Effiziente Topologiebestimmung von Vektor GIS Daten

Florian TWAROCH, Tibor STEINER und Richard MALITS

Dieser Beitrag wurde nach Begutachtung durch das Programmkomitee als „reviewed paper" angenommen.

Zusammenfassung

Zwei neue Ansätze zur automatischen Bestimmung der Topologie von Vektor GIS Daten werden vorgestellt. Die betrachteten Verfahren, deren Effizienz und Robustheit in diesem Beitrag untersucht wird, basieren auf Ansätzen der algorithmischen Geometrie. Im Rahmen von Testimplementierungen zeigt sich, dass theoretisch optimale Algorithmen in der Praxis jedoch ein anderes Verhalten aufzeigen als erwartet. Die Ergebnisse empirischer Tests mit realen Datensätzen werden präsentiert.

1 Einleitung

Die Bestimmung der Topologie von Vektor GIS Daten ist eine herausfordernde Aufgabe, zumal sie numerisch aufwendig und rechenintensiv ist. Im Rahmen eines K Plus Projekts wurden in Kooperation der Firmen Advanced Computer Vision GmbH – ACV und rm-DATA Datenverarbeitungsg. m. b. H. zwei Verfahren untersucht, welche auf einer Zerlegung der Ebene beruhen. Im Unterschied zu herkömmlichen Verfahren zur Topologiebildung, welche zunächst die Eingangsdaten bereinigen und hernach Topologien bilden, führen die zwei vorgestellten Verfahren die Fehlererkennung und –bereinigung während des Topologieaufbaus durch.
Wie im Verlauf der folgenden Ausführungen gezeigt werden wird, erfüllen die beiden zur Diskussion stehenden Algorithmen die nachstehend angeführten Kriterien:

- Bestimmung der gesamten Knoten-Kanten-Polygon Topologie eines Katasterplanes, wahlweise mit und ohne Berücksichtigung von Einsetzpunkten. (Unter Einsetzpunkten versteht man jene Punkte eines Polygons, denen semantische Informationen wie beispielsweise Namen zugeordnet werden.)
- Das Verfahren soll eine hohe Effizienz (günstiges Laufzeitverhalten) und geringen Speicherbedarf aufweisen. Die Verarbeitung von Massendaten (einige Millionen Liniensegmente) soll in einer Standard PC Umgebung möglich sein.
- Aufbau der Knoten-, Kanten- und Polygontopologie unter Berücksichtigung von Inselpolygonen.
- Berücksichtigung einer Fehlertoleranz beim Bilden der Topologie: Leichte Ungenauigkeiten bei der Datenerfassung führen zu Kanten mit freien Enden (Over- und Undershootkonfigurationen). Das sind Kanten, die in mindestens einem Endpunkt mit keinen weiteren Kanten inzidieren. Sie sind als Datenfehler zu interpretieren und sollen durch das Vorsehen einer Fehlertoleranz erkannt und bereinigt werden.

2 Algorithmische Geometrie und GIS

Die algorithmische Geometrie geht auf M. Shamos zurück, der in seiner Dissertation (SHAMOS 1978) Methoden zur Lösung geometrischer Probleme mit dem Computer analysiert. In ihrem Standardwerk beschreiben Preparata und Shamos (PREPARATA & SHAMOS 1985) eine Vielzahl von Algorithmen, welche dem Finden konvexer Hüllen von Polygonen, der Triangulation gegebener Punktmengen, etc. dienen. Seit der Veröffentlichung haben sich eine Vielzahl von Autoren, auch mit Aspekten der algorithmischen Geometrie im Konnex mit Problemen aus dem GIS Bereich beschäftigt (DE FLORIANI, PUPPO & MAGILLO 1999).

