

Institut für Formale Methoden der Informatik

Universität Stuttgart  
Universitätsstraße 38  
D–70569 Stuttgart

Bachelorarbeit

## **Points of Interest - Eine Suche nach adequaten Kartenbeschriftungen**

Lukas Baur

**Studiengang:** Informatik

**Prüfer/in:** Prof. S. Funke

**Betreuer/in:** F. Krumpe

**Beginn am:** 7. Dezember 2018

**Beendet am:** 13. Mai 2019



## **Kurzfassung**

Hilfreiche Kartenbeschriftungen tragen fundamental zur Benutzbarkeit von Kartendiensten bei. Dabei können sowohl Punkte als auch Flächen für den Nutzer potentiell interessante Orte (Points of Interest, POI) repräsentieren. Diese Arbeit untersucht den Prozess zur automatischen Point-of-Interest-Extraktion auf Eingabe roher Kartendaten der Plattform OpenStreetMap. Hierbei wird neben der Extraktion von Punkt-POIs ein besonderes Augenmerk auf Flächen gelegt: A priori kann von der Flächenform nicht auf einen Punkt-Repräsentanten geschlossen werden.

Dazu werden einleitend klassische Verfahren zur Mittelpunktbestimmung von Flächenobjekten diskutiert und deren Schwächen analysiert. Davon ausgehend widmen sich die folgenden Kapitel aktuellen Ansätzen, die eine Mittelpunkt-Lösung innerhalb der Fläche garantieren. Es fließen sowohl graphentheoretische Ansätze als auch morphologische Operationen in die Berechnungen ein. Ergänzt werden die theoretischen Überlegungen durch Beispiele aus eigener Implementierung und abschließenden Ergebnis-Evaluationen.

Basierend auf den behandelten Berechnungen wird ein Algorithmus zur Komposition der Einzelverfahren vorgestellt, der durch Klassifikation der Eingabe diese an möglichst optimale Methoden delegiert.

Abschließend werden Ansätze zur weiteren Verbesserung der Herangehensweise dargestellt, um für weiterführende Arbeiten in diesem Umfeld zu motivieren.



# Inhaltsverzeichnis

<b>1 Einleitung</b>	<b>13</b>
1.1 Thematischer Rahmen . . . . .	13
1.2 Aufgabenstellung . . . . .	16
1.3 Verwandte Arbeiten . . . . .	16
1.4 Beiträge dieser Arbeit . . . . .	17
<b>2 Beschriftungspipeline</b>	<b>19</b>
2.1 Datengrundlage . . . . .	19
2.2 POI-Extraction . . . . .	20
2.3 Labelpoint-Calculation . . . . .	22
2.4 POI-Classification . . . . .	23
2.5 Label-Generator . . . . .	23
2.6 Ausgabe . . . . .	24
<b>3 Berechnungskontext</b>	<b>25</b>
3.1 Definition Mittelpunkt . . . . .	25
3.2 Berechnungsgrundlage . . . . .	26
3.3 Topologische Umwandlung . . . . .	29
<b>4 Mittelpunktberechnung</b>	<b>33</b>
4.1 Gemittelte Summe . . . . .	33
4.2 Geometrischer Mittelpunkt . . . . .	35
4.3 Morphologische Erosion . . . . .	36
4.4 Point of Inaccessibility . . . . .	40
4.5 Graph-Skeleton-Centroid . . . . .	43
<b>5 Finale Lösung</b>	<b>57</b>
<b>6 Reflexion und Ausblick</b>	<b>61</b>
6.1 Zusammenfassung . . . . .	61
6.2 Problem-Reflexion . . . . .	61
6.3 Ausblick . . . . .	62
<b>Literaturverzeichnis</b>	<b>67</b>



# Abbildungsverzeichnis

1.1	Kartenvergleich mit und ohne Label . . . . .	14
1.2	Logo des OpenStreetMap Projekts . . . . .	15
1.3	Beispielhafte Verwendung der Typen Node, Way und Relation . . . . .	15
2.1	Die Beschriftungspipeline . . . . .	19
2.2	Schematische Darstellung der POI-Extractor-Komponente . . . . .	21
2.3	Schematische Darstellung der Labelpoint-Calculation-Komponente . . . . .	22
2.4	Schematische Darstellung der POI-Classification-Komponente . . . . .	23
2.5	Schematische Darstellung der Label-Generator-Komponente . . . . .	23
3.1	Kandidaten der Mittelpunktbestimmung . . . . .	26
3.2	Beispielhafte Polygon-Eingabe . . . . .	26
3.3	Eingabe-Fallbeispiele unter Berücksichtigung der Konsistenz-eigenschaften . . . . .	28
3.4	Schematische Interpretation einer Polygon-Relation . . . . .	29
3.5	Übersicht der Topologischen Umwandlung . . . . .	30
3.6	Einführendes Beispiel für komplexe Linienzugbildung . . . . .	30
3.7	Visualisierung eines Ausgabebaums . . . . .	31
3.8	Visualisierung der Ergebnisfamilien . . . . .	31
4.1	Negativbeispiel gemittelte Summe . . . . .	34
4.2	Pro- und Contra-Beispiel der Schwerpunkt-Variante . . . . .	36
4.3	Schaubild Erosion . . . . .	37
4.4	Grafiken zur Laufzeitbestimmung . . . . .	38
4.5	Worst-Case Rechnung morphologische Erosion . . . . .	38
4.6	Beispielhafte Erosionsergebnisse . . . . .	39
4.7	Erosionsergebnisse bei Variation von $\sigma$ . . . . .	39
4.8	Erweiterung der Erosion auf Polygonfamilienverbünde . . . . .	40
4.9	Beispiel Punktdefinition $P^\circ$ . . . . .	41
4.10	Positive Ergebnisse bei Einsatz des PoI-Verfahrens . . . . .	42
4.11	Exemplarische PoI-Gengenbeispiele . . . . .	42
4.12	Ergebnisse eigener Berechnungen des PoI . . . . .	43
4.13	Markierung der Feature Areas . . . . .	44
4.14	Beispielhafte Triangulierungen . . . . .	45
4.15	Beispielhafte AJJ-Skelettierung . . . . .	46
4.16	Gegenüberstellung verschiedener Skelette . . . . .	48
4.17	Beschriftungsübersicht Skeletongraph . . . . .	49
4.18	Darstellung einer Betweenness-Centrality-Komponente . . . . .	50
4.19	Tree-Shrink-Algorithmus an einem Beispiel . . . . .	52
4.20	Vergleich Skelettarten . . . . .	54
4.21	Ergebnisse Skeleton Centrality . . . . .	55

5.1	AJJ-Skelettierung auf abstrahierten Polygonkonturen . . . . .	58
5.2	Punkte-Reduktion . . . . .	58
6.1	Schwierige Eingaben . . . . .	62
6.2	Reparatur von Polygoneingaben . . . . .	63
6.3	Punkteverteilung innerhalb des Skeletts . . . . .	64
6.4	Skelett mit linearer Punkteanzahl zur Größe . . . . .	64
6.5	Anregungen zur Polygon-Detailreduktion . . . . .	65

# **Verzeichnis der Listings**

2.1	Beispielhafte Darstellung eines osm-Files . . . . .	20
2.2	Beispielhafte Darstellung des GeoJSON Formats . . . . .	21



# **Verzeichnis der Algorithmen**

4.1	Tree-Shrink-Algorithmus . . . . .	53
5.1	Meta-Ebene der Mittelpunkt-Berechnung . . . . .	59
5.2	Klassifizierung und Delegation der Polygonfamilien . . . . .	60



# 1 Einleitung

Die folgenden Abschnitte ebnen den Einstieg in das Themengebiet um das Verständnis für die Problemstellung zu fördern. Das Kapitel dient weiterhin dazu, dem Leser eine grob strukturierte Idee des behandelten Lösungsansatzes zu vermitteln.

## 1.1 Thematischer Rahmen

Eine für den Benutzer zufriedenstellende Kartenbeschriftung stellt ein entscheidendes Qualitätsmerkmal der Darstellung von (Land-)karten dar. Mit der Entwicklung von interaktiven Karten werden ganz neue Bedürfnisse an Kartendarstellungen gestellt: Nebst Verschieben muss der Kartendienst ein Zoomen, in Einzelanwendungen auch ein Rotieren der Karte anbieten können. Während im Gegensatz zu statischen Karten das Erscheinungsbild auf Grundlage des aktuellen Kartenausschnitt optimiert wird, sind für das dynamische Pendant weitere Anforderungen notwendig, die bereits von Been u.a. in [BDY06] definiert wurden:

1. Während eines monotonen Zoomens sollten Beschriftungslabel nicht mehr als einmal erscheinen und verschwinden.
2. Label sollten bei Karteninteraktion ihre Größe und Position nicht abrupt ändern.
3. Während dem Rotieren oder Verschieben der Karte sollten Label nicht verschwinden oder auftauchen. Eine Ausnahme stellen Label dar, die in den Bildbereich rein- oder rausgeschoben werden.
4. Die Labelauswahl und -positionierung ist ausschließlich von dem Bildbereich und dem Zoomlevel abhängig, also insbesondere nicht von der Interaktionshistorie.

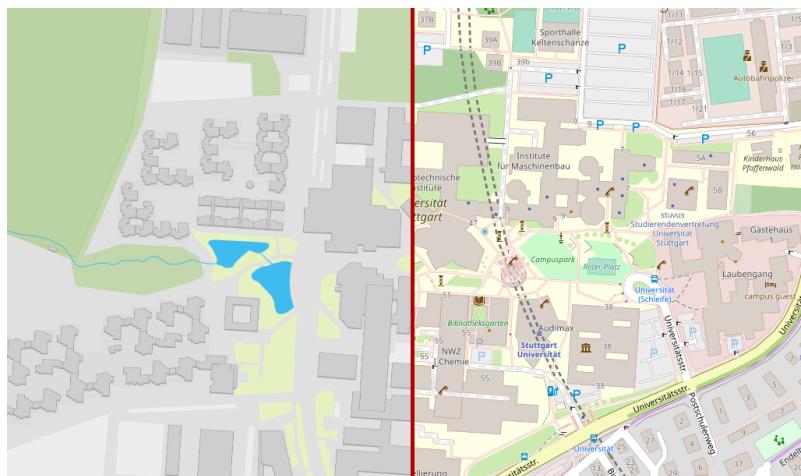
Außerdem definieren wir analog zu [Kru18b] eine zusätzliche Rahmenbedingung, auf die wir uns bei der Berechnung der Label-Hierarchie in Abschnitt 2.5 stützen:

5. Während dem Zoomen verschwindet ein Label ausschließlich, wenn es im Konflikt mit einem gleichwertigen oder wichtigeren Label steht.

Um die Informationsdarstellung einer Karte zu erleichtern, verwenden wir sogenannte *Points of Interest* (POI, POIs), die für den Benutzer relevante Kartenmerkmale darstellen und oft durch ein zugehöriges Icon repräsentiert werden[Ope]. Beispiele für POIs sind Krankenhäuser, Universitäten, Theater, Parkplätze und Bahnhöfe, während Straßen, Meere und Grenzen hauptsächlich die Navigation unterstützen und damit nicht als POIs fungieren.

## 1 Einleitung

---



**Abbildung 1.1:** Ein Vergleich von Kartendarstellungen der Universität Stuttgart: Der rote Strich trennt die Darstellung ohne Beschriftung von der Karte mit POIs (links). Die linke Hälfte wirkt wenig ansprechend und informativ

Quelle: <https://mc.bbbike.org> (15.03.18), Kartendaten ©2019 OpenStreetMap.org

In der Literatur wird zusätzlich der sinnverwandte Begriff *Place of Interest* verwendet, der wiederum durch einen Point of Interest dargestellt wird[MJ18]. Wie wir zeigen werden, ist die Abbildung von einer Fläche auf einen einzelnen Punkt im Allgemeinen nicht offensichtlich. Wir werden uns später in Kapitel 3 darauf konzentrieren.

Da dynamisch berechnete Daten zur Darstellung sinnvoller POI-Label ein wichtiges Know-How führender Kartendienste darstellt, bleiben diese ein gut geschütztes Eigentum.

Im Rahmen dieser Arbeit stützen wir uns auf Daten, die aus dem OpenStreetMap Projekt stammen. Um die Berechnungsschritte in Abschnitt 2.2 nachvollziehen zu können, arbeiten wir im folgenden Abschnitt wesentliche Kernkomponenten des Projekts heraus.

### OpenStreetMap Projekt

Das 2004 gegründete *OpenStreetMap* (OSM) Projekt[Wik19c] setzt sich das Ziel, eine freie Weltkarte zu erschaffen. OSM unterscheidet sich zu anderen Kartendiensten dadurch, dass Daten selbst erhoben und hinzugefügt werden können und die Gesamtheit aller Daten lizenzkostenfrei verwendet und weiterverarbeitet werden darf[Ope19]. Aktuell wird OpenStreetMap von 5,1 Millionen Nutzern verwendet, davon eine Millionen die aktiv Daten hinzufügen<sup>1</sup>. Weiter besteht die Datenbank aus fünf Milliarden Kartenknoten[Wik19d].

---

<sup>1</sup>Stand März 2019

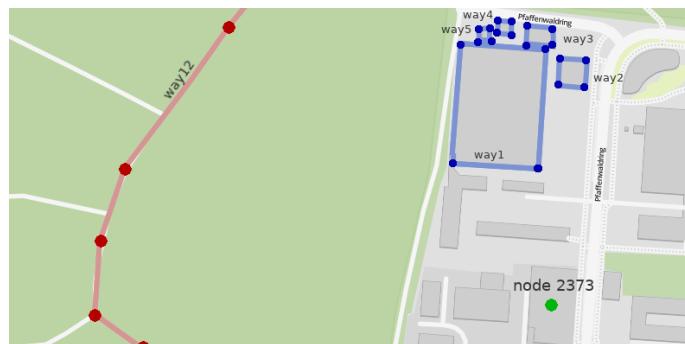

**Abbildung 1.2:** Das OpenStreetMap Logo

 Quelle: [www.openstreetmap.de](http://www.openstreetmap.de) (15.03.18)

Das gesamte Markierungskonzept von OSM unterscheidet die drei Primitive *Node*, *Way* und *Relation*, sowie einer Beschreibung durch *Tags* (siehe Abbildung 1.3). Ein Node repräsentiert einen Weltkartenpunkt mit zugehörigem Längen- und Breitengrad. Ein Way wird durch eine Menge von Punkten dargestellt, die in einer eindeutigen Reihenfolge vorliegen. Sind Start- und Endpunkt identisch spricht man von einem geschlossenen Weg. Die Relation-Datenstruktur verbindet mehrere Nodes und Ways miteinander. Wichtig dabei ist, dass die gruppierten Elemente geographisch nahe beieinander liegen. Eine Relation, die alle Bushaltestellen in Europa logisch zusammenfasst, wäre beispielsweise keine sinnvolle Relation, während eine Inselgruppe bestehend aus Einzelteilen hingegen sinnvollerweise durch eine Relation abgespeichert wird. Eine Relation sollte nicht als logisches Klassifizierungsschema dienen [Wik17a].

Zu jedem Primitiv lässt sich neben den notwendigen Eigenschaften wie osm-ID, Breiten- und Längenkoordinaten (bei Nodes) oder Verlinkungen auf andere Primitive (bei Ways und Relations) eine Liste an Tags anhängen [Wik19e]. Ein Tag hat die Form <Key> = <Value>. Mithilfe von Tags lassen sich weitere Metainformationen über ein Objekt abbilden, beispielsweise könnte ein geschlossener Way mit den Attributen 'building' = 'yes', 'amenity' = 'university', 'name' = 'Universität Stuttgart Fakultät 5' ein Campusgebäude beschreiben.

Als Schnittstelle stehen neben .osm-Files, die die obigen Konzepte in xml-Format umsetzen, ebenfalls .osm.pbf-Files zur Verfügung. Wir werden in Abschnitt 2.1 auf die notwendigen Implementierungsdetails eingehen.


**Abbildung 1.3:** Beispielhafte Darstellung von Nodes (rund), Ways (z.B. rot) und einer Relation (blau) bestehend aus fünf geschlossenen Wegen

 Quelle: [www.openstreetmap.de](http://www.openstreetmap.de) (15.03.18)

## 1.2 Aufgabenstellung

Auf Grundlage der vom OpenStreetMap Projekt bereitgestellten Daten solle ein sinnvoller Beschriftungssatz extrahiert werden. Eine Beschriftung wird durch folgende Attribute charakterisiert:

- einen Ankerpunkt als  $(x, y)$  Tupel
- eine Priorität
- eine Beschriftungs-Fontgröße

Ein Ankerpunkt stellt einen Punkt dar, der das Objekt möglichst passend repräsentiert. Den Begriff *passend* definieren wir im Abschnitt 3.1. Soll ein Label für einen einzelnen Node gefunden werden, so verwenden wir dessen Koordinaten. Es werden im Folgenden insbesondere Gebietsobjekte und die Bestimmung deren Ankerpunkte thematisiert.

Der Karten-Bildbereich lässt nur eine bestimmte Anzahl an Label zu. Daher muss zu jedem Bildsetting eine Teilmenge an überschneidungsfreien Label gefunden werden. Stehen zwei Label in Konflikt, wird das jeweils höher priorisierte der beiden auf der Karte gezeigt. Für  $k$  Label wiederholt man die Idee entsprechend. Daher muss die Arbeit eine Priorisierung hervorbringen, die ein späteres Vergleichen der Label ermöglicht.

Neben der Labelposition spielt auch dessen Größe eine entscheidende Rolle bei der visuellen Wahrnehmung einer Karte[Imh75]. Hier soll die Größe dynamisch aus dem Objekt generiert werden.

### Fokus

Die dynamische Extraktion von relevanten Points of Interest stellt ein vielseitig erforschtes Gebiet dar[BLM+14; MK08; NRH12; ZC15]. Aufbauend auf der Arbeit [Kru18b] konzentriert sich diese Arbeit auf die Generierung von diesbezüglich passenden Eingabedaten. Wie in Kapitel 2 beschrieben, bildet unserer Beschriftungsdatensatz die Eingabe der Label-Generierung.

Als Erweiterung zur Punktbestimmung in [Kru18b] erarbeiten wir verschiedene Varianten zur Mittelpunktbestimmung. Ein Fokus dieser Arbeit liegt auf dem Vergleichen der Verfahren anhand von Stabilität, Laufzeit und Einsetzbarkeit in Bezug auf die gegebenen Datenmengen.

## 1.3 Verwandte Arbeiten

Grundlage der Pipeline-Struktur ist die Arbeit von [Kru18b], die insbesondere die Vorberechnung der Label-Zoom-Schnittpunkte thematisiert. Sowohl in [RHB15] als auch in [NRH12] wird der Prozess der Objekt-Extraktion auf Grundlage der OSM-Daten beschrieben. Die Arbeit von [BLM+14] greift neben POIs auch auf Datensätze zur Landnutzung zu.

Zur Mittelpunktberechnung gibt es zunächst klassische Ansätze wie das Minimieren der Distanzfunktion eines Eingabepunktes zu allen anderen Punkten. Wir benennen dazu exemplarisch die Pionierarbeiten [LPS+88] und [AT87], die das formalisierte Problem analytisch aufarbeiten und jeweils einen Algorithmus vorschlagen, sowie einen Beitrag aktueller Forschung [ABB+16], in dem ein Linearzeitalgorithmus vorgestellt wird.

Eine aktuelle Veröffentlichung zur Berechnung von Mittelpunkten basierend auf einer Graph-Struktur ist [LA18]. In diesem Zusammenhang verweisen wir auf die Arbeit [AAJ15], in der untersucht wird, wie der entstehende Graph sich unter verschiedenen Triangulierungen verhält.

Als weiteren Ansatz zur Bestimmung der Mitte nennen wir [GL07], die in ihrer Arbeit einen Punkt mit maximaler Distanz in jegliche Richtung bestimmen.

## 1.4 Beiträge dieser Arbeit

In unserer Arbeit stellen wir den Berechnungsprozess dar, der als Ausgabe beschriftete Label einer Eingabe aus OpenStreetMap-Daten liefert. Konkret heißt das, es werden aus dem OSM-Datensatz Punkte extrahiert, die ein Label erhalten werden. Dabei sind sowohl Punkte als auch Flächen zu berücksichtigen.

Um einen geeigneten Mittelpunkt zu erhalten, konzentrieren wir uns auf entsprechende Methoden zur Kandidaten-Bestimmung. Es wird ein Vergleich zwischen grundlegenden geometrischen Verfahren und Methoden aus der Graphenanalyse aufgestellt, indem wir jede der beschriebenen Methoden implementieren und auf einen Testdatensatz anwenden. Insbesondere greifen wir auf aktuelle Forschung der Area-Feature-Bestimmung zurück [LA18]. In der Ergebnisdiskussion wird die Qualität der Mittelpunktergebnisse den Laufzeiten gegenübergesetzt.

Final werden die Verfahren zu einem Gesamt-Algorithmus vereinigt. So kann die Optimalität der einzelnen Verfahren für bestimmte Anwendungsfälle ausgenutzt werden. Wir stellen zudem die Detailreduktion als zusätzliches Hilfsmittel zur Berechnung signifikanter Eckpunkte vor.

Abschließend werden Ideen zur Erweiterung der Verfahrenskomposition eingebracht, um für verbesserte Sonderfallabdeckungen zu motivieren.

## Aufbau der Arbeit

Das Einsatzgebiet unseres finalen Algorithmus stellt die Labelberechnung auf Kartendaten dar. Der abstrakten Ablauf zur Generierung einer Kartenbeschriftung wird im nächsten Kapitel vorgestellt. Dazu durchlaufen wir sequentiell die Schritte von Datenaufbereitung, und –filterung, Mittelpunktberechnung und Bestimmung der Hierarchie, nach deren Regel sich Label auslösen.

Kapitel drei und vier bilden als Einheit den Kern der Arbeit. Während im ersten Teil grundlegende Definition eingeführt werden, betrachtet Kapitel 4 die Berechnungen zur Mittelpunktberechnung im Detail.

Kapitel 5 fasst die Ergebnisse zu einem finalen Algorithmus zusammen, bevor wir im letzten Abschnitt die Problemstellung reflektieren und einen Ausblick für zukünftige Arbeiten und Verbesserungen geben.



## 2 Beschriftungspipeline

Die Berechnung der Label lässt sich in drei grobe Schritte einteilen, die wir in Form einer Pipeline sequentiell ausführen. Die folgenden Abschnitte durchlaufen die in Abbildung 2.1 dargestellte Pipeline der Reihe nach. Sie baut grundlegend auf [Kru18a] auf und erweitert das Konzept um Komponenten.

Um einen Überblick über das Kapitel zu erhalten, folgt zuvor eine abstrakte Übersicht der Komponenten:

Die in Abschnitt 2.1 beschriebene Eingabe wird mithilfe des POI Extractors nach Points of Interest durchsucht. Im gleichen Schritt werden sinnvolle Label-Ankerpunkte bestimmt.

Auf Eingabe der entstandenen Liste an POIs und einer benutzerspezifischen Konfigurationsdatei berechnet der POI Classifier für jeden POI ein Ranking-Score.

Gemäß der Position und dem Ranking-Score kann im Label-Generator für jedes Label der Zeitpunkt bestimmt werden, an dem es durch ein lokal benachbartes Label bei fortschreitendem Zoomen ausgelöscht und überdeckt wird. Dieser Zeitpunkt wird in das Label gespeichert, um eine Auslöschung in Echtzeit realisierten zu können. Die Ausgabe bildet das Ende der Pipeline.

### 2.1 Datengrundlage

Das in Abschnitt 1.1 vorgestellte osm-Format eignet sich besonders gut, um Daten manuell zu analysieren und darin zu navigieren. Die zugrunde liegende Struktur ist xml und bildet die Primitive sinngerecht ab [Wik17b]. Ein Beispiel lässt sich dem Listing 2.1 entnehmen.

Für die maschinelle Verarbeitung ist die xml-basierte Variante allerdings ungeeignet. Aus Gründen der Performanz und des Speicherbedarfs verwenden wir bei der Implementierung Dateien, die .osm.pbf formatiert sind: Das Schreiben von .pbf-codierten Geo-Informationen ist, verglichen mit

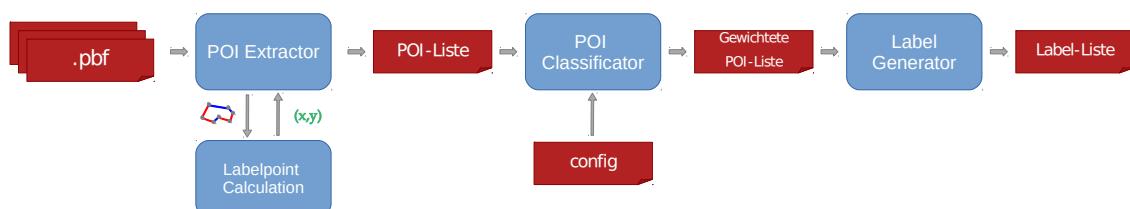


Abbildung 2.1: Schematische Darstellung der Beschriftungspipeline

## 2 Beschriftungspipeline

---

### Listing 2.1 Beispielhafte OSM-Einträge der Primitive Node, Way und Relation

---

```
<node id="283500188" lat="54.3281023" lon="-4.3905883" version="4" timestamp="2012-04-19T03:07:17Z" changeset="0">
  <tag k="name" v="The Parish Church of S. Olave. North Ramsey"/>
  <tag k="amenity" v="place_of_worship"/>
  <tag k="building" v="church"/>
  <tag k="religion" v="christian"/>
</node>

<way id="25985915" version="5" timestamp="2013-04-02T06:04:20Z" changeset="0">
  <nd ref="283486330"/>
  <nd ref="746330946"/>
  <nd ref="826954675"/>
  <nd ref="826954461"/>
  <nd ref="746330947"/>
  <nd ref="2244295314"/>
  <tag k="highway" v="footway"/>
  <tag k="surface" v="paved"/>
</way>

<relation id="7623634" version="9" timestamp="2017-10-29T01:25:30Z" changeset="0">
  <member type="way" ref="25986267" role="" />
  <member type="way" ref="170522889" role="" />
  <member type="relation" ref="7687840" role="forward" />
  <member type="relation" ref="7687840" role="backward" />
  <tag k="network" v="Isle of Man Public Transport" />
  <tag k="operator" v="Bus Vannin" />
  <tag k="route" v="bus" />
  <tag k="type" v="route" />
</relation>
```

---

den XML-codierten, bis zu fünf mal schneller bei halbem Speicherbedarf [Wik18]. Ein möglicher Datenexportanbieter ist die Plattform Geofabrik<sup>1</sup>, hier können insbesondere Subdatensätze wie Länder oder Regierungsbezirke geladen werden.

