Master Thesis

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

zum Thema

„Viewing von Geodaten“

Effizientes Viewing von Geodaten aus einer Oracle 11g Geodatenbank

vorgelegt von

Ing. Stefan Gamperl
U1361, UNIGIS MSc Jahrgang 2008

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

Gutachter:
Ao. Univ. Prof. Dr. Josef Strobl

Rohrbrunn, 20.03.2011

Danksagung

Ich möchte mich bei all denen bedanken, die mich bei dieser Arbeit und während des gesamten UNIGIS Lehrganges unterstützt haben. Dies gilt insbesondere Herrn Prof. Dr. Josef Strobl und dem gesamten UNIGIS Team für deren Engagement und Einsatz, um für jede erdenkliche Anfrage eine Lösung parat zu haben. Danken möchte ich meinem Arbeitgeber rmDATA, welcher mir die Durchführung des UNIGIS Fernlehrganges neben dem Arbeitsalltag ermöglicht hat, und meinen Kollegen bei rmDATA, welche mir fachliche Anregungen gaben und immer für interessante Diskussionen zur Verfügung standen. Nicht zuletzt gilt mein Dank meiner Familie und meinen Freunden, die mich im Laufe des Studiums sehr unterstützt und mir vieles abgenommen haben, damit ich meine Zeit für den UNIGIS Lehrgang investieren konnte.

Erklärung der eigenständigen Abfassung der Arbeit

Ich versichere, diese Master Thesis ohne fremde Hilfe und ohne Verwendung anderer als der angeführten Quellen angefertigt zu haben, und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat. Alle Ausführungen der Arbeit, die wörtlich oder sinngemäß übernommen wurden, sind entsprechend gekennzeichnet.

Rohrbrunn, am 20. März 2011

Ing. Stefan Gamperl

Kurzfassung

Systeme für die Verarbeitung von Geoinformation haben seit den 60er Jahren eine interessante Entwicklung erfahren. Dabei zeichnet sich ein immer stärker werdender Trend hin zu verteilten Softwaresystemen ab. Durch die Nutzung verteilter Systeme stellt sich die Bereitstellung von Geodatenbeständen als große Herausforderung dar. Je nach vorliegendem Client (Thin Client, Thick Client, Smart Client) müssen Geodaten je nach Ausgangsformat und Umfang auf den Client Computer übertragen werden. Steigt die Datenmenge, so wirkt sich dies negativ auf die Selektionsgeschwindigkeit beim Datenlieferanten (Datenbankserver, Dateiserver, Geodateninfrastruktur, …), der Übertragungszeit im Netzwerk und der Darstellungsgeschwindigkeit am Client aus. Da eine effiziente Bereitstellung ein immer wichtiger werdendes Thema scheint, soll mit der Arbeit ein Lösungsansatz beschrieben werden, welcher Geodaten durch eine effiziente Übertragung über das Netzwerk bereitstellt und eine möglichst rasche Darstellung der Geodaten auf einem Client ermöglicht. Verwendet wurde für die Ablage der Geodaten ein Oracle 11g Datenbankserver. Mittels der von Oracle bereitgestellten ODP.NET Anwendungsschnittstelle wird durch einen Testclient eine Abfrage durchgeführt. Die Ergebnisse der Testläufe wurden im Zuge der Arbeit dokumentiert. Durch die intelligente Kombination aus einer räumlichen und maßstabsabhängigen Datenselektion, einer automatischen Reduktion durch ein Entfernungskriterium und einer Vereinfachung von linienhaften Objekten, wird gegenüber dem Ausgangsdatenbestand eine derartige Datenreduktion erzielt, dass der Ergebnisdatenbestand rasch und ohne dem Verbrauch großer Ressourcen an den Client transportiert und angezeigt werden kann. Nachdem die räumliche und thematische Filterung in der Datenbank oftmals zum Einsatz kommt, ist die zusätzliche Datenreduktion aufgrund einer „On-the-fly“ Generalisierung die Kernidee der vorliegenden Arbeit. Durch die Definition des kleinsten am Bildschirm sichtbaren Objektes werden die Bereinigung kleiner Objekte (inkl. Topologieanpassungen) und die Vereinfachung linienhafter Objekte gesteuert. Da der Generalisierungsprozess direkt in der Datenbank durchgeführt wird, kann sowohl an Ressourcen als auch an Abfragezeit eingespart werden.

Abstract

Since the 60's there was an astonishing development in the area of geographic information processing. Actually there is an increasing trend of using distributed software systems to solve everyday problems. Also for GIS systems a distributed architecture becomes more and more common. Because of the distributed organization of the software components, the geographic data (geodata) may also be distributed over several distant locations. Dependent on the used client architecture the geodata has to be transported to the client machine. The amount of geodata is getting bigger and bigger and therefore it is more difficult to get the data to the client within a small period of time and without heavy resource usage. This challenging task is the basis for this paper and as a result an approach is mentioned to reduce the amount of data before the data is transported to a client.

For the storage of the geodata an Oracle 11g database server was used. Through the Oracle ODP.NET application interface a small client application sends a request to the database server, where the geodata gets selected and returned to the client. The whole process is recorded in terms of resource usage and duration and the result is displayed in a graphic viewer for visual verification. Through a clever combination of a spatial and thematic filtering, the reduction of small polygons (with automatic topology modifications) and a simplification of complex polylines, a good performance can be reached without the use of high network bandwidths and high resources on the client computer. The spatial and thematic filtering in the database is a common practice when data is selected from a database. The key idea for an efficient data reduction is the consideration of a Smallest-Visible-Object as part of the database request. The main advantage is that the geodata gets already reduced in the database and doesn't have to be delivered to the client computer. As a result, the Smallest-Visible-Object is a degree for the simplification process. A good overall result can be reached through a combination of the mentioned filtering and generalization methods within the database. The concept is implemented in a small test application, evaluated and verified on various test scenarios at the end of the paper.

Inhaltsverzeichnis

Begriffsdefinitionen

Begriff	Beschreibung
Darstellungsmaßstab	Aufgrund des dargestellten Datenausschnitts am Bildschirm ergibt sich ein aktueller Darstellungsmaßstab. Dieser gibt das Verhältnis zwischen einer Strecke am Bildschirm und der zugehörigen Strecke in der Wirklichkeit an. Wird das Anzeigefenster vergrößert oder verkleinert bzw. wird ein Zoomvorgang im Anzeigefenster durchgeführt, so ändert sich der aktuelle Darstellungsmaßstab.
Featureklasse	Eine Featureklasse fasst Geoobjekte zusammen, welche einer gemeinsamen Thematik zuordenbar sind (z.B. Featureklasse Gasleitungen, Featureklasse Grundstücke, Featureklasse Wassereinzugsgebiete, …). In den meisten Softwareprodukten haben alle Featureobjekte einer Featureklasse dieselbe Geometrieart (Punkt, Linie, Fläche, Oberfläche).
Feature oder Featureobjekt	Ist ein einzelnes Objekt, welches einer Featureklasse zugehörig ist. Ein Featureobjekt kann neben dessen Geometrieeigenschaften auch zusätzliche Sachattribute besitzen.
GIS-Administrator	Ein Administrator, welcher die Betreuung für das Viewing-Service übernommen hat. Er ist verantwortlich für die Konfiguration des Viewing-Service, damit dieses von GIS Anwendern eingesetzt werden kann.
Linienzug	= Linestring, Polylinie; eine Folge von einzelnen zusammenhängenden Liniensegmenten.
Liniensegment	Ein einzelnes Segment, welches Teil eines Linienzuges ist.
Maßstabsbereich	Unter einem Maßstabsbereich versteht man eine Von-Bis Maßstabsangabe in welcher die Objekte eine Featureklasse sichtbar sind. Wenn der aktuelle Darstellungsmaßstab innerhalb dieses Maßstabsbereiches liegt, werden die Objekte der Featureklasse dargestellt.
Serviceorientierte Architektur	Eine serviceorientierte Architektur (SOA) ist ein verteiltes System von Services, welche über eine einheitliche Nachrichtenschiene miteinander kommuniziert. Umgesetzt über offene Standards ist der SOA-Architekturansatz ein sehr mächtiger und erweiterbarer.
Smallest-Visible-Object	Laut Li et al. ist das Smallest-Visible-Object (SVO) das kleinste Objekt, welches in der aktuellen Darstellung sichtbar ist. Objekte, welche kleiner dem SVO sind, werden aus der Darstellung ent-

	fernt. Über das SVO-Kriterium wird der Grad der Generalisierung gesteuert.
Styling Rule	Eine Styling Rule beschreibt, wie die Featureobjekte einer Featureklasse dargestellt werden müssen (z.B. Featureklasse Gasleitung: Blaue Linien, durchgezogen, Strickstärke 1.5, …)
Topologie/topologische Beziehungen	Die Topologie beschreibt Beziehungen zwischen Objekten. Dies müssen nicht zwangsläufig die geometrischen Beziehungen zwischen Punkt, Linienzug und Polygon sein. Auch nichträumliche Beziehungen zwischen Objekten können eine Topologie bilden (z.B. ein Auto hat 4 Reifen, jeder einzelne Reifen gehört dabei zu genau diesem einen Auto, …).
Topologieänderungen „On-the-fly“	Sofortige Durchführung von Topologieänderungen im Zuge einer Abfrage
Viewing-Service	Bezeichnung des umgesetzten Lösungsansatzes im Prototyp

