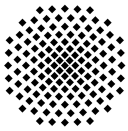


**Universität Stuttgart**

Institut für Formale Methoden der Informatik
Universität Stuttgart
Universitätsstraße 38
D - 70569 Stuttgart

Bachelorarbeit Nr. 216

# Vereinfachung von polygonalen Ebenenunterteilungen unter Topologieeinschränkungen

Maria Wiebe

| | |
|---|---|
| **Studiengang:** | Informatik |
| **Prüfer:** | Prof. Dr. Stefan Funke |
| **Betreuer:** | Dipl.-Inf. Martin Seybold, |
| | Dipl.-Inf. Filip Krumpe |
| **begonnen am:** | 15.04.2015 |
| **beendet am:** | 15.10.2015 |
| **CR-Klassifikation:** | F.2.2, G.2.2 |

# Kurzfassung

Verschiedene Geoinformationen, wie beispielsweise Straßenverläufe, Höhenlinen und Grenzverläufe, liegen häufig in großen Datenmengen vor. Zur Darstellung auf einem Bildschirm wird jedoch selten die volle Auflösung benötigt, sondern eine geringere Auflösung, die vom gewählten Zoombereich und von der Bildschirmauflösung abhängt. Daher müssen die Rohdaten vor der Übertragung und Darstellung bis zu einer gegebenen Fehlertoleranz vereinfacht werden.

In dieser Arbeit wird das Problem der Vereinfachung von polygonalen Ebenenunterteilungen untersucht. Dabei soll bei der Vereinfachung eine Fehlertoleranz eingehalten und die Topologie der Eingabe erhalten werden. Weitere Einschränkungen an die Vereinfachung können als Topologieeinschränkungspunkte gegeben sein, die nach der Vereinfachung in der topologisch selben Facette liegen müssen.

Es werden bekannte theoretische Ergebnisse sowie verschiedene Heuristiken zur Ebenenvereinfachung vorgestellt. Eine neue Heuristik, die mittels einer eingeschränkten Delaunay-Triangulierung das Problem auf viele kleine und lokale Teilprobleme reduziert, wurde im Rahmen dieser Arbeit implementiert. Zum Testen der Heuristik wurden sowohl verschiedene OpenStreetMap-Datensätze von Hamburg und von Baden-Württemberg verwendet als auch konstruierte Datensätze um die Laufzeit abzuschätzen. Anhand der ermittelten Laufzeiten für die Vereinfachung kann man von einer Laufzeit ausgehen, die superlinear jedoch nicht quadratisch ist.

# Inhaltsverzeichnis

# 1 Einleitung

Verschiedene Arten von Geoinformationen sind heutzutage unerlässlich für die Routenplanung. Für die Planung einer Radtour ist es praktisch über den genauen Straßenverlauf, Radwege, Höhenprofile, das öffentliche Verkehrsnetz und Grenzen von Tarifgebieten informiert zu sein. Will man auf solche Daten über ein Mobilgerät zugreifen, dann möchte man jedoch nicht die komplette Datenmenge abrufen, sondern maximal so viel, wie man mit der jeweiligen Bildschirmauflösung und dem gewählten Zoombereich darstellen kann. Die abgerufenen Karten sollen also vorher bis auf eine gegebene Ungenauigkeit bzw. Fehlertoleranz vereinfacht werden. Die Vereinfachung der Karten hat abgesehen von einer kleineren Datenmenge den Vorteil, dass darauf ausgeführte Methoden schneller von statten gehen, zum Beispiel das Finden einer kürzesten Strecke oder das Zeichnen und Einfärben der Karte.

Eine weitere Anforderung an die übertragenen Kartendaten ist, dass sie gewisse Topologieeinschränkungen einhalten sollen. Gehen wir davon aus, dass ein Höhenprofil in Form von Höhenlinien auf dem Mobilgerät dargestellt wird. Dann sollen sich zwei Höhenlinien auf der vereinfachten Darstellung nicht schneiden, da es sonst zu Inkonsistenzen führt, beispielsweise bei der Berechnung der steilsten Downhill-Route. Eine andere Einschränkung soll durch Punkte gegeben sein. Solch ein Punkt könnte die Position eine Tankstelle darstellen, die auf der richtigen Seite der Autobahn liegen muss.

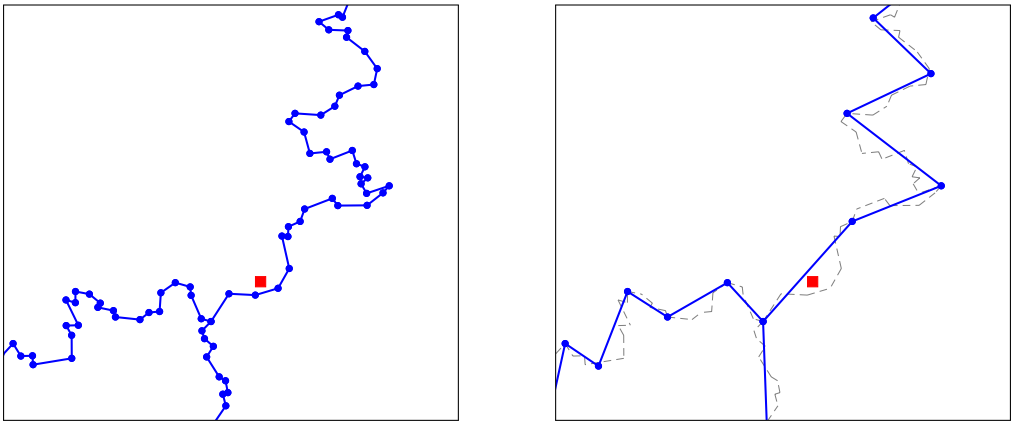


Abbildung 1.1: Beispiel einer Ebenenvereinfachung, bei der ein Einschränkungspunkt (rot) nach der Vereinfachung auf der anderen Seite des Polygonzugs liegt.

## Ziel und Aufbau der Arbeit

Ziel dieser Arbeit ist es, eine neue Heuristik für das Problem der Ebenenvereinfachung aufzustellen, zu implementieren und an verschiedenen Datensätzen zu testen.

Die Arbeit ist daher wie folgt aufgebaut. In Kapitel 2 werden die benötigten mathematischen Grundlagen eingeführt, sowie die Problembeschreibung formuliert. Außerdem beinhaltet Kapitel 2 die wichtigsten bereits bekannten theoretischen Ergebnisse zum Problem der Ebenenvereinfachung. Im nächsten Kapitel folgen bekannte (Teil-)-Algorithmen und Datenstrukturen, welche für das Verständnis der Lösungsansätze

nötig sind. Kapitel 4 stellt anschließend sowohl bekannte Heuristiken zur Ebenenvereinfachung, als auch eine neue, in dieser Bachelorarbeit untersuchte Heuristik vor. Eine Vereinfachungsdatenstruktur, die bei erneuten Anfragen zur Vereinfachung ein Ergebnis schnell zur Verfügung stellt, wird in Kapitel 5 vorgestellt. Die Vereinfachungsdatenstruktur ist in Hinsicht auf die Vereinfachung von Karten relevant und kann verschiedene Funktionalitäten realisieren, wie beispielsweise eine adaptive Vereinfachung von Karten. Im vorletzen Kapitel 6 wird die Heuristik auf verschiedenen Datensätzen getestet. Einerseits werden OpenStreetMap-Datensätze [HW08] als Eingabe verwendet um die praktische Tauglichkeit zu testen und andererseits werden konstruierte Datensätze getestet um die Laufzeitkomplexität abzuschätzen. Die erreichten Ergebnisse und mögliche Erweiterungen werden zuletzt in einem Überblick in Kapitel 7 zusammengefasst.

# 2 Grundlagen und Problembeschreibung

## 2.1 Ebenenunterteilungen und Segmente

**Definition 2.1 (Graph)** Ein *(ungerichteter) Graph* $G$ ist ein Tupel $(V, E)$ bestehend aus einer endlichen Menge $V$, genannt Vertexmenge, und einer Menge $E \subset \{\{v_1, v_2\} \,|v_1, v_2 \in V, v_1 \neq v_2\}$ genannt Kantenmenge. Ein Element $v \in V$ heißt *Vertex* oder *Knoten*, ein Element $e = \{v_1, v_2\} \in E$ heißt *Kante* zwischen den Vertices $v_1$ und $v_2$.

**Definition 2.2 (Grad)** Sei $v$ ein Vertex des Graphen $G = (V, E)$. Dann bezeichnet die Anzahl aller Kanten $e$ aus $E$, die den Vertex $v$ enthalten den *Grad* von $v$

$$\text{grad}(v) = \sum_{e \in E} |\{v\} \cap e|.$$

Zur visuellen Darstellung eines Graphen verwendet man üblicherweise eine Einbettung des Graphen in die $\mathbb{R}^2$-Ebene, welche die Vertices auf Punkte in $\mathbb{R}^2$ abbildet und die Kanten als Verbindungskurven, speziell Verbindungsgeraden, zwischen den Punkten darstellt.

**Definition 2.3 (Straight-line Einbettung)** Sei $V$ die Vertexmenge eines ungerichteten Graphen $G = (V, E)$. Eine Abbildung $\Phi : V \longmapsto \mathbb{R}^2$ heißt *Einbettung* des Graphen $G$ in $\mathbb{R}^2$. Wenn die Einbettung zusätzlich auf den Kanten $e = \{v_1, v_2\}$ definiert ist als $\Phi(e) := \{x \in \mathbb{R}^2 | x = \lambda\Phi(v_1) + (1-\lambda)\Phi(v_2),\ \lambda \in (0, 1)\}$, also als Verbindungsgerade zwischen beiden eingebetteten Vertices, dann heißt die Einbettung $\Phi$ *Straight-line Einbettung*.

**Definition 2.4 (Straight-line planar)** Ein Graph heißt *straight-line planar*, wenn es eine Straight-line Einbettung $\Phi$ gibt, so dass sich alle eingebetteten Kanten $\Phi(e)$ paarweise nicht schneiden.

Im Folgenden betrachten wir Graphen, die eine Knotenmenge $V$ besitzen, welche eine Teilmenge des $\mathbb{R}^2$ ist und straight-line planar sind mit der kanonischen Einbettung $\Phi : V \longrightarrow \mathbb{R}^2, v \longmapsto v$.

**Definition 2.5 (Ebenenunterteilung)** Eine *Ebenenunterteilung* ist das Abbild einer Straight-line Einbettung eines Straight-line-planaren Graphen. Die durch die eingebetteten Kanten begrenzten Flächen heißen *Facetten*.

**Definition 2.6 (Induzierter Teilgraph)** Sei $G = (V, E)$ ein Graph. Ein Graph $G' = (V', E')$ heißt *induzierter Teilgraph* von $G$ wenn folgende Eigenschaften gelten

- $V' \subset V$
- $E' \subset E$
- $e = \{v_1, v_2\} \in E' \iff v_1, v_2 \in V'$

Die Kantenzüge in einem Graphen, welche vereinfacht werden sollen, können als induzierte Teilgraphen der Ebenenunterteilung gesehen werden.

**Definition 2.7 (Segment)** Ein *Segment* ist ein induzierter Teilgraph $G_S = (V_S, E_S)$ eines Graphen $G$ mit den folgenden Eigenschaften.

- Das Segment besteht aus mindestens zwei Knoten, $N = |V_S| \geq 2$,
- es gibt ein Isomorphismus $\Psi$ vom Segment zum Graphen $G_E = (V_E, E_E)$ mit

$$V_E = \{1, \ldots, N\},$$
$$E_E = \{\{i, i+1\} | 1 \leq i \leq N-1\}$$

- alle nicht-Eckpunkte haben im übergeordneten Graphen $G$ den Grad 2,

$$\forall\, 2 \leq i \leq N-1 : \ \deg_G(\Psi^{-1}(i)) = 2.$$

Das Segment als *Polygonzug* im $\mathbb{R}^2$ wird bei gegebenen Graphen $G_S = (V_S, E_S)$ definiert als die Menge aller Punkte, die auf der Straight-line Einbettung des Segments liegen.

$$P_S = \{x \in \mathbb{R}^2 | \exists\, e \in E_S : \ x \in \Phi(e)\}. \tag{2.1}$$

Besteht das Segment aus genau zwei Knoten $V_S = \{v_1, v_2\}$, dann ist der Polygonzug die Strecke zwischen $v_1$ und $v_2$ und wir schreiben

$$P_S = \overline{v_1 v_2}.$$

## 2.2 Abstände und Fehlerschranken

**Definition 2.8 (Abstand zu einer Menge)** Sei ein Punkt in einem Vektorraum $x \in X$ mit Norm $\|\cdot\|$ und eine Menge $M \subseteq X$ gegeben. Dann wird der *Abstand* des Punktes $x$ zur Menge $M$ definiert als die größte untere Schranke vom Abstand zu allen Punkten $m \in M$

$$d(x, M) = \inf_{m \in M} \|x - m\|.$$

**Definition 2.9 ($\varepsilon$-Umgebung einer Menge)** Sei eine Menge $M \subseteq X$ gegeben. Die *$\varepsilon$-Umgebung* $U_\varepsilon(M)$ der Menge $M$ ist die Menge aller Punkte, die einen Abstand von maximal $\varepsilon$ zur Menge $M$ besitzen

$$U_\varepsilon(M) = \{x \in X | d(x, M) \leq \varepsilon\}.$$

Da wir in den folgenden Kapiteln Ebenenunterteilungen vergleichen wollen und insbesondere ob eine grobe Ebenenunterteilung maximal um $\varepsilon$ von einer feinen Ebenenunterteilung abweicht, werden wir die Ebenenunterteilungen als Mengen auffassen und prüfen, ob die eine Ebenenunterteilung in der $\varepsilon$-Umgebung der anderen liegt. Dabei muss betont werden, dass diese Eigenschaft nicht symmetrisch ist in dem Sinne dass für zwei beliebige Mengen $M$ und $N$ die Äquivalenz

$$M \subseteq U_\varepsilon(N) \Leftrightarrow N \subseteq U_\varepsilon(M) \tag{2.2}$$

**nicht** gilt. Zwei Gegenbeispiele sind in Abbildung 2.1 zu sehen.

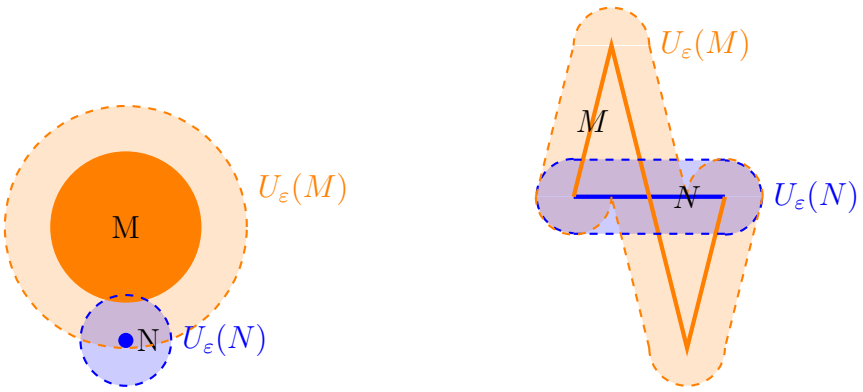



Abbildung 2.1: Zwei Gegenbeispiele zur Aussage (2.2). Es gilt in beiden Fällen $N \subseteq U_\varepsilon(M)$ und $M \nsubseteq U_\varepsilon(N)$.

