Marlene RACHHOLZ

Graduntersuchungen von Triangulierungen in 2D

DIPLOMARBEIT

zur Erlangung des akademischen Grades einer Diplom-Ingenieurin

Diplomstudium Technische Mathematik

Technische Universität Graz

Betreuer und Begutachter:
Assoc. Prof. Dipl.-Ing. Dr.-techn. Oswin Aichholzer

Institut für Softwaretechnologie

Graz, April 2011

Abstract

Triangulations are well-researched data structures used for geometric algorithms. A triangulation of a set of points in the plane is a complete disjoint decomposition of the convex hull into triangles. Those structures are used for complex algorithms that operate on sets of points in the plane. Examples for such algorithms can be found in many fields, such as simulation technology, computer graphics, and technical modelling. A very important research area is the optimization of triangulations with respect to different criteria, e.g., the minimization of the total length of the edges of a triangulation.

In this work, triangulations were analyzed with respect to the maximum degree of all vertices of the triangulation. This value is known to have a significant influence on the complexity of many algorithms using triangulations as data structures. Despite of this importance, however, this criterion has not yet been intensively investigated. The present thesis analyzes how both lossless and lossy modifications of the triangulation affect the maximum degree of its vertices. A number of algorithms were designed and implemented which perform such modifications by both adding and deleting points, without changing the convex hull. These algorithms were used to evaluate which degrees can be efficiently obtained with as little alteration of the point set as possible. In order to test these algorithms, a program was implemented that can produce point sets whose triangulations always have rather high degrees. The results were analyzed statistically.

It was found that randomized point sets could be reduced well using lossless techniques. However, for specially constructed point sets with relatively small degrees good results could be achieved using point removing algorithms, wheres for point sets with rather high degrees point inserting algorithms performed better.

Keywords: triangulations, degrees of vertices, reducing degrees, geometric algorithms

Kurzfassung

Triangulierungen sind eine viel studierte Datenstruktur im Gebiet der geometrischen Algorithmik. Eine Triangulierung einer Punktmenge ist eine vollständige, disjunkte Zerlegung der konvexen Hülle in Dreiecke. Sie werden als Datenstruktur für komplexere Algorithmen auf Punktmengen angewendet. Beispiele für solche Algorithmen liegen in Bereichen wie der Simulationstechnik, der Computergraphik oder sind im Bereich der Modellierung zu finden. Ein wichtiges Forschungsgebiet ist die Optimierung von Triangulierungen hinsichtlich verschiedener Kriterien, zum Beispiel der minimalen Kantenlängen.

In dieser Arbeit werden Triangulierungen hinsichtlich des maximalen Knotengrades analysiert. Die Höhe des Knotengrades beeinflusst etwa die Komplexität darüberliegender Anwendungen. Trotz der Bedeutung dieses Kriteriums ist es bisher noch nicht eingehend erforscht worden. Hier wird untersucht, wie sich sowohl verlustfreie als auch verlustbehaftete Modifikationen der Triangulierungen auf den maximalen Knotengrad auswirken. Dabei wurden Algorithmen entworfen, mit denen sowohl einige wenige Punkte entfernt, als auch hinzugefügt werden können, wobei die konvexe Hülle unverändert bleiben soll. Weiters wurde analysiert, welche Knotengrade damit effizient erzielt werden können. Hierfür wurde ein Programm implementiert mit dem gezielt Punktmengen mit Triangulierungen hohen Grades erstellt werden können. Auf diesen Punktmengen wurden dann die entworfenen Algorithmen getestet und die Ergebnisse statistisch ausgewertet.

Es wurde herausgefunden, dass randomisierte Punktmengen mit verlustfreien Methoden sehr gut reduzierbar waren. Bei speziell konstruierten Punktmengen mit relativ geringen Graden wurden gute Ergebnisse mit punktentfernenden Algorithmen erzielt. Wenn die Punktmengen jedoch hohe Grade aufwiesen, wurden mit punkthinzufügenden Algorithmen bessere Ergebnisse festgestellt.

Stichwörter: Triangulierungen, Knotengrade, Gradreduktion, geometrische Algorithmen

Deutsche Fassung:
Beschluss der Curricula-Kommission für Bachelor-, Master- und Diplomstudien vom 10.11.2008
Genehmigung des Senates am 1.12.2008

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 entnommene Stellen als solche kenntlich gemacht habe.

Graz, am ……. ……..
(Unterschrift)

Englische Fassung:

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.

……. ……..
date (signature)

Danksagungen

Zuerst möchte ich meinem Betreuer Herrn Professor Aichholzer für die Ratschläge und die vielen inspirierenden Ideen danken. Auch meinen Komilitonen, die mir teils geholfen, motiviert und mit Kritik und guten Anmerkungen zur Seite gestanden haben, möchte ich meinen Dank aussprechen. Vor allem Markus Plieschnegger möchte ich danken, denn ohne seine Hilfe beim Buildscript und ohne seine Ratschläge wäre ein großer Teil der Software um vieles weniger elegant ausgefallen. Weiters möchte ich Herrn DI Stefan Klampfl und meinem Vater DI Josef Rachholz dafür danken, dass sie mir bei diesem Schriftstück eine sehr große Hilfe waren, indem sie gelesen und korrigiert haben. Ein großer Dank gebührt auch meiner Familie und meinen Freunden die mir mit Rat und Tat zur Seite gestanden haben.

Graz, April 2011 Marlene Rachholz

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Triangulierungen von gegebenen Punktmengen in der Ebene sind mächtige Datenstrukturen, die in vielen Gebieten Anwendung finden (siehe [2], Kapitel 3 und 9). Eine der häufigsten Anwendungen ist im Gebiet der Computergeometrie zu finden, in der Triangulierungen meist als Grundstruktur verwendet werden, um damit komplexere Probleme zu lösen. Sie werden aber auch in der Computergraphik und in der Simulationstechnik eingesetzt, um Modelle in der Ebene und im Raum zu modellieren und damit zum Beispiel eine intelligente Interpolationsgrundlage für die Berechnung zu schaffen.

Es gilt zu beachten, dass eine gegebene Punktmenge in der Ebene und im Raum immer sehr viele verschiedene Triangulierungen besitzt. Es muss entschieden werden, welche Triangulierung aus dieser Menge aller möglichen Triangulierungen für den jeweiligen Fall optimal anzuwenden ist. Es kann zum Beispiel eine bestimmte Triangulierung für eine Anwendungen ideal sein, für andere Anwendungsgebiete dagegen nur bedingt brauchbar oder absolut ungeeignet sein. Für die Güte und Optimalität von Triangulierungen können verschiedene Kriterien herangezogen werden, wie etwa die Minimierung der Summe der Kantenlängen (die sogenannte *minimum-weight-Triangulierung* [6]) oder die Dualität zum Voronoi Diagramm (siehe [2] in Kapitel 7, Dualität in 9.2). Viele dieser Kriterien wurden bisher recht gut untersucht. Eine der am häufigsten betrachteten Triangulierungen ist die Delaunay-Triangulierung, da diese etliche der am öftesten benötigten Qualitätskriterien erfüllt. Diese Merkmale sind die Dualität zum Voronoi Diagramm, die Maximierung der Innenwinkel der zu Grunde liegenden Dreiecke und die Vermeidung annähernd kollinearer Dreiecke um in weiterer Folge Rundungsfehler in darüberliegenden Anwendungen möglichst auszuschließen. Natürlich wäre es optimal, verschiedene Qualitätskriterien gleichzeitig mit einer Triangulierung erfüllen zu können, aber oft schließen sich verschiedene Kriterien gegenseitig aus. Somit muss in diesen Anwendungsfällen ein Mittelweg zwischen den benötigten Kriterien gefunden werden. Eine Möglichkeit ist die Wahl eines primären und sekundären Merkmals, wobei das sekundäre nur dann zur Anwendung kommt, wenn das primäre nicht eindeutig ist. Es ist sehr schwer für zu erfüllende Merkmale eine optimale Triangulierung zu finden, daher wird oft mit der Näherung gearbeitet.

In dieser Arbeit soll ein weiteres, bisher recht selten betrachtetes Qualitätsmerkmal für Triangulierungen untersucht werden, das sich auf die Knotengrade bezieht. Es ist jene Triangulierung gesucht, bei der die maximale Anzahl der von einem Knoten ausgehenden Kanten für alle Knoten der Triangulierung minimiert wird.

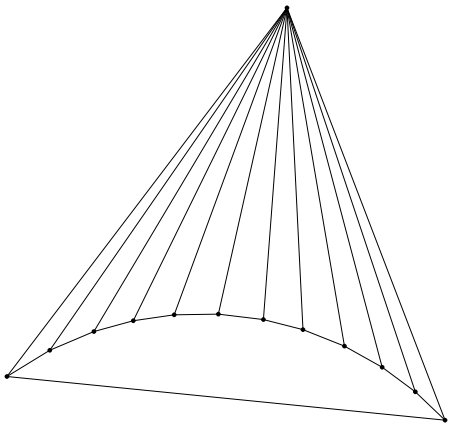


Abbildung 1.1: **Ein konkaver Fächer.** Dieses Bild zeigt einen konkaven Fächer mit allen unvermeidbaren Kanten, der aus einer Kette von Punkten auf einem Kreissegment und einem gegenüberliegenden Punkt erstellt wurde. Die Triangulierung dieses konkaven Fächers ist zwischen der Punktkette und dem gegenüberliegenden Punkt unvermeidbar. Der Grad dieses Punktes wird deshalb sehr hoch sein.

1.2 Aufgabenstellung

In dieser Arbeit soll mit einem bestimmten Qualtitätsmerkmal von Dreiecksnetzen gearbeitet werden. Dieses Merkmal bezieht sich auf den Grad einer Triangulierung. Der Grad der Triangulierung ist der maximale Knotengrad über alle Knoten, wobei der Grad eines Knotens die Anzahl der von diesem Knoten ausgehenden Kanten ist. Es soll untersucht werden, mit welchen Methoden man zu einer Triangulierung gelangen kann, die dieses Kriterium erfüllt. Da es sehr aufwändig ist, die gradmäßig optimale Triangulierung zu ermitteln, müssen wir uns mit einer Näherung an diese begnügen (vgl. [6] für die Komplexität solcher Probleme).

Selbst wenn die gradmäßig optimale Triangulierung vorliegt, besteht das Problem, dass der Grad sehr hoch sein kann. Um den Grad der zugrunde liegenden Triangulierung zu reduzieren, muss die darunterliegende Punktmenge verändert werden. Das ist notwendig, da es sogenannte unvermeidbare Kanten gibt, also Kanten die in jeder möglichen Triangulierung der Punktmenge vorhanden sind. Zu diesen Kanten zählen immer die Kanten der konvexen Hülle und alle Kanten, die nicht von anderen möglichen Kanten gekreuzt werden.

Hier sollen auch Situationen untersucht werden, die nur aus unvermeidbaren Kanten besteht und einen sehr hohen Grad eines der beteiligten Knoten erzeugten (ein sogenannter *konkaver Fächer*, siehe Abbildung 1.1). In diesem Fall kann der Grad nur durch Methoden gesenkt werden, die die Punktmenge in diesem Bereich verändern. Solche Methoden werden auch verlustbehaftete oder aggressive Optimierungsmethoden genannt. Diese sollten mit relativ wenigen Schritten durchgeführt werden, um möglichst wenig Information der Punktmenge zu verlieren.

In dieser Arbeit sollen zunächst Methoden untersucht werden, die die darunterliegende

Punktmenge nicht verändern, z.B., durch Flippen von Kanten (siehe [2] auf Seite 184). Mit verlustbehafteten Optimierungsmethoden soll der Grad durch möglichst wenige aggressive Schritte, die die Punktmenge verändern, auf einen Wunschgrad gesenkt werden. Der Bereich der Grade, auf die man ohne große Veränderung der Punktmenge reduzieren kann, soll ebenfalls herausgefunden werden. Hierfür sollen randomisierte Punktmengen verwendet werden. Die Methoden, die die Punktmenge gezielt verändern, sollen mit statistischen Mitteln auf ihre Effizienz hin überprüft werden. Hierfür müssen Punktmengen konstruiert werden, bei denen ein sehr großer Teil der Triangulierung eindeutig ist. Bei diesen Punktmengen sollen Optimierungsmethoden, welche die Punktmenge nicht verändern, ausgeschlossen werden, d.h., sie sollen den gewünschten Grad nicht erreichen. Weiters soll untersucht werden, welche Anfangstriangulierung die wengisten Optimierungsschritte braucht. Dies soll mit verschiedenen bekannten Triangulierungen realisiert werden.

Um die gesamte Thematik dieser Arbeit effizient untersuchen zu können, soll eine Software erstellt werden. Da man sehr viele bekannte geometrischen Datenstrukturen und Algorithmen verwendet, ist die Wahl der Programmiersprache auf C++ mit der bereits bestehenden CGAL Bibliothek [7] gefallen, die sehr viele Lösungsalgorithmen für geometrische Problemstellungen beinhaltet.

1.3 Gliederung

Diese Arbeit ist in etliche Kapitel gegliedert. Kapitel 1 beinhaltet die Einführung und die Motivation. Kapitel 2 enthält Definitionen, die in dieser Arbeit gebraucht werden. In Kapitel 3 werden Situationen in Punktmengen untersucht, die auf jeden Fall Triangulierungen mit hohen Graden erzeugen, um Testpunktmengen erstellen zu können, die nicht mit verlustfreien Methoden reduziert werden können. In Kapitel 4 werden die verwendeten Triangulierungsarten vorgestellt und es wird gezeigt, warum gewisse bereits bekannte Triangulierungsarten nicht optimal bezüglich des Grades sein können. Kaptiel 5 behandelt verlustfreie Optimierungsmethoden, die nur durch Änderungen der Triangulierung einer Punktmenge den Grad der Triangulierung senken. In Kapitel 6 werden verlustbehaftete Reduktionsmethoden behandelt, die den Grad mit einer gezielten Änderung der zugrundeliegenden Punktmenge senken soll. In Kapitel 7 wird die implementierte Software und die verwendeten Algorithmen vorgestellt. Außerdem wird hier gezeigt, wie die Software funktioniert und wie sie zu bedienen ist. Die Auswertung der Testläufe und die Analyse der Ergebnisse werden in Kapitel 8 gezeigt. Kapitel 9 beinhaltet das Resümee.

2 Definitionen

Diese Arbeit befasst sich mit einem großen Kapitel aus dem Bereich der geometrischen Algorithmen, den Triangulierungen. Zum Verständnis dieser Arbeit sind Grundkenntnisse von geometrischen Algorithmen und deren Begriffe Voraussetzung. Weiters ist die Kenntnis gewisser mathematischer Grunddefinitionen nötig. Trotzdem werden in diesem Kapitel einige der wichtigsten Definitionen aus dem Bereich der geometrischen Algorithmen genauer erläutert. Diese Definitionen sind zwar selbst verfasst, sind aber in sehr ähnlicher Form im Buch [1] enthalten. Für einen generellen Einstieg die Algorithmik ist dieses Buch sehr zu empfehlen.

Definition 2.1 (Konvexe Hülle einer Punktmenge)**.** Sei $P = \{p_1, \ldots, p_n\}$ eine Punktmenge in der Ebene. Die konvexe Hülle von P, kurz $KH(P)$, ist das kleinste konvexe Polygon, das P enthält, also $KH(P) = \bigcap_{S \supset P} S$.

Die konvexe Hülle einer Punktmenge wird durch die äußersten Punkte einer Menge beschrieben. Die konvexe Hülle ist ein zentrales geometrisches Konstrukt und kann für n Punkte in der Ebene in $O(n \log n)$ Zeit berechnet werden. Eine genauere Abhandlung über konvexe Hüllen kann zum Beispiel in [2] in Kapitel 1 und 11 gefunden werden.

Definition 2.2 (Triangulierung)**.** Eine Triangulierung $T = (V, E)$ einer Menge von Punkten P in der Ebene bezeichnet eine vollständige Zerlegung der konvexen Hülle der Punktmenge in disjunkte Dreiecke, wobei die Eckpunkte V der Dreiecke die aus P erzeugten Knoten sind. $V(p)$ ist der von p erzeugte Knoten, E ist die Menge aller Kanten. $\mathcal{T}(P)$ ist die Menge aller Triangulierungen einer Punktmenge P.

Der Rand jeder gültigen Triangulierung einer Punktmenge ist die konvexe Hülle. Im Zuge der Arbeit wird die Konvexe Hülle über die Triangulierung selbst erstellt. Triangulierungen und deren Aufbau sind unter [2] in Kapitel 3 und 9 erklärt.

Definition 2.3 (Gewicht einer Triangulierung)**.** Eine gewichtete Triangulierung besitzt eine Zuordnung $w : E \to \mathbb{R}_+$. Das Gewicht einer Triangulierung $w(T) := \sum_{e \in E} w(e)$ ist die Summe der Kantengewichte.

Ein Beispiel für Kantengewichte ist die Länge der Kanten.

Definition 2.4 (Minimum-Weight Triangulierung)**.** Die Minimum-Weight Triangulierung (MWT) ist jene Triangulierung aus der Menge der möglichen Triangulierungen einer gegebenen Punktmenge P, deren Gewicht am kleinsten ist: $T_{MW} := \min_{T \in \mathcal{T}} w(T)$

Sie kann unter Umständen nicht eindeutig sein, wenn es mehrere Möglichkeiten gibt die Punktmenge mit gleichem Gewicht zu triangulieren. Das Finden der MWT ist sehr aufwändig, da dieses Problem zur Klasse der NP-schweren Probleme gehört; dies bedeutet, dass hierfür kein polynomialer Algorithmus bekannt ist. Der Beweis hierfür steht in [6].

Definition 2.5 (Delaunay-Triangulierung)**.** Die Delaunay-Triangulierung DT ist eine Triangulierung aus $\mathcal{T}(P)$, die die Eigenschaft erfüllt, dass im Umkreis jedes Dreiecks der Triangulierung keine weiteren Knoten sind.

Diese Triangulierung ist bis auf Äquivalenzen eindeutig. Mehrdeutigkeiten entstehen nur dadurch, dass ein weiterer Punkt auf dem Umkreis eines Dreiecks liegt. Die Delaunay-Triangulierung erfüllt einige der bekanntesten Merkmale, wie zum Beispiel der Dualität zum Voronoi Diagramm. Ein weiteres Merkmal ist, dass die Maximierung der Winkelgrößen der Dreiecke der Triangulierung. Letztere Eigenschaft verhindert in weiterer Folge Rundungsfehler und macht die Delaunay-Triangulierung zu einer in der Praxis am häufigsten eingesetzten Triangulierungen. Sie sind in Kapitel 9 in [2] genauer erklärt.

Definition 2.6 (Greedy-Triangulierung)**.** Die Greedy-Triangulierung entsteht durch iteratives Einfügen von Kanten, wobei in jedem Schritt die kürzestmögliche Kante eingefügt wird, die sich mit keiner der schon eingefügten Kanten schneidet.

Diese Triangulierung ist eine - allerdings eine oft sehr schlechte - Annäherung an die MWT. Es existieren bereits relativ effiziente Algorithmen für den Erhalt der Greedy-Triangulierung, wobei hier aus der Menge aller möglichen Kanten durch eine Vorauswahl Kanten gelöscht werden, die unmöglich in einer Triangulierung mit diesem Merkmal sein können. Leider funktionieren die meisten dieser Algorithmen nur dann, wenn eine Punktmenge vorhanden ist, deren Punkte gleichverteilt auf der Ebene liegen. Aus diesem Grund sind diese Algorithmen für die in dieser Arbeit verwendeten Implementationen nicht brauchbar. Ein sehr effizienter Ansatz ist unter [3] zu finden.

Definition 2.7 (Grad eines Knotens, Grad einer Triangulierung)**.** Der Grad eines Knotens $\deg(P_k)$ ist die Anzahl aller Kanten der Triangulierung, die diesen Knoten als Endpunkt haben. Der Grad einer Triangulierung ist der maximale Knotengrad in dieser Triangulierung: $\deg(T) := \max_{P_k \in P} \deg(P_k)$.

Definition 2.8 (Gradoptimale Triangulierung)**.** Diese Triangulierung ist die bezüglich des Grades optimale Triangulierung: $T^* := \min_{T \in \mathcal{T}} \deg(T)$

Es wird vermutet, dass die Berechnung dieser Triangulierung NP-schwer ist.

Definition 2.9 (Konvexe Vierecke in Triangulierungen)**.** Sei $e \in E$ eine Kante von $T(V, E)$ und $e \notin KH(P)$. Dann hat e zwei benachbarte Dreiecke D_1 und D_2 in der Triangulierung $T(V, E)$. D_1 und D_2 bilden immer ein Viereck V_4. Diese Dreiecke bilden genau dann ein *konvexes Viereck* KV_4, wenn die konvexe Hülle dieses Vierecks aus vier Punkten gebildet wird. Ansonsten bilden sie ein *nicht konvexes Viereck*.

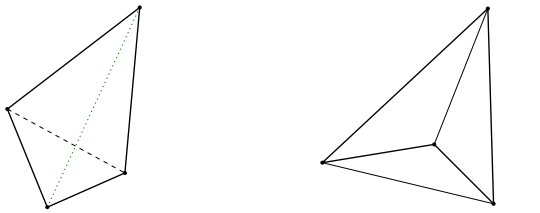


Abbildung 2.1: **Vergleich zwischen konvexen und nicht konvexen Vierecken.** Bei konvexen Vierecken besteht die konvexe Hülle aus vier Punkten, und die Diagonalen kreuzen sich. Nicht konvexe Vierecke besitzen eine konvexe Hülle mit 3 Punkten, und die Diagonalen kreuzen sich nicht. Daher kann bei konvexen Vierecken gewählt werden, welche Diagonale Teil der Triangulierung ist. Das Ersetzen der schwarzen, strichlierten durch die grüne, punktierte Diagonale nennt man Flip.

In Abbildung 2.1 ist der Unterschied deutlich zu sehen. Die konvexe Hülle der linken Abbildung hat vier Eckpunkte wobei die Konvexe Hülle der rechten Abbildung drei Punkte enthält.

Definition 2.10 (Kantenflips in Triangulierungen)**.** Sei KV_4 ein konvexes Viereck und e die zugrunde liegende Kante in $T(V, E)$. Dann ist ein Kantenflip von e möglich, indem die Kante e entfernt und die andere Diagonale von KV_4 eingefügt wird.

Damit kann aus einer gültigen Triangulierung $T(V, E)$ eine andere gültige Triangulierung $T(V, \hat{E})$ erstellt werden. Der Kantenflip beeinflusst nur die Punkte von KV_4 und die zugehörige Kante e. Somit wird auch nur der Grad der vier Eckpunkte von KV_4 verändert. Mit endlich vielen Kantenflips kann man von einer gültigen Triangulierung $T(V, E)$ zu jeder anderen gültigen Triangulierung in $\mathcal{T}(P)$ gelangen (siehe [4]). In Abbildung 2.1 ist zu sehen warum bei einem konvexen Viereck ein Kantenflip möglich ist, denn die schwarze, strichlierte Kante innerhalb des Vierecks kann jederzeit durch die grüne, punktierte ersetzt werden und umgekehrt (Definition siehe [2] auf Seite 184).

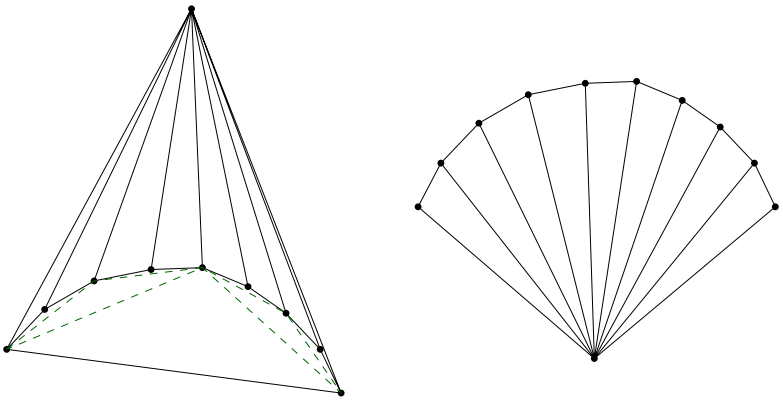
Definition 2.11 (Unvermeidbare Triangulierungskanten; Unvermeidbare Teiltriangulierung)**.** Unvermeidbare Triangulierungskanten sind Kanten e von $T(V, E)$ die nicht durch andere Kanten aus einer beliebigen Triangulierung $T \in \mathcal{T}(P)$ gekreuzt werden. Ein Bereich einer Triangulierung $T(V, E)$ wird unvermeidbare Teiltriangulierung genannt, wenn alle Triangulierungskanten dieses Bereichs unvermeidbar sind.

Damit gibt es keine Möglichkeit diese Kanten durch andere mögliche Kanten aus anderen möglichen Triangulierungen zu ersetzen. Diese Kanten müssen in jeder möglichen Triangulierung vorhanden sein. Die Kanten der konvexen Hülle sind unvermeidbar. Eine andere Situation mit sehr vielen unvermeidbaren Kanten ist der konkave Fächer, der noch genauer erläutert wird. Für eine unvermeidbare Teiltriangulierung gibt es keine Triangulierung aus $\mathcal{T}(P)$, die in diesem Bereich nicht eindeutig ist.

Definition 2.12 (Konkave Fächertriangulierung; Konvexe Fächertriangulierung)**.** Eine unvermeidbare Teiltriangulierung einer Punktmenge ist genau dann eine konkave Fächertriangulierung, wenn eine Punktkette entlang eines Kreissegments einem einzelnen Punkt auf der Seite gegenüberliegt, die nicht den Mittelpunkt des Segments enthält.

Bei einer konvexen Fächertriangulierung liegt der einzelne Punkt auf der Seite des Mittelpunkts des Segments.

Eine konkave Fächertriangulierung hat nur im Inneren des Kreissegments vermeidbare Kanten (in Abbildung 2.2 links), der konvexe Fächer besteht hingegen (in Abbildung 2.2 rechts), mit Ausnahme der konvexen Hülle, nur aus vermeidbaren Kanten.



konkaver Fächer konvexer Fächer

Abbildung 2.2: **Ein konkaver und ein konvexer Fächer.** Beide Fächer werden von einer Kette aus Punkten auf einem Kreissegment und einem externen Punkt erzeugt. Beim konvexen Fächer liegt der Punkt auf der Seite des Mittelpunktes des Kreissegments, beim konkaven Fächer auf der gegenüberliegenden Seite. Der konkave Fächer besitzt viele unvermeidbare Kanten, wogegen beim konvexen Fächer nur die konvexe Hülle des Fächers unvermeidbar ist. Die grünen, strichlierten Kanten sind nicht unvermeidbare Kanten des konkaven Fächers. Sie liegen immer innerhalb des zugrundeliegenden Segments, durch das die Punktkette definiert wurde.

Bei einem konvexen Fächer ist nur die konvexe Hülle des Fächers unvermeidbar, alle Kanten innerhalb der konvexen Hülle sind vermeidbar. Dies ist ein Beispiel für das Erzeugen eines unnötig hohen Grades.

Definition 2.13 (Nicht-aggressive Schritte; Aggressive Schritte)**.** Nicht-agressive Schritte sind Transformationen von $T(V, E) \rightarrow T(V, \hat{E})$, die angewendet werden um beispielsweise $\deg(v)$ zu senken. Sie verändern weder P noch V sondern nur E. Aggressive Schritte sind Methoden von $T(V, E) \rightarrow T(\hat{V}, \hat{E})$, die ebenso $\deg(v)$ senken sollen. Diese Schritte dürfen Kanten E, Knoten V und die zugrundeliegende Punktmenge P von $T(V, E)$ verändern.

Nicht-Aggressive Schritte dürfen die zugrunde liegende Punktmenge nicht verändern. Beispiele hierfür sind Kantenflips oder das Entfernen und das neuerliche Einfügen an

der gleichen Stelle von $p \in P$. Aggressive Schritte bestehen meist aus dem Entfernen oder Hinzufügen von einzelnen ausgewählten Punkten. Für eine Gradreduktion werden üblicherweise alle möglichen nicht-aggressiven Schritte ausgeführt, bevor man auf die aggressiven Schritte zurückgreift.

Definition 2.14 (Gradvektor eines konvexen Vierecks)**.** Sei KV_4 ein konvexes Viereck. Der Gradvektor eines konvexen Vierecks $\mathrm{Deg}(KV_4)$ ist ein absteigend sortierter Vektor mit den Graden der vier Eckpunkte $\deg(v) : v \in KV_4$ als Einträgen.

Dieser Vektor wird herangezogen um die Effektivität eines gültigen Kantenflips bezüglich der Grade berechnen zu können.

Definition 2.15 (Radiale Punktketten)**.** Sei KS ein Kreissegment mit Radius $r > 0$ und Winkel ω. Dann ist RPK eine Punktkette, die aus $m > 3$ äquidistanten Punkten entlang von KS gebildet wird, wobei die Endpunkte von K mit denen von KS übereinstimmen, eine *radiale Punktkette* mit Radius r und m Punkten.

3 Testpunktmengen

3.1 Einführung

Um Algorithmen, die mit dem Grad der Triangulierung arbeiten, aussagekräftig zu testen, werden gezielt Testpunktmengen erstellt. Randomisierte Punktmengen haben im allgemeinen Fall relativ niedrige Grade, wenn wenige, nichtaggressive Optimierungsschritte durchführt werden. Für beliebige Punktmengen gibt es normalerweise immer Triangulierungen, die einen hohen Grad haben; um jedoch aussagekräftig testen zu können, gilt es Punktmengen zu erzeugen, die nur Triangulierungen mit hohem Maximalgrad erzeugen. Es handelt sich hierbei um Punktmengen, deren zugehörige Triangulierungen sehr viele unvermeidbare Kanten und einen Knoten mit hohem Grad haben (unvermeidbare Kanten, vgl. stable lines in [5]).

Es gibt nicht viele Fälle, deren Triangulierungen viele unvermeidbare Kanten besitzen, deshalb müssen wir mit einer bekannten Situation, die auf konkave Fächer aufbaut, arbeiten. Darum wird im ersten Teil dieses Konstrukt genauer untersucht. Um die Bandbreite der Datensets zu erhöhen, werden im zweiten Teil des Kapitels noch mögliche Abänderungern vorgeschlagen und untersucht.

3.2 Konkave Fächer

Konkave Fächer bestehen immer aus Punktketten, die auf Kreissegmenten liegen, und mindestens einem Punkt, der auf der Seite des Mittelpunkts dieses Kreissegments liegt (die konkave Seite des Kreissegments). Um genauer untersuchen zu können, welche Auswirkungen ein zusätzlicher Punkt hat, muss auf die Lage des Punktes geachtet werden. Wir müssen dazu gewisse Bereiche und Zonen untersuchen, die aus der Punktkette, den Tangenten in den Endpunkten und der Kante zwischen den beiden Endpunkten gebildet werden.

An dieser Stelle sollen ein paar zusätzliche Definitionen, die in diesem Kaptiel notwendig sind, angeführt.

Definition 3.1 (Zonen von Radialen Punktketten mit Winkel kleiner als 180°)**.** Sei RPK eine radiale Punktkette mit ω <180°. Dann sind die Zonen (konkave Zone, echt konkave Zone, konvexe Zone, innere Zone, linker und rechter toter Winkel) dieser radialen Punktkette wie in Abbildung 3.1 definiert.

Definition 3.2 (Zonen von Radialen Punktketten mit Winkel größer gleich 180°)**.** Sei RPK eine radiale Punktkette mit ω >180°. Dann sind die Zonen (konkave Zone, konvexe

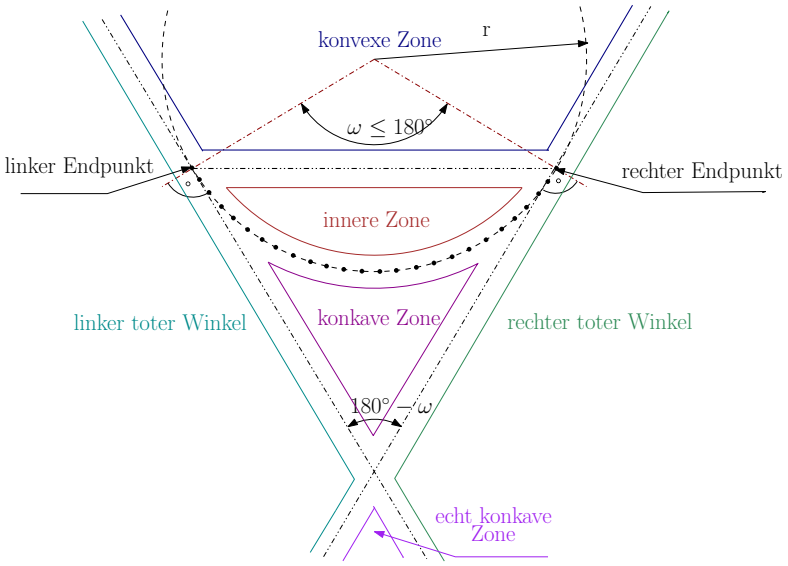


Abbildung 3.1: **Radiale Punktkette mit Winkel $\omega \leq 180°$.** Im Bild sind die konvexe, konkave, echt konkave und innere Zone, sowie die beiden toten Winkel eingezeichnet. Es kann nur bei einem Winkel kleiner als 180° einen Kreuzungspunkt geben, der auf der konkaven Seite des Kreissegments liegt. Dadurch gibt es in diesem Fall die echt konkave Zone.