Abkürzungsverzeichnis

Begriff	Bedeutung
BBOX	Bounding box; Ausdehnung eines Objektes in X- und Y-Richtung
DBMS	Datenbankmanagementsystem
DE-9IM	Dimensionally Extended 9-Intersection Matrix
DKM	Digitale Katastralmappe
EPSG	European Petroleum Survey Group; Sammlung von Koordinatensystemdefinitionen
FME	Feature Manipulation Engine; ETL Produkt der Firma Safe Software
GIS	Geoinformationssystem
ODP.NET	Oracle Data Provider for .NET; Oracle Anwendungsschnittstelle
SOA	Service-oriented Architecture / serviceorientierte Architektur
SVO	Smallest-Visible-Object
WFS	OGC Web Feature Service
WMS	OGC Web Map Service

Abbildungsverzeichnis

Tabellenverzeichnis

Quellcodeverzeichnis

Inhalt

1. Einführung

1.1) Aufbau von Geoinformationssystemen

Ein modernes Geoinformationssystem (GIS) setzt sich aus vielen einzelnen Softwarekomponenten zusammen. Der komponentenorientierte Ansatz versucht, Verantwortlichkeiten zusammenzufassen und diese in geschlossene Einheiten zu packen. Geoinformationssysteme werden heutzutage in sehr vielseitiger Weise eingesetzt. Die Bandbreite reicht von der Analyse kleinräumiger Phänomene (z.B. Anwendungen in der Medizin oder Biologie) bis hin zur Dokumentation und Untersuchung großräumiger Phänomene (z.B. Überwachung von Wasserständen oder Gletschergebieten, …).

Die Architektur von Geoinformationssystemen hat sich über die Jahrzehnte weiterentwickelt. Einen guten Überblick über die Entwicklung bzw. über einschneidende Ereignisse, die die Entwicklung von GIS-Systemen beeinflussten, gibt Longley et al. in (Longley 2002) an. In der ersten Generation von Geoinformationssystemen sprach man von Dateisystem-basierten Architekturen:

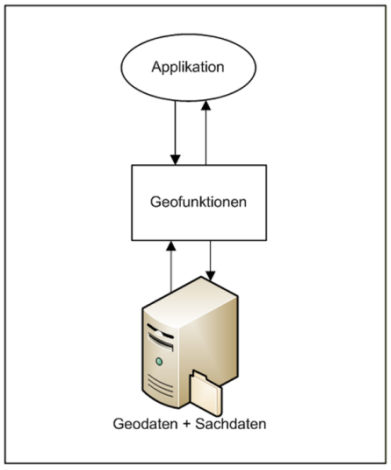


Abbildung 1-1: Geoinformationssystem der ersten Generation

Geodaten und Sachdaten wurden dabei gemeinsam in einer Sammlung von Dateien im Dateisystem abgelegt. Über eine spezielle Anwendung wurden Geofunktionen benutzt, um Analysen auf den Daten durchzuführen.

Dabei gab es einige Einschränkungen, welche dazu führten die Dateisystem-basierte Architektur weiterzuentwickeln:

- Keine Trennung von Geometrie, Thematik und Topologie
- Redundante Datenhaltung
- Keine Standardisierungen
- Ablage als einfache Dateien im Dateisystem

In den folgenden Jahren bzw. Jahrzehnten gab es einige wesentliche Weiterentwicklungen. Die räumliche Information wurde von der Sachinformation getrennt abgelegt. Als Datenspeicher dienten nicht mehr einzelne Dateien sondern Datenbanken. Geofunktionalität wanderte immer weiter in Richtung des verwendeten Datenbankmanagementsystems (DBMS). Zusätzlich wurde eine Vielzahl an Datenformaten definiert, um den Austausch zwischen den einzelnen GIS-Systemen zu erleichtern. In einem modernen Geoinformationssystem kommen verschiedene Varianten einer mehrschichtigen Architektur zum Einsatz. Dabei unterscheiden sich diese Varianten je nach Funktionalität und Aufbau des Clients bzw. Servers.

1.1.1) Architekturvarianten für Clientanwendungen

Thick Client

Der *Thick Clie*nt oder auch *Fat Client* genannt, ist ein Softwareclient, welcher die gesamte Datenverarbeitung übernimmt. Die Daten werden über eine Geodatenbank oder über Geodatendateien bereitgestellt. Sind die Daten am Client angelangt, erfolgt die weitere Verarbeitung. Das Styling der Geodaten auf der Basis von Darstellungsregeln (Styling Rules), die Darstellung der Geodaten auf einem Ausgabegerät, die Bearbeitung der Geodaten oder auch die Anwendung von Analysen auf Basis dieser Daten. Ein großer Teil der Informationsverarbeitung erfolgt demnach auf dem Client, welcher dementsprechend umfangreich mit Infrastruktur ausgestattet sein muss. Daher stammt die Bezeichnung des *Thick Client*. Clients dieser Art müssen daher entsprechende Rechenleistung zur Verfügung stellen können und es muss möglich sein, die Geodatenmenge für die Verarbeitung in den Speicher des Clients zu laden.

Thin Client

Ein *Thin Client* ist hingegen nur eine kleine Anwendung am Clientrechner, wobei dieser über das Intra- oder Internet Informationen bezieht und diese verarbeitet. Die Anwendung läuft dabei meist innerhalb eines Browsers und ist funktional entsprechend eingeschränkt. Zusätzlich ist lediglich ein eingeschränkter Zugriff auf das lokale Dateisystem möglich. Der Client sorgt dabei lediglich für die Darstellung der Geodaten bzw. übernimmt die Benutzerinteraktion. Die Ablage der Geodatenmenge ist dabei neben der vorhandenen Geschäftslogik auf dem Server angesiedelt. Analysen werden direkt am Server ausgeführt, wobei der Client nur mehr die Ergebnisse darstellt. Die Umsetzung dieser Architekturvariante wird oftmals über die Bereitstellung von Services am Server realisiert. Es wird von einer serviceorientierten Architektur (Booth u.a. 2004) gesprochen. Der Großteil der Geodatenverarbeitung wird auf einem Server (Applikationsserver, Datenbankserver) durchgeführt. Der Austausch zwischen dem Server und dem Client erfolgt dabei über standardisierte Services (z.B. OGC Web Map Service, OGC Web Feature Service, …).

Smart Client

Eine sehr populäre Hybridform stellt der Smart Client dar. Hier wird versucht die Vorteile einer Desktopanwendung mit der Flexibilität von *Thin Client* Lösungen zu vereinen. Der *Smart Client* arbeitet ähnlich wie ein *Thin Client*, wobei zusätzlich die Funktionalität erhalten bleibt, wenn die Verbindung zum Server getrennt wird. Geodaten und Komponenten zur Abbildung der Geschäftslogik werden synchronisiert und können offline verwendet werden (vgl. Hill 2005).

1.1.2) Serverseitige Entwicklungen

Auf der Serverseite gibt es in den letzten Jahren ebenfalls einige interessante Veränderungen in den letzten Jahren zu beobachten. Ein klarer Trend bei der Geodatenverarbeitung zeichnete sich bereits über einen längeren Zeitraum ab. Für die Verarbeitung von Geodaten wurde eine immer bessere Unterstützung in den Datenbankmanagementsystemen integriert. D.h., moderne DBMS können mit Geodaten hantieren und bieten Analysen auf Basis der Geodaten an.

