Master Thesis

im Rahmen des
Universitätslehrganges „Geographical Information Science & Systems“
(UNIGIS MSc) am Interfakultären Fachbereich für GeoInformatik (Z_GIS)
der Paris Lodron-Universität Salzburg

zum Thema

Geometrische Qualität und quantitative Eigenschaften eines ausgewählten Algorithmus zur Linienvereinfachung

vorgelegt von

Christiane Enderle
u104537, UNIGIS MSc Jahrgang 2016

Zur Erlangung des Grades
„Master of Science (Geographical Information Science & Systems) – MSc(GIS)”

Marburg, 8.8.2019

Danksagung

Ich danke Frau Prof. Dr. Gudrun Wallentin für die Betreuung dieser Master Thesis und Herrn Christoph Traun für die Korrekturlesung und seine stets freundlichen und beruhigenden Worte.

Herrn Dr. Dirk Tiede bin ich sehr dankbar für das hervorragend gestaltete Modul *Geoprozessierung mit Python* im Rahmen des UNIGIS-Studiums, das mir bei der Entwicklung der Python-Skripte sehr geholfen hat.

Herrn Michał Lupa danke ich sehr für die Übersendung des Python-Skripts, das die Evaluierung nach Chrobak et al. (2016) realisiert.

Herr Prof. Dr. Tinghua Ai hat mir freundlicherweise einen seiner Artikel übersandt, der anderweitig nicht zugänglich war.

Herrn Joachim Schuff sei für drei interessante Literaturhinweise gedankt.

Herrn PD Dr. Stefan Harnischmacher bin ich dankbar für den Austausch über die Geomorphologie der Küsten im Zusammenhang mit der Auswahl der Testdaten.

Herrn Prof. Dr. Duncan Whyatt danke ich herzlich für die Beantwortung meiner Frage zu seiner Thesis und die Übersendung einer gedruckten Version seiner Arbeit.

Ebenfalls sehr herzlich danke ich Frau Dr. Annika Surmeier und ihrer Firma *The Editing Enterprise* für den hervorragenden Service bei der Revision des extended abstract in englischer Sprache.

Ein besonderer Dank gilt meiner Familie, die mein Vorhaben stets unterstützt hat, sowie allen, die mich mit ihrer Anteilnahme begleitet haben.

Eigenständigkeitserklärung

Ich versichere hiermit, dass ich diese Master Thesis eigenständig, ohne fremde Hilfe und nur unter Nutzung der angegebenen Quellen erstellt habe. Diese Arbeit hat in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegen. Alle Ausführungen, die aus anderen Quellen wörtlich oder sinngemäß übernommen wurden, sind entsprechend gekennzeichnet.

DocuSigned by:
59918BF64F134C4...

Marburg, 8.8.2019 Christiane Enderle

Schlagwörter: Linienvereinfachung, Delaunay-Triangulation, Elementardreieck, Generalisierung, Kartographie

Keywords: Line simplification, Delaunay triangulation, elementary triangle, generalization, cartography

Kurzfassung

Die Vereinfachung von Linien ist ein wesentlicher Vorgang bei der Generalisierung von Karten für kleinere Maßstäbe. Im Verhältnis zu der großen Anzahl an Forschungsarbeiten, die sich seit mehr als 40 Jahren damit beschäftigen, diesen Vorgang zu automatisieren, sind nur wenige Algorithmen in etablierten Softwareprodukten zur Kartenerstellung implementiert worden. Die Ergebnisse solcher Tools sind aus kartographischer Sicht nach wie vor unbefriedigend. Das gab den Anlass zu der vorliegenden Arbeit, den im Jahr 2016 von Ai et al. publizierten Algorithmus auf einen Testdatensatz für verschiedenen Zielmaßstäbe anzuwenden und die Ergebnisse mit einer manuellen Vereinfachung zu vergleichen. Der Algorithmus konstruiert vereinfachte Linienabschnitte aus Segmenten von Dreiecken aus einer Delaunay-Triangulation bzw. aus den Mittelachsen so genannter envelope zones, die aus den Segmenten von Dreiecken gebildet werden. Da der Algorithmus nicht zur Verfügung stand, wurde er selbst programmiert, wobei nicht alle Optionen realisiert werden konnten. Die Ergebnisse wurden mit dem Verfahren von Chrobak et al. (2016) evaluiert. Es basiert auf folgender Überlegung: Je geringer die Anzahl an sichtbaren Abweichungen zwischen Originallinie und vereinfachter Linie, desto ähnlicher sind sie sich und desto besser ist die Qualität der vereinfachten Linie. Das Verfahren bildet aus den Abweichungen zwischen Originallinie und vereinfachter Linie Dreiecke, die mit ihrer Basis auf die Mindestdimensionen der drei Elementardreiecke nach Saliszczew verkleinert (standardisiert) werden. Dreiecke, die danach kleiner sind als diese Elementardreiecke, gelten als nicht sichtbar. Als Referenzdaten für den Vergleich diente eine selbst erstellte manuelle Vereinfachung der Testlinie. Die Untersuchung kommt zu dem Ergebnis, dass der Algorithmus von Ai et al. die Qualität einer manuellen Linienvereinfachung nicht erreicht.

Summary

Line simplification is an essential process in map generalisation on smaller scales. While a lot of research work has been done in more than 40 years in the past to automate this process there are only very few algorithms that have been implemented in common mapping software. From a cartographic point of view the results of those tools are not satisfying. For this reason this thesis investigates an algorithm published by Ai et al. in 2016. The algorithm is applied to a test line for several target scales and compares the results to a manually simplified version. The algorithm constructs simplified parts of the line from triangle sides derived from a Delaunay triangulation or from centrelines of so called envelope zones built up by triangle sides. Since the algorithm was not available it had to be reprogrammed. It was not possible to realize all the options of the algorithm. The results were evaluated using assessment methods as detailed by Song and Miao (2016) as well as Chrobak et al. (2016). The latter follow the concept that the fewer the number of visible deviations between original and simplified line the more similar they are, and the better the quality of the simplified line. They derive triangles from the deviations between original and simplified line and standardize them to the minimal dimensions of Saliszczew's elementary triangles. Those triangles that are smaller than the minimal dimensions after standardization are considered as invisible. Reference data for comparison was a self-made manual simplification. The conclusion of this investigation is that the algorithm developed after Ai et al. does not reach the quality of a manual line simplification.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1. Einleitung

Karten müssen mit kleiner werdendem Maßstab zunehmend generalisiert, d.h. auf das Wesentliche reduziert werden, um die Lesbarkeit trotz des geringeren Platzangebots zu erhalten. Generalisierung lässt sich nach Hake (1982, S. 228) in die Vorgänge *Vereinfachen, Vergrößern, Verdrängen, Zusammenfassen, Auswählen, Klassifizieren* bzw. *Typisieren* und *Bewerten* bzw. *Betonen* untergliedern. Bei der rechnergestützten Generalisierung kann man unterscheiden zwischen kartographischer Generalisierung, die Symbolgrößen in den Prozess einbezieht, und reiner Modellgeneralisierung, die ausschließlich die Geometrie von Geoobjekten ohne Berücksichtigung der Symbolisierung verarbeitet. Erste Ansätze zu einer Automatisierung von Generalisierungsvorgängen wurden von Perkal (1966) bzw. Douglas und Peucker (1973) schon vor dem Wandel von der analogen zur digitalen Kartographie veröffentlicht. Seitdem hat sich die automatische Generalisierung zu einem weiten Forschungsfeld entwickelt. Wissenschaftliche Untersuchungen und Lösungsansätze behandeln häufig Teilprobleme, etwa die Automatisierung einzelner Generalisierungsschritte wie z.B. Vereinfachung, Verdrängung, Zusammenfassung, konzentrieren sich auf bestimmte Objekte wie Gebäude, Linienelemente oder das Geländerelief, auf konkrete Eigenschaften wie Datenkomprimierung oder Verarbeitungsgeschwindigkeit, auf den Einsatz im Internet oder stellen nationale bzw. maßstabsabhängige Individuallösungen für die Generalisierung kompletter amtlicher Karten dar (Lafay et al. 2015; Regnauld 2014; Stoter et al. 2014).

Im folgenden werden einige Beispiele genannt, die zeigen, dass weiterhin Forschungsbedarf besteht:

- Stoter et al. (2016) beschreiben, wie weit die amtliche Kartographie in den Ländern Europas in der vollautomatische Ableitung von Karten kleineren Maßstabs aus einem größeren Maßstab fortgeschritten ist. Als ungelöste Fragen bzw. wenig erforschte Themen nennen sie unter anderem inkrementelle Updates für verschiedene Maßstäbe, die Generalisierung eines kompletten Landes in seiner Heterogenität und die Generalisierung von 3D- bzw. 4D-Daten.
- Die technologische Entwicklung hin zu Kartendiensten im Internet stellt neue Anforderungen an die automatische Generalisierung. Web-Services sollen Karten während der Übertragung zum Nutzer „on the fly“ generalisieren. Verarbeitungsgeschwindigkeit, hohe Kompressionsraten, Lesbarkeit auf Displays mobiler Endgeräte, stufenlose Generalisierung für beliebiges Zoomen und progressive Verarbeitung sind Gegenstand von Forschungsbemühungen (Huang et al. 2017; Miao et al. 2017; Enescu et al. 2015; Foerster et al. 2012; Gaffuri 2011).

1.1 Ziele der Arbeit

Die vorliegende Untersuchung geht folgenden Forschungsfragen nach:

1. Erreichen Algorithmen zur Linienvereinfachung die Qualität einer manuelle Generalisierung? Dabei ist mit einer hohen Qualität die optimale Wahrung der charakteristischen Form der ursprünglichen Linie gemeint.
2. In welchem Maß reduzieren sie die Datenmenge?
3. Inwieweit sind sie in der Lage, den Flächeninhalt von Polygonen zu wahren?
4. Welche Zeit benötigen sie zur Verarbeitung der Daten?

Aus Frage 1 ergibt sich folgende operative Teilfrage:

5. Wie kann die Qualität der vereinfachten Linie gemessen werden?

1.2 Relevanz

1.2.1 Linienvereinfachung

Im Generalisierungsprozess kommt der Vereinfachung von Linien eine große Bedeutung zu, und zwar aus folgenden Gründen: Die meisten Objekte einer topographischen Karte mittleren Maßstabs sind Linien (Müller 1991). Viele thematische Karten und einfache Übersichtskarten benötigen zur Orientierung neben ausgewählten Siedlungen nur wenig Basistopographie bestehend aus Küstenlinien, administrativen Grenzen und gegebenenfalls Gewässern und Verkehrswegen – also überwiegend lineare Elemente einschließlich Flächenbegrenzungen. Sie werden als Vektordaten zum Beispiel von Natural Earth, OpenStreetMap und Global Administrative Areas kostenlos zur Verfügung gestellt, von den beiden erstgenannten auch für kommerzielle Zwecke. Die Daten der beiden letztgenannten Anbieter weisen einen hohen Detailgrad auf, der für kleinere Maßstäbe eine Generalisierung erfordert. Natural Earth bietet seine Daten in drei Generalisierungsgraden für die Maßstäbe 1:10 Mio., 1:50 Mio. und 1:110 Mio. an, die jedoch teilweise wegen ihrer spitzwinkeligen Linienführungen kartographisch unbefriedigend sind. Zudem wären weitere Generalisierungsgrade für Zwischenmaßstäbe wünschenswert. Die Vereinfachung von Linien wird in der Literatur als einer der wichtigsten Vorgänge in der Generalisierung bezeichnet (Ai et al. 2016, S. 298; Samsonov und Yakimova 2017, S. 5). Grundsätzlich muss zwischen Linienvereinfachung und Liniengeneralisierung unterschieden werden. Nach Hake (1982) ist die Vereinfachung ein Teilprozess der Generalisierung. Dies wird auch in Fachaufsätzen zum Thema Linienvereinfachung häufig zum Ausdruck gebracht (Shivanth et al. 2014, S. 625; Shi und Cheung 2006, S. 27; Gruppi et al. 2015, S. 517; Saux 2003, S. 34; Touya und Girres 2013, S. 195). Als Ziel des Vorgangs wird meistens die Reduzierung der Punktzahl unter Wahrung der charakteristischen Formen und die damit einhergehende Einsparung von Speicherplatz bzw. die Beschleunigung des Datentransfers im Internet genannt (Douglas und Peucker 1973; McMaster 1987a; Jenks 1989; Veregin 1999; Cheung und Shi 2006, S. 463; Song und Miao 2016, S. 1; Moore et al. 2015). Robinson et al. (1978) bezeichnen Linienvereinfachung als einen Vorgang, der unerwünschte Details entfernt, während Liniengeneralisierung über die Vereinfachung hinaus Prozesse der Verdrängung, Übertreibung und Betonung durch Vergrößerung etc. umfasst (McMaster 1986, S. 103).

1.2.2 Vergleich mit manueller Generalisierung

In der Literatur gibt es Hinweise darauf, dass es immer noch schwierig bis unmöglich ist, mit einer automatischen Generalisierung ohne Nachbearbeitung durch einen Kartographen die Qualität einer manuellen Bearbeitung zu erreichen. Im folgenden seien einige Beispiele genannt, auch wenn sie sich nicht allein auf die Linienvereinfachung beziehen:

- Stoter (2010) untersuchte die Generalisierungstools der kommerziellen Softwareprodukte *ArcGIS*, *CPT* (Change, Push, Typify), *Radius Clarity* und *axpand* in der jeweiligen Version, die im Juni 2007 käuflich war, und stellte fest, dass bei der Evaluierung durch Experten bei allen getesteten Programmen die Lesbarkeit der Ergebnisse meistens als unterdurchschnittlich bewertet wurde, dass viele gravierende Fehler auftraten sowie erhebliche manuelle Nachbearbeitung notwendig war (S. 163–167).
- Stoter et al. (2014, S. 9–11) betonen, dass die vollautomatisch generalisierte Karte 1:50k der Niederlande nicht hundertprozentig die bisherige replizieren sollte, sondern im Vergleich zu dieser gewisse Anpassungen an die neue Technologie unumgänglich waren.

- Gruppi et al. (2015, S. 516) stellen fest: „Although generalization is a task that has been effortlessly done by humans, it is still hard to automate."
- Yan et al. (2015, S. 65) weisen darauf hin, dass Isobathen, die in Karten zur Navigation auf See das Relief des Meeresbodens darstellen, derzeit meistens manuell generalisiert werden, da sich die Ansätze zur Generalisierung des terrestrischen Reliefs nicht auf dasjenige des Meeresbodens übertragen lassen.
- Stoter et al. (2016, S. 647–648) berichten von einem Workshop mit Vertretern kartographischer Dienststellen europäischer Länder: „Some NMAs chose to go for fully automated processes while accepting a lower cartographic quality or a less rich content of the resulting maps, while others prefer to keep some manual edits to assure the best cartographic quality." (NMA: National Mapping Agency).

Die Vielfalt der Ansätze in jüngeren Forschungsarbeiten allein zum Thema Linienvereinfachung zeigt zum einen, wie sehr darum gerungen wird, die Vorgehensweise eines Experten der Kartographie in computergestützte Abläufe zu übersetzen und damit gleichwertige Ergebnisse zu erzielen („To imitate manual generalization, [...]" (Ai et al. 2014, S. 167)). Zum anderen wird deutlich, dass die Bemühungen ihr Ziel noch lange nicht erreicht haben. Im Literaturüberblick (Kapitel 1.3) werden einige Beispiele vorgestellt.

Obwohl seit der Veröffentlichung der Algorithmen von Douglas und Peucker (1973) bzw. Wang und Müller (1998) unzählige Ansätze zur Linienvereinfachung entwickelt worden sind, die durchaus bessere Ergebnisse liefern (Pallero 2013; Li und Openshaw 1992; Raposo 2013) (siehe auch Literaturüberblick), standen in ArcGIS Desktop bis zur Version 10.3 nur diese beiden Verfahren standardmäßig zur Verfügung. In Version 10.5 kam ein weiteres hinzu (siehe Literaturüberblick, Kapitel 1.3). Andere Algorithmen müssen separat eingebunden werden.

Obwohl sich die Entwickler von Algorithmen zur Linienvereinfachung an der manuellen Generalisierung orientieren, vergleichen sie häufig das Generalisierungsergebnis lediglich mit der ursprünglichen Linie bzw. mit den Ergebnissen anderer Algorithmen, nicht aber mit einer manuellen Generalisierung. Von 21 Artikeln über neu- oder weiterentwickelte Algorithmen zur Linienvereinfachung bzw. Evaluierung von Ergebnissen, die für diese Arbeit studiert wurden, enthielten 7 Artikel Vergleiche mit einer manuellen Bearbeitung. Dies mag teilweise damit zusammenhängen, dass manche Algorithmen vorrangig zu einem anderen Zweck wie z.B. der Reduzierung der Stützpunktzahl entwickelt wurden und weniger zur Vereinfachung der Linienform für einen kleineren Maßstab. Raposo (2013, S. 431–432) zieht es vor, die Ergebnisse seines Algorithmus mit der Originallinie im Ausgangsmaßstab zu vergleichen anstatt mit der entsprechenden Linie in einer vorhandenen Karte im Zielmaßstab, mit der Begründung, dass die Entscheidung darüber, ob letztere Linie akzeptabel ist oder nicht, subjektiv sein kann. Stoter et al. (2009, S. 215) gehen jedoch davon aus – wenn auch in einem etwas anderen Zusammenhang –, dass manuelle Generalisierung zufriedenstellende Ergebnisse liefert und dass die Ergebnisse einer automatischen Generalisierung mit den manuell erstellten vergleichbar sein sollten. Van Oosterom (2009, S. 303) bezeichnet die manuelle Generalisierung, unterstützt durch Werkzeuge der Automation, als Qualitätsmaßstab („benchmark").

1.2.3 Verfahren zur Evaluierung

Die Entwickler von Algorithmen zur Linienvereinfachung betreiben für die Bewertung der Ergebnisse einen unterschiedlich hohen Aufwand. Die Evaluierungsmethoden sind sehr verschieden und es werden keineswegs für jeden Algorithmus die gleichen Eigenschaften

betrachtet. Auch hier liegt die Ursache zum Teil in der jeweiligen Zielsetzung, für die das Verfahren zur Linienvereinfachung entwickelt wurde. Dennoch erscheint es sinnvoll, Algorithmen einer einheitlichen Evaluierung zu unterziehen, um sie besser vergleichbar zu machen. Zur Bewertung der Qualität vereinfachter Linien haben unter anderem Chrobak et al. (2016) sowie Song und Miao (2016) Methoden entwickelt, die in dieser Arbeit herangezogen werden sollen. Sie werden zusammen mit den Bewertungsverfahren einiger anderer Forschungsarbeiten im Abschnitt Literaturüberblick kurz vorgestellt und im Kapitel 3 *Evaluierungsmethoden* ausführlich erläutert.

1.2.4 Verarbeitungszeit und Reduzierung der Datenmenge

Diese beiden Aspekte gewinnen immer mehr an Bedeutung, da in zunehmendem Maß Daten zur Nutzung über das Internet angefordert werden. Ladezeiten sollen so gering wie möglich gehalten werden. Zu diesem Zweck können Daten entweder in bestimmten Detailgraden einer Maßstabsfolge auf dem Server vorgehalten oder während der Übertragung zum Nutzer generalisiert werden. Ziel einiger Forschungsbemühungen ist es, Algorithmen zu entwickeln, die je nach der vom Nutzer individuell gewählten Zoomstufe bzw. je nach Displaygröße des Endgerätes nahezu stufenlose Repräsentationen der Geodaten ermöglichen (van Oosterom 2009). Oberstes Kriterium ist dabei eine hohe Verarbeitungsgeschwindigkeit (Foerster et al. 2012).

1.3 Literaturüberblick

1.3.1 Algorithmen zur Linienvereinfachung

Von den zahlreichen Verfahren, die bisher zur Vereinfachung von Linien entwickelt wurden, kann hier nur eine Auswahl vorgestellt werden, die einige ältere, häufig verwendete bzw. in GIS-Software implementierte Algorithmen und deren Weiterentwicklungen umfasst sowie mehrere Verfahren, die in den vergangenen acht bis zehn Jahren publiziert wurden. Für einen umfassenderen Überblick sei auf mehrere Review-Artikel verwiesen, die Tong et al. (2015, S. 781) in ihrem Artikel aufführen.

Der bekannteste Algorithmus wurde von Douglas und Peucker (1973) entwickelt, bei dem es allerdings zu Überschneidungen innerhalb derselben Linie oder mit anderen Linien kommen kann. Saalfeld (1999) hat ein Verfahren zur Nachbearbeitung solcher Topologiefehler entwickelt. Pallero (2013) hingegen hat ihn so überarbeitet, dass derartige Topologiefehler gar nicht erst auftreten. Es gelang ihm, die Zahl der Stützpunkte um etwa 10% stärker zu reduzieren als Douglas und Peucker. Da sein Interesse neben der Reduzierung der Stützpunktanzahl auch der Verarbeitungsgeschwindigkeit galt, hat er nicht nur die robuste Variante, d.h. mit Prüfung und Reparatur von Topologiefehlern, sondern auch eine nicht-robuste Version entwickelt, die zugunsten der Verarbeitungszeit auf diese Prüfung verzichtet. Dennoch ist der Douglas-Peucker-Algorithmus selbst im Vergleich mit der nicht-robusten Variante von Pallero fast immer schneller. Der Douglas-Peucker-Algorithmus steht in ArcGIS Desktop 10.3.1 im Tool *Linie vereinfachen* bzw. *Simplify Line* unter dem Namen *Point Remove* zur Verfügung „mit Erweiterungen“ (ESRI 2016a). Er ist gut geeignet, um allzu dicht gesetzten Stützpunkte, die ein Linienobjekt repräsentieren, auszudünnen mit dem Ziel, die Datenmenge zu reduzieren und gleichzeitig den Verlauf der Originallinie zu bewahren. Bei größeren Schritten der Maßstabsänderung und bei Vereinfachungen für kleine Zielmaßstäbe – Vorgänge, bei denen komplette Objekte im Linienverlauf erkannt und entfernt werden müssen (Visvalingam und Whyatt 1993), führt er jedoch zu unschönen spitzwinkeligen Linienzügen. ArcGIS

Desktop 10.3.1 bietet daher einen weiteren Algorithmus *Bend Simplify* an, der von Wang und Müller (1998) entwickelt wurde (Raposo 2013, S. 428). Wang und Müller (1998) beschreiben Linien als eine fortlaufende Folge von Biegungen mit abwechselnder Richtung. Die Biegungen werden mit den Eigenschaften Größe, Form und in ihrem Kontext mit benachbarten Biegungen attributiert und in Abhängigkeit von diesen Attributwerten eliminiert, zusammengefasst oder vergrößert. Damit fließt kartographisches Wissen in die Linienvereinfachung ein. Nicht Charakteristika von einzelnen Punkten, sondern Charakteristika der Linie werden identifiziert. Das Verfahren gibt bei großen Maßstabsänderungen die Gestalt der Originallinie besser wieder als der Douglas-Peucker-Algorithmus (ESRI 2016a). Jedoch kann es auch bei *Bend Simplify* zu Selbstüberschneidungen der vereinfachten Linie kommen. Das ESRI-Tool erlaubt daher für beide Verfahren optional die Prüfung auf topologische Fehler und deren Auflösung.

In ArcGIS Desktop 10.5 wird erstmals ein zusätzlicher Algorithmus zur Linien- und Polygonvereinfachung angeboten: *Weighted Area*. Er basiert auf dem Verfahren von Zhou und Jones (2005), (ESRI 2018). Die Autoren greifen auf den Algorithmus von Visvalingam und Whyatt (1993) zurück. Visvalingam und Whyatt entfernen den unwichtigsten Punkt einer Linie und betrachten die verbleibenden Punkte als Stützpunkte der vereinfachten Linie, aus der wiederum der unwichtigste Punkt entfernt wird. Dieser Vorgang wird so lange wiederholt, bis der angestrebte Vereinfachungsgrad erreicht ist. Die Festlegung des Kriteriums oder der Kriterien, die den unwichtigsten Punkt bestimmen, sei offen für die Weiterentwicklung durch andere und sei nicht mehr Bestandteil des Algorithmus, sondern der Implementierung, so Visvalingam (2016). Visvalingam und Whyatt (1993) selbst verwendeten in einer Variante als Metrik die Größe derjenigen Fläche, die durch Verbindung eines Stützpunktes mit seinen beiden Nachbarpunkten gebildet wird. Je kleiner die eingeschlossene Fläche, desto weniger Veränderung bewirkt die Eliminierung des Punktes für den Linienverlauf („areal displacement“, Visvalingam 2016, S. 251). Diese Dreiecksflächen werden für alle Stützpunkte der Linie außer Anfangs- und Endpunkt berechnet. Nach Eliminierung des Punktes mit der kleinsten Dreiecksfläche werden die Dreiecke für die verbliebenen Punkte neu berechnet und wiederum der Stützpunkt mit der kleinsten Fläche entfernt. Um Daten für beliebige Grade der Vereinfachung vorzubereiten z.B. in Datenbanken, ohne jedes Mal aufs neue die Dreiecksflächen berechnen zu müssen, können die Stützpunkte in einer Vorverarbeitung einmalig mit der Reihenfolge ihrer Eliminierung attributiert werden (Visvalingam 2016). In einer anderen Variante verwenden Visvalingam und Whyatt die Metrik der „effective area“ (Visvalingam und Whyatt 1993, S. 47; Visvalingam 2016, S. 251). Dabei werden die Stützpunkte prinzipiell mit der Dreiecksfläche attributiert, die zu ihrer Eliminierung führen. Da sich jedoch durch das Entfernen eines Punktes und die Neuberechnung der Flächen für die Nachbarpunkte Flächengrößen ergeben können, die kleiner sind als diejenige, die im vorigen Iterationsschritt zur Entfernung eines Punktes geführt hat, erhalten solchermaßen betroffene Punkte als Attributwert nicht die neu berechnete, kleinere Flächengröße, sondern diejenige, die beim vorigen Iterationsschritt zur Eliminierung geführt hat – daher der Ausdruck „effective area“. Würde dies nicht berücksichtigt und die Linienvereinfachung strikt in der Größenreihenfolge der berechneten Flächengrößen durchgeführt, käme es an diesen Punkten zu Formverzerrungen. Durch die Verwendung der Metrik *effective area* können bei größeren Maßstabssprüngen komplette Objekte wie z.B. eine Landzunge oder ein Fjord eliminiert werden. Damit geht dieses Verfahren über die reine Linienvereinfachung hinaus und ermöglicht die Generalisierung der Linienform („caricatural generalization“, Visvalingam und Whelan 2016, S. 253). Weitere Varianten mit der Bezeichnung „weighted area“ gewichten auf unterschiedliche Weise die effective

area in Abhängigkeit von der Dreiecksform, die sich zum Beispiel durch den Winkel der Linie im betrachteten Stützpunkt beschreiben lässt (Visvalingam und Whelan 2016). Damit können bestimmte Formen bei der Linienvereinfachung bevorzugt erhalten oder unterdrückt werden. Der Visvalingam-Whyatt-Algorithmus und die verwendeten Testdaten sind frei verfügbar, um weitere Forschungsarbeiten durch andere zu ermöglichen (Bloch 2018; Visvalingam und Whelan 2016). Er wurde bei The New York Times verwendet und darüber hinaus in kommerzieller und Open-Source-Software implementiert (Visvalingam und Whelan 2016). Zhou und Jones (2005) haben diesen Algorithmus mit der Metrik der weighted area und einem nicht veröffentlichten Parameterset implementiert. In dieser Version setzt ihn der Ordnance Survey in Großbritannien ein (Revell et al. 2011). Visvalingam und Whelan (2016) weisen jedoch daraufhin, dass die Gewichtung der effective area gerade in der Version von Zhou und Jones (2005) auch zu unerwünschten Verzerrungen führen kann.

Einen anderen Ansatz haben Li und Openshaw (1990, 1992, 1993) gewählt, die die zu vereinfachenden Linien mit einem Quadratraster überziehen, dessen Zellgröße dem kleinsten vom menschlichen Auge noch wahrnehmbaren Objekt entspricht. Damit wird die Stärke der Vereinfachung vom Maßstab bestimmt. Alle Stützpunkte einer Linie, die in dieselbe Zelle fallen, werden zu einem einzigen Punkt zusammengefasst. Die verbleibenden Punkte werden zur generalisierten Linie verbunden. Selbstüberschneidungen können hierbei nicht auftreten. Raposo (2013) verwendet statt eines Quadratrasters ein Sechseckmosaik und erzielt damit größere Nähe zur Originallinie und weichere, natürlichere Linienverläufe. Er berücksichtigt außerdem Fälle, in denen ein- und dieselbe Linie eine Zelle mehrmals durchquert, indem er mehrere Punktcluster pro Zelle zulässt. Li und Openshaw fassen dagegen alle Punkte einer Zelle zu einem einzigen zusammen, wodurch wesentliche Linienabschnitte verlorengehen können. Raposo lässt die Zellgröße des Rasters auf dem Zielmaßstab und der Signaturbreite der generalisierten Linie basieren und ermöglicht damit kartographische Generalisierung. Nach der Linienvereinfachung wendet Raposo einen weiteren Algorithmus an, der Selbstüberschneidungen auflöst und auch separat zur Nachbearbeitung der Ergebnisse anderer Linienvereinfachungsalgorithmen verwendet werden kann. Raposos Verfahren wird von Chrobak et al. (2016, S. 248) als gebräuchlich bezeichnet. Sie stellen in diesem Artikel ein neues Evaluierungsverfahren vor, mit dem sie neben dem Douglas-Peucker-Algorithmus, dem Visvalingam-Whyatt-Algorithmus und einem von Chrobak (2000, 2010) selbst entwickelten Algorithmus auch Raposos Verfahren getestet haben, wobei Raposo am besten abschneidet. Dieses Evaluierungsverfahren wird weiter unten in diesem Abschnitt und im Kapitel 3 *Evaluierungsmethoden* detaillierter beschrieben.

Park und Yu (2011) arbeiten mit der Segmentierung von Linien in Teilstücke mit ähnlichen Formcharakteristika, um diese anschließend mit dem am besten geeigneten Algorithmus zu vereinfachen. Gegenüber den meisten früher publizierten Verfahren, die Liniensegmentierung einsetzen, beschreiben und analysieren Park und Yu die Linienformen mit quantitativen Eigenschaften (S. 1267). Zu diesem Zweck werden zunächst Linien in einem Testdatensatz mit drei Algorithmen vereinfacht. Park und Yu verwendeten das Verfahren von Douglas und Peucker, Sleeve-Fitting (Zhao und Saalfeld 1997) und Turning Function (Rangayyan et al. 2008). In jedem der drei Ergebnisse werden die Segmente mit mindestens drei aufeinander folgenden Punkten mit minimalen Lagefehlern selektiert, dem jeweiligen Algorithmus als „sections of exclusively high performance“ (SEHP, S. 1268) zugeordnet und zur quantitativen Charakterisierung der Form Länge und Winkel der Segmente ermittelt. Länge und Winkel der Segmente werden in Scatterplots gegeneinander aufgetragen, für jeden Algorithmus ein Scatterplot.

Es zeichneten sich in jedem Scatterplot drei Cluster ab, in deren Zentrum jeweils ein zentraler Vektor bestimmt wurde, der bei der anschließenden Segmentierung als Trainingsdatum diente.

Die zu segmentierenden und zu vereinfachenden Linien entstammten zwar denselben Daten wie der Testdatensatz, unterschieden sich jedoch räumlich von diesem (S. 1270, rechts oben). Für die Segmentierung wird mit dem Anfangssegment einer Linie, bestehend aus drei Stützpunkten, begonnen, dessen Länge und Winkel ermittelt, in die Scatterplots eingetragen und demjenigen Algorithmus zugewiesen, dessen Trainingsdatum das Segment am nächsten zu liegen kommt. Dann wird das Segment um einen Punkt erweitert und der gleichen Prüfung unterzogen. Fällt es aufgrund seiner Lage im Scatterplot demselben Algorithmus zu, wird es um einen weiteren Punkt verlängert. Der Prozess wird iterativ so lange fortgesetzt, bis das Segment dem Testdatum eines anderen Algorithmus näher kommt. Das Segment wird daraufhin mit dem vorherigen Punkt abgeschlossen, der gleichzeitig zum Anfangspunkt eines neuen Segments wird. Auf diese Weise werden sämtliche Linien segmentiert und die Segmente anschließend mit dem Algorithmus, dem sie zugeordnet worden sind, vereinfacht. Park und Yu wendeten ihr Verfahren auf einen Datensatz im Ausgangsmaßstab von 1:1000 für den Zielmaßstab 1:5000 an. Die hybride Linienvereinfachung zeigte im Blick auf Lage-, Flächen- und Winkelabweichung durchweg bessere Ergebnisse als die Vereinfachung aller Linien mit einem einzigen Algorithmus. Prinzipiell können für das Verfahren von Park und Yu auch andere Linienvereinfachungsalgorithmen verwendet werden, allerdings mit folgenden Einschränkungen: Die Algorithmen dürfen Anfangs- und Endpunkt der Linien nicht verschieben, da sonst die vereinfachten Segmente nicht mehr zusammengesetzt werden können. Algorithmen, die ausschließlich auf die Vereinfachung geschlossener Polygone ausgerichtet sind, können wegen der Liniensegmentierung nicht angewendet werden.

Samsonov und Yakimova (2017) kritisieren an dem Ansatz von Park und Yu, dass sie ausschließlich den Positionsfehler für die Auswahl eines Algorithmus zur Linienvereinfachung heranziehen. Nach dem oben beschriebenen Verfahren von Park und Yu ist diese Kritik nicht ganz zutreffend. Sie bemängeln außerdem, dass Clusterbildung prinzipiell nicht eindeutig sein könne und in diesem Fall noch unsicherer werde, da sie vom Anwender durchgeführt werden müsse. Als drittes beanstanden sie, dass die Daten, auf die der Algorithmus angewendet wurde, nämlich Gebäude, Straßen und Flüsse, jeweils in sich bereits einen homogenen Formcharakter besitzen, und verweisen auf administrative Grenzen, die, je nach dem ob sie künstlich gezogen wurden oder natürlichen Objekten folgen, innerhalb eines Datensatzes verschiedene Formcharakteristika haben können.

Samsonov und Yakimova (2017) verwenden ihrerseits die geometrischen Metriken *Segmentlänge* und *Winkel* sowohl für die qualitative Charakterisierung der Linienverläufe als auch für die Segmentierung. Die Testdaten sind administrative Grenzen der Rayons in Kami und Archangelsk in Russland sowie die County-Grenzen des US-Bundesstaates Montana. Rechtwinklige Segmente werden mit einem Verfahren vereinfacht, das auf Staufenbiel (1973) und Meulemans et al. (2010) zurückgeht. Für schematische Segmente verwenden sie den Douglas-Peucker-Algorithmus mit zusätzlicher topologischer Prüfung auf Selbstüberschneidungen und deren Beseitigung; nicht-schematische Segmente werden mit dem Li-Openshaw-Algorithmus vereinfacht.

Das Verfahren ist für kleine Maßstäbe entwickelt worden. Die Testdaten lagen in einem Ausgangsmaßstab von 1:1 Mio. vor und wurden für verschiedene Zielmaßstäbe bis 1:10 Mio. verarbeitet. Die Ergebnisse der formabhängigen segmentweisen Linienvereinfachung wurden

mit der Anwendung des Douglas-Peucker- bzw. Li-Openshaw-Algorithmus auf den kompletten Datensatz verglichen. Die modifizierten Hausdorff-Distanzen zur Originallinie und die Verarbeitungsgeschwindigkeit ihres eigenen Verfahrens wurden von den Autoren positiv bewertet, obwohl die sparklines in den Abbildungen meistens etwas anderes aussagen.

Bei der „Oblique-Dividing-Curve Method“ von Qian et al. (2016) handelt es sich um ein neu entwickeltes Verfahren der Liniensegmentierung. Segmente bestehen nur aus einer Linienbiegung und werden mithilfe mathematischer Beschreibungen automatisiert in die Kategorien *U*- oder *V-Form* bzw. große, mittlere und kleine Biegungen gruppiert. Theoretisch kann für die nun folgende Linienvereinfachung der einzelnen Segmentkategorien ein beliebiger Algorithmus angewendet werden. Die Autoren setzen den Douglas-Peucker-Algorithmus ein, dessen Parameter sie für große U- bzw. V-Formen verschieden wählen. Was den Ansatz von anderen Segmentierungsverfahren grundlegend unterscheidet, ist der iterative Verlauf. Wenn nach dem ersten Durchgang der Vereinfachung der ganzen Linie für das erste Segment eine Veränderung festgestellt wird, beginnt die Segmentierung von vorn und ermöglicht auf diese Weise, auf die Auswirkungen der Linienvereinfachung im Kontext des benachbarten Segments zu reagieren. Die Autoren streben weniger eine Generalisierung mit deutlicher Vereinfachung der Linienform an, sondern heben hervor, dass bei gleich starker Reduzierung der Stützpunktzahl die Formen gut erhalten bleiben im Gegensatz zur klassischen Anwendung des Douglas-Peucker-Algorithmus ohne Segmentierung. Das Ergebnis der Vereinfachung wird nur mäßig von dem Parameter beeinflusst, der für den Douglas-Peucker-Algorithmus gewählt wird. Qian et al. (2016) testeten ihre Methode mit Höhenlinien und stellten fest, dass bei dichter Scharung Überschneidungen mit Nachbarlinien nicht ganz auszuschließen sind, dass dieses Problem aber wesentlich seltener auftritt als beim Verfahren von Douglas-Peucker.

Lopes et al. (2013) setzen in ihrem publizierten Verfahren auf künstliche Intelligenz. Mit drei verschiedenen Algorithmen werden aus geometrischen Charakteristika der Linien die Spannungsparameter berechnet, die zur Vereinfachung der Linienbiegungen angewendet werden sollen. Diese Algorithmen werden mit Spannungsparametern trainiert, die während der manuellen Generalisierung durch eine erfahrene menschliche Fachkraft aufgezeichnet wurden. Ein vierter Algorithmus fungiert als Agent und wählt aus den Ergebnissen der drei Algorithmen die beste Lösung aus. Auf einen Datensatz von Höhenlinien mit einem Ausgangsmaßstab von 1:25k und einem Zielmaßstab von 1:50k angewendet ergab sich eine rund 80-prozentige Übereinstimmung mit der manuellen Generalisierung bei einer Zeitersparnis von 95%. Die Abweichungen bewegten sich im gleichen Umfang, der auch bei der manuellen Generalisierung durch verschiedene Personen auftritt.

Moore et al. (2015) verfolgen mit ihrem Ansatz keine Linienvereinfachung im Sinne einer Generalisierung, sondern die Reduzierung der Datenmenge, indem sie Linienverläufe in Anlehnung an Perkal (1966) durch Kreise unterschiedlicher Größe und deren Verbindung durch Tangenten nachbilden. Sie wendeten das Verfahren auf die Küstenlinie von Rarotonga im Maßstab 1:62.800 an und stellen anhand der Ergebnisse fest, dass ihr Ansatz zwar lineare Verläufe natürlicher geographischer Objekte wie Küstenlinien besser wiedergibt, dass jedoch der Douglas-Peucker-Algorithmus die Datenmenge stärker reduziert und die Flächeninhalte besser bewahrt.

Tong et al. (2015) haben einen Methode zur Vereinfachung von Polygonen unter Wahrung des Flächeninhalts entwickelt, die sie als Ergänzung zu klassischen Algorithmen wie diejenigen von Douglas und Peucker (1973) bzw. Visvalingam und Whyatt (1993) betrachten.

In einem vorbereitenden Schritt bestimmen sie mit einem der beiden genannten Verfahren diejenigen Punkte, die erhalten bleiben müssen, um die charakteristische Form des Polygons zu bewahren. Diese Punkte teilen die Polygonlinie in Abschnitte, die jeweils mit einer Geraden angepasst werden. In einem weiteren Schritt wird die Bedingung für den Erhalt der Flächengröße einbezogen, die mithilfe der Structured Total Least Square Methode in einer Matrizenberechnung realisiert wird. Zuletzt werden die Geraden von Schnittpunkt zu Schnittpunkt zur generalisierten Polygongrenze zusammengesetzt. Tong et al. testeten ihren Ansatz mit Landnutzungspolygonen der chinesischen Provinz Guangxi mit einem Ausgangsmaßstab von 1:2.000 und den Zielmaßstäben 1:10k, 1:20k und 1:50k. Sie kommen in allen drei Fällen zu dem beeindruckenden Ergebnis nahezu vollständiger Flächentreue. Der visuelle Eindruck vermittelt außerdem, dass die charakteristische Form des Polygons gewahrt bleibt.

Ai et al. (2014) haben sich in ihrer Forschungsarbeit darum bemüht, einen Algorithmus zu entwickeln, der bei der Linienvereinfachung Riasküsten als erkennbare geomorphologische Form bewahrt, weisen jedoch darauf hin, dass die Methode generell auf Küstenlinien anwendbar ist. Der Ansatz ist eine Weiterentwicklung von früher publizierten Verfahren (Ai 2007; Ai et al. 2000). Die seeseitigen Flächen der tiefen und nicht selten stark verästelten Mündungstrichter einer Riasküste werden per Delaunay-Triangulation in Dreiecke aufgeteilt, aus denen Skelette der Ästuare abgeleitet werden. Der längste Skelettast von der Mündung aus gesehen bestimmt den Haupttrichter. Von den Verzweigungspunkten ausgehend werden wiederum die längsten Äste gesucht, die als Trichter zweiter Ordnung kategorisiert werden. Dieser Vorgang wird fortgesetzt, bis die Hierarchie für das komplette Ästuarsystem aufgebaut ist. Die Länge und die Fläche der Ästuare inklusive ihrer Verzweigungen sowie die fraktale Dimension sind ausschlaggebend für die Auswahl der Ästuare, die in einem angestrebten Zielmaßstab erhalten bleiben sollen. Ai et al. (2014) testeten ihren Algorithmus mit einem Abschnitt der Riasküste der Chesapeake Bay in den USA mit Daten des NOAA National Geophysical Data Center (NOAA: National Oceanic and Atmospheric Administration). Der Ausgangsmaßstab betrug 1:100k, Zielmaßstäbe waren 1:150k, 1:200k, 1:250k und 1:500k. Die Küstenlinie der Kodiak-Island in der Antarktis, die keine Riasküste ist, wurde von einem Ausgangsmaßstab 1:1 Mio. für die Zielmaßstäbe 1:2 Mio., 4 Mio. und 5 Mio. generalisiert. Die Autoren verglichen ihr Verfahren mit dem Algorithmus von Wang und Müller, wie er in ArcGIS 9.3 unter dem Namen *Bend Simplify* zur Verfügung steht, und kommen zu dem Schluss, dass ihre Methode den geomorphologischen Charakter der Riasküste besser wiedergibt. Sie betonen den Vorteil, dass bei ihrem Ansatz im Gegensatz zu Wang und Müller keine Landvorsprünge, sondern ausschließlich wassergefüllte Formen abgeschnürt werden. Dies vergrößert zwar die Lageungenauigkeit bei Ai et al. (2014), hat aber den wesentlichen Vorteil, dass das Ergebnis die Navigation auf See nicht gefährdet. Bei Wang und Müller können außerdem Überschneidungen mit gegenüberliegenden Küstenabschnitten eines Ästuars auftreten, was bei Ai et al. (2014) verfahrensbedingt ausgeschlossen ist.

Ai et al. (2016) haben einen weiteren Algorithmus entwickelt. Er geht auf den Vorschlag von Perkal (1966) zurück. Perkal ließ einen Kreis mit Durchmesser epsilon auf beiden Seiten der zu generalisierenden Linie entlang rollen. Enge Krümmungen, die der Kreis nicht voll ausfahren kann, ohne die Linie zu schneiden, bilden auf beiden Seiten der Linie abgeschnittene Flächen, die als Hüllen („enveloping zones“, Ai et al. 2016, S. 299) betrachtet werden, innerhalb derer das Linienstück generalisiert werden muss. Da Perkals Vorgehensweise in bestimmten Linienkonstellationen zu unbefriedigenden Ergebnissen führt (Ai et al. 2016, S. 299), konstruieren die Autoren die Hüllen mithilfe der „constrained Delaunay triangulation“ (Ai et al. 2016,

S. 300). Die vereinfachten Linienabschnitte ergeben sich aus der rechten oder der linken Begrenzung der Hülle oder aus der Mittelachse. Das Verfahren schließt sowohl die Möglichkeit ein, charakteristische Punkte der Linie zu bewahren, als auch anhand der Bilanzierung der wegfallenden Flächen für jede enveloping zone zu entscheiden, welche der drei Varianten als vereinfachender Linienabschnitt verwendet werden soll. In Kapitel 2 *Algorithmus Ai et al. (2016)* wird der Algorithmus näher beschrieben.

1.3.2 Verfahren zur Evaluierung

Dieser Abschnitt gibt einen Überblick über einige grundsätzliche Überlegungen zur Bewertung von Algorithmen zur Linienvereinfachung und nennt Evaluierungsmethoden, die von den Autoren der im vorigen Abschnitt beschriebenen Algorithmen eingesetzt wurden.

McMaster (1986, 1987a) nennt sechs Metriken, mit denen vereinfachte Linien und ihr Original verglichen werden können:

- Prozentuale Reduzierung der Stützpunktzahl
- Prozentuale Veränderung der Winkeligkeit (angularity)
- Prozentuale Veränderung der Standardabweichung der Stützpunktzahl pro Längeneinheit in der Karte
- Summe der linearen Distanzen zwischen Original- und vereinfachter Linie pro Längeneinheit in der Karte. Distanzen links bzw. rechts der Linie gehen mit entgegengesetzten Vorzeichen in die Summe ein.
- Summe der Flächen, die sich zwischen vereinfachten und originalen Linienabschnitten auftun, pro Längeneinheit in der Karte. Flächen links bzw. rechts der Linie gehen mit entgegengesetzten Vorzeichen in die Summe ein.
- Prozentuale Veränderung der Anzahl von Kurvensegmenten. Ein Kurvensegment besteht aus aufeinander folgenden Stützpunkten, deren verbindende Kanten jeweils eine Winkeländerung in derselben Richtung besitzen.

Park und Yu (2011) verwenden neben einem visuellen Vergleich drei geometrische Metriken nach McMaster (1986): Summe der linearen Distanzen bzw. der flächenhaften Distanzen pro Längeneinheit und die prozentuale Veränderung der Winkeligkeit.

Qian et al. (2016) erfassen die Anzahl der Stützpunkte vor und nach der Linienvereinfachung und ermitteln daraus die Kompressionsrate. Die geometrische Qualität beurteilen sie visuell und achten dabei insbesondere auf topologische Fehler, die sich durch Überschneidung mit benachbarten Höhenlinien ergeben können.

Lopes et al. (2013) vergleichen ihre Ergebnisse anhand der Spannungsparameter, die ihr Algorithmus einerseits und die manuelle Generalisierung andererseits für die Linienvereinfachung angewendet hat. Der Spannungsparameter wird aus dem Höhenwert der Isolinien und sechs geometrischen Eigenschaften gebildet: Fraktale Dimension, Anzahl der Stützpunkte, Länge der Linie, Winkeligkeit, Mittelwert und Standardabweichung der Segmentlängen (S. 5). Diese Größen werden in ihrem Einfluss auf den Spannungsparameter unterschiedlich gewichtet. Darüber hinaus haben sie den Zeitaufwand bei rechnergestützter und manueller Bearbeitung gemessen.

Für Pallero (2013) stehen Verarbeitungszeit und Punktreduzierung sowie die qualitative Eigenschaft der korrekten Topologie im Vordergrund.

Moore et al. (2015) messen bei ihren Ergebnissen den benötigten Speicherplatz anhand der Anzahl der Stützpunkte im Vergleich zum Kreisbogen, der einen Linienabschnitt beschreibt und durch drei Werte (zwei Koordinaten für den Mittelpunkt, ein Wert für den Radius) eindeutig gekennzeichnet ist. Darüber hinaus verwenden sie eine spezielle Messgröße, die die Flächenveränderung beschreibt, um die Ähnlichkeit der Formen zu bewerten (S. 3). Das Produkt der Werte aus beiden Messgrößen dient ihnen als Gesamtgröße für die Beurteilung der Ergebnisse.

Tong et al. (2015) messen Flächenveränderungen in absoluten und prozentualen Werten. Sie ermitteln die orthogonalen Distanzen zwischen den Stützpunkten der Originallinie und dem Verlauf der vereinfachten Linie, klassifizieren die Distanzen und betrachten die Prozentanteile der Punktzahl in den jeweiligen Klassen an der Gesamtzahl der Stützpunkte. Neben diesen Metriken bewerten sie die Ergebnisse visuell.

Ai et al. (2016) berechnen die Kompressionsrate der Stützpunktzahl, den Mittelwert der prozentualen Flächenänderung und messen die Verarbeitungszeit. Außerdem bewerten sie ihre Ergebnisse visuell und achten dabei besonders auf den topologischen Fehler der unzulässigen Überschneidung.

Shi und Cheung (2006) ziehen zwei Messgrößen für die Bewertung der vereinfachten Linie heran:

- Als Metrik für eine entstandene Verschiebung die maximale orthogonale Distanz zwischen einem Stützpunkt der Originallinie und der vereinfachten Linie
- Als Kenngröße für die Formverzerrung ein „Unähnlichkeitsmaß“ („Shape dissimilarity measure“, S. 35). Die Winkelabweichungen zwischen den Segmenten der Originallinie, die bei der Linienvereinfachung zu einem einzigen Segment zusammengefasst werden, und dem vereinfachten Segment werden mit dem Anteil der jeweiligen Segmentlänge an der Summe der Einzelsegmentlängen gewichtet. Die Summe der gewichteten Winkelabweichungen ist der Wert des Unähnlichkeitsmaßes (siehe Cheung und Shi 2006, S. 471).
- Sie stellen für beide Messgrößen jeweils eine Variante vor, die eine mögliche Lageungenauigkeit der Stützpunkte in den Ausgangsdaten berücksichtigt. Diese führt insbesondere bei der Messung der Formverzerrung zu anderen Ergebnissen als die Variante ohne Berücksichtigung der Lageungenauigkeit.

Das Verfahren von Shi und Cheung ist darauf angewiesen, dass bei der Linienvereinfachung lediglich Punkte eliminiert, jedoch keine verschoben oder neu generiert werden, und dass die vereinfachten Linien aus geraden Teilstücken, nicht aber aus Kreisbögen bestehen (Cheung und Shi 2006). Diese Tatsache schränkt die Verwendbarkeit der Methode ein; z.B. ist sie nicht anwendbar auf den Algorithmus von Moore et al. 2015, der Kreisbögen und neue Punkte generiert, bzw. den von Tong et al. 2015, der neue Punkte berechnet. Werden beim Algorithmus von Ai et al. (2016) Mittelachsen in den envelopes als vereinfachte Linienabschnitte generiert, ist eine Evaluierung mit dieser Methode ebenfalls nicht möglich. Auch für geschlossene Polygone ist sie nicht geeignet, weil für die Messung der Formverzerrung vorausgesetzt wird, dass zwischen Anfangs- und Endpunkt eine Linie mit einer Länge größer als Null vorhanden ist. In diesem Fall wäre ein vorbereitender Schritt notwendig, der die Polygone jeweils in mindestens zwei Teilstücke zerlegt.

Shi und Cheung vergleichen neun Algorithmen. Überraschend ist das Ergebnis, dass der Douglas-Peucker-Algorithmus im Blick auf Verschiebung und Formverzerrung am besten abschneidet (Shi und Cheung 2006, S. 41), obwohl seine Ergebnisse allgemein für unschöne Spitzen bekannt sind (Samsonov und Yakimova 2017, S. 15). Hier spielt möglicherweise der Vereinfachungsgrad in Verbindung mit der Punktdichte auf der Originallinie eine Rolle, auch wenn die Autoren der Meinung sind, dass die Komplexität der Linie bei Douglas-Peucker grundsätzlich einen größeren Einfluss auf das Ergebnis hat als die Punktdichte (S. 33). Shi und Cheung reduzieren die 1.500 Punkte der ursprünglichen Linie (Ausgangsmaßstab 1:5.000) maximal auf 500. Dies ist zwar nur noch ein Drittel der Ausgangspunkte, jedoch ist dem Artikel nicht zu entnehmen, wie dicht die Punkte auf der Originallinie verteilt sind.

Nach Meinung von Song und Miao (2016) ist nicht die Kompressionsrate (Reduzierung der Stützpunktzahl) ein Maß für die Abweichung zwischen ursprünglicher und generalisierter Linie, sondern der Grad der Vereinfachung (S. 3). Denn die Kompressionsrate hängt stark davon ab, wie dicht die Stützpunkte in der ursprünglichen Linie gesetzt waren. Daher muss für einen Vergleich von Algorithmen vielmehr sichergestellt sein, dass ihre Ergebnisse den gleichen Vereinfachungsgrad aufweisen. Dies gilt für den Vergleich sowohl der Geometrie als auch der Performanz.

Die Evaluierungsmethode von Song und Miao konzentriert sich auf die Bildschirmdarstellung von Vektordaten. Hochauflösende Displays zeigen bei ein- und denselben Daten größere Differenzen zwischen abweichenden Linien als niedrig auflösende, bei denen Linien möglicherweise bereits zusammenlaufen, während sie auf den hochauflösenden noch deutlich getrennt wahrgenommen werden. Basierend auf dem Auflösungsvermögen des menschlichen Auges, dem Abstand zum Display und der Pixelgröße kann eine Mindestgröße in Pixeln errechnet werden, die das menschliche Auge am Display erkennen kann. Das Verfahren wird im Kapitel 3 *Evaluierungsmethoden* näher beschrieben.

Chrobak et al. (2016) haben eine eigenes Evaluierungsverfahren entwickelt, das die Genauigkeit und Lesbarkeit der generalisierten Linie gegenüber der Originallinie objektiv bewertet und den Informationsverlust durch verschiedene Metriken erfasst. Es ist grundsätzlich universell anwendbar, weil es die Ähnlichkeit der vereinfachten Linie mit dem Original und damit ihre Qualität anhand der Sichtbarkeit der Abweichungen zwischen Originallinie und vereinfachter Linie im Zielmaßstab bewertet. Dabei berücksichtigt es die Strichstärke im Zielmaßstab, die einen wesentlichen Einfluss auf die Sichtbarkeit besitzt. Einen besonderen Nutzen ihres Verfahrens sehen sie darin, dass die Parameter für die Linienvereinfachung, die bei der Vielfalt der Algorithmen sehr unterschiedlich sein können, durch die wiederholte Anwendung ihrer Methode so optimiert werden können, dass das Ergebnis keine sichtbaren Abweichungen mehr aufweist. Die Festlegung der Parameter wird damit unabhängig vom Vorwissen einer bearbeitenden Person. Das Verfahren wird im Kapitel 3 *Evaluierungsmethoden* näher beschrieben.

2. Algorithmus Ai et al. (2016)

2.1 Beschreibung

Bei der „constrained Delaunay triangulation“ (Ai et al. 2016, S. 300), die die Autoren verwenden, werden jeweils drei Stützpunkte der Linie zu einem Dreieck verbunden mit der Bedingung, dass die Fläche dieses Dreiecks keine weiteren Stützpunkte der Linie enthalten darf. Die Einschränkung besteht darin, dass jedes aus zwei benachbarten Stützpunkten bestehende Teilstück der Linie (Segment) Teil eines Dreiecks sein muss (Jones et al. 1995). Lange Segmente können bei der Triangulation Fehler produzieren. Deshalb werden sie in einem vorbereitenden automatisierten Schritt ermittelt und mit zusätzlichen Stützpunkten in kürzere Abschnitte unterteilt. Diese Stützpunkte werden nach der Generalisierung wieder entfernt. Die Originallinie teilt das von der Triangulation abgedeckte Gebiet in mehrere Flächen (siehe Abb. 1). Jede Fläche und der zugehörige begrenzende Abschnitt der Originallinie bilden ein Bendsystem (bend = Kurvenabschnitt). Nach der Delaunay-Triangulation werden die entstandenen Dreiecke abhängig von ihren Nachbarschaftseigenschaften in vier Typen eingeteilt (Abb. 2). Typ 1: Das Dreieck liegt am Rand des Triangulationsbereichs, an der Mündung eines Bendsystems. Typ 2: Das Dreieck hat einen Nachbarn auf der Innenseite des Bendsystems und zwei Seiten, die an ein benachbartes Bendsystem jenseits der Originallinie grenzen. Typ 3: Dreiecke, die zwei Nachbarn innerhalb desselben Bendsystems haben, füllen Haupt- und Seitenäste aus. Typ 4: Dreiecke, die mit allen drei Seiten an andere Dreiecke innerhalb desselben Bendsystems grenzen, liegen an Abzweigungen von Seitenarmen.

Vom Rand des Triangulationsbereiches ausgehend wird der Kurvenabschnitt der Originallinie, der ein Bendsystem begrenzt, von den Ecken des ersten Typ-4-Dreiecks in zwei Unterabschnitte (bends) zerteilt, die zwei Teiläste umschließen. In jedem Teilast können wiederum Typ-4-Dreiecke auftreten, die die Linienabschnitte in weitere bends unterteilen. Übergeordnete und nachgeordnete bends werden in eine Hierarchie gebracht, siehe Abb. 3.

Für jeden bend auf jeder Hierarchiestufe werden folgende messbare Charakteristika definiert: das Flächenmaß *Kurvengröße (bend size)*, der Tiefpunkt *bottom point* sowie die Längenmaße *Kurventiefe (bend depth)* und *Kurvenbreite (bend width)*. Als Kurvengröße wird der Flächeninhalt bezeichnet, der von dem bend und der ihn begrenzenden Kante des Typ-4-Dreiecks umschlossen wird. Der Tiefpunkt gehört im Scheitel des bends zum Dreieck des Typs 2 und liegt zwischen den beiden Dreieckskanten, die an das benachbarte Bendsystem grenzen. Die Kurventiefe ist die Länge derjenigen Linie, die die Mittelpunkte der querenden Dreieckskanten verbindet bis zum jeweiligen Tiefpunkt. Diese Linie wird im folgenden als „Skelettpfad“ bezeichnet (siehe Abb. 4). Die Kurvenbreite ist die Länge derjenigen Kante des Dreiecks von Typ 1 bzw. Typ 4, die das Bendsystem bzw. einen nachgeordneten bend in seinem so genannten Mündungsbereich abschließt. Diese Kante wird im folgenden „bend mouth“ genannt (Ai et al. 2016, S. 302).

Die Autoren speichern die bends aller Hierarchiestufen mit den genannten Eigenschaften in einem Binärbaum (binary tree). Dies ist möglich, weil jedes Dreieck vom Typ 4 genau zwei bends erzeugt.

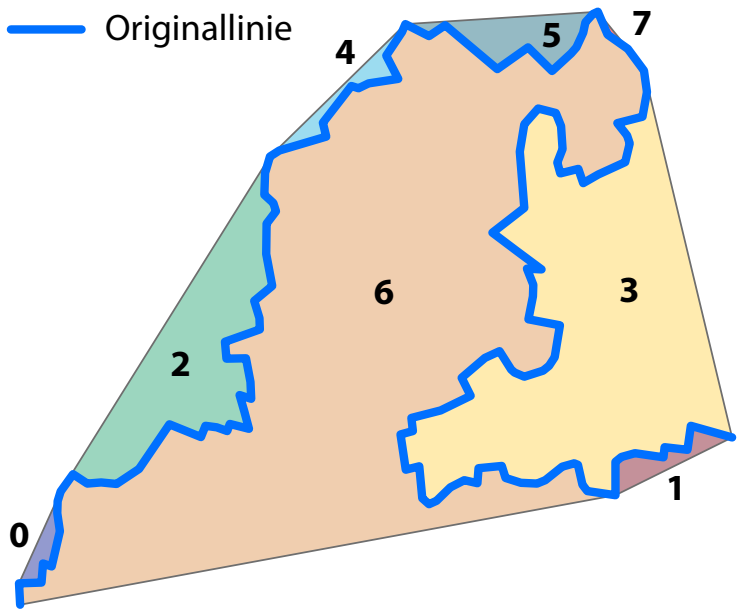


Abb. 1: Bendsysteme.
Sie ergeben sich aus der Domain der Triangulation und der Originallinie.

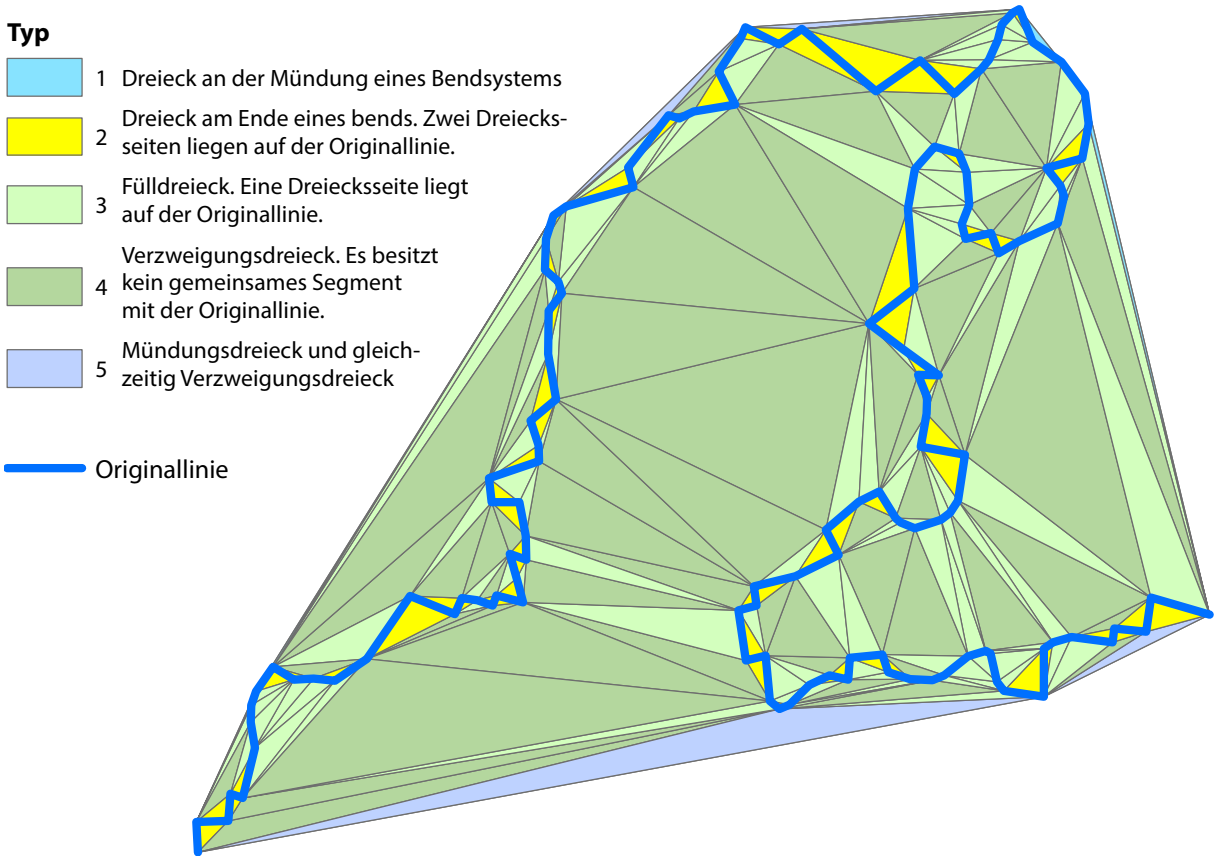


Abb. 2: Dreiecke der Delaunay-Triangulation nach ihren Typen.
Ai et al. (2016) weisen die Typen 1 – 4 aus. Typ 5 wurde aus programmiertechnischen Gründen hinzugefügt.

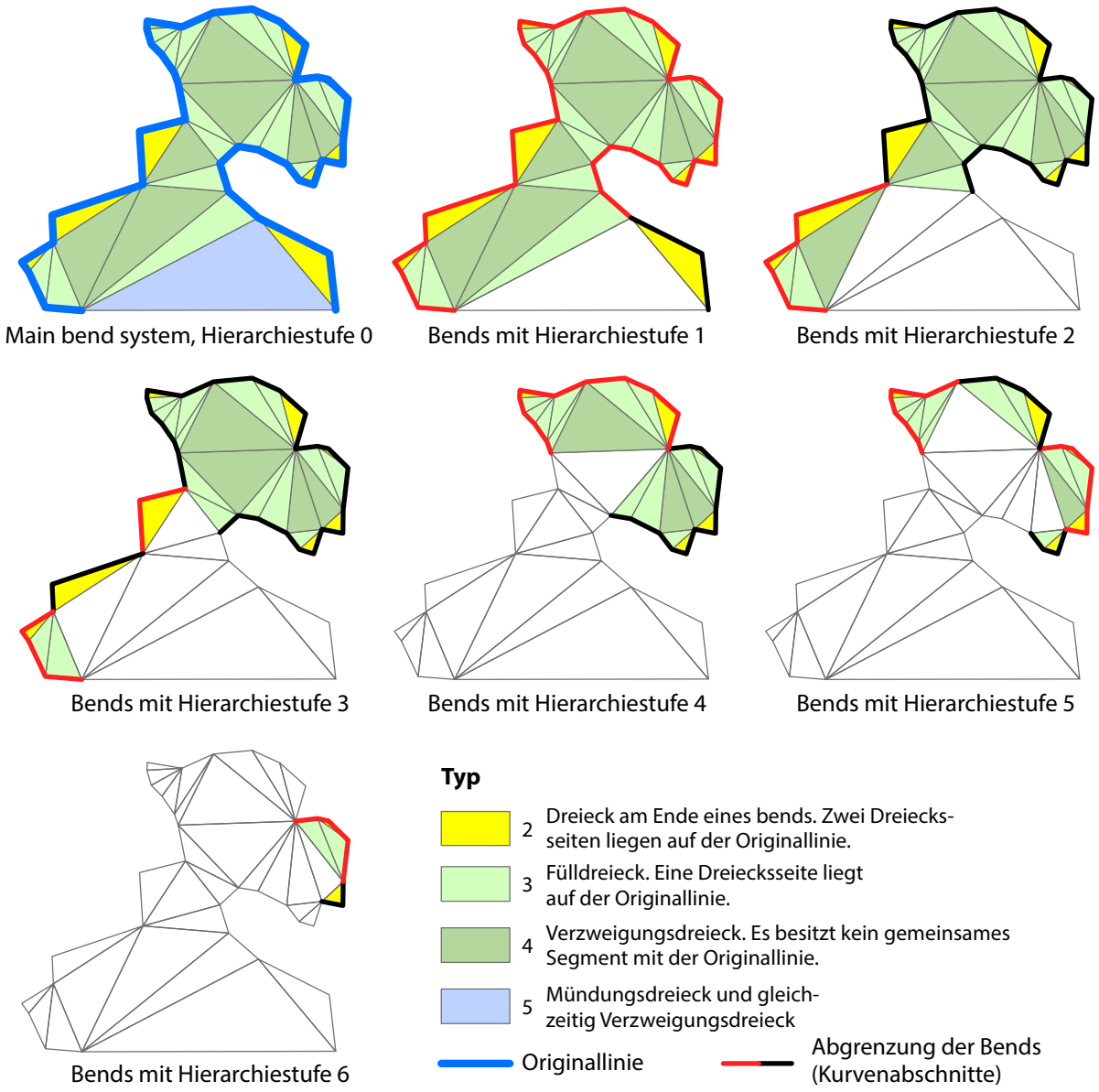


Abb. 3: Hierarchie der bends.
Bends sind Linien- bzw. Kurvenabschnitte.

Für die Bildung der enveloping zones (siehe Ende des Kapitels 1.3.1) werden alle Kurvenabschnitte zunächst in erhaltenswerte einerseits und zu verwerfende andererseits unterschieden. Als Auswahlkriterium kann ein Schwellenwert epsilon für die Kurvengröße, die Kurventiefe oder die Kurvenbreite herangezogen werden. Ai et al. (2016) verwenden die Kurventiefe (bend depth). Anschließend werden die enveloping zones aus den Flächen gebildet, die beiderseits des Linienverlaufs durch die bend mouths der verworfenen Kurvenabschnitte eingeschlossen werden (siehe Abb. 4).

Charakteristische Punkte wie zum Beispiel der Tiefpunkt der Gesamtkurve eines Bendsystems könnten jedoch auf diese Weise wegfallen, wenn die Tiefe der Gesamtkurve zwar über dem Schwellenwert epsilon liegt, die Kurventiefe des lokalen innersten Kurvenabschnitts, der in demselben Tiefpunkt endet, aber unter dem Schwellenwert liegt. Deshalb kann der Schwellenwert epsilon durch einen Parameter gamma modifiziert werden, so dass erhaltens-

werte Punkte bestimmt und verworfene Kurven in erhaltenswerte verwandelt werden. Der neue Schwellenwert *epsilon'* (im Python-Skript „epsilon_big" genannt) wird folgendermaßen gebildet: epsilon_big = gamma × epsilon

wobei gamma als Eingabeparameter vom Benutzer festgelegt wird und größer als 1 sein muss. Kurvenabschnitte, die durch epsilon weggefallen wären, die jedoch denselben Tiefpunkt wie ein übergeordneter bend besitzen, dessen Tiefe größer ist als das durch gamma bestimmte Vielfache von epsilon, werden dann erhalten.

