GERHILD GRINSCHGL, BSc

Weighted Straight Skeleton - Grundlagen und Implementierung

MASTERARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieurin

Informatik

eingereicht an der

Technischen Universität Graz

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

Betreuer: Univ.-Prof. Dipl.-Ing. Dr.techn. Franz Aurenhammer

Graz, Oktober 2016

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 literally or by content from the used sources.

Graz, _____ _____

Date Signature

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 _____ _____

Datum Unterschrift

Danksagung

An dieser Stelle möchte ich mich bei allen bedanken, die zum Gelingen der vorliegenden Masterarbeit beigetragen haben. An erster Stelle möchte ich mich bei Univ.-Prof. Dipl.-Ing. Dr.techn. Franz Aurenhammer für die Betreuung der Arbeit und bei Dr. Gernot Walzl für die Unterstützung bei der Implementierung des Tools bedanken.

Des Weiteren möchte ich mich bei DI Dirk Martin für die fachliche Unterstützung und die guten Anregungen und Ratschläge bedanken. Ebenso gilt mein Dank meiner Schwester Andrea für das Korrekturlesen und die moralische Unterstützung während dieser Zeit.

Mein abschließender Dank gebührt meinen Eltern und meinem Freund Lukas, die mich während des gesamten Studiums stets unterstützt und motiviert haben.

Kurzfassung

Das Weighted Straight Skeleton stellt eine Erweiterung des Straight Skeleton dar. Es wurde erstmals von Aichholzer und Aurenhammer [1] im Zusammenhang mit der Konstruktion von Dächern erwähnt. Wie auch die klassische Variante des Straight Skeleton ist auch das Weighted Straight Skeleton eine Struktur aus nur geraden Liniensegmenten. Es wird nicht über eine Distanzfunktion definiert, sondern wird durch die Anwendung eines sogenannten "Schrumpfprozesses" konstruiert. Der Unterschied zur ungewichteten Version besteht darin, dass sich die Polygonkanten mit unterschiedlicher Geschwindigkeit während des Schrumpfprozesses bewegen. Hierzu wird jeder Polygonkante ein Gewicht zugewiesen.

In der vorliegenden Arbeit werden die Eigenschaften des Weighted Straight Skeleton untersucht und Unterschiede zur ungewichteten Version aufgezeigt. Anhand eines ausgewählten Algorithmus werden die notwendigen Schritte zur Berechnung des Weighted Straight Skeletons beschrieben. Die Implementierung des Algorithmus ist Teil eines Tools, welches zur Berechnung und Visualisierung des Weighted Straight Skeleton im Rahmen dieser Arbeit entwickelt und für die Generierung der Testdaten verwendet wurde.

Abstract

The weighted straight skeleton represents an extension of the classical straight skeleton. Aichholzer and Aurenhammer [1] mentioned it the first time in connection with the construction of rooftops. As the classic approach of the straight skeleton, the weighted straight skeleton is defined as a structure of only straight line segments. It is not defined via a distance function but it is constructed using a so called "shrinking process". The difference between both versions lies within the fact, that the edges of the polygon move with different speed during the shrinking process. Therefore a weight gets assigned to each edge.

In this thesis the different characteristics of the weighted straight skeleton will be analyzed and the difference between the classical approach will be pointed out. All necessary steps to calculate the weighted straight skeleton will be shown by using an selected algorithm. The implementation of this algorithm is part of a tool which is used for the calculation and visualization of the weighted straight skeleton. This tool was developed within the scope of this thesis. Also all test data in this thesis was generated by this tool.

Inhaltsverzeichnis

Abbildungsverzeichnis

Algorithmenverzeichnis

Tabellenverzeichnis

1 Einführung

Ob nun in der Computergrafik, Architektur oder Biologie - “Skelette” können in unterschiedlichen Bereichen sinnvoll eingesetzt werden. Sie stellen eine weitere Repräsentationsmöglichkeit eines geometrischen Objektes dar, indem sie die Ebene in Bereiche zerlegen, so dass die geometrische Form des Objektes widergespiegelt wird. In Laufe der Jahre sind einige Skelett-Strukturen entwickelt worden. Die wohl bekanntesten und am häufigsten genutzten sind die Mediale Achse, das Voronoi-Diagramm und das Straight Skeleton.

Das Straight Skeleton ist eine Struktur aus nur geraden Liniensegmenten. Es ist nicht über eine Distanzfunktion definiert, sondern wird durch die Anwendung eines Schrumpfprozesses konstruiert. Das Straight Skeleton weist eine geringere kombinatorische Komplexität als die Mediale Achse und das Voronoi-Diagramm auf. Es eignet sich besonders gut als Skelett, da sich daraus die ursprüngliche geometrische Form leicht rekonstruieren lässt. Aufgrund seiner Vorteile gegenüber den anderen beiden Strukturen dient das Straight Skeleton als zentrales Forschungsobjekt in wissenschaftlichen Publikationen und wird daher auch fortlaufend erweitert. Eine dieser Erweiterung stellt das Weighted Straight Skeleton dar. Hier bewegen sich die Polygonkanten mit unterschiedlicher Geschwindigkeit.

Eines der Hauptanwendungsgebiete des Straight Skeletons in der Architektur ist die Konstruktion von Dächern. Unter Zuhilfenahme z.B. der gewichteten Version können Dächer mit unterschiedlicher Steigung der Dachflächen konstruiert werden. Neben seinen Einsatz zur Konstruktion von Dächern findet das Weighted Straight Skeleton auch in zahlreichen anderen Anwendungsgebieten, wie z.B. bei Geoinformationssystemen, seinen Einsatz.

1.1 Stand der Forschung

Das Weighted Straight Skeleton wurde das erste Mal von Aichholzer und Aurenhammer [1] im Zusammenhang mit der Konstruktion von Dächern erwähnt. Beim klassischen Straight Skeleton haben alle Dachflächen die gleiche Steigung. Bewegen sich die Kanten jedoch mit unterschiedlicher Geschwindigkeit ins Innere des Polygons, können individuelle Steigungen pro Dachfläche erreicht werden. Nach Aichholzer und Aurenhammer beschäftigen sich auch Eppstein und Erickson [11] ausführlicher mit dem Weighted Straight Skeleton. Sie entwickelten einen Algorithmus, der sowohl das ungewichtete als auch das gewichtete Straight Skeleton in $O(n^{8/5+\varepsilon})$ Zeit und Speicher berechnen kann. Dies stellt eine Verbesserung der vorher bekannten Laufzeit zur Berechnung des Weighted Straight Skeleton dar. Detaillierter mit dem Weighted Straight Skeleton und seinen Eigenschaften beschäftigen sich auch Biedl *et al.* [6]. Sie zeigen auf, dass nicht alle Eigenschaften vom ungewichteten Straight Skeleton auf die gewichtete Version übergehen und dass während des Schrumpfprozesses Sonderfälle auftreten können, bei denen das Weighted Straight Skeleton nicht klar definiert ist. Für monotone Polygone mit nur positiven Kantengewichten entwickelten Biedl *et al.* [7] einen Algorithmus, welcher das Weighted Straight Skeleton in $\mathcal{O}(n \log n)$ Zeit und $\mathcal{O}(n)$ Speicher berechnen kann.

1.2 Ziel der Arbeit

Ziel der Arbeit ist es einen Überblick über das Weighted Straight Skeleton und seine Eigenschaften zu geben. Unterschiede zur ungewichteten Version, dem klassischen Straight Skeleton, sollen hierbei besonders hervorgehoben werden. Es existieren bereits einige Algorithmen zur Berechnung des Straight Skeletons. In dieser Arbeit soll untersucht werden, ob und inwiefern diese Algorithmen auch für die Berechnung des Weighted Straight Skeletons verwendet werden können und welche Anpassungen gegebenenfalls vorgenommen werden müssen. Anhand eines ausgewählten Algorithmus sollen die Schritte zur Berechnung des Weighted Straight Skeletons aufgezeigt werden. Notwendige Anpassungen des ursprünglichen Algorithmus werden beschrieben und potentielle Herausforderungen sichtbar gemacht.

1.3 Gliederung der Arbeit

Im Kapitel 2 dieser Arbeit erfolgt zunächst die Definition des Weighted Straight Skeletons (siehe Kapitel 2.1). Da das Weighted Straight Skeleton eine Erweiterung der ungewichteten Version darstellt, wird am Anfang dieses Kapitels zunächst das Straight Skeleton definiert und wichtige Eigenschaften dessen angeführt. Danach erfolgt eine kurze Beschreibung des Roof- und Terrain-Models, welches eine Interpretationsmöglichkeit des Straight Skeletons im $\mathbb{R}^3$ darstellt und später bei Beweisen der Eigenschaften des Weighted Straight Skeleton zum Einsatz gebracht wird. Abgeschlossen wird das erste Unterkapitel durch die Definition und Beschreibung der Eigenschaften des Weighted Straight Skeletons. Aufgrund seiner positiven Eigenschaften findet das Weighted Straight Skeleton seinen Einsatz in den unterschiedlichsten Anwendungsgebieten. Ausgewählte Einsatzgebiete werden in Kapitel 2.2 näher erläutert. Abschließend werden in Kapitel 2.3 Algorithmen zur Berechnung des Weighted Straight Skeletons vorgestellt.

Im Rahmen dieser Arbeit wurde ein Tool zur Berechnung des Weighted Straight Skeleton entwickelt, welches in Kapitel 3 detaillierter beschrieben wird. Am Anfang dieses Kapitel wird die Benutzeroberfläche und die implementierte Funktionalität beschrieben (siehe Kapitel 3.1). Danach wird der Hauptaugenmerk auf den implementierten Algorithmus gelegt (siehe Kapitel 3.2). Unter Zuhilfenahme des Tools wurden Testdaten generiert, welche im Kapitel 4 ausgewertet wurden. Im letzten Kapitel der Arbeit erfolgt eine Zusammenfassung und ein Ausblick auf eine mögliche Erweiterung des Weighted Straight Skeletons.

2 Weighted Straight Skeleton

Das Weighted Straight Skeleton stellt eine Erweiterung des Straight Skeletons dar. Aufgrund dessen werden in Kapitel 2.1 zunächst das Straight Skeleton (siehe Kapitel 2.1.1) und seine Eigenschaften definiert. Da für die Beweise der Eigenschaften des Weighted Straight Skeleton (siehe Kapitel 2.1.3) das sogenannte "Roof-Model" zum Einsatz kommt, wird dieses in Kapitel 2.1.2 beschrieben. Das Weighted Straight Skeleton erfüllt einige positive Eigenschaften, wodurch es in den unterschiedlichsten Gebieten seine Anwendung findet (siehe Kapitel 2.2). Abschließend werden in Kapitel 2.3 Algorithmen zur Berechnung des Weighted Straight Skeletons beschrieben.

2.1 Definition

Bevor das Straight Skeleton beschrieben wird, werden zu Beginn dieses Kapitels noch grundlegende Begriffe definiert.

Definition 1. *Ein Polygon ist eine Teilmenge der Ebene, die* (1) *homeomorph zu einer Kreisscheibe ist, und* (2) *stückweise linear begrenzt ist.*

Definition 2. *Ein einfaches Polygon wird bezogen auf eine Linie l als monoton bezeichnet, wenn jeder Schnitt einer jeden Linie l' orthogonal zu l mit dem Polygon zusammenhängend ist. Als y-monotones Polygon wird ein monotones Polygon bezeichnet, wenn jeder Schnitt einer jeden Linie l' orthogonal zur y-Achse mit dem Polygon zusammenhängend ist.*

Definition 3. *Unter einer Jordan-Kurve (bzw. einfachen Kurve) wird eine geschlossene oder offene Kurve verstanden, die sich selbst niemals schneidet.*

Definition 4. *Unter einem Skelett versteht man die Partitionierung der Ebene in Regionen, sodass die geometrische Form des Objektes (z.B. Polygon) widergespiegelt wird.*

Das Straight Skeleton gehört wie auch die Mediale Achse und das Voronoi-Diagramm zu der Gruppe der Skelette (siehe Definition 4). Da in Kapitel 2.1.1 und Kapitel 2.1.3 die Vorteile des (Weighted) Straight Skeletons gegenüber den beiden anderen Skeletttypen beschrieben wird, werden diese nun genauer erläutert.

Mediale Achse und Voronoi-Diagramm
Die Mediale Achse wurde von Blum [9] das erste Mal beschrieben. Die Mediale Achse eines Polygons besteht aus all jenen Punkten, die mehr als einen nächsten Nachbarn am Rand des Polygons enthalten. In der Ebene ist die Mediale Achse als die Menge der Zentren von maximalen Kreisen eines Gebietes, resp. Polygon, definiert. Sie findet ihre Anwendung in Gebieten wie der Formerkennung in der Bildverarbeitung oder Bewegungsplanung in der Robotik.
Unter Voronoi-Diagramm versteht man die Zerlegung eines Raumes U in Regionen, den sogenannten Voronoi Regionen, welche durch ein Set von Objekten S in diesen Raum beeinflusst werden [1]. Es ist über eine Distanzfunktion definiert, welche jedem Element im $S \times U$ eine reelle Zahl zuweist. Eine Voronoi Region eines Objektes $s \in S$ beinhaltet jene Objekte des Raumes U, die näher am Objekt s liegen als zu einem anderen Objekt $\in S$. Wie auch die Mediale Achse findet das Voronoi-Diagramm in Anwendungsgebieten, wie z.B. Biologie und Robotik, seinen Einsatz.

2.1.1 Straight Skeleton

Aichholzer *et al.* [3] definieren das Straight Skeleton $S(P)$ als eine Struktur aus nur geraden Liniensegmenten, welche aus Teilen der Winkelhalbierende der Polygonkanten eines Polygons P entsteht. Es teilt das Innere eines Polygons mit n Kanten in n monotone Polygone; für jede Kante entsteht ein Polygon. Im Gegensatz zur Medialen Achse wird das Straight Skeleton nicht über eine Distanzfunktion definiert, sondern entsteht durch Anwendung eines "Schrumpfprozesses" auf das Polygon P. Während dieses Schrumpfprozesses bewegen sich die Polygonkanten parallel zu ihrer Ausgangsposition und mit gleicher Geschwindikgeit ins Innere des Polygons. Die Polygonknoten bewegen sich hierbei entlang der Winkelhalbierenden. Die Geschwindigkeit, mit der sich die Polygonknoten bewegen, ist abhängig vom aufgespannten Winkel der Kanten inzident zum Knoten [1]. Im Laufe des Prozesses kann es zur Topologieänderungen des Polygons kommen.

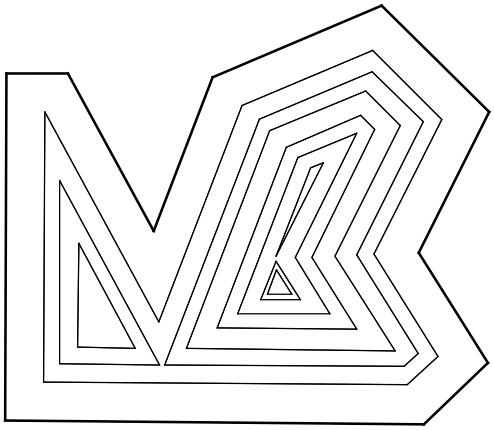

Abbildung 2.1: Polygon Hierarchie

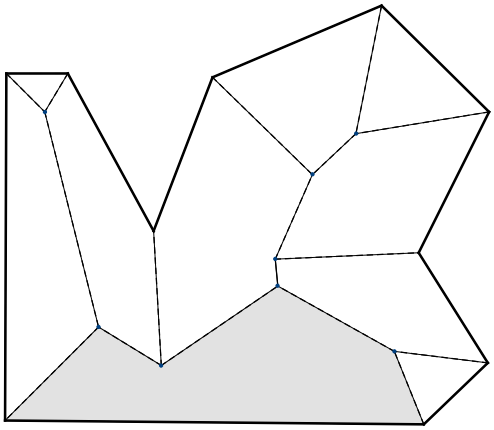

Abbildung 2.2: Straight Skeleton

Diese Änderungen werden als "Events" bezeichnet. Aichholzer *et al.* definieren zwei Eventtypen:

1. **Edge Event:** Ein Edge Event tritt dann auf, wenn eine Kante auf die Länge Null geschrumpft ist. Die benachbarten Kanten der Eventkante sind nach einem Edge Event adjazent.

2. **Split Event:** Bei einem Split Event trifft ein konkaver Knoten auf eine Kante und teilt diese in zwei Teile, wodurch zwei weitere Polygone entstehen. Die beiden Kanten inzident zum konkaven Knoten werden entsprechend mit dem der Eventkante verbunden.

Abhängig vom Eventtyp entstehen nach jedem Event ein oder zwei neue Polygone, für die wiederum der Schrumpfprozess durchgeführt wird. Das Ergebnis des Schrumpfprozess ist eine Hierarchie von verschachtelten Polygonen (siehe Abbildung 2.1).

Eigenschaften des Straight Skeletons

Laut Aichholzer *et al.* [3, 1] erfüllt das Straight Skeletons einige wertvolle Eigenschaften. Bevor diese näher erläutert werden, erfolgt zunächst die Definition grundlegender Begriffe, die in diesem Zusammenhang bedeutsam sind.

Begriffsdefinition
Wie am Anfang dieses Kapitel schon erwähnt, ist das Straight Skeleton als die Vereinigung von Teilen von Winkelhalbierenden "Kanten", die während des Schrumpfprozesses entstehen, definiert. Als "Face" wird jene Region des Polygons bezeichnet, welche von einer Kante e von P erreicht wird, bevor sie auf eine Kantenlänge von Null zusammenschrumpft. Die Polygonkanten werden parallel ins Innere des Polygons verschoben; die durch die Verschiebung entstanden Kanten werden als "Wavefront Kanten" (engl. *wavefront edges*) bezeichnet. Für jedes Edge- oder Split Event wird ein neuer Knoten in das Straight Skeleton eingefügt. Diese Knoten stellen die Endpunkte der Kanten des $S(P)$ dar, gehören nicht zum Polygon P und werden in weitere Folge als "Nodes" bezeichnet. Abbildung 2.2 zeigt das Straight Skeleton eines konkaven Polygons. Die strichlierten Linien stellen die Kanten des Straight Skeletons dar. Die Nodes des Straight Skeletons sind als blaue Punkte eingezeichnet. Die graue Fläche zeigt ein Face einer Polygonkante des konkaven Polygons.

Im ersten Schritt soll nun gezeigt werden, dass das Straight Skeleton ein Baum ist und seine Faces zusammenhängend sind.

Lemma 1. *Das $S(P)$ ist ein Baum mit n zusammenhängenden Faces, $n-2$ Nodes und $2n-3$ Kanten.*

Beweis. Sei $f(e)$ das Face einer Kante $e \in P$. $f(e)$ entsteht an der Kante e und kann nicht geteilt werden, auch nicht, wenn e aufgeteilt wird. Die Konstruktion des Faces endet, wenn alle Teile von e auf die Länge Null geschrumpft sind. Die Kante e kann danach nicht wieder erscheinen, wodurch $f(e)$ zusammenhängend ist und $S(P)$ azyklisch. Daraus folgt, dass $S(P)$ ein Baum mit n Knoten von P als Blätter sowie $n-2$ Nodes und $2n-3$ Kanten ist. □

In [1] definieren Aichholzer und Aurenhammer die Anzahl der Nodes eines Straight Skeletons für einen PSLG G.

Lemma 2. *Sei G ein PSLG mit t Knoten von Grad eins und gesamt n Knoten. Dann hat das $S(G)$ genau $2n+t-2$ Nodes.*

Unter Berücksichtigung von Lemma 1 lässt sich eine weitere Eigenschaft für die Faces eines $S(P)$ ableiten.

Lemma 3. *Die Faces eines $S(P)$ sind monotone Polygone.*

Beweis. Sei $f(e)$ ein Face, welches an der Kante e entsteht. Lemma 1 zeigt, dass $f(e)$ zusammenhängend ist. Das Face $f(e)$ ist monoton in Richtung der Kante e, wenn gilt, dass jede Überschneidung von $f(e)$ mit jeder Linie L, normal zu e zusammenhängend ist. Sei L eine Linie, die $f(e)$ an einem Punkt x verlässt und zu einem späteren Zeitpunkt am Punkt y wieder betritt. Zwischen den Punkten x und y wird L von Wavefront Kanten gekreuzt, die nicht mehr länger normal zu L sind. Diese Linien würden andernfalls y früher erreichen als die Wavefront Kanten der Kante e. Dies stellt einen Widerspruch zur Definition von y dar. Daraus folgt, dass Face $f(e)$ ein monotones Polygon ist. □

Das $S(P)$ kann aus konvexen oder konkaven Kanten bestehen. Aus jedem konvexen Polygonknoten entsteht eine konvexe Kante; aus jedem konkaven Knoten eine konkave Kante (siehe Lemma 4). Eine Besonderheit der konvexen Kanten ist es, dass sie auch zwei Nodes des $S(P)$ verbinden können, was jedoch bei konkaven Kanten nicht möglich ist.

Lemma 4. *Konkave Kanten von $S(P)$ entstehen nur durch konkave Knoten von P.*

Beweis. Sei uv jene Kante, die vom Punkt $v \in P$ aus hervorgeht. Der Node u kann entweder aus einem Edge- oder Split Event entstanden sein. Per Definition führt jeder konvexe Knoten $\in P$ zu einer konvexen Kante und jeder konkave Knoten zu einer konkaven Kante. Daher reicht es aus zu zeigen, dass nach einem Event die Kante $\in S(P)$, welche bei u startet, konvex ist. Sei vw jene Polygonkante, die nach einem Edge Event verschwindet. Da sich die Kanten wu und vu am Node u treffen, ist u zum Zeitpunkt des Events ein konvexer Knoten des neuen Polygons. Im Falle eines Split Events wird das Polygon am Node u geteilt. Es ist offensichtlich, dass u zu diesem Zeitpunkt ein konvexer Knoten für beide neuen Polygone ist. Daraus folgt, dass jeder Knoten, welcher während des Schrumpfprozesses entsteht, ein konvexer Knoten ist und somit auch alle Kanten, welche am diesem Node starten, konvex sind. Somit folgt, dass jede Kante, welche am Node u startet, konvex ist. □

Korollar 1. *Ein konkaver Knoten verschwindet nach einem Edge- oder Split Event.*

Neben den oben angeführten Eigenschaften, weist das $S(P)$, aufgrund seiner Baumstruktur, kombinatorisch eine kleinere Größe als die Mediale Achse [19] bei einem konkaven Polygon P auf. Sei P ein Polygon mit n Kanten und r konkaven Knoten,

dann besteht das $S(P)$ aus $2n-3$ Kanten (siehe Lemma 1), die Mediale Achse hingegen besteht aus $2n+r-3$ Kanten, wobei r davon parabolisch gekrümmt sind. Im Falle eines konvexen Polygons entspricht das Straight Skeleton der Medialen Achse. Die Mediale Achse, das Straight Skeleton und auch das Voronoi-Diagramm werden als "Skelette" bezeichnet. Ein Skelett eines planaren Graphen G mit geraden Kanten zerlegt die Ebene in Regionen, sodass jede Region die geometrische Form des Graphen widerspiegelt. Sie finden in vielen Bereichen, wie z.B. der Mustererkennung und Computergrafik, Anwendung. Die kombinatorische Komplexität des Straight Skeletons ist linear in der Anzahl der Knoten des Graphen G und im Allgemeinen kleiner als die des Voronoi-Diagramms. Aus diesem Grund und der Tatsache, dass sich der Graph G, unter der Voraussetzung, dass die Nodes des $S(G)$ nach ihrer Distanz zum Graphen beschriftet wurden, leicht aus dem $S(G)$ rekonstruiert lässt, eignet sich das Straight Skeleton besonders gut als Skeleton.