Um die Topologie von Liniensegmenten zu bestimmen, führen (KRIVOGRAD & ŽALIK 2000) in einem ersten Schritt eine Vorverarbeitung durch, in deren Rahmen zunächst die Polylinien in Liniensegmente aufgespaltet und – unter Berücksichtigung eventueller Spezialfälle – sämtliche Schnittpunkte ermittelt werden. Sofern ein konsistenter Datensatz vorliegt, wird in einem zweiten Schritt das äußere Hüllpolygon, welches nicht notwendigerweise konvex sein muss, bestimmt. Ausgehend von dieser Datenstruktur, werden in einem dritten Schritt Umfahrungspolygone, sowie Inselpolygone ermittelt; es wird jedoch nicht beschrieben wie man zu Einsetzpunkten gelangt. (KRIVOGRAD & ŽALIK 2000) führen in ihrer Arbeit auch Performanceuntersuchungen durch, welche jedoch nur auf kleinen und auch nicht realen Geodatenmengen beruhen.

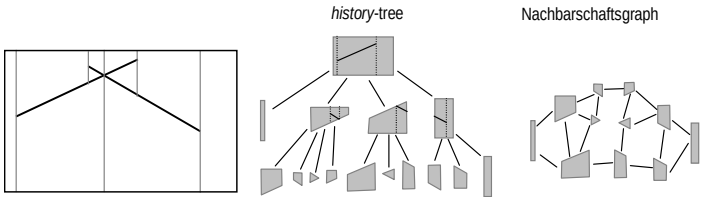
Um die automatische Generierung der Topologie bei gescannten Karten zu ermitteln, untersuchte (GOLD 1997) drei Verfahren, wobei das erste statisch (unveränderlicher Datensatz), die beiden letzteren hingegen dynamisch (veränderlicher Datensatz) arbeiten. Punkt Voronoi Diagramme, zwangsbasierte Delaunay Triangulierung und Voronoi Diagramm von Liniensegmenten und Punkten kommen dabei zum Einsatz. Details zu den Algorithmen oder deren Implementierungen findet man bei Gold nicht, er verweist jedoch auf die entsprechende Literatur der algorithmischen Geometrie.

3 Anordnung von Liniensegmenten

Die Bestimmung der Topologie von Geodaten ist in der algorithmischen Geometrie als Anordnung von Linien (Segmenten) bekannt. Gegeben sei eine endliche Menge von Linien L. Der durch sie induzierte Zellenkomplex wird Anordnung A(L) genannt. 0-Flächen (Knoten) sind die Schnittpunkte der Linien, 1-Flächen (Kanten) sind die maximalen Teile der Linien von L, die keinen Knoten enthalten, und schließlich 2-Flächen (Zellen genannt – in der Folge Umfahrungspolygone) sind die zusammenhängenden Komponenten des R^2. Der in 3.1 vorgestellte Algorithmus führt eine Zerlegung der Ebene in Trapeze durch, wohingegen der in 3.2 betrachtete Algorithmus die Anordnung der Liniensegmente in der Ebene durch eine Zerlegung in Dreiecke realisiert.

3.1 Trapezoidale Zerlegung

In (CHAZELLE, EDELSBRUNNER & GUIBAS 1991) und (DEBERG, DOBRINT & SCHWARZKOPF 1995) wird ein *randomized incremental algorithm* zur Berechnung einer Anordnung von Liniensegmenten vorgestellt. Die Grundlage dieser Methode ist eine Zerlegung der Ebene in Trapeze. Diese werden in einer baumartigen Datenstruktur (history-tree (Abb. 1b) verwaltet. Wird ein neues Liniensegment in die Datenstruktur eingefügt (Abb. 1a), werden zunächst alle möglichen Schnittpunkte mit bestehenden Trapezen ermittelt und gegebenenfalls weitere Trapeze erzeugt. Eine weitere Datenstruktur repräsentiert die Nachbarschaft der momentan aktuellen Trapeze (Abb. 1c).



a b c

Abb. 1: Der Nachbarschaftsgraph (a) verbindet ausschließlich die Blätter des *history*-tree (b) und wird beim Einfügen eines Liniensegmentes (a) in den history-tree, aktualisiert.