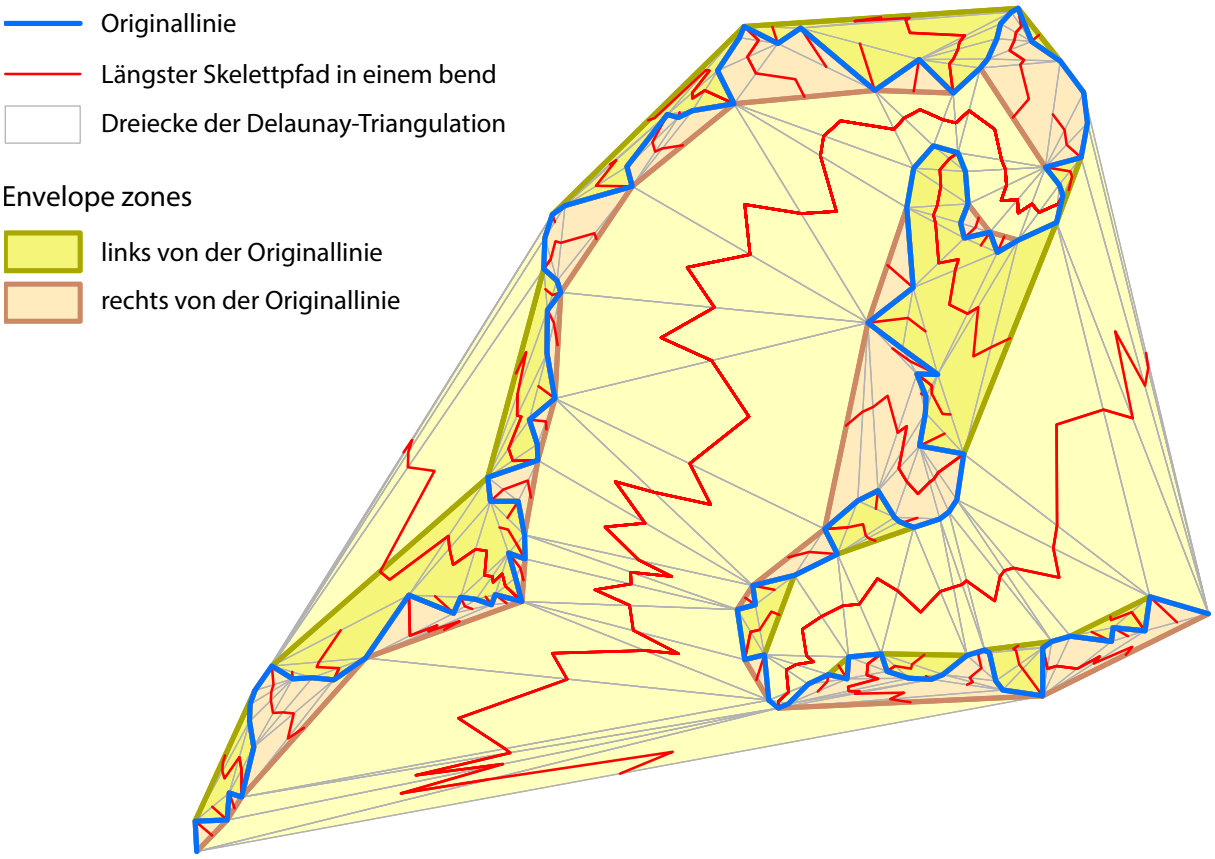


Abb. 4: Envelope zones und Skelettpfade.
Die Länge des längsten Skelettpfades in einem bend entspricht der bend depth und wird als Parameter epsilon verwendet.

Für die Bildung der generalisierten Linie gibt es prinzipiell drei Möglichkeiten:

Das generalisierte Linienteilstück kann als Mittelachse zwischen den beiden begrenzenden Hüllenrändern konstruiert werden; es kann jedoch auch der rechte oder der linke Rand der Hülle als generalisierter Abschnitt verwendet werden. Je nach gewähltem Verfahren fällt die Bilanz von wegfallenden und hinzukommenden Flächen unterschiedlich aus. Um dem Nutzer eine Kontrolle darüber zu geben, kann er einen Parameter m_0 mit einem Wert belegen. m ist folgendermaßen definiert:

m = (A - B) / (A + B),

wobei:

A = Summe der linksseitig abgeschnittenen Flächeninhalte
B = Summe der rechtsseitig abgeschnittenen Flächeninhalte

m kann demnach Werte zwischen -1 und 1 annehmen. Je größer der Unterschied zwischen linksseitig und rechtsseitig abgeschnittenen Flächeninhalten ist, desto näher liegt m an 1 oder -1. Sind die Flächeninhalte auf beiden Seiten nahezu gleich groß, liegt m nahe null. Im Lauf des Vereinfachungsvorgangs wird m für jede Hülle berechnet und mit dem Eingabeparameter m_0 verglichen. Wird m_0 per Benutzereingabe beispielsweise auf den Wert 0,3 gesetzt, wird in allen Hüllen, für die $-0{,}3 \leq m \leq 0{,}3$ gilt, die Mittelachse zwischen den Hüllenrändern als vereinfachte Linie konstruiert. Für $m < -0{,}3$ wird der rechte Hüllenrand als vereinfachte Linie verwendet, für $m > 0{,}3$ der linke Hüllenrand. Die Konstruktion der Mittelachse führt zu einer stärkeren Glättung der Linie und bei geschlossenen Polygonen zu geringeren Abweichungen der Polygonfläche. Die Verwendung der Hüllenränder bedeutet dagegen eine höhere geometrische Genauigkeit. Somit kann der Benutzer mit dem Eingabeparameter m_0 das Ergebnis der Generalisierung nach seinen Bedürfnissen beeinflussen.

Begründung für die Wahl des Algorithmus

Der Algorithmus von Ai et al. (2016) wurde aus folgenden Gründen für diese Arbeit ausgewählt: Er erfüllt das Kriterium, in jüngerer Zeit veröffentlicht worden zu sein. Die Autoren verfolgen das Ziel, die Liniencharakteristik zu wahren, indem sie Punkte identifizieren, die beim Vorgang der Vereinfachung erhalten bleiben müssen. Die präsentierten Ergebnisse sahen vielversprechend aus. Sie haben ihr Verfahren allerdings nur für relativ große Maßstäbe getestet: ausgehend vom Maßstab 1:10k für die Zielmaßstäbe 1:50k und 1:100k. Dagegen waren für die vorliegende Arbeit Zielmaßstäbe bis 1:10 Mio. geplant.

2.2 Programmierung des Algorithmus nach Ai et al. (2016)

Die Autoren haben den Algorithmus in dem von ihnen entwickelten Generalisierungssystem DoMap implementiert (S. 309). Auf Anfrage übersandten sie ihn als Toolbox zur Verwendung in ArcGIS zusammen mit einem Beispieldatensatz. Das Tool lief ohne Fehlermeldungen. Das Ergebnis war jedoch gegenüber dem Original unverändert, auch bei Verwendung verschiedener Parameterwerte. Die Autoren schickten daraufhin eine zweite Version. Das Problem bestand aber weiterhin. Bei beiden Versionen konnte nur ein Parameter festgelegt werden, während dagegen der Artikel drei Parameter beschrieb. Die Umstellung auf drei Parameter sei möglich, werde jedoch mehr Zeit beanspruchen, teilte einer der Autoren mit. Zwei weitere Nachfragen nach einer funktionierenden Version mit drei Parametern blieben innerhalb eines Jahres unbeantwortet.

Da der Algorithmus in dem Artikel sehr gut beschrieben war, wurde er von der Bearbeiterin selbst nachprogrammiert. Mithilfe der in ArcGIS Desktop 10.3.1 vorhandenen Tools, mit selbstgeschriebenen Skript-Tools in Python 2.7.1 und dem Modul *arcpy* wurden im Modelbuilder von ArcGIS zwei Modelle erstellt. Der nächste Abschnitt stellt die Probleme, die daraus folgenden Einschränkungen und das Ergebnis der Programmierung vor.

Probleme bei der Programmierung

Die Programmierung des Algorithmus in der vorliegenden Fassung beanspruchte 386 Arbeitsstunden. Ein großes Problem stellte die **Genauigkeit bei der Verarbeitung der Koordinaten** durch die Tools von ArcGIS 10.3 dar. Häufig ergaben sich Fehler, weil die Koordinaten durch die Geoprozessierung minimal verändert wurden. Sie konnten nur teilweise durch die Angabe von Toleranzwerten behoben werden.

Die **Methoden der arcpy-Geometrie-Objekte** machten den Eindruck, **unzuverlässig** zu arbeiten, z.B. bei einer Abfrage *if point1.touches(point2)*. Bei diesen Methoden gibt es nach dem Kenntnisstand der Bearbeiterin häufig keine Möglichkeit, Toleranzwerte anzugeben. Sie berücksichtigen auch nicht die Umgebungseinstellung *XYTolerance* (vgl. ESRI 2016b). In einigen Fällen mussten die Skripte daher umgearbeitet werden, so dass sie anstelle dieser Methoden die entsprechenden Tools aus den Toolboxen verwendeten, die die XYToleranz beachten. Für eine räumliche Abfrage von Punktobjekten, die auf dem Endpunkt einer Linie liegen, gibt es jedoch offenbar kein entsprechendes Tool in ArcGIS. Um die unzuverlässige Methode *touches* zu umgehen, müssten die Koordinaten der Punktobjekte und der Linienendpunkte ausgelesen und verglichen werden. Dabei könnte ein Toleranzwert eingeführt werden. Dieses recht umständliche Vorgehen wurde aus Zeitgründen nicht mehr umgesetzt.

Das ArcGIS-Skript-Tool ***Add Geometry Attributes***, das in einem selbst erstellten Python-Skript aufgerufen wird, versagt speziell an dieser Stelle hin und wieder seinen Dienst – ohne erkennbaren Grund und ohne dass am Skript oder an den Eingabedaten etwas verändert wurde. Eine Ursache konnte die Bearbeiterin nicht finden.

Die Autoren Ai et al. verstehen den **Begriff der „Constrained Delaunay-Triangulation“** anders als im ArcGIS-Tool *Create TIN* (triangulated irregular network) beschrieben. Ai et al. (2016, S. 300) beziehen sich auf Jones et al. 1995, demzufolge dabei alle Segmente der Linie zu Seiten von Dreiecken werden müssen. Zuvor verdichten sie allzu lange Segmente mit zusätzlichen Punkten. Im ArcGIS-Tool *Create TIN* wird die Originallinie als Bruchlinie (breakline) behandelt und „Constrained Delaunay-Triangulation“ bedeutet zwar ebenfalls, dass alle Segmente der Linie zu Dreiecksseiten werden müssen. Die Einschränkung besteht jedoch darin, dass die

Bruchlinien so stark honoriert werden, dass die Regeln der Delaunay-Triangulation nicht darauf angewendet werden, was wiederum bedeutet, dass sie nicht mit zusätzlichen Punkten verdichtet werden (ESRI 2016c). Die Option *Constrained Delaunay-Triangulation* wurde im ArcGIS-Tool daher nicht aktiviert, sondern die Verdichtung mit zusätzlichen Punkten wurde zugelassen.

Da der Schwerpunkt dieser Arbeit nicht auf der Programmierung von Algorithmen liegt, die Programmierkenntnisse der Bearbeiterin dem Anfängerniveau zuzuordnen sind und der Zeitaufwand im Rahmen gehalten werden musste, wurden bei der Programmierung folgende **Einschränkungen** gemacht:

- Der Algorithmus ist nicht auf Verarbeitungsgeschwindigkeit optimiert.
- Er ist nicht auf beliebige Linien anwendbar. Es darf beispielsweise keines der Linienenden innerhalb der Domain der Delaunay-Triangulation liegen, sondern sie müssen sich auf der Grenze der Domain befinden. Linien, bei denen die Länge des umschreibenden Rechtecks sehr viel größer ist als die Breite, können ihn zum Absturz bringen oder zu fehlerhaften Ergebnissen führen, weil in diesen Fällen bei der Delaunay-Triangulation sehr langgestreckte Dreiecke entstehen, die wegen der erwähnten geometrischen Ungenauigkeiten nicht mehr korrekt verarbeitet werden.
- Die von den Autoren beschriebene Vorgehensweise, für die Delaunay-Triangulation die Punktdichte auf der Linie bei Bedarf zu erhöhen, wurde nicht umgesetzt, sondern stattdessen das Tool *Create TIN* aus der Toolbox *3D-Analyst* von ArcGIS verwendet und dabei das Einfügen von Punkten zugelassen. Die Autoren ermitteln bei ihrem Verfahren zunächst den Mittelwert aller Segmentlängen der Originallinie. Segmente mit Längen, die größer sind als die Summe aus Mittelwert L_m und zweifacher „mean square deviation“, werden verdichtet (Ai et al. 2016, S. 300). Es bleibt unklar, ob die Autoren tatsächlich die mittlere quadratische Abweichung oder eher die Wurzel daraus meinen, da sich der Mittelwert in Metern nicht zur mittleren quadratischen Abweichung in Quadratmetern addieren lässt. Segmente L_i, deren Länge jedenfalls eine bestimmte Grenze überschreitet, werden mit einer Anzahl von Punkten gleichmäßig verdichtet, die dem nächst niedrigeren Ganzzahlwert des Quotienten L_i / L_m entspricht. Damit wollen die Autoren zu schmale Dreiecke bei der Triangulation und die damit verbundenen Probleme vermeiden. Die Bearbeiterin hat mit einer Beispiellinie einen Test durchgeführt und folgendes festgestellt: Das Verfahren nach Ai et al. – wenn man die Wurzel der mittleren quadratischen Abweichung verwendet – setzt mehr Punkte auf ein längeres Segment als das ArcGIS-Tool *Create TIN* mit der Option zum Einfügen von Punkten. Die erhoffte Vermeidung von schmalen Dreiecken konnte jedoch nicht allgemein bestätigt werden. An manchen Stellen wurden zwar schmale Dreiecke in breitere, anders gelagerte aufgelöst, was hier und da die Ausgangssituation für die Linienvereinfachung begünstigte und damit einen Qualitätsgewinn bedeuten könnte, in anderen Fällen wurden jedoch schmalere Dreiecke als zuvor generiert. Nach Einschätzung der Bearbeiterin hat dies keinen nennenswerten Einfluss auf das Ergebnis. Würde man den Autoren strikt folgen und die Summe aus Mittelwert der Segmentlängen und mittlerer quadratischer Abweichung als Grenzwert verwenden, wäre bei der Beispiellinie keine Segmentverdichtung erforderlich gewesen, weil kein Segment länger als diese Summe war.
- Es wurde für die ineinander verschachtelten Kurvenabschnitte kein Binary Tree aufgebaut. Stattdessen wurden Punktlisten, Eigenschaften und Hierarchiestufe eines jeden Kurvenabschnitts mit verschachtelten Listen verarbeitet. Nach dem Verständnis der Bearbeiterin beeinflusst dies nicht die Qualität oder Geometrie der vereinfachten Linie, sondern die Verarbeitungsgeschwindigkeit.

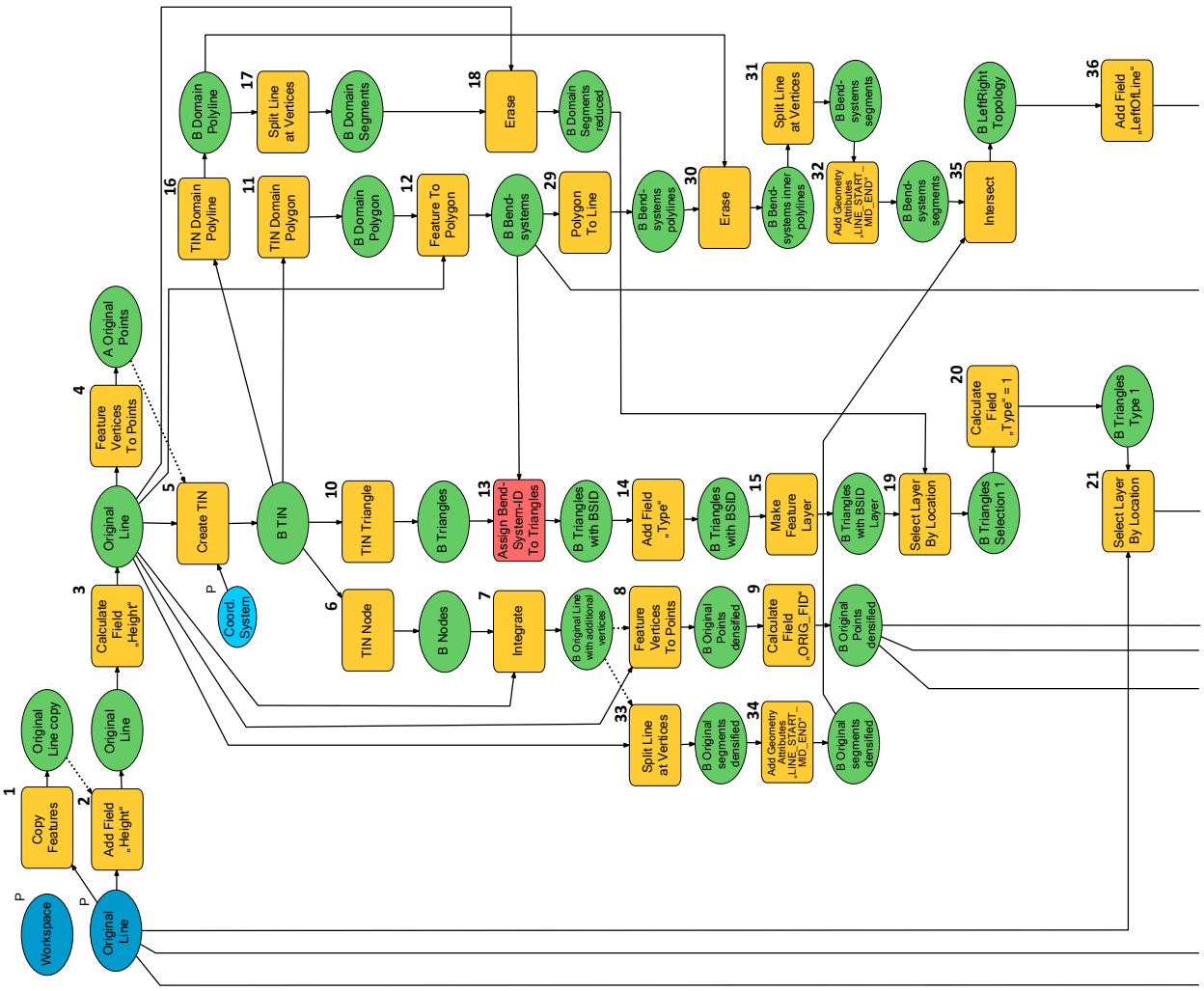
- Die Konstruktion der Mittelachse in den enveloping zones als vereinfachter Linienabschnitt wurde nicht umgesetzt.
- Die Methode *area preserving* mithilfe des Parameters m_0 wurde nicht umgesetzt. Stattdessen wurde eine vereinfachte Variante *area preserving simple* manuell ausgeführt, und zwar folgendermaßen: Für jede envelope zone wurde ermittelt, ob die links der Originallinie liegenden Abweichungen eine kleinere Flächenveränderung zur Folge haben als die rechts der Originallinie liegenden Abweichungen oder umgekehrt. Schließlich wurde die vereinfachte Linie jeweils entlang derjenigen Linienabschnitte digitalisiert, die zur kleineren Flächenveränderung führten.
- Um die verbliebenen Probleme durch geometrischer Ungenauigkeiten zu minimieren, musste die Originallinie einer Vorverarbeitung unterzogen werden. Die angestrebte hohe Genauigkeit bei der manuellen Erfassung hatte zu einer hohen Stützpunktzahl geführt. Dabei war die Richtungsänderung in vielen Punkten so gering – im Winkelmaß ausgedrückt nahe 0°, dass bei der Triangulation sehr flache, lang gezogene Dreiecke entstanden, was bei einigen weiteren Verarbeitungsschritten Ungenauigkeiten mit sich brachte, die zu fehlerhaften Ergebnissen oder zum Abbruch des Algorithmus führten. Deshalb wurde die manuell digitalisierte Linie zunächst mit dem ArcGIS-Tool *Simplify Line*, Methode *Point Remove*, vereinfacht. Die Methode *Point Remove* basiert auf dem Verfahren von Douglas und Peucker mit Erweiterungen (vgl. ESRI 2016a). Der Toleranzparameter, der bestimmt, wie stark das Ergebnis bei einem wegfallenden Punkt vom Original abweichen darf, wurde auf 10 Meter gesetzt. 10 Meter entsprechen im ersten Folgemaßstab der vorliegenden Untersuchung 1:100k einem Wert von 0,1 mm. Dieser Betrag liegt nach Chrobak et al. (2016, S. 240–241) unter der Wahrnehmungsgrenze.

  Auch nach den Überlegungen von Song und Miao (2016) kommt man zu diesem Ergebnis. Die Ermittlung der sichtbaren Mindestdimensionen für Displays, wie die Autoren sie vorstellen, kann ebenso auf gedruckte Medien angewendet werden. Song und Miao (2016, S. 5) beziehen sich für die Berechnung der Sichtbarkeitsgrenze auf Rayleighs Kriterium. Danach beträgt die vom menschlichen Auge wahrnehmbare Distanz eine Bogenminute. Gleichung (4) lautet nach Song und Miao (2016, S. 5):

$$D_{eye} = 2 \times d_{observe} \times \tan\frac{\theta}{2} = 2 \times \tan\frac{1}{120} \times d_{observe} = 0.000291 \times d_{observe}\,(\mathrm{mm}) \qquad (4)$$

  Dabei ist D_{eye} die gesuchte Mindestdimension. $d_{observe}$ entspricht dem Betrachtungsabstand. Der Winkel *Theta* ist die Bogenminute nach Rayleighs Kriterium. Bei einer angenommenen Lesedistanz von $d_{observe}$ = 35 cm ergeben sich damit ca. 0,1 mm.

Das Ergebnis der Programmierung sind die Modelle *Preparing Steps* und *FindParameters* (Abb. 5, Abb. 6). Das Modell *Preparing Steps* generiert die Hierarchie der Linienabschnitte. Es muss nur einmal ausgeführt werden, beansprucht den größten Teil der Verarbeitungszeit und wurde deshalb vom eigentlichen Vereinfachungsprozess getrennt. Es liefert als Ausgabe die Dateien, die für das zweite Modell *FindParameters* benötigt werden. Auch für eine operationelle Implementierung ist diese Trennung sehr sinnvoll. Mit dem Modell *Preparing Steps* bzw. einer ausgereiften Version davon könnten vorhandene Datenbestände vorab verarbeitet werden, um sie bei Bedarf mit beliebigen Parametern *epsilon, gamma* bzw. m_0 zu vereinfachen. Das zweite Modell nimmt als Eingabe neben dem Output des Modells *Preparing Steps* die Parameter *epsilon, gamma* und m_0 und besorgt damit die Linienvereinfachung. Für Details zu den selbst erstellten Python-Skript-Tools sei auf die ausführlich kommentierten Skripte im Anhang A verwiesen.



Für Details zu den im Rahmen der Arbeit erstellten Python-Skript-Tools siehe Kommentare in den Skripts im Anhang A1 - A9.

1 Die Originallinie wird kopiert, da sie während des Verarbeitungsprozesses verändert wird.

2, 3 Der Originallinie wird das Attribut *Height* hinzugefügt und das Feld mit dem Wert 0 gefüllt. Die Delaunay-Triangulation erfordert ein Attribut als dritte Dimension.

4 Die Stützpunkte (vertices) der Originallinie werden als Punktdatensatz gespeichert. Dies ist notwendig, weil bei der Triangulation der Originallinie auf langen, geraden Segmenten gegebenenfalls Punkte hinzugefügt werden, die später aus der vereinfachten Linie wieder entfernt werden müssen (43).

5 Delaunay-Triangulation. Sie erstellt aus der Originallinie ein triangulated irregular network (TIN) und erfordert die Angabe eines Koordinatensystems, sinnvollerweise dasjenige der Originallinie.

6 Es werden die Knoten des TIN gespeichert. Sie setzen sich aus den vertices der Originallinie und gegebenenfalls während der Triangulation hinzugefügten Punkten zusammen. Sie liegen nicht in der Reihenfolge des Linienverlaufs vor.

7 Deshalb werden die Knoten des TINs in die Originallinie integriert. Dabei erhöht sich die Anzahl der vertices in der Originallinie nur um die bei der Triangulation neu hinzugekommenen vertices. Die Originallinie wird also verdichtet („densified").

8 Die vertices der verdichteten Originallinie werden als Punktdatensatz gespeichert. Durch die vorherige Integration der neuen vertices in die Originallinie liegen nun alle vertices in der Reihenfolge des Linienverlaufs vor.

9 Das im vorigen Schritt automatisch erstellte Attribut *ORIG_FID* wird mit der Feature_ID der Punkte gefüllt.

10 Das TIN wird in seine Dreiecke aufgelöst. Diese werden als Polygone gespeichert.

11 Die durch die Triangulation eingeschlossenen Fläche ist die Domain. Sie entspricht der konvexen Hülle der Originallinie und wird in ein Polygon verwandelt.

12 In Kombination mit der Originallinie wird das Domain-Polygon in bend systems zerteilt (siehe Abb. 1).

13 Die Feature-ID der bend systems (BSID) wird den Dreiecken als Attribut zugewiesen. (Python-Skript siehe Anhang A1)

14 Den Dreiecken wird ein Attribut *Type* hinzugefügt, da sie nach Ai et al. (2016) typisiert werden müssen (siehe Abb. 2).

15 Das Tool für die raumbezogene Auswahl (19) verlangt als Eingabe einen Layer, der hier erstellt wird.

16 Die Domain wird in eine Polylinie verwandelt.

17 Die Domain-Polylinie wird an ihren vertices in Segmente aufgelöst.

18 Segmente der Domain-Polylinie, die auf der Originallinie liegen, werden gelöscht.

19, 20 Dreiecke, die mit den aus Schritt 18 verbliebenen Segmenten der Domain-Polyline ein gemeinsames Segment haben („SHARE A LINE SEGMENT WITH"), sind vom Typ 1 (Mündungsdreiecke, siehe Abb. 2).

21, 22, 23 Dreiecke, die kein gemeinsames Segment mit der Originallinie haben (21) („SHARE A LINE SEGMENT WITH", Invert spatial relationship), sind Verzweigungsdreiecke. Wenn sie in Schritt 20 den Typ 1 erhalten haben (22), sind sie gleichzeitig Mündungsdreiecke. Dreiecke, die gleichzeitig Verzweigungs- und Mündungsdreiecke sind, erhalten den Typ 5 (23), siehe Abb. 2.

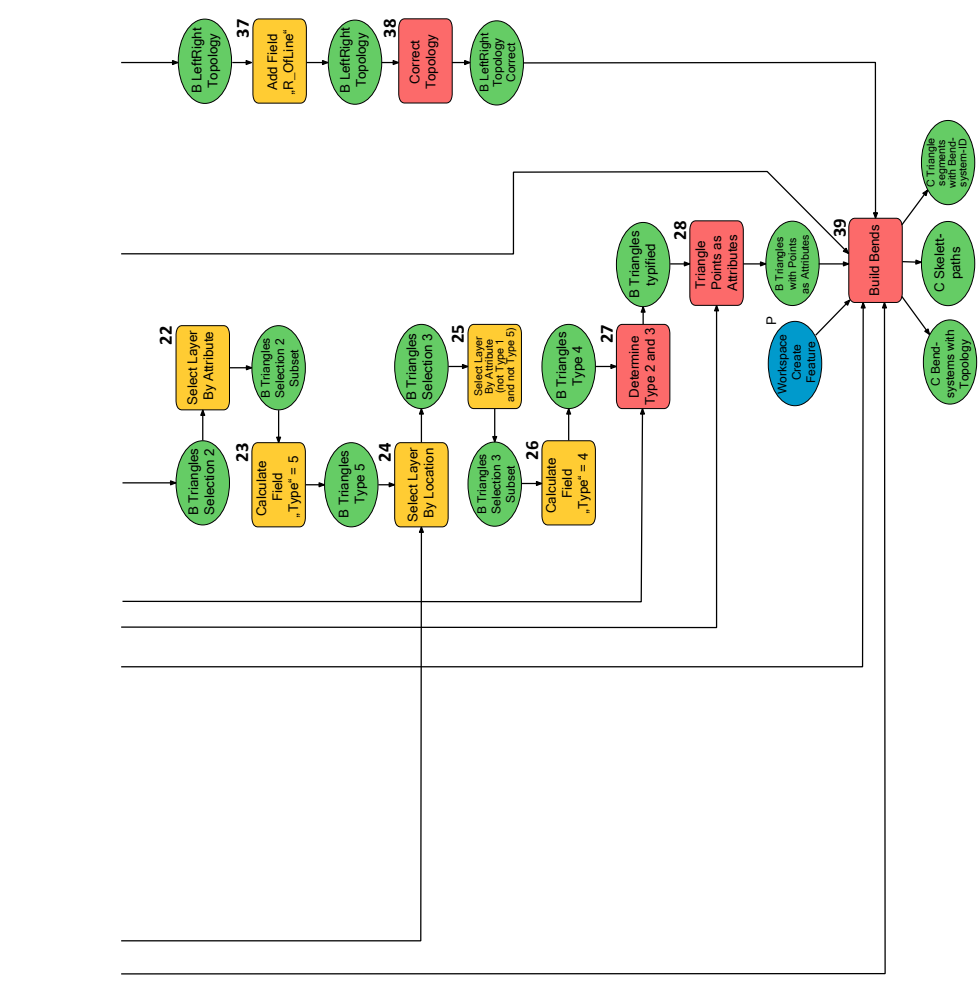


Abb. 5: Modell *Preparing Steps*.

24, 25, 26 Dreiecke, die kein gemeinsames Segment mit der Originallinie haben (24) („SHARE A LINE SEGMENT WITH", Invert spatial relationship), sind Verzweigungsdreiecke, wenn sie nicht bereits den Typ 1 oder 5 erhalten haben (25). Typ 1 können sie theoretisch mit der Auswahl (24) nie haben. Die Prüfung wurde jedoch sicherheitshalber in die SQL-Abfrage einbezogen. Verzweigungsdreiecke erhalten den Typ 4 (26).

27 Dreiecke von Typ 2 und 3 werden bestimmt. Dreiecke des Typs 2 haben zwei gemeinsame Segmente mit der Originallinie und liegen am inneren Ende eines Kurvenabschnitts. Dreiecke des Typs 3 haben ein gemeinsames Segment mit der Originallinie und sind Fülldreiecke. Vergleiche dazu Abb. 2. (Python-Skript siehe Anhang A2)

28 Für jedes Dreieck werden in drei neuen Attributen die Feature-IDs seiner Eckpunkte aus *B Original Points densified* eingetragen. (Python-Skript siehe Anhang A3)

29 Die bend systems werden in Polylinien konvertiert und dabei die Feature-IDs der bend systems (BSID) links und rechts von jeder Polylinie als Attribute *LEFT_FID* bzw. *RIGHT_FID* aufgezeichnet. Die Polylinien zwischen zwei benachbarten bend systems liegen auf der Originallinie, jedoch teilweise in entgegengesetzter Richtung, so dass die Attributwerte für *LEFT_FID* und *RIGHT_FID* in Bezug auf den Linienverlauf nicht immer richtig sind. Eine Korrektur dieser toplogischen Angabe ist notwendig.

30 Die Polylinien, die mit der Domain-Außengrenze zusammenfallen, werden gelöscht. Die Grenzen zwischen bend systems bleiben übrig.

31 Die verbliebenen Polylinien werden an ihren vertices in Segmente aufgelöst.

32 Diesen Segmenten werden die Koordinaten ihrer Start- und Endpunkte als Attribute hinzugefügt.

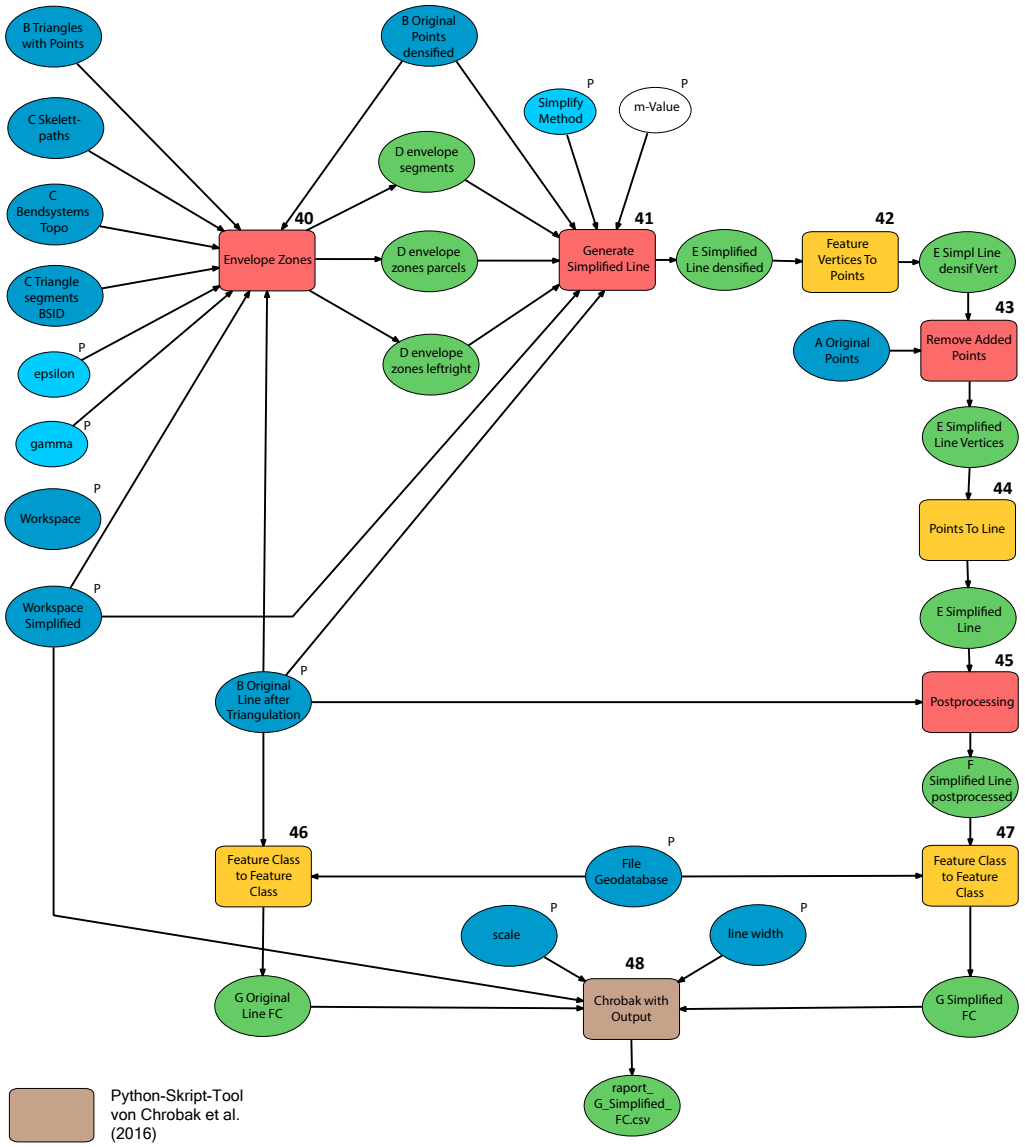
33, 34 Die verdichtete Originallinie wird ebenfalls an ihren vertices in Segmente aufgelöst (33). Die Richtung dieser Segmente entspricht nun dem Verlauf der Originallinie. Auch ihnen werden die Koordinaten ihrer Start- und Endpunkte als Attribute hinzugefügt (34).

35 Die Segmente der verdichteten Originallinie mit den korrekt ausgerichteten Start- und Endpunkten aus (34) werden mit den Segmenten aus den Grenzen der bend systems von (32) verschnitten und dabei die korrekten Start- und Endpunktkoordinaten an die Attributtabelle der bend-system-Segmente angehängt.

36, 37 Es werden Attribute für die korrekte topologische Angabe der BSIDs hinzugefügt (*LeftOfLine*, *R_OfLine*. RightOfLine ist als Attributname zu lang und daher unzulässig).

38 Das Python-Skript-Tool vergleicht die Koordinaten der korrekten Start- und Endpunkte mit den Koordinaten der potenziell vertauschten, weil in umgekehrter Richtung liegenden Start- und Endpunkte und überträgt oder vertauscht entsprechend die BSID aus den Attributen *Left_FID* bzw. *Right_FID* in die Attribute *LeftOfLine* bzw. *R_OfLine*. (Python-Skript siehe Anhang A4)

39 Das Python-Skript-Tool identifiziert bend systems und darin eingeschlossene untergeordnete bends (siehe Abb. 3), bringt sie in eine Hierarchie, bestimmt den jeweiligen bottom point, bend width, bend depth und bend size (Ai et al. (2016), S. 302; siehe Kapitel 2.2). Es gibt die bend systems mit einem Attribut zurück, das ihre Lage links oder rechts von der Originallinie speichert. Die Pfade, deren Länge die bend depth angibt, werden als Skelettpfade ausgegeben (siehe Abb. 4). Die in Segmente aufgelösten Dreiecke werden mit der zugehörigen BSID gespeichert. (Python-Skript siehe Anhang A5)



Weitere Erläuterungen siehe Abb. 5.

Für Details zu den im Rahmen der Arbeit erstellten Python-Skript-Tools siehe Kommentare in den Skripts im Anhang A1 - A9.

40 Mit diesem Python-Skript-Tool beginnt die eigentliche Linienvereinfachung. In Abhängigkeit von den Parametern epsilon (maximale bend depth, die wegfallen soll) und gamma (ein Faktor, mit dem epsilon multipliziert wird zur Auffindung charakteristischer Punkte) werden Envelope-Zonen (Hüllkurven) gebildet. Das Ergebnis sind drei Datensätze: Die Segmente der Envelope-Zonen, die potenziell ein Teilstück der Originallinie ersetzen werden, mit ihrer Lage links bzw. rechts von der Originallinie; ein Zwischenprodukt mit Teilflächen der Envelope-Zonen; die Envelope-Zonen mit ihrer Lage links bzw. rechts von der Originallinie (siehe Abb. 4). (Python-Skript siehe Anhang A6)

41 Dieses Python-Skript-Tool generiert die vereinfachte Linie in Abhängigkeit von der Nutzereingabe der Simplify-Methode, ob nur die links oder nur die rechts von der Originallinie liegenden Envelope-Segmente die entsprechenden Linienabschnitte ersetzen sollen. (siehe Kapitel 2.1, Python-Skript siehe Anhang A7)

42, 43, 44 Das Ergebnis von (41) enthält noch die vertices, die durch die Triangulation hinzugefügt wurden. Deshalb werden die vertices der vereinfachten Linie gespeichert (42), die hinzugefügten vertices im Python-Skript-Tool (43, siehe Anhang A8) durch Abgleich mit den vertices der Originallinie entfernt, soweit sie nicht für den Verlauf der Linie gebraucht werden, und aus den verbleibenden vertices die endgültige vereinfachte Linie zusammengesetzt (44).

45 Dieses Python-Skript-Tool ersetzt in der vereinfachten Linie vertices, die näher als 0,001 m an vertices der Originallinie liegen, mit den vertices der Originallinie. Dies ist erforderlich, weil das Python-Skript von Chrobak et al. (2016) bei vertices, die sowohl in der vereinfachten als auch in der Originallinie vorhanden sind und die gleichen Koordinaten haben sollten, bei geringsten Lageabweichungen mit einer Fehlermeldung abbricht. (Python-Skript siehe Anhang A9)

46, 47 Die Originallinie und die vereinfachte Linie müssen in einer File Geodatabase gespeichert werden, weil das Python-Skript-Tool von Chrobak et al. (2016) keine. Shapefiles als Input akzeptiert. Es wird die Originalline mit den zusätzlichen vertices, die die Triangulation hinzugefügt, hat benötigt, weil die hinzugefügten vertices potenziell relevante Stützpunkte der vereinfachten Linie sind. Das Skript von Chrobak erlaubt jedoch keine Punkte in der vereinfachten Linie, die nicht auch in der Originalline vorhanden sind.

48 Das Python-Skript-Tool von Chrobak et al. (2016) wurde von der Bearbeiterin mit einem Ausgabeparameter versehen, ansonsten jedoch unverändert verwendet. Für eine Beschreibung des Tools siehe Kapitel 3.3.1.

Abb. 6: Modell *FindParameters*.

3. Evaluierungsmethoden

3.1 Evaluierung nach Song und Miao (2016)

Abhängig vom Maßstab gibt ein Bildschirmpixel die räumliche Auflösung von Vektordaten an (S. 4). Wird die räumliche Auflösung mit der vom Auge erkennbaren Mindestgröße in Pixeln multipliziert, erhält man die räumliche Auflösung, die das Auge erkennen kann. Weiter als diese Auflösung dürfen ursprüngliche und generalisierte Linie nicht voneinander abweichen, damit sie noch als ähnlich wahrgenommen werden (S. 5).

Für die Bestimmung des Vereinfachungsgrades werden die größten Abweichungen zwischen jedem Segment der ursprünglichen und der generalisierten Linie berechnet und der Mittelwert gebildet (S. 6).

Da jeder Algorithmus andere Eingabeparameter besitzt, die den Grad der Vereinfachung bestimmen, gilt es jeweils denjenigen Wert für den Eingabeparameter zu finden, mit dem sich ein Mittelwert der Abweichungen knapp unter dem Auflösungsvermögen des Auges ergibt. Dafür stellen Song und Miao einen iterativen Prozess vor:

Es werden so viele äquidistante Werte für den Eingabeparameter gewählt, wie es der Anzahl der Liniensegmente entspricht. Der Algorithmus wird mit jedem dieser Werte angewendet und der Mittelwert der Abweichungen berechnet. Diejenigen beiden Eingabewerte, für die der Mittelwert dem Auflösungsvermögen des Auges am nächsten kommt, werden als die Grenzen eines Intervalls herangezogen, das wiederum in so viele Schritte wie vorhandene Liniensegmente zerlegt wird. Mit diesen neuen Eingabewerten wird der Algorithmus erneut angewendet, die mittlere Abweichung berechnet und diejenigen Eingabewerte ermittelt, für die der Mittelwert dem Auflösungsvermögen des Auges am nächsten kommt. Dieser Prozess wird so lange fortgesetzt, bis Ober- und Untergrenze des Intervalls für die Eingabewerte nahezu gleich sind. Damit ist derjenige Wert für den Eingabeparameter gefunden, mit dem der Algorithmus zu dem angestrebten Vereinfachungsgrad führt. Die Autoren haben auf diese Weise die Vereinfachungsparameter für vier verschiedene Algorithmen ermittelt und damit 504 Eisenbahnlinien in China mit insgesamt ca. 7300 Punkten vom Ausgangsmaßstab 1:1 Mio. für die Zielmaßstäbe 1:2,5 Mio., 1:5 Mio., 1:10 Mio., 1:25 Mio. und 1:50 Mio. vereinfacht. Die Ergebnisse evaluierten sie mit den Metriken der Standardabweichung der maximalen Distanzen, der Kompressionsrate und der Berechnungszeit.

3.2 Evaluierung nach Chrobak et al. (2016)

Die Autoren knüpfen die Lesbarkeit bzw. die Qualität der vereinfachten Linie daran, dass die Abweichungen zur Originallinie nicht mehr sichtbar sind. Dabei greifen sie auf die Arbeit von Saliszczew (1998) zurück, der aufgrund des Auflösungsvermögens des menschlichen Auges Mindestdimensionen für drei so genannte Elementardreiecke von unterschiedlicher Basislänge und Höhe entwickelt hat (siehe Abb. 7).

Chrobak et al. (2016) haben diese Maße in die digitale Kartographie übertragen und leiten zusätzlich die Länge des Dreiecksschenkels ab (siehe Abb. 8).

Während Saliszczew das Dreieck mit einer Strichstärke von 0,1 mm angenommen, diese Linienbreite jedoch in seine Überlegungen nicht einbezogen hat, berücksichtigen Chrobak et al.

beliebige Strichbreiten der vereinfachten Linie im Zielmaßstab, weil sie wesentlichen Einfluss auf die Lesbarkeit haben (Abb. 9). Die Autoren vertreten die Auffassung, dass eine generalisierte Linie der Originallinie ähnlich sieht und damit das Original gut wiedergibt, wenn die Abweichungen im Zielmaßstab geringer sind als die Dimensionen des oben beschriebenen Elementardreiecks. Die Abweichungen werden als Flächen erfasst, die durch das Schneiden der generalisierten Linie mit der Originallinie zwischen diesen beiden eingeschlossen werden.

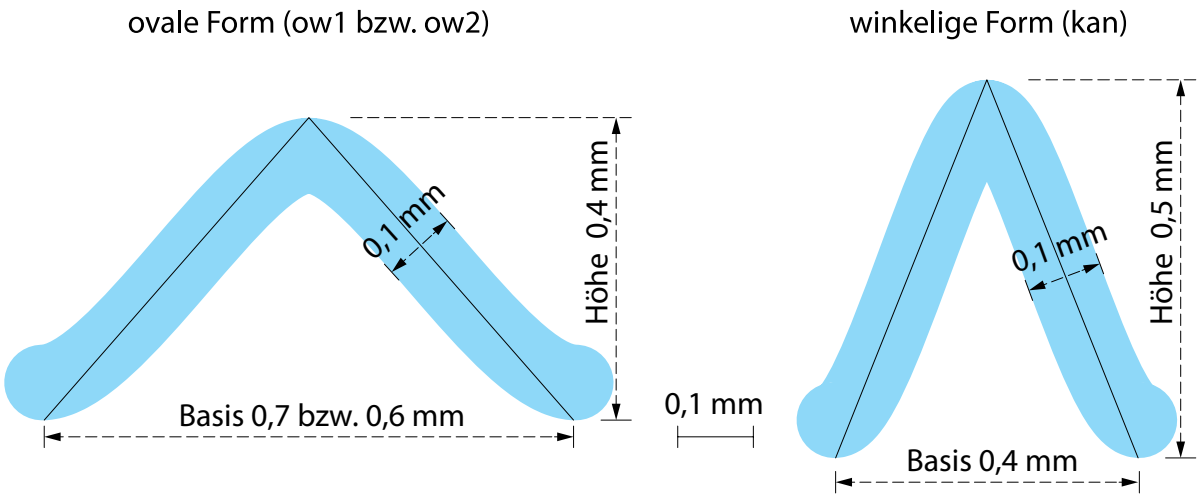


Abb. 7: Elementardreiecke nach Saliszczew (1998).
Die Abbildung entspricht Fig. 1a in Chrobak et al. (2016).

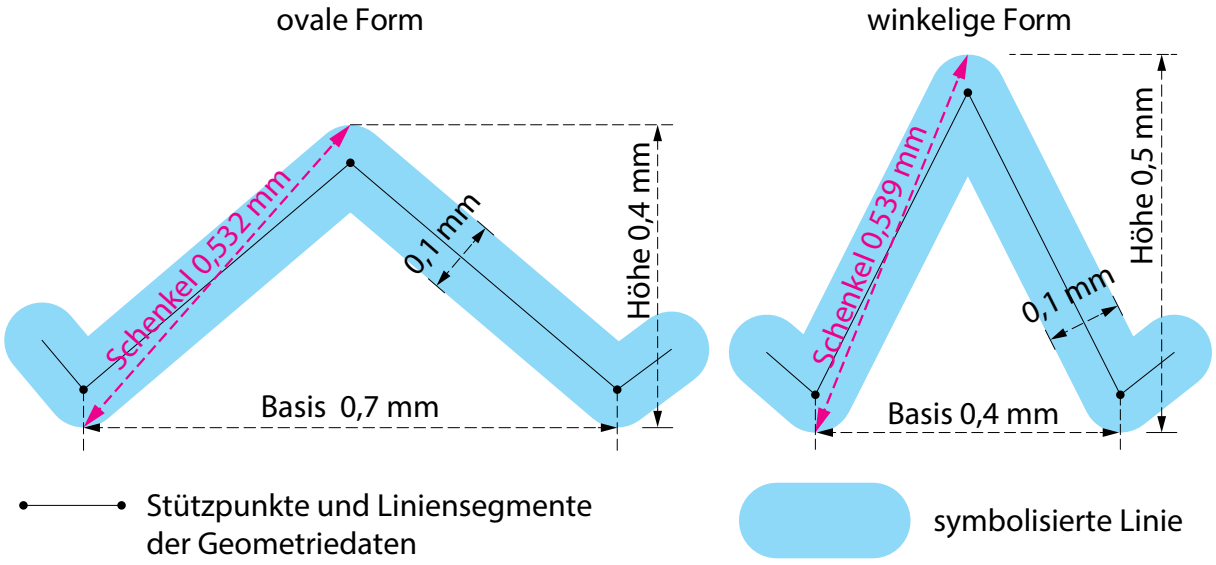


Abb. 8: Elementardreiecke nach Chrobak et al. (2016).
Die Abbildung entspricht Fig. 2 des Artikels.

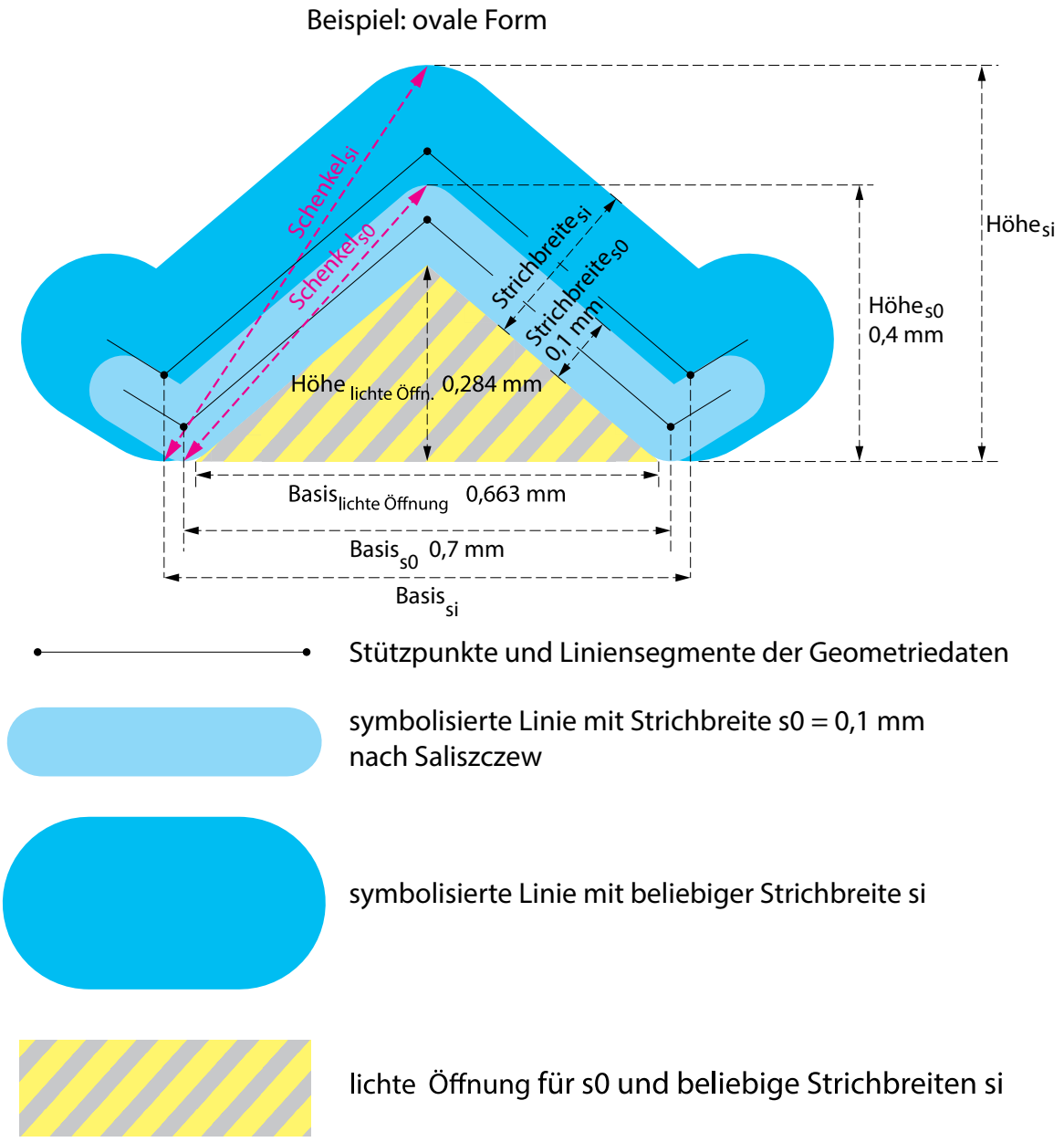


Abb. 9: Elementardreiecke für Linienbreiten s_0 und s_i nach Chrobak et al. (2016). Die Abbildung entspricht Fig. 3 des Artikels. Die Maße der lichten Öffnung können aus dem Elementardreieck von Saliszczew berechnet werden und müssen als Mindestmaße bei jeder beliebigen Strichbreite s_i erhalten bleiben.

Um festzustellen, ob die Abweichungen kleiner sind als das Auflösungsvermögen des menschlichen Auges, werden zunächst die Maße der Elementardreiecke für die gegebene Linienbreite berechnet und mithilfe des Zielmaßstabs, für den die Linie vereinfacht wurde, in Dimensionen der realen Welt umgerechnet. Dann wird für jede eingeschlossene Fläche derjenige Stützpunkt der Originallinie ermittelt, dessen lotrechter Abstand von der generalisierten Linie am größten ist. Er bildet die Spitze eines Dreiecks (im folgenden auch „Liniendreieck“ genannt, siehe Abb. 10).

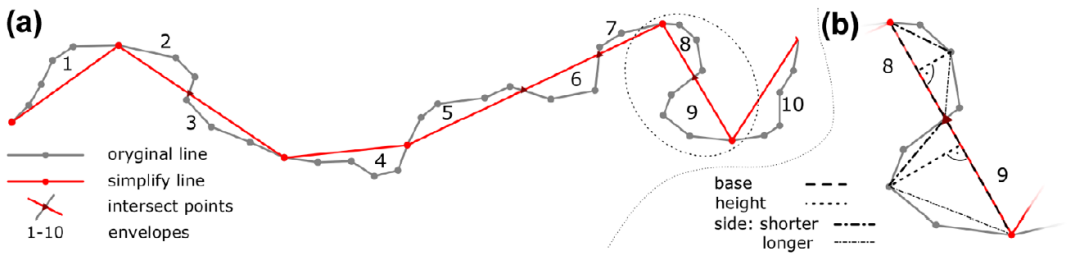


Abb. 10: Dreiecke simulieren wegfallende Flächen.
Entnommen aus Chrobak et al. (2016, Figure 7, S. 246): „The method of assessing the accuracy of simplified data (a) the construction of the envelopes, (b) the triangles built on the envelopes."

Der Abstand zur generalisierten Linie entspricht der Höhe des Dreiecks, die kürzere der beiden Verbindungen von der Dreiecksspitze zu den nächst gelegenen Schnittpunkten im Linienverlauf stellt einen Schenkel des Dreiecks dar; der zweite Schenkel wird als Verbindung zwischen der Dreiecksspitze und dem Schnittpunkt zur anderen Seite konstruiert. Die Verbindung zwischen den beiden Schnittpunkten auf der generalisierten Linie bildet die Basislinie. Diese Maße der Liniendreiecke in der Dimension der realen Welt werden zunächst mit den Mindestmaßen der Elementardreiecke verglichen. Die Elementardreiecke sind jedoch gleichschenkelig, während dagegen die Liniendreiecke jede beliebige Gestalt aufweisen können. Aus diesem Grund werden die Liniendreiecke standardisiert, d.h. mit dem Quotienten Elementardreiecksbasis / Liniendreiecksbasis skaliert (Abb. 11).

Ist eines der standardisierten Dreiecksmaße nun kleiner als im Elementardreieck, rechtfertigt dies das Weglassen des Linienabschnitts im Zuge der Generalisierung, da die Abweichung als nicht mehr sichtbar gilt. Sind alle drei Maße größer oder gleich im Vergleich zum Elementardreieck, gilt die Abweichung als sichtbar und die Generalisierung als zu stark; die Linie sieht an dieser Stelle dem Original nicht mehr hinreichend ähnlich. Auf diese Weise können die Parameter eines Vereinfachungsalgorithmus vom Ergebnis her als geeignet oder ungeeignet beurteilt und gegebenenfalls angepasst werden. Sie werden damit unabhängig vom Vorwissen eines Anwenders.

Zur weiteren Bewertung und zum Vergleich verschiedener Algorithmen verwenden die Autoren folgende Metriken:

- Die Anzahl der sichtbaren und nicht sichtbaren Abweichungen. Je weniger der wegfallenden Linienabschnitte als sichtbar gelten, desto besser wird das Ergebnis bewertet.
- Die Flächenbilanz. Die eingeschlossenen Flächen werden mit einem Attribut versehen, das ihre Lage auf der rechten bzw. linken Seite der vereinfachten Linie angibt. Lässt man nun als weiteres Attribut die Inhalte der eingeschlossenen Flächen berechnen und verwendet je nach Lage zur Linie entgegengesetzte Vorzeichen, ermöglicht dies eine Bilanzierung des Flächeninhalts.
- Die Summe der quadrierten mittleren Fehler der kürzeren Dreiecksseite in den Quelldaten, den envelopes und den coves. Als envelopes bezeichnen die Autoren Liniendreiecke, deren Grundseite aus einem vereinfachenden Liniensegment besteht. Coves dagegen sind lange, schlauchartige Gebilde, die der Linienvereinfachung anheim fallen, bei denen jedoch das vereinfachende Liniensegment die kürzere Dreiecksseite bildet, siehe Abb. 12.

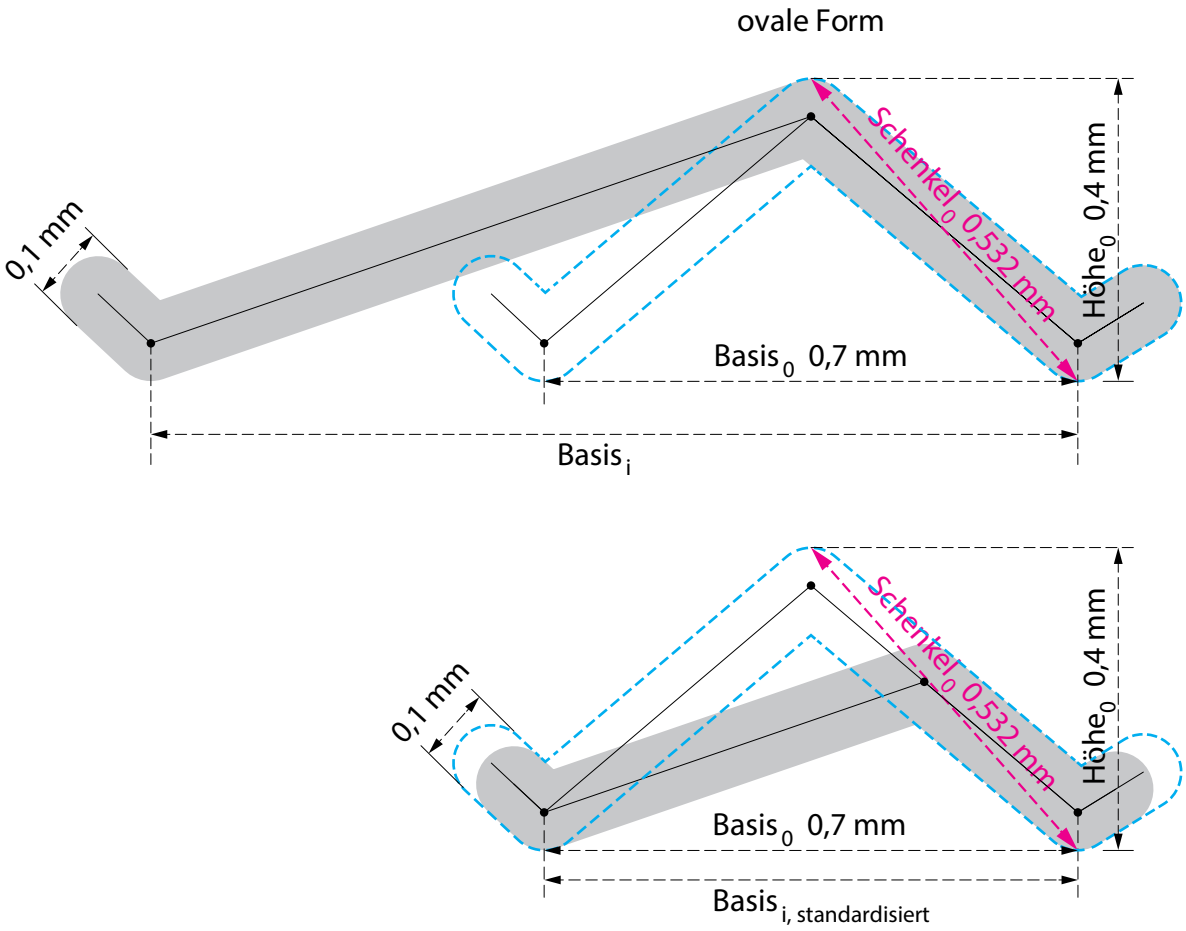


Abb. 11: Standardisierung.
Das Liniendreieck i wird mit dem Quotienten $Basis_0/Basis_i$ skaliert. In diesem Beispiel erreichen die Höhe, die kürzere Dreiecksseite sowie die lichte Öffnung des Liniendreiecks nach der Standardisierung nicht mehr die Mindestmaße des Elementardreiecks.

Auf welche Weise coves automatisiert identifiziert werden, beschreiben die Autoren nicht. Diese Metrik soll die Genauigkeit der vereinfachten Linie messen (Chrobak et al. 2016, S. 248).

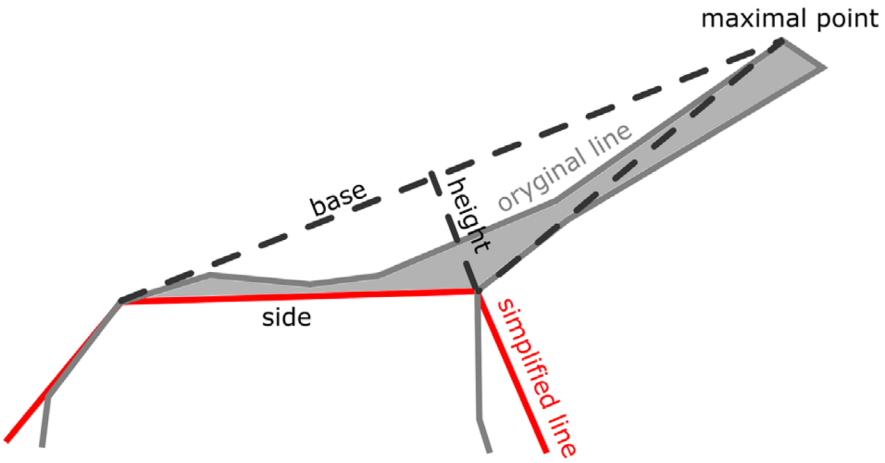


Abb. 12: Bildung des Dreiecks im Fall von coves.
Entnommen aus Chrobak et al. (2016, Figure 6, S. 245).

- p-Value, der es ermöglicht, die Anzahl der verbliebenen Stützpunkte zwischen den angewendeten Algorithmen zu vergleichen. Der p-Value basiert auf dem Wurzelgesetz von Töpfer und Pillewizer (1966). Nach Yu (1993) liegt er für Linienobjekte bei 1. Li (2007) hat für die Berechnung des p-Value folgende Formel entwickelt (Vorzeichen verändert durch Chrobak et al. (2016), siehe Abb. 13):

$$n_j = n_{j-1} \times \left(\frac{M_{j-1}}{M_j} \right)^p \rightarrow p = \frac{\lg\left(n_j / n_{j-1}\right)}{\lg\left(M_{j-1} / M_j\right)} \qquad (32)$$

Abb. 13: Berechnung des p-Value.
n_j: Anzahl der Stützpunkte der vereinfachten Linie, M_j: Maßstabszahl des Zielmaßstabs. n_{j-1}: Anzahl der Stützpunkte der Originallinie. M_{j-1}: Maßstabszahl des Ausgangsmaßstabs. Chrobak et al. (2016, Gleichung 32, S. 252).

Das Verfahren der Autoren zur Bewertung der Genauigkeit der vereinfachten Linie eignet sich nicht für alle Generalisierungsalgorithmen, sondern verlangt folgende Voraussetzungen:

- Die Linie muss Punkte besitzen, die während der Vereinfachung nicht verändert werden. (S. 254)
- Die Koordinaten von Anfangs- und Endpunkt einer Linie dürfen durch die Linienvereinfachung nicht verändert werden.

Begründung für die Wahl der Evaluierungsmethode

Ein Ziel dieser Arbeit war es, zu untersuchen, inwieweit Algorithmen zur Linienvereinfachung in der Lage sind, die Qualität einer manuellen Generalisierung zu erreichen. Mit einer hohen Qualität ist die optimale Wahrung der charakteristischen Form der ursprünglichen Linie gemeint.

Aus den beschriebenen Evaluierungsmethoden wurde das Verfahren von Chrobak et al. (2016) ausgewählt und nach Song und Miao (2016) der Mittelwert und die Standardabweichung der maximalen Distanzen zwischen Originallinie und vereinfachter Linie. Chrobak et al. (2016) verfolgen den Ansatz, das Ergebnis der Linienvereinfachung am Sehvermögen des menschlichen Auges zu messen. Das Sehvermögen bestimmt demnach den Vereinfachungsgrad und die Parameter des Algorithmus, der für die Vereinfachung verwendet wird. Die vereinfachte Linie sieht der Originallinie ähnlich, wenn die Abweichungen im Zielmaßstab nicht sichtbar sind. Wenn aber die vereinfachte Linie der Originallinie ähnlich sieht, ist es gelungen, deren charakteristische Form zu wahren. In diesen Punkten gleicht der Ansatz von Chrobak et al. (2016) dem von Song und Miao (2016). Jedoch behandeln Chrobak et al. (2016) das Sehvermögen differenzierter. Indem sie die Dreiecke, die die wegfallenden Flächen zwischen Originallinie und vereinfachter Linie repräsentieren, auf die Mindestdimensionen nach Saliszczew standardisieren (siehe Abb. 11), stufen sie teilweise auch Dreiecke als nicht sichtbar ein, deren Höhe ohne Standardisierung größer als die Mindestmaße nach Saliszczew ist. Das bedeutet, dass sie Dreiecke mit einer verhältnismäßig langen Basis, aber Höhen, die über 0,4 bzw. 0,5 mm besitzen, als nicht sichtbar einstufen. Song und Miao (2016) sind in diesem Punkt strenger, indem sie keine Dreiecke, sondern ausschließlich die maximalen Distanzen betrachten, die zwischen Originallinie und vereinfachter Linie bestehen. Chrobak et al. (2016) beziehen darüber hinaus die Linienbreite im Zielmaßstab in die Beurteilung der Sichtbarkeit ein.

Das Verfahren nach Shi und Cheung (2006) evaluiert die Ergebnisse ebenfalls nach den maximalen Distanzen zwischen Originallinie und vereinfachter Linie, misst jedoch auch Richtungs- und damit Formveränderungen, die sich durch die Vereinfachung ergeben. Zum einen erscheinen diese Charakteristika aber nicht mehr relevant zu sein, wenn eine Änderung unterhalb der Sichtbarkeitsgrenze liegt, zum anderen wäre für diese Art der Evaluierung weitere, zeitaufwändige Programmierarbeit notwendig gewesen.

3.3 Anwendung der Evaluierung nach Chrobak et al. (2016)

3.3.1 Beschreibung des Python-Skripts

Die Autoren haben für ihr Verfahren ein Python-Skript entwickelt, das Herr Michał Lupa freundlicherweise auf Anfrage zur Verfügung gestellt hat. Das Skript erwartet vom Anwender folgende Eingaben:

- den Datensatz der Originallinien
- den Datensatz der vereinfachten Linien
- die Maßstabszahl des Zielmaßstabs, für den die Linien vereinfacht wurde bzw. in dem sie dargestellt werden soll
- die Strichbreite der Linien im Zielmaßstab in Millimeter
- den Ordner für die Ergebnisdateien

Das Skript erstellt einen Bericht in Form einer Datei *raport_[Dateiname der vereinfachten Linie].txt* mit den verschiedenen Dreiecksmaßen und dem Ergebnis des Vergleichs. Sie kann als csv-Datei in Microsoft Excel eingelesen werden. In der Tabelle steht jede Zeile für eine eingeschlossene Fläche zwischen Originallinie und vereinfachter Linie. Die Bedeutung der Spalten ist in Tabelle 1 erläutert.