**Lemma 2.10 (Schachtellemma)** Seien $L$, $M$ und $N$ Teilmengen eines normierten Vektorraums. Falls $L$ in der $\varepsilon$-Umgebung von $M$ und $M$ in der $\delta$-Umgebung von $N$ liegt

$$L \subseteq U_\varepsilon(M) \wedge M \in U_\delta(N),$$

dann gilt $L \in U_{\varepsilon+\delta}(N)$.

Die Aussage des Lemmas ist sehr intuitiv. Ein Beweis wird dennoch kurz dargestellt.

Beweis: Aus der Dreiecksungleichung $\|l - n\| \leq \|l - m\| + \|m - n\|$ folgt die Abschätzung

$$\inf_{n\in N} \|l - n\| \leq \sup_{l\in L} \|l - m\| + \sup_{m\in M} \|m - n\|.$$

Diese gilt für beliebige $l \in L$, $m \in M$, $n \in N$, welche noch als freie Variablen in der Ungleichung stehen. Also folgt insbesondere

$$\sup_{l\in L} \inf_{n\in N} \|l - n\| \leq \sup_{l\in L} \inf_{m\in M} \|l - m\| + \sup_{m\in M} \inf_{n\in N} \|m - n\| \leq \varepsilon + \delta,$$

was nach Definition bedeutet, dass $L$ in der $(\varepsilon + \delta)$-Umgebung von $N$ liegt. □

## 2.3 Problembeschreibung

*„Everything should be made as simple as possible, but not simpler.“*

-Albert Einstein

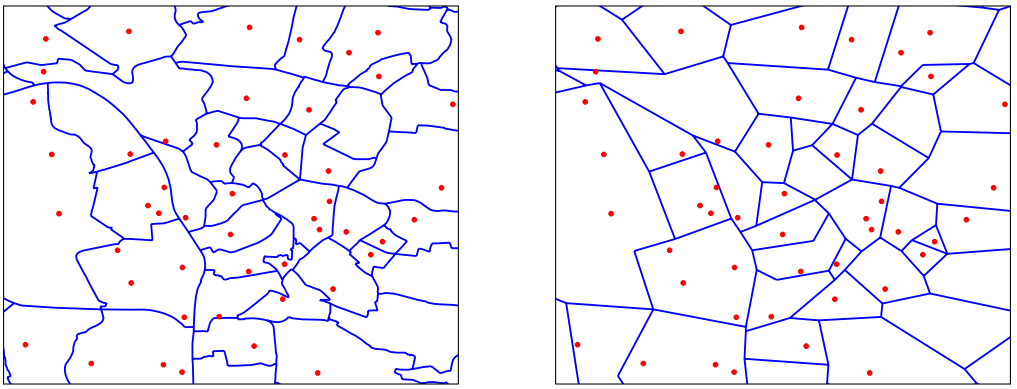
Sei eine Ebenenunterteilung in Form des Graphen $G = (V \subset \mathbb{R}^2, E)$ mit kanonischer Straight-line Einbettung gegeben. Gesucht ist eine Vereinfachung des Graphen $G$, in der möglichst viele Grad 2 Knoten ausgespart werden, d.h. es werden möglichst viele und lange Knotensequenzen $v_1, v_2, \ldots, v_n$ durch die einfache Kante $\overline{v_1 v_n}$ ersetzt, wobei alle Knoten $v_i$ mit $2 \leq i \leq n-1$ Grad-2 Knoten sind. Der bei der Vereinfachung entstandene Graph soll dabei die Planarität und die Topologie erhalten. Das Problem kann noch um folgende Nebenbedingungen erweitert werden.

**Fehlerschranke:** Sei eine Fehlerschranke $\varepsilon > 0$ gegeben. Zulässig sind nur die Vereinfachungen von Segmenten $S$ bestehend aus den Vertices $v_1, v_2, \ldots, v_n$ zu einer Kante $S' = \overline{v_1 v_n}$, wenn das Ausgangssegment $S$ in einer $\varepsilon$-Umgebung von der Kante $S'$ liegt, d.h.

$$S \in U_\varepsilon(S') \iff \max_{s \in S} \min_{s' \in S'} \|s - s'\| \leq \varepsilon.^*$$

**Topologieeinschränkungen:** Zusätzlich zur geforderten Planarität und Topologieerhaltung des vereinfachten Graphen $G'$ wird eine Menge $P \subset \mathbb{R}^2$ an Topolgieeinschränkungen gegeben. Die Punkte aus der Menge $P$ sollen vor und nach der Vereinfachung jeweils in der topologisch selben Facette liegen.

Zu einer gegebenen Ebenenunterteilung mit dem Graphen $G = (V, E)$, einer Fehlerschranke $\varepsilon$ und einer Menge $P$ an Topologieeinschränkungen ist eine vereinfachte Ebenenunterteilung $G' = (V', E')$ gesucht, welche die oberen Eigenschaften erfüllt mit minimaler Größe der Vertexmenge $V'$. Dieses Problem werden wir, angelehnt an die Namensgebung in [II88], *min-# Ebenenunterteilungsproblem* oder auch kurz *min-# Problem* nennen. Eine bildliche Darstellung des Problems ist in Abbildung 2.2 zu sehen.



(a) Höchste Auflösung. (b) Vereinfachte Ebenenunterteilung.

Abbildung 2.2: Unvereinfachte und vereinfachte Ebenenunterteilung.

## 2.4 Theoretische Untersuchungen des Problems

Das Problem der Segmentvereinfachung und aufbauend darauf das Problem der Vereinfachung von Ebenenunterteilungen ist schon lange ein bekanntes Problem in der Kartographie und Bildverarbeitung. Schon in den 70er Jahren wurden erste heuristische Lösungsansätze verfolgt, z.B. [DP73], [II88], [GHMS91], [CC92]. Das erste theoretische Ergebnis zu diesem Problem wurde in der Dissertationsschrift von

*Die Segmente können als Polygonzüge parametrisiert werden. Setzt man die Parametrisierungen in die Formel für den Abstand ein, dann wird sie somit zu einer stetigen Funktion auf einer kompakten Menge. Mit dem Satz von Weierstraß, [Rud05] Satz 4.16, nimmt der Abstand daher auch sein Minimum / Maximum an.

R. Estkowski [Est00] veröffentlicht, in der bewiesen wird, dass das Problem nicht in Polynomzeit lösbar ist, wenn P≠NP[†] gilt.

**Satz 2.11 (NP-vollständig [Est00])** Das min-# Problem ist für $P = \emptyset$ NP-vollständig.

Weiterhin wurde in der Veröffentlichung [EM01] das Problem als Minimierungsproblem untersucht und ein Ergebnis zur Approximierbarkeit gefunden. Die zum Verständnis benötigten Begriffe werden vorher kurz erläutert. Die Definitionen und Notationen sind hauptsächlich an [Kan94] angelehnt.

**Definition 2.12 (Minimierungsproblem)** Ein *Minimierungsproblem* über einem Alphabet $\Sigma$ ist ein Tupel $M = (I_M, S_M, m_M)$ wobei

- $I_M \subseteq \Sigma^*$ die Menge aller Eingaben und in Polynomzeit erkennbar ist,
- $S_M(x) \subseteq \Sigma^*$ die Menge aller zulässigen Lösungen für die Eingabe $x \in I_M$ ist, wobei es für jedes $x \in I_M$ ein Polynom $p$ und ein in Polynomzeit berechenbares Prädikat $\pi$ gibt, so dass

  $$S_M(x) = \{y \in \Sigma^* : |y| \leq p(|x|) \wedge \pi(x, y)\}.$$

  Man kann also mit $\pi$ in Polynomzeit bestimmen, ob $y$ eine zulässige Lösung ist und die Länge der Lösung $y$ wächst höchstens polynomiell mit der Länge der Eingabe $x$.
- $m_M : I_M \times \Sigma^* \longrightarrow \mathbb{N}$ bezeichnet die Zielfunktion, welche in Polynomzeit berechenbar ist. Der Wert $m_M(x, y)$ ist nur für $y \in S_M(x)$ definiert.

Weiterhin bezeichne $opt_M(x) = \inf_{y \in S_M(x)} m_M(x, y)$ den optimalen Wert der Zielfunktion.

**Definition 2.13 (Polynomiell beschränkt)** Sei ein Minimierungsproblem $M$ gegeben. Wenn die Zielfunktion eine obere Schranke besitzt, die polynomiell in der Eingabegröße ist, also

$$\exists \,\text{Polynom } p \,\forall\, x \in I_M \,\forall\, y \in S_M(x) : m_M(x, y) \leq p(|x|), \tag{2.3}$$

dann heißt die Zielfunktion *polynomiell beschränkt*.

**Definition 2.14 (MIN PB-Vollständigkeit)** Sei MIN PB die Klasse aller Minimierungsprobleme mit polynomiell beschränkter Zielfunktion. Dann heißt das Minimierungsproblem $M$ *MIN PB-vollständig* wenn

- $M \in$ MIN PB und
- jedes $N \in$ MIN PB ist polynomiell reduzierbar auf $M$.

[†]Für die Bedeutung der Komplexitätsklassen P und NP, siehe z.B. [AB02].

**Definition 2.15 (Approximierbarkeit)** Ein Minierungsproblem $M$ kann bis auf $q(n)$ ($q : \mathbb{Z}^+ \longrightarrow \mathbb{R}^+$) approximiert werden, wenn es einen Polynomzeitalgorithmus $A$ gibt, so dass für jedes $n \in \mathbb{Z}^+$ und alle Eingaben $x \in I_M$ mit $n = |x|$ gilt

$$A(x) \in S_M(x) \text{ und } m_M(x, A(x)) \leq q(n) \cdot opt_M(x).$$

Der Algorithmus $A$ berechnet also in Polynomzeit eine zulässige Lösung, die in der Zielfunktion maximal um den Faktor $q(n)$ von dem optimalen Wert abweicht.

**Satz 2.16 (MIN PB-vollständig, [EM01])** Das min-# Ebenenunterteilungsproblem ist für $P = \emptyset$ MIN PB-vollständig und kann nicht mit einem Polynomzeitalgorithmus, unter der Annahme dass P≠NP, bis auf einen Faktor von $q(n) = n^{1/5-\delta}$ zur optimalen Lösung approximiert werden für jedes $\delta > 0$, wobei $n$ die Anzahl der Vertices in der originalen Ebenenunterteilung bezeichnet.

# 3 Verwendete Algorithmen und Datenstrukturen

Bevor wir uns mit den Lösungsansätzen des min-# Problems beschäftigen, werden wir in diesem Kapitel einige Probleme mit Lösungsalgorithmen und Datenstrukturen anschauen, die Bestandteile der Lösungen sein werden. Dies umfasst Algorithmen zur Vereinfachung von Segmenten, Algorithmen auf Graphen und Triangulierungen.

## 3.1 Algorithmen zur Segmentvereinfachung

Zuerst stellen wir einige bekannte Algorithmen zur Segmentvereinfachungen vor, da diese Algorithmen die Grundlagen der Lösungsansätze des min-# Problems bilden. Aufgrund der Struktur von Segmenten werden wir im Folgenden Segmente statt als Graph $G_S = (V_S, E_S)$ als geordnete Menge der Vertices $(v_1, \ldots, v_n)$ schreiben, wobei es so zu verstehen ist, dass zwei aufeinanderfolgende Vertices im Segment über eine Kante verbunden sind.

### 3.1.1 Douglas-Peucker Algorithmus

Der Douglas-Peucker Algorithmus [DP73] ist ein heuristischer Algorithmus zur Vereinfachung von Segmenten. Bei gegebenen Ausgangssegment und einer unteren Schranke $\varepsilon$ liefert der Algorithmus als Ergebnis ein vereinfachtes Segment, zu dem das Ausgangssegment einen maximalen Abstand von $\varepsilon$ hat.

Die Idee ist, das Segment zuerst durch das Geradenstück zwischen Anfangs- und Endpunkt, $v_1$ und $v_n$ zu approximieren. Wenn alle dazwischen liegenden Segmentpunkte einen Abstand von $\varepsilon$ oder weniger haben, ist diese Approximation gültig. Wenn nicht, wählt man den Punkt $v_k$ mit dem größten Abstand zur Geraden und teilt das Problem rekursiv in die beiden Teilprobleme mit den Segmenten $v_1, \ldots, v_k$ und $v_k, \ldots, v_n$.

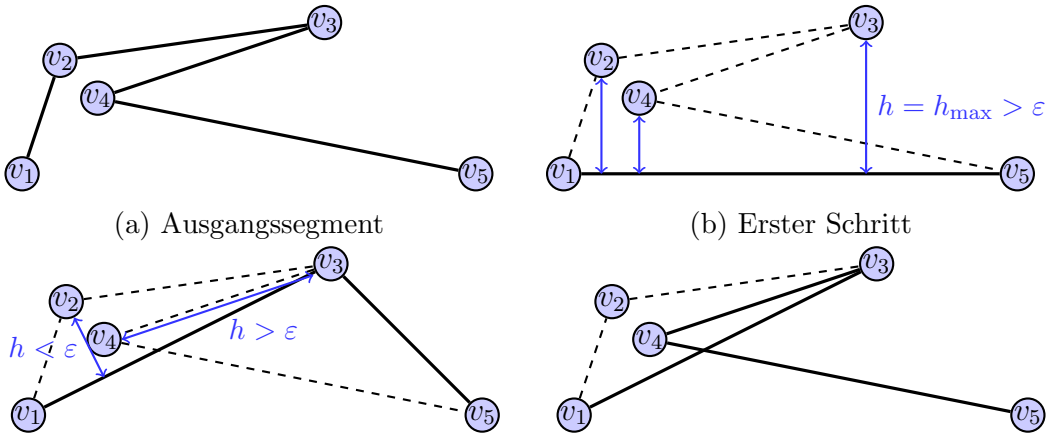
**Gegeben:** Segment $S = (v_1, \ldots, v_n)$, obere Schranke $\varepsilon > 0$.

**Gesucht:** Segment $S' = (v_1, v_{i_1}, \ldots, v_{i_m}, v_n)$ mit $2 \leq i_1 \leq \ldots \leq i_m \leq n-1$ und $S \subseteq U_\varepsilon(S')$.

**Algorithmus 3.1** Douglas-Peucker Algorithmus

```
1:  function DouglasPeucker(S = (v_1, ..., v_n), ε)
2:      S_approx = v_1 v_n
3:      d_max ← max d(v_i, S_approx)
4:      v_max ← arg max d(v_i, S_approx)
5:      if d_max < ε then
6:          return S_approx
7:      else
8:          S_L ← (v_1, ..., v_max)
9:          S_R ← (v_max, ..., v_n)
10:         return (DouglasPeucker(S_L, ε) ∪ DouglasPeucker(S_R, ε))
```

Wie bereits erwähnt, handelt es sich bei dem Douglas-Peucker Algorithmus um eine Heuristik, das heißt das Ergebnis ist im Allgemeinen nicht die optimale Lösung. Zudem werden keine Selbstüberschneidungen ausgeschlossen, wie in Abbildung 3.1 exemplarisch dargestellt.




(a) Ausgangssegment (b) Erster Schritt

(c) Zweiter Schritt (d) Ergebnis vom DP Algorithmus

Abbildung 3.1: Douglas-Peucker Iterationsschritte mit selbstüberschneidendem Ergebnis.