Die Methode eignet sich auch sehr gut, zu einem gegebenen Einsetzpunkt zugehörige Polygone aufzusuchen. Die Suche nach einem Polygon erfolgt in zwei Schritten:

1. Ein Trapez wird als Starttrapez in der Datenstruktur gekennzeichnet. Dazu wird der *history-tree* traversiert, solange bis ein Trapez gefunden wurde, das einem bestimmtem Kriterium entspricht (z. B. Enthaltensein eines Einsetzpunktes - Abb. 2 dunkelgraues Trapez).

2. Ausgehend vom Starttrapez werden rekursiv alle Nachbarn, welche über eine vertikale Seite benachbart sind, ermittelt und zu einer Region zusammengefasst (hellgraue Trapeze in Abb. 2). Hierzu werden die im Nachbarschaftsgraph gespeicherten Adjazenzbeziehungen verwendet.

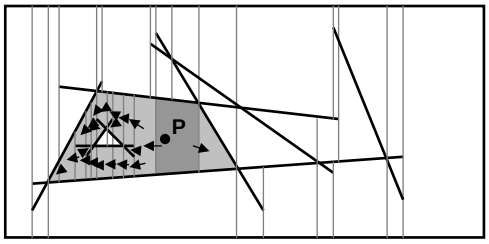


Abb. 2: Suche des Umfahrungspolygons zu einem Einsetzpunkt P - ausgehend von einem Starttrapez (dunkelgrau) werden benachbarte Trapeze ermittelt.

Die kombinatorische Komplexität einer Aufgabe gibt Auskunft über die Effizienz eines möglichen Algorithmus. Bei der Anordnung von Liniensegmenten ist die obere Grenze der Komplexität $O(n^2)$ für zufällig angeordnete Liniensegmente. Für ein einzelnes Polygon in der *Annordnung* von Linien ist die obere Grenze der Komplexität gleich der Anzahl der Kanten des Polygons. Für *m* Polygone und *n* Liniensegmente gilt laut (AGARWAL 1990):

$$O(m^{2/3}n^{2/3} + m + n)$$

In ersten empirischen Tests mit einem Demonstrator konnten für kleine Datenmengen ein Verhalten, das in etwa *O(n log(n))* entspricht, festgestellt werden.

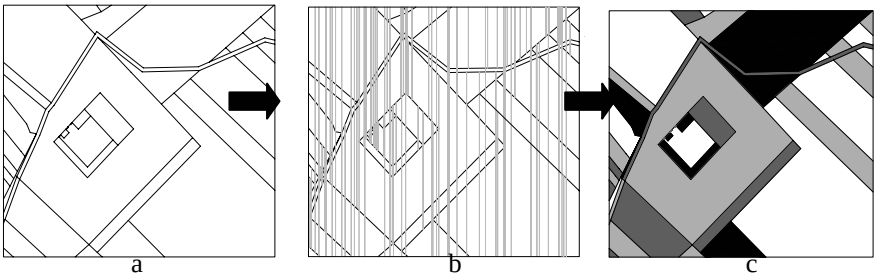


Abb. 3: Ausgehend von Spaghettidaten (a) werden in einem Dreischrittverfahren mittels Zerlegung der Ebene in Trapeze (b) Polygontopologien (c – grau schattiert) gebildet.

In einem ersten Schritt werden Fehler in den Ausgangsdaten bereinigt (z. B. doppelte Segmente, Segmente der Länge Null etc.). In einem zweiten Schritt erfolgt die Zerlegung der Ebene in Trapeze (Abb. 3b). Abb. 3b demonstriert auch das stark globale Verhalten des Algorithmus, der lang gezogene Feldweg wird vielfach in Trapeze zerteilt. In einem abschließenden Schritt erfolgt die Bildung von Ergebnispolygonen (Abb. 3c), sowie eine Fehlerdetektion.

