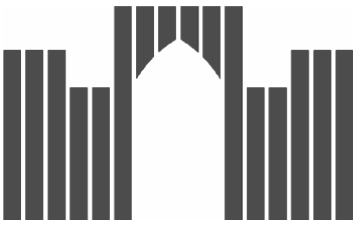
Arbeit zur Erlangung des akademischen Grades Bachelor of Science
in Informatik

Automatische Grenzfindung und Bereinigung von Karten unter kartografischen Aspekten anhand von ATKIS-Datensätzen

- Bachelorarbeit -



im Fachbereich Informatik und Medien
im Studiengang Informatik
der Fachhochschule Brandenburg an der Havel

vorgelegt von: Robin Kaluzny
Gerostraße 4
14770 Brandenburg an der Havel
0177 / 30 54 905

Matrikelnummer: 20052005

Erstbetreuer: Dipl.-Inform. Ingo Boersch
Zweitbetreuer: Dr. Rolf Lessing

Abgabetermin: 25.01.2010

Eidesstattliche Erklärung

Hiermit erkläre ich, dass ich diese Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und nicht veröffentlicht.

Brandenburg, den 21.01.2010

Kurzreferat

Diese Bachelorarbeit beschäftigt sich mit der automatischen Grenzziehung, bei der Satellitenbilder zum Einsatz kommen. Dabei wird gezeigt, wie durch die Zusatzinformation der *ATKIS-Grenzen* eine Verbesserung gegenüber des bisherigen Grenzziehungsprozesses erzielt werden kann. Die erweiterte Grenzziehung soll in die bestehenden Strukturen der Software *eConstruction* eingefügt werden. Dieses Konzept beruht auf Vergleichen und Berechnungen mit den geometrischen Abbildungen der ATKIS-Daten, wobei die Java-Bibliothek *JTS* zum Einsatz kommt. Es werden Hintergründe und Grundlagen der verwendeten Software erläutert sowie der Entwicklungsverlauf der entstandenen Komponenten beschrieben. Dieser beginnt im Kapitel „Theoretischer Ansatz“, geht über die „Architektur“ und endet in der „Implementierung“. Am Ende wird in der Schlussfolgerung der Funkionsnachweis erbracht.

Inhaltsverzeichnis

1 Einleitung **7**
1.1 Motivation . . . 7
1.2 Zielsetzung und Aufbau der Arbeit . . . 8
1.3 Szenario . . . 9

2 Grundlagen und Techniken **11**
2.1 Eingangsdaten . . . 11
2.1.1 RapidEye . . . 11
2.1.2 Vorklassifikation . . . 12
2.1.3 ATKIS . . . 14
2.2 eConstruction . . . 16
2.2.1 Umsetzung der kartografischen Aspekte durch das Objekt Repräsentant . . . 18
2.2.2 Das Setzen der Repräsentanten . . . 19
2.2.3 Grenzrepräsentanten und deren Assoziation zueinander . . . 20
2.2.4 Vom Grenzpunkt zur Teilgrenze . . . 22
2.2.5 Die Entstehung von Grenzen und Regionen . . . 23
2.3 Theoretischer Ansatz . . . 24
2.3.1 Veränderung der Grenzpunktberechnung durch ATKIS . . . 24
2.3.2 Die Neuberechnung des Ankerpunktes . . . 28

3 Architektur **40**
3.1 Die Klasse AtkisBPDetection.java . . . 42
3.2 Die Klasse AnchorCornerPointCreator.java . . . 43

4 Implementierung **45**
4.1 Line.java . . . 45
4.2 AtkisBPDetection.java . . . 47
4.3 AnchorCornerPointCreator.java . . . 49

5 Schlussfolgerung **55**

Abbildungsverzeichnis **I**

Tabellenverzeichnis **III**

Literaturverzeichnis **III**

1 Einleitung

Die menschliche Gestaltwahrnehmung ist die Fähigkeit, dreidimensionale Strukturen in Bildern zu erkennen. Dabei laufen mehrere Prozesse gleichzeitig sowie iterativ ab. Es werden Bildteile als auch das gesamte Bild wahrgenommen und mit vorhandenen Erkenntnissen verglichen.

Aufgrund dieses komplexen Prozesses stehen Satellitenbildinterpreten schon immer im Zwiespalt zwischen der Genauigkeit ihrer Analyse und der Arbeitszeit, die sie dafür benötigen.

Automatisierte Lösungen können es jedoch nicht mit der kognitiven Wahrnehmungsvielfalt des Menschen aufnehmen. eConstruction[1] ist eine Software der Firma Delphi IMM und beruht auf dem Prinzip, schon erworbene Information bei Abgrenzung von Objekten mit einzubeziehen.

Eine neue Vorinformation, die zur Grenzfindung hinzugenommen werden soll, sind die ATKIS-Datensätze. ATKIS ist das Amtliche Topographisch-Kartographische Informationssystem, dass Nutzungsdaten der Erdoberfläche bereitstellt. Die Einbindung dieses Vorwissens und die Bereinigung der Karten unter kartografischen Aspekten ist das Hauptziel dieser Arbeit.

1.1 Motivation

Die Software eConstruction war vor Beginn dieser Bachelorarbeit in der Lage, Satellitenbilder unter zwei Gesichtspunkten auszuwerten und Grenzen zu finden.

[1] siehe 2.2

Zum Einen das pixelorientierte Verfahren, bei dem jedem Pixel ein Farbwert zugeordnet wird. Im Folgenden wird angenommen, dass jeder Farbwert einer Objektart zugeordnet werden kann.

Zum Anderen das segmentbasierte Verfahren, bei dem zu Beginn Gruppierungen vorgenommen werden. Daraus können Formbewertungen entstehen.

Diese Verfahren sind recht erfolgreich, solange die zu trennenden Gruppen zumindest in einem Merkmal sich prägnant unterscheiden. Ist dies nicht so, fällt das automatische Erkennen sehr schwer.

Um das automatische Erkennen von sehr ähnlichen Flächen zu ermöglichen, wird durch das Hinzuziehen der Landnutzungsinformationen der ATKIS-Datensätzen, ein dritter Gesichtspunkt in das Grenzziehungsverfahren mit eingebunden.

1.2 Zielsetzung und Aufbau der Arbeit

Diese Bachelorarbeit hat als Ziel, eine funktionstüchtige Softwarekomponente zu erstellen, die die Grenzziehung unter Berücksichtigung der vorliegenden ATKIS-Grenzen durchführt und sich in eConstruction einfügt. Dabei sollen die für pixelbasierende Bilder typischen Rastermuster (siehe Abbildung 1.1) bei großem Zoom aus der Klassifikation elimeniert werden.

Zum besseren Verständnis werden im ersten Teil dieser Arbeit folgende Eckpunkte eingehend beantwortet:

- Welche Daten werden als Input benötigt?
- Welche kartografischen Aspekte werden von eConstruction umgesetzt?
- Was sind die Funktionsweisen von eConstruction?
- Welche Rolle spielt JTS?
- Wie werden die Informationen der ATKIS-Grenzen genutzt?
- Wie werden diese Informationen in eConstruction eingebunden?

Der zweite Teil der Arbeit wird sich mit der Architektur, der Funktionsweise und der Einordnung der Komponenten in eConstruction befassen.

Die Implementierung der erstellten Komponenten wird den dritten Teil dieser Arbeit beinhalten.

Viele der in der Arbeit verwendeten Quellen stammen aus den Aufzeichnungen von Herrn Christoph Kinkeldey, die während der Entwicklung von eConstruction entstanden. In der Sammlung ergeben sie die „Dokumentation eConstruction“.

1.3 Szenario

Die Firma Delphi IMM GmbH bietet seit 1997 Lösungen im Bereich der Geoinformatik an, die sich mit der Schaffung von Geoinfrastrukturen, Geoportalen und der Aktualisierung von Geodaten via Fernerkundung befassen. Diese Bachelorarbeit ist dem Thema der Aktualisierung von Geodaten zuzuordnen.

Der Auftraggeber die *Arbeitsgemeinschaft der Vermessungsverwaltungen der Länder der Bundesrepublik Deutschland* kurz AdV[2] möchte für das Projekt ATKIS[3] dem *Amtliches Topographisch-Kartographisches Informationssystem* eine Aktualisierung seines ATKIS-Datenbestandes durchführen.

Die Anforderungen des Auftraggebers sind, dass Änderungen in der Landschaft erfasst und dargestellt werden. Jedoch dürfen bestehende Grenzen nicht verändert werden, nur gegebenenfalls Neue hinzukommen. Das heißt auch, dass Grenzen die bis zu einem definierten Abstand in der Nähe der ATKIS-Grenzen liegen, nicht neu erfasst werden dürfen. Daher soll eConstruction Bezug auf die ATKIS-Grenzen nehmen, denn diese repräsentieren den Ist-Stand.

eConstruction soll den zur Zeit semi-automatischen Klassifikationsprozess ablösen, da das Verhältnis Arbeitszeit und Qualität nicht stimmen. Was heißt, dass Satellitenbildinterpreten, durch manuell einzustellende Software eine Vorklassifikation[4] erstellen, die jedoch sehr fehlerhaft ist. Dies macht es unumgänglich eine manuelle Prüfung des gesamten Gebietes vorzunehmen.

[2] http://www.adv-online.de
[3] siehe 2.1.3
[4] siehe 2.1.2

Abbildung 1.1: Ein Grenzverlauf der ohne eConstruction erstellt wurde. Deutlich sichtbar, das Rastermuster, des zugrunde liegende Satellitenbildes

Vor allem die Unterscheidung der Waldklassen und die Trennung von offenem Boden und Industriefläche stellen ein Problem dar. Bei diesen Objektklassen ist der Einsatz von eConstruction vorerst geplant.

Um dies zu realisieren, benötigt eConstruction:

- aktuelle Satellitenbilder, die durch die Firma RapidEye AG bereit gestellt werden
- die Vorklassifikation des zu betrachtenden Gebietes
- den ATKIS-Datensatz des zu betrachtenden Gebietes

2 Grundlagen und Techniken

In diesem Teil der Bachelorarbeit wird nun erläutert, welche Komponenten eConstruction benutzt, welche externen Techniken benötigt werden und wie eConstruction arbeitet. Zum Schluss wird der Ansatz beschrieben, wie die Nutzung des ATKIS-Datensatzes theoretisch erfolgen soll.

2.1 Eingangsdaten

eConstruction benötigt einige Eingangsdaten, die zur Zeit noch händisch erstellt werden müssen. Im Folgenden werden nun diese drei Datentypen genauer vorgestellt:

- RapidEye Satellitenbilder
- Vorklassifikation
- ATKIS-Datensatz

2.1.1 RapidEye

eConstruction benutzt Satellitenbilder zur Aktualisierung der ATKIS-Datensätze. Diese stammen von Satelliten der Firma RapidEye AG, eines in Brandenburg an der Havel ansässigen Unternehmens, welches 1998 in München gegründet wurde und 2004 seinen Sitz nach Brandenburg verlegte.

Am 29. August 2008 wurden die fünf Satelliten per Trägerrakete in eine sonnensynchrone Umlaufbahn gebracht.

Die technischen Spezifikationen[1] sind :

[1] Broschüre von RapidEye 15.01.2010

	Beschreibung	
Spektralkanäle	Blau	440–510 nm
	Grün	520–590 nm
	Rot	630–685 nm
	Red Edge	690–730 nm
	Nahes Infrarot	760–850 nm
Bodenauflösung (Nadir)	6,5 m	
Pixelgröße (orthorektifiziert)	5 m	
Breite der Bildstreifen (Nadir)	77 km	
Wiederholrate	1 Tag	
Äquatorüberflugszeit	ca. 11:00 Uhr lokale Sonnenzeit (von Nord nach Süd)	
Aufnahmekapazität	bis zu 4 Millionen km² täglich	

Abbildung 2.1: Technische Daten im Überblick

Die wichtigsten Daten, die diese Bachelorarbeit betreffen, sind die Auflösung und die auszuwertenden Kanäle des RapidEye-Sensors. Im Abschnitt eConstruction[2] werden diese beide Eigenschaften und deren Zusammenhang zur Software eingehend erläutert.

Ein vorzubereitender Schritt ist es, die erhaltenen Satellitenbilder auf acht Bit herunter zu rechnen - eConstruction kann mit einer höheren Bandbreite nicht umgehen - und in ihre fünf Kanäle zu zerlegen. Das ermöglicht, die Kanäle frei in Reihenfolge und in Anzahl zu variieren. Auf Grundlage dieser Satellitenbilder wird eine Vorklassifikation erstellt, die im nächsten Abschnitt beschrieben wird.

2.1.2 Vorklassifikation

Die Vorklassifikation ist ein wichtiges Hilfsmittel für eConstruction, denn es zeigt der Software eine ungefähre Abbildung der Verteilung der Objekte einer themati-

[2] siehe 2.2

schen Klasse aus dem *ATKIS-Objektartenkatalog*[3]. Die in Abbildung 2.2 zusehende Vorklassifikation liegt als Vektordatei vor, hat aber auf Grund der Ableitung von einem pixelbasierendem Bild dessen typisches Rastermuster.

Die Objekte in einer Vektordatei liegen als Polygone vor. Ein Polygon wird durch seinen Typ und den Punkten, aus denen es besteht, beschrieben.[2] Die Vektordatei im Format *.shp* von *ESRI*[4] vorhanden, wird als Standard angesehen. Dies ist notwendig, um die Datei ohne Aufwand weiterverarbeiten zu können.

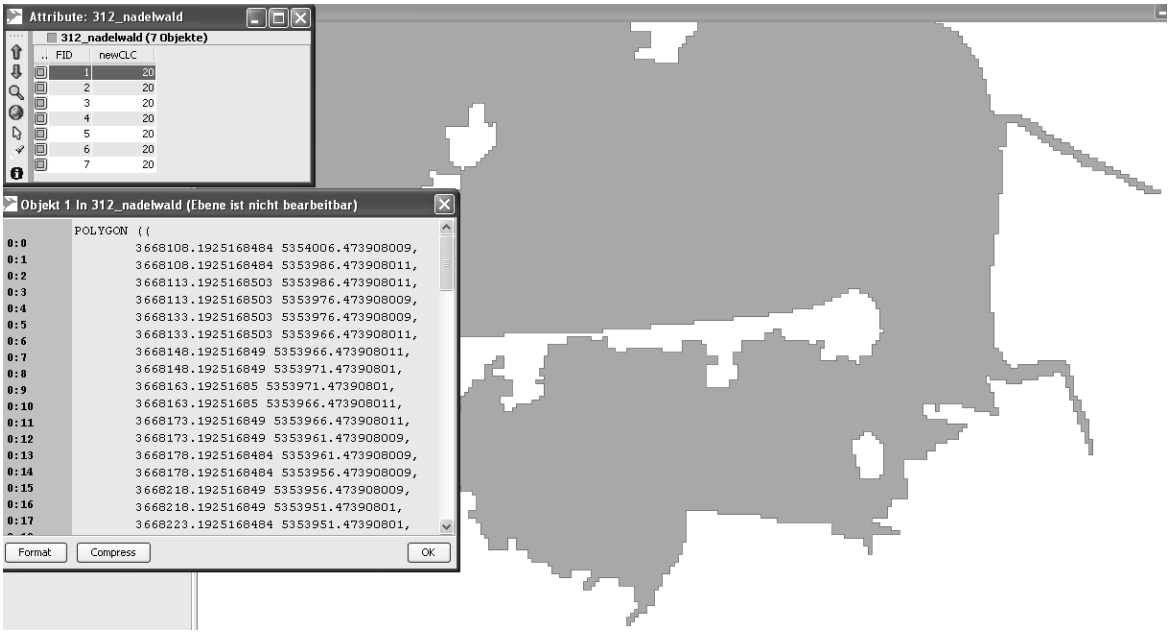


Abbildung 2.2: Vorklassifikation eines der Objektklasse Nadelwald, erstellt von einem Satellitenbildinterpreten

Die Objekte in einer Vektordatei sind alle vom selben Objekttyp, dementsprechend entstehen aus der Vorklassifikation so viele Vektordateien, wie gewünschte Objektklassen.[4] Das ist für die Firma Delphi IMM und das aktuelle Problem - die Unterscheidung der Waldklassen - sehr praktisch, da man auf diese Art alle unerwünschten Objektklassen zu einer Restklasse zusammenführen kann. Dies erspart viel Rechenaufwand und erleichtert das Prüfen der Ergebnisse.