**Bemerkung 3.2** Zu beachten ist, dass der Abstand eines Punktes $v_i$ zu dem Geradenstück $S_{\text{approx}} = \overline{v_1 v_n}$ im obigen Algorithmus im Allgemeinen nicht als Höhe von $v_i$ auf der Geraden durch $v_1$ und $v_n$ berechnet werden kann, da das Lot nicht notwendigerweise auf dem Geradenstück $\overline{v_1 v_n}$ stehen muss. Der Abstand muss daher mit der Fallunterscheidung

$$d(v_i, \overline{v_1 v_n}) = \begin{cases} h(v_i, \overline{v_1 v_n}) & 0 \leq \langle v_i - v_1, \frac{v_i - v_n}{\|v_i - v_n\|} \rangle \leq 1 \\ \min(d(v_i, v_1), d(v_i, v_n)) & \text{sonst} \end{cases} \tag{3.1}$$

berechnet werden, wobei $h(v_i, \overline{v_1 v_n}) = \langle v_i - v_1, \frac{(v_i - v_n)^\perp}{\|v_i - v_n\|} \rangle$ die Höhe des Dreiecks $v_1$, $v_i$, $v_n$ mit Grundseite $\overline{v_i v_n}$ ist und $\cdot^\perp$ eine 90°-Drehung in der Ebene bezeichnet.

### 3.1.2 Imai-Iri Algorithmus

Ein weiterer Algorithmus zur Vereinfachung eines Polygonzugs ist der Imai-Iri Algorithmus, welcher erstmals in [II88] vorgestellt wurde. Die Idee vom Imai-Iri Algorithmus ist, zuerst einen Graphen zu konstruieren, der alle zulässigen, also um maximal $\varepsilon$ abweichenden Abkürzungen enthält und auf diesem Graphen den kürzesten Pfad zu finden.

**Gegeben:** Segment $S = (v_1, \ldots, v_n)$, obere Schranke $\varepsilon > 0$

**Gesucht:** Teilsegment $S' = (v_1, v_{i_1}, \ldots, v_{i_m}, v_n)$ mit $2 \leq i_1 \leq \ldots \leq i_m \leq n-1$, $S \subseteq U_\varepsilon(S')$ und minimaler Anzahl an Punkten unter allen zulässigen Segmenten.

**Algorithmus 3.3** Imai-Iri Algorithmus

1: **function** IMAI-IRI($S = (v_1, \ldots, v_n), \varepsilon$)
2: $\quad$ E $= \{(v_i, v_j) | 1 \leq i < j \leq n, \overline{v_i v_j} \in U_\varepsilon(S)\}$
3: $\quad$ Graph $G = (S, E)$
4: $\quad$ $S' =$ shortest_path($G$, $v_1$, $v_n$)
5: $\quad$ **return** ($S'$)

Die Konstruktion der Kantenmenge des Graphen $G$ in Zeile 2 vom Algorithmus 3.3 kann mit einem naiven Algorithmus in $\mathcal{O}(n^3)$ berechnet werden. Chan und Chin [CC92] haben ein Algorithmus aufgestellt, der $G$ in $\mathcal{O}(n^2)$ berechnet, was auch die optimale Laufzeit ist, da $G$ im schlimmsten Fall bis zu $\mathcal{O}(n^2)$ Kanten hat. Der kürzeste Pfad auf $G$ kann ebenfalls in $\mathcal{O}(n^2)$ mit topologischer Sortierung berechnet werden. Eine offensichtliche topologische Sorierung ist gegeben durch die Reihenfolge, in der die Vertices entlang des Segments sortiert sind.

$$v_i < v_j \Longleftrightarrow i < j$$

Damit kann man beim Dijkstra-Algorithmus (siehe z.B. [Sch09], Kapitel 15) zur Berechnung des kürzesten Pfades die Vertices $v_i$ nach dieser Ordnung durchlaufen und nur Kanten betrachten, die zu Vertices $v_j > v_j$ führen.

**Bemerkung 3.4** Wie beim Douglas-Peucker Algorithmus 3.1 werden keine Selbstüberschneidungen ausgeschlossen. Der Imai-Iri Algorithmus berechnet jedoch im Gegensatz zum Douglas-Peucker-Algorithmus die optimale Segmentvereinfachung, also ein Segment mit minimaler Anzahl an verbleibenden Punkten, zu einer gegebenen Fehlerschranke $\varepsilon$. Das ist einfach nachzuvollziehen, da die (bzw. eine) optimale Segmentvereinfachung einerseits als Pfad in $G$ enthalten sein muss und andererseits minimal ist, da sonst eine kürzere Segmentvereinfachung berechnet werden würde.

## 3.2 Algorithmen auf Graphen

Im vorangehenden Abschnitt haben wir uns mit einigen Algorithmen auf Segmenten beschäftigt, also insbesondere nur auf lokalen Teilmengen eines Graphen. Für das min-# Problem sind aber zusätzlich zu den Segmentvereinfachungen auch Algorithmen auf dem ganzen Graphen von Bedeutung. Ein Problem, welches bei der Lösungsfindung auftreten wird, ist das Unabhängige-Menge Problem.

### 3.2.1 Unabhängige-Menge Problem

Ein Problem gegeben auf Graphen ist das sogenannte *Maximale-Unabhängige-Menge* (Englisch: *Maximum Independent Set*) *Problem*. Gesucht ist dabei die größte Menge an Knoten in einem Graphen, welche paarweise nicht benachbart sind.

**Gegeben:** Graph $G = (V, E)$.

**Gesucht:** Menge $M \subset V$ mit $\forall\, v_1 \neq v_2 \in M :\ \{v_1, v_2\} \notin E$ und $|M|$ maximal.

Es ist bekannt, dass das Maximale-Unabhängige-Menge Problem zur Klasse der NP-vollständigen Probleme gehört, siehe z.B. [AB02], Satz 4.3.6. Ein Approximationsalgorithmus für dieses Problem ist der randomisierte Algorithmus 3.5.

**Algorithmus 3.5** Random Unabhängige Menge

```
 1: function RANDOM_UNABHÄNGIGE_MENGE(G = (V, E))
 2:     M ← ∅
 3:     V_rest ← V
 4:     while V_rest ≠ ∅ do
 5:         v ← random(V_rest)
 6:         V_rest ← V_rest \ {v}
 7:         M ← M ∪ {v}
 8:         for all w ∈ V_rest do
 9:             if {v, w} ∈ E then V_rest ← V_rest \ {w}
10:     return M
```

Falls ein Graph gegeben ist, welcher eine obere Schranke für den Grad aller Knoten besitzt, also $\forall\, v \in V :\ \deg(v) \leq k$, dann gibt der randomisierte Algorithmus 3.5 als Ergebnis eine unabhängige Menge mit mindestens $\lfloor \frac{|V|}{k} \rfloor$ Elementen.

Ersetzt man im Algorithmus 3.5 die Auswahl des nächsten zu betrachtenden Knoten $v$ in Zeile 5 durch die Auswahl des Knotens mit dem geringsten Grad, dann erhält man einen Greedy-Algorithmus für die Berechnung einer unabhängigen Menge.

Es ist bekannt, dass Maximale-Unabhängige-Menge Problem zu den schwer approximierbaren Problemen gehört, da es sich nicht bis auf eine Konstante approximieren lässt. Mit anderen Worten: Es gibt unter der Annahme P≠NP keinen Polynomzeitalgorithmus, der für eine beliebige Eingabe $G = (V, E)$ eine unabhängige Menge berechnet, die nur konstant weniger Vertices als die maximale unabhängige Menge enthält.

## 3.3 Triangulierungen

In diesem letzten Abschnitt werden wir Triangulierungen betrachten. Triangulierungen werden uns dabei helfen das min-# Problem auf einzelne lokale Probleme zurückzuführen, indem wir zu jedem Punkt jeweils nur die benachbarten Dreiecke als Umgebung in Betracht ziehen. Als *Triangulierung einer Punktmenge P* bezeichnen wir - wie in [BCKO08] - Ebenenunterteilungen, bei denen jede beschränkte Facette ein Dreieck ist und die Vertices der Facetten Punkte aus $P$ sind.

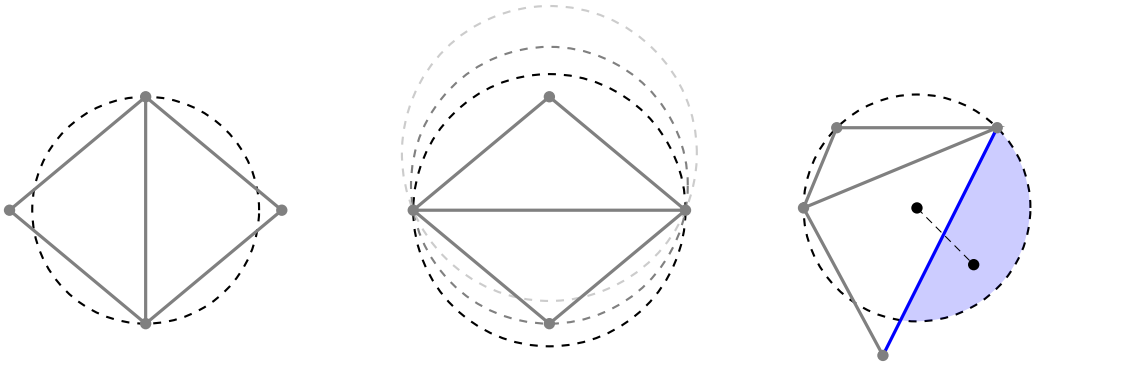
### 3.3.1 Delaunay-Triangulierungen

Triangulierungen auf Punktmengen sind nicht eindeutig und können sehr unterschiedlich aussehen. Manche Triangulierungen sehen weniger natürlich aus, da sie sehr lange Dreiecke mit spitzen Winkeln enthalten. Eine Triangulierung, welche solche Dreiecke so gut es geht vermeidet, ist die sogenannte Delaunay-Triangulierung.

**Definition 3.6 (Delaunay Triangulierung)** Sei eine Ebenenunterteilung $G = (V, E)$ gegeben, wobei es in der Vertexmenge $V \subset \mathbb{R}^2$ keine vier Punkte gibt, die auf einem Kreis liegen. Die Ebenenunterteilung $G$ heißt *Delaunay-Triangulierung* genau dann, wenn

$$\{v_1, v_2\} \in E \iff \text{Es gibt einen Kreis (Zeugenkreis), der durch } v_1 \text{ und } v_2 \text{ läuft und keinen weiteren Punkt aus } V \text{ im Inneren oder auf dem Rand enthält.} \quad (3.2)$$

Ein Bespiel für eine Kante mit Zeugenkreis und ein Beispiel für eine Kante ohne Zeugenkreis sind in Abbildung 3.2a bzw. 3.2b zu sehen.



(a) Zeugenkreis einer Kante.

(b) Kante ohne Zeugenkreis.

(c) Nicht sichtbarer Punkt im Umkreis.

Abbildung 3.2: Zulässige (a) und nicht zulässige Kante (b) in einer Delaunuay-Triangulierung nach der Zeugenkreis-Bedingung (3.2). Zulässiges Dreieck (c) in einer Eingeschränkten Delaunuay Triangulierung. Die blau gekennzeichnete Kante stellt eine Einschränkungskante dar.

Dass es sich bei der Definition 3.6 nicht nur um eine Ebenenunterteilung sondern tatsächlich um eine Triangulierung handelt, ist beispielsweise in [BCKO08], Abschnitt 9.2, bewiesen. Die Einschränkung an die Vertexmenge, also dass keine vier Punkte auf einem Kreis liegen, ist nur zur Vereinfachung der Definition und nicht als echte Einschränkung gedacht. Liegen vier Punkte von $V$ auf einem Kreis, dann muss es einen Kreis geben, der nur im Inneren keine weiteren Punkte von $V$ enthält. Es muss aber ausgeschlossen werden, dass sich die zwei Kanten der vier Punkte schneiden und die Delaunay Triangulierung ist in diesem Fall nicht mehr eindeutig.

Für die Konstruktion einer Delaunay-Triangulierung über einer Vertexmenge $V \subset \mathbb{R}^2$ mit $n = |V|$ gibt es einen randomisierten Algorithmus, der eine erwartete Laufzeit von $\mathcal{O}(n \log(n))$ hat (Siehe z.B. [BCKO08]: Algorithmus in Kapitel 9.3, Laufzeit in Theorem 9.12).

Eine zu Definition 3.6 alternative Definition einer Delaunay-Triangulierung lautet wie folgt: Eine Triangulierung ist eine Delaunay-Triangulierung, genau dann, wenn im Umkreis von jedem Dreieck kein weiterer Vertex der Triangulierung liegt. Mit dieser Bedingung ist es möglich die Definition 3.6 von Delaunay-Triangulierungen auf Eingeschränkte Delaunay-Triangulierungen zu erweitern.

### 3.3.2 Eingeschränkte Delaunay Triangulierungen

Will man nicht nur eine Delaunay-Triangulierung über einer Vertexmenge berechnen, sondern über einer Vertex-Menge zusammen mit einer Menge $M_P$ von fest gegebenen Polygonzügen, dann kann man Eingeschränkte Delaunay Triangulierungen (CDT) verwenden. Eine Eingeschränkte Delaunay Triangulierung kann im Allgemeinen die Zeugenkreis-Bedingung (3.2) nicht für alle Kanten erfüllen, da sie für die fest gegebenen Kanten der Polygonzüge nicht erzwungen werden kann. Die Bedingung an die Dreiecke der Eingeschränkten Delaunay Triangulierung muss also aufgelockert werden. Die folgenden Definitionen der Eingeschränkten Delaunuay Triangulierung richten sich nach [CGAb].

**Definition 3.7 (Sichtbarkeit in CDTs)** Ein Vertex $v \in \mathbb{R}^2$ ist von einem Punkt $p \in \mathbb{R}^2$ aus *sichtbar*, genau dann wenn die Verbindungsgerade $\overline{vp}$ kein Polygonzug aus $M_P$ schneidet.

**Definition 3.8 (Eingeschränkte Delaunay Triangulierung (CDT))** Eine Triangulierung über einer Vertexmenge $V \subset \mathbb{R}^2$ zusammen mit einer Menge $M_P$ von fest gegebenen Polygonzügen heißt *Eingeschränkte Delaunuay Triangulierung* genau dann, wenn im Umkreis von jedem Dreieck der Triangulierung kein Vertex (außer den Eckpunkten) vom Mittelpunkt des Umkreises sichtbar ist. Eine Kante eines Polygonszugs aus $M_P$ heißt *Einschränkungskante*.

Abbildung 3.2c verdeutlicht Definition 3.8. In dem eingezeichneten Umkreis des Dreiecks liegt ein Vertex, der jedoch vom Mittelpunkt des Umkreises nicht sichtbar ist, da eine Einschränkungskante die Verbindungsgerade schneidet.

# 4 Lösungsansätze

*„It is a mistake to think you can solve any major problems just with potatoes.“*

-Douglas Adams

Nachdem wir die grundlegenden Begriffe und Probleme in Kapitel 2 sowie bekannte Algorithmen in Kapitel 3 eingeführt haben, welche die Grundbausteine der Lösungsansätze für das min-# Problem sind, werden in diesem Kapitel verschiedene Lösungsansätze vorgestellt und diskutiert. Im Abschnitt 4.1 wird eine Auswahl von Algorithmen aus verschiedenen Artikeln vorgestellt. Abschnitt 4.2 behandelt im Gegensatz dazu den Algorithmus, welcher im Laufe dieser Arbeit implementiert und untersucht wurde.

## 4.1 Bekannte Ansätze

### 4.1.1 Einfacher-Umweg Heuristik

Eine in [EM01] vorgestellte Heuristik ist die Einfacher-Umweg Heuristik, welche in der gleichen Veröffentlichung für mehrere Datensätze bestehen aus Höhenlinien getestet wurde. Die Heuristik läuft in 4 Schritten ab.