Tab. 1: Bedeutung der Spalten im Bericht von Chrobak et al. (2016)

Spaltentitel	Erläuterung
Plik	Dateiname der vereinfachten Linie
Nr	ID der eingeschlossenen Fläche zwischen Originallinie und vereinfachter Linie
Obiekt	Wenn die vereinfachte Linie ein Multi-Part-Objekt ist, werden die einzelnen Bestandteile hier nummeriert
podstawa	Länge der Basis des Dreiecks, das die eingeschlossene Fläche repräsentiert, in der Realität
wysokosc	Höhe des Dreiecks in der Realität, inklusive Linienbreite
ramie	Kürzerer Schenkel des Dreiecks in der Realität, inklusive Linienbreite
N_ow1_b	Länge der Basis des ovalen Elementardreiecks ow1 in der Realität
N_ow1_h	Höhe des ovalen Elementardreiecks ow1 in der Realität
N_ow1_a	Schenkellänge des ovalen Elementardreiecks ow1 in der Realität
N_ow2_b	Länge der Basis des ovalen Elementardreiecks ow2 in der Realität
N_ow2_h	Höhe des ovalen Elementardreiecks ow2 in der Realität
N_ow2_a	Schenkellänge des ovalen Elementardreiecks ow2 in der Realität
N_kan_b	Länge der Basis des spitzwinkelingen Elementardreiecks kan in der Realität
N_kan_h	Höhe des spitzwinkelingen Elementardreiecks kan in der Realität
N_kan_a	Schenkellänge des spitzwinkelingen Elementardreiecks kan in der Realität
N_ow1_h_hi	Wert 1, wenn die Höhe des aktuellen Dreiecks größer ist als die Höhe von ow1 (wysokosc > N_ow1_h); sonst 0
N_ow1_a_ai	Wert 1, wenn der kürzere Schenkel des aktuellen Dreiecks länger ist als der von ow1 (ramie > N_ow1_a); sonst 0
il_ow1	Wert 1, wenn N_ow1_h_hi = 1 oder N_ow1_a_ai = 1; sonst 0
N_ow2_h_hi	Wert 1, wenn die Höhe des aktuellen Dreiecks größer ist als die Höhe von ow2 (wysokosc > N_ow2_h); sonst 0
N_ow2_a_ai	Wert 1, wenn der kürzere Schenkel des aktuellen Dreiecks länger ist als der von ow2 (ramie > N_ow2_a); sonst 0
il_ow (sic!)	Soll vermutlich „il_ow2“ heißen. Wert 1, wenn N_ow2_h_hi = 1 oder N_ow2_a_ai = 1; sonst 0
N_kan_h_hi	Wert 1, wenn die Höhe des aktuellen Dreiecks größer ist als die Höhe von kan (wysokosc > N_kan_h); sonst 0
N_kan_a_ai	Wert 1, wenn der kürzere Schenkel des aktuellen Dreiecks länger ist als der von kan (ramie > N_kan_a); sonst 0

Tab. 1: Fortsetzung

Spaltentitel	Erläuterung
il_kan	Wert 1, wenn N_kan_h_hi = 1 oder N_kan_a_ai = 1; sonst 0
k_ow1	Standardisierungsfaktor für ow1. „Länge der Basis des Elementardreiecks ow1“ geteilt durch „Länge der Basis des aktuellen Dreiecks“ (N_ow1_b / podstawa)
k_ow2	Standardisierungsfaktor für ow2. „Länge der Basis des Elementardreiecks ow2“ geteilt durch „Länge der Basis des aktuellen Dreiecks“ (N_ow2_b / podstawa)
k_kan	Standardisierungsfaktor für kan. „Länge der Basis des Elementardreiecks kan“ geteilt durch „Länge der Basis des aktuellen Dreiecks“ (N_kan_b / podstawa)
N_ow1_bi	Wert 1, wenn k_ow1 < 1
N_ow1_stan_h_hi	Erhält nur dann einen Wert, wenn N_ow1_bi = 1; sonst leer Erhält den Wert 1, wenn die standardisierte Höhe des aktuellen Dreiecks größer ist als bei ow1 (k_ow1*wysokosc > N_ow1_h); sonst 0.
N_ow1_stan_a_ai	Erhält nur dann einen Wert, wenn N_ow1_bi = 1; sonst leer Erhält den Wert 1, wenn der standardisierte kürzere Schenkel des aktuellen Dreiecks größer ist als bei ow1 (k_ow1*ramie > N_ow1_a); sonst 0.
il_ow1_h	Erhält nur dann einen Wert, wenn N_ow1_bi = 1; sonst leer. Wert 1, wenn die standardisierte Höhe UND die Länge der Basis des aktuellen Dreiecks größer sind als bei ow1 (N_ow1_bi = 1 UND N_ow1_stan_h_hi = 1); sonst 0.
il_ow1_a	Erhält nur dann einen Wert, wenn N_ow1_bi = 1; sonst leer. Wert 1, wenn der standardisierte kürzere Schenkel UND die Länge der Basis des aktuellen Dreiecks größer sind als bei ow1 (N_ow1_bi = 1 UND N_ow1_stan_a_ai = 1); sonst 0.
sum_ow1	Wert 1, wenn il_ow1_h = 1 ODER il_ow1_a = 1 (Logische Summe aus il_ow1_h und il_ow1_a); sonst 0 bzw. leer
N_ow2_bi	Wert 1, wenn k_ow2 < 1
N_ow2_stan_h_hi	Erhält nur dann einen Wert, wenn N_ow2_bi = 1; sonst leer. Erhält den Wert 1, wenn die standardisierte Höhe des aktuellen Dreiecks größer ist als bei ow2 (k_ow2*wysokosc > N_ow2_h); sonst 0.
N_ow2_stan_a_ai	Erhält nur dann einen Wert, wenn N_ow2_bi = 1; sonst leer. Erhält den Wert 1, wenn der standardisierte kürzere Schenkel des aktuellen Dreiecks größer ist als bei ow2 (k_ow2*ramie > N_ow2_a); sonst 0.

Tab. 1: Fortsetzung

Spaltentitel	Erläuterung
il_ow2_h	Erhält nur dann einen Wert, wenn N_ow2_bi = 1; sonst leer. Wert 1, wenn die standardisierte Höhe UND die Länge der Basis des aktuellen Dreiecks größer sind als bei ow2 (N_ow2_bi = 1 UND N_ow2_stan_h_hi = 1); sonst 0.
il_ow2_a	Erhält nur dann einen Wert, wenn N_ow2_bi = 1; sonst leer. Wert 1, wenn der standardisierte kürzere Schenkel UND die Länge der Basis des aktuellen Dreiecks größer sind als bei ow2 (N_ow2_bi = 1 UND N_ow2_stan_a_ai = 1); sonst 0.
sum_ow2	Wert 1, wenn il_ow2_h = 1 ODER il_ow2_a = 1 (Logische Summe aus il_ow2_h und il_ow2_a); sonst 0 bzw. leer
N_kan_bi	Wert 1, wenn k_kan < 1
N_kan_stan_h_hi	Erhält nur dann einen Wert, wenn N_kan_bi = 1; sonst leer. Erhält den Wert 1, wenn die standardisierte Höhe des aktuellen Dreiecks größer ist als bei kan (k_kan*wysokosc > N_kan_h); sonst 0.
N_kan_stan_a_ai	Erhält nur dann einen Wert, wenn N_kan_bi = 1; sonst leer. Erhält den Wert 1, wenn der standardisierte kürzere Schenkel des aktuellen Dreiecks größer ist als bei kan (k_kan*ramie > N_kan_a); sonst 0.
il_kan_h	Erhält nur dann einen Wert, wenn N_kan_bi = 1; sonst leer. Wert 1, wenn die standardisierte Höhe UND die Länge der Basis des aktuellen Dreiecks größer sind als bei kan (N_kan_bi = 1 UND N_kan_stan_h_hi = 1); sonst 0.
il_kan_a	Erhält nur dann einen Wert, wenn N_kan_bi = 1; sonst leer. Wert 1, wenn der standardisierte kürzere Schenkel UND die Länge der Basis des aktuellen Dreiecks größer sind als bei kan (N_kan_bi = 1 UND N_kan_stan_a_ai = 1); sonst 0.
sum_kan	Wert 1, wenn il_kan_h = 1 ODER il_kan_a = 1 (Logische Summe aus il_kan_h und il_kan_a); sonst 0 bzw. leer
Plik	Dateiname der vereinfachten Linie

Ein Sachverhalt, der in Tabelle 1 nicht leicht ersichtlich ist, muss betont werden: Eine Standardisierung erfolgt nur dann, wenn die Basis des Dreiecks größer ist als die Basis des Elementardreiecks im Realweltmaß (siehe N_ow1_stan_h_hi, N_ow1_stan_a_ai, N_ow2_stan_h_hi, N_ow2_stan_a_ai, N_kan_stan_h_hi, N_kan_stan_a_hi). Das bedeutet, dass die Höhe des Dreiecks für die Entscheidung über Sichtbarkeit oder Unsichtbarkeit gänzlich unberücksichtigt bleibt, wenn die Dreiecksbasis kleiner ist als im Elementardreieck.

Außerdem produziert das Skript vier Shapefiles:

- *[Dateiname der vereinfachten Linie]_OTO.shp*: die eingeschlossenen Flächen (Abweichungen) (Polygone)
- *[Dateiname der vereinfachten Linie]_POD.shp*: die Basen der Dreiecke, die die Abweichungen repräsentieren (Polylinie)
- *[Dateiname der vereinfachten Linie]_RAM.shp*: die kürzeren Schenkel der Dreiecke (Polylinie)
- *[Dateiname der vereinfachten Linie]_WYS.shp*: die Höhen der Dreiecke (Polylinie)

Die Werte der im Kapitel 3.2 *Evaluierung nach Chrobak et al. (2016)* beschriebenen Metriken werden nicht direkt vom Skript ausgegeben, sondern müssen aus den Dateien, die das Skript erstellt, abgeleitet werden.

3.3.2 Probleme mit der Evaluierung von Chrobak et al. (2016)

Etliche Unstimmigkeiten in Formeln, Berechnung und in Verweisen auf Abbildungen, Tabellen und Gleichungen erschwerten es, das Verfahren der Autoren nachzuvollziehen. Eine kommentierte Version des Artikels ist als Anhang B beigefügt.

Der Code war im **Skript in polnischer Sprache kommentiert**. Die Bearbeiterin hat die Kommentare mithilfe von Google Translator übersetzt.

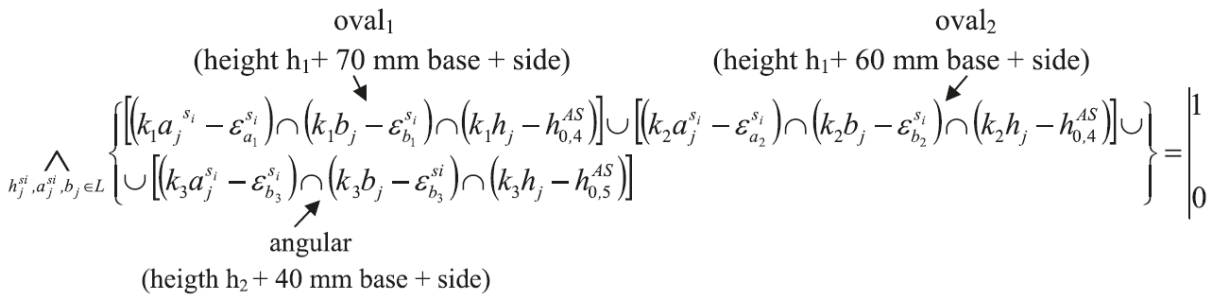
Im Python-Skript der Autoren wird das Kriterium für die Sichtbarkeit der Abweichungen zwischen Originallinie und vereinfachter Linie anders formuliert als das entsprechende Kriterium (23) im Artikel von Chrobak et al. (2016, S. 245). Zusätzlich scheinen im Artikel die ≥-Zeichen irrtümlich oder vielleicht durch den Druckprozess mit Minuszeichen ersetzt worden zu sein. Mit Minuszeichen ergibt das Kriterium keinen Sinn (siehe Abb. 14). Das Kriterium (23) im Artikel besagt, dass in einem Liniendreieck, das eine Abweichung zwischen Originallinie und vereinfachter Linie repräsentiert, der standardisierte kürzere Dreiecksschenkel UND die standardisierte Dreiecksbasis UND die standardisierte Dreieckshöhe größer als im Elementardreieck sein müssen, damit das Liniendreieck als sichtbar gilt. Die drei Bedingungen sind also durch den logischen Operator UND verknüpft, d.h. es wird das logische Produkt gebildet (vgl. Chrobak et al. 2016, Table 2, Spalten 20-22, S. 247).

Im Skript dagegen lautet die Bedingung für die Dreiecke, die die Abweichungen repräsentieren: standardisierte Höhe UND standardisierte Basis müssen größer sein als im Elementardreieck ODER standardisierter kürzerer Dreiecksschenkel UND standardisierte Basis müssen größer als im Elementardreieck sein, siehe Abb. 15. Diese Diskrepanz zwischen Artikel und Python-Skript bedeutet, dass Abweichungen, die gemäß dem Kriterium im Artikel als nicht sichtbar gelten, bei einer Evaluierung mit dem Python-Skript durchaus als sichtbar eingestuft werden können. Die Bearbeiterin hat sich für das Kriterium im Python-Skript entschieden, und zwar aus folgenden Gründen:

Nach dem visuellen Eindruck der Ergebnisse erschienen selbst Abweichungen als wahrnehmbar, die das Skript als nicht sichtbar bezeichnete. Wenn das Kriterium (23) des Artikels verwendet werden würde, könnten noch mehr Abweichungen sichtbar sein.

Sowohl im Artikel als auch im Skript werden die tatsächlichen Maße der Liniendreiecke durch die Standardisierung unter Umständen stark verkleinert, so dass sich die Frage stellt, ob sie in ihren realen Ausmaßen nicht doch sichtbar wären.

Deshalb wurde für die Evaluierung das „weichere“ Kriterium des Python-Skripts verwendet.



where:

h_j – the real height of the triangle built on line L,
b_j – the real lengths of the triangle,
$h_{0.4}^{AS}$ – the norm of the oval triangle height by Saliszczew 2003,
$h_{0.5}^{AS}$ – the norm of the angular triangle height by Saliszczew,
$b_j^{0.7}$ – the norm of the base length in the first oval triangle Saliszczew,
$b_j^{0.6}$ – the norm of the base length in the second oval triangle Saliszczew,
$b_j^{0.4}$ – the norm of the base length in the angular triangle Saliszczew.

Abb. 14: Kriterium (23) aus Chrobak et al. (2016). (S. 245–246)

```
#-------------------------ow1 ----- stan          # Elementardreieck, ovale Form ow1, Standardisierung
if bb >=  b_ver_ow1_szer:                        # bb: Basis des Abweichungsdreiecks vor der Standardisierung
    N_ow1_bi = 1                                 # b_ver_ow1_szer: Basis des Elementardreiecks der Form ow1 inklusive Strichbreite (szer)
else: N_ow1_bi = 0                               # N_ow1_bi: Logische Variable, die auch in der Berichtsdatei eingetragen ist

if k_ow1 < 1:                                    # k_ow1 = b_ver_ow1_szer / bb (Zeile 391 im Python-Skript ocena_1.py)
    if hh * k_ow1 >= h_out_ow1_szer:             # hh: Hoehe des Abweichungsdreiecks vor der Standardisierung
        N_ow1_stan_h_hi = 1                      # h_out_ow1_szer: Hoehe des Elementardreicks der Form ow1 inklusive Strichbreite (szer)
    else: N_ow1_stan_h_hi = 0                    # N_ow1_stan_h_hi: Logische Variable, die auch in der Berichtsdatei eingetragen ist

    if aa * k_ow1 >= a_out_ow1_szer:             # aa: Kuerzerer Schenkel des Abweichungsdreiecks vor der Standardisierung
        N_ow1_stan_a_ai = 1                      # a_out_ow1_szer: Kuerzerer Schenkel von ow1 inklusive Strichbreite (szer)
    else: N_ow1_stan_a_ai = 0                    # N_ow1_stan_a_ai: Logische Variable, die auch in der Berichtsdatei eingetragen ist

    if N_ow1_stan_h_hi == 1 and N_ow1_bi == 1:   # Bedingung (1) "Wenn die standardisierte Hoehe und die standardisierte Basis
        il_ow1_h = 1                             # des Abweichungsdreiecks groesser sind als im Elementardreieck: dann ist il_ow1_h = 1"
    else: il_ow1_h = 0                           # il_ow1_h: Logische Variable, die auch in der Berichtsdatei eingetragen ist

    if N_ow1_stan_a_ai == 1 and N_ow1_bi == 1:   # Bedingung (2) "Wenn der standardisierte kuerzere Schenkel und die standardisierte Basis
        il_ow1_a = 1                             # des Abweichungsdreiecks groesser sind als im Elementardreieck: dann ist il_ow1_a = 1"
    else: il_ow1_a = 0                           # il_ow1_a: Logische Variable, die auch in der Berichtsdatei eingetragen ist

    if il_ow1_h == 1 or il_ow1_a == 1:           # "Wenn Bedingung (1) ODER Bedingung (2) erfuellt ist:...
        sum_ow1 = 1                              # ...dann ist sum_ow1 = 1", d.h. das Abweichungsdreieck ist sichtbar.
    else: sum_ow1 = 0                            # sum_ow1: Logische Variable, die auch in der Berichtsdatei eingetragen ist
else:
    N_ow1_stan_h_hi = ""
    N_ow1_stan_a_ai = ""
    il_ow1_h = ""
    il_ow1_a = ""
    sum_ow1 = ""
```

Abb. 15: Ausschnitt aus dem Python-Skript zu Chrobak et al. (2016). Zeilen 438 – 468. Kommentare der rechten Seite von der Bearbeiterin hinzugefügt. Die Skriptzeilen lauten analog für Elementardreiecke der Form ow2 (ovale Form mit einer Basis von 0,6 mm) und kan (spitzwinkelige Form).

Die Identifizierung von coves und ihre Behandlung im Evaluierungsprozess wird von Chrobak et al. (2016) nicht im Detail beschrieben. Im Artikel heißt es auf S. 246: „Bei der Verifizierung von coves (Figure 6) muss die Kompatibilität mit Saliszczews Mindestmaßen durch entsprechende Maße für die Höhe, Basis und den kürzeren Dreiecksschenkel gewahrt wer-

den. Das Messkriterium wird nur dann erfüllt, wenn sich die Dreiecksseiten mit der Uferlinie des cove überlappen (und die Basis schließt das Dreieck).“ (Übersetzung Chr. Enderle; statt „überlappen“ vielleicht besser „schneiden“).

Abb. 12 zeigt die genannte Figure 6 des Artikels und illustriert die Handhabung von coves. Das ersetzende Segment wird nicht als Dreiecksbasis, sondern als der kürzere Dreiecksschenkel verwendet, der längere Dreiecksschenkel wird von der Dreiecksspitze aus bis zum entlegensten Punkt der Form („maximal point“) konstruiert, woraus sich dann auch die Dreiecksbasis und die Höhe des Dreiecks ergeben. Ob und wie die Identifizierung automatisiert abläuft, sagen die Autoren nicht. Die Ergebnisse des Python-Skripts zeigen keine Sonderbehandlung von coves. Eine mögliche mathematische Bedingung zur Identifizierung hat die Bearbeiterin in Abb. 16 formuliert.

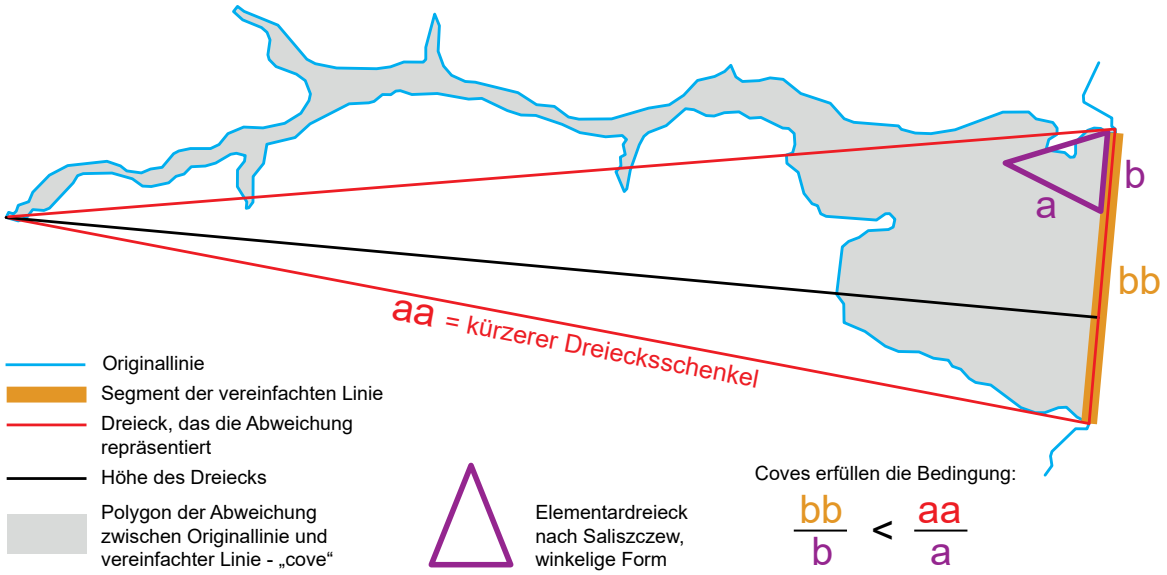


Abb. 16: Identifizierung von Coves

Die Berechnung bzw. das Zustandekommen des Wertes, der die Genauigkeit der vereinfachten Linie angibt, war für die Bearbeiterin nicht nachvollziehbar. Auf eine Nachfrage bei einem der Autoren erhielt sie keine Antwort.

Abb. 17 zeigt die Gleichung für die Berechnung dieses Wertes als „Average error of the side length“, der in Table 6 bei Chrobak et al. (2016, S. 253) ausgewiesen ist, mit den Erläuterungen der Variablen durch die Autoren.

$$m_{d_{env}} = \sqrt{\frac{\sum_{i=1}^{i=n} V_{ai} V_{ai}}{n-1}}, \tag{25}$$

where: $V_i = a_i \quad i = 1,\ 2,\ 3, \ldots, n$

Moreover, shorter sides in triangles which replaced the removed segments from the source line are random errors V_{a_i} in Equation (25).

Abb. 17: Gleichung (25) aus Chrobak et al. (2016, S. 248).

$$m_{d_{simp}}{}^2 = m_{d_{src}}{}^2 + m_{d_{env}}{}^2 + m_{cov}{}^2 \quad (26)$$

where:

m_{src} – the source data,
m_{env} – unrecognizable shorter envelope side,
m_{cov} – unrecognizable shorter cove side (Equation 25).

Abb. 18: Gleichung (26) aus Chrobak et al. (2016, S. 248).

Abb. 18 zeigt die Gleichung, die die Datengenauigkeit insgesamt berechnet.

Die Bearbeiterin legt im folgenden ihre Auslegung dar:

Die Gleichung (25) in Abb. 17 wird für die Berechnung von m_{src}, m_{env} und m_{cov} verwendet. Für m_{src} ist V_{ai} die Länge desjenigen Liniensegments der Originallinie, das durch den kürzeren Dreiecksschenkel repräsentiert wird. N ist die Anzahl der zwischen Originallinie und vereinfachter Linie eingeschlossenen Flächen, die als nicht sichtbar gelten. Für die Berechnung von m_{env} ist V_{ai} die Länge des kürzeren Dreiecksschenkels, ohne coves, und n ist die Anzahl der eingeschlossenen Flächen, jedoch abzüglich der Anzahl der coves. Für die Berechnung von m_{cov} ist V_{ai} die Länge des kürzeren Dreiecksschenkels bei coves und n die Anzahl der coves. Wie m_{simp} als Summe aus den Quadraten von m_{src}, m_{env} und m_{cov} ein Maß für die Genauigkeit der vereinfachten Linie darstellen kann, blieb der Bearbeiterin unklar. Unverständlich ist darüber hinaus, dass die Autoren diese Summe als Standardabweichung bezeichnen (Chrobak et al. 2016, S. 254). Aufgrund der unsicheren Sachlage entschied sich die Bearbeiterin dafür, auf die beschriebene Metrik zu verzichten. Die Evaluierung der geometrischen Qualität beschränkt sich im wesentlichen auf die visuelle Untersuchung.

Bei den Linienvereinfachungen für kleinere Maßstäbe ergaben sich teilweise größere Diskrepanzen zwischen den wegfallenden Polygonen und den Dreiecken, die diese Polygone repräsentieren sollen. Abb. 19 zeigt vier Beispiele.

Das Phänomen ist selbst in der Untersuchung, die Chrobak et al. (2016) durchgeführt haben, zu beobachten. Herr Michał Lupa hatte zusammen mit dem Python-Skript die untersuchte Testlinie übersandt. Es handelt sich dabei um einen Küstenabschnitt der Eddrachillis Bay in Schottland. Abb. 20 zeigt zwei Beispiele.

Angesichts dieser Diskrepanzen stellt sich die grundsätzliche Frage, ob das von Chrobak et al. (2016) entwickelte Verfahren, das die konstruierten Dreiecke mit den Elementardreiecken nach Saliszczew vergleicht, die Sichtbarkeit der Abweichungen und die Genauigkeit der vereinfachten Linie realistisch bewerten kann.

Trotz der aufgezeigten Unsicherheiten wurde das Verfahren von Chrobak et al. (2016) für die Ermittlung geeigneter Vereinfachungsparameter verwendet.

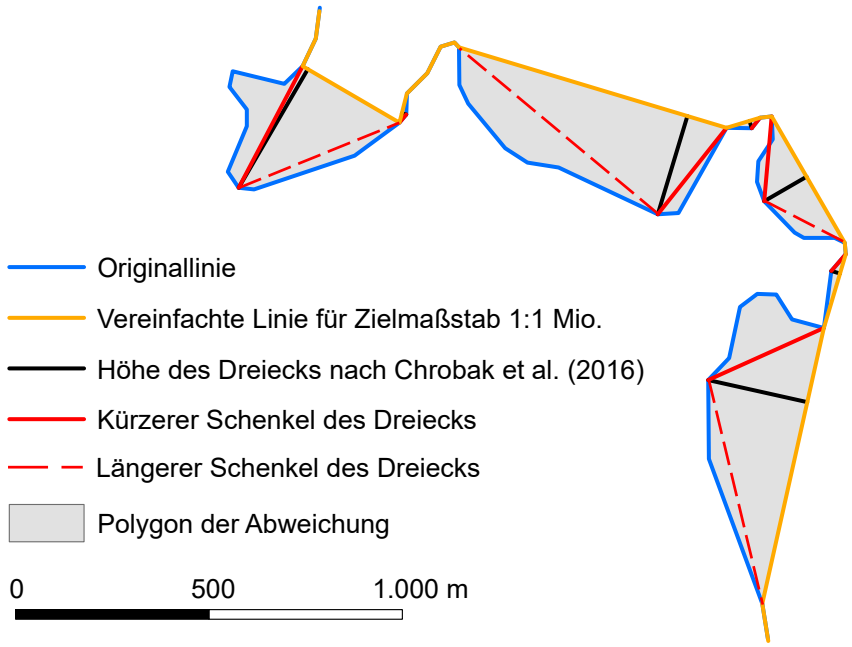


Abb. 19: Unterschiede zwischen Dreiecken und wegfallenden Polygonen (Bretagne). Die Originallinie – der untersuchte Küstenabschnitt der Bretagne – wurde mit dem Algorithmus nach Ai et al. (2016) für den Zielmaßstab 1:1Mio. vereinfacht. Die Abbildung hat einen größeren Maßstab.

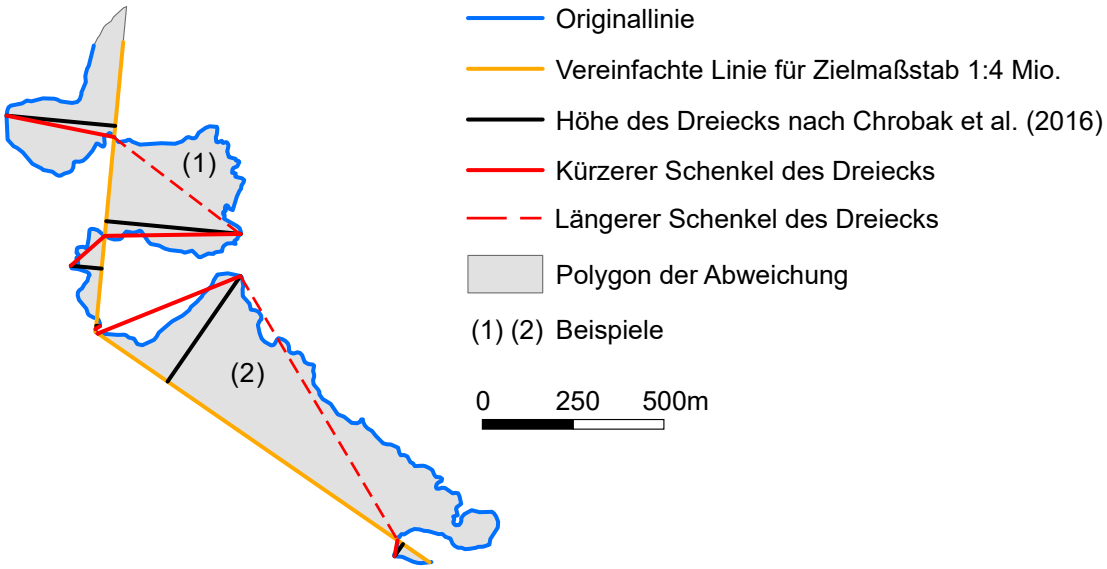


Abb. 20: Unterschiede zwischen Dreiecken und wegfallenden Polygonen (Schottland). Küstenlinie von Schottland und ihre Vereinfachung mit dem ArcGIS-Tool *Simplify Line*, Methode *Point Remove*, Toleranz 400 m, für den Zielmaßstab 1:4 Mio. Die Autoren verwenden für diesen Zielmaßstab eine Toleranz von 1200 m bzw. 1600 m (Chrobak et al. 2016, Table 4, S. 249). Die Abbildung hat einen größeren Maßstab.

3.4 Verwendete Metriken nach Chrobak et al. (2016)

Von den beschriebenen Metriken nach Chrobak et al. (2016) wurden folgende verwendet:

Anzahl der sichtbaren Abweichungen als Qualitätsmaßstab. Sie kann aus der csv-Datei, die das Python-Skript der Autoren als Ergebnis schreibt, abgeleitet werden, und zwar als logische Summe aus den Werten der Spalten sum_ow1, sum_ow2 und sum_kan (siehe Tabelle 1). Ergibt sich ein Wert von 1, ist die Abweichung sichtbar. Wie im Kapitel 4.2.2 erläutert, wurden die Eingabeparameter für den Algorithmus so gewählt, dass die Anzahl der sichtbaren Abweichungen 0 ergab.

p-Value – ein Maß für die Reduzierung der Stützpunkte, siehe Kapitel 3.2.

3.5 Verwendete Metriken nach Song und Miao (2016)

Zusätzlich wurden nach Song und Miao (2016) **Mittelwerte der maximalen orthogonalen Distanzen** in allen wegfallenden Flächen zwischen Ausgangsmaßstab 1:50k und dem jeweiligen Zielmaßstab und deren **Standardabweichung** ermittelt. Als maximale orthogonale Distanz wurden die Dreieckshöhen nach Chrobak et al. (2016) verwendet. In Abweichung zu Song und Miao (2016) wurde dabei auch die Strichstärke der Linien berücksichtigt.

3.6 Weitere Metriken

Kompressionsrate

Neben dem p-Value wurde die üblichere Metrik der Kompressionsrate der Stützpunktanzahl für die Reduzierung der Datenmenge berechnet, und zwar nach folgender Formel:

$r = 100 – v_{simplified} / v_{original}\ 100$

wobei r = Kompressionsrate, $v_{simplified}$ = Anzahl der Stützpunkte der vereinfachten Linie, $v_{original}$ = Anzahl der Stützpunkte der Originallinie, und zwar im Fall des Algorithmus von Ai et al. (2016) vor der Vereinfachung mit dem ArcGIS-Tool *Simplify Line*, Methode *Point Remove* (siehe Kapitel 2.2). Die Anzahl der Stützpunkte wurde im Fall des Algorithmus von Ai et al. (2016) aus der Attributtabelle des Shapefiles *F_Simplified_Points.shp* abgelesen, für die manuelle Vereinfachung aus den Eigenschaften der Skizze bei geöffneter Editiersitzung in ArcGIS.

Flächenbilanz

Das Python-Skript von Chrobak et al. (2016) erstellt die Datei *G_Simplified_Line_OTO.shp* mit den Polygonen der Abweichungen zwischen Originallinie und vereinfachter Linie ohne Raumbezug. Es wurde ihr also zunächst das Koordinatensystem der Originallinie zugewiesen. Anschließend wurde mit dem ArcGIS-Tool *Add Geometry Attribute* eine neue Attributspalte mit dem Flächeninhalt (Methode *AREA_GEODESIC*) in Quadratkilometern hinzugefügt. Für die Methoden *left* bzw. *right* wurden die Summen der Flächen aus dem Fenster, das mit dem Spaltenmenüpunkt *statistics* aufgerufen wird, abgelesen. Für die Methode *area preserving simple* konnte für die Unterscheidung der Abweichungen auf den beiden Seiten der Originallinie das Attribut *PL* verwendet werden, das das Python-Skript von Chrobak et al. (2016) den Abweichungen zuweist und die Abweichungen als links von der vereinfachten Linie mit dem Buchstaben *L* bzw. rechts mit dem Buchstaben *P* kennzeichnet.

Da es sich bei der Testlinie um einen Küstenabschnitt handelt, wurden die sich ergebenden Landgewinne auf der linken Seite der Originallinie jeweils mit einem positiven Vorzeichen festgehalten, die Landverluste auf der rechten Seite mit negativem Vorzeichen. Für die Ergebnistabelle (Tabelle 8) wurden die Werte der Flächenbilanz in die Einheit Hektar umgerechnet.

Berechnungszeit

Die Berechnungszeit wurde aus den Meldungen im Fenster der Geoverarbeitung in ArcGIS übernommen.

Visuelle Bewertung der Liniengestalt

Wie bereits erwähnt, wurden die Abweichungen zur Originallinie zusätzlich visuell auf Sichtbarkeit und Formtreue untersucht. Die visuelle Bewertung der vereinfachten Linien geschah sowohl in einer vergrößerten Ansicht am Computerdisplay als auch in einem Ausdruck im Zielmaßstab (siehe Anhang C1 - C11).

4. Methodik der Untersuchung

Die Qualität des Ergebnisses nach Anwendung eines Algorithmus zur Linienvereinfachung sollte am Ergebnis einer manuellen Generalisierung derselben Linie gemessen werden. Zu diesem Zweck wurde eine Testlinie aus analog erstellten Karten im Maßstab 1:50k als Vektordaten digitalisiert und mit dem zu untersuchenden Algorithmus für die Zielmaßstäbe 1:100k, 1:250k, 1:1 Mio., 1:2,5 Mio. und 1:12,5 Mio. vereinfacht. Das Ergebnis wurde zum einen jeweils mit einer manuellen Generalisierung für den entsprechenden Zielmaßstab visuell verglichen, zum anderen in Bezug auf die Originallinie evaluiert. Zusätzlich wurden die Werte für die in den Kapiteln 3.4 – 3.6 beschriebenen Metriken sowohl für die automatisiert als auch für die manuell vereinfachten Linien ermittelt und miteinander verglichen.

4.1 Testdaten für den Vergleich mit der manuellen Generalisierung

Die Forschungsfragen entspringen dem beruflichen Aufgabenfeld der Bearbeiterin, in dem häufig Linienelemente wie Küsten, Flüsse und Grenzen in den unterschiedlichsten Maßstäben benötigt werden.

4.1.1 Regionale Auswahl der Daten

Je nach Liniengestalt ist die Bewahrung der charakteristischen Form durch den Vereinfachungsvorgang hindurch eine Herausforderung sowohl für die manuelle Vorgehensweise als auch für rechnergestützte Verfahren. Die Gestalt von Küsten und Flüssen variiert von nahezu geradlinigen über kurvige bis hin zu stark zerklüfteten Formen. Grenzverläufe hingegen sind künstliche, häufig geradlinige oder eckige Gebilde, wo sie nicht natürlichen Objekten wie Flüssen oder Gebirgszügen folgen. Aus dem Spektrum der Küstenformen wurden anhand der Beschreibung von Valentin (1954) ein Abschnitt der Küste der Bretagne in Frankreich als Riasküste ausgewählt (siehe Anhang C1). Sie hat einen kurvigen Verlauf und weist tiefe, schmale Einschnitte ins Landesinnere auf.

4.1.2 Referenzdaten zur Beurteilung der manuellen Generalisierung

Als Vorlage für die Ausgangsdaten im Maßstab 1:50k sowie die Referenzdaten der Maßstäbe 1:100k, 1:250k dienten amtliche Karten. Die meisten Staaten des europäischen Raumes sind technisch, wirtschaftlich und militärisch hochentwickelt. Insbesondere das Militär ist auf genaue Karten angewiesen. Deshalb erscheint die Annahme berechtigt, dass amtliche Karten dieser Staaten höchsten Ansprüchen an Genauigkeit und kartographische Qualität genügen. Für den Maßstab 1:1 Mio. wurde ein Kartenblatt der Internationalen Weltkarte herangezogen, für die Maßstäbe 1:2,5 Mio. und 1:12,5 Mio. Karten aus dem Times-Atlas (1:2,5 Mio. aus Bartholomew 1967, S. 67; 1:12,5 Mio. aus Bartholomew 1955, S. 49). Die Auswahl der Maßstäbe musste sich danach richten, welche Maßstäbe in den amtlichen Kartenwerken und in den Atlanten vorhanden sind.

Die verwendeten Karten stammen aus der Zeit zwischen 1952 und 1967. Damit ist sichergestellt, dass sie manuell generalisiert worden sind, da bis zu diesem Zeitpunkt die automatisierte Generalisierung noch nicht im Produktionsprozess von Karten etabliert war (bezogen auf Frankreich Stoter 2005, S. 4).

Die Karten wurden auf einem Großformatscanner gescannt bzw. als Scans über das Internet zur Verfügung gestellt. Sie wurden mit der Software ArcGIS Desktop 10.3.1 georeferenziert und der

gewünschte Linienabschnitt ebenfalls mit ArcGIS in höchstmöglicher Genauigkeit und in einer jeweils 12,5-fachen Vergrößerung als Vektordaten digitalisiert. Tabelle 2 zeigt eine Übersicht der analog erstellten Kartenvorlagen, die für den Vergleich verwendet werden sollten.

Tab. 2: Übersicht der analogen Kartenvorlagen.

Maßstab	Kartenwerk, Kartenblatt	Jahr der Ausgabe	Bodenauflösung	Quelle
1:50k	Carte 1:50000, Nr. 0515, 0615, 0616	1957, 1957, 1952	1 Pixel entspricht 5 m x 5 m (254 dpi bei 1:50k)	IGN Institut National de l'Information Géographique et de Forestière (2017, 2014)
1:100k	Carte de France_1/100000, Morlaix (Feuille B-8)	1958	1 Pixel entspricht 8,47 m x 8,47 m (300 dpi bei 1:100k)	IGN Institut Géographique National (1958)
1:250.000	Carte de France_1/250000, Brest, Feuille NL-NM 30-10	1961	1 Pixel entspricht 21,17 m x 21,17m (300 dpi bei 1:250.000)	IGN Institut Géographique National (1961a)
1:250.000	Carte de France_1/250000, Saint-Brieuc, Feuille NM 30-11	1961	1 Pixel entspricht 21,17 m x 21,17m (300 dpi bei 1:250.000)	IGN Institut Géographique National (1961b)
1:1.000.000	International Map of the World, London, NM 30 & PTS. 29 & 31, Series 1301	1954	1 Pixel entspricht 151,5 m x 151,5 m (168 dpi bei 1:1 Mio.)	Ordnance Survey (1954)
1:2,5 Mio.	The Times Atlas of the World, Plate 67 France	1967	1 Pixel entspricht 106,04 m x106,04 m (600 dpi in 1:2,5 Mio.)	Bartholomew (1967)
1:12,5 Mio.	The Times Atlas of the World, Plate 49 Europe	1955	1 Pixel entspricht ca. 531,91 m x 531,91 m (600 dpi bei 1:12,5 Mio.	Bartholomew (1955)

4.2 Untersuchung des Algorithmus

Wie im Kapitel 2.2 beschrieben, musste die digitalisierte Originallinie 1:50k einer Vorverarbeitung unterzogen werden. Mit dem ArcGIS-Tool *Simplify Line*, Methode *Point Remove*, wurde die Punktzahl der Originallinie reduziert. Die Methode *Point Remove* basiert auf dem Douglas-Peucker-Algorithmus (ESRI 2016a). Das Ergebnis wird im folgenden als Originallinie_DP bezeichnet.

4.2.1 Aufbau der bend-Hierarchie

Anschließend wurde das von der Bearbeiterin entwickelte Modell *Preparing Steps* zum Aufbau des hierarchischen Systems der bends angewendet. Dieser Schritt ist unabhängig vom Zielmaßstab und damit nur ein einziges Mal erforderlich, da hiermit nur die Daten für die weitere Verarbeitung beschrieben und strukturiert werden. Er liefert in Form von drei Shapefiles die Informationen über die Liniengestalt, die für die eigentliche Linienvereinfachung mit Modell *FindParameters* notwendig sind, siehe Tabelle 3.

Tab. 3: Output des Modells *Preparing Steps*

Name	Geometrie	Relevante Attribute
C_Bendsystems_Topo.shp	Bend systems. Polygone, die sich aus der Verschneidung der Originallinie_DP mit der Domain (der Außengrenze) der Triangulation ergeben, siehe Abb. 1	Topology: gibt mit den Werten L bzw. R die Lage des Bendsystems links bzw. rechts vom Verlauf der Originallinie_DP an
C_Skelettpaths.shp	Längster Skelettpfad eines jeden bends, siehe Abb. 4	BSID: Feature-ID des zugehörigen Bendsystems TID: Feature-ID des lokalen Dreiecks, das in den bend (Kurvenabschnitt) führt HS: Hierarchiestufe des zugehörigen bends LENGTH: Länge des Skelettpfades in Karteneinheiten. BOPO: Feature_ID des Bottom Point (innerster Punkt des bends)
C_Triangle_segments_BSID.shp	Segmente der Dreiecke, ohne diejenigen, die auf der Originallinie_DP liegen. Sie werden zum Aufbau der envelope zones und der vereinfachten Linienabschnitte benötigt.	BS_FID: Feature-ID des zugehörigen bendsystems MID_X, MID_Y: Koordinaten des Segmentmittelpunktes

4.2.2 Eingabeparameter für die Linienvereinfachung

Ausgehend von den Ergebnissen des Modells *Preparing Steps* wurde die Linie für die genannten Zielmaßstäbe vereinfacht, und zwar jedes Mal ausgehend von der Originallinie_DP im Maßstab 1:50k.

Als Eingabeparameter epsilon, der sich unmittelbar auf die Vereinfachung auswirkt, diente wie bei Ai et al. (2016) die bend depth, also die Länge des jeweils längsten Skelettpfades im bend. Wird der Parameter gamma auf einen Wert > 1 gesetzt, bedeutet dies, dass bends, deren längster Skelettpfad kleiner ist als epsilon, dennoch erhalten bleiben, wenn dieser bend

Bestandteil eines übergeordneten, größeren bends ist, dessen längster Skelettpfad länger ist als das Produkt aus epsilon × gamma. Dies hat, wie im Kapitel 2.1 beschrieben, zum Ziel, die Endpunkte großer bends als charakteristisch zu identifizieren und zu erhalten. Für die Bewertung der Sichtbarkeit nach Chrobak et al. (2016) bedeutet dies:

- Abweichungen, die bei einer Nichtverwendung von gamma (gleichbedeutend mit gamma = 1) gegebenenfalls zu groß wären, werden bei einem gamma > 1 möglicherweise entfallen, weil der bend mit dem längeren Skelettpfad erhalten bleibt und daher keine Abweichung zur Originallinie entsteht.
- Allerdings können theoretisch nachgeordnete bends des erhalten gebliebenen bends dadurch als Abweichungen auftreten, die eventuell als sichtbar identifiziert werden, zwar nicht aufgrund der Länge des Skelettpfades, aber aufgrund der Form der Abweichung.

Um denjenigen Wert für den Eingabeparameter zu finden, mit dem die Abweichungen zur Originallinie_DP einerseits nicht sichtbar sind, die Vereinfachung andererseits jedoch auch nicht zu schwach, wurde aus den beiden Python-Skripts *EnvelopeZones.py* und *GenerateSimplifiedLine.py* zusammen mit weiteren notwendigen ArcGIS-Tools und dem Python-Skript von Chrobak et al. (2016) das Modell *FindParameters* erstellt, siehe Abb. 6. Ideal wäre es, dieses Modell für einen iterativen Prozess zu programmieren. Aus Zeitgründen wurde diese Programmierung nicht durchgeführt. Die Wahl der Parameter erfolgte jedoch nach einem Prinzip, das sich in eine solche Programmierung umsetzen lässt. Dieses Prinzip ist in Abb. 21 dargestellt.

Der Anfangswert für epsilon orientierte sich an den Höhen der Elementardreiecke nach Saliszczew (1998) für den jeweiligen Zielmaßstab. Dabei wurde der epsilon gegenüber diesem Mindestmaß etwas erhöht, da die Längen der Skelettpfade in der Regel nicht den maximalen orthogonalen Distanzen zwischen der Originallinie und der vereinfachten Linie entsprechen. Für die Festlegung des Anfangswertes von gamma wurde die Linienform visuell auf erhaltenswerte bends untersucht. Die Länge des zugehörigen längsten Skelettpfades bildet den Schwellenwert epsilon_big:

epsilon_big = epsilon × gamma

Führte der Anfangswert von epsilon zu keinen sichtbaren Abweichungen, wurde der Wert für epsilon verdoppelt und gamma so angepasst, dass epsilon_big konstant blieb. Dann wurde das Modell *FindParameters* mit den veränderten Parametern erneut auf die Originallinie_DP angewendet. Ergaben sich sichtbare Abweichungen, konnten sie aus den Ergebnisdateien des Python-Skripts von Chrobak et al. (2016) mithilfe der Feature-ID identifiziert werden. Es wurde die Abweichung mit dem kürzesten zugehörige Skelettpfad ermittelt, der neue Wert für epsilon auf den nächst niedrigeren Integer-Wert unter dieser Skelettpfadlänge gesetzt und gamma wieder angepasst, um das Produkt epsilon × gamma konstant zu halten.

Das Modell *FindParameters* wurde zunächst nur mit der Methode *left* angewendet. Wenn mit dieser Methode Werte für epsilon und gamma gefunden worden waren, die zu keinen sichtbaren Abweichungen führten, wurde das Modell mit der Methode *right* und diesen Werten für epsilon und gamma angewendet. Ergaben sich dabei sichtbare Abweichungen, wurden epsilon und gamma nach der gleichen Vorgehensweise erneut verändert, bis es auch mit der Methode *right* keine Abweichungen ergab. Allerdings mussten diese Parameterwerte dann nochmals mit der Methode *left* überprüft werden. Der Prozess wurde so lange fortgesetzt, bis sich mit denselben Parameterwerten weder bei der Methode *left* noch bei *right* sichtbare Abweichungen mehr ergaben. Der Grund dafür war, dass die Bilanz aus Flächenge-

winn und Flächenverlust als Folge der Linienvereinfachung ermittelt werden sollte. Dafür war es notwendig, dass die Linie mit beiden Methoden und den gleichen Parameterwerten für epsilon und gamma vereinfacht wurde.

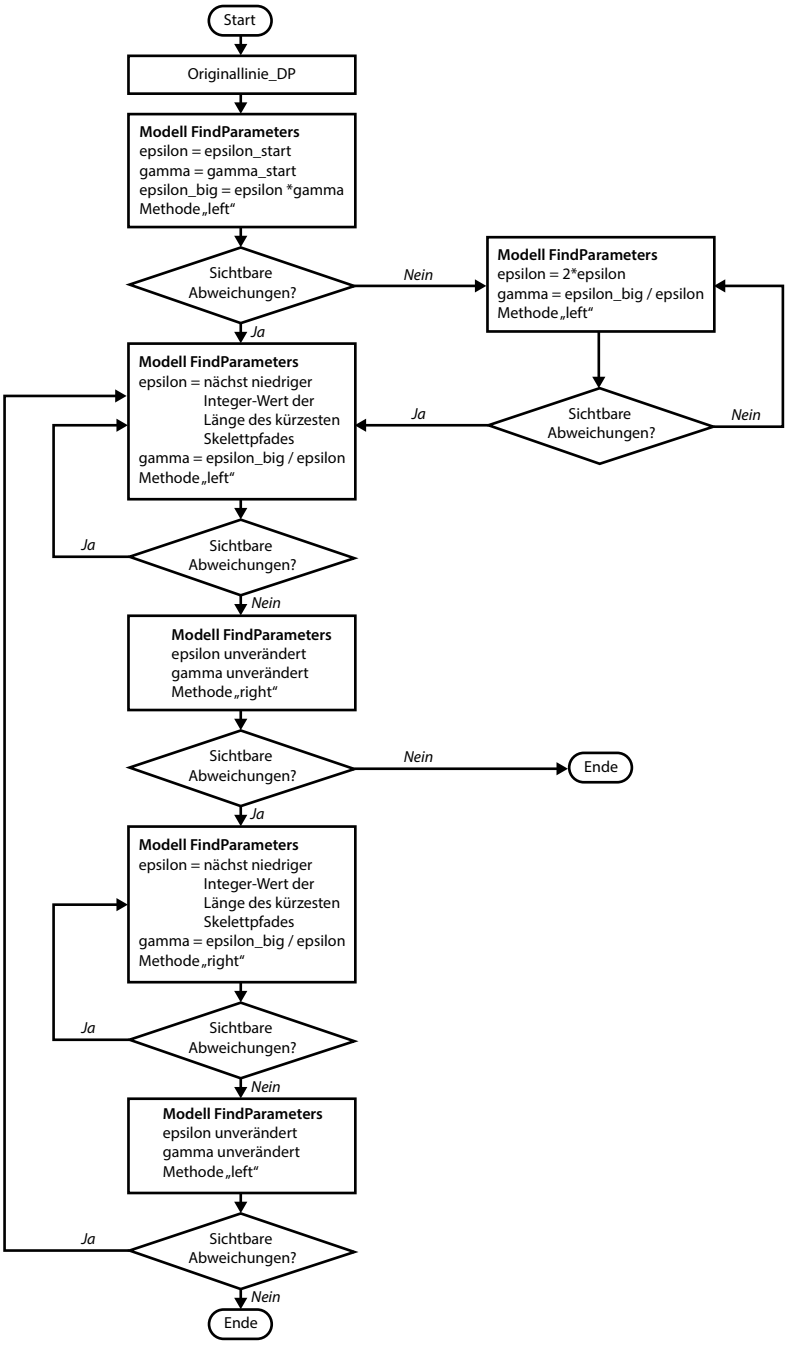


Abb. 21: Schema zur Auffindung eines geeigneten Wertes für den Parameter epsilon.

Während für den Anfangswert von epsilon_big und damit implizit für die Festsetzung des Anfangswertes für gamma die Originallinie visuell auf erhaltenswerte bends für den ersten Zielmaßstab 1:100k untersucht worden war, wurde für die weiteren Zielmaßstäbe epsilon_big, also die Schwelle der Länge des längsten Skelettpfades, nicht visuell bestimmt, sondern mit dem Änderungsfaktor zwischen dem vorherigen und dem neuen Zielmaßstab multipliziert.

Dies ist zugegebenermaßen nur begrenzt sinnvoll, da die Wahl von epsilon_big sehr stark von der Liniengestalt abhängt, siehe Abb. 22. Die konkreten Ergebnisse aller Anwendungen des Modells *FindParameters* bis zu den endgültigen Parametern sind in Tabelle 4 aufgelistet.

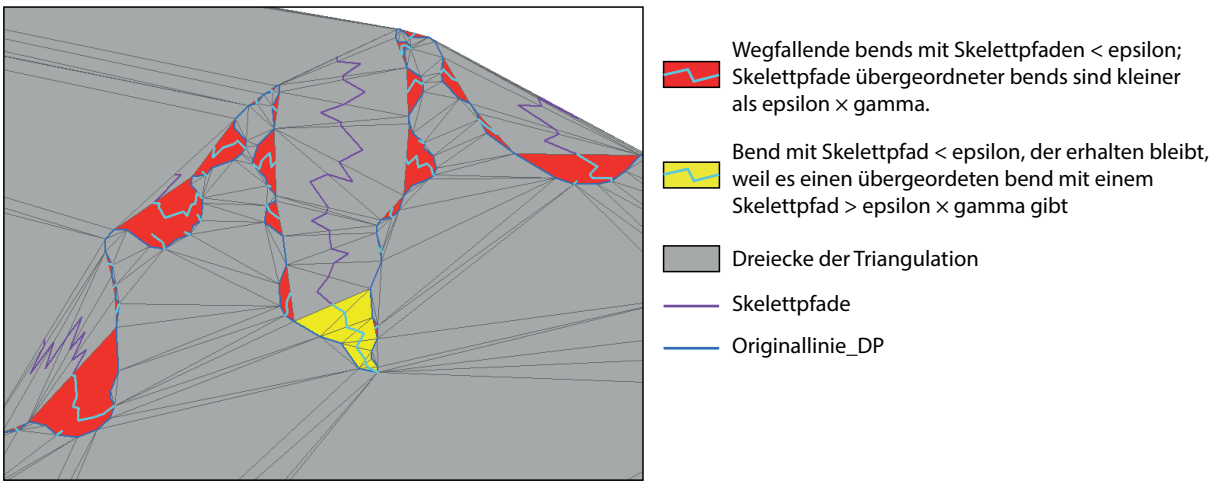


Abb. 22: Auswirkung des Parameters gamma

4.2.3 Umgehung von gamma

Eine automatisierte Bestimmung von gamma für gute, die Liniengestalt wahrende Ergebnisse erscheint problematisch, ebenso die Bestimmung nach dem visuellen Eindruck, da der Anwender normalerweise keinen Zugriff auf die Skelettpfade hat. Er kann sie weder betrachten, noch ihre Länge ablesen.

Im folgenden wird eine Möglichkeit skizziert, wie charakteristische Formen sozusagen unter Umgehung des Parameters gamma identifiziert werden können. Epsilon bestimmt sich wie beschrieben anhand der Nicht-Sichtbarkeit der Abweichungen zwischen Originallinie_DP und vereinfachter Linie. Skelettpfade, die sich überlagern, weil sie zu verschachtelten bends gehören, sind anhand des gleichen bottom points (Attribut *BOPO*) einfach zu ermitteln. Diejenigen bends, deren Skelettpfadlänge kleiner als epsilon ist, die jedoch Bestandteil eines übergeordneten bends mit einer Skelettpfadlänge größer als epsilon sind, werden anhand der Skelettpfadattribute *BOPO* und *Length* abgefragt und bleiben erhalten. Dies entspricht im Grunde genommen einem gamma-Wert von 1, weil epsilon = epsilon_big gesetzt wird. Bei einem gamma-Wert von 1 werden jedoch gemäß Ai et al. (2016) im Skript *EnvelopeZones.py* alle bends mit Skelettpfadlängen < epsilon eliminiert. Es wird nicht nach charakteristischen Punkten gesucht.

Die beschriebene Variante wurde nicht mehr programmiert, jedoch für den Zielmaßstab 1:2,5 Mio. manuell ausgeführt. Der Vergleich mit dem automatisiert gewonnenen Ergebnis erweckt den Eindruck, als würde man mit dieser Vorgehensweise dem Zielmaßstab besser gerecht, weil mehr charakteristische Formen erhalten geblieben sind.

Tab. 4: Anwendung des Modells *FindParameters* mit verschiedenen Parametern. Die finalen Parameter sind farbig hinterlegt (gelb: Methode left, grün: Methode right).

Lfd. Nr.	Maßstabszahl des Zielmaßstabs	epsilon (Meter)	gamma	epsilon (Meter) *gamma	Methode	Sichtbare Abweichungen gemäß Skript vorhanden (ja = 1, nein = 0)	Sichtbare Abweichungen gemäß Artikel vorhanden (ja = 1, nein = 0)	Dauer (Min.)
1	100k	60	4,5	270	left	1	0	8
2	100k	44	6,136	270	left	0	0	9,383
3	100k	44	6,136	270	right	0	0	6,383
4	250k	150	4,5	675	left	1	0	10,217
5	250k	113	5,973	675	left	0	0	11,717
6	250k	113	5,973	675	right	1	1	9,15
7	250k	103	6,553	675	right	0	0	6,067
8	250k	103	6,553	675	left	0	0	5,433
9	1 Mio.	600	4,5	2700	left	1	0	11,05
10	1 Mio.	551	4,900	2700	left	0	0	7,85
11	1 Mio.	551	4,900	2700	right	0	0	8,3
12	2,5 Mio.	1500	4,5	6750	left	0	0	10,15
13	2,5 Mio.	3000	2,25	6750	left	1	0	8,267
14	2,5 Mio.	2890	2,336	6750	left	1	0	10,733
15	2,5 Mio.	2101	3,213	6750	left	0	0	9,067
16	2,5 Mio.	2101	3,213	6750	right	1	1	4,283
17	2,5 Mio.	1930	3,497	6750	right	0	0	3,917
18	2,5 Mio.	1930	3,497	6750	left	0	0	5,05
19	12,5 Mio.	7500	4,5	33750	left	0	0	9,217
20	12,5 Mio.	15000	2,25	33750	left	0	0	8,95
21	12,5 Mio.	30000	1,125	33750	left	1	0	9,333
22	12,5 Mio.	17110	1,973	33750	left	0	0	6,317
23	12,5 Mio.	17110	1,973	33750	right	1	1	3,217
24	12,5 Mio.	15087	2,237	33750	right	0	0	4,433
25	12,5 Mio.	15087	2,237	33750	left	0	0	7,217

4.2.4 Methode *area preserving simple*

Da die Programmierung der Methode *area preserving* (Ai et al. 2016, S. 307) unter Verwendung des Parameters *m* (siehe Kapitel 2.1) nicht abgeschlossen werden konnte, wurde sie in abgewandelter Form manuell als Methode *area preserving simple* ausgeführt, indem für jede envelope zone die Flächen rechts bzw. links der Originallinie, die durch die Linienvereinfachung wegfallen sollten, aufsummiert und verglichen wurden. Ergaben die Flächen rechts der Originallinie die größere Summe, blieben sie erhalten und für die vereinfachte Linie wurden die neuen Segmente links der Originallinie verwendet, andernfalls geschah es umgekehrt.

4.3 Untersuchung der manuellen Linienvereinfachung

Bei der Untersuchung der Linien, die aus den manuell erstellten Karten als Referenz für die manuelle Linienvereinfachung als Vektordaten digitalisiert worden waren, zeigten sich große Abweichungen, die zum einen der Tatsache zuzuschreiben sind, dass die Karten der verschiedenen Zielmaßstäbe weder in der gleichen Behörde entstanden sind, zum anderen vermuten lassen, dass auch die verwendeten Karten des IGN nicht von derselben Person vereinfacht worden sind oder zum dritten möglicherweise auf verschiedenen Grundlagen basieren. Küstenverlagerungen natürlicher Art sind in einem solchen Ausmaß in dem begrenzten Erscheinungszeitraum der Karten in den allermeisten Fällen unwahrscheinlich, zumal wenn heutige Karten eher dem Zustand der Originallinie ähneln, deren Kartenvorlagen als älteste in den Jahren 1952 und 1957 erschienen sind. Allenfalls schmale, weit ins Meer reichende Molen, die in verschiedenen Jahren unterschiedliche Ausbauzustände aufweisen können, kommen in Einzelfällen als Erklärung in Frage. Darüber hinaus beschränkte sich die Bearbeitung der Küstenlinie für die Zielmaßstäbe in den Kartenvorlagen nicht auf die Vereinfachung der Linie, sondern die Küstenlinie muss notwendigerweise generalisiert und damit auch durch Vorgänge der Verdrängung und Vergrößerung verändert worden sein. Anhang C2 und C3 zeigen die digitalisierten Linien der verschiedenen Zielmaßstäbe.

Unter diesen Umständen konnten die gewählten Referenzdaten nicht verwendet werden. Die Bearbeiterin hatte hier eine größere Übereinstimmung zwischen den Maßstäben erwartet.

Die Anwendung des Python-Skripts von Chrobak et al. (2016) hat außerdem gezeigt, dass die vereinfachten Linien eine weitere Voraussetzung erfüllen müssen: die vereinfachten Abschnitte der Linie müssen gerade sein, d.h. sie dürfen nur aus zwei Punkten, nämlich den Schnittpunkten mit der Originallinie bestehen. Diese Bedingung ist beim Vergleich einer Originallinie mit einer manuell vereinfachten Linie in der Regel nicht gegeben (siehe Abb. 23).

Eine Recherche bei scopus ergab, dass der Artikel von Chrobak et al. (2016) bis zum Zeitpunkt des 25. März 2019 fünf Mal zitiert wurde. Die zitierenden Artikel beschreiben jedoch keine Ausdehnung des Verfahrens auf Fälle, in denen die vereinfachten Abschnitte aus mehr als zwei Punkten bestehen.

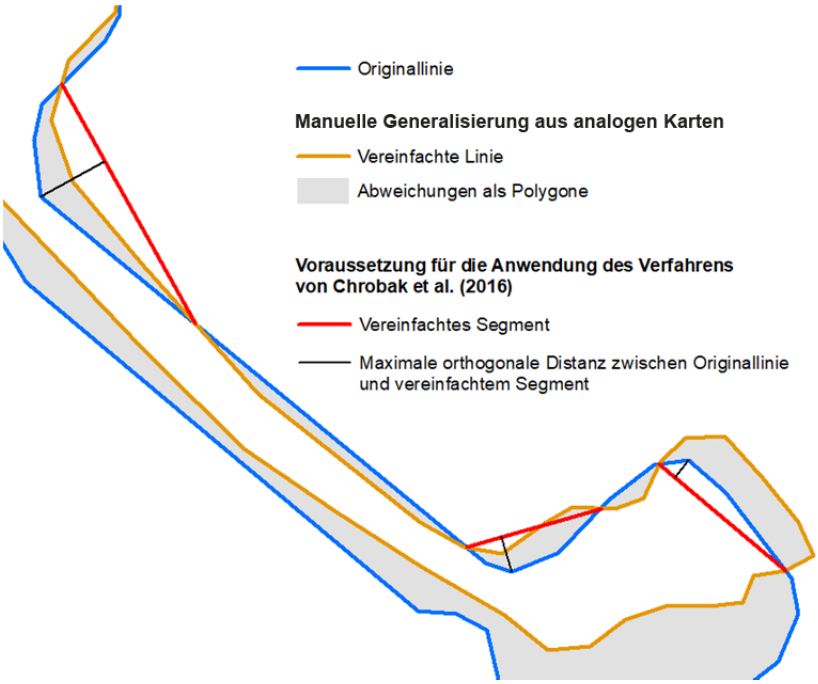


Abb. 23: Beispiel für Segmente der manuell vereinfachten Linie aus einer Kartenvorlage. Sie besitzen weitere vertices zwischen Anfangs- und Endpunkt.

Die Bearbeiterin erstellte daher auf der Grundlage der Originallinie 1:50k selbst manuell die Linienvereinfachungen für die Reihe der Zielmaßstäbe. Um die Ergebnisse mit dem Python-Skript von Chrobak et al. (2016) evaluieren zu können, wurden dabei ausschließlich Stützpunkte der Originallinie verwendet. Anhang C4 und C5 zeigen die eigenhändig vereinfachten Linien der verschiedenen Zielmaßstäbe.

5. Ergebnisse

5.1 Ergebnisse der Linienvereinfachung

Die maßstabsgetreuen graphischen Ergebnisse der Linienvereinfachung sind im Anhang C4 - C11 zusammengestellt.

5.2 Ergebnisse der metrischen Evaluierung

Die Tabellen 5 – 10 bzw. die Abbildungen 33 – 37 zeigen die Ergebnisse für die Metriken der sichtbaren Abweichungen, Kompressionsrate, p-Value, Flächenbilanz, Mittelwert und Standardabweichung der maximalen Distanzen sowie Berechnungszeit. Die Ergebnisse für das Verfahren nach Ai et al. (2016) wurden mit den Parametern erzielt, die in Tabelle 4 als final ausgewiesen sind, und für den Zielmaßstab 1:2,5 Mio. zusätzlich mit der Umgehung von gamma, wie in Kapitel 4.2.3 beschrieben. „Manuell CE“ meint die manuelle Vereinfachung durch die Bearbeiterin.

Tab. 5: Sichtbare Abweichungen nach Chrobak et al. (2016).
Die Gesamtzahl der Abweichungen ist in Klammern angegeben.

Zielmaßstab	1:100k	1:250k	1:1 Mio.	1:2,5 Mio.	1:12,5 Mio.
Manuell CE	3 (332)	12 (465)	5 (231)	2 (80)	0 (59)
Ai et al. (2016)					
Methode left	0 (71)	0 (130)	0 (160)	0 (106)	0 (52)
Methode left, Umgehung von gamma				0 (141)	
Methode right	0 (83)	0 (149)	0 (195)	0 (122)	0 (17)
Methode area preserving simple	0 (110)	0 (186)	0 (203)	0 (123)	0 (52)

Tab. 6: Kompressionsrate

Zielmaßstab	1:100k	1:250k	1:1 Mio.	1:2,5 Mio.	1:12,5 Mio.
Manuell CE					
Anzahl der Stützpunkte	1290	391	100	27	18
Kompressionsrate	33,47%	79,83%	94,84%	98,61%	99,07%
Ai et al. (2016)					
Methode left	47,60%	52,40%	69,83%	79,53%	92,16%
Methode left, Umgehung von gamma				73,6%	
Methode right	48,32%	53,69%	74,06%	86,38%	98,40%
Methode area preserving simple	49,56%	55,70%	72,10%	80,09%	92,16%

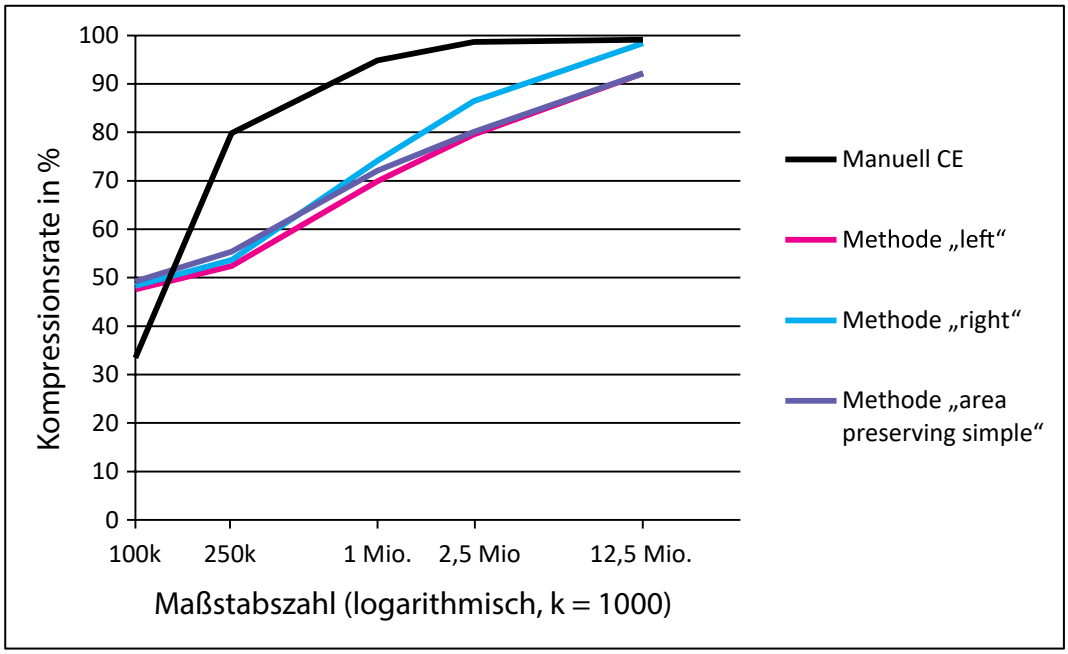


Abb. 24: Kompressionsrate

Tab. 7: p-Value.
Für die Berechnung des p-Value wurden als vorige Werte immer der Maßstab 1:50k und die Punktzahl in 1:50k verwendet.

Zielmaßstab	1:100k	1:250k	1:1 Mio.	1:2,5 Mio.	1:12,5 Mio.
Manuell CE	0,588	0,995	0,990	1,093	0,848
Ai et al. (2016)					
Methode left	0,932	0,461	0,400	0,405	0,461
Methode left, Umgehung von gamma				0,340	
Methode right	0,952	0,478	0,450	0,510	0,749
Methode area preserving simple	0,987	0,506	0,426	0,413	0,461

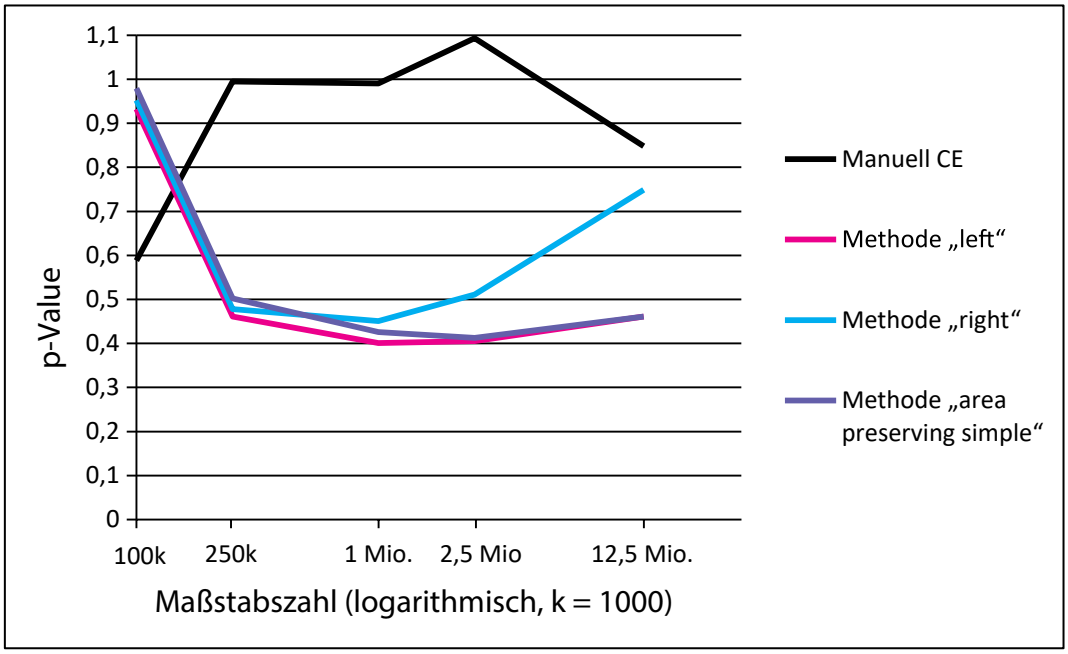


Abb. 25: p-Value.
Der Idealwert für Polylinien ist 1.

Tab. 8: Flächenbilanz in Hektar.
Positive Werte bedeuten einen Zugewinn an Landfläche, negative Werte bedeuten einen Landverlust.