Ein aktueller, sehr stark aufkommender Trend der Informationstechnologie ist der Einsatz von Cloud Computing anstatt bisheriger traditioneller Computerinfrastrukturen. Dabei wird versucht Applikationen, Daten und Rechenleistung von der hausinternen Computerinfrastruktur weg zu einem Dienstleister auszulagern. Es werden Services angemietet, welche von einem Softwaredienstleister zur Verfügung gestellt werden. Auch Geoinformationssoftware und Geodatenmanagement haben den Sprung in die Cloud geschafft. Beispiele dafür sind die Firma ESRI mit ihrer *ArcGIS* Produktpalette[1] oder auch die Lösung der Firma *GisCloud*[2]. Vorteil dieser oder ähnlicher Onlinelösungen ist die große Flexibilität für den Anwender. Je nach Lizenzmodell fallen für den Anwender nur dann Kosten an, wenn dieser Services in der „Cloud" nutzt. Dabei können Softwaredienstleister eine Vielzahl unterschiedlicher Services anbieten:

- Immer aktuelle Geodaten zu einem Themengebiet verfügbar.
- Die aktuellste Software ist durch das Service verfügbar, d.h., Softwareupdates werden automatisch durch den Dienstleister eingespielt.
- Ausfallsicherheit und Sicherung von Datensätzen
- Rechenleistung steht je nach Bedarf zur Verfügung (Elastizität, etwa für eine einmalige Analyse von großen Datenmengen)
- …

Ein weiterer aktueller Trend ist der Aufbau sogenannter Geodateninfrastrukturen mit Hilfe von serviceorientierten Architekturen. Dabei werden Geoinformationssysteme über offene Schnittstellen (vgl. Open Geospatial Consortium 2006 oder Open Geospatial Consortium 2010a) miteinander verbunden, um eine möglichst redundanzfreie Datenhaltung zu ermöglichen.

[1] http://www.esri.com/technology-topics/cloud-gis/arcgis-and-the-cloud.html
[2] http://www.giscloud.com/

1.2) Fragestellung

Datenmengen sind heutzutage derartig groß geworden, dass diese nicht immer einfach transportiert bzw. am Client mit ihnen gearbeitet werden kann. Die Datenmenge hängt dabei sehr stark von der jeweiligen Aufgabenstellung ab. Oftmals reicht es mit einigen wenigen Datendateien zu arbeiten. In vielen Anwendungsfällen, speziell in der langfristigen Haltung von Geodatenbeständen (z.B. Infrastrukturdokumentation, Digitaler Katastralmappe, …), sind jedoch sehr große Datenmengen im Spiel.

Es gilt diese Fülle an Information dem Nutzer am Clients entsprechend zugänglich zu machen. Dabei kann, je nach umgesetzter Architektur, der Datenbestand in unterschiedlichsten Varianten verfügbar sein:

- als Dateidatensatz direkt am Clientrechner
- auf einem Datenbank- oder Fileserver im Intranet
- oder auch verteilt in den Weiten des Internets

Verstärkt durch den Trend der Geodateninfrastrukturen und der Nutzung von Cloud Services, sind Geodaten in vielen Fällen nicht direkt auf dem Client verfügbar. Besonders bei größeren Datenmengen wird dabei die Verarbeitung schwierig. Daraus ergeben sich die folgenden Fragestellungen für die vorliegende Arbeit:

- Welche Möglichkeiten gibt es größere Geodatenmengen effizient zu einem Client zu transportieren, um die übertragenen Geodaten darzustellen?
- Ist es ohne die Vorberechnung von Geodaten (Rasterkacheln, generalisierten Ansichten) möglich, das Datenvolumen gering zu halten, welches über das Netzwerk transportiert wird?
- Ist der beschriebene Lösungsansatz dieser Arbeit für jeden Geodatenbestand anwendbar?

1.3) Hypothese

Die vorliegende Master Thesis stellt einen Lösungsansatz vor, der zeigt, wie eine Geodatenmenge effizient an einen Grafikclient transportiert und angezeigt werden kann. Der Lösungsansatz wird mit Hilfe eines Prototyps evaluiert, um entsprechende Aussagen über die Effizienz des Lösungsansatzes tätigen zu können.

Mit dem umgesetzten Prototyp sollen dabei die folgenden Thesen belegt werden:

- Durch die geschickte Kombination von räumlichen und thematischen Filtern, sowie der Anwendung von Generalisierungsoperatoren ist es möglich die Datenmenge stark zu reduzieren. Dadurch ist das Datenvolumen geringer, welches über das Netzwerk zum Client transportiert und anschließend angezeigt werden muss.
- Die Anwendung von Generalisierungsoperatoren kann sehr zeitaufwendig sein. Deshalb soll im Prototyp nur ein einfacher Generalisierungsoperator zum Einsatz kommen, welcher jedoch eine hohe Reduktion der zu übertragenden Objekte erreicht.
- Es wird direkt mit dem Originaldatenbestand hantiert. D.h., gibt es Änderungen am Originaldatenbestand, so stehen diese Änderungen sofort am Client zur Verfügung. Verschiedene Vorberechnungsvarianten auf Vektor- oder Rasterbasis fallen somit für das Lösungskonzept weg, da bei solchen Ansätzen die Änderungen erst nach einer Aktualisierung des verwendeten Caches zur Verfügung stehen.

1.4) Rahmenbedingungen

Da die formulierte Fragestellung durch eine Diskussion mit dem aktuellen Arbeitgeber aufgeworfen wurde und die gewonnenen Erkenntnisse in Softwareprodukte des Arbeitgebers einfließen sollen, gibt es für die durchgeführte Untersuchung definierte Rahmenbedingungen.

1.4.1) Systemvoraussetzungen

Für die Untersuchung im Rahmen dieser Arbeit gibt es folgende, technische Rahmenbedingungen:

- Die Datenhaltung erfolgt in einem *Oracle 11g*[3] Datenbankserver. Dabei handelt es sich um die *Standard Edition* der Datenbank. Es dürfen daher nur die *Oracle Locator* Features zur Hantierung mit räumlichen Geodaten verwendet werden. Die Features *Oracle Spatial* oder *Oracle Partitioning* sind erst mit der Option Enterprise Edition verfügbar.
- Abgefragte Geoinformationen werden in einem Datenviewer angezeigt. Für die Anzeige der empfangenden Geodaten wird eine C# Grafikkomponente der Firma rmDATA GmbH[4] eingesetzt. Der Windows-Client, welcher die Abfrage durchführt, ist eine Microsoft .NET[5] Anwendung. Implementiert wird der Windows-Client in der Programmiersprache C#[6].
- Da der Client als Microsoft.NET Anwendung umgesetzt ist, erfolgt der Zugriff auf die Oracle Datenbank ebenfalls über eine .NET Anwendungsschnittstelle. Von Oracle gibt es die Bibliothek ODP.NET[7], um aus .NET Programmen auf die Oracle Datenbank zugreifen zu können.

1.4.2) Abgrenzungen

- Die untersuchte Lösung ist für die Anzeige von Geodaten gedacht. Für eine Bearbeitung bedarf es weiterer Überlegungen, welche nicht Teil dieser Arbeit sind.
- In der Arbeit wird die Abfrage der Geodaten aus der Oracle-Datenbank, die Übertragung an den Client und die Anzeige der Geodaten beleuchtet. Die Herstellung des Ausgangsdatenbestandes, sprich das Füllen des definierten Datenmodells ist nicht Teil der Master Thesis.
- Für die Evaluierung auf Basis des Prototyps werden ausschließlich die im folgenden Kapitel genannten Testdaten verwendet. Hierbei handelt es sich um freie Datensätze, welche für die Master Thesis verwendet werden dürfen. Eine Über-

[3] http://www.oracle.com/us/products/database/StandardEdition/index.html
[4] http://www.rmdata-geospatial.com/
[5] http://www.microsoft.com/net/
[6] http://msdn.microsoft.com/en-us/vcsharp/default
[7] http://www.oracle.com/technetwork/topics/dotnet/index-085163.html

prüfung mit flächendeckenden Detaildatensätzen (z.B. DKM Österreich) ist aufgrund fehlender Nutzungsrechte nicht möglich. Anhand der durchgeführten Tests mit dem vorliegenden Testdatensatz ist allerdings eine Aussage über die Skalierbarkeit des Lösungsansatzes möglich. Die Ausweitung der Tests mit umfangreicheren Datensätzen könnte ein Thema für zukünftige Untersuchungen sein.

- Die Prüfung des Lösungskonzeptes mit dem Prototyp erfolgt durch Verarbeitung von Polygonfeatures. Die Behandlung von Linien- und Punktfeatureklassen ist vorerst keine Anforderung bei der Umsetzung des Prototyps.

1.4.3) Testdaten

Die durchgespielten Testszenarios zur Belegung der formulierten Thesen wurden mit folgenden freien Geodatenbeständen durchgeführt:

Datensammlung	Datensatz	Quelle	Anzahl Polygone	Anzahl Segmente
Natural Earth Data	Staatsgrenzen weltweit	(Kelso & Patterson 2009)	3994	532696
Global Administrative Areas	Österreich	(Hijmans u.a. 2009)	1	4175
Global Administrative Areas	Bundesländer Österreichs	ebd.	9	10559
Global Administrative Areas	Bezirke Österreichs	ebd.	99	31485
Digitales Oberösterreichisches Raum-Informations-System (DORIS)	Gemeindegrenzen Oberösterreichs	(DORIS 2008)	2285	4312

Tabelle 1-1 Verwendete Testdatensätze

Bei den verwendeten Datensätzen handelt es sich um Featureklassen der Geometrieart Polygon. Die Ausgangsdaten liegen als ESRI Shape Dateien vor. Neben der Polygonanzahl ist in der obigen Tabelle auch die Anzahl der Segmente gelistet, aus welchen die Polygone zusammengesetzt sind. Anhand dieser kann die Komplexität des Datensatzes

besser eingeschätzt werden. Wie man bei der Auswertung noch sehen wird, ist die Anzahl der darzustellenden Segmente ein wesentlicher Faktor für die Gesamtperformance.