3.2 Zwangsbasierte Triangulierung

Ein alternativer Algorithmus bedient sich einer zwangsbasierten Triangulierung einer Punktmenge in der Ebene. Für die vorgestellten Berechnungen ist keine spezielle Triangulierung erforderlich, wie z. B. die bekannte Delaunay Triangulierung (DELAUNAY 1934). Es sei hier auf die umfangreiche Literatur zur Bildung von Triangulierungen verwiesen (BERN & EPPSTEIN 1992, DWYER 1987, SHEWCHUK 1996).

In einem Dreischrittverfahren können topologisch unreine Massendaten verarbeitet werden (Abb. 4). Die Fehlerdetektion und -bereinigung erfolgt während des Topologieaufbaus. Der in einem Demonstrator umgesetzte Algorithmus arbeitet in 3 Schritten:

Schritt 1 (Vorverarbeitung): Es erfolgt eine Sortierung der Punkte nach der *x*-Koordinate, um doppelt vorkommende Punkte zu ermitteln und dafür Sorge zu tragen, dass sie nur einmal in der Datenstruktur abgelegt werden. Das bringt eine erhebliche Reduzie-

rung der Datenmenge mit sich. Liniensegmente werden in Referenzen zu Punktdaten umgewandelt. Identische Segmente werden hierbei entfernt.

Schritt 2 (Triangulierung): Zunächst werden die Endpunkte der vorgegebenen Liniensegmente nach einem beliebigen freien Verfahren trianguliert (Abb. 4a). Dann werden die Segmente in die Triangulierung eingefügt (Abb. 4b). Ist ein Segment bereits durch eine Kante der Triangulierung repräsentiert wird eine Referenz gesetzt, alle anderen Segmente werden in die Triangulierung gezwängt. Dazu werden alle Kanten gelöscht die von dem einzufügenden Segment (Zwangskante) geschnitten werden (Abb. 4c). Die beiderseits der Zwangskante verbleibenden Polygone werden erneut trianguliert (Abb. 4d). Das Ergebnis ist eine zwangsbasierte Triangulierung aller vorgegebenen Liniensegmente.

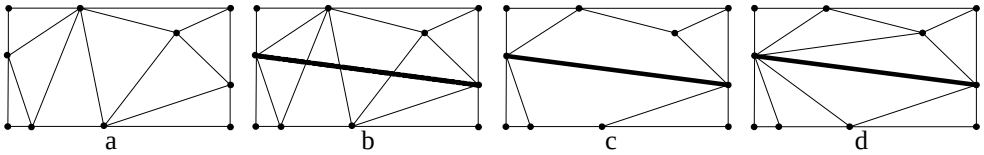


Abb. 4: Nach einer freien Triangulierung (a) der Endpunkte der Liniensegmente können Liniensegmente (b) in die Triangulierung als Zwangskanten eingefügt werden. Ist die Zwangskante nicht Teil der Triangulierung werden geschnittene Kanten gelöscht (c) und die beiderseits verbleibenden Polygone erneut trianguliert (d).

Schritt 3 (Topologiebildung): Die Topologiebildung läuft in zwei Schritten ab:

- Zuerst werden Dreiecke, die nicht über vorgegebene Kanten (Abb. 5a) benachbart sind, zu Gebieten zusammengefasst. Dies kann durch rekursives Durchsuchen der Dreiecksstruktur (Abb. 5b) erreicht werden. Elemente, die zweimal in die Struktur eingefügt werden, werden aus der Struktur gelöscht.
- Es verbleiben nur mehr Kanten, die Polygonen angehören. Die Kanten der Gebiete werden sortiert und als geschlossene Polygone ausgegeben (Abb. 5c). Treten bei der Sortierung der Kanten mehrere Umfahrungspolygone auf, so werden Inselpolygone identifiziert. Das Polygon mit der größten Fläche bildet das äußere Umfahrungspolygon.