Zielmaßstab	1:100k	1:250k	1:1 Mio.	1:2,5 Mio.	1:12,5 Mio.
Manuell CE	-5,91	-38,54	161,98	1143,43	227,63
Ai et al. (2016)					
Methode left	11,62	41,84	358,4	980,89	2178,69
Methode left, Umgehung von gamma				541,11	
Methode right	-16,31	-50,55	-397,41	-1342,23	-8339,33
Methode area preserving simple	-4,63	-10,23	-201,83	-430,56	2178,69

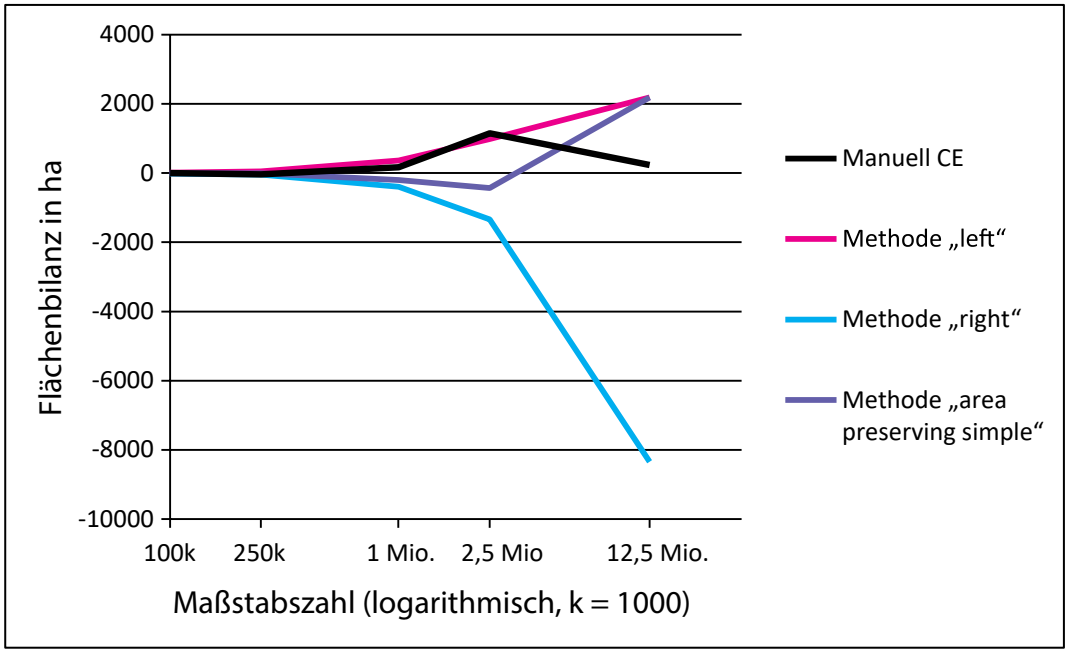


Abb. 26: Flächenbilanz.
Werte nahe 0 sind die besten.

Tab. 9: Mittelwert und Standardabweichung der maximalen orthogonalen Distanzen in Metern. Der obere Wert einer Zelle ist der Mittelwert, der untere die Standardabweichung. Die Schwelle für sichtbare Abweichungen nach Song und Miao (2016) wurde auf Druckerzeugnisse übertragen (siehe Kapitel 1.3.2 und 3.1) und mit 0,1 mm im Zielmaßstab angenommen. Die Werte schließen die Strichbreite von 0,1 mm, umgerechnet auf den Realwert im jeweiligen Zielmaßstab, mit ein.

Zielmaßstab	1:100k	1:250k	1:1 Mio.	1:2,5 Mio.	1:12,5 Mio.
Schwelle für sichtbare Abweichungen nach Song und Miao (2016), bezogen auf den Mittelwert	10	25	100	250	1250
Manuell CE	21,86 13,70	61,03 92,39	186,75 198,02	585,23 701,29	1738,89 1013,10
Ai et al. (2016)					
Methode left	31,68 8,21	56,47 15,91	189,19 86,87	443,35 253,40	1783,47 1275,79
Methode left, Umgehung von gamma				340,59 179,26	
Methode right	32,19 7,93	55,78 16,15	188,17 92,74	443,45 262,77	2748,57 1565,12
Methode area preserving simple	31,13 8,17	53,64 14,33	180,46 90,15	414,77 262,34	1783,47 1275,79

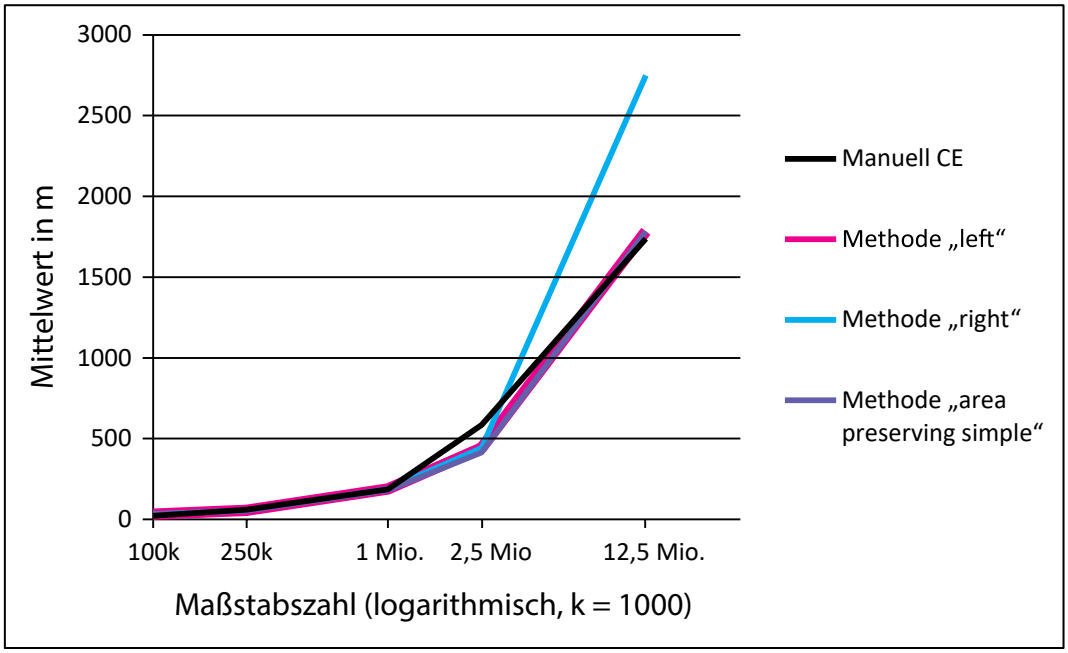


Abb. 27: Mittelwert der maximalen orthogonalen Distanzen. Kleine Werte sind besser.

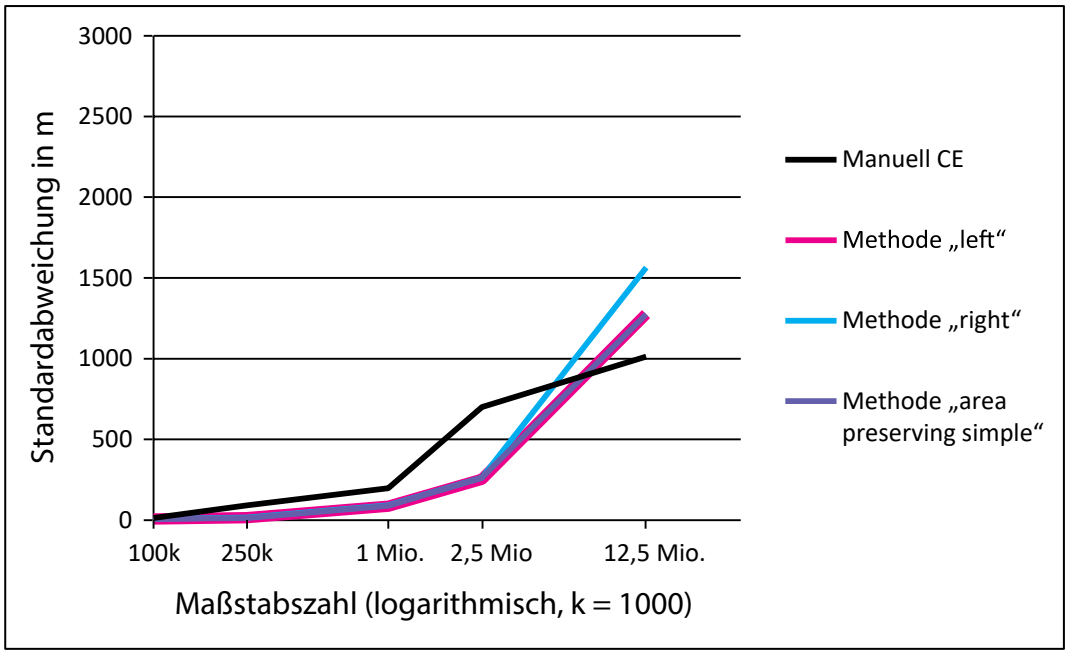


Abb. 28: Standardabweichung der maximalen orthogonalen Distanzen. Kleine Werte sind besser.

Tab. 10: Bearbeitungszeiten in Minuten. Erläuterungen im Text.

Zielmaßstab	1:100k	1:250k	1:1 Mio.	1:2,5 Mio.	1:12,5 Mio.
Manuell CE	360				
Ai et al. (2016)					
Vorverarbeitung mit ArcGIS-Tool Simplify Line (Point Remove)	Nur einmal notwendig vor dem Aufbau der bend-Hierarchie: 0,0077				
Modell *Preparing Steps*	Nur einmal notwendig vor der ersten Linienvereinfachung: 187,7				
Modell *FindParameters*	23,8	42,6	27,2	51,5	48,7
Methode left	9,0	5,3	7,5	4,9	7,0
Methode left, Umgehung von gamma	manuell				
Methode right	6,2	6,0	8,2	3,8	4,2
Methode area preserving simple	manuell				

Die Zeile Modell *FindParameters* in Tabelle 10 enthält für jeden Zielmaßstab die Summe der Berechnungszeiten aller Durchläufe, einschließlich des Durchlaufs mit den endgültigen Parametern und der abschließenden Prüfung auf sichtbare Abweichungen nach Chrobak et al. (2016).

Die Zeilen Methode *left* und Methode *right* geben die reine Berechnungszeit mit den endgültigen Parametern an, ohne die abschließende Prüfung auf sichtbare Abweichungen.

Zu Tabelle 10 ist folgende Anmerkung notwendig: Durch die Triangulation werden der Originallinie Stützpunkte hinzugefügt, die gemäß Ai et al. (2016) am Ende aus der vereinfachten Linie wieder entfernt werden müssen. Dies geschah zunächst mit dem ArcGIS-Tool *Intersect*, mit dem die Schnittmenge aus den Stützpunkten der Originallinie und der vereinfachten Linie gebildet und daraus die Linie mit ihrer endgültigen Stützpunktzahl generiert wurde. Auf dieses Vorgehen beziehen sich die Berechnungszeiten in dieser Tabelle.

Nach Abschluss sämtlicher rechnergestützter Vereinfachungen wurde der Bearbeiterin klar, dass die hinzugefügten Punkte durchaus für die Form der vereinfachten Linie relevant sein können, wenn in ihnen eine Richtungsänderung der Linie stattfindet. Durch die Bildung der Schnittmenge aus Stützpunkten der Originallinie und der vereinfachten Linie gehen jedoch alle hinzugefügten Punkte verloren. Deshalb wurde das ArcGIS-Tool *Intersect* an dieser Stelle durch das selbst geschriebene Python-Skript-Tool *RemoveAddedPoints* ersetzt, das diesem Problem Rechnung trägt. Von den vereinfachten Linien war jedoch nur der Zielmaßstab 1:1 Mio., jeweils mit Methode *left* und *right*, sowie der Zielmaßstab 1:2,5 Mio. mit Methode *right* betroffen. Alle Fehler, die sich daraus für die verwendeten Metriken ergaben, wurden korrigiert mit Ausnahme der Berechnungszeit, weil dies zum einen bedeutet hätte, den kompletten Prozess der Parameterfindung zu wiederholen, zum anderen der Algorithmus ohnehin nicht auf Verarbeitungsgeschwindigkeit optimiert ist.

Die manuelle Vereinfachung beanspruchte mit rund 360 Minuten weniger Zeit als die automatisierte Vereinfachung, für die die Zeiten in den Zeilen *Modell Preparing Steps* und *Modell FindParameters* aufsummiert 381,5 Minuten ergeben.

5.3 Interpretation und Diskussion

Der Interpretation und Diskussion der Ergebnisse seien folgende Aussagen von Forscherinnen und Forschern auf dem Gebiet der Linienvereinfachung vorangestellt:

„Mathematical measures, such as those proposed by McMaster (1987b), for objective evaluation of an algorithm's performance are useful when the output of filtering should be very similar to the source; i.e. when the primary aim is point reduction and when the database is designed to support not just visual mapping but also the calculation of statistics, such as distances and areas of polygons." (Visvalingam und Whelan 2016, S. 256)

„They [Mackaness und Ruas 2007, Anmerkung der Bearbeiterin] stated that an adequate evaluation framework should be able to handle the notion that the final output is a compromise between a set of sometimes competing map objectives. Such a framework should combine human evaluation and machine evaluation to meet the complexity of evaluation;" (Stoter 2010, S. 28)

Diese beiden Bemerkungen sollen helfen, der metrischen wie der visuellen Evaluierung im folgenden die rechte Bedeutung beizumessen.

5.3.1 Metrische Evaluierung

Bei Kompressionsrate und p-Value fallen drei Dinge auf:

1. Bei der Linienvereinfachung mit dem Algorithmus von Ai et al. (2016), im folgenden Ai genannt, sind die Werte beim Zielmaßstab 1:100k fast gleich. Sie liegen deutlich höher und sind damit besser als bei der manuellen Vereinfachung, im folgenden MV genannt.

2. Ab dem Zielmaßstab 1:250k fallen die Werte bei der manuellen Vereinfachung besser aus als in den drei Varianten von Ai.

3. Der p-Value der Ai-Methode *right* steigt mit abnehmendem Zielmaßstab deutlich höher als bei den Methoden *left* und *area preserving simple*.

Die erste Beobachtung ist damit zu erklären, dass es für die Anwendung von Ai notwendig war, die Originallinie zunächst mit dem ArcGIS-Tool *Point Remove* auszudünnen. Allein mit diesem Tool reduzierte sich die Anzahl der Punkte von 1939 auf 1093, also fast um die Hälfte. Ein solcher Eingriff ist bei MV nicht notwendig, so dass hier die Kompressionsrate niedriger ausfällt.

Die zweite Beobachtung ist auf unterschiedliche Vereinfachungsgrade zurückzuführen. Er ist bei MV in allen Zielmaßstäben höher als bei Ai, weil die Bearbeiterin mit ihrer manuellen Vereinfachung ein klareres Bild und damit eine bessere Lesbarkeit angestrebt hat. Das betrifft insbesondere den Zielmaßstab 1:250k, was sich auch in der höchsten Anzahl an sichtbaren Abweichungen und visuell im Anhang C4 und C5 widerspiegelt. Die Parameter für die Vereinfachung mit Ai hingegen wurden mithilfe des Verfahrens von Chrobak et al. (2016) ermittelt. Nach Ansicht der Autoren sind sich die Originallinie und die vereinfachte Linie umso ähnlicher, je geringer die Anzahl der sichtbaren Abweichungen ist. Dies kann auch dahingehend interpretiert werden, dass die Linie dadurch im Zielmaßstab vor einer zu starken Vereinfachung geschützt wird und mögliche Formverzerrungen unsichtbar bleiben. Um ein optimales Ergebnis in diesem Sinne zu erhalten, wurden die Parameter so gewählt, dass einerseits durch den Parameter epsilon die Anzahl der sichtbaren Abweichungen null war, andererseits durch den Parameter gamma charakteristische Endpunkte von Meeresarmen erhalten blieben. Das Ergebnis ist ein geringerer Vereinfachungsgrad als bei MV.

Die Ursache für die dritte Beobachtung ist hauptsächlich im Verlauf der Testlinie begründet. Bei Methode *left* liegen alle vereinfachten Linienabschnitte links von der Originallinie – im vorliegenden Testfall also meerseitig. Die tief eingeschnittenen Meeresarme besitzen lange Skelettpfade sowie kaum Verzweigungen mit kürzeren Skelettpfaden, die wegfallen könnten. Dadurch und infolge des gamma-Wertes blieben sie mit einem Großteil ihrer Stützpunkte bis zum Maßstab 1:2,5 Mio. vor der Eliminierung bewahrt. Der längste Meeresarm blieb auch im Zielmaßstab 1:12,5 Mio. erhalten und mit ihm eine große Anzahl von Stützpunkten. Bei der Methode *right* wurden bereits im Zielmaßstab 1:2,5 Mio. die Meeresarme landseitig sehr stark geglättet, wodurch viele Stützpunkte wegfielen. Im Zielmaßstab 1:12,5 Mio. war kein Skelettpfad länger als das Produkt aus den Parametern epsilon und gamma, so dass alle Landvorsprünge, die aufgrund des Parameters epsilon wegfallen sollten, tatsächlich eliminiert wurden. Dadurch ergaben sich in diesem Zielmaßstab wesentlich weniger Stützpunkte als bei der Methode *left* und in beiden beschriebenen Zielmaßstäben ein besserer p-Value. Die Werte für die Methode *area preserving simple* liegen nahe bei denjenigen der Methode *left*, weil insbesondere im Bereich der Meeresarme den links liegenden, meerseitigen vereinfachten Linienabschnitten der Vorzug gegeben wurde, da sie zu geringeren Flächenänderungen führen als die Methode *right* (siehe Anhang C10).

Die Kompressionsrate ist zwar für die Reduzierung der Datenmenge von Bedeutung und MV schneidet hier sehr gut ab. Inwieweit die verbliebene Stützpunktzahl aber in der Lage ist, die Linie im Zielmaßstab angemessen darzustellen, versucht der p-Value zu erfassen, der das Wurzelgesetz von Töpfer und Pillewizer (1966) berücksichtigt. Betrachtet man die Formel für die Berechnung des p-Value (Kapitel 3.2), dann wird deutlich, dass er den Idealwert 1 für Linienelemente in dem Fall erreicht, wenn das Verhältnis der Punktzahl in Ziel- und Originalmaßstab dem Verhältnis dieser Maßstäbe entspricht. Dies trifft zwar für MV nicht in allen Zielmaßstäben zu, in den meisten Fällen liegen die Werte jedoch deutlich näher an 1 als bei Ai.

Hinsichtlich der Flächenbilanz erfüllt die Methode *area preserving simple* ihren Zweck sehr gut. Erst im kleinsten Zielmaßstab 1:12,5 Mio. ist ein größerer Flächenzuwachs zu verzeichnen, nämlich um den gleichen Betrag wie bei Methode *left*. Dies ist damit zu erklären, dass die komplette Testlinie nur noch von einer einzigen envelope zone eingehüllt war, so dass sämtliche meerseitigen Linienabschnitte für die vereinfachte Linie verwendet wurden, weil dies zu einer besseren Flächenbilanz führte als mit den landseitigen. Das ist auch der Grund, weshalb die Methode *area preserving simple* bei allen Metriken im kleinsten Zielmaßstab den gleichen Wert aufweist wie Methode *left*. Die Methode *right* führt in 1:12,5 Mio. zu einem extremen Landverlust, was hier auch wieder in der Gestalt der Testlinie begründet ist. Wäre der Küstenabschnitt länger gewählt worden, so dass er eine konvexe Form in Richtung Meer angenommen hätte, hätten sich infolge einer größeren Triangulationsdomain auch für die Landseite längere Skelettpfade ergeben können, so dass möglicherweise Landvorsprünge erhalten geblieben wären. Bei MV liegt die Flächenbilanz in fast allen Zielmaßstäben am nächsten beim Idealwert 0. Im Zielmaßstab 1:1 Mio. sind die Landgewinne deutlich höher als die Landverluste, da MV stärker dahin tendierte, Meeresbuchten abzuschnüren als Landzungen.

Der Verlauf der Mittelwerte und Standardabweichungen der maximalen orthogonalen Distanzen ist bei den drei Methoden *left*, *right* und *area preserving simple* bis zum Zielmaßstab 1:2,5 Mio. praktisch gleich. Dies ist auf den Vereinfachungsparameter der Skelettpfadlänge zurückzuführen, dessen Wert für die drei Methoden jeweils der gleiche ist. Er beeinflusst die maximalen orthogonalen Distanzen, und zwar sowohl als Obergrenze epsilon, als auch als Untergrenze im Produkt epsilon × gamma, das charakteristische Endpunkte von Skelettpfaden mit größerer Länge vor der Eliminierung schützt. Wenn die Ergebnisse für die Mittelwerte bei den Methoden *left* und *right* fast gleich sind, müssen auch die Werte für die Methode *area preserving simple* ihnen gleichen, da *area preserving simple* eine Mischung aus den Methoden *left* und *right* darstellt. Der deutliche höhere Mittelwert und die größere Standardabweichung bei Methode *right* im kleinsten Zielmaßstab 1:12,5 Mio. haben die gleichen Ursachen wie die große Abweichung in der Flächenbilanz, nämlich die Lage von Anfangs- und Endpunkt der Testlinie, die daraus folgende Ausdehnung der Triangulationsdomain, die wiederum landseitig (rechts) zu kürzeren Skelettpfaden führte, die eliminiert wurden, während dagegen linksseitig einer der langen Meeresarme erhalten blieb.

Die Mittelwerte der maximalen orthogonalen Distanzen stimmen überdies mit den Werten der MV überein bzw. unterbieten diese sogar im Maßstab 1:2,5 Mio. Beim visuellen Vergleich werden die Ursachen deutlich (siehe Abb. 29): Ai-Methode *left* schnürt etliche breite Meeresbuchten aufgrund des epsilon-Parameters nur zur Hälfte ab, erhält jedoch aufgrund des gamma-Wertes die beiden längsten, sehr schmalen Meeresarme in voller Tiefe. Das senkt sowohl den Mittelwert als auch die Standardabweichung gegenüber MV, entspricht jedoch nicht kartographischer Praxis. Bei Ai-Methode *right* sorgt der Vereinfachungsparameter in

Verbindung mit der Gestalt der Originallinie insgesamt dafür, dass es viele bends mit gemäßigter bend depth sind, die hier entfallen und dadurch zu einem kleineren Mittelwert führen.

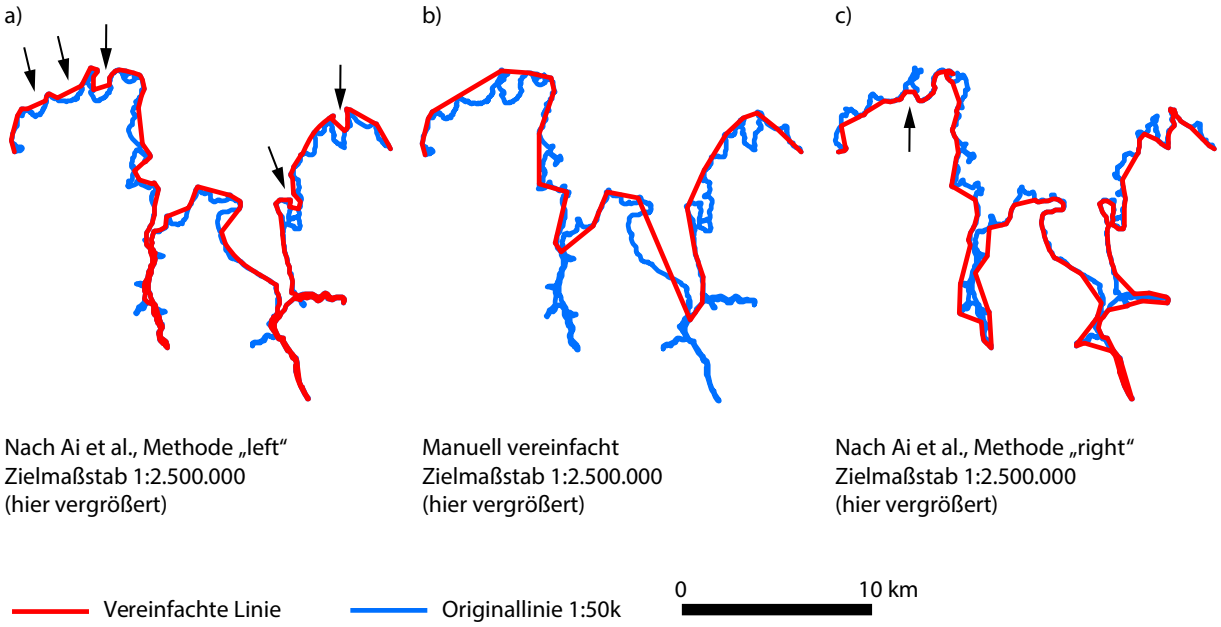


Abb. 29: Meeresbuchten im visuellen Vergleich.
a) Bei Methode *left* sind die Buchten etwa zur Hälfte abgeschnitten. b) Die manuelle Bearbeitung entfernt die Buchten komplett und hat den sichtbar gröberen Vereinfachungsgrad. c) Bei Methode *right* ist eine Landzunge nur teilweise abgeschnitten.

5.3.2 Visuelle Evaluierung

Qualität im Blick auf die Sichtbarkeit der Abweichungen

Generell kann festgestellt werden, dass es häufig nicht nachvollziehbar war, welche Abweichungen nach Chrobak et al. (2016) als sichtbar galten, während andere als nicht sichtbar eingestuft wurden. Das wurde zum einen während der Suche nach den geeigneten Parametern für den Algorithmus nach Ai et al. (2016) deutlich (siehe Abb. 30), aber auch bei der Analyse der verbliebenen sichtbaren Abweichungen bei der manuellen Linienvereinfachung (siehe Abb. 31).

Die Standardisierung im Verfahren von Chrobak et al. (2016) ist prinzipiell als fragwürdig anzusehen. Denn je länger das vereinfachende Segment ist, desto kleiner wird der Standardisierungsfaktor k, der als Quotient aus der in den Zielmaßstab umgerechneten Basislänge des Elementardreiecks und der Länge des vereinfachenden Segments berechnet wird. Desto größer darf jedoch auch die maximale orthogonale Distanz sein, da sie durch die Multiplikation mit dem Standardisierungsfaktor leichter unter der Höhe des Elementardreiecks bleibt als bei einem kurzen Segment. Umgekehrt werden Abweichungspolygone, deren Dreiecksbasis kürzer ist als die des Elementardreiecks im entsprechenden Zielmaßstab, nicht weiter berücksichtigt, unabhängig davon, wie groß die Höhe des Dreiecks, also die maximale orthogonale Distanz, ist (siehe Abb. 32). Sie können daher visuell sichtbar bleiben, auch wenn sie mit dem Verfahren von Chrobak et al. (2016) als nicht sichtbar eingestuft werden. Nach Auffassung der Bearbeiterin handelt es sich in diesen Fällen um coves, deren Verarbeitung mit dem Python-Skript-Tool von Chrobak et al (2016) unklar geblieben ist (siehe Kapitel 3.3.2).

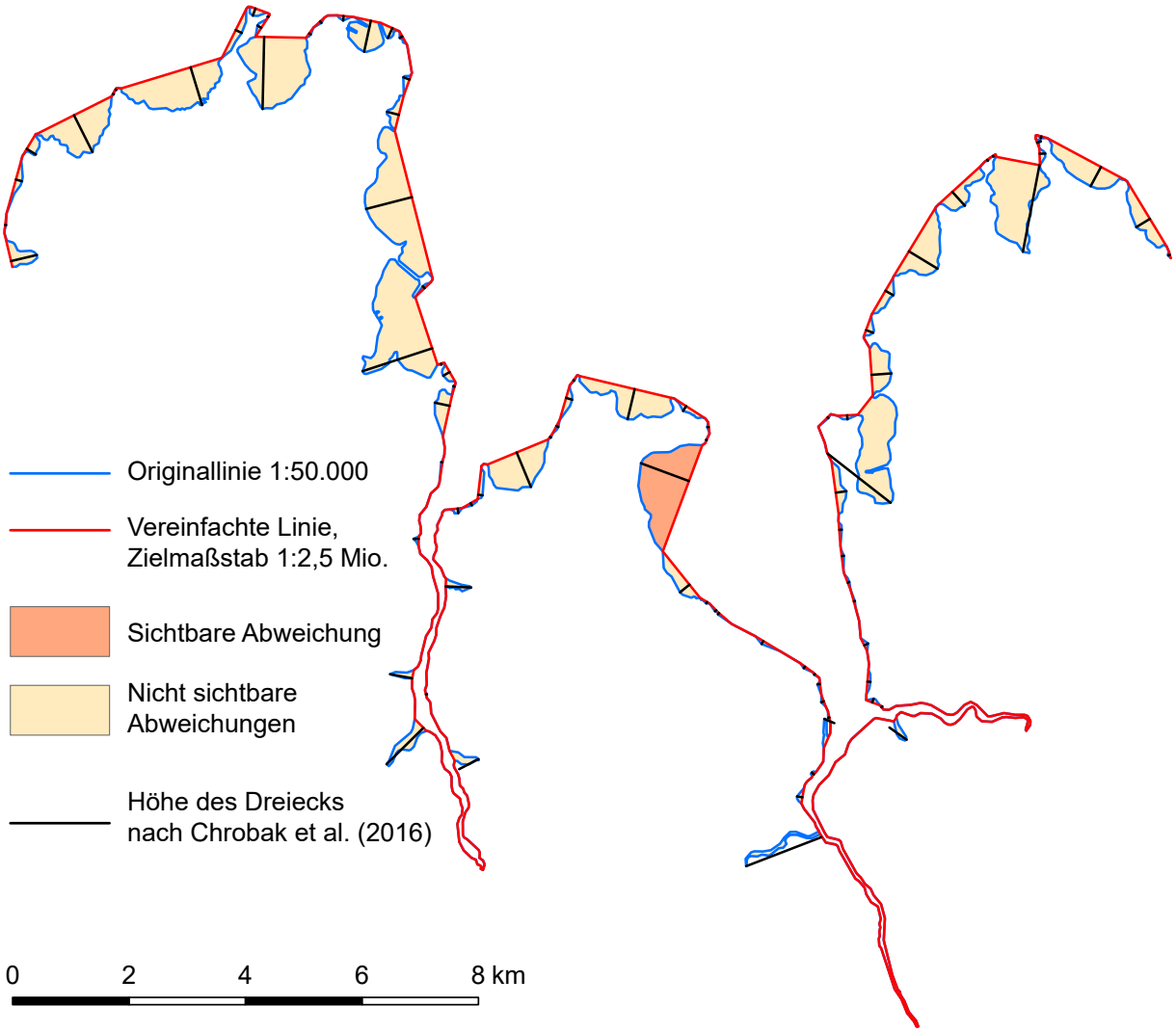


Abb. 30: Sichtbare und unsichtbare Abweichungen nach Chrobak et al. (2016) bei der automatisierten Vereinfachung nach Ai et al. (vergrößerte Darstellung).

Die Verwendung der maximalen orthogonalen Distanzen nach Song und Miao (2016) als Metrik für die Sichtbarkeit unter Einbeziehung der Strichstärke wie bei Chrobak et al. (2016) erscheint daher sinnvoller, vorausgesetzt man betrachtet nicht nur den Mittelwert für alle Abweichungen, sondern die individuellen Werte.

Im maßstabsgerechten Ausdruck der nach Ai et al. (2016) vereinfachten Linien in Kombination mit der Originallinie in den verschiedenen Zielmaßstäben (siehe Anhang C6, C8, C10) sind an einigen Stellen durchaus Abweichungen zu erkennen, die nach Chrobak et al. (2016) nicht sichtbar sein dürften. Aus Sicht der Bearbeiterin ist das jedoch nicht störend, sondern im Gegenteil eine angenehm empfundene Reduzierung der Liniendetails und eine Steigerung der Lesbarkeit, die dem Sinn und Zweck der Linienvereinfachung entspricht. Eine Ausnahme bildet die Entfernung des westlichen Meerestrichters im Maßstab 1:12,5 Mio. bei Methode *left*, während der östliche erhalten bleibt. Dieses Ergebnis ist visuell unbefriedigend.

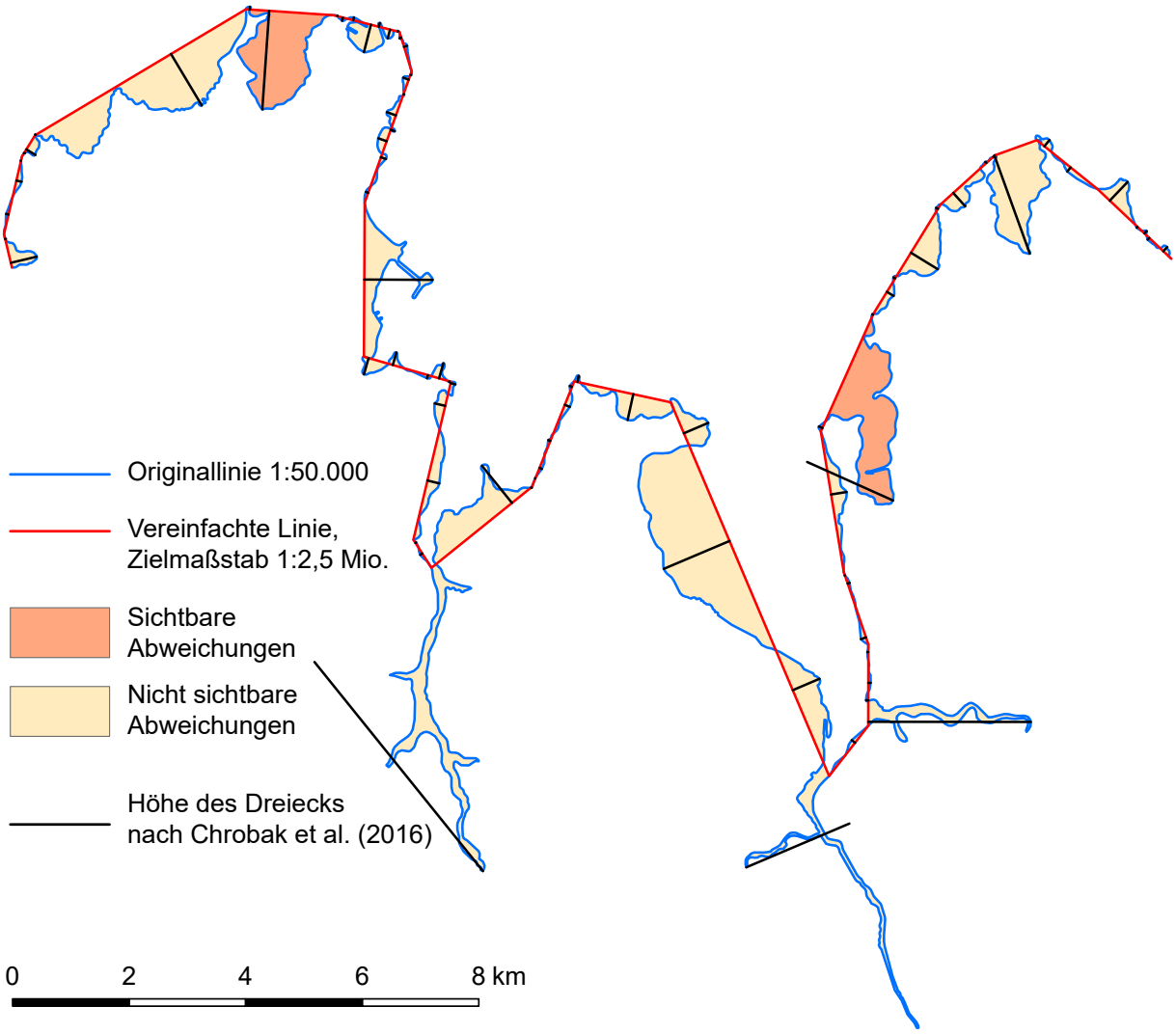


Abb. 31: Sichtbare und unsichtbare Abweichungen nach Chrobak et al. (2016) bei der manuellen Vereinfachung.

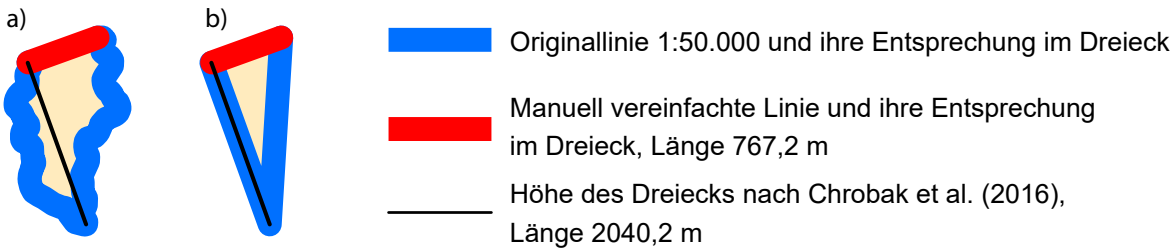


Abb. 32: Abweichungspolygone mit einem kurzen vereinfachten Linienabschnitt und hoher maximaler orthogonaler Distanz.
a) Polygon der Abweichung. b) Dreieck, das aus dem Abweichungspolygon abgeleitet wurde. Die Basis des Dreiecks ist mit 767,2 m kürzer als die Basis im spitzwinkligen Elementardreieck „kan“ im Maßstab 1:2,5 Mio. (1000 m); der Wert für die Höhe des Dreiecks – 2040,2 m – liegt deutlich über dem des Elementardreiecks (1250 m). Die Abbildung ist gegenüber dem Zielmaßstab 1:2,5 Mio. vergrößert. Dadurch ergeben sich große Strichbreiten, da diese bei Chrobak et al. (2016) berücksichtigt werden.

Qualität im Blick auf den Grad der Vereinfachung

Vereinfachung nach Ai et al. (2016)

Obwohl einige Abweichungen sichtbar sind, ist der Vereinfachungsgrad in den Zielmaßstäben ab 1:250k und kleiner zu schwach ausgeprägt, und dies bei allen drei Methoden, siehe beispielhaft Abb. 33.

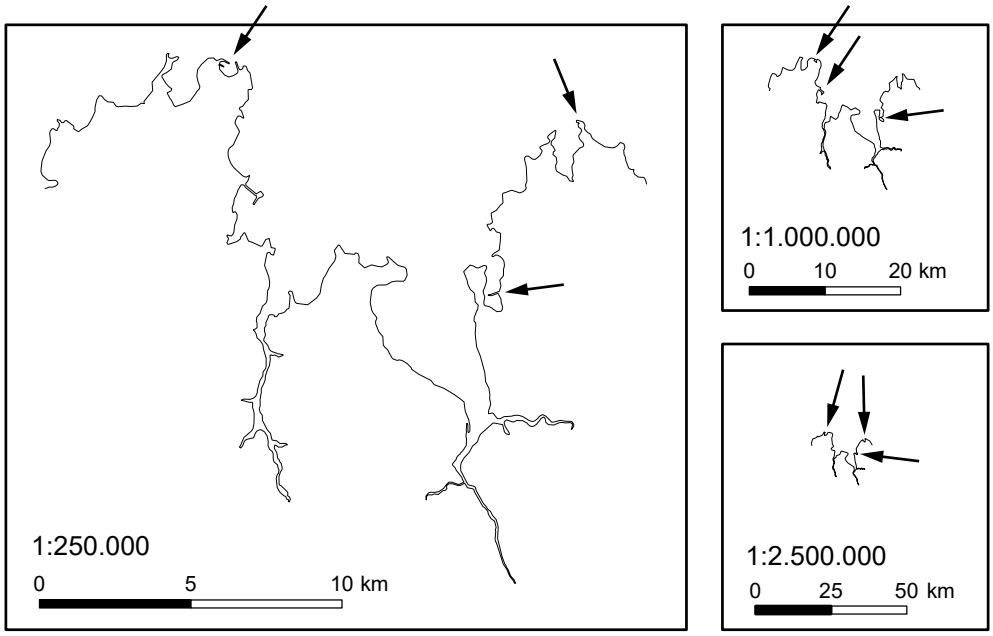


Abb. 33: Zu geringer Vereinfachungsgrad beim Algorithmus nach Ai et al. (2016), Methode *left*.

Die Methode *right* führt in Abb. 33 im Zielmaßstab 1:250k beim westlichen und zum Teil auch beim östlichen Meerestrichter zu einem besseren Ergebnis, weil diejenigen vereinfachenden Linienabschnitte verwendet werden, die landseitig liegen. Sie führen damit zu einer Verbreiterung des Trichters. Dies ist jedoch ein zufälliges Ergebnis, denn der am weitesten einschneidende Meeresarm des östlichen Trichters profitiert kaum von der Methode *right*. Als positiv ist die Tatsache anzusehen, dass bei der Methode *right* im Zielmaßstab 1:12,5 Mio. beide Meerestrichter andeutungsweise erhalten bleiben, während bei der Methode *left* der kürzere, westliche entfällt und der längere, östliche stehen bleibt (siehe Anhang C6 und C8).

Der Vereinfachungsgrad ist bei den Methoden *left* und *right* nach Ai et al. (2016) nicht konsistent. Viele, auch schwache Knicke der Originallinie bleiben stehen, wenn die vereinfachenden Segmente auf der rechten Seite der Originallinie liegen, weil bei Methode *left* nur die links liegenden Segmente zur Vereinfachung verwendet werden, und umgekehrt. Dies ist konzeptionell im Verfahren von Ai et al. (2016) begründet.

Bei der Methode *area preserving simple*, bei der sowohl von der rechten als auch von der linken Seite der Originallinie Segmente zur Vereinfachung herangezogen werden, erscheint der Vereinfachungsgrad einheitlicher.

Visuell unbefriedigend sind die engen Meeresarme, deren gegenüberliegende Ufer bereits im Maßstab 1:250k sich zu berühren beginnen, weil die Strichbreite nicht unter das Minimum

von 0,1 mm abgesenkt werden darf. Dabei muss man jedoch im Gedächtnis behalten, dass der Algorithmus nach Ai et al. (2016) sich auf den Generalisierungsvorgang der Linienvereinfachung beschränkt und nicht den Anspruch erhebt, eine Linie nach allen kartographischen Gesichtspunkten zu generalisieren.

Der Einsatz der Delaunay-Triangulation und der Aufbau der bend-Hierarchie bringt grundsätzlich das Problem mit sich, dass bestimmte Knicke in einem Kurvenabschnitt nie geglättet werden können, sondern nur mit dem kompletten Kurvenabschnitt wegfallen können, vgl. Abb. 34. Ein Kurvenabschnitt reicht immer von einer Ecke eines Verzweigungsdreiecks zur nächsten (siehe auch Abb. 3).

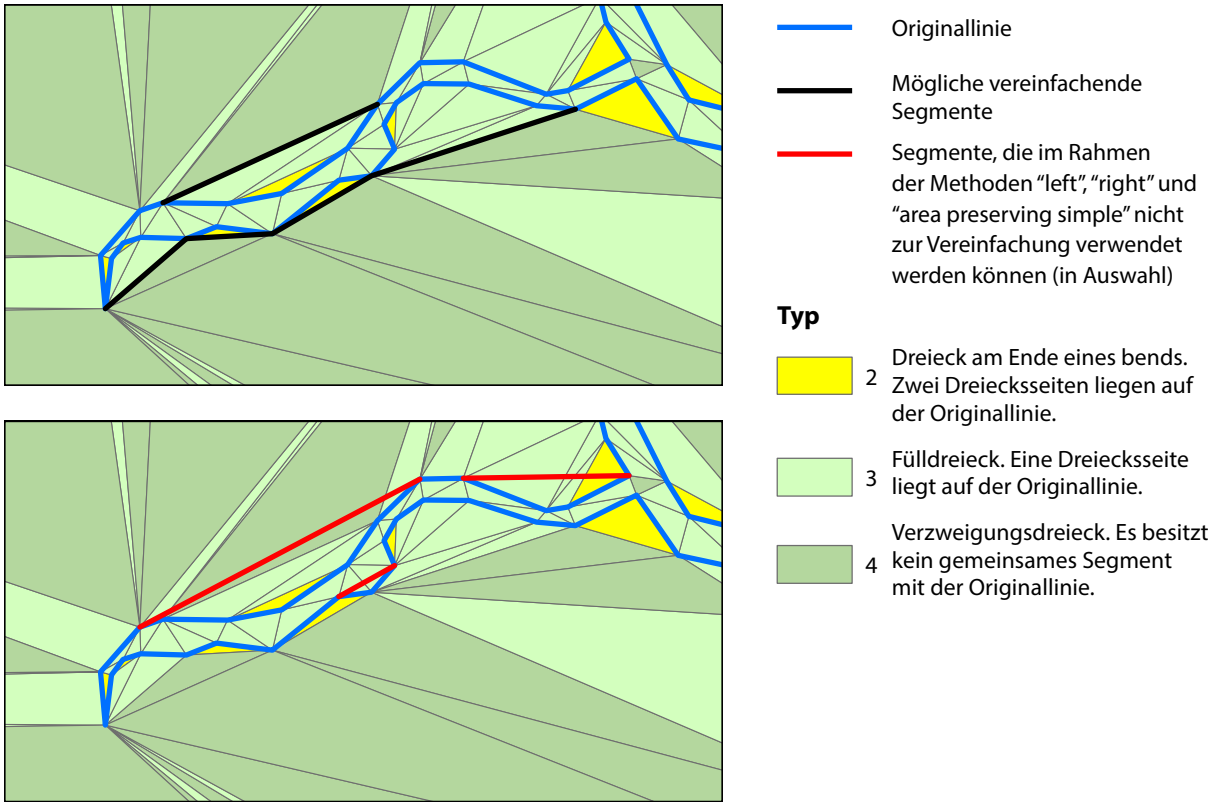


Abb. 34: Triangulation und bend-Struktur als Ursache für einen inkonsistenten Vereinfachungsgrad.

Abb. 35 verdeutlicht außerdem, dass durch dieses Konzept unter Umständen neue Formen geschaffen werden können, die für den Zielmaßstab zu klein sind.

Es darf nicht unerwähnt bleiben, dass Ai et al. (2016, S. 307) eine interaktive Kontrolle bei der Verwendung ihres Algorithmus in Betracht ziehen. Wie sich diese Interaktivität gestalten könnte oder müsste, ob sie sich auf das Austesten mehrerer relevanter Parameter oder auf die individuelle Beeinflussung einzelner Kurvenabschnitte bezieht, geht aus dem Artikel nicht hervor.

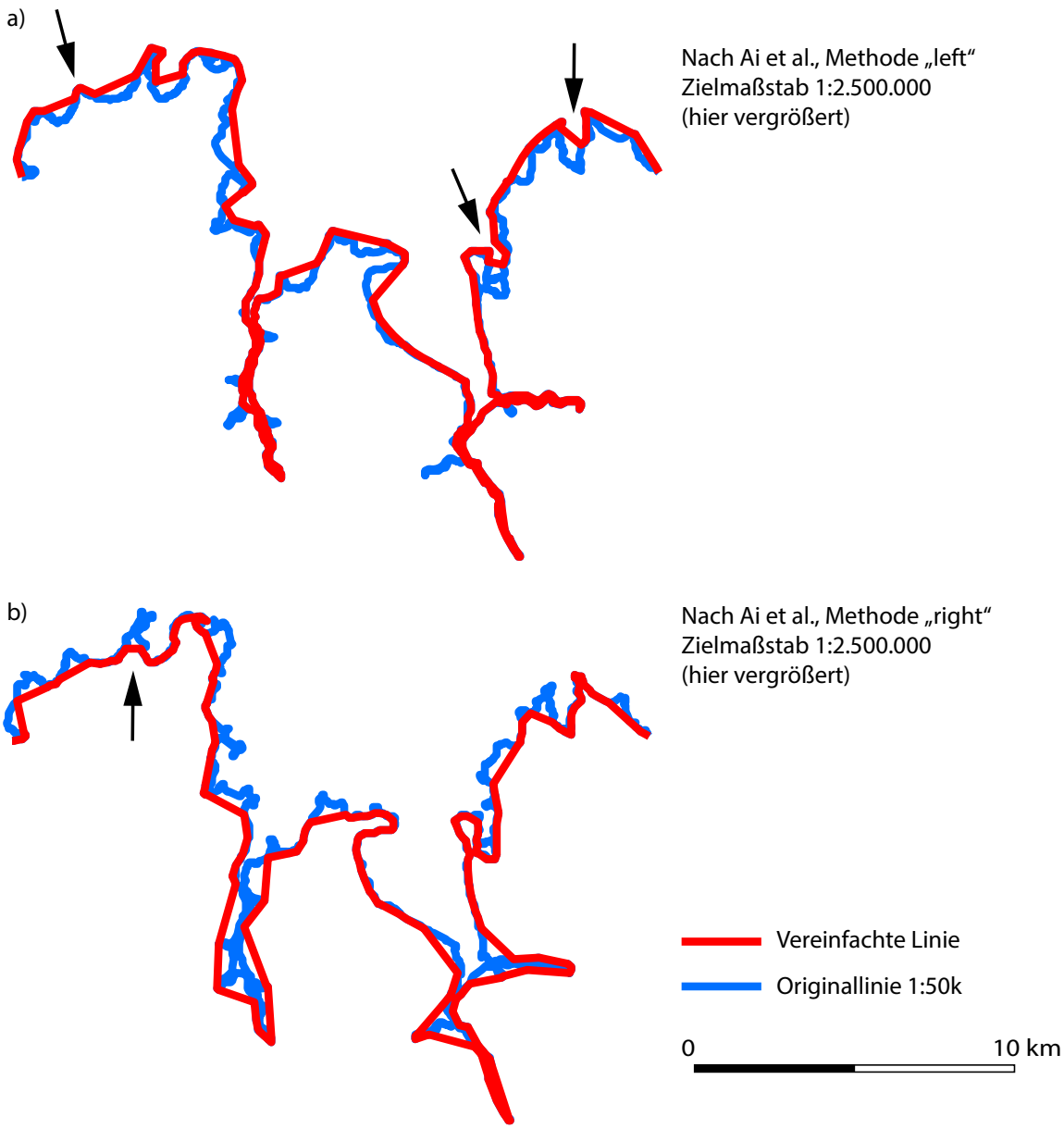


Abb. 35: Triangulation und bend-Struktur als Ursache für neue, zu kleine Formen.

Manuelle Vereinfachung

Der Vereinfachungsgrad erscheint in allen Zielmaßstäben insgesamt angemessen. Er ist gröber als bei der automatisierten Vereinfachung und ihr damit überlegen, denn das Ergebnis ist dadurch besser lesbar. An dieser Stelle sei noch einmal daran erinnert, dass der geringe Vereinfachungsgrad in der Automatisierung dadurch zustande kommt, dass die Parameter für den Algorithmus nach Ai et al. (2016) mithilfe des Verfahrens von Chrobak et al. (2016) ermittelt wurden, um sichtbare Abweichungen zu vermeiden.

Qualität im Blick auf die Wahrung der Formen

In den Zielmaßstäben 1:100k und 1:250k besitzen die Formen nach der Linienvereinfachung mit dem Algorithmus nach Ai et al. (2016) in den meisten Fällen die Qualität einer manuellen Vereinfachung. Abb. 36 zeigt einige Beispiele des Zielmaßstabs 1:100k, in denen eine Person, die die Linie manuell vereinfacht, nach kartographischen Gesichtspunkten anders hätte entscheiden müssen.

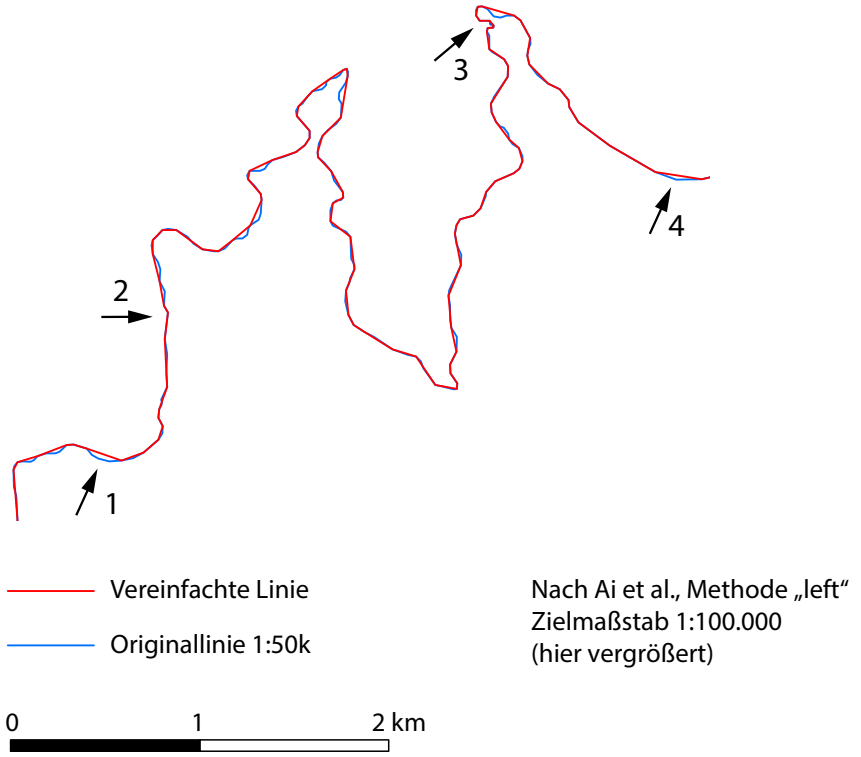


Abb. 36: Unzulänglichkeiten des Algorithmus nach Ai et al. (2016) im Zielmaßstab 1:100k. Situationen 1 und 4: Die Form der Bucht wäre besser unverändert übernommen worden. Situationen 2 und 3: Der Knick wäre im manuellen Verfahren weggefallen.

Abb. 37 zeigt beispielhaft die unterschiedlichen Ergebnisse von rechnergestütztem und manuellem Verfahren für den Zielmaßstab 1:1 Mio. Bei der Linienvereinfachung ist es ein grundsätzliches Anliegen der Kartographie, die Hauptrichtungen der Linie zu erhalten. In Abb. 37a, Situation 1, ist dies nicht gelungen. Die selbst erstellte manuelle Vereinfachung hingegen beachtet die Hauptrichtungen (Abb. 37b). Situation 2 in Abb. 37a wahrt zwar die Hauptrichtungen, lässt jedoch Formen stehen, die für den Zielmaßstab zu klein sind. Abb. 37b wird den Hauptrichtungen gerecht. Der Vereinfachungsgrad ist zwar deutlich stärker als bei Abb. 37a, was sich jedoch im Zielmaßstab eher positiv auf den visuellen Eindruck auswirkt. Situation 3 in Abb. 37a ist unbefriedigend, weil anstelle der Bucht mit breiter Basis eine Spitze entstanden ist. Die selbst erstellte manuelle Vereinfachung in Abb. 37b umgeht das Problem durch einen stärkeren Vereinfachungsgrad. Situation 3 in Abb. 37a hat ihre Ursache in der Konzeption des Algorithmus von Ai et al. (2016), wie Abb. 38 verdeutlicht.

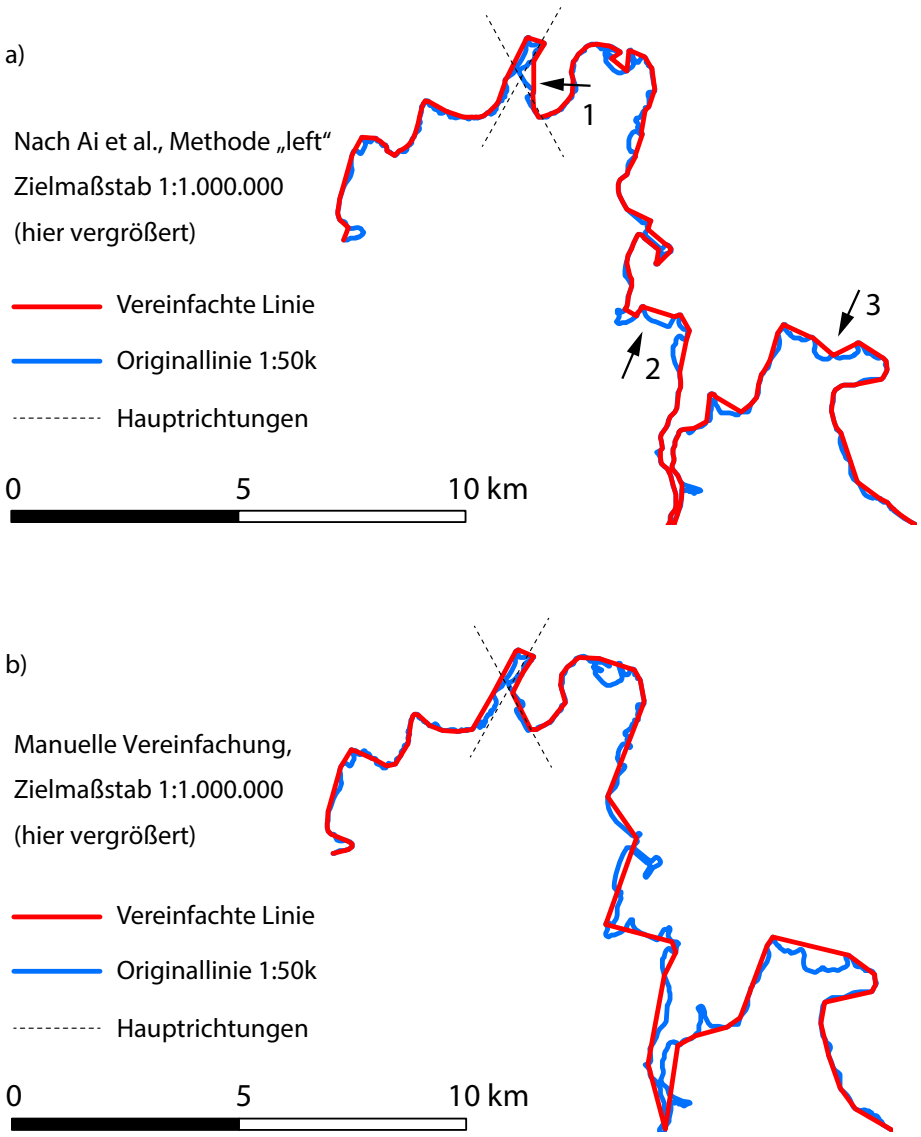


Abb. 37: Formveränderungen durch den Algorithmus von Ai et al. (2016), Methode *left*. a) Algorithmus nach Ai et al. (2016). b) Manuelle Vereinfachung. Erläuterungen im Text.

Abb. 39 zeigt kartographisch unerwünschte Formveränderungen im bei der Methode *area preserving simple*.

An dieser Stelle sei noch einmal auf Abb. 35 verwiesen, die verdeutlicht, dass durch die Verwendung der bend depth als Parameter für den Vereinfachungsgrad entgegen kartographischer Generalisierungspraxis Formen verkleinert werden können anstatt sie entweder durch Vergrößerung zu betonen oder ganz zu eliminieren. Wie auch immer der Schwellenwert für die bend depth festgesetzt wird – er kann für eine Situation perfekt und für eine andere nachteilig sein.

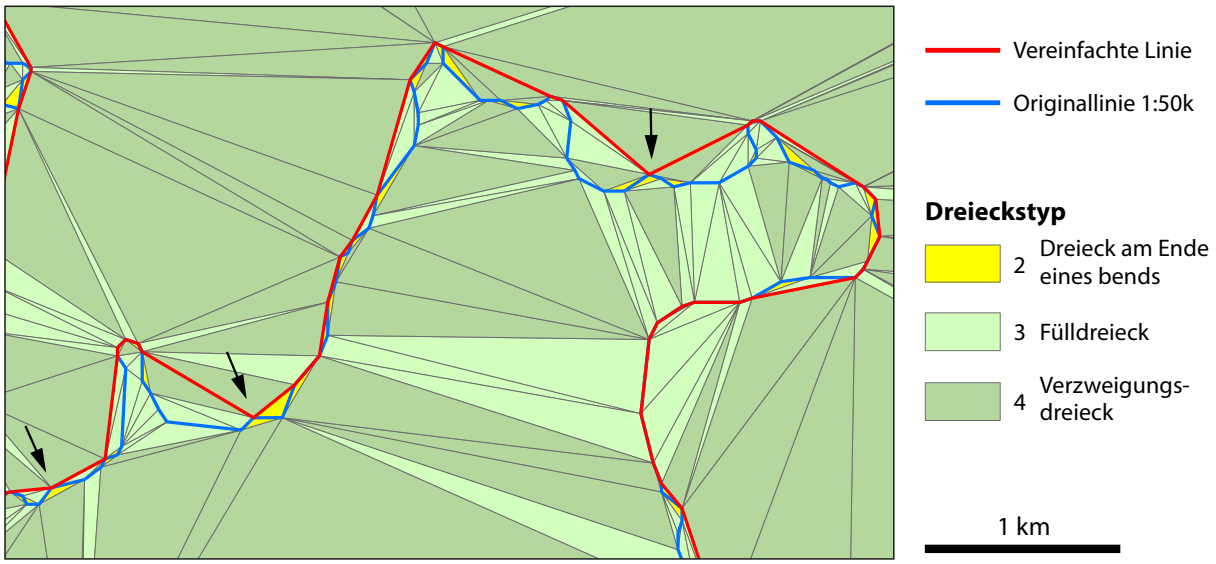


Abb. 38: Triangulation und bend-Struktur als Ursache für Formveränderungen. Vereinfachte Segmente können nur aus Seiten eines Verzweigungsdreiecks bestehen. Zielmaßstab 1:1 Mio., hier vergrößert. Algorithmus nach Ai et al. (2016), Methode *left*. Strichbreiten sind nicht maßstäblich.

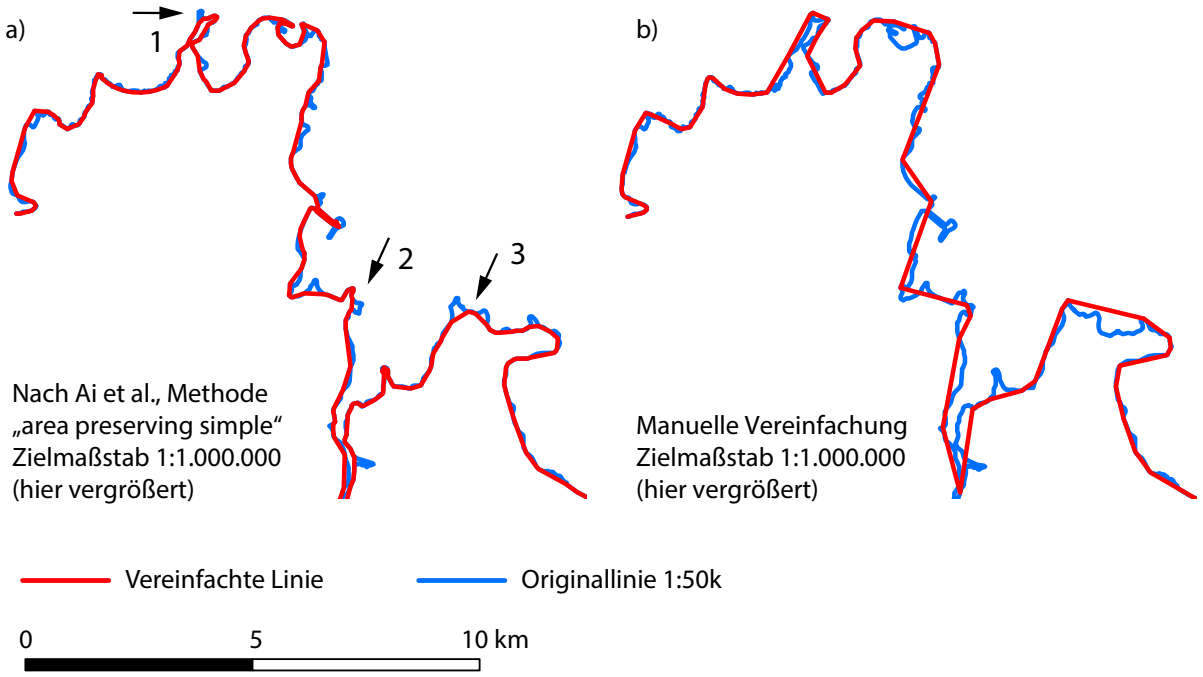


Abb. 39: Formveränderungen durch den Algorithmus von Ai et al. (2016), Methode *area preserving simple*.

6. Fazit und Ausblick

Weder der Algorithmus nach Ai et al. (2016), wie er in dieser Arbeit eingesetzt wurde, noch das Evaluierungsverfahren von Chrobak et al. (2016) konnten nach diesem intensiven Studium die Bearbeiterin überzeugen, dass sie die Qualität einer manuellen Vereinfachung erreichen bzw. bewerten können. Wird der Vereinfachungsgrad so gering gewählt, dass keine Abweichungen zur Originallinie sichtbar sind, bleiben zu viele Details erhalten. Lässt man einen gröberen Vereinfachungsgrad zu, besteht die Gefahr, dass Formveränderungen sichtbar werden, die den kartographischen Regeln widersprechen. Dabei muss jedoch zum einen noch einmal darauf hingewiesen werden, dass der Algorithmus, wie Ai et al. (2016) ihn in ihrem Artikel beschreiben, nicht vollständig umgesetzt werden konnte. Insbesondere die Option, die Mittelachsen der envelope zones zu verwenden, wurde nicht realisiert. Sie würde die Verwendung des Parameters *m* und damit eine Steuerung der Flächenbilanz ermöglichen. Der visuelle Eindruck würde sich jedoch möglicherweise nur eingeschränkt verbessern, da die Mittelachse aus der Verbindung der Mittelpunkte der Dreiecksseiten innerhalb der envelope zone gebildet wird. Das kann zu unerwünschten kleinen Knitterungen führen. Das haben Beobachtungen beim Versuch, diese Option zu programmieren, gezeigt. Zum anderen sollten Studien mit weiteren Testlinien durchgeführt werden.

In dieser Arbeit wurde wie von Ai et al. (2016) ausschließlich der Parameter der bend depth (Skelettpfadlänge) für die Linienvereinfachung verwendet. Das Verfahren birgt jedoch durch Triangulation und bend-Struktur konzeptionelle Probleme, die auch durch die Verwendung von bend width oder bend size als Parameter für die Vereinfachung nicht gelöst werden können. Interessant wäre es zu untersuchen, ob sich durch die Verwendung der Mittelachsen und gegebenenfalls eine wiederholte Anwendung des Algorithmus die Ergebnisse verbessern ließen. Dafür sollte allerdings zunächst die Verarbeitungsgeschwindigkeit optimiert werden.

Für die Ermittlung geeigneter Vereinfachungsparameter wäre eine andere Strategie zu entwickeln als die iterative Anwendung des Verfahrens von Chrobak et al. (2016).

7. Literaturverzeichnis

Ai, T. (2007): The drainage network extraction from contour lines for contour line generalization. In: *ISPRS Journal of Photogrammetry and Remote Sensing* 62 (2), S. 93–103. DOI: 10.1016/j.isprsjprs.2007.04.002

Ai, T.; Guo, R.; Liu, Y. (2000): A binary tree representation of curve hierarchical structure based on Gestalt principles. In: Proceedings of 9th international symposium on spatial Data handling. Beijing, China, 10–12 August 2000, S. 2a30–2a43.

Ai, T.; Ke, S.; Yang, M.; Li, J. (2016): Envelope generation and simplification of polylines using Delaunay triangulation. In: I*nternational Journal of Geographical Information Science* 31 (2), S. 297–319. DOI: 10.1080/13658816.2016.1197399.

Ai, T.; Zhou, Q.; Zhang, X.; Huang, Y.; Zhou, M. (2014): A Simplification of Ria Coastline with Geomorphologic Characteristics Preserved. In: *Marine Geodesy* 37 (2), S. 167–186. DOI: 10.1080/01490419.2014.903215.

Bartholomew, J. (Hg.) (1955): The Times Atlas of the World. Mid-Century Edition. Volume III Northern Europe.

Bartholomew, J. (Hg.) (1967): The Times Atlas of the World. Comprehensive Edition.

Bloch, M. (2018): mapshaper. Online verfügbar unter https://github.com/mbloch/mapshaper, zuletzt geprüft am 29.04.2018.

Cheung, C. K.; Shi, W. (2006): Positional error modeling for line simplification based on automatic shape similarity analysis in GIS. In: *Computers & Geosciences* 32 (4), S. 462–475.

Chrobak, T. (2000): Modelling of spatial data in a database for the needs of cartographic generalization. Modelowanie danych przestrzennych w bazie danych dla potrzeb generalizacji kartograficznej. In: *Geodezja i Kartografia* 49 (1), S. 7–19.

Chrobak, T. (2010): The role of least image dimensions in generalization of object in spatial databases. In: *Geodesy and Cartography* 59 (2), S. 99–120. DOI: 10.2478/v10277-012-0004-y.

Chrobak, T.; Szombara, S.; Kozoł, K.; Lupa, M. (2016): A method for assessing generalized data accuracy with linear object resolution verification. In: *Geocarto International* 32 (3), S. 238–256. DOI: 10.1080/10106049.2015.1133721.

Douglas, D. H.; Peucker, T. K. (1973): Algorithms for the Reduction of the Number of Points Required to present a Digitized Line or its Caricature. In: *Cartographica: The International Journal for Geographic Information and Geovisualization* 10 (2), S. 112–122. DOI: 10.3138/FM57-6770-U75U-7727.

Enescu, I. I.; Panchaud, N. H.; Heitzler, M.; Enescu, C. M. I.; Hurni, L. (2015): Towards Better WMS Maps Through the Use of the Styled Layer Descriptor and Cartographic Conflict Resolution for Linear Features. In: *The Cartographic Journal* 52 (2), S. 125–136. DOI: 10.1080/00087041.2015.1119468.

ESRI (2016a): Funktionsweise von „Linie vereinfachen“—Hilfe | ArcGIS for Desktop. Online verfügbar unter http://desktop.arcgis.com/de/arcmap/10.3/tools/cartography-toolbox/how-simplify-line-works.htm, zuletzt aktualisiert am 23.03.2016, zuletzt geprüft am 20.02.2018.