Dieses Prinzip lässt aus einer großen Menge von Objektklassen, eine leicht überschaubare Menge von Laub-, Nadel- und Mischwald sowie der Restklasse entste-

[3] siehe 2.1.3
[4] http://www.esri-germany.de/

hen. Dieses Beispiel wird auch später bei der Erläuterung von eConstruction, dem theoretischen Ansatz, der Architektur und der Implementierung verwendet.

Bei der objektorientierten Klassifikation werden spektralähnliche Pixel zu Objekten zusammengeführt. Diese Objekte werden anhand des spektralen Mittelwertes einer Klasse zugewiesen, die durch den Objektartenkatalog von *ATKIS* festgelegt sind.

Dabei können zwei grundsätzliche Fehlerarten auftreten. Zum Einen der natürliche Fehler, nämlich dass zum Zeitpunkt der Aufnahme Dunst, Nebel oder Wolken auf dem Satellitenbild die Erkennung erschweren oder unmöglich machen.

Zum Anderen, Fehler, die durch die verwendete Software selbst entstehen. Bei spektralähnlichen Objektklassen z.B. Industrieflächen und offenen Boden ist eine Klassifikation so nicht möglich. Ein anderes Problem stellt die Unterscheidung der unterschiedlichen Waldklassen (Laub-, Nadel- und Mischwald) dar. Denn die große spektrale Ausprägungsvielfalt der Waldarten gepaart mit der hohen spektralen Ähnlichkeit (junger Nadelwald ähnelt Laubwald, junger Nadelwald ähnelt Mischwald) macht die Klassifikation und somit die Grenzziehung sehr anspruchsvoll.

Basierend auf dieser Vorklassifikation erstellt eConstruction nun Repräsentanten, die die kartografischen Aspekte *Generalisierung* und *Mindestgröße* umsetzen. Darauf wird später bei der Funktion der Repräsentanten im Abschnitt *eConstruction*[5] eingegangen.

2.1.3 ATKIS

Das amtliche Topographisch-Kartographische Informationssystem vom AdV bietet Informationen über die Topografie der Erdoberfläche in digitaler Form an, um die Datenverarbeitung zu vereinfachen. Diese Daten sind offen, um sie mit anderen Daten von Nutzern zu verknüpfen und liegen in leicht veränderter Form vor. Es sind keine Polygone - Objekte mit einer Fläche - im Datensatz vorhanden, sondern sogenannte Polylines, die Grenzlinien der Objekte. Diese werden später mit den Grenzen von eConstruction verglichen.

[5] siehe 2.2

Das Herzstück von ATKIS ist dessen Objektartenkatalog[6], in dem festgelegt ist, welche Attribute eine Objektart besitzt. Auf Grundlage dieses Kataloges erfolgt die thematische Unterscheidung zwischen den einzelnen Objektarten.

Wie auch die Vorklassifikation ist der ATKIS-Datensatz, siehe Abbildung 2.3, im Format *.shp*[2] gespeichert und besteht grundlegend aus dem Polygontyp *MultiLine* (mehrfach Linie) und ist ein zyklischer Graph. Eine MultiLine besteht aus mehreren *LineStrings*(Linien), die azyklisch sind. Ein LineString wird durch seine enthaltenen Punkte beschrieben. Teilstücke eines LineStrings sind Geraden von einem Punkt zu seinem Nachbarpunkt. Zur Auswertung ist es teilweise notwendig, jedes Teilstück eines ausgewählten LineStrings abzugleichen. Dies wird ausführlich im Abschnitt *Verbesserung der Grenzpunkte durch ATKIS*[7] beschrieben.

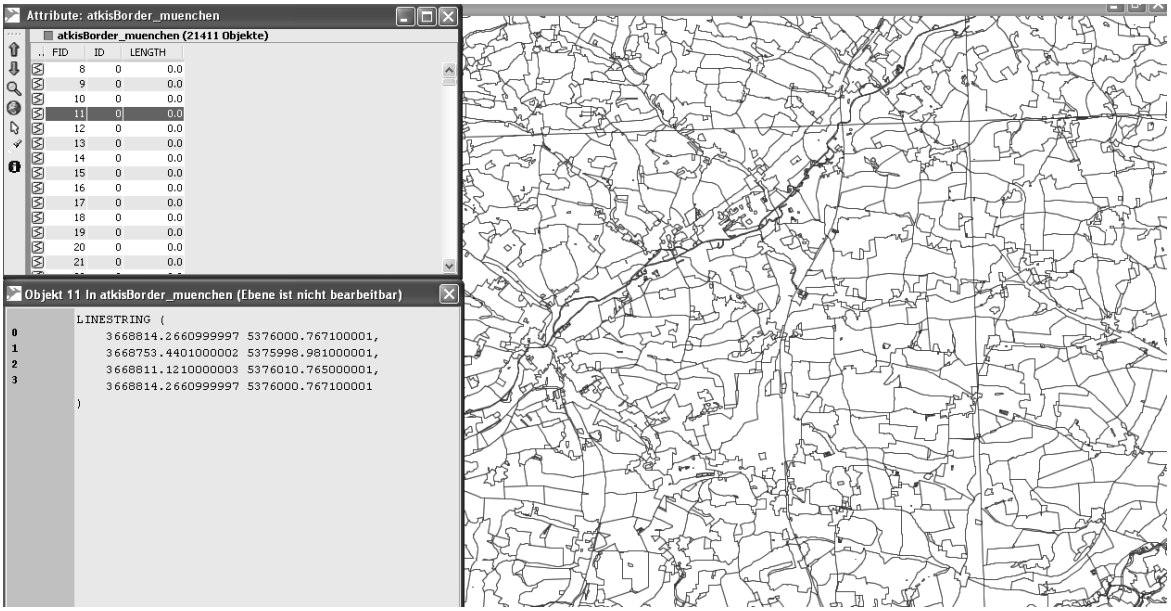


Abbildung 2.3: Ein ATKIS-Datensatz: bestehend aus LineStrings

eConstruction ist für die Aktualisierung auf die Referenz der ATKIS-Daten angwiesen. Jedoch bezieht sich die Software nur auf die geometrischen Eigenschaften der ATKIS-Datensätze. Die thematischen Eigenschaften sind für die Berechnung der Grenzen nicht notwendig. Dem Nutzer ist, durch das Prinzip der Objektklassen, die thematische Einordnung überlassen. Anhand der Eingangsdaten konstruiert eConstruction Regionen, die der thematischen Darstellung der Objektarten von

[6]Objektartenkatalog
[7]siehe 2.3.1

ATKIS entsprechen. Aus dieser thematischen Trennung erfolgt die Grenzziehung. Dieser komplexe Vorgang ist Inhalt des nächsten Abschnittes *eConstruction*.

2.2 eConstruction

eConstruction ist eine voll funktionstüchtige Software, war vor dieser Bachelorarbeit vorhanden und wurde von Christoph Kinkeldey entwickelt. Sein Wissen um eConstruction, welches in diese Arbeit mit einfließt, kommt aus der Sammlung „eConstruction Dokumentation“[4]. Um ein besseres Verständnis zu vermitteln, was im Laufe dieser Arbeit an Mehrwert hinzugekommen ist, wird der Arbeitsablauf beschrieben. An den entscheidenden Stellen wird ein Hinweis erfolgen und auf die neuen Komponenten im theoretischen Ansatz verwiesen.

eConstruction setzt Vorwissen bei der Abgrenzung von Objekten ein. Dazu werden die voneinander abzugrenzenden Objekte, Schritt für Schritt anhand von Homogenitätskennzahlen und Farbwerten verglichen.[8]

Diese Abgrenzung erfolgt unter Berücksichtigung zweier kartografischer Aspekte:

- Generalisierung
- Mindestgröße

Diese beiden Merkmale dienen der besseren Visualisierung von Karten für den Nutzer.

Der *Grad der Generalisierung* ist das wichtigste Merkmal in der Kartografie. Es dient der Beschreibung der Umwelt und basiert auf zwei Prinzipien. Bereits Grundkarten stehen zur Realität in einem Verkleinerungsverhältnis und erfordern einen gewissen Generalisierungsgrad. Würde man Karten fotografisch verkleinern, so würden bei fortgesetzter Verkleinerung, Objekte unterhalb der *grafischen Mindestgröße* fallen. Da dies nicht umsetzbar ist, muss man sich für eines der zwei Prinzipien entscheiden.

Beachtet man das *Prinzip der Lesbarkeit*, muss man das Objekt unverhältnismäßig vergrößern und schränkt damit die Richtigkeit ein.

[8] Delphi IMM, 15.12.2009, 15:33

Betreibt man *Verzicht auf Wiedergabe*, durch die angesprochene Verkleinerung, verzichtet man auf die Vollständigkeit. Dieser Verzicht kann durch die geringe Objektbedeutung oder aus Mangel an Darstellungsfläche begründet werden.[1][5]

In eConstruction wird vor allem das Prinzip Verzicht auf Wiedergabe angewendet.

Die Mindestgröße ist deshalb für eConstruction so relevant, weil die Auflösung von RapidEye fünf Meter pro Pixel ist. Interne Analysen der Firma Delphi IMM haben ergeben, dass Strukturen erst ab einer Mindestgröße von vier mal vier Pixel erkannt werden können. Die zwei folgenden Abbildungen zeigen das beispielhaft für eine besiedelte Fläche von ca. einhundert mal achtzig Meter.

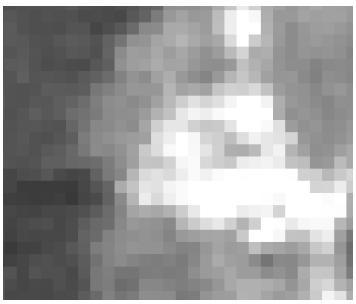


Abbildung 2.4: Gezeigt wird eine besiedelte Fläche im großen Zoom.

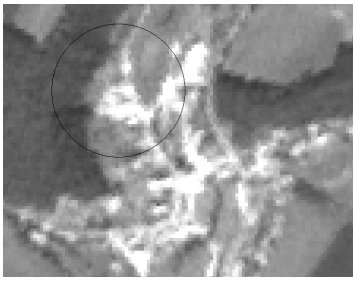


Abbildung 2.5: Selbige Fläche, etwas heraus gezoomt

Dies ergibt eine quadratische Fläche von zwanzig mal zwanzig Meter. Der wie schon erwähnte Haupteinsatzzweck soll die Grenzziehung in Waldgebieten sein. Dementsprechend vernachlässigt eConstruction, wenn sich Bäume oder kleinere Baumgruppen in dieser Größenordnung befinden.

2.2.1 Umsetzung der kartografischen Aspekte durch das Objekt Repräsentant

Um das Merkmal der *Generalisierung* umzusetzen, wurde das Objekt *Repräsentant* erschaffen. Ein Repräsentant (siehe Abbildung 2.6) ist kreisförmig und definiert sich über seinen Radius und den Abstand zu seinen Nachbarn. Diese Größen sind beliebig wählbar, um sich an verschieden große Objekte anpassen zu können. Dargestellt wird ein Repräsentant jedoch als zwanzig-eckiges Polygon.

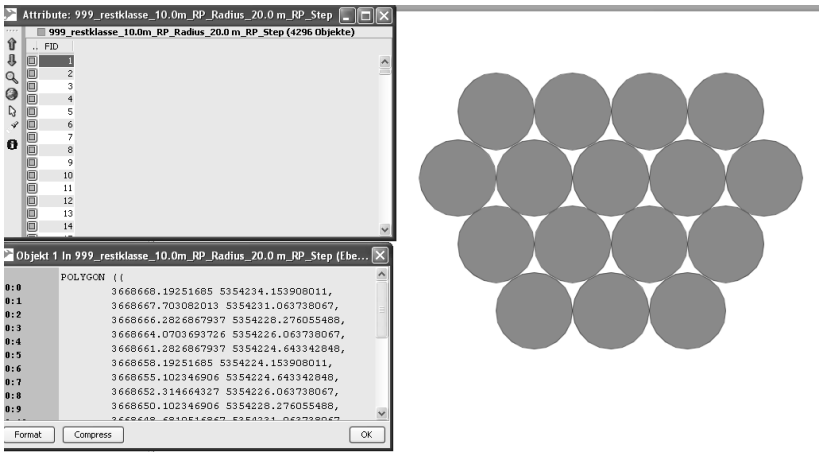


Abbildung 2.6: Repräsentanten im typischen Anordnungsmuster

Ein Repräsentant besitzt ein Gitter von Scanpunkten, welches unterschiedlich von Struktur und Engmaschigkeit sein kann. Pro Scanpunkt wird der Farbwert ermittelt und aus der Summe von Farbwerten Attribute für den gesamten Repräsentanten abgeleitet. Dabei stechen dominantere Attribute - die von eConstruction bewertet wurden - heraus und werden übernommen, während Rezessive wegfallen. Basierend auf Größe des Repräsentanten und die Anzahl seiner Scanpunkte sowie dem Abstand zu seinen Nachbarn wird das Prinzip *Verzicht auf Wiedergabe* umgesetzt, da zu kleine Objekte oder Strukturen mit Attributen, die nicht die Mehrheit der Scanpunkte widerspiegelt, durch das Scangitter fallen.

Wie Abbildung 2.7 zeigt, wird dadurch auch das Rastermuster aus der Vorklassifikation im Informationsfluss elemeniert, da nicht mehr jedes einzelne Pixel betrachtet wird. Das genaue Prinzip wird in den folgenden drei Abschnitten erläutert.

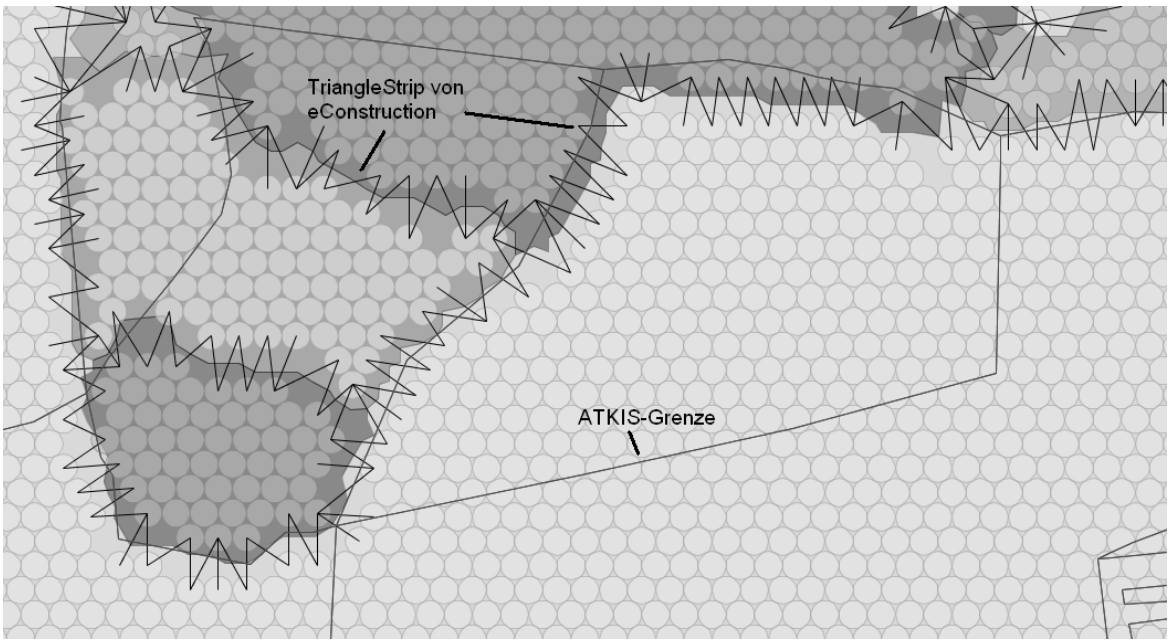


Abbildung 2.7: Grenzziehung mit eConstruction, ohne Hinzunahme von ATKIS-Daten

Ein Repräsentant steht also für eine bestimmte Fläche in der Realität, welche für diesen Bereich dominante Attribute besitzt. Bei den verschiedenen Klassen des Waldes setzen sich die Attribute der Inhomogenität und der spektrale Abstrahlung im nahen Infrarot-Bereich durch.