2.1.2 Roof- und Terrain-Model

Aichholzer *et al.* definieren das Roof- und Terrain-Model als eine Interpretationsmöglichkeit des Straight Skeletons für einfache Polygone [3] und planaren Graphen mit nur geraden Kanten (PSLG, engl. *planar straight line graph*) [1] im $\mathbb{R}^3$. Das $S(P)$ oder $S(G)$ induziert eine Distanzfunktion auf P oder G. Sie gibt für einen Punkt x in der Ebene genau jenen Zeitpunkt an, an dem er das erste Mal von einer Wavefront Kante getroffen wird. Die Grundidee ist es, den Schrumpfprozess im $\mathbb{R}^3$ durchzuführen, wobei die z-Achse jene Zeit darstellt [6]. Während des Prozesses wird eine Fläche das sogenannte "Roof-Model" durchlaufen. Sei $\mathcal{W}_P(t)$ die Vereinigung aller Wavefront Kanten zu einem bestimmten Zeitpunkt (alternativ das geschrumpfte Polygon P zum Zeitpunkt t) dann ist $\mathcal{T}(\mathcal{P}) = \bigcup_{t\geq 0} W_P(t) \times \{t\}$. Werden die Kanten des $\mathcal{T}(\mathcal{P})$ auf die Ebene $\mathbb{R}^2 \times \{0\}$ projektiert, erhält man das $S(P)$. Im Gegensatz dazu erhält man $\mathcal{T}(\mathcal{P})$, indem jeder Knoten von $S(P)$ anhand seiner orthogonalen Distanz zur entsprechenden Polygonkante in den $\mathbb{R}^3$ gehoben wird. Wie bereits beschrieben, entspricht diese Distanz jenem Zeitpunkt, an dem der Knoten von der ersten Wavefront Kante erreicht wurde.
Eine Sonderform des Roof-Models bildet das "Terrain". Ein Roof-Model wird als Terrain bezeichnet, wenn jede Linie parallel zur z-Achse diese maximal in einen Punkt schneidet.

Abhängig vom Einsatzgebiet definieren Aichholzer *et al.* [3, 2] zwei Begriffe: "Roof" und "Island".

Roof und Island
Gegeben ist der Grundriss eines Hauses als Inputpolygon P. Gesucht wird eine einfache Methode um ein geeignetes Dach für das gegebene Polygon zu generieren. Eine Möglichkeit ist, dass $\mathcal{T}(P)$ basierend auf dem $S(P)$ zu konstruieren. Sei P eingebettet in die horizontale Ebene Π_0 und jede Kante $e \in P$ mit einer Halbebene Π_e im $\mathbb{R}^3$ assoziiert. Jede dieser Halbebenen Π_e ist durch die Linie $l(e)$ begrenzt. Des Weiteren besitzt sie eine fixe Steigung (z.B. 45°) bezogen auf Π_0 und ist schräg in Richtung P. Für zwei unterschiedliche Kanten e_1 und e_2 projiziert der Schnitt der Halbebenen Π_{e_1} und Π_{e_2} vertikal auf den Bisektor von e_1 und e_2. Das beschriebene Objekt wird als "Dach (engl. *Roof*)" bezeichnet und kann durch die Berechnung des $S(P)$ konstruiert werden. Die konvexen Kanten des $S(P)$ erzeugen die Dachfirste und konkave Kanten stellen die Dachkehlen dar. Die Endpunkte der Kanten werden als "Corners" bezeichnet und projektieren zu den Nodes des $S(P)$.

Das $S(P)$ bietet eine einzigartige Möglichkeit, um Dächer für ein einfaches Polygon zu konstruieren, ohne dass, wie bei anderen planaren Bisektor Graphen (engl. *plane bisector graph*), Anomalien auftreten. So kann es im Gegensatz zum $S(P)$ bei anderen Graphen passieren, dass eine Kante des Polygons P mehrere Faces im Graphen erzeugt. Auch lokale Minima sind möglich. Dies impliziert, dass die Größe des Graphen nicht linear ist. Eine obere Grenze wäre hier $\mathcal{O}(n^3)$, da jede Node des Graphen von einem anderen Tripel aus Polygonkanten entsteht. Des Weiteren stellt nur das $S(P)$ sicher, dass das Regenwasser abrinnen kann und sich bei Regen nicht am Dach sammelt. Eine Eigenschaft, die für Dächer unabdingbar ist.

Neben der Verwendung des $\mathcal{T}(\mathcal{P})$ als Dach, kann es auch als Insel interpretiert werden. Das einfache Polygon P stellt dabei die Küste dar. Der Wasserlevel steht bei Ebene Π_0 und steigt während des Schrumpfprozesses kontinuierlich an. Ein Split Event tritt dann auf, wenn das Wasser ein lokales Maximum der Insel umschließt. Das daraus resultierende Objekt wird als Insel des Polygons P bezeichnet.

2.1.3 Definition und Eigenschaften des Weighted Straight Skeletons

Das Weighted Straight Skeleton SK_W [4] stellt eine Erweiterung des im Kapitel 2.1.1 beschriebenen Straight Skeleton dar. Es unterscheidet sich von der ungewichteten Version dadurch, dass sich die Polygonkanten mit unterschiedlichen Geschwindigkeiten ins Innere des Polygon P bewegen. Hierzu wird jeder Kante $e \in P$ ein Gewicht w_i zugewiesen. Bei negativem Kantengewicht bewegt sich die Kante außerhalb des Polygons, wohingegen bei einem Kantengewicht $w_i == 0$ keine Bewegung der Kante stattfindet. Das Weighted Straight Skeleton entspricht der ungewichteten Version

genau dann, wenn alle Kantengewichte $W = (w_1, \ldots, w_n)$ positiv und gleich sind. Sind alle Kantengewichte positiv, so entspricht das Weighted Straight Skeleton der gewichteten Medialen Achse.
Während des Schrumpfprozesses können, wie auch bei der ungewichteten Variante, Edge- und Split Events auftreten. Die Definition des Edge Events von Kapitel 2.1.1 kann für das Weighted Straight Skeleton vollständig übernommen werden. Bei der Definition des Split Events gibt es jedoch Unterschiede zur ungewichteten Variante [6]. Ein Split Event tritt dann auf, wenn ein Knoten v auf eine Wavefront Kante trifft. Im Gegensatz zur ungewichteten Version muss es sich hierbei nicht um einen konkaven Knoten handeln. Es besteht unter anderem auch die Möglichkeit, dass die Kante nicht in mehrere Teile zerlegt wird. Des Weiteren kann es nach dem Split Event mehrere Varianten geben, wie der Schrumpfprozess weitergeführt werden kann.

Beim Weighted Straight Skeleton können bei oder nach einem Edge- oder Split Event Sonderfällen auftreten, bei denen das Skeleton nicht eindeutig definiert ist. So ist es nach einem Edge Event möglich, dass parallele Wavefront Kanten mit unterschiedlichen Gewichten adjazent geworden sind. Würde dieser Fall gleich wie der wohldefinierte behandelt werden, so würde, aufgrund der unterschiedlichen Geschwindigkeiten, eine fortlaufend größer werdende Kluft zwischen den beiden Kanten entstehen. Laut Biedl *et al.* [6] könnte diese Lücke durch eine weitere Wavefront Kante geschlossen werden. Jedoch hätte diese Kante keinerlei Verbindung zu einer Polygonkante, noch wäre sie parallel zu einer. Eine bessere Variante laut Biedl *et al.* ist es, eine Kante zugunsten der anderen aufzugeben. Des Weiteren besteht die Möglichkeit, dass zwei Kanten e_1 und e_2, welche sich in die gleiche Richtung bewegen, nach einem Edge Event benachbart sind. Der gemeinsame Endpunkt wird als Knoten zur Wavefront hinzugefügt und bewegt sich perpendikular zu den Kanten e_1 und e_2. Dieser neue Knoten wird als “Ghost Knoten” bezeichnet.

Einen weiteren Sonderfall bilden mehrere Split Events, die gleichzeitig am selben Ort p stattfinden. In diesem Fall ist es laut Biedl *et al.* [6, 8] nicht eindeutig, wie sich die Topologie der Wavefront Kante verändert. Sichergestellt werden muss, dass die Wavefront Kanten nach dem Event in der Nähe von p keinerlei Kreuzungen enthalten.

Eigenschaften des Weighted Straight Skeletons

Für die Beschreibung der Eigenschaften $SK_W(P)$ gehen Biedl *et al.* [6] davon aus, dass keiner der in diesem Kapitel erwähnten Sonderfälle eintritt. Daraus folgt, dass keine zwei parallelen Kanten am Beginn oder später nach einem Edge Event benachbart sind und dass nicht mehrere Events gleichzeitig am gleichen Ort stattfinden können.

Im ersten Schritt soll gezeigt werden, dass das gewichtete Straight Skeleton unter bestimmten Bedingungen keine Kreuzungen enthält. Hierzu ist es laut Biedl *et al.* sinnvoll zu betrachten unter welchen Umständen das $\mathcal{T}_W(P)$ ein Terrain ist.

Lemma 5. *Sei P ein Polygon mit oder ohne Löcher. Wenn alle Kantengewichte positiv sind, dann ist das Roof-Model $\mathcal{T}_W(P)$ ein Terrain und die z-Projektion entspricht P.*

Beweis. Da alle Kantengewichte positiv sind, bewegt sich die Wavefront ins Innere des Polygons P. Nach jedem Event zeigen die Trajektorie der neuen Wavefront Knoten auf jene Region im Polygon, die noch nicht von der Wavefront erreicht wurde, d.h. es ist ausgeschlossen, dass eine Region im Polygon mehrmals von einer Wavefront erreicht wird. Des Weiteren wird jedoch jede Region des Polygons zumindest einmal von der Wavefront erfasst. Wäre dies nicht der Fall, würde der Rand der noch nicht erreichten Region die Wavefront darstellen und somit wäre $\mathcal{W}_W(P) \neq \emptyset$. □

Von Lemma 5 lässt sich folgende Aussage über die Kreuzungsfreiheit des Weighted Straight Skeleton ableiten:

Lemma 6. *Sei P ein Polygon mit oder ohne Löcher. Wenn alle Kantengewichte positiv sind, dann ist $S_W(P)$ planar.*

Beweis. Laut Lemma 5 kann kein Punkt p mehrmals von einer Wavefront Kante erfasst werden. Wäre $S_W(P)$ nicht kreuzungsfrei, wäre dies jedoch der Fall. □

Für den späteren Beweis, dass das $S_W(P)$ unter bestimmten Bedingungen ein Baum ist, ist es hilfreich die Eigenschaften der Faces der Polygonkanten zu betrachten. Das Face einer Kante e ist die Vereinigung aller Geraden zum Zeitpunkt t, die von der Polygonkante e entstanden sind $f(e) = \bigcup_{t>0} e(t)$. Wie auch bei der ungewichteten Version ist das Face $f(e)$ zusammenhängend, da Teile einer Kante nicht während des Prozesse plötzlich (wieder) erscheinen können. Der Rand eines Faces bd $f(e)$ besteht aus Kanten des $S_W(P)$ und der Polygonkante e.

Lemma 7. *Sei P ein einfaches Polygon mit positiven Kantengewichten. Dann ist bd $f(e)$ für jede Polygonkante e ein einfaches Polygon.*

Beweis. Wenn ein Face $f(e)$ in mehrere Teile zerlegt wird, können diese durch die Ghost Knoten später nicht wieder zusammengefügt werden. Die Faces sind daher offene, einfach zusammenhängende Sets und ihr Rand ist ein schwaches, einfaches Polygon. Es gibt nur einen Fall, bei dem der Rand eines Faces bd $f(e)$ kein einfaches Polygon ist. In diesem Fall werden zwei Teile eines Faces s_1, s_2 in der Wavefront benachbart. Wenn der Verlauf von s_1 und s_2 zu der Kante e innerhalb des Faces $f(e)$ zurückverfolgt wird, kann eine geschlossene Jordan Kurve $\mathcal{C}$ innerhalb des Abschlusses $cl(f(e))$ gefunden werden, die einen Punkt beinhaltet, welcher nicht zu $cl(f(e))$ gehört. Lemma 5 zeigt, dass bei positiven Kantengewichten das Roof-Model $\mathcal{T}_W(P)$ auf P projektiert. Somit gilt: $\mathcal{C} \subset cl(f(e)) \subset P$. Da P ein einfaches Polygon ist, gilt, dass jeder Punkt in $\mathcal{C}$ auch in P ist. Da ein Punkt von $\mathcal{C}$ nicht zu $cl(f(e))$ gehört, muss noch ein anderes Face $f(e_2)$ in $\mathcal{C}$ sein. Laut Lemma 6 ist das $S_W(P)$ kreuzungsfrei und wodurch folgt, dass die Kante e_2 des Faces $f(e_2)$ auch im Inneren von $\mathcal{C}$ ist. Da $\mathcal{C}$ jedoch keine Kanten von P beinhaltet, kann e_2 nicht mit dem äußeren Rand von P verbunden sein. Das Polygon P hat daher ein Loch, was einen Widerspruch darstellt. □

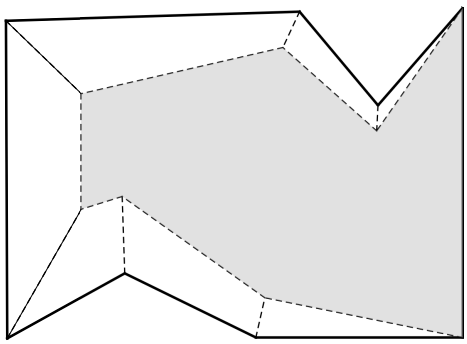

Abbildung 2.3: Nicht monotones Face

Laut Lemma 3 sind die Faces bei der ungewichteten Version des Straight Skeletons monotone Polygone. Wird für jede Kante das gleiche Gewicht gewählt, so entspricht das $S_W(P)$ dem $S(P)$, d.h auch in diesem Fall sind die Faces monotone Polygone. Dies gilt unabhängig davon, ob das Polygon Löcher besitzt oder nicht. Werden jedoch unterschiedliche Kantengewichte gewählt, kann es vorkommen, dass diese Eigenschaft nicht mehr erfüllt wird. Abbildung 2.3 zeigt ein Beispiel für ein nicht monotones Face.

Handelt es sich beim Inputpolygon jedoch um ein konvexes Polygon sind die Faces des Weighted Straight Skeleton nicht nur monotone sondern konvexe Polygone.

Theorem 1. *Sei P ein gewichtetes konvexes Polygon, dann ist der Körper umschlossen von $P \cup \mathcal{T}_W(P)$ ein konvexes Polyeder.*

Korollar 2. *Sei P ein gewichtetes konvexes Polygon, dann sind die Faces des Weighted Straight Skeleton konvex.*

Wie in Kapitel 2.1.1 beschrieben, ist das Straight Skeletons ein Baum. Auch bei der gewichteten Version definiert Aurenhammer [4], dass das $SK_W(P)$ ein Baum mit n Blättern und höchstens $n-2$ Nodes ist, wobei n die Anzahl der Polygonknoten ist. Biedl *et al.* [6] gibt an, dass das Weighted Straight Skeleton $SK_W(P)$ dann ein Baum ist, wenn nur positive Kantengewichte gewählt wurden. Als Basis für den Beweis nutzen Biedl *et al.* die Tatsache, dass $P \cup \mathcal{T}_W(P)$ eine 2-Mannigfaltigkeit ist und daher die Euler-Poincaré Formel angewendet werden kann. Dabei ist $\mathcal{T}_W(P)$ das gewichtete "Roof-Model" (siehe Kapitel 2.1.1). Sei n die Anzahl der Polygonknoten, v die Anzahl der Weighted Straight Skeleton Nodes, e die Anzahl der Kanten, sowie h die Anzahl der Löcher im Polygon und g der Genus von $T_W(P)$, dann gilt:

$$e = n + v - 1 + 2g - h.$$

Beweis. Die Faces des Roof-Models $\mathcal{T}_W(P)$ sind schwache, einfache Polygone (siehe Lemma 7) und P hat per Definition h Löcher. Das Polygon P wird zu einem schwachen einfachen Polygon transformiert, indem für jedes Loch h eine Kante zwischen zwei Polygonknoten erstellt wird. Diese Kante verbindet entweder ein Loch direkt mit dem Polygonrand oder indirekt über ein bereits verbundenes Polygonloch. Der Körper $P \cup \mathcal{T}_W(P)$ besteht nun aus $v' = v + n$ Knoten und $e' = e + n + h$ Kanten sowie aus $f' = n + 1$ Faces. Jedes dieser Faces kann, ohne die Euler-Poincaré Charakteristik zu verlieren, trianguliert werden. Somit folgt $e = n + v - 1 + 2g - h$. □

Lemma 8. *Das $S_W(P)$ eines einfachen Polygons ist genau dann ein Baum, wenn der Körper umschlossen von $P \cup \mathcal{T}_W(P)$ Genus Null hat.*

Beweis. Für den Beweis muss vorerst gezeigt werden, dass $S_W(P)$ zusammenhängend ist. Seien p und q zwei Punkte, die über einen Pfad in $S_{W_t}(P) \cup \mathcal{W}_{P,W}(t)$ miteinander verbunden sind. $S_{W_t}(P)$ stellt das Weighted Straight Skeleton zum Zeitpunkt t dar. Es gilt zu zeigen, dass diese Verbindung auch noch nach dem Auftreten von Events besteht. Nach einem Edge- oder Split Event wird ein Straight Skeleton Knoten v hinzugefügt, der jene Kanten des $S_W(P)$ miteinander verbindet, die am Event beteiligt

waren. Diese Kanten werden von Punkten der am Event beteiligten Komponenten erzeugt. Bei einem Split Event wird eine Wavefront Kante in mehrere Komponenten zerlegt. Jede dieser Komponenten ist über eine Straight Skeleton Kante und somit auch wiederum über einem Knoten der Komponente mit dem Knoten v verbunden. Daraus folgt, dass ein Event eine Komponente nie vom Event erzeugten Knoten trennt. Nach Induktion sind die Knoten p und q in $S_{W'_t}(P) \cup \mathcal{W}_{P,W}(t')$, für t' nach dem letzten Event miteinander verbunden. Bei negativen Kantengewicht kann es vorkommen, dass Wavefront Komponenten unendlich lange propagiert werden. In diesem Fall wird für jede dieser Komponente ein spezieller Knoten mit unendlicher x- und y-Koordinate in $S_W(P)$ eingefügt. Indem für jede Wavefront Komponente $\in \mathcal{W}_{P,W}(t')$ ein Knoten erstellt wird, kann das $S_W(P)$ aus $S_{W'_t}(P) \cup \mathcal{W}_{P,W}(t')$ abgeleitet werden. Somit gilt, dass die Knoten p und q pfad-zusammenhängend in $S_W(P)$ sind. Da einfache Polygone initial aus einer zusammenhängenden Komponente bestehen, gilt, dass das $S_W(P)$ zusammenhängend für einfache Polygone ist.

Das $S_W(P)$ ist genau dann ein Baum, wenn $e = n + v - 1$ gilt. Wie weiter oben beschrieben, gilt $e = n + v - 1 + 2g$. Daraus folgt, dass $S_W(P)$ genau dann ein Baum ist, wenn $g = 0$.

□

Für einfache Polygonen mit nur positiven Kantengewichten gehen sehr viele Eigenschaften der ungewichteten auf die gewichtete Version über. Lediglich die Eigenschaft, dass die Faces des Straight Skeletons monotone Polygone sind, geht verloren (siehe Lemma 3). Wird der Polygontyp auf konvexe Polygone beschränkt, wird laut Biedl *et al.* [6] auch diese Eigenschaft beim Weighted Straight Skeleton erfüllt.

Anders sieht es aus, wenn auch negative Kantengewichte erlaubt werden. Hier kann der Fall eintreten, dass für ein einfaches Polygon das $S_W(P)$ nicht mehr kreuzungsfrei und zyklenfrei ist. Biedl *et al.* haben dies anhand eines einfachen Polygons mit Kantengewichten von $\{1, -1\}$ gezeigt und damit gleichzeitig aufgezeigt, dass für ein einfaches Polygon mit einer Kombination aus positiven und negativen Kantengewichten das Weighted Straight Skeleton kein Baum sein muss. Bei den konvexen Polygonen bleibt diese Eigenschaft jedoch auch bei negativen Kantengewichten erhalten. Aber auch hier existieren Gewichtskonfiguration, die verursachen, dass das Weighted Straight Skeleton $S_W(P)$ nicht mehr kreuzungsfrei ist.

2.2 Anwendungsgebiete

Das Weighted Straight Skeleton findet in unterschiedlichen Gebieten Anwendung. Aurenhammer [4] zeigt, dass das Weighted Straight Skeleton sich zum Zerlegen eines konvexen Polygons n konvexe Teile mit vordefinierten Bedingungen eignet (siehe Kapitel 2.2.1). Barequet *et al.* setzen [5] das Weighted Straight Skeleton ein, um die initiale Topologie für die Berechnung des Straight Skeletons eines allgemeinen Polyeder zu konstruieren (siehe Kapitel 2.2.2). Haunert *et al.* [13] hingegen verwenden das Weighted Straight Skeleton, um die Topologie bei der Generalisierung von Landkarten aufrechtzuerhalten (siehe Kapitel 2.2.3). Sugihara [23] hingegen benutzt das Weighted Straight Skeleton für das Design von Pop-up Karten (siehe Kapitel 2.2.5).

2.2.1 Polygonzerlegung

Aurenhammer [4] verwendet das Weighted Straight Skeleton, um ein konvexes Polygon unter Einhaltung von vordefinierten Bedingungen in n konvexe Teile zu zerlegen. Jedes dieser Subpolygone beinhaltet eine Kante des konvexen Polygons und erfüllt bestimmte Bedingungen. Bedingungen können hierbei z.B. ein bestimmter Flächeninhalt pro Polygon oder eine bestimmte Anzahl von Punkten einer Punktmenge in den Subpolygonen sein. Eine Zerlegung eines konvexen n-Eck P gilt als korrekt (engl. *proper*) wenn sie aus n einfachen Polygonen besteht und jedes dieser Polygone genau eine Kante des Polygon P enthält. Wenn keiner Polygonkante das Gewicht Null zugewiesen wurde, liefert das Weighted Straight Skeleton eine korrekte Polygonzerlegung. Neben dem Weighted Straight Skeleton gibt es auch andere Polygonzerlegungen, die diese Bedingungen erfüllen. Jedoch erfüllen laut Aurenhammer [4] jene Polygonzerlegungen, die durch die Verwendung des Weighted Straight Skeleton entstanden sind, einige Optimalitätskriterien.

Optimalitätskriterien
Sei $\nu(x, e_i)$ der Normalabstand eines Punkt x zu einer Geraden durch die Endpunkte einer Polygonkante e_i von P und $\Pi : P \to \{e_1, \ldots, e_n\}$ eine Polygonzerlegung mit fixen Polygonflächeninhalten $A_1, \ldots, A_n$. Das Weighted Straight Skeleton des Polygons P minimiert dann

$$\int_{x \in P} \nu(x, \Pi x) \, \mathrm{d}x$$

über alle Polygonzerlegungen Π. Sei die Höhe eines Polygons $Q = \Pi^{-1}(e_i)$ definiert als der maximaler Normalabstand $\nu(x, e_i)$, der für einen Punkt $x \in Q$ auftritt und die

Höhe einer Zerlegung Π die maximale Höhe, die für ein Polygon in Π auftritt. Dann gilt, dass die Polygone des Weighted Straight Skeleton minimalste Höhe aufweisen.