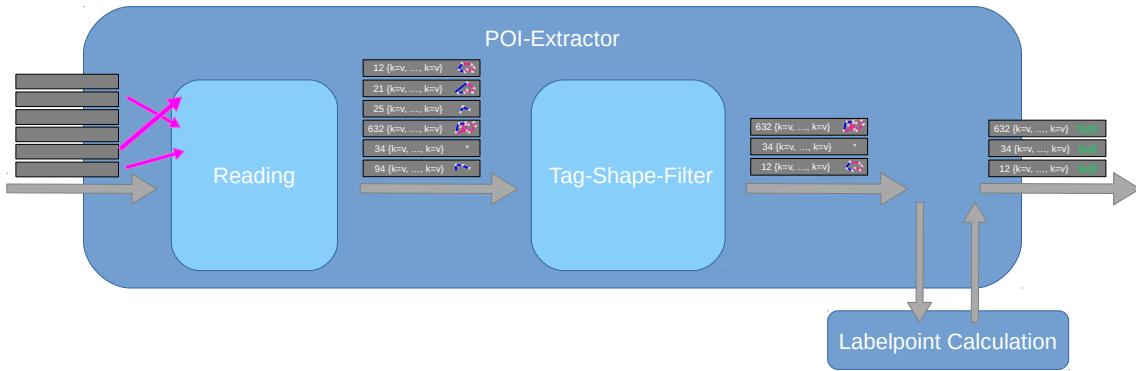
Eine Datei ist blockweise angeordnet. Jeder Block lässt sich unabhängig von den anderen decodieren, was insbesondere durch paralleles Auslesen zu Speedup beiträgt.

## 2.2 POI-Extraction

Nicht alle in OpenStreetMap hinterlegten Daten sind für unsere Anwendungen relevant. Der POI Extractor (Abbildung 2.2) filtert relevante Points of Interest heraus, für die wir später ein Label auf der Karte spendieren möchten. Dazu extrahieren wir die vorläufigen Label in vier Phasen.

---

<sup>1</sup><http://download.geofabrik.de/>



**Abbildung 2.2:** Schematische Darstellung der Komponente POI-Extractor

Im ersten Schritt werden alle zur Verfügung stehenden Daten eingelesen: Pro .pbf-Datei lesen parallele Threads aus den Datenblöcken und übergeben die gelesenen Informationen an die Filter-Komponente.

Dort wird das jeweilige Objekt gegen eine Liste von Bedingungen geprüft: Nebst Konsistent der Einträge wird vor allem anhand der Tags aussortiert. Wir übernehmen nur Label, die einen Namen besitzen, sowie ein Key-Value-Paar aufweisen, dessen Key als *Feature Point* verwendet werden kann. Map Features sind von der OSM-Community einheitlich festgelegte Tag-Vorschläge für wiederkehrende Objekte wie Flughäfen, öffentliche Einrichtungen, Freizeitmöglichkeiten, Straßenelemente, Vorkommen in der Natur, Einkaufsmöglichkeiten und vieles Weiteres. Eine ausführliche Auflistung wichtiger Key-Value-Paare ist auf der zugehörigen Wiki-Seite<sup>2</sup> zu finden.

**Listing 2.2** Beispielhafte Darstellung des GeoJSON Formats anhand des Knotens aus Listing 2.1

```

1  {
2    "type": "Feature",
3    "id": 283500188,
4    "geometry": {
5      "type": "Point",
6      "coordinates": [-4.3905883, 54.3281023]
7    },
8    "properties": {
9      "amenity": "place_of_worship",
10     "building": "church",
11     "religion": "christian"
12   },
13     "title": "The Parish Church of S. Olave. North Ramsey"
14   }
15

```

<sup>2</sup>[https://wiki.openstreetmap.org/wiki/Map\\_Features](https://wiki.openstreetmap.org/wiki/Map_Features)

## 2 Beschriftungspipeline

---

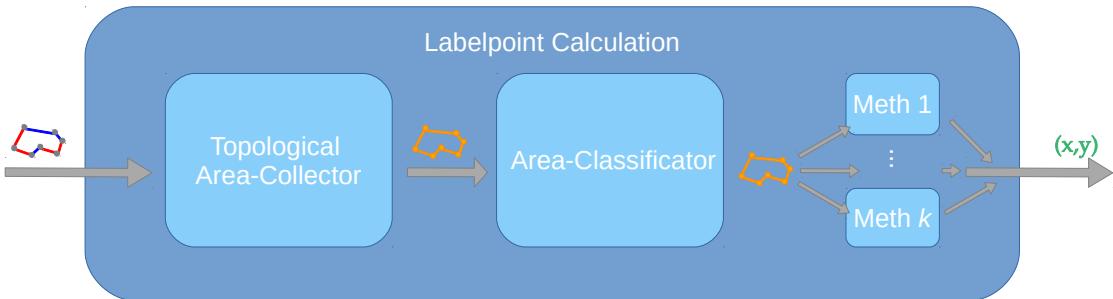
Die Ausgaben des POI Extractors bilden die Attribute Name und OSM-ID, Label-Koordinate, sowie eine Unterscheidung, ob es sich um ein Flächen- oder Punktlabel handelt. Zur Klassifizierung wird zudem die Tag-Liste angehängt. All diese Informationen werden als *GeoJSON* formatiert.

GeoJSON ist ein offener Standard für die Darstellung geografischer Features und basiert auf der Beschreibungssprache JSON[BDD+16]. Eine beispielhafte Darstellung ist Listing 2.2 zu entnehmen.

Handelt es sich bei einem Feature um ein Node, so werden dessen Koordinaten übernommen. Bei Eingabe eines beliebig komplexen zusammengesetzten Geometrie-Objektes wird die Struktur an die Labelpoint Calculation Komponente übergeben, die dynamisch einen Ankerpunkt für die Beschriftung ermittelt.

### 2.3 Labelpoint-Calculation

Wie wir in Abschnitt 3.1 zeigen werden, ist die Bestimmung eines repräsentativen Punktes innerhalb des Polygons keineswegs trivial. Für diese komplexe Reduktion ist Labelpoint Calculation (Abbildung 2.3) zuständig, die ihre Funktionalität in drei Stufen umsetzt.



**Abbildung 2.3:** Schematische Darstellung der Komponente Labelpoint Calculation

Dass die Eingabemenge nicht direkt als Berechnungsgrundlage verwendet werden kann, zeigen wir in Abschnitt 3.2. Zuvor müssen wir eine Format-Wandlung durchführen, die den ersten Schritt der Berechnung darstellt. Auf die Umwandlungsdetails werden wir in Abschnitt 3.3 eingehen.

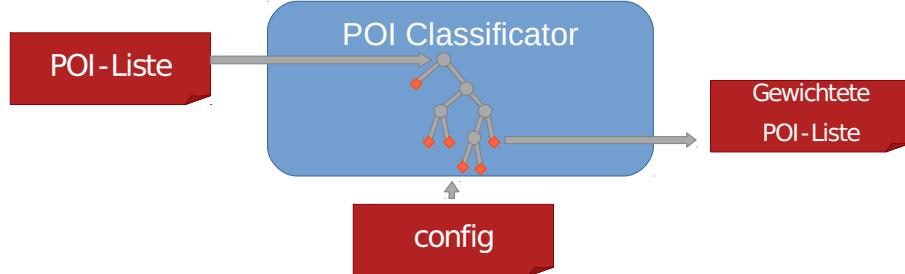
Im zweiten Teil untersuchen wir die konvertierte Polygonpunktmenge um eine optimale Berechnungsstrategie zu entwickeln. Für ein segmentiertes degeneriertes Polygon mit Löchern müssen angepasstere Verfahren zum Einsatz kommen, als bei einem ungelochten konvexen. Für genaue Spezifikationen verweisen wir auf Kapitel 5.

In Kapitel 4 beschäftigen wir uns mit verschiedenen Verfahren zur Mittelpunktbestimmung. Gemäß dem Ergebnis der Polygontyp-Analyse der vorherigen Schritte übergeben wir die Polygon-Datenstruktur final an die jeweilige Berechnungskomponente. Deren Ergebnis liefert unsere Lösung.

Die vorgestellte Struktur bietet den Vorteil, beliebige andere Berechnungsstrategien einbinden zu können, ohne die Grobstruktur dadurch zu verletzen. Es sind hierzu lediglich Änderungen an dem Polygontyp-Analyse-Entscheidungsbaum vorzunehmen.

## 2.4 POI-Classification

Die Klassifizierung der einzelnen POIs schließt signifikant an der Vorarbeit von [Kru18a] an. Eine grobe Idee der Funktionsweise ist der Abbildung 2.4 zu entnehmen.

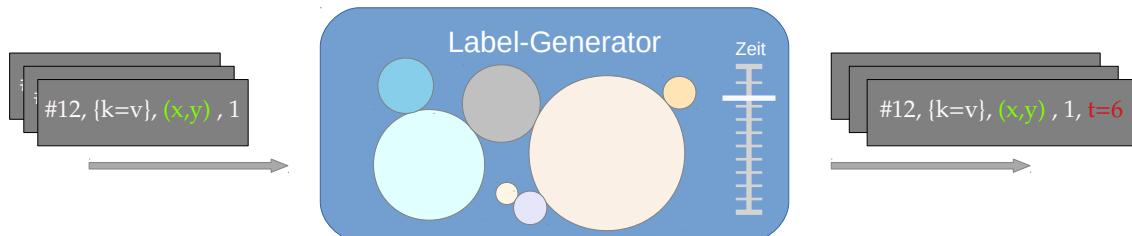


**Abbildung 2.4:** Schematische Darstellung der Komponente POI-Classification

Passend zu jedem Filter sind Regeln zu Tag-Einträgen definiert. Aus diesen Definitionen der config generieren wir einen Entscheidungsbaum. Jedes Blatt des Baumes enthält eine bestimmte Gewichtung. Jedes Objekt durchläuft den Entscheidungsbaum. Je nachdem ob das aktuelle Objekt einen bestimmten Key oder Key-Value-Eintrag vorweisen kann, steigt es links oder rechts ab. So kann für jedes POI deterministisch ein eindeutiges Scoring bestimmt werden, das dem jeweiligen Eintrag als Attribut hinzugefügt wird.

## 2.5 Label-Generator

Abbildung 2.5 skizziert den Ablauf der finalen Berechnung: Wir verwenden den in [Kru18a] zur Verfügung gestellten Code, um die Schnittpunkte der Label zu bestimmen.



**Abbildung 2.5:** Schematische Darstellung der Komponente Label-Generator

Intern werden um die  $(x, y)$ -Koordinaten der Ankerpunkte Kreise gezogen. Diese wachsen mit derselben Geschwindigkeit so lange an, bis sich zwei oder mehr Kreise zum ersten Mal schneiden. Die in Konflikt stehenden Label werden anhand ihres Scores aus Abschnitt 2.4 verglichen. Das Label mit der höchsten Gewichtung setzt sich durch, die anderen hinterlegen den Auslösungszeitpunkt und werden vom weiteren Wachstumsprozess ausgeschlossen. Je später der Auslösungszeitpunkt, desto länger ist das Label beim späteren Zoomvorgang sichtbar: Es ist relativ gesehen wichtig.

## **2 Beschriftungspipeline**

---

Der Vorgang wird so lange wiederholt bis nur noch ein Label übrig ist oder eine obere Schranke für den Auslösungszeitpunkt überschritten wird.

### **2.6 Ausgabe**

Nach dem erfolgreichen Durchlaufen der Pipeline erhalten wir alle in Abschnitt 1.2 geforderten Eigenschaften:

Für jeden POI erhalten wir einen Ankerpunkt sowie eine Priorität.

Die Beschriftungsgröße lässt sich wiederum von der Priorität ableiten und je nach Anwendungsfall beliebig skalieren.

Das Verfahren zur Prioritätsbestimmung bietet einen enormen Vorteil im späteren Einsatz der Daten: Soll ein Kartenausschnitt gerendert werden, benötigt man die Maßen des Bildausschnittes und das aktuelle Zoomlevel. Es können nun sehr effizient in Echtzeit Label ausgeschlossen werden, es bedarf lediglich einer Prüfung aller in Frage kommenden Label, ob deren Auslösungszeitpunkt größergleich dem aktuellen Zoomlevel ist.

# 3 Berechnungskontext

Ein wesentlicher Bestandteil dieser Arbeit stellt die Bestimmung eines sinnvollen Mittelpunktes einer gegebenen Polygonstruktur dar. Dazu definieren wir in Abschnitt 3.1 die Begrifflichkeit für unseren Kontext und legen eine syntaktische Grundlage zur späteren Berechnung in Abschnitt 3.2 fest. Die Lücke zwischen den gegebenen Daten und dem für die Anwendung der Verfahren notwendigen Format schließen wir im letzten Abschnitt 3.3.

## 3.1 Definition Mittelpunkt

Im allgemeinen Sprachgebrauch wird der Begriff *Mittelpunkt* multipel verwendet: Er beschreibt beispielsweise einen Ort, der zentral gelegen ist. Im geometrischen Kontext beschreibt er einen  $d$ -dimensionierten Punkt, der zu allen Punkten einer gegebenen Punktmenge denselben Abstand bezüglich einer gemessenen Norm aufweist.

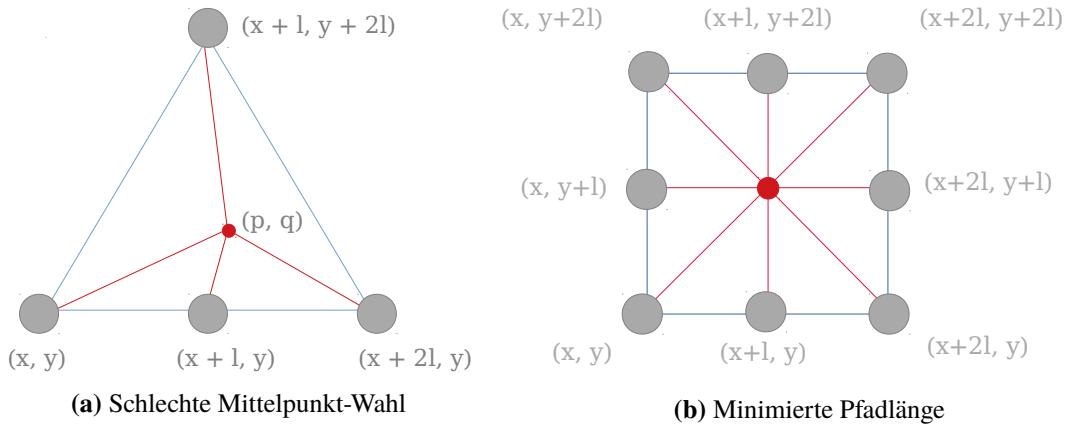
Ein solcher Punkt existiert nicht immer: Bei vier im Dreieck angeordneten Punkten, lässt sich nicht immer ein Punkt finden, der den selben Abstand zu allen anderen misst. Ein Beispiel hierfür zeigt Abbildung 3.1a. Als zweckmäßige Lösung wird ein Punkt als Mittelpunkt bezeichnet, wenn er den Fehler des Abstands minimiert, wie in Abbildung 3.1b.

Leider ist ein Mittelpunkt nach dieser Definition für unsere Anwendung unzureichend. Bei Eingabe von Polygonen könnte der minimierende Punkt außerhalb des Objekts liegen. Wir definieren einen Mittelpunkt zu einem Polygon mit  $n$  unterschiedlichen Punkten in  $d = 2$  wie folgt:

Ein Polygon-Mittelpunkt  $C$  repräsentiert möglichst visuell ansprechend das gesamte Objekt und hält folgenden Rahmenbedingungen ein:

- Es handelt sich um einen einzelnen Punkt  $(x, y)$
- Der Punkt befindet sich in derselben Ebene wie das Objekt
- Der Punkt ist innerhalb des Polygons
- Der Punkt ist deterministisch und effizient bestimmt worden

Insbesondere schließen wir für unsere Betrachtungen alle Algorithmen  $A$  aus, für deren Laufzeit  $t(A) \in \omega(n^k)$ ,  $\forall k \in \mathbb{N}$  gilt. Später mit der Evaluation der praxisorientierten Verfahren wird  $k$  auf drei fixiert.



**Abbildung 3.1:** Mittelpunktkandidaten (rot)

## 3.2 Berechnungsgrundlage

In Vorbereitung auf die Untersuchung der Mittelpunktverfahren wollen wir zu Beginn dieses Kapitels das theoretische Rahmenwerk aufbauen, das die Grundlage für spätere Analysen bildet. Dazu wird explizit das EingabefORMAT und die erlaubte Eingabemenge definiert, darüber hinaus wird eine Mittelpunktbenennung angegeben, auf die wir im Laufe dieser Arbeit mehrfach Bezug nehmen. Im zweiten Abschnitt werden wir feststellen, dass dieses Eingabe-Format nicht damit übereinstimmt, wie Objekte in OSM konstruiert sind. Zur sinnvollen Verwendung der Eingabedaten wird eine Umwandlung notwendig sein, auf die wir in Abschnitt 3.3 eingehen werden.

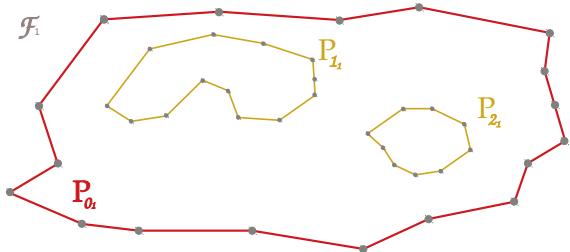
### 3.2.1 Theoretische Berechnungsgrundlage

Um die in Kapitel 4 vorgestellten Verfahren analysieren zu können, definieren wir für diesen Zweck die mathematische Grundlage. Eine Eingabe  $E$  zur Mittelpunktbestimmung hat die Form:

$$E = \mathcal{F}_1, \mathcal{F}_2, \dots, \mathcal{F}_f \quad \text{für } f > 0 \quad (3.1)$$

$$\mathcal{F}_j = \mathcal{P}_{0,j}, \mathcal{P}_{1,j}, \dots, \mathcal{P}_{k_j} \quad \text{für } k_j \geq 0 \text{ und } 1 \leq j \leq f \quad (3.2)$$

$$\mathcal{P}_{i,j} = (x_{1,i,j}, y_{1,i,j}), (x_{2,i,j}, y_{2,i,j}), \dots, (x_{n_{i,j}}, y_{n_{i,j}}), (x_{1,i,j}, y_{1,i,j}) \quad \text{für } n_{i,j} \geq 3 \text{ und } 0 \leq i \leq k_j \quad (3.3)$$



**Abbildung 3.2:** Eine beispielhafte gültige Eingabe  $E = \mathcal{F}_1$

Zur besseren Übersicht werden wir manche Indices an Stellen, an denen aus dem Kontext hervor geht, welche Elemente gemeint sind, weglassen.

Eine Eingabe  $E$  besteht also aus einer Liste von Polygonfamilien  $\mathcal{F}_j$ . Jede Familie besteht aus genau einem sogenannten *äußeren Polygon*  $\mathcal{P}_{0,j}$ , dem  $k_j$  Löcher (die im Folgenden *innere Polygone* genannt werden) zugeordnet sind. Jedes Polygon besteht aus  $n_{i,j}$  Punkten, die aus  $\mathbb{R}^2$  stammen.

Für unsere Anwendung lassen wir nur Eingaben zu, die folgende Konsistenz-Eigenschaften erfüllen:

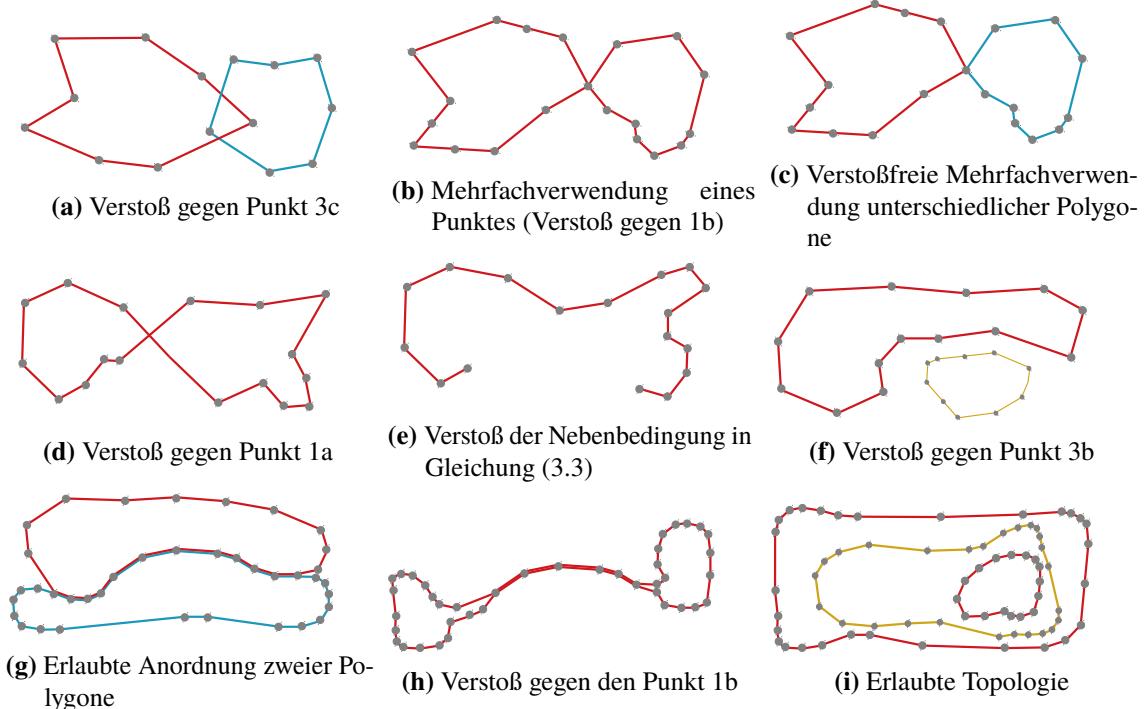
1. Schnittfreie Kanten innerhalb  $\mathcal{P}$ 
  - (a) Keine Polygonkante schneidet eine andere
  - (b) Jedes Polygon hat genau  $n_i$  verschiedene Punkte
2. Nichtleere Fläche (jedes Polygon hat eine Fläche  $> 0$ )
3. Schnittfreie Polygone innerhalb  $\mathcal{F}$ 
  - (a) Jede Familie hat  $k_j$  unterschiedliche Polygone
  - (b) Jedes innere Polygon befindet sich innerhalb des äußeren Polygon
  - (c) Keines der Polygone schneidet sich
4. Jede Eingabe besteht aus  $f$  unterschiedlichen Familien  $\mathcal{F}_j$
5. Die Polygone einer Familie  $\mathcal{F}_l$  befinden sich innerhalb eines Loches einer anderen  $\mathcal{F}_m$  oder sind freistehend

Die Eigenschaft aus Punkt 2 folgt direkt aus Punkt 1a, Punkt 1b und Gleichung (3.3): Jedes Polygon besteht aus mindestens 3 verschiedenen Punkten (und einem vierten, der identisch zum ersten ist), die kreuzungsfrei eine Fläche aufspannen. Der Abbildung 3.3 können beispielhafte Fälle spezieller gültiger und ungültiger Eingaben entnommen werden.

Der Begriff des Schnittes ist im Kontext der Eingabedefinition zweifach belegt. Wir unterscheiden zwischen Polygon-Selbst-Schnittpunkten und Polygon-Polygon-Schnittpunkten, denen wir uns im nächsten Absatz annähern. Ein Schnitt innerhalb eines Polygons ist in jeglicher Hinsicht ungültig: Mit Ausnahme des ersten und letzten Punktes darf kein Knoten mehrfach verwendet werden, um Zyklen innerhalb eines Polygons zu verhindern (siehe dazu Abbildung 3.3b). Weiterhin darf sich keine Kante mit einer anderen schneiden (siehe dazu Abbildung 3.3d). Ein Berühren ist demnach lediglich zwischen Kanten erlaubt, die einen gemeinsamen Knoten verbinden.

Innerhalb des Kontextes von Familien ist der Schnittpunktbegehr schwächer formuliert. Echte Polygon-Polygon-Schnitte sind untersagt, allerdings sind Berührungen erlaubt. Beispielsweise kann ein Punkt oder eine Kante von einem angrenzenden Polygon mitverwendet werden. Abbildung 3.3g zeigt eine gültige Berührung zweier Polygone, während Abbildung 3.3h eine ungültige Form eines Polygons aufweist.

Der Punkt 5 sagt aus, dass sich kein äußeres Polygon direkt innerhalb eines anderen äußeren Polygons befinden darf. Konkret formuliert, schneiden sich keine zwei Flächen, die von einem äußeren Polygon aufgespannt werden, aus dem die aufgespannten Flächen der inneren Polygone ausgeschnitten wurden. Ein gültiges Beispiel liefert Abbildung 3.3i, das ungültig werden würde, wenn die innersten zwei Polygone ihre Rolle tauschen würden.



**Abbildung 3.3:** Mögliche Eingaben und deren Konsistenzbewertungen: Äußeres Polygon von  $\mathcal{F}_1$  (rot), inneres Polygon (ocker) innerhalb  $\mathcal{F}_1$ , äußeres Polygon (cyan) von  $\mathcal{F}_2$ .