**(1)** Zuerst wird die einfachste Ebenenunterteilung berechnet, welche eine gegebene Fehlertoleranz erfüllt. Dafür verwendet man auf jedem Segment separat ein Segmentvereinfachungsalgorithmus, wie beispielsweise den Douglas-Peucker-Algorithmus 3.1. Da die Segmentvereinfachungen separat berechnet wurden, ist es nicht ausgeschlossen (bzw. je nach Datensatz sehr wahrscheinlich), dass Überschneidungen paarweise zwischen den Segmenten und im Segment selbst auftreten.

**(2)** Im zweiten Schritt werden alle Paare von sich schneidenden Segmenten bestimmt, beispielsweise mit einem Sweep-Line-Algorithmus.

**(3)** Im nächsten Schritt werden nacheinander alle sich schneidenden Segmentpaare $s$ und $s'$ betrachtet und davon jeweils ein Segment ausgewählt, welches verfeinert wird um einen Umweg zu finden. (Für eine gute Strategie zur Auswahl dieses Segments, siehe [EM01].) Zu diesem Segment $s$ wird zuerst der Umweg-Graph berechnet, der alle Kanten enthält, die das andere Segment $s'$ nicht schneiden und die die Fehlertoleranz erfüllen. Sobald dieser Umweg-Graph konstruiert ist, wird der kürzeste Pfad vom Anfangs- zum Endpunkt bestimmt und als Verfeinerung des Segments $s$ genommen. Existiert solch ein Pfad nicht, so werden $s$ und $s'$ beide jeweils um einen Punkt verfeinert.

**(4)** Gehe wieder zu Schritt (2) und bestimme die neu entstandenen Segmentüberschneidungen. Der Algorithmus terminiert sobald keine Überschneidungen von Segmenten mehr auftreten.

### 4.1.2 Konkatenierte Imai-Iri Heuristik

Ein weiterer Ansatz für das Lösen des min-# Problems wurde in [BKS95] vorgestellt. Bei diesem Ansatz wird eine Approximationslösung innerhalb von $\mathcal{O}(n(n+m)\log(n))$ berechnet, wobei $n$ die Anzahl der Vertices der Ebenenenunterteilung und $m$ die Anzahl der Einschränkungspunkte in $P$ ist. Leider wurden zu dieser Heuristik keine experimentellen Untersuchungen gemacht.

Die Konkatenierte Imai-Iri Heuristik baut auf dem Algorithmus von Imai und Iri aus Abschnitt 3.1.2 auf. Anders als die Einfacher-Umweg Heuristik werden in diesem Ansatz keine Vereinfachungen berechnet, aus der anschließend die Überschneidungen beseitigt werden. Stattdessen wird in [BKS95] der Imai-Iri-Algorithmus so erweitert, dass das dabei entstehende Segment die Topologieeinschränkungen aus $P$ einhält und weder sich selbst schneidet noch andere Segmente.

**Erweiterung um Topologieeinschränkungen:** Die Erweiterung des Imai-Iri Algorithmus erfolgt zuerst für $x$-monotone Segmente. Dabei wird zusätzlich zum Graph $G$, der in Algorithmus 3.3 die bezüglich Fehlertoleranz zulässigen Abkürzungen beschreibt, ein weiterer Graph $H$ eingeführt, der alle bezüglich Topologieeinschränkungen zulässigen Kanten enthält. Sei ein Segment $S = (v_1, \ldots, v_n)$ gegeben. Dann werden Vertexmenge $V_H$ und Kantenmenge $E_H$ der Graphen $H$ definiert als

$$\begin{aligned} V_H =& \{v_1, \ldots, v_n\} \\ E_H =& \{\{v_i, v_j\} | 1 \leq i < j \leq n, \text{ in der zwischen } S_{ij} = (v_i, \ldots, v_j) \text{ und } \overline{v_i v_j} \\ & \text{eingeschlossenen Fläche liegen keine Punkte aus } P\} \end{aligned}$$

Der Schnitt der Kantenmengen von den Graphen $G$ und $H$ gibt dann alle Abkürzungskanten, die beide Eigenschaften erfüllen. Der kürzeste Pfad von $v_1$ nach $v_n$ auf dem Schnittgraphen ist demzufolge die Vereinfachung des Segments $S$ mit kürzester Länge, welche die Topologieeinschränkungen $P$ einhält. Die Erweiterung auf nicht $x$-monotone Segmente erfolgt durch eine Zerlegung des Segments in möglichst lange (ggf. mit rotiertem Koordinatensystem) $x$-monotone Teilsegmente. Für nicht $x$-monotone Segmente liefert die Erweiterung des Imai-Iri-Algorithmus daher nicht mehr die minimale Vereinfachung.

**Keine Überschneidungen:** Jedes $x$-monotone Teilsegment besitzt nur Vereinfachungen, die keine Selbstüberschneidungen aufweisen. Sei also ein $x$-monotones Teilsegment $(v_i, \ldots, v_j)$ eines Segments $(v_1, \ldots, v_i, \ldots, v_j, \ldots, v_n)$ gegeben. Füge dann, um Selbstüberschneidungen zu vermeiden, zu der Menge aller Topologieeinschränkungen $P$ die Vertices $v_1, \ldots, v_{i-1}$ und $v_{j+1}, \ldots, v_n$ hinzu.

Soweit werden Segmente vereinfacht, sodass keine Selbstüberschneidungen auftreten. Um Überschneidungen mit anderen Segmenten zu vermeiden, muss man zu der Menge $P$ noch die Vertices aller anderen Segmente der Ebenenunterteilung hinzufügen. Die Anzahl der Punkte, die zu $P$ hinzugefügt werden, kann man reduzieren, indem man nur Punkte hinzufügt, die in der konvexen Hülle des Segments liegen.

## 4.2 Ansatz über Delaunay-Triangulierungen

Ein neuer Ansatz, der auch den Schwerpunkt dieser Bachelorarbeit bildet, ist die gegebene Ebenenunterteilung zu triangulieren und mit der erhaltenen Triangulierung das Problem auf viele lokale Probleme zu reduzieren. Natürlich wird durch die Reduktion auf lokale Probleme die Ebenenunterteilung nicht optimal vereinfacht, daher handelt es sich um eine heuristische Methode.

### 4.2.1 Erzeugen der entsprechenden Delaunay Triangulierung

Sei eine Ebenenunterteilung $G = (V, E)$ und eine Menge von Punkten $P$ als Topologieeinschränkungen gegeben. Dann erzeugen wir aus der Punktmenge $V_D = V \cup P$ eine Eingeschränkte Delaunay Triangulierung $G_D$, wobei alle Kanten aus der Menge $E$ in $G_D$ enthalten sein müssen, wie in Abbildung 4.1 dargestellt. Die Kanten aus $E$ werden als *Einschränkungskanten* der Delaunay-Triangulierung bezeichnet.

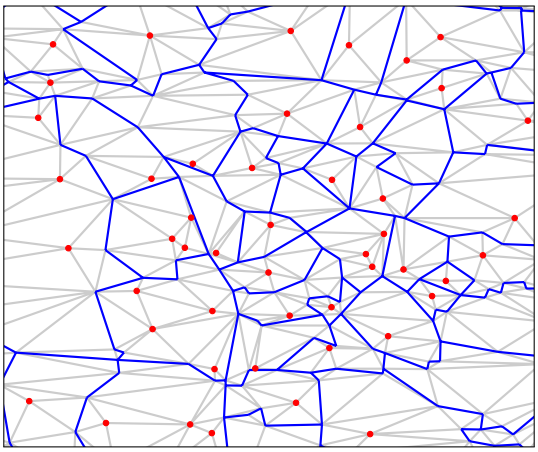


Abbildung 4.1: Ebenenunterteilung als eingeschränkte Delaunay Triangulierung. Einschränkungskanten sind blau, alle weiteren Kanten der Triangulierung sind grau gezeichnet. Die Topologieeinschränkungen sind als rote Punkte gezeichnet und Teil der Triangulierung.

Solch eine Triangulierung existiert unter der Voraussetzung, dass keiner der Punkte aus $P$ auf einer Kante $e \in E$ liegt. Jede Kante der Ebenenunterteilung entspricht also einer Einschränkungskante in der Eingeschränkten Delaunay Triangulierung. Alle weiteren Kanten in der Delaunay Triangulierung haben keine Entsprechung in der Ebenenunterteilung, werden aber im folgenden Algorithmus verwendet um heuristisch zu bestimmen ob ein Punkt aus der Ebenenenunterteilung entfernt werden kann.

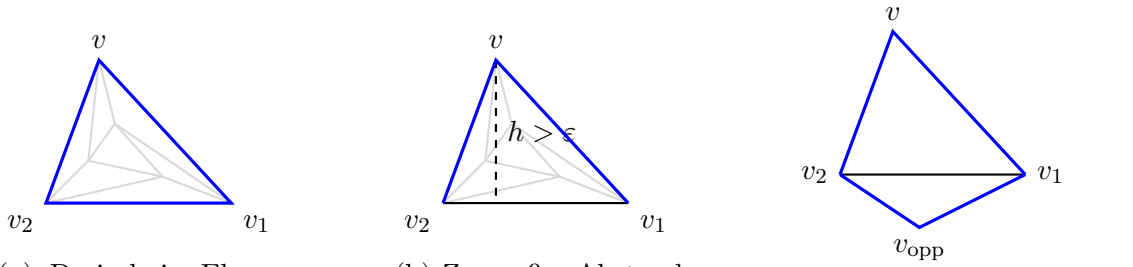
| Ebenenunterteilung $G = (V, E)$ | | Eingeschränkte Delaunay Triangulierung $G_D = (V_D, E_D)$ |
|---|---|---|
| Vertex $v \in V$ | $\leftrightarrow$ | Vertex $v \in V_D$ |
| Topologieeinschränkung $p \in P$ | $\leftrightarrow$ | Vertex $p \in V_D$ |
| Kante $e \in E$ | $\leftrightarrow$ | Einschränkungskante $e \in E_D$ |
| - | $\leftrightarrow$ | Nicht einschränkende Kante $e \in E_D$ |
| $\deg_G(v \in V)$ | $\leq$ | $\deg_{G_D}(v \in V_D)$ |

### 4.2.2 Finden von entfernbaren Punkten

Die Delaunay Triangulierung wird genutzt um zu bestimmen, ob ein Grad-2 Punkt $v$ aus der Ebenenunterteilung entfernt werden kann, ohne dass es zu Überschneidungen oder zur Verletzung von Topologieeinschränkungen kommt: Sei ein Punkt $v \in V$ mit $\deg_G(v) = 2$ gegeben. Die in der Ebenenunterteilung angrenzenden Punkte an $v$ werden im Folgenden $v_1$ und $v_2$ genannt. Wir wollen nun prüfen, ob der Punkt $v$ entfernt werden kann. Dies kann in 2 Fällen sofort ausgeschlossen werden.

**Fall 1.** Die Kante $\{v_1, v_2\}$ ist ebenfalls eine Kante der Ebenenunterteilung, also $\{v_1, v_2\} \in E$, siehe Abbildung 4.2a.

**Fall 2.** Der Abstand von Punkt $v$ zur Strecke $\overline{v_1 v_2}$, welcher wie bei der Abstandsberechnung (3.1) im Douglas-Peucker-Algorithmus berechnet wird, ist größer als die gegebene Fehlerschranke $\varepsilon$, siehe Abbildung 4.2b.



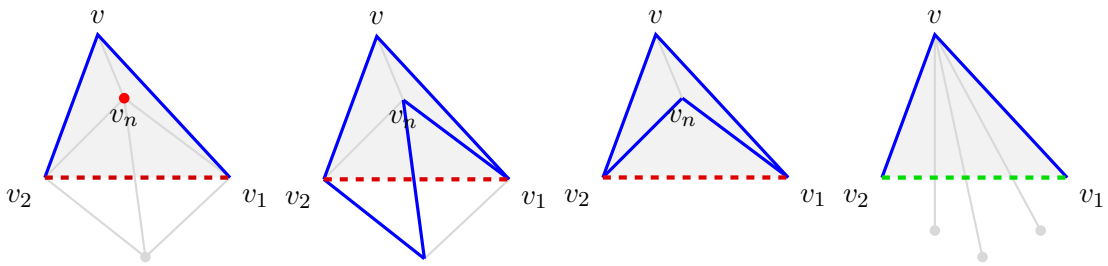

(a) Dreieck in Ebenenunterteilung

(b) Zu großer Abstand zur Kante $\overline{v_1 v_2}$

(c) Viereck in Ebenenunterteilung

Abbildung 4.2: Ausschlussfälle für das Entfernen des Punktes $v$.

Trifft keiner dieser beiden Fälle zu, kann man weiterhin prüfen, ob die Vereinfachung der Knotensequenz $v_1, v, v_2$ zum einfachen Segment $v_1, v_2$ zulässig bezüglich Topologieeinschränkungen ist. Wir prüfen für jeden Punkt $v_n \neq v_1, v_2$, welcher adjazent zum Punkt $v$ in der eingeschränkten Delaunay Triangulierung ist, ob er sich innerhalb des Dreiecks $\triangle(v, v_1, v_2)$ befindet. Befindet sich mindestens ein Punkt $v_n$ innerhalb des Dreiecks, dann führt die Vereinfachung zu einer Verletzung der Topologiebedingungen: Im Fall dass der Punkt $v_n$ ein Topologiepunkt $v_n = p \in P$ in der Ebenenunterteilung ist, wandert durch die Vereinfachung der Punkt $p$ auf die andere Seite des vereinfachten Segments, siehe Abbildung 4.3a. Im Fall dass der Punkt $v_n$ ein Punkt der Ebenenunterteilung $v_n \in V$ ist, gibt es ein Segmentstück, welches durch $v_n$ läuft. Die Vereinfachung der Sequenz $v_1, v, v_2$ zu $v_1, v_2$ führt also entweder zu einer (Selbst-)Überscheidung oder zu einer Änderung der Orientierung einer Facette, wie jeweils in den Abbildungen 4.3b und 4.3c dargestellt.




(a) Topologieeinschränkung (b) Überschneidung des Segments (c) Änderung der Orientierung (d) Kein Konflikt, $\forall\, v_n \notin \triangle(v, v_1, v_2)$

Abbildung 4.3: Konflikte bei der Vereinfachung der Punktsequenz $v_1, v, v_2$ zu $v_1, v_2$.

Befindet sich kein zu $v$ adjazenter Punkt $\neq v_1, v_2$ im Dreieck $\triangle(v, v_1, v_2)$, dann führt die Vereinfachung zu keinem Konflikt.

### 4.2.3 Gleichzeitiges Entfernen von Punkten

Im vorangehenden Abschnitt 4.2.2 wurde beschrieben, wie man für einen einzelnen Punkt mithilfe der Delaunay Triangulierung in $\mathcal{O}(\deg_{G_D}(v))$ herausfinden kann, ob er aus der Ebenenunterteilung entfernt werden kann. Wir wollen aber für die Vereinfachung der Ebenenunterteilung eine möglichst große Menge an Punkten finden, die wir gleichzeitig entfernen können. Dafür ist eine unabhängige Menge notwendig. Beim Entfernen einer unabhängigen Punktmenge aus der Triangulierung, wobei für jeden Punkt dieser Menge die Topologieeinschränkungen und die Fehlerschranke $\varepsilon$ eingehalten werden, wird auch auf der gesamten Ebenenenunterteilung die Topologieeinschränkungen und die Fehlerschranke nicht verletzt.