1.5) Zielgruppe

Diese Master Thesis richtet sich an Personen, welche Vektordaten aus einer Oracle Datenbank auslesen und anschließend anzeigen möchten. In der vorliegenden Arbeit wird versucht, theoretische Überlegungen zu einer effektiven Ablage und Abfrage dieser Geodaten in der Datenbank umzusetzen. Dabei soll bewiesen werden, dass durch intelligente Selektion und einfache Generalisierung eine effiziente Übertragung und Darstellung ermöglicht wird. Wichtig dabei war, dass die theoretischen Überlegungen auch in einem konkreten Prototyp umgesetzt werden. Es bleibt dadurch nicht nur bei einer theoretischen Ausarbeitung eines Lösungsansatzes bzw. neuer Datenstrukturen. Auf der anderen Seite wurden die Möglichkeiten der eingesetzten Oracle Datenbank nur bedingt ausgereizt. Es war nicht Ziel, durch Oracle-spezifische Tuningtricks eine gute Performance zu erreichen. Aus diesem Grund ist die fachliche Tiefe sowohl im Theorieteil, als auch bei der Umsetzung im Prototyp überschaubar. Soll der beschriebene Ansatz in einem Softwareprodukt eingesetzt werden, ist die Klärung weiterer Detailfragen notwendig. Die Arbeit richtet sich demnach an GIS-Administratoren, deren Aufgabe die Ablage eines Geodatensatzes und die effiziente Darstellung der Daten auf einem Client ist.

1.6) Struktur der Arbeit

Im folgenden Kapitel wird auf verschiedene Lösungsmöglichkeiten in der Literatur eingegangen. Dabei wird kein Anspruch auf Vollständigkeit erhoben. Das Kapitel beschreibt die Varianten nur soweit, wie diese für die weiteren Betrachtungen notwendig sind. In Kapitel 3 findet sich der Theorieteil, in welchem auf die Problemstellung eingegangen und eine Lösungsbeschreibung formuliert wird. Das folgende Kapitel versucht den beschriebenen Ansatz im Theorieteil mit einem Prototyp umzusetzen. Dieser Abschnitt ist relativ techniklastig, da hier auf die konkrete Umsetzung in der Oracle Datenbank eingegangen wird. In Kapitel 5 wird dann anschließend versucht, die aufgeworfenen Fragestellungen durch Testszenarien zu beantworten. Das Ende dieses Kapitels bildet eine kritische Betrachtung des umgesetzten Prototyps. Es wird versucht bestehende

Nachteile der Umsetzung aufzuzeigen. Im letzten Kapitel der Thesis werden die gewonnenen Erkenntnisse zusammengefasst und Verbesserungsvorschläge gemacht, welche im Zuge der Arbeit nicht mehr beleuchtet wurden.

2. Zugänge in der Literatur

Bei der Anzeige von Geodaten gilt es oftmals der Fülle an Daten Herr zu werden. Der gesamte Datenbestand, welcher oftmals so groß ist, dass dieser in einer Datenbank gespeichert werden muss, kann aufgrund von Einschränkungen beim Netzwerk, dem physikalischen Speicherplatz oder dem vorhandenen Arbeitsspeicher nur schwierig weiterverarbeitet werden. Daher gibt es verschiedene Ansätze, welche durch geschickte Filterung oder Vorberechnung die Datenmenge reduzieren, damit diese von einem Client verarbeitet werden können. Bei der Auswahl der in den folgenden Kapiteln beschriebenen Literaturverweisen wurde dabei auf die Relevanz für die vorliegende Arbeit geachtet. Demnach sind hier nur Lösungsansätze erwähnt, welche aufgrund ihrer spezifischen Eigenschaften einen Einfluss bei der Findung eines Lösungsansatzes hatten oder deshalb im Kapitel aufgenommen wurden, um auf den beschriebenen Lösungsansatz hinzuführen.

2.1) Dynamischer Lösungsansatz

In einem möglichen Lösungsansatz für die Problemstellung wird versucht eine Reduktion der Datenmenge durch Abbildungsregeln zu erzielen. Da die Reduktion direkt auf dem Originaldatenbestand angewendet wird, kann immer der aktuellste Stand der Geodaten angezeigt werden. Veränderte Geoobjekte werden sofort bei einer Anzeige am Client berücksichtigt. Aufgrund der dynamischen Anpassung an den neuen Geodatenbestand wird der Ansatz in der Folge als *dynamischer Ansatz* bezeichnet. Bei einer Abfrage werden nur jene Geodaten auf dem Client angezeigt, welche für den Anwender aktuell von Interesse sind. Um dies zu erreichen, wird ein Geodatenbestand abhängig von einer thematischen Konfiguration dargestellt. D.h., Featureklassen werden nur dann angezeigt, wenn diese im aktuell angezeigten Kartenausschnitt mit dem aktuellen Darstellungsmaßstab als sichtbar konfiguriert sind. Dabei ist der Darstellungsmaßstab das Verhältnis einer Strecke am Bildschirm zur entsprechenden Strecke in der Wirklichkeit. Verändert sich die Größe des Kartenausschnitts am Bildschirm oder vergrößert bzw. verkleinert der Anwender den Inhalt des Kartenausschnitts (Zoom-Befehle), ergibt sich ein neuer Darstellungsmaßstab. Aufgrund der Maßstabsänderung ändert sich die Liste der sichtbaren Featureklassen.

Im österreichischen digitalen Kataster (DKM) sind Verwaltungsgrenzen entsprechend ihres hierarchischen Ranges geordnet. Dies geht von der Featureklasse Staatsgrenze bis hin zu den Grenzen niederen Ranges, wie Grundstücksgrenzen oder Nutzungsgrenzen. In einer Katasterdarstellung eines ganzen Landes ist die Darstellung von Grundstückgrenzen oder Nutzungsgrenzen erst ab einem Darstellungsmaßstab von etwa 1:5000 sinnvoll. Bevor der Anwender nicht in diesen Maßstabsbereich navigiert, werden keine Grundstücksgrenzen dargestellt, da die zusätzliche Fülle an Informationen für den Betrachter keinen Mehrwert bringen würde. Die Filterung der Featureklassen ist demnach abhängig von der Thematik. Daher wird diese Art der Einschränkung auch als *thematische Filterung* bezeichnet. Eine weitere Möglichkeit der Filterung von darzustellenden Objekten ist die Filterung auf räumlicher Basis. Es werden nur Geodaten innerhalb eines räumlich abgegrenzten Gebietes geladen und angezeigt. Der dafür anzuwendende Filter wird durch ein räumliches Abfragerechteck oder manchmal auch durch ein Abfragepolygon definiert. Der Vorgang wird als *räumliche Filterung* bezeichnet. Durch die Kombination von thematischen und räumlichen Filtermethoden, können in einem Vektor-GIS gute Performancewerte erreicht werden. In (Bartelme 2005) beschreibt Bartelme, wie räumliche und thematische Filter zu kombinieren sind, um ein gutes Abfrageergebnis zu erhalten. Dabei führt er neben der räumlichen und thematischen Filterung noch die Unterscheidung von Grob- und Feintest ein. Dabei wird durch einen ersten groben räumlichen und thematischen Test eine Reduzierung der Datenmenge erreicht. Der Grobtest ist wesentlich effizienter als der Feintest, hat jedoch Genauigkeitseinschränkungen. Im anschließenden Feintest werden nur mehr jene Featureobjekte geprüft, welche den Grobtest positiv überstanden haben. Thematische und räumliche Filterung ist sowohl für Vektordaten als auch für Rasterdaten möglich.

Problematisch bei der Umsetzung des dynamischen Ansatzes sind Anwendungsfälle, in denen Bearbeitungen bzw. Darstellungen von großen räumlichen Gebieten gefragt sind. Je mehr Featureobjekte gleichzeitig von der Datenbank zum Client transportiert, gestylt und dargestellt werden müssen, desto schlechter sind die Werte für Darstellungsgeschwindigkeit, Datentransfer übers das Netzwerk und benötigter Arbeitsspeicher am Client.