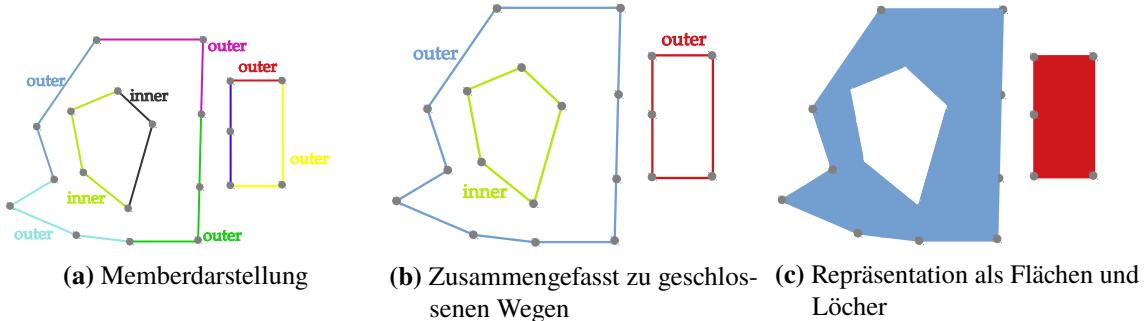
Eine Ausgabe besteht aus einem Tupel des  $\mathbb{R}^2$  der Form  $(x_c, y_c)$ . Für ihn gelten im Speziellen die in Abschnitt 3.1 geforderten Eigenschaften. Im Falle einer ungültigen Eingabe wird, sofern diese nicht repariert werden kann, der Berechnungsvorgang gestoppt und eine entsprechende Meldung ausgegeben.

### 3.2.2 OpenStreetMap Berechnungsgrundlage

In Abschnitt 2.1 und 1.1 haben wir die Grobstruktur, nach der Flächen in OpenStreetMap definiert sind, kennen gelernt. Wir werden diese im Folgenden verfeinern, um den Umwandlungsschritten des übernächsten Abschnittes folgen zu können. Die Umwandlung ist erforderlich, um die Lücke zwischen dem vorliegenden OSM-Datenformat und der abstrakteren Berechnungssyntax der Mittelpunktrechner zu überbrücken.

In der Markierungssyntax von OpenStreetMap existiert kein Primitiv, welches das Objekt *Fläche* explizit kapselt. Eine Klassifizierung wird ausschließlich über Tags realisiert [Joh18]. Dies bietet einerseits den Vorteil, die Primitivanzahl übersichtlich und das Modell dadurch leicht verständlich zu halten, erschwert andererseits die Validierung eines Tags; es wäre theoretisch denkbar einen nicht-geschlossenen Pfad als Fläche zu deklarieren.

Im Laufe der Zeit haben sich de-facto Standards zum Taggen einer Fläche entwickelt, auf die wir aufbauen. Eine Fläche wird entweder durch einen geschlossenen Linienzug (Way) oder ein Multipolygon definiert [Wik16].



**Abbildung 3.4:** Schematische Interpretation der acht Way-Member

Ein geschlossener Linienzug ähnelt stark dem in Abschnitt 3.2.1 forcierten Ansatz, indem eine geschlossene Punktliste die Grundlage bietet. Abgeschlossen wird das Objekt mit dem Tag 'area' = 'yes'. In besonderen Fällen werden geschlossene Wege ebenfalls als Fläche gerendert, wenn sie stattdessen Key-Value-Kombinationen aufweisen, die auf eine Fläche schließen lassen. Beispiele hierfür sind geschlossene Wege mit Key 'building' oder 'landuse' [Wik19a]. Da wir Objekte dieser Form direkt in eine Eingabe bestehend aus einer Familie  $E = \mathcal{F} = \mathcal{P}$ , überführen können, beschränken wir uns bei der weiteren Untersuchung auf die zweite Art der Polygondarstellungen.

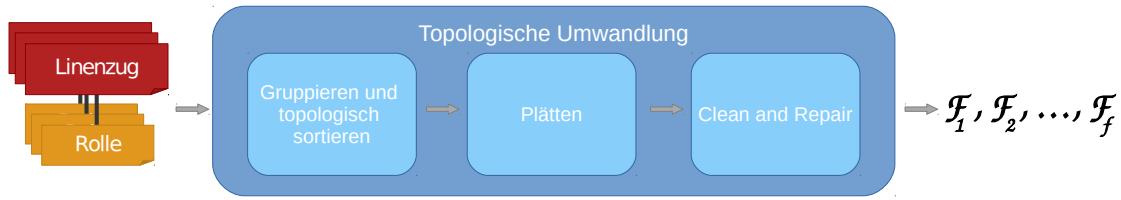
Das zweite Konstrukt zur Flächendefinition baut auf dem Primitiv der Relation auf. Eine Relation kann als Fläche interpretiert werden, wenn es das Schlüsselpaar 'type' = 'multipolygon' aufweist. Eine Relation enthält, neben einer Liste von Tags, mindestens einen *Member*. Hierbei handelt es sich um einen Verweis auf ein anderes Primitiv mit dessen OSM-ID und einer Rolle. Obwohl theoretisch auf ein beliebiges Primitiv verwiesen werden könnte, wird im Kontext von Multipolygonen ausdrücklich darauf hingewiesen, ausschließlich auf Wege oder geschlossene Wege zu referenzieren [Wik19b]. Jedem Weg wird demnach eine Rolle zugewiesen. Man unterscheidet zwischen den Rollen *outer* und *inner*. Mehrere adjazente Wegzüge werden als ein Wegzug aufgefasst, so entstehen auch geschlossene Wege. Die Rolle definiert die Fläche, die das Polygon einschließt. Ein geschlossener Wegzug in der Rolle als Outer definiert eine Fläche analog zum obigen Ansatz. Ein in der Rolle Inner befindlicher geschlossener Wegzug schneidet in die Fläche, in der er sich befindet, ein Loch. Die Abbildung 3.4 zeigt eine beispielhafte Darstellung einer Relation mit acht Wegen, davon zwei in der Rolle Inner.

Die Konvertierung der Multipolygon-Relation in das mathematische Modell ist nicht so trivial, wie es intuitiv vermuten lässt. Daher widmen wir das nächste Kapitel diesem Umwandlungsschritt.

### 3.3 Topologische Umwandlung

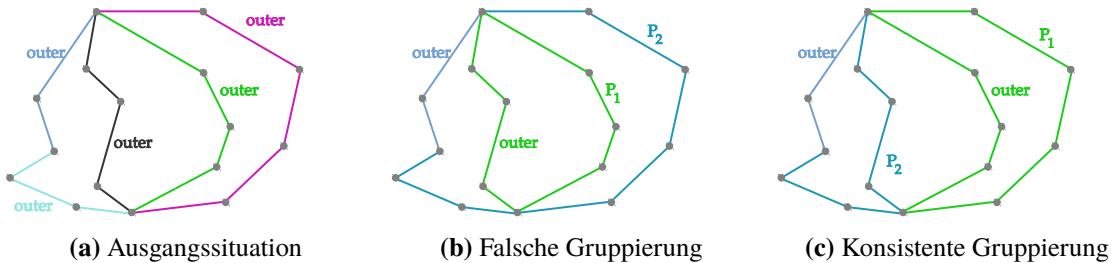
Die Umwandlung lässt sich abstrakt in drei logische Schritte einteilen: Zu Beginn werden die Linienzüge geladen und zu geschlossenen Linienzügen verbunden. Im Anschluss werden diese topologisch sortiert, es entsteht ein Baum. Im zweiten Schritt werden aus dem Baum einzelne Polygonfamilien extrahiert, bevor im letzten Schritt die Daten auf Konsistenz geprüft und andernfalls bereinigt werden. Wir verweisen an dieser Stelle auf Abbildung 3.5, die den Prozess schematisch darstellt.

### 3 Berechnungskontext



**Abbildung 3.5:** Die topologische Umwandlung im Überblick

Ausgehend von allen Liniensegmenten eines Multipolygons mit jeweils zugehörigen Rollen, werden daraus geschlossene Linienzüge gebildet. Für einen nicht-geschlossenen Linienzug kommen nur angrenzende Liniensegmente in Frage, die einen gemeinsamen Endpunkt verbinden und derselben Rolle sind. Zusätzlich ist jedoch zu beachten, dass topologische Eigenschaften weiterhin eingehalten werden: Ein äußeres Polygon  $P_a$  darf ausschließlich innerhalb eines anderen äußeren  $P_b$  liegen, wenn es sich gleichzeitig innerhalb eines Loches in  $P_b$  befindet. Um einen Eindruck zu vermitteln, welche Probleme auftreten können, verweisen wir auf Grafik 3.6. Es lassen sich leicht, analog zum Vorgehen in Abbildung 3.6, umfangreichere Beispiele generieren, die den Aufwand schnell groß werden lassen. Es reicht offensichtlich nicht aus, die Linienzüge per Tiefensuche zu ermitteln, vielmehr bietet sich eine Backtracking-Lösung an. In unserer Arbeit greifen wir zur Problemlösung auf ein am Institut<sup>1</sup> entwickeltes Tool<sup>2</sup> zurück, das auf die C++-Library *CGAL* aufbaut.



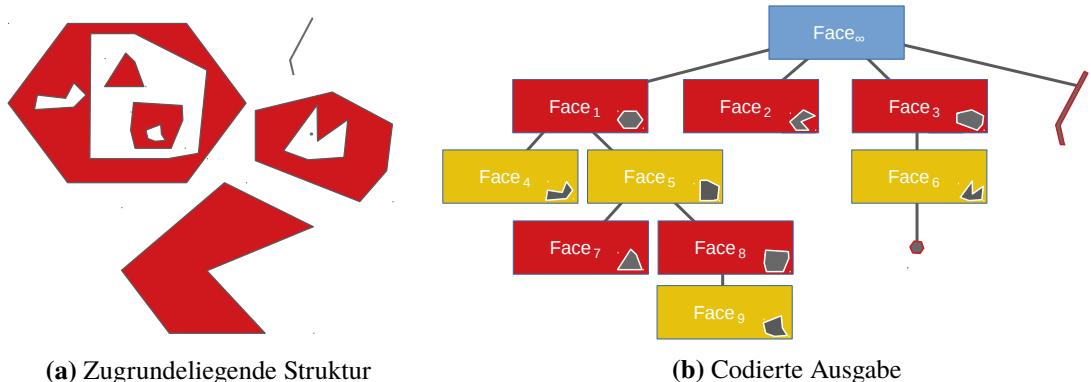
**Abbildung 3.6:** Möglichkeiten zur Linienzugbildung

Als Ausgabe erhalten wir einen Baum, der die topologischen Beziehungen codiert. Die abstrakte Darstellung lässt sich beispielhaft in Abbildung 3.7 nachvollziehen. Beginnend vom Infinite-Face, die der gesamten aufgespannten Fläche des  $\mathbb{R}^2$  entspricht, beschreiben wir mithilfe von Kanten eine binäre antisymmetrische *ist-enthalten-in*-Relation  $R$ . Je Ebene wechselt sich die Rolle der definierten Flächen ab: Die erste Ebene charakterisiert äußere Polygone, die zweite Löcher darin, dargestellt mithilfe von inneren Polygonen. Innerhalb der Löcher können weitere äußere Polygone enthalten sein. Die Ebenen lassen sich beliebig erweitern. Zusätzlich sind Primitive erlaubt: Nicht geschlossene Linienzüge und Punkte. Als Kind einer Fläche kann entweder ein Face in entsprechend anderer Rolle stehen, oder alternativ ein Primitiv.

Im zweiten Verarbeitungsschritt trennen wir den Baum auf, indem wir zunächst jedes Outer-Polygon mit dessen Kindern der direkten Ebene darunter als eigene Familie auffassen.

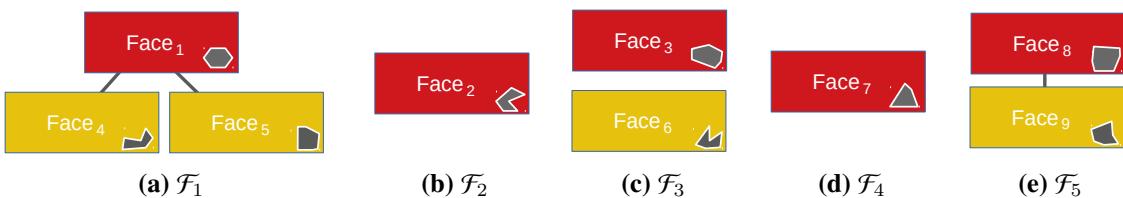
<sup>1</sup>Institut für Formale Methoden der Informatik (FMI) Abteilung Algorithmik (ALG) der Universität Stuttgart

<sup>2</sup>Polygonizer (in C++) von F. Krumpe, T. Mendel (Universität Stuttgart)



**Abbildung 3.7:** Beispiel einer topologisch sortierten Szene

Anschließend werden die in Abschnitt 3.2.1 geforderten Konsistenz Eigenschaften geprüft und ungültige Familien repariert oder entfernt. Das Ergebnis der Umwandlung lässt sich Abbildung 3.8 entnehmen.



**Abbildung 3.8:** Die entstandene Menge  $E = \mathcal{F}_1, \mathcal{F}_2, \mathcal{F}_3, \mathcal{F}_4, \mathcal{F}_5$

Nach Durchlaufen der Pipeline erhalten wir die gewünschte mathematische Form. Diese stellt die Grundlage für die im folgenden Kapitel vorgestellten Mittelpunkt-Bestimmungsverfahren.



## 4 Mittelpunktberechnung

In den folgenden Kapiteln werden Verfahren vorgestellt, die wir für unsere Implementierung in Betracht gezogen haben und beleuchten deren Vor- und Nachteile, insbesondere in Hinblick auf Laufzeit und visuelles Ergebnis. Zur besseren Übersicht wird davon ausgegangen, dass unsere Eingabe  $E$  auf eine Familie  $\mathcal{F}$  beschränkt ist. Für den  $|E| > 1$  Fall können die Verfahren mehrfach verwendet werden, die Zeiten skalieren entsprechend mit. Wenn nicht anders gegeben, gehen wir also von der Eingabe folgender Form aus:

$$\mathcal{F} = \mathcal{P}_0, \mathcal{P}_1, \dots, \mathcal{P}_k \quad (4.1)$$

$$\mathcal{P}_i = (x_{1_i}, y_{1_i}), (x_{2_i}, y_{2_i}), \dots, (x_{n_i}, y_{n_i}), (x_{1_i}, y_{1_i}) \quad (4.2)$$

Unser Ziel wird es sein, innerhalb dieser Polygonfamilie einen Mittelpunkt  $C = (x_c, y_c)$  gemäß Abschnitt 3.1 zu bestimmen. Zur Vereinfachung der Komplexitätsbetrachtungen wird vorausgesetzt, dass die Eingabe aus  $n_i$  Punkten für das Polygon  $\mathcal{P}_i$  besteht. Wir bezeichnen die Anzahl aller Punkte als  $n$ . In den ersten drei Kapiteln wird die Notation  $\mathcal{P}_i$  verwendet, um die Polygone zu unterscheiden. In Abschnitt 4.4 und 4.5 beschränken wir uns auf ein Polygon. Darin werden wir neue Punkte im  $\mathbb{R}^2$  definieren, die  $P$  als Bezeichnung erhalten.

### 4.1 Gemittelte Summe

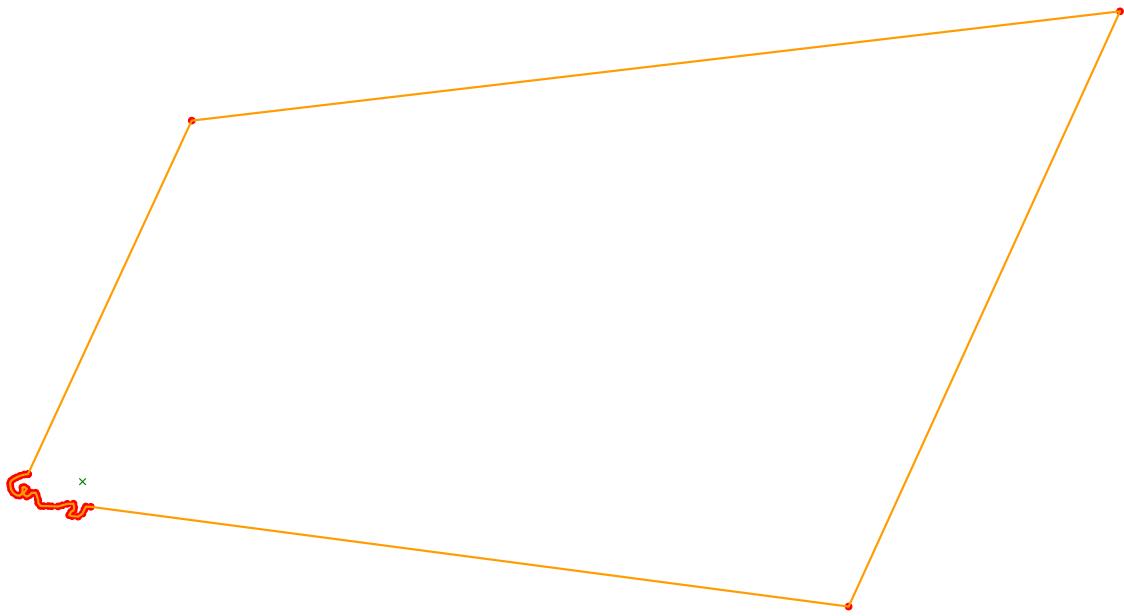
Eine Möglichkeit zur Berechnung eines Punktes stellt das zweidimensionale arithmetische Mittel dar. Die Idee dahinter ist, einen Punkt zu erhalten, der den Durchschnitt aller Punkte bildet und dadurch recht zentral liegt.

Hierzu erweitern wir die Formel im eindimensionalen Fall und erhalten den Mittelpunkt zum  $i$ . Polygon:

$$C_i = \frac{1}{n_i} \sum_{j=1}^{n_i} (x_{j_i}, y_{j_i}) = \frac{1}{n_i} \left( \sum_{j=1}^{n_i} x_{j_i}, \sum_{j=1}^{n_i} y_{j_i} \right) \quad (4.3)$$

Bei der Berechnung erhalten wir einen simplen Algorithmus mit optimaler Laufzeit in der Anzahl der Polygonpunkte.

Da lediglich ein Mittelpunkt zu bestimmen ist, wird für  $E$  stellvertretend  $C = C_0$  berechnet. Obwohl der ermittelte Punkt für regelmäßige Polygone das Optimum darstellt, ist das Ergebnis im Allgemeinen nicht als Mittelpunkt tauglich. Da die Kanten des Polygons beliebiger Länge sein dürfen, verlagert sich das Ergebnis in Richtung der meisten Ecken. Punkthaufungen ziehen den Ergebnispunkt an. Abbildung 4.1 veranschaulicht diesen Gedanke. In dem oben gezeigten Fall werden mögliche Löcher außer Acht gelassen, außerdem könnte  $C$  außerhalb des Polygons



**Abbildung 4.1:** Konstruiertes Negativbeispiel: Die Häufung der Polygonpunkte (rot) zieht den Mittelpunkt (grün) bei Verwendung der gemittelten Summe an.

liegen. Da die inneren Polygone nicht mitgewichtet werden, ist der Ergebnispunkt zudem nicht repräsentativ. Auch wenn man diese Überlegung in Gleichung (4.3) einfließen, und die inneren Punkte mitgewichten lässt, bleiben die restlichen Probleme ungelöst.

Wir verwenden daher an keiner Stelle die gewichtete Summe.

Analog zur Anwendung im Eindimensionalen, könnte man auch den d-dimensionalen (hier  $d = 2$ ) Median  $M_i$  zum  $i$ . Polygon berechnen:

$$M_i = \underset{(x_{M_i}, y_{M_i}) \in \mathbb{R}^2}{\operatorname{argmax}} \sum_{j=1}^{n_i} \|(x_{j_i}, y_{j_i}) - (x_{M_i}, y_{M_i})\|_2$$

Dieser kommt der Idee der geringsten mittleren Abstände aus Abschnitt 3.1 am nächsten, hat dadurch aber ähnliche Eigenschaften wie der Ansatz in Formel 4.3. Die Bestimmung eines entsprechenden Punktes  $(x_{M_i}, y_{M_i})$  ist im Allgemeinen rechnerisch aufwendig, man kann sogar zeigen, dass weder eine explizite Formel noch ein exakter Algorithmus zur Lösung des Problems existiert [Baj86]. Man müsste stattdessen einem numerisch iterativen Ansatz nachgehen. Um den Abstand zu allen Punkten zu minimieren, bietet es sich in unserem Extrembeispiel an, in der Nähe der Punktewolke südwestlich zu liegen. Der Punkt neigt sich also noch weiter in diese Richtung. Zudem vererbt sich die Eigenschaft, dass der Mittelpunkt innerhalb eines Loches liegen könnte.

## 4.2 Geometrischer Mittelpunkt

Bei dem geometrischen Mittelpunkt interpretiert man die Eingabe als Fläche, in die die inneren Polygone Löcher schneiden. Im Anschluss wird der Punkt berechnet, der den Schwerpunkt dieser Fläche repräsentiert. Gewichten wir die Polygonstücke gemäß deren *Masse* (hier mit  $d = 2$  die Fläche), können wir das Verfahren auf Komponenten, bestehend aus mehreren Flächen, ausweiten. Berechnet man die Summe aller gerichteten Flächen zwischen einer Kante und der x-Achse, kann man daraus die Fläche des Polygons  $\mathcal{P}_i$  bestimmen[Bou98]:

$$A_{\mathcal{P}_i} = \sum_{j=1}^{n_i} \frac{1}{2} (x_{j_i} + x_{j+1_i}) (y_{j+1_i} - y_{j_i}) = \frac{1}{2} \sum_{j=1}^{n_i} a_{j_i} \quad (4.4)$$

Hierbei bezeichnet  $a_{j_i} = x_{j_i} y_{i+1_j}$  den Term, der nach Entfernen der Teleskopenteile übrig bleibt. Man beachte, dass wir hier  $x_{(n_i+1)_j} := x_{1_j}$  und  $y_{(n_i+1)_j} := y_{1_j}$  verwenden, um die letzte Kante von  $(x_{n_i}, y_{n_i})$  nach  $(x_1, y_1)$  zu modellieren. Bestimmt man von jedem Trapez-Teilstück den Schwerpunkt und gewichtet diesen mit der zugehörigen gerichteten Fläche, erhält man schließlich den Schwerpunkt des Polygons[Bou88]:

$$\begin{aligned} C = (x_c, y_c) &= \left( \frac{1}{6A} \sum_{j=1}^{n_i} (x_{j_i} + x_{j+1_i}) a_{j_i}, \frac{1}{6A} \sum_{j=1}^{n_i} (y_{j_i} + y_{j+1_i}) a_{j_i} \right) \\ &= \frac{1}{6A} \left( \sum_{j=1}^{n_i} (x_{j_i} + x_{j+1_i}) a_{j_i}, \sum_{j=1}^{n_i} (y_{j_i} + y_{j+1_i}) a_{j_i} \right) \\ &= \frac{1}{6A} \sum_{j=1}^{n_i} a_{j_i} (x_{j_i} + x_{j+1_i}, y_{j_i} + y_{j+1_i}) \end{aligned} \quad (4.5)$$

Auch hier sind die überlaufenden  $j$ -Indices Modulo  $n_i$  zu verstehen.

Sowohl die Fläche als auch die Gewichtung der Schwerpunktstücke lässt sich in linearer Zeit berechnen, sodass wir zur Berechnung insgesamt in  $\theta(n_i)$  und damit insbesondere in  $\theta(n)$  liegen.

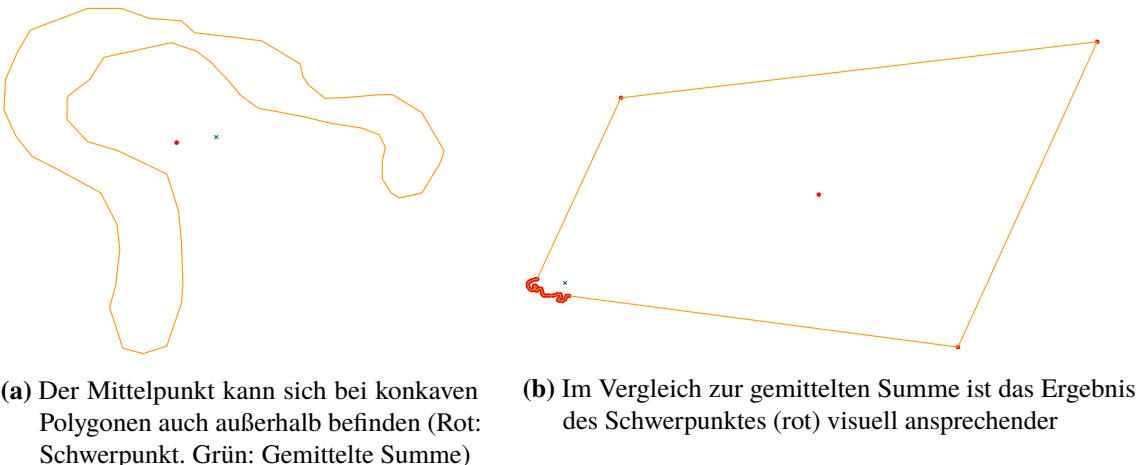
Handelt es sich bei der Eingabe um ein äußeres konvexes Polygon ohne Löcher, garantiert das Verfahren für den Punkt  $C$ , dass es innerhalb der Eingabe liegt. Im Allgemeinen liegt der Punkt aber nicht notwendigerweise innerhalb des Polygons (Abbildung 4.2a).