Es kann beim gleichzeitigen Entfernen von zwei Punkten $v^a$, $v^b$ nur dann zu einem Konflikt kommen, wenn $v^a$ und $v^b$ inzident sind. Wenn es zwei Punkte $v_1$ und $v_2$ gibt, die beide in der Ebenenunterteilung zu $v^a$ oder $v^b$ inzident sind, dann kann es zu einer Degenerierung einer Facette kommen (d.h. die Facette würde nur noch zwei Eckpunkte besitzen). Die Punkte $v^a$, $v^b$, $v_1$ und $v_2$ bilden also ein Viereck wie in Abbildung 4.2c.

Wir suchen also eine unabhängige Menge von entfernbaren Punkten und schließen ebenfalls aus, dass je zwei Punkte in dieser Menge vorhanden sind, die auf einem Viereck der Ebenenenunterteilung liegen. Dazu bestimmen wir die unabhängige Menge mit einem Greedy-Algorithmus. Dieser Algorithmus hat auch den Vorteil, dass das Entfernen von Punkten mit geringem Grad weniger Laufzeit benötigt.

### 4.2.4 Zusammenfassung des Algorithmus

Der Algorithmus wird so formuliert, dass in jedem Iterationsschritt eine unabhängige Menge von Punkten entfernt wird. Zuerst werden alle Grad-2 Knoten in eine Punktliste eingefügt und nach aufsteigendem Grad in der Triangulierung sortiert. Für

jeden dieser Punkte wird bestimmt, ob beim Herauslöschen Konflikte erzeugt werden. Wenn nicht, werden alle angrenzenden Punkte aus der Punktliste entfernt, der untersuchte Punkt aus der Triangulierung gelöscht und die erhaltene Ebenenenunterteilung wird wieder zu einer Triangulierung ergänzt. Dieses Vorgehen wird solange wiederholt, bis aus der Liste der Grad-2 Punkte keine Punkte mehr gelöscht werden können.

**Algorithmus 4.1** Vereinfachungs-Algorithmus

```
1:  function SIMPLIFY(G, ε_min, ε_max)
2:      ε = ε_min
3:      num_removed_points = 0
4:      while (ε < ε_max) do
5:          δ = ε/2
6:          do
7:              num_removed_points = 0
8:              δ = δ/2
9:              Set S = deg_2_vertices(G)
10:             for all v ∈ S do
11:                 if no_conflicts(δ, v) then
12:                     remove_point(G,v)          ▷ Mit Triangulierung der Lücken
13:                     remove_incident_points(S,v)
14:                     num_removed_points = num_removed_points + 1
15:         while ( num_removed_points > 0 )
16:         ε = 2 * ε
17:     return (G)
```

Der Algorithmus garantiert für jedes durchlaufene $\varepsilon$, welches im Bereich $[\varepsilon_{min}, \varepsilon_{max}]$ liegt, dass die erhaltene Ebenenunterteilung $G^\varepsilon$ maximal um $\varepsilon$ von der ursprünglichen Ebenenunterteilung $G$ abweicht.

Um das nachzuvollziehen, betrachten wir zuerst die innere **do-while**-Schleife in Zeile 6-15 vom Algorithmus 4.1. In jedem Schleifendurchlauf entsteht dabei eine vereinfachte Ebenenunterteilung $G_i^\varepsilon$, so dass wir eine Sequenz von Ebenenunterteilungen erhalten

$$G_0^\varepsilon, G_1^\varepsilon, \ldots, G_k^\varepsilon = G^\varepsilon.$$

Die Variable $\varepsilon$ hat während dem Durchlauf der inneren do-while-Schleife einen unveränderten Wert und für diesen Parameter $\varepsilon$ liegt die resultierende Ebenenunterteilung $G^\varepsilon$ in einer $\frac{\varepsilon}{2}$-Umgebung der ersten Ebenenenunterteilung $G_0^\varepsilon$, denn es gilt

$$G_1^\varepsilon \subseteq U_{\varepsilon/4}(G_0^\varepsilon) \wedge G_2^\varepsilon \subseteq U_{\varepsilon/8}(G_1^\varepsilon) \wedge \ldots \wedge G^\varepsilon = G_k^\varepsilon \subseteq U_{\varepsilon/2^{k+1}}(G_{k-1}^\varepsilon)$$

und somit mit dem Schachtellemma 2.10

$$G^\varepsilon \subseteq U_{\varepsilon/4+\varepsilon/8+\ldots+\varepsilon/2^{k+1}}(G_0^\varepsilon) \subset U_{\varepsilon/2}(G_0^\varepsilon). \tag{4.1}$$

Betrachten wir nun die äußere **while**-Schleife in Zeile 4-16. Innerhalb dieser Schleife durchläuft $\varepsilon$ die Werte $\{\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_l\}$ mit $\varepsilon_i = 2^{i-1} \cdot \varepsilon_{min} < \varepsilon_{max}$. Wir wollen

nun zeigen, dass jede, nach einem Durchlauf der äußeren Schleife erhaltene Ebenenunterteilung $G^{\varepsilon_i}$ um nicht mehr als $\varepsilon_i$ von der Ausgangsebenenunterteilung $G$ abweicht, das heißt in einer $\varepsilon_i$-Umgebung von $G$ liegt. Wir wissen, analog zur oberen Überlegung, dass mit (4.1) die Beziehungen

$$G^{\varepsilon_i} \subseteq U_{\varepsilon_i/2}(G^{\varepsilon_{i-1}}) \wedge G^{\varepsilon_{i-1}} \subseteq U_{\varepsilon_{i-1}/2}(G^{\varepsilon_{i-2}}) \wedge \ldots \wedge G^{\varepsilon_1} \subseteq U_{\varepsilon_1/2}(G)$$

gelten. Es folgt wieder mit dem Schachtellemma

$$G^{\varepsilon_i} \subseteq U_{\varepsilon_i/2+\varepsilon_{i-1}/2+\ldots+\varepsilon_1/2}(G).$$

Mit der Wahl $\varepsilon_i = 2^{i-1}\varepsilon_{min}$ folgt für die Summe der Abstände

$$\begin{aligned}\frac{\varepsilon_i}{2} + \frac{\varepsilon_{i-1}}{2} + \ldots + \frac{\varepsilon_1}{2} &= 2^{i-1}\frac{\varepsilon_{min}}{2} + 2^{i-2}\frac{\varepsilon_{min}}{2} + \ldots + \frac{\varepsilon_{min}}{2} \\ &= 2^{i-1}\varepsilon_{min} \cdot \left(\frac{1}{2} + \frac{1}{4} + \ldots + \frac{1}{2^{i-2}}\right) \\ &< 2^{i-1}\varepsilon_{min} = \varepsilon_i.\end{aligned}$$

Damit ist $G^{\varepsilon_i} \subseteq U_{\varepsilon_i}(G)$.

### 4.2.5 Eigenschaften des Verfahrens

1. Das Verfahren ist ein top-down Verfahren. Man beginnt mit der feinsten Ebenenunterteilung und entfernt sukzessive Grad-2 Punkte so, dass das Zwischenergebnis weiterhin eine gültige Ebenenunterteilung ist. Die Einfacher-Umweg Heuristik aus Abschnitt 4.1 ist im Gegensatz dazu ein bottom-up Verfahren, da man mit der gröbsten Ebenenunterteilung anfängt und nach und nach Punkte hinzufügt bis eine zulässige Ebenenunterteilung entstanden ist.

2. Das Verfahren ist ein heuristisches Verfahren. Im Allgemeinen wird die optimale Lösung nicht gefunden, da nur das Löschen von Grad-2 Punkten erlaubt ist, wenn die erhaltene Ebenenunterteilung zulässig ist. Es kann aber eine optimale Ebenenunterteilung geben, die nicht durch eine Sequenz von zulässigen Ebenenenunterteilungen erreichbar ist, wenn nur das Entfernen einzelnen Punkten bzw. unabhängigen Mengen erlaubt ist. Ein Beispiel ist in Abbildung 4.4 dargestellt.

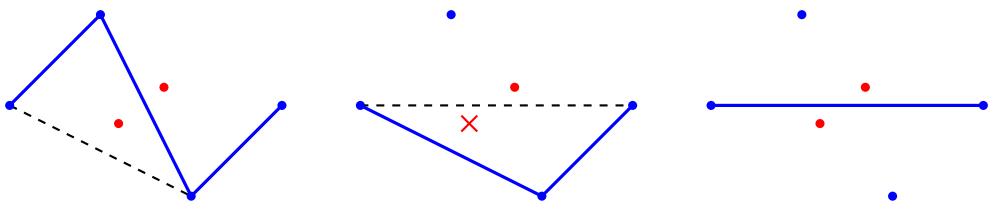


Abbildung 4.4: Sequenz von Segmentvereinfachungen die über ein nicht-zulässiges Segment zum Optimum führen.

# 5 Erweiterung um eine Vereinfachungs-Datenstruktur

Bisher haben wir mit Algorithmus 4.1 ein Verfahren vorgestellt, dass bei Eingabe eines Wertes $\varepsilon_{max}$ eine zulässige Ebenenunterteilung $G^{\varepsilon_{max}}$ als Ergebnis liefert. Es ist in der Anwendung aber auch sinnvoll die Zwischenergebnisse $G^{\varepsilon_i}$ der Berechnung ebenfalls zu speichern. Betrachtet man als Beispiel eine Landkarte, dann wird üblicherweise zuerst die größte Zoom-Ebene dargestellt, also mit $\varepsilon = \varepsilon_{max}$. Nachdem der Nutzer einen bestimmten Ort gefunden hat, zoomt er näher ran um den interessanten Bereich detaillierter darzustellen. Mit dem bisherigen Verfahren müssten wir nun erneut eine Ebenenunterteilung berechnen für ein neues $\varepsilon < \varepsilon_{max}$, was aber schon ein Zwischenergebnis der ersten Berechnung gewesen sein kann.

Speichert man die Sequenz von Vereinfachungen in einer Vereinfachungsdatenstruktur, dann fällt die erneute Berechnung der vereinfachten Ebenenunterteilung weg. Ein weiterer Vorteil einer Vereinfachungs-Datenstruktur liegt darin, dass man die Delaunay-Triangulierung nach der Berechnung nicht mehr benötigt. Dadurch spart man sich den Overhead der Datenstruktur der Delaunay-Triangulierung.

Die Vereinfachungs-Datenstruktur wird als Wald von Binärbaumen dargestellt mit der folgenden Struktur eines Knotens.

```
struct Simple_edge
{
    Point_2 *p_begin, *p_end;

    bool is_leaf;

    Simple_edge *parent,
                *child1,
                *child2;

    R eps; // Distance: this edge <-> original arrangement
};
```

Ein Knoten bezeichnet dabei eine Kante, die durch die zwei Eckpunkte `p_begin` und `p_end` eindeutig gegeben ist. Die Kante kann dabei entweder eine ursprüngliche Kante der Ebenenunterteilung sein oder eine Kante, welche durch das Zusammenfassen von zwei benachbarten Kanten entstanden ist. Die Blattkanten werden durch die Variable `is_leaf` gekennzeichnet und bilden zusammen die Ausgangsebenenunterteilung. Alle Wurzelknoten des Waldes bilden hingegen die gröbste (bzw. aktuelle) Ebenenunterteilung. Weiterhin hat jeder Knoten `se` $\in$ `Simple_edge` einen Wert `eps`, welcher eine Näherung ist an den Abstand der gegeben Kante zu der Ebenenunterteilung.

## 5.1 Aufbau der Vereinfachungs-Datenstruktur

Der Aufbau der Vereinfachungs-Datenstruktur ist sehr eingängig. Füge zuerst alle Kanten der Ebenenunterteilung als Blattknoten in den Wald ein. Bei jedem Löschen

von Punkten in Algorithmus 4.1 in Zeile 12, finde zu jedem gelöschten (Grad-2) Punkt $v$ die beiden angrenzenden Kanten $e_L = (v_1, v)$ und $e_R = (v, v_2)$ und füge eine neue Kante $e_M = (v_1, v_2)$ als Vaterkante zu diesen beiden Kanten ein. Die zu löschenden Kanten haben alle keine gemeinsamen Eckpunkte, da eine unabhängige Punktmenge aus der Ebenenunterteilung gelöscht wird. Daher spielt die Reihenfolge, in der die Punkte $v$ durchlaufen werden keine Rolle und der Wald wird um genau eine Ebene tiefer. Die Konstruktion der Vereinfachungs-Datenstruktur ist an einem einfachen Beispiel in Abbildung 5.1 vorgeführt.

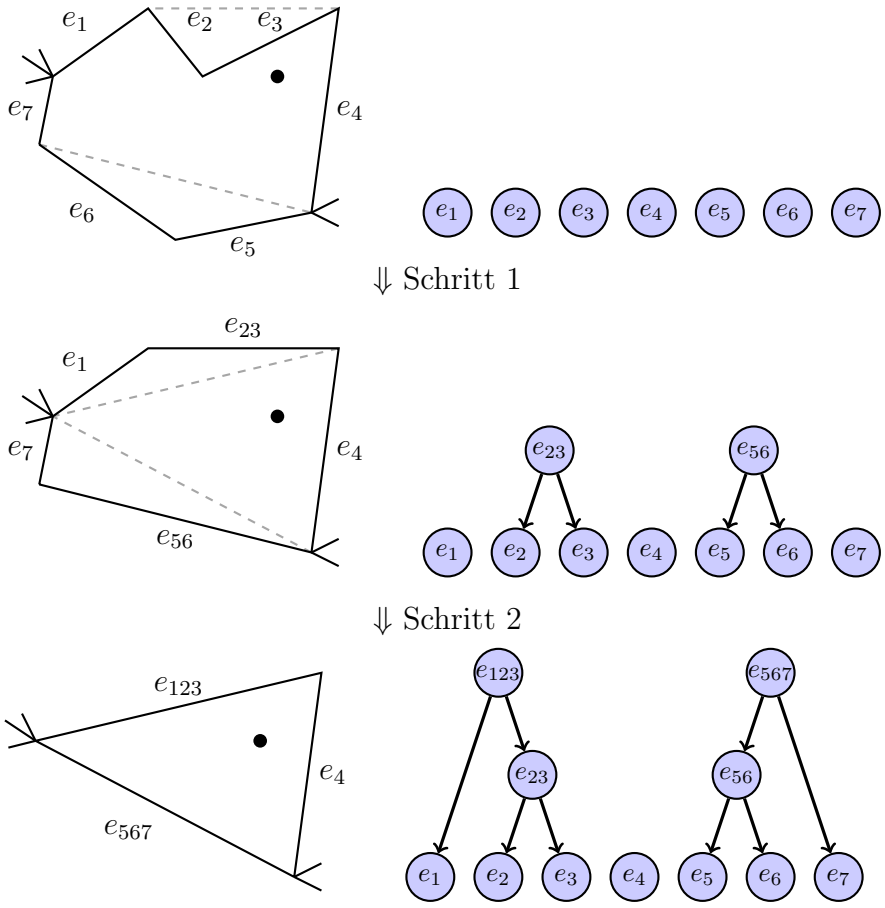



Abbildung 5.1: Aufbau der Vereinfachungs-Datenstruktur.