2.1.1) Vorteile des dynamischen Lösungsansatzes

- Darstellung des aktuellen Standes am Client: Features werden in der Version dargestellt, wie diese aktuell in der Datenbank verfügbar sind. D.h., werden Geometrieobjekte in der Datenbank verändert, so sind diese Änderungen direkt am Client verfügbar.
- Darstellungsreihenfolge von Featureklassen ist änderbar: Featureklassen werden in einer definierten Reihenfolge dargestellt. Diese Reihenfolge ist oftmals abhängig vom jeweiligen Anwendungsfall und sollte daher flexibel änderbar sein. Dies ist sowohl für Vektor- als auch Rasterdaten möglich, sofern bei Rasterdaten jede Featureklasse als eigener Rasterlayer verfügbar ist.
- Flexibilität beim Styling von Featureklassen: Die Darstellungskonfiguration (Styling Rules) von Featureklassen kann jederzeit flexibel verändert werden. Des Weiteren sind ad hoc Sichtbarkeitsschaltungen auf Featureklassenebene möglich. Die durchgeführten Änderungen sind direkt am Client verfügbar.

2.1.2) Nachteile

- Umgang mit großen Datenmengen wird schwierig: Die Probleme zeigen sich in der schlechten Ladeperformance und im hohen Verbrauch an Systemressourcen, wenn man den Transport zum Client und die Anzeige am Client betrachtet. Dies kann zum Teil durch Konfiguration von Maßstabsbereichen ausgeglichen werden. Da Geodaten ohne Veränderung an den Client übertragen werden, ergibt sich zumindest ein Netzwerkoverhead aufgrund einer großen Zahl an Features.
- Die Verarbeitung von größeren Datenmengen am Client wird schwierig. Wird die Anzahl der Featureobjekte zu groß, so muss am Client für die Anzeige von Geodaten mit einer längeren Darstellungszeit gerechnet werden. Jede Änderung am Styling der Featureklassen hätte eine Aktualisierung der Darstellung mit derselben Dauer zur Folge. Weitere Bearbeitungsschritte funktionieren womöglich Träge. Die Anzeige- und Verarbeitungsgeschwindigkeit ist natürlich stark vom verwendeten Client abhängig. Je nach Effizienz der umgesetzten Datenstrukturen und Algorithmen unterscheiden sich die Performancewerte der einzelnen Clients.

2.2) Verbesserung durch Vorkalkulation - Statischer Lösungsansatz

Ein weiterer möglicher Ansatz zur effizienten Übertragung und Darstellung von Geodaten ist die Aufbereitung des Originaldatenbestandes, um für den Transport zum Client und der dortigen Darstellung eine Performancesteigerung zu erreichen. Die Reduktion der Datenmenge wird dabei durch Generalisierung erreicht. Dies kann zum einen die Erstellung von Rasterbildern aus dem Originaldatenbestand, oder auch die Generalisierung von Vektordaten für bestimmte Darstellungsmaßstäbe bedeuten. Die Erstellung des generalisierten Datenbestandes wird vorab durchgeführt. Änderungen am Originaldatenbestand werden im generalisierten Datenbestand nicht automatisch nachgeführt. Da die generalisierten Daten statisch für einen Datensatz zu einem bestimmten Zeitpunkt erstellt werden, wird der theoretische Ansatz in der Folge als *statischer Ansatz* bezeichnet.

2.2.1) Erstellung eines Rasterabbildes

Der Vorteil einer Abbildung als Raster ist die natürliche Generalisierung des darzustellenden Gebietes durch Rasterpunkte (Pixel). Je nach Bildmaßstab entspricht ein Pixel des Bildes einem entsprechenden Gebiet. Ein Satellitenbild wird zum Beispiel mit einer Bodenauflösung von 10 Metern erzeugt. Dies bedeutet, dass ein Bildpunkt (Pixel) einem Gebiet von 10 x 10 Metern in der Realität entspricht.

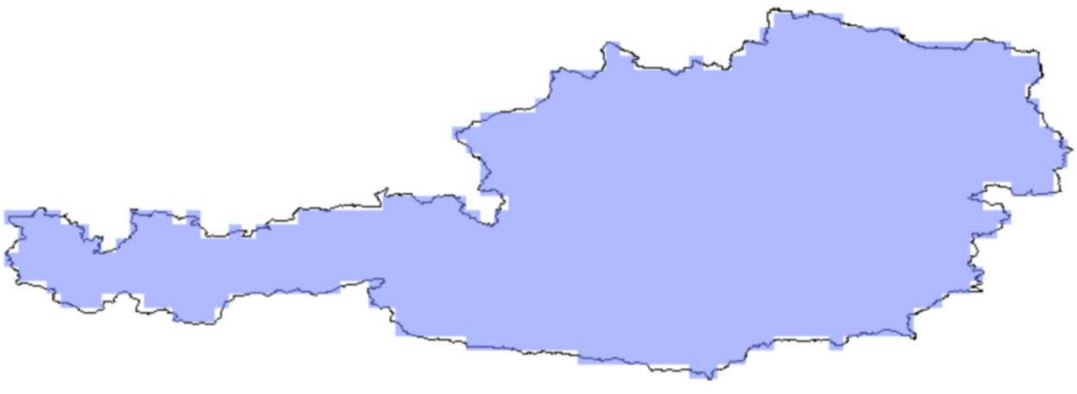


Abbildung 2-1: Generalisierung von Vektordaten durch Erzeugung von Rasterdaten (Daten: Hijmans u.a. 2009)

Neben der natürlichen Filterung der Datenmenge durch die Rasterabbildung gibt es weitere Möglichkeiten, um ein großes Gebiet mithilfe von Rasterbildern abdecken zu können.

Verwendung von Farbpaletten und Farbindizes

Eine Technik zur Reduktion der Datenmenge bei der Modellierung von Rasterdaten ist die Verwendung von Farbtabellen bzw. Farbpaletten. Dabei wird pro Pixel nicht der vollständiger Farbwert sondern lediglich ein Indexwert gespeichert. Mit diesem Farbindex wird über eine Farbpalette die Farbe für einen Pixel ermittelt. In einem Farbraster muss nun nur mehr ein Wert pro Pixel abgelegt werden. Ohne Farbpaletten, muss pro Pixel für jedes Rasterband ein Wert gespeichert werden.

0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0
0	1	0	2	0	0	0	0	0
0	1	0	0	2	0	0	0	0
0	1	0	0	0	2	0	0	0
0	1	0	0	0	0	2	0	0
0	0	0	0	0	0	0	2	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	3	3	3	3	3	3	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0

Farbpalette	RGB-WERT
0	255/255/255
1	218/150/148
2	141/180/226
3	196/189/151

Abbildung 2-2: Beispiel einer Farbpalette

Berechnung und Verwendung von Kacheln

Um ein Gebiet in sehr kleinem Maßstab mit hohem Detailgrad in einem Rasterbild abbilden zu können, benötigt man ein Rasterbild mit enorm großer Ausdehnung in Pixel. Dieses Rasterbild ist meist zu groß, um es in einem Schritt in den Arbeitsspeicher des Clients zu laden. Um dem entgegenzuwirken wird das gesamte Rasterabbild in einzelne kleine Rasterbilder, sogenannte Kacheln (eng. „Tiles“), aufgeteilt.

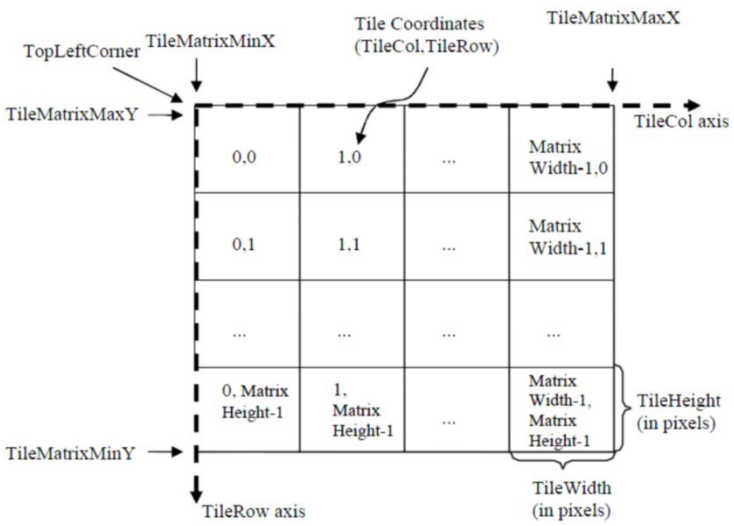


Abbildung 2-3: Aufteilung eines Rasterbildes in Tiles (Bild: Open Geospatial Consortium 2010b)

Die Aufteilung in einzelne Rasterkacheln bringt den Vorteil, dass für eine räumliche Abfrage nur mehr jene Kacheln an den Client transportiert werden müssen, welche innerhalb des Abfragefensters sind oder dieses überschneiden. Sofern die betroffenen Kacheln rasch ermittelt werden, kann für ein kleines Abfragefenster hier ein wesentlicher Performancegewinn erreicht werden.