### Zusammengesetzte Polygone

Besteht die Eingabe aus einem äußeren und  $k$  inneren Polygonen, so kann der Schwerpunkt leicht bestimmt werden, indem man die Flächen der Polygone mit konjugiertem Vorzeichen gewichtet einbringt[BB15]:

$$C_{\mathcal{F}} = \frac{C^{\text{outer}} A_{\mathcal{P}_0} - \left( \sum_{i=1}^k C_i^{\text{inner}} A_{\mathcal{P}_i} \right)}{A_{\mathcal{P}_0} - \sum_{i=1}^k A_{\mathcal{P}_i}}$$

Hier entsprechen  $C_i^{\text{inner}}$  dem Mittelpunkt und  $A_{\mathcal{P}_i}$  der Fläche eines inneren Polygons  $\mathcal{P}_i$ .



**Abbildung 4.2:** Beispiele zu verschiedenen Schwerpunkteigenschaften

Nach dem selben Prinzip kann man den Schwerpunkt beliebig zusammengesetzter Familien berechnen:

$$C_E = \frac{\sum_{j=1}^f C_{\mathcal{F}_j} A_{\mathcal{F}_j}}{\sum_{j=1}^f A_{\mathcal{F}_j}}$$

## Evaluation

Intuitiv betrachtet charakterisiert ein Mittelpunkt nach Gleichung (4.5), sofern er denn innerhalb der Polygonfläche liegt, visuell sehr ansprechend. Das liegt vor allem daran, dass der Mittelpunkt im Gegensatz zu den bisher vorgestellten Verfahren nicht auf Grundlage der Eckpunkte, sondern der Fläche berechnet wird. Dadurch sind die Ergebnisse unabhängig von der Punktverteilung auf dem Polygonzug. Abbildung 4.2b verdeutlicht diesen Sachverhalt.

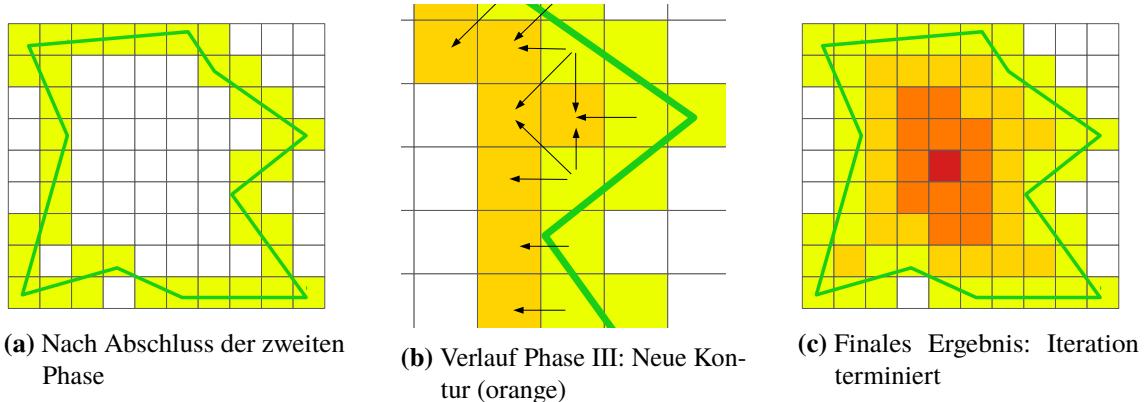
Zudem wird das Verfahren durch die Tatsache gestärkt, dass die Berechnung in der linearen Anzahl an Polygonpunkten ein optimales Verfahren bezüglich des Zeitbedarfs darstellt.

Schwächen weißt das Verfahren auf, sobald mehrere Familien im Spiel, Löcher in der Polygonfläche vorhanden, oder die Polygone stark entartet sind. Hierzu findet sich ein Beispiel in Abbildung 4.2a.

## 4.3 Morphologische Erosion

Die Idee hinter morphologischer Erosion lässt sich am Leichtesten durch eine Metapher erklären: Stellen wir uns vor, unsere Fläche wäre eine Insel, und die Löcher innerhalb des Polygons Seen. Lässt man nun den Meeres- und Wasserspiegel steigen, so wird die Fläche von den Rändern aus überschwemmt. Wir suchen uns den Punkt heraus, der am längsten *trocken* bleibt.

Zurück zur formalen Eingabe, berechnen wir auf einem Gitter iterativ engere Polygonumrandungen. Dabei wird die Fläche, die das Polygon einnimmt, stets kleiner. Hat die Fläche Minimalgröße erreicht, stoppen wir den Vorgang und geben den Flächenmittelpunkt aus.



**Abbildung 4.3:** Die Phasen der morphologischen Erosion anhand eines Beispiels

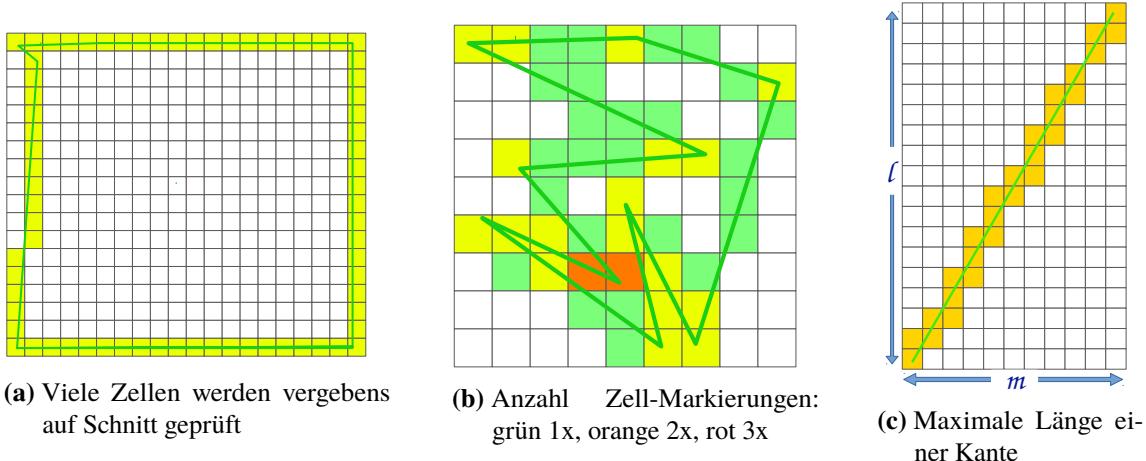
Algorithmisch lösen wir das Problem in drei Phasen: Zuerst berechnen wir uns ein hinreichend genaues Gitter, das die gesamte Polygonfamilie abdeckt. Anschließend markieren wir Zellen, die von äußeren oder inneren Polygonen überdeckt werden (Grafik 4.3a). Im dritten Schritt berechnen wir zur aktuellen Kontur eine engere, indem wir Zellen markieren, die adjazent zur aktuellen Kontur und innerhalb der Fläche sind. Wir stoppen Phase drei sobald die Fläche einen Querschnitt von einer Gitterzelle hat. Anschließend können wir die Mittelpunkte der final markierten Zellen als Lösungskandidaten ausgeben. In Abbildung 4.3c entspricht das dem Mittelpunkt der roten Zelle.

## Laufzeit

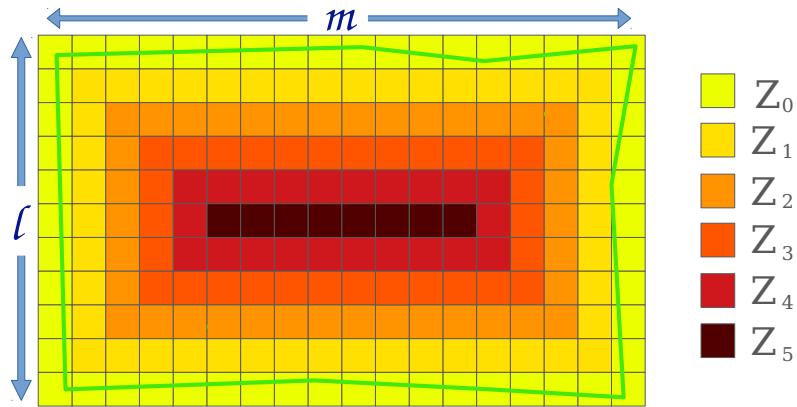
Offensichtlich hängt die Laufzeit neben der Anzahl an Polygonpunkten auch signifikant von der verwendeten Zellenanzahl ab: Starten wir mit  $l \times m$  vielen Zellen, benötigen wir zur Initialisierung  $\theta(lm)$  Rechenschnitte für Phase I. Falls die Maße der Polygon-Bounding-Box nicht bekannt sind, erhöht sich diese Abschätzung zur Bestimmung um einen additiven Term in  $\theta(n)$ .

In Phase II durchlaufen wir jede Kante des Polygons und markieren Zellen, die durch diese überdeckt werden. Hierfür existieren zwei Ansätze: Man könnte einerseits jede Zelle durchlaufen und prüfen, ob diese einen Schnittpunkt mit einer Kante hat. Hierfür werden  $O(lmn)$  Operationen benötigt. Das Verfahren hat den Nachteil, dass möglicherweise (wie in Abbildung 4.4a illustriert) viele Zellen betrachtet werden, die fernab möglicher Kanten sind. Alternativ durchläuft man die Eingabe kantenweise und markiert betroffene Zellen. Hierbei hängt die Laufzeit stark von der Implementierung und Kantenlänge ab. Obwohl die Laufzeit bei wenigen, langen Kanten gegenüber Variante eins deutlich besser ist, werden bei naiver Implementierung Rechenoperationen unnötigerweise mehrfach ausgeführt. Trotzdem werden für jede der  $n$  Kanten maximal  $c\sqrt{m^2 + l^2}$  Zellen markiert. Für  $\hat{d} = \max\{l, m\}$  ergibt sich eine obere Schranke von  $O(nc\sqrt{2\hat{d}^2}) = O(n\hat{d}) \subset O(n(l + m))$ .

In der dritten Phase iterieren wir einen Verkleinerungs-Algorithmus so lange, bis die Fläche einen Querschnitt von einer Zelle hat. Aufgrund der Stauchung von allen Seiten, benötigen wir für  $\check{d} = \min\{l, m\}$  maximal  $\left\lfloor \frac{\check{d}}{2} \right\rfloor$  Durchläufe. Wir benennen die neu markierten Zellen in der  $i$ . Iteration  $Z_i$ . Offensichtlich richtet sich die Anzahl an Operationen innerhalb einer Iteration  $i$  nach der Elementanzahl von  $Z_{i-1}$ . Für unsere Worst-Case-Analyse nehmen wir an, dass alle Randzellen



**Abbildung 4.4:** Verständnisunterstützende Grafiken zur Laufzeitbestimmung



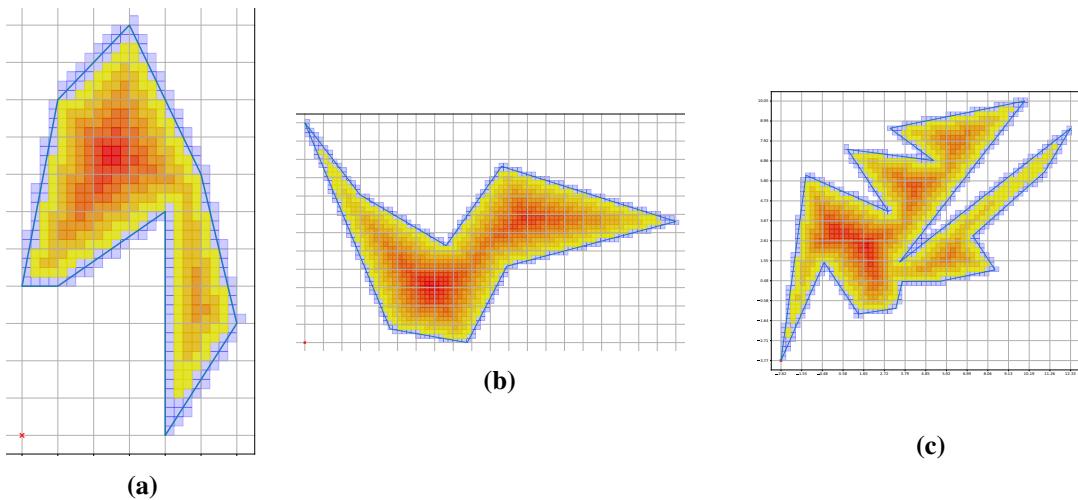
**Abbildung 4.5:** Ergebnis nach Terminierung im Worst-Case

$(0, j), (l - 1, j), (i, 0)$  und  $(i, m - 1)$  für  $0 \leq i \leq l$  und  $0 \leq j \leq m$  initial markiert sind. Das sind  $|Z_0| = 2m + 2(l - 2)$  Zellen zu Beginn der ersten Iteration. Im Allgemeinen lässt sich die Anzahl an Zellen im  $i$ . Durchlauf mit  $|Z_i| = 2(l - 2i) + 2(m - 2i) - 4 = 2(l + m) - 4(2i + 1)$  beschreiben.

Wir können die Struktur ausnutzen und folgern  $\sum_{i=0}^{\lfloor 0.5d \rfloor} |Z_i| = ml$  zu berechnenden Zellen. Naiv prüfen wir für jede Zelle konstant viele Nachbar-Zellen, die jeweils einen Punkt-in-Polygontest durchführen, für den sie  $n$  Kanten testen. Wir erhalten nach großzügiger Abschätzung eine  $O(lmn)$  Schranke, die gleichzeitig eine obere Schranke für den gesamten Algorithmus darstellt.

## Evaluation

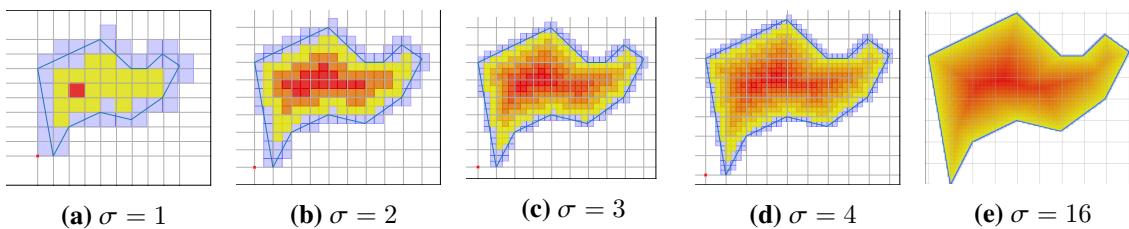
Die implizite Behandlung von Löchern innerhalb der Polygonfläche stellt einen entscheidenden Vorteil gegenüber den bisher gezeigten Verfahren dar. Das Verfahren garantiert bei ausreichend kleiner Wahl der Gitter das Finden von Mittelpunktkandidaten innerhalb des Polygons. Wir können diese Eigenschaft garantieren, da neue Zellen nur dann markiert werden, wenn deren Mittelpunkt



**Abbildung 4.6:** Verschiedene Polygone und deren Ergebnisse

einen Punkt-in-Polygontest bestand. Außerdem befinden sich die Endkandidaten aller besuchten Zellen maximal weit von der Kontur entfernt, daher garantiert das Verfahren auf Grundlage des Gitters Zentralität.

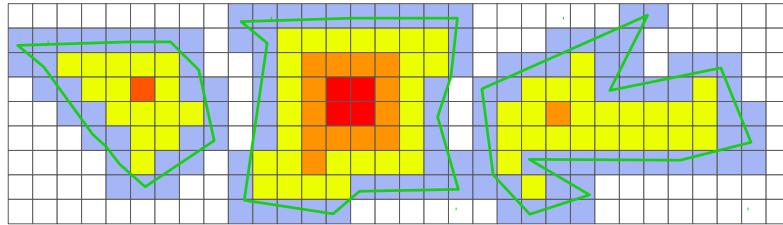
Diese Eigenschaft zeigt auch beispielhaft Abbildung 4.6. Die Ergebnisse sind visuell ansprechend und repräsentieren das Polygon ausreichend. Abbildung 4.7 zeigt zudem, dass bereits eine geringe Anzahl an Gitterzellen eine approximativ gute Näherung bieten. In der Grafik wurden die Gittergrößen durch mehrfaches Unterteilen des Grundgitters ( $\sigma = 1$ ) erzeugt. Ein quadratisches Gitter mit  $\sigma = 4$  hat beispielsweise die  $4^2$  fache Zellenmenge.



**Abbildung 4.7:** Erosionsergebnisse bei Variation von  $\sigma$

Zusätzlich zur Robustheit gegenüber Löchern kann das Verfahren auf beliebige Polygongruppen ausgeweitet werden. Legt man bei einer Eingabe jede der  $f$  Polygonfamilien auf ein Gitter derselben Auflösung  $\sigma$ , kann man global Punkte finden, die am weitesten innerhalb eines Polygons liegen. Abbildung 4.8 verdeutlicht dies.

Der flexiblen Einsetzbarkeit stehen dem Verfahren unerwünschte Lösungs-Charakteristika gegenüber. Während bei den Verfahren bisher die Lösungsmenge einelementig ausfiel, werden bei der morphologischen Erosion mit Gitter oft mehrere Lösungskandidaten bestimmt. Abbildung 4.5 ist ein gutes Beispiel, dass bei insbesondere einfachen, nahezu parallel verlaufenden Polygonkonturen Lösungskandidaten-Linien auftreten können. Es wäre ein nachgeschalteter Bewertungsschritt notwendig, um einen möglichen Optimum-Kandidaten zu bestimmen.



**Abbildung 4.8:** Ergebnis einer Eingabe mit drei Komponenten: Es verbleiben 4 Mittelpunktkandidaten

Ein weiterer signifikanter Nachteil des Verfahrens ist der zusätzliche unbekannte Parameter zur Bestimmung der Gitter-Größe. Intuitiv könnte man die Gittergröße abhängig von der eingenommenen Fläche machen. Dies lässt sich dadurch begründen, dass es bei großen Flächen eher in Kauf genommen werden kann, auf Kontur-Details zu verzichten, als bei kleinen.

Die Laufzeit hängt stark von der Anzahl der Zellen  $w$  ab, die durch die Parameter  $l$  und  $m$  definiert wird ( $w = lm$ ). Verwenden wir ein Gitter mit einer fixen Anzahl an Zellen, für die die Werte  $l$  und  $m$  gemäß dem Verhältnis der Bounding-Box des Polygons gewählt werden, so ergibt sich linearer Aufwand. Wählen wir die Werte anhand der absoluten Größe des Polygons, so kann die Laufzeit durch bloße Skalierung des Polygons beliebig erhöht werden. Eine naive Wahl der Parameter in Abhängigkeit der Kantenanzahl ist ebenfalls keine Universallösung. Es lassen sich leicht strukturell einfache Polygone mit großer Kantenanzahl bestimmen, indem die Polygonlinien beliebig oft unterteilt werden. Im Umkehrschluss können wenige Polygonpunkte eine ausreichend komplexe Kontur darstellen, für die die Zellenanzahl zu gering wäre.

Wir halten abschließend fest, dass eine naive Verwendung der morphologischen Erosion durch Schätzen der Parameter  $l$  und  $m$  im Allgemeinen nicht ausreicht, um eine anschauliche Lösung zu garantieren.

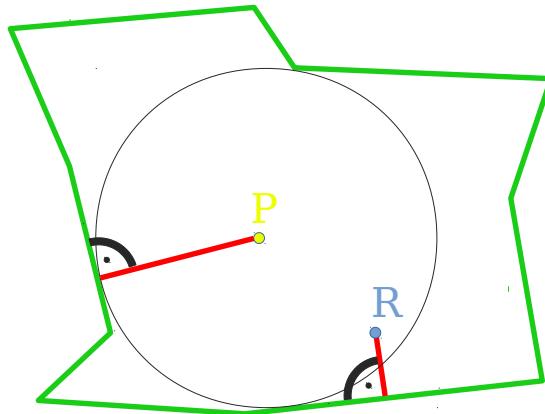
## 4.4 Point of Inaccessibility

Wir greifen die Überlegungen aus Abschnitt 4.3 erneut auf und wiederholen deren Charakteristika: Jeder Punkt, der in der Ergebnismenge liegt, stellt den Mittelpunkt einer Zelle dar, die verglichen zu nicht-finalen Zellen weiter im Inneren des Polygons liegt. Weiten wir diese Vorüberlegungen auf die kontinuierlich definierte Punktemenge innerhalb des Polygons aus, definieren wir eine bestimmte Punktemenge  $P^\circ$  in unserer Eingabefläche, indem wir den Übergang der Gitterbreite  $\rightarrow 0$  durch Gitterunterteilung  $\sigma \rightarrow \infty$  betrachten:

$$P^\circ = \{P \in A_E \mid dist_E(P) \geq dist_E(R), R \in A_E\} \quad (4.6)$$

Die Funktion  $dist_E : \mathbb{R}^2 \rightarrow \mathbb{R}$  gibt den minimalen Abstand von dem Punkt zu allen Punkten auf den Polygonrand wieder. Wir fügen also einen Punkt  $P$  zu  $P^\circ$  hinzu, sobald dieser Teil der Polygonfläche ist und dessen Minimalabstand zum Rand maximal ist. Abbildung 4.9 skizziert dies.

Da gemäß Definition alle Punkte innerhalb  $P^\circ$  dieselbe maximale Distanz aufweisen, reicht es für die Praxis aus, einen solchen Punkt zu bestimmen. Wir haben uns für diesen Zweck an der Idee von [GL07] orientiert, die analog zur kartografischen Begrifflichkeit  $P^\circ$  als *Poles of inaccessibility*



**Abbildung 4.9:** Exemplarische Darstellung der Punktes  $R \notin P^\circ$  und  $P \in P^\circ$  und deren Distanzen

(Pole der Unzugänglichkeit) bezeichnet. Garcia-Castellanos und Lombardo beschreiben in ihrer Arbeit einen Suchalgorithmus, zur Bestimmung eines Punktes auf der Weltkarte, der maximale Distanz zu einer Küste hat. Für unseren Rust-implementierten Algorithmus orientieren wir uns an der Grundlage des Codes von [Map18].

Prinzipiell funktioniert der Algorithmus nach einem vergleichbaren Grundprinzip wie in Abschnitt 4.3, indem ein Gitter über das Polygon gelegt und das Maximum der Funktionswerte  $dist_E(P_{l,m})$  für Zellenmittelpunkte  $P_{l,m}$  bestimmt wird. Allerdings sind die Zellen adaptiv, sodass nur relevante Teile des Polygons getestet werden. Die entscheidende Weiterentwicklung von [Aga16] besteht in den folgenden Berechnungsvarianten.

Bezeichnen wir eine quadratische Zelle als  $Z$  mit deren Diagonale  $d_Z$  und deren Radius  $r_Z = \frac{d_Z}{2}$ , sowie deren Mittelpunkt mit  $C_Z$ . Weiterhin bezeichnen wir die Fläche, die  $Z$  abdeckt als  $A_Z$ . Dann gilt für jeden Punkt  $R \in A_Z$  [Aga16]:

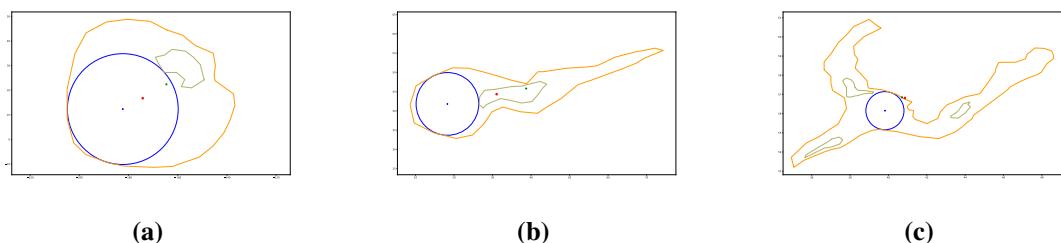
$$dist_E(R) \leq dist_E(C_Z) + r_Z =: cellmax_Z$$

Ist zudem der bisher größte tatsächlich gemessene Abstand  $\hat{dist}$  größer gleich dem aktuell berechneten  $cellmax_Z$ , kann diese Zelle übersprungen werden, da sich in ihr kein Maximum befinden kann.

## Laufzeit

Die Laufzeit hängt von dem darunterliegenden Polygon und dem Exaktheitsparameter ab. In einer Priority-Queue werden noch nicht berechnete Zellen verwaltet. Es wird so lange rekursiv abgestiegen, bis keine neuen Zellen hinzugefügt werden können. Eine Zelle wird der Queue hinzugefügt, wenn deren Eltern-Zelle aus der Queue stammt und sich vierteilte. Eine Zelle teilt sich sobald die echt berechnete Distanz in der Zelle besser ist als die aktuell beste und die Abweichung noch größer als die Genauigkeitsschranke ausfällt. Die Queue ist gemäß dem  $cellmax_Z$  sortiert.

Nach eigenen Angaben ist der gezeigte Algorithmus 40 mal schneller als die originale Implementierung. [Aga16]



**Abbildung 4.10:** Beispiele für einen geeigneten Einsatz des Verfahrens (rot: geometrischer Mittelpunkt, grün: gemittelte Summe, blau: Point of Inaccessibility mit Umkreis)

## Evaluation

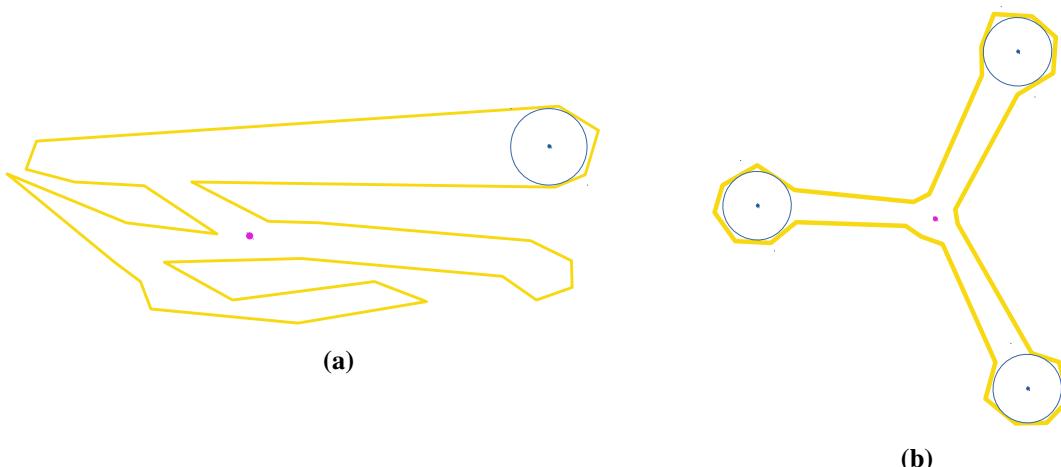
Das Verfahren hat die Eigenschaft innerhalb einer vorgegebenen Präzision garantiert den optimalen *Point of Inaccessibility* zu finden.