2.2.2 Das Setzen der Repräsentanten

Der vorgenannten Darstellung eines Repräsentanten folgt nachstehend die Beschreibung zum Einsatz. Die Vorklassifikation, die aus mehren Objektklassen besteht, wird sukzessiv einzeln befüllt (Abbildung 2.8).

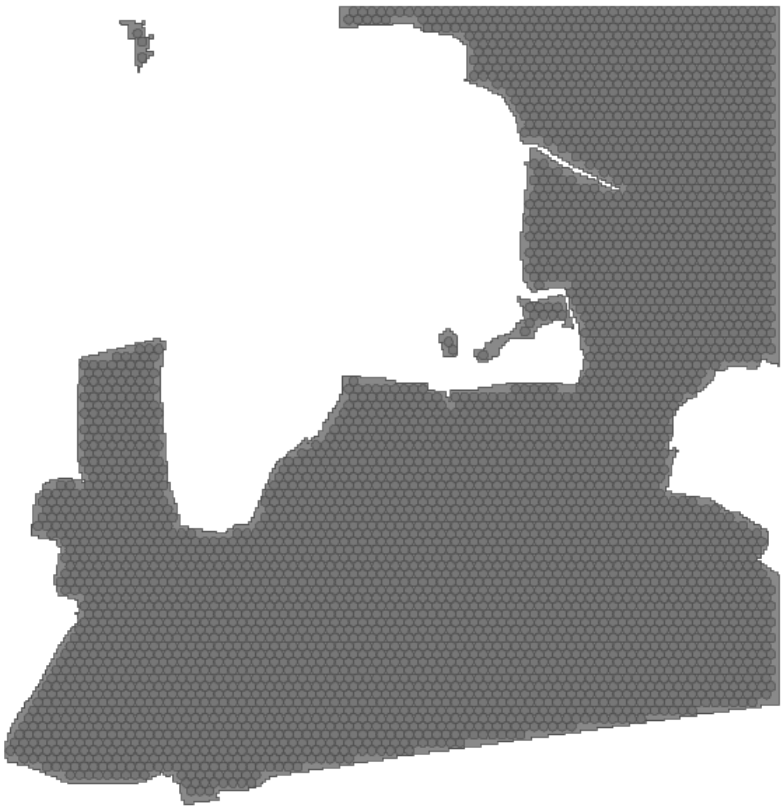


Abbildung 2.8: Befüllung einer Objektklasse mit Repräsentanten

Durch das Abgleichen, ob sich ein Repräsentant innerhalb eines Polygons der Objektklasse befindet, wird ein Repräsentant gesetzt oder nicht. So werden alle Polygone einer Objektklasse mit Repräsentanten befüllt.

Es entsteht aus der Vorklassifikation eine Ableitung, die ähnliche Strukturen wie die Vorklassifikation besitzt, jedoch aus Repräsentanten besteht und einen gewissen Grad der Generalisierung beinhaltet.

2.2.3 Grenzrepräsentanten und deren Assoziation zueinander

Der Nachbar eines Repräsentanten, der nicht vom selben Objekttyp ist, muss ein *Grenzrepräsentant* sein. Alle Grenzrepräsentanten, die eine geschlossene Fläche bilden, gehören zu einer *Repräsentantengruppe*. Aus diesen Repräsentantengruppen entstehen später konkrete Polygone, die einer Objektklasse zuzuordnen sind.

Zwischen einem Repräsentanten und seinen direkten Nachbarn der verschiedenen Repräsentantengruppen werden Verbindungslinien konstruiert, die sogenannten Connections (siehe Abbildung 2.9). Dies geschieht auf Grundlage der **Delaunay-Triangulation**, einem Verfahren, welches genutzt wird, um aus einer Punktmenge - in diesem Fall die Mittelpunkte der Repräsentanten - ein Dreiecksnetz zu erstellen.[3] Der Algorithmus wird nur erwähnt, nicht erläutert. Für weitere Informationen siehe [3] und [4].

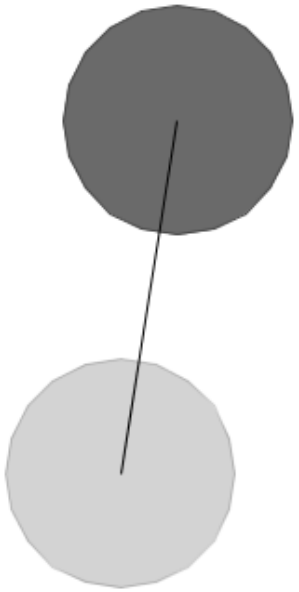


Abbildung 2.9: Repräsentanten die durch eine Connection verbunden sind

Die gebildeten Connections verlaufen von Mittelpunkt zu Mittelpunkt der zwei beteiligten Repräsentanten. Auf diesen Connections findet die eigentliche Grenzziehung statt. Dies wird mit verschiedene Algorithmen realisiert, die vom Benutzer frei wählbar sind. Zwei davon sind zum Beispiel *treshhold* und *maxGradient*. Der Erste arbeitet mit Schwellen von Grauwerten, der Zweite mit dem mathematischen Differentialoperator Gradient.

An diesem Punkt sollen die Informationen der ATKIS-Daten eingebracht und von eConstrution ausgewertet werden.[9]

[9] siehe 2.3.1

2.2.4 Vom Grenzpunkt zur Teilgrenze

Connections die einen gemeinsamen Punkt besitzen, bilden einen TriangleStrip, eine zusammenhängende Kette die von zwei Dreiecken begrenzt wird (siehe 2.10). Mehrere dieser Ketten bilden einen Graph, dessen Knoten die erwähnten Dreiecke sind. Im Fall von eConstruction ist das ein Dreieck, bei dem mindestens zwei Eckpunkte Repräsentanten besitzen, die verschiedenen Objektklassen angehören.

Diese Dreiecke, auch *Ankerdreiecke* genannt, nehmen bei der Grenzziehung eine spezielle Rolle ein. Betrachet man nur die Grenzpunkte auf den Connections aus denen ein Ankerdreieck besteht, würde die Fläche des Dreieckes bei der Grenziehung nicht berücksichtigt werden.

Zur Vermeidung wurde ein Zusatzpunkt für alle Ankerdreiecke geschaffen. Dieser befand sich vor dieser Bachelorarbeit immer an der selben Stelle. Es handelt sich dabei um den Inkreismittelpunkt, der aus den Winkelhalbierenden eines Dreieckes berechnet wird.

Im theoretischen Ansatz wird gezeigt, dass dies nicht immer erwünscht ist und wie nach festgelegten Kriterien anders entschieden wird.[10] Dies ist der zweite Punkt an dem eine Bewertung der ATKIS-Daten durch eConstruction erfolgt.

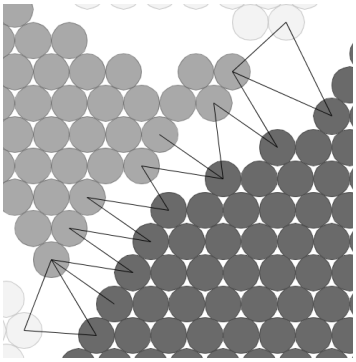


Abbildung 2.10: TriangleStrip zwischen zwei Ankerdreiecken

Zwischen zwei Ankern verläuft also ein Teil des Triangulationsnetzes, welches den Namen *TriangleStrip* besitzt. Dieser TriangleStrip besteht aus einer Anzahl von Connections, die jeweils einen Grenzpunkt beinhalten. Durch die Verbindung der einzelnen Grenzpunkte ergibt sich eine neue Teilgrenze, in dem die ermittelten Punkt

[10] siehe 2.3.2

einer Liste hinzugefügt werden. Diese Liste wird einem ShapeWriter übergeben der daraus ein Polygone erschafft (siehe Abbildung 2.11). Da diese Teilgrenze nicht mehr auf der Vorklassifikation beruht, sondern auf den Grenzpunkten der Connections zwischen den Repräsentanten, ist das Rastermuster des ursprünglichen Satellitenbildes elemeniert.

2.2.5 Die Entstehung von Grenzen und Regionen

Durch das Zusammenfügen von Teilgrenzen (TriangleStrips), entstehen geschlossene Grenzen, die eine Fläche einschließen. Aus dieser Fläche wird ein Polygon konstruiert, welches eine thematische Zugehörigkeit hat. Diese Fläche wird auch als Region in eConstruction bezeichnet und beschreibt die geografische Ausdehnung einer realen Objektart. Diese Regionen und Grenzen werden als Vektordatei im .shp-Format abgespeichert. Eine, in Abbildung 2.11 zusehende, Regionen-Vektordatei beinhaltet nur Regionen der selben Objektart, während die Vektordatei des ATKIS-Datensatzes alle Grenzen enthält.

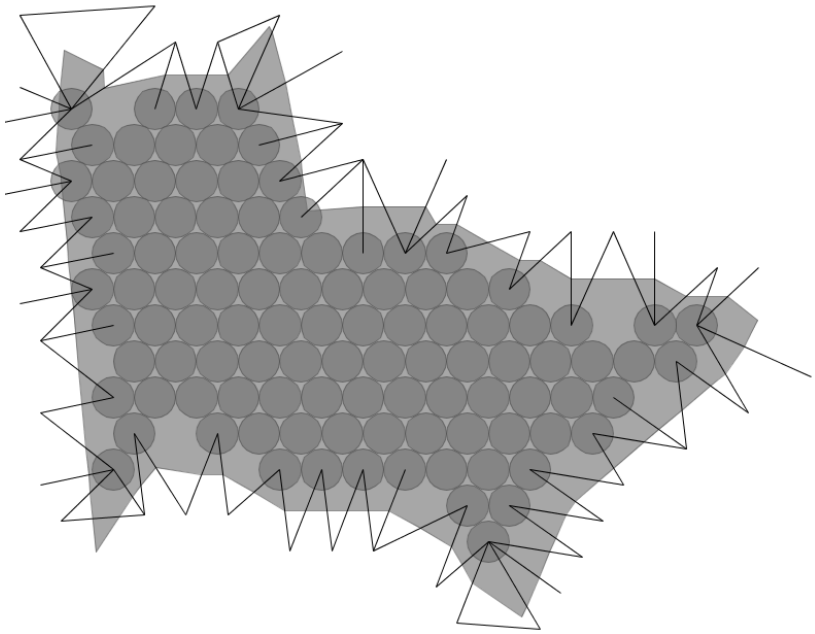


Abbildung 2.11: TriangleStrips bilden eine geschlossene Fläche. Eine Region entsteht.

Das vorstehende Kapitel gibt eine grobe Beschreibung des Arbeitsablaufs von eConstruction wieder. Dadurch wird verdeutlicht, in welchem Arbeitsabschnitt die ATKIS-

Daten benutzt werden. Die nachstehende detaillierte Erläuterterung erklärt die Benutzung und Auswertung.

2.3 Theoretischer Ansatz

In diesem Abschnitt sollen die Überlegungen und Ideen erläutert werden, die während der Bachelorarbeit entwickelt und zum Einsatz gebracht wurden:

- Die Einbeziehung der ATKIS-Grenzen,
- und deren korrekte Auswertung.

Inwiefern das passierte, wird im Schlussteil zusammen gefasst. Es wird darauf eingegangen, wie die ATKIS-Daten auszuwerten sind, welche Probleme dabei auftreten und die dadurch entstehenden Vorteile beschrieben. Im Kapitel Architektur werden diesen Überlegungen konkretisiert, sodass im darauffolgenden Abschnitt „Implementierung“ die entstandene Lösung gezeigt wird. Die Grenzziehung soll an genau zwei Stellen, durch das Einfügen der ATKIS-Grenzen erweitert werden:

- bei der Berechnung der Grenzpunkte auf den Connections
- und der Neuberechnung der Ankerpunkte

2.3.1 Veränderung der Grenzpunktberechnung durch ATKIS

Die Überlegung ist, dass während der Grenzziehung, entschieden werden muss, ob die von eConstruction berechnete oder die vorgegebene ATKIS-Grenze benutzt werden soll. Das Kriterium dafür ist, ob eine ATKIS-Grenze eine Connection schneidet. Ist kein Schnittpunkt vorhanden, wird der vorhandene Grenzziehungsalgorithmus benutzt.

Wie im Abschnitt ATKIS beschrieben, besteht eine solche Grenze aus Punkten eines zweidimensionalen Koordinatensystems. Schneidet eine ATKIS-Grenze eine

Connection ist das ein, aus zwei nebeneinander liegenden Punkte, bestehendes Teilstück der ATKIS-Grenze (siehe Abbildung 2.12). Dementsprechend beruht diese Art der Grenzfindung auf geometrischen Berechnungen. Vor allem Schnittpunkte von Geraden werden berechnet.

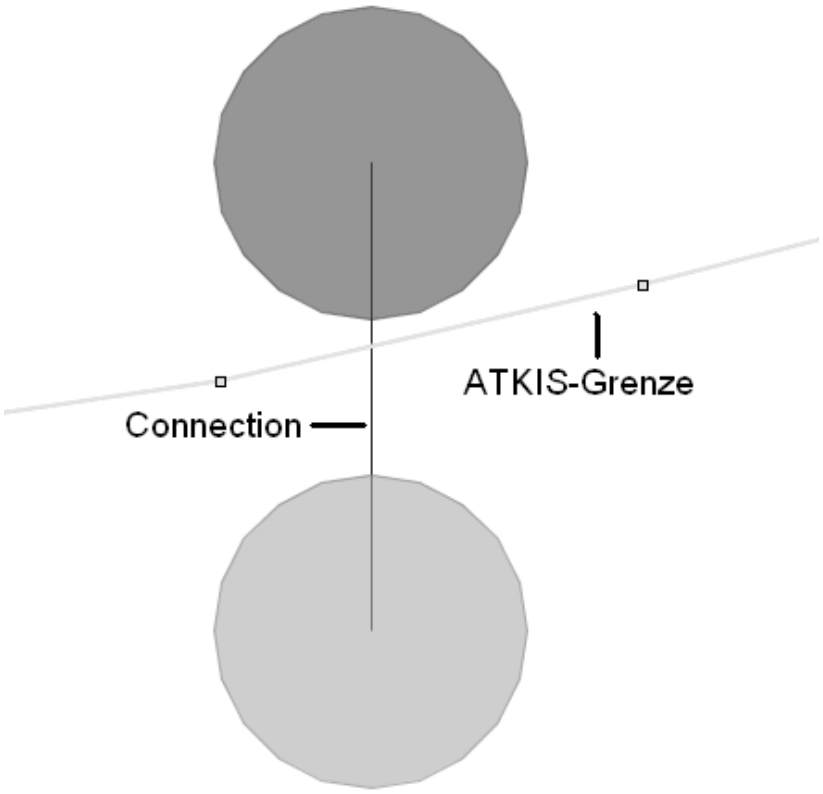


Abbildung 2.12: Connection schneidet ein Teilstück der ATKIS-Grenze

Für diese Berechnungen wird die JTS[11] benutzt. JTS ist eine freie in Java geschriebene Programmbibliothek. Diese Suite stellt ein räumliches Objektmodell und grundlegende geometrische Funktionen bereit.

Vor der Berechnung müssen drei Fälle unterschieden werden. Jede Connection wird auf Schnittmenge mit allen Teilstücken des ATKIS-Datensatz geprüft. Bis zu einem gewissen Grad an Komplexität übernimmt das die JTS-Bibliothek.