Zone, innere Zone, linker und rechter toter Winkel) dieser radialen Punktkette wie in Abbildung 3.2 definiert.

Bemerkung 3.1. Die Namen der Zonen wurden sprechend gewählt; So beschreibt zum Beispiel die innere Zone den Bereich im Inneren des Kreissegments. Der Name der konvexen beziehungsweise konkaven Zone stammt aus der Mathematik und der Definition von konvex bzw. konkav. Die toten Winkel erinnern an die toten Winkel eines Rückspiegels, bei denen ein Objekt umso schwerer gesehen werden kann, je weiter hinten im toten Winkel es sich befindet.

Wir betrachten nun den elementaren Fall, dass ein Punkt hinzugefügt wird. Da wir festgelegt haben, dass die Punkte nicht kollinear sein dürfen, kann der Punkt immer nur in einer Zone liegen. Vorerst werden die Auswirkungen der Lage dieses Punktes und die Größe der durch die Anordnung entstehenden Fächer untersucht.

3.2.1 Eigenschaften von Triangulierungen radialer Punktketten und einem zusätzlichen Punkt

Satz 3.1. (Auswirkungen des Hinzufügen eines Punktes in der inneren Zone einer radialen Punktkette) *Sei RPK eine radiale Punktkette und p ein Punkt in der inneren Zone der Punktkette. Dann sind nur die Kanten der konvexen Hülle unvermeidbar und es existiert eine Triangulierung mit Grad fünf.*

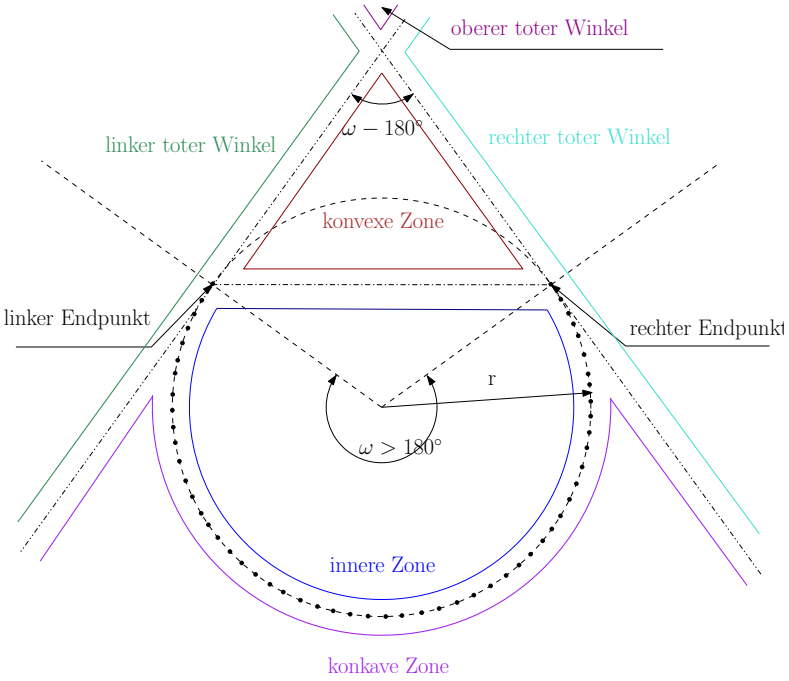


Abbildung 3.2: **Radiale Punktkette mit Winkel $\omega \geq 180°$.** Im Unterschied zum vorigen Bild gibt es hier keine echt konkave Zone, dafür gibt einen oberen toten Winkel. Die Eigenschaften der Zonen sind gleich wie in Abbbildung 3.1, jedoch kann die Größe des Fächers für einen fixen Punkt variieren.

Beweis. Gestartet wird mit einer Triangulierung von RPK. Es werden nun die beiden Endpunkte betrachtet und ein Dreieck wird mit einem der Nachbarpunkte auf der Punktkette, o.B.d.A. der nächste Nachbar des linken Endpunktes gebildet. Nun wird so lange ein Dreieck mit dem gegenüberliegenden nächsten Punkt gebildet, bis die Punktkette vollständig trianguliert ist, siehe Abbildung 3.3. Wie zu sehen ist, hat diese Triangulierung Grad vier. Wenn nun ein Punkt p im Inneren der Triangulierung eingefügt wird, so kann er, weil die Punkte nicht kollinear sein dürfen, nur im inneren eines dieser vorhandenen Dreiecke liegen. Somit können aus diesem Dreieck drei Dreiecke gebildet werden, indem der Punkt p durch eine Kante mit den Punkten des Dreiecks verbunden werden. Durch diesen Schritt erhöht sich nur der Grad der Dreieckspunkte des betroffenen Dreiecks um eins und es wurde eine Triangulierung gefunden, deren Grad fünf beträgt.

□

Bemerkung 3.2. Diese Triangulierung besitzt definitiv nicht die kürzesten Kanten und genügt auch nicht der Umkreiseigenschaft. Sie ist bezüglich anderer Merkmale wie beispielsweise die Kantenlängen denkbar schlecht. Trotzdem ist sie bezüglich des Grades sehr gut und wird im weiteren Verlauf dieser Arbeit sehr häufig verwendet. Sie ist ein Beispiel dafür, dass eine Triangulierung gut bezüglich eines Merkmals sein kann und gleichzeitig absolut schlecht bezüglich eines anderen.

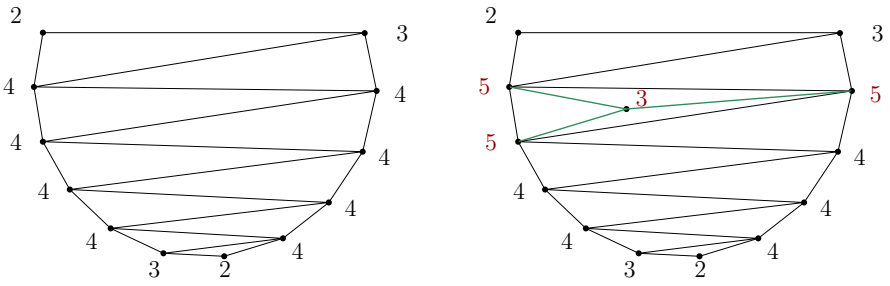


Abbildung 3.3: **Einfügen eines Punktes in der inneren Zone.** Links im Bild ist eine bezüglich des Grades gute Triangulierung der inneren Zone zu sehen. Die Grade bewegen sich im Bereich zwischen 2 und 4. Es ist auch sofort zu erkennen, dass diese Triangulierung nicht die kürzesten Kantenlängen besitzt, denn eine Triangulierung in der jeder Punkt mit dem nächsten und übernächsten ein Dreieck bildet, würde dies besser erfüllen. Im Bild rechts sieht man welche Auswirkungen ein hinzugefügter Punkt hat und dass keine weiteren Operationen benötigt werden (außer dem Einfügen), um eine Triangulierung des gesamten Systems mit geringen Graden zu erhalten.

Satz 3.2. (Auswirkungen des Hinzufügen eines Punktes in der konvexen Zone einer radialen Punktkette) *Sei RPK eine radiale Punktkette und p ein Punkt in der konvexen Zone der Punktkette. Dann wird p zur konvexen Hülle hinzugefügt und es gibt eine Triangulierung mit Grad vier.*

Beweis. Es wird eine Triangulierung in der inneren Zone wie im Beweis von Satz 3.1 erzeugt. Da p nicht in der inneren Zone liegt, kann er nicht innerhalb der konvexen Hülle liegen. p muss also zur der konvexen Hülle der Punktkette hinzugefügt werden. Um einen Punkt zur konvexen Hülle hinzuzufügen, müssen die vom Punkt ausgehenden Tangenten auf die konvexe Hülle betrachtet werden. Trianguliert wird, indem p mit jedem Punkt der konvexen Hülle verbunden wird, der zwischen den beiden Tangenten auf KH liegt. Wenn nun die Beschaffenheit der konvexen Zone betrachtet wird, so ist zu sehen, dass die konvexe Zone mit den Tangenten in den Endpunkten des Segments beschränkt ist. Somit kann innerhalb der Tangenten auf die KH kein weiterer Punkt liegen und p wird nur mit den beiden Endpunkten verbunden und bildet damit ein Dreieck. Dieser Schritt erhöht nur den Grad der Endpunkte. Da diese in der oben gezeigten Triangulierung Grad zwei beziehungsweise drei haben, können sie nach dem Einfügen des Punktes höchstens Grad vier haben. (siehe Abbildung 3.4)

□

Bemerkung 3.3. Diese ersten beiden Fälle sind offensichtlich für die Erzeugung von Punktmengen, deren Triangulierungen hohe Grade aufweisen sollen nicht gut. Sie sollten bei der Erstellung von Testdaten vermieden werden.

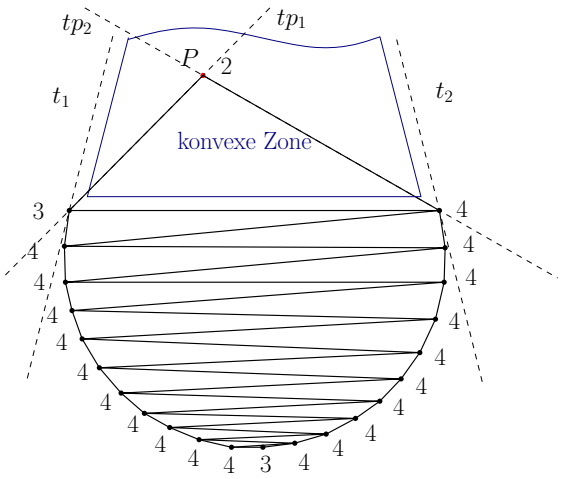


Abbildung 3.4: **Einfügen eines Punktes in der konvexen Zone.** Hier ist zu sehen, dass ein Punkt in der konvexen Zone mit geringem Grad in die Triangulierung eingefügt werden kann. Dafür werden nur die Kanten entlang der Tangenten tp_1 und tp_2 erzeugt. Da die Triangulierung im Inneren bestehen bleibt, erhöhen sich die Grade der Triangulierung durch das Hinzufügen des Punktes nicht wesentlich.

Satz 3.3. (Auswirkungen des Hinzufügen eines Punktes in der konkaven Zone einer radialen Punktkette) *Sei RPK eine radiale Punktkette und p ein Punkt in der konkaven Zone der Punktkette. Dann bildet die Triangulierung in der konkaven Zone immer einen konkaven Fächer und der Grad der Triangulierung beträgt m, die Anzahl der Punkte auf RPK, die zwischen den Tangenten des Punktes p auf dem zugrundeliegenden Kreis liegen.*

Beweis. Sei T die Triangulierung der Punktkette. Dann gehören alle Kanten entlang der Punktkette und die Kante, die zwischen den beiden Endpunkten entsteht, zur KH von T. Um p hinzuzufügen, müssen die Tangenten tp_1 und tp_2 von p auf den zugrundeliegenden Kreis ermittelt werden. Es sind drei Fälle zu betrachten:

- *Fall 1: $\omega = 180°$.*

  Aufgrund der Definition der Zone sind die beiden begrenzenden Tangenten t_1 und t_2 in den Endpunkten des Kreissegmentes parallel. Werden die Tangenten des Punktes p (tp_1 und tp_2) auf den Kreis erzeugt, so werden zwei Berührungspunkte q_1 und q_2 erstellt. Um den Punkt p hinzuzufügen, müssen alle Punkte, die zwischen q_1 und q_2 entlang des Kreissegments liegen, durch Kanten mit p verbunden werden. Damit wird eine Teiltriangulierung dieses Bereichs erzeugt, bestehend aus den Dreiecken, die durch Nachbarpunkte von RPK und p generiert wurden.

  Seien nun D_1 und D_2 zwei beliebige benachbarte Dreiecke die so entstanden sind und p enthalten. Aufgrund der Krümmung können D_1 und D_2 kein konvexes Viereck bilden (siehe Abbildung 3.5 links). Daher können die Kanten zwischen p und Punkten aus RPK nicht flipbar sein.

Übrig bleiben in diesem Bereich nur noch die Kanten zwischen benachbarten Punkten auf RPK. Damit eine Kante flipbar ist, müssen die benachbarten Dreiecke dieser Kante ein konvexes Viereck bilden. Dazu müsste ein Punkt r zwischen den beiden blauen Halbgeraden (also im schraffierten Bereich) in Abbildung 3.5 rechts liegen. Da ω aber 180° beträgt, kann so ein Punkt nicht existieren. Es kann keine dieser so entstandenen Kanten geflippt werden und die Teiltriangulierung besteht nur aus unvermeidbaren Kanten. Der Punkt p hat einen Grad, der der Anzahl der von ihm ausgehenden Kanten entspricht, was in diesem Fall die Anzahl der Punkte auf RPK, die zwischen q_1 und q_2 liegen, ist.

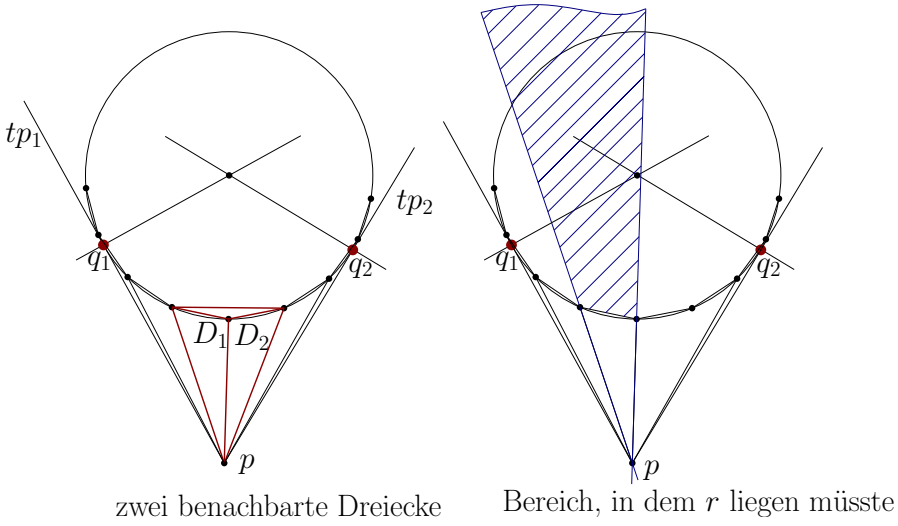


Abbildung 3.5: **Mögliche Flips eines konkaven Fächers.** Links: Kanten zwischen p und Punkten von RPK. D_1 und D_2 sind die an diese Kante direkt angrenzenden Dreiecke. Es ist zu erkennen, dass D_1 und D_2 zusammen kein konvexes Viereck bilden können. Rechts: Kanten zwischen benachbarten Punkten von RPK. Das Dreieck unterhalb der Kante ist bereits fix vorgegeben, damit das zweite Dreieck diese Voraussetzung erfüllt, muss der weitere Punkt r im blauen Bereich liegen. Dieser Bereich wird durch die Verlängerung der Kanten des ersten Dreiecks definiert.

- *Fall 2:* $\omega < 180°$. Der einzige Unterschied zum obigen Fall ist, dass die Tangenten in den Endpunkten sich hier schneiden und einen Punkt k erzeugen. Es wird nun unterschieden, ob der Punkt p in der konkaven oder in der echt konkaven Zone liegt. Liegt p in der echt konkaven Zone, so hat tritt der Fall auf, dass alle Punkte von RPK innerhalb der vom Punkt p erzeugten Tangenten tp_1 und tp_2 sind (siehe Abbildung 3.6 links). Damit besteht der Fächer aus allen Punkten von RPK und der Grad von p beträgt m, der Anzahl der Punkte von RPK.

  Sollte p nicht in der echt konkaven Zone liegen, so liegen die Strecken vom Punkt p entlang der Tangenten tp_1 und tp_2 bis zum Kreis komplett in der konkaven Zone (siehe Abbildung 3.6 rechts). Das bedeutet, dass die Punkte rechts und links der

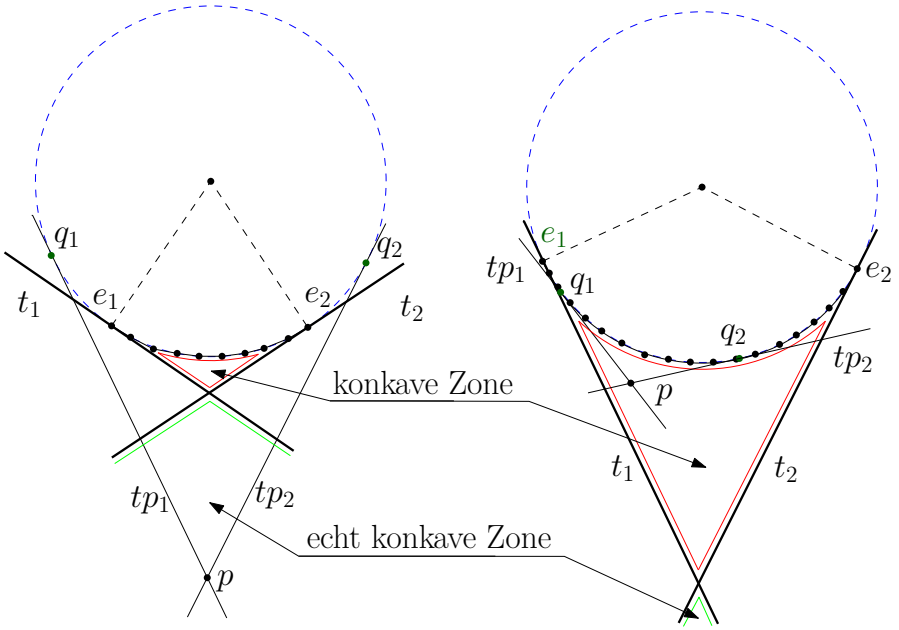


Abbildung 3.6: **Unterschied zwischen dem Hinzufügen eines Punkts in konkaver und echt konkaver Zone.** Links: eine RPK mit Winkel $\omega \leq 180°$, in deren echt konkaver Zone ein Punkt eingefügt wurde. Es ist zu sehen, dass die Berührungspunkte der Tangenten tp_1 und tp_2, q_1 und q_2 nicht im Bereich von RPK liegen. Dadurch gehören alle Punkte von RPK zum konkaven Fächer (nicht eingezeichnet). Rechts: ein Punkt wird in der konkaven Zone eingefügt. Die Berührungspunkte der Tangenten q_1 und q_2 sind im Bereich von RPK und es werden links und rechts Punkte vom konkaven Fächer weggeschnitten.

jeweiligen Tangente vom Punkt p aus nicht direkt erreicht werden können. Alle m Punkte, die zwischen den beiden Tangentenberührungspunkten liegen, erzeugen mit p einen Fächer. Daher hat auch p Grad m.

- *Fall 3:* $\omega > 180°$.

  Dieser Fall ist fast ident zu Fall 1, der einzige Unterschied besteht darin, dass unter Umständen die Kanten entlang von RPK nicht unvermeidbar sind, da es einen Punkt l geben könnte, der mit p und zwei benachbarten Punkten auf RPK ein konvexes Viereck bildet. Somit wäre diese Kante flippbar. Die Auswirkungen eines solchen Flips sind allerdings nicht besonders wünschenswert, denn die Kanten zwischen p und Punkten von RPK sind nach wie vor nicht vermeidbar und jeder dieser Flips erhöht den Grad von p um eins, was wiederum den Grad der Triangulierung erhöht. Die Situation könnte dann wie in Abbildung 3.7 aussehen.

□

Bemerkung 3.4. In Abbildung 3.5 links ist zu sehen, dass drei auf RPK benachbarte Punkte mit p immer nur ein nicht konvexes Viereck bilden können. Das resultiert aus

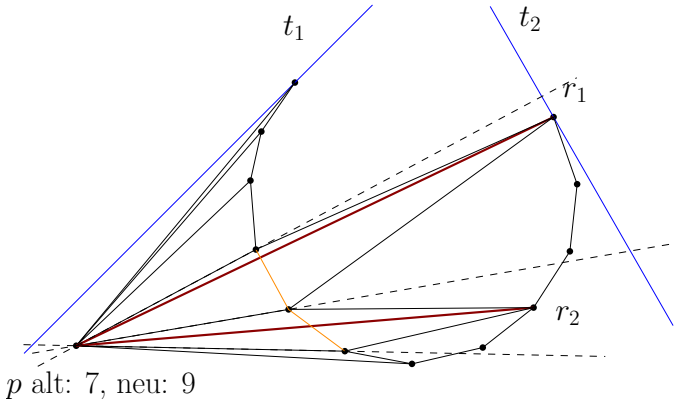


Abbildung 3.7: **Mögliche Flips eines konkaven Fächers.** Dieses Bild zeigt die einzige Möglichkeit, bei einem konkaven Fächer zu flippen. Dazu wird eine *RPK* mit Winkel größer 180° und einen Punkt in der konkaven Zone benötigt. Der Flip muss ein Dreieck des Fächers beeinflussen, und wie in Abbildung 3.5 rechts zu sehen ist, muss einer der Punkte im vorgegebenen Bereich (hier strichliert eingezeichnet) liegen. Der Flip ist möglich und es sind auch die Auswirkungen zu sehen, denn der Grad des Punktes p wird dadurch erhöht.

der Krümmung zwischen den drei Punkten entlang von *RPK*. Genau aus diesem Grund kann die Kante zwischen dem mittleren dieser drei Punkte und p, die die beiden Dreiecke teilt, nie geflippt werden.

Bemerkung 3.5. *(Auswirkungen des Hinzufügen eines Punktes in einem toten Winkel einer radialen Punktkette)* Sei *RPK* eine radiale Punktkette und p ein Punkt einem der toten Winkeln der Punktkette. Dann wird ein konkaver Fächer gebildet dessen Größe (dh. die Anzahl der beteiligten Punkte) sehr stark vom Winkel der Geraden $p\bar{m}$ zu $e_1\bar{m}$ bzw. $e_2\bar{m}$ abhängt.

Angenommen $\hat{r}$ ist die Entfernung von Mittelpunkt des Kreises m zu p. Man bilde einen Kreis mit Radius $\hat{r}$ und lässt den Punkt p von einem Ende des toten Winkels zum anderen Ende entlangwandern, zwischen den Schnittpunkten mit den Tangenten t_1 und t_2. Diese Schnittpunkte liegen jeweils am Rand des toten Winkels, daher sind sie auch Teil der angrenzenden Zone. Je näher p an der konkaven Zone liegt, desto größer muss der Fächer sein. Wenn er jedoch in der Nähe der konvexen Zone liegt, so ist der Fächer und damit der Grad von p klein. In Abbildung 3.8 wird gezeigt, wie sich die Größe des Fächers aufgrund der Lage verändert. Punkt p_0, der direkt auf dem Schnittpunkt zwischen Tangente und Kreis liegt, hat hier den größten Grad. Je weiter der Punkt entlang des Kreises weiterwandert, desto kleiner wird der Grad bis schließlich bei Punkt p_3.

Satz 3.4. (Platzierung eines Punktes bei einer *RPK* mit Winkel kleiner als 180°) *Sei RPK eine radiale Punktkette. Die Geraden, die durch die Endpunkte e_1 und m, sowie durch e_2 und m definiert sind, definieren auf der konkaven Seite von RPK eine Kernzone (siehe Abbildung 3.10 für RPK mit Winkel kleiner 180° und Abbildung 3.11 für RPK*

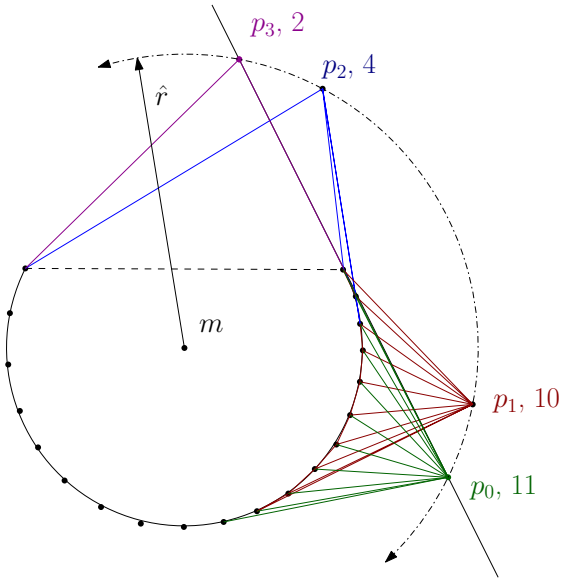


Abbildung 3.8: **Größe des Fächers aufgrund der Lage des Punktes im seitlichen toten Winkel.** In diesem Bild sind die Auswirkungen der Lage von p in einem seitlichen toten Winkel auf die Größe des konkaven Fächers zu sehen. Der Punkt p liegt auf einem Kreis, der den selben Mittelpunkt hat wie RPK, nur einen größeren Radius $\hat{r}$. Wenn p näher zur konvexen Zone kommt, so wird der Fächer kleiner. Je weiter der Punkt Richtung konkaver Zone kommt, desto größer wird der Fächer.

mit Winkel größer 180°). Wenn ein Punkt p in der Kernzone platziert wird, so enthält die Triangulierung einen konkaven Fächer.

Beweis. Wenn RPK Grad kleiner 180° hat, so sind die Zonen wie in Abbildung 3.1 definiert. In diesem Fall wird die Kernzone durch die konvexe Zone, einem großen Teil der echt konkaven Zone und Teilen der beiden seitlichen toten Winkel überdeckt. Hat RPK dagegen Grad größer oder gleich 180°, so sind die Zonen wie in Abbildung 3.2 definiert und die Kernzone kann nur von der konvexen Zone und Teilen der seitlichen toten Winkel überdeckt werden.

Es gibt drei Fälle für einen Punkt p in der Kernzone:

- p liegt in der konvexen Zone
- p liegt in der echt konvexen Zone
- p liegt in einem der beiden seitlichen toten Winkel

Fall 1: p liegt in der konvexen Zone. Laut Satz 3.3 wird ein konkaver Fächer erzeugt mit $m < n$ Punkten, wobei n die Anzahl der Punkte von RPK ist.

Fall 2: p liegt in der echt konkaven Zone. Laut Satz 3.3 wird ein konkaver Fächer mit allen Punkten der Punktkette erzeugt.

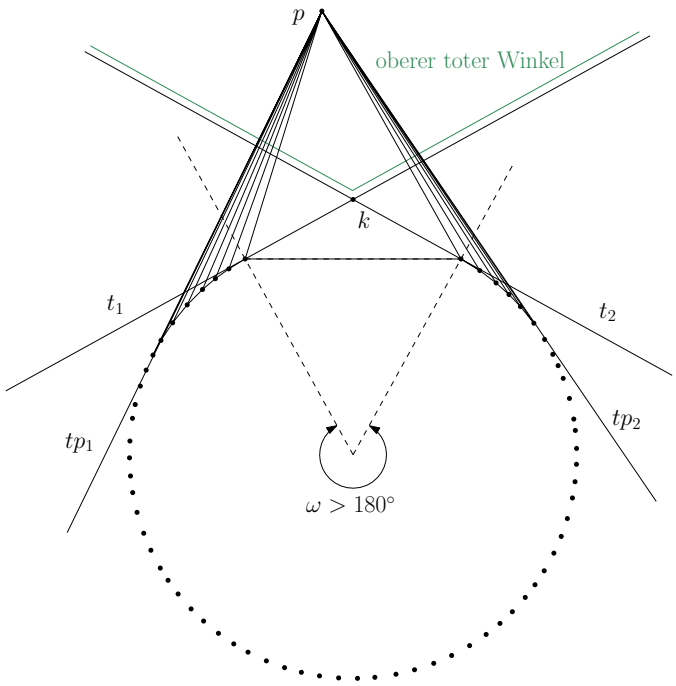


Abbildung 3.9: **Einfügen eines Punkts im oberen toten Winkel.** Wenn ein Punkt im oberen toten Winkel eingefügt wird, so entsteht zwar ein konkaver Fächer, dieser hat in der Mitte jedoch ein Loch. Dieser Effekt entsteht nur bei radialen Punktketten mit sehr großen Winkeln. Dies ist ein Sonderfall, denn der Punkt liegt eigentlich auf der konvexen Seite des Kreissegments.

Fall 3: p liegt in einem der seitlichen toten Winkeln, oBdA. dem linken toten Winkel. Wie in Abbildung 3.10 ersichtlich ist, stimmt nur ein kleiner Teil des toten Winkels mit der Kernzone überein. Dadurch wird laut Bemerkung 3.5 ein konkaver Fächer erzeugt. Dieser kleine Teil grenzt an die konkave Zone an. Dadurch wird laut Bemerkung 3.5 ein konkaver Fächer erzeugt. Daher erzeugt ein Punkt innerhalb dieses Teils einen relativ großen Fächer und der Punkt hat einen großen Grad.

□

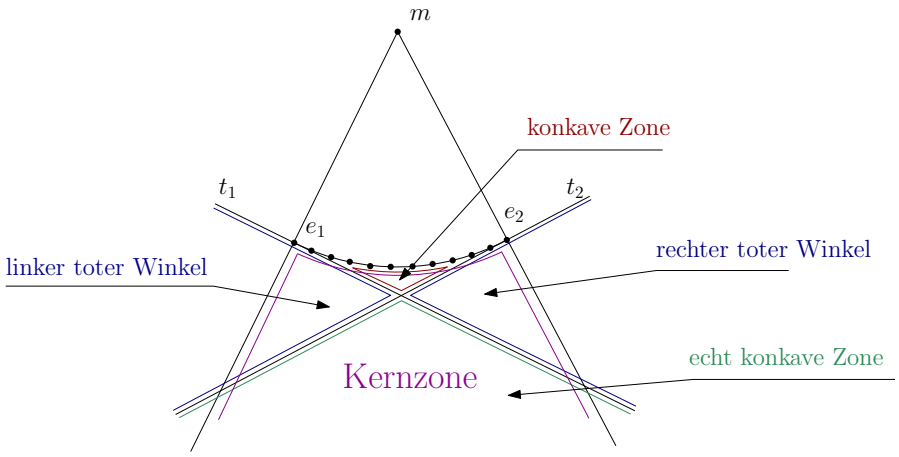


Abbildung 3.10: **Kernzone einer Punktkette mit Winkel kleiner als 180°.** In diesem Bild ist die Kernzone an einer radialen Punktkette mit Winkel kleiner als 180° ersichtlich. Es sind auch die überdeckten Zonen eingezeichnet.

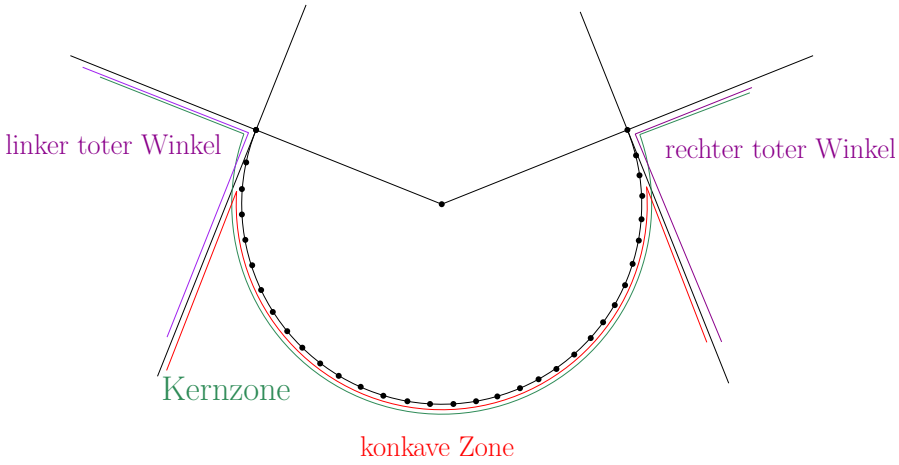


Abbildung 3.11: **Kernzone einer Punktkette mit Winkel größer als 180°.** Hier sind Kernzone an einer radialen Punktkette mit Winkel größer als 180° zu sehen. Es sind hier auch die überdeckten Zonen eingezeichnet.

3.2.2 Eigenschaften verwandter Systeme

Der Fall des echten konkaven Fächers ist erweiterbar und es werden damit viele verschiedene Punktmengen mit Triangulierungen hohem Grades erzeugt. Hierzu gibt es mehrere Möglichkeiten mit verschiedenen Auswirkungen:

- Vergrößerung des Winkels des zugrundeliegenden Kreissegments
- Erzeugung mehrerer Kreissegmente mit Punktketten

- Einfügen mehrerer isolierte Punkte in der konkaven Zone bzw. in den toten Winkeln

Vergrößerung des Winkels

Bei diesem Fall werden die Effekte der Vergrößerung des Kreissegmentwinkels untersucht. Es wird eine fixe Anzahl der Punkte n vorausgesetzt. Je größer der Winkel der radialen Punktkette ist desto größer wird der Distanzabstand zwischen den Punkten und die Kernzone. Bei gleicher Lage des externen Punktes (wenn er in beiden Fällen in der gleichen Zone landet), enthält der Fächer weniger Punkte. Daher wird auch der Grad des externen Punktes kleiner.