Da die Laufzeit die der morphologischen Erosion aussticht, bevorzugen wir hinsichtlich Geschwindigkeit dieses Verfahren.

Ähnlich dem Ansatz des vorhergehenden Kapitels, liegt der gefundene Punkt stets innerhalb des Polygons. Auch Löcher werden implizit behandelt.

Außerdem ist das Verfahren für unsere Datenmenge sehr schnell und findet auf  $10^{-8}$  Längen- und Breitengrad die korrekte Lösung. Dies entspricht einer Abweichung von weniger als einem Meter sowohl auf Höhe Stuttgarts (ca.  $49^\circ$ ) als auch im Worst-Case-Fall auf Höhe des Äquators<sup>1</sup>.

Wie die naiven Mittelpunktberechnungen kann das Verfahren nur auf eine Polygonfamilie angewendet werden, bei zusammengesetzten Polygonen müssen zusätzliche Überlegungen vorgenommen werden. Wir werden diesen Punkt in Kapitel 5 noch einmal aufgreifen.

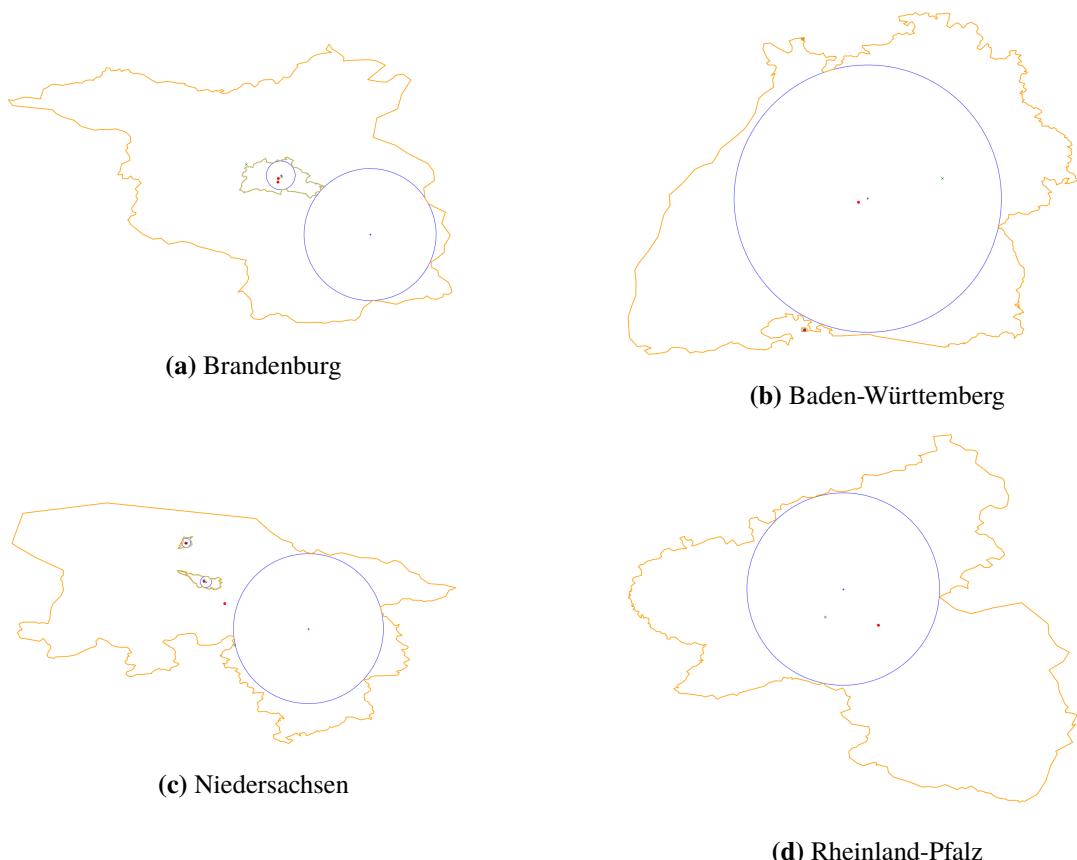


**Abbildung 4.11:** Gegenbeispiele (blau: PoI, magenta: von Hand gesetzter Punkt)

<sup>1</sup>Bezugspunkte: Universität Stuttgart (48.74566, 9.106436) und Worst-Case Golf von Guinea (0, 9.106436). Errechnet mit <https://rechneronline.de/geo-koordinaten>

Auch wenn die Suche nach einem maximal großen Kreis innerhalb einer Polygonfamilie gute Ergebnisse vermuten lässt, muss auf Problemfälle hingewiesen werden. Bei länglichen, dünnen Polygonen kann eine Weitung, in die die PoI-Kreisscheibe passt, nicht repräsentativ sein. Ein plakatives Beispiel ist mit Abbildung 4.11 zu finden.

Die Bestimmung eines einzelnen Punktes ist aus Sicht der Aufgabenstellung vorteilhafter, es kann jedoch auch als Nachteil gegenüber dem Verfahren mit Erosionsansatz gewertet werden: Während wir uns mit dem gefundenen Lösungspunkt zufrieden geben müssen, haben wir bei der Erosion in einem Nachbearbeitungsschritt die Möglichkeit, von mehreren Kandidaten den für uns passendsten zu suchen.



**Abbildung 4.12:** Berechnungsbeispiele auf Eingabe deutscher Bundesländer (PoI in blau)

## 4.5 Graph-Skeleton-Centroid

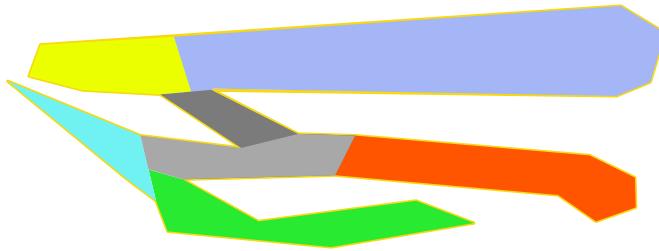
Im vorangegangenen Kapitel haben wir die Idee forciert, Punkte zu finden, die einen besonders großen Abstand zur Kontur des Polygons aufweisen. Für ein relativ regulär-förmiges Polygon haben wir gute Chancen einen solchen visuell ansprechenden Punkt zu finden. Betrachten wir das Gegenbeispiel aus Abbildung 4.11, so können wir daraus neue Ideen generieren. Viel besser, als die Distanz in alle Richtungen zu maximieren, ist es, die Distanzen in Bezug auf die Polygonform zu

## 4 Mittelpunktberechnung

---

betrachten. Bei einer sternförmigen Eingabe spielt der Punkt, der die Zacken verbindet, eine große Rolle und das obwohl dieser nicht die größte Minimal-Distanz zu allen umliegenden Konkurstücken hat.

Die in diesem Kapitel verfolgte Idee ist das Ersetzen von Polygonrand-Bezugspunkte durch Punkte, die das Polygon charakterisieren. Wir können uns auf bereits existierende Ansätze wie [LA18] stützen. Einen solchen Punkt, der repräsentativ für eine charakteristische Fläche innerhalb des Polygons (eine sogenannte Feature Area) steht, nennen wir *Area Feature Point* oder kurz Feature Point. Dazu extrahieren wir aus der Form des Polygons Punkte, die eine besondere formelle Ausprägung des Polygons repräsentieren. In den Beispielen der Abbildung 4.11 könnten beispielsweise die Endpunkte der Polygonausläufer mögliche Featurepoints sein. Eine beispielhafte farbliche Zuordnung kann der Abbildung 4.13 entnommen werden.



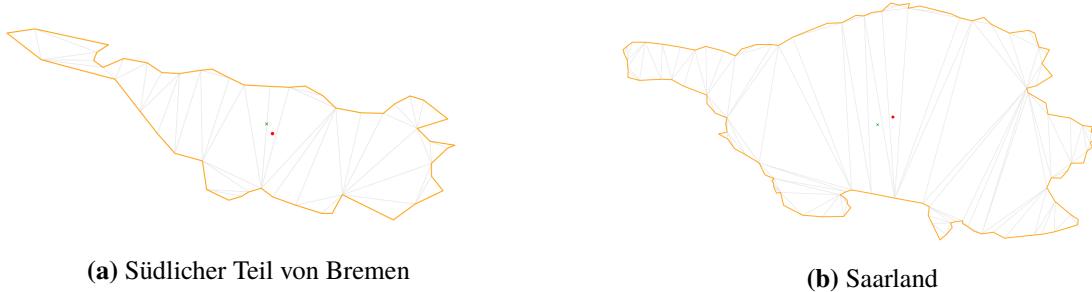
**Abbildung 4.13:** Mögliche Einteilung des Polygons aus Abbildung 4.11a in Feature Areas

Wären wir in der Lage, die Feature Areas exakt zu bestimmen, könnten wir angrenzende Flächen verbinden und berechnen, welche Fläche möglichst viele andere Flächen verbindet, oder welche Fläche innerhalb des gesamten Polygons eine zentrale Rolle spielt. In unserem grafischen Beispiel aus dem letzten Absatz (Abbildung 4.13) sind die grau markierten Flächen jene, die das Polygon in seiner Gänze zusammenhalten. Von ihnen gehen die Ausläufer oben, links, rechts und nach unten weg. Es wäre also naheliegend, die Punkte innerhalb der grauen Flächen als Mittelpunktkandidaten in Betracht zu ziehen. Insbesondere könnte der Mittelpunkt der Geraden, die die grauen Flächen separiert, ein guter Kandidat sein.

Wir werden in den folgenden Absätzen diesen Ansatz konkretisieren, indem wir das Problem in drei Stufen lösen. Initial werden wir das Polygon in Dreiecke aufteilen, um die Gesamtfläche des Polygons aufzuteilen. Im zweiten Schritt verbinden wir benachbarte Teilstücke und bauen eine Graph-Struktur auf, bevor wir im letzten Schritt diese mit geeigneten Zentralitätskriterien untersuchen und Kandidaten für mögliche Mittelpunkte extrahieren.

### Phase 1: Triangulierung

Um die Polygonfläche stückweise aufzuteilen, zerlegen wir das Polygon in Dreiecke. Dabei sind die Flächenstücke nicht notwendigerweise gleich groß. Wir argumentieren, dass viele Polygonpunkte auf engem Raum auf eine größere Änderung der Richtung hindeuten, oder als Mittel zur Darstellung von Details fungiert. Abbildung 4.11a unterstützt den Gedankengang: In der blau markierten Teilfläche konzentrieren sich die Punkte auf die Rundung am Ende, während die langgezogene primitive Fläche mit wenigen Punkten auskommt.



**Abbildung 4.14:** Beispielhafte Triangulierungen (hellgrau: Triangulierung, rot: geometrischer Mittelpunkt, grün: gewichtete Summe)

Für unsere Berechnungen greifen wir auf das C++ Framework `poly2tri`<sup>2</sup> zurück, das uns eine Constrained Delaunay Triangulierung liefert. Für eine Einbindung in unsere Gesamtberechnungen nutzen wir den gleichnamigen Rust-Wrapper<sup>3</sup>.

Prinzipiell ist eine Triangulierung auch für Polygone mit Löchern möglich, um die Berechnungen innerhalb von Abschnitt 4.5 zu beschleunigen, beschränken wir uns im Folgenden auf Polygone ohne Löcher. Wir werden diese Einschränkung in Kapitel 5 berücksichtigen.

## Phase 2: Graph-Konstruktion

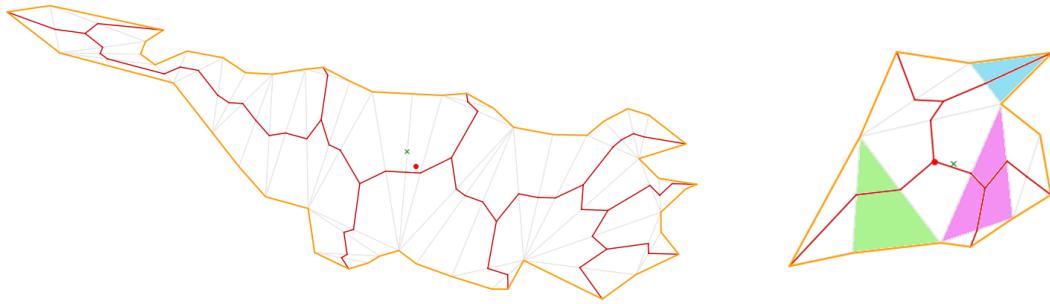
Im Rahmen dieser Arbeit haben wir zwei Methoden implementiert, die Dreiecke als Graph miteinander verbinden. Um die Verfahren voneinander abgrenzen zu können, verwenden wir sie unter explizit verschiedenen Namen. Zu Beginn untersuchen wir den Graphen der in dem Paper [AAJ15] beschrieben ist. In der Veröffentlichung wird zudem der Einfluss verschiedener Triangulierungstypen auf den Graphen geschildert. In diesem Kontext benennen wir den nach diesem Verfahren entstandenen Graphen gemäß deren Autoren *AAJ-Graph*. Anschließend werden wir die grundsätzliche Idee aufgreifen und für unsere Bedürfnisse anpassen. Das Ergebnis wird den Namen *FMI-Skelettierung* tragen.

### AAJ-Graph

Zunächst klassifizieren wir die Dreiecke in drei Typen. Jedes Polygon der Triangulierung besteht aus drei Kanten, von der jede entweder bei der Triangulierung neu (künstlich) hinzugefügt oder als Teil der Outer-Polygonkantenmenge wiederverwendet wurde. Aigner et al. benennen ein Dreieck als *ear triangle* wenn es eine, als *link triangle* wenn es zwei, und als *branch triangle* wenn es drei künstliche Kanten besitzt. Anschließend werden Berechnungsvorschriften für jeden der drei Typen benannt, auf die wir in den nächsten Absätzen eingehen werden. Jede Berechnungsvorschrift gibt einen ungerichteten zusammenhängenden zyklusfreien Graphen  $G_{\Delta_i} = (V_i, E_i)$  des Dreiecks  $\Delta_i$  zurück. Eine wichtige Eigenschaft der Teilgraphen ist, dass sich adjazente liegende Dreiecke jeweils

<sup>2</sup><https://github.com/greenm01/poly2tri>

<sup>3</sup><https://docs.rs/poly2tri/0.1.0/poly2tri/>



**Abbildung 4.15:** AJJ-Skelettierungen mit beispielhaft eingefärbten Ear- (cyan), Link- (mint) und Branch-Dreiecken (magenta)

genau einen Graphen-Knotenpunkt teilen. Folglich bildet der Graph  $G_{\Delta_{a,b}}$  bestehend aus den zwei benachbarten Graphen  $G_{\Delta_a}$  und  $G_{\Delta_b}$  weiterhin einen Baum:

$$G_{\Delta_{a,b}} = (V_a \cup V_b, E_a \cup E_b) \text{ sowie } |V_a \cap V_b| = |v_{ab}| = 1, E_a \cap E_b = \emptyset$$

Per Induktion können wir leicht zeigen, dass der Graph der entsteht, indem alle Teilgraphen benachbarter Dreiecke zusammengesetzt werden, ebenfalls die Baumeigenschaften erfüllt.

Einem Ear-Triangle, das eine wichtige Rolle für unsere Feature-Part Analyse spielen wird, wird ein Teilgraph  $G_{\Delta_e}$  – bestehend aus einer Kante – zugeordnet:  $V_e = \{P_e, M_{\bar{e}}\}, E_e = \{\{P_e, M_{\bar{e}}\}\}$  Hierbei beschreibt  $M_{\bar{e}}$  den Mittelpunkt der künstlich hinzugefügten Diagonalen.  $P_e$  erhält die Koordinaten des Polygonpunktes, der sich im Dreieck gegenüber dem Punkt  $M_{\bar{e}}$  befindet. Da die Punkte  $P_e$  von Bedeutung sein werden, erhalten auch diese (wie in [AAJ15]) einen eigenen Namen: *ear vertex*.

Jedem Link-Triangle  $\Delta_l$  wird ein Teilgraph  $G_{\Delta_l}$  zugeordnet. Benennen wir die künstlichen Dreiecks-Kanten  $r$  und  $s$ , erhalten deren Mittelpunkte die entsprechende Bezeichnung  $M_r$  und  $M_s$ . Das reicht bereits aus, um den simplen Teilgraphen  $G_{\Delta_l}$  zu konstruieren:  $G_{\Delta_l} = (\{M_r, M_s\}, \{\{M_r, M_s\}\})$ .

Für den dritten Dreieckstyp, bestehend aus den künstlich hinzugefügten Diagonalen  $o, p$  und  $q$ , lässt sich der Teilgraph  $G_{\Delta_b}$  eines Branch-Triangles bilden. Zuvor berechnen wir den Mittelpunkt des Dreiecks, den wir als  $C_b$  bezeichnen. Als Rückgabe erhalten wir  $E_b = \{\{M_o, C_b\}, \{M_p, C_b\}, \{M_q, C_b\}\}$  mit den entsprechenden Knoten  $V_b = \{M_o, M_p, M_q, C_b\}$ . Analog zu Ear- und Link-Graphen entspricht  $M_o$  dem Mittelpunkt der Strecke  $o$ .

### FMI Skelettierung

Das Verfahren nach diesem ursprünglichen Ansatz garantiert uns eine Graphstruktur innerhalb der Polygongrenzen. Wir werden diese Garantie zugunsten eines glätteren Ergebnisses in den kommenden Ansätzen aufgeben. Betrachten wir die Ergebnisse der Skelettierung nach dem AJJ-Ansatz, stellen wir ein charakteristisches Zickzack-Muster fest. Dieses tritt typischerweise auf, da die Dreiecke einer zum Teil starken Deformation ausgesetzt sind. Die meisten der Dreiecke – insbesondere die Link-Dreiecke – haben eine kurze Grundseite (nämlich die Seite zwischen zwei Outer-Punkten) und zwei sehr lange Kanten (welche die Diagonalen durch das Polygon darstellen). Da der Mittelpunkt eines einzelnen Dreiecks so gewählt wurde, dass es dessen geometrischen Schwerpunkt entspricht, kann es zu starken Schwingungen kommen, wenn lange dünne Dreiecke

sich angrenzend mit der Polygonseite gegenüberliegen. Ein Beispiel wurde in Abbildung 4.16a aufbereitet. Derselbe Effekt wird auch in der Arbeit von McAllister und Snoeyink beschrieben, zudem werden weitere Alternativen vorgestellt[MS00].

Für unseren Ansatz greifen wir diese Feststellung auf und passen unsere Graphstruktur entsprechend an. Wir erhalten für jedes Dreieck einen Kanten-bezogenen Höhen-Mittelpunkt: Für das Dreieck  $\Delta_d$  bestimmen wir zu Beginn dessen kürzeste Seite  $c_d$  sowie deren gegenüberliegenden Punkt  $E_d$ . Wir berechnen den Mittelpunkt  $M_d$  der Strecke  $c_d$  und interpolieren anschließend  $M_d$  und  $E_d$  mittig. Den entstandenen Punkt nennen wir  $C_d$ .

Wir erhalten den Teilgraphen eines Ear-Triangel  $\Delta_e$ , indem wir eine Kante von dem Ear-Vertex  $P_e$  (siehe Abschnitt 4.5) zu  $C_e$  ziehen. Außerdem fügen wir noch eine Kante von  $C_e$  zu  $C_k$  ein. Hier ist  $C_k$  der Höhen-Mittelpunkt des adjazenten Link- oder Branchdreiecks.

Den Teilgraphen eines Link-Dreiecks  $\Delta_l$  erhalten wir durch Hinzufügen der Kanten  $P_l$  nach  $P_r$  und  $P_l$  nach  $P_s$ . Auch hier sind  $\Delta_r$  und  $\Delta_s$  wieder angrenzende Dreiecke von  $l$ .

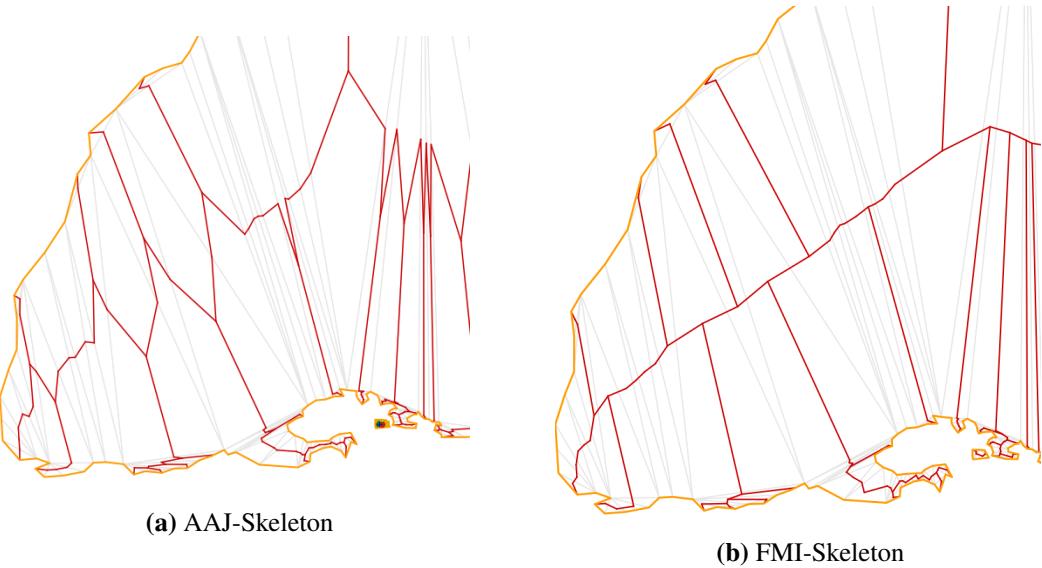
Nach demselben Verfahren generieren wir das Dreieck-Skeleton eines Branch-Dreiecks  $\Delta_b$ . Benötigt werden lediglich die Höhenmittelpunkte der drei umliegenden Dreiecke.

Offensichtlich ist der Schnitt beim Zusammenfügen zweier Teilgraphen nicht leer, durch das Verbot von Mehrfachkanten ist die Baumeigenschaft dennoch erfüllt:

$$G_{\Delta_a,b} = (V_a \cup V_b, E_a \cup E_b) \text{ mit } |V_a \cap V_b| = |\{C_a, C_b\}| = 2, E_a \cap E_b = \{\{C_a, C_b\}\}$$

Bei unserer Implementierung verwenden wir eine HashSet, um bereits bearbeitete Dreiecke zu verwalten. Außerdem knüpfen wir das Einfügen von Nachbarschaftskanten an die Bedingung, dass sich das angrenzende Dreieck im Zustand *unbearbeitet* befinden muss. Dadurch entsteht derselbe Graph implizit.

Neben der optisch visuell ansprechenderen Aufteilung der Graphkanten, werden wir bei der späteren Knotenanalyse weitere Vorteile des FMI-Verfahrens vorstellen. Zum aktuellen Zeitpunkt halten wir lediglich fest, dass die verwendeten Kanten die äußere Polygonstruktur besser repräsentieren als die des AAJ-Ansatzes, da im Gegensatz dazu weniger Strecke zurückgelegt wird. Die Kanten adjazenter Dreiecke verlaufen direkt und weisen weniger Höhenunterschied auf.



**Abbildung 4.16:** Skeleton-Typen am Beispiel von Süd-West-Baden-Württemberg

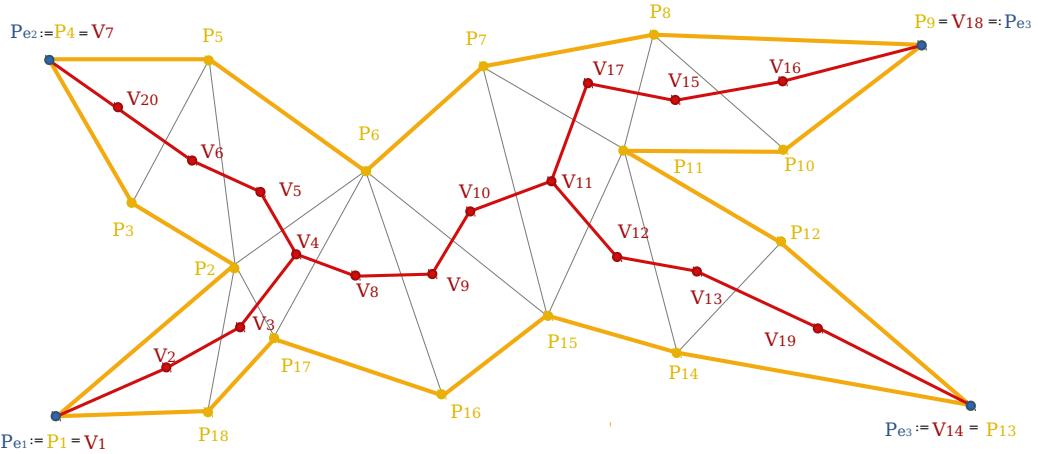
### Phase 3: Zentralitätsanalyse

In den bisherigen Phasen haben wir eine von der Fläche abhängige Graphstruktur aufgebaut. Insbesondere sind Kantenlängen und -orientierung stark von den darunterliegenden Dreiecken bestimmt, die sich wiederum an dem Outer-Polygon orientieren. Die letzte Phase zielt darauf ab, den Graphen auf Grundlage von Heuristiken nach wichtigen Knoten zu untersuchen, die einen Rückschluss auf, in unserem Kontext gemessen, zentrale Komponenten des Polygons schließen lässt. Wir erinnern in diesem Zusammenhang noch einmal an das Beispiel in Abbildung 4.13, bei dem der darunterliegende Graph wünschenswerterweise innerhalb der grauen Flächen markante Punkte aufweist.