ESRI (2016b): PointGeometry—Hilfe | ArcGIS for Desktop. Online verfügbar unter http://desktop.arcgis.com/de/arcmap/10.3/analyze/arcpy-classes/pointgeometry.htm, zuletzt aktualisiert am 23.3.2016, zuletzt geprüft am 1.8.2019.

ESRI (2016c): TIN erstellen—Hilfe | ArcGIS for Desktop. Online verfügbar unter http://desktop.arcgis.com/de/arcmap/10.3/tools/3d-analyst-toolbox/create-tin.htm, zuletzt aktualisiert am 23.03.2016, zuletzt geprüft am 1.8.2019

ESRI (2018): Linie vereinfachen—Hilfe | ArcGIS Desktop. Online verfügbar unter http://desktop.arcgis.com/de/arcmap/10.5/tools/cartography-toolbox/simplify-line.htm, zuletzt aktualisiert am 19.03.2018, zuletzt geprüft am 18.06.2019.

Foerster, T.; Stoter, J.; van Oosterom, P. (2012): On-demand base maps on the web generalized according to user profiles. In: *International Journal of Geographical Information Science* 26 (1), S. 99–121. DOI: 10.1080/13658816.2011.574292.

Gaffuri, J. (2011): Improving Web Mapping with Generalization. In: *Cartographica: The International Journal for Geographic Information and Geovisualization* 46 (2), S. 83–91. DOI: 10.3138/carto.46.2.83.

Gruppi, M. G.; De Magalhaes, S. V. G.; Andrade, M. V. A.; Franklin, W. R.; Li, W. (2015): An efficient and topologically correct map generalization heuristic. In: Proceedings of the 17th International Conference on Enterprise Information Systems. Barcelona, Spain, 27–30 April 2015, Band 1. S. 516–525.

Hake, G. (1982): Kartographie. Band 1. Berlin, New York.

Huang, L.; Ai, T.; van Oosterom, P.; Yan, X.; Yang, M. (2017): A Matrix-Based Structure for Vario-Scale Vector Representation over a Wide Range of Map Scales. The Case of River Network Data. In: *ISPRS International Journal of Geo-Information* 6 (7), Artikel 218, S. 1–20. DOI: 10.3390/ijgi6070218.

IGN Institut Géographique National (1958): Carte de France_1/100000. Morlaix. Feuille B-8. Paris.

IGN Institut Géographique National (1961a): Carte de France _ 1/250000. Brest. Feuille NL-NM 30-1-10. Paris.

IGN Institut Géographique National (1961b): Carte de France _ 1/250000. Saint-Brieuc. Feuille NM 30-11. Paris.

IGN Institut National de l'Information Géographique et de Forestière (2014): SCAN Historique®SCAN 50® – Version 1.0. Descriptif de contenu. Online verfügbar unter http://professionnels.ign.fr/doc/DC_SCAN50_Historique_1-0.pdf, zuletzt aktualisiert am 2016, zuletzt geprüft am 31.03.2019.

IGN Institut National de l'Information Géographique et de Forestière (2017): SCAN Historique®. SCAN 50® années 1950, sur la France métropolitaine. Paris. Internet. Online verfügbar unter http://professionnels.ign.fr/scanhisto, zuletzt geprüft am 15.10.2018.

Jenks, G. F. (1989): Geographic Logic In Line Generalization. In: *Cartographica: The International Journal for Geographic Information and Geovisualization* 26 (1), S. 27–42. DOI: 10.3138/L426-1756-7052-536K.

Jones, C. B.; Bundy, G. L.; Ware, M. J. (1995): Map Generalization with a Triangulated Data Structure. In: *Cartography and Geographic Information Systems* 22 (4), S. 317–331. DOI: 10.1559/152304095782540221.

Lafay, S.; Braun, A.; Chandler, D.; Michaud, M.; Ricaud, L.; Mustière, S. (2015): Automatic mapping and innovative on-demand mapping services at IGN France. In: *Cartography and Geographic Information Science* 42 (1), S. 54–68.

Li, Z. (2007): Algorithmic foundation of multi-scale spatial representation. London.

Li, Z.; Openshaw, S. (1990): A Natural Principle of Objective Generalization of Digital Map Data and Other Spatial Data. RRL Research Report. University of Newcastle upon Tyne, Newcastle upon Tyne. Centre for Urban and Regional Development Studies (CURDS).

Li, Z.; Openshaw, S. (1992): Algorithms for automated line generalization1 based on a natural principle of objective generalization. In: *International journal of geographical information systems* 6 (5), S. 373–389. DOI: 10.1080/02693799208901921.

Li, Z.; Openshaw, S. (1993): A natural principle for the objective generalization of digital maps. In: *Cartography and Geographic Information Systems* 20 (1), S. 19–29.

Lopes, T.; António, J.; Catalão, J. (2013): A new method for line generalization based on artificial intelligence algorithms. In: 8th Iberian Conference on Information Systems and Technologies (CISTI), 2013. Lisboa, Portugal, 19–22 June 2013. S. 1–6.

Mackaness, W. A.; Ruas, A. (2007): Evaluation in Map Generalisation Process. In: W. A. Mackaness, A. Ruas, L. T. Sarjakoski (Hg.): Generalisation of geographic information. Cartographic modelling and applications. S. 89–112.

McMaster, R. B. (1986): A Statistical Analysis of Mathematical Measures for Linear Simplification. In: *The American Cartographer* 13 (2), S. 103–116. DOI: 10.1559/152304086783900059.

McMaster, R. B. (1987a): Automated Line Generalization. In: *Cartographica: The International Journal for Geographic Information and Geovisualization* 24 (2), S. 74–111. DOI: 10.3138/3535-7609-781G-4L20.

McMaster, R. B. (1987b): The Geometric Properties of Numerical Generalization. In: *Geographical Analysis* 19 (4), S. 330–346. DOI: 10.1111/j.1538-4632.1987.tb00134.x.

Meulemans, W.; van Renssen, A.; Speckmann, B. (2010): Area-preserving subdivision schematization. In: Geographic information science: Proceedings of 6th international conference, GIScience 2010. Zurich, Switzerland, 14–17 September 2010. S. 160–174.

Miao, R.; Song, J.; Feng, M. (2017): A feature selection approach towards progressive vector transmission over the Internet. In: *Computers & Geosciences* 106, S. 150–163. DOI: 10.1016/j.cageo.2017.06.005.

Moore, A.; Mason, C.; Whigham, P.; Thompson-Fawcett, M. (2015): Polygon Generalization with Circle Arcs. In: Proceedings of the Geospatial Science Research 2 Symposium. Melbourne, Australia, 10–12 December 2012, Band 1328. 7 Seiten. Online verfügbar unter http://ceur-ws.org/Vol-1328, zuletzt geprüft am 19.02.2018.

Müller, J.-C (1991): Generalization of spatial databases. In: D. Maguire, M. Goodchild und D. Rhind (Hg.): Geographical Information Systems: Principles and Applications. Band 1. S. 457–475.

Ordnance Survey (1954): International Map of the World. NM 30 & PTS. 29 & 31 London. Series 1301. Edition 4 -GSGS. Online verfügbar unter http://legacy.lib.utexas.edu/maps/imw/txu-oclc-6654394-nm-30-31-4th-ed.jpg, zuletzt geprüft am 06.05.2019.

Pallero, J. L. G. (2013): Robust line simplification on the plane. In: *Computers & Geosciences* 61, S. 152–159. DOI: 10.1016/j.cageo.2013.08.011.

Park, W.; Yu, K. (2011): Hybrid line simplification for cartographic generalization. In: *Pattern Recognition Letters* 32 (9), S. 1267–1273. DOI: 10.1016/j.patrec.2011.03.013.

Perkal, J. (1966): An attempt at objective generalization. Discussion paper, 10, Michigan. Michigan Inter-University Community of Mathematical Geographers.

Qian, H.; Zhang, M.; Wu, F. (2016): A New Simplification Approach Based on the Oblique-Dividing-Curve Method for Contour Lines. In: *ISPRS International Journal of Geo-Information* 5 (9), Artikel 153, S. 1–21. DOI: 10.3390/ijgi5090153.

Rangayyan, R. M.; Guliato, D.; Carvalho, J. D. de; Santiago, S. A. (2008): Polygonal approximation of contours based on the turning angle function. In: *Journal of Electronic Imaging* 17 (2), Artikel 023016. DOI: 10.1117/1.2920413.

Raposo, P. (2013): Scale-specific automated line simplification by vertex clustering on a hexagonal tessellation. In: *Cartography and Geographic Information Science* 40 (5), S. 427–443. DOI: 10.1080/15230406.2013.803707.

Regnauld, N. (2014): Automatic Map Derivation at Ordnance Survey GB. In: D. Burghardt, C. Duchêne und W. A. Mackaness (Hg.): Abstracting Geographic Information in a Data Rich Wor-

ld Methodologies and Applications of Map Generalisation. Lecture Notes in Geoinformation and Cartography. S. 351–355.

Revell, P.; Regnauld, N.; Bulbrooke, G. (2011): OS VectorMap® District: automated generalisation, text placement and conflation in making public data public. In: Proceedings of the 25th International Cartographic Conference. Paris, France, 3–8 July 2011. 13 Seiten.

Robinson, A. H.; Sale, R. D.; Morrison, J. L. (1978): Elements of cartography. 4. ed. New York u.a.

Saalfeld, A. (1999): Topologically Consistent Line Simplification with the Douglas-Peucker Algorithm. In: *Cartography and Geographic Information Science* 26 (1), S. 7–18.

Saliszczew, K. A. (1998): General cartography. 2. ed. Warschau.

Samsonov, T. E.; Yakimova, O. P. (2017): Shape-adaptive geometric simplification of heterogeneous line datasets. In: *International Journal of Geographical Information Science* 31 (8), S. 1–36. DOI: 10.1080/13658816.2017.1306864.

Saux, Eric (2003): B-spline Functions and Wavelets for Cartographic Line Generalization. In: *Cartography and Geographic Information Science* 30 (1), S. 33–50. DOI: 10.1559/152304003100010938.

Shi, W.; Cheung, C. K. (2006): Performance Evaluation of Line Simplification Algorithms for Vector Generalization. In: *The Cartographic Journal* 43 (1), S. 27–44. DOI: 10.1179/000870406X93490.

Shivanth, M. P.; Kale, Sandeep; Sarda, N. L.; Bellur, Umesh; Goje, Vishal (2014): Moving window based geometry simplification with topology constraints. In: 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (ACM SIGSPATIAL GIS 2014). Dallas, USA, 4–7 November 2014. S. 625–628.

Song, J.; Miao, R. (2016): A Novel Evaluation Approach for Line Simplification Algorithms towards Vector Map Visualization. In: *ISPRS International Journal of Geo-Information* 5 (12), Artikel 223, S. 1–13. DOI: 10.3390/ijgi5120223.

Staufenbiel, W. (1973): Zur Automation der Generalisierung topographischer Karten mit besonderer Berücksichtigung großmaßstäbiger Gebäudedarstellungen. Dissertation. Universität Hannover, Hannover.

Stoter, J. (2005): Generalisation within NMA's in the 21st century. Paper. In: Mapping approaches into a changing world. International Cartographic Conference. Coruña, Spain, 9–16 July 2005. 11 Seiten.

Stoter, J. (2010): State-of-the-art of automated generalisation in commercial software. In: *Official Publication - EuroSDR* 58, S. 9–231.

Stoter, J.; Post, M.; van Altena, V.; Nijhuis, R.; Bruns, B. (2014): Fully automated generalization of a 1:50k map from 1:10k data. In: *Cartography and Geographic Information Science* 41 (1), S. 1–13. DOI: 10.1080/15230406.2013.824637.

Stoter, J.; van Altena, V.; Post, M.; Burghardt, D.; Duchêne, C. (2016): Automated generalisation within NMAs in 2016. In: XXIII ISPRS Congress. Prague, Czech Republic, 12–19 July 2016, Volume XLI-B4. (The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Volume XLI-B4), S. 647–652.

Stoter, J.; van Smaalen, J.; Bakker, N.; Hardy, P. (2009): Specifying Map Requirements for Automated Generalization of Topographic Data. In: *The Cartographic Journal* 46 (3), S. 214–227. DOI: 10.1179/174327709X446637.

Tong, X.; Jin, Y.; Li, L.; Ai, T. (2015): Area-preservation Simplification of Polygonal Boundaries by the Use of the Structured Total Least Squares Method with Constraints. In: *Transactions in GIS* 19 (5), S. 780–799. DOI: 10.1111/tgis.12130.

Töpfer, F.; Pillewizer, W. (1966): The Principles of Selection. In: *The Cartographic Journal* 3 (1), S. 10–16.

Touya, G.; Girres, J.-F. (2013): ScaleMaster 2.0. A ScaleMaster extension to monitor automatic multi-scales generalizations. In: *Cartography and Geographic Information Science* 40 (3), S. 192–200. DOI: 10.1080/15230406.2013.809233.

Valentin, H. (Hg.) (1954): Die Küsten der Erde. Beiträge zur allgemeinen und regionalen Küstenmorphologie. In: *Petermanns Geographische Mitteilungen*. Ergänzungshefte 246.

van Oosterom, P. (2009): Research and development in geo-information generalisation and multiple representation. In: *Computers, Environment and Urban Systems* 33 (5), S. 303–310. DOI: 10.1016/j.compenvurbsys.2009.07.001.

Veregin, H. (1999): Line Simplification, Geometric Distortion, and Positional Error. In: *Cartographica: The International Journal for Geographic Information and Geovisualization* 36 (1), S. 25–39. DOI: 10.3138/D7R4-M1M7-6632-73X1.

Visvalingam, M. (2016): The Visvalingam Algorithm. Metrics, Measures and Heuristics. In: *The Cartographic Journal* 53 (3), S. 242–252. DOI: 10.1080/00087041.2016.1151097.

Visvalingam, M.; Whelan, J. C. (2016): Implications of Weighting Metrics for Line Generalization with Visvalingam's Algorithm. In: *The Cartographic Journal* 53 (3), S. 253–267. DOI: 10.1080/00087041.2016.1149906.

Visvalingam, M.; Whyatt, J. D. (1993): Line generalisation by repeated elimination of points. In: *The Cartographic Journal* 30 (1), S. 46–51. DOI: 10.1179/000870493786962263.

Wang, Z.; Müller, J.-C. (1998): Line Generalization Based on Analysis of Shape Characteristics. In: *Cartography and Geographic Information Systems* 25 (1), S. 3–15.

Yan, J.; Guilbert, E.; Saux, E. (2015): An Ontology of the Submarine Relief for Analysis and Representation on Nautical Charts. In: *The Cartographic Journal* 52 (1), S. 58–66. DOI: 10.1179/1743277413Y.0000000050.

Yu, Z. (1993): The effects of scale change on map structure. PhD. Clark University, Worcester.

Zhao, Z.; Saalfeld, A. (1997): Linear-time sleeve-fitting polyline. In: AutoCarto 13 Conference. Seattle, USA, 7–10 April 1997, Band 5. S. 214–223.

Zhou, S.; Jones, C. B. (2005): Shape-aware line generalisation with weighted effective area. In: Developments in Spatial Data Handling. 11th International Symposium on Spatial Data Handling. Leicester, UK, 23–25 August 2004. S. 369–380. Online verfügbar unter https://users.cs.cf.ac.uk/C.B.Jones/Zhou_JonesSDH04.pdf, zuletzt geprüft am 24.04.2018.

8. Anhang

A1 AssignBSIDtoTriangles

```
# -*- coding: utf-8 -*-
# Autorin: Christiane Enderle
# Datum: 29.1.2019
# Fuer ArcGIS Desktop 10.3.1 mit Python 2.7.8

# Das Python-Skript-Tool ist nur im Rahmen des Modells, das den Algorithmus von Ai et al.
(2016) implementiert, sinnvoll einsetzbar.
# Dieses Skript weist den aus der Triangulation hervorgegangenen Dreiecken die Feature-ID
des bend systems zu (BSID), in dem sie liegen.
# Wegen Ungenauigkeiten zwischen Bend_Systems und Dreiecken im Randbereich der Triangulation
wird bei der raeumlichen Abfrage
# auch auf Ueberlappung geprueft. Ausserdem kann es vorkommen, dass ein Bend-System aus nur
einem Dreieck besteht. Deshalb zusaetzlich
# die Abfrage "equals".
# Intersect ist hier ungeeignet, weil es die Anzahl der Dreiecke um sliver-polygons erhoeht.
Das Aufloesen der sliver-polygons wuerde
# Polygone mit mehr als drei Punkten ergeben, die also keine Dreiecke mehr waeren.
# Dreiecke sind jedoch zwingend notwendig fuer die weitere Verarbeitung.
# Dieses Skript kann vermutlich auch durch das ArcGIS-Tool Spatial Join mit der raeumlichen
Abfrage nach Dreiecken,
# die "Have their center in" Bendsystems, ersetzt werden (Toolbox Analysis Tools, Toolset
Overlay).

import arcpy
arcpy.env.overwriteOutput = 1

input_triangles = arcpy.GetParameterAsText(0) # B_Triangles.shp
input_bends = arcpy.GetParameterAsText(1) # B_Bend_systems.shp
output_triangles = arcpy.GetParameterAsText(2) # B_Triangles_with_BSID.shp"

# Den Dreiecken wird ein Attribut fuer die BSID hinzugefuegt.
arcpy.AddField_management(input_triangles,"BS_FID","LONG")
# Das Feld wird fuer alle Dreiecke zunaechst mit dem Wert -2 gefuellt,
# ein Wert, der als FID nicht auftreten kann, damit bei der Schleife durch die Dreieicke
# all diejenigen uebergangen werden koennen, die bereits eine BSID erhalten haben (siehe
erste if-Anweisung).
arcpy.CalculateField_management(input_triangles,"BS_FID", -2)
with arcpy.da.SearchCursor(input_bends,["SHAPE@","FID"]) as cursor_bend:
    for bend in cursor_bend:
        geom_bend = bend[0]
        FID = bend[1]
        with arcpy.da.UpdateCursor(input_triangles, ["SHAPE@","BS_FID"]) as cursor_tria:
            for tri in cursor_tria:
                BS_FID = tri[1]
                if BS_FID == -2: # Wenn das Dreieck noch keinem Bendsystem zugeordnet ist...
                    geom_tria = tri[0]
                    if geom_tria.within(geom_bend) or geom_tria.overlaps(geom_bend)or
                    geom_tria.equals(geom_bend):
                        tri[1] = FID
                        cursor_tria.updateRow(tri)

# Kopiere "B_Triangles.shp", damit das Ergebnis dieses Skripts im Modell dem Skript
"BuildBends" zur Verfuegung steht.
arcpy.CopyFeatures_management(input_triangles,output_triangles)
```

A2 DetermineType2and3

```
# Autor: Christiane Enderle
# Datum: 8.12.2018
# Fuer ArcGIS Desktop 10.3.1 mit Python 2.7.8

# Das Python-Skript-Tool ist nur im Rahmen des Modells, das den Algorithmus von Ai et al.
(2016) implementiert, sinnvoll einsetzbar.
# Es identifiziert Delaunay-Dreiecke vom Typ 2 (End-Dreiecke einer Bucht) und Typ 3
(Fuelldreiecke) im Rahmen des Algorithmus von Ai et al (2016).
# Dreiecke des Typs 2 haben zwei gemeinsame Segmente mit der Originallinie und liegen am
inneren Ende eines Kurvenabschnitts (bends).
# Dreiecke des Typs 3 haben ein gemeinsames Segment mit der Originallinie und sind
Fuelldreiecke.
# Der Typ wird in das Feld "Type" in der Featureclass der Delaunay-Dreiecke eingetragen.
# Voraussetzung fuer die Anwendung dieses Skripts:
# Typ 1 (Muendungsdreiecke), Typ 4 (Verzweigungen), Typ 5 (Dreiecke, die gleichzeitig
Muendungs- und Verzweigungsdreiecke sind)
# muessen bereits eingetragen sein.

import arcpy

# Eingabefeld fuer den Layer der Delaunay-Dreiecke (es muss sich um einen Layer handeln,
weil das Select-Tool nichts anderes akzeptiert):
input_triangles = arcpy.GetParameterAsText(0)
# Eingabefeld fuer die Punkte der Originallinie, die durch die Delaunay-Triangulation
gegenueber dem Original moeglicherweise verdichtet
# und dann in die Originallinie integriert wurden:
input_points = arcpy.GetParameterAsText(1) # B_Original_Points_densified.shp
# Eingabefeld fuer den output. Es muss eine Output-Datei generiert werden, weil die
Typisierung Bedingung fuer das Skript BuildBends ist.
output_typified = arcpy.GetParameterAsText(2) # "B_Triangles_typified.shp"

# Auswahl der noch nicht typisierten Dreiecke (Typ 0):
arcpy.SelectLayerByAttribute management(input triangles,"NEW SELECTION",'"Type" = 0')

# Fuer den Dreiecke-Cursor werden nur die Felder Geometrie und Type benoetigt:
fields_triangles = ['SHAPE@','Type']
# Ein UpdateCursor wird angelegt fuer das Eintragen des Typs beim jeweiligen Dreieck
# und fuer das Verschneiden der Dreiecke mit den Punkten der verdichteten Originallinie.
# Der Cursor umfasst jetzt nur die oben ausgewaehlten Dreiecke vom Typ 0:
curT = arcpy.da.UpdateCursor(input_triangles,fields_triangles)

# Fuer den Punkt-Cursor werden die Felder Geometrie und die Original-Feature-ID benoetigt,
# weil diese der Reihenfolge der Punkte auf der Linie entspricht (wichtig fuer
Typ-2-Dreiecke).
fields_points = ['SHAPE@','ORIG_FID']
# Ein Cursor wird angelegt, um zu pruefen, welche Punkte auf dem jeweiligen Dreieck liegen
und welche Feature-ID sie haben:
curP = arcpy.da.SearchCursor(input_points,fields_points)

# Eine leere Liste fuer die Punkte, die jeweils auf einem einzelnen Dreieck liegen:
trianglePointList = []

# Der Datensatz der Dreiecke wird iteriert:
for t in curT:
# Ein Cursor (der unten folgende Punkte-Cursor) kann nur einmal durchlaufen werden.
# Soll er mehrmals durchlaufen werden, muss er nach jedem Durchlauf geloescht und neu
angelegt werden:
    del curP
    # Die Variable fields_points ist durch den Loeschvorgang ebenfalls weg.
    # Die benoetigten Felder werden hier nun als Literals angegeben:
    curP = arcpy.da.SearchCursor(input_points,['SHAPE@','ORIG_FID'])
    # Variable tri speichert die Geometrie des aktuellen Dreiecks:
    tri = t[0]
    # Zaehler fuer Trefferliste der folgenden Verschneidungspruefung:
    i = 0

    # Der Datensatz der Punkte wird iteriert:
    for p in curP:
```

```
            # Variable poi speichert die Geometrie des aktuellen Punktes:
            poi = p[0]
            # Pruefen, ob der Punkt einer der Ecken des aktuellen Dreiecks ist.
            # Dafuer muss erstens mit boundary() die Umrisslinie des Dreiecks generiert werden.
            # Zweitens liefert der Operator within nur fuer diejenigen Punkte den Wert true, die
            nicht Anfangs- oder Endpunkt sind.
            # Der Operator touches liefert fuer Punkte, die mit dem Anfangs- oder Endpunkt der
            Umrisslinie des Dreiecks zusammenfallen, den Wert true.
            if poi.within(tri.boundary())or poi.touches(tri.boundary()):
                # Schreibe die ORIG_FID des Punktes in die Trefferliste:
                trianglePointList.append(p[1])
                # Der Zaehler kann nach dem ersten Treffer zweimal erhoeht werden.
                # Nach drei Treffern sind fuer das aktuelle Dreieck keine weiteren Treffer mehr
                zu erwarten, weil ein Dreieck nur drei Eckpunkte hat.
                if i < 2:
                    i = i+1
                # Nach drei Treffern wird die Trefferliste mit den Original-Feature-IDs sortiert,
                # so dass sie die Punkte in der Reihenfolge, wie sie auf der verdichteten Linie
                liegen, wiedergibt.
                else:
                    trianglePointList.sort()
                    f = trianglePointList[0]
                    # Wenn die Original-Feature-IDs drei aufeinanderfolgende Zahlen sind, muss
                    es sich um ein Dreieck vom Typ 2 handeln.
                    if trianglePointList[1] == f+1 and trianglePointList[2] == f+2:
                        # Der Typ 2 wird vom UpdateCursor in den Datensatz des aktuellen
                        Dreiecks geschrieben:
                        t[1] = 2
                        curT.updateRow(t)
                    else:
                        # Andernfalls muss es ein Dreieck vom Typ 3 sein.
                        # Der Typ 3 wird vom UpdateCursor in den Datensatz des aktuellen
                        Dreiecks geschrieben:
                        t[1] = 3
                        curT.updateRow(t)
                    # Die Trefferliste wird geleert und damit fuer die Pruefung des naechsten
                    Dreiecks vorbereitet:
                    trianglePointList = []
                    # Die Schleife durch den Punkt-Cursor kann beendet werden und es geht mit
                    dem naechsten Dreieck weiter:
                    break
    del curT

    # Die Auswahl auf dem Layer wird aufgehoben:
    arcpy.SelectLayerByAttribute_management(input_triangles,"CLEAR_SELECTION",'"Type" = 2')
    # Der Layer wird in ein output-file geschrieben:
    arcpy.CopyFeatures_management(input_triangles,output_typified)
```

A3 TrianglePoints

```
# Autor: Christiane Enderle
# Datum: 15.12.2018
# Fuer ArcGIS Desktop 10.3.1 mit Python 2.7.8

# Das Python-Skript-Tool ist nur im Rahmen des Modells, das den Algorithmus von Ai et al.
(2016) implementiert, sinnvoll einsetzbar.
# Es bestimmt fuer jedes Dreieck der Delaunay-Triangulation die ORIG_FID der zugehoerigen
Punkte (Original Feature-ID auf der Originallinie),
# sortiert die ORIG_FIDs aufsteigend und traegt sie als Attribute P1, P2, P3 in den
Datensatz der Dreiecke ein.

import arcpy

arcpy.env.overwriteOutput = 1

# Eingabefeld fuer den Datensatz der Delaunay-Dreiecke (nach der Typisierung der Dreiecke):
input_triangles = arcpy.GetParameterAsText(0) # "B_Triangles_typified.shp"
# Eingabefeld fuer die Punkte der Originallinie, die durch die Delaunay-Triangulation
gegenueber dem Original moeglicherweise verdichtet
# und dann in die Originallinie integriert wurden:
input_points = arcpy.GetParameterAsText(1) # "B_Original_Points_densified.shp"
# Eingabefeld fuer Ergebnis dieses Skripts:
output_triangles = arcpy.GetParameterAsText(2) # "B_Triangles_with_Points.shp"
# Der Dreieckdatensatz erhaelt fuer die ORIG_FID eines jeden Dreieckspunkts ein Attribut:
arcpy.AddField_management(input_triangles,"Point1","LONG")
arcpy.AddField_management(input_triangles,"Point2","LONG")
arcpy.AddField_management(input_triangles,"Point3","LONG")
# Fuer den Cursor werden die Felder Geometrie, Point1, Point2 und Point3 benoetigt:
fields_triangles = ['SHAPE@','Point1','Point2','Point3']
# Ein UpdateCursor wird angelegt fuer das Eintragen der ORIG_FIDs der Punkte beim jeweiligen
Dreieck
# und fuer das Verschneiden der Dreiecke mit den Punkten der verdichteten Originallinie.
curT = arcpy.da.UpdateCursor(input_triangles,fields_triangles)

# Eine leere Liste fuer die Punkte, die jeweils auf einem einzelnen Dreieck liegen:
trianglePointList = []

# Der Datensatz der Dreiecke wird iteriert:
for t in curT:
    # Variable tri speichert die Geometrie des aktuellen Dreiecks:
    tri = t[0]
    # Ein Cursor wird angelegt, um zu pruefen, welche Punkte auf dem jeweiligen Dreieck
    liegen und zum Auslesen der ORIG_FIDs.
    # Ein Cursor (der unten folgende Punkte-Cursor) kann nur einmal durchlaufen werden.
    # Soll er mehrmals durchlaufen werden, muss er nach jedem Durchlauf geloescht und neu
    angelegt werden:
    # Es werden die Felder Geometrie und die Original-Feature-ID (ORIG_FID) benoetigt,
    # weil diese der Reihenfolge der Punkte auf der Linie entspricht (wichtig fuer
    Typ-2-Dreiecke und die Ermittlung der bends (Kurvenabschnitte)).
    curP = arcpy.da.SearchCursor(input_points,['SHAPE@','ORIG_FID'])
    # Zaehler fuer Trefferliste der folgenden Verschneidungspruefung:
    i = 0
    # Der Datensatz der Punkte wird iteriert:
    for p in curP:
        # Variable poi speichert die Geometrie des aktuellen Puntkes:
        poi = p[0]
        # Pruefen, ob der Punkt einer der Ecken des aktuellen Dreiecks ist.
        # Dafuer muss erstens mit boundary() die Umrisslinie des Dreiecks generiert werden.
        # Zweitens liefert der Operator within nur fuer diejenigen Punkte den Wert true, die
        nicht Anfangs- oder Endpunkt sind.
        # Der Operator touches liefert fuer Punkte, die mit dem Anfangs- oder Endpunkt der
        Umrisslinie des Dreiecks zusammenfallen, den Wert true.
        if poi.within(tri.boundary())or poi.touches(tri.boundary()):
            # Schreibe die ORIG_FID des Punktes in die Trefferliste:
            trianglePointList.append(p[1])
            # Der Zaehler kann nach dem ersten Treffer zweimal erhoeht werden.
            # Nach drei Treffern sind fuer das aktuelle Dreieck keine weiteren Treffer mehr
            zu erwarten, weil ein Dreieck nur drei Eckpunkte hat.
```

```
            if i < 2:
                i = i+1
            # Nach drei Treffern wird die Trefferliste mit den Original-Feature-IDs sortiert,
            # so dass sie die Punkte in der Reihenfolge, wie sie auf der verdichteten Linie
            liegen, wiedergibt.
            else:
                trianglePointList.sort()
                t[1] = trianglePointList[0]
                t[2] = trianglePointList[1]
                t[3] = trianglePointList[2]
                curT.updateRow(t)
                # Die Trefferliste wird geleert und damit fuer die Pruefung des naechsten
                Dreiecks vorbereitet:
                trianglePointList = []
                # Die Schleife durch den Punkt-Cursor kann beendet werden:
                break
    # Der Punkt-Cursor wird geloescht, damit er fuer das naechste Dreieck erneut angelegt
    und durchlaufen werden kann.
    del curP
del curT
# Kopiere input_triangles in ein neues Shapefile, damit im Modell sichergestellt werden kann,
# dass das Skript "BuildBends" auf das Ergebnis dieses Skriptes hier zugreift:
arcpy.CopyFeatures_management(input_triangles,output_triangles)
```

A4 CorrectTopology

```
# Autorin: Christiane Enderle
# Datum: November 2018
# Fuer ArcGIS Desktop 10.3.1 mit Python 2.7.8

# Das Python-Skript-Tool ist nur im Rahmen des Modells, das den Algorithmus von Ai et al.
(2016) implementiert, sinnvoll einsetzbar.
# Es nimmt als input die Segmente der durch die Triangulation verdichteten Originallinie,
# die aus der bisherigen Verarbeitung in den Attributen LEFT_FID und RIGHT_FID die BSID
(Bendsystem-Feature-ID) haben,
# wobei die Lage links oder rechts von der Originallinie teilweise vertauscht ist.
# Das input-file enthaelt aus der bisherigen Verarbeitung ausserdem Attribute mit den
Koordinaten der Start- und Endpunkte in der korrekten Richtung
# und Koordinaten von Start- und Endpunkten, die teilweise in der Richtung im Vergleich zum
Verlauf der Originallinie vertauscht sind.
# Das Skript vergleicht die Koordinaten der korrekten Start- und Endpunkte mit den
Koordinaten der potenziell vertauschten
# und uebertraegt oder vertauscht entsprechend die BSID aus den Attributen "Left_FID" bzw.
"Right_FID" in die Attribute "LeftOfLine" bzw. "R_OfLine".

import arcpy

def CorrectTopology(file,file_out):
    fields = ['LEFT_FID','RIGHT_FID','START_X','START_Y','END_X','END_Y','LeftOfLine',
    'R_OfLine']
    cur_correct_left = arcpy.da.UpdateCursor(file,fields, '(-0.005<"START_X_1"- "START_X")
    AND ("START_X_1"- "START_X"< 0.005) AND (-0.005<"START_Y_1"- "START_Y") AND
    ("START_Y_1"- "START_Y"< 0.005)')
    for row in cur_correct_left:
        row[6] = row[0]
        cur_correct_left.updateRow(row)
    cur_correct_right = arcpy.da.UpdateCursor(file,fields, '(-0.005<"START_X_1"- "START_X")
    AND ("START_X_1"- "START_X"< 0.005) AND (-0.005<"START_Y_1"- "START_Y") AND
    ("START_Y_1"- "START_Y"< 0.005)')
    for row in cur_correct_right:
        row[7] = row[1]
        cur_correct_right.updateRow(row)
    cur_false_left = arcpy.da.UpdateCursor(file,fields, 'NOT((-0.005<"START_X_1"- "START_X")
    AND ("START_X_1"- "START_X"< 0.005) AND (-0.005<"START_Y_1"- "START_Y") AND
    ("START_Y_1"- "START_Y"< 0.005))')
    for row in cur_false_left:
        row[6] = row[1]
        cur_false_left.updateRow(row)
    cur_false_right = arcpy.da.UpdateCursor(file,fields, 'NOT((-0.005<"START_X_1"-
    "START_X") AND ("START_X_1"- "START_X"< 0.005) AND (-0.005<"START_Y_1"- "START_Y") AND
    ("START_Y_1"- "START_Y"< 0.005))')
    for row in cur_false_right:
        row[7] = row[0]
        cur_false_right.updateRow(row)
    arcpy.CopyFeatures_management(file,file_out) # "B_LeftRightTopology_Correct.shp"
    return file_out

if __name__ == '__main__':
    file = arcpy.GetParameterAsText(0) # "B_LeftRightTopology.shp"
    file_out = arcpy.GetParameterAsText(1)
    CorrectTopology(file,file_out)
```

A5 BuildBends

```
# Autorin: Christiane Enderle
# Datum: 10.12.2018
# Fuer ArcGIS Desktop 10.3.1 mit Python 2.7.8

# Das Python-Skript-Tool ist nur im Rahmen des Modells, das den Algorithmus von Ai et al.
(2016) implementiert, sinnvoll einsetzbar.
# Es identifiziert Bendsysteme und darin eingeschlossene untergeordnete bends, erfasst sie
als Liste von vertices der Originallinie,
# bringt sie in eine Hierarchie und bestimmt den jeweiligen bottom point,
# bend width, bend depth und bend size (nach Ai et al. (2016), S. 302).
# Die in Segmente aufgeloesten Dreiecke werden mit der zugehoerigen BSID (Feature-ID des
Bendsystems) gespeichert: "C_Triangle_segments_BSID.shp"
# Die Bendsysteme erhalten ein Attribut "Topology", das ihre Lage links oder rechts von der
Originallinie speichert: "C Bendsystems Topo.shp"
# Die Pfade, deren Laenge die bend depth angibt, werden als Skelettpfade ausgegeben:
"C_Skelettpaths.shp"
# Abweichend von Ai et al. werden die bend systems nicht als binary tree, sondern in Listen
abgelegt.

import arcpy
arcpy.env.overwriteOutput = 1

input_line = arcpy.GetParameterAsText(0) # Originallinie
input_bendsystems = arcpy.GetParameterAsText(1) # "B_Bend_Systems.shp"
input_triangles = arcpy.GetParameterAsText(2) # "B Triangles with Points.shp"
input_leftright = arcpy.GetParameterAsText(3) # "B LeftRightTopology Correct.shp"
input_points = arcpy.GetParameterAsText(4) # "B_Original_Points_densified.shp"
output_tri_BSID_seg = arcpy.GetParameterAsText(5) # "C_Triangle_segments_BSID.shp"
output_skelettpfade = arcpy.GetParameterAsText(6) # "C_Skelettpaths.shp"
output_BS_Topo = arcpy.GetParameterAsText(7) # "C_Bendsystems_Topo.shp"
worksp = arcpy.GetParameterAsText(8)
arcpy.env.workspace = worksp

# Kopiere die Bendsystems, damit sie im Cursor curBS ein weiteres Mal verwendet werden
koennen:
arcpy.CopyFeatures_management(input_bendsystems, "C_Bend_Systems_copy.shp")

# Erstelle einen Layer aus den Punkten, um spaeter eine raumbezogene Auswahl treffen zu
koennen:
arcpy.MakeFeatureLayer_management(input_points, "input_points_Layer")

# Erstelle einen Layer aus den Dreiecken, um spaeter eine Auswahl nach Attributen treffen zu
koennen:
arcpy.MakeFeatureLayer_management(input_triangles,"B_Triangles_Layer")

# Erstelle eine Liste fuer die main bend systems (jedes item dieser Liste ist eine bendList
# bestehend aus 1 main bend mit seinen nachgeordneten bends).
# Das ist notwendig, um die items (bendList) nach Hierarchiestufe sortieren zu koennen fuer
die Berechnung der Skelettpfade.
# Die Begriffe main bend system und bend system werden synonym verwendet.
list_bendsystems = [] # Die items dieser Liste sind bend systems.

# Bend systems ihrerseits werden nach dem Muster
[[BSID,TID,BT,PList,Length,HS],[BSID,TID,BT,PList,Length,HS],...]
# als verschachtelte Liste "bendList" angelegt.
# Die items dieser Liste sind bends.
# BSID: FeatureID des bend systems
# TID: FeatureID des lokalen Dreiecks, das in den bend (Kurvenabschnitt) fuehrt (Typ 1, Typ
5 und im Fall eines nachgeordneten bends Typ 4)
# BT: Typ des bends: M fuer main bend (die durch das Schneiden der Originallinie mit der
Delaunay-Domain entstanden sind),
#                         L fuer einen Kurvenabschnitt, der vom linken Schenkel des
Verzweiungsdreiecks ausgeht,
#                         R fuer einen Kurvenabschnitt, der vom rechten Schenkel des
Verzweiungsdreiecks ausgeht.
#                         Im Nachhinein hat sich herausgestellt, dass die Unterscheidung in
L und R nicht weiter verwendet wird.
# PList: zugehoerige Punktliste mit den ORIG_FID (Original Feature-ID) derjenigen
```

```
Linienpunkte, die zu dem bend gehoeren.
# Length: Laenge der Punktliste
# HS: Hierarchiestufe des bends

# Ein Cursor fuer die bend systems:
curBS = arcpy.da.SearchCursor(input_bendsystems, ["Shape@","FID"])
for BS in curBS:
    BSID = BS[1]
    # Erstelle eine Liste nur fuer die ORIG_FIDs der Punkte des aktuellen main bends bzw.
    leere die Liste nach dem Schleifendurchlauf:
    BSpointList = []
    # Erstelle eine Liste fuer das aktuelle main bend system mit seinen nachgeordneten bends
    bzw. leere die Liste nach dem Schleifendurchlauf:
    bendList = []
    # Suche das zugehoerige Muendungsdreieck (Typ 1 oder Typ 5) des aktuellen main bends.
    # where_clause fuer arcpy.SelectLayerByAttribute_management muss vorab als string in
    eine Variable geschrieben werden,
    # um dynamisch auf die Variable BSID zugreifen zu koennen. Quelle: Harrison, F.,
    PolyGeo, Marcin & blah238 (2012-2015):
    # Including variable in where clause of arcpy.Select_analysis()?.-
    https://gis.stackexchange.com/questions/27457/including-variable-in-where-clause-of-arcpy-
    select-analysis (Stand: 3.11.2015) (Zugriff: 13.12.2018)
    whereclause = '("BS_FID" = ' + str(BSID) + 'and "Type" = 1) or ("BS_FID" = ' + str(BSID)
    + 'and "Type" = 5)'
    arcpy.SelectLayerByAttribute_management("B_Triangles_Layer","NEW_SELECTION",whereclause)
    # Die FID des Muendungsdreiecks wird ausgelesen. Im folgenden Cursor duerfte jetzt nur
    das Muendungsdreieck sein:
    with arcpy.da.SearchCursor("B_Triangles_Layer","FID") as cursor: # Dieser Cursor loescht
    sich von selbst wieder, wenn die Schleife durchlaufen ist.
        for row in cursor:
            TID = row[0]
    # Die folgende Liste MB ist das erste item in der bendList des aktuellen main bend
    systems.
    # Sie speichert zunaechst nur die FID des aktuellen main bend systems (BSID), die ID des
    Muendungsdreiecks (TID) sowie den bend-Typ:
    MB = [BSID,TID,'M'] # Main bend
##    print "MB = ", MB # zu Testzwecken
    # Ermittle die Punktliste des aktuellen main bends per Tool Intersect.
    FID_clause = "FID = "+ str(BSID)
    arcpy.MakeFeatureLayer_management("C_Bend_Systems_copy.shp", "Bend_Systems_copy_Layer",
    FID_clause)
    arcpy.SelectLayerByLocation_management("input_points_Layer","INTERSECT",
    "Bend_Systems_copy_Layer","1.0 Meters","NEW_SELECTION")
    with arcpy.da.SearchCursor("input_points_Layer", "ORIG_FID") as curP:
        for p in curP:
            pnt = p[0] # ORIG_FID des aktuellen Punktes
            BSpointList.append(pnt)
    # Die Punktliste wird sortiert:
    BSpointList.sort()
    # Die Punktliste wird an die Liste MB angehaengt:
    MB.append(BSpointList)
    # Die Laenge der Punktliste des main bends wird ermittelt und an die Liste MB angehaengt:
    lenBSM = len(BSpointList)
    MB.append(lenBSM)
    # Die Hierarchiestufe wird an die Liste angehaengt:
    MB.append(0)
    bendList.append(MB)

# Hierarchie der nachgeordneten bends im bend system bestimmen
# Erstens: die bends im oben beschriebenen Schema erfassen: [BSID,TID,BT,PList,Length,HS].
    HS = 0
    # Ein Cursor fuer die Verzweigungsdreiecke (Typ 4 oder Typ 5) des aktuellen main bend
    systems:
    bs_clause = '("BS_FID" = ' + str(BSID) + 'and "Type" = 4) or ("BS_FID" = ' + str(BSID) +
    'and "Type" = 5)'
    curT = arcpy.da.SearchCursor(input_triangles, ["BS_FID","FID","Type","Point1","Point2",
    "Point3"], bs_clause)
    for t in curT:
```

```
        # Die ID des aktuellen Verzweigungsdreiecks und die ORIG_FID seiner Eckpunkte werden
        ausgelesen:
        TID = t[1]
        TP1 = t[3]
        TP2 = t[4]
        TP3 = t[5]
        # Die bends werden ebenfalls nach dem obigen Schema aufgebaut:
        # bend = [BSID des Hauptbendsystems, TID des Dreiecks, L bzw. R fuer links bzw.
        rechts liegenden Kurvenabschnitt,
        #         Liste bendL bzw. bendR, Laenge der Punktliste, Hierarchiestufe wird mit 0
        belegt]
        # Punktlisten der bends erstellen. Jedes Verzweigungsdreieck hat zwei child-bends,
        die sich am mittleren Punkt des Verzweigungsdreiecks scheiden.
        # Demnach besteht die Punktliste eines bends aus allen Punkten Point1 bis Point2
        bzw. Point2 bis Point 3.
        # Die Punktlisten der bends werden durch iteratives Erhoehen der ORIG_FID zwischen
        Punkt 1 und Punkt 2 bzw. Punkt 2 und Punkt 3 des Verzweigungsdreiecks generiert.
        # bendL = [ORIG_FID von Punkt 1, ...Zwischenpunkte..., ORIG_FID von Punkt 2]
        # bendR = [ORIG_FID von Punkt 2, ...Zwischenpunkte..., ORIG_FID von Punkt 3]
        bendL = [TP1]
        bendR = [TP2]
        bend = [BSID,TID]
        for FID in range(TP1,TP2): # Der letzte Wert in range wird grundsaetzlich nicht mehr
        verarbeitet. Deshalb darf im range hier TP2 am Ende stehen, nicht TP2-1.
            FID = FID + 1
            bendL.append(FID)
        length = len(bendL)
        bend.append('L')
        bend.append(bendL)
        bend.append(length)
        bend.append(HS)
        # bend wird an die bendList des aktuellen main bend systems angehaengt:
        bendList.append(bend)
        # bend muss fuer die Konstruktion des rechtsliegenden Kurvenabschnitts
        zurueckgesetzt werden:
        bend = [BSID,TID]
        for FID in range(TP2,TP3): # TP2 und TP3 koennen nie zwei aufeinanderfolgende
        Zahlenwerte sein, weil ein Verzweigungsdreieck kein gemeinsames Segment mit der
        Originallinie hat.
            FID = FID + 1 # Der letzte Wert in range wird grundsaetzlich nicht mehr
            verarbeitet. Deshalb darf im range hier TP3 am Ende stehen, nicht TP3-1.
            bendR.append(FID)
        length = len(bendR)
        bend.append('R')
        bend.append(bendR)
        bend.append(length)
        bend.append(HS) # HS = 0
        # bend wird wiederum an die bendList des aktuellen main bend systems angehaengt:
        bendList.append(bend)
    # Der Dreiecke-Cursor kann geloescht werden, weil alle Dreiecke des Typs 4 nun mit ID
    und zugehoeriger Punktliste in bendList gespeichert sind.
    del curT
# Zweitens: die bends in eine Hierarchie bringen.
# Grundidee: ein bend hoeherer Hierarchiestufe enthaelt alle Punkte der untergeordneten
Hierarchiestufen.
    # Für Vergleich der Punktlisten wird bendList kopiert:
    bendListC = bendList[:]
    for a in bendList: # a steht fuer ein bend und hat die Form [BSID, TID, BT, PList,
    Length, HS]
        HS = 0
        for aC in bendListC:
            # Die zu vergleichenden Punktlisten werden in Variablen gespeichert:
            PL = a[3]
            PLC = aC[3]
            # Es wird geprueft, ob die zu vergleichenden Listen gleich sind. In dem Fall
            kann in den naechsten Durchlauf gesprungen werden (continue).
            # Das kann vorkommen im Falle, dass das Originalelement mit dem entsprechenden
            kopierten Element verglichen wird.
```

```
            # Dieser Fall kann nicht ueber die sich gleichenden Dreiecks-IDs ausgeschaltet
            werden.
            if PL == PLC:
                continue
            # Es wird ermittelt, ob PL eine Teilmenge von PLC ist: Mit Sets. Quelle:
            https://www.python-forum.de/viewtopic.php?t=8714 (Stand: 9.1.2007)(Zugriff:
            15.12.2018)
            schnitt = set(PLC) & set(PL)
            if schnitt == set(PL):
                # Wenn es eine Teilmenge ist, wird die Hierarchiestufe um 1 erhoeht.
                # Dies geschieht mit jedem Schleifendurchlauf aufs neue, wenn die Pruefung
                ergibt, dass PL eine Teilmenge von PLC ist.
                HS = HS + 1 # HS = 1
        # Ist der Vergleich mit allen Elementen der Vergleichsliste abgeschlossen,
        # wird die resultierende Hierarchiestufe in die Liste des bends 'a' geschrieben.
        a[5] = HS
    # Die bendList ist komplett und wird der Liste aller bend systems hinzugefuegt.
    list_bendsystems.append(bendList)
# Der Cursor curBS wird geloescht:
del curBS
# Die Auswahl im "B_Triangles_Layer" wird aufgehoben, damit der Layer im folgenden wieder
komplett zur Verfuegung steht:
arcpy.SelectLayerByAttribute_management("B_Triangles_Layer","CLEAR_SELECTION")

#-------------------------------------------------------------------------------------------
----------
# Berechnung der Eigenschaften der bends: bend width, bend depth, bend bottom, bend size
#-------------------------------------------------------------------------------------------
----------
# Berechnung von bend width (Laenge der Aussenseite des Muendungs- bzw. Verzweigungsdreiecks).
# Das ist die Distanz zwischen PList[0] und PList[-1], d.h. zwischen dem ersten und letzten
Punkt eines bends:
# Erstelle einen Cursor fuer den Zugriff auf die ORIG_FID und die Geometrie der Punkte.
# Ueber die ORIG_FIDs sind die Punkte sowohl in den Listen als auch im Datensatz
identifizierbar.
arcpy.MakeFeatureLayer_management(input_points,"Densified_Points_Layer")
# Zwei Variablen fuer die Kennzeichnung von Treffern bei der Suche nach dem ersten und
letzten Punkt von PList im "Densified_Points_Layer":
treffer1 = 0
treffer2 = 0
for list in list_bendsystems: # list ist ein main bend system mit allen nachgeordneten bends.
    for a in list: # a ist ein bend im main bend system. a hat die Form: [BSID, TID, BT,
    PList, Length, HS]
        PList = a[3]
        with arcpy.da.SearchCursor("Densified_Points_Layer", ["Shape@","ORIG_FID"]) as cursor
        :
            for p in cursor:
                if p[1] == PList[0]:
                    geom1 = p[0]
                    treffer1 = 1
                elif p[1] == PList[-1]:
                    geom2 = p[0]
                    treffer2 = 1
                if treffer1 == 1 and treffer2 == 1:
                    # Die Distanz zwischen erstem und letztem Punkt von PList wird durch die
                    Methode des Geometrie-Objekts "angleAndDistanceTo" ermittelt:
                    property = geom1.angleAndDistanceTo(geom2) # property ist ein Tuple aus
                    Winkel und Distanz zwischen dem ersten und dem letzten Punkt von PList.
                    bendwidth = property[1]
                    a.append(bendwidth)
##                    print "bend mit bendwidth: ",a # zu Testzwecken
                    treffer1 = 0
                    treffer2 = 0
                    break

# Berechnung von bend depth (laengster Skelettpfad im bend):
# Erstelle ein Shapefile fuer die Skelettpfade (zunaechst zum Teil nur Teilstuecke):
arcpy.CreateFeatureclass_management(worksp, "C_skelettpfade_teilstuecke.shp","Polyline","",""
```

```
,"",input_line)
# Fuege C_skelettpfade_teilstuecke.shp die Attribute BSID, TID, BT, HS, LENGTH und die
ORIG_FID des bottom points (BOPO) hinzu.
# LENGTH soll die Laenge des skelettpfades speichern.
arcpy.AddField_management("C_skelettpfade_teilstuecke.shp","BSID","SHORT")
arcpy.AddField_management("C_skelettpfade_teilstuecke.shp","TID","SHORT")
arcpy.AddField_management("C_skelettpfade_teilstuecke.shp","BT","TEXT","","",1)
arcpy.AddField_management("C_skelettpfade_teilstuecke.shp","HS","SHORT")
arcpy.AddField_management("C_skelettpfade_teilstuecke.shp","LENGTH","DOUBLE")
arcpy.AddField_management("C_skelettpfade_teilstuecke.shp","BOPO","LONG")
# Wandle die Delaunay-Dreiecke in Liniensegmente um.
# neighbors sollen mit FID aufgezeichnet werden.
arcpy.PolygonToLine_management(input_triangles, "C_Triangle_segments.shp", 1)
# Bei PolygonToLine gehen alle Attribute der Dreiecke verloren.
# Es werden aber die BSID und ein Attribut gebraucht, um doppelte Liniensegmente zu loeschen.
# Um die Attribute wieder hinzuzufuegen, werden die Liniensegmente mit den Dreiecken
verschnitten:
arcpy.Intersect_analysis([["C_Triangle_segments.shp",2],[input_triangles,1]],
"Triangle_BSID_segments.shp") # HIER PASSIEREN LEICHT UNGEAUIGKEITEN !!!
# Die Liniensegmente benachbarter Dreiecke sind doppelt vorhanden.
# Doppelt vorhandene Liniensegmente werden geloescht:
arcpy.DeleteIdentical_management("Triangle_BSID_segments.shp","FID_C_Tria")
# Die Dreiecksegmente, die auf der Originallinie liegen, werden fuer den Skelettpfad nicht
benoetigt und werden deshalb geloescht:
with arcpy.da.UpdateCursor("Triangle_BSID_segments.shp","SHAPE@") as cursor1:
    for x1 in cursor1:
        trisegment = x1[0]
        with arcpy.da.SearchCursor(input_line,"SHAPE@") as cursor2:
            for x2 in cursor2:
                linesegment = x2[0]
                if trisegment.within(linesegment):
                    cursor1.deleteRow()
# Berechne die Mittelpunktkoordinaten der verbliebenen Dreiecksseiten:
arcpy.AddGeometryAttributes_management("Triangle_BSID_segments.shp","LINE_START_MID_END",
"METERS","","")
# Generiere die Mittelpunkte der Dreiecksseiten.
# Vor der Schleife werden noch die Felder fuer den SearchCursor fuer die Dreieckssegmente in
# Triangle_BSID_segments.shp bzw. Triangle_BSID_segments_copy (Layer) in eine Variable
geschrieben:
fields_segments1 = ["SHAPE@XY","LEFT_FID", "RIGHT_FID", "MID_X", "MID_Y"]
fields_segments2 = ["LEFT_FID", "RIGHT_FID", "MID_X", "MID_Y"]
# Der Datensatz mit den Segmenten der Dreiecke wird dupliziert, um ihn verschachtelt zu
iterieren:
arcpy.CopyFeatures_management("Triangle_BSID_segments.shp","Triangle_BSID_segments_copy.shp")
arcpy.MakeFeatureLayer_management("Triangle_BSID_segments_copy.shp",
"Triangle_BSID_segments_copy")
for list in list_bendsystems: # Die Liste mit den kompletten bendsystems wird iteriert.
    # Die Mittelpunkte sollen in der Reihenfolge der Hierarchiestufen (HS) erstellt werden,
    vom hoechsten HS-Wert (= unterste Hierarchiestufe)
    # zum kleinsten HS-Wert (0) (hoechste Hierarchiestufe).
    # Nur main bends haben HS = 0. Ihr Skelettpfad wird (mit Ausnahme des ersten, von der
    Aussenseite des Muendungsdreicks startenden Teils)
    # aus den Skelettpfaden der nachgeordneten bends zusammengesetzt.
    # Deshalb muessen die bendLists zunaechst nach dem HS-Wert absteigend sortiert werden.
    list.sort(key=lambda x: int(x[5]), reverse=True) # Quelle: Gray, D., Taymon,
    inspectorG4dget (2013): Python - How to sort a list of lists by the fourth element in
    each list? [duplicate].
    https://stackoverflow.com/questions/17555218/python-how-to-sort-a-list-of-lists-by-the-fou
    rth-element-in-each-list, (last modified: 2.12.16) (last access: 18.12.2018)
    for a in list: # a ist ein bend im main bend system. a hat die Form: [BSID, TID, BT,
    PList, Length, HS, BW]. BW = bend width
##        print a # fuer Testzwecke
        # Ein Array fuer die Punkte des Skelettpfades wird angelegt:
        PathPoints = arcpy.Array()
        PL = a[3] # Die Punktliste PList des aktuellen bends wird aufgerufen.
        # Auswahlbedingung fuer den folgenden cursor5 ist: die ORIG_FID des Punktes im
        cursor5 muss
        # der ORIG_FID des ersten Punktes in der Punktliste des bends a entsprechen:
```

```
clause1 = '"ORIG_FID" ='+str(PL[0])
with arcpy.da.SearchCursor(input_points, ["Shape@X","Shape@Y","ORIG_FID"], clause1)
as cursor5:
    for p in cursor5:
        start_X = p[0]
        start_Y = p[1]
# Auswahlbedingung fuer den folgenden cursor6 ist: die ORIG_FID des Punktes im
cursor6 muss
# der ORIG_FID des letzten Puntkes in der Punktliste des bends a entsprechen:
clause2 = '"ORIG_FID" ='+str(PL[-1])
with arcpy.da.SearchCursor(input_points, ["Shape@X","Shape@Y","ORIG_FID"], clause2)
as cursor6:
    for p in cursor6:
        ende_X = p[0]
        ende_Y = p[1]
# Errechne die Koordinaten des Mittelpunktes zwischen erstem und letztem Punkt in
PList:
Centro_X = abs(start_X + ende_X)/2
Centro_Y = abs(start_Y + ende_Y)/2
# Eine Variable, um die Schleife durch die Dreiecksegmente vorzeitig beenden zu
koennen, wenn ein bend sich verzweigt.
control = 0
# Die Dreieckssegmente werden im folgenden "Quersegmente" genannt, weil
Triangle_BSID_segments.shp nur noch Segmente enthaelt,
# die nicht auf der Originallinie liegen, sondern quer durch das bend verlaufen.
with arcpy.da.SearchCursor("Triangle_BSID_segments.shp",fields_segments1) as cursor1:
 # fields_segments1 = ["SHAPE@XY","LEFT_FID", "RIGHT_FID", "MID_X", "MID_Y"]
    # Bestimme das erste Quersegment des bends.
    # Sein Mittelpunkt (MidX,MidY) muss mit der Mitte zwischen erstem und letztem
    Punkt aus PList (Centro_X,Centro_Y) übereinstimmen.
    # Diese umstaendliche Vorgehensweise wurde notwendig, weil die raeumlichen
    Abfragen der arcpy-Geometrie-Objekte keine zuverlaessigen Ergebnissse lieferten.
    for segment in cursor1:
        MidX = segment[3]
        MidY = segment[4]
        if abs(Centro_X-MidX)<0.001 and abs(Centro_Y-MidY)<0.001:
            PathPoints.append(
                arcpy.Point(MidX,MidY)) # Ein Punktobjekt wird aus den Koordinaten
                erstellt und an einen Array der PathPoints fuer den Skelettpfad
                angehaengt.
            # Das aktuelle Quersegment trennt zwei Dreiecke:
            # das Verzweigungsdreieck, gekennzeichnet durch TID, das in das aktuelle
            bend fuehrt,
            # und das erste Fuelldreieck. Eines der Dreiecke hat die LEFT_FID, das
            andere die RIGHT_FID (LEFT_FID und RIGHT_FID sind Attribute der
            Quersegmente).
            # Die TID des Verzweigungsdreiecks steht in der aktuellen Liste a.
            # Um das naechste Quersegment fuer den naechsten Mittelpunkt zu finden,
            # wird nicht die ID des ausserhalb liegenden Verzweigungsdreiecks
            benoetigt,
            # sondern diejenige des ersten Fuelldreiecks. Diese wird mit dem
            folgenden if-statement ermittelt.
            # LEFT-FID oder RIGHT_FID sind -1, wenn das Quersegment die Aussenseite
            eines Muendungsdreiecks ist.
            # -1 kennzeichnet die gesamte Region ausserhalb der Domain der
            Triangulation.
            if segment[1] == -1 or segment[2] == -1: # Falls es sich also um ein
            Muendungsdreieck handelt:
                id = a[1]
                vorige = -1
            elif segment[1] == a[1] and segment[2] != -1:
                id = segment[2]
                vorige = a[1]
            elif segment[2] == a[1] and segment[1] != -1:
                id = segment[1]
                vorige = a[1]

            # Suche das Quersegment, das zwischen dem aktuellen und dem folgenden
```

```
Dreieck liegt,
# d.h. es muss die voranstehende id im Datensatz besitzen, nicht aber
die id des Verzweigungsdreiecks ("vorige"):
select_clause = '("LEFT_FID" = ' + str(id) + 'and not "RIGHT_FID" = ' +
str(vorige) + ') or ("RIGHT_FID" = ' + str(id) + 'and not "LEFT_FID" = '
+ str(vorige) + ')'
sel = arcpy.SelectLayerByAttribute_management(
"Triangle_BSID_segments_copy","NEW_SELECTION",select_clause)
result = arcpy.GetCount_management(sel)
count = int(result.getOutput(0))
# Mit der folgenden while-Schleife wird durch die Quersegmente des
aktuellen bends a iteriert, bis der bottom erreicht ist
# (dann ist count = 0, weil es zu "vorige" keine passende id mehr gibt).
while count > 0:
    # Wenn 2 Quersegmente gefunden werden, dann gehoeren sie zu einem
    Verzweigungsdreieck.
    # Deshalb enstehen nun zwei Teilstuecke fuer zwei Skelettpfade, die
    in die beiden child-bends laufen.
    # Gemaess Ai et al. (2016), Fig. 5, S. 302, wird zunaechst der
    Centroid des Verzweigungsdreiecks eingefuegt.
    # Der naechste Punkt ist der Mittelpunkt zunaechst des einen, dann
    des anderen naechsten Quersegments.
    # Er bildet jeweils den letzten Punkt in diesem Skelettpfad, der
    jetzt nur ein Teilstueck darstellt.
    # und spaeter mit anderen Teilstuecken zusammengesetzt wird.
    if count == 2:
        # Die Variable id hat als aktuellen Wert die FID des
        Verzweigungsdreiecks.
        clause8 = '"FID" =' + str(id) # Formulierung der clause fuer den
        folgenden cursor8.
        # Der folgende cursor8 enthaelt nun nur die beiden zu den
        inneren bends hinweisenden Quersegmente des Verzweigungsdreiecks.
        with arcpy.da.SearchCursor("B_Triangles_Layer",["Shape@","FID",
        "Type"],clause8) as cursor8:
            for row in cursor8:
                geom = row[0]
                type = row[2]
                # centroid ist eine property von geom und gibt ein
                Punkt-Objekt zurueck
                TIDcentroid = geom.centroid
                PathPoints.append(TIDcentroid)
                # Die folgende Variable countdown kontrolliert, dass
                LENGTH und BOPO nur einmal an die aktuelle Liste a
                angehaengt werden.
                # Die Liste a repraesentiert das aktuelle bend.
                # In einem bend, das sich spaeter noch verzweigt, treten
                zwei Teilstuecke von Skelettpfaden auf.
                # Erst spaeter im Skript, nach dem Zusammensetzen der
                Teilstuecke, wird die bend-depth
                # (d.h. die Laenge des laengsten Skelettpfades im bend)
                ermittelt und in die Liste a eingetragen.
                # Deshalb steht zu diesem Zeitpunkt auch der bottomPoint
                BOPO noch nicht fest.
                # Daher wird fuer das aktuelle bend a genau einmal die
                vorlaeufige Laenge 0 und als vorlaeufiger BOPO der Wert
                -1 eingetragen.
                countdown = 1
                with arcpy.da.SearchCursor(sel,fields_segments2) as
                cursor2: # fields_segments2 = ["LEFT_FID", "RIGHT_FID",
                "MID_X", "MID_Y"]
                    for segm in cursor2:
                        MidX = segm[2]
                        MidY = segm[3]
                        PathPoints.append(
                            arcpy.Point(MidX,MidY)) # Der Mittelpunkt
                            des Quersegments wird an die Liste der
                            PathPoints angehaengt.
                        skelettpfad = arcpy.Polyline(PathPoints) # Das
```

```
                        Teilstueck des Skelettpfades wird gebildet.
                        BSID = a[0]
                        TID = a[1]
                        BT = a[2]
                        HS = a[5]
                        LENGTH = 0
                        BOPO = -1
                        if countdown == 1: # Wenn countdown = 1, d.h.
                        beim ersten Durchlauf durch diese Schleife mit 2
                        Quersegmenten, ...
                            a.append(LENGTH) # dann werden die
                            vorlaeufigen Werte fuer Laenge und BOPO an
                            die Liste a angehaengt.
                            a.append(BOPO)
                            countdown = 0 # countdown = 0 besagt, dass
                            eines der Quersegmente abgearbeitet ist.
                        # Speichere das Teilstueck mit seinen
                        bend-Attributen im Shapefile:
                        bone = arcpy.da.InsertCursor(
                        "C_skelettpfade_teilstuecke.shp", ["Shape@",
                        "BSID","TID","BT","HS","LENGTH","BOPO"])
                        bone.insertRow([skelettpfad, BSID, TID, BT, HS,
                        LENGTH, BOPO])
                        del bone
                        # Der letzte Punkt wird aus PathPoints entfernt,
                        um an die bestehende Liste den Mittelpunkt des
                        anderen
                        # Quersegments anzuhaengen.
                        num = PathPoints.count - 1

                        PathPoints.remove(num)
            # Die Control-Variable wird hier noch einmal benutzt, um
            vorzeitig aus der aeusseren Schleife durch die Quersegmente
            auszusteigen.
            # Denn mit den inneren Segmenten eines Verzweiungsdreiecks
            beginnen zwei neue bends.
            # Deshalb muss jetzt mit der Schleife durch die bends a
            fortgefahren werden.
            # (siehe unten nach dem else-Block).
            control = 1
            count = 0

        else:
            # Wenn nur ein Quersegment gefunden wurde, wird dessen
            Mittelpunkt der Liste PathPoints hinzugefuegt,
            # und die Variablen der LEFT_FID und RIGHT_FID -- id und vorige
            -- fuer den naechsten Durchlauf der while-Loop aktualisiert.
            with arcpy.da.SearchCursor(sel,fields_segments2) as cursor2:
                for segm in cursor2:
                    MidX = segm[2]
                    MidY = segm[3]
                    PathPoints.append(
                        arcpy.Point(MidX,MidY))
                    if id == segm[0]:
                        vorige = id
                        id = segm[1]
                    else:
                        vorige = id
                        id = segm[0]
            # Eine neue Auswahl muss am Ende der while-Loop getroffen
            werden, um einen neuen Wert fuer count zu erhalten,
            # der darueber entscheidet, ob die while-Loop ein weiteres Mal
            durchlaufen wird oder nicht.
            select_clause = '("LEFT_FID" = ' + str(id) + 'and not
            "RIGHT_FID" = ' + str(vorige) + ') or ("RIGHT_FID" = ' + str(id)
            + 'and not "LEFT_FID" = ' + str(vorige) + ')'
            sel = arcpy.SelectLayerByAttribute_management(
            "Triangle_BSID_segments_copy","NEW_SELECTION",select_clause)
```

```
                result = arcpy.GetCount_management(sel)
                count = int(result.getOutput(0))

        if control == 1:
            break
        # Wenn es zu einer id keine weitere LEFT_ID oder RIGHT_ID gibt,
        # ist die Schleife am bend bottom angekommen.
        # Der letzte Wert von id ist die ID eines Endedreiecks (Typ 2).
        # Dieses Dreieck wird nun per cursor3 im Datensatz der Dreiecke gesucht.
        # Dabei wird cursor3 auf Dreiecke des Typs 2 beschraenkt bzw. auch Typ 1
        ist zugelassen,
        # denn wenn ein bendsystem aus nur einem Dreieck besteht, ist das
        Typ1-Dreieck gleichzeitig das Endedreieck:
        clause3 = '"Type" = 2 or "Type" = 1'
        with arcpy.da.SearchCursor(input_triangles, ["FID","Type","Point2"],
        clause3) as cursor3:
            for triangle in cursor3:
                if triangle[0] == id:
                    # Point2 als Attribut des Dreiecks ist die ORIG_FID des
                    Punkts in der Spitze des Dreiecks und damit der bottom point
                    des bends:
                    bottomPoint = triangle[2]
                    # Die Koordinaten des bottom points werden mithilfe eines
                    cursors aus dem Punktdatensatz ausgelesen:
                    clause = '"ORIG_FID" = ' + str(bottomPoint)
                    with arcpy.da.SearchCursor(input_points, ["Shape@XY",
                    "ORIG_FID"], clause) as cursor4:
                        for xy in cursor4:
                            coordinates = xy[0]
                            # Aus den Koordinaten des bottom points wird ein
                            Punktobjekt erstellt und dieses an die PathPoints
                            angehaengt.
                            PathPoints.append(
                                arcpy.Point(coordinates[0],coordinates[1]))
                            # Der Skelettpfad wird aus den Punktobjekten im
                            PathPoints-Array generiert und an die Liste der
                            Skelettpfade angehaengt.
                            skelettpfad = arcpy.Polyline(PathPoints)
                            BSID = a[0]
                            TID = a[1]
                            BT = a[2]
                            HS = a[5]
                            # Fuer bends ohne Verzweigungen ist der aktuelle
                            Skelettpfad der laengste (weil es keinen anderen
                            gibt).
                            # Seine Laenge entspricht damit der bend-depth und
                            wird an das aktuelle bend a angehaengt:
                            LENGTH = skelettpfad.length
                            a.append(LENGTH)
                            a.append(bottomPoint) # Die ORIG_FID des bottom
                            points wird an das aktuelle bend a angehaengt.
                            bone = arcpy.da.InsertCursor(
                            "C_skelettpfade_teilstuecke.shp", ["Shape@","BSID",
                            "TID","BT","HS","LENGTH","BOPO"])
                            bone.insertRow([skelettpfad, BSID, TID, BT, HS,
                            LENGTH, bottomPoint])
                            del bone
        break # Beendet die aeussere Schleife durch die Quersegmente. Es wird
        mit dem naechsten bend a fortgefahren.

# Kopiere "Triangle_BSID_segments.shp" in eine Output-Datei, damit sie dem
Python-Script-Tool "EnvelopeZones.py" im Modell zur Verfuegung steht.
arcpy.CopyFeatures_management("Triangle_BSID_segments.shp",output_tri_BSID_seg)

# Zusammensetzen der Skelettpfade aus den Teilstuecken:
# Ausgehend von den Pfaden, die in einem bottomPoint enden, wird jeweils das Teilstueck
gesucht, das den vorigen Pfad beruehrt, aber eine hoehere Hierarchiestufe
# (= kleineren HS-Wert) besitzt. Ist es gefunden, wird es mit dem vorigen Pfad
```

```
zusammengefuegt und als neues Linienobjekt in ein Shapefile geschrieben.
# Der neue Skelettpfad wird zur Eingabegeometrie, zu der das folgende Teilstueck gesucht wird.

# Um verschachtelt durch die Skelettpfade bzw. Teilstuecke iterieren zu koennen, werden sie
separiert in:
# -> Teilstuecke der Skelettpfade oberhalb der niedrigsten Hierarchiestufen mit einen
bottomPoint -1.
# -> Skelettpfade der niedrigsten Hierarchiestufen, die in einem bottomPoint enden (sie
werden per SQL-Abfrage im cursor10 separiert).
# Die Teilsstuecke mit bottomPoint -1 werden ausgewaehlt und separat gespeichert:
arcpy.MakeFeatureLayer_management("C_skelettpfade_teilstuecke.shp",
"skelettpfade_teilstuecke_Layer1")
arcpy.SelectLayerByAttribute_management("skelettpfade_teilstuecke_Layer1","NEW_SELECTION",
'"BOPO" = -1')
arcpy.CopyFeatures_management("skelettpfade_teilstuecke_Layer1",
"C_skelettpfade_teilstuecke_copy.shp")
arcpy.SelectLayerByAttribute_management("skelettpfade_teilstuecke_Layer1","CLEAR_SELECTION")
# Die die Skelettpfade der niedrigsten Hierarchiestufen werden ausgewaehlt und in ein neues
Shapefile unter dem Parameter output_skelettpfade kopiert,
# in das mit einem InsertCursor dann auch die zusammengesetzten Teilstuecke geschrieben
werden:
arcpy.SelectLayerByAttribute_management("skelettpfade_teilstuecke_Layer1","NEW_SELECTION",
'"BOPO" <> -1')
arcpy.CopyFeatures_management("skelettpfade_teilstuecke_Layer1", output_skelettpfade)

# Die Skelettpfade, die auf den hoeheren Hierarchiestufen nur als Teilstuecke existieren,
werden zusammengesetzt.
# Der InsertCursor bone haengt die Ergebnisse an den Datensatz "skelettpfade" an.
bone = arcpy.da.InsertCursor(output_skelettpfade, ["Shape@","BSID","TID","BT","HS","LENGTH",
"BOPO"])
# Fuer die spaetere Schleife for i in range(0,HS_max) muss die Hierarchiestufe mit dem
hoechsten Wert ermittelt werden:
HS_value_list = []
for list in list_bendsystems:
    for a in list: # a ist ein bend im main bend system. a hat die Form: [BSID, TID, BT,
    PList, Length, HS, BW].
        HS_value = a[5]
        HS_value_list.append(HS_value)
HS_value_list.sort()
HS_max = HS_value_list[-1]

# Der folgende SearchCursor wird mit der SQL-Abfrage BOPO <> -1 eingeschraenkt,
# so dass er nur diejenigen Skelettpfade enthaelt, fuer die bereits ein BottomPoint
eingetragen ist.
# Die Attribute Geometrie, Hierarchiestufe und die ORIG_FID des BottomPoint werden benoetigt.
with arcpy.da.SearchCursor("C_skelettpfade_teilstuecke.shp", ["Shape@","HS","BOPO"], '"BOPO"
<> -1') as cursor10:
    for path in cursor10: # Ein Skelettpfad niedriger Hierarchie wird mit seinen Attributen
    eingelesen.
        geom1 = path[0]
        HS1 = path[1]
        BOPO = path[2]
        # Solange Teilstuecke einer hoeheren HS ( = niedrigerer HS-Wert) gefunden werden,
        # wird der im folgenden Code aus zwei Teilstuecken zusammengesetzte neue Skelettpfad
        zur neuen Eingabe-Geometrie:
        for i in range(0,HS_max):
            if HS1 != 0: # 0 ist die hoechste Hierarchiestufe und der kleinstmoegliche
            HS-Wert.
##                print "HS1 = ",HS1, "BOPO = ", BOPO # zu Testzwecken
                with arcpy.da.SearchCursor("C_skelettpfade_teilstuecke_copy.shp", ["Shape@",
                "BSID","TID","BT","HS"]) as cursor11:
                    for path_part in cursor11: # Mit diesem Cursor wird das angrenzende
                    Teilstueck gesucht.
                        geom2 = path_part[0]
                        BSID = path_part[1]
                        TID = path_part[2]
                        BT = path_part[3]
                        HS2 = path_part[4]
```

```
##                        if geom2.touches(geom1) and HS1 > HS2: # Diese raeummliche Abfrage
ist unzuverlaessig !!! Deshalb wird eine andere Methode gewaehlt:
                       if geom2.distanceTo(geom1) < 1.0 and HS1 > HS2:
                           skelettpfad = geom2.union(geom1)
                           LENGTH = skelettpfad.length
                           # Der neue Skelettpfad wird mit der errechneten Laenge und
                           weiteren Eigenschaften des bends
                           # in das Shapefile unter dem Parameter output_skelettpfade
                           geschrieben.:
                           bone.insertRow([skelettpfad, BSID, TID, BT, HS2, LENGTH, BOPO])
                           # Der neue Skelettpfad wird zur Eingabegeometrie, zu der im
                           naechsten Durchlauf (for i in range(0,HS_max))
                           # das naechste angrenzende Teilstueck gesucht wird:
                           geom1 = skelettpfad
                           # Die Hierarchiestufe HS1 wird auf HS2 heruntergesetzt.
                           HS1 = HS2
                           # Wenn ein passendes Teilstueck gefunden wurde, muss der
                           cursor11 verlassen werden,
                           # damit in der folgenden Suchschleife wieder alle Teilstuecke
                           zur Verfuegung stehen.
                           break
del bone

# Suche den laengsten Skelettpfad fuer jeden bend. Diese Laenge ist die bend-depth des bends.
# Wenn ein bend weitere Verzweigungen umfasst, koennen in output_skelettpfade fuer ein bend
mehrere Pfade mit verschiedener Laengen auftreten.
# Die 4 Attribute BSID, TID, BT und HS kennzeichnen in ihrer Kombination ein bend eindeutig.
# Es geht also darum, in der Gruppe von Skelettpfaden mit gleicher BSID, TID, BT und HS den
Skelettpfad mit der groessten Laenge zu finden.
# Erstelle eine Liste fuer die Laengen innerhalb einer Gruppe (um daraus dann das Maximum zu
ermitteln):
lengths = []
# Erstelle als Control-Variable ein set, dessen Elemente aus Text-Kombinationen
BSID_TID_BT_HS bestehen:
control_set = set()
# control_set wird mit allen in output_skelettpfade vorhandenen Kombinationen aus BSID, TID,
BT, HS gefuellt:
with arcpy.da.SearchCursor(output_skelettpfade, ["BSID","TID","BT","HS"]) as cursor15:
    for skelett in cursor15:
        BSID = skelett[0]
        TID = skelett[1]
        BT = skelett[2]
        HS = skelett[3]
        kombi = str(BSID)+'_'+str(TID)+'_'+str(BT)+'_'+str(HS)
        control_set.add(kombi)
# Kopiere output_skelettpfade -> C_skelettpfade_copy.shp
arcpy.CopyFeatures_management(output_skelettpfade,"C_skelettpfade_copy.shp")
while len(control_set) > 0: # Solange Elemente im control_set enthalten sind (Laenge > 0),
laeuft die while-Loop.
    # Erstelle einen Cursor fuer C_skelettpfade_copy.shp:
    with arcpy.da.SearchCursor("C_skelettpfade_copy.shp", ["BSID","TID","BT","HS"]) as
    cursor12:
        for path in cursor12:
            # Die folgenden 4 Attribute kennzeichnen in ihrer Kombination ein bend eindeutig:
            BSID = path[0]
            TID = path[1]
            BT = path[2]
            HS = path[3]
            kombi = str(BSID)+'_'+str(TID)+'_'+str(BT)+'_'+str(HS)
            # Mit clause13 wird der folgende cursor13 auf alle Objekte eines bends mit den
            aktuellen Werten fuer BSID,TID,BT,HS eingeschraenkt:
            clause13 = '"BSID" = '+ str(BSID) + ' and "TID" = ' + str(TID) + ' and "BT"' + "
            = '" + BT + "'" + ' and "HS" = ' + str(HS) # Quelle: Quinn, Sterling; Detwiler,
            Jim; Hardisty, Frank; O'Brien, James (2018): GEOG 485: GIS Programming and
            Software Development. 3.2.3 Retrieving records using an attribute query. -
            https://www.e-education.psu.edu/geog485/node/129, Stand: 2.1,2019, Zugriff:
            2.1.2019.
            with arcpy.da.SearchCursor(output_skelettpfade,["BSID","TID","BT","HS","LENGTH"],
```

```
            clause13) as cursor13:
                for selection in cursor13:
                    leng = selection[4] # Fuer jeden Skelettpfad des aktuellen bends wird
                    die Laenge ausgelesen.
                    lengths.append(leng) # Die Laengen werden in einer Liste gespeichert.
            lengths.sort() # Die Liste wird sortiert, um das Maximum (= die gesuchte
            bend-depth) zu ermitteln.
            leng_max = lengths[-1]
            lengths = [] # Die Liste wird geleert fuer den naechsten Durchlauf.
            with arcpy.da.UpdateCursor(output_skelettpfade,["BSID","TID","BT","HS","LENGTH",
            "BOPO"], clause13) as cursor16:
                for selection in cursor16:
                    leng = selection[4]
                    BOPO = selection[5] # BOPO wird fuer den Eintrag in die bendLists
                    benoetigt.
                    if leng != leng_max: # Aus output_skelettpfade werden alle Skelettpfade
                    des aktuellen bends, ausser dem laengsten, geloescht.
                        cursor16.deleteRow()
                    else: # Der laengste Skelettpfad im aktuellen bend wird mit seiner
                    Laenge und seinem bottomPoint BOPO in die entsprechende bendList
                    eingetragen:
                        for list in list_bendsystems:
                            for a in list:
                                if a[0] == BSID and a[1] == TID and a[2] == BT and a[5] == HS:
                                    a[7] = leng_max
                                    a[8] = BOPO
            # cursor12 wird hier abgebrochen, damit alle bends der aktuellen Kombination
            BSID,TID,BT,HS aus "C_skelettpfade_copy.shp" geloescht werden koennen,
            # damit sie beim naechsten Durchlauf nicht noch einmal gefunden werden:
            break
    # Aus C_skelettpfade_copy.shp werden alle bends der aktuellen Kombination BSID,TID,BT,HS
    (clause13) geloescht:
    with arcpy.da.UpdateCursor("C_skelettpfade_copy.shp", ["BSID","TID","BT","HS","LENGTH"],
    clause13) as cursor14:
        for row in cursor14:
            cursor14.deleteRow()
    control_set.remove(kombi) # Die aktuelle Kombination BSID,TID,BT,HS wird aus dem
    control_set geloescht.

# Berechne bend-size, d.h. die Flaeche, die durch ein bend eingeschlossen wird.
# Lies die Punktlisten der bends aus, trage den ersten Punkt ein weiteres Mal als letzten
Punkt ein, generiere daraus eine Flaechengeometrie,
# berechne den Flaecheninhalt und trage ihn fuer jedes bend ein.
# Lies die Punktlisten der bends aus:
for list in list_bendsystems:
    for a in list:
        PList = a[3]
        start = PList[0]
        PList.append(start) # Der erste Punkt von PList wird am Ende der PList angehaengt,
        damit daraus ein geschlossenes Polygon gebildet werden kann.
        PolygonPoints = arcpy.Array() # Ein Array fuer die Stuetzpunkte des bend-Polygons.
        for id in PList: # Die Punkte von PList werden in einer Schleife aus den
        input_points ausgelesen.
            clause17 = '"ORIG_FID" = ' + str(id) # Der folgende Cursor wird auf die aktuelle
            id eingeschraenkt.
            with arcpy.da.SearchCursor(input_points, ["Shape@XY","ORIG_FID"], clause17) as
            cursor17:
                for p in cursor17:
                    coordinates = p[0]
                    # Aus den Koordinaten des aktuellen Punktes wird ein Punktobjekt
                    erstellt und dieses an die PolygonPoints angehaengt.
                    PolygonPoints.append(
                        arcpy.Point(coordinates[0],coordinates[1]))
        # Wenn alle Punkte aus der PList des aktuellen bends a in PolygonPoints gespeichert
        sind, wird das Polygon erstellt:
        bend_area = arcpy.Polygon(PolygonPoints)
        bend_size = bend_area.area
        a.append(bend_size)
```

```
#-------------------------------------------------------------------------------------------------
----------
# Lage der bends rechts oder links der Linie
#-------------------------------------------------------------------------------------------------
----------
# Jedem bend a wird ein weiteres Element "L" oder "R" hinzugefuegt, je nachdem auf welcher
Seite der Linie es liegt.
# Aus dem Shapefile B_LeftRightTopology_Correct.shp (input_leftright) werden die BSIDs in
den Feldern "LeftOfLine" und "R_OfLine"
# ausgelesen und in Mengen (sets) gespeichert.
# Fuer jede Seite wird ein set angelegt:
left = set()
right = set()
# Der folgende Cursor liest die BSIDs in den beiden Feldern aus und speichert sie im
entsprechenden set:
with arcpy.da.SearchCursor(input_leftright, ["LeftOfLine","R_OfLine"]) as cursor18:
    for row in cursor18:
        BSID_L = row[0]
        BSID_R = row[1]
        left.add(BSID_L)
        right.add(BSID_R)
for list in list_bendsystems:
    for a in list:
        BSID = a[0]
        if BSID in left:
            a.append('L')
        elif BSID in right:
            a.append('R')
##        print a # zu Testzwecken

# Schreibe die Rechts-/Linkslage als Attribut in B_Bend_Systems.shp.
# Fuege B_Bend_Systems.shp ein Feld hinzu:
arcpy.AddField_management(input_bendsystems,"Topology","TEXT","","",1)
# Ein UpdateCursor fuer die Aktualisierung von B_Bend_Systems.shp:
with arcpy.da.UpdateCursor(input_bendsystems,["FID","Topology"]) as cursor:
    for bendsystem in cursor:
        BSID = bendsystem[0]
        for list in list_bendsystems:
            # list ist eine Liste mit allen bends eines bestimmten bend systems.
            # D.h. alle bends a in list haben in a[0] die gleiche BSID.
            # Deshalb genuegt es, fuer das erste item list[0] die BSID a[0] auszulesen und
            mit der BSID des aktuellen bend systems im cursor zu vergleichen.
            a = list[0]
            if a[0] == BSID: # Stimmen sie ueberein, wird dem bend system im cursor die
            Rechts- bzw. Linkslage eingetragen.
                bendsystem[1] = a[10]
                cursor.updateRow(bendsystem)
                break # Die Schleife for list in list_bendsystems kann abgebrochen werden.

# Kopiere B_Bend_Systems.shp in eine neue Datei, damit sie dem Python-Script-Tool
"EnvelopeZones.py" im Modell zur Verfuegung steht.
arcpy.CopyFeatures_management(input_bendsystems,output_BS_Topo)

### Zu Testzwecken:
##BS_textfile = open("2019_02_23_a bendLists.txt","w+")
##for list in list_bendsystems:
##    for a in list:
##        print a
##        BS_textfile.write(str(a)+'\n')
##BS_textfile.close()


# Loesche Zwischenergebnisse aus BuildBends:
del list_bendsystems
del BSpointList
del bendList
del MB
del bend
del bendListC
del a
del PL
del PList
del lengths
```