Für eine gegebene Polygonzerlegung Π eines Polygon P kann einfach festgestellt werden, ob es sich um das Weighted Straight Skeleton handelt oder nicht.

Lemma 9. *Sei $L(e)$ eine gerade Linie, welche das Liniensegment e beinhaltet und Π eine Polygonzerlegung des Polygons P, welche das Polygon in konvexe Polygone $Q_1, \ldots, Q_n$ zerlegt. Jedes dieser konvexen Polygone beinhaltet eine Seite e_i des Polygons P. Die Zerlegung Π ist genau dann für ein gegebenes Set von Gewichten das Weighted Straight Skeleton, wenn für jede Kante b der Zerlegung $L(e_i) \cap L(e_j) \in L(b)$ gilt.*

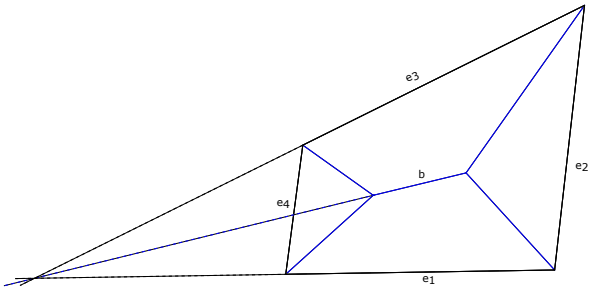

Abbildung 2.4: Polygonzerlegung mit Weighted Straight Skeleton. Es gilt $L(e_1) \cap L(e_2) \in L(b)$.

Aurenhammer [4] beschreibt auch Algorithmen, wie eine Polygonzerlegung berechnet werden kann. Die einfachste Variante hierbei ist, dass ein konvexes Polygon und ein Tupel von Kantengewichten gegeben sind und das Weighted Straight Skeleton ohne weitere Bedingungen konstruiert werden soll. In diesem Fall existiert ein Algorithmus, welcher das Weighted Straight Skeleton in linearer Zeit berechnet. Schwieriger ist es jedoch, wenn zusätzlich noch Bedingungen, wie zum Beispiel ein bestimmter Flächeninhalt pro Subpolygon, gefordert werden. In diesem Fall muss zuerst ein entsprechendes Tupel von Kantengewichten gefunden werden. Abhängig von den Bedingungen gibt es unterschiedliche Varianten, wie das Tupel gefunden werden kann. Sichergestellt ist hierbei, laut Aurenhammer, dass so ein Tupel von Kantengewichten existiert (siehe Theorem 2).

Theorem 2. *Sei ρ eine beliebige kontinuierliche Dichtefunktion auf dem konvexen Polygon P mit Kanten $e_1, \ldots, e_n$ und μ das Maß definiert von ρ auf P. Für beliebig positive Zahlen $A_1, \ldots, A_n$, deren Summe eins ist, existiert ein n-Tupel von Gewichten, sodass für jede Region des $S_W(P)$ $\mu(reg(e_i)) = A_i$ gilt.*

2.2.2 Erstellung der initialen Topologie zur Berechnung des Straight Skeletons von Polyedern

Bei der Berechnung von Straight Skeletons allgemeine Polyeder kann der Fall eintreten, dass während des Schrumpfprozesses mehrere gültige Topologiekonfigurationen von einem Zustand aus existieren. Dieser Fall tritt unter anderem dann ein, wenn ein oder mehrere Knoten einen Grad von größer Drei besitzen. Um diese Anomalie zu beheben, verwenden Barequet *et al.* [5] das Weighted Straight Skeleton. Da es sich beim Inputpolyeder um ein allgemeines Polyeder handelt, kann es aus Knoten mit einem Grad größer Drei bestehen. Ziel ist es also, diese Knoten auf mehrere Knoten von Grad Drei aufzuteilen. Hierzu schneiden Barequet *et al.* jene Faces, die einen solchen Knoten umgeben, mit einer oder mehreren Ebenen. Jede Schnittebene, welche alle Faces schneidet und nicht parallel zu einen von ihnen ist, kann hierfür verwendet werden. Kann keine solche Schnittebene gefunden werden, kann laut Walzl [25] der von Barequet *et al.* beschriebene Ansatz nicht verwendet werden. Existiert sie jedoch, dann liefert sie, durch die Überschneidung der Faces mit der Schnittebene, das Inputpolygon für die Berechnung des Weighted Straight Skeletons. Die Gewichte zur Berechnung des Weighted Straight Skeletons werden aus dem Diederwinkel der Faces mit der Schnittebene berechnet. Vorteil dieses Ansatzes ist es, dass er immer eine eindeutige Lösung liefert. Da jeder Knoten des Weighted Straight Skeletons in der Ebene einen Knoten des schrumpfenden Polyeder in Raum repräsentiert, zeigt dieser Ansatz wie die Knoten von einem Grad größer Drei aufgeteilt werden müssen.

2.2.3 Erhaltung von Topologien bei der Generalisierung von Landkarten

Unter der Generalisierung von Landkarten wird das Vereinfachen, Zusammenfassen und Typisieren sowie Selektieren von Objekten verstanden. Im Zuge der Generalisierung, wenn zum Beispiel der Maßstab verkleinert wird, müssen einige kleinere Gebiete eliminiert werden. Im Allgemeinen wird das zu eliminierende Gebiet einem Nachbargebiet zugeordnet. Da die Landkarte vereinfacht werden soll, muss sichergestellt werden, dass durch das Zusammenfassen der Gebiete keine komplexe Form entsteht. In manchen Fällen kann dies nicht umgesetzt werden, weshalb das Gebiet in mehrere Teilgebiete zerlegt werden muss, welche dann wiederum unterschiedlichen Nachbargebieten zugeordnet werden. Gewährleistet muss hierbei sein, dass sich die Gebiete nicht überlappen und dass keine Löcher entstehen. Des Weiteren soll sich die Form des Nachbargebietes so wenig wie möglich verändern und topologische Beziehungen und Grenzen mit hoher Priorität erhalten bleiben. Haunert *et al.* [13] verwenden hierzu das Straight Skeleton. Das Straight Skeleton teilt das Innere eines

Polygons mit n Kanten in n monotone Polygone (siehe Kapitel 2.1.1). Jedes Face des Straight Skeleton wird durch genau eine Polygonkante begrenzt, wodurch eine eindeutige Zuordnung zu einem Nachbarn möglich ist. Wie weiter oben erwähnt, muss in allen Fällen sichergestellt werden, dass topologische Beziehungen sich nicht verändern und dass Grenzen mit hoher Priorität beibehalten werden. Als Beispiel definieren Haunert *et al.* einen Fluss, der in einem See mündet. Wird in diesem Fall das ungewichtete Straight Skeleton verwendet, geht die topologische Beziehung verloren, d.h. dass der Fluss nicht mehr mit dem See verbunden ist. Wird jedoch das Weighted Straight Skeleton eingesetzt, kann durch die Variation der Gewichte die topologische Beziehung aufrechterhalten bleiben. Im oben angeführten Beispiel würden alle Polygonkanten, welche den Fluss und den See miteinander verbinden, mit einem größeren Gewicht versehen werden.

2.2.4 Konstruktion von Dächern

Wie bereits in Kapitel 2.1.2 beschrieben, eignet sich das Straight Skeletons besonders gut zur Konstruktion von Dächern. Unter Verwendung des Straight Skeletons kann sichergestellt werden, dass das Regenwasser auch abrinnen kann. Die konvexen Kanten des $S(P)$ erzeugen die Dachfirste und konkave Kanten stellen die Dachkehlen dar. Wird statt dem ungewichteten Straight Skeleton die gewichtete Version verwendet, können unterschiedliche Steigungen der Dachflächen erreicht werden.

Im Jahr 2011 präsentierten Kelly *et al.* [18] ein Modellierungssystem für die Außenansicht von Architekturmodellen. Es ermöglicht die Konstruktion von komplexen Oberflächen, wie z.B. gebogene oder überhängende Dächer, Dach- und Erkerfenstern und Schornsteinen. Für die Konstruktion dieser Oberflächen wird ein Sweep-Algorithmus verwendet, welcher auf dem (gewichteten) Straight Skeleton basiert. Ziel des Algorithmus ist es, eine architektonische Hülle (Netz im $\mathbb{R}^3$) zu konstruieren. Als Input wird dem Algorithmus ein Grundriss übergeben. Dieser Grundriss stellt eine kreuzungsfreie Partitionierung der Ebene dar und besteht aus Ecken und Kanten. Da der Grundriss parallel zu einer xy-Ebene einbettet wird, haben alle Ecken die gleiche Höhe (z-Koordinate). Die Ränder des Grundriss bilden Polygone, die typischerweise gegen den Uhrzeigersinn orientiert sind. Einzige Ausnahme stellen Polygone dar, die ein Loch beschreiben. Diese müssen per Definition im Uhrzeigersinn orientiert sein. Zu jeder Kante existiert eine Richtungsebene, welche durch den Winkel θ definiert ist und die Kante beinhaltet. Der Winkel θ_i einer Richtungsebenen kann durch Polygonlinien, den sogenannten “Profilen”, kontrolliert werden. Der von Kelly *et al.* [18] vorgestellte Algorithmus verwendet eine Sweep-Ebene. Für jede Höhe, die die Sweep-Ebene annehmen kann, definiert sie einen Querschnitt des Gebäudes und

somit einen weiteren $2D$ Plan. Dieser wird als aktiver Plan bezeichnet. Die Kanten des Plans bewegen sich kollinear zur Überschneidung der Richtungsebene mit der Sweep-Ebene. Während der Bewegung der Kanten treten wie beim Straight Skeleton (siehe Kapitel 2.1.1) Events auf. Da bei Architekturmodellen sehr oft Ausnahmefälle auftreten, kann laut Kelly *et al.* die klassische Kategorisierung in Edge-, Split- und Knoten Events nicht vollständig übernommen werden. Aus diesem Grund definieren Kelly *et al.* vier neue Eventtypen:

1. General Intersection Event
2. Edge Direction Event
3. Profile Offset Event
4. Anchor Event

Der Hauptunterschied zu einem Straight Skeleton Algorithmus ist hierbei, dass alle Events, ausgenommen das General Intersection Event, vom Benutzer des Modellierungssystems ausgelöst werden. Gemeinsam haben die unterschiedlichen Eventtypen jedoch, dass jedes Event zur Folge hat, dass eine Änderung am aktiven Plan vorgenommen werden muss. Kelly *et al.* beschreiben hierzu für jeden Eventtyp die notwendigen Änderungen. Wie auch bei einigen in Kapitel 2.3 beschriebenen Algorithmen werden mögliche Events in einer Warteschlange nach ihrer Höhe sortiert gespeichert. Der Algorithmus terminiert, wenn sich kein Event mehr in der Warteschlange befindet.

2.2.5 Design von Pop-up Karten

Sugihara [23] verwendet das Weighted Straight Skeleton zur Konstruktion von Pop-up Karten. Wobei das Hauptaugenmerk auf jene Klassen von Pop-up Strukturen gelegt wird, die durch Schneiden und Falten eines einzelnen Blatt Papiers entstehen können. Als eine Falteinheit (engl. *folding unit*) wird ein Set von zwei Schnittlinien c_1, c_2 und fünf Faltlinien $b_1, b_2, e_1, e_2, g_{12}$ bezeichnet. Zusätzlich bilden die zwei Liniensegmente b_1 und b_2 die Basislinie. Die Winkel zwischen b_1 und e_1 sowie b_2 und e_2 werden als ϕ_1 und ϕ_2 bezeichnet. Die Winkeln zwischen den Faltkanten e_1, e_2 und g_{12} werden als θ_1 und θ_2 definiert. Es gilt:

$$\theta_1 + \theta_2 = \phi_1 + \phi_2.$$

Jackson [17] definiert eine Basiseigenschaft, die erforderlich für das Design von faltbare Strukturen ist.

Lemma 10. *Vorausgesetzt, dass die Faltlinien e_1, e_2 und g_{12} nicht parallel zu b_1 und b_2 sind, kann eine Falteinheit gefunden werden, die den Diederwinkel zwischen b_1 und b_2 auf null reduziert.*

Dies ist genau nur dann der Fall wenn folgende zwei Eigenschaften erfüllt werden:

1. Die Faltlinien können so erweitert werden, dass sie einen gemeinsamen Schnittpunkt haben.
2. Es gilt $\phi_1 = \theta_2$ und $\phi_2 = \theta_1$.

Diese zwei Eigenschaften werden erfüllt, wenn die Falteinheit sich zu einer ebenen Fläche falten lässt. Dies ist wiederum laut Sugihara [23] nur der Fall, wenn sich die Faltkante g_{12} am Bisektor der Kanten e_1 und e_2 befindet und die Kantengewichte $w_{e_1} = \sin\theta_2$ und $w_{e_2} = \sin\theta_1$ sind. Der Bisektor der Kanten e_1 und e_2 ist hierbei jene Kante, die für jedes Paar von Gewichten w_{e_1} und w_{e_2} durch den gemeinsamen Schnittpunkt der beiden Kanten geht.

Sugihara stellt einen Algorithmus vor, der unter Zuhilfenahme des ungewichteten und gewichteten Straight Skeletons die gewünschte Pop-up Struktur erstellt. Hierzu wird im ersten Schritt ein Polygon mit der gewünschten Form konstruiert. Danach wird das Straight Skeleton berechnet. Im nächsten Schritt wählt der Benutzer aus dem Set der Straight Skeleton Kanten jene Kanten g_i für $i = 1, 2, \ldots, n$ aus, die die Region zweier nicht adjazenten Polygonkanten e_i und e_i' verbinden. Für jedes dieser Paare wird der gemeinsame Schnittpunkt berechnet. Danach wird das Ausgangspolygon so umgeformt, dass die Schnittpunkte kollinear werden. Die daraus entstandene Linie stellt die Basislinie dar. Anhand der Basislinie und den Kantenpaaren e_i und e_i' werden die Falteinheiten konstruiert. Hierzu werden den Kantenpaaren, wie weiter oben beschrieben, folgende Kantengewichte zugeordnet:

$$\begin{aligned} w_{e_i} &= \sin\theta_i' \\ w_{e_i'} &= \sin\theta_i \end{aligned}$$

Im vierten Schritt wird das Weighted Straight Skeleton berechnet. Danach werden zu den vorher ausgewählten Faltlinien die dazugehörigen Schnittlinien ergänzt. Im letzten Schritt können je nach Bedarf weitere Details zur Struktur hinzugefügt werden.

2.3 Berechnung des Weighted Straight Skeletons

In diesem Kapitel werden einige Algorithmen zur Berechnung des Straight Skeletons vorgestellt. Jeder der hier beschriebenen Algorithmen eignet sich sowohl zur Berechnung des ungewichteten wie auch zur Berechnung des gewichteten Straight Skeletons. Falls vorhanden, werden Besonderheiten bei der Berechnung des Weighted Straight Skeletons hervorgehoben.

In weiterer Folge werden die Algorithmen in zwei Kategorien eingeteilt:

1. Wavefrontbasierte Algorithmen
2. Triangulierungsbasierte Algorithmen

Tabelle 2.1 zeigt eine Aufstellung der Laufzeiten und Speicherbedarf der einzelnen beschriebenen Algorithmen.

Algorithmus	**Zeit**	**Speicher**
Aichholzer *et al.* [2]	$\mathcal{O}(nr \log n)$	$\mathcal{O}(n)$
Felkel und Obdržálek [12]	$\mathcal{O}(nr + n \log n)$	$\mathcal{O}(n)$
Eppstein und Erickson [11]	$O(n^{8/5+\varepsilon})$	$O(n^{8/5+\varepsilon})$
Aichholzer und Aurenhammer [1]	$\mathcal{O}(n^3 \log n)$	$\mathcal{O}(n)$

Tabelle 2.1: Laufzeit und Speicherbedarf der Algorithmen

Aichholzer und Aurenhammer [1] geben an, dass für typische Inputgraphen eine Laufzeit von fast $\mathcal{O}(n \log n)$ erreicht werden kann. Bei der oben beschriebenen Laufzeit von Eppstein und Erickson [11] handelt es sich um eine theoretische Laufzeit, da dieser Algorithmus schwer zu implementieren ist. In der Praxis kann nur eine schlechtere Laufzeit von $\mathcal{O}(n \log n + nr)$ und Speicher von $\mathcal{O}(nr)$ erreicht werden. Aufgrund der besseren Laufzeit und der Nicht-Verwendung von komplexen Datenstrukturen, dient der Algorithmus von Aichholzer und Aurenhammer als Basis für den in Kapitel 3.2 implementierten Algorithmus.

2.3.1 Wavefrontbasierte Algorithmen

Aichholzer et al.

Aichholzer *et al.* [2] beschreiben einen Algorithmus, welche das Straight Skeleton für ein einfaches Polygon in $\mathcal{O}(nr \log n)$ Zeit,und $\mathcal{O}(n)$ Speicher berechnet. Der Faktor r repräsentiert die Anzahl der konkaven Knoten des Polygons P.

Handelt es sich beim Inputpolygon P um ein konvexes Polygon, treten ausschließlich Edge Events auf. Ein Edge Event tritt dann auf, wenn eine Kante e auf die Länge Null geschrumpft ist. Die Endpunkte der Kante verschmelzen zu einem Punkt, welcher eine Node im $S(P)$ darstellt. Ehemals benachbarte Kanten der Kante e sind nun adjazent. Die Endpunkte der Kante e bewegen sich während des Schrumpfprozesses entlang der Winkelhalbierenden. Laut Aichholzer *et al.* [2] lassen sich Edge Events einfach vorhersagen, da lediglich berechnet werden muss, wann sich die zwei Strahlen der beiden Punkte schneiden. Die Events werden in einer Warteschlange, sortiert nach dem Zeitpunkt ihres Auftretens, gespeichert. Nach einem Edge Event muss für die nun benachbarten Kanten die Zeit für ein mögliches Edge Event neu berechnet und die Warteschlange entsprechend aktualisiert werden. Da nur eine konstante Anzahl von Events in der Warteschlange aktualisiert werden muss, kann ein Edge Event in $\mathcal{O}(\log n)$ abgehandelt werden [15]. Für konvexe Polygone kann daher das Straight Skeleton in $\mathcal{O}(n \log n)$ berechnet werden.

Im Gegensatz dazu lassen sich Split Events lokal nicht so einfach feststellen. Ein Split Event tritt dann auf, wenn ein konkaver Knoten auf eine Kante e trifft. Um herauszufinden ob ein Knoten v zu einem bestimmten Zeitpunkt auf eine Kante e trifft, ist es nicht ausreichend, dass der Strahl ausgehend von v irgendwann auf e trifft, da sich diese Kante im Laufe des Schrumpfprozess von v wegbewegen oder vor dem berechneten Zeitpunkt verschwunden sein könnte. Bei dieser Vorgehensweise müsste während des Schrumpfprozess ständig überprüft werden, ob die initial berechneten Split Events noch Gültigkeit besitzen. Dies stellt sich laut Aichholzer *et al.* [2], bezogen auf Laufzeit und Speicher, als sehr kostspielig heraus, da sich eine Kante während des Prozesses $\mathcal{O}(n)$ Mal verändern und ein Polygon aus $\mathcal{O}(n)$ konkaven Knoten bestehen kann.

Aichholzer *et al.* [2] präsentieren eine Methode, wie Split Events effizient erkannt werden können. Nach jedem Edge Event wird das geschrumpfte Polygon $W_P(t)$ auf Selbstüberschneidungen untersucht. Treten welche auf, müssen zwischen dem Zeitpunkt t und dem Zeitpunkt t' des vorherigen Edge Events Split Events aufgetreten sein. Da das letzte Event immer ein Edge Event ist, können so alle Split Events

ermittelt werden. Diese können laut Aichholzer *et al.* [2] von der Struktur des $W_P(t)$ abgelesen werden. Beim ersten Split Event trifft ein Knoten v auf eine Polygonkante e, wodurch das Polygon in zwei Polygone geteilt wird. Diese zwei Teile werden durch ein Dreieck bestehend aus dem Punkt v und den zwei neuen Schneidungspunkten, der sogenannten Brücke verbunden. Treten mehrere Split Events bis zum Zeitpunkt t auf, wird der gesamte Prozess wiederholt, d.h. Polygonteile werden wiederum aufgeteilt und Brücken konstruiert. Eine Überlappung einzelner Polygonteile ist ausgeschlossen, da $W_P(t')$ sonst Überschneidungen hätte. Die Brücken werden durch eine baumartige Struktur verbunden. Diese Brücken können sich jedoch mit anderen Brücken oder Polygonteilen überlappen. Des Weiteren behalten sie immer ihre initiale Dreiecksform bei.

Um alle Split Events im Zeitintervall $[t, t']$ zu finden, müssen alle Brücken und somit alle Überschneidungspunkte gefunden werden. Der Zeitpunkt des Auftretens des Events kann dann anhand der Struktur der Brücke abgelesen werden. Wenn ein Standard Algorithmus für Schnitte von Liniensegmente [21] verwendet wird, können alle Schnittpunkte von $W_P(t)$ in $\mathcal{O}(n \log n)$ Zeit gefunden werden. Sei t das k-te Edge Event und wird zusätzlich noch eine Exponentielle Suche verwendet, dann sind $O(\log k)$ statt $O(k)$ Test notwendig, um alle Selbstüberschneidungen zu finden. Wird des Weiteren ein Algorithmus zur Polygontriangulierung, wie von Chazelle [10] beschrieben, angewandt, kann jeder dieser Tests $\mathcal{O}(n)$ Zeit durchgeführt und somit alle Überschneidungen in $\mathcal{O}(n \log k)$ Zeit ermittelt werden. Wie in diesem Kapitel erwähnt, können Edge Events in $\mathcal{O}(n \log n)$ Zeit behandelt werden, wodurch sich eine Gesamtlaufzeit für den Algorithmus von $\mathcal{O}(n^2 \log n)$ ergibt. Da jedes Split Event von einem konkaven Knoten ausgelöst wird, kann die oben beschrieben Gesamtlaufzeit noch verfeinert werden. Sei r die Anzahl der konkaven Knoten und $s \leq r$ die Anzahl der Split Events, dann berechnet der Algorithmus das Straight Skeleton in $\mathcal{O}(ns \log n)$ Zeit und $O(n)$ Speicher.