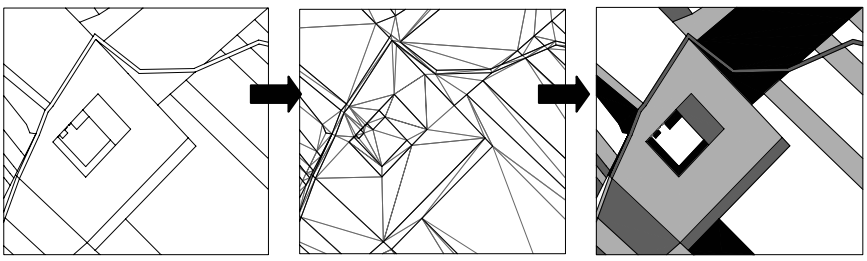


Abb. 5: Aus den Eingangsdaten (a) entsteht in einem Dreischrittverfahren eine zwangsbasierte Triangulierung (b); in einem abschließenden Schritt wird die Polygontopologie (c) gebildet.

Mit einem Demonstrator kann auch für große Datenmengen (1.000.000 Liniensegmente) quasilineares Leistungsverhalten aufgezeigt werden. Dies ist auf die Lokalität des Verfahrens zurückzuführen.

4 Adaptionen für einen GIS Algorithmus

Bei der theoretischen Untersuchung geometrischer Algorithmen werden häufig praktische Aspekte außer Acht gelassen, die für die Funktionsfähigkeit eines GIS Algorithmus wesentlich sind. Ein GIS Algorithmus für Vektordaten benötigt z. B. Fließkommaarithmetik, Berücksichtigung von Fehlertoleranzen etc. Auch die Verarbeitung großer Datenmengen muss in Betracht gezogen werden. Auf zwei wesentliche Punkte unserer Implementierung soll hier näher eingegangen werden.

4.2 Bestimmung eindeutiger Einsetzpunkte

In vielen digitalen Katasterplänen existieren Einsetzpunkte, über die semantische Informationen Polygonen zugeordnet werden. Dabei kann es vorkommen, dass kein, ein oder mehrere Einsetzpunkt(e) einem Polygon zugeordnet werden können. Ziel ist es, jedem Polygon genau einen Einsetzpunkt zuzuordnen. Ein begrenzendes Rechteck zum Plazieren von Texten bzw. Symbolen soll berücksichtigt werden. Die folgende Abb. 6 beschreibt die Funktionsweise des von uns entwickelten Algorithmus.

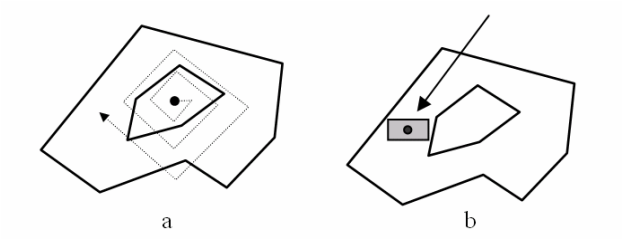


Abb.6: Ausgehend vom Schwerpunkt des Polygons wird das Polygon spiralförmig abgetastet (a), bis eine Position für einen Einfügepunkt (b) gefunden wurde.

In den, von uns durchgeführten Tests erwiesen sich die Schwerpunkte der Polygone in 90 % der Fälle als gültige Einsetzpunkte, während in den restlichen 10 % geeignete Einsetzpunkte durch eine spiralförmige Abtastung der jeweiligen Polygone ermittelt wurden.

4.3 Behandlung von Over- und Undershoots

Eine wichtige Funktionalität ist die Behandlung von Over- und Undershoots. Das sind vorgegebene Liniensegmente, die in mindestens einem Endpunkt mit keinem weiteren Liniensegmente inzidieren. Sind solche Segmente kleiner als ein definiertes Toleranzkriterium handelt es sich um einen Overshoot, andernfalls liegt ein Undershoot vor. Zunächst soll die Triangulierung eines Over-/Undershoot freien Polygon betrachtet werden (Abb. 7a). Hinzufügen eines weiteren Liniensegments (Abb. 7b) verursacht eine stärkere Seg-

mentierung des Polygons. Im Unterschied zu Abb. 7a enthalten in Abb. 7b, Dreiecke im Inneren des Polygons ebenfalls Referenzen auf vorgegebene Kanten. So können fehlerhafte Liniensegmente identifiziert werden.