## 5.2 Berechnung des Abstands einer Kante zum Originalsegment

Wie bereits erwähnt, soll in der Variable `eps` eine Näherung an den Abstand der gegebenen Kante zur unvereinfachten Ebenenenunterteilung gespeichert werden. Im Folgenden werden wir als Näherung den Abstand zu dem zugrunde liegenden Segment, also zu allen Kanten, die Blattknoten des Teilbaumes von `se` sind, betrachten. Da das unvereinfachte Segment Teil der unvereinfachten Ebenenunterteilung ist, ist diese Näherung größer oder gleich dem geschätzten Wert. Die Größe der Vergröberung wird also nicht unterschätzt. Wir untersuchen zwei Ansätze an die Näherung des Wertes `eps`

**Ansatz 1.** Sei die Kante `se` mit Eckpunkten $v_L$, $v_R$ gegeben. Falls `se` kein Blattknoten ist, bezeichnen `ch1` und `ch2` die beiden Kindknoten mit Eckpunkten $v_L$, $v_M$ bzw. $v_M$, $v_R$. Für den Wert `eps` wird folgende Schätzung gewählt

$$\texttt{eps}^1 = \begin{cases} 0 & : \text{falls } \texttt{se} \text{ Blattknoten} \\ d(v_M, \overline{v_L v_R}) + \max(\texttt{eps}^1_{\texttt{ch1}}, \texttt{eps}^1_{\texttt{ch2}}) & : \text{sonst} \end{cases} \tag{5.1}$$

**Ansatz 2.** Wir betrachten wieder die Kante `se` mit Eckpunkten $v_L$, $v_R$. Weiterhin bezeichne $G_{\texttt{se}} = (V_{\texttt{se}}, E_{\texttt{se}})$ das Segment, aus dem `se` durch Vereinfachung hervorgegangen ist. Dann setzen wir den Wert für `eps` als

$$\texttt{eps}^2 = \max_{v \in V_{\texttt{se}}} \{d(v, \overline{v_L v_R})\} \tag{5.2}$$

Mit dieser Wahl ist $\varepsilon$ also der (bis auf Maschinenrundung) genaue Abstand der Kante `se` zum ursprünglichen Segment.

Wenn man beide Ansätze vergleicht, dann sieht man, dass Ansatz 1 einerseits nur eine Näherung an den Abstand zum Segment liefert, aber andererseits lediglich zwei Zugriffe auf die beiden Kinderknoten `ch1` und `ch2` benötigt und damit sehr schnell berechnet werden kann. Ansatz 2 berechnet den genauen Abstand zum Originalsegment und liefert daher eine bessere Schätzung an den Abstand zur Ebenenunterteilung. Damit bekommt in der Regel eine vereinfachte Ebenenunterteilung mit weniger Kanten als mit dem ersten Ansatz bei gleicher Wahl der Schranke $\varepsilon_{tol}$. Um den Wert $\texttt{eps}^2$ zu berechnen muss man jedoch für jedes Segment `se` alle Blätter des an `se` hängenden Teilbaumes durchlaufen. Man muss also zwischen Performanz und Approximationsgüte abwägen. Einige Beispiele mit Laufzeiten werden in Kapitel 6 für die Berechnung von `eps` mit Ansatz 2 vorgestellt.

Ein weiterer Unterschied zwischen beiden Ansätzen ist der Aufbau der Vereinfachungsstruktur. Beim Ansatz 1 bekommt jeder in die Vereinfachungsstruktur eingefügte Knoten einen Wert `eps` zugeordnet, der größer oder gleich zu den $\texttt{eps}^1$-Werten der Kinderknoten ist, da

$$\texttt{eps}^1 = d(v_M, (v_L, v_R)) + \max(\texttt{eps}^1_{\texttt{ch1}}, \texttt{eps}^1_{\texttt{ch2}}) \geq \max(\texttt{eps}^1_{\texttt{ch1}}, \texttt{eps}^1_{\texttt{ch2}}).$$

Die $\texttt{eps}^1$-Werte sinken daher mit wachsender Tiefe der Vereinfachungsstruktur mit einer unteren Schranke von 0 (Blattknoten). Bei Ansatz 2 gilt diese Eigenschaft für $\texttt{eps}^2$ jedoch nicht.

Mit der Einführung der Vereinfachungsstruktur und den `eps`-Werten ist es nicht mehr nötig in Algorithmus 4.1 in der inneren Schleife die maximale Fehlerschranke $\delta$ in jedem Zeitschritt zu halbieren. Statt dessen wird im Laufe der Vereinfachung die Vereinfachungsstruktur aufgebaut und diese zum Schätzen des maximalen Fehlers verwendet. Das angepasste Verfahren ist der Algorithmus 5.1.

**Algorithmus 5.1** Vereinfachungs-Algorithmus mit Vereinfachungsstruktur

```
1:  function SIMPLIFY(G = (V, E), ε_min, ε_max)
2:      Simplification_structure simpli(E)                ▷ Blattknoten einfügen
3:      ε = ε_min
4:      num_removed_points = 0
5:      while (ε < ε_max) do
6:          do
7:              num_removed_points = 0
8:              Set S = deg_2_vertices(G)
9:              for all v ∈ S do
10:                 if no_conflicts(ε, v) then
11:                     remove_point(G,v)          ▷ Mit Triangulierung der Lücken
12:                     remove_incident_points(S,v)
13:                     num_removed_points = num_removed_points + 1
14:                     simpli.merge_edges(v)
15:         while ( num_removed_points > 0 )
16:         ε = 2 * ε
17:     return (G)
```

## 5.3 Anwendungen

### 5.3.1 Konsistente und nicht-konsistente Level-views

Nach Durchlauf des Vereinfachungsalgorithmus 5.1 erhalten wir die Vereinfachungsstruktur, welche eine hierarchische Vereinfachung der Ebenenunterteilung darstellt. Uns interessiert bei gegebener Vereinfachungsstruktur und Fehlertoleranz $\varepsilon_{tol}$ aber nur ein Schnitt durch diesen Wald. Ein *Schnitt* durch einen Wald ist eine Menge an Knoten $C \subseteq V$, so dass jeder Blattknoten von genau einem Knoten aus $C$ durch herabsteigen im Wald erreicht werden kann. Einen Schnitt durch die Vereinfachunsstruktur bezeichnen wir als *Level-view*, siehe Abbildung 5.2.

Ein Level-view bezeichnen wir als *konsistent*, wenn es keine (Selbst-)überschneidungen und keine Verletzungen von Topologieeinschränkungen aufweist, also die Problemstellung erfüllt, und *nicht-konsistent*, wenn mindestens eine dieser Eigenschaften verletzt ist.

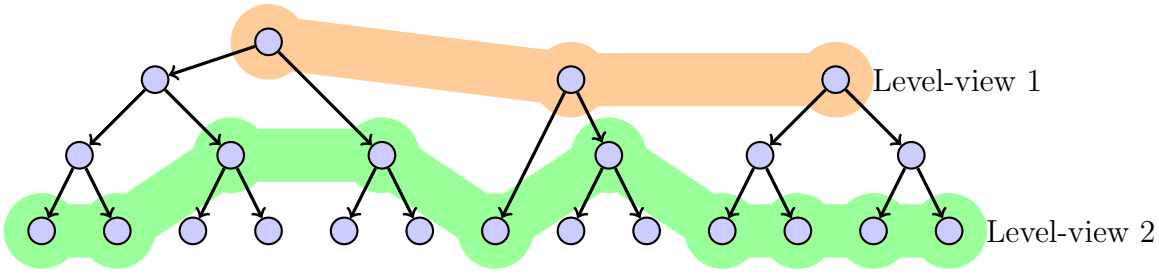
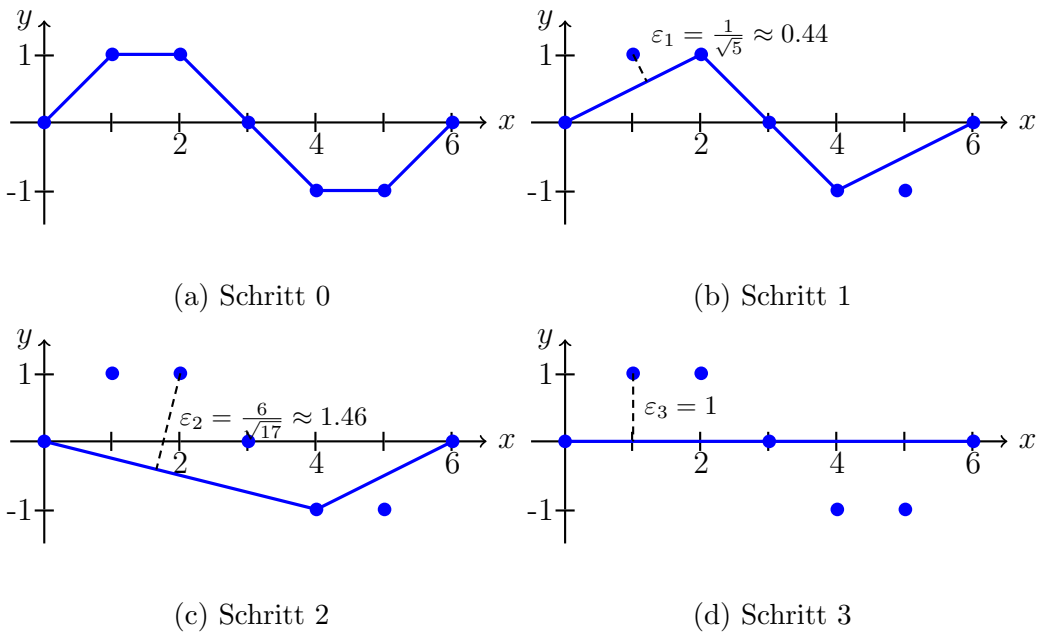



Abbildung 5.2: Zwei Level-views in einer Vereinfachungsstruktur.

Falls nicht anders erwähnt, ist ab jetzt die Fehlerschranke `eps` mit Formel (5.2) berechnet. Wie im vorangehenden Kapitel erwähnt, ist der `eps`-Wert nicht wachsend mit zunehmender Tiefe der Vereinfachungsstruktur. Es kann also ein Fall auftreten, in der für 3 gegebene Toleranzen $\varepsilon_1 < \varepsilon_2 < \varepsilon_3$ mit resultierenden Ebenenverfeinerungen $A_1$, $A_2$ und $A_3$ die gröbste Ebenenunterteilung $A_3$ die Fehlertoleranz $\varepsilon_1$ erfüllt. Abbildung 5.3 zeigt solch ein Fall.




(a) Schritt 0

(b) Schritt 1

(c) Schritt 2

(d) Schritt 3

Abbildung 5.3: Vereinfachung eines Segments mit nicht monoton wachsendem Abstand $\varepsilon$ zum Ausgangssegment.

Viel wahrscheinlicher ist es aber, dass nur ein Segment von $A_3$ die Fehlertoleranz $\varepsilon_1$ erfüllt. Hieran sieht man die Bedeutung der nicht-konsistenten Level-views. Während zu jeder Ebenenenunterteilung $A_{1/2/3}$ ein konsistentes Level-view erzeugt wurde, kann man einen weiteren Level-view definieren, welcher Kanten von sowohl $A_1$ als auch $A_3$ enthält und die Fehlertoleranz $\varepsilon_1$ erfüllt. Vorteil von solch einem Level-view ist, dass es gleich viele oder weniger Kanten beinhaltet als das zu $A_1$ zugeordnete Level-view. Es ist jedoch nicht garantiert, dass es konsistent ist.

### 5.3.2 Adaptive Ebenenverfeinerungen

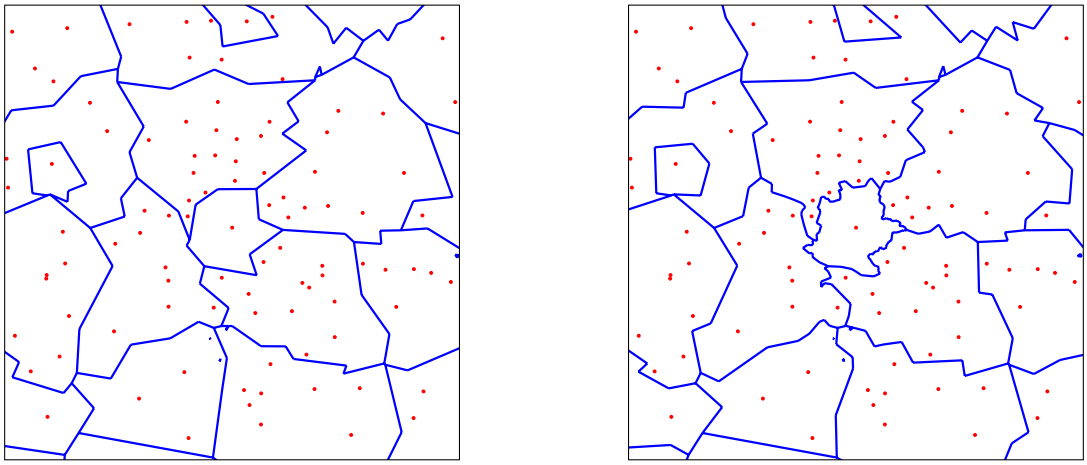
Bisher wurden die Level-views immer so gewählt, dass die Ebenenunterteilung überall die gleiche Auflösung / Verfeinerungsstufe ausweist. Möchte man aber auf einer Landkarte eine Route planen, so interessiert man sich in der Regel für den genauen Streckenverlauf entlang der Strecke. Die Details, welche weit weg von der Strecke liegen, sind jedoch für den Fahrtverlauf wenig relevant. Man benötigt also einen Level-view, der an ausgewählten Bereichen eine niedrige Fehlertoleranz hat als bei den übrigen Bereichen. Wir geben die Fehlertoleranz daher als Funktion in Abhängigkeit von der Position an

$$\varepsilon_{tol} : \mathbb{R}^2 \longrightarrow \mathbb{R}, p \longmapsto \varepsilon_{tol}(p).$$

Die Erzeugung der Level-views mit solch einer Fehlertoleranz läuft analog zu einer konstanten Fehlertoleranz ab, mit dem einzigen Unterschied, dass zu jeder Kante der jeweilige Toleranzwert ermittelt werden muss. Den Toleranzwert einer Kante haben wir als Minimum der beiden Toleranzwerte an den Eckpunkten der Kante gewählt.

$$\varepsilon_{tol}(e = (p_1, p_2)) := \min(\varepsilon_{tol}(p_1), \varepsilon_{tol}(p_2)) \tag{5.3}$$

Eine mit einer ortsabhängigen Fehlertoleranz erhaltene Ebenenunterteilung nennen wir *adaptiv*.



(a) $\varepsilon = 200000$.

(b) $\varepsilon(x) \in [0, 200000]$.

Abbildung 5.4: Beispiel einer nicht-adaptiven und adaptiven Ebenenunterteilung von Baden-Württemberg im Umkreis von Stuttgart.

# 6 Testrechnungen und Anwendungen

*„Irren ist menschlich. Aber wer richtigen Mist bauen will, braucht einen Computer!“*

-Dan Rather

In diesem Abschnitt zeigen wir Testrechnungen für einige OpenStreetMaps Datensätze [HW08], [osm]. Die im Abschnitt 4.2 vorgestellten Datenstrukturen und Algorithmen wurden in C++ mithilfe von CGAL [cgaa], einer Bibliothek für zuverlässige und effiziente geometrische Algorithmen, implementiert.