Erstens, Connection und ATKIS-Grenze haben keinen Schnittpunkt. Man geht davon aus, dass es eine im ATKIS-Datensatz nicht vorhandene Unterscheidung von

[11] Java Topology Suite

Objektklassen gibt. Der Grenzfindungsalgorithmus von eConstruction wird benutzt. Dadurch entsteht eine neue Grenze.

Zweitens, Connection und ATKIS-Grenze schneiden sich einmal. Dies ist der Fall, wenn ATKIS-Grenze und die konstruierte Grenze von eConstruction ähnlich sind. Die ATKIS-Grenze muss auf Grund der geringen Abweichung benutzt werden. Denn wie schon beschrieben, sind Strukturen erst ab einer bestimmten Mindestgröße erkennbar. Diese Mindestgröße ist ein Repräsentant mit dem Radius von zehn Meter.

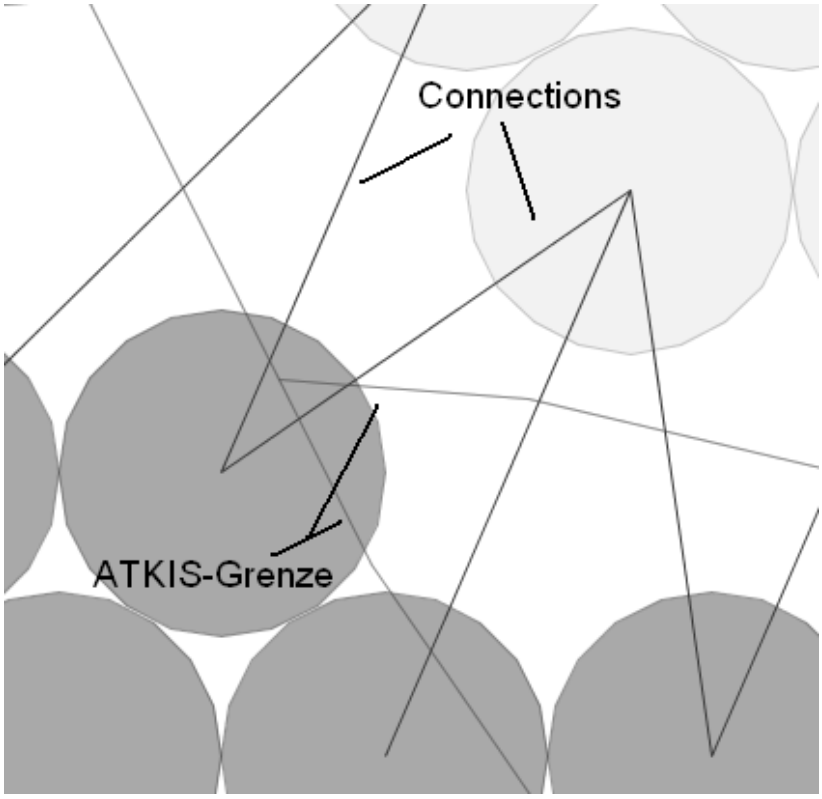


Abbildung 2.13: TriangleStrip wird von ATKIS-Grenze geschnitten. Resultat: Ein oder Zwei Schnittpunkt/e

Drittens, Connection und ATKIS-Grenze besitzen zwei Schnittpunkte. Das tritt ein, wenn mehr als zwei Objektarten von ATKIS aufeinander treffen. Dabei entsteht eine Kreuzung der LineStrings. Es ist vorgesehen, dass der Grenzpunkt dann in diese Kreuzung gesetzt wird (siehe Abbildung 2.14). An dieser Stelle wird absichtlich nicht von einem Schnittpunkt gesprochen. Denn alle gemeinsamen Punkte der LineStrings werden von JTS als Schnittpunkte ausgegeben.
Aus diesem Grund nimmt man einen notwendigen Umweg in Kauf.

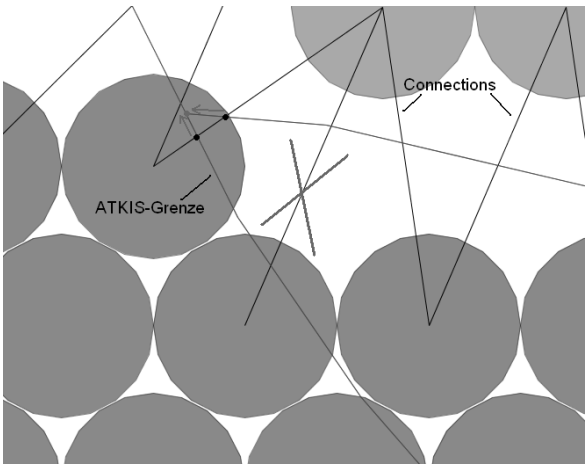


Abbildung 2.14: Berechnung eines neuen Anker durch Erzeugung zweier Hilfsgeraden.

Es ist möglich, den Punkt (*KP*) dieser Kreuzung zu konstruieren. Es werden die Teilstücke der beteiligten LineStrings ermittelt und mit den am nächsten liegenden Punkten der Teilstücke zu *KP* zwei Geraden konstruiert, die einen Schnittpunkt ergeben. Dieser ist im besten Fall der gleiche wie der ursprüngliche *KP*.

Nun kann ein zusätzlicher Fall eintreten, dass eine zweite Connection hinter der Ersten entstanden ist. Diese weiter Entfernte darf nicht berücksichtigt werden, da die Connection nicht die Teilstücke schneidet, die *KP* als Punkt besitzen. Dadurch würde es zu Ungenauigkeiten kommen, welche man durch einen zusätzlichen Filter minimieren möchte.

Es können nicht die Distanzen von einer Connection zu *KP* für einen Vergleich genutzt werden. Denn die vorhandenen Strukturen von eConstruction geben vor, dass zur Laufzeit nur eine Connection betrachtet wird. Es wurden Testreihen durchgeführt, um einen Schwellwert zu finden. Dieser Schwellwert sagt aus, bis zu welcher Entfernung eine Connection nutzbar ist. Die Länge der Connection und somit die Fähigkeit zwei Schnittpunkte bei großer Distanz zum *KP* zu erhalten, hängt direkt mit der Größe der Repräsentanten zusammen. Resultat der Testreihe ist, dass der 1,75-fache Radius der Repräsentanten zum besten Ergebnis führt, wie die folgende Tabelle belegt. Das heißt: 95% der Connections sind korrekt und ergeben den gleichen Schnittpunkt wie *KP*.

Zusammenfassend kann man sagen, dass es nicht möglich ist alle Spezialfälle, die die Realität vorgibt, ab- oder nachzubilden. Doch die hohe Trefferquote der Über-

Versuch	Multiplikator des Radius	Nutzbar	Fehler
1	1,75	ja	-
2	1,75	ja	-
3	1,75	ja	-
4	1,75	ja	-
5	1,75	ja	-
6	1,75	ja	-
7	1,75	ja	-
8	1,75	ja	-
9	1,75	ja	-
10	1,75	ja	-
11	1,75	ja	-
12	1,75	ja	-
13	1,75	ja	-
14	1,75	nein	ja
15	1,75	ja	-
16	1,75	ja	-
17	1,75	ja	-
18	1,75	ja	-
19	1,75	ja	-
20	1,75	ja	-

Tabelle 2.1: Übersicht der Testreihe

einstimmungen in Verbindung mit der Ungenauigkeit unterhalb der Mindestgröße ist durchaus zufriedenstellend. Noch dazu sind im Laufe dieser Bachelorarbeit weitere Ideen zu diesem Projekt entstanden, die den Arbeitsablauf optimieren werden.

Im nächsten Abschnitt wird nun gezeigt, wie unter Berücksichtigung der ATKIS-Daten die Ankerpunktberechnung verändert werden soll.

2.3.2 Die Neuberechnung des Ankerpunktes

Ankerdreiecke sind Anfang und Ende der *TriangleStrips*. Da ein TriangleStrip eine noch nicht konstruierte Teilgrenze ist, verbinden Ankerdreieck nicht nur die Teilgrenzen sondern halten die unterschiedlichen Regionen zusammen. Dieses Zusammentreffen sollte idealerweise in einem Punkt geschehen.

Ein Dreieck ist eine Fläche. Es muss also ein Punkt gefunden werden, der diese Aufgabe übernimmt. Dieser Punkt nennt sich **Anker** und muss konstruiert, da er nicht Teil der Input-Daten ist. Aktuell wird das über den Mittelpunkt des Inkreises realisiert, der durch die Winkelhalbierenden eines Dreieckes zustande kommt. Dieser Punkt berücksichtigt nicht die ATKIS-Daten. Dadurch entstehen Unregelmäßigkeiten im Grenzverlauf, die beseitigt werden müssen.

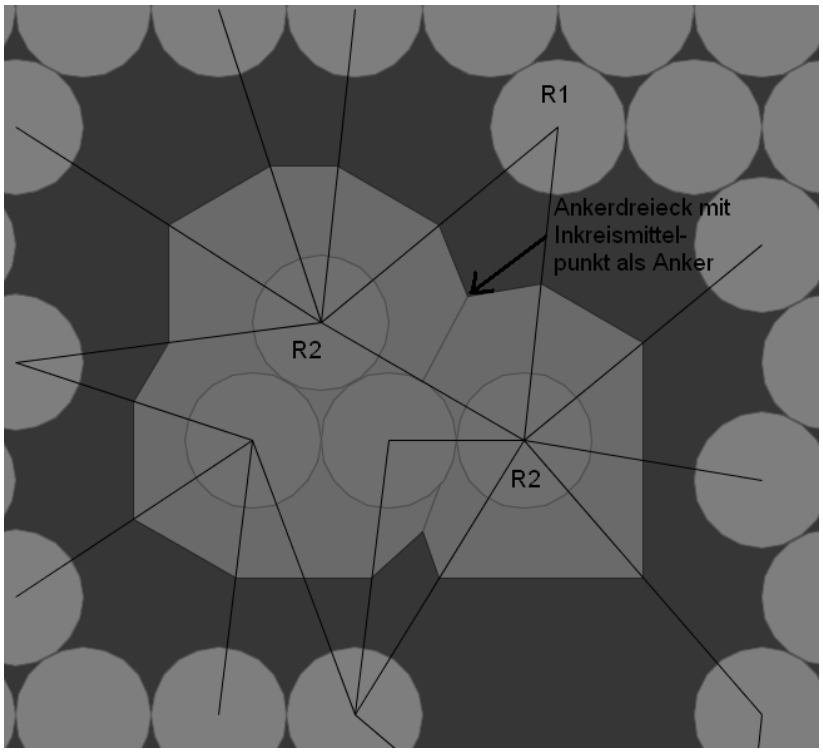


Abbildung 2.15: Entstehung von Einschnitten mit Ankerpunkten, die auf dem Inkreis beruhen.

Das Ankerdreieck gibt vor, in welcher räumlichen Ausdehnung der Anker liegen kann. Dementsprechend muss das Ankerdreieck auf Schnittmengen mit den ATKIS-Daten geprüft werden. Der Anker selbst muss dann so berechnet werden, dass dieser im besten Fall auf einer ATKIS-Grenze liegt. Ein gewünschter gleichmäßiger Grenzverlauf wäre die Folge.

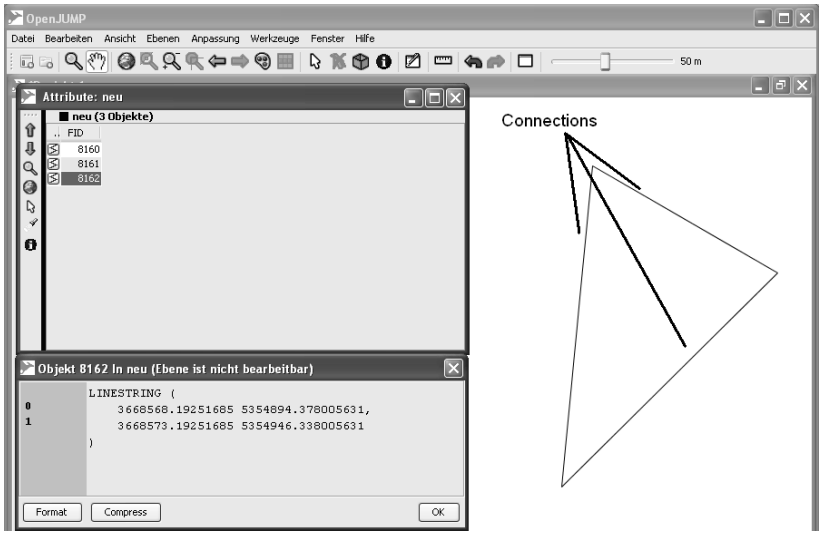


Abbildung 2.16: Ankerdreieck bestehend aus drei Connections

Da eine Fläche von unterschiedlichen Geraden - in diesem Fall, Teilstück der ATKIS-Grenze - unendlich oft geschnitten werden kann, ergibt das unendlich viele Schnittpunkte. Dies führt beim ersten Anblick zu einem Problem, dass durch eine klare Vorbedingung gelöst werden kann. Es wird nur auf maximal vier Schnittpunkte geprüft. Das entspricht dem Schneiden zweier Teilstücke der ATKIS-Grenze mit dem Ankerdreieck. Bei mehr als vier Schnittpunkten geht man davon aus, dass die Struktur so komplex ist, dass sie maschinell nicht korrekt bewertet werden kann.

Es folgt eine Fallunterscheidung mit den Bedingungen:

- keine Schnittmenge
- zwei Schnittpunkte
- drei Schnittpunkte
- vier Schnittpunkte
- mehr als vier Schnittpunkte

Der Fall, dass ein Schnittpunkt existiert, ist unmöglich und wird nicht beachtet, da es als Voraussetzung hätte, dass eine Grenze in ein Ankerdreieck hinein läuft, jedoch nicht wieder hinaus.

Kein Schnittpunkt

Ankerdreiecke bestehen aus drei einzelnen nicht verknüpften Connections (Abbildung 2.15). Für den Fall, dass **kein Schnittpunkt** existent ist, kann wie bei der Grenzpunktberechnung angenommen werden, dass es noch keine Unterteilung dieses Gebietes in Unterklassen gibt.