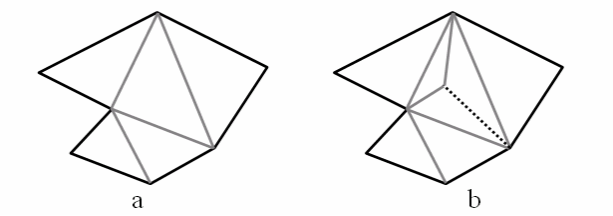


Abb.7: Im Gegensatz zur Triangulierung eines fehlerfreien Polygons (a), enthält ein topologisch unreines Polygon (b) im Inneren Referenzen auf vorgegebene Liniensegmente.

Bei der Bildung der Umfahrungspolygone (vgl. 3.2) werden solche Polygone speziell markiert und in einem Nachbearbeitungsschritt bereinigt. Overshootkonfigurationen beeinflussen die Funktionsweise des Algorithmus nicht.

5 Empirische Tests

Im Verlauf von empirischen Tests (Pentium IV - 2,2 GHz, 512 MB, Windows XP Professional) wurde das Leistungsverhalten der prototypischen Implementierungen bezüglich Laufzeit und Speicherbedarf wiederholt untersucht, wobei sich zeigte, dass durchgeführte Erweiterungen und Verbesserungen des Algorithmus einen nachhaltigen Einfluss auf das Verhalten ausübten. Der Algorithmus, welcher eine Unterteilung des Untersuchungsgebiets in Trapeze vornimmt, wurde objektorientiert in C++ umgesetzt, während der Algorithmus, der eine zwangsbasierte Triangulierung anwendet, funktionsorientiert in C umgesetzt wurde.

Wegen des globalen Verhaltens des Trapez-Algorithmus bezüglich Laufzeit und Speicherbedarfs konnte eine Verarbeitung von Massendaten nicht ohne weiteres in Angriff genommen werden. Eine Aufteilung der Daten in kleinere Recheneinheiten musste umgesetzt werden, dabei wurden räumliche Indizierungsmethoden, wie Quadtree und R-Tree (SAMET 1990) angewendet. In weiteren Leistungstests stellte sich sehr bald heraus, dass der Algorithmus mit der Dreieckszerlegung ein wesentlich besseres Leistungsverhalten aufweist, als der Algorithmus mit der Trapezzerlegung. Aus diesem Grund werden im Folgenden nur die Ergebnisse des leistungsstarken Algorithmus „zwangsbasierte Triangulierung“ präsentiert.

Die verwendeten fünf Datensätze, welche von der Firma rmDATA zur Verfügung gestellt worden sind, dienten ausgiebigen Leistungtests. Es sind dies Testdatensätze aus dem Burgenland mit dem Ortsgebiet von Leitha Prodersdorf (32.535 Liniensegmente) und Stadt Schlaining (91.055 Liniensegmente), sowie ein Testdatensatz aus Deutschland vom Gebiet Ingolstadt (982.891 Liniensegmente). Die weiteren Datensätze Ingolstadt 500 k (497.396

Liniensegmente) und Ingolstadt 250 k (249.243 Liniensegmente) sind Ausschnitte des zuvor genannten deutschen Gesamtdatensatzes.

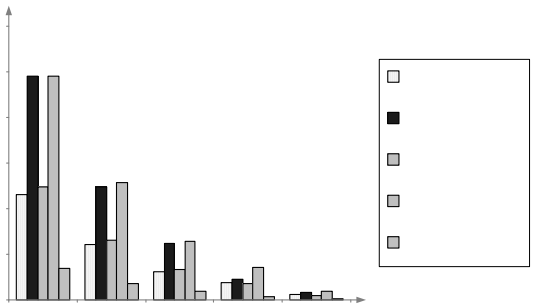


Abb.8: Die Graphik stellt für die zur Verfügung stehenden Datensätze die Anzahl der vom Algorithmus gebildeten Element gegenüber. Einen dominierenden Anteil haben vorgegebene Liniesegmente und ausgegebene Dreiecke; die Anzahl der gebildeten Polygone ist verhältnismäßig klein.