In Anlehnung der theoretischen Überlegung in [LA18] implementieren wir in unseren Algorithmus zwei Metriken, die jedem Knotenpunkt des Graphen einen Score zuweisen. Wir greifen die Terminologie hier auf und unterscheiden die Metriken in *Betweenness*- und *Closeness-Centrality*. Beide Methoden sind der Graphentheorie entnommen und sind wichtige Werkzeuge der Zentralitätsanalyse [Mül08].

Bevor wir die beiden Heuristiken vorstellen, geben wir vorab eine Übersicht der verwendeten Begrifflichkeiten. Wir benennen die Punkte des Polygons, auf dem unsere Berechnungen resultieren,  $P_1, P_2, \dots, P_n$ . Die Triangulierung  $T$  unseres  $n$ -Punkte-Polygons liefert uns  $n - 2$  Dreiecke. Auf dieser Grundlage berechnet uns die Skelettierung den Dualgraphen  $G_T = (V_T, E_T)$ . Wir erhalten als Resultat die Knoten  $v_1, v_2, \dots, v_{|V_T|}$ . Wie bereits in Abschnitt 4.5 angedeutet, werden die Graph-Endpunkte  $P_{e_i} \in V_T$  eine besondere Rolle spielen, da diese dieselbe Koordinaten wie die zugehörigen Outer-Polygonpunkte aufweisen. Wir gehen im Folgenden davon aus, dass die Triangulierung  $m < n$  Endknoten (Earpoints) bestimmt. Abbildung 4.17 soll die in den kommenden Absätzen beschriebenen Verfahren gedankenstützend untermalen.

Widmen wir uns nun den beiden Verfahren.



**Abbildung 4.17:** Das Polygon  $\mathcal{P} = P_1, P_2, \dots, P_n$  (hier  $n = 18$ , orange) bringt auf Triangulierung  $T$  (grau) den Skeletongraphen  $G_T = (V_T, E_T)$  ( $|V_T| = |\{v_1, v_2, \dots\}| = 20$ , rot) hervor. Es entstehen  $m = 4$  Ear-Points  $P_{e1}, \dots, P_{e4}$  (blau).

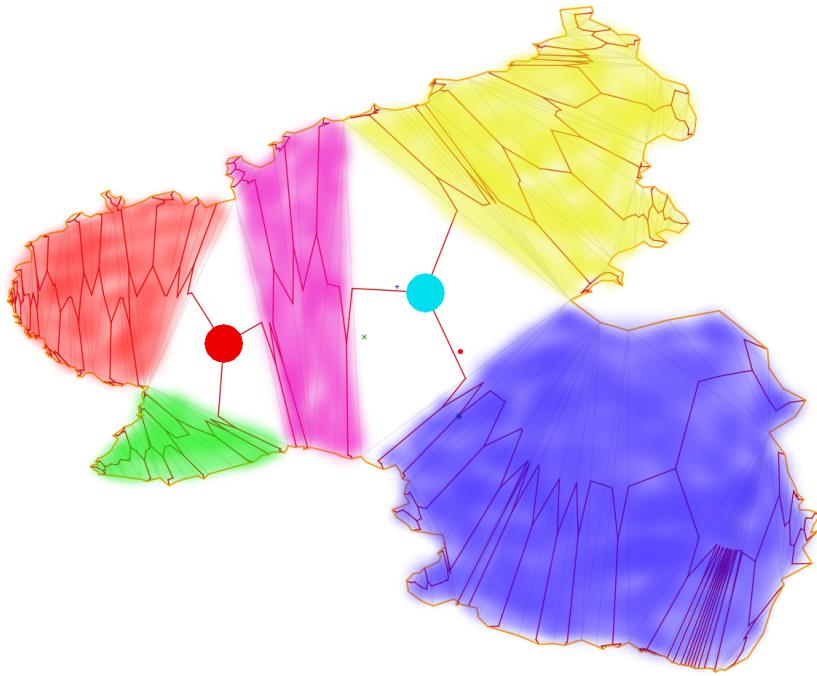
### Betweenness-Centrality

Die erste Methode zur Bewertung der Knoten zielt darauf ab, Punkte nach deren Position innerhalb des Graphen einzurichten. Die Idee der Betweenness-Heuristik ist Folgende: Punkte, die mehrere Zusammenhangskomponenten miteinander verbinden, sind zentraler als Punkte, die beispielsweise lediglich als Endpunkte fungieren. Um ein schönes Beispiel innerhalb unserer Testgraphen miteinzubeziehen, verweisen wir auf Abbildung 4.18. Hier wurde das Bundesland Rheinlandpfalz gemäß visuellen Zusammenhangskomponenten eingefärbt. Die zwei Punkte repräsentieren Knoten mit hohen Score-Werten. Es lässt sich leicht erkennen wie diese Knoten die signifikanten Flächenteile des gesamten Graphen zusammenhalten.

Wir gehen davon aus, dass die  $P_{e_i}$  Endpunkte einer zusammenhängenden Komponente darstellen. Durch Branch- und Link-Dreiecke werden wiederum kleine Komponenten zu größeren verbunden. Für unsere Betrachtungen suchen wir nun Knoten, die möglichst weit oben in dieser Verbindungs-Hierarchie stehen. Zu diesem Zweck berechnen wir für jeden Knoten  $v_i$  innerhalb des Graphen den sogenannten Betweenness-Score  $\Xi_{v_i}^B$ . Er soll hoch sein, wenn der Knoten viele Komponenten (transitiv) verbindet, und niedrig vice-versa. Zu jedem Knoten  $v_i$  erhalten wir demnach den Betweenness-Score  $\Xi_{v_i}^B$ , indem wir die kürzesten Wege zählen, die durch  $v_i$  verlaufen [LA18]:

$$\Xi_{v_i}^B = \sum_{(r,s) \in \text{pairs}(m)} X_{r,s}(v_i) \quad (4.7)$$

Die Funktion  $\text{pairs} : \mathbb{N} \rightarrow \mathbb{N} \times \mathbb{N}$  liefert uns die gewünschten, paarweise verschiedenen Endpunkt-Kombinationen:  $m \mapsto \{(r, s) | 1 \leq r < s \leq m\}$ . Die Funktion  $X_{r,s} : V_T \rightarrow \{0, 1\}$  wird genau dann eins, wenn der Knoten  $v_i$  im kürzesten Weg von  $P_{e_r}$  nach  $P_{e_s}$  (bzw. vice-versa) vorkommt.



**Abbildung 4.18:** AJJ-Skelettierung mit beispielhaft eingefärbten Zusammenhangskomponenten. Die hervorgehobenen Punkte in rot und cyan stellen Knoten mit besonders hohen Betweenness-Scores dar.

### Closeness-Centrality

In diesem Ansatz konzentrieren wir uns auf die Suche nach einem Punkt, der innerhalb des Graphen möglichst nächstgelegen zu allen anderen liegt. Um den Unterschied zum ersten Verfahren zu verdeutlichen, stellen wir uns ein Polygon vor, das sehr viele Details auf der rechten Hälfte des Polygons und sehr grobe, kantige Komponenten auf der linken hat. Gemäß unseres ersten Ansatzes würden wir versuchen, einen Punkt zu wählen, der möglichst viele Komponenten transitiv verbindet. Durch den Detailreichtum entstehen viele Ear-Points auf der rechten Seite. Dadurch wird sich der Punkt mit dem höchsten Betweenness-Score höchstwahrscheinlich deutlich im rechten Teil befinden.

Um unabhängig von der konkreten Punkteverteilung zu bleiben, bietet es sich an, einen Punkt zu finden, der innerhalb aller Komponenten so liegt, dass die Abstände zu allen Ear-Points in etwa dieselben sind. So würden wir den Punkt aus unserem gedanklichen Beispiel in etwa der Mitte erwarten dürfen.

Den Abstand von einem Punkt zu einem Ear-Punkt bestimmen wir durch die Länge des kürzesten Pfads. Wollen wir einen Punkt auf dem Graphen finden, der zu allen anderen Ear-Punkten in etwa den gleichen Abstand hat, können wir äquivalent dazu, einen Punkt suchen, der die Standardabweichung der kürzesten Pfade minimiert. Diese Überlegungen gießen wir nun in die Formel zum Berechnen des Closeness-Scores eines inneren Konten  $v_i$  [Fra; Hen10; LA18]:

$$\begin{aligned}\mathfrak{S}_{v_i}^C &= (-1)std \left( d_{eucl}(v_i, P_{e_j}) \right) \\ &= -\sqrt{\frac{1}{m-1} \sum_{j=1}^m \left( d_{eucl}(v_i, P_{e_j}) - \bar{d}_{eucl}(v_i) \right)^2}\end{aligned}\tag{4.8}$$

Wir bezeichnen die Distanz des kürzesten Weges von  $v_j$  nach  $P_{e_j}$  mit  $d_{eucl}(v_i, P_{e_j})$ . Hierbei verwenden wir als Gewichte der Kanten aus  $E_T$  deren jeweilige euklidische Länge. Weitere Gewichtungs-Maße wie beispielsweise die partiellen Flächen der Dreiecke, die eine Kante verbindet, sind denkbar und werden in [LA18] ausgiebig diskutiert. Analog stellt  $\bar{d}_{eucl}(v_i)$  das arithmetische Mittel aller kürzesten Wege zu Ear-Points ausgehend von  $v_i$  dar.

Entgegen der üblichen Notation, verwenden wir in unserer Arbeit aus numerischen Gründen nicht den Kehrwert der Standardabweichung, sondern deren negativen Betrag. Bei Eingabe einer regelmäßigen Geometrie könnte die Abweichung 0 werden. Zudem sind wir nicht an den genauen Werten, vielmehr an dem maximalen (oder den  $k$ -maximalen) aller  $\mathfrak{S}_{v_i}^C$  interessiert.

## Laufzeiten

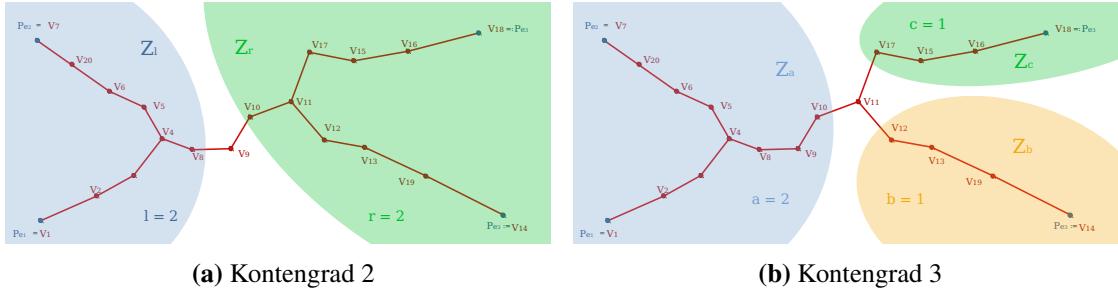
Die Rechenzeit in Phase *I* hängt hauptsächlich von der Laufzeit der poly2tri-Berechnung ab. Während die theoretischen Laufzeiten für die Delaunay-Triangulierung im Bereich  $O(n \log(n))$  liegen[SD97] (und eine Triangulierung von simplen Polygonen im Allgemeinen sogar in linearer Zeit[Cha91] berechnet werden kann), begnügen wir uns mit schlechteren oberen Schranken, wie  $O(n^2)$ , da die späteren Analyse-Methoden diese Zeitschranke sowieso nicht unterbieten werden.

Die Phase-*II*-Laufzeit hängt ausschließlich von der Anzahl an Dreiecken der Triangulierung ab. Die Triangulierung liefert uns exakt  $n - 2$  Dreiecke [Düv09; VI01]. Da wir durch die Verwendung einer geeigneten Datenstruktur über jedes Dreieck einmalig mit konstantem Aufwand iterieren, beschränkt uns das die Laufzeit linear.

In Phase *III* erhalten wir je nach Verfahren unterschiedliche Laufzeiten. Widmen wir uns zuerst dem Verfahren zur Bestimmung der Betweenness-Centrality-Scores. Hierfür müssen wir für jeden Knoten  $v_i$  die Anzahl an kürzesten Wege bestimmen, die ihn kreuzen. Naiv müssen wir also alle kürzesten Wege von  $P_{e_r}$  nach  $P_{e_s}$  für  $(r, s) \in pairs(m)$  bestimmen. Das sind  $|pairs(m)| = \binom{m}{2} = \frac{m(m-1)}{2}$  Wege. Berechnen wir nun für unseren Graphen mit  $n - 2$  inneren Knoten und  $m$  äußeren Knoten (und demnach  $m + (n - 2) - 1$  Kanten) eine Dijkstra-Suche an, benötigen wir pro Pfad im Worst Case mindestens  $(|V_T| + |E_T|) \cdot \log(|V_T|) = ((n - 2 + m) + (n - 3 + m)) \cdot \log((n - 2 + m)) \in O((n + m) \log(n + m))$  Rechenschritte [WW13]. Für alle Pfade ergibt das eine Summe von  $O(m^2(n + m) \log(n + m))$ . Würden wir als pessimistische Schätzung annehmen, dass jeder tau-sendste Polygonknoten ein Ear-Knoten ist, verläuft der Algorithmus mit einer Worst-Case-Laufzeit  $O(n^3 \log(n))$  zwar noch im Rahmen der gewünschten Bedingungen in Abschnitt 3.1, schließt ihn für die praktische Anwendung auf Eingabe größerer Graphen allerdings nahezu aus.

Da wir für einen Suchlauf im schlechtesten Fall den ganzen Dijkstra-Baum aufbauen, bietet es sich an, statt  $\binom{m}{2}$  Einzelsuchen lediglich  $m - 1$  One-To-All Berechnungen durchzuführen. Das reduziert unsere Laufzeit zumindest auf  $O(m(n + m) \log(n + m))$ .

## 4 Mittelpunktberechnung



**Abbildung 4.19:** Visualisierung der Komponentenunterteilung am Beispielgraphen aus Abbildung 4.17

Abstrahieren wir das Problem weiter, so können wir die Struktur des Problems besser an unseren Algorithmus anpassen: Ziel ist es, die kürzesten Pfade zwischen allen Endpunkten des Baums zu berechnen. Da der kürzeste Weg zwischen zwei Punkten in einem Baum stets eindeutig ist und keine Schleifen vorhanden sind, reicht es, *einen* Pfad zu finden, mit dem keine Kanten doppelt verwendet wurden. Weiter noch, benötigen wir nicht die genaue Länge des Pfades, sondern nur dessen Verlauf, also die Menge an Knoten, die er durchläuft. Somit können wir die Anzahl der Pfade argumentativ bestimmen: Wir klassifizieren einen Knoten anhand der Anzahl seiner angrenzenden Kanten.

Offensichtlich gehen von jedem Endknoten mit Knotengrad eins  $m - 1$  kürzeste Wege aus. Daher können wir den Score  $\mathfrak{S}_{P_{e_i}}^B$  fix für jeden der  $m$  Konten auf  $m - 1$  setzen.

Fixieren wir als Nächstes einen Knoten  $v_x$  vom Grad zwei und dessen Anzahl an kürzesten Wegen, die ihn beinhalten. Offensichtlich verbindet der Knoten die disjunkten Zusammenhangskomponenten  $Z_l$  und  $Z_r$  (siehe Abbildung 4.19a). Nehmen wir an,  $Z_l$  hat  $l$  und  $Z_r$   $r$  äußere Knoten, so gehen gemäß der Baumeigenschaft  $lr$  Wege durch  $v$  (die gleichzeitig auch die kürzesten sind). Man beachte, dass sich der Algorithmus jeweils nur eine Knotenanzahl merken muss, indem man  $m = l + r$  benutzt. Wie können den Score als  $\mathfrak{S}_{v_x}^B = l \cdot (m - l) = r \cdot (m - r)$  angeben.

Für den dritten der möglichen Fälle sei der Knotengrad eines Knoten  $v_y$  drei, sodass der Graph in drei Zusammenhangskomponenten zerfällt, die wir  $Z_z$  mit  $z \in \{a, b, c\}$  nennen. Abbildung 4.19b skizziert den Sachverhalt. Nun verlaufen von jedem Ear-Knoten aus  $Z_a$  offensichtlich  $b$  viele kürzeste Wege nach  $Z_b$  und  $c$  viele nach  $Z_c$ . Weiterhin verlaufen durch den Knoten die  $b \cdot c$  vielen Wege zwischen den Knoten aus  $Z_b$  beziehungsweise  $Z_c$ . Für den Knoten ergibt sich der Gesamt-Score  $\mathfrak{S}_{v_y}^B = a \cdot b + b \cdot c + c \cdot a$ .

Unsere Überlegungen bringen uns auf den einfachen Algorithmus 4.1.

Grafisch können wir uns vorstellen, dass wir beginnend von den äußeren Knoten die Berechnungen nach innen propagieren. Für Knoten mit Grad eins (initial gegeben) und Grad zwei können wir den Score direkt berechnen. Für Knoten, die drei ausgehende Kanten haben, an denen wiederum drei Knoten hängen, führen wir nur eine Berechnung durch, falls zwei Knoten davon bereits bearbeitet wurden. Die Baum-Struktur garantiert uns dabei ein Terminieren. Offensichtlich kann ein Score erst bestimmt werden, wenn sich alle angrenzenden Konten bereits in der *closedList* befinden. Bei der Analyse der Code-Struktur in Algorithmus 4.1 fällt auf, dass zur Behandlung eines aus der Queue entnommenen Knoten konstant Zeit benötigt wird, um dessen Score zu berechnen. Es

**Algorithmus 4.1** Tree-Shrink-Algorithmus

---

```

procedure CALCULATEBETWEENNESSSCORES( $G_T = (E_T, T_V)$ , earpoints =  $\{P_{e_1}, P_{e_2}, \dots, P_{e_m}\}$ )
    closedList  $\leftarrow \emptyset$ 
    lockedQueue  $\leftarrow$  Queue.empty()
    ADDALLToQUEUE(earpoints, lockedQueue)
    while lockedQueue  $\neq \emptyset$  do
        vertex  $\leftarrow$  lockedQueue.dequeue()
        if vertex  $\in$  closedList then
            continue
        end if
        if deg(vertex) = 1 then                                // Earpoint
             $\Xi_{\text{vertex}}^B \leftarrow m - 1$ 
            representative(vertex) = 1
            closedList.put(vertex)
            adjacentVertex  $\leftarrow$  GETOPENNEIGHBOURVERTEX(vertex)
            lockedQueue.enqueue(adjacentVertex)
        else if deg(vertex) = 2 then                      // Zwei Zusammenhangskomponenten
            closedNeighbour  $\leftarrow$  GETCLOSEDNEIGHBOURVERTEX(vertex)
            r  $\leftarrow$  representative(closedNeighbour)
             $\Xi_{\text{vertex}}^B \leftarrow r \cdot (m - r)$ 
            representative(vertex)  $\leftarrow r$ 
            closedList.put(vertex)
            adjacentVertex  $\leftarrow$  GETOPENNEIGHBOURVERTEX(vertex)
            lockedQueue.enqueue(adjacentVertex)
        else if deg(vertex) = 3 then                      // Drei Zusammenhangskomponenten
            N  $\leftarrow$  GETCLOSEDNEIGHBOURVERTICES(vertex)
            if  $|N| \neq 2$  then
                continue
            end if
             $\{v_a, v_b\} = N$ 
            a  $\leftarrow$  representative( $v_a$ )
            b  $\leftarrow$  representative( $v_b$ )
            c  $\leftarrow m - (a + b)$ 
             $\Xi_{\text{vertex}}^B \leftarrow a \cdot b + b \cdot c + c \cdot a$ 
            representative(vertex)  $\leftarrow a + b$ 
            closedList.put(vertex)
            adjacentVertex  $\leftarrow$  GETOPENNEIGHBOURVERTEX(vertex)
            lockedQueue.enqueue(adjacentVertex)
        end if
    end while
end procedure

```

---

## 4 Mittelpunktberechnung

---

bleibt die Anzahl an *enqueue*-Befehlen zu bestimmen. Ein Nicht-Ear-Knoten wird dann zur Queue hinzugefügt, wenn dessen angrenzender Knoten in den Zustand *closed* wechselt, was höchstens  $O(n)$  mal vorkommt und uns gleichzeitig eine obere optimale Laufzeitschranke liefert.

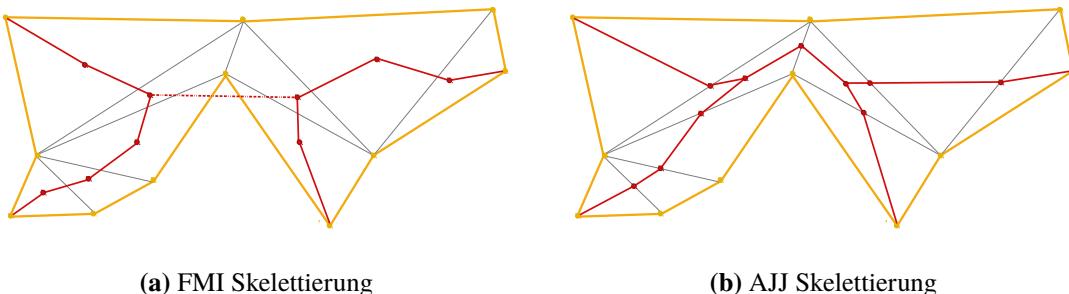
Für die Berechnung der Centrality-Scores  $\mathbb{S}_{v_i}^C$  gehen wir analog zum zweiten Ansatz der Betweenness-Scores vor. Wir berechnen dazu von jedem inneren Knoten die Distanzen zu allen Earpoints mithilfe des One-To-All-Dijkstra Algorithmus, wofür jeweils  $O(n \log(n + m)) = O(n \log(n))$  Schritte notwendig sind. Wie erhalten eine Schranke von  $O(n^2 \log(n))$ . Bei dem Untersuchen der Struktur des Baumes stellen wir zudem fest, dass die Suche von einem Punkt zu allen Endpunkten dem Traversieren des Baumes entspricht[LA18]. Für  $n$ -maliges Traversieren erhalten wir folglich eine verbesserte Schranke von  $O(n^2)$ .

### Evaluation

Im Zusammenhang mit Graph-Skelett-Berechnungen existieren viele Parameter, die das Endergebnis beeinflussen: Es kann die Triangulierung verändert, neue Skelett-Verfahren eingeführt, verschiedene Vorgehen bei der Zentralitätsanalyse gewählt werden und dort wiederum weitere Freiheitsgrade wie Kantengewichtungsfunktionen (wir erinnern hier beispielhaft an Gewichtungsfunktionen wie euklidische Kantenlänge oder partielle Dreiecksflächen) angepasst werden. Für eine konkrete Implementierungsumgebung lohnt es sich, Testpolygone zu generieren, um die Parameter entsprechend zu trainieren. Für unsere theoretische Betrachtung werden die genannten Möglichkeiten durchgegangen und die jeweiligen Vor- und Nachteile herausgearbeitet.

Bei der Bestimmung einer Dreiecksanordnung, kann aus  $C_n$  vielen verschiedenen Triangulierungen ausgewählt werden, wenn  $C_i$  die i. Catalan-Zahl ist [LRS10]. Wir verweisen hierzu auf die Veröffentlichung von Aigner, Aurenhammer und Jüttler, die die Auswirkungen verschiedener Triangulierungen auf die Skelettform untersuchen[AAJ15].

Aktuell werden lediglich Eingaben betrachtet, die aus einem gültigen äußeren Polygon bestehen. Das Verfahren kann daher insbesondere nicht mit Polygonen umgehen, die mindestens ein Loch aufweisen.



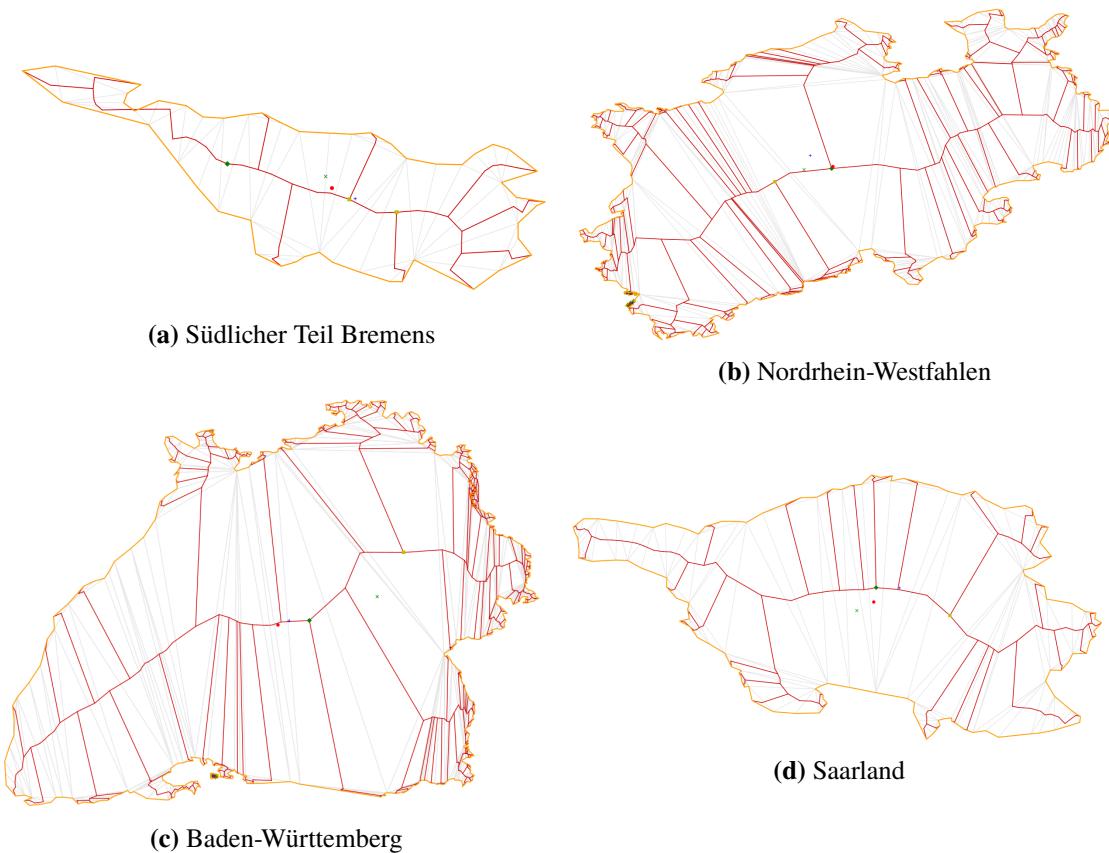
**Abbildung 4.20:** Unterschiedliche Skelettierungen

Es wurden zwei mögliche Graph-Strukturen vorgestellt, die sich beide linear in der Anzahl der Dreiecke, also insbesondere linear in  $n$  bestimmen lassen. Die AJJ-Skelettierung ist sehr robust und hat gegenüber der FMI-Variante den Vorteil, stets innerhalb der Dreiecksflächen zu verlaufen.