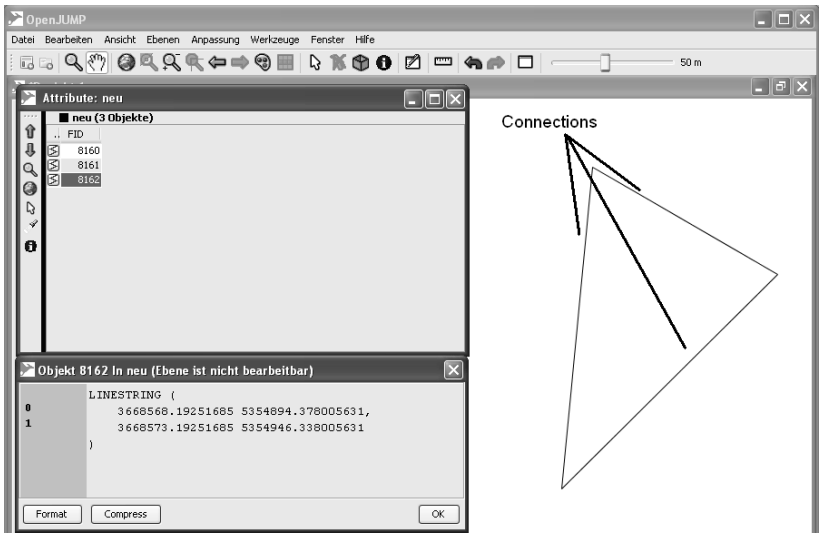


Abbildung 2.17: Ankerdreieck bestehend aus drei Connections

Dennoch kann eine Verbesserung erwirkt werden. Wenn zwei (R2) der drei Repräsentanten einer Objektart angehören, sind an diesem Ankerpunkt auch nur zwei Regionen verknüpft. Eine Grenze sehr nah betrachtet kann als Gerade angesehen werden. Diese verbindet die beiden Mittelpunkte der Connections der Repräsentanten R2 und des einzelnen Repräsentanten R1. Der Ankerpunkt ist nun der Mittelpunkt dieser konstruierten Geraden. Verdeutlichen soll das folgende Abbildung:

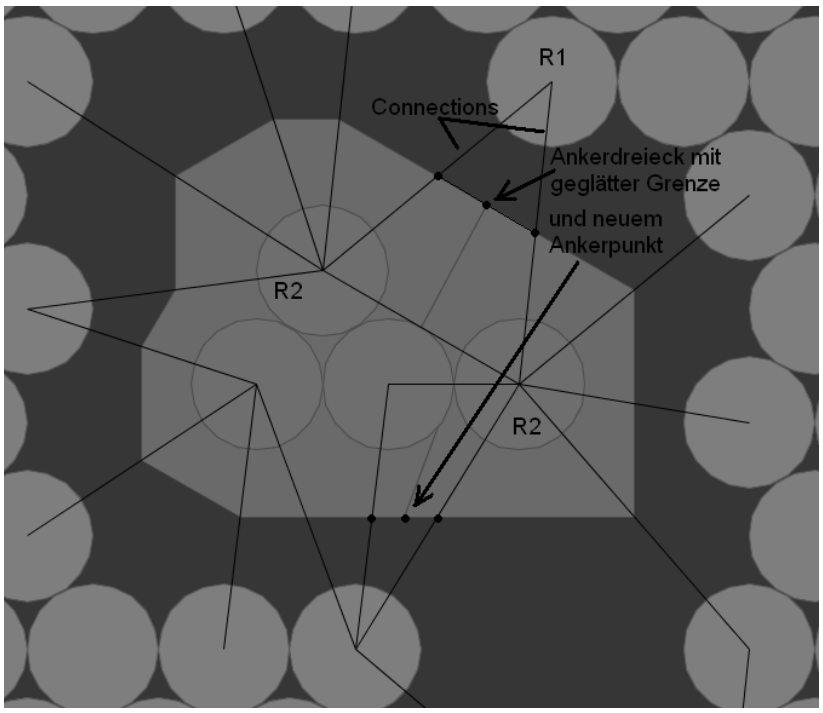


Abbildung 2.18: Ankerdreieck mit korrigiertem Anker

Wenn alle drei Repräsentanten von unterschiedlichen Objektarten stammen, werden drei Regionen miteinander verknüpft. Um eine gleichmäßige Grenzziehung zu erhalten, wird der herkömmliche Anker benutzt, da dieser zentral im Ankerdreieck sitzt.

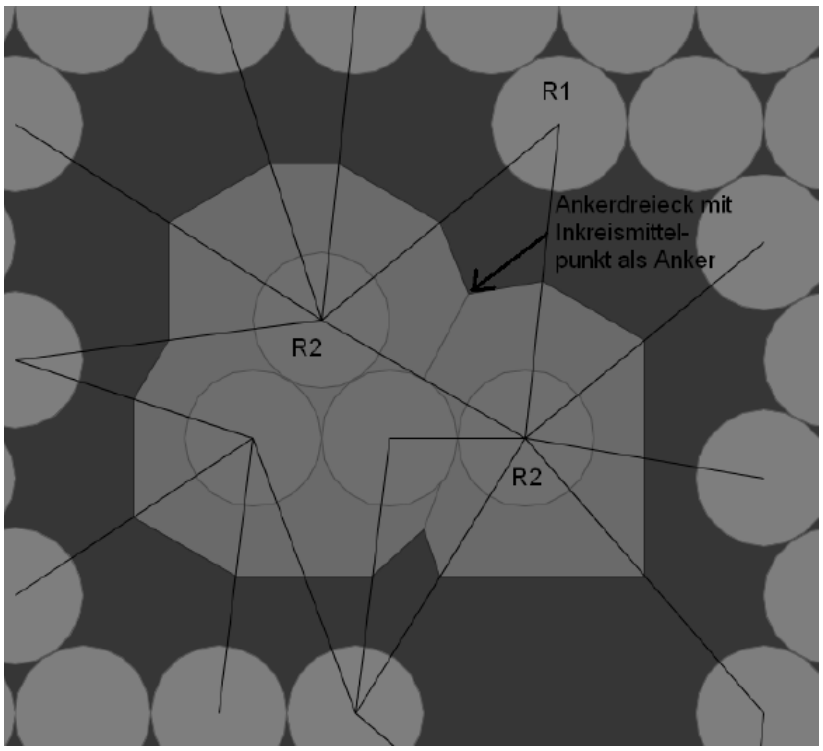


Abbildung 2.19: Ankerdreieck mit drei Repräsentanten unterschiedlicher Objektart und herkömmlichen Anker

Zwei Schnittpunkte

Hat das Ankerdreieck zwei Schnittpunkte (siehe Abbildung 2.19) mit einer ATKIS-Grenze, wird wieder davon ausgegangen, dass eine Grenze, sehr stark vergrößert, eine Gerade ist. Solange der schneidende Teil der ATKIS-Grenze eine Gerade ist, stimmt das auch. Das macht diese Berechnung recht einfach, indem der Mittelpunkt einer Geraden zwischen den beiden Schnittpunkten mit der ATKIS-Grenze konstruiert wird. Der Mittelpunkt ist der neue Anker.

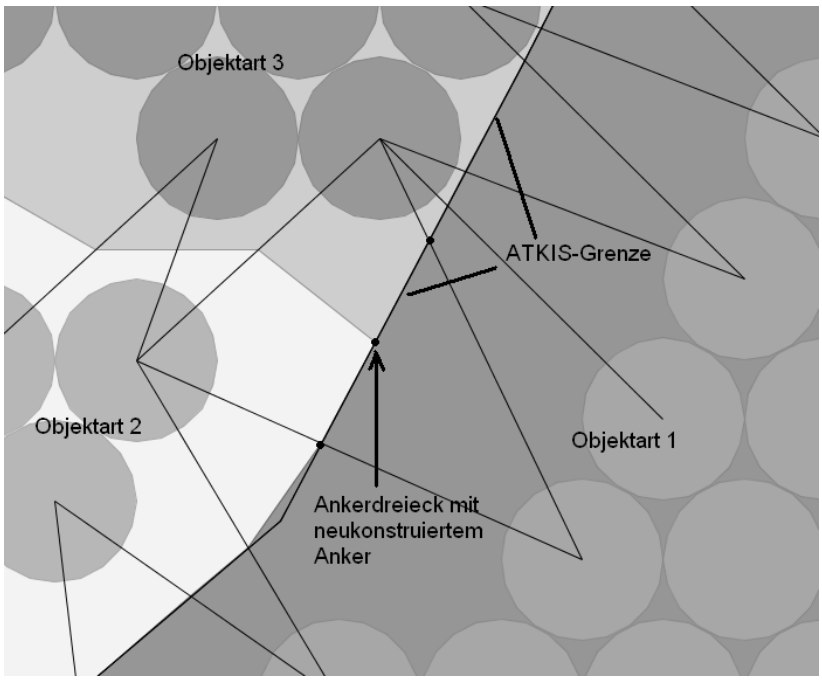


Abbildung 2.20: Ankerdreieck mit neuem an ATKIS-Grenze orientiertem Anker

Das Reduzieren einer Grenze auf eine Gerade, birgt in einigen Fällen Risiken. Denn in der Realität gibt es für jede Regel eine Ausnahme. So besteht dieser schneidende Teil dann aus mehreren Teilstücken, welche nur im günstigsten Fall einer Geraden entsprechen. Für diese Art von Ausnahmen wurde bis jetzt keine Lösung erarbeitet, weil Aufwand und Nutzen in keinem rentablen Verhältnis stehen.

Drei Schnittpunkte

Wenn das Ankerdreieck drei Schnittpunkte (siehe Abbildung 2.20) mit der ATKIS-Grenze aufweist, wird davon ausgegangen, dass drei Teilstücke in einem Punkt KP zusammen laufen. Entspricht die Anordnung der ATKIS-Teilstücke bis zu einem gewissen Grad den einer T-Kreuzung wird der Anker in den KP verlegt.

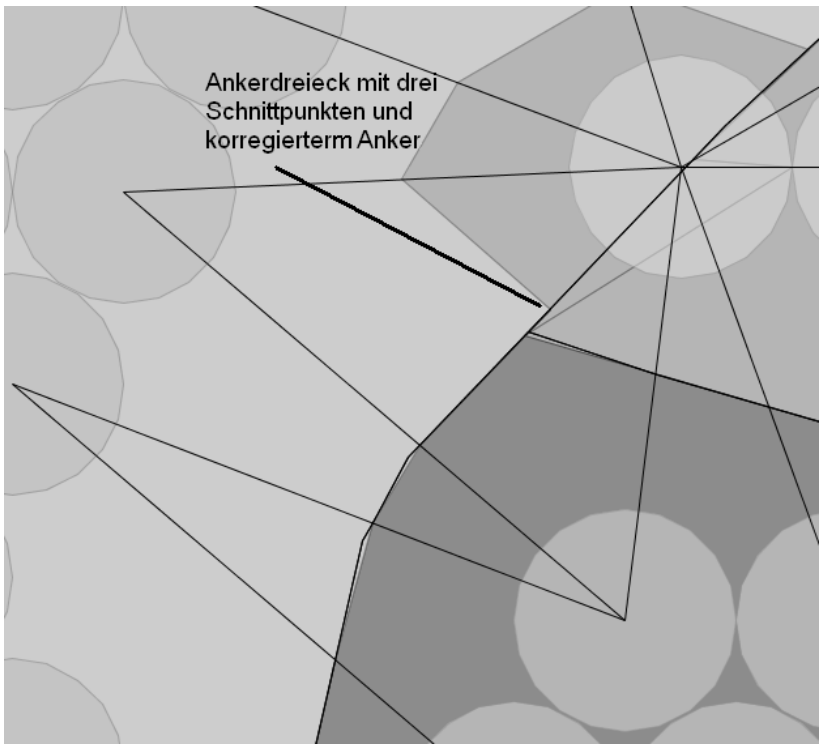


Abbildung 2.21: Ankerdreieck mit drei Schnittpunkten und neuem an ATKIS-Grenze orientiertem Anker

Ist die Anordnung der Teilstücke so, dass alle drei den gleichen Winkel zueinander haben, wird der ursprüngliche Anker verwendet.

Vier Schnittpunkte

Bei vier Schnittpunkten (siehe Abbildung 2.21)mit der ATKIS-Grenze schneiden zwei Teilgrenzen das Ankerdreieck. Da es nicht möglich ist, den Anker einer Grenze zuzuschreiben und somit diese zu bevorzugen, wird ein gemeinsamer Punkt der Teilgrenzen gesucht.

Um diesen Punkt zu berechnen, wird angenommen, dass die beiden Teilgrenzen Geraden sind. Ist der Winkel der beiden Teilgrenzen zueinander ungleich Null, schneiden sich die Teilgrenzen. Das Ziel ist, diesen Schnittpunkt zu finden.

Dazu werden zwei Hilfsgeraden mit den zusammengehörenden Schnittpunkten erschaffen. Der Schnittpunkt dieser beiden Geraden ist der neue Anker.

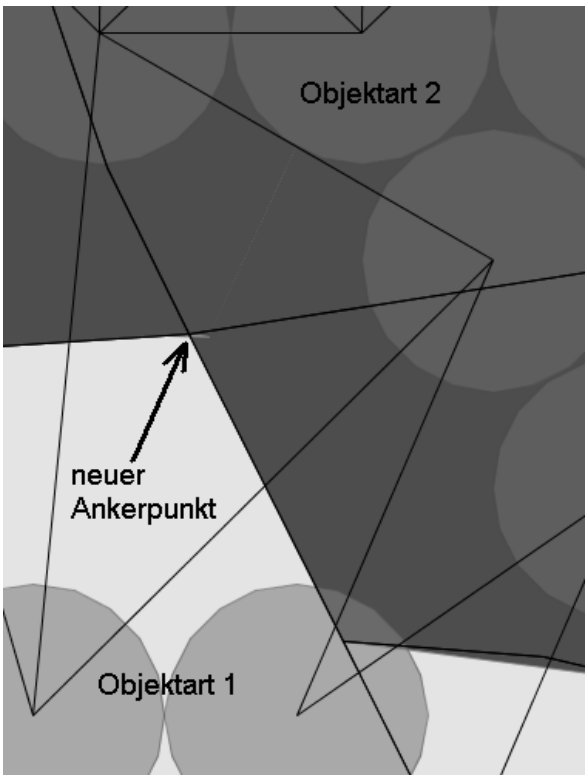


Abbildung 2.22: Ankerdreieck mit vier Schnittpunkten und neuem an ATKIS-Grenze orientiertem Anker

Natürlich fällt diese Idee in sich zusammen, wenn die Teilgrenzen keine Geraden sind oder der Winkel zueinander sehr klein ist. Unter letzter Bedingung würde ein Punkt in weiter Entfernung konstruiert. Diese beiden Probleme werden abgedeckt, in dem man diesem Konstrukt gewisse Toleranzen erlaubt z.B. den Ankerpunkt auch außerhalb des Ankerdreieckes zu setzen.

Dazu wird für den Fall, dass die Teilgrenzen keine Geraden sind, der Abstand von Mittelpunkt der konstruierten Gerade zur Teilgrenze ermittelt. Dieser darf nicht größer sein als zwei Meter.

Für die Lage zueinander, wird geprüft wie der Winkel der Geraden zur X-Achse ist. Die Differenz der beiden Winkel ergibt einen Winkel $\Delta\ \phi$, der nach folgenden Intervallen ausgewertet wird.

$[30° < \Delta\ \phi < 150°]$ $[210° < \Delta\ \phi < 330°]$

Der Wert des Abstandes und die Differenz der Winkel sind jedoch noch nicht optimal und befinden sich zur Zeit in der Erprobungsphase.

Mehr als vier Schnittpunkte

Für mehr als vier Schnittpunkte wurden bis jetzt keine ausreichenden Testfälle gefunden, um diese untersuchen und auswerten zu können. Bei dem einzigen Testfall der bisher gefunden wurde, versuchte man über Geradenkonstruktionen, die möglichst lange Geraden hervorbringen, eine allgemeingültige Beschreibung zu finden. Diese scheiterte aber. Zufällig verlief die Grenzziehung für dieses Beispiel positiv.

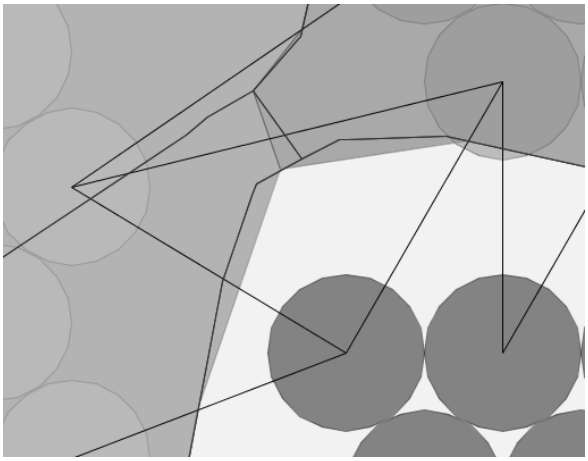


Abbildung 2.23: Einzelfall: Ankerdreieck mit mehr als vier Schnittpunkten

Der Ansatz dieser Bachelorarbeit verfolgt das Prinzip, geometrische Zusammenhänge zwischen ATKIS-Grenzen und dem Grundkonzept von eConstruction (Repräsentanten, Connection, LineStrings) aufzuzeigen und zu bewerten. Dieses Vorgehen hat sowohl Vor- als auch Nachteile. Vorteile sind die schnelle und einfache Berechenbarkeit zwischen den geometrischen Objekten und die korrekte Auswertung von vordefinierten Fällen. Die nachfolgende Abbildung zeigt das selbe Gebiet wie Abbildung 1.1 und 2.7. Gut zu erkennen ist der verändertete Grenzverlauf, sowie das Herausfallen des Rastermusters.