Auch wenn bei größeren Winkeln Kanten entlang des Kreissegments flippbar sind, würde ein solcher Flip den Grad des betroffenen Punktes noch zusätzlich erhöhen.

Sollte man einen größeren Winkel verwenden, so ist der Distanzabstand zwischen den Punkten auf dem Kreissegment größer (siehe Abbildung 3.12).

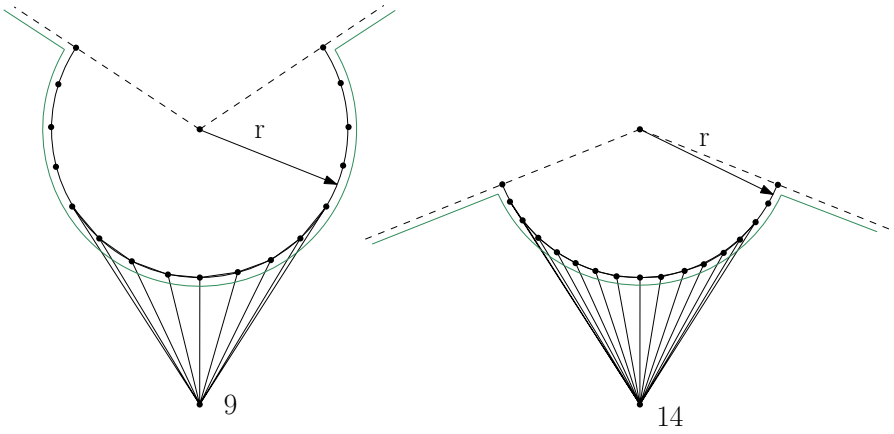


Abbildung 3.12: **Auswirkung einer Vergrößerung des Winkels auf den Grad.** Im linken und im rechten Bild wird jeweils eine RPK mit externem Punkt gezeigt. Sie haben den gleichen Radius und besitzen beide 18 Punkte. Der Unterschied in den Bildern liegt lediglich am Winkel von RPK. Links sieht man, dass der entstandene konkave Fächer weniger Kanten beinhaltet als im rechten Bild mit kleinerem Winkel. Dafür sieht man, dass die Kernzone größer geworden ist und man mehr Möglichkeiten hat, einen Punkt zu platzieren, der einen konkaven Fächer erzeugt.

Zusammenfassend kann folgendes gesagt werden: Wenn ω vergrößert wird, so hat der externe Punkt p zwar geringere Grade, aber man hat bei der Konstruktion einen insgesamt größeren Bereich, in dem p liegen darf, damit er einen Fächer erzeugt. Ist ω so groß, dass man einen ganzen Kreis hat, so hat man zwar nur die konkave Zone (es gibt keine Tangenten) aber die Punktkette hat trotzdem immer noch n Punkte. Damit kann ein Fächer nur kleiner werden. Wenn n sehr groß ist, so kann dies sinnvoll sein, aber man muss bedenken, dass man bei einem Kreis zumindest die Hälfte der Punkte der Kreiskette nicht benötigt um einen Fächer zu erzeugen, da diese Punkte auf der anderen Seite des Kreises sind und niemals erreicht werden könne.

Mehrere Kreissegmente

Hier werden statt einem isolierten Punkt mehrere Kreissegmente erzeugt, auf denen die Punkte zu liegen kommen. Wenn nun zum Beispiel zwei Kreissegmente gegenüberliegen, so kann der Grad auf die Punkte aufgeteilt und durch eine andere Triangulierung, deren Grad kleiner ist, gesenkt werden (siehe Abbildung 3.13). Selbst wenn zwei Kreissegmente normal aufeinander stehen (anhand der Gerade, die durch die Endpunkte geht, gemessen), kann der Grad gesenkt werden. Zu relativ guten Punktmengen führt nur die Wahl einer sehr großen Punktkette mit wenigen kleinen Punktketten, die höchstens bis zu drei Punkten enthalten. Hier hat man zum Beispiel eine Aufteilung eines großen Grades in drei Punkte mit dem Drittel dieses Grades plus der Triangulierung der kleinen Punktkette. Die Abbildung 3.14 zeigt, dass bei der gleichen Ausgangspunktkette ein insgesamt größerer Fächer entsteht, wenn in diesem Falle drei Punkte auf der zweiten Punktkette existieren. Es ist ebenso zu sehen, dass im Bild links die Grade im Vergleich zu nur einem Punkt nicht signifikant kleiner sind, da auch die Triangulierung der kleineren Punktkette berücksichtigt werden muss. Insgesamt erhöht es jedoch die Wahrscheinlichkeit Fächer zu erzeugen, weil der betroffene Bereich (die Anzahl der Punkte die zwischen den beiden rot markierten Endpunkte der Fächer sind) größer ist als bei nur einem Punkt. Wenn eine sehr große Punktmenge erzeugt wurde, sind auch die aufgeteilten Grade noch viel zu hoch.

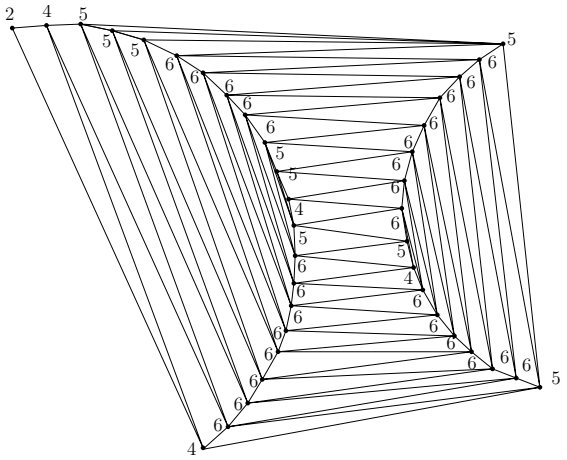


Abbildung 3.13: **Eine Triangulierung mit niedrigen Graden bei gegenüberliegenden Punktketten.** Im Bild kann man sehen, dass mehrere große Kreissegmente nicht unbedingt Triangulierungen mit hohen Graden haben müssen. Hier werden zwei radiale Punktketten gezeigt, die eigentlich optimal gegenüberliegen; das heißt jeweils komplett in der Kernzone der anderen Punktkette. In dieser Situation werden nur Grade in der Höhe von sechs erreicht, obwohl man 39 Punkte erzeugt hat. Im Vergleich dazu man kann eine Anordnung finden mit einem konkaven Fächer, der alle Punkte beinhaltet, d.h., p hat Grad 38.

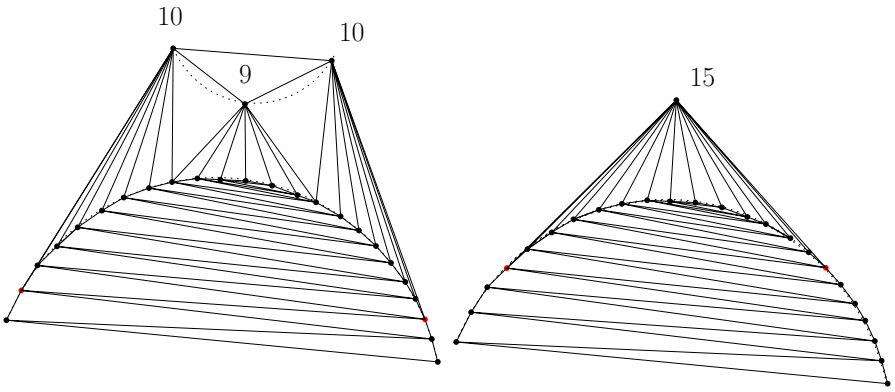


Abbildung 3.14: **Triangulierung einer radialen Punktkette mit mehreren unabhängigen externen Punkten.** Hier ist zu sehen, dass unter Umständen eine zweite radiale Punktkette mit sehr wenigen Punkten, die in der Kernzone einer größeren radialen Punktkette platziert wurde, Sinn machen kann. Der Grad der drei Punkte links ist zwar geringer als der Grad des einzelnen Punktes rechts, aber er ist nach wie vor hoch. Die Punkte teilen den Grad des einzelnen Punktes rechts untereinander auf, aber auch die Breite des konkaven Fächers ist größer, das heißt, mehr Punkte von *RPK* sind betroffen (der Bereich, der durch die roten Punkte begrenzt ist). Der Fächer links besteht nicht aus unvermeidbaren Kanten, aber die anderen Möglichkeiten weisen einen noch höheren Grad auf.

Mehrere isolierte Punkte und eine Punktkette

Hier werden mehrere isolierte Punkte in der konkaven Zone bzw. in den toten Winkeln eingefügt. Wenn mehrere Punkte in der gleichen Zone zu liegen kommen, teilen diese den Maximalgrad eines Punktes in der Zone zwischeneinander auf (gleiche Situation wie im vorigen Fall). Man sollte also sicherstellen, dass die Punkte nicht in der gleichen Zone liegen, damit hohe Grade erzeugt werden. Diese Art Punktmengen zu konstruieren setzt allerdings voraus, dass man relativ große Winkel der radialen Punktketten hat, denn ansonsten beeinflussen sich die Punkte der verschiedenen Zonen gegenseitig. Im optimalen Fall hat man hier eine radiale Punktkette mit großem Winkel (zum Beispiel $180° \leq \omega \leq 270°$) und je einen Punkt in jedem toten Winkel in der Nähe der konkaven Zone. Das stellt sicher, dass man große Fächer hat, und das beide Fächer sich sehr wenig beeinflussen (siehe Abbildung 3.15).

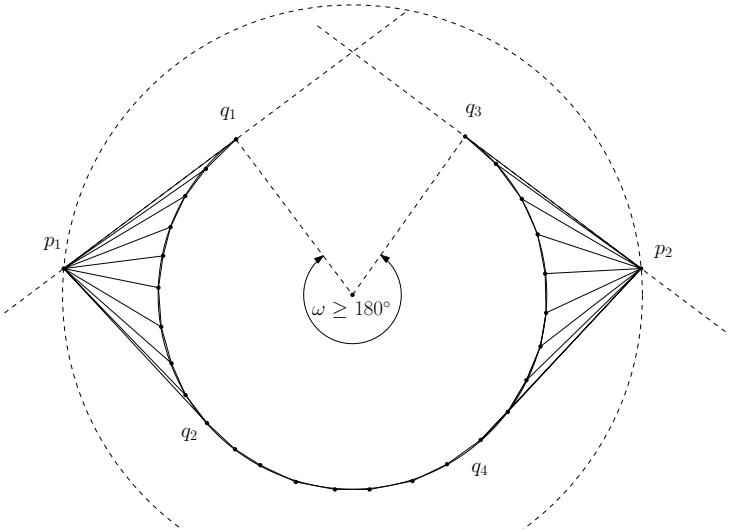


Abbildung 3.15: **Eine große radiale Punktkette mit zwei externen Punkten in der Kernzone.** Hier ist eine große radiale Punktkette zu sehen, die zwei externe Punkten besitzt, die beide in der Kernzone liegen, aber deren erzeugte konkaver Fächer einander nicht beeinflussen. Die Fächer haben keinen einzigen gemeinsamen Punkt und erzeugen deshalb beide den größtmöglichen Fächer.

4 Triangulierungen und Grade

4.1 Einführung

In diesem Kapitel sollen bisher bekannte Triangulierungen, die relativ effizient berechnet werden können, genauer bezüglich ihres Grades untersucht werden.

4.2 Standardtriangulierung

Diese Triangulierung wird von der Bibliothek CGAL (zu finden in [7] im Kapitel über Triangulierungen in 2D)bereitgestellt. Sie wird durch iteratives Einfügen der Punkte erstellt ohne Rücksicht auf Merkmale zu nehmen. Sie ist keine besondere Triangulierung aber sie kann sehr effektiv und schnell erzeugt werden ($O(n \log n)$).

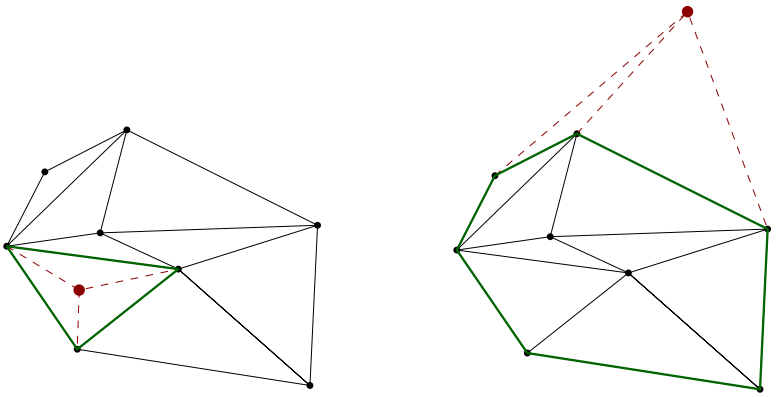
4.2.1 Erzeugung einer Standardtriangulierung

Im ersten Schritt werden drei Punkte aus P eingefügt und ein Dreieck wird damit gebildet. Danach werden die Punkte iterativ eingefügt, indem die Lage von p_i lokalisiert wird. Sollte p_i innerhalb der konvexen Hülle liegen, so muss er entweder auf einem bereits eingefügten Punkt, einer bereits erstellten Kante oder im Inneren eines Dreiecks liegen. Da in dieser Arbeit Kollinearitäten nicht betrachtet werden, kann p_i nur in einem bereits bestehenden Dreieck D_j liegen. Um p_i einzufügen, wird wie in Abbildung 1.1 links vorgegangen, indem die roten, strichlierten Kanten hinzugefügt werden. Wenn p_i außerhalb der konvexen Hülle liegt, so wird er zur konvexen Hülle hinzugefügt, indem man innerhalb der beiden Tangenten auf KH alle sichtbaren Punkte auf KH durch Kanten mit p_i verbindet (wie in Abbildung 1.1 rechts).

4.2.2 Eigenschaften einer Standardtriangulierung

Wenn die Punktmenge P aus der die Triangulierung erzeugt werden soll, randomisiert wurde, so kann keine Aussage über den Grad gemacht werden, da es nur auf die Einfügereihenfolge der Punkte $p \in P$ ankommt. Die Triangulierung selbst sieht dann meist sehr "unordentlich" aus, da sie sehr viele spitze Winkel enthält. Auffällig ist nur, das man mit dieser Methode sehr viele Pseudodreiecke erzeugt. Da diese aber auf den Grad keine Auswirkungen hat, wird hier nicht weiter darauf eingegangen.

Sollte die Punktmenge P jedoch generiert worden sein, so besitzen die Punkte eine gewisse Ordnung. In unserem Fall wäre folgende Reihenfolge gegeben: zuerst alle Punkte der Kette vom linken Randpunkt des Segments zum rechten Randpunkt des Segments und anschließend die externen Punkte. Bei Vorliegen dieses Falles erfolgt das Einfügen



Einfügen innerhalb der KH Einfügen außerhalb der KH

Abbildung 4.1: **Einfügen innerhalb und außerhalb der konvexen Hülle.** Dieses Bild zeigt wie Punkte in eine bestehende Triangulierung eingefügt werden müssen. Links liegt der einzufügende Punkt innerhalb der konvexen Hülle und liegt somit in einem bestehenden Dreieck. Der Punkt wird eingefügt, indem das Dreieck dreigeteilt wird. Rechts liegt der Punkt außerhalb der konvexen Hülle. Daher muss der Punkt an die konvexe Hülle angefügt werden, indem die Kanten entlang der Tangenten eingefügt werden und die dazwischenliegenden Kanten zwischen den Tangenten, der konvexen Hülle und dem Punkt generiert werden.

der Punkte immer nach dem gleichen Schema. Triangulierungen, die aus mehreren verschiedenen, aber doch gleich erzeugten Punktmengen erstellt wurden, haben eine große Ähnlichkeit. Auf den Einfluss, den dieses Schema auf den Grad hat, wird in einem späteren Kapitel eingegangen.

4.3 Randomisierte Standardtriangulierung

Diese Triangulierungsart ist ein Sonderfall der Standardtriangulierung. Das fixe Einfügeschema wird durch eine zufällige Einfügereihenfolge ersetzt.

4.3.1 Erzeugung einer randomisierten Standardtriangulierung

Hier wird im Gegensatz zur normalen Standardtriangulierung die Punktmenge vor dem Erstellen der Triangulierung randomisiert. Dadurch wird die starre Reihenfolge der Punkte, die durch die Erzeugung der Punktmenge erstellt wurde, gebrochen. Die restliche Algorithmus der Standardtriangulierung bleibt unverändert.

4.3.2 Eigenschaften einer randomisierten Standardtriangulierung

Diese Triangulierungsart hat den Vorteil wirklich chaotisch zu sein (siehe Abbildung 4.2). Es kann passieren, dass zufällig eine bekannte Triangulierungsart erstellt wurde, die eine bestimmte Eigenschaft erfüllt (z.B. Delaunay). Diesen Fall wird man jedoch aufgrund der geringen Wahrscheinlichkeit nur dann haben, wenn man eine sehr begrenzte Anzahl

von Triangulierungen in $\mathcal{T}(P)$ hat. Dies ist dann der Fall, wenn entweder sehr wenige Punkte in der Punktmenge P vorhanden sind, oder wenn die Triangulierung sehr viele unvermeidbare Kanten besitzt.

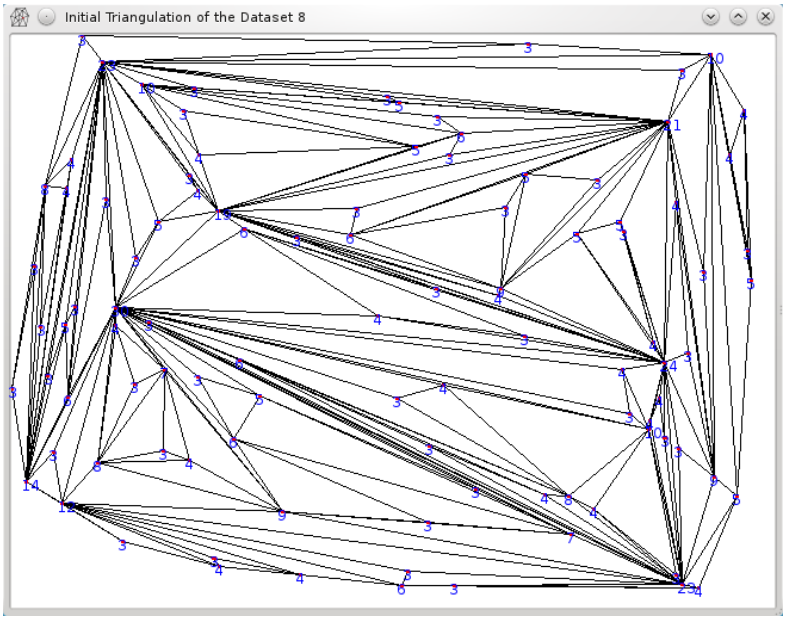


Abbildung 4.2: **Beispiel für eine randomisierte Standardtriangulierung.** Auf diesem Screenshot ist zu sehen, wie chaotisch eine solche Triangulierung aussehen kann.

4.4 Delaunay-Triangulierung

Diese Triangulierung muss mit jedem Dreieck die Umkreiseigenschaft erfüllen. Es darf also im Umkreis eines Dreiecks kein weiterer Punkt liegen. Wenn ein Punkt innerhalb des Umkreises liegen würde, so würde dieser Punkt mit den Punkten des Dreiecks ein konvexes Viereck KV_4 bilden, und man müsste flippen um die Eigenschaft nicht zu brechen. Um eine Delaunay-Triangulierung zu erstellen wird von CGAL (siehe Manual [12] im Kapitel über Delaunay-Triangulierungen in 2D) der unter [2] in Kapitel 9 beschriebene Algorithmus ausgeführt. In diesem Kapitel sind auch die Eigenschaften und Anwendungen zu finden.

4.4.1 Erzeugung einer Delaunay-Triangulierung

Für Delaunay-Triangulierungen gibt es mehrere effiziente Algorithmen. Einer davon verwendet eine modifizierte Version des iterativen Algorithmus der eine Standardtriangulierung erzeugt, mit dem Unterschied, das nach jedem Einfügevorgang so lange geflippt wird, bis die Umkreiseigenschaft erfüllt ist. Wenn die Umkreiseigenschaft nicht erfüllt ist, muss immer nur lokal mit den direkten Nachbardreiecken des betroffenen Dreiecks

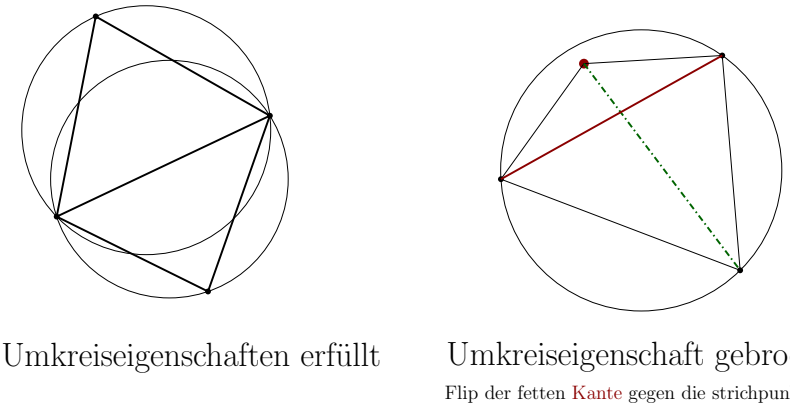


Abbildung 4.3: **Umkreiseigenschaft der Delaunay-Triangulierung.** Im linken Bild ist sie für jedes Dreieck erfüllt, denn im Umkreis der Dreiecke befindet sich kein weiterer Knoten. Rechts dagegen sieht man, dass im Umkreis des unteren Dreiecks ein weiterer Knoten liegt. Hier muss geflippt werden, damit die Umkreiseigenschaft nicht mehr verletzt ist.

geflippt werden. Danach wird so lange über die Kanten der Triangulierung iteriert, bis die Umkreiseigenschaft in der gesamten Triangulierung eingehalten wird.

4.4.2 Eigenschaften einer Delaunay-Triangulierung

Die Delaunay-Triangulierung erfüllt viele Merkmale (zB Voronoi Diagramm, Maximierung der Innenwinkel...). Geringe Grade werden allerdings nicht optimal erfüllt, denn für die Delaunay-Triangulierungen gibt es Beispiele, die mit fast jeder anderen Triangulierungsart besser gelöst werden würden. Man nehme einen konvexen Fächer, dessen Segment einen minimal größeren Radius hat als die Umkreise der Dreiecke, die zwischen externem Punkt und den Punkten der Kette gebildet werden. Sei KS ein Kreissegment mit äquidistanten Punkten auf dem Segment und p_e der externe Punkt auf der Seite, auf der der Mittelpunkt des Segments liegt. Man wähle den Radius von KS so, dass er um ein ϵ größer ist, als der größte Umkreisradius der Dreiecke, die mit einer Verbindung zwischen p_e und jeweils zwei Nachbarpunkten auf KS erzeugt wurden. Dann wird die Umkreiseigenschaft nie gebrochen und es wird die gradmässig schlechteste Triangulierung durch die Delaunay-Triangulierung erzeugt.

Abbildung 4.4 zeigt, das diese Triangulierung die Umkreiseigenschaft erfüllt, da kein Punkt der Punktkette innerhalb eines der Kreise liegt. Der Grad des unteren Punktes ist darum sehr hoch.

Dieses Beispiel zeigt, warum die Delaunay-Triangulierung nicht optimal bezüglich des Grades ist, denn diese Situation ist in einer Punktmenge mitunter sehr oft enthalten. Sogar wenn man keine Punktkette hat, kann sie entstehen.

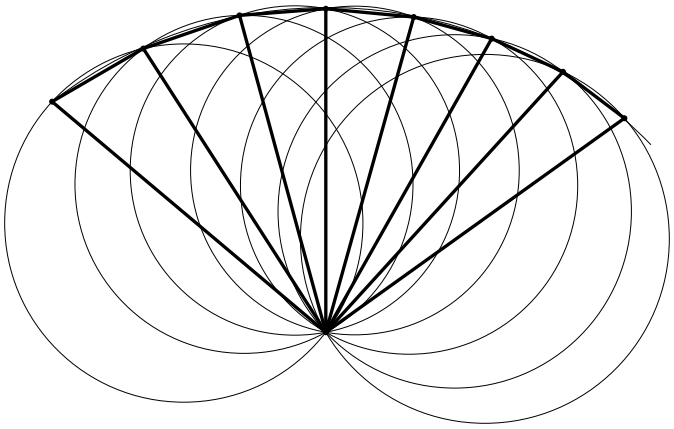


Abbildung 4.4: **Ein Beispiel für das Versagen der Delaunay-Triangulierung.** Dieser Fall zeigt genau den Fall auf, bei dem Delaunay unnötigerweise einen sehr hohen Grad erzeugt. Die Umkreiseigenschaft von Delaunay wird hier bei keinem der Dreiecke verletzt, wobei jedoch bei einer Kante, die zum Beispiel vom linken Endpunkt zum rechten geht, die Umkreiseigenschaft des unteren Dreiecks verletzt werden würde. Dies würde jedoch den Grad des unteren Punktes reduzieren.

4.5 Greedy-Triangulierung

Die Greedy-Triangulierung ist eine Annäherung an die Minimum Weight Triangulierung und wird deshalb auch untersucht. Sie mag zwar die MWT oft nur sehr schlecht annähern, aber vielleicht ist sie gerade deshalb bezüglich des Grades der Triangulierung besser.

4.5.1 Erzeugung einer Greedy-Triangulierung

Wenn nur Punktmengen vorhanden wären, die randomisiert erzeugt wurden, könnte auf einen der vielen Algorithmen zu diesem Thema zurückgegriffen werden. Diese können sogar eine Laufzeit von $O(n \log n)$ erreichen. In unserem Fall können wir darauf nicht zurückgreifen da wir Punktmengen generieren, wodurch die Punkte nicht mehr normalverteilt in der Ebene liegen. Daher wird auf den Brute-Force Algorithmus zurückgegriffen: Es wird eine Menge $\hat{E}$ erstellt, die alle möglichen Kanten enthält, die aus der Punktmenge generiert werden können. Dann wird iterativ immer die kürzestmögliche Kante aus $\hat{E}$ eingefügt und alle Kanten aus der Menge $\hat{E}$, die diese Kante schneiden würden, werden gelöscht.

4.5.2 Eigenschaften einer Greedy-Triangulierung

Auch hier gibt es Beispiele, wo eine Greedy-Triangulierung einen nicht notwendigen hohen Grad besitzt (siehe Abbildung).

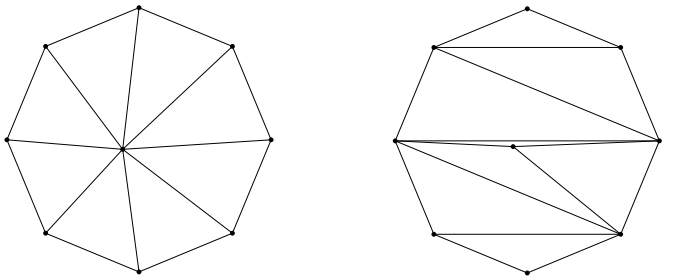


Abbildung 4.5: **Ein Beispiel für eine Punktmenge, bei der die Greedy-Triangulierung versagt.** Ein Beispiel, bei dem die Greedy-Triangulierung einer identen Punktmenge einen höheren Grad aufweist als eine beliebige andere Triangulierung. Es ist sofort ersichtlich, dass der Punkt in der Mitte im linken Bild einen höheren Grad hat als rechts.

5 Optimierungsmethoden

In diesem Kapitel werden Optimierungsmethoden erläutert, die den Grad reduzieren sollen ohne die Punktmenge zu verändern. Es werden zwei Methoden, und zwar die Methode Optimierung durch Entfernen und Wiedereinfügen und die Methode Optimierung durch Flips vorgestellt und untersucht.

5.1 Optimierung durch Entfernen und Wiedereinfügen

Die Idee dieser Methode beruht auf einer Beobachtung aus einem sehr frühen Stadium der Software. Am Anfang gab es eine Zeichenebene in der Punkte einzugeben waren und eine Triangulierung wurde mit dem Einfügealgorithmus erzeugt. Darauf wurden dann die Reduktionsmethoden, wie zum Beispiel die Entfernung eines Punktes mit dem höchsten Grad, angewandt. Als recht früh der Bedarf einer Rückgängig-Funktion entstand, trat der Effekt erstmals auf. Es zeigte sich, dass ein Punkt der zuletzt eingefügt wurde, einen geringeren Grad aufweist, als ein Punkt, der am Anfang eingefügt worden ist.

Dies ist nachvollziehbar, da die Arbeitsweise des Einfügealgorithmus dieses Verhalten erzwingt. Daraus entstand die Idee einen Optimierungsalgorithmus zu implementieren, der diesen Effekt benutzt.

In diesem Kapitel wird jedoch das Phänomen generell untersucht.

Satz 5.1. (Wiedereinfügen von Punkten bei einer randomisierten Punktmenge) *Sei P eine randomisierte Punktmenge mit n Punkten und $T(V, E)$ eine zugehörige Triangulierung die mit dem Einfügealgorithmus erzeugt wurde. Sei ein Punkt $p \in P$ dessen zugehöriger Knoten $v(p)$ den Grad $k = \deg(v(p))$ hat. Wenn $v(p)$ aus der Triangulierung entfernt wird, die restliche Punktmenge zu einer Triangulierung vervollständigt wird, und p an der gleichen Stelle wieder eingefügt wird, so hat $v(p)$ nun einen Grad kleiner gleich k.*

Beweis. Wie in Kapitel 3 erläutert wurde, werden im Normalfall generierte Punktmengen gebraucht, damit unvermeidbare Kanten erzeugt werden können. Bei randomisierten Punktmengen tritt dagegen eher der Fall auf, dass aufgrund einer schlecht gewählten Triangulierung hohe Grade vorkommen.

Wenn $v(p)$ aus der Triangulierung entfernt wird, so müssen alle Kanten, die von $v(p)$ ausgehen, gelöscht werden und es treten zwei Fälle auf:

- *Fall 1:* $v(p)$ war ein Teil der konvexen Hülle
- *Fall 2:* $v(p)$ war im Inneren der Triangulierung

Ist *Fall 1* eingetreten, so ist unter Umständen keine gültige Triangulierung mehr vorhanden. Abbildung 5.1 rechts zeigt, wie ein Entfernen eines Knotens mit den dazugehörigen Kanten immer noch gültig sein kann. In Abbildung 5.1 links ist jedoch der andere Fall, der eine ungültige Triangulierung erzeugt, ersichtlich. Die resultierende Triangulierung ist nicht mehr gültig, da sie nun nicht mehr komplett ist, denn die Außenhülle der Triangulierung entspricht nun nicht mehr der konvexen Hülle der Punktmenge $P \setminus \{p\}$. In diesem Fall muss der Bereich um den entfernten Knoten $v(p)$ neu trianguliert werden damit wieder eine gültige Triangulierung vorhanden ist. Dies wird in Abbildung 5.1 links mit den blauen strichlierten Kanten gezeigt.

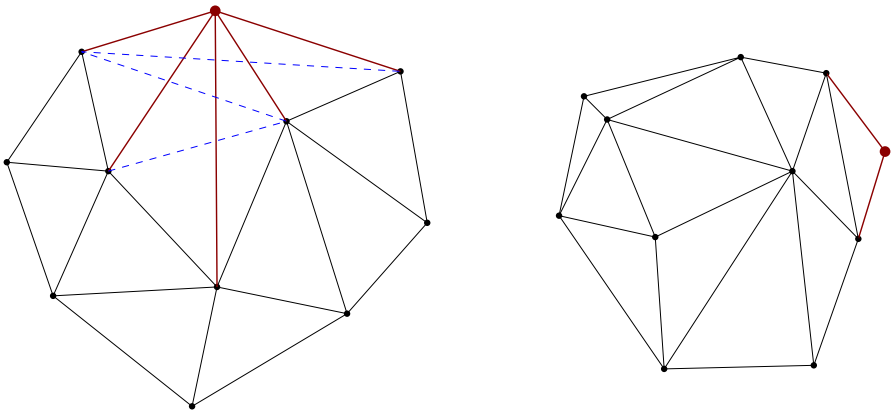


Abbildung 5.1: **Entfernen eines Punktes auf der konvexen Hülle.** Die roten Kanten werden mit dem Punkt entfernt. Links: die entstandene Lücke muss neu trianguliert werden. Rechts: Die Triangulierung bleibt gültig.