Auf besondere Eingaben kann die FMI-basierte Skelettierung eine Polygonkante schneiden. Abbildung 4.20 skizziert diesen Sachverhalt. Die Berechnungsregeln stellen jedoch sicher, dass sich die Graph-Knoten immer innerhalb des Dreiecks befinden und die Ergebnisse der Graphanalyse folglich ebenfalls.

Das FMI-Verfahren hat gegenüber der AJJ-Variante den Vorteil, glattere Übergänge zwischen benachbarten Dreiecken zu ermöglichen. Die Abbildung 4.16 bringt das zum Ausdruck. Durch die Glattheit des Graphen ist die Kantenlängen-Summe kleiner.

Abbildung 4.21 zeigt exemplarisch Ergebnisse unserer Eingabe. Auffallend ist, dass der Closeness-Centerpoint in der Umgebung des Schwerpunktes liegt. Eine Erklärung hierfür ist, dass die Eingabepolygone wenig entartet sind, und die Pfadmitte in etwa der Flächenmitte entspricht. Der Betweenness-Score ist prinzipiell nicht eindeutig, wie Abbildung 4.21b zeigt. In deren Eingabe gibt es zwei Graphknoten, die gleich oft Teil eines kürzesten Weges sind. Für Polygone in ‘I’-Form, die über lediglich zwei Ear-Knoten verfügen, ist jeder Knotenpunkt Teil der Ergebnismenge. In Kapitel 5 wird durch das Vergrößern von Dreiecken eine geeignete Abstraktion eingeführt, um insbesondere die Betweenness-Ergebnisse weiter zu verbessern.



**Abbildung 4.21:** Beispielhafte Ergebnisse der Zentralitätsanalyse (grüner Punkt: Closeness Top-Score, gelbe Punkte: Betweenness Top-Scores)

#### 4 Mittelpunktberechnung

---

Auch wenn die Laufzeit deutlich oberhalb der linearen Schranke liegt, garantiert das Verfahren in akzeptabler Zeit stabile Ergebnisse.

Später werden beide Bewertungsfunktionen parallel verwendet und der optimale Punkt unter den Kandidaten ermittelt.

## 5 Finale Lösung

Zusammenfassend aus Kapitel vier existiert kein Verfahren, das alle Kriterien restlos zufriedenstellend erfüllt. Jeder der analysierten Algorithmen zeichnet sich je nach Art der Eingabe als bevorzugt oder benachteiligt ab.

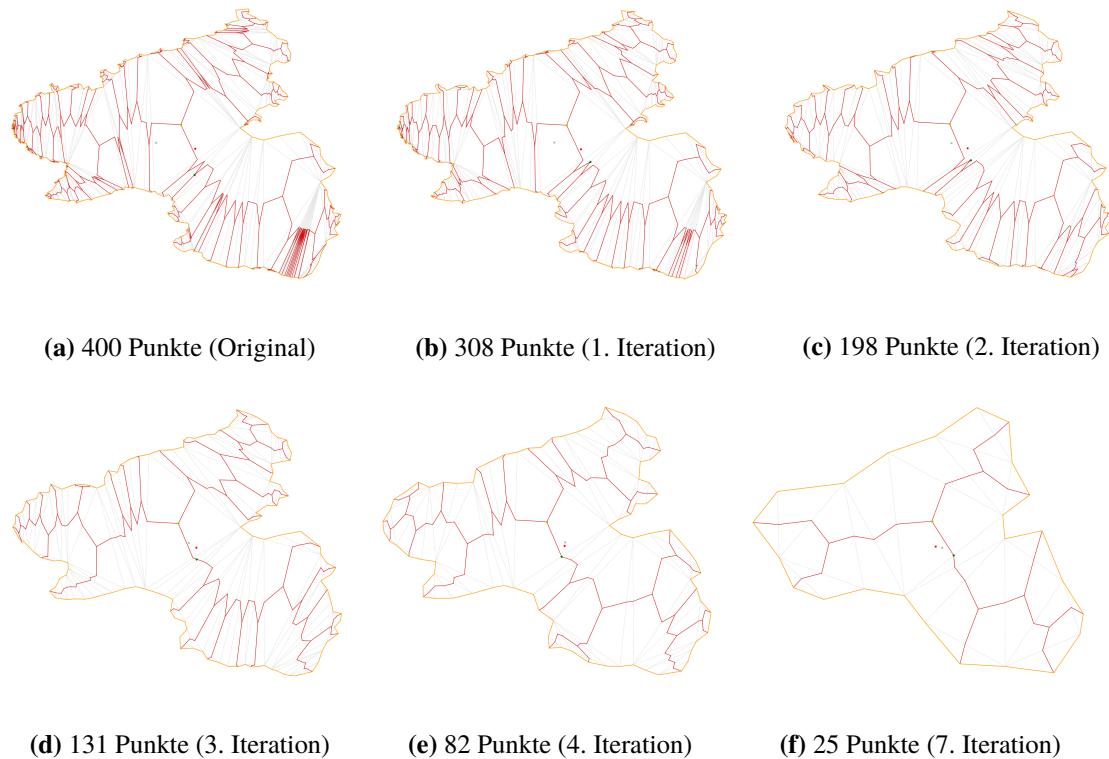
Um dennoch gute Resultate liefern zu können, werden die Verfahren eingabesensitiv eingesetzt. Dazu gehen wir in drei Schritten vor. Initial werden die Polygoneingaben in die in Abschnitt 3.2.1 vorgesehene Ausgangsform umgewandelt. Ohne die Details vorwegzunehmen, wird im zweiten Schritt gegebenenfalls eine Vereinfachung durchgeführt, um insgesamt stabilere Ergebnisse zu erhalten. Um die genaue Umsetzung kümmern wir uns im folgenden Abschnitt *Polygondetailreduktion*. Als finaler Schritt wird die Eingabe auf Charakteristika analysiert, die für oder gegen eine bestimmte Berechnungsvariante sprechen, um die Anfrage an die Verfahren entsprechend zu delegieren.

### Polygondetailreduktion

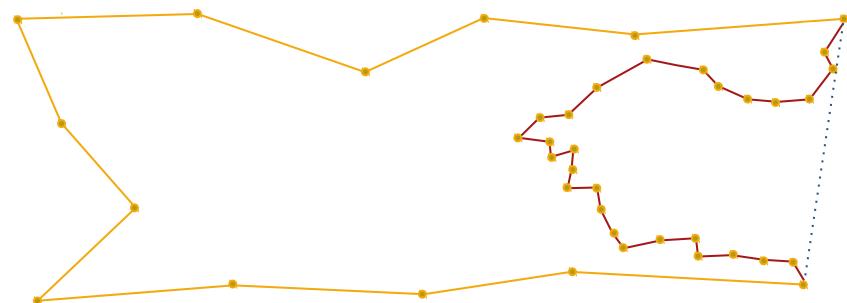
Motiviert wird dieses Kapitel mit einem Berechnungsbeispiel der Polygon-Skelettierung. Betrachten wir dazu Abbildung 5.1a, in der sich zur besseren Darstellung des Detailreichtums der Eingabe für die AJJ-Skelettierung entschieden wurde. Die feingranulare Auflösung des Polygons, bezogen auf seine eingenommene Fläche, bezahlen wir mit der Konsequenz, dass die Dreiecke der Triangulierung in vielen Fällen stark entarten. Der süd-westliche Teil des Polygons bezeugt dies musterhaft. Durch die ungleiche Verteilung der Dreiecksgrößen sind die Strecken, die ein Maß für die Struktur des Polygons darstellen, nicht mehr repräsentativ. Dadurch verlagert sich der Betweenness-Score (in grün dargestellt) der detailreichen Regionen entgegen in südliche Richtung.

Wir werden dieses Problem im Folgenden angehen, indem zwei Dinge gezeigt werden. Erstens kann die Struktur einer Fläche mit verhältnismäßig wenig Punkten überraschend gut approximiert werden. Zweitens führen weniger Outer-Punkte zu gleichmäßigeren Dreiecken. Dies unterstützt wiederum die Aussagekraft der Zentralitäts-Analyse signifikant.

Unser Ziel wird es sein, die Dreieckanzahl bei gleichbleibender Fläche zu reduzieren, indem wir Eingabepunkte des Polygons miteinander verschmelzen. In unserer beispielhaften Implementierung wird dazu das arithmetische Mittel aller Strecken benachbarter Punkte berechnet und jede Strecke zu einem Punkt kollabiert, deren Länge deutlich unter dem Durchschnitt liegt. Wichtig ist jedoch, dass beim Zusammenfügen der Punkte darauf geachtet wird, mehrfaches Kollabieren an aufeinanderfolgenden Strecken zu unterbinden. Würden wir naiv Punkte, die zu kürzeren Strecken der Abbildung 5.2 gehören, verschmelzen, erhielten wir das in orange-blau markierte Ergebnis. Daher wird mindestens eine Strecke übersprungen, sobald ein Merge durchgeführt wurde. Das Ergebnis der Detailreduktion ist in Abbildung 5.1 festgehalten. Dieses wird in Folge von Iterationen jeweils linearer Laufzeit berechnet, indem die durchschnittliche Länge der Kanten bestimmt und die Punkte anschließend gemäß Vorschrift zusammengefügt werden.



**Abbildung 5.1:** Verschieden detaillierte Rheinland-Pfalz' AJJ-Skelettierungen mit den Ergebnispunkten der geometrische Mitte (rot), Betweenness- (gelb) und Closeness-Analyse (grün)



**Abbildung 5.2:** Naiv würden detailreiche Streckenzüge (rot) ausgelöscht werden (neue Linie in blau)

---

**Algorithmus 5.1** Meta-Ebene der Mittelpunkt-Berechnung

---

```
procedure CALCULATELABELPOINT(outerSegments, innerSegments)
     $[\mathcal{F}_1, \mathcal{F}_2, \dots, \mathcal{F}_f] \leftarrow \text{CONVERTTOPOLOGICALLY}(\text{outerSegments}, \text{innerSegments})$ 
     $C = (x_c, y_c) \leftarrow \text{GETCENTERBYDELEGATEDMETHOD}([\mathcal{F}_1, \mathcal{F}_2, \dots, \mathcal{F}_f])$ 
    return  $(x_c, y_c)$ 
end procedure
```

---

Wie aus den Ergebnissen visuell hervorgeht, ändert sich die grobe Struktur des Polygons bis zur viersten Iteration kaum. Dies kann insbesondere daran beobachtet werden, dass der Flächenschwerpunkt in den ersten Iterationen eine konstante Position beibehält. Gleichzeitig verändert sich der Skelett-Graph des Polygons signifikant. Nicht nur die Anzahl an Kanten verringert sich drastisch, vielmehr deren Aussagekraft. Das spiegelt sich auch merklich in den Zentralitäts-Analyse-Ergebnissen wider: Während der Betweenness-Mittelpunkt zu Beginn suboptimal liegt, konvergiert er bei fortlaufender Detailreduktion in Richtung des geometrischen Mittelpunktes.

Zu den besseren Ergebnissen in Bezug auf Zentralität über einem Skelett verbessert sich auch die Laufzeit für praktische Anwendungen. Kann die Eingabegröße auf 30% reduziert werden, kommt uns der quadratische Faktor der One-to-All Berechnungen zu Gute, indem die Laufzeit der Bewertungsanalyse damit um mindestens 91% gesenkt wird.

Für unsere Testdaten haben wir eine Punkte-Reduktion zwischen 71.1% und 84.4% (Im Schnitt 78.73%) je Polygon als optimal bewertet. Außerdem wurden lediglich Polygone an Detailgrad reduziert, die zu Beginn eine Mindestanzahl ( $n > 100$ ) an Punkten aufweisen konnten.

## Polygon-Klassifikation

Zur finalen Anwendung unserer Primitiv-Ansätze konzentrieren wir uns an dieser Stelle auf die Komposition, die den finalen Algorithmus (Algorithmus 5.1) darstellt.

Die Pipeline aus Kapitel 2 liefert uns zwei Mengen an Kanten, jeweils eine für Inner- und Outer-Segmente. Im ersten Schritt (in Abschnitt 2.3 Topological Area-Collector genannt) wird die Eingabe in die gewünschte Polygonfamilien-Kollektion umgewandelt. In diesem Schritt werden auch mögliche ungültige Teile der Eingabe entfernt oder repariert. Anschließend findet in Algorithmus 5.2 die eigentliche Klassifizierung und anschließende Delegierung statt.

Wir gehen im der angegebenen Vorgehensweise davon aus, dass alle Familien gültiges Format haben und insbesondere nicht leer sind. Dieser Prüfcode wurde zur besseren Übersicht entfernt.

Zu Beginn wird der Fall abgedeckt, dass die Eingabe aus mehren zusammengehörigen Outer-Polygonen besteht, die wiederum selbst innere Polygone aufweisen. In unserem Fall reduzieren wir die Berechnungen auf das Flächengröße. Mehr Details werden später in Abschnitt 6.3 eingebracht.

Für Eingaben mit einer Familie wird zunächst geprüft, ob sich unser für Lochstruktur angepasste Algorithmus verwenden lässt. Andernfalls kann mit dem Basisfall gerechnet werden, dass es sich lediglich um eine Fläche ohne Löcher handeln muss. Falls es sich bei der Eingabe zudem um ein konkav polygon handelt, wird die Optimallösung des geometrischen Mittelpunktes verwendet. In allen anderen Fällen werden Polygondetails reduziert und erneut auf Konvexität geprüft. Final generieren wir Mittelpunktkandidaten, aus denen die besten ausgewählt und zurückgegeben werden.

**Algorithmus 5.2** Klassifizierung und Delegation der Polygonfamilien

---

```

procedure GETCENTERBYDELEGATEDMETHOD( $[\mathcal{F}_1, \mathcal{F}_2, \dots, \mathcal{F}_f]$ )
    if  $f > 1$  then
         $\hat{\mathcal{F}} \leftarrow \text{GETLARGESTFAMILY}([\mathcal{F}_1, \mathcal{F}_2, \dots, \mathcal{F}_f])$ 
        return GETCENTEROFFAMILY( $\hat{\mathcal{F}}$ )
    end if
    return GETCENTEROFFAMILY( $\mathcal{F}_1$ )
end procedure

procedure GETCENTEROFFAMILY( $\mathcal{F} = [\mathcal{P}_0, \mathcal{P}_1, \dots, \mathcal{P}_k]$ )
    if  $k > 0$  then
        return GETPOINTOFINACCESSIBILITY( $\mathcal{F}$ )                                // Polygon hat  $k$  Löcher
    end if
     $\mathcal{P} = \mathcal{P}_0$                                                         // Polygon besteht nur aus Outer
    if isConvex( $\mathcal{P}$ ) then
        return GETGEOMETRICCENTROID( $\mathcal{P}$ )
    end if
     $\mathcal{P} \leftarrow \text{REDUCEPOLYGONDETAILS}(\mathcal{P})$ 
    if isConvex( $\mathcal{P}$ ) then
        return GETGEOMETRICCENTROID( $\mathcal{P}$ )
    end if
    pointCandidates  $\leftarrow \emptyset$ 
    pointCandidates.push(GETBETWEENNESSCENTROIDS( $\mathcal{P}$ ))
    pointCandidates.push(GETCLOSENESSCENTROIDS( $\mathcal{P}$ ))
    return GETBESTCANDIDATE(pointCandidates,  $\mathcal{P}$ )
end procedure

```

---

Das Vergleichen von Punktkandidaten kann wiederum mehrere Heuristiken wie Polygonabstände (ähnlich dem Point of Inaccessibility) oder die euklidische Distanz zum geometrischen Mittelpunkt enthalten. Wir werden diese Thematik in Abschnitt 6.3 ebenfalls nochmals aufgreifen.

# 6 Reflexion und Ausblick

In diesem letzten Kapitel werden unsere Ergebnisse im Überblick zusammengefasst, um anschließend über Probleme der Ansätze und Rahmenbedingungen zu diskutieren. Im abschließenden Teil werden Ideen für zukünftige aufbauende Arbeiten vorgestellt und auf bestehende Lücken aufmerksam gemacht, die mit intelligenten Lösungen zu schließen sind.

## 6.1 Zusammenfassung

Nachdem einleitend die Problemstellung im Kontext der OpenStreetMap-Umgebung erläutert und ein Bezug zu verwandten Arbeiten hergestellt wurde, skizziert das zweite Kapitel die Pipeline zur Erstellung adäquater Karten-Beschriftungslabel auf Eingabe einer OSM-formatierten Kartendatei.

Die Pipeline stellt das Rahmenwerk dar, in das die, in den folgenden Kapiteln thematisierten, Algorithmen und Überlegungen eingefügt werden.

Abschnitt drei schließt einerseits die theoretische Lücke zum Umwandeln der OSM-Eingabe-Struktur in eine neutrale, universell einsetzbare und definiert andererseits unsere Berechnungsgrundlagen formal.

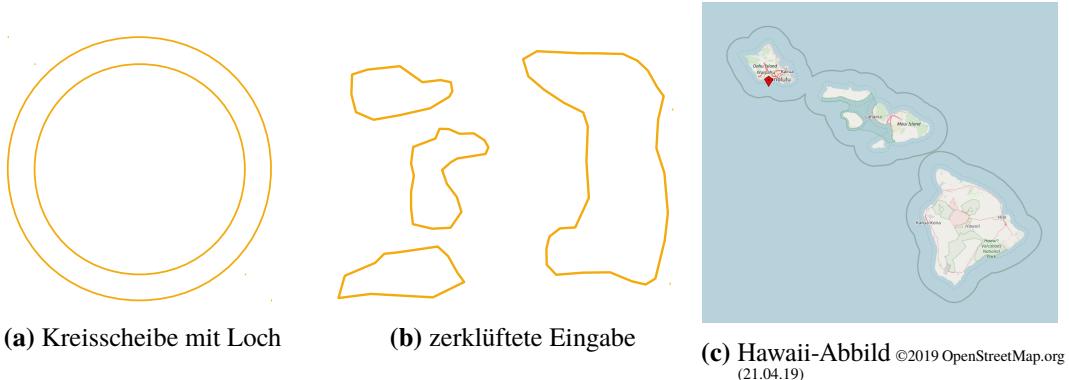
Die Vorstellung und Herleitung der Verfahren zur Mittelpunktbestimmung einer Polygonfläche diskutieren wir im nächsten Abschnitt. Hieran knüpft sich hauptsächlich die selbst erstelle Implementierung der Arbeit. Jeder Algorithmus wird in Bezug auf Laufzeit und Ergebnisqualität evaluiert.

Kapitel 5 versucht die positiven Eigenschaften der Verfahren zu einer Gesamtlösung zu verknüpfen.

Es folgt in den kommenden Abschnitten eine rückblickende Betrachtung der Problemstellung. In kritischer Retrospektive werden Schwachstellen der aktuellen Implementierung diskutiert, bevor diese Arbeit mit Ideen zur weiteren Forschung abgerundet wird.

## 6.2 Problem-Reflexion

Im Rahmen der Mittelpunktbestimmung wurden verschiedene Verfahren und deren Eigenschaften aufgezeigt. Während es im Eingabe-Fall eines konvexen Polygons eine optimale Lösung gibt, stoßen wir auf Probleme im allgemeinen Fall, da es kein Messwerkzeug für die Güte einer Lösung gibt. Daher stellt es sich als schwer heraus, die Ergebnisse messbar zu vergleichen. Wird beispielsweise eine kreisrunde Eingabe betrachtet, aus der mittig eine Kreisfläche ausgeschnitten wurde, lässt sich



**Abbildung 6.1:** Problematische Eingaben, deren Mittelpunkte das Objekt nur unzureichend repräsentieren

intuitiv keine optimale Lösung finden (Abbildung 6.1a). Jeder Punkt auf dem Mittelkreis der Fläche scheint äquivalent, und wirkt unbefriedigend, da wir für eine bidirektional-symmetrische Eingabe eine Lösung auf der Spiegelachse (bevorzugt auf deren Schnittpunkt) erwarten.

Ein analoges Problem tritt auf, wenn die Eingabe aus mehreren Familien besteht, wie beispielsweise in Abbildung 6.1b gezeigt. Eine sinnvolle Repräsentation einer fiktiven Inselgruppe durch einen einzelnen Mittelpunkt innerhalb der Fläche ist nicht immer zielführend.

In diesen Extrembeispielen bietet es sich an – entgegen den Erwartungen aus Abschnitt 3.1 – einen zentralen Punkt zu wählen, der außerhalb liegt. Man müsste zum erwartungsgemäßem Beantworten aller Eingaben eine abstrakte Form der Polygonklassifikation implementieren, die bestimmt, ob eine sinnvolle Beantwortung möglich ist. In Grenzfällen bietet es sich daher an, für kleine Zoomstufen einen Labeltext einzubinden, während für große Zoomstufen der Mittelpunkt zwar außerhalb der Fläche aber mittig platziert wird.

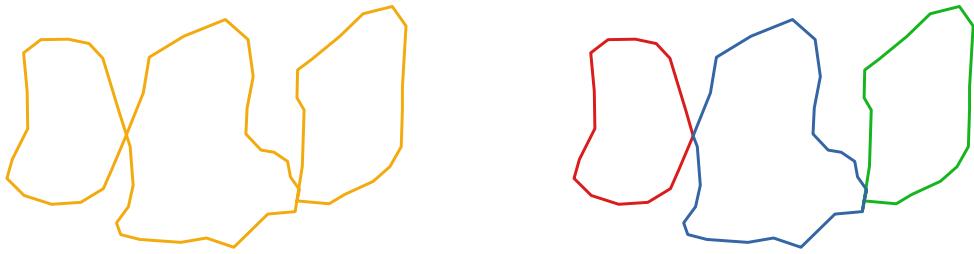
Dass eine sinnvolle Darstellung als Punkt nicht immer sinnvoll ist, zeigt auch ein Beispiel aus OpenStreetMap. Abbildung 6.1c verweist auf die unvorteilhafte Markierung des Mittelpunktes.

### 6.3 Ausblick

Im Verlauf dieser Arbeit sind an einigen Stellen Ideen entstanden, die für zukünftige Verbesserungen aufgegriffen werden können. Außerdem müssen bestimmte Stellen kritisch hinterfragt werden, um Anstöße für verbesserte Verfahren zu geben. Diese Ideen finden in den kommenden Abschnitten Platz.

#### Skelett-Graph mit Zyklen

Wie in Abschnitt 4.5 (Evaluation) beschrieben, werden zum jetzigen Stand nur lochfreie Eingaben betrachtet. Prinzipiell kann diese Einschränkung leicht aufgehoben werden, indem eine Triangulierung auch auf Polygone mit Löchern erlaubt wird. Dadurch ist der Tree-Shrink-Algorithmus



(a) Ungültige Eingabe

(b) Reparierte Eingabe

**Abbildung 6.2:** Beispielhafte Umwandlung einer ungültigen Eingabe in eine gültige

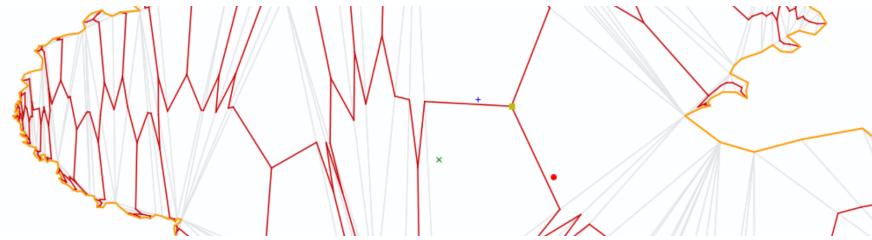
(Algorithmus 4.1) aufgrund der nicht garantierten Zyklendifreiheit nicht mehr anwendbar. An dieser Stelle müsste auf die ursprünglich vorgestellte One-to-All-Dijkstra-Methode zurückgegriffen werden.

### Polygon-Reparatur

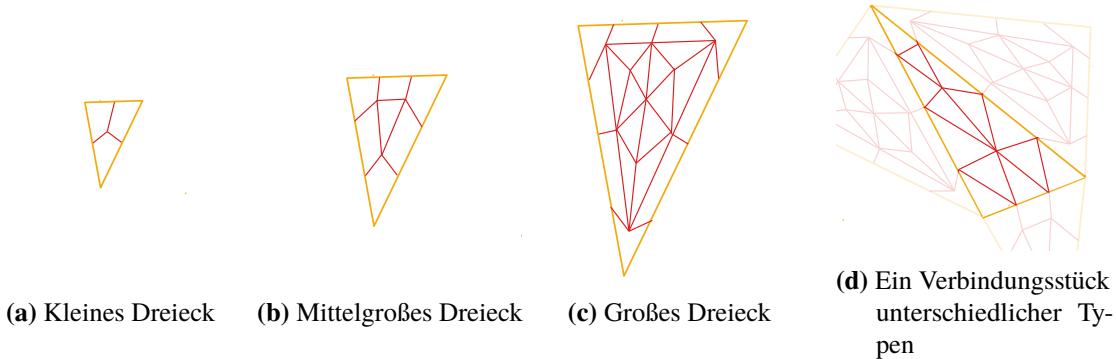
Im Zusammenhang mit der Eingabe wurden auch die Konsistenz-eigenschaften eines Eingabe-Polygons untersucht. Da die Daten von OSM unkontrolliert dem Verwaltungsprogramm zur Verfügung gestellt werden, kann deren Korrektheit nicht garantiert werden. Um diesem Problem entgegen zu wirken, wird in Abschnitt 2.3 rudimentär die Idee der Polygongeingebe-Reparatur eingeführt. Tatsächlich verbirgt sich dahinter ein eigenes Feld der Analyse, um Fehler von Benutzern zu korrigieren. Beispielhaft könnte hier die Idee genannt werden, Polygone mit Schleifen in mehrere Einzel-Polygone umzuwandeln (Abbildung 6.2). Eine umgesetzte Idee ist auch die automatische Schließung nicht-geschlossener Polygonzüge.