Felkel und Obdržálek

Aufbauend auf den Algorithmus von Aichholzer *et al.* [2] haben Felkel und Obdržálek [12] einen Algorithmus und dessen Implementierung entwickelt. Im Falle eines konvexen Polygons wird das Straight Skeleton gleich wie beim von Aichholzer *et al.* [2] entwickelten Algorithmus berechnet. Die Berechnung des Straight Skeletons für ein konkaves Polygon erfolgt jedoch auf andere Weise. Wie auch beim Algorithmus von Aichholzer *et al.* [2] werden die Events in einer Warteschlange gespeichert. Um ein Split Event zu ermitteln, werden im ersten Schritt die Koordinaten des Schnittpunktes i berechnet. Sei r der konkave Knoten, der das Split Event auslöst. Jeder Knoten,

welcher den gleichen Abstand zu einer gegenüberliegenden Polygonkante wie auch zu den zum Punkt r adjazenten Kanten hat, stellt hierbei einen möglichen Kandidaten dar. Um die gegenüberliegende Kante e zu finden, werden alle Polygonkanten auf eine mögliche Überschneidung mit dem von Knoten r ausgehenden Strahl getestet. Hierbei muss sichergestellt werden, dass der Schnittpunkt sich in der von der getesteten Kante und von Winkelhalbierenden der Endpunkte aufgespannten Region befindet. Aus der Liste der gefundenen Schnittpunkten wird dann jener ausgewählt, welcher die kürzeste Distanz zum Punkt r hat. Der Algorithmus berechnet das Straight Skeleton in $\mathcal{O}(nr + n \log n)$ Zeit und $\mathcal{O}(n)$ Speicher. Laut Huber [15] kann jedoch durch die oben beschriebene Vorgehensweise nicht immer sichergestellt werden, dass das richtige Split Event ermittelt wird.

Eppstein und Erickson

Eppstein und Erickson [11] entwickelten einen Algorithmus, der das ungewichtete und gewichtete Straight Skeleton in $\mathcal{O}(n^{8/5+\varepsilon})$ für jedes $\varepsilon > 0$ Zeit und Speicher berechnet. Wie auch bei den anderen Algorithmen wird das Straight Skeleton durch Anwendung eines Schrumpfprozesses konstruiert. Im Gegensatz zu den Algorithmen von Aichholzer *et al.* [2] und Felkel und Obdržálek [12] können neben Edge- und Split Events auch sogenannte Vertex Events auftreten. Diese treten dann auf, wenn zwei oder mehrere konkaven Knoten gleichzeitig den gleichen Punkt erreichen. Im Gegensatz zu einem Edge- oder Split Event kann durch ein Vertex Event ein neuer konkaver Knoten entstehen. Jedoch sinkt die Anzahl der konkaven Knoten laut Eppstein und Erickson während des Schrumpfprozesses. Eppstein und Erickson betrachten das Problem im $\mathbb{R}^3$ (siehe Kapitel 2.1.2). Jede Polygonkante formt ein Dreieck im $\mathbb{R}^3$ sowie jeder konkaver Knoten einen Strahl im $\mathbb{R}^3$. Zusätzlich setzen sie eine Sweep-Ebene ein, welche sich entlang der z-Achse im $\mathbb{R}^3$ bewegt [15]. Ein Edge Event tritt dann auf, wenn die Sweep-Ebene das obere Ende eines Dreieck erreicht hat. Ein Split- oder Vertex Event tritt dann auf, wenn die Sweep-Ebene einen Schnittpunkt eines Strahls mit einem Dreieck erreicht. Nach jedem Event muss die Datenstruktur aktualisiert werden. Dies kann mit einer konstante Anzahl von Einfügen und Löschen durchgeführt werden. Bei jedem Vertex Event mit k beteiligten konkaven Knoten werden $\mathcal{O}(k)$ Dreiecke und Strahlen gelöscht und eingefügt. Über die gesamte Laufzeit des Algorithmus werden $O(n)$ solcher Operationen durchgeführt. Edge Events werden sortiert nach ihren z-Koordinate in einer Warteschlange gespeichert, welche mit einer Gesamtzeit von $\mathcal{O}(n \log n)$ behandelt werden kann. Um das nächste Split- oder Vertex Event zu finden, verwenden Eppstein und Erickson eine spezielle Datenstruktur, welche es ermöglicht, die früheste Überschneidung von einem Set von Strahlen mit

einem Set von Dreiecken zu finden. Eppstein und Erickson gelang es dieses Problem auf folgende zwei Bereichssuchprobleme zu reduzieren:

1. Gegeben ist ein Set von Dreiecken im $\mathbb{R}^3$. Welches dieser Dreiecke wird zuerst von einem Strahl getroffen?
2. Gegeben ist ein Set von Strahlen im $\mathbb{R}^3$. Wo findet die früheste (minimalste z-Koordinate) Überschneidung eines beliebigen Strahls mit dem gegebenen Dreieck statt?

Die verwendete Datenstruktur setzt sich aus zwei anderen Datenstrukturen zusammen. Beide dieser Strukturen ermöglichen es die früheste Überschneidung in $O(n^{1+\varepsilon}/s^{1/4})$ abzufragen und dass das Einfügen und das Löschen von Elementen in $O(n^{1+\varepsilon}/n)$ Zeit durchgeführt werden kann. Somit ergibt sich eine Gesamtlaufzeit und Speicher von $\mathcal{O}(n^{8/5+\varepsilon})$. Wie am Anfang dieses Kapitels erwähnt, kann mit diesem Algorithmus auch die gewichtete Version des Straight Skeletons berechnet werden. Der Unterschied zur ungewichtete Version besteht nur darin, dass ein Edge Event mit zwei konkaven Knoten stattfinden kann und dass der daraus resultierende Knoten ein konkaver Knoten ist.

Aichholzer und Aurenhammer [1] geben an, dass ihr Algorithmus (siehe Kapitel 2.3.2) in der Praxis für "typische" Inputgraphen eine Laufzeit von nahezu $\mathcal{O}(n \log n)$ erreicht. Aufgrund der Verwendung von komplexen Algorithmen für die Bereichssuche erreicht der von Eppstein und Erickson [11] vorgestellte Algorithmus in der Praxis eine schlechtere Laufzeit als unter theoretischer Betrachtung. Dennoch kann unter Verwendung einer Quadtree-basierten Datenstruktur das (gewichtete) Straight Skeleton in $\mathcal{O}(n \log n + nr)$ Zeit und $\mathcal{O}(nr)$ Speicher konstruiert werden.

2.3.2 Triangulierungsbasierter Algorithmus

Aichholzer und Aurenhammer [1] beschreiben, dass das Straight Skeleton für einen PSLG G keine Voronoi-ähnliche Struktur besitzt. Die klassischen Methoden zur Berechnung des Voronoi-Diagrammes, wie z.B. inkrementales Einfügen oder Divide and Conquer, können daher nicht angewandt werden. Würde z.B. eine Kante e erst später hinzugefügt werden, würden mit hoher Wahrscheinlichkeit Teile des bereits berechneten Skeletons gelöscht oder rekonstruiert werden müssen, da die neu hinzugefügte Kante den gesamten Schrumpfprozess beeinflussen kann. Der Schrumpfprozess muss daher simultan für alle Kante durchgeführt werden. Aichholzer und Aurenhammer [1] präsentieren einen Algorithmus, welcher während des gesamten Schrumpfprozesses eine Triangulierung der von der Wavefront noch nicht erreichten Fläche aufrechterhält. Die Knoten der Triangulierung entsprechen den Knoten der

aktuellen Wavefront und bewegen sich entlang der Winkelhalbierenden ins Innere des Polygons. Die Besonderheit des Algorithmus ist es, dass jedes Edge- oder Split Event durch ein zusammengeklapptes Dreieck der Triangulierung angezeigt wird. Im ersten Schritt wird die initiale Wavefront für alle "Figures" des Graphen G erzeugt. Als Figures werden alle zusammenhängenden Komponenten des Graphen, wie z.B. Polygonkanten, einfache Polygone oder Liniensegmente, bezeichnet. Im Gegensatz dazu werden Knoten des Graphen mit Grad Eins als "Terminals"bezeichnet. Im nächsten Schritt werden die Wavefront Knoten von G trianguliert. Die neuen Kanten, die durch die Triangulierung entstanden sind, werden als "Spokes" bezeichnet. Diese müssen so gewählt werden, dass, nachdem der Schrumpfprozess gestartet wurde, jene Regionen, die bereits von der Wavefront erfasst wurden, nicht mehr trianguliert sind, ihr Komplement allerdings schon noch.

Lemma 11. *Sei G ein Graph mit n Knoten, wovon t Terminals sind. Dann besteht die initiale Triangulierung der Wavefront Knoten von G aus genau $2n + t - 2$ Dreiecken.*

Während des Schrumpfprozesses verändern die Dreiecke ihre Form und werden unter bestimmten Umständen zerstört. Der Zeitpunkt zu dem ein Dreieck verschwindet oder aktualisiert wird, wird als "collapsing time" bezeichnet und von drei Eventtypen ausgelöst:

1. **Flip Event:** Der Wavefront Knoten v trifft auf einen Dreieckskante s. Damit weiterhin eine gültige Triangulierung besteht, wird s entfernt und eine neue Kante t eingefügt.

2. **Edge Event:** Der Knoten v trifft auf einen anderen Wavefront Knoten, der gerade eine Kante e verloren hat. Es werden die zwei beteiligten Knoten identifiziert, um die Triangulierung zu aktualisieren. Danach wird die Kante e gelöscht.

3. **Split Event:** Der Knoten v ist auf eine Wavefront Kante e gestoßen. Die Kante e wurde hierbei in zwei Teile e' und e'' zerteilt. Die Triangulierung wird angepasst, indem der Knoten v dupliziert wird und die Kanten e' und e'' sowie alle Dreieckskanten, welche v beinhaltet haben, entsprechend aktualisiert werden. Danach wird die Kante e gelöscht.

Jedes Edge- oder Split Event erzeugt eine neue Node in $S(G)$. Der Algorithmus terminiert, wenn die Collapsing Time aller Dreiecke in der Warteschlange unendlich ist.

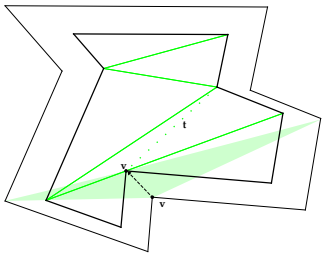

Abbildung 2.5: Flip Event

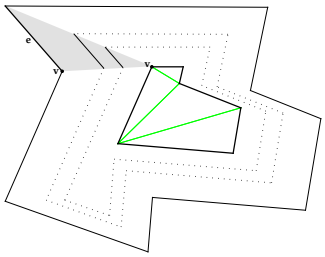

Abbildung 2.6: Edge Event

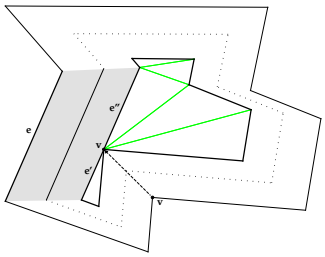

Abbildung 2.7: Split Event

Laufzeit- und Speicheranalyse
Aichholzer und Aurenhammer definieren eine obere Grenze für die Anzahl der Edge- und Split Events, welche gleichzeitig die Anzahl der Nodes im $S(G)$ repräsentiert.

Lemma 12. *Sei G ein Graph mit n Knoten und t Terminals, dann treten in Summe $2n + t - 2$ Edge- und Split Events auf.*

Beweis. Laut Lemma 11 besteht die initiale Triangulierung aus $2n + t - 2$ Dreiecken. Tritt ein Flip Event auf, bleibt die Anzahl der Dreiecke gleich. Bei einem Edge- oder Split Event verringert sich die Anzahl der Dreiecke um eins. Daraus ergibt sich, dass in Summe maximal $2n + t - 2$ Edge- und Split Events auftreten. □

Für die Laufzeitanalyse ist es noch wichtig, eine obere Grenze für die Anzahl der Flip Events zu definieren. Abhängig davon ob ein Wavefront Knoten, bezogen auf die noch nicht erreichte Region, lokal konvex ist oder nicht, unterscheidet man zwischen konvexen und konkaven Wavefront Knoten. Ein konvexer Knoten kann nie ein Flip Event auslösen, da die Dreieckskante "Spoke" in diesem Fall sonst einen bereits von der Wavefront besuchte Region schneiden würde. Da jene Region aber nicht trianguliert ist, kann dies niemals der Fall sein. Daraus folgt, dass Flip Events nur von konkaven Knoten ausgelöst werden. Ein einzelnes Flip Event braucht $\mathcal{O}(\log n)$ Zeit, da nur jene zwei Dreiecke aktualisiert werden müssen, die die Dreieckskante beinhalten [15]. Es können nicht mehr als $O(n^3)$ Flip Events auftreten, wie das folgende Lemma zeigt.

Lemma 13. *Die maximale Anzahl von Flip Events beträgt $\mathcal{O}(n^3)$.*

Beweis. Jeder Wavefront Knoten bewegt sich entlang einer geraden Linie mit konstanter Geschwindigkeit. Gegeben sind drei Punkte p, q, r eines Dreiecks $\triangle$ und $p(t), q(t), r(t)$ ihre Positionen zu einen bestimmten Zeitpunkt t. Des Weiteren sind v_p, v_q, v_r ihre Bewegungsvektoren. Das Dreieck klappt zusammen, wenn die Punkte p, q, r kollinear werden, d.h. wenn gilt:

$$\begin{vmatrix} p_x(t) & q_x(t) & r_x(t) \\ p_y(t) & q_y(t) & r_y(t) \\ 1 & 1 & 1 \end{vmatrix} = \begin{vmatrix} p_x(0) + t * v_{p,x} & q_x(0) + t * v_{q,x} & r_x(0) + t * v_{r,x} \\ p_y(0) + t * v_{p,y} & q_y(0) + t * v_{q,y} & r_y(0) + t * v_{r,y} \\ 1 & 1 & 1 \end{vmatrix} = 0$$

Die quadratische Gleichung wird von keinem, einem oder zwei Werten von t erfüllt. Daraus folgt, dass ein Dreieck maximal zweimal zusammenklappt und somit maximal zwei Flip Events pro Tripel auftreten können. Da es $\binom{n}{3}$ mögliche Tripel von Knoten gibt, ist die Anzahl der möglichen Flip Events durch $\mathcal{O}(n^3)$ begrenzt. □

Neben den Dreiecken müssen während dem Ausführen des Algorithmus auch die Bewegungsvektoren der Wavefront Knoten aktualisiert werden. Nach jedem Edge- oder Split Event ändert sich die Richtung und die Geschwindigkeit für die im Event beteiligten Knoten. Bei einem Flip Event findet keine Veränderung statt. Dies hat wiederum zur Folge, dass sich jener Zeitpunkt, zu dem ein Dreieck zusammenklappt, für jene Dreiecke verändert, die diesen Knoten beinhalten. Lemma 12 impliziert, dass der Knotengrad $\mathcal{O}(n)$ ist und daher $O(n^2)$ Dreiecke aktualisiert werden müssen. Somit können alle Edge- und Split Events in $\mathcal{O}(n^2 \log n)$ Zeit abgehandelt werden. Da laut Lemma 12 und Lemma 13 $\mathcal{O}(n^3)$ Dreiecke aktualisiert werden, erhält man somit eine Laufzeit von $\mathcal{O}(n^3 \log n)$. Lemma 11 definiert die Anzahl der Dreiecke und da maximal $2n + t - 2$ Dreiecke gespeichert werden müssen, ergibt sich ein Speicherbedarf von $\mathcal{O}(n)$. Die oben beschrieben Laufzeit ist schlechter als die Laufzeit des trivialen Algorithmus, der einfach für jeden Wavefront Knoten überprüft, ob ein Event mit einer der Wavefront Kanten stattfinden kann. Dieser weist eine Laufzeit von $\Theta(n^3)$ auf. In der Praxis kann laut Aichholzer und Aurenhammer [1] für typische Graphen aber eine Laufzeit von fast $\mathcal{O}(n \log n)$ erreicht werden. Also eine Laufzeit, die in etwa der Konstruktion der initialen Triangulierung entspricht.

Offen bleibt die Frage, ob die Anzahl der Flip Events $\mathcal{O}(n^2)$ beschränkt werden kann, um eine Gesamtlaufzeit von $\mathcal{O}(n^2 \log n)$ zu erreichen. 2010 zeigten Huber und Held [16], dass beim oben beschrieben Algorithmus von Aichholzer und Aurenhammer [1] für ein Polygon mit n Knoten $\Omega(n^2)$ Flip Events auftreten. Hierfür beschäftigten sie sich mit der Fragestellung, wie oft eine Diagonale der Triangulierung wieder

erscheinen kann. Wäre dies $\mathcal{O}(1)$ so könnte die Anzahl der Flip Events auf $\mathcal{O}(n^2)$ beschränkt werden, da $\binom{n}{2}$ Paare durch eine Diagonale miteinander verbunden sein können. Dies ist jedoch nicht der Fall, wie folgendes Lemma zeigt:

Lemma 14. *Es existieren Polygone P_n mit $\Theta(n)$ Knoten und Triangulierungen T_n, so dass $\Omega(n)$ Diagonalen von T_n $\Omega(n)$ Mal während des Schrumpfprozesses wieder erscheinen.*

Um das Lemma 14 zu beweisen, beschreiben Huber und Held [16] eine geometrische Konfiguration von sich bewegenden Punkten, welche eine Sequenz von Topologieänderungen auslösen, so dass die Diagonalen $\Omega(n)$ Male erscheinen. Zuerst wurde ein Polygon P und eine Triangulierung T konstruiert, so dass eine Diagonale zweimal erscheint. Danach wurde die Konstruktion erweitert, um zu zeigen, dass eine einzelne Diagonale $\Omega(n)$ Mal erscheint. Im letzten Teil des Beweises erweitern sie die Konstruktion so, dass $\Omega(n)$ Diagonalen $\Omega(n)$ Mal wieder entstehen.

In der beschriebenen Konstruktion verwenden Huber und Held [16] bewusst Triangulierungen, die zu $\Omega(n^2)$ Flip Events führen. Basis dieser Vorgehensweise war die Fragestellung, ob für ein Polygon P immer eine Triangulierung existiert, so dass die Anzahl der Flip Events auf $o(n^2)$ oder $\mathcal{O}(n)$ beschränkt werden kann. Auch hier konnten Huber und Held [16] beweisen, dass dies nicht immer der Fall ist.

3 Weighted Straight Skeleton - Tool

In diesen Kapitel wird die Implementierung eines Tools zur Berechnung des Weighted Straight Skeleton beschrieben. Im ersten Teil (siehe Kapitel 3.1) wird ein Überblick über die Benutzeroberfläche und deren Funktionen gegeben. Wobei auch ein kurzer Einblick in die Generierung von randomisierten Polygonen gegeben wird. In Kapitel 3.2 wird der verwendete Algorithmus näher beschrieben.

3.1 Benutzeroberfläche

Die Benutzeroberfläche des Weighted Straight Skeleton Tools wurde einfach und intuitiv gestaltet. Das Hauptfenster unterteilt sich in folgende Bereiche:

1. **Menüleiste:** Der Menüpunkt „File“ ermöglicht dem Benutzer, Polygone des Tools zu laden oder zu speichern. Um dem Benutzer das Erstellen des Polygons zu erleichtern, wurde ein weiteres Tool in das Weighted Straight Skeleton Tool integriert. Dieses erzeugt unter Verwendung von drei unterschiedlichen Algorithmen randomisierte Polygone und kann unter dem Menüpunkt „Random Polygon“ aufgerufen werden [22].

2. **Zeichenbereich** („Drawing area“)**:** Der Zeichenbereich stellt die zentrale Komponente des Tools dar. In diesem Bereich können Polygone erstellt und editiert werden. Auch die Kantengewichte können direkt dort verändert werden. Erlaubt sind nur Kantengewichte > 0.

3. **Bearbeitungsbereich:** In diesem Bereich wird eine Liste der berechneten Weighted Straight Skeletons anzeigt. Über eine Checkbox wird dem Benutzer ermöglicht, ein einzelnes Weighted Straight Skeleton für die Bearbeitung im Zeichenbereich auszuwählen. Ist kein Skeleton selektiert, wird automatisch beim Verschieben der Punkte oder Änderung der Kantengewichte ein neues Weighted Straight Skeleton erzeugt und der Liste hinzugefügt. Des Weiteren wird auch die Möglichkeit geboten einzelne Skeletons zu löschen und deren Anzeigefarbe

zu ändern. Um bei einer größeren Anzahl an Skeletons nicht den Überblick zu verlieren, verfügt das Tool über die Funktionalität, Skeletons temporär im Zeichenbereich zu deaktivieren.

4. **Action Panel:** Im unterem Bereich des Hauptfenster befindet sich das Action Panel. Hier werden die grundlegenden Funktionen „Play", „Step" und „Reset"des Tools angeboten. Eine Besonderheit bietet hier der „Step"-Button. Dieser erlaubt dem Benutzer Schritt-für-Schritt die Entstehung des Skeletons nachzuvollziehen.

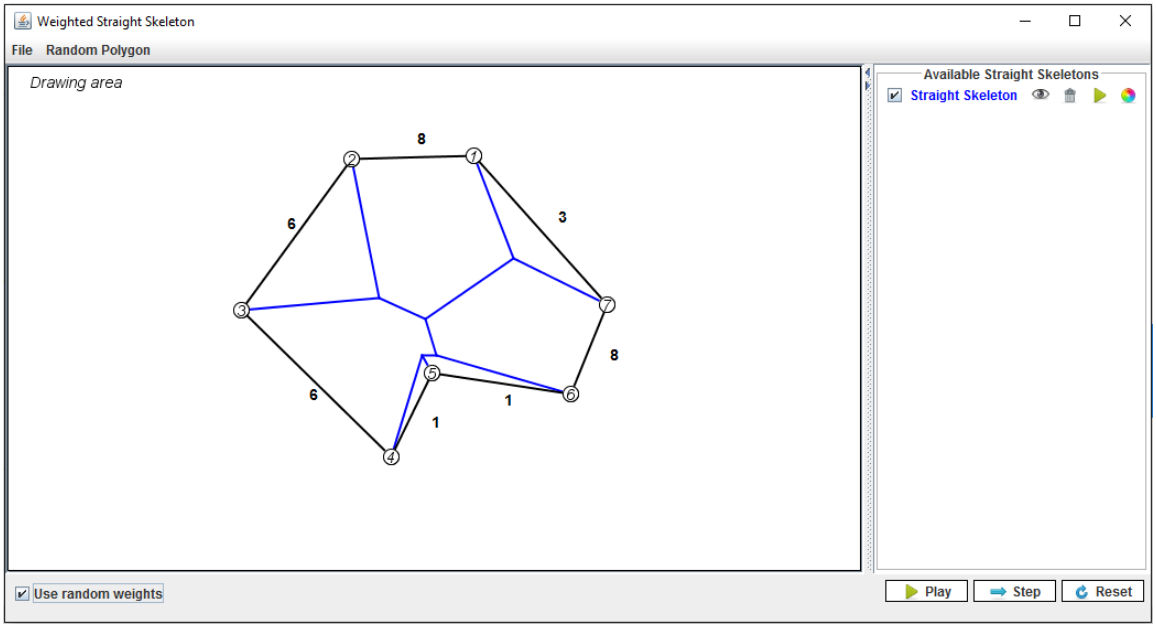

Abbildung 3.1: Benutzeroberfläche des Weighted Straight Skeleton Tools

3.1.1 Generierung von Polygonen

Das Programm bietet zwei Varianten, wie Polygone generiert werden können. Bei der ersten Variante kann das Polygon via Mausklick direkt im Zeichenbereich konstruiert werden. Die Orientierung des Polygons spielt hierbei keine Rolle; die Mausklicks können im oder gegen den Uhrzeigersinn erfolgen. Die Kantengewichte werden bei der Erstellung des Polygons zufällig vergeben und können bei Bedarf direkt im Zeichenbereich an der entsprechenden Kante verändert werden. Negative Kantengewichte sind nicht erlaubt. Sollten einzelne Punktpositionen nicht der Vorstellung des

Benutzers entsprechen, können diese im Nachhinein durch Verschieben der einzelnen Punkte verändert werden.