Sei nun $\hat{KH}$ die Konvexe Hülle von $P \setminus \{p\}$ und $\bar{E}$ die Menge der Kanten die Teil dieser Hülle sind. Es wird nun die Menge der Kanten E aus der ursprünglichen Triangulierung $T(V, E)$ mit der Menge $\bar{E}$ verglichen. Sei m die Anzahl der Kanten in $\bar{E}$, die nicht in E enthalten sind. Wenn $m = 0$ ist, so ist nach dem Entfernen von p eine gültige Triangulierung vorhanden und es wird nach dem erneuten hinzufügen von p wieder genau der gleiche Grad $\deg(v(p))$ vorhanden sein. Sollte $m > 0$ sein, so tritt die in Abbildung 5.2 gezeigte Situation auf. Es ist bekannt, dass man mit endlich vielen Flips zu jeder anderen gültigen Triangulierung gelangen kann. Da die Kanten von $\bar{E}$ gültige Kanten sind, ist es möglich, zu einer Triangulierung zu kommen, für deren Kantenmenge $\bar{E} \in E$ gilt. Die m Kanten müssen zumindest geflippt werden, damit diese neue Triangulierung entsteht. Wenn nun der Knoten $v(p)$ mit dem normalen Einfügealgorithmus wieder zur Knotenmenge hinzufügt wird, so werden mit diesem Algorithmus um m Kanten weniger generiert als in der ursprünglichen Triangulierung und $v(p)$ hat demnach $\deg(v(p)) - m$ als neuen Grad.

Liegt jedoch *Fall 2* vor, so wird ein Knoten aus dem Inneren der Triangulierung entfernt und es wird damit ein Loch im Inneren der Triangulierung erzeugt. Damit kann die neue Triangulierung nicht mehr gültig sein. Um die Triangulierung wieder gültig zu machen, muss der Bereich um den entfernten Knoten neu trianguliert werden. Dieser

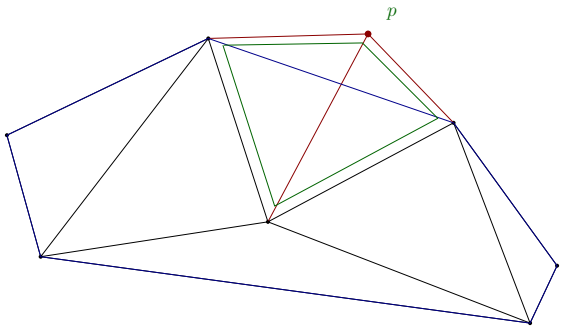


Abbildung 5.2: **Veranschaulichung des Entfernens eines Punktes auf der konvexen Hülle.** Durch endlich viele Flips kann man eine Triangulierung erzeugen, die nach dem Entfernen der Kanten von p gültig bleibt.

Bereich ist definiert durch alle Punkte, die eine zugehörige Kante verloren haben (siehe Abbildung 5.3). In der Abbildung ist dieser Bereich dunkelrot untermalt. Genau jener Bereich muss gültig trianguliert werden.

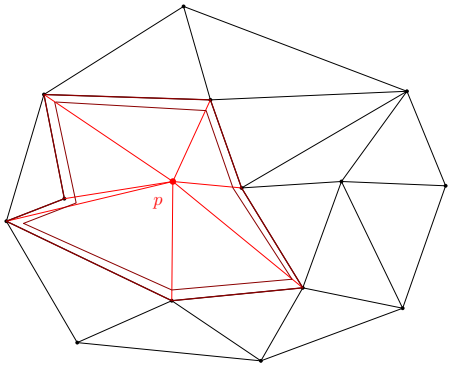


Abbildung 5.3: **Entfernen eines Punktes innerhalb der konvexen Hülle.** Die Lücke, die durch das Löschen der Kanten von p entsteht, muss neu trianguliert werden.

Wenn p nun wieder eingefügt wird, so wird dieser in eine bereits bestehende gültige Triangulierung eingefügt und kann, da dieser Punkt im Inneren liegt, nur innerhalb eines Dreiecks D liegen. Beim Einfügen werden nun nur die drei Kanten von p zu allen Punkten von D eingefügt. Damit kann $v(p)$ nur drei Kanten haben und muss somit $\deg(v(p)) = 3$ haben. Das kann nur kleiner oder gleich dem vorherigen Grad sein, denn im Inneren einer Triangulierung muss der Mindestgrad drei betragen. □

Diese Optimierungsmethode bringt nur unter Umständen eine Verbesserung des Grades, da der Einfluss auf die benachbarten Knoten nicht berücksichtigt wird. Bei nur wenigen Punkten einer allgemeinen Punktmenge kann diese Methode sehr gut und effizient funktionieren aber die Auswahl der Punkte, die entfernt werden, um sie wiedereinzufügen, kann sich als sehr schwierig herausstellen. Es kann entweder immer der Knoten

Diff.	Kl.	Bedeutung
+ - - + + - + - + + - -	1	höchsten Grad erhöht sich
+ 0 0 - + 0 - 0 + - 0 0	1	höchster Grad erhöht sich, nur ein Grad senkt sich
0 + 0 - 0 + - 0	1	höchsten Grad bleibt gleich, zweithöchster Grad erhöht sich
0 0 + -	1	höchster und zweithöchster Grad neutral, dritthöchster erhöhtk sich
0 0 0 0	2	neutraler Flip
0 0 - +	3	erste und zweite Stelle neutral, dritter Grad besser
0 - + 0 0 - 0 +	4	erste Stelle neutral, verbessert den zweithöchsten Grad
- + 0 0 - + + - - + - +	5	höchster Grad besser, zweithöchster schlechter
- 0 0 + - 0 + 0	6	höchster Grad besser, zweithöchster neutral
- - + +	7	höchster und zweithöchster Grad verbessert sich

Tabelle 5.1: Die Bedeutung der Differenzen der Gradvektoren

mit dem höchsten Grad entfernt werden, was sich als problematisch herausstellen kann, da dieser Knoten unter Umständen zu viele unvermeidbare Kanten besitzt und somit kann mit dieser Methode keine Verbesserung erzielt werden. Wenn bedacht wird, dass die Nachbarn des Knotens in der Regel einen höheren Grad aufweisen können als zuvor, dann kann mit dieser Methode unter Umständen durch das Verbessern eines Grades ein anderer hoher Grad erzeugt werden.

5.2 Optimierung durch Flips

Diese Methode beruht im Vergleich zur vorherigen Methode auf einer gezielten lokalen Verbesserung der Triangulierung. Man benutzt hierzu folgenden Satz:

Satz 5.2. (Lokale Verbesserung des Grades durch einen Flip) *Sei KV_4 ein konvexes Viereck einer Triangulierung $T(V, E)$ einer Punktmenge P und* $\mathrm{Deg}(KV_4)$ *der zugehörige Gradvektor. Dann kann laut Definition 2.10 ein anderes, gültiges konvexes Viereck $\hat{KV}_4$ mit Gradvektor* $\mathrm{Deg}(\hat{KV}_4)$ *erzeugt werden. Dieser Flip ist genau dann günstig, wenn die Differenzen der Gradvektoren laut Tabelle 5.1 in Klasse 3 bis 7 liegen.*

Beweis. Wenn KV_4 geflippt wird, so wird die vorhandene Kante entlang der ersten Diagonale im Zentrum gelöscht. Dadurch muss der Grad der beiden Punkte, zwischen denen diese Kante verlaufen ist, um 1 gesenkt werden. Fügt man nun die andere Diagonale als Kante ein, so erhöht sich der Grad der beiden anderen Knoten um 1. Damit erzeugt man nur wenige Möglichkeiten eines Differenzvektors. Der Differenzvektor muss, sollten Unterschiede entstanden sein, immer zwei gekoppelte Einträge haben. Für jeden Eintrag, der um 1 besser wurde, muss ein Eintrag vorhanden sein, der um 1 schlechter wurde. Gleich bleiben kann ein Eintrag nur dann, wenn er mit einem anderen Eintrag Platz getauscht hat. Das kann nur passieren, wenn zwei Einträge des Vektors um 1 verschieden sind, und der kleinere Eintrag erhöht und der größere verringert wird. Dadurch erhält man alle in der Tabelle aufgeführten Möglichkeiten. Eine lokale Verbesserung des Grades kann nur dann passieren, wenn der erste Eintrag, der nicht Null ist, kleiner ist. Das passiert bei Klasse 3,4,5,6 und 7. □

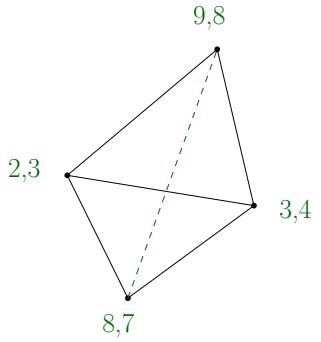


Abbildung 5.4: **Ein Flip mit Gradvektoren der Klasse 7.** Durch den Flip verbessert sich der höchste und der zweithöchste Grad jeweils um eins. Die beiden anderen, kleineren Grade erhöhen sich jeweils um eins.

Mit Hilfe dieses Satzes wird eine Klassifikation jeder flipbaren, inneren Kante nach diesem Schema erstellt. Es wird so lange der beste Flip ausgeführt und die Klassifikationen aktualisiert, bis kein weiterer guter Flip mehr möglich ist. Da unvermeidbare Kanten nicht flipbar sein können, kann diese Methode auch keine konkaven Fächer verhindern. Die so erzeugte Triangulierung ist zwar recht gut, aber mitunter nicht optimal, wie folgender Satz zeigen wird.

Satz 5.3. *Die Methode, die lokale Flips zur Verbesserung des Grades benutzt, erreicht nicht immer die grad-optimale Triangulierung.*

Beweis. Dieser Beweis benutzt ein einfaches Gegenbeispiel: Sei $T(V, E)$ eine Triangulierung, deren Grad reduzierbar ist und $T(V, E)$ besteht aus einem KV_4 und sonst nur nicht konvexen Vierecken. Wenn KV_4 geflippt werden würde, so würde dieser lokale Flip den Gradvektor verschlechtern. Um jedoch den Grad des gesamten Systems zu senken muss dieser Flip ausgeführt werden, denn er ermöglicht andere Flips, die den Grad der Triangulierung insgesamt senken. Da der Algorithmus terminiert, wenn nur ein schlechterer Flip ausgeführt werden kann, werden diese anderen Flips nicht ermöglicht. Damit kann nie auf die grad-optimale Triangulierung reduziert werden. □

Am Ende dieses Kapitels wird noch ein konkretes Beispiel gezeigt, das veranschaulichen soll, warum es mit dieser Methode nicht immer möglich ist, die gradoptimale Triangulierung zu erhalten. In Abbildung 5.5 ist eine Punktmenge mit einer bestehenden Triangulierung und deren möglicher Flips zu sehen. Wenn die Gradvektoren dieser möglichen Arten von Flips betrachtet werden, so ist zu sehen, dass egal welcher mögliche Flip augeführt wird, nur eine lokale Verschlechterung des Grades hervorruft. Beim Flip des Typen 1 (senkrecht und dunkelrot schraffiert) beträgt der momentane Gradvektor (8;8;5;4), wogegen der Gradvektor der geflippten Kante (9;7;6;3) hat. Ein Flip des Typen 2 (grün und nach rechts unten schraffiert) hat die Gradvektoren (8;8;7;5) und geflippt (9;7;6;6). Typ 3 (magenta und waagrecht schraffiert) hat (8;8;8;7) und (9;8;7;7); Typ 4 (orange und nach rechts oben schraffiert) (8;8;8;8) und (9;9;7;7). Die Differenzvektoren betragen damit (1;-1;1;-1), (1;-1;-1;1), (1;0;-1;0) und (1;1;-1;-1). Daher kann mit jedem dieser möglichen Flips nur eine lokale Verschlechterung erzielt werden und der hier implementierte Optimierungsalgorithmus würde keinen dieser Flips ausführen. In Abbildung 5.6 ist die optimale Triangulierung dieser Punktmenge zu sehen, deren maximaler Grad 7 beträgt.

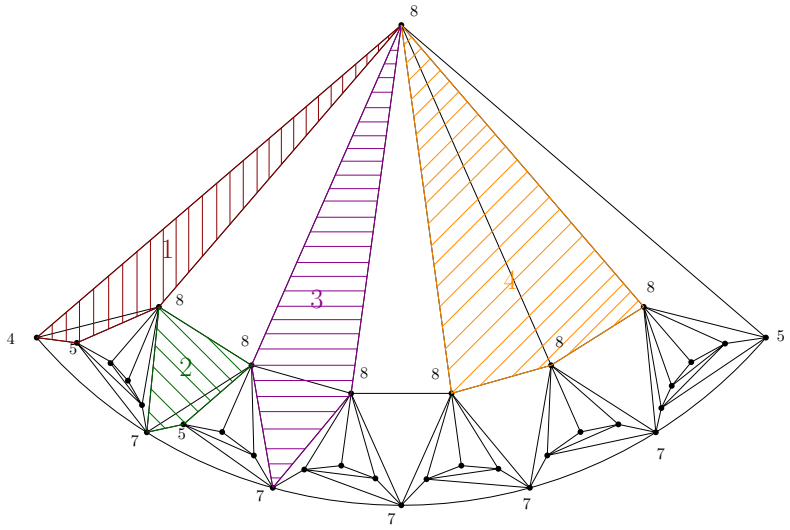


Abbildung 5.5: **Kein lokaler gradverbessernder Flip ist möglich.** Dieses Bild zeigt eine Punktmenge mit einer Triangulierung, bei der kein Kantenflip eine lokale Verbesserung des Grades bringt. In der nächsten Abbildung kann man jedoch sehen, dass diese Triangulierung nicht optimal ist. Die schraffierten Vierecke geben hier die möglichen arten der möglichen Flips an, wobei für jede Art ein Beispiel herangezogen wurde.

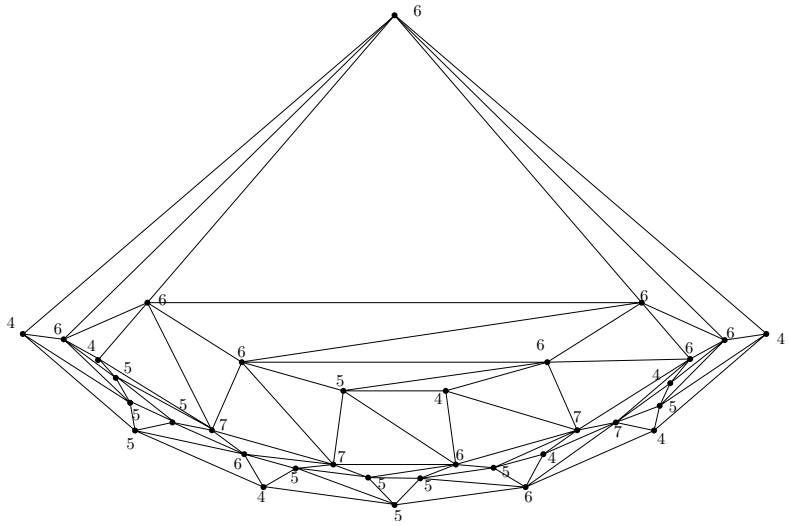


Abbildung 5.6: **Gradoptimale Triangulierung der vorherigen Punktmenge.** Hier wird eine der gradoptimalen Triangulierungen der Punktmenge, die auch in Abbildung 5.5 verwendet wurde, gezeigt. Eine andere, ebenfalls gradoptimale Triangulierung wäre eine Spiegelung der Triangulierung.

6 Reduktionsmethoden

In diesem Kapitel werden Reduktionsmethoden betrachtet. Dabei handelt es sich um Methoden, die den Grad einer Triangulierung senken sollen. Im Gegensatz zum vorigen Kapitel werden in diesem Kapitel Methoden betrachtet, die gezielt die Punktmengen verändern. Es wird versucht mit sehr wenigen Operationen auszukommen, denn wenn Punkte gelöscht oder hinzugefügt werden, gehen auch immer Informationen verloren oder werden verändert. Aus diesem Grund müssen die angewendeten Methoden so effizient wie möglich sein. Da Kollinearitäten sehr viele weitere Probleme verursachen, wie beispielsweise ungültige Triangulierungen, sollen sie vermieden werden. Das heißt, es dürfen keine Punkte eingefügt werden, die auf einer bereits bestehenden Gerade, die definiert durch zwei in P vorhandene Punkte, liegen. Auch essenzielle Informationen, wie der Rand der Punktmenge, sollen nicht verloren gehen. Deshalb werden keine Punkte auf der konvexen Hülle entfernt oder hinzugefügt.

6.1 Auf "Punktentfernung" basierende Reduktionsmethoden

Hier werden Reduktionsmethoden untersucht, die gezielt einzelne Punkte löschen. Wobei darauf geachtet wird, dass keine Punkte auf der konvexen Hülle entfernt werden. Darum wird immer nur ein Punkt entfernt. Es werden verschiedene Selektierungsmethoden vorgestellt, die Punkte zum Löschen auswählen sollen. Es wird die Anwendbarkeit der Reduktionsmethoden auf Punktmengen verglichen, wie sie in Kapitel 3 erstellt wurden. Auf jeden Fall muss beachtet werden, dass ein innerer Punkt, der gelöscht wird, ein Loch hinterlässt, welches geschlossen werden muss. Das kann durch die Triangulierung selbst gelöst werden (siehe [12] CGAL Manual im Kapitel über Triangulierungen).

- *Entfernung des gradhöchsten Punktes:* Die erste Methode um einen Punkt zu selektieren ist den gradhöchsten Punkt auszuwählen, um diesen zu löschen. Dazu wird der gradhöchste Punkt selektiert, der nicht auf der konvexen Hülle liegt. Das kann in der Praxis ein Punkt sein, dessen Grad sehr niedrig ist.

- *Entfernung des gradhöchsten Nachbarn des gradhöchsten Punktes:* Bei dieser Methode wird zuerst der gradhöchste Punkt gesucht (absolut, darf auf der konvexen Hülle liegen). Danach wird über die Nachbarn iteriert um den gradhöchsten Nachbarpunkt zu finden, der nicht auf der konvexen Hülle liegt. Dieser Nachbarpunkt wird gelöscht.

- *Entfernung eines beliebigen Nachbarpunktes des gradhöchsten Punktes:* Hier wird ebenfalls zuerst der absolut gradhöchste Punkt gesucht. Dann wird ein beliebiger (zufälliger) Nachbarpunkt selektiert, der sich nicht auf der konvexen Hülle befindet.

- *Entfernen eines beliebigen Punktes:* Die letzte Methode ist das Entfernen eines beliebigen Punktes, der nicht auf der konvexen Hülle liegt.

6.2 Auf "Punkthinzufügen" basierende Reduktionsmethoden

Es werden Reduktionsmethoden untersucht, die Punkte hinzufügen, mit der Zielsetzung den Grad zu reduzieren. Es ist darauf zu achten, dass die hinzugefügten Punkte nicht kollinear sind. D.h. wenn der hinzugefügte Punkt kollinear zu zwei weiteren Punkten ist, so wird er um ein Epsilon verschoben um sicherzustellen, dass kein Problem erzeugt wurde. Vorerst werden verschiedene Ideen für Algorithmen vorgestellt, die Punkte hinzufügen.

- *Hinzufügen des Schwerpunktes des gradhöchsten Nachbardreiecks:* Hier wird ein Punkt als Schwerpunkt eines Dreiecks errechnet und eingefügt, das erstens ein Nachbar des gradhöchsten Punktes ist und zweitens die höchste Summe über die Grade der Eckpunkte hat.

- *Hinzufügen des Schwerpunkts eines beliebigen Nachbardreiecks:* Hier wird ein Punkt als Schwerpunkt eines Dreieck errechnet, wobei der gradhöchste Punkt ein Eckpunkt dieses Dreiecks sein soll.

- *Hinzufügen des Schwerpunkts eines speziell berechneten, bisher noch nicht benutzten Nachbardreiecks:* Hier wird ein Punkt als Schwerpunkt eines Dreiecks eingefügt. Sei m die Anzahl der Nachbardreiecke von p, wobei p der Punkt höchsten Grades ist und sei k der Grad, auf den reduziert werden soll. Berechne $l = m/k$. Angefangen mit dem linkest möglichen Dreieck wird geprüft, ob dieses Dreieck bereits einmal zum Einfügen eines Punktes benutzt worden ist. Wenn dies nicht der Fall war, wird der Schwerpunkt in das Dreieck eingefügt und wird markiert. Ansonsten iteriert man um l Dreiecke weiter.

Weitere Ideen wären, Punkte an anderen Stellen als im Schwerpunkt des Dreieck einzufügen, zum Beispiel auf der Schwerlinie des Dreiecks, die durch den Punkt p geht, oder einen Punkt komplett randomisiert innerhalb des Dreiecks zu berechnen. Um Kollinearitäten zu vermeiden, muss der eingefügte Punkt mit allen anderen Punkten auf Kollinearität untersucht werden und gegebenenfalls normal zur Kollinearitätsgerade verschoben werden. Aufgrund der Beschaffenheit der Punktmenge im Bereich eines hochgradigen Punktes (muss ein konkaver Fächer sein), haben die benachbarten Dreiecke alle einen gemeinsamen Punkt p und die anderen Punkte der Dreiecke liegen entlang eines Kreissegments. Da die Schwerpunkte (bzw. Punkte, die auf dieselbe Weise in den einzelnen Dreiecken generiert wurden) genau diese Eigenschaft erben müssen (ohne Beweis, aber klar wenn man die Schwerpunkteigenschaften nachschlägt), können auch diese generierten Punkte nicht kollinear sein. Im Programm wurde nur die Generierung der Punkte im Dreieck anhand des Schwerpunktes herangezogen, weil die Effekte genau die gleichen sein müssten. Es ist wichtiger, in welchem Dreieck ein Punkt generiert wird, als wo dieser im Dreieck generiert wird (siehe Auswertung).

6.3 Auswirkungen dieser Methoden

Wenn ein Punkt aus der Punktmenge entfernt wird, so verliert man immer auch Informationen mit dem Entfernen dieses Punktes. Um die genauen Auswirkungen abschätzen zu können, müssten Informationen über die darüberliegenden Algorithmen bekannt sein. Da eine Triangulierung jedoch eine Datenstruktur für sehr viele Algorithmen sein kann, kann man nie genau bemessen, wie essenziell diese Information ist, die man mit einem Punkt löscht. Wenn zum Beispiel die Simulationstechnik herangezogen wird, so kann mit einem Punkt eine ganze Gleichung verlorengehen und in den benachbarten Punkten wird de Komplexität der Gleichungen erhöht. Werden aber Punkte hinzugefügt, so erhalten sie möglicherweise nicht relevante oder sogar falsche Informationen. Da man nicht weiß, wofür die Triangulierung im konkreten Fall benutzt wird, sollten nur wenige aber effiziente Schritte durchgeführt werden.

Liegt ein konkaver Fächer vor, so liegt der schlechteste Fall vor. Das kann allein dadurch gesehen werden, dass die normalen Optimierungsalgorithmen wie der Flip-Algorithmus versagen muss. Leider kann mit den Methoden, die Punkte entfernen, auch kein großer Erfolg erzielt werden, weil der gradhöchste Punkt bei einem eindeutigen konkaven Fächer immer auf der konvexen Hülle sein muss. Dieser kann jedoch genau deshalb nicht gelöscht werden. Sollte der Fächer nicht eindeutig sein und seinen höchsten Grad bei einem Punkt haben, der im Inneren der Triangulierung liegt, so hat das Entfernen dieses Punktes immer eine Verschlechterung des Grades zur Folge, denn die Punkte, die neben diesem Punkt liegen, müssen den Verlust dieses Punktes ausgleichen. Diese liegen dafür meist auf der konvexen Hülle (siehe Abbildung 6.1). Daher muss man sich mit entweder dem zweithöchsten Punkt begnügen, der nicht auf der konvexen Hülle ist, oder mit dem höchsten Nachbarpunkt. Weil der konkave Fächer, wie wir in Kapitel 3 gesehen haben, nur aus einer Punktkette und höchstens ganz wenigen hochgradigen, externen Punkten, die immer auf der konvexen Hülle liegen, besteht, können nur Punkte gelöscht werden, die auf der radialen Punktkette und innerhalb der Berührungspunkte der Tangenten liegen. Somit wird die Auswahl der zu löschenden Punkte extrem beschränkt und egal welcher Selektionsalgorithmus angewendet wird, man hat nur sehr bedingten Erfolg. Ein Löschen eines dieser Punkte hat nicht sehr große Auswirkungen auf den Grad des damit verbundenen extremen Punktes, denn der Grad dieses Punktes wird lediglich um den Faktor 1 vermindert. Damit müsste man theoretisch $m - k$ Punkte löschen, wobei k der Höhe des Grades im extremen Punkt (sollte es nur einer sein) entspricht und m der Anzahl der Punkte auf der Punktkette, die zwischen den Tangenten von p auf RPK. Damit würden genau l Punkte auf der Punktkette übrigbleiben, wobei l der zu erreichende Grad ist. Um dies zu Veranschaulichen: RPK hat einen Winkel $\omega \leq 180°$ und besitzt 99 Punkte. Der externe Punkt liegt auf der Mittellinie des Kreissegments und liegt innerhalb der echt konkaven Zone. Damit hat dieser Punkt p Grad 99. Um auf Grad 9 zu senken, müsste man damit 90 Punkte löschen.

Es ist also klar zu erkennen, dass mit dem Entfernen von Punkten in diesem Fall sehr wenig zu erreichen ist. Werden Punkte hinzugefügt, so kann der Grad nur dann reduziert werden, wenn ein einzelner großer konkaver Fächer durch das Einfügen eines Punktes unterteilt wird. Wie in Kapitel 3 zu sehen war, wird der Grad eines Punktes gesenkt,

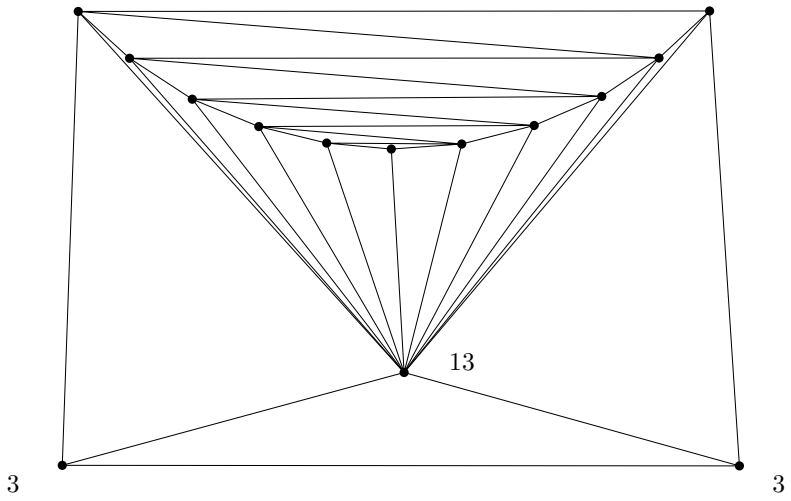


Abbildung 6.1: **Der gradhöchste Punkt ist nicht auf der konvexen Hülle** Dieser Fächer ist nicht eindeutig, denn es wäre möglich, den Fächer auf die drei Punkte aufzuteilen. Sollt ein solcher Fall vorliegen, könnte der gradhöchste Punkt entfernt werden. Das würde jedoch den Grad der beiden äußeren unteren Punkte sehr erhöhen und diese Reduktion würde keine Verbesserung bringen.

wenn er einen direkten Nachbarpunkt bekommt. Da hier die konvexe Hülle beibehalten werden soll, muss dieser Nachbarpunkt innerhalb der konvexen Hülle liegen.

Um dies zu bewerkstelligen liegen nun zwei Möglichkeiten vor. Die erste besteht darin, dass eine Art Fächerkette erzeugt wird, deren Punkte ebenso auf einem Kreissegment zu liegen kommen; dies bewirkt eine Senkung des Grades eines externen Punktes und wird in Abbildung 6.2 veranschaulicht. Bei sehr großen Punktketten werden möglicherweise mehrere Schichten benötigt. Bei der zweiten Möglichkeit wird ebenso eine Fächerkette gebildet, aber die einzelnen Punkte sind annähernd kollinear entlang einer Geraden zwischen dem Mittelpunkt des Kreissegments und dem externen Punkt p. Dies wird in Abbildung 6.3 veranschaulicht.

Beide Möglichkeiten haben ihre Vor- und Nachteile. Die Vorteile sind bei beiden Varianten ähnlich, denn beide Varianten reduzieren den Grad, indem sie den maximalen Grad unter den eingefügten Punkten aufteilen. Bei der ersten Methode besteht der Vorteil darin, dass diese leichter zu berechnen ist und für allgemeinene Fälle besser anwendbar ist, da die zu erwartenden konkaven Fächer kleiner sind. Der Nachteil jedoch ist, dass man mindestens einen Punkt mehr braucht als bei der zweiten Methode, denn man benötigt mehr Verbindungskanten zwischen den neuen Punkten. Methode zwei ist dagegen etwas komplexer zu implementieren, da man den Fächer mit der Software erkennen muss und dann die zugrundeliegende Mittellinie des Segments errechnen muss auf der die Punkte platziert werden sollen. Danach muss man noch die Punkte jeweils so verschieben, dass sie nicht kollinear sind und so lange flippen, bis man den Grad reduziert hat. Der Vorteil ist, dass man sehr wenige Verbindungskanten zwischen den neuen, eingefügten Punkten

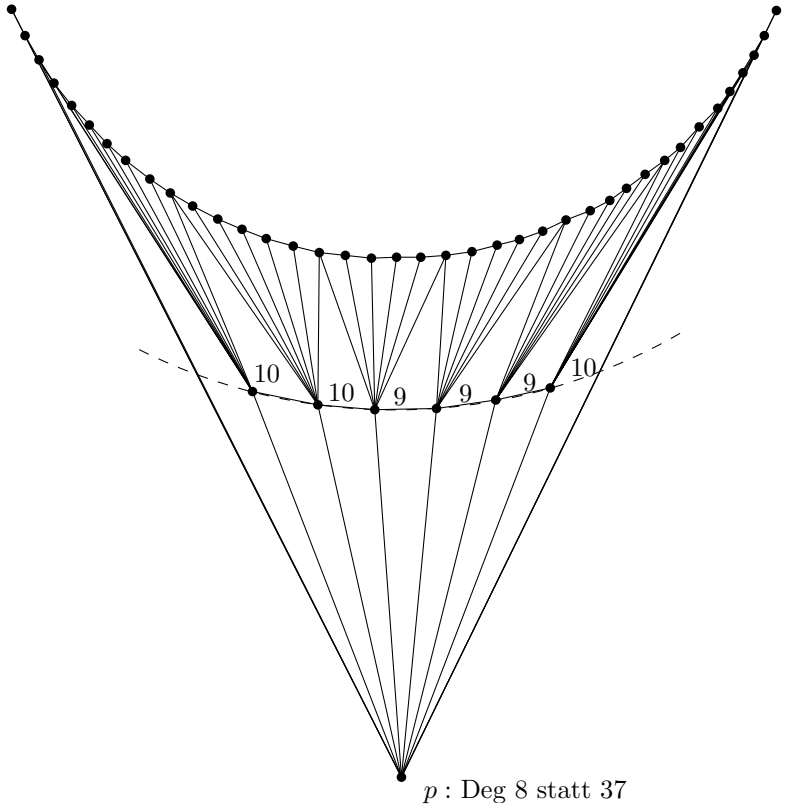


Abbildung 6.2: **Eine eingefügte Punktkette entlang eines Kreissegments.** In diesem Bild ist die erste Methode zu sehen, in der eine weitere radiale Punktkette erzeugt wird (hier strichliert eingezeichnet), auf der die eingefügten Punkte zu liegen kommen. Es ist erkennbar, dass diese Methode den Grad von Punkt p sehr effizient senken kann.

braucht (eine!) und somit mindestens einen Punkt weniger als bei der ersten Methode benötigt. Beide Methoden können bei l eingefügten Punkten den Grad des Punktes ungefähr auf k/l senken, wobei dies von der Anzahl der benötigten Verbindungskanten abhängig ist.

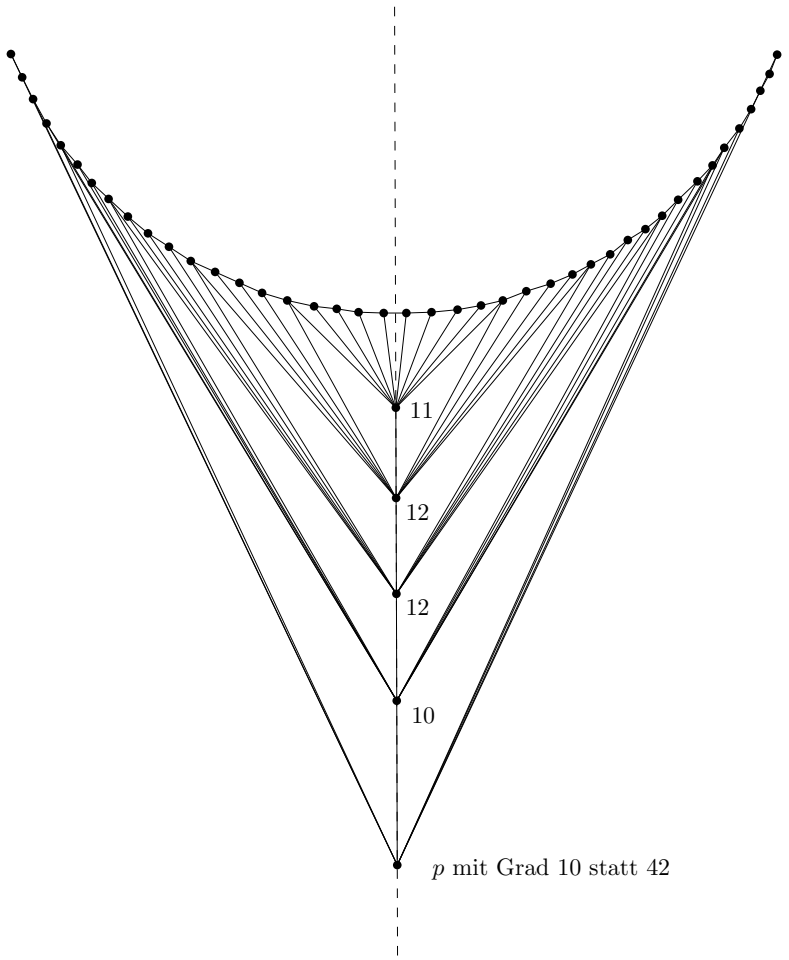


Abbildung 6.3: **Eine eingefügte Punktkette entlang der Gerade zwischen m und p.** In diesem Bild ist die zweite Methode zu sehen, in der die Punkte senkrecht entlang der strichlierten Linie eingezeichnet wurden. Hier wurde auf kollineare Punkte nicht geachtet, da dieses Bild nur zur Veranschaulichung der Situation dienen soll. In einem Algorithmus, der auf dieser Methode basiert, sollte jedoch darauf geachtet werden. Es ist ebenfalls erkennbar, dass diese Methode den Grad von Punkt p ungefähr so gut wie Methode 1 senken kann.