A6 EnvelopeZones

```
# Autorin: Christiane Enderle
# Datum: 4.1.2019
# Fuer ArcGIS Desktop 10.3.1 mit Python 2.7.8

# Das Python-Skript-Tool ist nur im Rahmen des Modells, das den Algorithmus von Ai et al.
(2016) implementiert, sinnvoll einsetzbar.
# Mit diesem Python-Skript-Tool beginnt die eigentliche Linienvereinfachung.
# In Abhaengigkeit von den Parametern epsilon (maximale bend depth, die wegfallen soll) und
# gamma (ein Faktor, mit dem epsilon multipliziert wird zur Auffindung charakteristischer
Punkte) werden Envelope-Zonen (Huellkurven) gebildet.
# Das Ergebnis sind drei Datensaetze, die fuer das folgende Skript
"GenerateSimplifiedLine.py" benoetigt werden:
# D_envelope_segments.shp: Die Segmente der Envelope-Zonen, die potenziell ein Teilstueck
der Originallinie ersetzen werden,
#                          mit ihrer Lage links bzw. rechts von der Originallinie
# D_envelope_zones_parcels.shp: Teilflaechen der Envelope-Zonen, die fuer die Konstruktion
von Mittelpfaden durch die Envelope-Zonen
#                               gebraucht werden. Sie wurden im Skript belassen, auch wenn
die Option der Mittelpfade ("centerlines")
#                               im Rahmen dieser Arbeit nicht fehlerfrei realisiert wurde.
# D_envelope_zones_leftright.shp: aus den Envelope-Zonen per Trennung durch die
Originallinie hervorgegangene Polygone
#                                mit dem Attribut der Lage links oder rechts von der
Originallinie

import arcpy
arcpy.env.overwriteOutput = 1

input_line = arcpy.GetParameterAsText(0) # Orignallinie nach der Triangulation
input_triangles = arcpy.GetParameterAsText(1) # "B_Triangles_with_Points.shp"
input_triangles_segments = arcpy.GetParameterAsText(2) # "C_Triangle_segments_BSID.shp"
input_points = arcpy.GetParameterAsText(3) # "B_Original_Points_densified.shp"
input_bendsystems = arcpy.GetParameterAsText(4) # "C_Bendsystems_Topo.shp"
input_skelettpfade = arcpy.GetParameterAsText(5) # "C_Skelettpaths.shp"
output_envelope_segments = arcpy.GetParameterAsText(6) # "D_envelope_segments.shp"
output_parcels = arcpy.GetParameterAsText(7) # "D_envelope_zones_parcels.shp" = Dreiecke,
die eine envelope zone bilden
output zones leftright = arcpy.GetParameterAsText(8) # "D envelope zones leftright.shp"
worksp = arcpy.GetParameterAsText(9)
arcpy.env.workspace = worksp


# Ein Generalisierungsparameter wird angegeben. Hier ein Schwellenwert fuer die bend-depth.
# Alle bends (Kurvenabschnitte), deren laengster Skelettpfad (bend-depth) kuerzer ist als
epsilon, werden verworfen.
epsilon = arcpy.GetParameterAsText(10)

# Optional kann der Parameter gamma angegeben werden, durch den charakteristische Punkte im
Linienverlauf erhalten bleiben
# (vgl. Ai et al. (2016), S. 303 ff.)
# gamma wird folgendermassen verwendet:
# epsilon_big = gamma * epsilon ## mit gamma > 1
# epsilon_big dient als zweiter Schwellenwert.
# bottomPoints von bends, deren Skelettpfad laenger ist als epsilon_big, sollen auf jeden
Fall erhalten bleiben.
# Damit koennen charakteristische Punkte in fraktal-artig gewundenen Linien identifiziert
werden und erhalten bleiben.
gamma = 1.0 # Falls der Nutzer nichts angibt, wird der Defaultwert 1 verwendet, d.h. es wird
nicht nach charakteristischen Punkten gesucht.
gamma = arcpy.GetParameterAsText(11)
if gamma == 1.0:
    # Lege ein Shapefile fuer diejenigen Dreieckssegmente an, die der Anfangspunkt des
    jeweiligen Skelettpfades mit LENGTH < epsilon beruehrt.
    # Sie werden zur aeusseren Begrenzung der envelope zones.
    arcpy.CreateFeatureclass_management(worksp,"D_envelope_segments_raw.shp","POLYLINE","","
    ","",input_line)
    # Ein InsertCursor fuer Segmente der envelope zones:
    envelope = arcpy.da.InsertCursor("D_envelope_segments_raw.shp",["Shape@"])
```

```
    # Der folgende Cursor fuer die Skelettpfade wird eingeschraenkt auf die Skelettpfade mit
    LENGTH < epsilon.
    clause19 = '"LENGTH" < ' + str(epsilon)
    with arcpy.da.SearchCursor(input_skelettpfade,["Shape@","BSID","LENGTH"],clause19) as
    cursor19:
        for path in cursor19:
            geom1 = path[0]
            BSID = path[1]
            clause20 = '"BS_FID" = ' + str(BSID) # Der folgende Cursor wird eingeschraenkt
            auf die Dreieckssegmente des bendsystems (BSID), in dem der Skelettpfad liegt.
            with arcpy.da.SearchCursor(input_triangles_segments,["SHAPE@","BS_FID","MID_X",
            "MID_Y"],clause20) as cursor20:
                for triseg in cursor20:
                    segment = triseg[0]
                    MidX = triseg[2] # Die berechneten Mittelpunktkoordinaten werden
                    ausgelesen.
                    MidY = triseg[3]
                    MidP = arcpy.Point(MidX,MidY) # Ein Punktobjekt wird aus den Koordinaten
                    erstellt.
                    if MidP.touches(geom1): # Wenn der Mittelpunkt des aktuellen
                    Dreieckssegments den Skelettpfad (naemlich in dessen Anfangspunkt)
                    beruehrt...
                        envelope.insertRow([segment]) #... dann wird dieses Dreieckssegment
                        den envelope_segments_raw hinzugefuegt.
    del envelope

elif gamma < 1:
    arcpy.AddError("Der Wert fuer gamma muss groesser oder gleich 1 sein.")
else:
    epsilon_big = float(gamma) * float(epsilon)
    # Vorgehensweise:
    # 1) Schreibe die bottomPoints der Skelettpfade < epsilon in ein set_epsilon.
    # 2) Schreibe die bottomPoints der Skelettpfade > epsilon_big in ein set_gamma.
    #    Warum kann man nicht gleich alle Skelettpfade < epsilon_big auswaehlen und ihre
    bottomPoints weggeneralisieren?
    #    Weil sowohl kurze als auch lange Skelettpfade zu ein und demselben bottomPoint
    fuehren koennen
    #    und eben diejenigen bottomPoints als charakteristisch gelten und erhalten bleiben
    sollen, zu denen lange Skelettpfade fuehren.
    # 3) Bilde die Differenz aus set_epsilon - set_gamma = set_diff
    # 4) Waehle im folgenden die Skelettpfade anhand der bottemPoints in set_diff aus.

    # 1) Schreibe die bottomPoints der Skelettpfade < epsilon in ein set_epsilon.
    # Ein set fuer die bottomPoints der Skelettpfade mit Laenge < epsilon:
    set_epsilon = set()
    # Der folgende Cursor fuer die Skelettpfade wird eingeschraenkt auf die Skelettpfade mit
    LENGTH < epsilon.
    clause19 = '"LENGTH" < ' + str(epsilon) #
    with arcpy.da.SearchCursor(input_skelettpfade,["LENGTH","BOPO"],clause19) as cursor19:
        for path in cursor19:
            BOPO = path[1]
            set_epsilon.add(BOPO)

    # 2) Schreibe die bottomPoints der Skelettpfade > epsilon_big in ein set_gamma.
    # Ein set fuer die bottomPoints der Skelettpfade mit Laenge > epsilon_big:
    set_gamma = set()
    # Der folgende Cursor fuer die Skelettpfade wird eingeschraenkt auf die Skelettpfade mit
    LENGTH > epsilon_big.
    clause21 = '"LENGTH" >= ' + str(epsilon_big)
    with arcpy.da.SearchCursor(input_skelettpfade,["LENGTH","BOPO"],clause21) as cursor21:
        for path in cursor21:
            BOPO = path[1]
            set_gamma.add(BOPO)

    # 3) Bilde die Differenz aus set_epsilon - set_gamma = set_diff
    set_diff = set_epsilon - set_gamma

    # 4) Waehle im folgenden die Skelettpfade anhand der bottomPoints in set_diff aus.
```

```
    # Lege zunaechst ein Shapefile fuer diejenigen Dreieckssegmente an, die der Anfangspunkt
    des jeweiligen Skelettpfades beruehrt.
    # Aus ihnen wird die aeussere Begrenzung der envelope zones ermittelt.
    arcpy.CreateFeatureclass_management(worksp,"D_envelope_segments_raw.shp","POLYLINE","","
    ","",input_line)
    # Ein InsertCursor fuer Segmente der envelope zones:
    envelope = arcpy.da.InsertCursor("D_envelope_segments_raw.shp",["Shape@"])

    # Der Cursor fuer die Skelettpfade wird vorab schon mal auf die Skelettpfade mit LENGTH
    < epsilon eingeschraenkt (clause19, siehe oben):
    with arcpy.da.SearchCursor(input_skelettpfade,["Shape@","BSID","LENGTH","BOPO"],clause19)
     as cursor22:
        for path in cursor22:
            geom1 = path[0]
            BSID = path[1]
            BOPO = path[3]
            if BOPO in set_diff:
                # Fuer die Pruefung auf Beruehrung wird der folgende Cursor eingeschraenkt
                auf die Dreieckssegmente des bendsystems (BSID),
                # in dem der Skelettpfad liegt.
                clause20 = '"BS_FID" = ' + str(BSID)
                with arcpy.da.SearchCursor(input_triangles_segments,["SHAPE@","BS_FID",
                "MID_X","MID_Y"],clause20) as cursor20:
                    for triseg in cursor20:
                        segment = triseg[0]
                        MidX = triseg[2] # Die berechneten Mittelpunktkoordinaten werden
                        ausgelesen.
                        MidY = triseg[3]
                        MidP = arcpy.Point(MidX,MidY) # Ein Punktobjekt wird aus den
                        Koordinaten erstellt.
                        if MidP.touches(geom1): # Wenn der Mittelpunkt des aktuellen
                        Dreieckssegments den Skelettpfad (naemlich in dessen Anfangspunkt)
                        beruehrt...
                            envelope.insertRow([segment]) #... dann wird dieses
                            Dreieckssegment den envelope_segments_raw hinzugefuegt.
    del envelope

# Generiere envelope-zones aus den envelope-segments und der Originallinie. Alle
eingeschlossenen Flaechen sind Bestandteil von envelope-zones:
arcpy.FeatureToPolygon_management(["D_envelope_segments_raw.shp",input_line],output_parcels)

# Weise den output_parcels (= Dreiecke, die eine envelope zone bilden) ein Attribut fuer die
Lage links oder rechts von der Originallinie zu
# durch Intersect mit input_bendsystems, die dieses Attribut ("Topology") bereits besitzen.
arcpy.Intersect_analysis([output_parcels,input_bendsystems],
"D_envelope_zones_parcels_leftright.shp")

# Entferne Binnengrenzen zwischen aneinandergrenzenden Flaechen auf derselben Linienseite:
arcpy.Dissolve_management("D_envelope_zones_parcels_leftright.shp",
"D_envelope_zones_leftright_multi.shp","Topology")

# Wandle Multipart-Features in Single-Features um:
arcpy.MultipartToSinglepart_management("D_envelope_zones_leftright_multi.shp",
output_zones_leftright)

# Speichere von den D_envelope_segments_raw nur die, die auf der Aussenkante der
output_zones_leftright liegen,
# in einem neuen Shapefile:
arcpy.MakeFeatureLayer_management("D_envelope_segments_raw.shp","envelope_segments_raw_Layer")
arcpy.SelectLayerByLocation_management("envelope_segments_raw_Layer",
"SHARE_A_LINE_SEGMENT_WITH",output_zones_leftright,"","NEW_SELECTION")
arcpy.CopyFeatures_management("envelope_segments_raw_Layer",
"D_envelope_outer_segments_raw.shp")
# Weise diesen Segmenten ein Attribut fuer die Lage links oder rechts von der Originallinie zu
# durch Intersect mit input_bendsystems, die dieses Attribut ("Topology") bereits besitzen:
arcpy.Intersect_analysis(["D_envelope_outer_segments_raw.shp",input_bendsystems],
"D_envelope_outer_segments.shp")
# Verbinde envelope_outer_segments, die sich beruehren und auf derselben Seite der
Originallinie liegen:
arcpy.UnsplitLine_management("D_envelope_outer_segments.shp",output_envelope_segments,
"Topology")
```

A7 GenerateSimplifiedLine

```
# Autorin: Christiane Enderle
# Datum: 7.1.2019
# Fuer ArcGIS Desktop 10.3.1 mit Python 2.7.8

# Das Python-Skript-Tool ist nur im Rahmen des Modells, das den Algorithmus von Ai et al.
(2016) implementiert, sinnvoll einsetzbar.
# Es generiert die vereinfachte Linie in Abhaengigkeit von der Nutzereingabe der
Simplify-Methode (left, right, aera preserving),
# ob nur die links oder nur die rechts von der Originallinie liegenden Envelope-Segmente die
entsprechenden Linienabschnitte ersetzen sollen,
# oder ob fuer jede Envelope-Zone anhand der Flaechenbilanz entschieden werden soll, ob das
linke oder das rechte Envelope-Segment verwendet wird.
# Im letzteren Fall kann ein Wert fuer m angegeben werden. Fuer die Bedeutung von m siehe Ai
et al. (2016), S. 307, bzw. Kapitel "Algorithmus von Ai et al. (2016)".
# m = (Summe der linkseitig abgschnittenen Flaecheninhalte - Summe der rechtsseitig
abgeschnittenen Flaecheninhalte) /
# (Summe der linkseitig abgschnittenen Flaecheninhalte + Summe der rechtsseitig
abgeschnittenen Flaecheninhalte)

import arcpy
arcpy.env.overwriteOutput = 1
arcpy.env.XYTolerance = 0.1

input_line = arcpy.GetParameterAsText(0) # Orignallinie nach der Triangulation
input_points = arcpy.GetParameterAsText(1) # "B_Original_Points_densified.shp"
input_segments = arcpy.GetParameterAsText(2) # "D_envelope_segments.shp"
input_parcels = arcpy.GetParameterAsText(3) # "D_envelope_zones_parcels.shp" (nur fuer
Methode "centerlines", die nicht fehlerfrei realisiert wurde)
input_zones_leftright = arcpy.GetParameterAsText(4) # "D_envelope_zones_leftright.shp"
output_simplified = arcpy.GetParameterAsText(5) # "E_simplified_line_densified.shp"
worksp = arcpy.GetParameterAsText(6)
arcpy.env.workspace = worksp

method = arcpy.GetParameterAsText(7) # 'right'/'left'/('area preserving' wurde nicht
fehlerfrei realisiert)

#
-------------------------------------------------------------------------------------------------
-----------------------------------------------
# -------------------- Zusammensetzung der simplified line in Abhaengigkeit von der Methode,
die der Nutzer wuenscht ------------------------
#
-------------------------------------------------------------------------------------------------
-----------------------------------------------

# Die Variable method ist eine Nutzereingabe und kann folgende Werte annehmen (vgl. Ai et
al. (2016), S. 307):
# left oder right: Es werden nur die envelope-segments auf der gewaehlten Seite der
Originallinie
#                   fuer die Konstruktion der simplified_line herangezogen.
# shape preserving: aus dem Artikel von Ai et al. (2016) ging nicht hervor, wie sich diese
Methode von der Methode "area preserving" unterscheidet.
#                    Sie wurde daher nicht umgesetzt.
# area preserving: Es wird fuer jede envelope-zone individuell entschieden, ob ein
envelope-segment von rechts oder links
#                   oder ob die centerline verwendet wird.
# In Abweichung von Ai et al. (2016) soll ausserdem die Methode "centerlines" zugelassen
werden. Dann werden nur die Mittelpfade verwendet.
# Die Methode "centerlines" konnte jedoch im Rahmen dieser Arbeit nicht fehlerfrei
realisiert werden.

# -----------------------------------------------------
# ------ Generieren der generalisierten Linie --------
# -----------------------------------------------------
# Prinzipieller Ablauf:
# 1) Trenne die Originallinie an den Punkten, in denen die envelope_segments oder
centerlines sie beruehren.
# 2) Bringe die Segmente der Originallinie in die richtige Reihenfolge.
```

```
# 3) Setze die generalisierte Linie aus Teilen der Originallinie und den envelope_segments
der rechten oder linken Seite oder den centerlines,
#    je nach Nutzereingabe fuer "method", zusammen.

#
-------------------------------------------------------------------------------------------
-------------
# -------- Vorarbeiten fuer die Varianten 'left', 'right', 'area preserving' ----------
#
-------------------------------------------------------------------------------------------
-------------

#
if method != 'centerlines':
    # Die input_segments stammen aus den Delaunay-Dreiecken und haben einen z-Wert.
    # Die Linien, die im Rahmen dieser Untersuchung vereinfacht werden sollen, sowie die
    centerlines haben keinen z-Wert.
    # Damit envelope-segments mit verbleibenden Abschnitten der Originallinie und/oder
    centerlines verbunden werden koennen,
    # werden im folgenden die envelope-segments in ein neues, ohne z-value angelegtes
    Shapefile geschrieben.
    arcpy.CreateFeatureclass_management(worksp, "E_envelope_segments_2D.shp", "POLYLINE", "",
     "DISABLED", "DISABLED", input_line)
    arcpy.AddField_management("E_envelope_segments_2D.shp","Topology","TEXT","","",1)
    segments_2D = arcpy.da.InsertCursor("E_envelope_segments_2D.shp",["SHAPE@","Topology"])
    with arcpy.da.SearchCursor(input_segments,["SHAPE@","Topology"]) as copycursor:
        for row in copycursor:
            segments_2D.insertRow(row)
    del segments_2D
# ---------------------------------------------------------------------
# ----- 'left' --------------------------------------------------------
# ---------------------------------------------------------------------
if method == 'left':
    side_clause_L = '"Topology" = ' + "'L'" # Einschraenkung auf die envelope_segments links
    von der Originallinie.
    # Ermittle die Anzahl der envelope-segments auf der linken Seite der Originallinie.
    arcpy.MakeFeatureLayer_management("E_envelope_segments_2D.shp","segments_left_Layer",
    side_clause_L) # Es wird ein Layer mit der Auswahl-Klausel side_clause_L erstellt.
    result = arcpy.GetCount_management("segments_left_Layer") # Die Features darin werden
    gezaehlt...
    countEnvelopes_L = int(result.getOutput(0)) # ...und in der Variable countEnvelopes_L
    gespeichert.
    arcpy.CopyFeatures_management("segments_left_Layer","E_envelope_segments_left.shp")
    if countEnvelopes_L == 0:
        arcpy.AddError("Es gibt keine vereinfachten Linienabschnitte auf der linken Seite.")
    elif countEnvelopes_L > 0:

# ----- 1) 'left': Trenne die Originallinie an den Punkten, in denen die envelope_segments
sie beruehren.
        # Finde die Punkte, in denen die envelope_segments die Originallinie beruehren:
        point_list_L = []
        with arcpy.da.SearchCursor(input_points,"SHAPE@") as cursor_point:
            for point in cursor_point:
                geom_point = point[0]
                with arcpy.da.SearchCursor("E_envelope_segments_left.shp","SHAPE@") as
                cursor_segment:
                    for segment in cursor_segment:
                        geom_segment = segment[0]
                        if geom_point.touches(geom_segment):
                            point_list_L.append(geom_point)
                            break # Nach dem ersten Treffer kann die Segment-Schleife
                            abgebrochen werden.
        # Kopiere die gefundenen Punkte in "E_split_points_L.shp":
        arcpy.CopyFeatures_management(point_list_L,"E_split_points_L.shp")

        # Fuer das Tool "SplitLineAtPoint" muss offenbar ein Search-Radius angegeben werden,
        # obwohl die Punkte genau von der Linie stammen, die sie jetzt splitten sollen,
        # so dass eigentlich keine Koordinatenabweichung bestehen sollte.
```

```
        # Der Suchradius soll kleiner sein, als der kleinste Abstand zwischen zwei
        benachbarten Punkten.
        # Der Abstand zwischen allen Punkten wird in eine Tabelle geschrieben:
        arcpy.GenerateNearTable_analysis("E_split_points_L.shp","E_split_points_L.shp",
        "NearTable_L")
        # Finde den minimalen Abstand zwischen zwei Punkten:
        arcpy.Statistics_analysis("NearTable_L", "NearTable_L_statistics", [["NEAR_DIST",
        "MIN"]])
        Minimum = [row[0] for row in arcpy.da.SearchCursor("NearTable_L_statistics",
        "MIN_NEAR_DIST")]
        # Um ganz korrekt zu sein, muesste die lineare Einheit fuer den folgenden
        search_radius aus dem Koordinatensystem der Originallinie abgefragt werden.
        # Darauf wird hier verzichtet. Der Suchradius wird auf 1/100 des minimalen Abstands
        gesetzt. Als Einheit wird Meter verwendet:
        search_radius_L = str(Minimum[0] / 100) + " Meters" # Quelle: Lippmann, Eric (2012):
        Python: Division von Gleitkomma- und komplexen Zahlen. -
        https://www.netways.de/blog/2012/08/09/python-division-von-gleitkomma-und-komplexen-za
        hlen/, Stand: 9.8.2012, Zugriff: 7.1.2019

        # Trenne die Originallinie an den Beruehrpunkten:
        arcpy.SplitLineAtPoint_management(input_line,"E_split_points_L.shp",
        "E_Original_segments_L.shp",search_radius_L)


# ----- 2) 'left': Wichtig fuer die anschliessende Zusammensetzung ist, dass die
Original_segments mit dem ersten Segment am Anfangspunkt der Originallinie beginnen
        # und die richtige Reihenfolge entlang der Originallinie haben. Das gleiche gilt
        fuer die envelope-segments.
        # Original_segments:
        # Lies die Geometrie des Anfangspunktes der Originallinie aus:
        with arcpy.da.SearchCursor(input_points,["SHAPE@","ORIG_FID"],'"ORIG_FID" = 0') as
        cursor:
            for geom in cursor:
                startpunkt = geom[0]

        LineGeometryList_L = [] # Eine leere Liste, die mit den geordneten Original_segments
        gefuellt wird. Das Suffix _L steht fuer left.

        # Bringe die Original_segments in die richtige Reihenfolge:
        # Dafuer wird die Anzahl der Original_segments benoetigt:
        result = arcpy.GetCount_management("E_Original_segments_L.shp")
        countLines_L = int(result.getOutput(0))

        voriges = startpunkt
        for i in range(0,countLines_L):
            with arcpy.da.UpdateCursor("E_Original_segments_L.shp","SHAPE@") as cursor:
                for line in cursor:
                    geom_line = line[0]
                    if voriges.touches(geom_line):
                        LineGeometryList_L.append(geom_line)
                        voriges = geom_line
                        cursor.deleteRow()
                        break # Der Cursor wird jeweils nach dem ersten Treffer nicht mehr
                        benoetigt und daher abgebrochen.

        # Schreibe die geordneten Segmente in ein Shapefile E_Original_segments_L_order.shp:
        arcpy.CopyFeatures_management(LineGeometryList_L,"E_Original_segments_L_order.shp")

        # Envelope segments:
        SegmGeometryList_L = [] # Eine leere Liste, die mit den geordneten envelope_segments
        gefuellt wird. Das Suffix _L steht fuer left.
        # Bringe die envelope_segments in die richtige Reihenfolge:
        # Dafuer wird die Anzahl der envelope_segments benoetigt:
        result = arcpy.GetCount_management("E_envelope_segments_left.shp")
        countSegm_L = int(result.getOutput(0))

        for i in range(0,countSegm_L):
            # Es genuegt, dass jeweils der uebernaechste Punkt fuer die Pruefung auf
```

```
            Beruehrung mit einem envelope-segment ausgewaehlt wird.
            # Denn jedes envelope-segment ist von zwei Split-Points begrenzt und die
            envelope-segments beruehren sich nicht in den Split-Points,
            # weil zwischen zwei envelope-segments immer ein Segment der Originallinie liegt.
            clause_points = '"FID" = ' + str(2*i)
            # Die split_points liegen in der Reihenfolge des Linienverlaufs vor.
            arcpy.MakeFeatureLayer_management("E_split_points_L.shp","split_points_L",
            clause_points)
            arcpy.SelectLayerByLocation_management("segments_left_Layer", "BOUNDARY_TOUCHES",
             "split_points_L", "", "NEW_SELECTION")
            for segment in arcpy.da.SearchCursor("segments_left_Layer", "SHAPE@"):
                geom_segm = segment[0]
                SegmGeometryList_L.append(geom_segm)

        arcpy.CopyFeatures_management(SegmGeometryList_L,"E_envelope_segments_L_order.shp")
        arcpy.SelectLayerByAttribute_management("segments_left_Layer", "CLEAR_SELECTION",
        '"FID" = 0')


# ----- 3) 'left': Zusammensetzen der vereinfachten Linie. -----
        LineGeometryList = [] # Eine leere Liste fuer die folgende Zusammensetzung aus
        envelope-segments und original-segments
        # Diejenigen Segmente der Originallinie, die durch ein envelope-segment ersetzt
        werden, werden nicht verwendet.
        # Gibt es bereits am Anfangspunkt ein envelope-segment?
        voriges = startpunkt # Die Variable voriges wird auf den Startpunkt der
        Originallinie zurueckgesetzt.
        # Es wird geprueft, ob es bereits am Anfangspunkt ein envelope-segment gibt:
##        with arcpy.da.UpdateCursor("segments_left_Layer","SHAPE@") as cursor_envelope:
##            for segment in cursor_envelope:
##                geom_segment = segment[0]
##                if voriges.touches(geom_segment): # Wenn ja, wird es als erstes Feature in
die LineGeometryList eingetragen und dann aus dem Shapefile geloescht.
##                    LineGeometryList.append(geom_segment)
##                    voriges = geom_segment
##                    cursor_envelope.deleteRow()
##                    countEnvelopes_L = countEnvelopes_L - 1 # Entsprechend reduziert sich
die Anzahl der Features in segments_left_Layer um 1.
##                    break
        # Der voranstehende auskommentierte Code funktioniert nicht zuverlaessig.
        # Deshalb wurden die raeumlichen Abfragen mit arcpy-Geometrie-Objekten durch
        ArcGIS-Tools ersetzt:
        arcpy.MakeFeatureLayer_management("E_envelope_segments_L_order.shp","segments_left")
        arcpy.MakeFeatureLayer_management(input_points,"input_points_Layer",'"ORIG_FID" = 0')
        arcpy.SelectLayerByLocation_management("segments_left", "BOUNDARY_TOUCHES",
        "input_points_Layer", "", "NEW_SELECTION")
        result = arcpy.GetCount_management("segments_left")
        num = int(result.getOutput(0))
        if num == 1:
            with arcpy.da.UpdateCursor("segments_left","SHAPE@") as cursor_envelope:
                for s in cursor_envelope:
                    geom_segment = s[0]
                    LineGeometryList.append(geom_segment)
                    voriges = geom_segment
                    cursor_envelope.deleteRow()
                    countEnvelopes_L = countEnvelopes_L - 1
##        else:
##
arcpy.MakeFeatureLayer_management("E_Original_segments_L_order.shp","original_segments_left")
##            arcpy.SelectLayerByLocation_management("original_segments_left",
"BOUNDARY_TOUCHES", "input_points_Layer", "", "NEW_SELECTION")
##            with arcpy.da.UpdateCursor("original_segments_left","SHAPE@") as
cursor_original:
##                for orig in cursor_original:
##                    geom_orig = orig[0]
##                    LineGeometryList.append(geom_orig)
##                    ## cursor_original.deleteRow()
##                    ## countLines_L = countLines_L - 1
```

```
##                    voriges = geom_orig

        # Wenn es am Anfangspunkt der Originallinie kein envelope-segment gibt, behaelt
        "voriges" als Wert den Punkt mit ORIG_FID = 0.
        # Im folgenden wird abwechselnd ein Segment der Originallinie und ein
        envelope-segment aneinandergehaengt und
        # sofort aus dem jeweiligen Quell-Shapefile geloescht:
        while countLines_L > 0:
            with arcpy.da.UpdateCursor("E_Original_segments_L_order.shp","SHAPE@") as
            cursor_line:
                for line in cursor_line:
                    geom_line = line[0]
                    if voriges in LineGeometryList:
                        # "voriges" ist entweder der Startpunkt, der sich nicht in
                        LineGeometryList befindet. Dann geht es  mit dem else-statement
                        weiter.
                        # Oder "voriges" ist ein envelope-Segment und befindet sich in
                        LineGeometryList.
                        # Dann ist geom_line ein zum envelope-Segment "parallel"
                        verlaufendes Original-Segment, das ersetzt werden soll.
                        # Dieses Teilstueck der Originallinie wird daher nicht an
                        LineGeometryList angehaengt.
                        voriges = geom_line
                        cursor_line.deleteRow()
                        countLines_L = countLines_L -1
                    # Das naechste Segment geom_line ist ein Teilstueck der Originallinie,
                    das erhalten bleiben soll.
                    else:
                        LineGeometryList.append(geom_line)
                        voriges = geom_line
                        cursor_line.deleteRow()
                        countLines_L = countLines_L - 1
                        break # Der cursor_line wird hier abgebrochen, weil als naechstes
                        wieder ein envelope-segment an die LineGeometryList angehaengt
                        werden muss.
            if countEnvelopes_L > 0:
                with arcpy.da.UpdateCursor("E_envelope_segments_L_order.shp","SHAPE@") as
                cursor_envelope:
                    for segment in cursor_envelope:
                        geom_segment = segment[0]
##                        if voriges.touches(geom_segment):
                        LineGeometryList.append(geom_segment)
                        cursor_envelope.deleteRow()
                        countEnvelopes_L = countEnvelopes_L - 1
                        break

        # Erstelle aus der LineGeometryList ein neues Shapefile:
        arcpy.CopyFeatures_management(LineGeometryList,"E_simplified_line_segments.shp")
        # Verbinde die Einzelsegmente miteinander:
        arcpy.UnsplitLine_management("E_simplified_line_segments.shp",output_simplified)


# ---------------------------------------------------------------------
# ----- 'right' -------------------------------------------------------
# ---------------------------------------------------------------------
if method == 'right':
    side_clause_R = '"Topology" = ' + "'R'" # Einschraenkung auf die envelope_segments
    rechts von der Originallinie.
    # Ermittle die Anzahl der envelope-segments auf der rechten Seite der Originallinie.
    arcpy.MakeFeatureLayer_management("E_envelope_segments_2D.shp","segments_right_Layer",
    side_clause_R) # Es wird ein Layer mit der Auswahl-Klausel side_clause_R erstellt.
    result = arcpy.GetCount_management("segments_right_Layer") # Die Features darin werden
    gezaehlt...
    countEnvelopes_R = int(result.getOutput(0)) # ...und in der Variable countEnvelopes_R
    gespeichert.
    arcpy.CopyFeatures_management("segments_right_Layer","E_envelope_segments_right.shp")
    if countEnvelopes_R == 0:
```

```
        arcpy.AddError("Es gibt keine vereinfachten Linienabschnitte auf der rechten Seite.")
    elif countEnvelopes_R > 0:

# ----- 1) 'right': Trenne die Originallinie an den Punkten, in denen die envelope_segments
sie beruehren.
        # Finde die Punkte, in denen die envelope_segments die Originallinie beruehren:
        point_list_R = []
        with arcpy.da.SearchCursor(input_points,"SHAPE@") as cursor_point:
            for point in cursor_point:
                geom_point = point[0]
                with arcpy.da.SearchCursor("E_envelope_segments_right.shp","SHAPE@") as
                cursor_segment:
                    for segment in cursor_segment:
                        geom_segment = segment[0]
                        if geom_point.touches(geom_segment):
                            point_list_R.append(geom_point)
                            break # Nach dem ersten Treffer kann die Segment-Schleife
                            abgebrochen werden.
        # Kopiere die gefundenen Punkte in "E_split_points_R.shp":
        arcpy.CopyFeatures_management(point_list_R,"E_split_points_R.shp")

        # Fuer das Tool "SplitLineAtPoint" muss offenbar ein Search-Radius angegeben werden,
        # obwohl die Punkte genau von der Linie stammen, die sie jetzt splitten sollen,
        # so dass eigentlich keine Koordinatenabweichung bestehen sollte.
        # Der Suchradius soll kleiner sein, als der kleinste Abstand zwischen zwei
        benachbarten Punkten.
        # Der Abstand zwischen allen Punkten wird in eine Tabelle geschrieben:
        arcpy.GenerateNearTable_analysis("E_split_points_R.shp","E_split_points_R.shp",
        "NearTable_R")
        # Finde den minimalen Abstand zwischen zwei Punkten:
        arcpy.Statistics_analysis("NearTable_R", "NearTable_R_statistics", [["NEAR_DIST",
        "MIN"]])
        Minimum = [row[0] for row in arcpy.da.SearchCursor("NearTable_R_statistics",
        "MIN_NEAR_DIST")]
        # Um ganz korrekt zu sein, muesste die lineare Einheit fuer den folgenden
        search_radius aus dem Koordinatensystem der Originallinie abgefragt werden.
        # Darauf wird hier verzichtet. Der Suchradius wird auf 1/100 des minimalen Abstands
        gesetzt. Als Einheit wird Meter verwendet:
        search_radius_R = str(Minimum[0] / 100) + " Meters" # Quelle: Lippmann, Eric (2012):
        Python: Division von Gleitkomma- und komplexen Zahlen. -
        https://www.netways.de/blog/2012/08/09/python-division-von-gleitkomma-und-komplexen-za
        hlen/, Stand: 9.8.2012, Zugriff: 7.1.2019

        # Trenne die Originallinie an den Beruehrpunkten:
        arcpy.SplitLineAtPoint_management(input_line,"E_split_points_R.shp",
        "E_Original_segments_R.shp",search_radius_R)


# ----- 2) 'right': Wichtig fuer die anschliessende Zusammensetzung ist, dass die
Original_segments mit dem ersten Segment am Anfangspunkt der Originallinie beginnen
        # und die richtige Reihenfolge entlang der Originallinie haben.
        # Lies die Geoemtrie des Anfangspunktes der Originallinie aus:
        with arcpy.da.SearchCursor(input_points,["SHAPE@","ORIG_FID"],'"ORIG_FID" = 0') as
        cursor:
            for geom in cursor:
                startpunkt = geom[0]

        LineGeometryList_R = [] # Eine leere Liste, die mit den geordneten Original_segments
        gefuellt wird. Das Suffix _R steht fuer right.

        # Bringe die Original_segments in die richtige Reihenfolge:
        # Dafuer wird die Anzahl der Original_segments benoetigt:
        result = arcpy.GetCount_management("E_Original_segments_R.shp")
        countLines_R = int(result.getOutput(0))

        voriges = startpunkt
        for i in range(0,countLines_R):
            with arcpy.da.UpdateCursor("E_Original_segments_R.shp","SHAPE@") as cursor:
```

```
                for line in cursor:
                    geom_line = line[0]
                    if voriges.touches(geom_line):
                        LineGeometryList_R.append(geom_line)
                        voriges = geom_line
                        cursor.deleteRow()
                        break # Der Cursor wird jeweils nach dem ersten Treffer nicht mehr
                        benoetigt und daher abgebrochen.

        # Schreibe die geordneten Segmente in ein Shapefile E_Original_segments_order.shp:
        arcpy.CopyFeatures_management(LineGeometryList_R,"E_Original_segments_R_order.shp")


# ----- 3)'right': Zusammensetzen der vereinfachten Linie. -----
        LineGeometryList = [] # Eine leere Liste fuer die folgende Zusammensetzung aus
        envelopes und original-segments
        # Diejenigen Segmente der Originallinie, die durch ein envelope-segment ersetzt
        werden, werden nicht verwendet.
        # Gibt es bereits am Anfangspunkt ein envelope-segment?
        voriges = startpunkt # Die Variable voriges wird auf den Startpunkt der
        Originallinie zurueckgesetzt.
        # Es wird geprueft, ob es bereits am Anfangspunkt ein envelope-segment gibt:
##        with arcpy.da.UpdateCursor("envelope_segments_right.shp","SHAPE@") as
cursor_envelope:
##            for segment in cursor_envelope:
##                geom_segment = segment[0]
##                if voriges.touches(geom_segment): # Wenn ja, wird es als erstes Feature in
die LineGeometryList eingetragen und dann aus dem Shapefile geloescht.
##                    print "Es gibt am Anfangspunkt ein envelope_R-segment." # zu Testzwecken
##                    LineGeometryList.append(geom_segment)
##                    voriges = geom_segment
##                    cursor_envelope.deleteRow()
##                    countEnvelopes_R = countEnvelopes_R - 1 # Entsprechend reduziert sich
die Anzahl der Features in segments_right_Layer um 1.
##                    break
        # Der voranstehende auskommentierte Code funktioniert nicht zuverlaessig.
        # Deshalb wurden die raeumlichen Abfragen mit arcpy-Geometrie-Objekten durch
        ArcGIS-Tools ersetzt:
        arcpy.MakeFeatureLayer_management("E_envelope_segments_right.shp","segments_right")
        arcpy.MakeFeatureLayer_management(input_points,"input_points_Layer",'"ORIG_FID" = 0')
        arcpy.SelectLayerByLocation_management("segments_right", "BOUNDARY_TOUCHES",
        "input_points_Layer", "", "NEW_SELECTION")
        result = arcpy.GetCount_management("segments_right")
        num = int(result.getOutput(0))
        if num == 1:
            with arcpy.da.UpdateCursor("segments_right","SHAPE@") as cursor_envelope:
                for s in cursor_envelope:
                    geom_segment = s[0]
                    LineGeometryList.append(geom_segment)
                    voriges = geom_segment
                    cursor_envelope.deleteRow()
                    countEnvelopes_R = countEnvelopes_R - 1

        # Wenn es am Anfangspunkt der Originallinie kein envelope-segment gibt, behaelt
        "voriges" als Wert den Punkt mit ORIG_FID = 0.
        # Im folgenden wird abwechselnd ein Segment der Originallinie und ein
        envelope-segment aneinandergehaengt und
        # sofort aus dem jeweiligen Quell-Shapefile geloescht. Weitere Kommentare siehe oben
        bei method "left".
        while countLines_R > 0:
            with arcpy.da.UpdateCursor("E_Original_segments_R_order.shp","SHAPE@") as
            cursor_line:
                for line in cursor_line:
                    geom_line = line[0]
                    if (voriges.touches(geom_line) or voriges.crosses(geom_line)) and voriges
                     in LineGeometryList:
                        cursor_line.deleteRow()
                        countLines_R = countLines_R -1
```

```
                voriges = geom_line
            elif voriges.touches(geom_line) and not voriges in LineGeometryList:
                LineGeometryList.append(geom_line)
                voriges = geom_line
                cursor_line.deleteRow()
                countLines_R = countLines_R - 1
                break
    if countEnvelopes_R > 0:
        with arcpy.da.UpdateCursor("E_envelope_segments_right.shp","SHAPE@") as
        cursor_envelope:
            for segment in cursor_envelope:
                geom_segment = segment[0]
                if voriges.touches(geom_segment):
                    LineGeometryList.append(geom_segment)
                    cursor_envelope.deleteRow()
                    countEnvelopes_R = countEnvelopes_R - 1
                    break

# Erstelle aus der LineGeometryList ein neues Shapefile:
arcpy.CopyFeatures_management(LineGeometryList,"E_simplified_line_segments.shp")
# Verbinde die Einzelsegmente miteinander:
arcpy.UnsplitLine_management("E_simplified_line_segments.shp",output_simplified)
```

A8 RemoveAddedPoints

```
# Autorin: Christiane Enderle
# Datum: 10.6.2019
# Fuer ArcGIS Desktop 10.3.1 mit Python 2.7.8

# Das Python-Skript-Tool ist nur im Rahmen des Modells, das den Algorithmus von Ai et al.
(2016) implementiert, sinnvoll einsetzbar.
# Es entfernt diejenigen Punkte aus der vereinfachten Linie, die durch die Triangulation
hinzugefuegt worden sind
# und in der vereinfachten Linie keine Richtungsaenderung bewirken, d.h. auf einer Geraden
mit dem vorigen und dem folgenden Punkt liegen.
# Diese Bedingung wird geprueft, indem der Winkel zum vorigen und der Winkel zum folgenden
Punkt berechnet wird.
# Die Betraege der Winkel muessen in der Summe 180 Grad ergeben. Minimale Abweichungen
werden durch einen Toleranzwert aufgefangen.

import arcpy
arcpy.env.overwriteOutput = 1

input_original = arcpy.GetParameterAsText(0) # "A_Original_Points.shp" - Vertices der
Originallinie
input_simpl_plus = arcpy.GetParameterAsText(1) # "E_Simpl_Line_densif_Vert.shp" - Vertices
der vereinfachten Linie inklusive der zusaetzlichen Punkte
output_simplified = arcpy.GetParameterAsText(2) # "E_Simplified_Line_Vertices.shp"

# Kopiere den input_simpl_plus in ein neues Shapefile, aus dem die ueberfluessigen Punkte
dann geloescht werden:
arcpy.CopyFeatures_management(input_simpl_plus, output_simplified)

# Erstelle Layer aus den Eingabe-Shapefiles, um eine Auswahl treffen zu koennen:
arcpy.MakeFeatureLayer_management(input_original,"original_points_Layer")
arcpy.MakeFeatureLayer_management(input_simpl_plus,"densified_vertices_Layer")

# Waehle aus den verdichteten Punkten per Intersect die Original-Punkte aus und kehre die
Auswahl um ("INVERT").
# Das Ergebnis sind die durch die Triangulation hinzugefuegten Punkte.
arcpy.SelectLayerByLocation_management ("densified_vertices_Layer", "INTERSECT",
"original_points_Layer", "", "NEW_SELECTION","INVERT")

# Erstelle eine leere Liste fuer die Feature_IDs der hinzugefuegten Punkte:
Point_ID = []
# Schreibe die Feature IDs der hinzugefuegten Punkte in diese Liste:
for pnt in arcpy.da.SearchCursor("densified_vertices_Layer", "FID"):
    ID = pnt[0]
    Point_ID.append(ID)
arcpy.AddMessage("Feature-IDs der hinzugefuegten Punkte: {0}" .format(Point_ID))

# Erstelle eine leere Liste fuer die zu entfernenden Punkte:
Delete_Points = []
# Bestimme fuer jeden durch die Triangulation hinzugefuegten Punkt die Feature_ID des
vorigen und des folgenden Punktes:
for ID in Point_ID:
    ID_vor = ID - 1
    ID_nach = ID + 1
    # Erstelle eine Liste mit tupeln fuer den vorigen, den hinzugefuegten und den folgenden
    Punkt.
    # Jedes tupel wird seinerseits eine Liste aus FID und Punktgeometrie.
    point_list = []
    # Erstelle eine clause fuer die Auswahl dieser drei Punkte aus dem Layer:
    clause1 = '"FID" = ' + str(ID_vor) + ' OR "FID" = ' + str(ID) + ' OR "FID" = ' + str(
    ID_nach)
    for pnt in arcpy.da.SearchCursor(output_simplified, ["FID","SHAPE@"], clause1):
        FID = pnt[0]
        geom = pnt[1]
        tuple = [FID,geom]
        point_list.append(tuple)

    # Sortiere die point_list aufsteigend nach der FID in den tupeln
    point_list.sort(key=lambda x: int(x[0]))
```

```
    # Lies fuer jeden Punkt die FID und die Geometrie aus der Liste und schreibe sie in
    Variablen:
    tuple_vor = point_list[0]
    geom_vor = tuple_vor[1]
    tuple_added = point_list[1]
    geom_added = tuple_added[1]
    tuple_nach = point_list[2]
    geom_nach = tuple_nach[1]

    # Erstelle eine clause fuer die Auswahl des aktuellen hinzugefuegten Punkt aus dem
    Layer:
    clause2 = '"FID" = ' + str(ID)
    with arcpy.da.SearchCursor(output_simplified, "FID", clause2) as cursor:
        for vertex in cursor:
            FID_added = vertex[0]
            # Pruefe sicherheitshalber, ob seine FID mit der FID aus dem tupel
            uebereinstimmt:
            if FID_added == tuple_added[0]:

                # Miss den Winkel zum vorigen und zum folgenden Punkt:
                vor = geom_vor.angleAndDistanceTo(geom_added,"PLANAR")
                winkel_vor = vor[0]
                nach = geom_added.angleAndDistanceTo(geom_nach,"PLANAR")
                winkel_nach = nach[0]
                arcpy.AddMessage("ID = {0}" .format(ID))
                arcpy.AddMessage("winkel_vor = {0}" .format(winkel_vor))
                arcpy.AddMessage("winkel_nach = {0}" .format(winkel_nach))

                # Wenn die Winkel gleich sind (Toleranzwert ist hier 0.1), dann ist der
                Punkt ueberfluessig und wird in die Liste der zu loeschenden Punkte
                geschrieben.
                if abs(winkel_vor - winkel_nach) < 0.1:
                    Delete_Points.append(ID)

arcpy.AddMessage("Feature-IDs der zu loeschenden Punkte: {0}" .format(Delete_Points))
# Die Punkte in der Liste Delete_Points werden aus output_simplified geloescht:
with arcpy.da.UpdateCursor(output_simplified, "FID") as cursor:
    for vertex in cursor:
        FID = vertex[0]
        if FID in Delete_Points:
            cursor.deleteRow()
```

A9 PostProcessing

```
# Autorin: Christiane Enderle
# Datum: 13.3.2019
# Fuer ArcGIS Desktop 10.3.1 mit Python 2.7.8

# Das Python-Skript-Tool ist nur im Rahmen des Modells, das den Algorithmus von Ai et al.
(2016) implementiert, sinnvoll einsetzbar.
# Es ersetzt Punkte der vereinfachten Linie, die infolge der bisherigen Verarbeitung nicht
mehr exakt auf den Punkten der Originallinie liegen,
# mit den entsprechenden Punkten der Originallinie. Dabei bewegen sich die Distanzen in der
Realwelt im Zehntel-Millimeter-Bereich.
# Dieses Postprocessing ist notwendig, weil das Evaluierungsprogramm von Chrobak et al
andernfalls Fehler produziert und abbricht.
# Die Vorgangsweise ist folgende:
# Die vertices der Originallinie und der vereinfachten Linie werden gespeichert (Tool
Feature vertices to Points).
# Anschliessend werden die Distanzen der Punkte der vereinfachten Linie zu denjenigen der
Originallinie untersucht.
# Sind sie kleiner als ein Schwellenwert, wird der jeweilige Punkt durch den Punkte der
Originallinie ersetzt.
# Anschliessend werden die Punkte mit dem Tool wieder zur vereinfachten Linie verbunden.

import arcpy
arcpy.env.overwriteOutput = 1

input_original = arcpy.GetParameterAsText(0) # Originallinie nach der Triangulation
input_simplified = arcpy.GetParameterAsText(1) # "E_Simplified_Line.shp"
output_postprocessed = arcpy.GetParameterAsText(2) # "F_Simplified_Line_postprocessed.shp"

# Die vertices der Originallinie und der vereinfachten Linie werden gespeichert:
arcpy.FeatureVerticesToPoints_management(input_original,"F_Original_Points.shp")
arcpy.FeatureVerticesToPoints_management(input_simplified,"F_Simplified_Points.shp")

# Die Punkte der Originallinie und der vereinfachten Linie werden iteriert:
with arcpy.da.UpdateCursor("F_Simplified_Points.shp","Shape@") as cursor_simpl:
    for pnt_simpl in cursor_simpl:
        simpl = pnt_simpl[0]

        with arcpy.da.SearchCursor("F_Original_Points.shp","Shape@") as cursor_orig:
            for pnt_orig in cursor_orig:
                orig = pnt_orig[0]
                # Die Distanz zwischen den Punkten wird geprueft und ggf. der Punkt der
                vereinfachten Linie ersetzt:
                if simpl.distanceTo(orig) < 0.001:
                    pnt simpl[0] = orig
                    cursor simpl.updateRow(pnt simpl)
                    break # Wenn ein entsprechender Punkt auf der Originallinie gefunden
                    wurde, kann die innere Schleife verlassen werden.

# Die Punkte der vereinfachten Linie werden wieder zu einer Polyline zusammengesetzt.
# Dieses Ergebnis muss als input fuer das Evaluierungsprogramm von Chrobak et al. (2016)
verwendet werden.
arcpy.PointsToLine_management("F_Simplified_Points.shp",output_postprocessed)
```

B Kommentare zum Artikel von Chrobak et al. (2016)

Geocarto International

ISSN: 1010-6049 (Print) 1752-0762 (Online) Journal homepage: http://www.tandfonline.com/loi/tgei20

A method for assessing generalized data accuracy with linear object resolution verification

Tadeusz Chrobak, Stanisław Szombara, Krystian Kozoł & Michal Lupa

To cite this article: Tadeusz Chrobak, Stanisław Szombara, Krystian Kozoł & Michal Lupa (2017) A method for assessing generalized data accuracy with linear object resolution verification, Geocarto International, 32:3, 238-256, DOI: 10.1080/10106049.2015.1133721

To link to this article: http://dx.doi.org/10.1080/10106049.2015.1133721

Accepted author version posted online: 18 Dec 2015.
Published online: 05 Feb 2016.

GEOCARTO INTERNATIONAL, 2017
VOL. 32, NO. 3, 238–256
http://dx.doi.org/10.1080/10106049.2015.1133721

Taylor & Francis
Taylor & Francis Group

A method for assessing generalized data accuracy with linear object resolution verification

Tadeusz Chrobak[a], Stanisław Szombara[a], Krystian Kozoł[a] and Michal Lupa[b]

[a]Department of Geomatics, AGH University of Science and Technology, Krakow, Poland; [b]Department of Geoinformatics and Applied Computer Science, AGH University of Science and Technology, Krakow, Poland

ABSTRACT

The article is composed of two sections. In the first section, the authors describe the application of minimum line dimensions which are dependent on line shape, width and the operational scale of the map. The proposed solutions are based on the Euclidean metric space, for which the minimum dimensions of Saliszczew's elementary triangle (*Elementary triangle* – is the term pertaining to model, standard triangle of least dimensions securing recognizability of a line. Its dimensions depend on scale of the map and width of the line representing it. The use of a triangle in the simplification process is as follows: triangles with sides (sections) on an arbitrary line and bases (completing the sides) are compared with lengths of the shorter side and the base of the elementary triangle.) were adapted. The second part of the article describes an application of minimum line dimensions for verifying and assessing generalized data. The authors also propose a method for determining drawing line resolution to evaluate the accuracy of algorithm simplification. Taking advantage of the proposed method, well-known simplification algorithms were compared on the basis of qualitative and quantitative evaluation. Moreover, corresponding with the methods of simplified data accuracy assessment the authors have extended these solutions with the rejected data. This procedure has allowed the identification of map areas where graphic conflicts occurred.

ARTICLE HISTORY
Received 17 August 2015
Accepted 14 December 2015

KEYWORDS
Generalization; minimum object dimensions; generalization assessment; graphic conflicts; cartography

1. Introduction

Maps are a medium containing information about real-world objects, distribution of geographical space phenomena, as well as the relationships between these components. Modern cartography is based on multiresolution spatial databases, which are the source of derivative works, i.e. topographical, thematic or general geographic content maps that are created dynamically at smaller scales. These databases (multiresolution databases – MRDB) are created using cartographic generalization, allowing a certain level of generalisation of data depending on a certain scale to be achieved (Lupa et al. 2015).

Along with the production of maps at smaller scales, generalized shapes are created based on invariable original points (obtained from the source object), as well as those points created during the simplification process. However, generalization methods, and in particular automatic generalization, often generate outcomes which are far from ideal. Therefore, choice of appropriate algorithms is difficult due to the differences between their effectiveness which leads to the need for objective

CONTACT Michal Lupa mlupa@agh.edu.pl

evaluation methods which will help in examining and checking whether the desired characteristics of the resulting data are satisfactory for a given task (Stoter et al. 2014).

Among the studies on modern cartography, we can find works related to assessing the quality of the generaliZation process (Mackaness 1991; Weibel 1993, 1995; Ruas and Plazanet 1995; Stoter et al., 2014). Nonetheless, there is a lack of methods concerning the verification of object resolution.[1] Thus, the readability of the resulting maps cannot be defined and assessed clearly. This, in turn, is due to the fact that the commonly used generalization algorithms (or their input parameters) are not based on map readability or recognizability formulas.

The first proposed method concerns the verification of drawing line resolution, depending on minimum line recognition dimensions, line width and the operational map scale. Firstly, the difference between the original and the generalized line shape is calculated. The determined difference is called an envelope. The map area covered by the envelope will be not recognizable in the target scale of the map being developed. Secondly, the line resolution is verified on elementary triangle standards, which are compared to standardized dimensions of the triangles formed on the examined lines. Finally, verification of line resolution is conducted on a deterministic model, using the principles of the elementary triangle (and its minimum dimensions). This model takes into account the number of deleted envelopes and coves, their maximum heights as well as the number of points (vertices) after the simplification.

A further section of the article covers the experimental studies. The accuracy of the simplified line and the verification of line resolution were tested with a propagation of the uncertainty law. Four well-known line simplification algorithms were used for the testing and the results were verified with methods suggested in the article. Moreover, the simplified line is assessed unambiguously. An objective assessment of the process (with no user influence on the outcomes) allows the determining of the optimal simplification algorithm, as shown in Chapter 4. The result is affected by the following factors:

- Line resolution, which is verified based on the minimum size obtained from the elementary triangle,
- the number of vertices of the generalized line,
- the amount of unrecognized and recognizable envelopes and coves,
- the accuracy of the data was determined on the basis of the methods of McMaster (1986) and Weibel (1997). The method was expanded by adding determination of locations of the envelopes' and coves' deleted during the simplification.

The location of envelopes could facilitate graphic conflict elimination due to the safe area (envelope) in which objects can be moved without causing further conflicts.

(Burghardt et al. 2014) based on work (Mackaness & Ruas 2007) have distinguished three types of evaluation in cartographic generalization: *evaluation for tuning, evaluation for controlling* and *evaluation for assessing*. Evaluation for assessing, according to (Bard & Ruas 2005), could be divided into three main types:

- *Evaluation for editing* – Concentrates on the detection of errors arising from the generalization process (errors are removed automatically or semi-automatically).
- *Evaluation for grading* – Designed to assess the quality of the data after processing and the comparison of different generalization solutions.
- *Descriptive evaluation* – Giving general information about the changes that have occurred in the data as a result of generalization.

The methods of line resolution verification and accuracy assessment, according to the article (Burghardt et al. 2014), can be classified into each of the three aforementioned groups of evaluation in cartographic generalization.