Bei der zweiten Variante handelt es sich um ein weiteres Programm, welches in das Weighted Straight Skeleton Tool integriert wurde. Wie schon am Anfang dieses Kapitels (3.1) erwähnt, kann es über die Menüleiste unter dem Punkt „Random Polygon"gestartet werden. Es ermöglicht dem Benutzer eine schnelle Generierung von randomisierten Polygonen mit einer beliebigen Punktmenge. Hierbei stehen dem Benutzer vier unterschiedliche Algorithmen zur Verfügung. Beim Schließen des Programmes wird das aktuell verfügbare Polygon automatisch in den Zeichenbereich des Weighted Straight Skeleton Tools übernommen und kann dort dann weiterverarbeitet werden.

Als Grundlage für die in Kapitel 4 beschriebenen Ergebnisse, wurden randomisiert generierte Polygone verwendet. Diese werden im folgenden Kapitel nun näher erläutert.

Randomisiert generierte Polygone

Problemstellung: Gegeben ist ein Set von Punkten in der Ebene. Wie müssen diese Punkte verbunden werden, so dass ein geschlossenes Polygon, bei dem sich die Polygonkanten nicht überschneiden, entsteht?

Es existieren einige Algorithmen, die dieses Problem lösen. In diesem Kapitel wird auf die vier, welche im Tool verwendet werden, genauer eingegangen [24].

1. **Random Two Peasants:** Bei diesem Algorithmus werden zuerst die Punkte mit dem niedrigsten oder höchsten x-Koordinate ermittelt. Als Alternative ist es auch möglich, zufällig zwei Punkte auszuwählen. Danach werden diese Punkte mit einer imaginären Linie verbunden, um die Punktmenge in eine untere und obere Hälfte einzuteilen. Im nächsten Schritt werden die Punkte nach ihren Winkel sortiert. Danach werden, ausgehend vom linken Endpunkt der imaginären Linie, die Punkte der oberen Hälfte miteinander verbunden bis der rechte Endpunkt erreicht wird. Analog dazu werden die Punkte der unteren Hälfte vom rechten zum linken Endpunkt miteinander verbunden. Die Laufzeit des Algorithmus ist $O(n \log n)$.

2. **Steady Growth:** Beim Steady Growth Algorithmus wird das Polygon durch Hinzufügen von Punkten zu einem Startpolygon generiert. Im ersten Schritt wird hierzu das Startpolygon mit drei Punkten erstellt. Die Punkte werden

so gewählt, dass sich in ihrer Konvexen Hülle kein anderer Punkt befindet. Danach werden fortlaufend Punkte zum Polygon hinzugefügt. Auch hier wird sichergestellt, dass sich kein weiterer Punkt in der Konvexen Hülle des erweiterten Polygons befindet. Der Algorithmus terminiert, wenn alle Punkte des Sets hinzugefügt wurden. Die Laufzeit des Algorithmus beträgt $O(n^2)$.

3. **Two Opt Moves:** Bei diesem Algorithmus wird das Polygon durch Entfernen von sich überschneidenden Kanten erstellt. Als Input für diesen Algorithmus wird ein zufällig erstelltes Polygon verwendet. Dieses Polygon muss nicht kreuzungsfrei sein. Beim Two Opt Moves Algorithmus wird nach sich kreuzenden Kanten gesucht und diese durch Vertauschen der Endknoten aufgelöst, so dass ein kreuzungsfreies Polygon entsteht. Im Worst Case müssen $O(n^3)$ solcher Vertauschungen durchgeführt werden, wodurch sich eine Laufzeit von $O(n^4)$ ergibt.

4. **Convex Layers:** Wie auch beim Random Two Peasants Algorithmus wird hier die Punktmenge in zwei Hälften geteilt. Hierzu werden zwei Punkte ausgewählt und diese mit einer imaginären Linie verbunden. Beide Hälften werden getrennt voneinander betrachtet und am Schluss zu einem Polygon zusammengefügt. Für jede Hälfte wird zuerst die Konvexe Hülle berechnet. Jene Punkte, die nicht darauf liegen, werden danach entsprechend mit denen auf der Konvexen Hülle verbunden. Abschließend werden beide Polygone miteinander verbunden. Das randomisierte Polygon kann mit diesem Algorithmus in $O(n \log n)$ erstellt werden.

Abbildung 3.2 zeigt den Output der vier Algorithmen für eine Punktmenge von 20 Punkten.

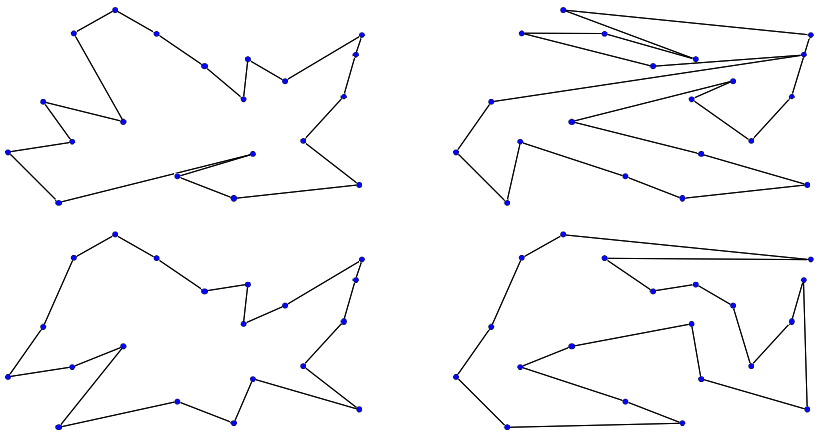

Abbildung 3.2: Randomisiert generierte Polygone

3.1.2 Speichern und Laden von Polygonen

Um erstellte Polygone wiederverwenden zu können, bietet das Tool eine Speicher- und Lademöglichkeit an. Beim Speichern wird über die Liste der vorhandenen Kanten iteriert und für jede Kante folgende Information, getrennt durch Bindestriche, in eine Textdatei gespeichert:

1. Anfang- und Endpunkt der Kante: Für den Anfangs- und Endpunkt der Kante wird die Punktnummer („ID“) sowie die Koordinaten „x“ und „y“, getrennt durch Semikolons, gespeichert.
2. Kantengewicht

Ein Beispiel des Ausgabeformats „Punkt - Punkt - Kantengewicht“:

$$
\begin{aligned}
&1;523.0;114.0-2;193.0;104.0-3\\
&2;193.0;104.0-3;339.0;169.0-9\\
&3;339.0;169.0-4;230.0;289.0-5\\
&4;230.0;289.0-5;445.0;409.0-8\\
&5;445.0;409.0-6;433.0;306.0-5\\
&6;433.0;306.0-7;502.0;251.0-9\\
&7;502.0;251.0-1;523.0;114.0-4
\end{aligned}
$$

Neben dem oben angeführten Dateiformat ist es auch möglich, Polygone mit dem Dateiformat des Random Polygon Programms zu laden. Der Benutzer kann hierzu, unabhängig vom Dateiformat, die Funktion „Open“ aus dem File-Menü verwenden.

3.2 Algorithmus

Die Implementierung basiert auf dem Algorithmus von Aichholzer *et al.* [1]. Wie in Kapitel 2.3.2 beschrieben, bewegen sich bei diesem Algorithmus die Kanten des Polygons mit unterschiedlicher Geschwindigkeit parallel zur ihrer Ursprungskante ins Innere des Polygons. Aufgrund der unterschiedlichen Kantengewichte bewegen sich die Punkte nicht entlang der Winkelhalbierenden. Die Bewegungsvektoren der Punkte werden daher über die Berechnung des Schnittpunktes der adjazenten Kanten ermittelt (siehe Kapitel 3.2.2). Während des gesamten Schrumpfprozesses wird eine Triangulierung aufrechterhalten. Die initiale Triangulierung wird unter Verwendung von zwei Algorithmen (siehe Kapitel 3.2.1) konstruiert. Im Laufe des Schrumpfprozesses treten Edge-, Split- und Flip Events auf. Diese treten dann auf, wenn die Punkte eines Dreiecks der Triangulierung kollinear werden (siehe Kapitel 3.2.3). Die berechneten Events werden sortiert nach dem Zeitpunkt ihres Auftretens in einer Warteschlange gespeichert. Befindet sich kein Event mehr in der Warteschlange, terminiert der Algorithmus.

Algorithmus 1 Weighted Straight Skeleton Algorithmus

Initialisierung
Trianguliere das Polygon P
for jeden Punkte $p \in P$ **do**
    Berechne Bewegungsvektor
Berechne initiale Events und speichere diese in die Warteschlange Q

while $\exists$ Event $\in Q$: Collapsing Time $\neq \infty$ **do**
    $ev :=$ Event in Q mit kleinster Zeit
    Entferne ev aus Q
    if $ev ==$ Edge Event $\vee$ $ev ==$ Split Event **then**
        Entferne Dreieck aus der Liste der verfügbaren Dreiecke
    Aktualisiere Triangulierung abhängig vom Eventtyp
    Berechne Bewegungsvektor für alle neuen Knoten
    Berechne Events und aktualisiere Warteschlange Q
    if $ev ==$ Edge Event $\vee$ $ev ==$ Split Event **then**
        Füge neuen Knoten in das Weighted Straight Skeleton ein

3.2.1 Triangulierung

Als ersten Schritt wird das Polygon trianguliert. Abhängig davon ob es sich um konvexes oder konkaves Polygon handelt, werden unterschiedliche Triangulierungsalgorithmen angewandt. Handelt es sich um ein konvexes Polygon, wird eine Fächertriangulierung (siehe Algorithmus 2) verwendet. Im Falle eines konkaven Polygon wird die Triangulierung mittels Sweep-Line-Algorithmus (siehe Algorithmus 3.2.1) ermittelt.

Konvexes Polygon - Fächertriangulierung

Bei der Fächertriangulierung wird ein beliebiger Punkt des Polygons mit allen nicht benachbarten Punkten verbunden. Laufzeit des Algorithmus ist $O(n)$.

Algorithmus 2 Fächertriangulierung

1: $n :=$ Anzahl der Punkte des Polygons
2: Erstelle $\triangle(p_1, p_{n-1}, p_2)$ und füge es zur Triangulierung hinzu
3: **for** $i = n - 1$ to $i > 2$ **do**
4: Erstelle $\triangle(p_{i-1}, p_i, p_2)$
5: Füge $\triangle$ zur Triangulierung hinzu
6: $i = i - 1$

Konkaves Polygon - Sweep-Line-Algorithmus

Für die Berechnung der Triangulierung von konkaven Polygonen wurde die Poly2tri Bibliothek [20] verwendet. Wie in Kapitel 3.2.1 beschrieben können konvexe Polygone sehr einfach und effizient trianguliert werden. Neben den konvexen können jedoch auch monotone Polygone (siehe Definition 2 in Kapitel 2) in linearer Zeit trianguliert werden. Poly2tri verwendet einen Sweep-Line-Algorithmus, der im ersten Schritt ein konkaves Polygon in monotone Polygone zerlegt und diese dann trianguliert. Ein Knoten p liegt unterhalb eines anderen Knotens q, wenn $p_y < q_y$ oder $p_y = p_y$ und $p_x > q_x$ gilt. Im Gegensatz dazu ist ein Knoten p oberhalb eines Knotens q, wenn $p_y > q_y$ oder $p_y = p_y$ und $p_x < q_x$ gilt.

Es werden fünf Typen von Polygonknoten unterschieden:

1. **Startknoten:** Ein Knoten v ist ein Startknoten, wenn beide Nachbarknoten unterhalb liegen und der innere Winkel am Knoten v kleiner als 180° ist.
2. **Endknoten:** Als Endknoten wird ein Knoten v bezeichnet, wenn beide Nachbarknoten oberhalb liegen und der innere Winkel am Knoten v kleiner als 180° ist.
3. **Trennknoten:** Liegen beide Nachbarknoten unterhalb von einem Knoten v und ist der innere Winkel bei v größer als 180°, so wird der Knoten v als Trennknoten bezeichnet.
4. **Mergeknoten:** Im Gegensatz zum Trennknoten liegen beide Nachbarknoten oberhalb vom Knoten v. Der innere Winkel bei v ist aber auch hier größer als 180°.
5. **regulärer Knoten:** Als reguläre Knoten werden alle Knoten bezeichnet, die weder Start-, End, Trenn- oder Mergeknoten sind.

Ein Polygon ist y-monoton, wenn es keine Trenn- oder Mergeknoten besitzt. Ziel des Algorithmus ist es daher, alle Knoten dieser zwei Typen durch das Einfügen von Diagonalen zu eliminieren (siehe Algorithmus 3.2.1).

Sei n die Anzahl der Knoten des Polygons P. Die Warteschlange Q kann in $\mathcal{O}(n \log n)$ initialisiert werden. Für jedes Element $\in Q$ wird eine Methode aufgerufen. In dieser Methode werden Operationen in der Warteschlange und Splay-Baum durchgeführt. Einfügen, Löschen und Suchen in einem Splay-Baum sowie in einer Warteschlange ist in $\mathcal{O}(\log n)$ Zeit möglich. Daher ergibt sich für die Zerlegung des Polygons in monotone Polygone eine Laufzeit von $\mathcal{O}(n \log n)$. Da ein monotones Polygon in $\mathcal{O}(n)$ Zeit trianguliert werden kann, ergibt sich eine Gesamtlaufzeit von $\mathcal{O}(n \log n)$.

Algorithmus 3 Sweep-Line-Algorithmus

Input: Array von Polygonknoten sortiert gegen den Uhrzeigersinn
Output: Triangulierung des Polygons P

Zerlege das Polygon P in monotone Polygonteile

1: Für jeden Polygonknoten ermittle den Knotentyp
2: Speichere Polygonknoten in eine Warteschlange Q sortiert nach ihren y-Koordinaten oder bei gleichen y-Koordinaten nach ihren x-Koordinaten
3: Initialisiere einen leeren Splay-Baum
4: **while** $Q \neq 0$ **do**
5: Entferne Knoten mit der höchsten Priorität von Q
6: Abhängig von Polygonknoten, rufe die entsprechende Methode auf und aktualisiere den Splay-Baum

Trianguliere die monotonen Polygonteile

7: m = Anzahl der monotonen Polygonteile
8: **for** $i = 0$ to $i < m$ **do**
9: Trianguliere monotones Polygon i

3.2.2 Berechnung Bewegungsvektor

Neben dem Aufbau der initialen Triangulierung muss im Initialisierungsschritt auch der Bewegungsvektor für jeden Polygonknoten ermittelt werden. Aufgrund der Gewichtung der Kanten bewegen sich die Punkte des Polygons nicht entlang der Winkelhalbierenden. Der Bewegungsvektor muss daher auf andere Weise berechnet werden.

Der Bewegungsvektor $\mathbf{v_x}$ für einen Punkt $\mathbf{p_x}$ wird über die Berechnung des Schnittpunktes der adjazenten Kanten e_1 und e_2 ermittelt.

Definition:
$\mathbf{v_x}$: Bewegungsvektor des Punktes $\mathbf{p_x}$
$\mathbf{l_1}$: Vektor der Kante e_1
$\mathbf{l_2}$: Vektor der Kante e_2
$\mathbf{n_1}$: gewichteter Normalvektor der Kante e_1
$\mathbf{n_2}$: gewichteter Normalvektor der Kante e_2

Gesucht ist der Schnittpunkt in der Form $\mathbf{s} = \mathbf{p_x} + (\mathbf{n_1} + u\mathbf{l_1})$.

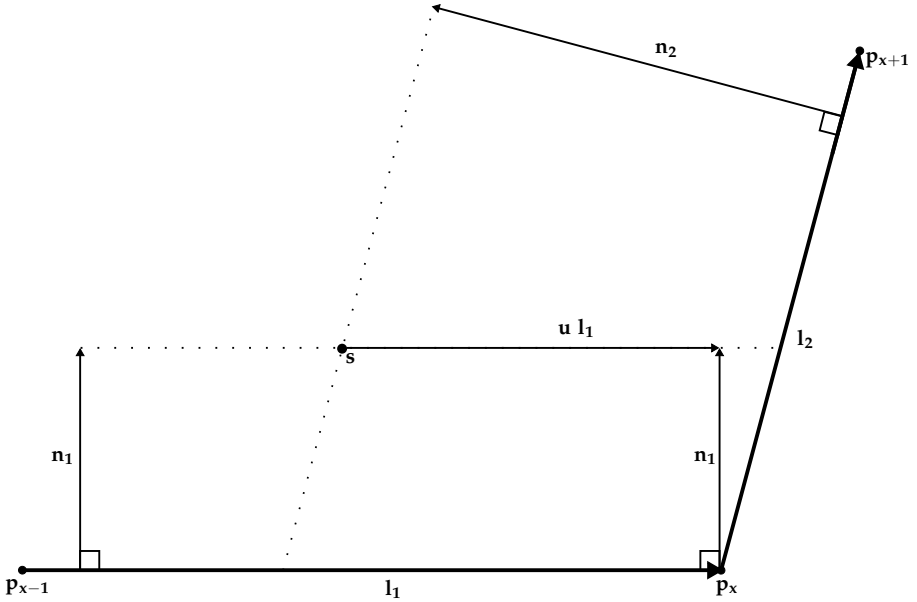

Abbildung 3.3: Schnittpunkt s der adjazenten Kanten e_1 und e_2

Sei $\mathbf{a}$ der Fußpunkt des Schnittpunktes $\mathbf{s}$ auf der Kante e_1. Der Abstand des Punktes $\mathbf{a}$ vom Punkt $\mathbf{p_x}$ ist gleich der Unbekannten u.

Seien $P_{\mathbf{ap_xp_{x+1}c}}$ und $P_{\mathbf{p_{x-1}p_xp_{x+1}q}}$ zwei Parallelogramme mit den gemeinsamen Seiten $\mathbf{l_2}$ und $u\mathbf{l_1}$ bzw. $\mathbf{l_1}$. Die beiden Punkte $\mathbf{c}$ und $\mathbf{q}$ können mithilfe der Punkte $\mathbf{a}$ und $\mathbf{p_{x-1}}$ und dem Vektor $\mathbf{l_2}$ ermittelt werden.

$$\begin{aligned} \mathbf{q} &= \mathbf{p_{x-1}} + \mathbf{l_2}, \\ \mathbf{c} &= \mathbf{a} + \mathbf{l_2}. \end{aligned}$$

Das Seitenverhältnis der Parallelogramme beträgt $1 : u$, wodurch sich der unbekannte Faktor u aus dem Verhältnis der Flächeninhalte beider Parallelogramme ableiten lässt:

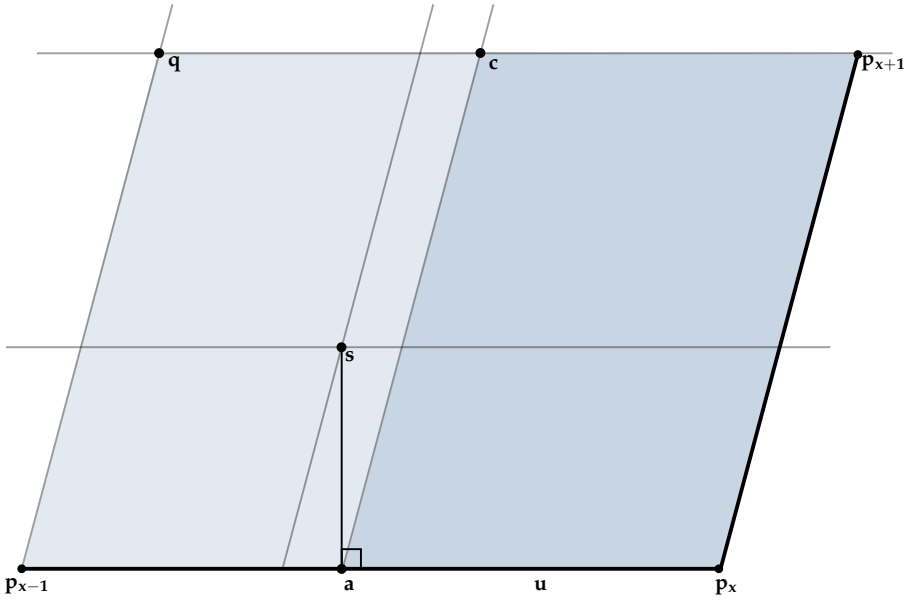

Abbildung 3.4: Berechnung des Parameters u aus dem Verhältnis der Flächeninhalte der Parallelogramme $P_{\mathbf{ap_xp_{x+1}c}}$ und $P_{\mathbf{p_{x-1}p_xp_{x+1}q}}$.

$$u = \frac{\|P_{\mathbf{ap_xp_{x+1}c}}\|}{\|P_{\mathbf{p_{x-1}p_xp_{x+1}q}}\|}.$$

Der Flächeninhalt des Polygons $\|P_{\mathbf{p_{x-1}p_xp_{x+1}q}}\|$ kann mit den Vektoren $\mathbf{l_1}$ und $\mathbf{l_2}$ ermittelt werden.

$$\|P_{\mathbf{p_{x-1}p_xp_{x+1}q}}\| = det(\mathbf{l_2}, \mathbf{l_1})$$

Weiterhin unbekannt bleibt der Flächeninhalt des Parallelogramms $P_{\mathbf{ap_xp_{x+1}c}} = det(\mathbf{l_2}, u\mathbf{l_1})$.

Es wird ein weiteres Parallelogramm $P_{\mathbf{dp_xp_{x+1}b}}$ mit den bekannten Größen

$$\begin{aligned} \mathbf{b} &= \mathbf{p_{x+1}} + \mathbf{n_2} - \mathbf{n_1} \in g, \\ \mathbf{d} &= \mathbf{p_x} + \mathbf{n_2} - \mathbf{n_1} \in g \end{aligned}$$

konstruiert.