Berechnung und Verwendung von Bildpyramiden

Eine weitere Technik zur effizienteren Darstellung von Geodaten ist die Berechnung von Bildpyramiden auf Basis eines Ausgangsrasterbildes. Dabei wird für unterschiedliche Darstellungsmaßstäbe jeweils ein Rasterbild mit entsprechender Auflösung erzeugt und gespeichert. Die Auflösung des Rasterbildes gibt dabei den Detailgrad der Abbildung an. Zoomt der Anwender aus dem Darstellungsgebiet, so kann das Rasterbild mit einer geringeren Auflösung verwendet werden, da der Betrachter am Bildschirm ohnehin nicht alle Details des Bildes erkennen kann. Das Bild benötigt aufgrund der geringeren Auflösung weniger Speicherplatz. Dies ermöglicht eine schnellere Übertragung zum Client und eine effizientere Darstellung.

Burt und Adelson beschreiben in (Burt & Adelson 1983), wie durch geschickte Anwendung von Tiefpassfilterung aus einem Rasterbild ein verkleinertes Abbild mit geringerer Auflösung erzeugt werden kann.

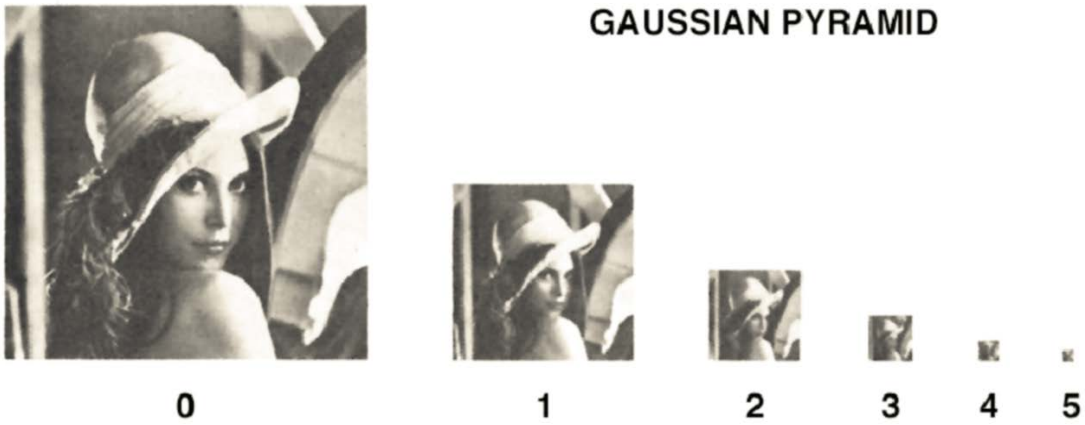


Abbildung 2-4: Abbildung eines Teils der Gauß-Laplace Bildpyramide (Bild: Burt & Adelson 1983)

Sogenannte Gauß-Laplace Pyramiden (vgl. Burt & Adelson 1983) werden bei einer Vielzahl von Anwendungsgebieten der Bildverarbeitung eingesetzt. Die Speicherung der zusätzlichen Ebenen der Bildpyramide benötigt dabei lediglich 1/3 mehr Speicherplatz als das Originalrasterbild.

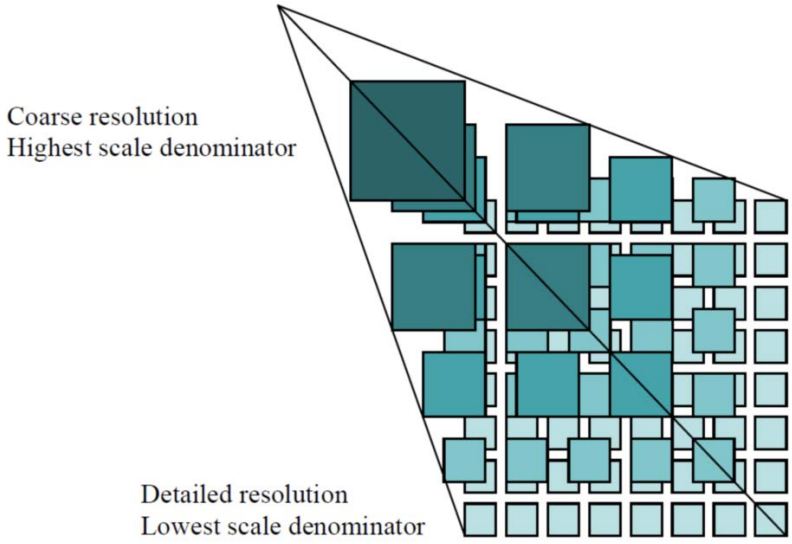


Abbildung 2-5 Bildpyramide mit Kacheln (Bild: Open Geospatial Consortium 2010b)

In der Praxis werden Bildpyramiden oftmals mit einem Kachelsystem zu einem *Tile Service* kombiniert. Dabei ist pro Pyramidenstufe das Rasterbild in Kacheln unterteilt. Bewegt sich der Anwender durch das Darstellungsgebiet, so müssen nur jene Kacheln geladen werden, welche sich gerade im Darstellungsausschnitt befinden. Eine gute Beschreibung, wie beide Ansätze kombiniert werden können, findet sich in der Arbeit von Schütze (Schütze 2007). Für eine Standardisierung dieser Vorgehensweise im Web Map Bereich arbeitet das Open Geospatial Consortium (OGC) mit der *Web Map Tile Service* Spezifikation (Open Geospatial Consortium 2010b) an einem Standard für *Tile Services*.

Die *OGC Web Map Tile Service* Spezifikation ist bisher von offizieller Seite noch nicht freigegeben. Zahlreiche Umsetzungen von *Tile Services* sind auch heute schon im Einsatz. Darunter zählen unter anderen Google Maps[8], Bing Maps[9]oder auch OpenStreetMap[10].

2.2.2) Generalisierung von Vektordaten

Bei der Erzeugung von Bildpyramiden aus Rasterdateien, erfolgt durch den Herstellungsprozess eine natürliche Generalisierung der Daten. Jede Ebene einer Bildpyramide bildet das Gebiet mit einer geringeren Auflösung ab, als die vorhergegangene Ebene. Durch die Zusammenfassung von Pixeln verschwinden sichtbare Objekte, welche in der vorhergegangenen Ebene noch sichtbar waren. Die Komplexität der Generalisierung von Vektordaten ist hingegen um einiges höher. Dabei werden, ähnlich dem Rasteransatz, für bestimmte Darstellungsmaßstäbe generalisierte Ansichten erzeugt und gespeichert. Bei Anfragen durch einen Client wird auf Basis des aktuellen Darstellungsmaßstabes eine geeignete, generalisierte Sicht auf die Geodaten geladen und angezeigt.

Ein Feature in der Karte ist durch folgende Merkmale klassifiziert:

- Information über die Geometrie des Objektes (Position, Form, Ausdehnung)
- Information über topologischen Beziehungen zu anderen Features
- Thematische Information über die Art des Features (Featureklasse) bzw. über dessen Priorität für die Anzeige

Auf Basis dieser Merkmale kann durch Einführung von Kriterien eine Generalisierung von Vektordaten erfolgen (vgl. Li 2007:10–17).

Geometrische Transformation von Features

In (Li 2007:17–25) oder (Galanda 2003) findet man einen Überblick über mögliche geometrischen Transformationen. Je nach Geometrieart (Punkt, Linie, Fläche) gibt es unterschiedliche Möglichkeiten die Geometrie eines Features zu generalisieren. Ein Punkthaufen kann ab einen bestimmten Darstellungsmaßstab zu einem Punkt, ein komplexer Linienzug mit vielen Stützpunkten zu einem vereinfachten Linienzug generali-

[8] http://maps.google.at/
[9] http://www.bing.com/maps/
[10] http://www.openstreetmap.org/

siert werden. Es gibt eine Vielzahl von Generalisierungsmethoden, welche die Geometrie eines Objektes vereinfachen. Transformationsmethoden haben unterschiedlichen Komplexitätsgrad und sind daher mit unterschiedlich großem Aufwand durchzuführen. Dies muss bei der Verwendung bedacht werden. Besonders in zeitkritischen Anwendungen ist die Wahl der Transformationsmethoden entscheidend.