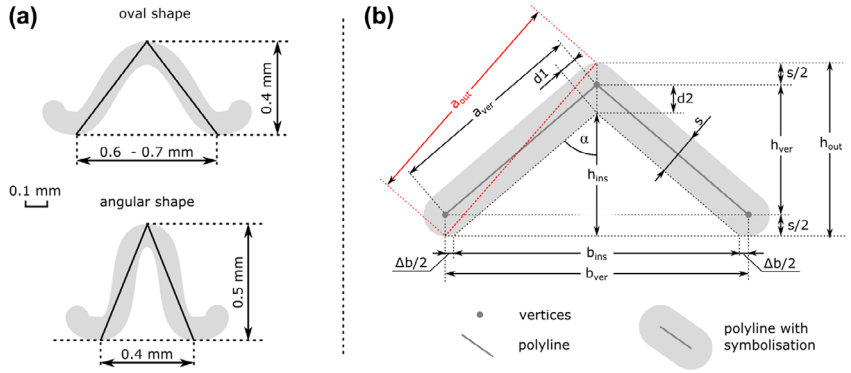


Figure 1. (a) The minimum line dimensions on the map given by Saliszczew (2003), (b) the markers of elementary triangles measures used by the authors (out – outside, ins – inside, ver – vertices) – other explanations are located in the text.

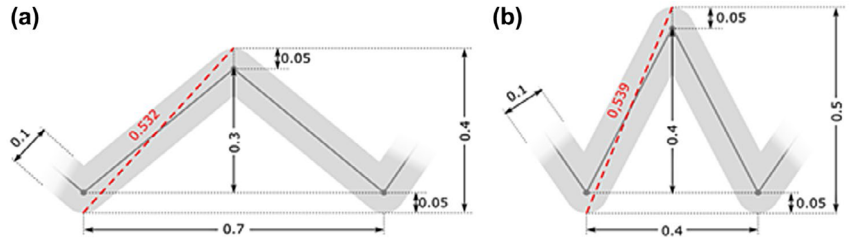


Figure 2. The triangles described on the line, which retain the minimum dimensions of elementary triangles. (a) an oval shape, b) an angular shape.

2. Determination of the minimum dimensions of any arbitrary line using an elementary triangle

In the metric space of a map which preserves the Euclidean's metric assumptions, each line can be converted into a finite number of segments. Their course is described using triangle inequality, expressed by the following metric:

$$d(x,z) \leq d(x,y) + d(y,z) \vee x,y,z \in X \tag{1}$$

where the distance between x and z is not greater than the sum of the distances between x and y as well as y and z.

Moreover, each arbitrary line can also be described as a finite number of triangles which are built on three adjacent vertices. A line described using triangles is subjected to verification of its resolution. For verification purposes, two versions of the line shape were used: *oval* and *angular* (Figure 1) (Saliszczew 1998).

The triangle height h_{out} and base length b_{ver} (Figure 1(b)), which are given by Saliszczew, were extended with the minimum dimensions of the shorter triangle side a_{out}. It should also be mentioned that the elementary triangles were defined for lines with 0.1 mm width (Figure 2). Moreover, the dimension of 0.6 mm base length, defined by Saliszczew, (Figure 1(a)) was also included.

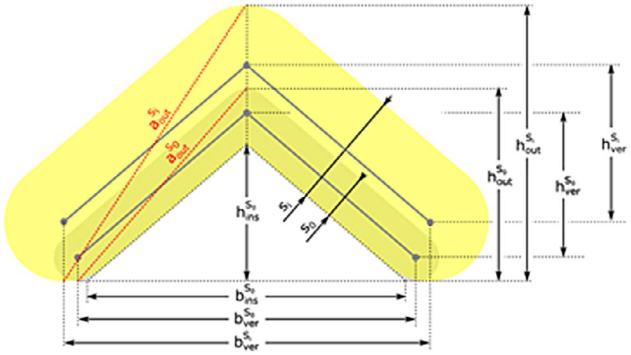


Figure 3. The comparison of the minimum dimensions of elementary triangles for two line widths.

Table 1. The values of minimum dimensions of an elementary triangle for sample line widths. The minimum base values for three types of triangles are marked in green. Red denotes minimum length of triangle sides. All dimensions are in millimetres.

	Oval triangle 1 (ow_1)			Oval triangle 2 (ow_2)			Angular triangle (kan)		
s_i	Δb^{s_i}	$b^{s_i}_{ver}$	$a^{s_i}_{out}$	Δb^{s_i}	$b^{s_i}_{ver}$	$a^{s_i}_{out}$	Δb^{s_i}	$b^{s_i}_{ver}$	$a^{s_i}_{out}$
	–	0.663	0.461		0.558	0.424	–	0.338	0.447
0.1	0.037	0.663	0.532	0.041	0.558	0.500	0.062	0.338	0.539
0.25	0.092	0.755	0.687	0.104	0.662	0.669	0.155	0.493	0.782
0.5	0.185	0.848	0.9619	0.207	0.766	0.9623	0.309	0.647	1.192
1	0.370	1.033	1.532	0.414	0.973	1.564	0.618	0.956	2.014
2	0.740	1.403	2.694	0.828	1.387	2.781	1.236	1.574	3.660
3	1.110	1.773	3.863	1.243	1.801	4.003	1.854	2.192	5.307

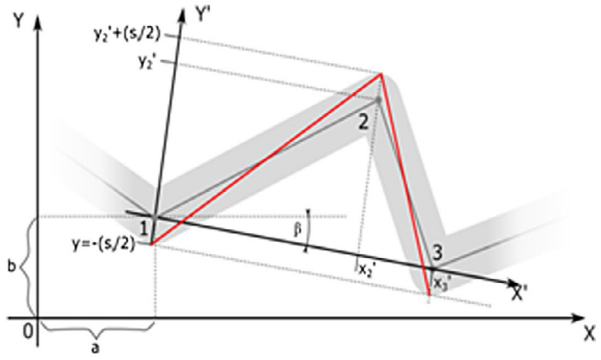


Figure 4. Transformation of the coordinates of vertices. Red line indicates the calculated shorter side of the triangle.

2.1. Extension of the minimum dimensions of lines of any width

The issue of the minimum object dimensions has been raised repeatedly in the literature (Saliszczew 2003; Speiss 2005). However, these studies have not included variable line widths or problems with their recognizability on maps. This chapter contains a discussion on this topic, showing proprietary solutions in this area.

Seite: 5

Verfasser: Christiane Thema: Notiz Datum: 21.06.2019 22:51:57

Die Tabelle besitzt entgegen der Angabe in der Tabellenüberschrift keine grün und rot markierten Bereiche. Auch in der Online-Version ist dies nicht der Fall. Die gelb markierten Zahlenwerte sind nach Meinung der Bearbeiterin falsch. In der Spalte b_ver, zweite Zeile, geben sie die Werte nicht für b_ver (b_ver wäre bei ow1 0,7mm, bei ow2 0,6mm, bei kan 0,4mm, d.h. die Dreiecksbasis bezogen auf die Punktgeometrie), sondern für b_ins (Dreiecksbasis der lichten Öffnung, also unter Berücksichtigung der Strichbreite) an. In der Spalte a_out, Zeile 1, rechnen sie nicht mit dem Wert in der Spalte b_ver, sondern mit den Werten 0,7, 0,6 und 0,4. a_out müsste in der ersten Zeile aber mit 0,663, 0,559 und 0,338 gerechnet werden, da im Fall der reinen Geometrie, ohne Strichstärke, b_ver = b_ins ist. b_ins aber ist nicht 0,7, 0,6 oder 0,4, denn Saliszczew hat die Strichstärke nicht berücksichtigt, auch nicht Strichstärke 0,1, vgl. Fig. 1a und 2.

Verfasser: Christiane Thema: Hervorheben Datum: 01.11.2017 22:06:01 +01'00'

Verfasser: Christiane Thema: Hervorheben Datum: 01.11.2017 22:06:03 +01'00'

Verfasser: Christiane Thema: Hervorheben Datum: 01.11.2017 22:06:05 +01'00'

Verfasser: Christiane Thema: Hervorheben Datum: 01.11.2017 21:52:57 +01'00'

Verfasser: Christiane Thema: Hervorheben Datum: 01.11.2017 21:53:01 +01'00'

Verfasser: Christiane Thema: Hervorheben Datum: 01.11.2017 21:53:06 +01'00'

Based on the assumptions of minimum dimensions for an elementary triangle (line width $s_0 = 0.1$ mm), the authors made necessary transformations for broadening these statements on lines of any width ($s_i \geq 0.1$ mm).

The function s_i has been developed to take account of the invariability of an elementary triangle's (line width $s_0 = 0.1$ mm) interior dimension (*aperture*[2]). The interior (aperture) of the triangle aims to unambiguously determine the dimensions of base b_{ins} and height h_{ins} (Figure 1(b)).

The condition for elementary triangles, having s_0 *i* s_i widths take the following form:

$$\begin{cases} h_{ins}^{s_i} = h_{ins}^{s_0} \\ b_{ins}^{s_i} = b_{ins}^{s_0} \end{cases} \tag{2}$$

To determine the height $h_{ins}^{s_0}$, which includes the *aperture*, two lengths were calculated: $d_1^{s_0}$ *i* $d_2^{s_0}$:

$$tg\alpha = \frac{b_{ver}^{s_0}/2}{h_{ver}^{s_0}} = \frac{s_0/2}{d_2^{s_0}} \rightarrow d_1^{s_0} = s_0 \frac{h_{ver}^{s_0}}{b_{ver}^{s_0}}. \tag{3}$$

$$\left(d_2^{s_0}\right)^2 = \left(d_1^{s_0}\right)^2 + \left(\frac{s_0}{2}\right)^2 = \left(s_0 \frac{h_{ver}^{s_0}}{b_{ver}^{s_0}}\right)^2 + \left(\frac{s_0}{2}\right)^2 = \left(\frac{s_0}{2b_{ver}^{s_0}}\right)^2 \left[4\left(h_{ver}^{s_0}\right)^2 + \left(b_{ver}^{s_0}\right)^2\right] \tag{4}$$

that is:

$$\left(d_2^{s_0}\right)^2 = \left(\frac{s_0}{2b_{ver}^{s_0}}\right)^2 \left[4\left(h_{ver}^{s_0}\right)^2 + \left(b_{ver}^{s_0}\right)^2\right]. \tag{5}$$

Internal height (in the aperture) of the triangle is:

$$h_{ins}^{s_0} = h_{out}^{s_0} - d_2^{s_0} - \frac{s_0}{2} = h_{ver}^{s_0} - d_2^{s_0} + \frac{s_0}{2} \tag{6}$$

The internal base of the triangle is defined by the relationship:

$$\frac{b_{ins}^{s_0}}{h_{ins}^{s_0}} = \frac{b_{ver}^{s_0}}{h_{ver}^{s_0}} \rightarrow b_{ins}^{s_0} = h_{ins}^{s_0} \frac{h_{ver}^{s_0}}{h_{ver}^{s_0}}. \tag{7}$$

The correction of the triangle base length:

$$\Delta b^{s_0} = b_{ver}^{s_0} - b_{ins}^{s_0} \tag{8}$$

for line width s_i (Figure 1), the following corrections were calculated (Figure 3):

- the length of the base Δb^{s_i}:

$$\frac{\Delta b^{s_i}}{\Delta b^{s_0}} = \frac{s_i}{s_0} \rightarrow \Delta b^{s_i} = \Delta b^{s_0} \frac{s_i}{s_0} \tag{9}$$

- the length retains the *aperture* $d_2^{s_i}$:

$$\frac{d_2^{s_i}}{d_2^{s_0}} = \frac{s_i}{s_0} \rightarrow d_2^{s_i} = d_2^{s_0} \frac{s_i}{s_0} \tag{10}$$

- For an elementary triangle of any width (but with an aperture like a triangle with 0.1 mm width), the following were calculated:

Seite: 6

Verfasser: Christiane Thema: Hervorheben Datum: 21.06.2019 22:49:51
muss heißen: d1

Verfasser: Christiane Thema: Hervorheben Datum: 21.06.2019 22:49:45
muss heißen: b_ver

- bases:

$$b_{\text{ver}}^{s_i} = b_{\text{ins}}^{s_0} + \Delta b^{s_i} \tag{11}$$

- heights:

$$h_{\text{out}}^{s_i} = h_{\text{ins}}^{s_0} + d_2^{s_i} + \frac{s_i}{2} \tag{12}$$

- side:

$$a_{\text{out}}^{s_i} = \left[\left(h_{\text{out}}^{s_i}\right)^2 + \left(\frac{b_{\text{ver}}^{s_i}}{2}\right)^2 \right]^{0.5}. \tag{13}$$

Based on the Equations (2–13), the minimum dimensions of an elementary triangle were calculated (Table 1). The calculations covered the selected line widths s_i as well as its oval and angular shapes (Figure 1). There are also two types of oval triangles, due to the two base lengths given by Saliszczew 2003.

Based on the values collected in the above table, it can be seen that if line width is 2 mm, the base of the angular triangle is greater than the oval one. This would not have occurred if the invariability aperture condition (*line width* = s_0) had not been preserved. The minimum base, height and triangle side dimensions depend on the line width. Moreover, the verification of the line resolution process seems to confirm the need to distinguish these three measures.

The comparison of the shorter side in the oval triangle $a_{\text{out}}^{s_i}(ow_1)$, $a_{\text{out}}^{s_i}(ow_2)$ and its counterpart in the angular side $a_{\text{out}}^{s_i}(\text{kan})$ (Table 1) shows that both measures are dependent on triangle shape and line width s_i.

The aforementioned measures were used to define the criteria for verifying and assessing the accuracy of the data in the study of drawing line resolution. A line of any width s_i which can be described by triangles has three main parameters: height $h_j^{S_i}$, base b_j and shorter triangle side $a_j^{s_i}$. These values determine the drawing resolution, based on the following assumptions:

- The drawing line resolution is maintained if each triangle fulfils the following conditions (the resolution is not maintained in any other case):

$$a_j^{s_i} \geq a_{\text{out}}^{s_i}(\text{kan}) \wedge h_j^{s_i} \geq h_{\text{out}}^{s_i}(\text{kan}) \wedge b_j \geq b_{\text{ver}}^{s_i}(\text{kan}) \tag{14a}$$

$$a_j^{s_i} \geq a_{\text{out}}^{s_i}(ow_1) \wedge h_j^{s_i} \geq h_{\text{out}}^{s_i}(ow_1) \wedge b_j \geq b_{\text{ver}}^{s_i}(ow_1) \tag{14b}$$

$$a_j^{s_i} \geq a_{\text{out}}^{s_i}(ow_2) \wedge h_j^{s_i} \geq h_{\text{out}}^{s_i}(ow_2) \wedge b_j \geq b_{\text{ver}}^{s_i}(ow_2) \tag{14c}$$

In studies of drawing line resolution, there is additionally included the denominator of the map scale – M. The drawing line resolution conditions take the following form:

- for the first oval triangle (ow_1):

$$\varepsilon_{a1}^{s_i} = a_{\text{out}}^{s_i}(ow_1) \times 10^{-3} \times M[m], \quad \varepsilon_{h1}^{s_i} = h_{\text{out}}^{s_i}\left(ow_1\right) \times 10^{-3} \times M[m], \quad \varepsilon_{b1}^{s_i} = b_{\text{ver}}^{s_i}(ow_1) \times 10^{-3} \times M[m], \tag{15}$$

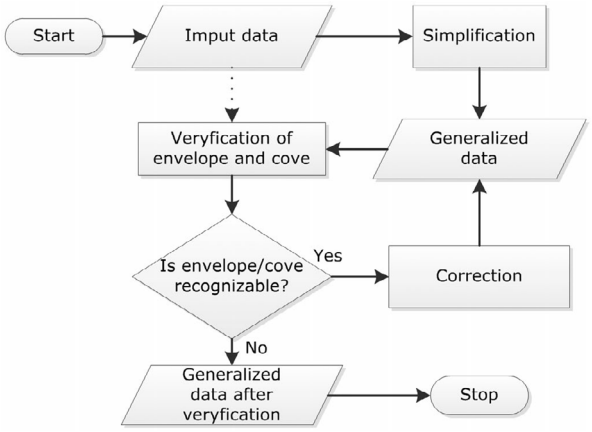


Figure 5. The schema of verification of drawing line resolution.

- for the second oval triangle (ow_2)

$$\varepsilon_{a2}^{s_i} = a_{\text{out}}^{s_i}(ow_2) \times 10^{-3} \times M[m], \quad \varepsilon_{h2}^{s_i} = h_{\text{out}}^{s_i}(ow_2) \times 10^{-3} \times M[m], \quad \varepsilon_{b2}^{s_i} = b_{\text{ver}}^{s_i}(ow_2) \times 10^{-3} \times M[m], \tag{16}$$

- for the angular triangle (kan)

$$\varepsilon_{a3}^{s_i} = a_{\text{out}}^{s_i}(\text{kan}) \times 10^{-3} \times M[m], \quad \varepsilon_{h3}^{s_i} = h_{\text{out}}^{s_i}(\text{kan}) \times 10^{-3} \times M[m], \quad \varepsilon_{b3}^{s_i} = b_{\text{ver}}^{s_i}(\text{kan}) \times 10^{-3} \times M[m], \tag{17}$$

The drawing line resolution is checked based on the line vertices and its width s_i. The next three vertices of the line (which are forming the triangle) are transformed to the local coordinate system (Figure 4), according to the relation below (18).

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x - a \\ y - b \end{bmatrix} \times \begin{bmatrix} \cos\beta - \sin\beta \\ \sin\beta + \cos\beta \end{bmatrix} \tag{18}$$

where:

$$\beta = \text{arctg}\frac{y_3 - y_1}{x_3 - x_1}, 0° < \beta < 90°$$

After the transformation, the new coordinates are used to calculate the length of the triangle sides (side 1–2):

$$a_j^{s_i}(1,\ 2) = \left[\left(x_2'\right)^2 + \left(y_2' + s_i\right)^2\right]^{0.5}. \tag{19}$$

side 2–3:

$$a_j^{s_i}(2,\ 3) = \left[\left(x_3' - x_2'\right)^2 + \left(y_2' + s_i\right)^2\right]^{0.5}. \tag{20}$$

The assessment of the simplification results is based on the shorter triangle side:

$$a_j^{s_i} = d_{\min}\left[a_j^{s_i}(1,\ 2),\ a_j^{s_i}\left(2\,,3_i\right)\right]. \tag{21}$$

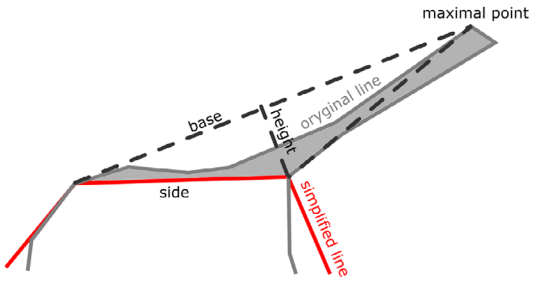


Figure 6. Compatibility of triangle dimensions (built on the cove) with the Saliszczew's standards.

2.2. The verification of drawing line resolution of any width on a map on any scale

The verification of drawing line resolution requires the standardization of the actual data into the form in which Saliszczew specified the minimum dimensions. The standardization uses the following transformation:

(X, d) – the metric space, wherein for any two points **F**, **N** and the number k (which takes the values of the oval and angular triangles) there is a relationship:

$$\mathbf{d}(\mathbf{F}', \mathbf{N}') = \mathbf{k}(\mathbf{F}, \mathbf{N}), \tag{22}$$

where: points $\mathbf{F}'$, $\mathbf{N}'$ are images of points **F**, **N**

d – the metric space (distance) of any two points of set ***X,***

k – the number, called ***the scale* or *the proportion*** of similarity

and the similarity scale **k*i*** (i = 1, 2, 3).

where:

$$k_1 = \frac{0.7M}{b_{\text{real}}}, \quad k_2 = \frac{0.6M}{b_{\text{real}}}, \quad k_3 = \frac{0.4M}{b_{\text{real}}}, \tag{23}$$

b_{real} – real length of the triangle base,

M – the scale denominator of a generalized map.

The resulting standardized data, in accordance with (22) is the basis of verification of drawing line resolution. Verification is performed by checking whether the condition (23) is maintained for the minimum dimensions specified by Saliszczew (2003). This, in turn, is done by comparing triangles built on three adjacent vertices with the dimensions of the elementary triangle.

Determination of drawing line L resolution is shown in the following verification schema (Figure 5). This schema takes into account:

- invariable object points, i.e.:
  - o envelopes in the shape of elementary triangles (base lengths: 0.4; 0.6; 0.7 mm),
  - o coves (e.g. sea water that cuts into the land) o in the shape of elementary triangles,
- the map scale 1:M after generalization,
- line width s_i on the map,
- criterion (23) for verifying the line resolution, based on the triangle sides' dimensions:

$$\bigwedge_{h_j^{si}, a_j^{si}, b_j \in L} \left\{ \begin{array}{l} \left[\left(k_1 a_j^{\,s_i} - \varepsilon_{a_1}^{s_i}\right) \cap \left(k_1 b_j - \varepsilon_{b_1}^{s_i}\right) \cap \left(k_1 h_j - h_{0.4}^{AS}\right) \right] \cup \left[\left(k_2 a_j^{s_i} - \varepsilon_{a_2}^{s_i}\right) \cap \left(k_2 b_j - \varepsilon_{b_2}^{s_i}\right) \cap \left(k_2 h_j - h_{0.4}^{AS}\right) \right] \cup \\ \cup \left[\left(k_3 a_j^{s_i} - \varepsilon_{b_3}^{s_i}\right) \cap \left(k_3 b_j - \varepsilon_{b_3}^{si}\right) \cap \left(k_3 h_j - h_{0.5}^{AS}\right) \right] \end{array} \right\} = \begin{vmatrix} 1 \\ 0 \end{vmatrix}$$

oval$_1$ (height h$_1$+ 70 mm base + side)

oval$_2$ (height h$_1$+ 60 mm base + side)

angular (heigth h$_2$ + 40 mm base + side)

Seite: 9

Verfasser: Christiane Thema: Notiz Datum: 21.06.2019 23:02:31

Die Minuszeichen sind hier möglicherweise ein Fehler im Druckprozess des Artikels. Im Pythonscript steht ein >= (z.B. Zeilen 438 - 469 im Original).
Im Skript lautet das Kriterium außerdem anders: standardisierte Basis und standardisierte Höhe müssen größer sein als im Elementardreieck ODER standardisierte Basis und standardisierter kürzerer Dreiecksschenkel müssen größer als im Elementardreieck sein.

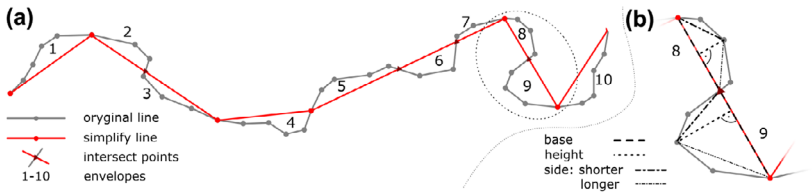


Figure 7. The method of assessing the accuracy of simplified data (a) the construction of the envelopes, (b) the triangles built on the envelopes.

where:

h_j – the real height of the triangle built on line L,
b_j – the real lengths of the triangle,
$h_{0.4}^{AS}$ – the norm of the oval triangle height by Saliszczew 2003,
$h_{0.5}^{AS}$ – the norm of the angular triangle height by Saliszczew,
$b_j^{0.7}$ – the norm of the base length in the first oval triangle Saliszczew,
$b_j^{0.6}$ – the norm of the base length in the second oval triangle Saliszczew,
$b_j^{0.4}$ – the norm of the base length in the angular triangle Saliszczew.

Line L, with a width of s_i, on the assumed scale 1:M (based on measures of envelopes and coves, which were formed after generalization) will preserve drawing resolution if the standardized triangle lengths meet the conditions (15–17) as well as (23) (logical sum equals 1). However, if one of the line segments fails to meet the condition (23), the resolution won't be maintained (logical sum equals 0).

Within the verification of the coves (Figure 6), it is required to maintain the compatibility of Saliszczew's minimum measures with corresponding measures of height, base and shorter triangle side. Measurement criterion will be met only if triangle sides are overlapped by the cove shoreline (and the base closes the triangle).

3. The method of assessing the accuracy of line simplification algorithms

Existing methods for assessing the quality of generalization are not objective. This is due to the fact that different generalization algorithms are based on different input parameters. Therefore, it is worth knowing how to compare the results of an algorithm based on the scale against an algorithm dependent on a tolerance threshold. Furthermore, the methods applied so far do not include the accuracy of lost data (information) as a result of their removal from generalized maps. Therefore, for the sake of objectivity and uniformity in the assessment, the authors have proposed a method based on:

- The aforementioned standards (standardized Saliszczew's dimensions),
- envelopes (defining the area of deleted data), i.e. triangles constructed according to the Figure 7,
- coves (as above – Figure 7),
- the average error of the shorter triangle sides (triangles built on envelopes – Figure 7(b)).

3.1. The assessment of simplified line accuracy

The accuracy of the simplified line is specified using the previously described standards (Chapter 2). If the assumptions are met (depending on the map scale) and the technical conditions (graphical loads, line resolution) (Chrobak et al. 2007) are not too burdersome, the map is readable. The analysis of line resolution is based on the minimum dimensions (Figure 1(a)) established by Saliszczew (the authors adapted those dimensions to digital cartography).

Seite: 10

Verfasser: Christiane Thema: Hervorheben Datum: 09.04.2019 10:13:46
Figure 7 - muss heißen Figure 6

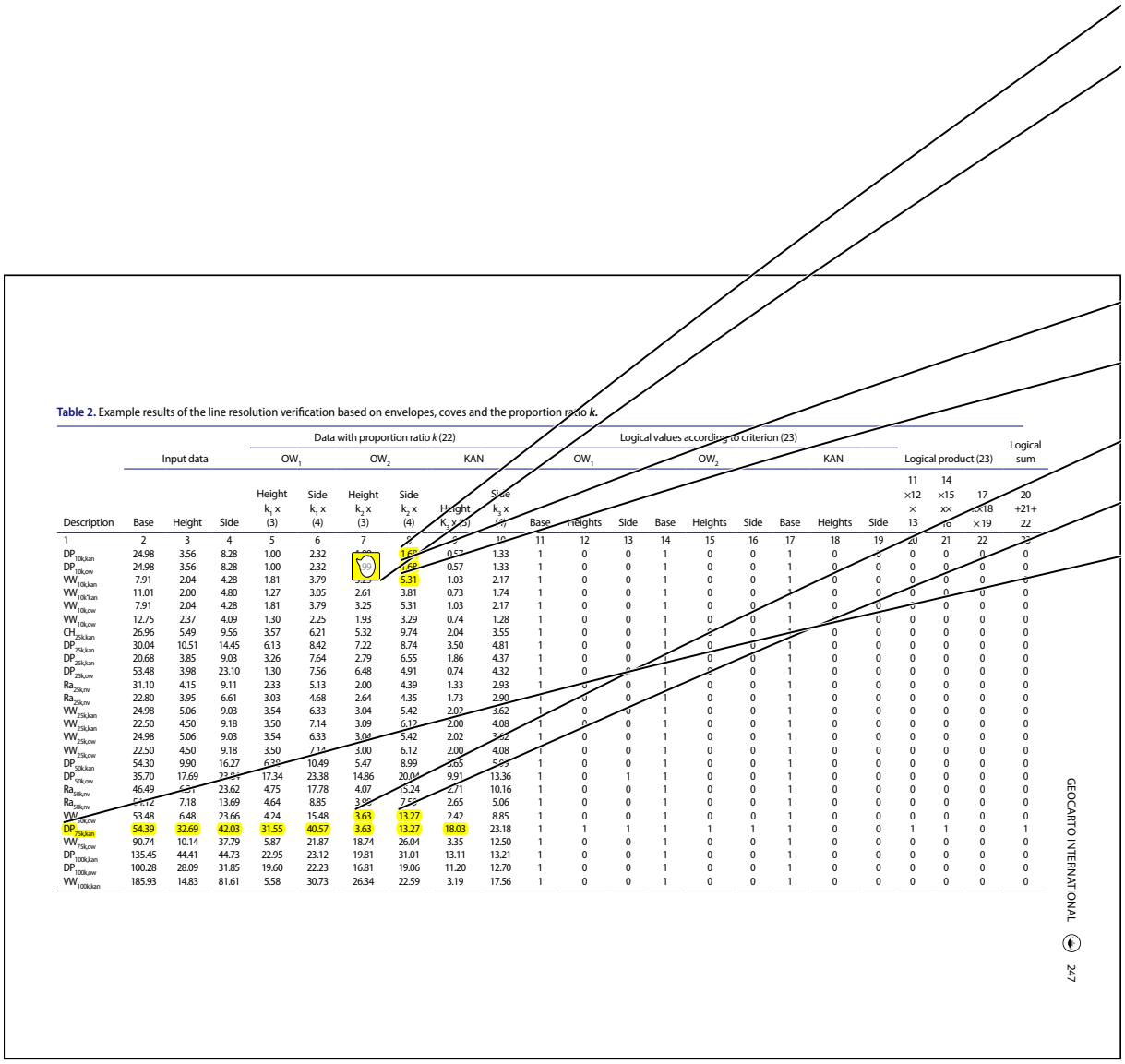


Table 2. Example results of the line resolution verification based on envelopes, coves and the proportion ratio ***k***.

	Input data			Data with proportion ratio *k* (22)						Logical values according to criterion (23)									Logical product (23)			Logical sum
				OW_1		OW_2		KAN		OW_1			OW_2			KAN			11 ×12 ×	14 ×15 ××	17 ××18 × 19	20 +21+ 22
Description	Base	Height	Side	Height k_1 x (3)	Side k_1 x (4)	Height k_2 x (3)	Side k_2 x (4)	Height k_3 x (3)	Side k_3 x (4)	Base	Heights	Side	Base	Heights	Side	Base	Heights	Side	13	16		
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
$DP_{10k,kan}$	24.98	3.56	8.28	1.00	2.32	1.88	1.68	0.57	1.33	1	0	0	1	0	0	1	0	0	0	0	0	0
$DP_{10k,ow}$	24.98	3.56	8.28	1.00	2.32	1.88	1.68	0.57	1.33	1	0	0	1	0	0	1	0	0	0	0	0	0
$VW_{10k,kan}$	7.91	2.04	4.28	1.81	3.79	3.25	5.31	1.03	2.17	1	0	0	1	0	0	1	0	0	0	0	0	0
$VW_{10k,kan}$	11.01	2.00	4.80	1.27	3.05	2.61	3.81	0.73	1.74	1	0	0	1	0	0	1	0	0	0	0	0	0
$VW_{10k,ow}$	7.91	2.04	4.28	1.81	3.79	3.25	5.31	1.03	2.17	1	0	0	1	0	0	1	0	0	0	0	0	0
$VW_{10k,ow}$	12.75	2.37	4.09	1.30	2.25	1.93	3.29	0.74	1.28	1	0	0	1	0	0	1	0	0	0	0	0	0
$CH_{25k,kan}$	26.96	5.49	9.56	3.57	6.21	5.32	9.74	2.04	3.55	1	0	0	1	0	0	1	0	0	0	0	0	0
$DP_{25k,kan}$	30.04	10.51	14.45	6.13	8.42	7.22	8.74	3.50	4.81	1	0	0	1	0	0	1	0	0	0	0	0	0
$DP_{25k,kan}$	20.68	3.85	9.03	3.26	7.64	2.79	6.55	1.86	4.37	1	0	0	1	0	0	1	0	0	0	0	0	0
$DP_{25k,ow}$	53.48	3.98	23.10	1.30	7.56	6.48	4.91	0.74	4.32	1	0	0	1	0	0	1	0	0	0	0	0	0
$Ra_{25k,nv}$	31.10	4.15	9.11	2.33	5.13	2.00	4.39	1.33	2.93	1	0	0	1	0	0	1	0	0	0	0	0	0
$Ra_{25k,nv}$	22.80	3.95	6.61	3.03	4.68	2.64	4.35	1.73	2.90	1	0	0	1	0	0	1	0	0	0	0	0	0
$VW_{25k,kan}$	24.98	5.06	9.03	3.54	6.33	3.04	5.42	2.02	3.62	1	0	0	1	0	0	1	0	0	0	0	0	0
$VW_{25k,kan}$	22.50	4.50	9.18	3.50	7.14	3.09	6.12	2.00	4.08	1	0	0	1	0	0	1	0	0	0	0	0	0
$VW_{25k,ow}$	24.98	5.06	9.03	3.54	6.33	3.04	5.42	2.02	3.62	1	0	0	1	0	0	1	0	0	0	0	0	0
$VW_{25k,ow}$	22.50	4.50	9.18	3.50	7.14	3.00	6.12	2.00	4.08	1	0	0	1	0	0	1	0	0	0	0	0	0
$DP_{50k,kan}$	54.30	9.90	16.27	6.39	10.49	5.47	8.99	3.65	5.99	1	0	0	1	0	0	1	0	0	0	0	0	0
$DP_{50k,ow}$	35.70	17.69	23.91	17.34	23.38	14.86	20.01	9.91	13.36	1	0	1	1	0	0	1	0	0	0	0	0	0
$Ra_{50k,nv}$	46.49	5.31	23.62	4.75	17.78	4.07	15.24	2.71	10.16	1	0	0	1	0	0	1	0	0	0	0	0	0
$Ra_{50k,nv}$	54.12	7.18	13.69	4.64	8.85	3.99	7.59	2.65	5.06	1	0	0	1	0	0	1	0	0	0	0	0	0
$VW_{50k,ow}$	53.48	6.48	23.66	4.24	15.48	3.63	13.27	2.42	8.85	1	0	0	1	0	0	1	0	0	0	0	0	0
$DP_{75k,kan}$	54.39	32.69	42.03	31.55	40.57	3.63	13.27	18.03	23.18	1	1	1	1	1	1	1	0	0	1	1	0	1
$VW_{75k,ow}$	90.74	10.14	37.79	5.87	21.87	18.74	26.04	3.35	12.50	1	0	0	1	0	0	1	0	0	0	0	0	0
$DP_{100k,kan}$	135.45	44.41	44.73	22.95	23.12	19.81	31.01	13.11	13.21	1	0	0	1	0	0	1	0	0	0	0	0	0
$DP_{100k,ow}$	100.28	28.09	31.85	19.60	22.23	16.81	19.06	11.20	12.70	1	0	0	1	0	0	1	0	0	0	0	0	0
$VW_{100k,kan}$	185.93	14.83	81.61	5.58	30.73	26.34	22.59	3.19	17.56	1	0	0	1	0	0	1	0	0	0	0	0	0

Seite: 11

Verfasser: Christiane Thema: Hervorheben Datum: 08.11.2017 20:06:11 +01'00'

Verfasser: Christiane Thema: Notiz Datum: 21.06.2019 22:37:57

1,99 ist das Ergebnis von k2 mal Side (8,28). Der korrekte Wert für k2 x Height fehlt. 1,68 für k2 x Side ist falsch.
k1 ist 0,7 mm x 10000 / 24,98 m = 0,28
k2 ist 0,6 mm x 10000 / 24,98 m = 0,24
k3 ist 0,4 mm x 10000 / 24,98 m = 0,16

So laut Gleichung (23), S. 245.

Für 5,31 ist ebenfalls falsch. Es müsste den Wert 3,25 aus der Spalte links davon haben.

Verfasser: Christiane Thema: Hervorheben Datum: 08.11.2017 20:06:13 +01'00'

Verfasser: Christiane Thema: Hervorheben Datum: 08.11.2017 20:16:25 +01'00'

Verfasser: Christiane Thema: Hervorheben Datum: 21.04.2019 20:49:08

3.63

Verfasser: Christiane Thema: Hervorheben Datum: 21.04.2019 20:49:10

13.27

Verfasser: Christiane Thema: Hervorheben Datum: 21.04.2019 20:42:35

DP75k,kan 54.39 32.69 42.03 31.55 40.57 3.63 13.27 18.03

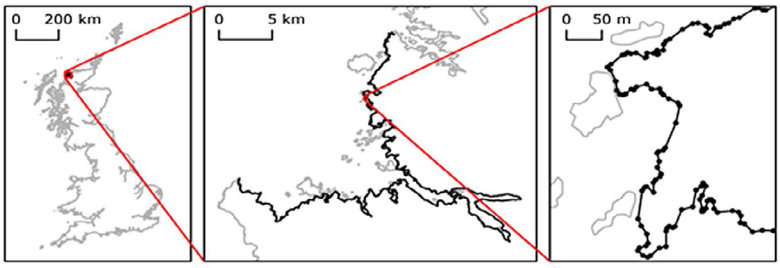


Figure 8. The coastline of Eddrachillis bay used in the tests (black). The background of entire layer (grey).

The proposed method of assessment is based on a comparison between standardized real dimensions (base, height and shorter triangle side) and the dimensions obtained from an elementary triangle (Table 2). The shorter side of each envelope acts as a metric representation of the resolution of the rejected (original) line segments. The side length depends on the scale, width and shape of the lines on the map. This fact in the accuracy assessing is characterized by random errors.

The accuracy of the simplified line is defined by the following relationship:

$$m_{d_{\text{env}}} = \sqrt{\frac{\sum_{i=1}^{i=n} V_{ai} V_{ai}}{n-1}}, \tag{25}$$

where: $V_i = a_i \quad i = 1,\ 2,\ 3, \ldots, n$

Moreover, shorter sides in triangles which replaced the removed segments from the source line are random errors V_{a_i} in Equation (25).

When condition 23 is not satisfied, i.e. the line is not recognizable, the process repeats until the line reaches recognizability. Maintaining the condition of recognizability (23) allows for the assessment of the generalized data with Equation (25). Furthermore, in each envelope the maximum height's vertex is determined (Figure 6) with its coordinates located on the original line. These coordinates are useful in determining the shift direction of the line causing the graphical conflict.

The data accuracy m_{simp} of the simplified line is the sum of the squares of average lengths' errors:

$$m_{d_{\text{simp}}}{}^2 = m_{d_{\text{src}}}{}^2 + m_{d_{\text{env}}}{}^2 + m_{\text{cov}}{}^2 \tag{26}$$

where:

m_{src} – the source data,
m_{env} – unrecognizable shorter envelope side,
m_{cov} – unrecognizable shorter cove side (Equation 25).

3.2. The experiment assumptions

Experiments were conducted based on four commonly used simplification algorithms:

- Douglas–Peucker (1973),
- Visvalingam–Whyatt (1993),
- Chrobak (2000, 2010),
- Raposo (2013).

Moreover, input parameters were determined in accordance with Saliszczew's norm. This makes it possible to retain the consistency of comparison.

Table 3. The average lengths errors of shorter sides in envelope triangles.

Scale	Ra Av. len. errors m_d (m)	CH Av. len. errors m_d (m)	VW Av. len. errors m_d (m)	DP Av. len. errors m_d (m)
1	5	3	6	4
1:10,000	–	±2.8	±4.5	±8.0
1:25,000	±6.2	±5.4	±7.5	±11.0
1:50,000	±8.0	±8.5	±10.4	±13.8
1:75,000	±9.6	±10.8	±12.8	±17.2
1:100,000	±11.0	±12.7	±15.3	±20.2

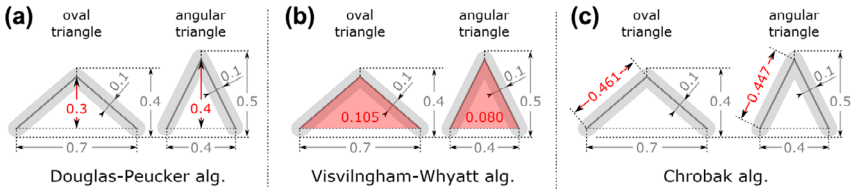


Figure 9. The method of calculating threshold parameters for algorithms and their geometrical interpretation (in red). The threshold parameter for VW algorithm (in mm^2).

Table 4. Threshold parameters of compared algorithms.

Parameter	$\varepsilon_j^{DP}(ow_1)$	$\varepsilon_j^{VW}(ow_1)$	$\varepsilon_j^{Ch}(ow_1)$	ε_j^{DP}(kan)	ε_j^{VW}(kan)	ε_j^{Ch}(kan)	ε_j^{Ra}
Scale	(m)	(m^2)	(m)	(m)	(m^2)	(m)	(m)
1:1	0.3×10^{-3}	0.105×10^{-6}	0.461×10^{-3}	0.4×10^{-3}	0.080×10^{-6}	0.447×10^{-3}	0.5×10^{-3}
1:10,000	3	10.5	4.61	4	8	4.47	5*
1:25,000	7.5	65,625	11,525	10	50	11,175	12.5
1:50,000	15	262.5	23.05	20	200	22.35	25
1:75,000	22.5	590,625	34,575	30	450	33,525	37,5
1:100,000	30	1050	46.1	40	800	44.7	50
1:200,000	60	4200	92.2	80	3200	89.4	100
1:300,000	90	9450	138.3	120	7200	134.1	150
1:400,000	120	16,800	184.4	160	12,800	178.8	200
1:500,000	150	26,250	230.5	200	20,000	223.5	250
1:1,000,000	300	105,000	461	400	80,000	447	500
1:2,000,000	600	420,000	922	800	320,000	894	1000
1:3,000,000	900	945,000	1383	1200	720,000	1341	1500
1:4,000,000	1200	1,680,000	1844	1600	1,280,000	1788	2000

*Unused parameter – description in text.

The tests were carried out using a test object *Eddrachillis bay* situated off the coast of Scotland. The data-set used © Ordnance Survey is licenced by Crown copyright 2011. The coastline object was obtained from *high_water_polyline* layer (Boundary-line™ data-set). The analysed fragment in the background of the entire class is shown in Figure 8.

The object has a complicated course (164,263 m length and 11,716 vertices). Simplification of the coastline was made for scales: 1: 10, 1: 25, 1: 50, 1: 75 and 1: 100 k.

Based on the methods proposed in Chapter 2, the logical sum of condition 23 was checked (Table 2, columns 23).

The study also includes an accuracy assessment of a simplified line which is based on deleted envelopes and coves formed between the original and the simplified line.

Many examples were analysed in order to draw the first observations:

- the standardized lengths fulfil the resolution criterion (23) if they implement the condition:

$$\Delta \varepsilon_i \leq 0.04 \times k_1 \times M \times 10^{-3}\,[m], \quad i = 1,\ 2,\ 3 \tag{27}$$

where:

- 0.04 the difference of shorter sides of elementary triangle (triangle bases: 0.6 and 0.7 mm)
- k_1 – the similarity scale in the Equation (22)
- cove shaded in Table 2 is recognizable and was subjected to user correction,
- the range of the envelope determines the map area which was lost after generalization,
- this area is a place of frequent graphic conflicts, caused by combining thematic overlays in the digital map editing,
- the coves (Figure 6) of the generalized line are presented or eliminated, depending on their size on the generalized map.

3.3. The parameters of tested algorithms

The average length errors of shorter sides in envelope triangles are shown in Table 3. These results were obtained from the tested algorithms, taking into account various scale denominators and 0.1 mm line width. The generalization covered four well-known algorithms for variable operating scales.

To calculate the Douglas–Peucker (DP) algorithm the *Simplify Line* tool (*remove point* method) available in 'ArcGIS 10.2 software' was used. The algorithms of Visvalingam–Whyatt and Chrobak (WV) were implemented by the authors using Python and the *arcpy* package. Data simplification using Raposo's algorithm (Ra) was performed using the original author's code (available at http://github.com/paulojraposo/HexQuant). It should be noted that in the case of Raposo's algorithm, the calculations for the scale 1:10k could not be realized because of their excessive computational complexity which stems from the lack of code optimization for large datasets. The Ra algorithm was used in its original form, whereas its tolerance parameter *tessera width* ε_j^{Ra} was calculated based on the formula given by author:

$$\varepsilon_j^{Ra} = 5 \times s_i \times M_j[m] \tag{28}$$

where: $s_i = 0.1[mm]$ – line width, M_j – scale denominator (signs were modified from the original)

The threshold parameters of other algorithms were configured to the dimensions of the elementary triangle (line width = 0.1 and two line shapes: oval (triangle base = 0.7) and angular) (Kozioł & Szombara 2013).

The DP algorithm is based on a tolerance threshold parameter ε_j^{DP}. This parameter was determined based on the height of elementary triangles $h_{ver}^{s_0}(ow_1)$, $h_{\text{ver}}^{s_0}(\text{kan})$ whose values were 0.3 and 0.4 mm (Figure 9(a)).

The threshold DP algorithm parameters were calculated based on formulas below:

$$\begin{aligned} \varepsilon_j^{DP}(ow_1) &= 0.3 \times M_j \times 10^{-3}[m], \\ \varepsilon_j^{DP}(kan) &= 0.4 \times M_j \times 10^{-3}[m] \end{aligned} \tag{29}$$

The VW algorithm is based on threshold parameters, which are calculated as *effective areas*. The effective areas are the surfaces of triangles with bases $b_{\text{ver}}^{s_0}(ow_1)$, $b_{\text{ver}}^{s_0}(\text{kan})$ and heights $h_{\text{ver}}^{s_0}(ow_1)$, $h_{\text{ver}}^{s_0}(\text{kan})$ (Figure 8(b)). These surfaces are 0.105 *i* 0.080 mm^2. For individual scales, the threshold parameters for the algorithm VW are calculated using the two following formulas:

$$\begin{aligned} \varepsilon_j^{VW}(ow_1) &= 0.105 \times M_j \times 10^{-6}[m^2], \\ \varepsilon_j^{VW}(kan) &= 0.080 \times M_j \times 10^{-6}[m^2] \end{aligned} \tag{30}$$

Seite: 14

Verfasser: Christiane Thema: Notiz Datum: 07.11.2017 19:12:18 +01'00'
in Tabelle 2 gibt es keine Schattierung

Verfasser: Christiane Thema: Hervorheben Datum: 09.04.2019 10:16:09
Figure 8(b)). Muss heißen Figure 9b

Table 5. The number of line vertices after the simplification and the percentage of variation.

	Algorithm						
Scale	DP_{ov1}	VW_{ov1}	Ch_{ov1}	DP_{ang}	VW_{ang}	Ch_{ang}	Ra
1:10,000	6589 (56.2%)	10,476 (89.4%)	11,469 (97.9%)	5389 (46%)	10,929 (93.3%)	11,504 (98.2%)	
1:25,000	3310 (28.3%)	5700 (48.7%)	8470 (72.3%)	2643 (22.6%)	6325 (54%)	8636 (73.7%)	9339 (79.7%)
1:50,000	1866 (15.9%)	2826 (24.1%)	4866 (41.5%)	1405 (12%)	3291 (28.1%)	5021 (42.9%)	6529 (55.7%)
1:75,000	1276 (10.9%)	1878 (16%)	3244 (27.7%)	978 (8.3%)	2178 (18.6%)	3345 (28.6%)	4846 (41.4%)
1:100,000	978 (8.3%)	1401 (12%)	2369 (20.2%)	760 (6.5%)	1602 (13.7%)	2460 (21%)	3821 (32.6%)
1:200,000	499 (4.3%)	665 (5.7%)	1102 (9.4%)	353 (3%)	769 (6.6%)	1140 (9.7%)	2036 (17.4%)
1:300,000	319 (2.7%)	416 (3.6%)	708 (6%)	257 (2.2%)	496 (4.2%)	724 (6.2%)	1363 (11.6%)
1:400,000	257 (2.2%)	302 (2.6%)	512 (4.4%)	187 (1.6%)	349 (3%)	529 (4.5%)	1031 (8.8%)
1:500,000	203 (1.7%)	246 (2.1%)	401 (3.4%)	150 (1.3%)	280 (2.4%)	409 (3.5%)	851 (7.3%)
1:1,000,000	91 (0.8%)	108 (0.9%)	178 (1.5%)	64 (0.5%)	127 (1.1%)	183 (1.6%)	428 (3.7%)
1:2,000,000	58 (0.5%)	41 (0.3%)	76 (0.6%)	35 (0.3%)	50 (0.4%)	82 (0.7%)	216 (1.8%)
1:3,000,000	31 (0.3%)	25 (0.2%)	51 (0.4%)	20 (0.2%)	35 (0.3%)	54 (0.5%)	148 (1.3%)
1:4,000,000	20 (0.2%)	20 (0.2%)	32 (0.3%)	18 (0.2%)	22 (0.2%)	33 (0.3%)	104 (0.9%)

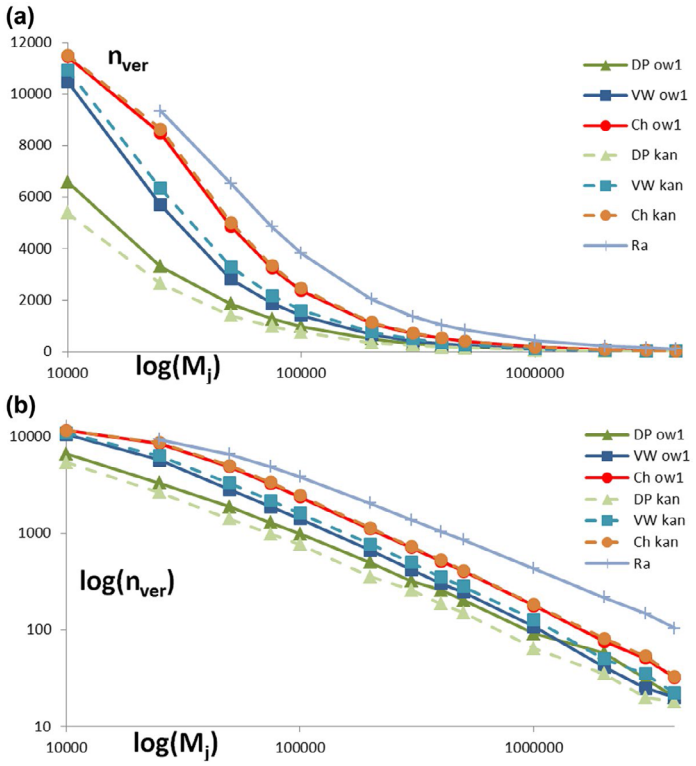


Figure 10. The number of simplified line vertices and the logarithm of this number as a logarithmic function the map scale denominator.

Diese Seite wurde freigelassen, um die folgenden Textseiten und ihre zugehörigen Kommentare auf gegenüberliegenden Seiten zu platzieren.

The Ch algorithm threshold parameters are defined by triangle sides $b_{\text{ver}}^{s_0}(ow_1)$ and heights $h_{\text{ver}}^{s_0}(ow_1)$, $h_{\text{ver}}^{s_0}(\text{kan})$ (Figure 8(c)). The side lengths are 0.461 i 0.447 mm. For individual scales, the threshold parameters for the algorithm Ch are calculated using the two following formulas:

$$\begin{aligned} \varepsilon_j^{Ch}(ow_1) &= 0.461 \times M_j \times 10^{-3}[m], \\ \varepsilon_j^{Ch}(kan) &= 0.447 \times M_j \times 10^{-3}[m] \end{aligned} \tag{31}$$

In Table 4 shows the calculated parameter values for the 13 tested scales. All calculations were performed in ArcGIS 10.2 environment.

The overall conclusion is that the approach of the presented method for parameter calculation is universal. We compared the three different algorithms (controlled by various parameters), independently of their original parameters. Each of the original algorithm parameters was made dependent on the target scale and line width (the Raposo algorithm respects by definition this assumption). So, we can say that the comparison was objective and can be used to compare other algorithms.

3.4. Comparison of the algorithms' results

Table 5 depicts the number of generalized line vertices. As was previously mentioned, the authors used four algorithms and 13 levels of scale. Figure 10 shows the number of vertices and the logarithm of the number of vertices in the logarithmic function of the map scale denominator. The use of a logarithmic scale (horizontal axis) results from a wide range of generalization scales.

The results shown in Table 3 are compatible in an ordinal scale with elements of Table 5.

To illustrate the differences between the results in terms of the number of vertices the Radical Law (Töpfer and Pillewizer 1966) was used. The law was applied to the results showing the number of vertices for subsequent pairs of simplification scales (starting from 1:25 k due to lack of results of 1:10 k scale for algorithm R). Li (2007) stated that this law can be written as follows:

$$n_j = n_{j-1} \times \left(\frac{M_{j-1}}{M_j}\right)^p \rightarrow p = \frac{\lg\left(n_j/n_{j-1}\right)}{\lg\left(M_{j-1}/M_j\right)} \tag{32}$$

where: n_j – the number of vertices in given scale 1:M_j, n_{j-1} – the number of vertices in the previously analysed scale 1:M_{j-1} (signs were modified from the original).

Yu (1993) stated that based on Equations (31), the value of p equals 2 for surface objects, 1.5 for points and 1 for line objects. This approach can be used to describe simplification algorithms. Similar conclusions about the Radical Law were part of the work of (Ratajski 1989). Figure 10 shows a plot of p-values calculated using (30) formula. Based on these results, it can be observed that:

- The p-value charts of the Raposo and Chrobak algorithms are smoother than the other two algorithms. The rationale behind this is that these algorithms maintain the shape of the simplifying line better than the others.
- On scales 1:300 and 1:400k most algorithms cause a decline of p-value (DP algorithm had this decline previously). This shows the close correlation between scale and the results of the simplification process
- Significant deviations from the trend line connecting the p-values for individual scales testifies to the declining line resolution of these scales
- As is shown in Figure 9, the p-values obtained in the author's study in most cases are close to 1. This in turn confirms their compliance with the values obtained in the study carried out by (Yu 1993).

Seite: 16

Verfasser: Christiane Thema: Hervorheben Datum: 09.04.2019 10:17:14
Figure 8(c) - muss heißen Figure 9c

Verfasser: Christiane Thema: Hervorheben Datum: 09.04.2019 10:21:21
Figure 10 - muss heißen Figure 11

Verfasser: Christiane Thema: Hervorheben Datum: 09.04.2019 10:19:47
(30) - muss heißen (32)

Verfasser: Christiane Thema: Hervorheben Datum: 09.04.2019 10:22:09
Figure 9, muss heißen Figure 11

Table 6. Summary results of the tested algorithms.

Scale	Algorithm	Number of envelopes	Number of recognizable envelopes	Number of recognizable coves	Average error of the side length	Minimum dimension of the triangle side
	2	3	3a	3b	4	5
10k	CH	731	10	2	3.29	4.61
	VW	754	6	0	5.34	
	DP	5218	13	5	9.08	
25k	CH	4074	25	5	5.97	11.52
	Ra	2257	16	1	7.51	
	VW	4545	7	0	8.66	
	DP	4930	39	16	12.96	
50k	CH	5273	12	1	10.31	23.05
	VW	4973	14	3	11.98	
	Ra	4237	5	1	10.60	
	DP	3598	36	12	18.27	
75k	CH	4836	9	0	14.11	34.58
	Ra	4638	0	0	14.14	
	VW	4298	20	5	15.73	
	DP	2898	39	15	27.80	
100k	CH	4251	10	1	18.25	46.10
	Ra	4460	0	0	17.77	
	VW	3630	14	5	20.50	
	DP	2453	32	9	27.73	
200k	Ra	3285	0	0	31.85	92.20
	CH	2827	3	1	33.10	
	VW	2292	11	3	35.32	
	DP	1446	14	2	48.45	
300k	Ra	2617	0	0	45.32	138.30
	CH	2062	2	0	48.37	
	VW	1704	8	1	50.25	
	DP	1142	10	5	64.30	
400k	Ra	2041	0	0	59.89	192.40
	CH	1609	2	0	63.96	
	VW	1334	4	1	65.77	
	DP	916	12	2	81.83	
500k	Ra	1835	0	0	72.51	230.50
	CH	1386	4	1	77.33	
	VW	1114	6	2	81.56	
	DP	739	8	3	99.50	

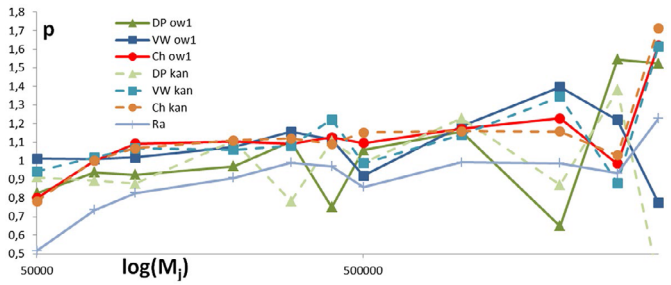


Figure 11. The *p*-value for each simplification result as a logarithmic function of the scale denominator.

3.5. The accuracy assessment

The minimum dimensions according to Saliszczew's 2003 and Chrobak's (2010) criteria were compared for the investigated algorithms. The following set of scales was used: 1:10, 1:25, 1:50, 1:75 and 1:100k. The results are described in Section 3.1 and the accuracy assessments are shown in Table 6.

Seite: 17

Verfasser: Christiane Thema: Hervorheben Datum: 06.11.2018 09:47:00 +01'00'
4.61. Bei meinem Test: ow1_a = 5,315, ow2_a = 5, kan_a = 5,385

Verfasser: Christiane Thema: Hervorheben Datum: 06.11.2018 09:36:08 +01'00'
5218; bei meinem Test ergeben sich 4585 envelopes.

This table shows the number of triangles representing the envelopes, where average side lengths and the bases have fulfilled the condition (23) (Table 6, column 3a). Column 4 shows the average side lengths, ordered from the lowest to the highest depending on the scales. The average value of the side lengths was calculated without coves (Figure 11).

In the assessment of the data accuracy, the standard deviation m_{dsimpl}, defined by Equation (26) appears as an indicator of the simplified line accuracy on digital maps.

The results from Tables 3 and 6 are ordered according to the algorithms used. For the assumed scales set, the DP algorithm achieved the worst results. In the scale 1:10 k, the smallest average length errors were produced by the CH algorithm. The results for scales 1:25–1:75 k were quite similar for the CH, R and VW algorithms (except DP). However, for small scales (1:100 k and smaller), the most favourable results were generated by the Ra method.

4. Conclusions

Based on the results presented in this article, it is concluded that the proposed method of verification of line resolution is objective, due to the restrictive data model preserving:

- The map space treated as Euclidean metric,
- the norm concerning minimum object dimensions, depending on the line width and the map scale,
- the scale and visibility of map objects,
- user-independent process and parameters.

The proposed method of evaluation includes the perception properties of the eye, related to three typical shapes (two types of oval line and one angular) and their minimal measures. Based on these properties, the authors defined the drawing line resolution condition (23).

According to Equation (26), the accuracy assessment of the generalized line is based on the measure which determines the amount of lost data.

The objective method of spatial data verification allows for the continued development of simplification algorithms which are necessary to process data in MRDB.

The developed methods could be used to:

(a) Verify the simplification algorithms currently used in digital cartographic generalization processes.
(b) Verify quality data characteristics:
  - Shape resolution, based on psychological user qualities,
  - Number of points rejected in the process,
  - The p-value, defined by Yu.
(c) Verify quantitative characteristics:
  - Process repeatability,
  - The average length errors of the shorter triangle side (dependent on the map scale).
(d) Graphic conflicts elimination.

The authors have also identified certain conditions for the proposed methods:

(a) The line width s_i on digital maps should be defined with an accuracy of 0.1 mm.
(b) The input data should be classified, hierarchized and ordered
(c) Every object should have invariable points (stable during the generalization)
(d) Data whose values are close to Saliszczew's minimum dimensions does not need standardization.

Seite: 18

Verfasser: Christiane Thema: Hervorheben Datum: 09.04.2019 10:24:33
Figure 11. Es kann sich nicht um Figure 11 handeln, sondern um eine nicht vorhandene Abbildung.

Notes

1. [GIS Dictionary, *support.esri.com*] *Resolution* – the detail with which a map depicts the location and shape of geographic features. The larger the map scale, the higher the possible resolution. As scale decreases, resolution diminishes and feature boundaries must be smoothed, simplified or not shown at all; for example, small areas may have to be represented as points.
2. as an analogy to the typography.

Disclosure statement

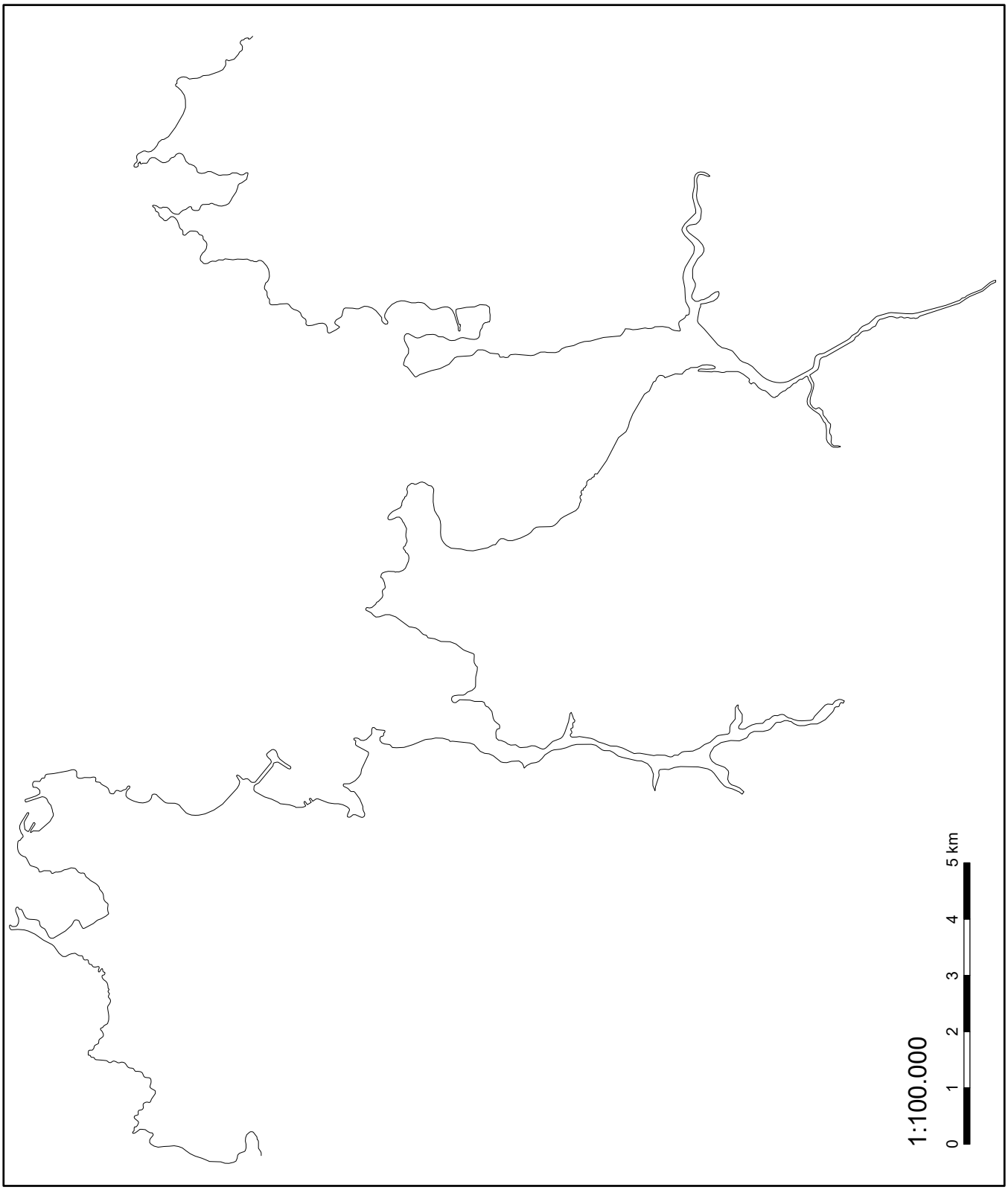
No potential conflict of interest was reported by the authors.

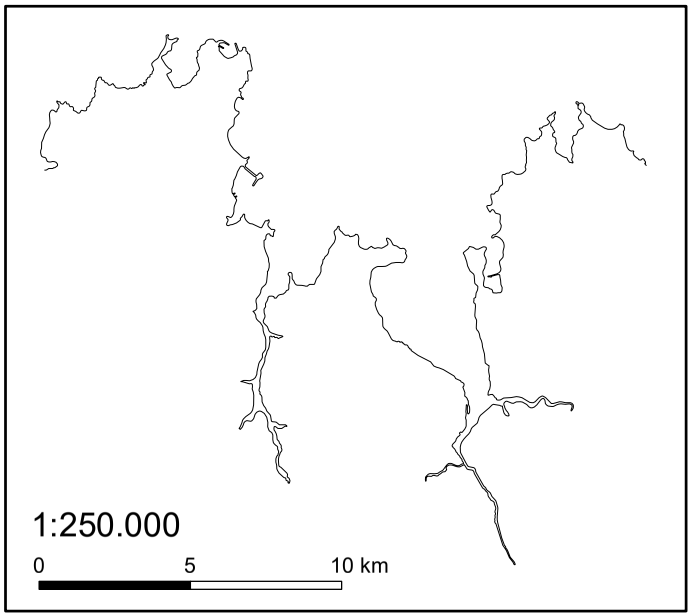
References

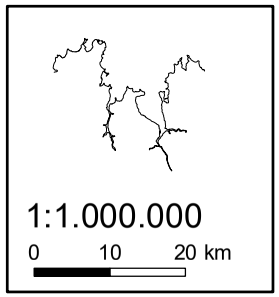
Bard S, Ruas A. 2005. Why and How Evaluating Generalised Data? [W:] In: Fisher PF, editor. Developments in spatial data handling. Berlin: Springer; p. 327–342.
Burghardt D, Duchêne C, Mackaness W. 2014. Methodologies and Applications of Map Generalisation. In: Burghardt Dirk, Duchene Cécile, Mackaness William, editors. Abstracting geographic information in a data rich world. Switzerland: Springer; p. 197–225.
Chrobak T. 2000. Modelling of spatial data in a database for the needs of cartographic generalization. Geodesy and Cartography. 49:7–19.
Chrobak T. 2010. The role of least image dimensions in generalized of object in spatial databases. Geodesy and Cartography. 59:99–120.
Chrobak T, Keller SF, Kozioł K, Szostak M, Żukowska M. 2007. Podstawy cyfrowej generalizacji kartograficznej [The basics of digital cartographic generalization]. wyd 1. Kraków: Uczelniane Wydawnictwa Naukowo-Dydaktyczne AGH.
Douglas DH, Peucker TK. 1973. Algorithms for the reduction of the number of points required to represent a digitised line or its caricature. Cartographica: The International Journal for Geographic information and Geovisualization. 10:112–122.
Kozioł K, Szombara S 2013. New method of creation data for natural objects in MRDB based on new simplification algorithm. [W:] In: Buchroithner M.F., editor. Proceedings of 26th International Cartographic Conference. Drezno. Sierpnia: International Cartographic Association; 25–30.
Li Z. 2007. Algorithmic foundation of multi-scale spatial representation. London: CRC Press.
Lupa M., Kozioł K., Leśniak A. 2015. An attempt to automate the simplification of building objects in multiresolution databases. [W:] In: Stanisław Kozielski, editor. Beyond Databases, Architectures and Structures: 11th international conference, Ustroń, Poland. Switzerland: Springer International Publishing; p. 448–459.
Mackaness WA. 1991. Integration and evaluation of map generalisation. In: Buttenfield BP, McMaster RB, editors. Map generalisation: making rules for knowledge representation. London: Longman; p. 217–226.
Mackaness WA, Ruas A. 2007. Evaluation in the Map Generalisation Process. [W:] In: Mackaness WA, Ruas A, Sarjakoski LT, editors. Generalisation of geographic information: cartographic modelling and applications. Amsterdam: Elsevier; p. 89–111.
McMaster RB. 1986. A statistical analysis of mathematical measures for linear simplification. Cartography and Geographic Information Science. 13:103–116.
Raposo P. 2013. Scale-specific automated line simplification by vertex clustering on a hexagonal tessellation. Cartography and Geographic Information Science. 40:427–443.
Ratajski L. 1989. Metodyka kartografii społeczno-gospodarczej [Methodology of the socio-economic cartography]. wyd 2. Warszawa: Państ. Przedsiębiorstwo Wydawnictw Kartograficznych im. Eugeniusza Romera.
Ruas A, Plazanet C. 1995. Data and knowledge modelling for generalisation. In: Müller JC, Lagrange JP, Weibel R, editors. GIS and generalisation: methodology and practice. London: Taylor and Francis; p. 73–90.
Saliszczew KA. 1998. General cartography. Ed. 2. Warsaw: PWN.
Speiss E. 2005. Swiss Society of Cartography: Topographic Maps. Maps Graphic and Generalisation. Cartographic Publication Series No. 17Töpfer, F., Pillewizer, W., 1966. The principles of selection Cartographic Journal. 3:10–16.
Stoter J., Zhang X., Stigmar H., Harrie L. 2014. Evaluation in Generalisation. In: Burghardt D, Duchene C, Mackaness W, editors. Abstracting geographic information in a data rich world. Switzerland: Springer; p. 259–297.
Topfer F, Pillewizer W. 1966. The Principles of Selection. The Cartographic Journal. 3:10–16.
Visvalingam M, Whyatt JD. 1993. Line generalisation by repeated elimination of points. The Cartographic Journal. 30:46–51.
Weibel R. 1993. Knowledge acquisition for map generalization: methods and prospects. Position paper in Proceedings of Specialist Meeting for Initiative 8 'Formalizing Cartographic Knowledge'. Buffalo, NY: National Center for Geographic Information and Analysis (NCGIA). p. 223–232.

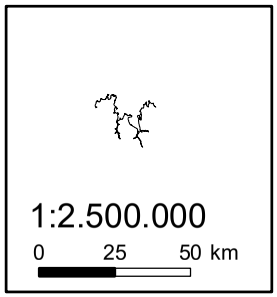
Weibel R. 1995. Three essential building blocks for automated generalisation. In: Müller J-C, Lagrange J-P, Weibel R, editors. GIS and generalisation: methodology and practice, vol 1 of Gisdata. London: Taylor and Francis; p. 56–69.
Weibel R. 1997. Generalization of spatial data: principles and selected algorithms. [W:] In: Van Kreveld M, Nievergelt J., Roos T., Widmayer P., editors. *Algorithmic Foundations of Geographic Information Systems*. Berlin: Springer; p. 99–152.
Yu Z. 1993. The effects of scale change on map structure [PhD]. Worcester, MA: Clark University.