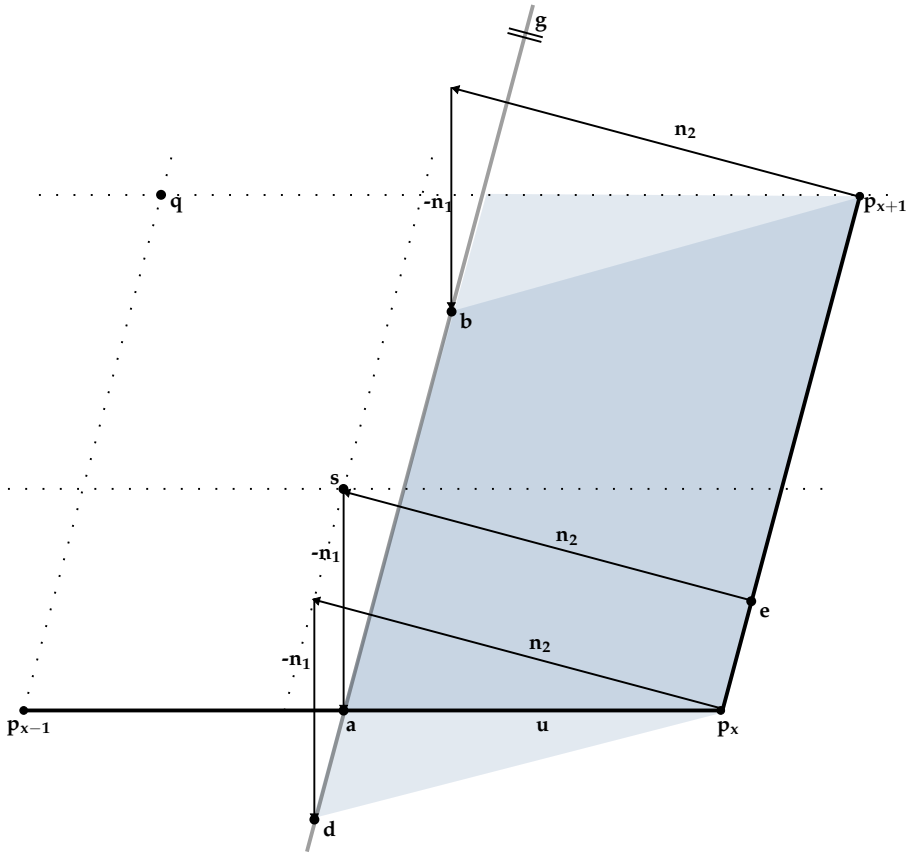

Abbildung 3.5: Konstruktion des Parallelogramms $P_{\mathbf{dp_xp_{x+1}b}}$

Annahme: $\|P_{\mathbf{dp_xp_{x+1}b}}\| = \|P_{\mathbf{ap_xp_{x+1}c}}\|$

Beweis. Sei g eine Gerade durch den Fußpunkt $\mathbf{a}$ mit Abstand $\mathbf{n_2} - \mathbf{n_1}$ von $\mathbf{l_2}$. Per Definition beträgt der Abstand des Punktes $\mathbf{a}$ zum Schnittpunkt $\mathbf{s}$ $\|\mathbf{n_1}\|$. Sowie $\|\mathbf{n_2}\|$ auf e_2. Daraus folgt, dass ein Punkt $\mathbf{e}$ auf der Kante e_2 existiert, so dass $\mathbf{a} = \mathbf{e} + \mathbf{n_2} - \mathbf{n_1}$. Somit ist $\mathbf{e}$ der Fußpunkt des Schnittpunktes $\mathbf{s}$ auf die Kante e_2. Per Definition beträgt der Abstand des Fußpunktes $\mathbf{e}$ zum Schnittpunkt $\|\mathbf{n_2}\|$. Hieraus folgt:

$$\mathbf{a} \quad = \quad \mathbf{e} + \mathbf{n_2} - \mathbf{n_1} \in g.$$

□

Da beide Parallelogramme dieselbe Grundlinie $\mathbf{l_2}$ und dieselbe Höhe u besitzen, folgt $det(\mathbf{l_2}, u\mathbf{l_1}) = det(\mathbf{l_2}, \mathbf{n_2} - \mathbf{n_1})$ und des Weiteren:

$$\begin{aligned} \|P_{\mathbf{dp_x p_{x+1} b}}\| &= \|P_{\mathbf{ap_x p_{x+1} c}}\| = det(\mathbf{l_2}, \mathbf{n_2} - \mathbf{n_1}), \\ u &= \frac{det(\mathbf{l_2}, \mathbf{l_1})}{det(\mathbf{l_2}, \mathbf{n_2} - \mathbf{n_1})}, \\ v &= \mathbf{n_1} + u\mathbf{l_1}. \end{aligned}$$

Die Berechnung des Bewegungsvektors kann in konstanter Zeit für einen Punkt durchgeführt werden. Der Bewegungsvektor muss am Anfang für jeden Punkt berechnet werden und auch während des Ausführung kann unter bestimmten Umständen, nach z.B. einem Edge- oder Split Event, eine Neuberechnung des Bewegungsvektor der beteiligten Knoten notwendig sein. In Summe ergibt sich somit eine Gesamtlaufzeit von $\mathcal{O}(n)$ für diesen Schritt.

3.2.3 Berechnung Events

Wie in Kapitel 2.3.2 erläutert, wird jedes Event durch ein zusammengeklapptes Dreieck ausgelöst. Ein Dreieck klappt dann zusammen, wenn die drei Punkte p, q, r des Dreiecks kollinear werden (siehe Beweis 13). Gegeben sind die drei Dreieckspunkte und deren Bewegungsvektoren v_p, v_q, v_r. Die Positionen der Punkte zu einen bestimmten Zeitpunkt t werden als $p(t), q(t), r(t)$ bezeichnet. Die drei Punkte sind kollinear, wenn gilt:

$$\begin{vmatrix} p_x(t) & q_x(t) & r_x(t) \\ p_y(t) & q_y(t) & r_y(t) \\ 1 & 1 & 1 \end{vmatrix} = \begin{vmatrix} p_x(0) + t * v_{p,x} & q_x(0) + t * v_{q,x} & r_x(0) + t * v_{r,x} \\ p_y(0) + t * v_{p,y} & q_y(0) + t * v_{q,y} & r_y(0) + t * v_{r,y} \\ 1 & 1 & 1 \end{vmatrix} = 0$$

Die Gleichung wird nach t aufgelöst. Es entsteht eine quadratische Gleichung, die eine, zwei oder keine Lösung/en für t liefert. Eine Lösung ist gültig, wenn gilt $t \geq 0$.

Existiert eine gültige Lösung für t für das Dreieck $\triangle_{p,q,r}$ muss noch herausgefunden werden, ob es sich um eine Edge-, Split- oder Flip Event handelt. Ein Edge Event tritt dann auf, wenn die zwei Endpunkte $\mathbf{p_1}, \mathbf{p_2}$ einer Kante e zu einem gemeinsamen Punkt verschmelzen, oder äquivalent dazu die Kantenlänge auf Null geschrumpft ist. Seien p, q die Endpunkte einer Polygonkante, dann tritt ein Edge Event auf, wenn gilt:

$$\begin{aligned} p_x(0) + t * v_{p,x} &= q_x(0) + t * v_{q,x}, \\ p_y(0) + t * v_{p,y} &= q_y(0) + t * v_{q,y}. \end{aligned}$$

Bei Split- und Flip Events trifft ein konkaver Knoten auf eine gegenüberliegende Kante. Die gegenüberliegende Kante kann entweder eine Polygon- oder Dreieckskante sein.

Sei der Punkt r der konkave Knoten und p, q die Endpunkte einer Polygon- oder Dreieckskante (Spoke). Da die drei Punkte kollinear sind, ist sichergestellt, dass sie auf einer Linie liegen. Überprüft werden muss daher nur noch, ob $r(t)$ zwischen $p(t)$ und $q(t)$ liegt. Ist dies der Fall, handelt es sich um ein Split- oder Flip Event.

3.2.4 Behandlung von Sonderfällen

Kollinearität von benachbarten Kanten nach Edge Event

Sind nach einem Edge Event die benachbarten Kanten mit unterschiedlichen Gewichten kollinear, so ist die Definition des Weighted Straight Skeleton nicht eindeutig. Laut Biedl *et al.* [6] gibt es verschiedene Methoden dieses Problem zu beheben. Eine Möglichkeit wäre, diesen Fall gleich wie den eindeutig definierten zu behandeln, d.h. beide Kanten werden weiterhin ins Innere des Polygons verschoben. Dies hat jedoch zur Folge, dass während dem Schrumpfprozesses ein immer größer werdendes Loch zwischen den zwei Kanten e_1 und e_2 entsteht. Um dies zu verhindern, könnte laut Biedl *et al.* [6], eine weitere Kante als Verbindung zwischen den beiden Kanten e_1 und e_2 eingefügt werden. Bei genauerer Betrachtung stellt sich diese Vorgehensweise als nicht sehr ratsam heraus, da eine neue Kante weder eine Verbindung zu einer Inputkante hat, noch parallel zu einer verlaufen muss. Abhängig von der Definition dieser neuen Kante, würde in den meisten Fälle ein Kantengewicht von Null gewählt werden, was wiederum zu einem degenerierten Face führen würde. Eine bessere Variante laut Biedl *et al.* [6] ist es, das Schrumpfen einer Kante zugunsten der anderen aufzugeben. Ob hierbei das Schrumpfen der langsameren oder die schnelleren gestoppt wird, spielt keine Rolle, solange bei jedem Auftreten die gleiche Option gewählt wird.

Aufgrund der angeführten, negativen Aspekte von Variante 1 wurde beschlossen, die zweite Variante, wie von Biedl *et al.* [6] beschrieben, zu implementieren. Für den in dieser Arbeit implementierten Algorithmus wurde entschieden, die schnellere Kante zu behalten und die langsamere aufzugeben. Seien e_1 und e_2 jene kollinearen und benachbarten Kanten und sei e_1 die Kante mit dem kleinerem Gewicht. Füge e_1 zum Weighted Straight Skeleton hinzu und vergrößere e_2, so dass die Länge von e_2 der Länge von $e_1 + e_2$ entspricht. Die Abbildung 3.6 zeigt ein Polygon und sein Weighted Straight Skeleton. Die Kante $(2, 3)$ wurde hier nach einem Edge Event der Kante $(3, 4)$ zugunsten von Kante $(4, 5)$ aufgegeben.

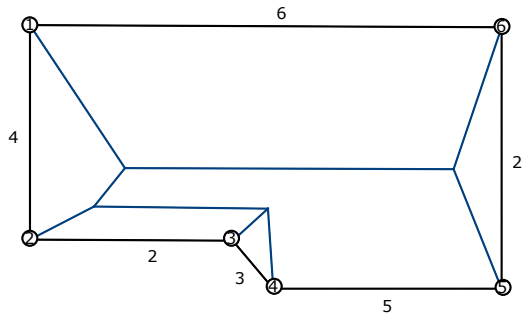

Abbildung 3.6: Kollineare benachbarte Kanten nach Edge Event

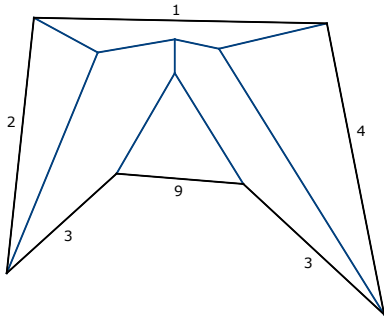

Abbildung 3.7: Konkaver Knoten nach Edge Event

Konkaver Knoten nach Edge Event

Da sich die Kanten des Polygons mit unterschiedlicher Geschwindigkeit ins Innere des Polygons bewegen, kann es nach einem Edge Event vorkommen, dass der aus diesem resultierende Knoten konkav ist (siehe Abbildung 3.7). In diesem Fall reicht es nicht aus, nur die sich aktuell in der Warteschlange befindlichen Events zu aktualisieren, da potentiell auch neue Split- und Flip Events auftreten können.

Wie schon in Kapitel 2.3.2 beschrieben, ermöglicht der triangulierungsbasierte Algorithmus das Weighted Straight Skeleton in $\mathcal{O}(n^3 \log n)$ Zeit und $\mathcal{O}(n)$ Speicher zu berechnen. Die Berechnung der initialen Triangulierung und der Bewegungsvektoren kann in $\mathcal{O}(n^3 \log n)$ durchgeführt werden. Die von Aichholzer und Aurenhammer [1] beschriebene bessere Laufzeit von $\mathcal{O}(n \log n)$ für typische Inputgraphen konnte mit der vorliegenden Implementierung nicht erreicht werden (siehe Kapitel 4.1).

4 Ergebnisse

Im Rahmen der Arbeit werden konkret drei Fragestellungen untersucht. Fokus liegt dabei darauf, die Unterschiede zwischen der gewichteten und ungewichteten Version des Straight Skeletons aufzuzeigen. Zur Berechnung des Weighted Straight Skeletons und zur Generierung der Testdaten wurde das in Kapitel 3 implementierte Tool verwendet.

Folgende Problemstellungen werden untersucht:

1. Welche Auswirkung hat das Kantengewicht auf die Verteilung des Polygonflächeninhaltes auf die Faces?
2. Besteht ein linearer Zusammenhang zwischen Kantengewicht und Flächeninhalt der Faces?
3. Besteht ein linearer Zusammenhang zwischen Kantenanzahl und Flächeninhalt der Faces?

Im Rahmen dieser Arbeit wurde eine Laufzeitmessung des implementierten Algorithmus durchgeführt. Die Ergebnisse dieser Messung werden in Kapitel 4.1 beschrieben. Bei jeder der oben angeführten Problemstellung wurden konvexe und konkave Polygone untersucht. Hierzu wurden Testdaten generiert, welche im Kapitels 4.2 beschrieben werden. Danach wird der Fokus dieses Kapitels auf die oben beschriebenen Problemstellung gelegt (siehe Kapitel 4.3, 4.4 und 4.5).

4.1 Laufzeitanalyse

Die Implementierung des Algorithmus (siehe Kapitel 3.2) weist für randomisiert erstellte Polygone eine Laufzeit von $\mathcal{O}(n^2)$ auf und ist damit schlechter als die von Aichholzer *et al.* [1] beschriebene Laufzeit von $\mathcal{O}(n \log n)$ für typische Inputgraphen. Tabelle 4.2 zeigt die Laufzeiten für konkave Polygone mit unterschiedlicher Knotenanzahl. Deutlich zu sehen ist, dass der Initialisierungsschritt sehr viel weniger Zeit als die eigentliche Berechnung des Weighted Straight Skeletons in Anspruch nimmt.

Anzahl Punkte	**Initialisierung (ms)**	**Algorithmus (ms)**
10	0.11	0.11
20	0.10	0.16
30	0.12	0.26
40	0.16	0.38
50	0.18	0.53
60	0.22	0.55
70	0.24	0.93
80	0.27	1.22
90	0.30	1.16
100	0.33	1.57
110	0.38	2.04
120	0.43	2.38
130	0.48	2.51
140	0.51	3.06
150	0.55	2.90
160	0.59	3.45
170	0.64	3.79
180	0.69	4.88
190	0.75	5.35
200	0.78	5.91

Tabelle 4.1: Laufzeit

Optimierungsbedarf besteht dementsprechend bei der Abarbeitung der unterschiedlichen Eventtypen. Werden die einzelnen Laufzeiten zur Behandlung der drei Eventtypen betrachtet, stellt sich heraus, dass die Behandlung eines Split Events mit 32885 ns deutlich zeitintensiver ist als die Behandlung eines Edge- oder Flip Events (2366.5 ns bzw. 3060.8 ns). Im Falle eines Split Events wird das Polygon in zwei Teile aufgeteilt, wodurch bei der im Kapitel 3.2 beschriebenen Implementierung alle Dreiecke

überprüft und bei Bedarf aktualisiert werden müssen. Bei den anderen zwei Eventtypen ist dies nicht notwendig. Hier werden nur jene Dreiecke aktualisiert, welche die Eventknoten beinhalten. Da bei konvexen Polygonen keine Split- oder Flip Events auftreten können, liegt die benötigte Zeit zur Berechnung des Weighted Straight Skeleton deutlich unter jener für konkave Polygone (siehe Abbildung 4.1).

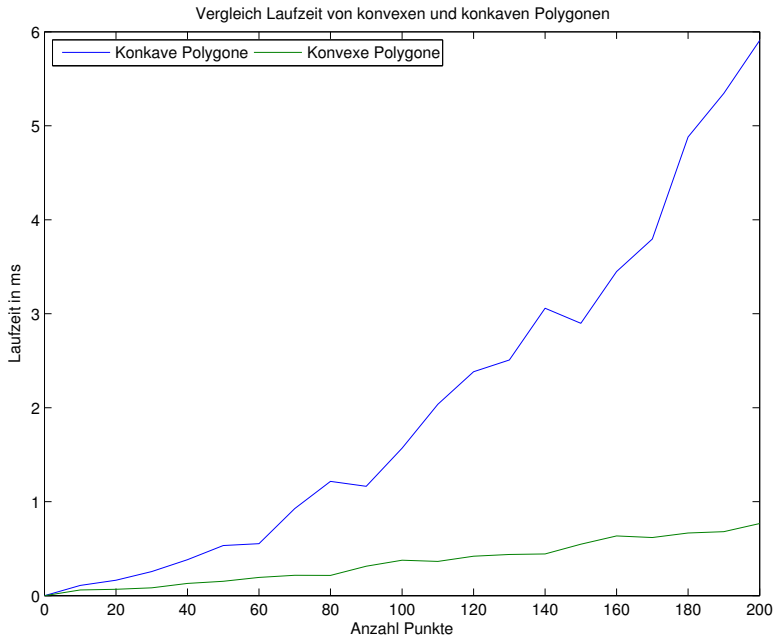

Abbildung 4.1: Vergleich Laufzeit von konvexen und konkaven Polygonen

4.2 Testdaten

Für die in diesem Kapitel beschriebenen Ergebnisse wurden Testdaten generiert. Die generierten Testdaten bestehen aus einem Set mit konkaven Polygonen, welches unter Zuhilfenahme der in Kapitel 3.1.1 beschriebenen Algorithmen erzeugt wurde, sowie aus einem Set von konvexen Polygonen, welche unter Verwendung des Programmes Matlab generiert wurden. Tabelle 4.2 gibt eine Übersicht über die Eckdaten der generierten, gewichteten Polygone.

	Konkave Polygone	**Konvexe Polygone**
Anzahl der Punkte	50	
max. Kantengewicht	50	
Kantenlänge	$\approx$ 5.24 cm	$\approx$ 1.62 cm
Flächeninhalt	$\approx$ 273.11 cm^2	$\approx$ 485.65 cm^2
Anzahl Edge Events	$\approx$ 37	48
Anzahl Split Events	$\approx$ 11	-
Anzahl Flip Events	$\approx$ 17	-

Tabelle 4.2: Eckdaten der generierten Polygone

Für jedes Polygon wurde sowohl das ungewichtete als auch das gewichtete Straight Skeleton berechnet. Im Falle der gewichteten Version wurden die Kantengewichte zufällig den Polygonkanten zugeordnet. Jedes Kantengewicht $1 \leq w_i \leq 50$ wurde dabei genau einmal vergeben.

Da nur positive Kantengewichte bei den Testpolygonen verwendet wurden, ist das Weighted Straight Skeleton $SK_W(P)$ ein Baum mit $n-2$ Nodes (siehe Kapitel 2.1.3). Wie in Kapitel 2 beschrieben, erzeugen nur Edge- und Split Events einen Node im $SK_W(P)$, wodurch bei den generierten Testpolygonen immer in Summe 48 Edge- und Split Events auftreten. Interessant ist es auch, die Anzahl der Flip Events zu betrachten. Diese liegt mit durchschnittlich 17 Events weit unter der von Aichholzer und Aurenhammer [1] beschriebenen Anzahl von $\mathcal{O}(n^3)$.

Abbildung 4.2 zeigt die Anzahl der Events für die vier Algorithmen zur Berechnung des ungewichteten und gewichteten Straight Skeletons. Abhängig vom Algorithmus lassen sich kleine Unterschiede zwischen ungewichteter und gewichteter Version feststellen. So treten beim Convex Layers Algorithmus bei der ungewichteten Version im Durchschnitt drei Flip Events mehr als bei der gewichteten Version des Straight Skeletons auf. Beim Steady Growth und Two Opts Moves Algorithmus tritt bei der ungewichteten Variante lediglich ein Flip Event mehr auf. Beim Random Two Peasants

tritt bei der gewichteten Variante ein Flip Event mehr auf. Bei allen Algorithmen lässt sich feststellen, dass bei der ungewichteten Version mehr Split Events gegenüber der gewichteten Version auftreten. Werden die vier Algorithmen, die zur Generierung von randomisierten Polygonen eingesetzt werden, miteinander verglichen, fällt auf, dass je nach verwendeten Algorithmus die Anzahl der Flip Events variiert. So treten z.B. bei jenen Polygonen, die mit dem Convex Layer Algorithmus generiert werden, mit durchschnittlich 25 Events die höchste Anzahl von Flip Events auf. Beim Steady Growth Algorithmus hingegen liegt die Anzahl der Flip Events bei durchschnittlich nur neun Events. Beim Random Two Peasants Algorithmus müssen im Durchschnitt 20 Flip Events behandelt werden, beim Two Opt Moves Algorithmus 13 Flip Events. Bei der Anzahl der Edge- und Split Events kann nur ein kleiner Unterschied zwischen den vier Algorithmen festgestellt werden. So treten beim Random Two Peasants Algorithmus mit 38 Edge Events die meisten und beim Steady Growth Algorithmus mit 35 Edge Events die wenigstens auf.

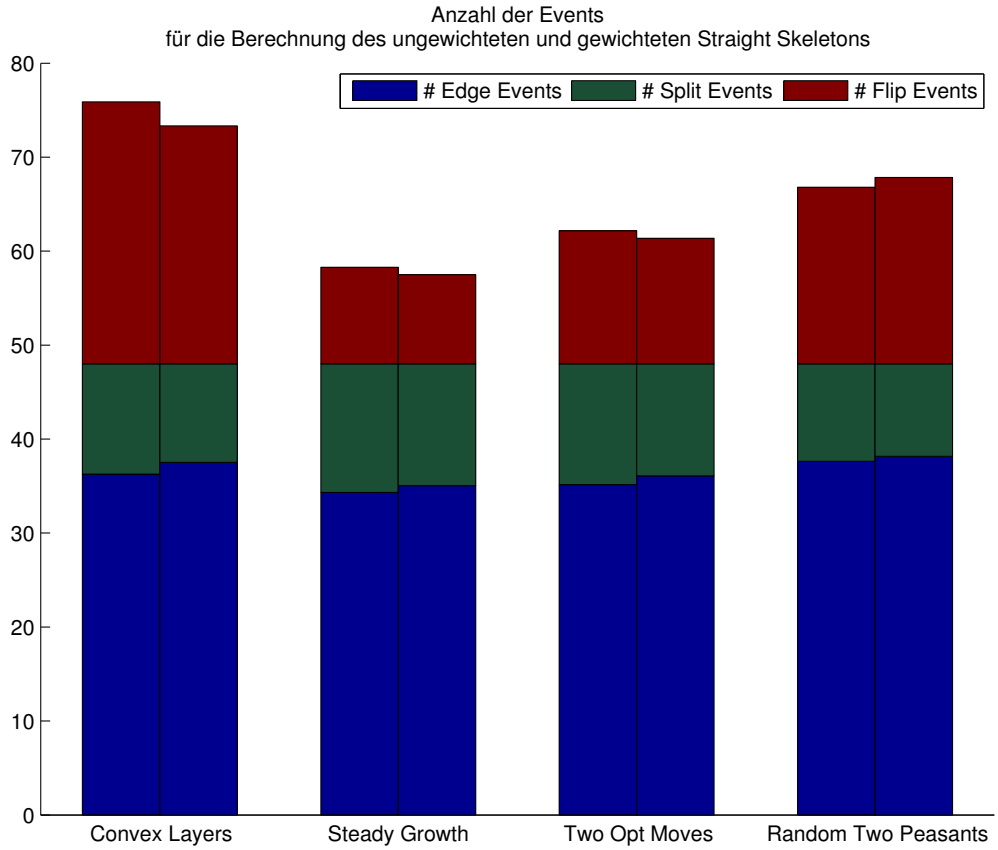

Abbildung 4.2: Anzahl der Events der Algorithmen

Im Durchschnitt haben die konkaven Polygone einen Flächeninhalt von 273.11 cm^2 (siehe Tabelle 4.2). Abbildung 4.3 zeigt die Flächeninhalte der Polygone für die vier Algorithmen. Mit 298 cm^2 weisen die Polygonen, welche mit dem Convex Layer Algorithmus erstellt wurden, den größten Flächeninhalt auf. Polygone generiert mit dem Random Two Peasants Algorithmus haben mit 258 cm^2 den kleinsten Flächeninhalt.