Topologische Transformation von Features

Features in einer Karte stehen in topologischen Beziehungen zueinander. Topologische Beziehungen beschreiben die Beziehungen zwischen den Geometriearten Punkt, Linie und Polygon beschreiben. Diese Beziehungen zwischen einzelnen Geometrieobjekten können durch das DE-9IM-Intersecton Model (vgl. Clementini 1996) beschrieben werden. Nach einer Generalisierung kann mittels Vergleich der DE9-IM Matrix eine Veränderung der topologischen Beziehungen zwischen den Geometrieobjekten geprüft werden. Neben der Prüfung von Überschneidungen kann der Vergleich von Richtungen bedeutend sein. Eine Stadt, welche sich zum Beispiel vor einer Transformation nördlich von Salzburg befindet, soll natürlich auch nach der Transformation nördlich zu finden sein. Je nach Anwendungsgebiet können Kriterien definiert werden, welche auch nach einer Transformation erfüllt sein müssen.

Thematische Transformation von Features

Neben der Generalisierung aufgrund der räumlichen Objektausdehnung spielen oft auch thematische Gesichtspunkte eine wichtige Rolle. Beispiel: In einer kleinmaßstäblichen Straßenkarte werden nur Autobahnen visualisiert, wohingegen in einer Karte mit größerem Maßstab auch Straßen niederen Ranges von Bedeutung sind und dargestellt werden. Die Thematik entscheidet über den Grad der Generalisierung. Thematische Regeln für die Erstellung von generalisierten Sichten auf die Daten müssen in diesem Fall bereits bei der geometrischen und topologischen Transformation berücksichtigt werden. Vertiefende Literatur zu diesem Thema findet man unter (Li & Choi 2002) und (Gao, Gong & Li 2002).

2.2.3) Vorteile des statischen Ansatzes

- Performancegewinn bei der Anzeige der Geodaten: Durch die Reduktion der Datenmenge können Geodaten effizienter zum Client transportiert und dargestellt werden.
- Durch die Reduktion der Datenmenge kann der Client die Daten einfacher und schneller verarbeiten.

2.2.4) Nachteile

- Erheblicher Ressourcenaufwand: Bei der Verwendung von Rasterdaten müssen diese erst erzeugt und abgelegt werden. Daher ist für die Erstellung Rechenzeit und für die Ablage der Rasterdaten eine große Mengen an Speicherplatz notwendig.
- Die Generalisierung der Vektordaten kann je nach Anforderungen zu einer sehr schwierigen und komplexen Aufgabe werden, welche nicht oder nur bedingt automatisiert gelöst werden kann.
- Die Erstellung der generalisierten Ansicht für ein großes abzudeckendes Gebiet ist meist sehr zeitaufwändig.
- Bereitet man Rasterdaten auf, so ist die Änderung der Darstellungsreihenfolge von Featureklassen nur bedingt möglich, da die Rasterdaten für eine fixe Thematik erzeugt wurden. Ansätze sind heute schon vorhanden, bei welchen Rasterdaten pro Featureklasse am Server erzeugt werden und je nach gewünschter Darstellungsreihenfolge „On-the-fly“ zusammengefügt werden (vgl. Helle & Tonnhofer 2011).
- Änderungen am Originaldatenbestand werden nicht sofort in den generalisierten Ansichten verfügbar. Erst nachdem diese auf Basis der neuen Ausgangsdaten aktualisiert wurden, stehen diese Änderungen am Client zur Verfügung.

3. Reduktion durch intelligente Selektion

Im vorigen Abschnitt wurden zwei mögliche Ansätze beschrieben, um die effiziente Darstellung eines größeren Gebietes zu ermöglichen. Beide Ansätze (dynamisch und statisch) haben Vor- und auch Nachteile. Für die Lösung der in Kapitel 1 aufgeworfenen Problemstellung wird durch die Kombination der Vorteile beider Strategien ein möglicher Lösungsansatz beschrieben.

3.1) Beschreibung des Lösungsansatzes

Durch eine Mischung aus dem beschriebenen *statischen* und dem *dynamischen* Ansatz wird die Ablage der Geodaten in der Datenbank derart optimiert, sodass eine Datenbankabfrage und die Darstellung am Client möglichst effizient durchführbar sind. Die vorliegenden Vektordaten werden mit Zusatzinformationen angereichert, damit eine generalisierte Ansicht abhängig vom Darstellungsmaßstab bestimmt werden kann.
Beschrieben wird demnach eine Möglichkeit, eine Datenbank für die Ablage von Multidimensionalen Daten zu nutzen, wobei zusätzlich die Definition einer maßstabsabhängigen Abfrage möglich ist. Das Ergebnis der Untersuchung soll eine Mischung aus intelligenter Datenspeicherung in der Datenbank und effektiver Ableitung der maßstabsbedingten Ansicht sein. Für die Ableitung der maßstabsabhängigen Sicht auf die Geodaten müssen diese überprüft und modifiziert werden. In der Folge ist daher von der Durchführung einer „On-the-fly“ Generalisierung die Rede. Die größte Herausforderung ist dabei das richtige Maß für das Verhältnis zwischen redundanter Datenspeicherung und benötigter Abfragezeit zu finden. Mehrere vorberechnete maßstabsabhängige Sichten garantieren eine gute Performance auf Kosten der redundanten Ablage. Andererseits beansprucht eine möglichst speicherschonende Ablage mehr Zeit, um die angeforderten Datenmengen in der Datenbank zu selektieren, da ohne die durchgeführte Generalisierung die Datenmenge sehr groß sein kann.
In der Literatur findet man bereits einige Ansätze und Ideen zur „On-the-fly“ Generalisierung von Vektordaten bzw. zur Ablage mehrerer maßstabsabhängigen Sichten in dafür geeigneten Datenstrukturen. Diese sind unter dem Begriff *Multi-Scale Data Structures* zu finden. Ein Beispiel dieser Datenstrukturart trägt den Namen *Reactive data structure* und wurde in (van Oosterom 1993) und (van Oosterom 1995) dokumentiert.

In einer *Reactive data structure* werden Geodaten derart effizient abgelegt, sodass eine Ansicht für einen bestimmten Darstellungsmaßstab sehr einfach und effizient abgeleitet werden kann. Generalisierte Ansichten beinhalten in der Regel weniger Geometrieobjekte als der Originaldatenbestand. Durch die Generalisierung wird versucht immer in etwa dieselbe Objektanzahl an den Client zu transportieren. Durch redundante Datenhaltung kann eine generalisierte Sicht auf die Geodaten mit wenig Aufwand abgeleitet werden. Nähere Informationen über die Datenstruktur und eine Implementierung in Oracle findet man in (Meijers 2006). Meijers zeigt sehr gut, dass durch die Erhöhung des Redundanzgrades eine ansprechende Abfragezeit erreicht werden kann. Steigt die Tabellengröße stark an, ist jedoch mit längeren Ausführungszeiten der SQL-Abfragen zu rechnen. Die Idee der Anreicherung von Informationen zu den gespeicherten Vektorelementen wurde für diese Arbeit aufgegriffen. Es sollen jedoch nicht Featureobjekte mehrfach in der Datenbank abgelegt werden. Das zusätzliche Speichern von Informationen soll dabei nur dazu dienen Featureobjekte effizienter abzufragen.

3.2) Reduktion durch intelligente Selektion und Modifikation

Bei einem Einsatz von Bildpyramiden und Rasterkacheln wird eine einfache und natürliche Generalisierung durchgeführt und diese ausgenutzt, um eine Datenreduktion herbeizuführen. Verkleinert der Anwender den aktuellen Darstellungsmaßstab, so werden Featureobjekte in einem Rasterbild reduziert, da diese aufgrund von Pixelzusammenlegungen nicht mehr sichtbar sind. Ähnlich der Rastergeneralisierung wird bei der intelligenten Reduktion ein virtueller Raster über den Datenbestand gelegt. Jene Featureobjekte, deren Ausdehnung die virtuelle Pixelgröße unterschreitet, werden aus dem Datenbestand entfernt. Li et al. (Li & Openshaw 1992, Li & Openshaw 1993) bezeichnet das kleinste sichtbare Objekt als Smallest-Visible-Object (SVO). Diese Bezeichnung wird in der Folge für den Grad der Generalisierung verwendet.

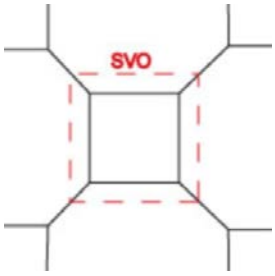


Abbildung 3-1: Prüfung jeder Ausdehnung mit dem SVO-Kriterium

Featureobjekte, welche für den aktuellen Maßstab eine zu kleine Ausdehnung besitzen, dürfen nicht naiv aus dem Datenbestand entfernt werden. Ein Löschen von Features würde erkennbare Lücken in der Darstellung ergeben.

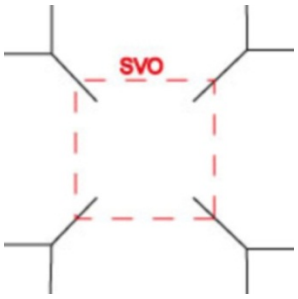


Abbildung 3-2: Lücken nach dem naiven Entfernen eines Feature

Neben der Generalisierung auf Basis des SVO-Kriteriums soll durch den kombinierten Einsatz von Filtern die Datenmenge zusätzlich eingeschränkt werden.