7 Software

In diesem Kapitel wird die implementierte Software genauer erläutert. Es wird sowohl die Installation als auch der Funktionsumfang, sowie die Implementierung wichtiger Algorithmen erläutert.

Die Software hat einige wichtige Aufgaben zu erfüllen. Sie muss die in den vorangegangenen Kapitel beschriebenen Algorithmen testen können. Dazu werden Punktmengen benötigt, die nach dem Schema in Kapitel 3 erzeugt werden können. Auch die Optimierungsmethoden aus Kapitel 5 und einige der Reduktionsmethoden aus Kapitel 6 werden implementiert und mit Hilfe der erstellten Testumgebung untersucht. Um die Bedienbarkeit der Software benutzerfreundlich zu gestalten, wurde eine umfassende grafische Oberfläche implementiert. Dadurch wurde sowohl die Auswertbarkeit der Ergebnisse erleichtert, als auch die Vergleichbarkeit der Testläufe verbessert.

Der Funktionsumfang setzt sich aus folgenden Hauptgruppen zusammen:

- *Generierung von aussagekräftigen Testdatensätzen*
- *Implementation der Algorithmen zur Gradreduktion*
- *Aussagekräftige statistische Auswertungen*
- *Möglichkeiten, die verschiedenen Phasen der Reduktion zu analysieren*
- *Eine grafische Benutzeroberfläche für die Bedienung*

7.1 Voraussetzung und Installation

Die Software läuft unter folgenden Vorausetzungen: Als Betriebssystem wird eine aktuelle Linux Distribution benötigt, zum Beispiel Debian stable oder besser unstable (Stand Oktober 2010, unstable ist besser, da neuere Versionen von CGAL und QT im Standardumfang enthalten sind.). Es werden dazu noch wenige, additiv installierbare Pakete benötigt wie QT 4.6.2 oder höher (qt-core und qt-dev), sowie CGAL 3.6.1. Das kann zum Beispiel durch die üblichen Packetmanager wie aptitude, emerge usw. erfolgen. Sollte man ein aktuelles System haben, so ist die benötigte Software als Standardprogramme verfügbar. Sollten die benötigten Bibliotheken nicht vorhanden sein, so werden sie mit dem eingebauten build System lokal installiert. Dabei sollte jedoch darauf geachtet werden, dass hierfür sehr viel Speicherplatz benötigt wird, denn QT und CGAL und das Anwendungsprogramm selbst benötigen 5.8GB Speicherplatz. Das lokale Kompilieren von QT dauert sehr lange (auf einem normalen System muss ungefähr mit 2 Stunden 30 Minuten für die Installation gerechnet werden). Darum wird empfohlen sowohl QT als auch CGAL bereits am System zu installieren.

Das Programm wird als .tar.gz Datei geliefert. Somit muss es vorher entpackt werden:

```
(1) tar zxf StatisticalOutput-2.0.0-slim.tar.gz
(2) tar zxf StatisticalOutput-2.0.0-with-deps.tar.gz
(3) tar zxf StatisticalOutput-2.0.0.tar.gz
```

- (*1*) wird verwendet, wenn Qt und CGAL bereits installiert sind
- (*2*) wird verwendet, wenn die Pakete zu installieren sind, aber kein Zugang zum Internet vorhanden ist
- (*3*) wird verwendet, wenn die Pakete zu installieren sind und ein Zugang zum Internet vorhanden ist

Nach dem Entpacken muss in den nun erhaltenen Ordner gesprungen und der Befehl "make" aufgerufen werden. Die Installation sollte mit den jeweiligen Voraussetzungen funktionieren, wenn das richtige Paket für die jeweilige Situation entpackt wurde. Da das Programm dafür ausgelegt ist auf neueren Linux Distributionen zu funktionieren, kann nicht garantiert werden, dass es auf anderen Systemen funktioniert. In diesem Fall muss der jeweilige Benutzer auf den Homepages von CGAL beziehungsweise QT und der seiner Distribution nachsehen, welche Pakete außer CGAL und QT noch benötigt werden. Theoretisch kann das Programm auch in einem Windows installiert werden, was aber nicht getestet wurde.

Um das Programm auszuführen, muss in den Ordner

```
.../files/binary
```

gewechselt werden. Wenn mit Option (1) installiert wurde, so wird lediglich der Aufruf in Zeile (1a) im folgenden Code Block benötigt, ansonsten muss der Aufruf in Zeile (2a) benutzt werden, denn dieser stellt sicher, dass die lokale Version benutzt wird.

```
(1a) ./StatisticalOutput
(2a) LD_LIBRARY_PATH=MY_PATH/files/binary/libs/lib/ ./StatisticalOutput
```

7.2 Übersicht über die Software

Wenn das Programm aufgerufen wird, so öffnet sich ein Hauptfenster, das sowohl die Menüführung beinhaltet, als auch eine Tabelle mit Testlaufparametern und Ausgabefeldern. Gestartet wird mit drei Popupfenstern, die nacheinander geöffnet werden und das Hauptfenster überlagern. Mit ihnen werden die Hauptwerte festgelegt, wie die Anzahl der Punkte pro Punktmenge, die Anzahl der Punktmengen und der Grad, auf den der Benutzer reduzieren möchte. Nach der Eingabe dieser Werte springt die Software von einer Phase zur nächsten, denn die Phasen benötigen ein erfolgreiches Abschließen der vorangegangenen Phasen.

Im Laufe eines Testlaufs werden folgende Phasen durchlaufen:

- *Datensatzerstellungsphase (Abschnitt 7.3)*
- *Triangulierungsphase (Abschnitt 7.4)*
- *Optimierungsphase (Abschnitt 7.5)*
- *Reduktionsphase (Abschnitt 7.6)*
- *Auswertungsphase (am Ende des Abschnitts 7.6)*

Keine dieser Phasen kann übersprungen werden und in jeder Phase hat der Benutzer eine Auswahlmöglichkeit, was er in dieser Phase tun möchte. Jede Phase verändert die Daten; um die Daten im nachhinein auswerten zu können, werden sie in jeder Phase abgespeichert. Da unter Umständen sehr viele Testdatensätze erstellt wurden, hat der Benutzer die Möglichkeit jede erstellte und bearbeitete Punktmenge zu selektieren um sie durch das Auslösen diverser Buttons anzusehen. Dazu wird pro selektierten Datensatz ein eigenes Fenster aufgemacht, in dem die Daten dieses Datensatzes angezeigt werden. Sobald eine Phase abgeschlossen wurde, können Optionen der Phase nicht mehr benutzt werden.

7.3 Punkterzeugungsphase

Es wurden drei Möglichkeiten zum Erzeugen von Punktmengen implementiert.

- Manuelle Eingabe
- Randomisieren
- Generieren

7.3.1 Manuelle Eingabe

Die manuelle Eingabe kann vom Benutzer in der Datensatzerstellungsphase aufgerufen werden. Das kann er tun, indem er den Button mit dem grünen Stift betätigt.

Es wird ein Fenster zu Eingabe geöffnet werden. Der Benutzer hat die Möglichkeit, Punkte zu definieren, indem er mit der Maus an die Stelle klickt, an der ein Punkt erzeugt werden soll. Er wird allerdings dazu gezwungen, die vorher eingegebene Anzahl der Punkte einzuhalten. Dies wird dem Benutzer Anhand eines Zählers im oberen Teil des Fensters angezeigt. Wenn die benötigte Anzahl erreicht wurde, so erscheint in der Menüleiste ein grüner Haken, mit dem der Benutzer die Punktmenge speichern kann (siehe Abbildung 7.1) Es steht auch die Option zur Verfügtung, die bisher definierten Punkte zu verwerfen und neu anzufangen. Sollte das Fenster geschlossen werden, wo wird die bisher eingegebene Punktmenge verworfen.

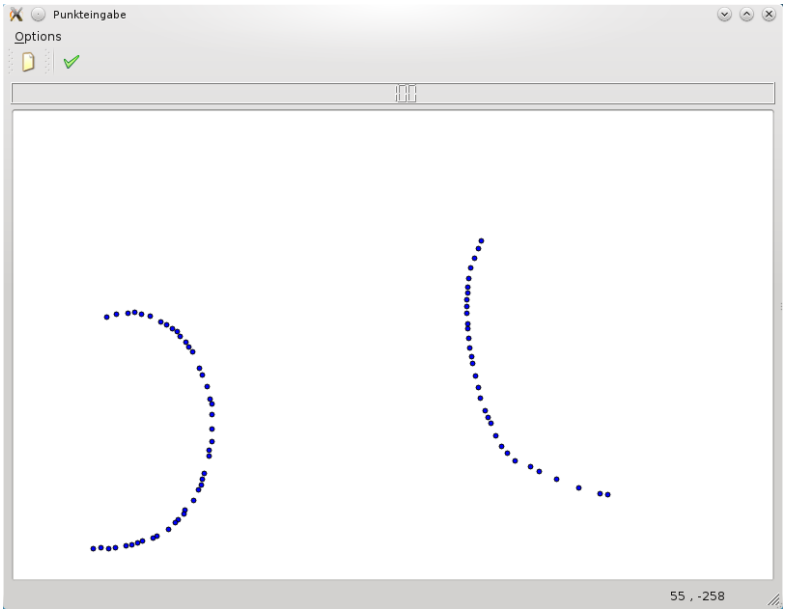


Abbildung 7.1: **Manuelle Eingabe** In diesem Bild ist das Fenster, in dem eine manuelle Eingabe der Punkte erfolgt ist zu sehen. Die Punkte werden mit der Maus gesetzt. Mit dem Linken Button (weißes Blatt) kann die momentane, eingegebene Punktmenge gelöscht werden und der Benutzer kann neu starten.

Mit dem Rechten Button (grüner Haken) kann die Punktmenge bestätigt werden. Damit wird die Eingabe bestätigt und die Punktmenge wird im Hauptprogramm übernommen und an die erste freie Stelle der Punktmengen gestellt.

Die manuelle Eingabe sollte nur ein bisher nicht betrachtetes Beispiel zu anderen generierten Punktmengen hinzufügen, um auszuprobieren, wie eine andere Punktmenge, die nicht generiert und nicht randomisiert wurde sich im Vergleich zu den anderen Punktmengen verhält.

7.3.2 Randomisiertes Erzeugen von Punktmengen

Der Algorithmus 1 wird aufgerufen, in dem der Benutzer in der Datensatzerstellungsphase den links stehenden Button betätigt. Diese Methode generiert Punkte in der Ebene, indem ein Rahmen mit fixer Größe erstellt wird um dann Punkte randomisiert innerhalb dieses Rahmens zu erzeugen. Hiermit werden die restlichen offenen Punktmengen erzeugt. Nach diesem Algorithmus wird noch ein Algorithmus aufgerufen, der die Kolliniaritäten beseitigt (Algorithmus 4). Ein Beispiel für eine randomisiert erzeugte Punktmenge wird in Abbildung 7.2 gezeigt.

Algorithmus 1 Pseudocode für randomisiertes Erstellen von Punkten.

```
 for dataset= current ... num_datasets_
        PointContainer points(num_points_);
        for counter = 0 ... num_points_
            x_coord = rand() % 6000 +20;
            y_coord = rand() % 4400 +20;
            Point_2 point(x_coord,y_coord);
            ok = points->SetPoint(point);
        endfor
endfor
```

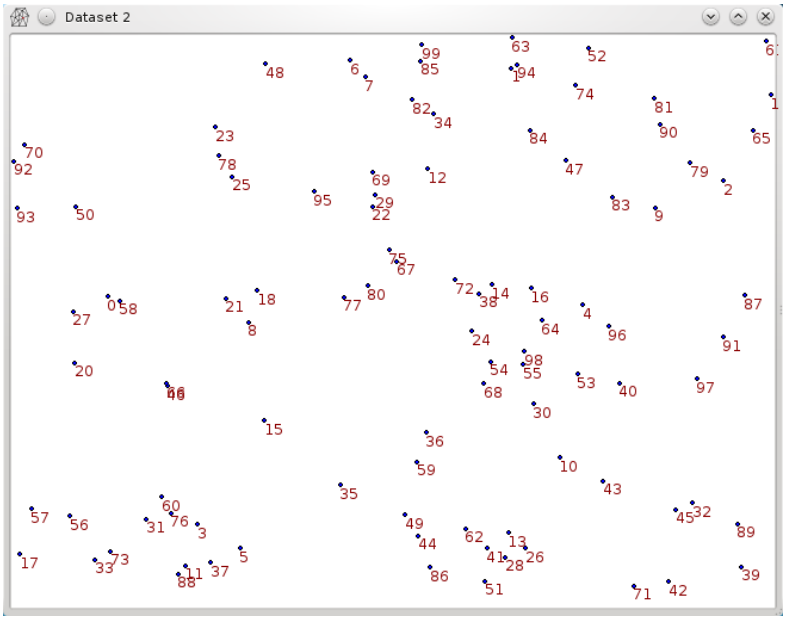


Abbildung 7.2: **Beispiel für eine randomisierte Punktmenge.** In diesem Bild ist ein Beispiel für eine randomisierte Punktmenge zu sehen, wie sie mit dem Programm erzeugbar sind und angezeigt werden können.

7.3.3 Generierung der Punktmengen

Dieser Algorithmus wird aufgerufen, indem der Benutzer den links stehenden Button betätigt. Um die Punktmengen so zu generieren, damit sie denen entsprechen, die in Kapitel 3 zu finden sind, müssen einige Schritte durchlaufen werden. Die Anzahl der Punkte, die erreicht werden soll, ist bekannt, da die Eingabe am Anfang vom Benutzer gemacht werden muss. Es wird der begrenzende Rahmen definiert analog dem Algorithmus für randomisierte Punktmengen. Aufgrund dieses Rahmens werden Bereiche um den Mittelpunkt dieses Rahmens definiert, indem die Länge zwischen Mittelpunkt und den Rahmen jeweils gedrittelt werden. Damit werden drei Bereiche in Form eines Fensters generiert. In der inneren Zone wird lediglich der Mittelpunkt des großen Kreissegments definiert. Der Radius dieses großen Segments wird so gewählt, dass das Kreissegment ganz in den inneren beiden Zonen zu liegen kommt. In der äußersten Zone werden die externen Punkte generiert.

Nachdem diese Zonen definiert wurden, werden nun die externen Punkte generiert. Es werden einer bis drei externe Punkte im Randbereich generiert. Für jeden generierten Punkt, wird die Anzahl der noch zu berechnenden Punkte nach unten korrigiert. Danach werden bis zu zwei Kreissegmente im Randbereich generiert, auf denen von zwei bis drei Punkte zu liegen kommen. Auch hier wird die Anzahl der noch zu berechnenden Punkte nach unten korrigiert.

Danach wird ein Kreis generiert, dessen Mittelpunkt im Innersten Bereich liegt und dessen Radius so gewählt ist, dass kein Punkt des Kreises im Randbereich zu liegen kommt. Nun werden zwei Winkel generiert (startingangle und segmentangle). Der Startpunkt und der Endpunkt des Kreissegments kann mit Algorithmus 2 berechnet werden.

Um die restlichen Punkte äquidistant auf diesem Kreissegment zu verteilen, wird Algorithmus 3 angewendet.

Bemerkung 7.1. *(Exaktheit des Algorithmus)* Diese Generierung der Punktmenge erreicht nicht ganz ihr Ziel, denn die Zonen, wie sie in Kapitel 3 definiert wurden, werden nicht eingehalten. Das ist auch nicht unbedingt nötig, denn die Wahrscheinlichkeit, eine Punktmenge zu generieren, deren externe Punkte in der konvexen Zone liegen, liegt derzeit bei 5 bis 20 Prozent der Datenbeispiele. Das gibt den Punktmengen zum einen eine gewisse Variation und zeigt zum anderen, dass die Optimierungsalgorithmen in diesem Fall trotzdem funktionieren. Unter Abbildung 7.3 ist ein Beispiel für eine solche Punktmenge zu sehen.

Man könnte den Algorithmus um Punktmengen zu erzeugen noch deutlich verbessern, indem zuerst das Kreissegment definiert wird und dann die Zonen des Kreissegments bestimmt werden. Aufgrunddessen können dann die externen Punkte in der jeweiligen Zone definiert werden. Der Rest dieses Algorithmuses bleibt gleich.

Am Ende dieses Kapitels (Algorithmus 4) wird noch der Pseudocode für das Entfernen von Kollinearitäten gezeigt.

Algorithmus 2 Pseudocode für das Berechnen des Start- bzw Endpunktes eines Segments.

```
value = segmentangle/numrest

x= big_center.x()
x = x + radius * cos(value)
y = big_center.y()
y = y + radius * sin(value)

Point_2 start_point = Point_2(x, y);

value = (startingangle + segmentangle)*PI/180

x = big_center.x();
x = x + radius * cos(value);
y = big_center.y();
y = y + radius * sin(value);

Point_2 end_point = Point_2(x, y);
```

Algorithmus 3 Pseudocode für das Finden von Kollinearitäten.

```
distance_angle = segment_angle/num_rest_points

for point_counter = 0 ... num_rest_points
   angle = distance_angle*PI/180

   x = circle_center.x()
   y = circle_center.y()

   x = x + radius * cos(angle)
   y = y + radius * sin(angle);

   Point_2 calc_point(x,y);
   ok = points->SetPoint(calc_point);
   distance_angle += calc_angle;
endfor
```

Algorithmus 4 Pseudocode für das Finden und entfernen von Kollinearitäten durch Verschieben des Punktes entlang der Normalgeraden.

```
is_col = false;

for idx1 = 0 ... num_points_
   for idx2 = idx1+1 ... num_points_
      for idx3 = idx2+1 ... num_points_
        is_col = collinear(points[idx1], points[idx2], points[idx3])
        if(is_col)
        {
           norm_vec.x() = (points->at(idx1).y()-points->at(idx2).y())
           norm_vec.y() = -(points->at(idx1).x() - points->at(idx2).x())

           norm_vec = norm_vec * epsilon
           points[idx3].x() += norm_vec.x()
           points[idx3].y() += norm_vec.y()
           is_col = false;
   idx3--;
        }
      endfor
   endfor
endfor
```

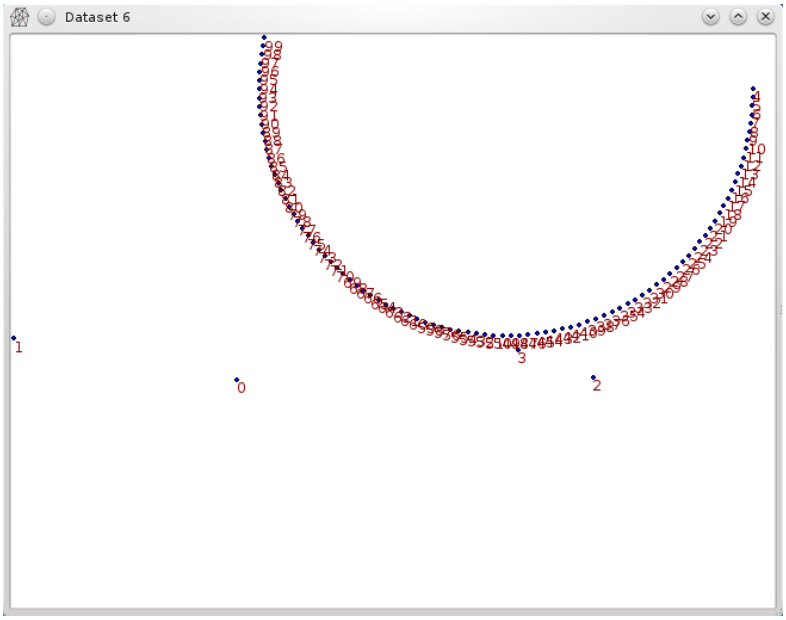


Abbildung 7.3: **Eine generierte Punktmenge mit 100 Punkten.** In diesem Bild sieht man ein Beispiel für eine generierte Punktmenge, wie sie mit dem Programm erzeugbar ist und angezeigt werden kann.

7.4 Triangulierungsphase

Die Software benutzt vier Triangulierungsalgorithmen. Sie wurden in Kapitel 4 genauer beschrieben. Die vier Methoden stehen im Kombinationsfeld zur Auswahl und können mit dem Button links erstellt werden. Zur Auswahl stehen die Standardtriangulierung, die Delaunay-Triangulierung, die Greedy-Triangulierung und die randomisierte Standardtriangulierung.

7.4.1 Standardtriangulierungsalgorithmus

Eine Standardtriangulierung wird in dieser Software durch die Ausführung der normalen Triangulierungsalgorithmen von CGAL. Algorithmen und Funktionalitäten sowie Laufzeitverhalten wurden im Kapitel über Triangulierungen der Referenz [12] behandelt.

7.4.2 Delaunay-Triangulierung

Eine Delaunay-Triangulierung wird in dieser Software ebenfalls durch die Ausführung der in CGAL bereits vorhanden Delaunay-Algorithmen erzeugt. Algorithmen und Funktionalitäten sowie Laufzeitverhalten wurden im Kapitel über Delaunay-Triangulierungen in der Referenz [12] behandelt.

7.4.3 Greedy-Triangulierung

Die Greedy-Triangulierung basiert auf die Constained Triangulations der CGAL Bibliothek. Diese Triangulierungsart basiert auf ein Kantenskelett, das von CGAL selbst trianguliert wird. Hier wird ebenfalls die Referenz [12] empfohlen.

Der Algorithmus um eine Greedy-Triangulierung zu erzeugen folgt dem Brute-force Schema:

1. Erzeuge alle möglichen Kanten der Punktmenge und speichere sie in einen Container (zum Beispiel List)
2. solange Kanten im Container vorhanden sind führe Schritt 3 bis 6 aus
3. Hole die kürzeste Kante aus dem Container, nenne sie e
4. Entferne alle Kanten aus dem Container, die e schneiden (intersect Algorithmus von CGAL wird aufgerufen)
5. Füge e in die Constrained Triangulierung ein

Die Brute-Force Implementation der Greedy-Triangulierung ist sehr langsam. Im Vergleich dazu ist die in diesem Paper [3] um vieles effizienter umgesetzt.

7.4.4 Randomisierte Standardtriangulierung

Laut Kapitel 4 ist die Standardtriangulierung sehr stark von der Einfügereihenfolge der Punkte in die Triangulierung abhängig. Um eine gleichverteilte Randomisierung der Punkte zu erreichen, wurde folgendes Verfahren angewandt.

1. Erstelle einen Vektor (ist in der Standardbibliothek stl vorhanden) mit Einträgen von 0 bis n, wobei n die Anzahl der Punkte ist
2. Rufe den Randomisierungsbefehl des Vektors "randomshuffle" (Algorithmus der Standardbibliothek) auf
3. Iteriere über diesen Vektor und füge den Punkt an der Position, die der Vektor liefert, in die Standardtriangulierung ein

7.5 Optimierungsphase

Im Programm wurden die zwei Optimierungsalgorithmen implementiert, die in Kapitel 5 beschrieben wurden. Er entscheidet das in der Optimierungsphase des Programms, indem er im Kombinationsfeld die Art der Optimierung auswählt und betätigt.

Die Software unterstützt drei Varianten der Optimierung:

- keine Optimierung
- Wiedereinfügealgorithmus
- Flipalgorithmus

7.5.1 Wiedereinfüge-Algorithmus

Dieser Algorithmus (Algorithmus 6) sucht den gradhöchsten Knoten einer Punktmenge um diesen dann aus der Triangulierung zu entfernen. Danach fügt er ebendiesen Knoten wieder in die Triangulierung ein, indem der CGAL- Einfügealgorithmus aufgerufen wird.

Die Hauptschleife des Algorithmuses:

Die Funktion in Zeile (3) berechnet den aktuellen höchsten Grad der Triangulierung. In Zeile (4) ist die Abbruchbedingung für den Algorithmus zu finden. Der Wiedereinfügealgorithmus soll abbrechen, wenn entweder keine weiteren Schritte mehr möglich sind, zweimal die Anzahl der Punkte der Punktmenge durchlaufen wurden, oder der Zielgrad erreicht wurde. In Zeile (8) wird die eigentliche Optimierungsfunktion aufgerufen.

Unter Algorithmus 7 ist die Optimierungsfunktion zu sehen.

Algorithmus 5 Pseudocode: Hauptschleife des Wiedereinfügealgorithmuses.

```
(1) for num_loops = 0 ... 2 * num_points_
(2)  {
(3)      deg = get_max_degree_absolute();
(4)      if(!done_something || (deg < target_degree_))
(5)      {
(6)          break;
(7)      }
(8)      done_something = remove_highest_and_reinsert_it()
(9)  }
```

Algorithmus 6 Pseudocode: Optimierungsfunktion der Wiedereinfügemethode.

```
done_something = true
max_degree_index = get_max_degree_index()
if(max_degree_index != num_points_)
{
   Point_2 point

   if(vertex.at(max_degree_index)->getDegree() > target_degree_)
   {
     worst_vertex = triang_datasets_.at(max_degree_index)->getVertex()
     point = worst_vertex->point()
     current_degree = worst_vertex->degree()
     t_->remove(worst_vertex)

     worst_vertex = t_->insert(point)
     new_degree = worst_vertex->degree()

     for point_idx = 0 ... num_points_
        triang_datasets_.at(point_idx)->update()
     endfor
   }
   else
   {
       done_something = false
   }
   return done_something;
}
```

7.5.2 Flip-Algorithmus

Dieser Algorithmus ist einem Greedy-Algorithmus sehr ähnlich, denn er berechnet ebenso wie Greedy den momentan nächstbesten Schritt und führt diesen aus. Der Flipp-Algorithmus iteriert hierbei über alle inneren Kanten und entscheidet über den Gradvektor der angrenzenden Dreiecke, ob ein Flip möglich ist und ob dieser eine Verbesserung bringt. Um eine Kante zu klassifizieren, geht dieser Kante laut 5.1 in Kapitel 5 vor. Hier wird veranschaulicht, wie die Datenstruktur aussieht, mit der dies bewerkstelligt werden kann und wie der Flip gemacht wird.

Datenstruktur: neighborpairfaces In dieser Datenstruktur besteht im wesentlichen aus der betrachteten Kante und den angrenzenden Dreiecken. Sollte die Kante auf der konvexen Hülle liegen, so ist eines dieser angrenzenden Dreiecke bereits mit "infinitive" von CGAL selbst gekennzeichnet worden. Die Datenstruktur hat Aufgrund der Dreiecke und der Beschaffenheit von "face" (in CGAL siehe Manual) Zugriff auf alle Knoten, die zu dieser Kante gehören. Die Datenstruktur "edge" von CGAL wurde hier bewusst nicht herangezogen (die Kante selbst ist durch ihre Endpunkte definiert), weil diese in CGAL sehr wenig Eigenschaften hat (zum Beispiel kennt die Datenstruktur "edge" ihre angrenzenden Dreiecke nicht, ein "vertex" dagegen schon). neighborpairfaces speichert außerdem den Gradvektor des entstehenden Vierecks, wenn die Kante nicht auf der konvexen Hülle war. Sie hat eine interne Funktion "isflippable()" die berechnet, ob die Kante flipbar ist und einen Wert in der Struktur setzt. Gewisse Werte dieses Vierecks müssen nur einmal ermittelt werden (sie ändern sich für ein unveränderliches Viereck nicht), wie *flipbar* und *auf KH*.

Diese Klasse hat vier Funktionen, auf die hier genauer eingegangen wird:

- *"isflippable()"*
- *"filllists()"*
- *"classify()"*
- *"flip()"*

Die Funktion "isflippable()" untersucht, ob die beiden gespeicherten, benachbarten Dreiecke ein konvexes Viereck bilden. Das wird durch folgendem Pseudocode erreicht:

Wobei die CGAL Funktion "intersection(segment1,segment2)" aus zwei Segmenten, in diesem Fall zwei Kanten, einen Schnittpunkt errechnet. Falls einer gefunden wird, so ist laut der Definition der konvexen Vierecke aus Kapitel 2 ein konvexes Viereck vorhanden, das ebenso laut diesem Kapitel flipbar ist.

Die Funktion "filllists()" simuliert einen Flip, falls er möglich ist und berechnet den neuen Gradvektor, der nach dem Flip gültig wäre. Dazu wird die Knoteneingenschaft shared- Knoten ist Teil beider Dreieck oder isolated - Knoten ist nur Teil eines Dreiecks, verwendet. Wenn ein Flip durchgeführt werden würde, so wird diese Kante gelöscht und durch die Kante zwischen den beiden isolierten Knoten ersetzt. Dadurch wird der Grad der shared Knoten um eins gesenkt und der Grad der isolated Knoten um eins erhöht. Daher kann der neue Gradvektor sehr einfach berechnet werden. Beide Gradvektoren

Algorithmus 7 Pseudocode: Prüfen, ob zwei benachbarte Dreiecke ein konvexes Viereck bilden.

```
Point_2 points[4];

points[0] = vertex_1->point()
points[1] = vertex_2->point()
points[2] = vertex_3->point()
points[3] = vertex_4->point()

S segment1(points[0],points[1])
S segment2(points[2],points[3])


intersect = false;
Point_2 point;

CGAL::Object obj = intersection (segment1, segment2)
if (assign(point, obj))
   intersect = true;

 if (intersect)
   flippable
 else
  not flippable
```

werden absteigend sortiert und die Differenz der beiden Vektoren gibt den Differenzvektor, der nur aus 0, 1 und -1 bestehen kann, wobei immer eine gerade Anzahl an Nullen vorhanden sein muss, sowie gleich viele Einsen wie negative Einsen. Aufgrund dieses Differenzvektors kann erst klassifiziert werden.

Die Funktion “classify()“ klassifiziert die Kante mit ihrem Differenzvektor laut 5.1 in Kapitel 5. Zusätzlich wird berücksichtigt, dass die Kante auf der konvexen Hülle liegen oder nicht flipbar sein kann (”classified” wird auf 0 gesetzt). Wenn “classified“ mit 2 gesetzt wurde, so liegt ein neutraler Flip vor, also ein Flip der weder gut noch schlecht ist. Ist klassified 1 so bringt dieser Flip nur eine momentane Verschlechterung.

Die Funktion ”flip()” führt den vorher simulierten Flip aus.

Optimierung mit Flips Die aufrufende Funktion hat nur eine Aufgabe: Sie muss eine Liste über alle möglichen neighborpairfaces verwalten und nach der höchsten Klassifikation in dieser Liste suchen. Falls die höchste Klassifikation nur mehr 2 ist, so wird noch eine fixe Anzahl an neutralen Flips augeführt, in der Hoffnung durch diesen Flip einen anderen, besseren Flip zu ermöglichen. Wenn die höchste Klassifikation 1 ist, so wird abgebrochen, denn man kann mit diesem Algorithmus nun keine Verbesserung mehr erzielen. Sollte geflippt worden sein, so wird diese Liste geleert und wieder neu erzeugt, denn nach einem Flip sind weder die benachbarten Dreiecke, noch die betroffenen Knoten gültig. Es ist effizienter die Liste neu aufzubauen.