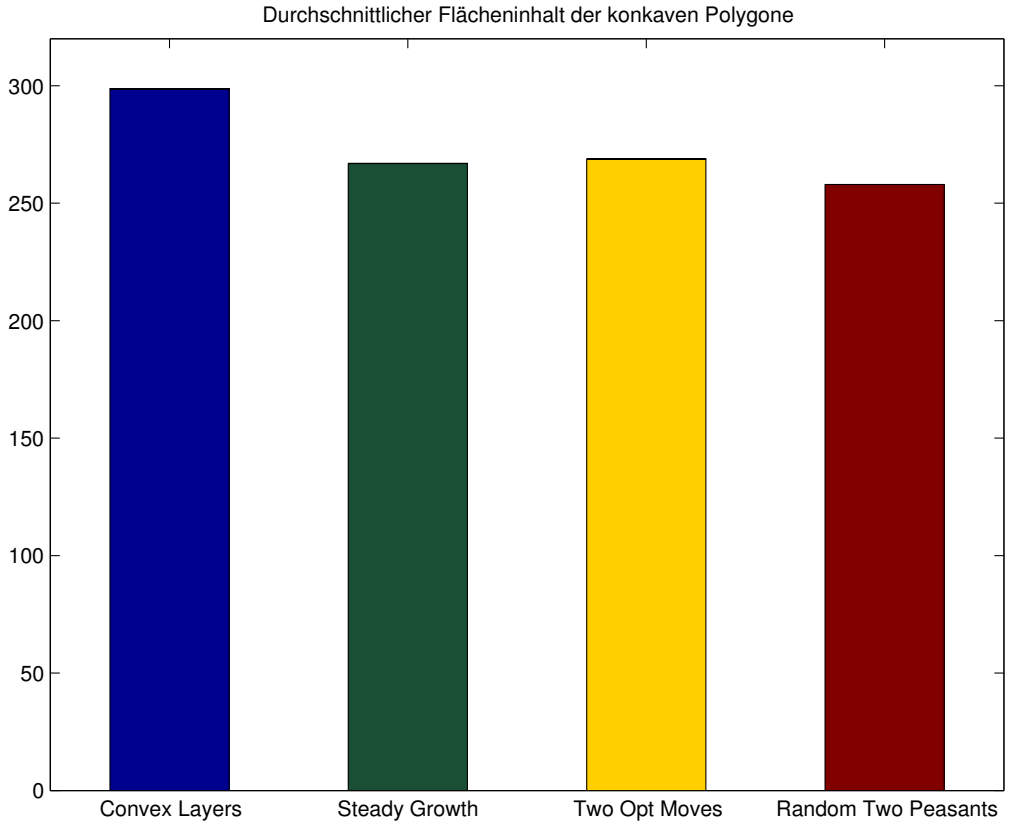

Abbildung 4.3: Durchschnittlicher Flächeninhalt der konkaven Polygone

4.3 Auswirkung des Kantengewichtes auf die Verteilung des Flächeninhaltes

Das Weighted Straight Skeleton teilt das Innere eines Polygon P mit n Kanten in n Polygone auf. Jede Kante erzeugt ein Face im Straight Skeleton. Die Flächeninhalte der Faces summieren sich immer zum Flächeninhalt des Polygons auf, wodurch sich der gleiche Mittelwert für den Flächeninhalt der Faces für beide Varianten ergibt. Offen bleibt jedoch die Frage, inwiefern die Gewichtung der Kanten Einfluss auf die Verteilung des vorhandenen Flächeninhaltes hat.

Abbildung 4.4 zeigt den Mittelwert und die Standardabweichung des Flächeninhaltes der Faces für die generierten Polygone. Bei den konkaven Polygonen kann ein kleiner Unterschied (Standardabweichung von 6.92 statt 5.25) zwischen der ungewichteten und gewichteten Version festgestellt werden. Bei den konvexen Polygonen ist jedoch deutlich zu erkennen, dass bei der gewichteten Version eine größere Standardabweichung (21.82 statt 5.30) auftritt. In diesem Fall treten Faces mit sehr großen und sehr kleinen Flächeninhalt auf.

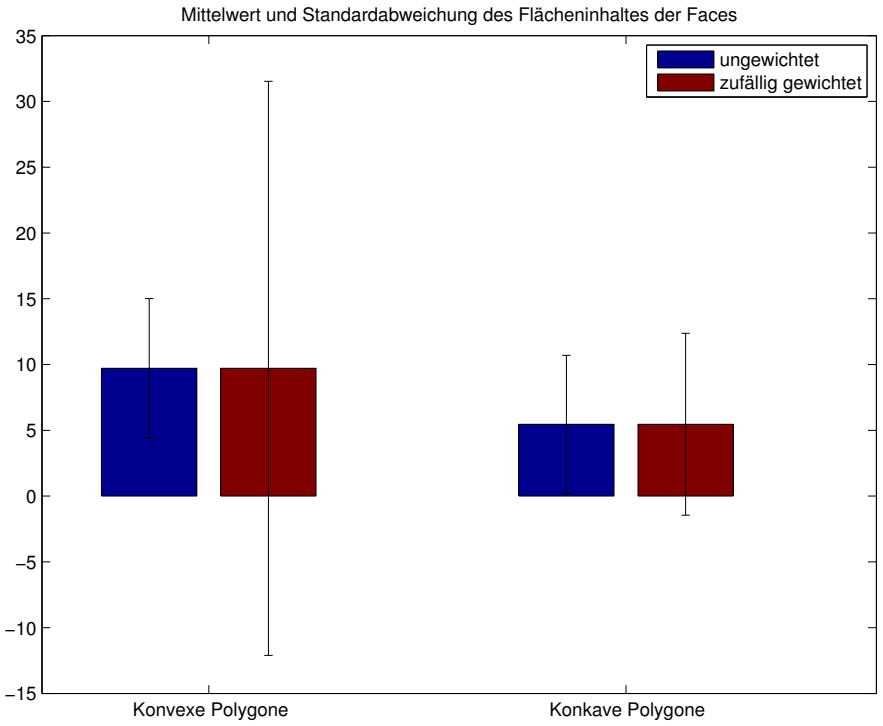

Abbildung 4.4: Mittelwert und Standardabweichung der Flächeninhalte der Faces

Deutlich wird dies auch, wenn man die Häufigkeitsverteilung der Flächeninhalte der Faces betrachtet (siehe Abbildung 4.5).

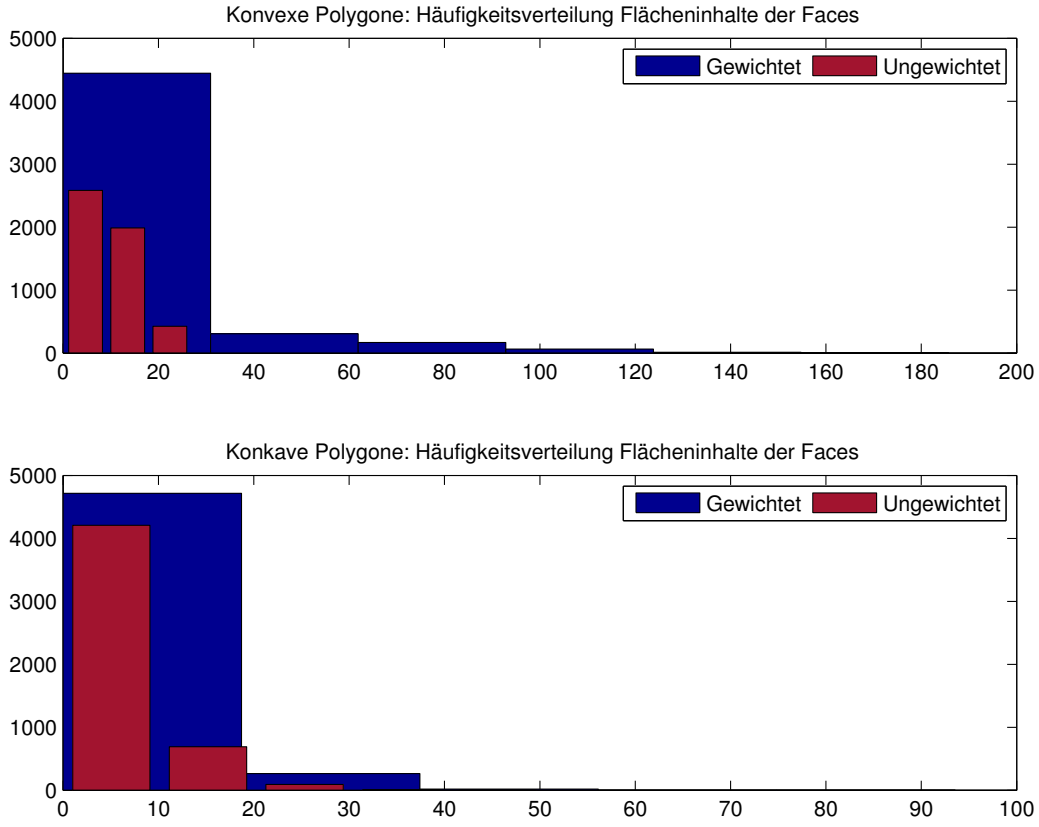

Abbildung 4.5: Häufigkeitsverteilung Flächeninhalte der Faces

Bei den ungewichteten, konvexen Polygonen haben alle Faces einen Flächeninhalt von maximal 27 cm^2. Bei der gewichteten Variante treten hingegen Flächeninhalte von bis zu 186 cm^2 auf. Der Großteil der Faces hat aber auch in diesem Fall einen Flächeninhalt von ≤ 27 cm^2. Auffallend ist jedoch, dass, im Gegensatz zu der ungewichteten Variante, eine Vielzahl der Faces (≈ 3000) einen Flächeninhalt von ≤ 1 cm^2 aufweisen.
Bei den konkaven Polygonen treten hingegen im Vergleich zur ungewichteten bei der gewichteten Variante etwas weniger als doppelt so viele Faces mit einem Flächeninhalt ≤ 1 cm^2 auf. Auch beim maximalen Flächeninhalt lässt sich ein Unterschied feststellen. Bei der ungewichteten Variante haben die Faces einen Flächeninhalt von bis zu 51 cm^2. Bei der gewichteten Variante hingegen tritt ein Flächeninhalt von bis zu 94 cm^2 auf. Die Differenz ist in diesem Fall aber um einiges kleiner als bei den konvexen Polygonen. Der Großteil der Faces hat allerdings bei beiden Varianten einen

Flächeninhalt von maximal 20 cm^2. Der Unterschied zwischen konkaven und konvexen Polygonen lässt sich darauf zurückführen, dass sich bei konvexen Polygonen die Faces kontinuierlich vergrößern, wenn das Gewicht erhöht wird. Bei konkaven Polygonen ist dies nicht der Fall [4].

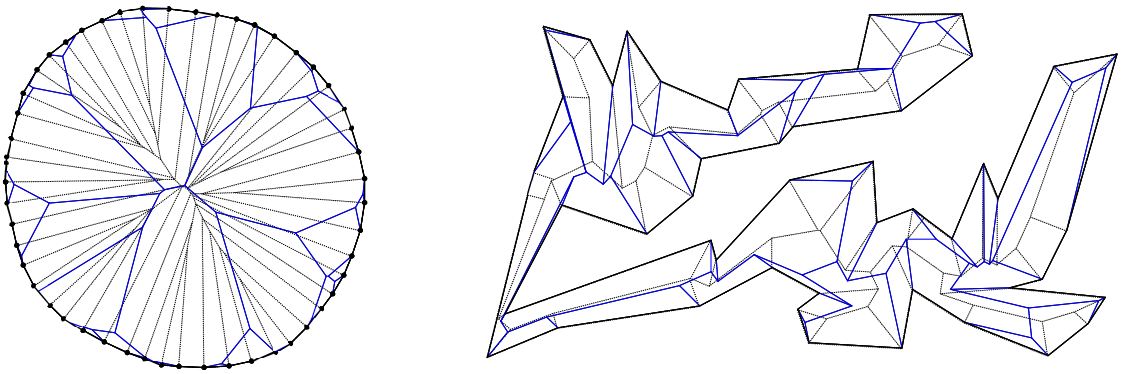

Abbildung 4.6: Verteilung des Flächeninhaltes auf die Faces des Straight Skeletons eines konvexen und konkaven Polygons

Abbildung 4.6 zeigt das ungewichtete (strichlierte Linie) und gewichtete Straight Skeleton für ein konvexes und konkaves Polygon. Beim konvexen Polygon kann ein deutlicher Unterschied zwischen beiden Varianten festgestellt werden. Im Gegensatz zum ungewichteten Straight Skeleton verteilt sich bei der gewichteten Version ein Großteil des verfügbaren Flächeninhaltes auf eine geringe Anzahl (8) der Faces des Weighted Straight Skeletons. Auch beim konkaven Polygon ist ein Unterschied zwischen beiden Varianten sichtbar. Bei einigen Faces hat sich bei der gewichteten Version der Flächeninhalt vergrößert bzw. verkleinert. Die Auswirkung auf die Verteilung des Flächeninhaltes ist jedoch nicht so groß wie beim konvexen Polygon.

4.4 Auswirkung des Kantengewichtes auf den Flächeninhalt eines Faces

Ein Face des Straight Skeletons ist jene Region, die von einer Polygonkante während des Schrumpfprozesses erreicht wird, bevor sie auf Länge Null zusammenschrumpft. Unter der Voraussetzung, dass alle Kanten das gleiche Gewicht besitzen, bestimmt die Kantenlänge den Flächeninhalt eines Faces maßgeblich mit. Längere Polygonkanten benötigen mehr Zeit bis sie vollständig verschwunden sind und erzeugen dadurch einen größeren Flächeninhalt. Bei der ungewichteten Version bzw. wenn das gleiche Gewicht für jede Kante vergeben wird, bewegen sich die Kanten mit gleicher Geschwindigkeit ins Innere des Polygons. Dies ist bei der gewichteten Variante nicht mehr der Fall. Hier wird durch das Zuordnen eines Gewichtes zur Kante die Geschwindigkeit der Kante beeinflusst. Abbildung 4.7 zeigt ein regelmäßiges Fünfeck mit gleichen Gewichten für jede Polygonkante. Die Faces des Straight Skeletons weisen hier den gleichen Flächeninhalt auf. Wird einer Kante ein größeres Gewicht zugeordnet, vergrößert sich der Flächeninhalt des Faces (siehe Abbildung 4.8).

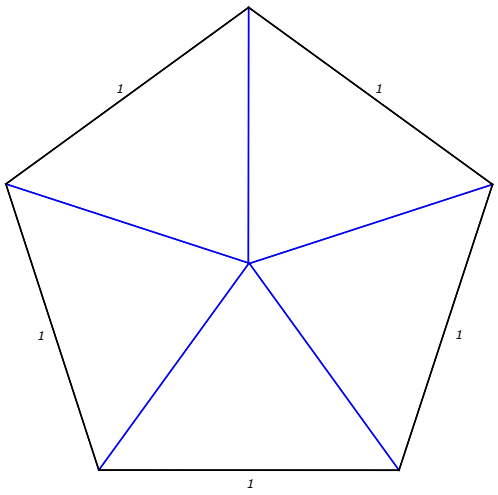

Abbildung 4.7: Fünfeck mit gleichen Kantengewichten

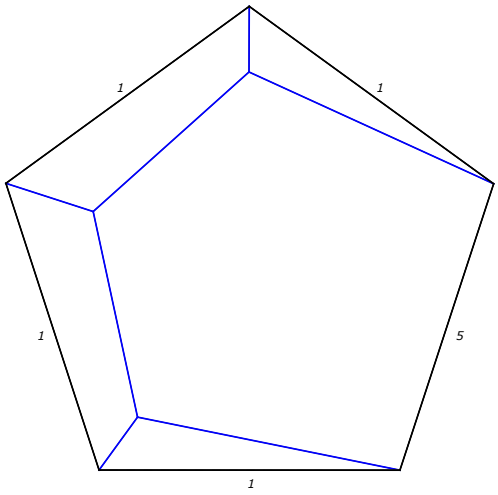

Abbildung 4.8: Fünfeck mit einem großen Kantengewicht

Es gilt nun herauszufinden, ob neben der Kantenlänge auch ein größeres Kantengewicht automatisch zu einem größeren Flächeninhalt der Faces führt.

Da bereits bekannt ist, dass die Kantenlänge ausschlaggebend für den Flächeninhalt ist, wurde für die weiteren Analysen der Flächeninhalt A folgendermaßen normiert:

$$A_n = \frac{A}{\text{Kantenlänge}}.$$

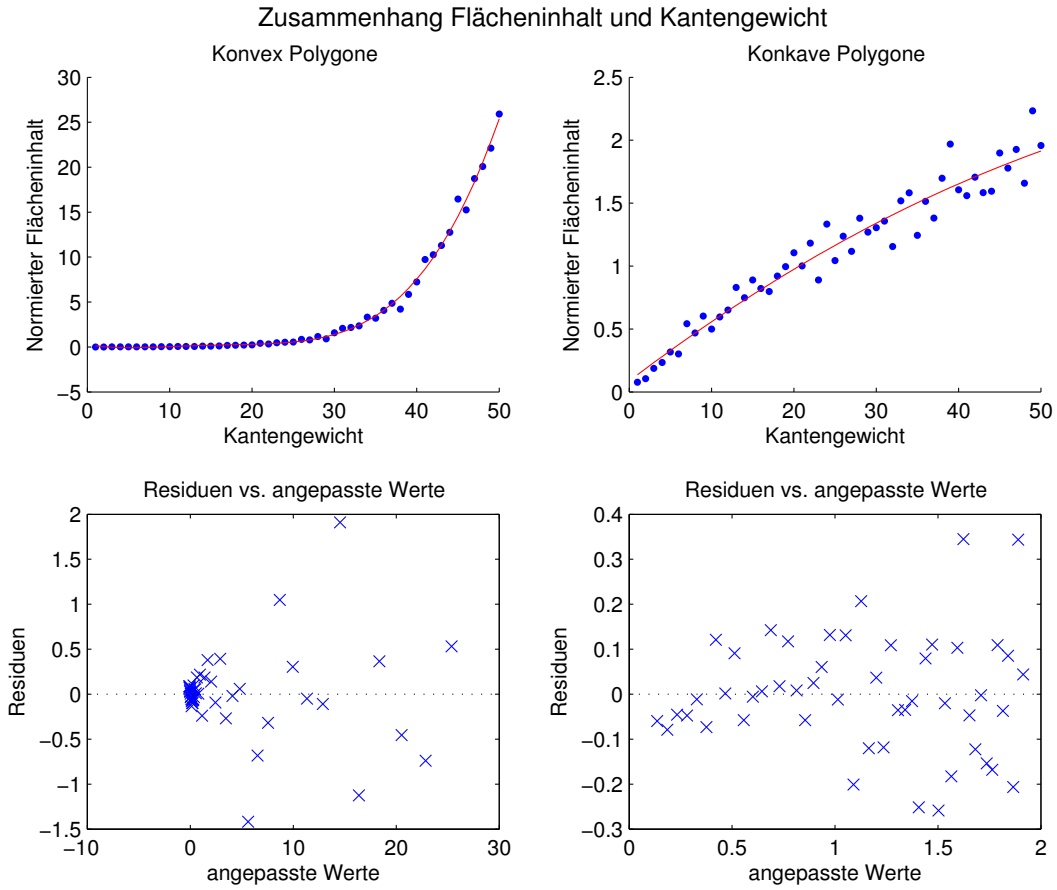

Abbildung 4.9: Zusammenhang Gewicht und Flächeninhalt

Abbildung 4.9 zeigt, dass kein linearer Zusammenhang zwischen dem normierten Flächeninhalt und dem Kantengewicht besteht. Sowohl bei den konvexen als auch bei den konkaven Polygonen eignet sich ein Fit mit einer Funktion von Grad ≥ 2 besser für die Daten als ein Fit mit einer linearen Funktion. Die Residuenplot der konkaven Polygonen zeigt eine inhomogene Varianz der Residuen. Ein größeres Kan-

tengewicht kann, muss aber nicht zwingend zu einem größeren Flächeninhalt führen. Der Flächeninhalt eines Faces wird daher noch von anderen Faktoren bestimmt.

Abbildung 4.10 zeigt ein regelmäßiges Fünfeck mit einer Kantengewichtskonfiguration. Trotz eines größeren Kantengewichtes weist die Kante mit einem Gewicht von Fünf einen kleineren Flächeninhalt auf, als jene Kante mit einem Gewicht von Drei. Ein Grund hierfür ist, dass beiden Nachbarkanten hohe Gewichte zugeordnet wurden und deren Faces einen Großteil des verfügbaren Flächeninhaltes und einen großen Teil des Faces der Kante mit Gewicht Fünf einnehmen. Diese Faces beeinflussen auch den Flächeninhaltes des Faces mit der Kante von Gewicht Drei. Jedoch wurde einer der Nachbarkante ein kleines Gewicht zugeordnet, wodurch das Face der Kante mit Gewicht Drei vergrößert wurde. Zusammengefasst kann gesagt werden, dass der Flächeninhalt eines Faces zusätzlich noch durch alle anderen Faces des Straight Skeletons beeinflusst wird. Eine lokale Änderung kann daher global große Auswirkungen haben, wodurch der Schrumpfprozess immer simultan für alle Kanten gleichzeitig durchgeführt werden muss.

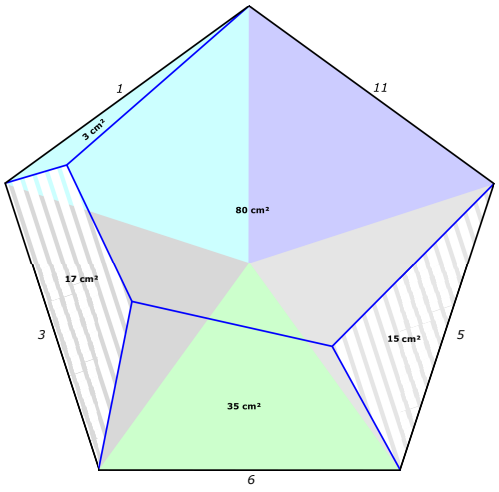

Abbildung 4.10: Fünfeck mit unterschiedlichen Kantengewichten

4.5 Zusammenhang Kantenanzahl und Flächeninhalt der Faces

In Kapitel 4.4 wurde gezeigt, dass kein linearer Zusammenhang zwischen Flächeninhalt eines Faces und dem Kantengewicht besteht. Bei weiterer Betrachtung von Abbildung 4.8 wird sichtbar, dass jenes Face mit dem größten Flächeninhalt auch die größte Anzahl von Kanten besitzt. Von den restlichen vier Faces weisen jeweils immer zwei den gleichen Flächeninhalt und in diesem Fall auch die gleichen Kantenanzahl auf. Des Weiteren ist ersichtlich, dass die Kantenanzahl mit dem Flächeninhalt steigt. Die Frage, die sich nun stellt, ist, ob sich mit größer werdenden Flächeninhalt auch die Kantenanzahl erhöht. Jedes Face des Straight Skeletons besteht aus mindestens drei Kanten, einer Polygonkante und jenen zwei Straight Skeleton Kanten, die von den Endpunkten der Polygonkante erzeugt werden. Dies ist genau dann der Fall, wenn eine Kante durch ein Edge Event vor ihren adjazenten Kanten verschwindet. Nach jedem Edge- und Split Event wird der neu eingefügte Node mit den Nachbarkanten der Eventkante verbunden (siehe Kapitel 2.1.1), wodurch eine neue Kante in den Faces der Nachbarkanten entsteht. Abbildung 4.11 zeigt für die konvexen und konkaven Polygone die aufgetretenen Kantenanzahlen der Faces.