Anhand von Abb. 8 lässt das quasilineare Verhalten der Methode der „zwangsbasierten Triangulierung“ erkennen. Die von rechts nach links dargestellte zunehmende Anzahl vorgegebener Elemente (Punkte, Liniensegmente) bewirkt nur eine lineare Steigerung der vom Algorithmus ausgegebenen Elemente (Punkte, Dreiecke, Polygone). Wie bereits erwähnt, lassen sich mit der zwangsbasierten Triangulierung wesentlich größere Datenmengen ohne eine Zerlegung der Eingabedaten in kleinere Recheneinheiten verarbeiten, als mit der Zerlegung der Ebene in Trapeze. Das Verfahren zeichnet sich durch sein lokales Verhalten aus. Das schlägt sich auch im Speicherbedarf nieder, der in der folgenden Tabelle 1 abgebildet ist.

Tab. 1: Speicherbedarf des Algorithmus (in Megabyte)

Ingolstadt	Ingolstadt 500k	Ingolstadt 250k	Stadt Schlaining	Leithaprodersdorf
213	112	59	29	8

Eine Ermittlung des Gesamtzeitbedarfs des Algorithmus - sowohl inklusive als auch exklusive des Zugriffs auf den Sekundärspeicher - bestätigt die Laufzeit des Verfahrens, was bedeutet, dass eine Vervielfachung der Anzahl der Eingabesegmente um den Faktor *k* eine Vervielfachung der Gesamtrechenzeit um den Faktor *k* nach sich zieht (Abb. 9).

Abb. 9 bzw. Tab. 2 geben Auskunft darüber, welche Schritte des Algorithmus für Performanzeinbußen verantwortlich sind. Wie sich gezeigt hat, müssen sowohl während der Vor- als auch der Nachbearbeitungsphase die Zugriffe auf den Sekundärspeicher optimiert werden. Dies lässt sich dadurch realisieren, dass in Zukunft anstelle von ASCII-Dateien binäre Dateien verwendet werden.

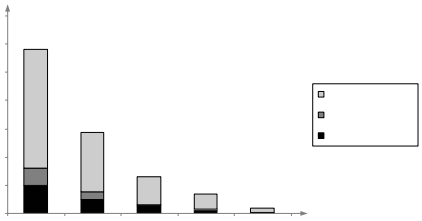


Abb. 9: Für die zur Verfügung stehenden Testdatensätze wurde die Gesamtlaufzeit des Algorithmus ermittelt. Die Grauschattierungen der Balken geben dabei den Zeitbedarf der einzelnen Schritte (Vorbereitung, Triangulierung, Nachbearbeitung) des Algorithmus an.

Tab. 2: Analyse des Zeitbedarfs (in sec.) der einzelnen Schritte des Algorithmus

	Daten erstellen	**Triangulierung**	**Ergebnis bilden**	**Gesamt (inkl. Disk)**	**Gesamt (exkl. Disk)**
Ingolstadt	20	12	84	116	42
Ingolstadt 500k	10	5	42	57	20
Ingolstadt 250 k	5	1	20	26	9
Stadt Schlaining	2	1	11	14	4
Leitha Prodersdorf	1	0	3	4	1

6 Resümee

In der vorliegenden Arbeit wurden zwei Verfahren aus der algorithmischen Geometrie zur automatischen Bildung der Topologie von Vektor GIS Daten vorgestellt. Die zwangsbasierte Triangulierung erwies sich dabei als die bessere Methode um die Topologie „realer" Geodaten zu ermitteln. Aufgrund des effizienten Speicherbedarfs der Methode ist die Behandlung großer Datenmengen, ohne eine Zerlegung der Daten in kleinere Recheneinheiten, möglich. Mit den, im Beitrag getesteten Datensätzen (bis zu 1.000.000 Liniensegmente) konnte die Leistungsgrenze auf einem Standard PC nicht ermittelt werden. In den Ausführungen wurde zudem gezeigt, wie die Ermittlung bzw. Elimination redundanter Punkte und Segmente, topologische Fehlerbereinigung und das eindeutige Zuordnen von Einsetzpunkten in den Ablauf des Algorithmus eingebunden sind.