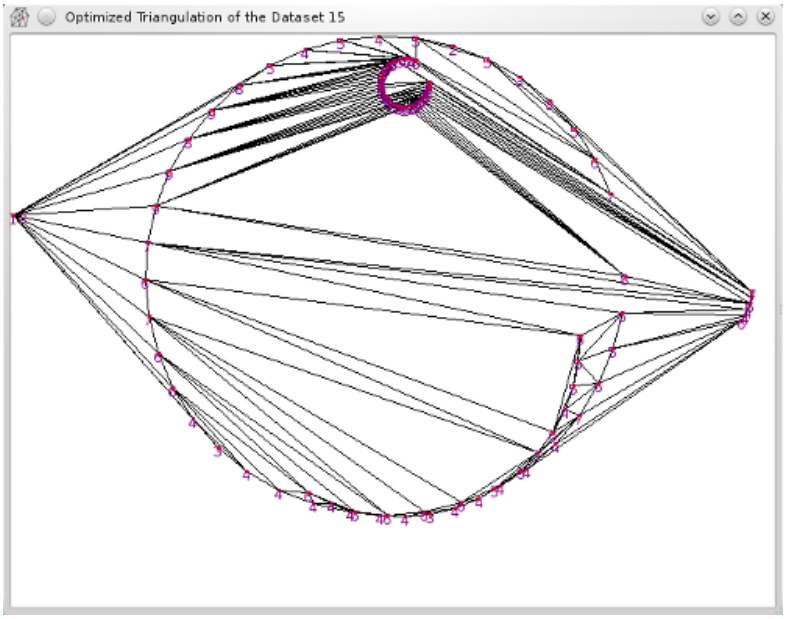


Abbildung 7.4: **Punktmenge mit optimierter Triangulierung.** In diesem Fall sieht man eine triangulierte Punktmenge, wobei die Triangulierung bereits optimiert wurde. Es ist sofort ersichtlich, dass diese Methode versagen muss, wenn ein konkaver Fächer (im Bild links) entsteht.

Bemerkung 7.2. *(Dieser Algorithmus findet nicht unbedingt die gradoptimale Triangulierung)* Die gradoptimale Triangulierung kann nicht erreicht werden, weil unter Umständen

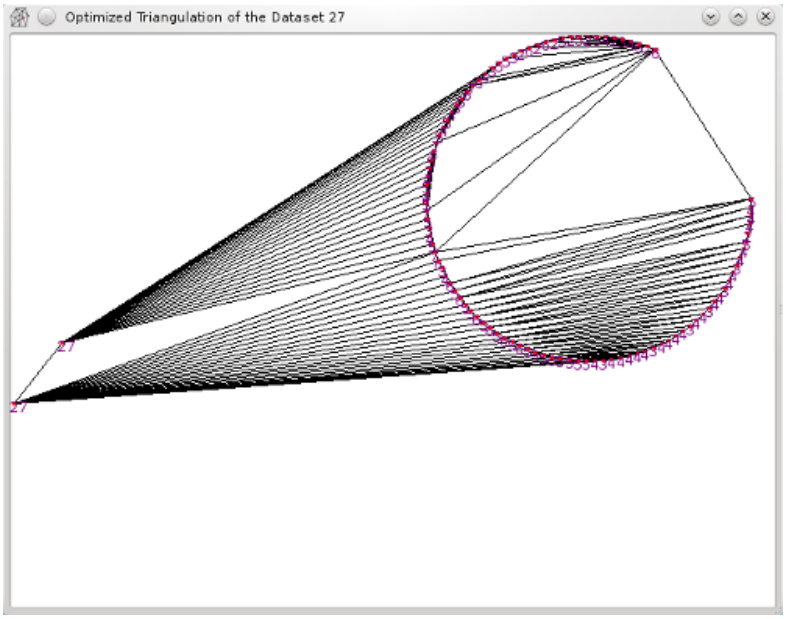


Abbildung 7.5: **Punktmenge mit einem Punktpaar als externe Punkte.** Hier ist zu sehen, dass diese Optimierungsmethode die Grade von zwei extremen Punkten aufteilt.

ein schlechter Flip ausgeführt werden muss um einen essenziellen Flip zu ermöglichen. Das kann ganz einfach in folgender Abbildung 7.6 gesehen werden.

Unter Abbildung 7.4 ist eine Optimierung mit dem Flip-Algorithmus ausgeführt worden. Es wurde nicht der Zielgrad (hier 10 erreicht, aber an Stellen im Inneren der radialen Punktkette wurde erfolgreich optimiert. Unter Abbildung 7.5 ist zu sehen, dass bei Versagen der Methode mit einer Mehrdeutigkeit (links im Bild) der Grad auf die zwei Punkte aufgeteilt wird. In Abbildung 7.6 ist zu sehen, dass die Methode versagen kann.

7.6 Reduktionsphase

In diesem Teil werden die Reduktionsalgorithmen und deren Implementation erläutert, wie sie in Kapitel 6 beschrieben wurden. Zuerst wird auf die Entfernung von Punkten eingegangen, dann wird das Thema “Hinzufügen von Punkten” behandelt. Der Benutzer startet diese Algorithmen in der Reduktionsphase des Programms, indem er den gewünschten Reduktionsalgorithmus im Kombinationsfeld auswählt und den Button links betätigt.

7.6.1 Auf Punktentfernung basierende Algorithmen

Der Kern dieser Algorithmen ist immer eine Schleife, die abgebrochen wird, wenn der Grad erreicht wurde oder kein vernünftiger Knoten mehr vorhanden ist. In dieser Schleife

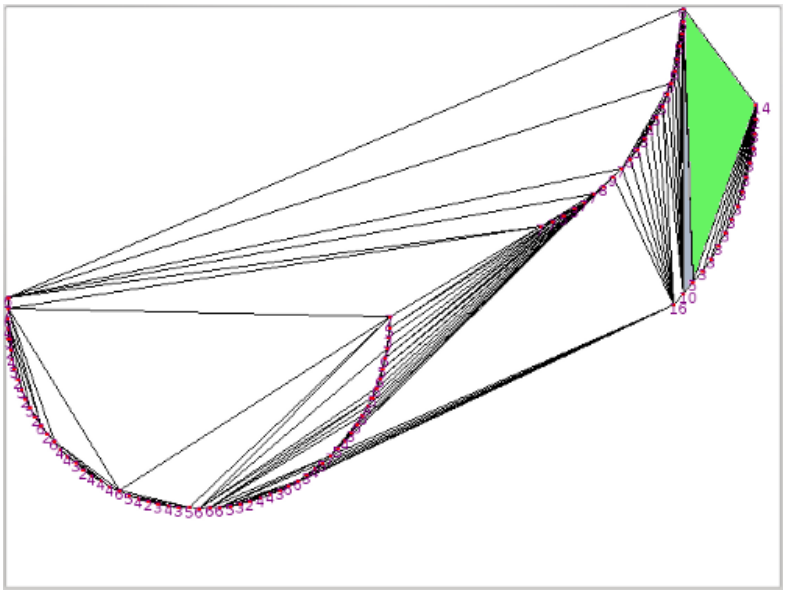


Abbildung 7.6: **Ein schlechter Flip müsste ausgeführt werden.** Bei diesem Bild ist die Situation, die am Ende von Kapitel 5 beschrieben wurde eingetreten. Im markierten Bereich müsste ein verschlechternder lokaler Flip ausgeführt werden, damit der Rest optimiert werden kann. Da dies nur ein Screenshot mit sehr schlechter Qualität ist, wird hier auf die Abbildungen im oben referenzierten Kapitel verwiesen (Abbildung 5.5 und Abbildung 5.6).

wird ein Knoten selektiert um diesen zu löschen. Alle Container, in denen der Knoten enthalten war, müssen aktualisiert werden. Dann muss die Triangulierung optimiert werden (denn wie wir im nächsten Kapitel sehen werden, kann ein solcher Algorithmus ansonsten nie effizient laufen). Der Algorithmus wird so lange wiederholt, bis entweder keine Knoten mit dem gewünschten Merkmal vorhanden sind, oder der gewünschte Grad erreicht wurde. In dieser Arbeit wurden drei Selektionsmethoden getestet:

- *Selektieren des gradhöchsten Punkt, der nicht auf der konvexen Hülle liegt*
- *Selektieren des gradhöchsten Nachbarn vom gradhöchsten Punkt, der nicht auf der konvexen Hülle liegt*
- *Selektieren eines randomisiert ausgewählten Nachbarn des gradhöchsten Punktes, der nicht auf der konvexen Hülle liegt*

Selektieren des gradhöchsten Punktes, der nicht auf der konvexen Hülle liegt

Dafür wurde Algorithmus 8 implementiert.

Dieser Algorithmus iteriert lediglich über alle Knoten der Triangulierung und vergleicht den Grad des aktuellen Knotens mit dem bisher gefundenen Knoten, wenn der aktuelle Knoten nicht auf der konvexen Hülle liegt. Sollte der Knoten einen höheren Grad aufweisen, so wird sowohl der aktuell höchste Grad, als auch der Knoten selbst gespeichert.

Algorithmus 8 Pseudocode: Selektieren des gradhöchsten Punktes, der nicht auf der konvexen Hülle liegt.

```
highest_degree_index = 0;
   highest_degree = 0;

   found_something = false;
   point_idx = 0;

for it = vertex.begin() ... vertex.end()
   if(not_on_bound(it))
   {
       if(it.getDegree() > highest_degree)
       {
           highest_degree = it->getDegree()
           highest_degree_vertex = it
       }
   }
endfor
return highest_degree_vertex
```

Selektieren des gradhöchsten Nachbarn des gradhöchsten Knotens

Dieser Algorithmus benötigt eine Funktion um den Knoten mit dem höchsten Grad zu selektieren. Dieser Knoten darf auf der konvexen Hülle liegen, was den einzigen Unterschied zum vorherigen Algorithmus darstellt. Sobald der gradhöchste Punkt gefunden wurde, wird über die Nachbarn iteriert. Bei der Iteration wird geprüft, ob der Nachbar auf der konvexen Hülle ist. Sollte er nicht auf der Hülle liegen, so wird der Grad des Nachbarn mit dem bisher gefundenen Nachbarn verglichen. Wurde ein höherer Nachbar gefunden, so wird sowohl der Grad, als auch der Knoten selbst gespeichert. Am Ende der Iteration sollte genau ein Knoten selektiert sein.

Selektieren eines randomisiert ausgewählten Nachbarn des gradhöchsten Knotens

Hier wird der selbe Algorithmus verwendet, der in Sektion 2 verwendet wurde, um den gradhöchsten Knoten zu finden. Am Anfang wird eine Zufallszahl zwischen 0 und k generiert, wobei k der Grad des gradhöchsten Knotens ist. Danach wird über die Nachbarknoten iteriert und eine Zahl wird hochgezählt, wenn der Knoten nicht auf der konvexen Hülle liegt. Entspricht die Zahl der Zufallszahl, so wird dieser Knoten selektiert.

7.6.2 Punkthinzufügen

Um Punkte in die Triangulierung hinzuzufügen, muss zuerst die Position an der ein Punkt eingefügt wird, ermittelt werden. Der Rest wird ebenfalls über die CGAL Tri-

angulierungsstruktur gelöst. Sobald ein neuer Punkt hinzugefügt wurde, wird die Triangulierung mit den Optimierungsalgorithmen optimiert, bevor erneut selektiert werden kann. Die Selektion hängt vorallem von der Wahl des Dreiecks ab, in dem der Punkt erzeugt werden soll. Es ist klar, dass man innerhalb eines Dreiecks einen Punkt einfügen muss, denn es sind weder Kollinearitäten erwünscht, noch darf ein Punkt außerhalb der konvexen Hülle generiert werden. Zu beachten ist, dass die Wahl des Dreiecks wichtiger ist als die Berechnung des Punktes innerhalb dieses Dreiecks, denn durch die Wahl des Dreiecks erhält man wesentlich mehr Einschränkungen als mit der Berechnung des Punktes. In Abbildung 7.7 sind drei mögliche Methoden zur Punktplatzierung innerhalb eines Dreiecks dargestellt.

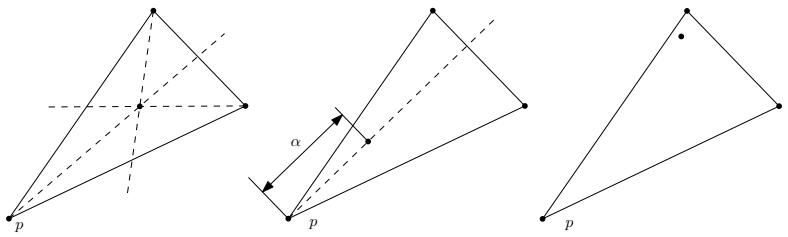


Abbildung 7.7: **Möglichkeiten um einen Punkt in ein Dreieck einzufügen.** Hier wird gezeigt, mit welchen Möglichkeiten ein Punkt in einem Dreieck eingefügt werden kann. Im linken Bild wird gezeigt, wie ein Punkt an der Stelle des Schwerpunktes platziert wird, wogegen im mittleren eine Möglichkeit mit Distanzberechnungen auf gezeigt wird. In diesem Fall wird auf der Schwerline ein Punkt platziert, indem die Distanz zwischen p und der gegenüberliegenden Kante mit dem goldenen Schnitt geteilt wird. Im rechten Bild wird ein randomisierter Algorithmus benutzt, der einen Punkt an einer beliebigen Stelle im Dreieck generiert.

Im Programm gibt es zwei Methoden, um ein Dreieck zu selektieren:

- *Das gradhöchste Dreieck*
- *Ein beliebiges Dreieck, das an den Knoten mit dem höchsten Punkt enthält*

Das gradhöchste Dreieck

Jedem Dreieck kann ein Wert zugewiesen werden, der dem Grad des Dreiecks entspricht. Dieser Wert wird durch die Summe der einzelnen Grade der Knotenpunkte, die zum jeweiligen Dreieck gehören, berechnet. Aus diesem Grund muss das Dreieck mit dem höchsten Grad den Knotenpunkt mit dem höchsten Grad beinhalten.

Um das gradhöchste Dreieck zu finden ist es daher nur notwendig, dass die dem gradhöchsten Punkt angrenzenden Dreiecke betrachtet werden. Damit kann über die dem gradhöchsten Punkt p benachbarten Dreiecke iteriert werden, um das gradhöchste Dreieck zu berechnen. Sobald dieses Dreieck gefunden wurde, kann mit der gewählten Methode ein Punkt innerhalb dieses Dreiecks berechnet werden (siehe Abbildung 7.8).

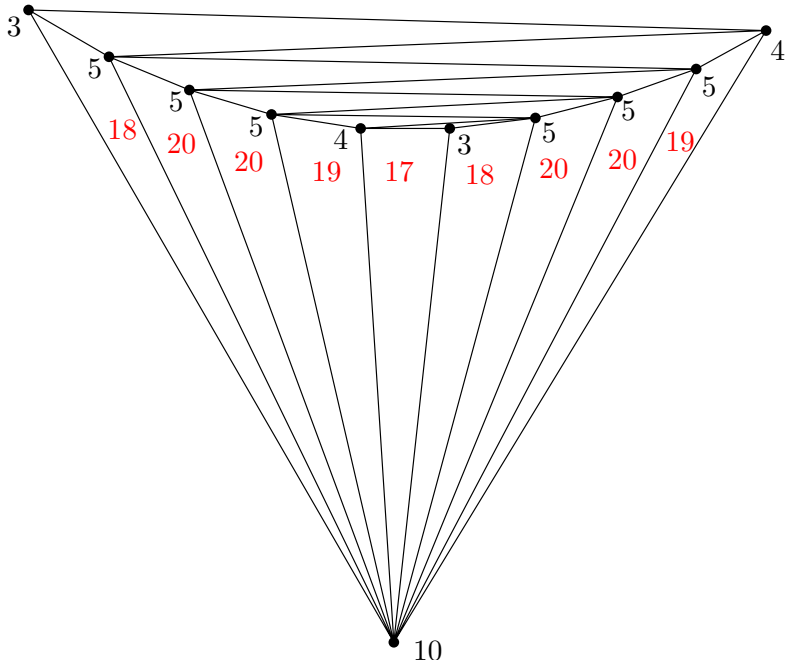


Abbildung 7.8: **Grade der Dreiecke einer Triangulierung.**
In diesem Bild werden Grade der Dreiecke veranschaulicht. Die schwarzen Zahlen beziffern hier die Grade der Knoten und die roten Zahlen beziffern den Grad der einzelnen Dreiecke. Der Dreiecksgrad errechnet sich durch die Summe über die Knotengrade der Knoten, die zum Dreieck gehören.

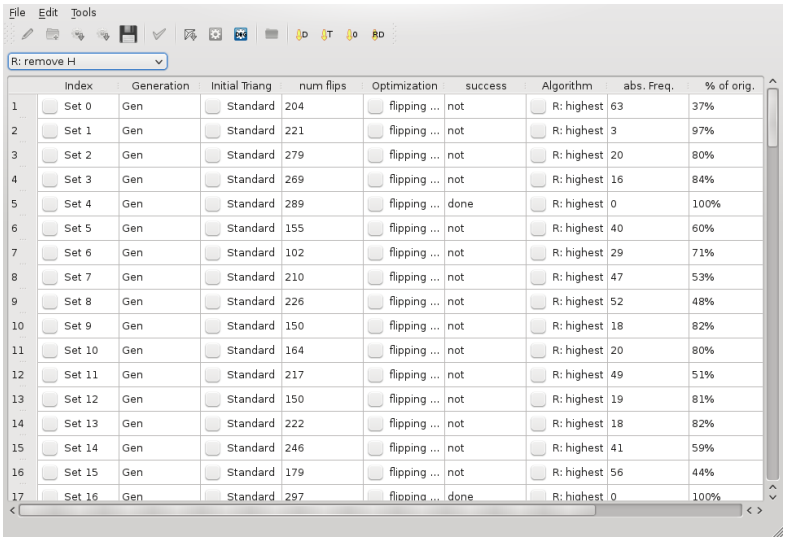
Ein radomisiert ausgewähltes Nachbardreieck

In diesem Fall wird ein Dreieck ausgewählt, das als einzige Bedingung eine unmittelbare Nachbarschaft an den gradhöchsten Knoten haben muss. Dafür wird wieder über die an den Punkt p angrenzenden Dreiecke iteriert um ein beliebiges auszuwählen. Danach wird der Punkt in diesem Dreieck generiert und dieser wird in die Triangulierung eingefügt.

7.7 Benutzung und GUI

Damit die Bedienbarkeit des Programms erleichtert wird, wurde mit QT ein Grafisches User Interface (GUI) implementiert. Es wurde durch gezielte Deaktivierung und Aktivierung von Schaltflächen sichergestellt, dass der richtige Ablauf durch das Programm funktioniert. Dazu wird in Abbildung 7.9 ein Bild des Hauptfensters mit dessen Teilen gezeigt.

Am Anfang steht nur eine begrenzte Auswahl zur Verfügung, denn der Benutzer befindet sich noch in der Phase, in der er die Punktmengen definieren muss (Datenseterzeugungsphase). Daher sind nur sehr wenige Auswahlmöglichkeiten vorhanden und die Tabelle im Herz des Fensters ist noch leer. Er hat folgende Menüleiste zur Verfügung



	Index	Generation	Initial Triang	num flips	Optimization	success	Algorithm	abs. Freq.	% of orig.
1	Set 0	Gen	Standard	204	flipping ...	not	R: highest	63	37%
2	Set 1	Gen	Standard	221	flipping ...	not	R: highest	3	97%
3	Set 2	Gen	Standard	279	flipping ...	not	R: highest	20	80%
4	Set 3	Gen	Standard	269	flipping ...	not	R: highest	16	84%
5	Set 4	Gen	Standard	289	flipping ...	done	R: highest	0	100%
6	Set 5	Gen	Standard	155	flipping ...	not	R: highest	40	60%
7	Set 6	Gen	Standard	102	flipping ...	not	R: highest	29	71%
8	Set 7	Gen	Standard	210	flipping ...	not	R: highest	47	53%
9	Set 8	Gen	Standard	226	flipping ...	not	R: highest	52	48%
10	Set 9	Gen	Standard	150	flipping ...	not	R: highest	18	82%
11	Set 10	Gen	Standard	164	flipping ...	not	R: highest	20	80%
12	Set 11	Gen	Standard	217	flipping ...	not	R: highest	49	51%
13	Set 12	Gen	Standard	150	flipping ...	not	R: highest	19	81%
14	Set 13	Gen	Standard	222	flipping ...	not	R: highest	18	82%
15	Set 14	Gen	Standard	246	flipping ...	not	R: highest	41	59%
16	Set 15	Gen	Standard	179	flipping ...	not	R: highest	56	44%
17	Set 16	Gen	Standard	297	flipping ...	done	R: highest	0	100%

Abbildung 7.9: **Hauptfenster des Programms.** In diesem Bild ist das Hauptfenster der GUI zu sehen. Am oberen Rand befindet sich die Menüleiste. Im Inneren des Fensters ist eine Tabelle zu sehen, über die die einzelnen generierten Bilder zum öffnen selektiert werden können.

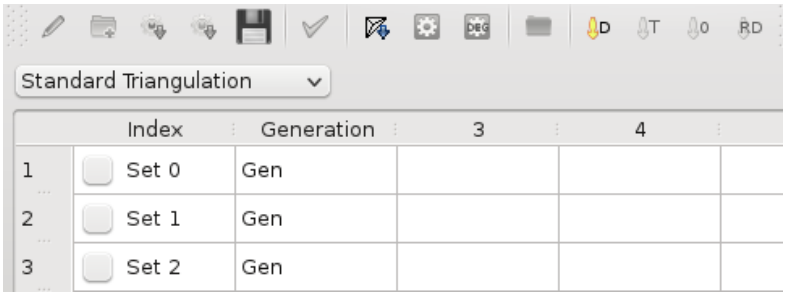
(Abbildung 7.10):

	grüner Stift: Damit kann ein Fenster geöffnet werden, das eine manuelle Eingabe der Punkte durch den Benutzer ermöglicht.
	blauer Folder mit grünem Plus: Für eine erneute Simulation dieser Punktmengen können bereits erstellte und gespeicherte Datensets geladen werden.
	rotes Zahnrad mit rotem Pfeil: Dies ruft den Algorithmus auf, der die restlichen, noch nicht erstellten Datensets mit generierten Punktmengen füllt.
	blaues Zahnrad mit blauem Pfeil: Damit werden die restlichen noch offenen Punktmengen randomisiert generiert.

Abbildung 7.10: **Menüführung der Dateneingabephase mit der Funktion der Buttons.**

Sobald die vorher eingegebene Anzahl an Datensets erreicht wurde, können die erstellten Punktmengen gespeichert werden. Durch den grünen Haken, der nun freigeschalten wurde, kann der Benutzer in die nächste Phase (Triangulierungsphase) wechseln. In dieser Phase steht ein Kombinationsfeld zur Verfügung, mit der die Art der Anfangstriangulierung auswählbar ist (siehe Abbildung 7.11). In der ersten Spalte des Hauptfensters

werden die Datensets angezeigt mit einer Auswahlmöglichkeit und in der zweiten Spalte steht ein Kürzel für die Art mit der die Punktmenge erstellt wurde (Gen für Generiert, Rand für Randomisiert und Usr für manuelle Eingabe).



	Diskettensymbol: Damit können die Punktmengen gespeichert werden. Es wird ein Fenster geöffnet, in dem der Speicherdialog erscheint.
	Triangulierungssymbol mit Pfeil: Das Drücken dieses Buttons erstellt die Triangulierungen der Punktmengen mit dem im Kombinationsfeld ausgewählten Triangulierungsalgorithmus.
	gelber Pfeil nach unten mit "D": Mit diesem Symbol können die im Hauptfenster selektierten Datensets angesehen werden. Dazu wird ein Fenster geöffnet in dem die Punkte angezeigt werden, mit der Reihenfolge, in der die Punkte erstellt wurden.

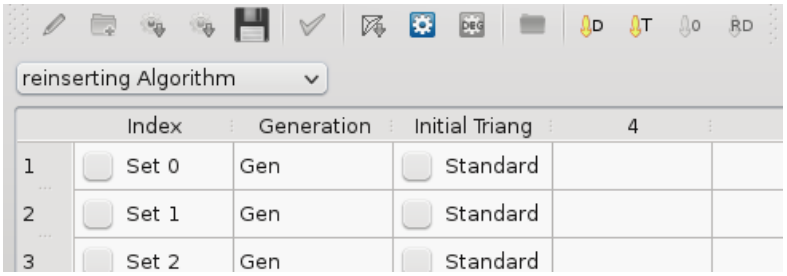
Abbildung 7.11: **Menüführung der Triangulierungsphase mit der Funktion der Buttons.**

Das Erzeugen der Triangulierung fügt noch weitere Spalten im Hauptfenster hinzu in der die Punktmenge mit der Triangulierung selektiert werden kann. Es werden weitere Optionen freigeschalten, denn das Programm befindet sich wieder in einer weiteren Phase (Optimierungsphase). In dieser Phase wird das Kombinationsfeld mit anderen Werten bestückt damit die Auswahl der Optimierungsmethoden möglich ist. Die Menüführung wird in Abbildung 7.12 gezeigt.

Nachdem die Optimierung erfolgt ist, werden zwei Fenster geöffnet, in denen eine Statistik über die Optimierung und deren Erfolge abzulesen ist. Das erste Fenster zeigt den Mittelwert der Anzahl Flips über alle Datenbeispiele (wenn keine Optimierung oder der Wiedereinfügealgorithmus aufgerufen wurde, so beträgt der Mittelwert und die Varianz 0). Ein Beispiel für dieses Fenster wird in Abbildung 7.13 gezeigt.

Im zweiten Fenster ist die Varianz der Anzahl der Flips zu sehen (ohne Abbildung). Im dritten Fenster wird die Erfolgsrate der Optimierung ausgegeben. Sie gibt prozentuell an, wieviele der Beispiele mit dem Algorithmus erfolgreich auf den Wunschgrad gebracht werden konnten. Ein Beispiel für dieses Fenster wird in Abbildung 7.14 gezeigt.

Die Fertigstellung der Optimierung leitet erneut eine Phase ein (Reduktionsphase), in der dem Kombinationsfeld neue Werte zugewiesen werden und auch die optimierten



	blaues Symbol mit weißem Zahnrad: Damit kann der im Kombinationsfeld ausgewählte Optimierungsalgorithmus aufgerufen werden.
	gelber Pfeil nach unten mit "T": Das zeigt die selektierten Triangulierungen in seperaten Fenstern an.

Abbildung 7.12: **Menüführung der Optimierungsphase.**

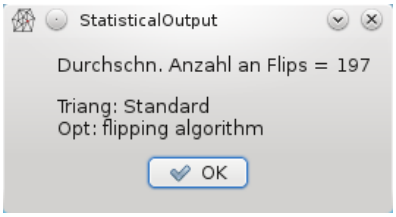


Abbildung 7.13: **Fenster mit Mittelwert über die Anzahl der Flips.**

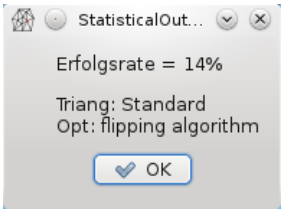
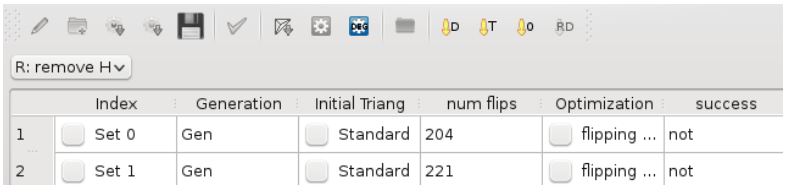


Abbildung 7.14: **Erfolgsrate der Optimierung.** Fenster mit Prozentsatz der erfolgreich optimierten Beispiele

Triangulierungen werden zur Tabelle hinzugefügt. Diese sind selektierbar um die Datenmengen anzeigen zu können. Die Menüführung ist in Abbildung 7.15 zu sehen.

Das Reduzieren leitet die letzte Phase (Auswertungsphase) ein. Hier wird ebenso ein Auswertungsfenster geöffnet, das den Mittelwert über die letzte Spalte anzeigt (siehe Abbildung 7.16).



	blaues Symbol mit weißem Zahnrad und “DEG”: Das ruft den im Kombinationsfeld augewählten Reduktionsalgorithmus auf.
	Gelber Pfeil nach unten mit “O”: Das zeigt die selektierten optimierten Triangulierungen in seperaten Fenstern an.

Abbildung 7.15: **Menüführung der Reduktionsphase.**

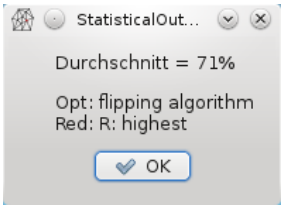


Abbildung 7.16: **Erfolgsrate der Reduktionsmethode.** Fenster mit der Erfolgsrate der Reduktionsmethode

Auch die reduzierten Triangulierungen können selektiert werden und durch das letzte Symbol (gelber Pfeil nach unten mit “RD”) in einem seperaten Fenster angezeigt werden. Es besteht nun die Möglichkeit, durch Selektionen in den einzelnen Spalten jede Phase einer Punktmenge und deren Reduktion einzusehen. In der “num flips“ Spalte des Hauptfensters wird die Anzahl der Optimierungsschritte des jeweiligen Beispiels ausgegeben. Die ”success” Spalte des Hauptfensters gibt an, ob der Durchlauf für dieses Beispiel bereits nach der Optimierung erfolgreich war.

Die Heuristik des Programms wird in den letzten beiden Spalten des Hauptfensters abgelesen. Die “num ops“ Spalte enthält die Anzahl der Reduktionsschritte, wogegen die ”relops” Spalte die prozentuelle Veränderung der Punktmenge angibt.

In der Auswertungsphase kann der Benutzer die einzelnen Datenbeispiele in den verschiedenen Phasen miteinander vergleichen.

Speichern und Laden

Wenn der Benutzer diesen Button drückt, so werden die Testdatensätze gespeichert. Nachdem der Benutzer definiert hat, wo gespeichert werden soll, wird folgendermassen gespeichert:

- *In der ersten Zeile wird die Anzahl der Punkte pro Datensatz (numpoints) gespei-*

chert

- *In der zweiten Zeile wird die Anzahl der Datensätze (numsets) geschrieben*
- *In der dritten Zeile wird die Kennzahl der Art des folgenden Testsets gespeichert*
- *Ab Zeile 4 werden die Punkte gespeichert (x-Koordinate, y-Koordinate)*
- *Sobald ein Datensatz fertig ist, fängt der nächste wieder mit der Kennzahl an.*

Zum Laden wird ein Navigationsfenster für den Benutzer zur Auswahl der Datei geöffnet. Hier ist nur wichtig, dass diese Datei von diesem Programm erstellt wurde. Es werden nun nacheinander alle Testsätze mit den dazugehörigen Punkten ausgelesen um sie weiterverwenden zu können.

7.8 Anhang, Anmerkungen und Referenzen zur Software

buildscript

Das buildscript für dieses Programm wurde mit Hilfe von Herrn Markus Plieschnegger und Florian Rieger sehr umfangreich gestaltet. Es soll sicherstellen, dass ausgehend von Debian (Lenny) immer eine Umgebung geschaffen werden kann, in der die Software läuft. Das Problem ist oftmals, dass die Software lokal funktionieren muss, ohne dass der Benutzer Administratorrechte, beziehungsweise das Knowhow hat, um die Voraussetzungen zu schaffen. Das buildscript liegt im Ordner prrmake/ und besteht aus mehreren Teilen:

- *File: config.cmake*
- *Ordner: prrmake*
- *Ordner: deps*

Im Ordner prrmake befinden sich die drei Dateien *functionality.cmake*, *defines.cmake* und *prrmake.cmake*. *defines.cmake* prrmake Definitionsfile *functionality.cmake* prrmake Implementationsfile *prrmake.cmake* prrmake Hauptfile

Im Ordner deps liegen die Abhängigkeitsdefinitionsfiles *cgal.cmake* und *qt.cmake*

Das Makefile ist so ausgelegt, dass es bei der Ausführung von *make* automatisch nach den benötigten Dependencies sucht. Wenn diese gefunden werden, so erkennt dieses Makefile, dass kompiliert werden kann. Findet es jedoch eine der Dependencies nicht, so werden sie installiert. Hier muss allerdings angemerkt werden, dass das Makefile nur in den lokalen Ordnern sucht, wenn nicht die slim Version verwendet wird.

CGAL

CGAL ist ein Open Source Projekt (siehe CGAL unter der Seite [7]), das es sich zum Ziel gemacht hat, gut verwendbare, effizient programmierte und verlässliche geometrische Algorithmen in Form einer C++ Bibliothek zur Verfügung zu stellen. Diese Bibliothek

wird in den verschiedensten Bereichen, die geometrische Algorithmen benötigen, verwendet. Beispiele dafür sind die Bereiche Computergrafik, wissenschaftliche Visualisierungen, computergestützte Modellierungs- und Designaufgaben, Geographische Informationssysteme, Molekularbiologie, medizinische Bildbearbeitung, Netzgenerierungen, Numerische Methoden und viele andere. Für weitere Informationen und Projekte, die CGAL verwenden wird auf die CGAL Homepage [7] und das CGAL Manual unter [12] verwiesen.

QT

QT ist eine sehr bekannte Software (zum Download unter [9] und zur Programmierungsanleitung unter [10]), die umfassende UI Funktionen für sehr viele Plattformen anbietet. QT gehört mittlerweile bei vielen Linux Distributionen zu einem der Pflichtpakete (der Window Manager KDE (siehe [8])verwendet zum Beispiel QT). Andere Anwendungen sind Oberflächen für Mobiltelefone, Webapplikationen, Datenbankfrontends und ähnliches.