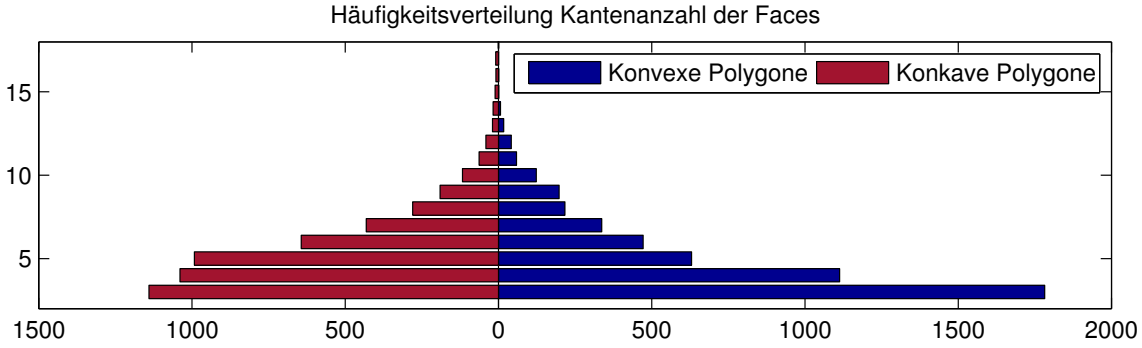

Abbildung 4.11: Häufigkeitsverteilung Kantenanzahl

Nur ein sehr geringer Anteil an Faces weist sowohl bei den konvexen als auch bei den konkaven Polygonen eine Kantenanzahl von > 10 auf. Der Großteil der Faces besitzt eine Kantenanzahl von $3 \leq$ Anzahl Kanten ≤ 6. Wie am Anfang dieses Kapitels (siehe Kapitel 4.2) beschrieben, hat auch der Großteil der Faces einen sehr kleinen Flächeninhalt (siehe Abbildung 4.5).

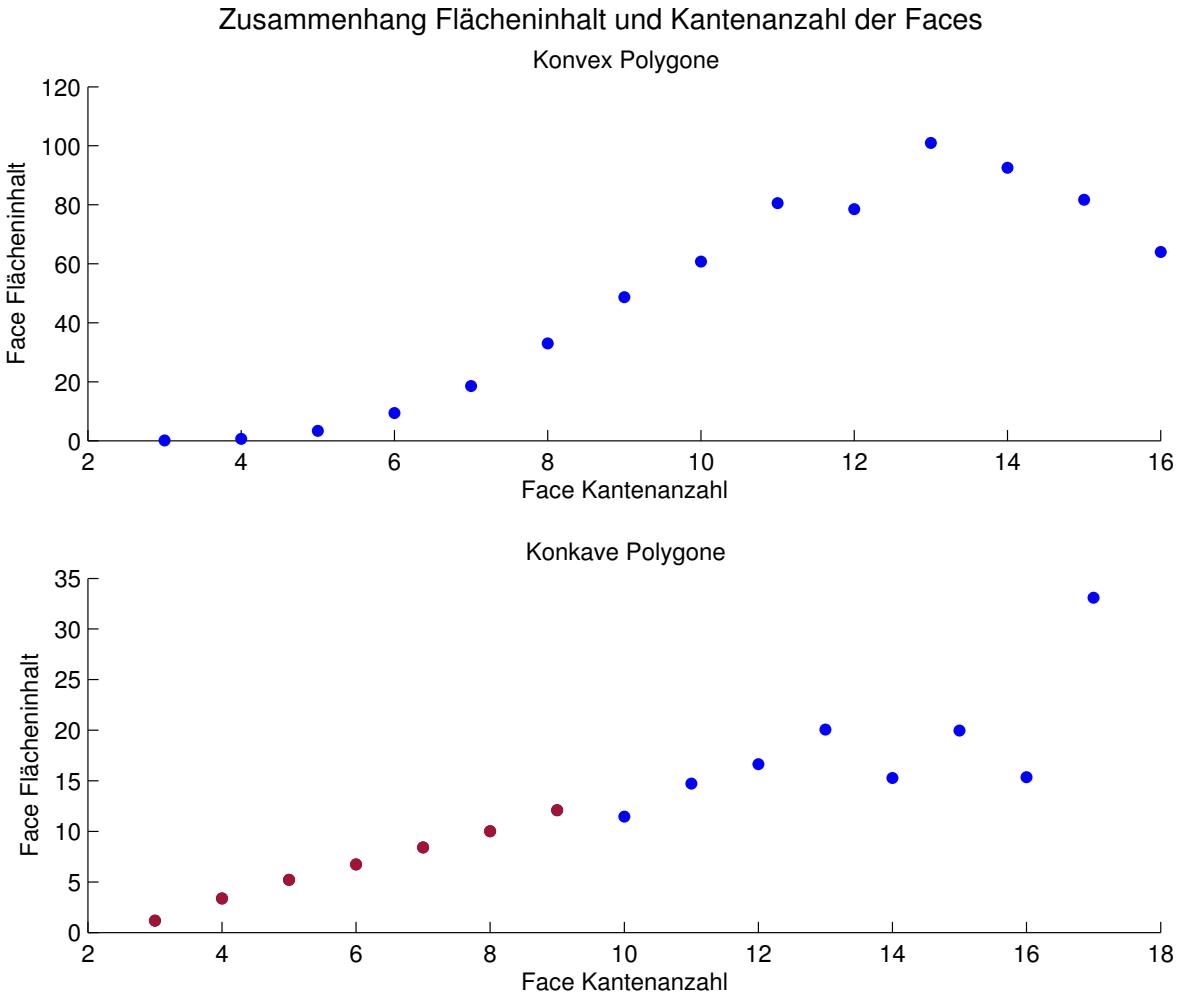

Abbildung 4.12: Zusammenhang Kantenanzahl und Flächeninhalt

Abbildung 4.12 zeigt allerdings, dass kein linearer Zusammenhang zwischen Kantenanzahl und Flächeninhalt für beide Polygontypen besteht. Bei den konkaven Polygonen hat es bis zu einem Flächeninhalt von $\leq 12\ cm^2$ zunächst den Anschein, als würde die Annahme zutreffen. Ab einem Flächeninhalt von $> 12\ cm^2$ haben die Faces bei steigendem Flächeninhalt aber nicht mehr zwingend auch eine höhere Kantenanzahl. Sowohl eine kleine als auch eine große Kantenanzahl ist möglich.

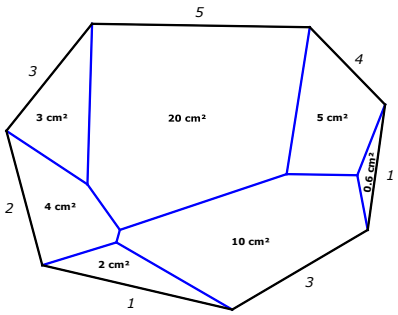

Abbildung 4.13: Konvexes Polygon mit unterschiedlichen Kantenanzahl pro Face

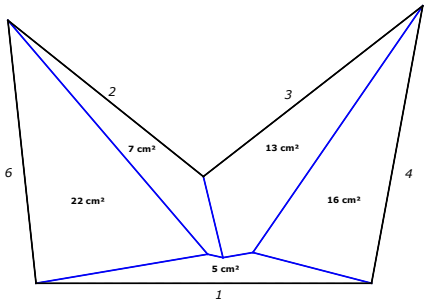

Abbildung 4.14: Konkaves Polygon mit unterschiedlichen Kantenanzahl pro Face

Abbildung 4.13 und Abbildung 4.14 zeigen zwei Polygonen mit ihren Weighted Straight Skeleton. Die Faces des Straight Skeletons besitzen unterschiedlichste Kantenanzahlen. Deutlich sichtbar wird bei beiden Beispielen, dass ein großer Flächeninhalt nicht unbedingt zu einer großen Kantenanzahl führt. Beim Straight Skeleton des konkaven Polygons (siehe Abbildung 4.14) hat sogar jenes Face mit dem kleinsten Flächeninhalt die größte Kantenanzahl. Wie auch schon im vorherigen Kapitel 4.4 beschrieben, wird auch die Kantenanzahl eines Faces von den anderen Faces des Straight Skeletons bestimmt.

5 Zusammenfassung und Ausblick

Das Weighted Straight Skeleton stellt eine Erweiterung des Straight Skeletons dar. Im Gegensatz zur ungewichteten Variante des Straight Skeletons wird beim Weighted Straight Skeleton jeder Polygonkante ein Gewicht zugeordnet, wodurch sich die Polygonkanten mit unterschiedlicher Geschwindigkeit während des Schrumpfprozesses ins Innere des Polygons bewegen. Ein Ausnahmefall stellt hierbei das Kantengewicht Null dar; bei diesem Gewicht findet keine Bewegung der Kante statt. Um die Unterschiede der ungewichteten und gewichteten Version des Straight Skeletons aufzuzeigen, wurden in dieser Arbeit zuerst das ungewichtete Straight Skeleton und seine Eigenschaften beschrieben. Bei der Gegenüberstellung der Eigenschaften der ungewichteten und gewichteten Version wird deutlich, dass nicht alle Eigenschaften der ungewichteten auf die gewichtete Version übergehen. Im Gegensatz zum ungewichteten Straight Skeleton müssen die Faces beim Weighted Straight Skeleton keine monotonen Polygone sein. Andere Eigenschaften werden wiederum nur durch eine Beschränkung auf einen bestimmten Polygontyp (z.B. einfaches Polygon) oder eine Gewichtskonfiguration erfüllt. So ist das Weighted Straight Skeleton nur genau dann ein Baum, wenn nur positive Kantengewichte gewählt wurden. Des Weiteren muss beim Weighted Straight Skeleton berücksichtigt werden, dass Sonderfälle, wie z.B. parallele Wavefront Kanten nach einem Edge Event, während des Schrumpfprozesses auftreten können. In diesen Fällen ist nicht klar definiert, wie der Schrumpfprozess weitergeführt werden kann. Eine Behandlung dieser Anomalien ist daher unabdingbar.

Es existieren eine Vielzahl von Algorithmen, welche sowohl das ungewichtete als auch das gewichtete Straight Skeleton mit unterschiedlichen Laufzeiten und Speicherbedarf berechnen können. Einige Algorithmen müssen für die Berechnung des Weighted Straight Skeleton geringfügig adaptiert werden. Ein essentieller Unterschied zur Berechnung des Straight Skeleton besteht darin, wie die Bewegungsvektoren der Polygonknoten ermittelt werden. Im Gegensatz zur ungewichteten Version des Straight Skeletons bewegen sich die Knoten beim Weighted Straight Skeleton aufgrund der Kantengewichtung nicht entlang der Winkelhalbierenden. Der Bewegungsvektor kann jedoch über den Schnittpunkt der adjazenten Kanten für einen Polygonknoten

berechnet werden. Zusätzlich muss, wie bereits beschrieben, die Behandlung von Sonderfällen berücksichtigt werden. Im Gegensatz zum ungewichteten Straight Skeleton kann beim gewichteten der Fall eintreten, dass nach einem Edge Event zwei parallele Kanten mit unterschiedlichen Gewichten benachbart worden sind oder, dass der neue Node konkav ist. Auch mehrere Split Events am gleichen Ort sind möglich. In der vorliegenden Arbeit wurde beim Auftreten des ersten Falles (hier: parallele Kanten) entschieden, die langsamere Kante zugunsten der schnelleren aufzugeben. Zusätzlich wird nach jedem Edge Event überprüft, ob es sich beim neuen Node um einen konvexen oder konkaven Knoten handelt.

Im Rahmen dieser Arbeit wurde der Algorithmus von Aichholzer *et al.* [1] adaptiert und implementiert. Bei diesem Algorithmus wird während des gesamten Schrumpfprozesses eine Triangulierung aufrecht erhalten. Da ein konkaver Knoten auf eine Diagonale eines Triangulierungsdreieck stoßen kann, wurde ein weiteres Event, das sogenannte "Flip Event", definiert. Die Anzahl der Flip Events beträgt laut Aichholzer *et al.* $O(n^3)$, wodurch sich eine Gesamtlaufzeit von $\mathcal{O}(n^3 \log n)$ ergibt. Hierbei handelt es sich eher um eine theoretische, obere Grenze, wie Kapitel 4.2 der Arbeit zeigt. Für konkave Polygone mit 50 Knoten wurden zur Berechnung des Weighted Straight Skeleton durchschnittlich 17 Flip Events benötigt.

Im nächsten Schritt wurde untersucht, ob und inwiefern sich die Kantengewichtung auf die Verteilung des Polygonflächeninhaltes auf die Faces des Straight Skeletons auswirkt. Hier wurde festgestellt, dass die Auswirkung der Gewichtung bei konvexen größer als bei konkaven Polygonen ist. Im Fall von konvexen Polygonen erreichen einige wenige Faces einen sehr großen Flächeninhalt, wodurch wiederum Faces mit sehr kleinen Flächeninhalt existieren. Bei den konkaven Polygonen konnte ein so großer Unterschied nicht festgestellt werden. Auch in diesem Fall kann durch den Einsatz von Kantengewichten der Flächeninhalt der einzelnen Faces beeinflusst werden. Im Gegensatz zu den konvexen Polygonen wächst der Flächeninhalt aber nicht kontinuierlich, wenn das Gewicht erhöht wird. Zusätzlich wurde die Auswirkung des Kantengewichtes auf den Flächeninhalt der Faces des Weighted Straight Skeletons untersucht. Hier wurde festgestellt, dass ein großes Kantengewicht nicht immer zu einem großen Flächeninhalt des Faces führen muss. Auch die Annahme, dass ein großer Flächeninhalt zu einer großen Kantenanzahl der Faces führt, konnte widerlegt werden.

Bei den in dieser Arbeit verwendeten Testpolygonen handelt es sich um einfache, konkave oder konvexen Polygone mit nur positiven Kantengewichten. Die Berechnung des Weighted Straight Skeletons von Polygonen mit negativen Kantengewichten ist

mit dieser Implementierung nicht möglich. Die Verwendung einer Kombination aus positiven und negativen Kantengewichte würde dazu führen, dass manche Eigenschaften des Weighted Straight Skeleton nicht mehr immer erfüllt werden können. Biedl *et al.* [6] haben aufgezeigt, dass mit einer Kombination aus positiven und negativen Kantengewichten beim Straight Skeleton eines einfachen Polygon nicht sichergestellt werden kann, dass es zu einem kreuzungsfrei ist und zum anderem keinen Zyklus beinhaltet. Somit muss das Straight Skeleton auch nicht zwingend ein Baum sein. Bei nur positiven Kantengewichten sind die Faces des Straight Skeletons einfache Polygone. Diese Eigenschaft wird, wenn auch negative Kantengewichte erlaubt sind, nicht immer erfüllt. Sichergestellt ist jedoch auch bei negativen Kantengewichten und einem einfachen Polygon, dass das Straight Skeleton zusammenhängend ist.

Ausblick: Additiv gewichtetes Straight Skeleton
Bei den in dieser Arbeit beschriebenen Algorithmen startet der Schrumpfprozess simultan für jede Polygonkante. Held und Palfrader [14] haben Anfang 2016 eine neue Variante des Straight Skeleton, das additiv gewichtete Straight Skeleton, für einfache Polygone vorgestellt. Bei dieser Variante können Polygonkanten zu unterschiedlichen Zeitpunkt ihre Bewegung starten. Eine Kombination mit dem in dieser Arbeit beschriebenen Weighted Straight Skeleton ist möglich, solange keine negativen Kantengewichte verwendet werden. Negative, additive Gewichte sind jedoch bei dieser Variante möglich. In diesem Fall würde der Schrumpfprozess einfach zum Zeitpunkt des kleinsten Gewichtes starten. Wie auch beim (Weighted) Straight Skeleton (siehe Kapitel 2) sind auch beim additiv gewichteten Straight Skeleton die Wavefronts ineinander verschachtelt. Das Ergebnis des Schrumpfprozesses ist auch in diesem Fall eine Hierarchie von verschachtelten Polygonen. Unterschied zu den beiden Varianten besteht darin, dass einzelne Wavefronts sich überlappen können.

Die additiven Gewichte legen für jede Kante fest, zu welchen Zeitpunkt ihre Bewegung ins Innere des Polygons startet. In Kapitel 2.1.3 dieser Arbeit wurde definiert, dass $e(t)$ die Vereinigung aller Wavefront Kanten einer Polygonkante e bis zum Zeitpunkt t ist. Im Falle des additiven, gewichteten Straight Skeleton sind diese Fragmente Teil von $\mathbf{e} + max(0, t - \delta(e)) * n_e$. Wobei $\delta(e)$ jene Funktion darstellt, die einer Kante einen Startzeitpunkt zuweist. Wie auch beim Weighted Straight Skeleton bewegen sich die Polygonknoten nicht unbedingt entlang der Winkelhalbierenden. Ausschlaggebend hierfür ist, ob beide benachbarten Kanten bereits mit ihrer Bewegung begonnen haben oder nicht. Laut Held und Palfrader [14] kann der triangulierungsbasierte Algorithmus, welcher in Rahmen dieser Arbeit implementiert wurde, auch für die Berechnung des additiv gewichteten Straight Skeleton verwendet werden. Es muss jedoch ein zusätzliches Event, das sogenannte "Speed Change Event" berücksichtigt

werden. Dieses Event zeigt an, dass eine neue Kante ihre Bewegung ins Innere des Polygons gestartet hat. Die Geschwindigkeit und die Richtung der beteiligten Wavefront Knoten muss aktualisiert werden.

Als ein Anwendungsfall des Weighted Straight Skeleton wurde die Konstruktion von Dächern (siehe Kapitel 2.2.4) genannt. Durch den Einsatz von Kantengewichten können unterschiedliche Steigungen der Facetten des Daches erreicht werden. Werden noch additive Gewichte verwendet, kann zusätzlich noch die Höhe, in der das Dach anfängt sich nach innen zu neigen, festgelegt werden. Im Gegensatz zum ungewichteten Roof-Model ist das additiv gewichtete Roof-Model nicht mehr z-monoton, da Wavefront Kanten sich während des Schrumpfprozesses nicht bewegen müssen. In diesem Fall entstehen vertikale Facetten. Sichergestellt wird hierbei weiterhin, dass das Regenwasser abrinnen kann. Eine nähere Betrachtung des additiv gewichteten Straight Skeletons und seine Eigenschaften in weiterführenden Arbeiten ist aus diesem Grund sinnvoll.

Literaturverzeichnis

[1] Oswin Aichholzer and Franz Aurenhammer. Straight skeletons for general polygonal figures in the plane. In *Proceedings of the Second Annual International Conference on Computing and Combinatorics*, COCOON '96, pages 117–126, London, UK, UK, 1996. Springer-Verlag.

[2] Oswin Aichholzer, Franz Aurenhammer, David Alberts, and Bernd Gärtner. Straight skeletons of simple polygons. In *Proc. 4th Internat. Symp. of LIESMARS*, pages 114–124, Wuhan, P.R. China, 1995.

[3] Oswin Aichholzer, Franz Aurenhammer, David Alberts, and Bernd Gärtner. A Novel Type of Skeleton for Polygons. In Hermann Maurer, Cristian Calude, and Arto Salomaa, editors, *J.UCS The Journal of Universal Computer Science*, pages 752–761. Springer Berlin Heidelberg, 1996.

[4] Franz Aurenhammer. Weighted skeletons and fixed-share decomposition. *Comput. Geom. Theory Appl.*, 40(2):93–101, July 2008.

[5] Gill Barequet and Amir Vaxman. Straight skeletons of three-dimensional polyhedra. In *Proceedings of the Twenty-fifth Annual Symposium on Computational Geometry*, SCG '09, pages 100–101, New York, NY, USA, 2009. ACM.

[6] Therese Biedl, Martin Held, Stefan Huber, Dominik Kaaser, and Peter Palfrader. Weighted straight skeletons in the plane. *Comput. Geom. Theory Appl.*, 48(2):120–133, October 2014.

[7] Therese Biedl, Martin Held, Stefan Huber, Dominik Kaaser, and Peter Palfrader. A simple algorithm for computing positively weighted straight skeletons of monotone polygons. *Information Processing Letters*, 115(2):243 – 247, 2015.

[8] Therese Biedl, Stefan Huber, and Peter Palfrader. Stable roommates for weighted straight skeletons. In *Proc. 30th Europ. Workshop on Comp. Geom. (EuroCG '14)*, Dead Sea, Israel,, March 2014.

[9] Harry Blum. A Transformation for Extracting New Descriptors of Shape. In Weiant Wathen-Dunn, editor, *Models for the Perception of Speech and Visual Form*, pages 362–380. MIT Press, Cambridge, 1967.

[10] Bernard Chazelle. Triangulating a simple polygon in linear time. *Discrete Comput. Geom.*, 6(5):485–524, August 1991.

[11] David Eppstein and Jeff Erickson. Raising roofs, crashing cycles, and playing pool: Applications of a data structure for finding pairwise interactions. In *Proceedings of the Fourteenth Annual Symposium on Computational Geometry*, SCG '98, pages 58–67, New York, NY, USA, 1998. ACM.

[12] Petr Felkel and Štěpán Obdržálek. Straight skeleton implementation. In *Proceedings of Spring Conference on Computer Graphics*, pages 210–218, 1998.

[13] Jan-Henrik Haunert and Monika Sester. Area collapse and road centerlines based on straight skeletons. *Geoinformatica*, 12(2):169–191, June 2008.

[14] Martin Held and Peter Palfrader. Straight skeletons with additive and multiplicative weights and their application to the algorithmic generation of roofs and terrains. *CoRR*, abs/1604.03362, 2016.

[15] Stefan Huber. *Computing Straight Skeletons and Motorcycle Graphs: Theory and Practice*. Shaker Verlag, April 2012.

[16] Stefan Huber and Martin Held. Straight skeletons and their relation to triangulations. In *Proc. 26th Europ. Workshop on Comp. Geom. (EuroCG '10)*, volume vol 41(5), pages 327–338, Dortmund, Germany, March 2010.

[17] P. Jackson. *The Pop-Up Book*. Henry Holt and Company, New York, 1993.

[18] Tom Kelly and Peter Wonka. Interactive architectural modeling with procedural extrusions. *ACM Trans. Graph.*, 30(2):14, 2011.

[19] D. T. Lee. Medial axis transformation of a planar shape. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, PAMI-4(4):363–369, July 1982.

[20] Wu Liang. Poly2tri: Fast and robust simple polygon triangulation with/without holes by sweep line algorithm. http://sites-final.uclouvain.be/mema/Poly2Tri/, 2005. Accessed: 2016-01-17.

[21] Franco P. Preparata and Michael I. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag New York, Inc., New York, NY, USA, 1985.

[22] Roman Purgstaller. *Randomisierte Polygone*. Technische Universität Graz, Institut für Grundlagen der Informationsverarbeitung, 2015.

[23] Kokichi Sugihara. Design of pop-up cards based on weighted straight skeletons. In *Proceedings of the 2013 10th International Symposium on Voronoi Diagrams in Science and Engineering*, ISVD '13, pages 23–28, Washington, DC, USA, 2013. IEEE Computer Society.

[24] University of Bonn. Geometry lab: Random polygon algorithms. `http://web.informatik.uni-bonn.de/I/GeomLab/RandomPolygon/index.html.en#pops`, 02 2009. Accessed: 2015-09-27.

[25] Gernot Christian Walzl. *STRAIGHT SKELETONS - From Plane to Space*. PhD thesis, Graz University of Technology, 2015.