### Skalierende Skelett-Punkte

In Abschnitt 4.5 wurde das wirkungsvolle Konzept der Graphanalyse vorgestellt. Nachteil des Verfahrens ist, dass nur Punkte auf dem Skelett Kandidaten für Mittelpunkte werden. Es können insbesondere ausschließlich Skelett-Kanten-Endpunkte zu späteren Mittelpunkten ernannt werden. Aufgrund der baumartigen Konstruktion des Graphen entsteht der Effekt, auf viele Punktkandidaten in Randnähe zu treffen, während Punkte im Inneren des Polygons große Flächen repräsentieren. Betrachten wir das Beispiel in Abbildung 6.3, fällt entgegen der Intuition auf, dass die Anzahl an Punkten linear in der Anzahl an Dreiecken, nicht aber der Flächen steigt. Große Dreiecke werden – obwohl die Wahrscheinlichkeit um ein Vielfaches größer ist, innerhalb der Fläche auf einen guten Mittelpunkt zu stoßen – analog zu kleinen Dreiecken mit nur einem Punkt repräsentiert. Es würde sich lohnen, bei der Skelettierung Punkte gemäß der repräsentierten Fläche einzuführen. So könnte das gesamte Polygon gleichmäßig mit Punkten abgedeckt werden. Abbildung 6.4 zeigt eine exemplarische Skelettierung, die um entsprechende Konnektoren (wie beispielhaft in Abbildung 6.4d zu erkennen) erweitert werden müsste.



**Abbildung 6.3:** Die Punktdichte des Graphen wächst nicht linear mit dessen repräsentierten Fläche



**Abbildung 6.4:** Die Anzahl Punkte wächst mit der Fläche des Dreiecks

### Polygondetailreduktion

Die Ergebnisse aus *Polygondetailreduktion* (Kapitel 5) sprechen für die Idee, für die Berechnung unnötige Rand-Details in einem Vorverarbeitungsschritt zu entfernen. Zum aktuellen Stand kann kein verlässlicher Wert für die Parameter angeben werden, wie viele Punkte ausreichen, um ein *gutes* Ergebnis zu erhalten. Um bessere Vorhersagen treffen und beliebig von der Polygon-Punktemenge abstrahieren zu können, sind weitere Untersuchungen notwendig.

Zudem ist der aktuelle Algorithmus lediglich gut geeignet, um sich ein erstes Bild von dem Verlauf der Reduktion zu machen. Für eine finale Implementierung sollten mehr Überlegungen in den Reduktionsprozess einfließen. Aktuell könnte es zu einem Kantenschnitt kommen, wenn der Polygonzug enge Passagen enthält (Abbildung 6.5a). Zudem ist aktuell nur eine Reduktion auf Familien ohne Löcher möglich, da nicht ausgeschlossen werden kann, dass das Outer-Polygon durch die Reduktion das innere schneidet. Außerdem wäre ein iterationsfreies Verfahren zu bevorzugen, sodass mehrere Polygondetails, die kaum zur Gesamterscheinung beitragen (sogenannte *schwache* Punkte) in einem Durchlauf eliminiert werden, wie Abbildung 6.5b zeigt. Dabei könnten Ansätze des Douglas-Peucker-Algorithmus[DP73] aufgegriffen werden, indem in einem ersten Schritt markante Eckpunkte gefunden werden, die in einem zweiten Schritt die Endpunkte der Segmente für den Douglas-Peucker-Algorithmus bilden. Alternativ könnte auf die Arbeit von Mendel zurückgegriffen werden, der in seinem Paper die Simplifizierung eines Polygonstreckenzuges untersucht, in der Fläche und Umfang des entstandenen Polygons möglichst geringfügig abweichen[Men].

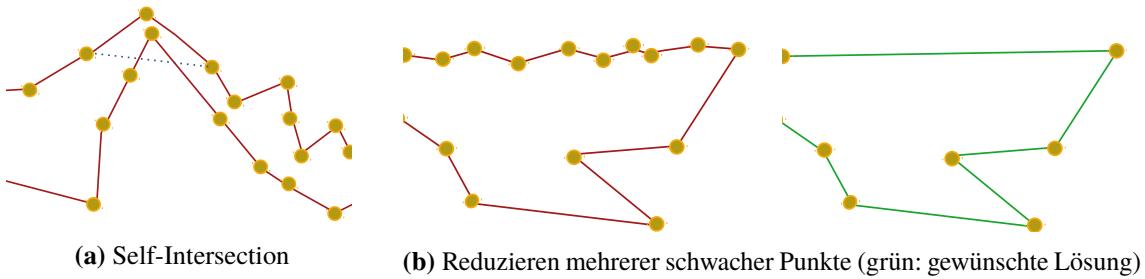


Abbildung 6.5

### Ausgewählte Earpoints für kürzeste Wege

Eine weitere Variante der Verbesserung stellt ein zweistufiges Verfahren dar. In einem ersten Schritt wird eine Polygondetailreduktion durchgeführt, um signifikante Ear-Points zu ermitteln. So können Earpoints, die ein charakteristisches Flächenstück repräsentieren, von Earpoints, die durch eine detailreiche Kontur entstanden sind, unterschieden werden. In einem zweiten Schritt werden die Betweenness- und Closeness-Scores nur mit den Earpoints aus der ersten Phase als Referenz berechnet. Dies sollte vor allem die Betweenness-Score-Ergebnisse stabiler werden lassen, da nur kürzeste Werte zwischen signifikanten Enden benutzt werden.

Alternativ bietet es sich an, eng benachbarte Earpoint-Komponenten zu einer Komponente zusammenzufassen. Ziel ist es, wenige Zusammenhangskomponenten zu erhalten, von denen dann zentrale Punkte bestimmt werden. Wir erinnern an das Schaubild 4.13.

Die Idee lehnt abstrakt an eine Multigitter-Berechnung an. Zu Beginn werden aus kleinen Dreiecken größere benachbarte Blöcke gebildet, davon wird mit denselben Methoden wie in Abschnitt 4.5 ein zentraler Block bestimmt, in dem dann wiederum ein passender Mittelpunkt errechnet wird.

So kann die Idee aus *Graph-Skeleton-Centroid* ohne eine Erhöhung der Graphknotenanzahl umgesetzt werden. Dies würde sich in der Laufzeit bemerkbar machen.

### Unterschätzen der geometrischen Mitte

Mit der Analyse des geometrischen Mittelpunktes in Abschnitt 4.2 wurde gezeigt, dass das Verfahren im Allgemeinen nicht für beliebige Polygone geeignet ist. Trotzdem wurde in Bezug auf die Lösungsqualität der geometrische Mittelpunkt als Bezugspunkt betrachtet. Dies ist intuitiv logisch, da viele Bundesländer, die einen signifikanten Teil der Testgraphen ausmachen, wenig entartet sind. Für einen besseren Einsatz des Schwerpunkt-Verfahrens wäre es gut, ein Maß zu finden, wie *konkav* beziehungsweise entartet ein Polygon sein darf, bis die geometrische Mitte eine schlechte Wahl darstellt.

Stellt man sich beispielsweise eine Eingabe als eine der drei Polygone in Abbildung 6.2b vor, so ist die geometrische Mitte das optimale Verfahren – und das, obwohl keines der Polygone konvex ist. Es wird im späteren Verlauf eine weitere Idee, wie diese Entscheidung gegebenenfalls algorithmisch getroffen werden könnte, präsentiert.

## Maschinelle Unterstützung bei der Mittelpunktbestimmung

Neben der bisherigen algorithmischen Klassifikation könnten Teile des Programms mit Ansätzen des maschinellen Lernens verknüpft werden. Es werden im Folgenden drei Ideen hierfür vorgestellt.

Wie im vorigen Abschnitt mit Nachdruck hervorgeing, sind die Ergebnisse des geometrischen Schwerpunktes oft sehr gut. Trotzdem wird das Verfahren aktuell nur bei konvexen Eingaben eingesetzt, da Ergebnispunkte in der Nähe einer Kante oder außerhalb liegen könnten. Um das Verfahren sinnvoll auf einer breiteren Klasse von Eingaben nutzen zu können, müsste man vorab entscheiden, ob das vorliegende Polygon für ein Anwenden des Verfahrens geeignet ist. Für diese Aufgabe bietet es sich besonders an, maschinell gelernte Algorithmen zu verwenden, da sehr leicht Trainingsdaten generiert werden können: Dazu werden Polygone aus OSM extrahieren, oder selbst dynamisch generiert, bevor auf diese die geometrische Schwerpunktstreuung angewendet wird. In einem letzten Schritt markiert man händisch all jene Polygone, deren Ergebnisse gut geeignet sind. Polygone deren Schwerpunkte außerhalb liegen könnten davor automatisch markiert werden lassen.

Erweitert man diese Idee, könnte der gesamte Entscheidungsbaum durch eine Klassifikations-Engine ersetzt werden. Man würde analog zum Lernen des Schwerpunktes vorgehen, indem alle Verfahren parallel Berechnungen anstellen und anschließend für jedes Polygon entschieden wird, welcher der Ergebnis-Punkte der beste ist.

Einmal gelernt, könnte der künstliche Polygon-Klassifikator auf eine Eingabe genau das Verfahren, welches die besten Ergebnisse verspricht, vorschlagen.

Als letzte Stufe könnte die gesamte Bestimmung des Mittelpunktes als Aufgabe eines neuronalen Netzes vorgegeben werden. Auch hier könnten in einem zweistufigen Verfahren Trainingsdaten erstellt werden, indem zuerst Polygone generiert und anschließend händisch Mittelpunkte (und optimalerweise Kreisintervalle darum herum) bestimmt werden. Zu beachten gilt jedoch, dass die notwendige Testdatenmenge sehr groß sein müsste, um die Aufgabe ausreichend gut zu trainieren.

## Verbesserte Multi-Familien Berechnung

Zum aktuellen Stand wird auf Eingabe mehrerer Familien nur die Flächengröße weiter verwendet. Da die topologische Struktur der Eingaben völlig außer Acht gelassen wird, besteht hinsichtlich dieser viel Verbesserungspotential. Stelle man sich vor, innerhalb des ausgeschnittenen Lochteils der Abbildung 6.1a befände sich eine weitere Kreisscheibe. Aktuell würden wir per POI-Ansatz im äußeren Polygon einen Repräsentanten suchen. Offensichtlich wäre die Wahl des mittigen, kleineren Polygons die bessere.

Mit diesen weiterführenden Ideen schließen wir unsere Arbeit ab. Zur Reproduktion der Berechnungen stehen die im Rahmen dieser Abschlussarbeit entwickelten Algorithmen, sowie die zugehörigen Testdaten in einem öffentlichen Repository<sup>1</sup> zur Verfügung.

---

<sup>1</sup><https://github.com/baurls/POI-Extraktion-mit-erweiterter-Area-Centroid-Bestimmung>

# Literaturverzeichnis

- [AAJ15] W. Aigner, F. Aurenhammer, B. Jüttler. „On triangulation axes of polygons“. In: *Information Processing Letters* 115 (2015) 45–51. 2015 (zitiert auf S. 17, 45, 46, 54).
- [ABB+16] H.-K. Ahn, L. Barba, P. Bose, J.-L. De Carufel, M. Korman, E. Oh. „A linear-time algorithm for the geodesic center of a simple polygon“. In: *Discrete & Computational Geometry* 56.4 (2016), S. 836–859 (zitiert auf S. 17).
- [Aga16] V. Agafonkin. *A new algorithm for finding a visual center of a polygon*. [Online; accessed 10-April-2019]. Aug. 2016. URL: <https://blog.mapbox.com/a-new-algorithm-for-finding-a-visual-center-of-a-polygon-7c77e6492fbc> (zitiert auf S. 41).
- [AT87] T. Asano, G. Toussaint. „Computing the geodesic center of a simple polygon“. In: *Discrete Algorithms and Complexity*. Elsevier, 1987, S. 65–79 (zitiert auf S. 17).
- [Baj86] C. Bajaj. „Proving geometric algorithm non-solvability: An application of factoring polynomials“. In: *Journal of Symbolic Computation* 2.1 (1986). [Online; accessed 6-April-2019], S. 99–102. doi: 10.1016/S0747-7171(86)80015-3. URL: <http://www.sciencedirect.com/science/article/pii/S0747717186800153> (zitiert auf S. 34).
- [BB15] A. Böge, W. Böge. „Schwerpunktslehre“. In: *Technische Mechanik: Statik - Reibung - Dynamik - Festigkeitslehre - Fluidmechanik*. Wiesbaden: Springer Fachmedien Wiesbaden, 2015, S. 77–90. ISBN: 978-3-658-09155-2. doi: 10.1007/978-3-658-09155-2\_2. URL: [https://doi.org/10.1007/978-3-658-09155-2\\_2](https://doi.org/10.1007/978-3-658-09155-2_2) (zitiert auf S. 35).
- [BDD+16] H. Butler, M. Daly, A. Doyle, S. Gillies, S. Hagen, T. Schaub. *The GeoJSON Format*. Internet Engineering Task Force (IETF) ISSN: 2070-1721. Aug. 2016. URL: <https://tools.ietf.org/html/rfc7946> (zitiert auf S. 22).
- [BDY06] K. Been, E. Daiches, C. Yap. „Dynamic map labeling“. In: *IEEE Transactions on visualization and computer graphics* 12.5 (2006), S. 773–780 (zitiert auf S. 13).
- [BLM+14] M. Bakillah, S. Liang, A. Mobasher, J. Jokar Arsanjani, A. Zipf. „Fine-resolution population mapping using OpenStreetMap points-of-interest“. In: *International Journal of Geographical Information Science* 28.9 (2014), S. 1940–1963 (zitiert auf S. 16).
- [Bou88] P. Bourke. *Calculating the area and centroid of a polygon, Satz 4.2*. [Online; accessed 4-April-2019]. Juli 1988. URL: <http://paulbourke.net/geometry/polygonmesh/> (zitiert auf S. 35).
- [Bou98] P. Bourke. *Determining whether or not a polygon (2D) has its vertices ordered clockwise or counter-clockwise*. [Online; accessed 4-April-2019]. März 1998. URL: <http://paulbourke.net/geometry/polygonmesh/> (zitiert auf S. 35).

- [Cha91] B. Chazelle. „Triangulating a simple polygon in linear time“. In: *Discrete & Computational Geometry* 6.3 (Sep. 1991). [Online; accessed 14-April-2019], S. 485–524. ISSN: 1432-0444. doi: [10.1007/BF02574703](https://doi.org/10.1007/BF02574703). URL: <https://doi.org/10.1007/BF02574703> (zitiert auf S. 51).
- [DP73] D. H. Douglas, T. K. Peucker. „Algorithms for the reduction of the number of points required to represent a digitized line or its caricature“. In: *Cartographica: the international journal for geographic information and geovisualization* 10.2 (1973), S. 112–122 (zitiert auf S. 64).
- [Düv09] N. Düvelmeyer. *Algorithmische Geometrie: Triangulierung von Polygonen*. [Online; accessed 14-April-2019]. Nov. 2009. URL: <https://www-m9.ma.tum.de/foswiki/pub/WS2009/AlgorithmischeGeometrie/FolienVAlgGeomV4.pdf> (zitiert auf S. 51).
- [Fra] M. Franceschet. *Closeness Centrality*. [Online; accessed 15-April-2019]. URL: <https://www.sci.unich.it/~francesc/teaching/network/closeness.html> (zitiert auf S. 50).
- [GL07] D. Garcia-Castellanos, U. Lombardo. „Poles of inaccessibility: A calculation algorithm for the remotest places on earth“. In: *Scottish Geographical Journal* 123.3 (2007), S. 227–233. doi: [10.1080/14702540801897809](https://doi.org/10.1080/14702540801897809). URL: <https://doi.org/10.1080/14702540801897809> (zitiert auf S. 17, 40).
- [Hen10] N. Henze. *Stochastik für Einsteiger: eine Einführung in die faszinierende Welt des Zufalls ; mit über 220 Übungsaufgaben und Lösungen*. Norbert Henze. Vieweg + Teubner, 2010. ISBN: 9783834808158 (zitiert auf S. 50).
- [Imh75] E. Imhof. „Positioning Names on Maps“. In: *The American Cartographer* 2.2 (1975), S. 128–144. doi: [10.1559/152304075784313304](https://doi.org/10.1559/152304075784313304). URL: <https://doi.org/10.1559/152304075784313304> (zitiert auf S. 16).
- [Joh18] A. Johannessen. „Algorithmen zur automatisierten Generalisierung durch Zusammenfassung von Linienzügen in OpenStreetMap“. [Online; accessed 03-April-2019] Kapitel 2. Diss. 2018. URL: <https://www.geofabrik.de/media/2018-03-02-johannessen-diplomarbeit-generalisierung.pdf> (zitiert auf S. 28).
- [Kru18a] F. Krumpe. *Label Pipeline Project*. [Online; accessed 20-March-2019]. Jan. 2018. URL: [https://github.com/krumpfp/label\\_pipeline](https://github.com/krumpfp/label_pipeline) (zitiert auf S. 19, 23).
- [Kru18b] F. Krumpe. „Labeling Points of Interest in Dynamic Maps using Disk Labels“. In: *10th International Conference on Geographic Information Science (GIScience 2018)*. 2018, S. 14. URL: <http://drops.dagstuhl.de/opus/volltexte/2018/9336/pdf/LIPIcs-GISCIENCE-2018-8.pdf> (zitiert auf S. 13, 16).
- [LA18] W. Lu, T. Ai. „Center Point of Simple Area Feature Based on Triangulation Skeleton Graph (Short Paper)“. In: *10th International Conference on Geographic Information Science (GIScience 2018)*. Bd. 114. Leibniz International Proceedings in Informatics (LIPIcs). Berechnung eines Beschriftungspunkts. Dagstuhl, Germany, 2018, 41:1–41:6. ISBN: 978-3-95977-083-5. doi: [10.4230/LIPIcs.GISCIENCE.2018.41](https://doi.org/10.4230/LIPIcs.GISCIENCE.2018.41). URL: <http://drops.dagstuhl.de/opus/volltexte/2018/9369> (zitiert auf S. 17, 44, 48–51, 54).
- [LPS+88] W. Lenhart, R. Pollack, J. Sack, R. Seidel, M. Sharir, S. Suri, G. Toussaint, S. Whitesides, C. Yap. „Computing the link center of a simple polygon“. In: *Discrete & Computational Geometry* 3.3 (1988), S. 281–293 (zitiert auf S. 17).

- 
- [LRS10] J. D. Loera, J. Rambau, F. Santos. *Triangulations*. Springer-Verlag Berlin Heidelberg, 2010. doi: [10.1007/978-3-642-12971-1](https://doi.org/10.1007/978-3-642-12971-1) (zitiert auf S. 54).
- [Map18] Mapbox. *polylabel*. [Online; accessed 10-April-2019]. 2018. URL: <https://github.com/mapbox/polylabel> (zitiert auf S. 41).
- [Men] T. Mendel. „Area-Preserving Subdivision Simplification with Topology Constraints: Exactly and in Practice“. In: *2018 Proceedings of the Twentieth Workshop on Algorithm Engineering and Experiments (ALENEX)*, S. 117–128. doi: [10.1137/1.9781611975055.11](https://doi.org/10.1137/1.9781611975055.11). URL: <https://pubs.siam.org/doi/pdf/10.1137/1.9781611975055.11> (zitiert auf S. 64).
- [MJ18] G. McKenzie, K. Janowicz. „OpenPOI: An Open Place of Interest Platform“. In: *10th International Conference on Geographic Information Science (GIScience 2018)*. 2018. URL: <http://drops.dagstuhl.de/opus/volltexte/2018/9375/pdf/LIPIcs-GISCIENCE-2018-47.pdf> (zitiert auf S. 14).
- [MK08] L. N. Mummidi, J. Krumm. „Discovering points of interest from users’ map annotations“. In: *GeoJournal* 72.3-4 (2008), S. 215–227 (zitiert auf S. 16).
- [MS00] M. McAllister, J. Snoeyink. „Medial Axis Generalization of River Networks“. In: *Cartography and Geographic Information Science* 27.2 (2000), S. 129–138. doi: [10.1559/152304000783547966](https://doi.org/10.1559/152304000783547966). URL: <https://doi.org/10.1559/152304000783547966> (zitiert auf S. 47).
- [Mül08] C. Müller. „Graphentheoretische Analyse der Evolution von Wiki-basierten Netzwerken für selbstorganisiertes Wissensmanagement“. In: GITÖ mbH Verlag, 2008, S. 170–174 (zitiert auf S. 48).
- [NRH12] E. Nuhn, W. Reinhardt, B. Haske. „Generation of landmarks from 3D city models and OSM data“. In: *Proceedings of the AGILE’2012 International Conference on Geographic Information Science, Avignon, France*. 2012, S. 24–27 (zitiert auf S. 16).
- [Ope] OpenStreetMap-Wiki. *Points of interest*. [Online; accessed 13-March-2019]. URL: [https://wiki.openstreetmap.org/wiki/Points\\_of\\_interest](https://wiki.openstreetmap.org/wiki/Points_of_interest) (zitiert auf S. 13).
- [Ope19] OpenStreetMap. *OpenStreetMap - Deutschland — FAQ*. [Online; accessed 15-March-2019]. 2019. URL: [https://www.openstreetmap.de/faq.html#was\\_ist\\_osm](https://www.openstreetmap.de/faq.html#was_ist_osm) (zitiert auf S. 14).
- [RHB15] A. Rousell, S. Hahmann, M. Bakillah, A. Mobasher. „Extraction of landmarks from OpenStreetMap for use in navigational instructions“. In: *Proceedings of the AGILE Conference on Geographic Information Science, Lisbon, Portugal*. 2015, S. 9–12 (zitiert auf S. 16).
- [SD97] P. Su, R. L. S. Drysdale. „A comparison of sequential Delaunay triangulation algorithms“. In: *Computational Geometry* 7.5-6 (1997), S. 361–385 (zitiert auf S. 51).
- [VI01] D. Vlasic, P. Indyk. *Polygon Triangulation: Existence*. [Online; accessed 14-April-2019; Page 4]. 2001. URL: <https://people.csail.mit.edu/indyk/6.838-old/handouts/lec4.pdf> (zitiert auf S. 51).
- [Wik16] O. Wiki. *Area — OpenStreetMap Wiki*. [Online; accessed 3-April-2019]. 2016. URL: <https://wiki.openstreetmap.org/w/index.php?title=Area&oldid=1280374> (zitiert auf S. 28).

- [Wik17a] O. Wiki. *Elements — OpenStreetMap Wiki*. [Online; accessed 15-March-2019]. 2017. URL: <https://wiki.openstreetmap.org/w/index.php?title=Elements&oldid=1479648> (zitiert auf S. 15).
- [Wik17b] O. Wiki. *OSM XML — OpenStreetMap Wiki*. [Online; accessed 18-March-2019]. 2017. URL: [https://wiki.openstreetmap.org/w/index.php?title=OSM\\_XML&oldid=1419416](https://wiki.openstreetmap.org/w/index.php?title=OSM_XML&oldid=1419416) (zitiert auf S. 19).
- [Wik18] O. Wiki. *PBF Format — OpenStreetMap Wiki*. [Online; accessed 18-March-2019]. 2018. URL: [https://wiki.openstreetmap.org/w/index.php?title=PBF\\_Format&oldid=1696345](https://wiki.openstreetmap.org/w/index.php?title=PBF_Format&oldid=1696345) (zitiert auf S. 20).
- [Wik19a] O. Wiki. *Key:area — OpenStreetMap Wiki*. [Online; accessed 3-April-2019]. 2019. URL: <https://wiki.openstreetmap.org/w/index.php?title=Key:area&oldid=1808273> (zitiert auf S. 29).
- [Wik19b] O. Wiki. *Relation:multipolygon — OpenStreetMap Wiki*. [Online; accessed 3-April-2019]. 2019. URL: <https://wiki.openstreetmap.org/w/index.php?title=Relation:multipolygon&oldid=1768351> (zitiert auf S. 29).
- [Wik19c] O. Wiki. *Stats — OpenStreetMap Wiki*. [Online; accessed 14-March-2019]. 2019. URL: <https://www.openstreetmap.de> (zitiert auf S. 14).
- [Wik19d] O. Wiki. *Stats — OpenStreetMap Wiki*. [Online; accessed 15-March-2019]. 2019. URL: <https://wiki.openstreetmap.org/w/index.php?title=Stats&oldid=1785660> (zitiert auf S. 14).
- [Wik19e] O. Wiki. *Tags — OpenStreetMap Wiki*. [Online; accessed 15-March-2019]. 2019. URL: <https://wiki.openstreetmap.org/w/index.php?title=Tags&oldid=1819155> (zitiert auf S. 15).
- [WW13] K. Weicker, N. Weicker. *Algorithmen und Datenstrukturen*. Page 265. Springer-Verlag, 2013. ISBN: 978-3-8348-1238-4 (zitiert auf S. 51).
- [ZC15] J.-D. Zhang, C.-Y. Chow. „GeoSoCa: Exploiting geographical, social and categorical correlations for point-of-interest recommendations“. In: *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM. 2015, S. 443–452 (zitiert auf S. 16).

Alle URLs wurden zuletzt am 08.05.2019 geprüft.

### **Erklärung**

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

---

Ort, Datum, Unterschrift