## 6.1 Implementierungsdetails

Alle in Kapitel 3, 4 und 5 beschriebenen Algorithmen und Berechnungen bauen auf der Annahme, dass Berechnungen auf den reellen Zahlen exakt sind. Diese Berechnungen liefern jedoch nicht die gleichen Ergebnisse auf den reellen Zahlen wie auf der endlichen Menge der Gleitkommazahlen. Bei Rechnungen auf Gleitkommazahlen treten aufgrund ihrer Darstellung mit einem Exponenten und einer Mantisse endlicher Länge Rundungsfehler auf. Bei geometrischen Algorithmen kann das zu unerwartetem Verhalten führen, beispielsweise erzeugen sie inkonsistente Ergebnisse, können abstürzen oder in Endlosschleifen hängen bleiben.

Für die Lösung des min-# Problems mithilfe von Triangulierungen kann es zum folgenden Fall kommen. Beim Prüfen, ob sich ein Punkt innerhalb eines Dreiecks befindet (siehe Abschnitt 4.2.2 und Abbildung 4.3) kann das Prädikat $p \in \triangle(p_1, p_2, p_3)$ als falsch berechnet werden, obwohl sich der Punkt $p$ im Dreieck berfindet. Das führt dazu, dass ein Segment vereinfacht wird und es zu einer Verletzung der Topologiebedingungen kommt, wenn $p$ eine Topologieeinschränkung ist, oder es kommt zu einer Überschneidung, wenn $p$ zu einem Segment der Ebenenunterteilung gehört. Der Lösungsansatz würde also ein inkonsistentes Ergebnis liefern und somit die Aufgabenstellung nicht erfüllen.

Um solche Prädikate exakt auszuwerten stellt CGAL Kernels bereit, die die exakte Berechnung von Prädikaten (z.B. $p \in \triangle(p_1, p_2, p_3)$) und exakte Konstruktionen (z.B. die Konstruktion eines Schwerpunkts, eines Dreiecks) ermöglichen.

```
typedef CGAL::Exact_predicates_exact_constructions_kernel K;
// ...
typedef CGAL::Triangle_2<K> Triangle_2;
```

Alle Operationen auf Eingeschränkten Delaunay Triangulierungen, also das Einfügen von Punkten, das Entfernen von Punkten und die anschließende Triangulierung des entstandenden Loches in der Triangulierung wurden mit built-in Methoden der Klasse `Constrained_Delaunay_triangulation_2` mit Hierarchie-Datenstruktur umgesetzt.

```
typedef CGAL::Constrained_Delaunay_triangulation_2<K, TDS,
        Itag> CDT;
typedef CGAL::Triangulation_hierarchy_2<CDT> CDTh_2;
```

Alle weiteren Methoden zur Ebenenvereinfachung wurden mit der C++ - STL-Bibliothek implementiert. Der komplette Quellcode zur Segmentvereinfachung mittels Eingeschränkten Delaunuay Triangulierungen befindet sich auf der beigefügten CD.

## 6.2 Testrechnungen

Zum Testen vom Algorithmus 5.1 verwenden wir zwei verschiedene Arten von Datensätzen. Zuerst werden OpenStreetMaps (OSM) Datensätze von Hamburger Stadtbezirken (kleiner Datensatz, Abschnitt 6.2.1) und von Bundeslandgrenzen, Regierungsbezirken und Landkreisen von Baden-Württemberg (größerer Datensatz, Abschnitt 6.2.2) verwendet. Als Topologieeinschränkungen werden die Positionen von Städten verschiedener Größen verwendet. Ein weiter Datensatz für Testrechnungen ist ein künstlicher Testdatensatz, der aus einem Segment besteht, welches spiralförmig verläuft und bei maximaler Vereinfachung auf ein Segment bestehend aus 2 Punkten vereinfacht werden kann. Dieser Datensatz kann mit beliebiger Anzahl an Vertices erzeugt werden und wird in Abschnitt 6.2.3 verwendet, um die Laufzeitkomplexität des vorgestellten Algorithmus abzuschätzen.

Alle folgenden Testrechnungen wurden auf einem Laptop ausgeführt mit 4 Intel i7-4600U Kernen (2.10 GHz) und dem Betriebssystem Ubuntu LTS 14.04, wobei für die Berechnungen jeweils nur ein Kern verwendet wurde. Der auf dem Betriebssystem installierte Compiler ist der gcc 4.8.4 Compiler. Als Flags für die Kompilierung wurde -frounding-math zur Unterstützung von CGAL und -02 zur Optimierung verwendet.

### 6.2.1 Testrechnungen auf einem kleinen Datensatz - Hamburg

Als erstes Testbeispiel wählen wir einen möglichst kleinen OSM-Datensatz um eine grobe Einschätzung für die benötigten Laufzeiten zu erhalten. Die bereitgestellten OSM-Datensätze enthalten für die Punkte der Ebenenunterteilung Längen- und Breitengrade, daher müsste man streng genommen die Daten in 3-dimensionale Koordinaten (auf einer Kugel) abbilden und anschließend auf eine Ebene projizieren. Die betrachteten Daten befinden sich jedoch nur auf einem sehr kleinen Teil der Kugel und werden dann als annähernd linear betrachtet. Für die $x$-Koordinate wird $\frac{2}{3}$ des Breitengrades und für die $y$-Koordinate wird der Längengrad gewählt und mit $10^7$ skaliert.

Für die ersten Testrechnungen haben wir 3 verschiedene Unterteilungen von Hamburg als Eingabe gewählt (Bundeslandgrenzen, Stadtbezirke und Stadtviertel). Die Größen der Vertexmenge, Kantenmenge und die Anzahl der Topologieeinschränkungen sind in Tabelle 6.1 aufgelistet.

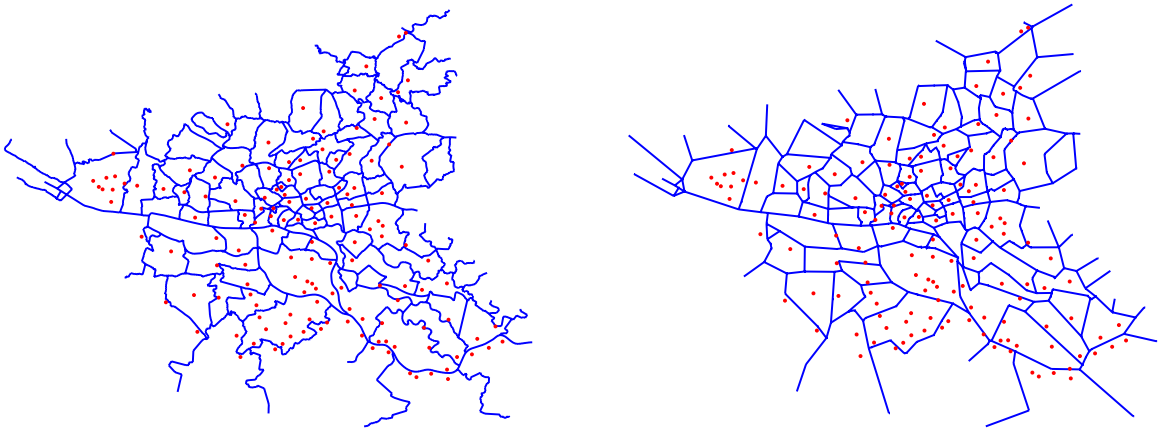
| Datensatz | $\|V\|$ | $\|E\|$ | $\|P\|$ | Kommentar |
|---|---|---|---|---|
| HH-4.graph | 3566 | 3679 | 145 | Bundesland |
| HH-9.graph | 5956 | 6234 | 145 | Stadtbezirke |
| HH-11.graph | 9839 | 10557 | 145 | Stadtviertel |

Tabelle 6.1: Eckdaten der Hamburger OSM-Datensätze.

Bei der Zeitmessung unterscheiden wir die Zeit für das Einlesen des Datensatzes und der Generierung der eingeschränkten Delaunay Triangulierung $t_{\text{init}}$, sowie die Zeit für die eigentliche Vereinfachung $t_{\text{simpl}}$. In Tabelle 6.2 sind die wesentlichen Laufzeiten und Ergebnisse der Vereinfachung aufgelistet, unter anderem der Anteil der nach der Vereinfachung verbleibenden Vertices $|V'|/|V|$. Weiterhin wurde in Tabelle 6.2 die Laufzeit für die Vereinfachung $t_{\text{simpl}}$ aufgeschlüsselt in Laufzeiten für Teilroutinen:

- $t_{\text{deg-2}}$, die Zeit zum Finden und Sortieren der Grad-2 Vertices,
- $t_{\text{check}}$, die Zeit zum Überprüfen, ob ein Grad-2 Punkt entfernt werden kann,
- $t_{\text{remove}}$, die Zeit zum Entfernen von Vertices,
- $t_{\text{build\_s}}$, die Zeit zum Aufbauen der Vereinfachungsdatenstruktur.

Das Ergebnis zur Vereinfachung der Stadtviertel ist in Abbildung 6.1 dargestellt.



(a) $\varepsilon = 0$. (b) $\varepsilon = 200000$.

Abbildung 6.1: Hamburger Stadtviertel.

Man sieht in Tabelle 6.2, dass selbst für die Unterteilung in Stadtviertel die Gesamtlaufzeit $t = t_{\text{init}} + t_{\text{simpl}}$ unter 0.3 Sekunden liegt.

| Datensatz | $\frac{\|V'\|}{\|V\|}$ | $t_{\text{init}}$[ms] | $t_{\text{simpl}}$[ms] | $\frac{t_{\text{deg-2}}}{t_{\text{simpl}}}$ | $\frac{t_{\text{check}}}{t_{\text{simpl}}}$ | $\frac{t_{\text{remove}}}{t_{\text{simpl}}}$ | $\frac{t_{\text{build\_s}}}{t_{\text{simpl}}}$ |
|---|---|---|---|---|---|---|---|
| HH-4.graph | 3.4% | 17 | 88 | 20.5% | 38.6% | 8.0% | 30.7% |
| HH-9.graph | 4.4% | 24 | 146 | 21.9% | 38.4% | 8.2% | 30.1% |
| HH-11.graph | 5.5% | 39 | 245 | 24.1% | 38.4% | 8.2% | 28.2% |

Tabelle 6.2: Vereinfachung von Hamburger OSM-Datensätzen. $\varepsilon = 200000$.

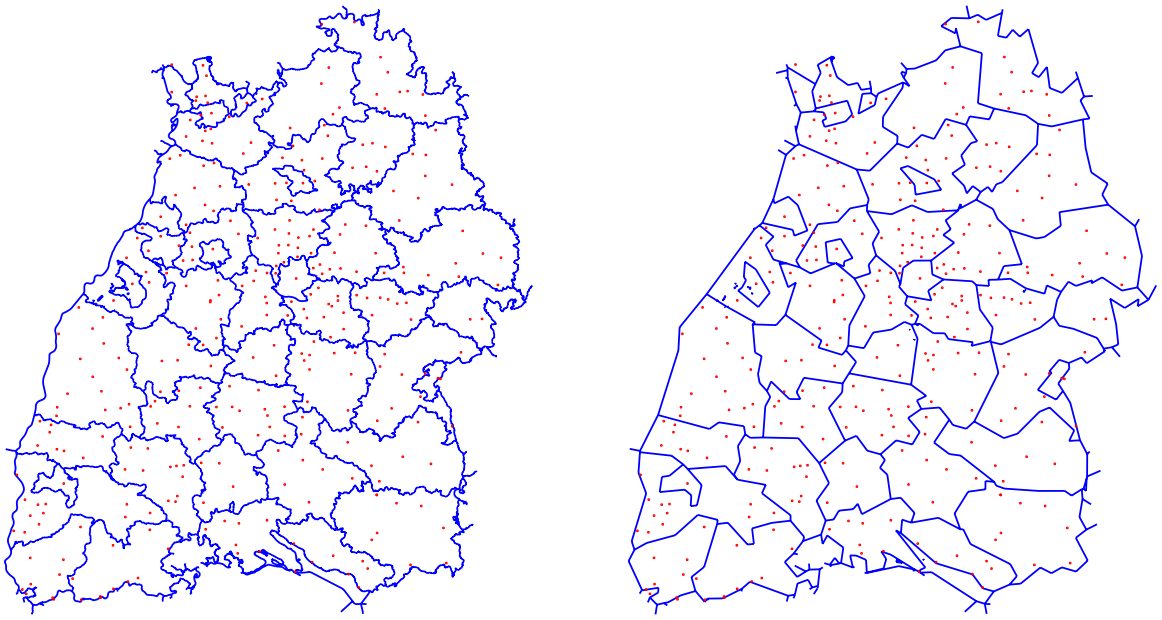
### 6.2.2 Testrechnungen auf einem größeren Datensatz - Baden-Württemberg

Da die Vereinfachungen der OSM-Daten von Hamburg im vorangehenden Abschnitt sehr schnell berechnet werden konnten, soll jetzt ein etwas größerer Datensatz getestet werden. Wir wählen diesmal 5 verschiedene Unterteilungen von Baden-Württemberg, diesmal die Bundeslandgrenzen, Regierungsbezirke, Landkreise, Verwaltungsgemeinschaften und Stadtgrenzen, siehe Tabelle 6.3.

| Datensatz | $\|V\|$ | $\|E\|$ | $\|P\|$ | Kommentar |
|---|---|---|---|---|
| bw_lvl4.graph | 35428 | 35427 | 332 | Bundesland |
| bw_lvl5.graph | 57055 | 57060 | 332 | Regierungsbezirke |
| bw_lvl6.graph | 133905 | 133953 | 332 | Landkreise |
| bw_lvl7.graph | 291307 | 291693 | 332 | Verwaltungsgemeinschaft |
| bw_lvl8.graph | 450940 | 452222 | 332 | Städte |

Tabelle 6.3: Eckdaten der Baden-Württemberg OSM-Datensätze.

Die Landkreise sind in Abbildung 6.2 dargestellt.



(a) $\varepsilon = 0$. (b) $\varepsilon = 200000$.

Abbildung 6.2: Baden-Württembergische Landkreise.

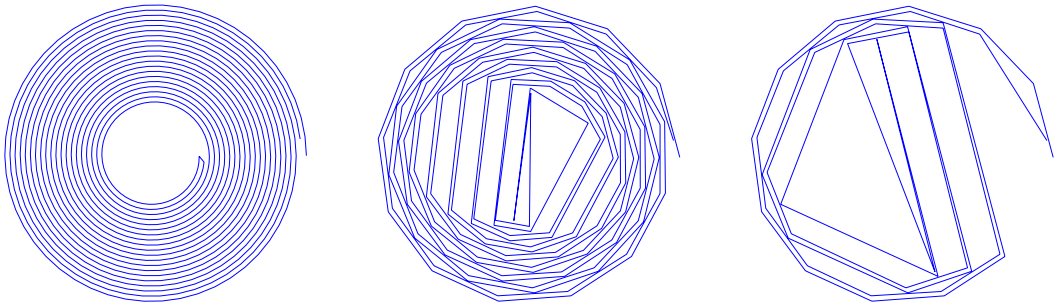
Für diese Datensätze sind die Laufzeiten diesmal signifikant größer. Für die Vereinfachung der feinsten Unterteilung in Städte und Verwaltungsgemeinschaften wurden etwa 22 Sekunden benötigt, siehe Tabelle 6.4. Bedenkt man jedoch, dass das min-# Problem ein NP-vollständiges Problem ist, dann liefert die Heuristik sehr zufriedenstellende Laufzeiten.

| Datensatz | $\frac{\lvert V'\rvert}{\lvert V\rvert}$ | $t_{\text{init}}$[ms] | $t_{\text{simpl}}$[ms] | $\frac{t_{\text{deg-2}}}{t_{\text{simpl}}}$ | $\frac{t_{\text{check}}}{t_{\text{simpl}}}$ | $\frac{t_{\text{remove}}}{t_{\text{simpl}}}$ | $\frac{t_{\text{build s}}}{t_{\text{simpl}}}$ |
|---|---|---|---|---|---|---|---|
| bw_lvl4.graph | 0.49% | 150 | 1510 | 20.9% | 46.1% | 5.1% | 27.4% |
| bw_lvl5.graph | 0.52% | 248 | 2475 | 22.8% | 45.3% | 5.1% | 26.4% |
| bw_lvl6.graph | 0.47% | 667 | 6242 | 28.9% | 41.0% | 4.8% | 24.9% |
| bw_lvl7.graph | 0.51% | 1407 | 13179 | 33.0% | 36.9% | 5.1% | 24.5% |
| bw_lvl8.graph | 0.76% | 2241 | 19688 | 36.0% | 34.3% | 5.3% | 23.7% |

Tabelle 6.4: Vereinfachung von Baden-Württemberg OSM-Datensätzen. $\varepsilon = 200000$.