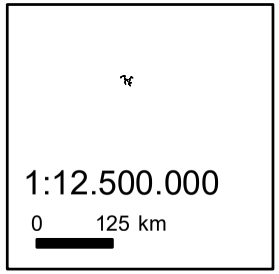
C1 Küstenabschnitt der Bretagne, ohne Vereinfachung



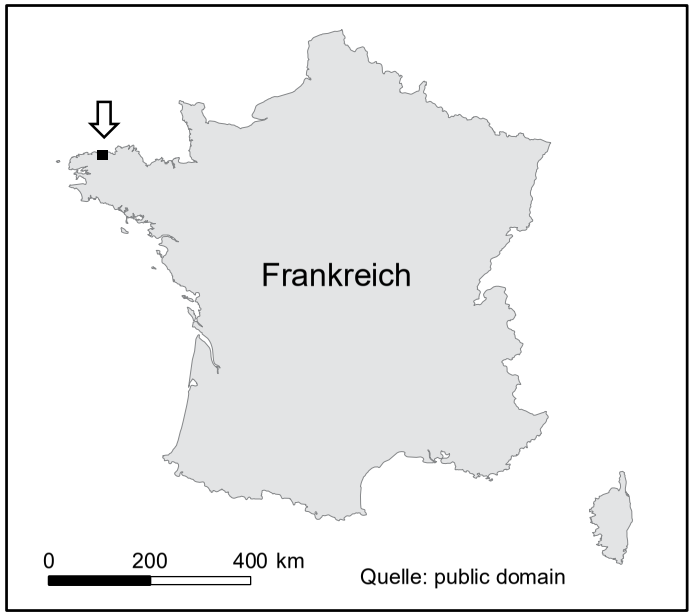




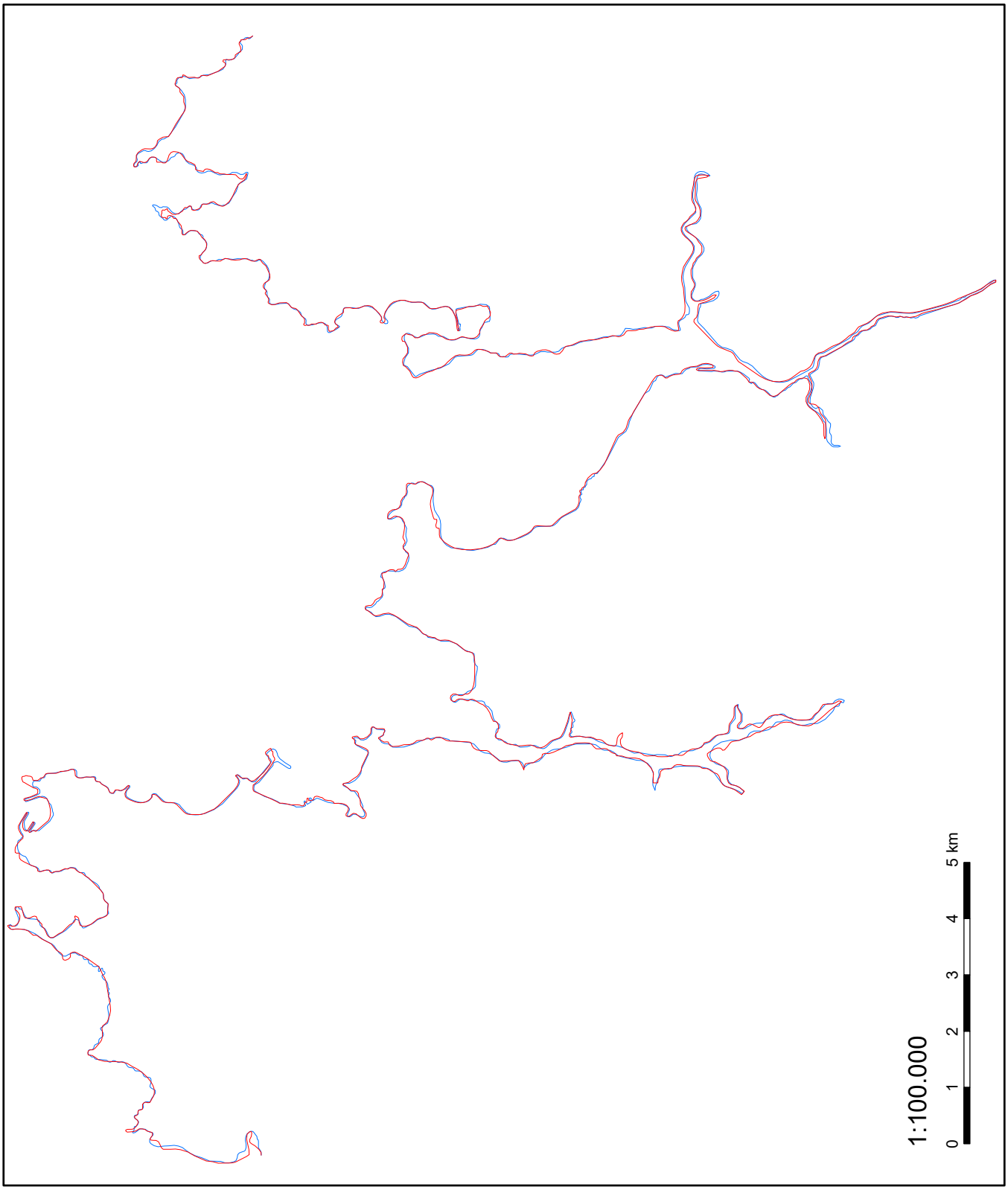


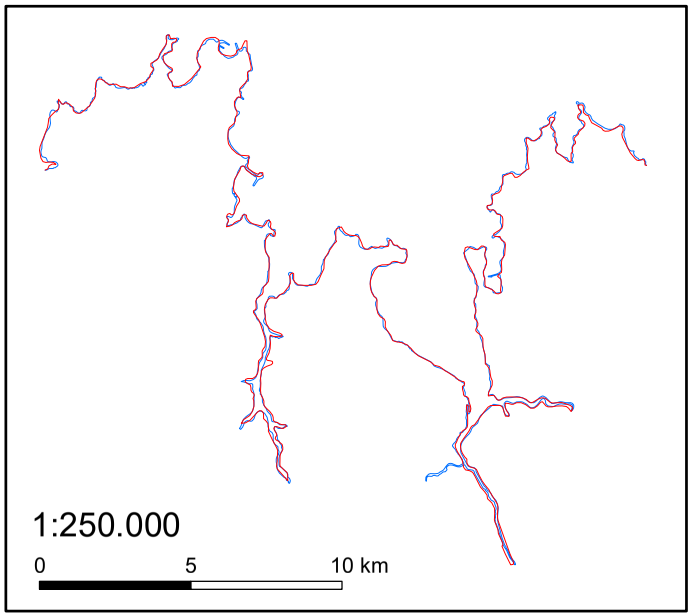


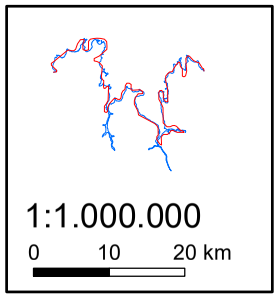
Quelle:
IGN Institut National de l'Information
Géographique et de Forestière (2017)

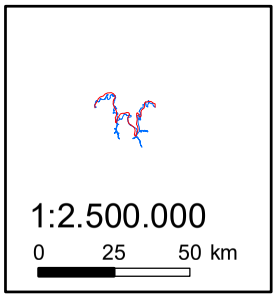


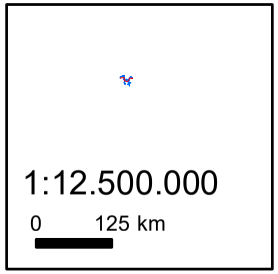
C2 Manuelle Linienvereinfachung nach Karten







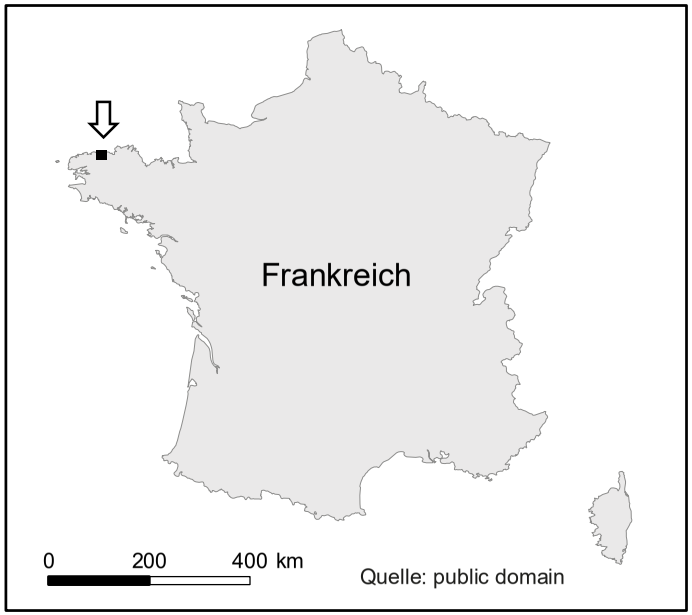




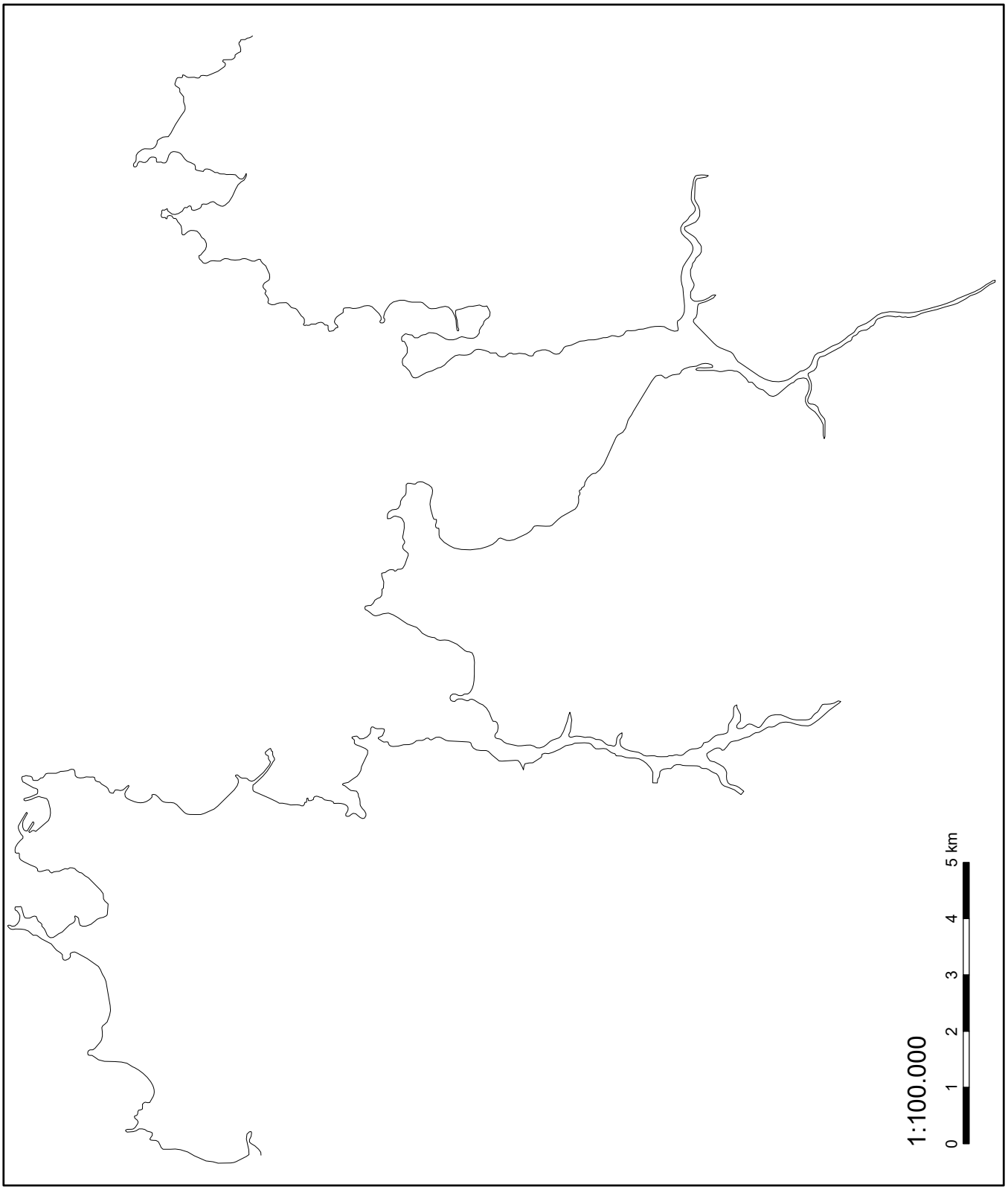
Originallinie ohne Anwendung von „Point Remove“

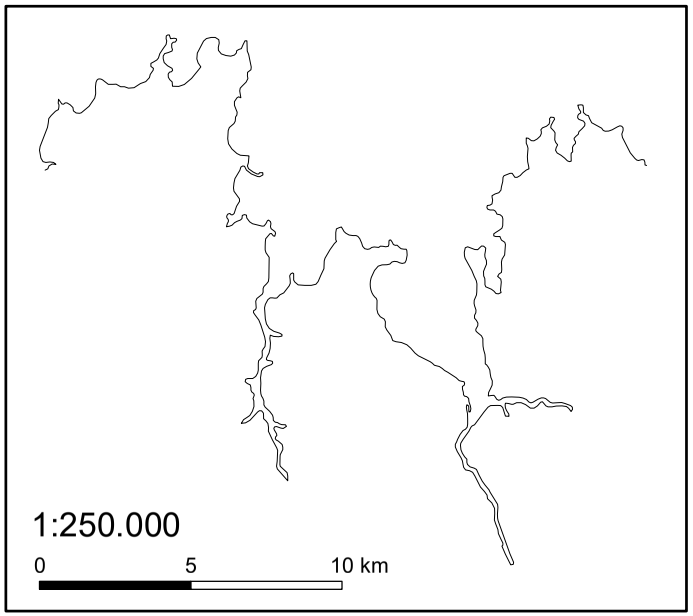
Manuell vereinfachte Linie

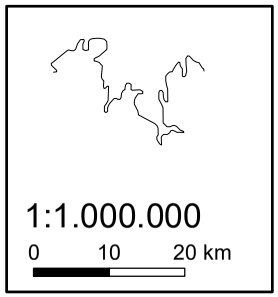
Quellen:
Bartholomew, John (Hg.) (1955, 1967)
IGN Institut Géographique National (1958, 1961)
IGN Institut National de l'Information Géographique et de Forestière (2017)
Ordnance Survey (1954)

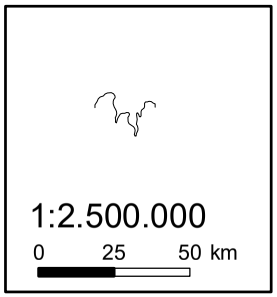


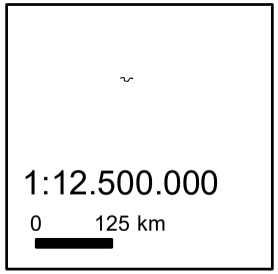
C3 Manuelle Linienvereinfachung nach Karten, ohne Originallinie



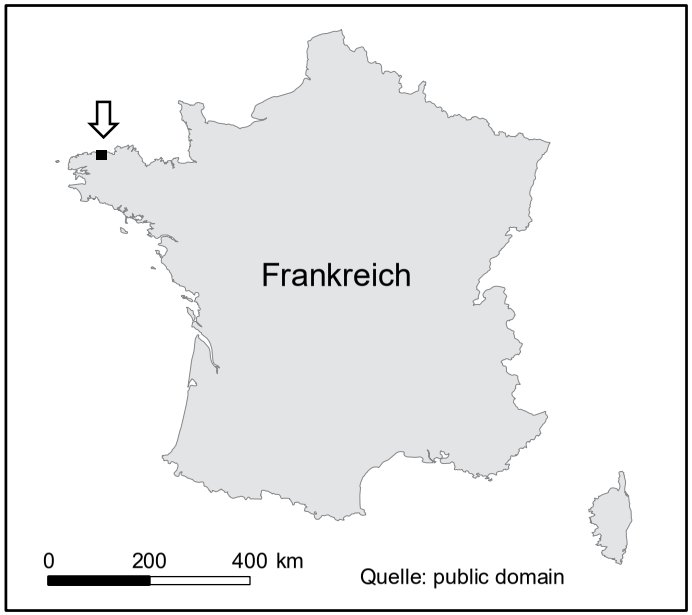




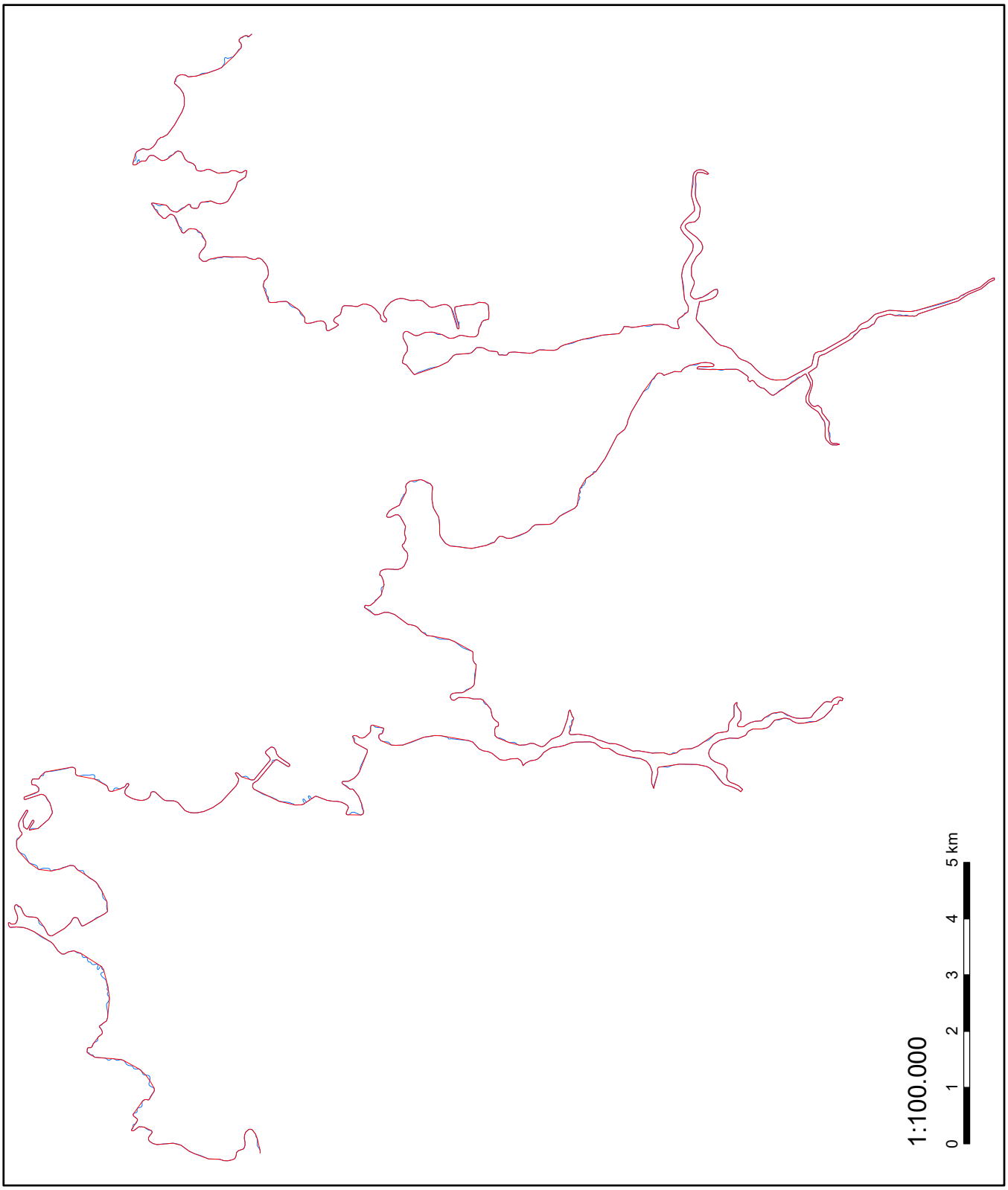


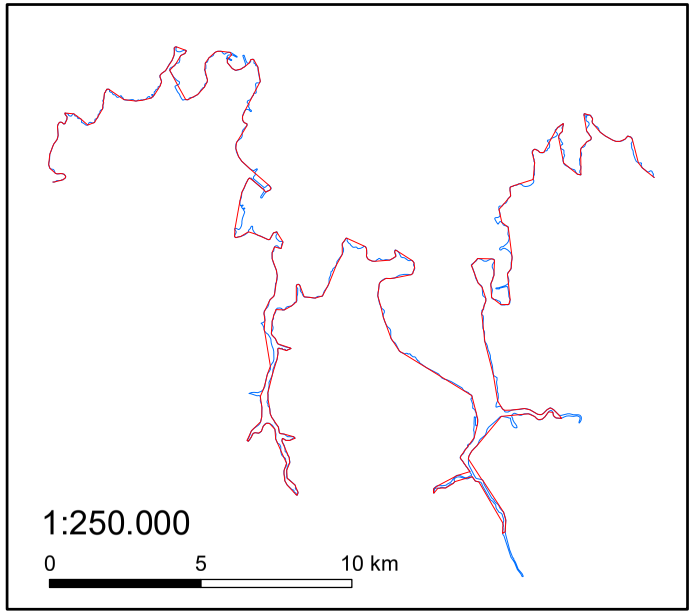


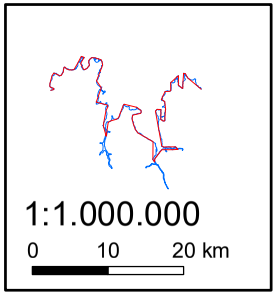
Quellen:
Bartholomew, John (Hg.) (1955, 1967)
IGN Institut Géographique National (1958, 1961)
Ordnance Survey (1954)

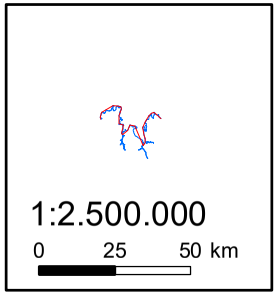


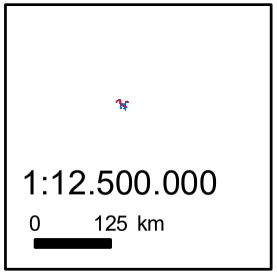
C4 Manuelle Linienvereinfachung, selbst erstellt







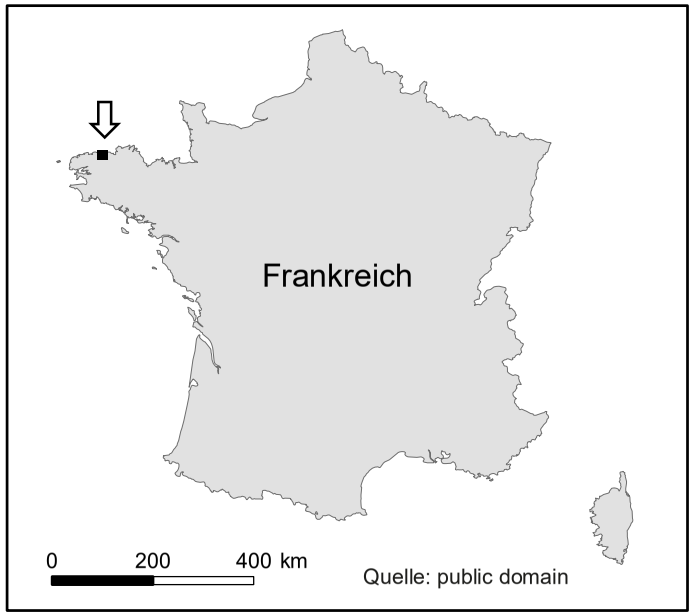




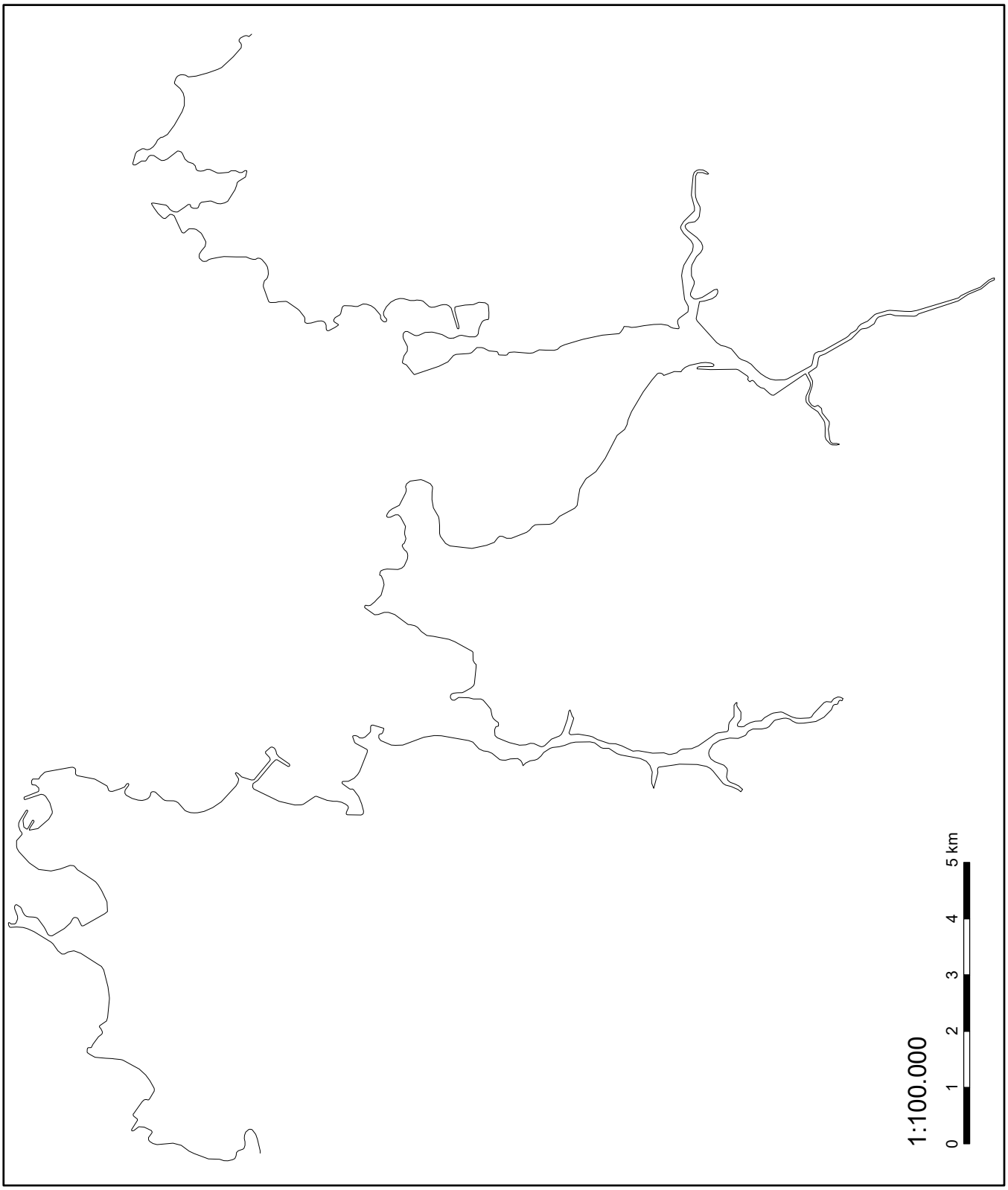
Originallinie ohne Anwendung von „Point Remove“

Vereinfachte Linie

Quelle:
IGN Institut National de l'Information Géographique et de Forestière (2017)

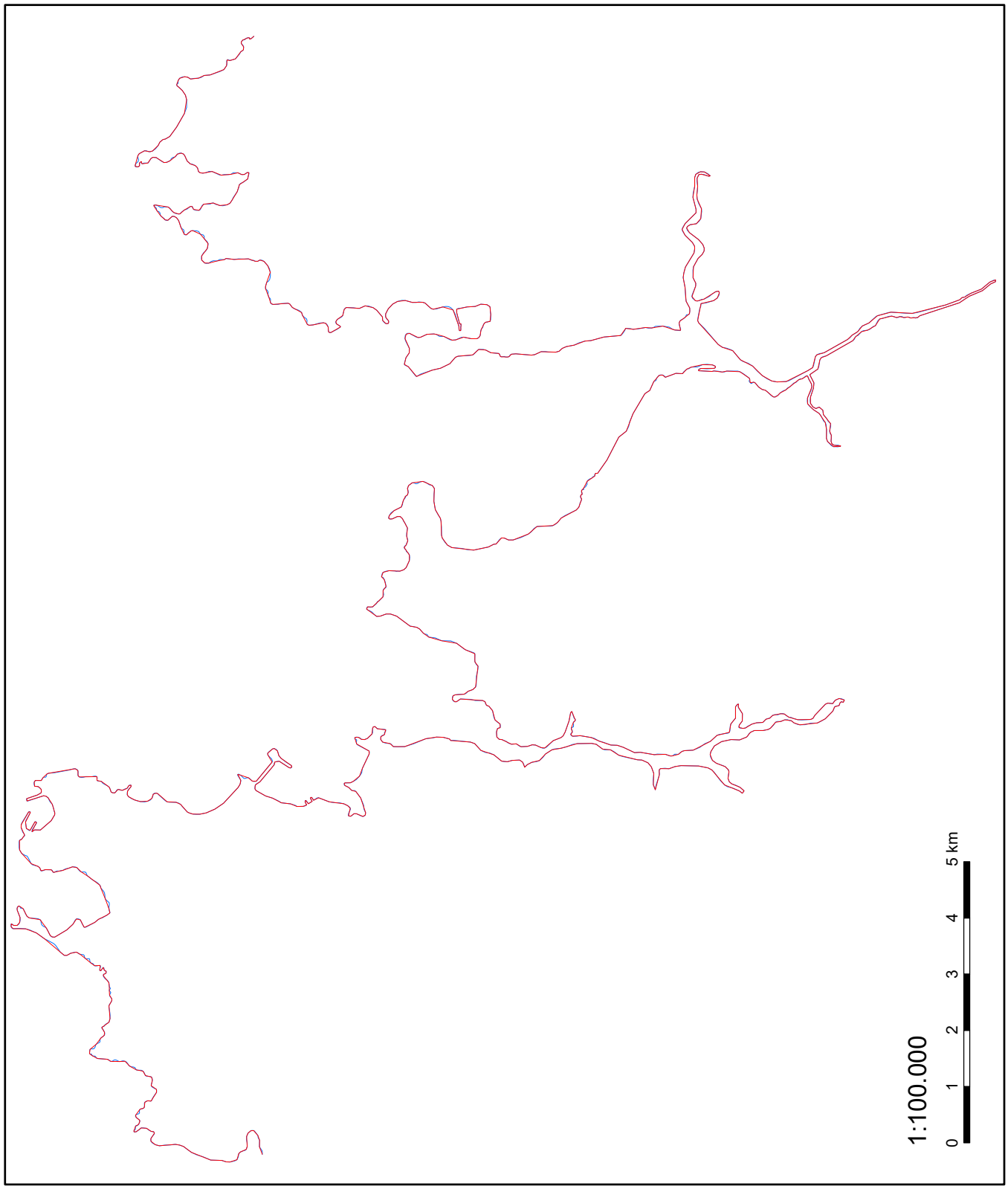


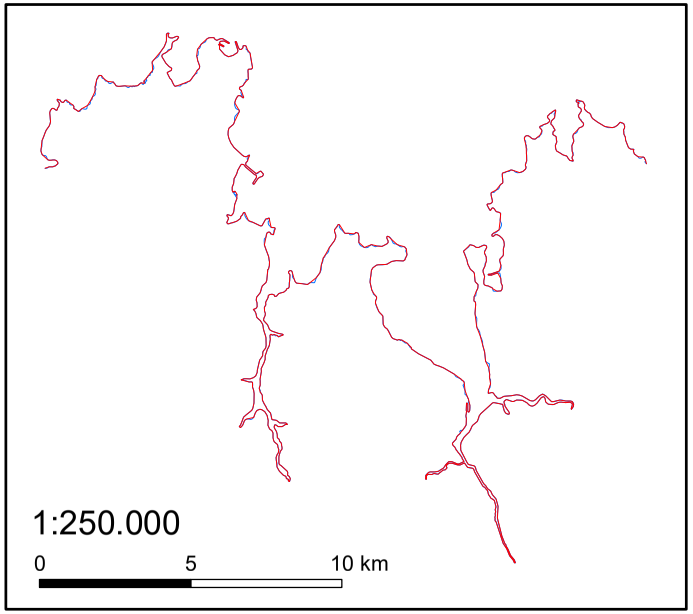
C5 Manuelle Linienvereinfachung, selbst erstellt, ohne Originallinie

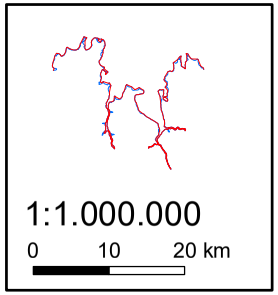


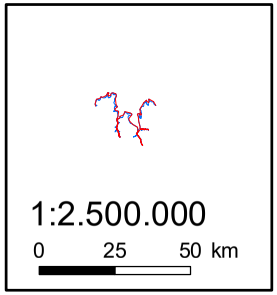
1:250.000
0
5
10 km
1:1.000.000
0
10
20 km
1:2.500.000
0
25
50 km
1:12.500.000
0
125 km
Frankreich
0
200
400 km
Quelle: public domain

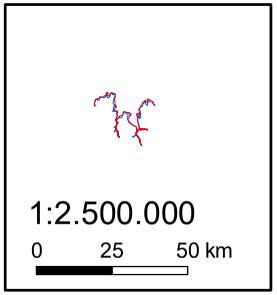
C6 Linienvereinfachung nach Ai et al. (2016), Methode *left*

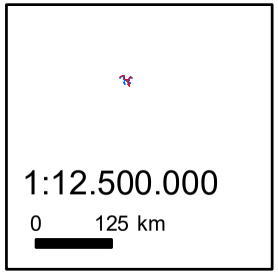










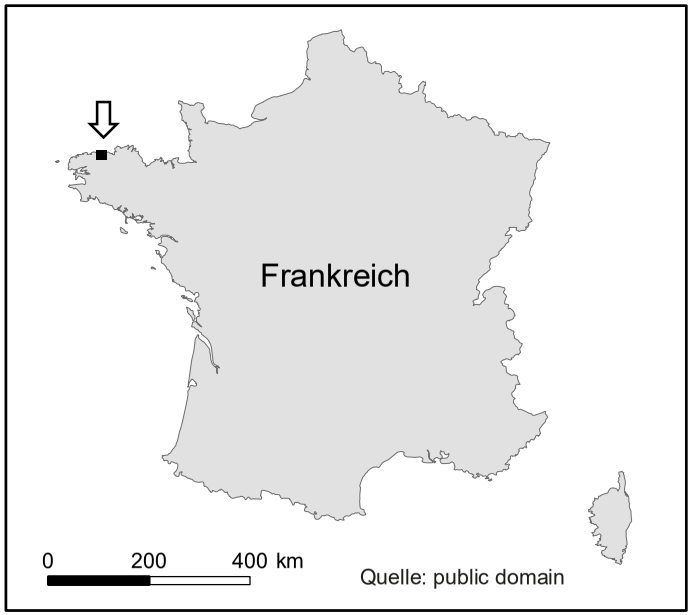


Umgehung von gamma

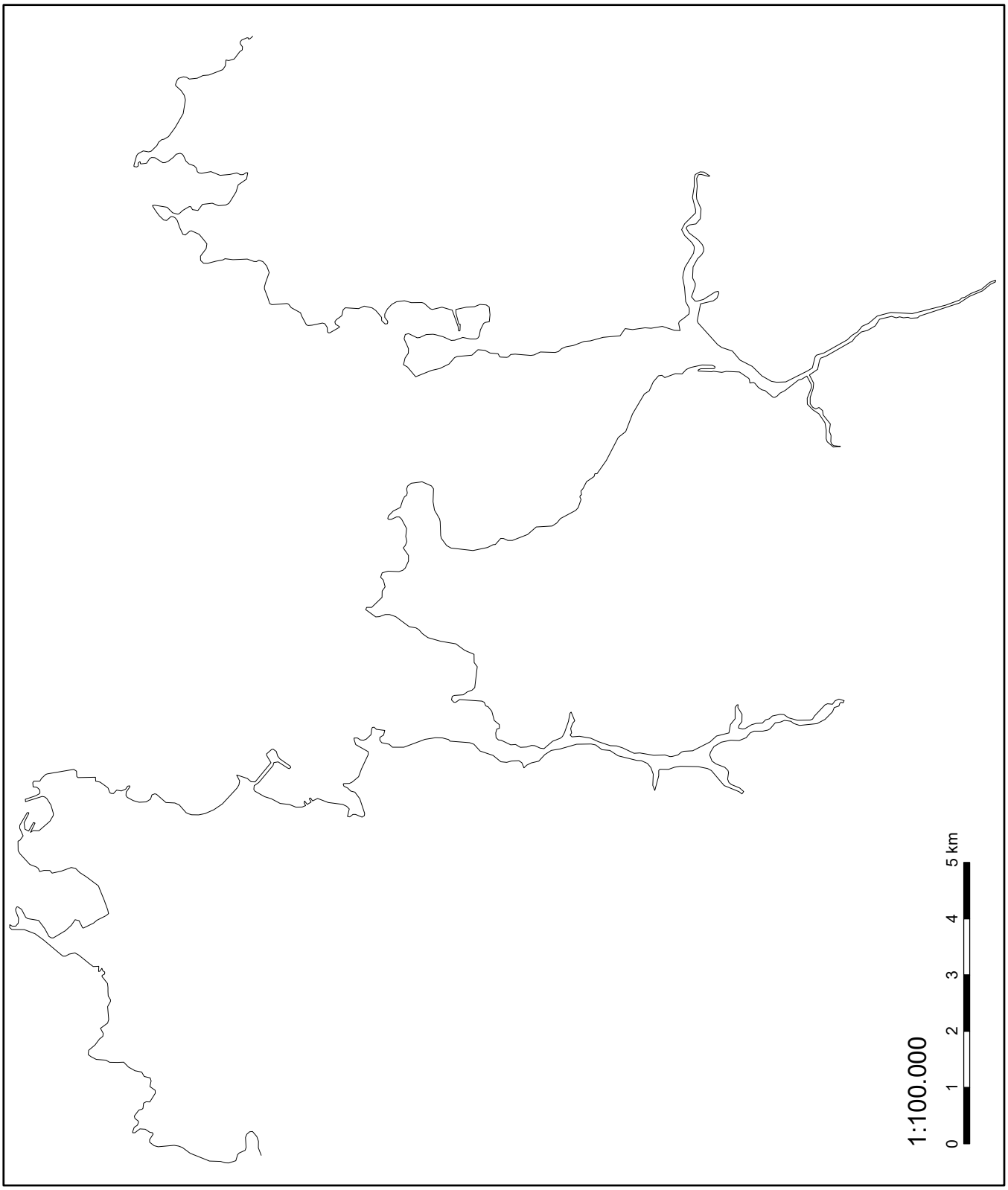
Originallinie ohne Anwendung von „Point Remove“

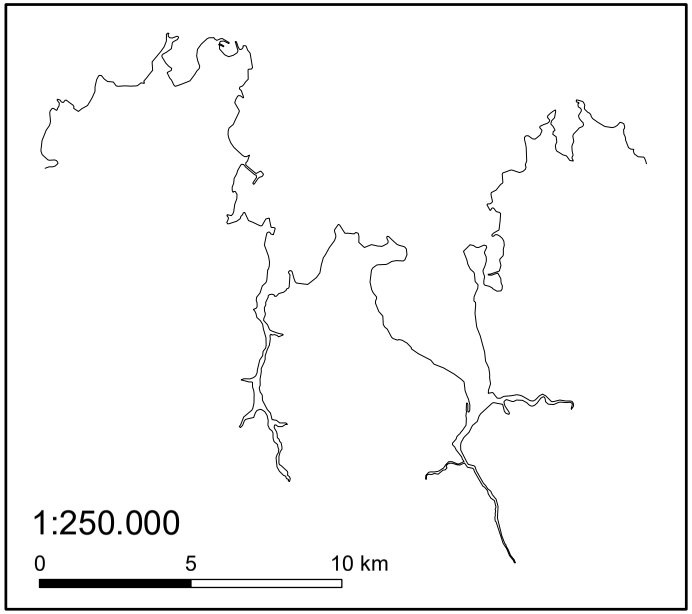
Vereinfachte Linie

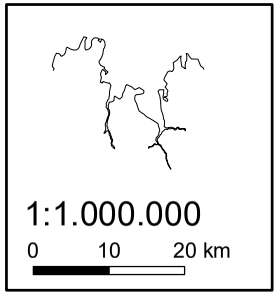
Quelle:
IGN Institut National de l'Information Géographique et de Forestière (2017)

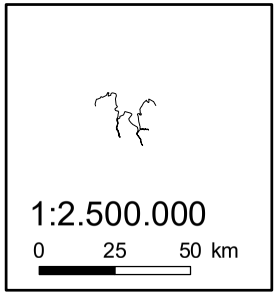


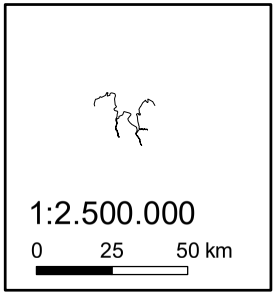
C7 Linienvereinfachung nach Ai et al. (2016), Methode *left*, ohne Originallinie



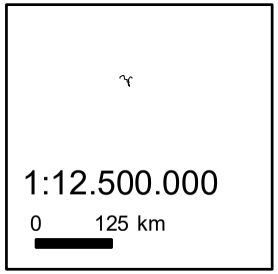


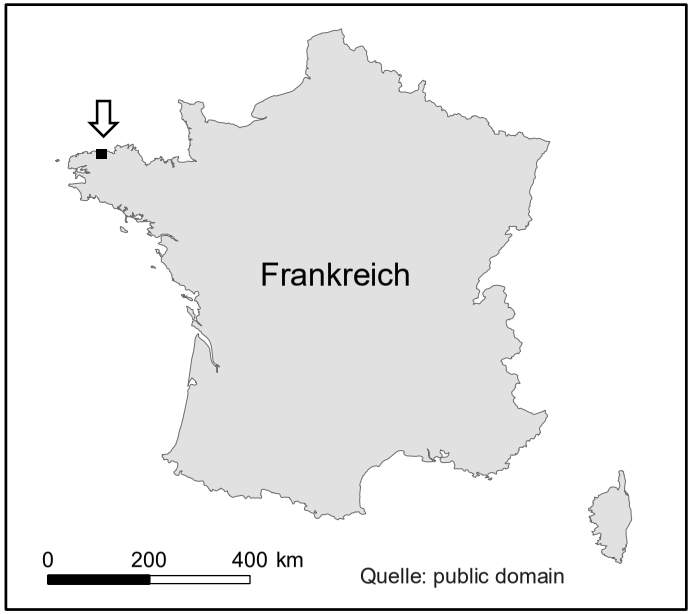




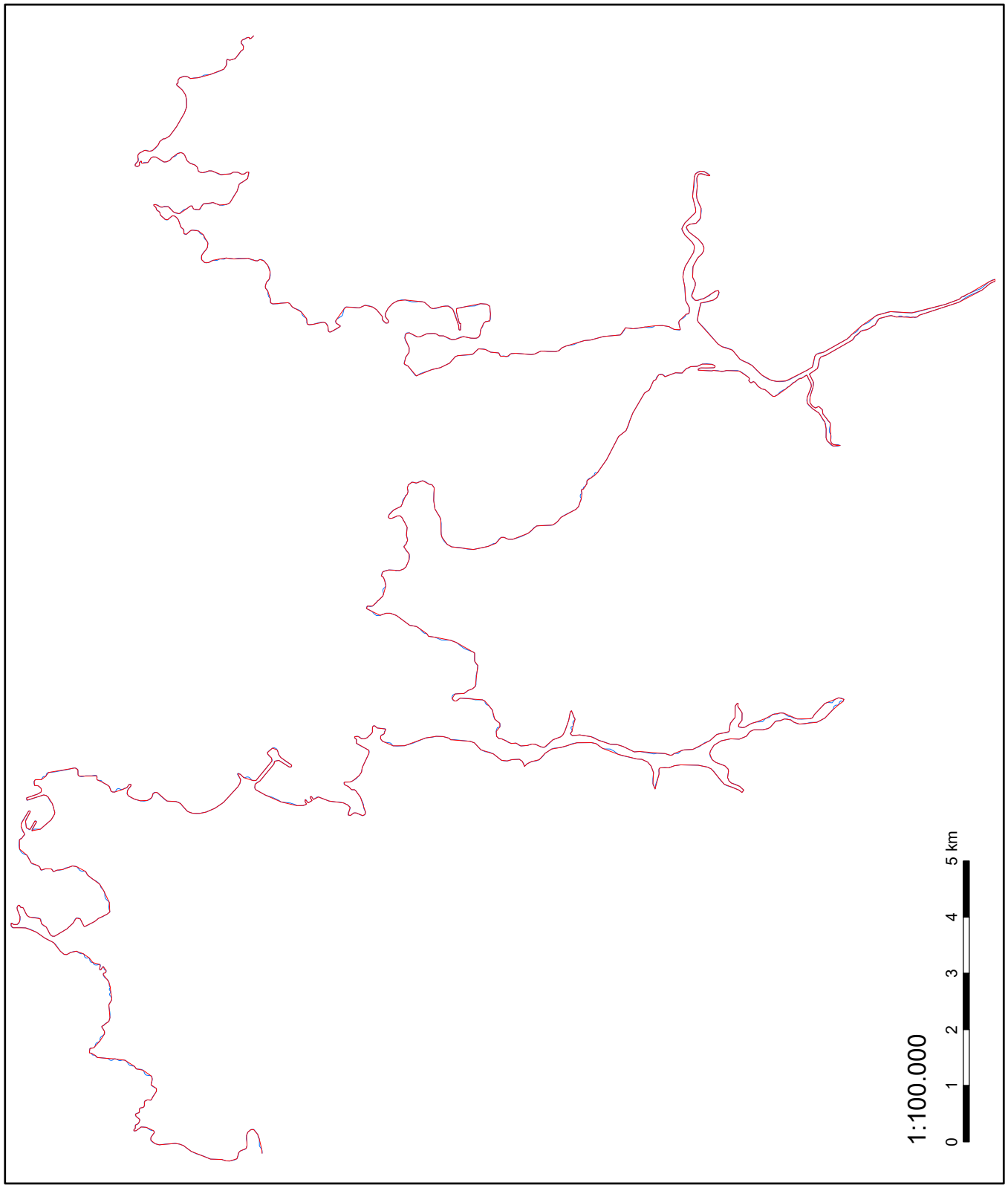


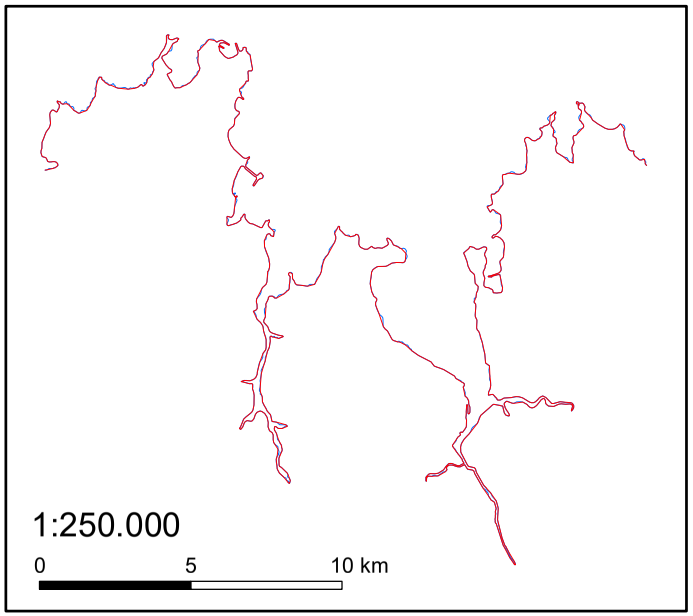
Umgehung von gamma

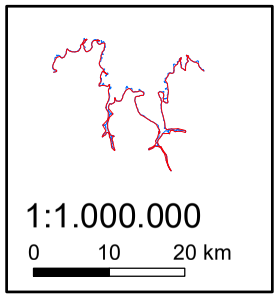


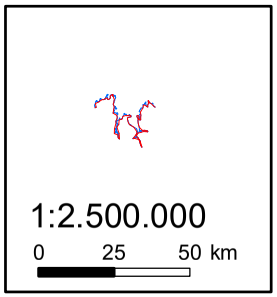


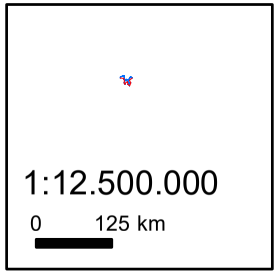
C8 Linienvereinfachung nach Ai et al. (2016), Methode *right*







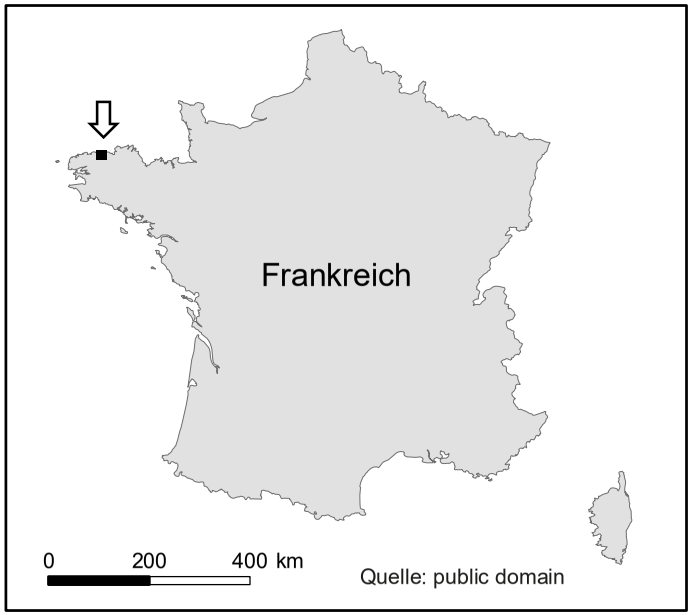




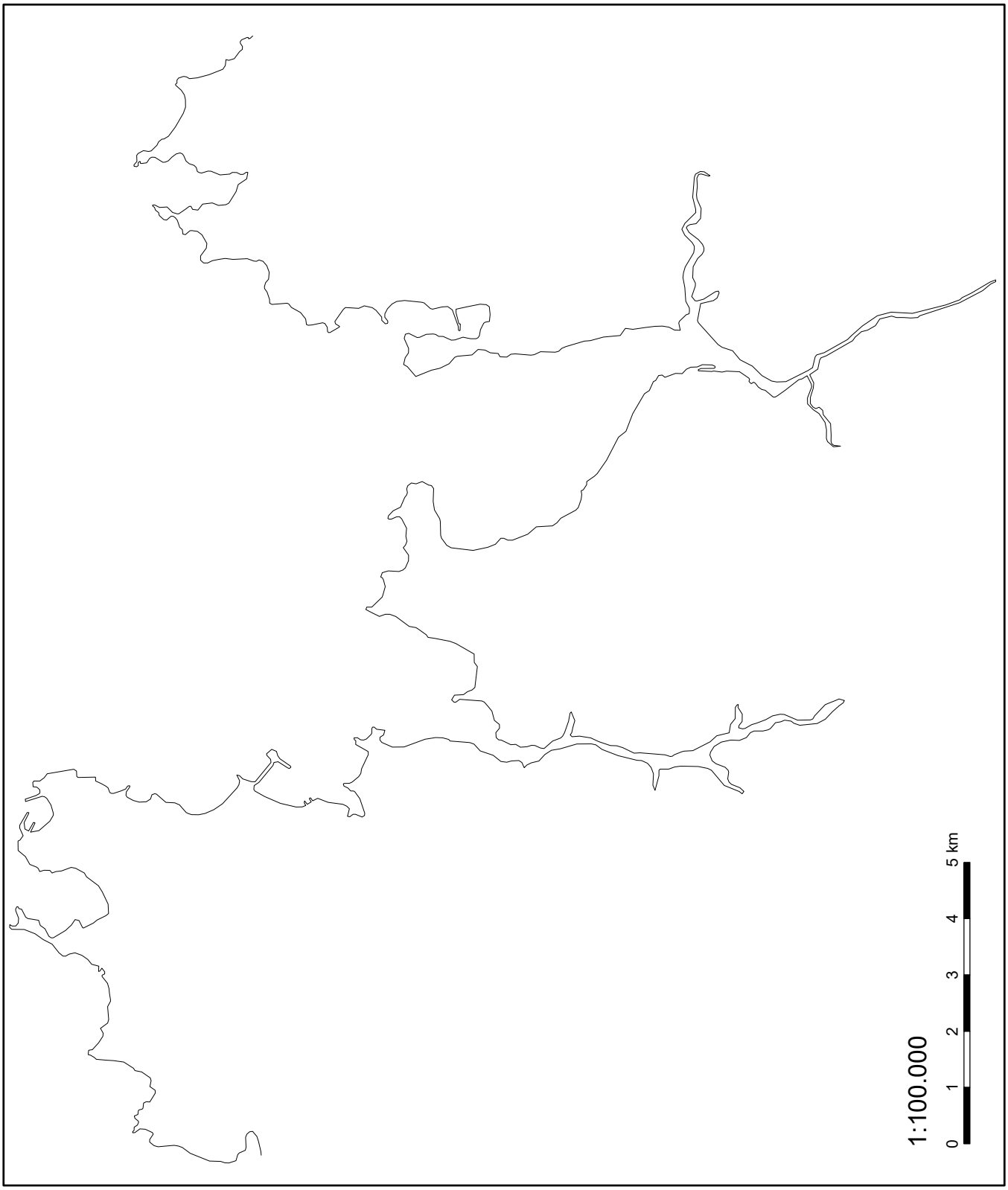
Originallinie ohne Anwendung von „Point Remove“

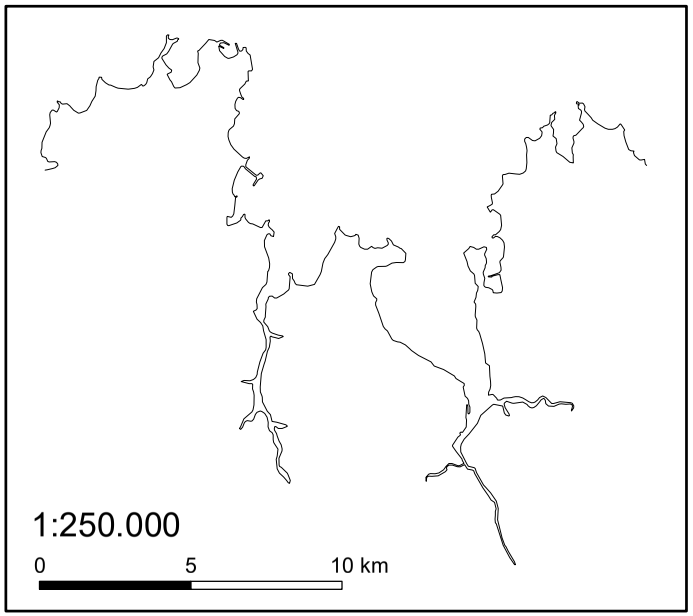
Vereinfachte Linie

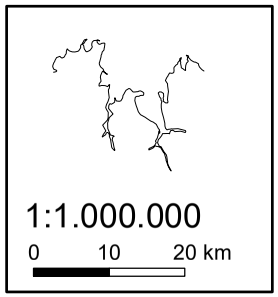
Quelle:
IGN Institut National de l'Information Géographique et de Forestière (2017)

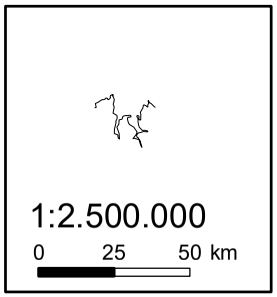


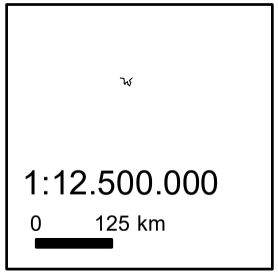
C9 Linienvereinfachung nach Ai et al. (2016), Methode *right*, ohne Originallinie

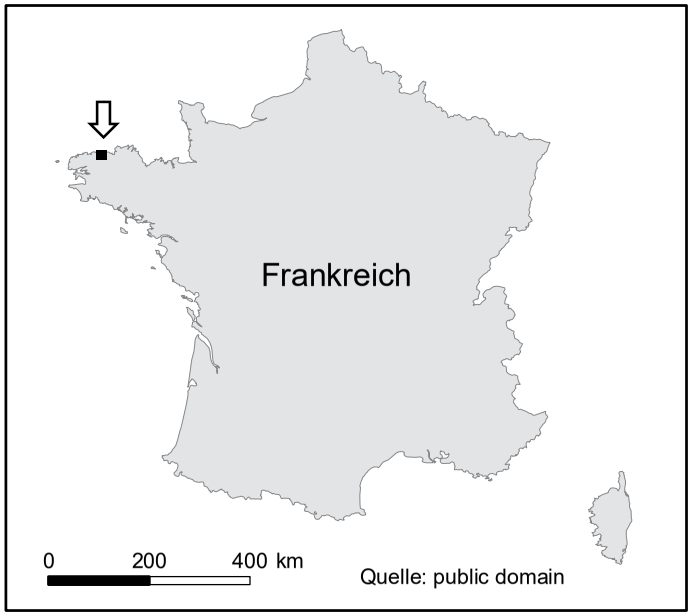



1:250.000
0 5 10 km

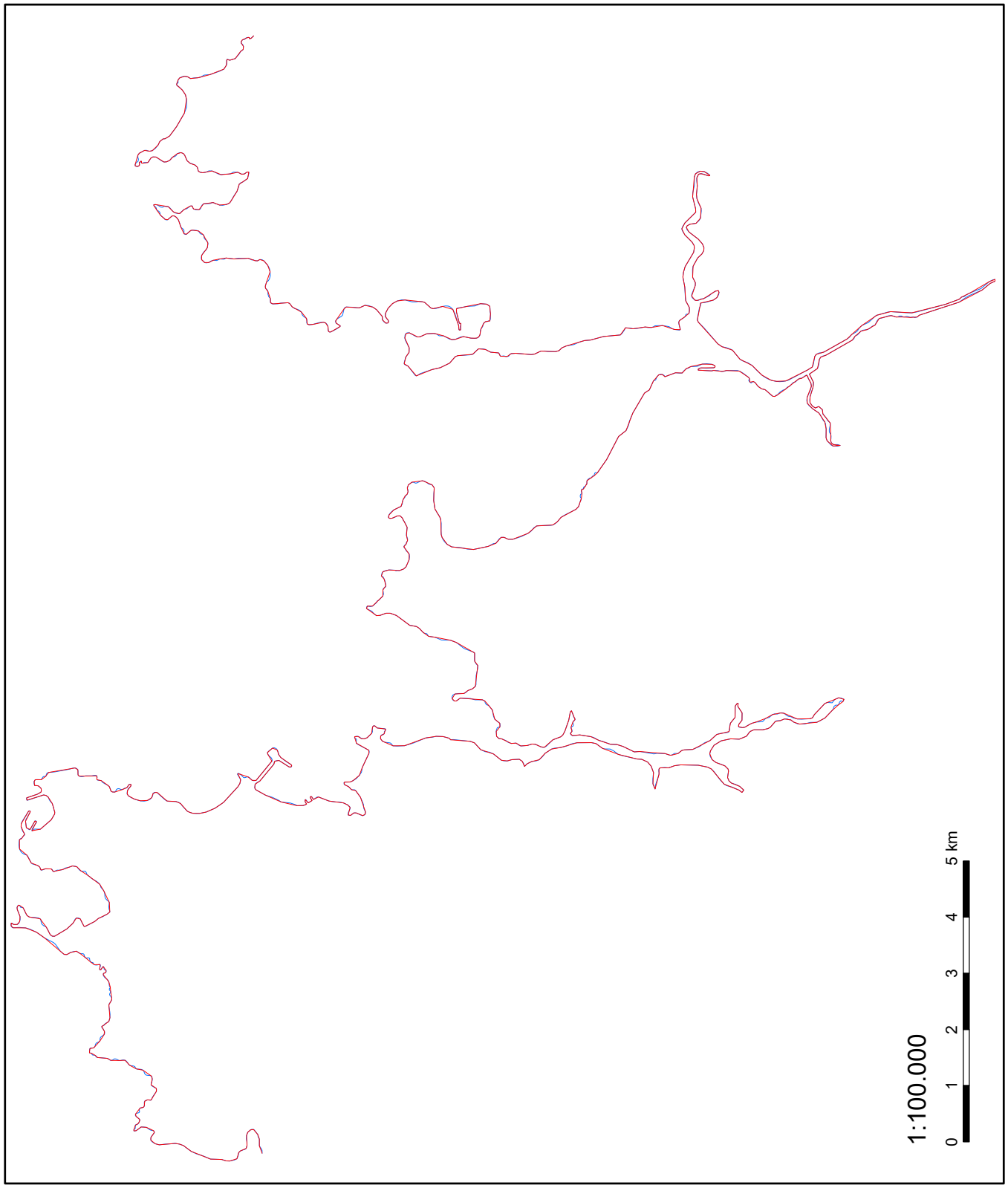

1:1.000.000
0 10 20 km

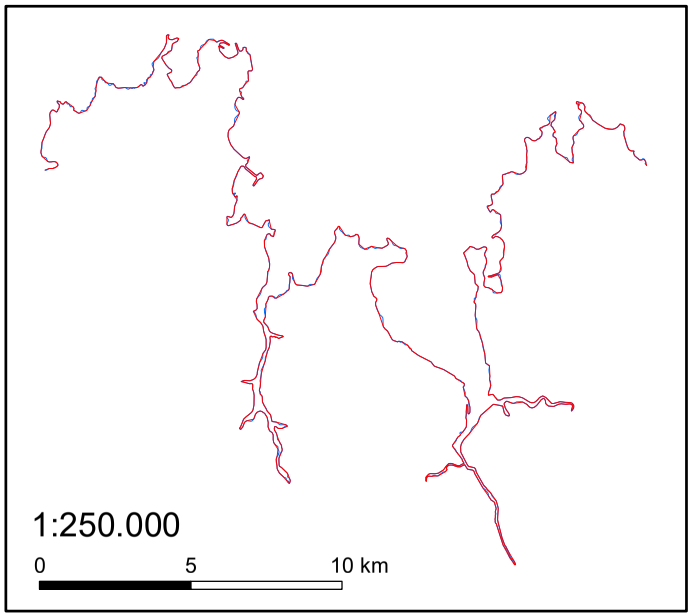

1:2.500.000
0 25 50 km

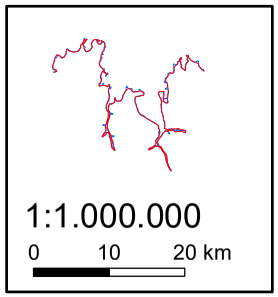

1:12.500.000
0 125 km

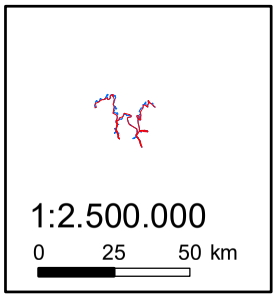

Frankreich
0 200 400 km
Quelle: public domain

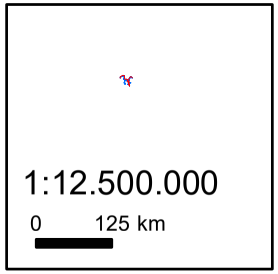
C10 Linienvereinfachung nach Ai et al. (2016), Methode *area preserving simple*







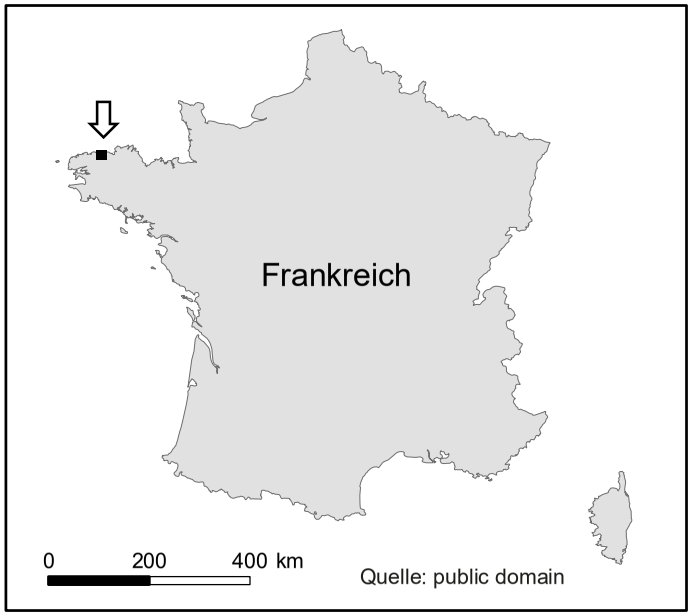




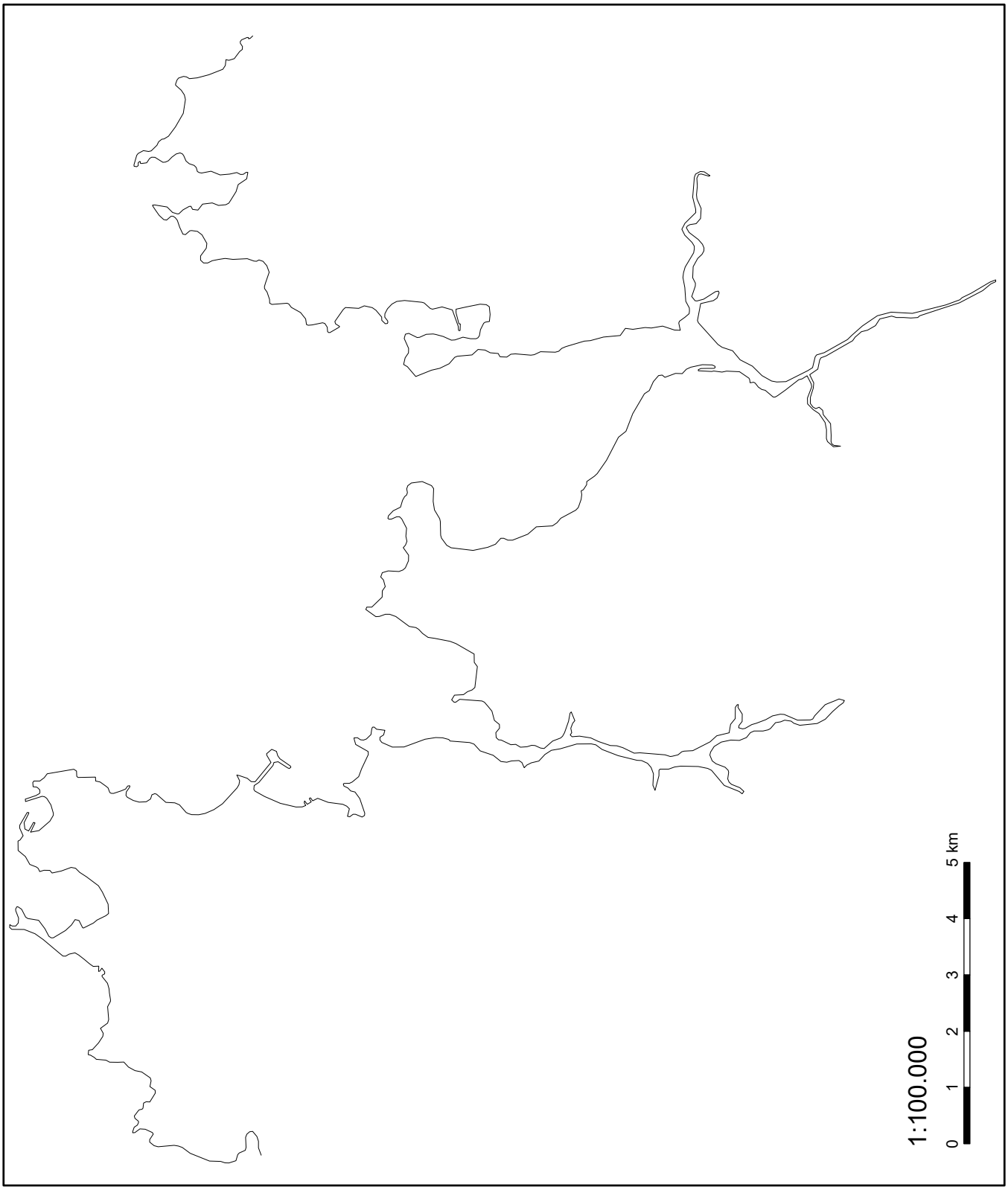
Originallinie ohne Anwendung von „Point Remove“

Vereinfachte Linie

Quelle:
IGN Institut National de l'Information Géographique et de Forestière (2017)



C11 Linienvereinfachung nach Ai et al. (2016), Methode *area preserving simple*, ohne Originallinie



1:250.000
0
5
10 km
1:1.000.000
0
10
20 km
1:2.500.000
0
25
50 km
1:12.500.000
0
125 km
Frankreich
0
200
400 km
Quelle: public domain