Weitere Anwendungen, die von den entwickelten Algorithmen und der guten Performance profitieren, sind unter anderem: automatische Erzeugung von topologischen Vektordaten aus bestehenden Rasterkarten, Topologieüberlagerungen, sowie der digitale Teilungsplan.

Unabhängig von proprietären Formaten ermöglichen die vorgestellten Verfahren eine automatische Bestimmung der Topologie von Geodaten. Nur wenige Datenformate unterstützen topologische Informationen, so ist z. B. das beliebte Datenaustauschformat DXF nicht in der Lage, mit diesen Informationen umzugehen. Geodaten mit Topologie sollten auch in komponentenbasierte Gesamtsysteme integriert werden, die Operationen auf topologischen Geodaten ausführen können.

6.1 Danksagung

Besonderer Dank gilt Dr. David Heitzinger und Doz. Martin Peternell, die Wesentliches zum Projekt beitrugen. Die Forschungsergebnisse sind im K plus Kompetenzzentrum Advanced Computer Vision entstanden. Die Arbeit wurde aus dem K plus Programm gefördert.

7 Literatur

AGARWAL P. K. (1990): *"Partitioning Arrangment of Lines II: Application"*, In: Handbook of Discrete and Computational Geometry, Springer, New York, 449 – 483.

BERN M. & EPPSTEIN D. (1992): *Mesh Generation and Optimal Triangulation. Computing in Euclidian Geometry*, In: Lecture Notes Series on Computing, World Scientific, Singapore, Vol. 1,23-90.

CHAZELLE B., EDELSBRUNNER H., GUIBAS L. & SHARIR M. (1991): *Computing a Face in an Arrangement of Line Segments*, In: Proc. 2nd ACM - SIAM Symposium on Discrete Algorithms, 441 - 448.

DEBERG M., DOBRINDT K. & SCHWARZKOPF O. (1995): *On Lazy Randomized Incremental Construction*, In: Discrete Computational Geometry, Springer, New York, Vol. 14, 261 – 286.

DELAUNAY B. N. (1934): *Sur la Sphère Vide*, Izvestia Akademia Nauk SSSR, VII Seria, Otdelenie Matematicheskii I Estestvennyka Nauk 7: 793-800.

DE FLORIANI L., PUPPO E. & MAGILLO P. (1999): *Applications of computational geometry to geographical information systems*, In: Handbook of Computational Geometry, Elsevier Science, 333-388.

DWYER R. (1987): *A Faster Divide-and-Conquer Algorithm for Constructing Delaunay Triangulations.* Algorithmica 2(2), Springer, New York, 137-151.

GOLD, C.M. (1997): *Simple topology generation from scanned maps.* In: Proc. Auto-Carto 13, ACM/ASPRS; Seattle, WA, 337-346.

KRIVOGRAD, S. & ŽALIK B. (2000): *Constructing the topology from a set of line segments.* In: Proc. Spring conference on computer graphics 2000, Budmerice, Slovakia, 231-240.

PREPARATA F. & SHAMOS M. (1985): *Computational Geometry. An Introduction*, Springer Verlag, New York.

SAMET H. (1990): The *Design and Analysis of Spatial Data Structure*, Addison Wesley, Boston.

SHAMOS M. (1978): *Computational Geometry*, Dissertation, Yale University

SHEWCHUK J.R. (1996): *Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator*, 1st Workshop on Applied Computational Geometry, Philadelphia, Pennsylvania, 124-133.