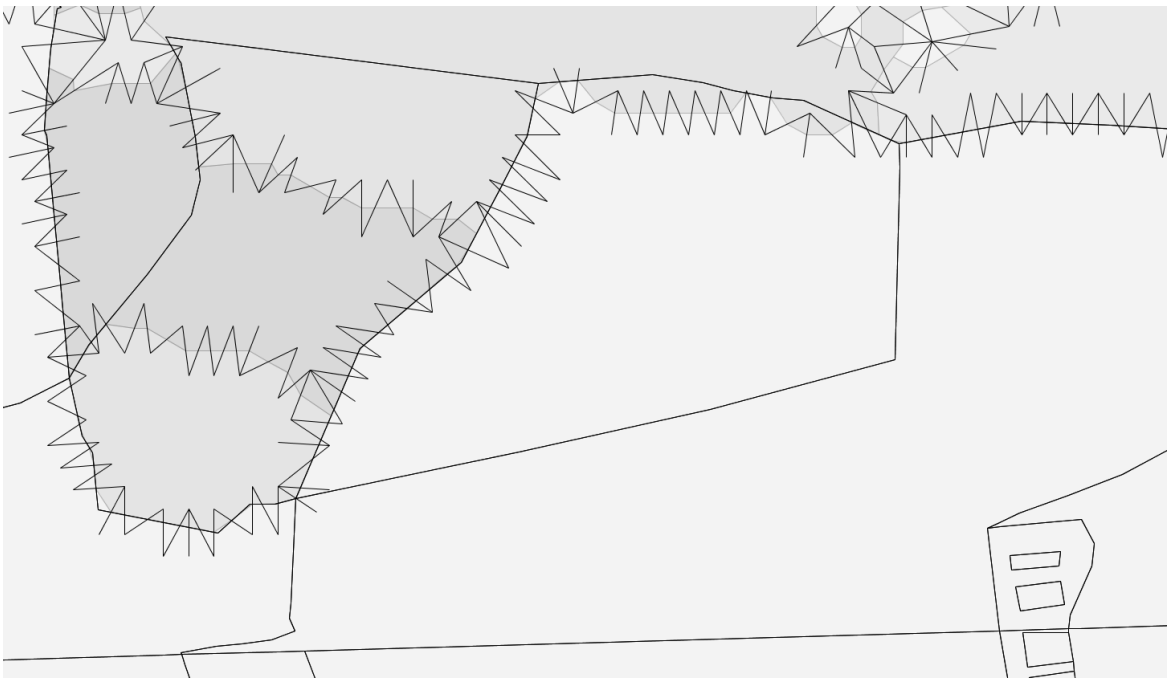


Abbildung 2.24: Grenzziehung mit eConstruction und der Hinzunahme der ATKIS-Daten

Abbildung 2.25: ATKIS-Daten vom selben Gebiet wie Abbildung 2.23

Abbildung 2.26: Satellitenbild des Testgebietes

Zu den Nachteilen zählen die, auf die vordefinierten Fälle begrenzten Handlungsmöglichkeiten und die Akzeptanz von Ungenauigkeiten, durch das vorgegebene Repräsentanten-Prinzip. Nachdem die konzeptionellen und technischen Grundlagen im vorangegangenen Kapitel erläutert wurden, folgt nun der praktische Teil der Arbeit. Dieser gliedert sich in die Architektur der zwei erdachten Komponenten und deren Umsetzung im Kapitel Implementierung.

3 Architektur

In diesem Kapitel wird beschrieben, wie eConstruction aufgebaut [4] ist und zu welchem Zeitpunkt die einzelnen Klassen in Aktion treten. Dabei wird auf die zwei Klassen AtkisBPDetection.java und AnchorCornerPointCreator.java eingegangen. Die Klasse Line.java und die Methode EConIO.loadGeometries() dienen als Hilfskonstrukte.

Die Grenzziehung wird von der Hauptklasse BoundaryConstruction.java gesteuert. Diese stellt im Laufe ihres Lebenszyklus die folgenden wichtigen Daten mit jeweiligem Datentyp für die Grenzziehung bereit. Die Aufzählung erfolgt nach der Zeit des Entstehens.

Die *basicLayerList* ist eine Liste von Layer. Ein Layer ist ein Interface, dass den Zugriff auf die verschiedenen Kanäle des Satellitenbildes steuert. Die Klasse, die das Interface implementiert, muss die Transformation der Koordinaten des Satellitenbildes in die Lokalen vornehmen.

Die *objClassList* ist eine Liste von Objektklassen. Diese besteht aus Listen von Listen, die Strings enthalten. Die Klasse ObjectClass ist ein Interface für Objektklassen und hilft bei deren geografische Klassifizierung. In einer Objektklasse sind Regelobjekte, die thematische Bezeichnung und der Radius der Repräsentanten enthalten.

Der Datentyp der *groupMap* ist eine Map, die auf dem Key-Value-Prinzip beruht. Der Key dieser Map ist der Name der Objektklasse. Das Value ist eine Collection von RPGroups. Eine RPGroup beschreibt eine Anzahl von Repräsentanten, die eine geschlossene Fläche bilden.

Die boundConnUnits ist eine Klasse, die einzelne Units steuert. Eine Unit besteht aus den Connections zwischen zwei Repräsentantengruppen, dessen Anfang und Ende Ankerdreiecke sind, also TriangleStrips. Strukturell setzt sich eine Unit aus einer Liste wie folgt zusammen: den Connections, den Ankerpunkten und einem

Indikator der angibt, ob dieser TriangleStrip zyklisch ist. Im Fall eines zyklischen Strips ist es einer (Anfang und Ende sind gleich) beim Nichtzyklischen sind es zwei.

Die vier Datentypen sind Übergabeparameter der Methode createBoundaries(). In dieser Methode werden zuerst die Grenzen (boundaries) konstruiert, danach die Ankerpunkte berechnet.

Die für die Grenzziehung zuständige Methode ruft den Klassenmember createBoundaries() der Klasse BoundaryFaktory auf . Diese Methode wird im Arbeitsablauf immer wieder überladen. Dabei wird der komplexe Zusammenhang zwischen Grenzen, Teilgrenzen - die zyklisch oder azyklisch sein können - und Ankerpunkten auf die Konstellationen heruntergebrochen, dass zwei Repräsentanten durch eine Connection verbunden sind, auf der sich der Grenzpunkt befinden muss.

Für jede Connection wird die Methode createBoundaryPoints() der Klasse BoundaryFactory.java aufgerufen. Sie besitzt unter anderem die Connection als Übergabeparameter. Die Grenzfindungsmethoden für ATKIS-Datensätze und die vorhandene von eConstruction werden ausgeführt. Liefert die AtkisBorderDetection einen gültigen Punkt hat dieser Vorrang.

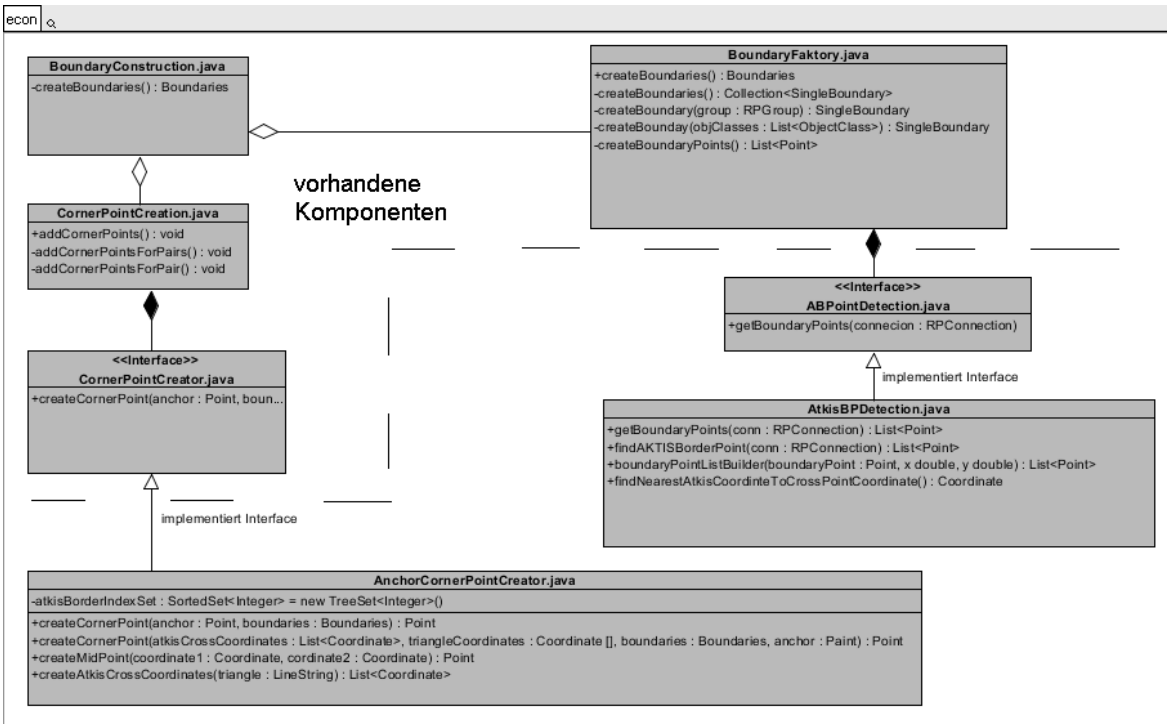


Abbildung 3.1: UML Klassendiagramm [6] der Beziehungen zwischen Klassen den verwendeten Klaasen und Methoden

3.1 Die Klasse AtkisBPDetection.java

Der Aufbau der Klasse AtkisBPDetetction.java ist sehr prozedural, da die Zeit für ein umfangreiches Refactoring nicht ausreichend war. Die Klasse enthält die vier Methoden:

- getBoundaryPoints() (implementiert das vorgegebene Interface)
- findATKISBorderPoint() (setzt die Hauptarbeit um)
- findNearestAtkisCoordinateToCrossPointCoordinate()
- boundaryPointListBuilder()

Die genaue Funktionsweise wird im Kapitel Implementierung erläutert.

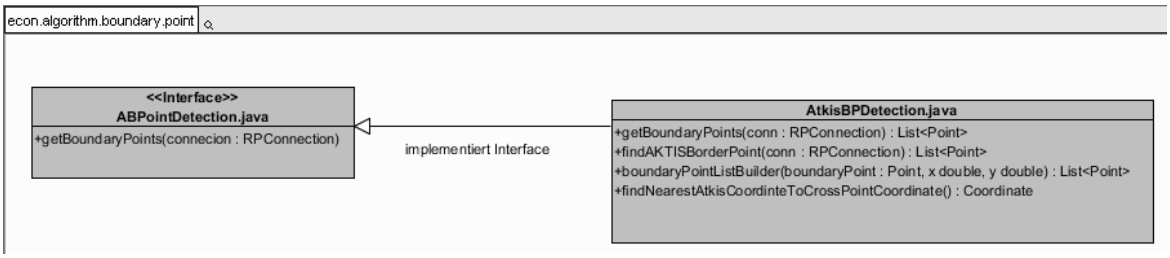


Abbildung 3.2: UML Klassendiagramm [6] der Klasse AtkisBPDetection.java

Als Rückgabewert besitzt die getBoundaryPoints() eine Liste von Punkten, obwohl sie nur einen Grenzpunkt berechnet. Dies ist der Eingliederung in eConstruction geschuldet.

Die Methode findATKISBorderPoint(), die den Hauptteil dieses Prozesses ausmacht, kann als großer Entscheider angesehen werden und besteht hauptsächlich aus vielen If-Anweisungen. Sie stellt durch die Instanziierung des Objektes Line, Methoden zur Verfügung, die zur Berechnung von Geraden und deren Schnittpunkt dienen und das Erzeugen eines Polygons ermöglichen. Die Funktionalität der JTS-Bibliothek ergänzt die erwähnten Methoden.

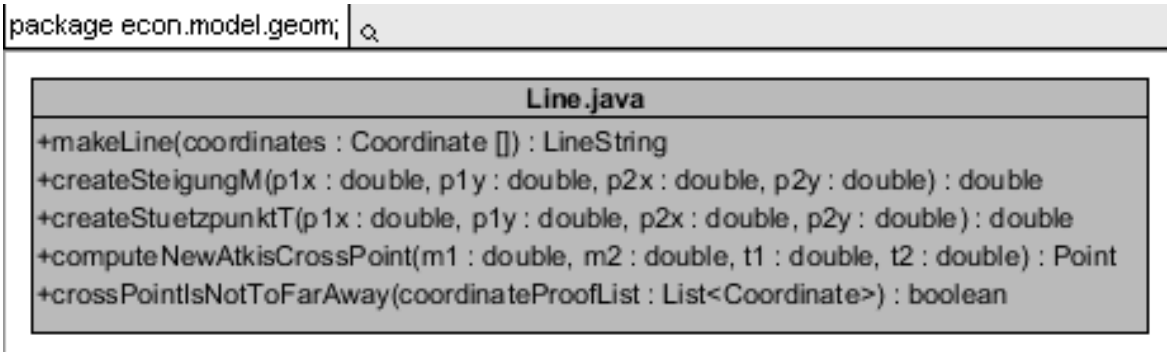


Abbildung 3.3: UML Klassendiagramm der Klasse Line.java

Die Ankerpunkterzeugung hat als Grundlage die Ankerdreiecke, die während der Grenzziehung beim Entstehen der Connections erzeugt wurden. Im Zuge dessen wurde auch der Inkreismittelpunkt für jedes Ankerdreieck berechnet und als Anker deklariert.

Da die Ankerpunktberechnung nach der Grenzziehung stattfindet, stehen die Daten zum Ankerdreieck und den daran beteiligten Repräsentanten, Objektklassen und Connections in einem komplexen Datentyp zur Verfügung.

Ähnlich der Grenzziehung findet auch in diesem Algorithmus das Prinzip der Teilung eines großen Problems in mehrere kleine, Anwendung. Zuerst werden alle vorhandenen Anker betrachtet. Anschließend werden alle einem Repräsentantengruppen-Paar angehörende Anker (aneinander liegende Repräsentantengruppen (RPGroup)) ermittelt. Abschließend werden die Anker, die einer Teilgrenze zu zuschreiben sind, herausgefiltert. Die Teilgrenzen können zyklisch oder azyklisch sein. Die Ankerpunktberechnung erzeugt einen Punkt, der der Liste von Grenzpunkten hinzugefügt wird. Die Klasse AnchorCornerPointCreator implementiert das Interface CornerPointCreator.java, dass später eine gewisse Flexibilität beim Hinzufügen anderer Konzepte erlaubt.

3.2 Die Klasse AnchorCornerPointCreator.java

Die Architektur der Klasse AnchorCornerPointCreator.java ähnelt dem der AtkisBPDetection, da sich die grundsätzliche Arbeit gleicht. Denn auch diese Klasse entscheidet über die im theoretischen Ansatz erläuterten Fälle. Darum besteht auch sie aus vielen If-Anweisungen. Die Klasse besitzt die Methode createCornerPoint(), die im

Lebenszyklus einmal überladen wird, weil das vorgegebene Interface CornerPoint-Creator.java implementiert werden muss. Der Rückgabewert dieser Methode ist ein Punkt.