CORE

Diese Bibliothek (CORE unter [11])ist bereits im Lieferumfang von CGAL ([7]) eingebaut und bietet Datentypen, mit denen man sehr genau rechnen kann. Andere Bibliotheken haben sehr oft den Nachteil, dass die genauen Datentypen gewisse Operationen wie die Quadratwurzel nicht genau rechnen können. Das wurde in dieser Bibliothek gelöst. Der Nachteil ist, dass die Benützung dieser Datentypen die Laufzeiten extrem verschlechtern. Die Datentypen von CORE kommen in der Software zur Anwendung, weil die Rechengenauigkeit sehr oft Probleme bereitet hat.

Buttons

Die Buttons, die für die grafische Oberfläche gebraucht wurden, mussten aus Mangel an Zeichen und Designtalent von anderen Quellen "geborgt" werden. Sie wurden größtenteils vom KDE Projekt (siehe Standardicons aus [8], das KDE Projekt, lizenziert unter LGPL 3) entnommen und enthalten höchstens einige kleine Abänderungen um die Funktionalität anzuzeigen.

8 Auswertung

In diesem Kapitel werden die mit Hilfe der Software erstellten Simulationen ausgewertet. Dabei wird sowohl auf die Preformance als auch auf die Effizienz der einzelnen Algorithmen eingegangen. Das Kapitel wird in sieben Bereiche unterteilt.

- Analyse der Punktmengengenerierung (unter Abschnitt 8.1)
- Laufzeitverhalten der verschiedenen Methoden (unter Abschnitt 8.2)
- Auswirkungen der Auswahl verschiedener Wunschgrade (unter Abschnitt 8.3)
- Analyse der Triangulierungsarten (unter Abschnitt 8.4)
- Performace der Optimierungsalgorithmen (unter Abschnitt 8.5)
- Vergleich verschiedener aggressiver Reduktionsmethoden (unter Abschnitt 8.6)
- Vergleich verschiedener Ansätze von Reduktionsmethoden (unter Abschnitt 8.7)

Für die Testläufe wurden zwei verschiedene Punktmengengenerierungsmethoden verwendet. Diese Datensätze wurden gespeichert und sind zu finden unter:

```
(1) StatisticalOutput/files/data/gentestset.txt
(2) StatisticalOutput/files/data/randtestset.txt
(3) StatisticalOutput/files/data/test1000/gentestset.txt
(4) StatisticalOutput/files/data/test1000/randtestset.txt
```

Unter (1) ist ein Testlauf zu finden, der aus 100 generierten Datensätzen mit je 100 Punkten besteht. Unter (2) findet man den gleichen Testlauf mit randomisiert erzeugten Punktmengen. (3) und (4) enthalten jeweils einen Testlauf mit 1000 Punktmengen zu je 100 Punkten, wobei die Punktmengen in (3) generiert und in (4) randomisiert wurden.

8.1 Analyse der Punktmengengenerierung

In diesem Bereich wird analysiert, warum eine abgeschwächte Version der Punktmengenerzeugung gewählt wurde und warum diese trotzdem für die Erfordernisse dieser Arbeit ausreichend funktioniert hat. Die Punktmengenerzeugung funktioniert, wie in Kapitel 7 zu sehen war, nicht genau nach dem Schema, das in Kapitel 3 vorgestellt wurde. Die implementierte Punktmengenerzeugung hat lediglich eine relativ hohe Wahrscheinlichkeit Punktmengen mit konkaven Fächern zu erzeugen. Dazu wurden zwei Testläufe erstellt,

der mit generierten Datensets arbeitet. Der erste besteht aus 100 generierten Punktmengen zu je 100 Punkten und der zweite besteht aus 1000 generierten Punktmengen mit je 100 Punkten.

Für die Triangulierung wurde eine randomisierte Standardtriangulierung gewählt und es wurde mit dem Flip-Optimierungsalgorithmus optimiert. Der Wunschgrad lag bei diesem Testlauf bei 10, was, wie im nächsten Abschnitt zu sehen ist, recht leicht erreicht werden kann. Diese Auswertung soll zeigen, dass selbst ein Wunschgrad, der normalerweise relativ leicht erreichbar ist, bei generierten Punktmengen schwer erreicht werden kann.

Der zweite, größere Testlauf wurde mehrfach ausgeführt und ist statistisch aussagekräftiger (Testlauf mit 1000 generierten Datensätzen zu je 100 Punkten, mehrfache Ausführung wegen der randomisierten Standardtriangulierung). Die Erfolgsrate betrug hierbei immer zwischen 17.6 und 17,9 Prozent (die gleichen Punktmengen wurden zu verschiedenen Tageszeiten ausgeführt, insgesamt acht mal).

Die Wahl ist aus mehreren Gründen auf die etwas weniger extremen Punktmengen gefallen. Zum einen, weil die Wahl der extremen Herangehensweise die Variation zwischen den Punktmengen extrem beschränkt hätte. Mit der Abschwächung der Generierungsmethode wurde zwar zugelassen, dass einige der Datensätze optimiert werden können, aber es wurden auch viel mehr der Randbereiche abgedeckt, die mit dem extremen Algorithmus nicht abgedeckt worden wären. Dies ist aber auch der zweite Grund für die Wahl der abgeschwächten Methode, denn wie zu sehen ist, wenn die einzelnen Punktmengen betrachtet werden, werden sehr viele Punkte in den toten Winkeln generiert. Diese Punkt mögen zwar keine so hohen Grade erzeugen, weil die zugehörigen Fächer etwas kleiner sind, sie erzeugen dennoch Probleme für die Optimierung und die Reduktion. Ein weiterer Grund war, dass auch bei der abgeschwächten Version noch immer sehr viele extreme konkave Fächer (Fächer mit sehr vielen Punkten und extrem hohen Graden) vorhanden sind.

8.2 Laufzeitverhalten der verschiedenen Methoden

In diesem Teil der Auswertung wird eine ungefähre Analyse der Laufzeit gegeben. Bei einigen verwendeten Algorithmen ist die Laufzeit bereits bekannt (zum Beispiel die Triangulierungsalgorithmen), wogegen bei anderen Algorithmen nur eine sehr grobe Abschätzung geliefert wird.

Punktmengen erstellen

Sollten die Punktmengen randomisiert erzeugt werden, so wird die Laufzeit auf $O(n)$ geschätzt, da jeder Punkt nur einmal betrachtet werden muss. Wird die Punktmenge dagegen generiert erzeugt, so verändert sich dieses Laufzeitverhalten nicht sehr stark. Das liegt daran, dass auch hier die Punkte jeweils nur einmal betrachtet werden. Das Erzeugen der Zonen und des Kreissegments, sowie das Platzieren der externen Punkte funktioniert in $O(n)$ Zeit. Daher ist bei diesem Algorithmus nur die Konstante schlechter

als beim randomisierten Erzeugen der Punktmengen. Der Unterschied in der Laufzeit ist auch bei der Ausführung des Programms kaum zu merken.

Kollinearität prüfen

Da die Punktmengen keine Kollinearitäten aufweisen sollen, werden sie nach der Erstellung noch einmal auf Kollinearität geprüft und Punkte, die kollinear zu anderen bereits erzeugen Punkten sind, werden um ein Epsilon verschoben. Die Kollinearität abzuprüfen ist jedoch ein aufwändiger Algorithmus, denn dieser Algorithmus muss jedes mögliche Punktetripel prüfen. Er läuft in $O(n^3)$ Zeit (es gibt jedoch effizientere Implementierungen, die in $O(n^2)$ Zeit laufen) und ist für Verzögerung bei der Erzeugung der Punktmengen verantwortlich.

Triangulierungen erstellen

Drei der Triangulierungen (Delaunay, Standard und randomizied Standard) werden zum größten Teil von CGAL selbst erzeugt. Wie man im CGAL Manual [12] nachlesen kann, funktionieren diese Algorithmen in $O(n \log(n))$. Die Randomisierung beim Algorithmus für randomisierte Standardtriangulierungen kann in linearer Zeit gemacht werden.

Der vierte Triangulierungsalgorithmus, Greedy, ist dagegen ein sehr aufwändiger Algorithmus. Um alle Kanten zu erstellen, muss man jedes mögliche Punktpaar erzeugen. Davon gibt es $\Theta(n^2)$ Paare. Diese Kantenmenge muss sortiert werden, um immer die kleinste selektieren zu können. Das funktioniert in $O(n^2 \log(n))$ Zeit. Daraufhin wird immer die kürzeste Kante aus dem Container selektiert, in die Triangulierung eingefügt und die Kanten, die diese Kante schneiden werden aus dem Container entfernt. Das läuft insgesamt schneller als das Sortieren der Kanten.

Wiedereinfüge-Optimierungsalgorithmus

Dieser Algorithmus zählt zu einem der schnellsten Algorithmen die in dieser Software benutzt wird. Es ist lediglich in $O(n)$ Zeit der Punkt mit dem höchsten Grad zu selektieren. Dieser wird von CGAL aus der Triangulierung entfernt. Da nicht bekannt ist, wie lange CGAL für diese Operation braucht, wird die Oberschranke (ein Neuaufbau der Triangulierung) als Referenz herangezogen. Da die Triangulierung nach der Triangulierungsphase als beliebige Standardtriangulierung gespeichert wird, wird hier mit $O(n \log(n))$ Zeit gerechnet. Für das neuerliche Einfügen des Punktes verwendet CGAL den Lokalisierungsalgorithmus, der laut CGAL Manual [12] logarithmischen Rechenaufwand benötigt.

Flip-Optimierungsalgorithmus

Dieser Algorithmus ist schwer abschätzbar, denn die Anzahl der möglichen Flips ist kaum vorherzusehen, da diese sehr stark von der maximalen Anzahl der Flips abhängt, die zwischen zwei beliebigen Triangulierungen aus der Menge aller Triangulierungen möglich sind. Zum Laufzeitverhalten dieser Methode kann nur eine Abschätzung in Abhängigkeit der benötigten Flips gegeben werden. Die Auswahl der möglichen Flips ist mit der Anzahl

der Kanten m, die in der Triangulierung vorhanden sind, abzuschätzen. Die Anzahl der Kanten m ist $O(n)$. Die Klassifikation der Kante kann in konstanter Zeit gemacht werden. Die Wahl des besten Flips ist linear in der Anzahl der Kanten m. Ein Flip kann in konstanter Zeit durchgeführt werden. Nach einem Flip müssen alle Kanten neu erstellt und klassifiziert werden, weil die Container voneinander unabhängig sind. Ansonsten müsste jeder Container durchlaufen werden und auf seine Gültigkeit überprüft werden, was sogar langsamer als ein Neuaufbau sein würde. Dh. es wird wieder $O(m)$ Zeit gebraucht, insgesamt muss mit $O(m^2)$ abgeschätzt werden. Damit läuft dieser Algorithmus in $O(n^2)$ Zeit. Diese Grenze ist deshalb vorhanden, weil nicht nur alle momentan möglichen Flips gemacht werden können, sondern alle Flips aller möglichen Kanten.

Punktentfernende Algorithmen

Die Laufzeit der punktentferndenden Algorithmen hängt von der Anzahl der entfernten Punkten ab. Wenn berücksichtigt wird, dass CGAL die Triangulierung aktuell halten muss, so kann insgesamt mit $O(kn\log(n))$ abgeschätzt werden, wobei k kleiner als n sein muss und die Anzahl der entfernten Punkte angibt. Wenn also ohne Optimierung gearbeitet wird, so beträgt die Laufzeit $O(n^2\log(n))$. Wird nach jedem Schritt optimiert, so dominiert die Laufzeit, die für das Optimieren verwendet wird. Dann kann mit k mal der Laufzeit der gewählten Optimeriungsmethode gerechnet werden.

Punkthinzufügende Algorithmen

Das Selektieren der Dreiecke funktioniert in $O(d)$ Zeit, wobei d dem höchsten Grad entspricht. Es werden in dieser Software höchstens $n/4$ Punkte eingefügt. Das Einfügen eines Punktes funktioniert in logarithmischer Zeit. Wenn kein Optimierungsalgorithmus verwendet wird, so beträgt die Laufzeit $O(n\log n)$. Genau wie bei den punktentfernenden Algorithmen ist auch hier die Optimierung der teuerste Schritt. Es kann mit $n/4$ mal der Laufzeit der gewählten Optimeriungsmethode gerechnet werden.

8.3 Auswirkungen der Auswahl verschiedener Wunschgrade

Die Auswahl des Wunschgrades hat einen sehr starken Einfluss auf den Erfolg der Reduktion des Grades. Gewisse Wunschgrade können nicht erreicht werden, weil sie zu klein sind, wogegen hohe Grade sehr schnell erreicht werden können. In diesem Teil der Auswertung wird die Wahl verschiedener Grade durch einen Test mit randomisierten Punktmengen und der Delaunay-Triangulierung erläutert. Hierzu wurde ein Testdatensatz mit 1000 randomisierten Punktmengen mit je 100 Punkten verwendet. Getestet wurden mehrere Grade, deren Ergebnisse in Tabelle 8.1 einzusehen sind.

Wie dieser Tabelle zu entnehmen ist, ist bei Grad kleiner als 7 kein Erfolg erzielbar. Daher macht es keinen Sinn auf diese Grade reduzieren zu wollen. Ab Grad 7 kann festgestellt werden, dass sehr viele randomisiert erzeugte Punktmengen durch den Flip-Optimierungsalgorithmus auf den gewünschten Grad reduziert werden kann.

Grad	Erfolgsrate	Mittelwert	Std.abw.
4	0%	59	88,67
5	0%	51	88,52
6	0%	51	88,51
7	98,7%	51	88,51
8	100%	51	88,52
9	100%	51	88,51
10	100%	51	88,51
12	100%	51	90,03
15	100%	51	90,03

Tabelle 8.1: **Erfolgsrate und Anzahl der Flips bei verschiedenen Wunschgraden einer Delaunay-Triangulierung.**

Bei hohen Graden ist bei randomisierten Punktmengen immer ein Erfolg zu sehen. Um den Optimierungsalgorithmus bei hohen Wunschgraden versagen zu lassen, müssen generierte Punktmengen herangezogen werden. Es ist zu sehen, dass die Höhe des Wunschgrades die Anzahl der Flips nicht beeinflusst. Die große Standardabweichung ergibt sich aufgrund einiger Outlier, wie in Abbildung 8.1 zu sehen. In diesen Fällen mussten alle möglichen Flips ausgeführt werden, weil die Delaunay-Triangulierung und die gradoptimale Triangulierung sehr weit voneinander entfernt waren.

8.4 Analyse der Triangulierungsarten

Die Triangulierungsart hat klarerweise einen Einfluss auf die Grade, wie im Laufe dieser Arbeit bereits festgestellt wurde. In diesem Abschnitt sollen die verschiedenen Triangulierungsarten und deren Grade miteinander verglichen werden. Das wurde mit einem Testlauf bei einem fixen Grad, randomisierten Datenmengen und mit dem Flip-Optimierungsalgorithmus gemacht. Die Wahl des Grades hat, wie in Tabelle 8.1 zu sehen ist, keinen Einfluss auf die Anzahl der Flips. Daher wurde im nächsten Testlauf der Grad auf 10 festgelegt. Die Anzahl der Flips gibt Auskunft über die Nähe einer Triangulierung zu einer Triangulierung, deren Grad gut ist. In Tabelle 8.2 wird gezeigt, welche der getesteten Triangulierungsarten wie viele Flips im Mittel benötigt wurden, um mit dem Optimierungsalgorithmus eine Triangulierung nierigem Grades zu erreichen. In Abbildung 8.1 wurde ein Histrogramm über die Anzahl der Flips eines Testlaufs mit einer Delaunay-Triangulierung gezeigt. Das gleiche Histogramm mit einer Greedy-Triangulierung sieht diesem Histogramm sehr ähnlich und der Mittelwert lag hier nur ein wenig höher bei 59 mit einer Standardabweichung von 91,73. Die Testläufe mit den Standardtriangulierungen waren wieder ähnlich. Das Histogramm hierzu ist in Abbildung 8.2 zu finden.

Dieser Tabelle ist zu entnehmen, dass die Delaunay-Triangulierung die besten Ergebnisse erzielt, wogegen die Standardtriangulierungen die schlechteste Preformance besitzen. Bei der randomisierten Standardtriangulierung kann man Glück haben und es werden

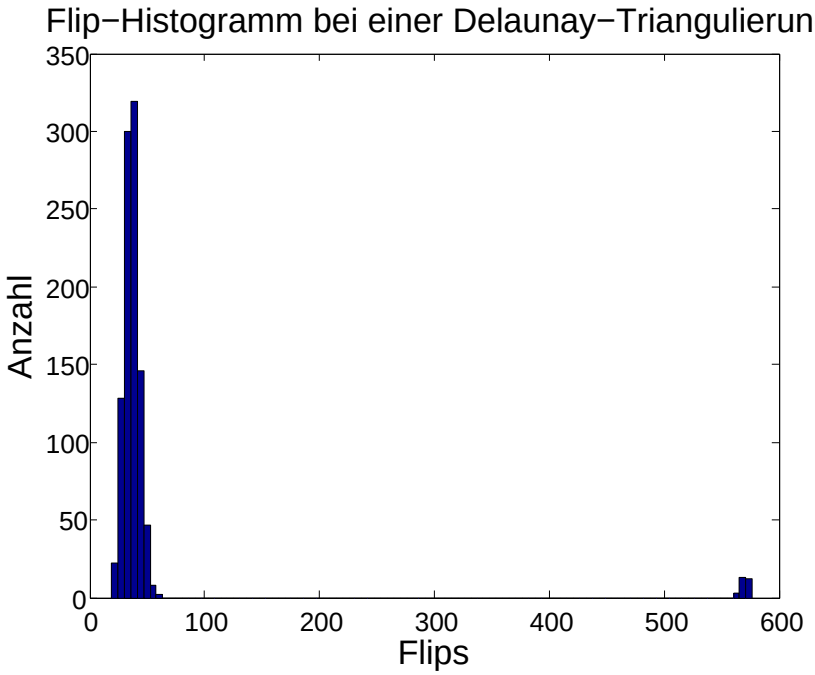


Abbildung 8.1: **Beispiel für ein Histogramm über die Anzahl der Flips.** Dieses Histogramm veranschaulicht die Verteilung der Anzahl der Flips eines Testlaufs mit 1000 randomisierten Punktmengen mit je 100 Punkten. Optimiert wurden hier Delaunay-Triangulierungen. Der Mittelwert liegt bei 51 Flips mit einer Standardabweichung von 88,51. Die große Standardabweichung ergibt sich durch Outlier, bei denen alle möglichen Flips ausgeführt werden mussten, weil bei diesen Beispielen die Delaunay-Triangulierung und die gradoptimale Triangulierung sehr weit voneinander entfernt waren.

sehr wenige Flips benötigt. Normalerweise verhält sie sich jedoch nur etwas besser als die normale Standardtriangulierung.

Die Greedy-Triangulierung hat relativ wenige Flips, ist aber im Vergleich zur, um sehr viel effizienter berechenbaren, randomisierten Standardtriangulierung, nicht gut genug.

8.5 Performace der Optimierungsalgorithmen

In diesem Teil wird die Erfolgsrate zwischen den beiden Optimierungsalgorithmen verglichen. Es wird getestet, ob der Wiedereinfügealgorithmus, dessen Laufzeit deutlich unter der der Flipmethode liegt, gleich gute Ergebnisse liefert. Vorerst wird die Preformance der beiden Algorithmen auf randomisierten Punktmengen verglichen (erneut Grad 10). Dazu wird die Anzahl der erfolgreich reduzierten Datenbeispiele im Vergleich zur Gesamtanzahl herangezogen.

Triangulierung	Mittelwert der Flips	Standardabw.
`Standardtriangulierung`	159	57,2
`Delaunay-Triangulierung`	51	88,5
`Greedy-Triangulierung`	59	91,7
`rand. Standardtriangulierung`	159	59,9

Tabelle 8.2: **Die Anzahl der Flips in Abhängigkeit der Triangulierungsart.**

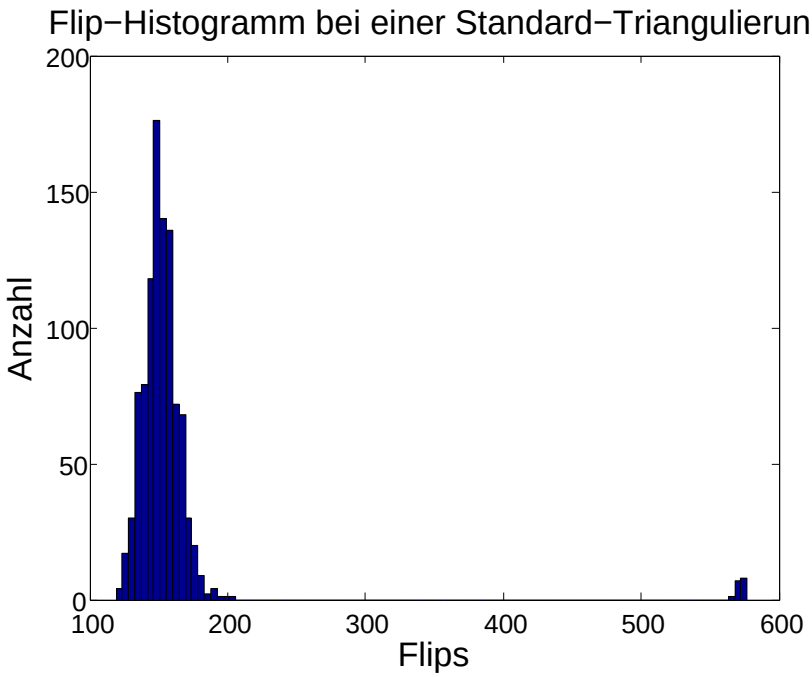


Abbildung 8.2: **Ein weiteres Beispiel für ein Histogramm über die Anzahl der Flips.** Dieses Histogramm veranschaulicht die Verteilung der Anzahl der Flips eines Testlaufs mit 1000 randomisierten Punktmengen mit je 100 Punkten einer Optimierung einer Standardtriangulierung. Der Mittelwert liegt bei 159 Flips mit einer Standardabweichung von 57,16. Es ist zu sehen, dass die Verteilung wesentlich breiter ist als in Abbildung 8.1.

Randomisierter Testlauf

Beim Test mit randomisierten Datenmengen lieferten beide Algorithmen einen Erfolg von 100 Prozent. Das bedeutet, dass die Algorithmen bei randomisierten Testsätzen gleich gut funktionieren, wobei der Wiedereinfügealgorithmus sehr schnell war und der Flipalgorithmus langsam war.

Generierter Testlauf

Bei diesem Test war eindeutig die Flipmethode besser, denn diese Methode schaffte eine um vieles höhere Erfolgsrate als der Wiedereinfügealgorithmus. Der Wiedereinfügealgorithmus versagt recht schnell, sollten die Punktmengen komplexer werden.

Resultat

Wie die beiden Testläufe gezeigt haben, kommt es hier sehr stark auf Struktur der Datensätze an, welcher Algorithmus empfohlen wird. Der Flipalgorithmus hat eine viel bessere Erfolgsrate als der Wiedereinfügealgorithmus, weil er eine genauere Auswahl für die Optimierung besitzt, denn er benutzt immer eine lokale Verbesserung. Trotzdem ist es nicht zu empfehlen, den Flipalgorithmus zu verwenden, wenn man weiß, dass man keine extremen Punktmengen hat. Will der Anwender mit Sicherheit reduzieren, so muss er mit dem Flipalgorithmus arbeiten. Der Unterschied der Laufzeiten der beiden Algorithmen ist hier entscheidend, denn der Wiedereinfügealgorithmus kann als wirklich schnell beschrieben werden, wogegen der Flipalgorithmus langsam ist. Eine Möglichkeit wäre das sequenzielle Ausführen der beiden Algorithmen. Durch den Wiedereinfügealgorithmus könnte voroptimiert werden und die Punktmengen, die dieser Algorithmus nicht erfolgreich reduzieren kann, werden noch einmal mit dem Flipalgorithmus optimiert. Dadurch kann sehr viel Rechenzeit eingespart werden.

8.6 Vergleich verschiedener aggressiver Reduktionsmethoden

In diesem Abschnitt werden die aggressiven Reduktionsmethoden und deren Ergebnisse miteinander verglichen. Dazu mussten sie aufgrund der Voroptimierung betrachtet werden. Die Wahl der Anfangstriangulierung fiel wieder auf die Delaunay-Triangulierung, denn wie in Kapitel 8.4 zu sehen war, benötigt diese im Schnitt die wenigsten Optimierungsschritte.

Aggressive Reduktionsmethoden ohne Optimieriung

Tabelle 8.3 zeigt die Auswertung der Algorithmen ohne Voroptimierung. Es wird die Prozentanzahl des Beispiels, also die Anzahl der im Schnitt übriggebliebenen Punkte bei den entfernenden Algorithmen, bzw. die Originalität bezüglich der ursprünglichen Punktmenge verglichen.

Dieser Tabelle (Tabelle 8.3) ist zu entnehmen, dass alle implementierten Methoden ohne Optimierungsmethoden nicht gut funktionieren. Das kann Anhand der Erfolgsrate herausgelesen werden. Diese Tabelle wurde durch einen Testlauf erstellt, der mit 1000 generierten Punktmengen mit je 100 Punkten gearbeitet hat. Die Wahl der Triangulierung fiel bei diesem Testlauf auf eine Delaunay-Triangulierung (wegen Abschnitt 8.4). Es wurde wieder versucht, den Grad auf 10 zu senken, da dies laut Abschnitt 8.3 relativ leicht erreichbar ist.

Algorithmus	MWT Originalität	Stdabw.	Erfolgsrate
`Entf. höchster Punkt`	62 %	24,18 %	16 %
`Entf. höchster Nachbar`	65 %	24,22 %	17 %
`Entf. random Nachbar`	66 %	23,84 %	17 %
`Add. in höchstem Dr.`	75 %	0 %	0 %
`Add. in random Dr.`	77 %	8,5 %	2 %

Tabelle 8.3: **Originalität der Punktmenge bezüglich der Reduktionsmethoden ohne Optimierung.** Die Originalität gibt an, wieviel Prozent der Punktmenge noch vorhanden sind (bei Entfernen von Punkten), bzw. wieviel Prozent die originale Punktmenge von der aktuellen Punktmenge ausmachen (beim Hinzufügen von Punkten). Der Testlauf bestand aus 1000 generierten Datensätzen mit je 100 Punkten.

Die punktentfernenden Algorithmen haben dabei noch relativ gut funktioniert. Das kommt daher, weil die Triangulierungen bei konkaven Fächern sehr viele unvermeidbare Kanten haben und die gradhöchsten Punkte immer bei den externen Punkten der Fächer liegen. Trotzdem wurden hier keine großen Erfolgsraten erzielt, denn die punktentferndenden Algorithmen versagen, wenn nur noch Punkte auf der konvexen Hülle vorhanden sind. Diese Punktmengen müssten jedoch nur optimiert werden, denn gerade diese Punktmengen sind relativ einfach auf vernünftige Grade reduzierbar. Die beiden Reduktionsalgorithmen, die Punkte hinzufügen, versagen hier komplett, denn sie hängen sehr stark von den Optimierungsalgorithmen ab. Sie können nur dann gute Ergebnisse liefern, wenn die bewusst durch das Einfügen von Punkten erzeugten Mehrdeutigkeiten geschickt trianguliert werden. Das kann ein Einfügealgorithmus jedoch niemals erreichen. Die Originalität der punkthinzufügenden Algorithmen ist mit 75 Prozent beschränkt, da diese Algorithmen eine Abbruchbedingung besitzen. Zwischen den punktentfernenden Algorithmen gibt es hier relativ geringe Unterschiede, wobei der Algorithmus, der beliebige Nachbarpunkte löscht, geringfügig besser abgeschnitten hat als die anderen Algorithmen.

Ein weiteres Beispiel einer Punktmenge mit einem punktentfernenden Algorithmus findet sich in Abbildung 8.3. Welcher Optimierungsalgorithmus am besten funktioniert, wird in den nächsten beiden Abschnitten erläutert.

Mit Wiedereinfügealgorithmus als Optimieriung

Hier ist die Auswertung der Algorithmen mit dem Wiedereinfügealgorithmus als Optimierung getestet worden. Beim Test hat sich sehr schnell gezeigt, dass diese Optimierungsmethode sehr schlecht für Reduktionsalgorithmen geeignet ist. Die entfernenden Algorithmen haben sehr oft sehr viele Punkte entfernt, die mit dem Flipalgorithmus erhalten geblieben sind. Der Wiedereinfügealgorithmus als Optimierungsmethode war für die hinzufügenden Algorithmen oft zu schwach war. Der Tabelle 8.4 ist zu entnehmen, dass der Wiedereinfügealgorithmus als Optimierung bei den punktentfernenden Algorithmen nicht zu empfehlen ist, denn die Erfolgsrate ist sogar geringer als ohne Optimierung. Bei den hinzufügenden Algorithmen hat diese Optimierung geringe Erfolge gebracht. Der

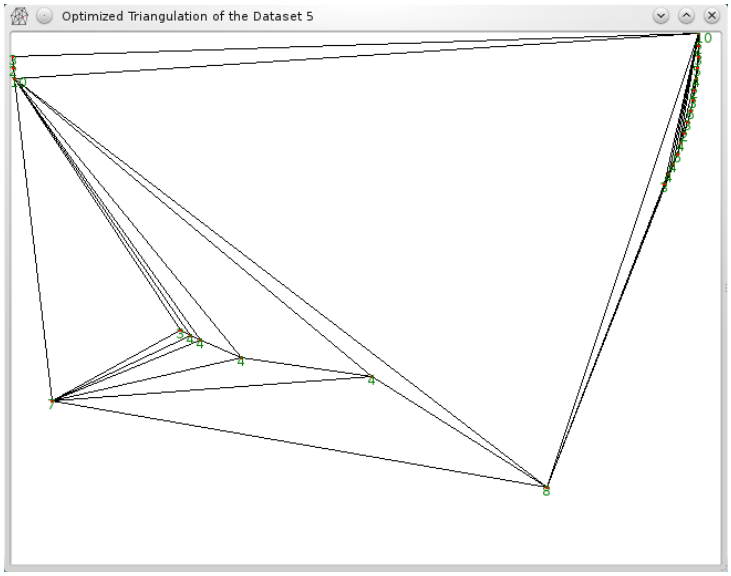


Abbildung 8.3: **Ohne Optimierung mit Entfernalgorithmus.** In diesem Bild ist ersichtlich, dass von einer Punktmenge nicht sehr viel übrigbleibt, wenn ohne Optimierung gearbeitet wird. Die Punktkette entlang des Kreissegments wurde fast vollständig aufgelöst, sodass die Punktkette selbst schon schwer erkennbar ist.

Testlauf, der zu dieser Tabelle geführt hat, hat die gleichen Testsätze verwendet, wie der Testlauf ohne Optimierung. In Abbildung 8.4 ist ein Histogramm über die Erfolgrate einer Methode, die Punkte entfernt hat, zu sehen (in diesem Fall entfernen des Nachbarpunktes mit dem höchsten Grad).

Abbildung 8.5, Abbildung 8.6 und Abbildung 8.7 verdeutlichen den Ausgang dieses Testlaufs.

Mit Flipalgorithmus als Optimieriung

Im letzten Teil der Auswertung werden die Reduktionsalgorithmen mit dem Flipalgorithmus als Optimierungsmethode getestet. Dieser Testlauf hatte die gleichen Parameter, wie die vorangegangenen Testläufe, mit dem Unterschied, dass als Optimierungsmethode der Flipalgorithmus gewählt wurde. Es wurde also ein Testlauf mit 1000 generierten Punktmengen zu je 100 Punkten und einer Delaunay-Triangulierung gewählt, der mit dem Flipalgorithmus und den Reduktionsalgorithmen den Grad auf 10 senken soll, verwendet. Unter Tabelle 8.5 ist die Originalität der Datenbeispiele zu sehen.

Punktentfernende Reduktionsalgorithmen

Im Testlauf war zu sehen, dass diese Methoden relativ gut funktioniert haben. Sie haben allerdings auch, wie in Kapitel 6 vermutet wurde, relativ viele Punkte gelöscht. Im Anschluss werden unter Abbildung 8.8, Abbildung 8.9 und Abbildung 8.10 drei Screens-

Algorithmus	MWT Originalität	Stdabw.	Erfolgsrate
`Entf. höchster Punkt`	63 %	23,05 %	11 %
`Entf. höchster Nachbar`	64 %	22,13 %	12 %
`Entf. random Nachbar`	64 %	22,65 %	11 %
`Add. in höchstem Dr.`	75 %	1,15 %	2 %
`Add. in random Dr.`	81 %	8,78 %	4 %

Tabelle 8.4: **Originalität der Punktmenge bezüglich der Reduktionsmethoden mit dem Wiedereinfügealgorithmus als Optimierungsalgorithmus.** Die Originalität gibt an, wieviel Prozent der Punktmenge noch vorhanden sind (bei Entfernen von Punkten), bzw. wieviel Prozent die originale Punktmenge von der aktuellen Punktmenge ausmachen (beim Hinzufügen von Punkten).