### 6.2.3 Testrechnungen für einen konstruierten Datensatz

In diesem Abschnitt werden Testrechnungen auf einem Datensatz durchgeführt, welcher künstlich erstellt wird, so dass man die Laufzeit des Algorithmus in Abhängigkeit von der Eingabegröße $n = |V|$ untersuchen kann. Der Datensatz besteht aus einem einzigen Segment, das sich spiralförmig aufwindet, siehe Abbildung 6.3.



(a) $\varepsilon = 0$. (b) $\varepsilon = 250000$. (c) $\varepsilon = 500000$.

Abbildung 6.3: Testdatensatz bestehend aus einer Linie.

Im Datensatz sind weiterhin keine Topologieeinschränkungen gegeben, also $P = \emptyset$, daher ist für $\varepsilon \longrightarrow \infty$ die optimale Lösung des min-# Problems gegeben als Segment welches aus einer einzigen Kante besteht.

Die ermittelten Laufzeiten für verschiedene Eingabegrößen sind in Tabelle 6.5 aufgelistet.

| $n$ | $t_{\text{init}}$ | $t_{\text{simpl}}$ | EOC |
|---|---|---|---|
| 1000 | 6 | 95 | 0.95 |
| 2000 | 15 | 180 | 1.24 |
| 4000 | 56 | 406 | 1.19 |
| 8000 | 223 | 831 | 1.32 |
| 16000 | 809 | 1819 | 1.17 |
| 32000 | 2494 | 3420 | 1.28 |
| 64000 | 7073 | 7268 | 1.26 |
| 128000 | 18945 | 15517 | 1.21 |
| 256000 | 43501 | 36359 | |

Tabelle 6.5: Laufzeiten [ms] und EOCs.

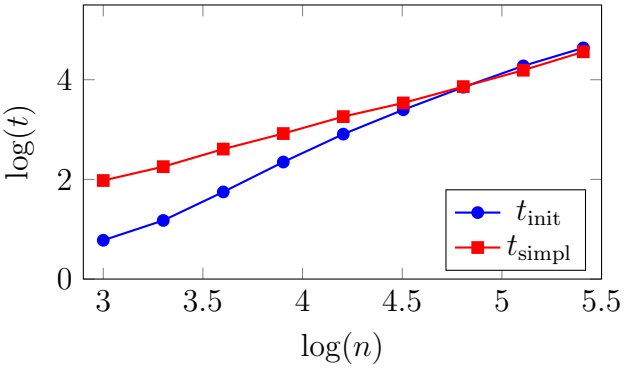



Abbildung 6.4: Zunahme der Laufzeiten.

Wenn man davon ausgeht, dass die Laufzeit polynomiell mit der Eingabegröße wächst, dann kann man die Laufzeit für große Eingabegrößen $n$ approximieren durch

$$t^{(n)} = t_{\text{init}}^{(n)} + t_{\text{init}}^{(n)} \approx C \cdot n^p.$$

Um den Grad $p$ der polynomiellen Laufzeit zu bestimmen, kann man Laufzeiten von zwei nacheinanderfolgenden Vereinfachungen vergleichen und erhält die Formel für den EOC (expected order of complexity)

$$\text{EOC} = p = \frac{\log(t^{(n_1)}/t^{(n_2)})}{\log(n_1/n_2)}.$$

In Tabelle 6.5 wurden die EOCs für das konstruierte Problem berechnet, die sich für große Werte von $n$ auf etwa 1.2 einpendeln. Für dieses Problem läuft der Algorithmus daher in etwas mehr als linearer Laufzeit.

Die Teillaufzeiten $t_{\text{deg-2}}$, $t_{\text{check}}$, $t_{\text{remove}}$ und $t_{\text{build\_s}}$ der Vereinfachung stehen in Tabelle 6.6. Anhand von Abbildung 6.5 sieht man, dass der prozentuale Anteil der Zeit $t_{\text{deg-2}}$ zum Finden und Sortieren der Grad-2 Vertices mit wachsender Eingabegröße zunimmt und somit asymptotisch die Laufzeit dominiert.

| $n$ | $t_{\text{deg-2}}$ | $t_{\text{check}}$ | $t_{\text{remove}}$ | $t_{\text{build\_s}}$ |
|---|---|---|---|---|
| 1000 | 2 | 49 | 3 | 40 |
| 2000 | 5 | 91 | 5 | 77 |
| 4000 | 10 | 207 | 9 | 179 |
| 8000 | 24 | 427 | 17 | 360 |
| 16000 | 58 | 943 | 33 | 781 |
| 32000 | 149 | 1758 | 71 | 1437 |
| 64000 | 384 | 3668 | 139 | 3069 |
| 128000 | 1390 | 7549 | 321 | 6239 |
| 256000 | 3851 | 17617 | 646 | 14195 |

Tabelle 6.6: Aufgeschlüsselte Laufzeiten [ms] für die Vereinfachung.

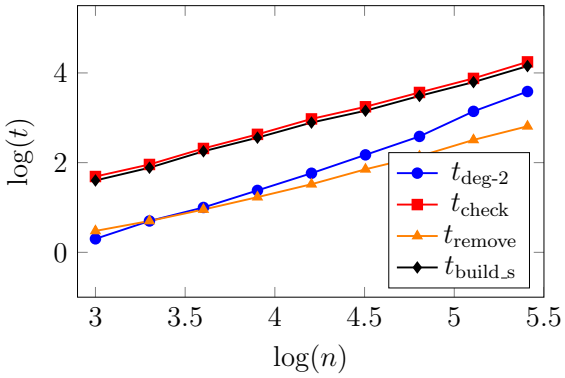



Abbildung 6.5: Zunahme der Laufzeiten für die Vereinfachung.

## 6.3 Diskussion

Die Testrechnungen für die spiralförmigen Daten aus Abschnitt 6.2.3, legen die Vermutung nahe, dass die implementierte Heuristik eine Laufzeit hat, die etwas mehr als linear ist, wahrscheinlich also $\mathcal{O}(n \log(n))$. Allerdings wurde die Komplexität nur mittels eines konstruierten Beispiels und ohne Topologieeinschränkungen geschätzt, daher muss man für genauere Aussagen den Algorithmus theoretisch auf seine Laufzeit untersuchen.

Anhand von Tabelle 6.5 und Abbildung 6.4 kann man weiterhin vermuten, dass die Laufzeit $t = t_{\text{init}} + t_{\text{simpl}}$ durch das Einlesen und das Konstrieren der Eingeschränkten Delaunay Triangulierung dominiert wird, da ab $n = 128000$ die Laufzeit für die Initialisierung $t_{\text{init}}$ die Laufzeit für die Vereinfachung $t_{\text{simpl}}$ überholt. Für die Hamburg- und Baden-Württemberg-Datensätze ist der große Anstieg von $t_{\text{init}}$ aber **nicht** zu

sehen. Eine mögliche Ursache könnte sein, dass die Punkte der Spirale in der Reihenfolge, in der sie entlang der Spirale auftauchen, eingefügt werden. Es ist jedoch bekannt, dass eine Delaunay-Triangulierung für eine **randomisierte** Eingabereihenfolge in erwartet $\mathcal{O}(n \log(n))$ konstruiert wird, siehe beispielsweise [BCKO08] Theorem 9.12. Die Einfügereihenfolge der Punkte der Spirale ist also wahrscheinlich ungünstig gewählt und sollte mit zufälligen Einfügereihenfolgen verglichen werden.

Geht man also davon aus, dass die Laufzeit durch die Zeit für die Vereinfachung dominiert wird, so wie in den Testrechnungen für die OSM-Datensätze in Abschnitt 6.2.1 und 6.2.2, dann dominiert asymptotisch die Zeit für das Finden und Sortieren der Grad-2 Vertices $t_{\text{deg-2}}$ (Abbildung 6.5, Tabellen 6.2 und 6.4). Um die Gesamtlaufzeit zu verbessern, muss man daher zuerst diese Teilroutine verbessern.

# 7 Fazit

## 7.1 Zusammenfassung

In dieser Arbeit wurde das Problem der Ebenenvereinfachung eingeführt zusammen mit drei Bedingungen: (1) die Einhaltung einer gegebenen Fehlerschranke, (2) der Ausschluss von Überschneidungen von Segmenten sowie (3) zusätzliche Topologieeinschränkungen. Zu diesem Problem wurden bekannte theoretische Ergebnisse zur Komplexität und zur Approximierbarkeit vorgestellt: Es handelte sich dabei um ein NP-vollständiges Problem, welches nicht bis auf einen Faktor $n^{1/5-\delta}$ für jedes $\delta > 0$ an die Optimallösung in Polynomzeit approximiert werden kann. (Natürlich unter der Annahme P $\neq$ NP.) In Kapitel 4 wurden bekannte Algorithmen zur Segmentvereinfachung sowie Heuristiken für Ebenenvereinfachung vorgestellt und eine neue Heuristik, welche zulässige Vereinfachungsschritte über Eingeschränkte Delaunay-Triangulierungen bestimmen kann. Diese Heuristik wurde im Laufe der Arbeit mithilfe von CGAL [cgaa] implementiert und getestet. Die Laufzeiten für die Vereinfachung von OSM-Datensätzen blieben für Eingaben mit bis zu 500000 Vertices unter einer halben Minute, wobei während der Vereinfachung 94.5-99.5% der Vertices entfernt wurden. Dabei war jedoch nicht bekannt, wie viele Vertices bei einer optimalen Ebenenvereinfachung übrig bleiben würden. Mit einem konstruierten Datensatz wurden zudem Laufzeiten abgeschätzt, die etwas mehr als linear in der Eingabegröße waren.

## 7.2 Ausblick

Als mögliche Fortsetzung dieser Arbeit könnte man die genaue Laufzeit der hier vorgestellten Heuristik theoretisch untersuchen. Ebenfalls kann man versuchen die Approximationsgüte abzuschätzen, entweder im Vergleich zu anderen Heuristiken oder zu einer unteren Schranke an die optimale Vereinfachung einer Instanz.

Interessant wären auch Testrechnungen auf weiteren Datensätzen, wie beispielsweise die Vereinfachung von Höhenlinien, da diese anfälliger für Überschneidungen sind. Möglicherweise treten bei solchen Datensätzen andere Laufzeiten auf, da häufiger iteriert werden muss.

Um die Ebenenvereinfachung zu beschleunigen, wäre auch eine Parallelisierung des Codes denkbar. Eine Möglichkeit wäre erst eine unabhängige Menge auf der Menge aller Grad-2 Knoten zu berechnen und anschließend auf dieser Menge das Prüfen der Entfernbarkeit, das Entfernen der Punkte und die Konstruktion der Vereinfachungsdatenstruktur parallel auszuführen. Bei sehr großen Datensätzen (z.B. Länder, Kontinente, Weltkarten) könnte dies signifikant schnellere Laufzeiten ergeben.

# Literatur


[AB02] A. Asteroth and C. Baier. *Einfuerung in Die Theoretische Informatik:*. Informatik - Pearson Studium. Pearson Education Deutschland GmbH, 2002.

[BCKO08] Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag TELOS, Santa Clara, CA, USA, 3rd ed. edition, 2008.

[BKS95] Marc de Berg, Marc van Kreveld, and Stefan Schirra. A new approach to subdivision simplification. In *ACSM/ASPRS Annual Convention & Exposition Technical Papers*, pages 79–88, Charlotte, North Carolina, USA, 1995. ACSM.

[CC92] W.S. Chan and F. Chin. Approximation of polygonal curves with minimum number of line segments. In Toshihide Ibaraki, Yasuyoshi Inagaki, Kazuo Iwama, Takao Nishizeki, and Masafumi Yamashita, editors, *Algorithms and Computation*, volume 650 of *Lecture Notes in Computer Science*, pages 378–387. Springer Berlin Heidelberg, 1992.

[cgaa] CGAL Computational Geometry Algorithms Library. `http://www.cgal.org`. Letzter Zugriff: 12.10.2015.

[CGAb] CGAL Triangulation_2 User Manual. `http://doc.cgal.org/latest/Triangulation\_2/index.html`. Letzter Zugriff: 12.10.2015.

[DP73] David H Douglas and Thomas K Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: The International Journal for Geographic Information and Geovisualization*, 10(2):112–122, 1973. doi:10.3138/FM57-6770-U75U-7727.

[EM01] Regina Estkowski and Joseph S. B. Mitchell. Simplifying a polygonal subdivision while keeping it simple. In *Proceedings of the Seventeenth Annual Symposium on Computational Geometry*, SCG '01, pages 40–49, New York, NY, USA, 2001. ACM.

[Est00] Regina Inez Estkowski. *Algorithms and Complexity Results for Three Problems in Applied Computational Geometry: Subdivision Simplification, Stripification of Triangulations, and Unions of Jordan Regions*. PhD thesis, Stony Brook, NY, USA, 2000. AAI9989025.

[GHMS91] Leonidas J. Guibas, John Hershberger, Joseph S. B. Mitchell, and Jack Snoeyink. Approximating polygons and subdivisions with minimum link paths. In *Proceedings of the 2Nd International Symposium on Algorithms*, ISA '91, pages 151–162, London, UK, UK, 1991. Springer-Verlag.

[HW08] Mordechai (Muki) Haklay and Patrick Weber. Openstreetmap: User-generated street maps. *IEEE Pervasive Computing*, 7(4):12–18, October 2008.

[II88] Hiroshi Imai and Masao Iri. Polygonal approximations of a curve – formulations and algorithms. In G. T. Toussaint, editor, *Computational Morphology*, pages 71–86. Elsevier Science, 1988.

[Kan94] Viggo Kann. Polynomially bounded minimization problems that are hard to approximate. *Nordic J. of Computing*, 1(3):317–331, September 1994.

[osm] OSM, OpenStreetMap. `http://www.openstreetmap.de`. Letzter Zugriff: 12.10.2015.

[Rud05] W. Rudin. *Analysis*. Oldenbourg, 2005.

[Sch09] M. Schubert. *Mathematik für Informatiker*. Vieweg+Teubner Verlag, 2009.

# Erklärung

Hiermit erkläre ich, dass ich die vorliegende Bachelorarbeit selbstständig angefertigt habe und sie nicht den Inhalt einer anderen Prüfung bildet. Es wurden nur die in der Arbeit ausdrücklich genannten Quellen und Hilfsmittel benutzt. Wörtlich oder sinngemäß übernommenes Gedankengut habe ich als solches kenntlich gemacht. Das eingereichte elektronische Exemplar stimmt mit den anderen Exemplaren überein.

---

Stuttgart, 15. Oktober 2015