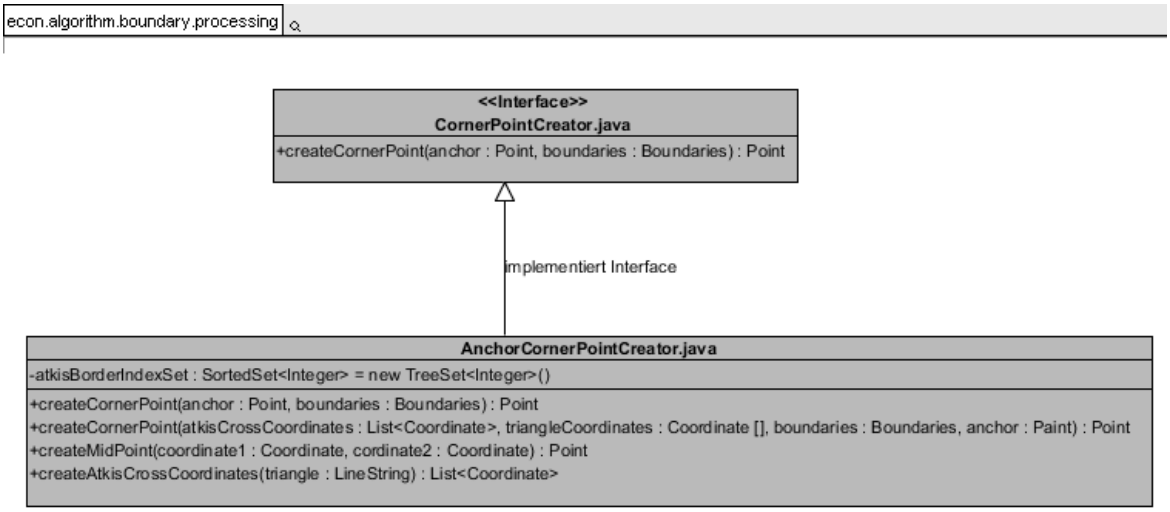


Abbildung 3.4: UML Klassendiagramm [6] der Klasse AnchorCornerPointCreator.java

Die Klasse AnchorCornerPointCreator.java besitzt außerdem die Methoden createMidPoint() und createAtkisCrossCoordinates(), die der Übersichtlichkeit dienen. Die eigentliche Funktionalität wird durch die Funktionen der JTS-Bibliothek und des Objekts der Klasse Line.java erzeugt.

4 Implementierung

Im vorangegangenen Kapitel wurde darauf eingegangen in welchem Kontext die Klassen benutzt werden. In diesem Kapitel wird die Imlementierung der drei erstellten Klassen beschrieben.

4.1 Line.java

Die Klasse bietet vier Funktionalitäten, die JTS nicht implementiert. Zudem fasst die Methode makeLine() die Arbeitsschritte, die zur Erstellung des geometrischen Objekts Gerade (unter JTS LineString) notwendig sind, für eConstruction zusammen. Dazu werden zwei Koordinaten in die GeometryFaktory gegeben.

```
1 public LineString makeLine(Coordinate[] coordinates){
2     LineString line = new GeometryFactory().createLineString(coordinates
          );
3     return line;
4   }
```

Da dieser LineString nur aus zwei Punkten besteht, resultiert daraus eine Strecke.

Den Schnittpunkt zweier Geraden kann man durch lineare Funktion berechnen. Dazu wird die Steigung und der Stützvektor beider Geraden benötigt. Die Gleichung lautet:

$$S\left(\frac{t2-t1}{m1-m2'}\frac{(m1*t2)-(m2*t1)}{m1-m2}\right)$$

Diese beiden Größen lassen sich über die X- und Y-Koordinaten der Punkte, die eine Geraden definieren, wie folgt berechnen:

$$m=\frac{p2y-p1y}{p2x-p1x}$$

$$t = p1y - \frac{p2y - p1y}{p2x - p1x} * p1x$$

Die drei Gleichungen wurden in folgende Methoden umgesetzt.

```
1 public double createSteigungM(double p1x, double p1y, double p2x, double
      p2y){
2     return((p2y -p1y)/(p2x-p1x));
3   }
```

```
1 public double createStuetzpunktT(double p1x, double p1y, double p2x,
     double p2y){
2     return(p1y -(((p2y - p1y)/(p2x - p1x))* p1x));
3   }
```

```
1 public Point computeNewAtkisCrossPoint(double m1, double m2, double t1,
     double t2){
2     return new Point((t2 - t1)/(m1 - m2),(((m1 * t2)-(m2 * t1))/(m1 - m
          2)));
3   }
```

Die vierte Methode hat den Rückgabewert Boolean und prüft zwei Geraden auf ihre Lage zueinander. Dazu wird per JTS-Methode der Winkel zur X-Achse berechnet. Der Winkel wird im Bogenmaß ausgegeben. Der Betrag der Differenz muss in folgenden Intervallen liegen:

$[30° < \Delta\, \phi < 150°]$

$[210° < \Delta\, \phi < 330°]$

```
1 public boolean crossPointIsNotToFarAway(List<Coordinate>
     coordinateProofList){
2     double deltaPhi = Math.abs(Angle.angle(coordinateProofList.get(0),
          coordinateProofList.get(1)) - Angle.angle(coordinateProofList.
          get(2), coordinateProofList.get(3)));
3     if(deltaPhi > 0.52 && deltaPhi < 2.62 && deltaPhi > 3.67 && deltaPhi
           < 5.76){
4       return true;
5     }else{
6       return false;
7     }
8   }
```

4.2 AtkisBPDetection.java

Wie schon beschrieben, entscheidet diese Klasse hauptsächlich, wie die Auswertung des Grenzpunktes zu erfolgen hat. Sie muss die Methode getBoundayPoints() implementieren und bekommt eine einzelne Connection übergeben. Um die ATKIS-Daten nicht bei jedem Aufruf der Methode in den Speicher laden zu müssen, wurde das beim Start der BoundaryConstruction.java durchgeführt.

```
1  public class BoundaryConstruction {
2  \vdots
3  private static List<Geometry> atkisBorder;
4
5    public static List<Geometry> getAtkisBorder() {
6      return atkisBorder;
7    }
8  \vdots
9   try {
10            BoundaryConstruction.atkisBorder = EConIO.loadGeometries(
                  properties.getAktisPath());
11      } catch (FileNotFoundException e) {
12        LOG.error("could not load AtkisDataSet : " + properties.
              getAktisPath() + ": " + e);
13      }
14 \vdots
15 }
```

Diese wird per Klassenmember an die Methode findAKTISBorderPoint() weitergeben. In einer Schleife durchlaufend wird jede Geometrie aus der Liste auf Schnittmengen mit der Connection geprüft und in eine Map gespeichert, um die Zugehörigkeit zwischen Schnittpunkt und Geometrie nicht zu verlieren. Für die Schnittmengenprüfung wird die JTS-Methode intersection() benutzt, die die Anzahl und die Koordinaten der Schnittpunkte ausgibt. Der Vorteil ist, dass komplette Polygone an die Methode übergeben werden können.

```
1  for (int j = 0; j < atkisBorderList.size(); j++) {
2        crossPointCoordinates = connGeometry.intersection(atkisBorderList.
             get(j)).getCoordinates();
3        if(crossPointCoordinates.length!=0){
4          for(int a = 0; a < crossPointCoordinates.length; a++){
5            crossPointCoordinatesMap.put(crossPointCoordinates[a],
                 atkisBorderList.get(j));
```

```
6         }
7       }
8     }
9     Set<Coordinate> keys = crossPointCoordinatesMap.keySet();
```

Per If-Anweisung werden die unterschiedlichen Auswertungsschritte festgelegt. Wenn die Größe des Sets gleich eins ist, wird der Schnittpunkt noch auf den Abstand zu den beteiligten Repräsentanten geprüft, sodass dieser nicht zu weit vom Originalschnittpunkt abweicht. Ist dies der Fall, übernimmt der boundaryPointListBuilder den Grenzpunkt, erzeugt eine Liste, fügt der Liste den Punkt hinzu und gibt die Liste der aufrufenden Methode zurück.

```
1 if (keys.size() == 1) {
2       for(Coordinate crossCoordinate : keys) {
3         if (crossCoordinate.distance(connCoordinates[0]) > (0.501 * conn
              .getRP0().getRadius()) ||                  crossCoordinate.
              distance(connCoordinates[1]) > (0.501 * conn.getRP1().
              getRadius())){
4           return boundaryPointListBuilder(boundaryPoint, crossCoordinate
                .x, crossCoordinate.y);
5         }
6       }
7     }
```

Für den Fall, dass der *key* zwei Schnittpunkte enthält, soll nun ein Punkt konstruiert werden, der dem Schnittpunkt der beiden LineStrings des ATKIS-Datensatzes entspricht. Dafür wird die Map benötigt. Es wird der am nächsten liegende Punkt des LineString zu seinem zugehörigen Schnittpunkt gesucht und aus beiden Punkten eine Gerade berechnet.

```
1 public Coordinate findNearestAtkisCoordinteToCrossPointCoordinate(
      Coordinate crossPointCoordinate, Coordinate[]
      atkisBorderPointCoordinates){
2     SortedMap<Double, List<Coordinate>> nearAtkisBorderCoordinatesMap =
3     new TreeMap<Double, List<Coordinate>>();
4     double distance = 0.;
5     for (int i = 0; i < atkisBorderPointCoordinates.length; i++ ){
6       distance = crossPointCoordinate.distance(
            atkisBorderPointCoordinates[i]);
7       nearAtkisBorderCoordinatesMap.put(distance,Arrays.asList(
            atkisBorderPointCoordinates[i],
8       crossPointCoordinate));
```

```
9     }
10    return nearAtkisBorderCoordinatesMap.get(
          nearAtkisBorderCoordinatesMap.firstKey()).get(0);
11  }
```

Der Arbeitsschritt wiederholt sich für den anderen Schnittpunkt. Dazu nutzt man die Funktionalitäten der Klasse Line.java.

4.3 AnchorCornerPointCreator.java

Für die Ankerpunktbewertung muss zuerst das passende Ankerdreieck gefunden werden. Per Schleife wird geprüft, in welchem Ankerdreieck der Ankerpunkt liegt. Über die ausgelesenen Punkte des passenden Dreieckes erhält man die Koordinaten, die dazu benutzt werden, ein geometrisches Objekt (LineString) in der Form eines Dreieckes zu erzeugen.

```
1  public Point createCornerPoint(Point anchor, Boundaries boundaries) {
2
3       Collection<Triangle> allAnchorTriangles = RPConnCreation.
            allAnchorTriangles;
4       Coordinate [] triangleCoordinates = new Coordinate[4];
5       for(Triangle anchorTriangle : allAnchorTriangles){
6         if(anchorTriangle.isPointInside(anchor)){
7           for(int i = 0; i <= 2 ; i++){
8             triangleCoordinates[i] = anchorTriangle.getPoint(i).
                  getCoordinate();
9           }
10          triangleCoordinates[3] = anchorTriangle.getPoint(0).
                getCoordinate();
11          LineString triangle = new GeometryFactory().createLinearRing(
                triangleCoordinates);
12
13          List<Coordinate> atkisCrossCoordinates =
                createAtkisCrossCoordinates(triangle);
14              return createCornerPoint(atkisCrossCoordinates,
                    triangleCoordinates, boundaries, anchor);
15        }
16      }
17        return anchor;
18    }
```

Die Methode prüft den erschaffenen LineString mit den ATKIS-Daten auf Schnittmengen. Sind Schnittpunkte vorhanden, werden die dazugehörenden LineString der ATKIS-Grenzen in einem Set gespeichert. Dafür wird die Methode intersection() benutzt.

```
public List<Coordinate> createAtkisCrossCoordinates(LineString triangle)
    {

      List<Coordinate> atkisCrossCoordinates = new ArrayList<Coordinate
          >();
      Set<Coordinate> tmpSet = new HashSet<Coordinate>();
      Coordinate[] tmpCrossCoordinates;
      List<Geometry> atkisBorder = BoundaryConstruction.getAtkisBorder()
          ;
      int atkisBorderSize = atkisBorder.size();

      for (int j = 0; j < atkisBorderSize; j++) {
          tmpCrossCoordinates = triangle.intersection(atkisBorder.get(j)
              ).getCoordinates();
          for (int i = 0; i < tmpCrossCoordinates.length; i++){
            tmpSet.add(tmpCrossCoordinates[i]);
            atkisBorderIndexSet.add(j);
          }
         }
      atkisCrossCoordinates.addAll(tmpSet);
      return atkisCrossCoordinates;
    }
```

Nachdem die Anzahl und die Koordinaten der Schnittpunkte feststehen, müssen diese ausgewertet werden. Ist kein Schnittpunkt vorhanden, wird geprüft ob zwei der drei Punkte des LineStrings einer Objektklasse angehören. Dies wird über ein Array realisiert, welches Listen enthält. In einer Liste stehen alle Repräsentantenkoordinaten, die einer Objektklasse zuzuordnen sind. Entsteht beim Vergleich eine Konstellation, in der eine Liste ein Element, die andere zwei Elemente enthält, ist der obengenannte Fall erfüllt.

```
\vdots
List<Coordinate> [] rpCoordinateListArray = EConIO.
    getRPCoordinateListArray();
List<Coordinate> [] tmpArray = new List[EConIO.getRPCoordinateListArray
    ().length];
\vdots
```

```
for (int j = 0; j < rpCoordinateListArray.length; j++){
            for (int i = 0; i < triangleCoordinates.length-1; i++){
              if (rpCoordinateListArray[j].contains(triangleCoordinates[
                  i])){
                tmpArray[x].add(triangleCoordinates[i]);
              }
                }
            x++;
          }
//          fill lineCoordinatesArray  to find the points of a triangle
    which are in the same class
          for (int l = 0; l < tmpArray.length; l++){
            if (tmpArray[l].size() == 1){
              lineCoordinatesArray[0].addAll(tmpArray[l]);
            }
            if (tmpArray[l].size() == 2){
              lineCoordinatesArray[1].addAll(tmpArray[l]);
            }
            if (tmpArray[l].size() == 3){
              return anchor;
            }
          }
//          anchorPoint between two classes, anchorTriangle is with one
    Point in the first objectClass and two in the second
          if ((lineCoordinatesArray[0].size() == 1) && (
              lineCoordinatesArray[1].size() == 2)){
          \vdots
```

Der Mittelpunkt aus der Geraden, die die Connections verbindet, die zwischen den Repräsentanten der unterschiedlichen Objektklassen verläuft, wird als Anker genommen.

Ergibt die Auswertung der Methode intersection() zwei Schnittpunkte, wird der Mittelpunkt zwischen den Schnittpunkten berechnet.

Bei drei Schnittpunkten muss ersichtlich sein, ob zwei der drei Schnittpunkte eine Gerade bilden. Dazu wird eine N mal M Matrix verwendet. Diese setzt sich aus zwei Listen zusammen. Durch eine Vorbedingung werden keine Dopplungen zugelassen.

```
for (int g = 0; g < atkisCrossCoordinates.size()-1; g++){
            for (int h = 1; h < atkisCrossCoordinates.size(); h++){
```

```
3               go:
4               if((g != h) && (g != h + 1) && (g != h + 2)){
```

Liegt die Entfernung des Mittelpunktes der konstruierten Gerade (G1) von der ATKIS-Grenze unter zwei Meter, wird das als geradenförmig angesehen. Damit wird der Mittelpunkt von G1 als neuer Ankerpunkt gesetzt.

```
1 \vdots
2 if (line.getCentroid().distance(BoundaryConstruction.getAtkisBorder().
      get(k))< .01) {
3 \vdots
4   }
5   \vdots
```

Wenn die Schnittmenge vier ergibt, wird ähnlich dem vorherigen Arbeitschritt gearbeitet. Es werden statt einer Geraden zwei gesucht. Die Matrix wird um ein Element erweitert.