Algorithmus	MWT Originalität	Stdabw.	Erfolgsrate
`Entf. höchster Punkt`	73 %	22,93	100 %
`Entf. höchster Nachbar`	76 %	19,83	100 %
`Entf. random Nachbar`	77 %	19,66	100 %
`Add. in höchstem Dr.`	90 %	9,79	74 %
`Add. in random Dr.`	94 %	6,37	80 %

Tabelle 8.5: **Originalität der Punktmenge bezüglich der Reduktionsmethoden mit dem Flipalgorithmus als Optimierung.**

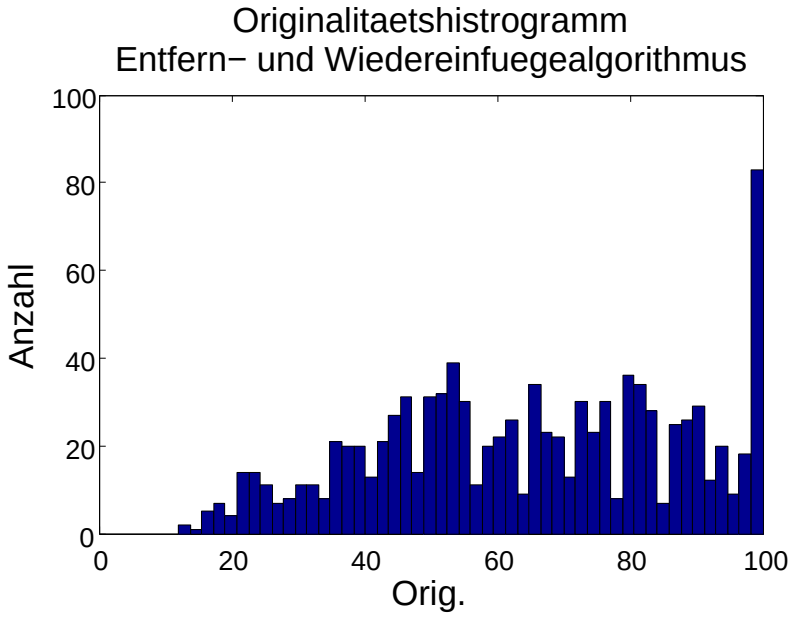


Abbildung 8.4: **Histogramm über die Originalität einer Reduktion mit dem Wiedereinfügealgorithmus.** Dieses Histogramm veranschaulicht die Originalität der einzelnen Punktmengen in einem Histogramm. Für die Erstellung wurde ein Testlauf benutzt, der mit 1000 generierten Punktmengen mit je 100 Punkten gearbeitet hat. Ausgegangen wurde von einer Delaunay-Triangulierung und es wurde mit dem Wiedereinfügealgorithmus optimiert. Der Mittelwert liegt bei 64 Prozent mit einer Standardabweichung von 22.65. Es ist zu beachten, dass die Erfolgsrate bei diesem Testlauf lediglich bei 11 Prozent lag.

hots gezeigt, die jeweils die gleiche Punktmenge zeigen, die aber mit den verschiedenen Methoden reduziert wurden.

Man erkennt anhand der Tabelle 8.5, dass die Methode mit randomisierter Auswahl des Punktes am besten funktioniert hat. Sie hat sowohl den besten Mittelwert der Originalität als auch die kleinste Standardabweichung der drei punktentfernenden Algorithmen. Es ist ebenso zu sehen, dass diese drei Algorithmen mit dieser Optimierungsmethode immer in der Lage war, die Punktmenge auf einen vernünftigen Grad zu senken. Ein weiterer Testlauf (hier nicht weiter erwähnt), der die gleichen Parameter verwendet hat, aber auf Grad 6 reduzieren sollte, hatte hier eine Erfolgsrate von 20 Prozent.

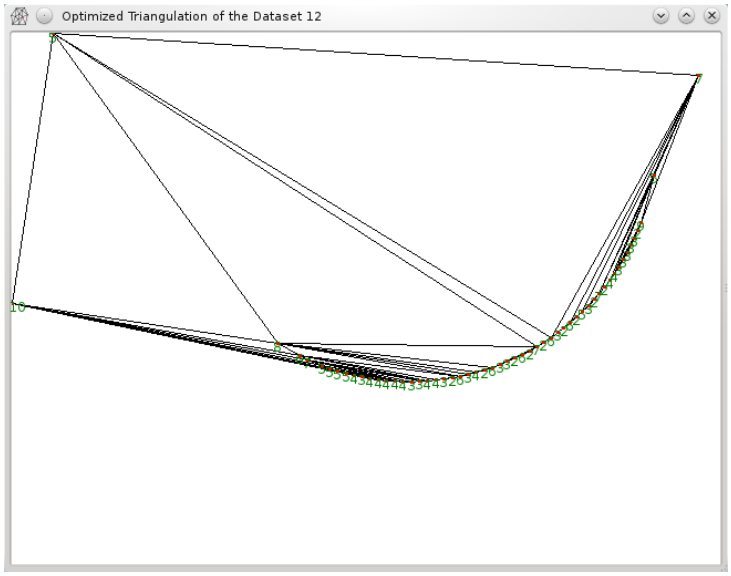


Abbildung 8.5: **Screenshot einer erfolgreich reduzierten Punktmenge mit Wiedereinfüge und Entfernalgorithmus.** Eine Punktmenge, die zwar erfolgreich, aber schlecht mit Wiedereinfüge und einem Entfernalgorithmus reduziert wurde. Es ist zu sehen, dass nicht sehr viele Punkte übrigbleiben.

Punkthinzufügende Reduktionsalgorithmen

Der Testlauf hat den Unterschied zwischen den Ansätzen in denen Punkte hinzugefügt bzw. entfernt werden verdeutlicht. Generell ist zu sehen (beim Durchsehen der Ergebnisse der Spalte "Algorithm"), dass die Datenbeispiele bei einer Auswahl der Dreiecke mit dem höchsten Grad eher Methode zwei (Punkte entlang der Mittelachse) aus Kapitel 6 und eine randomisierte Auswahl eher Methode eins (Fächerkette) aus diesem Kapitel entspricht. Das war zu erwarten, denn wenn das Dreieck mit dem höchsten Grad gewählt wird, ein Punkt eingefügt wird, dann teilt die Optimierung die Grade zwischen dem neuen Punkt und dem Punkt mit dem höchsten Grad auf. Dadurch ist das nächste Dreieck mit dem höchsten Grad wieder vorgegeben, denn es muss ein Nachbardreieck der Kante zwischen diesen beiden Punkten sein. Damit entsteht eine Art Kette vom Mittelpunkt des Kreissegments zum externen Punkt und das Verhalten von Methode zwei ist daraus entstanden.

Beim Ausführen dieser Methode sind allerdings sehr viele Probleme mit Rundungen zu sehen (werden von der Software abgefangen), denn die Punkte werden nicht äquidistant eingefügt, sondern es entsteht eine Folge, die ihren Häufungspunkt im externen Punkt besitzt.

Es ist ebenso klar ersichtlich, dass diese Methode sehr schnell versagt, sollte die Punktmenge mehr als nur einen konkaven Fächer besitzen, denn sie reduziert meist nur einen Fächer.

Der zweite Algorithmus funktioniert jedoch wider Erwarten sehr gut. Damit wird das

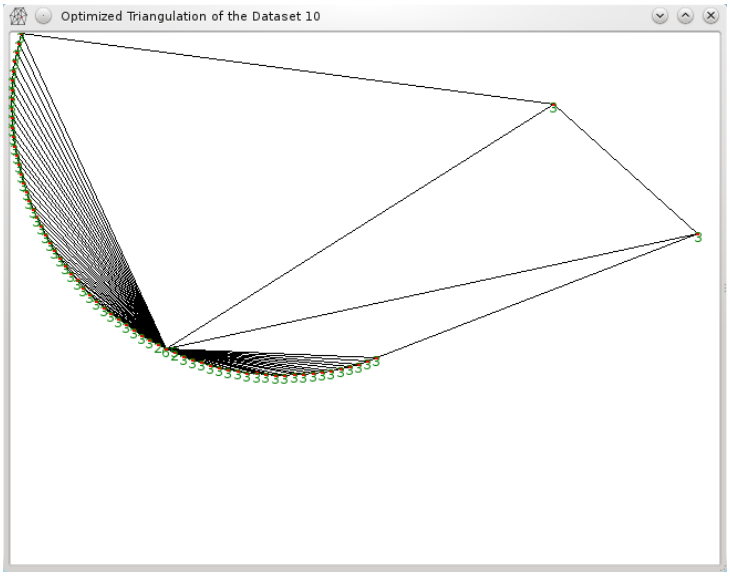


Abbildung 8.6: **Screenshot einer Punktmenge, bei der das reduzieren mit Wiedereinfüge und Entfernalgorithmus versagt hat.** Es ist zu sehen, dass hier der Optimierungsalgorithmus versagt, denn es existiert ein Punkt der keinen hohen Grad haben müsste. Dieser Fall ist sehr häufig aufgetreten. Alle Punkte im Inneren der Punktmenge wurden gelöscht.

Verhalten ähnlich dem einer Fächerkette erreicht, denn die Dreiecke, deren Eckpunkt p ist, sind sehr ähnlich. Ein Einfügen von Punkten an der selben Stelle erzeugt eine radiale Punktkette, die allerdings keine äquidistanten Punkte besitzt, außer es wurden zufällig Nachbardreiecke ausgewählt. Der einzige Unterschied zum Verhalten einer Fächerkette liegt darin, dass die Punktkette durch das Einfügen in neu erstellte Dreiecke durchbrochen wird. Es ist ersichtlich, dass dieses Verhalten dem Gesamtergebnis nicht wirklich schadet. Anhand des Histogramms über die Originalitäten dieses Testlaufs in Abbildung 8.11 ist abzulesen, dass nur relativ wenige Punkte hinzugefügt werden mussten, wenn die Methode erfolgreich war. Die Spitze bei 75 Prozent wurde durch das Versagen der Methode erzeugt. In den Abbildungen 8.12, 8.13 und 8.14 wird dieser Testlauf noch einmal verdeutlicht.

8.7 Vergleich verschiedener Ansätze von Reduktionsmethoden

Wie in Kapitel 6 zu sehen ist, ist schon die Theorie diesbezüglich relativ eindeutig. In den Testläufen hat sich die Theorie bestätigt, denn es war klar ersichtlich, dass Reduktionsmethoden, die Punkte hinzufügen um vieles effizienter arbeiten können als Methoden, die Punkte entfernen, weil sie nicht linear reduzieren, sondern den Grad durch die Anzahl der eingefügten Punkte dividieren. Es ist jedoch auch ersichtlich, dass Methoden, die Punkte löschen, öfter zu einem Ergebnis kommen, als Methoden, die Punkte hinzufügen. Das ist

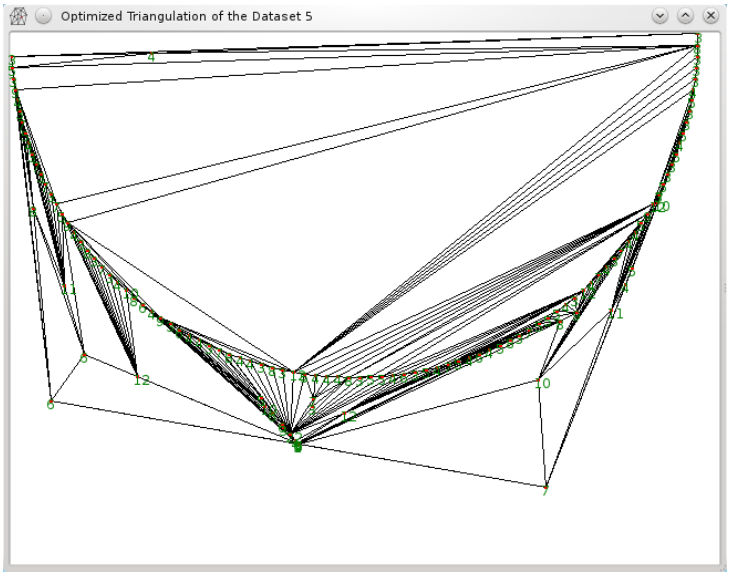


Abbildung 8.7: **Eine Punktmenge, die mit Wiedereinfügealgorithmus und einem Einfügealgorithmus reduziert wurde.** Diese Bild zeigt, wie dieser Versuch geendet hat. Es wurde nach 25 eingefügten Punkten abgebrochen. Hier ist die bessere (randomisierte Dreiecksauswahl) der beiden Methoden zum Einsatz gekommen. Es ist klar ersichtlich, dass der Wiedereinfügealgorithmus für diese Methoden viel zu schwach ist.

Anhand der Erfolgsraten in den jeweiligen Tabellen abzulesen. Es ist vorstellbar, dass ein ausgefeilter Algorithmus, der mit dem Hinzufügen von Punkten arbeitet, dieses Problem beheben könnte. Es muss vor Benutztung der Methoden abgewogen werden, welcher Fall vorliegt.

Sollte mit dichtbesetzten, radialen Punktketten, die konkave Fächer sehr hoher Ordnung erzeugen, gerechnet werden, so sollte die Wahl auf punkthinzufügende Algorithmen fallen. Sind jedoch eher mehrere kleinere konkave Fächer vorhanden, so sollte die Wahl auf punktentfernende Algorithmen fallen. Um die Erfolgsrate zu erhöhen, wäre es denkbar, Datensätze, bei denen punkthinzufügende Methoden versagen, nachträglich mit punktentfernenden Methoden zu reduzieren. Bei den punkthinzufügenden Algorithmen hat der dritte Algorithmus, der einen beliebigen Nachbarn löscht, am besten funktioniert, wie den Tabellen in diesem Kapitel zu entnehmen ist. Auch bei den punkthinzufügenden Algorithmen hat der Algorithmus, der den Schwerpunkt in einem beliebigen Nachbardreieck hinzufügt, die besseren Ergebnisse erzielt.

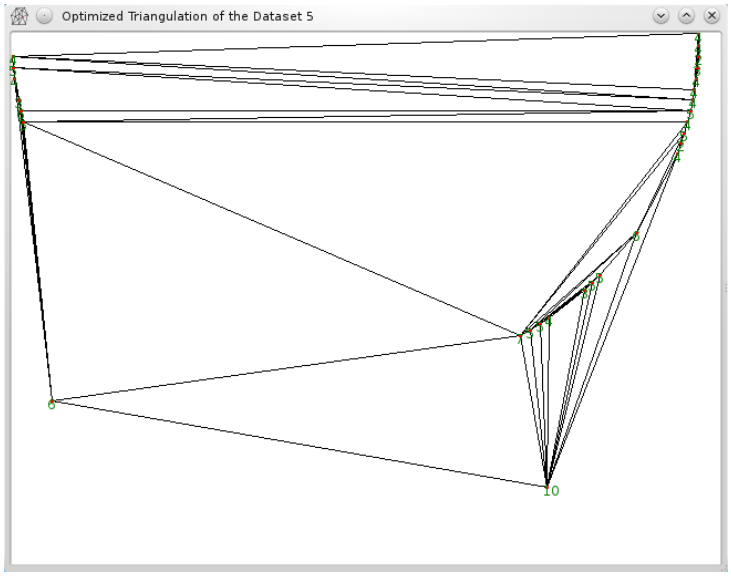


Abbildung 8.8: **Reduktion durch Löschen des gradhöchsten Punkts.** Eine Punktmenge, die durch den ersten punktentfernenden Algorithmus (siehe Tabelle 8.5) und dem Flipalgorithmus reduziert wurde.

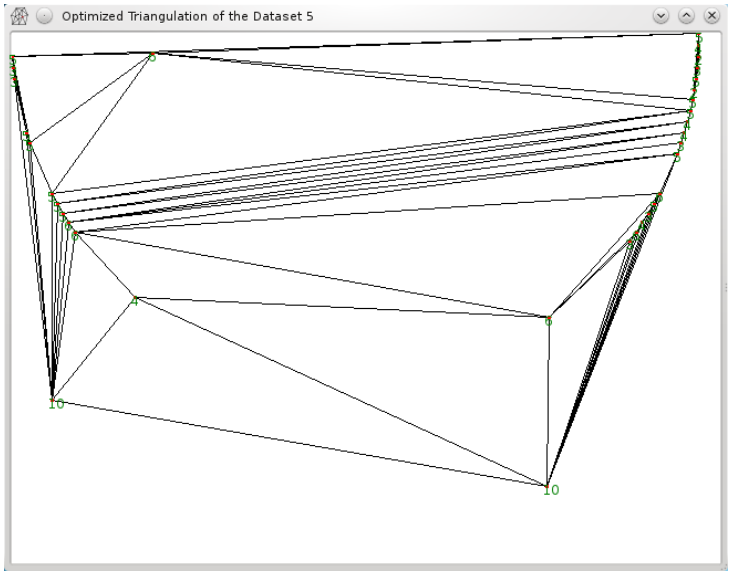


Abbildung 8.9: **Reduktion durch Löschen des gradhöchsten Nachbarpunktes.** Eine Punktmenge, die mit dem zweiten punktentfernenden Algorithmus (siehe Tabelle 8.5) und dem Flipalgorithmus reduziert wurde.

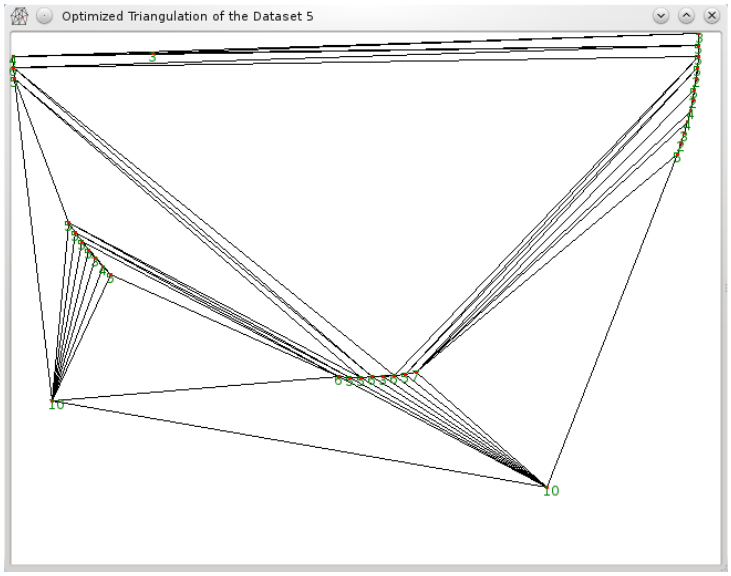


Abbildung 8.10: **Reduktion durch Entfernen eines beliebigen Nachbarn des gradhöchsten Punktes.** Eine Punktmenge, die mit dem letzten punktentfernenden Algorithmus (siehe Tabelle 8.5) und dem Flipalgorithmus reduziert wurde.

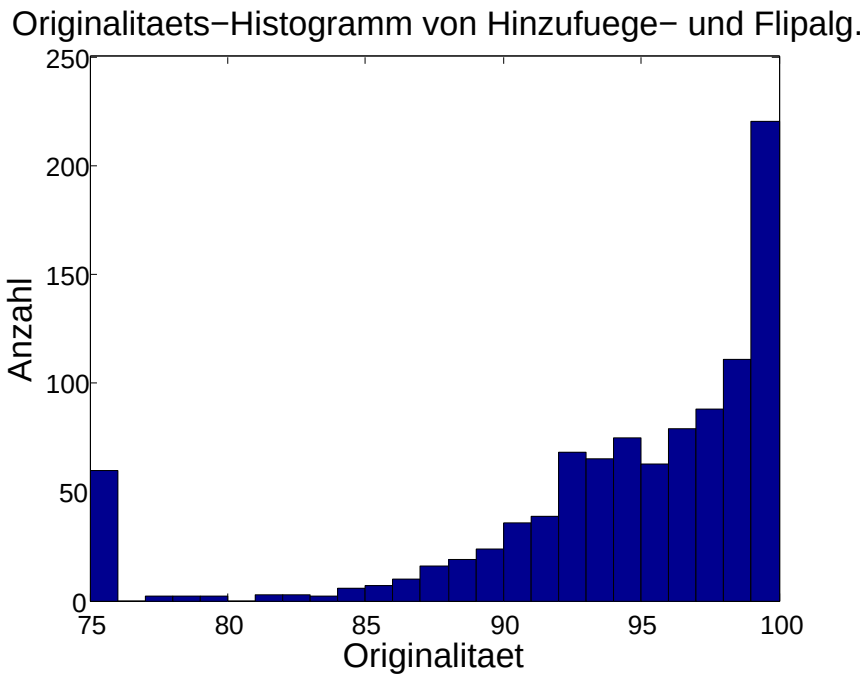


Abbildung 8.11: **Originalitätshistogramm der randomisierten, punkthinzufügenden Methode mit Flipalgorithmus.** In diesem Histogramm ist der Verlauf der Originalität der Punktmenge des Testlaufs zu entnehmen. Die Spitze bei 75 Prozent wurde durch ein versagen der Methode (Abbruchbedingung) erzeugt. Es ist zu sehen, dass die Punktmengen ansonsten mit relativ wenigen zusätzlichen Punkten reduziert werden konnten.

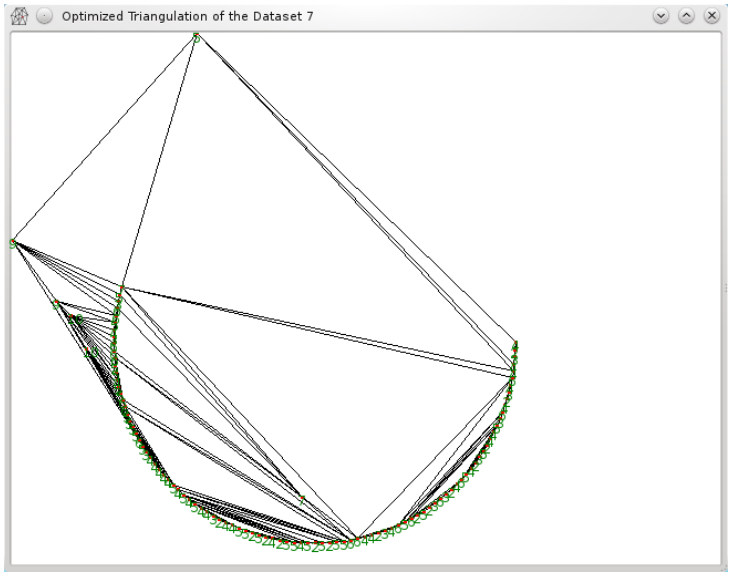


Abbildung 8.12: **Erfolg der punkthinzufügenden Methode, die das höchstgradigste Dreieck auswählt** Hier ist eine erfolgreiche Reduktion mit dem ersten Algorithmus zu beobachten. Das oben beschriebene Verhalten ist ebenfalls ersichtlich.

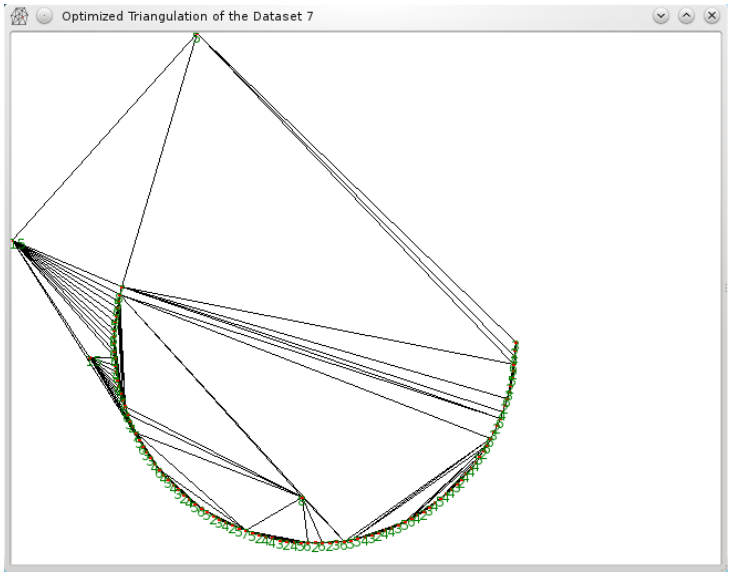


Abbildung 8.13: **Erfolg der punkthinzufügenden Methode, die ein beliebiges, benachbartes Dreieck auswählt** Dieses Bild zeigt einen Erfolg der zweiten Reduktionsmethode. Es ist zu sehen, dass sehr wenige Punkte gebraucht wurden um einen Erfolg zu erzielen.

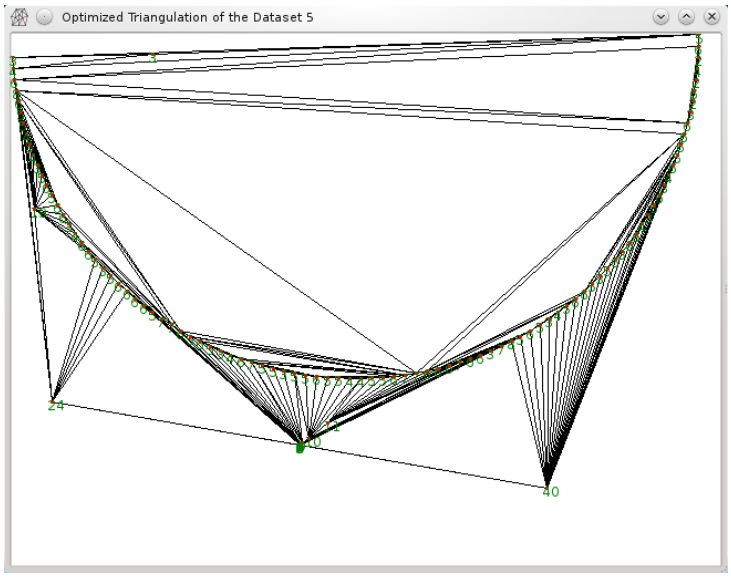


Abbildung 8.14: **Ein Versagen der punkthinzufügenden Methode, die das gradhöchste Dreieck auswählt** Hier ist klar ersichtlich, dass der erste Algorithmus zwar einen Fächer reduzieren kann, aber viel zu viele Punkte verbraucht hat, um die anderen Fächer noch reduzieren zu können. Der zweite Algorithmus konnte dieses Datenbeispiel ohne Probleme reduzieren.

9 Resümee

Die Komplexität des Gebiets der Triangulierungen mit geringen Graden ist sehr hoch. Es darf nicht vergessen werden, dass die optimale Triangulierung bezüglich des Grades derzeit als NP-schwer gilt, obwohl der Beweis noch aussteht. In dieser Arbeit wurden Näherungen an diese gradoptimale Triangulierung untersucht, die sowohl auf Algorithmen basieren, die die Punktmenge unverändert lassen als auch jene, die eine Veränderung hinnehmen.

Dabei war es nötig, den Bereich des Grades klarer zu definieren, auf welchen effizient reduziert werden kann. Im Kapitel 8.3 wurde ersichtlich, dass es nicht sinnvoll ist, wenn auf Grade reduziert werden soll, die kleiner als acht sind. Es wurde auch herausgefunden, dass die Reduktion auf höhere Grade leichter erfüllbar ist. Das bedeutet, dass jemand, der Grade senken möchte, mit einem "Trade off" leben muss.

Bei den Methoden, die eine angenäherte Triangulierung an die gradoptimale Triangulierung erzeugen sollten, ohne die darunterliegende Punktmenge zu verändern, wurden zwei wesentliche Ansätze untersucht. Die erste Methode basierte auf der Idee, Punkte mit hohen Graden zu löschen und durch einen Einfügealgorithmus an derselben Stelle wieder einzufügen. Bei der zweiten Methode wurde auf eine Hintereinanderausführung von gradverbessernden Flips zurückgegriffen. Der Preformancevergleich der beiden Methoden ergab klar, dass die zweite Methode von der Effizienz her der ersten Methode vorzuziehen ist, denn sie versagt weniger oft, wogegen die erste Methode um vieles schneller arbeitet. Es war klar zu sehen, dass beide Methoden auf gleichverteilten Punktmengen gleich effizient funktionieren. Der erste Algorithmus versagte jedoch relativ schnell, sobald die darunterliegenden Punktmengen komplexer wurden.

Es wurde auch die Verträglichkeit von verschiedenen Merkmalen, die Triangulierungen erfüllen können, untersucht. Es wurde ersichtlich, dass das hier zu erfüllende Merkmal eines geringen Grades sehr häufig nicht mit anderen bekannten Merkmalen zusammenpasst. Das kam in Kapitel 8.4 zum Vorschein, als die implementierte Flipoptimierungsmethode bei der Delaunay-Triangulierung wesentlich mehr Schritte benötigt hat, als bei einer Greedy-Triangulierung oder einem simplen randomisierten Triangulierungsalgorithmus, der die Triangulierung durch iteratives Einfügen der Punkte erzeugt.

Beide Optimierungsansätze versagten, wenn die Punktmenge bestimmte Strukturen aufwies und dadurch sehr viele unvermeidbare Triangulierungskanten erzeugte. Darum mussten sowohl aggressivere Methoden, die die Punktmengen selbst verändern, gesucht werden um den Grad zu reduzieren. Auch die Situation selbst, die Strukturen mit derart hohen Graden erzeugt, musste eingehend betrachtet werden.

Auch hierfür gab es zwei Ansätze. Der erste Ansatz versuchte den Grad der Triangulierung zu senken indem er gezielt Punkte aus der Punktmenge darunter löscht. Beim zweiten Ansatz wurden dagegen Punkte an ausgewählten Stellen hinzugefügt um den

Grad zu senken. Für beide Ansätze gab es verschiedene Implementierungen, deren Unterschiede in der Selektierung der Punkte beziehungsweise in der Selektion der Position für neue Punkte, liegen.

Beide Ansätze haben sich nur als bedingt brauchbar herausgestellt. Der Ansatz, der auf das Entfernen von Punkten basiert, hat sehr oft zu viele Punkte gelöscht, wogegen sich beim anderen Weg die Auswahl der Position als sehr komplex herausstellte und daher oft versagte. Der erste Ansatz schaffte es mit zwischenoptimerenden Schritten immer den Grad der Triangulierung zu reduzieren. Für den zweiten Weg gab es zwei verschiedene Implementierungen. Die erste Implementierung war dabei eher ein Fehlschlag, wogegen die Zweite relativ gut funktioniert hat. Es ist vorstellbar, dass die Preformance dieser Algorithmen durch eine Änderungen noch deutlich verbessert werden kann.

Diese Arbeit hat sehr viele neue, bisher noch nicht untersuchte Ansätze hervorgebracht. Es wäre zum Beispiel möglich, Reduktionsalgorithmen zu entwerfen, die ganze Punktketten statt einzelner Punkte einfügen, um den Grad der Triangulierung zu senken, denn dies würde viele Probleme lösen, die durch ein nicht zusammenhängendes Einfügen der Punkte entstanden sind.

Eine weitere Idee für einen Algorithmus wäre eine Zusammensetzung beider Reduktionsansätze, also ein Algortithmus, der sowohl Punkte einfügt, als auch welche löscht. Auch das Einfügen von Punkten auf der konvexen Hülle der Punktmenge wurde in dieser Arbeit nicht untersucht. Das größte Ziel wäre allerdings ein Algorithmus, der nicht versucht eine bestehende Triangulierung zu ändern, sondern eine Triangulierung erzeugt, deren Grade von Haus aus relativ klein sind. Man kann auf jeden Fall sagen, dass dieses Thema noch nicht vollends augereizt ist.

Eine wesentliche Hilfe beim Vergleich der verschiedenen Verfahren und der Implementierung neuer Algorithmen stellte die Bibliothek CGAL dar. Weiters zeigte sich, dass das Schaffen einer grafischen Oberfläche mit Hilfe von QT zum Generieren der Punktmengen, zur Auswahl der verschiedenen selbst implementierten Algorithmen, sowie der in CGAL verfügbaren Triangulierungsverfahren unumgänglich war und aussagekräftige Darstellungen der Ergebnisse ermöglichte.

Literaturverzeichnis

[1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms.* MIT press, 3 edition, 2009.

[2] Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications.* Springer, 3rd edition, April 2008.

[3] Robert L. Scot Drysdale, Günter Rote, and Oswin Aichholzer. A simple linear time greedy triangulation algorithm for uniformly distributed points. In *Report IIG-408, Institutes for Information Processing, Technische Universit at Graz*, 1995.

[4] C. Lawson. Transforming triangulations. *Discrete Mathematics*, 3(4):365–372, 1972.

[5] Andranik Mirzaian, Cao An Wang, and Yin feng Xu. On stable line segments in triangulations.

[6] Wolfgang Mulzer and Günter Rote. Minimum-weight triangulation is NP-hard. *J. ACM*, 55:11:1–11:29, May 2008.

[7] . CGAL, Computational Geometry Algorithms Library. http://www.cgal.org.

[8] . KDE. http://www.kde.org/.

[9] . QT. http://qt.nokia.com.

[10] . QT. http://doc.qt.nokia.com/4.6/index.html.

[11] . QT. http://cs.nyu.edu/exact/.

[12] The CGAL Project. *CGAL User and Reference Manual.* CGAL Editorial Board, 3.7 edition, 2010. http //www.cgal.org/Manual/3.7.

Abbildungsverzeichnis