Nun kann es sein, dass die zwei Geraden sehr dicht beieinander liegen, sodass ein LineString im Distanzbereich der Geraden, die zum anderen LineString gehört, liegt. Daher wird eine Map geschaffen, in der nach dem kleinsten Abstand sortiert werden kann. So wird der passende LineString des ATKIS-Datensatzes der richtigen Geraden zugeordnet.

```
1 for (int g = 0; g < atkisCrossCoordinates.size()-1; g++){
2                 for (int h = 1; h < atkisCrossCoordinates.size(); h++){
3                   go:
4                   if((g != h) && (g != h + 1) && (g != h + 2)){
5                     coordinates[0]=atkisCrossCoordinates.get(g);
6                     coordinates[1]=atkisCrossCoordinates.get(h);
7                     LineString line = tmpLine.makeLine(coordinates);
8                     for (int k  : atkisBorderIndexSet) {
9                       if (line.getCentroid().distance(
                            BoundaryConstruction.getAtkisBorder().get(k))<
                             .3) {
10                        distance = line.getCentroid().distance(

                                                BoundaryConstruction.
                              getAtkisBorder().get(k));
11                        nearestAtkisBorderMap.put(distance,Arrays.asList
                              (atkisCrossCoordinates.get(g),
                                                atkisCrossCoordinates.get(h)));
```

```
12                          break go;
13                        }
14                      }
15                    }
16                  }
17                }
18              keys = nearestAtkisBorderMap.keySet();
19              for (double k : keys){
20                m.add(tmpLine.createSteigungM(nearestAtkisBorderMap.get(k)
                      .get(0).x,
                      nearestAtkisBorderMap.get(k).get(0).y,
                      nearestAtkisBorderMap.get(k).get(1).x,
                                  nearestAtkisBorderMap.get(k).get(1).y));
21                t.add(tmpLine.createStuetzpunktT(nearestAtkisBorderMap.get
                      (k).get(0).x,
                      nearestAtkisBorderMap.get(k).get(0).y,
                      nearestAtkisBorderMap.get(k).get(1).x,
                                  nearestAtkisBorderMap.get(k).get(1).y));
22                if(m.size()==2 && t.size()==2){
23                      if (tmpLine.crossPointIsNotToFarAway(
                           coordinateProofList)){
24                          return tmpLine.computeNewAtkisCrossPoint(m.get
                                (0), m.get(1), t.get(0), t.get(1));
25                      }else{
26                          return anchor;
27                      }
28                }
```

Stimmt der Winkel der Geraden zueinander, kann der Schnittpunkt konstruiert werden.

Auf den Fall, dass es mehr als vier Schnittpunkte gibt, wird verzichtet, da dieser Ansatz keinem wissenschaftlichen Grundsatz folgt und als Experiment angesehen werden kann. Es wird nur die Abarbeitung gezeigt.

```
1 if (atkisCrossCoordinates.size() > 4){
2         for (int i = 0; i < atkisCrossCoordinates.size()-1; i++){
3             for (int j = 1; j < atkisCrossCoordinates.size(); j++){
4               if((i != j) && (i != j + 1) && (i != j + 2) && (i != j +
                      3)){
5                 distance = atkisCrossCoordinates.get(i).distance(
                        atkisCrossCoordinates.get(j));
```

```
6                   segmentCoordintesMap.put(distance,Arrays.asList(
                        atkisCrossCoordinates.get(i),
                            atkisCrossCoordinates.get(j)));
7                 }
8               }
9             }
10          return createMidPoint(segmentCoordintesMap.get(
                segmentCoordintesMap.lastKey()).get(0),
11              segmentCoordintesMap.get(segmentCoordintesMap.lastKey()).get
                    (1));
```

5 Schlussfolgerung

Der Prozess der Klassifikation, der auch den Arbeitsschritt Grenzziehung beinhaltete, gehört zum Tagesgeschäft der Satellitenbildinterpreten der Firma Delphi IMM. Ein Großteil der Zeit wird damit verbracht, die von der Software erstellte Klassifkation zu prüfen und zu berichtigen, wobei der Aufwand von Fall zu Fall sehr unterschiedlich ist. Die Unterteilung des Waldes in seine Unterklassen ist dabei am Arbeitsintensivsten. Die ATKIS-Grenzen, zur Informationsgewinnung, mit einfließen zu lassen, ist nicht neu. Dieser Prozess wird aber manuell durchgeführt. Diese Klassifikationen variieren in der Qualität, da der Einflussfaktor Mensch eine große Rolle spielt. Zum Beispiel: Wieviel Zeit hat der Auswerter, Details zu analysieren? Wie ist seine Konstitution? Hat er etwas übersehen? Ist seine Arbeitsleistung über acht Stunden konstant?

Die automatische Beachtung der ATKIS-Grenzen soll diese Fehlerquelle minimieren. Mit den zwei neuen Komponenten *AtkisBPDetection.java* und *AnchorCornerPointCreator.java*, die in eConstruction eingefügt wurden, ist das größtenteils gelungen.

Funkionsnachweis und Wertung

Dieser Nachweis zeigt, dass eConstruction die Wahrnehmungsvielfalt des Menschen nicht ersetzen kann und dass das Prüfen von automatischen Erzeugnissen nicht entfällt. Zum anderen wird man aber zeigen, dass eConstruction in der Lage ist, wiederkehrende und stupide Arbeit zu ersetzen.

In Abbildung 5.1 ist eine Klassifikation eines Satellitenbildinterpreten zu sehen, die semi-automatisch entstanden ist. Zu sehen sind verschiedene Regionen, ATKIS-Grenzen und das typische Rastermuster, dass von der verwendeten Software, die

das Satellitenbild auswertet, übernommen wurde. Gut zu erkennen sind die Bereiche in denen nachbearbeitet wurde und die Schwankungen in der Orientierung an ATKIS. Während manche Regionen sich dicht an die ATKIS-Grenzen schmiegen, weichen einige erheblich ab. Das entspricht nicht den Anforderungen des Auftraggebers[1]. Diese Klassifikation wurde in einem zweiten Korrekturdurchlauf als Fehler identifiziert.

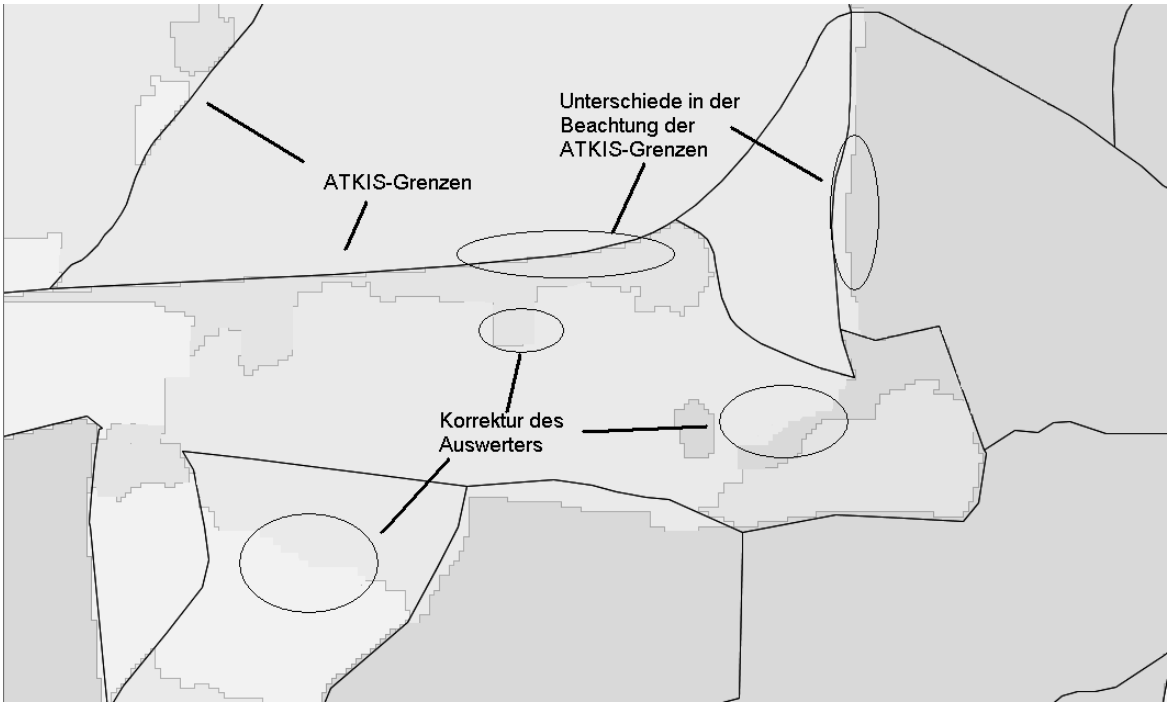


Abbildung 5.1: Klassifikation eines Satellitenbildinterpreten von Gebiet A

Wie in Abbildung 5.2 zu sehen, ist der Grenzverlauf allein durch das Elimieren des Rastermusters gleichmäßiger und leichter für das menschliche Auge zu erfassen. Dies wurde von Experten der Firma Delphi IMM bestätigt. Positiv zu bewerten ist auch, die gute Orientierung zum Verlauf der ATKIS-Grenze, die sich im betrachteten Gebiet durchzieht.

Negativ bewertet wurde, dass einige Gebiete vernachlässigt wurden und somit nicht existent sind. Dieses Problem ist der Mindestgröße zu zuschreiben.

Prinzipiell ist diese Grenzziehung aber völlig korrekt und hat im Bereich der ATKIS-Grenzen klare Vorteile gegenüber der der menschlichen Klassifikation:

[1] siehe 1.3

- automatisiert, menschliche Arbeitskraft wird gespart
- konstante Qualität der Klassifikaion im Bereich der ATKIS-Grenzen
- schnellere Klassifikation

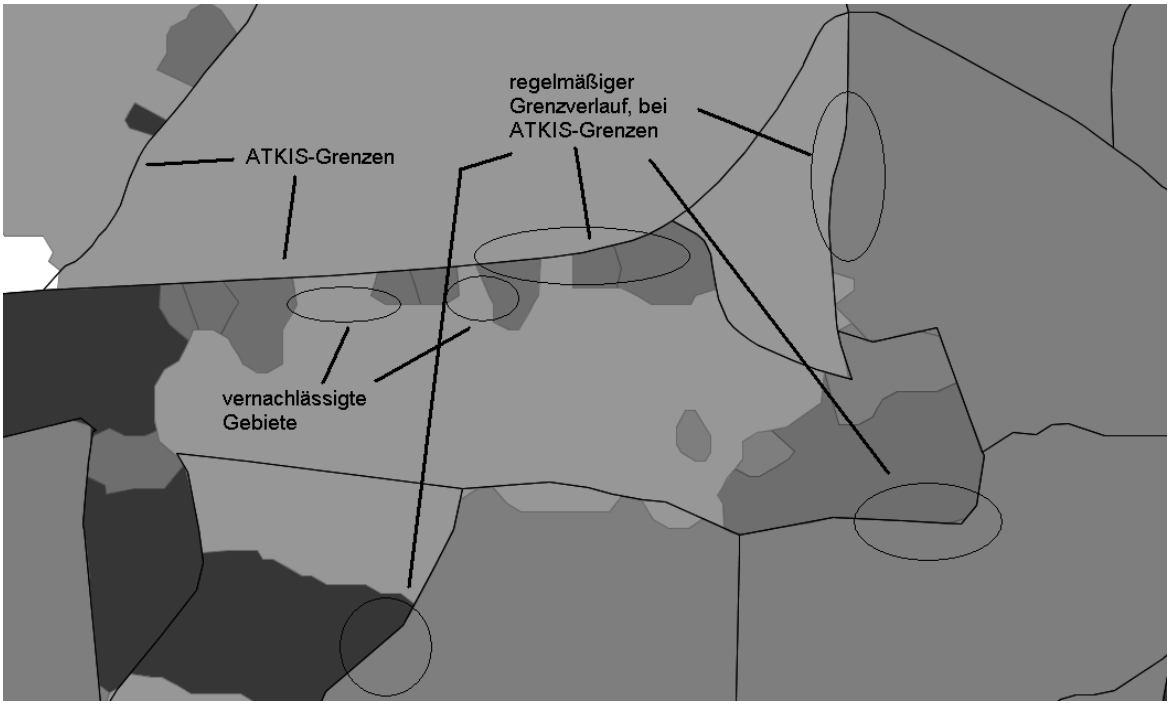


Abbildung 5.2: Klassifikation von eConstruction des Gebietes A

Ausblick

eConstruction fehlt es zur Zeit an Perfomance. Diese zu verbessern, ist ein Punkt für die Zukunft. Ein Anderer ist der, weitere Informationen zum Zeitpunkt der Grenzpunktanalyse zur Verfügung zu haben. Der thematische Layer der ATKIS-Grenzen wäre solch eine Information. Denn nur durch eine große Wissensbasis kann grundsätzlich eine genauere Grenzziehung erfolgen. Als anzustrebendes Beispiel für eine möglichst große Wissensbasis und dessen vielfältige Auswertung steht die des Menschen, auch Erfahrung genannt.

Abbildungsverzeichnis

1.1 Ein Grenzverlauf der ohne eConstruction erstellt wurde. Deutlich sichtbar, das Rastermuster, des zugrunde liegende Satellitenbildes . . 10

2.1 Technische Daten im Überblick 12
2.2 Vorklassifikation eines der Objektklasse Nadelwald, erstellt von einem Satellitenbildinterpreten 13
2.3 Ein ATKIS-Datensatz: bestehend aus LineStrings 15
2.4 Gezeigt wird eine besiedelte Fläche im großen Zoom. 17
2.5 Selbige Fläche, etwas heraus gezoomt 17
2.6 Repräsentanten im typischen Anordnungsmuster 18
2.7 Grenzziehung mit eConstruction, ohne Hinzunahme von ATKIS-Daten 19
2.8 Befüllung einer Objektklasse mit Repräsentanten 20
2.9 Repräsentanten die durch eine Connection verbunden sind 21
2.10 TriangleStrip zwischen zwei Ankerdreiecken 22
2.11 TriangleStrips bilden eine geschlossene Fläche. Eine Region entsteht. 23
2.12 Connection schneidet ein Teilstück der ATKIS-Grenze 25
2.13 TriangleStrip wird von ATKIS-Grenze geschnitten. Resultat: Ein oder Zwei Schnittpunkt/e 26
2.14 Berechnung eines neuen Anker durch Erzeugung zweier Hilfsgeraden. 27
2.15 Entstehung von Einschnitten mit Ankerpunkten, die auf dem Inkreis beruhen. 29
2.16 Ankerdreieck bestehend aus drei Connections 30
2.17 Ankerdreieck bestehend aus drei Connections 31
2.18 Ankerdreieck mit korrigiertem Anker 32
2.19 Ankerdreieck mit drei Repräsentanten unterschiedlicher Objektart und herkömmlichen Anker 33
2.20 Ankerdreieck mit neuem an ATKIS-Grenze orientiertem Anker . . . 34

2.21 Ankerdreieck mit drei Schnittpunkten und neuem an ATKIS-Grenze orientiertem Anker 35
2.22 Ankerdreieck mit vier Schnittpunkten und neuem an ATKIS-Grenze orientiertem Anker 36
2.23 Einzelfall: Ankerdreieck mit mehr als vier Schnittpunkten 37
2.24 Grenzziehung mit eConstruction und der Hinzunahme der ATKIS-Daten 38
2.25 ATKIS-Daten vom selben Gebiet wie Abbildung 2.23 38
2.26 Satellitenbild des Testgebietes 39

3.1 UML Klassendiagramm [6] der Beziehungen zwischen Klassen den verwendeten Klaasen und Methoden 41
3.2 UML Klassendiagramm [6] der Klasse AtkisBPDetection.java 42
3.3 UML Klassendiagramm der Klasse Line.java 43
3.4 UML Klassendiagramm [6] der Klasse AnchorCornerPointCreator.java 44

5.1 Klassifikation eines Satellitenbildinterpreten von Gebiet A 56
5.2 Klassifikation von eConstruction des Gebietes A 57

Tabellenverzeichnis

2.1 Übersicht der Testreihe 28

Literaturverzeichnis

[1] DIETMAR GRÜNREICH, Günther H.: *Kartographie*. 7. Walter de Gruyter, 1994. – ISBN 3–11–013398–9

[2] ESRI: *ESRI Shapefile Technical Description, letzter Aufruf 20.01.2010*. http://www.esri.com/library/whitepapers/pdfs/shapefile.pdf,

[3] FORTUNE, Steven: *Voronoi Diagrams and Delaunay Triangulations. In: Euclidean Geometry and Computers. Hg. von D. A. Du u. F. K. Hwang S.192-230*. 2. World Scientific Publishing, 1992. – ISBN 3–935042–92–2

[4] KINKELDEY, Christoph: *eConstruction Dokumentation*. 2008-2009

[5] PETER SCHRÖDER, Armin H.: *Kartographie in Stichworten*. 7. Gebrüder Borntraeger Verlagsbuchhandlung, 2002. – ISBN 3–443–03112–9

[6] SCHMIDT, Prof. G.: *Architekturdiagramme*. Vorlesungsskript Software Engeneering, Fachhochschule Brandenburg, 2007