3.2.1) Intelligente Nutzung von Filtern

Die Filterung der Geodaten erfolgt auf zweifache Weise. Zum einen werden nur jene Featureklassen selektiert, welche im aktuellen Darstellungsmaßstab aufgrund konfigurierter Maßstabsbereiche sichtbar sein müssen. Des Weiteren erfolgt zusätzlich die Reduktion der Datenmenge aufgrund der Angabe eines räumlichen Selektionsfensters. Um diese zweistufige Filterung zu unterstützen werden in der Datenbank Indizes erzeugt, die den Zugriff auf Datenbanktabellen beschleunigen.

Problematisch ist dabei, dass in Oracle bei der Kombination von räumlichen und nichträumlichen Prädikaten lediglich ein Filter zu einem Zeitpunkt bearbeiten werden kann. Es können zwar Abfragen mit mehreren Filterbedingungen (Where-Klausel mit mehreren Bedingungen) definiert werden, die Evaluierung des Filters erfolgt jedoch in einer sich aus Regeln ergebenden Reihenfolge. Die Formulierung einer Abfrage, in welcher die räumliche Einschränkung und der thematischer Filter gleichzeitig ausgewertet wird, ist nicht möglich. Durch die Verwendung eines räumlichen Filters mit Hilfe eines Spatial Domain Index in Oracle (Murray 2001) und einer räumlichen Abfrage, können die Geodaten in einem angegebenen Abfragefenster aus der Datenbank selektiert werden. Dabei liefert der räumliche Filter sehr gute Performancewerte, wenn das Abfragefenster einen kleinen Anteil der Gesamtausdehnung des Gebietes abdeckt. Je größer der abge-

fragte Bereich wird, umso nutzloser wird der räumliche Index in Hinblick auf Performanceverbesserungen.
Bei der Einschränkung von Featureklassen auf thematischer Ebene wird für jede Featureklasse überprüft, ob für den aktuellen Darstellungsmaßstab die Featureklasse sichtbar ist. Für die Evaluierung wird pro Featureklasse ein Sichtbarkeitsbereich durch Definition eines VON-BIS Maßstabsbereiches verwaltet, welcher bei der Datenbankabfrage ausgewertet wird. Weitere durchgeführte Schritte werden nur mehr mit jenen Features durchgeführt, welche beide Filterkriterien erfüllen.

Warum bringt diese Kombination Vorteile gegenüber einer Anwendung eines räumlichen Filters ohne thematische Filterung? In vielen Datenbeständen, besonders bei der Abdeckung großräumiger Gebiete, macht eine Darstellung aller Detailinformationen bei sehr kleinen Maßstäben ohne thematische Filterung wenig Sinn. Zu viele Features würden angezeigt werden, sodass nur bedingt Information aus der Darstellung abgeleitet werden kann. Deshalb erfolgt die Einteilung von Featureklassen in sogenannte Sichtbarkeitsbereiche.
In der folgenden Tabelle sind für die Verwaltungsgrenzen des österreichischen digitalen Katasters (DKM) beispielhaft Maßstabsbereiche festgelegt. Ist der aktuelle Darstellungsmaßstab 1:800.000, so werden die Featureklassen Staatsgrenzen und Landesgrenze dargestellt. Erst bei einer Verkleinerung des Maßstabes und einem daraus resultierenden kleineren Abfragefenster, werden zusätzliche Featureklassen dargestellt.

Featureklasse	Sichtbarkeitsbereich
Staatsgrenze	1:1 – 1: unendlich
Landesgrenze	1:1 – 1:1.000.000
Vermessungsbezirksgrenze	1:1 – 1:400.000
Gerichtsbezirksgrenze	1:1 – 1:400.000
politische Gemeindegrenze	1:1 – 1:400.000
Katastralgemeindegrenze	1:1 – 1:100.000
Grundstücksgrenze	1:1 - 1:5.000

Tabelle 3-1: Beispielhafte Einteilung in Sichtbarkeitsbereiche

Wie bereits erwähnt wird neben der thematischen Filterung ein räumlicher Filter angewendet und es werden nur jene Features aus der Datenbank gelesen, welche ein gewähltes Abfragerechteck schneiden.

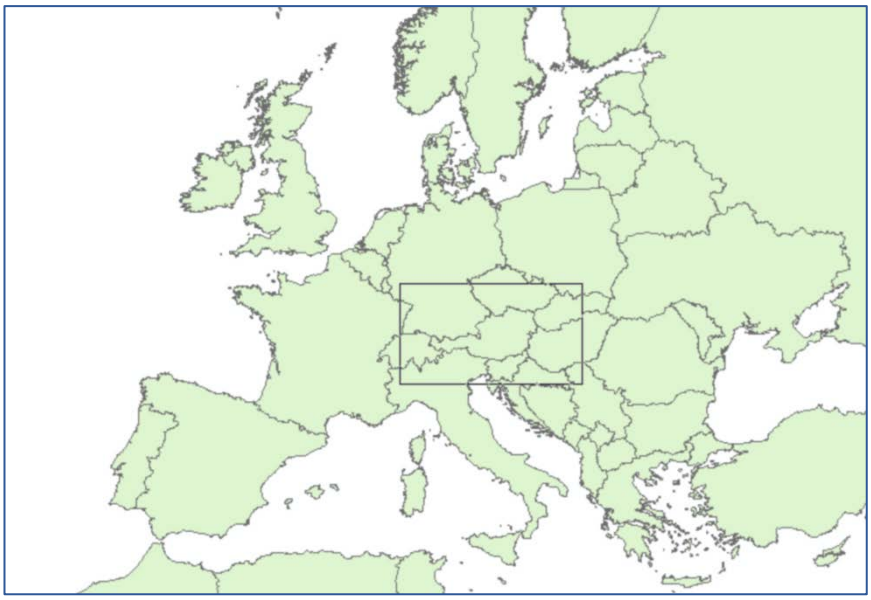


Abbildung 3-3: räumliche Abfrage (Abfragefenster wurde hervorgehoben / Daten: Kelso & Patterson 2009)

Abbildung 3-4: Selektierte Geodaten im gewählten Bereich (Daten: Kelso & Patterson 2009)

3.2.2) Intelligente Topologieänderungen „On-the-fly“

Für die Ableitung generalisierter Ansichten müssen eine Menge von Transformationsoperatoren auf die Daten angewendet werden. Diese zum Teil aufwendigen Transformationen erschweren das Ziel der verbesserten Abfrageperformance. Im vorigen Kapitel wurde bereits erwähnt, dass naives Entfernen von Features zu Lücken im Ergebnis führt. Daher ist für die Durchführung dieser Transformationsmethoden die Berücksichtigung der topologischen Beziehungen zwischen den einzelnen Featureobjekten wichtig.

“Topology is the science and mathematics of relationships used to validate the geometry of vector entities, and for operations such as network tracing and tests of polygon adjacency.” (Longley 2002:190).

Nach Longley werden über die Topologie die Nachbarschaftsbeziehungen zwischen Vektorelementen beschrieben. Diese sind für die Anwendung verschiedenster Analysen bedeutend (z.B. Flächenberechnung eines Polygondatenbestandes, Nachbarschaftsabfragen, …). Für die Generalisierung des Datenbestandes sind diese Beziehungen enorm wichtig, damit beim Löschen eines Features benachbarte Features angepasst werden können.

Da Transformationen im Zuge der Generalisierung sehr aufwendig sein können, werden in dieser Arbeit nur Vereinfachungen betrachtet, welche mit wenig Zeitaufwand durchführbar und einen großen Teil zur Datenreduktion beitragen können. Vereinfachungen werden dabei nur durchgeführt, damit alle Features das definierte SVO-Kriterium erfüllen.

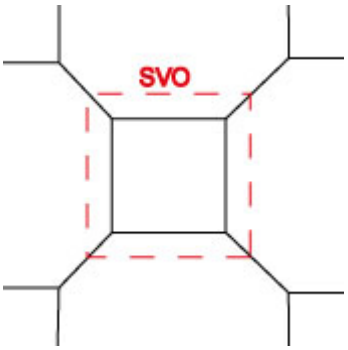


Abbildung 3-5: Prüfung jeder Featureausdehnung mit dem SVO-Kriterium

Das SVO-Kriterium definiert das kleinste sichtbare Feature im Ergebnis. Wenn nun dieses Kriterium passend gewählt wird, bleiben Topologieänderungen im Rahmen und die Datenreduktion aufgrund der Vereinfachung ist im Vergleich zum Ausgangsdatenbestand trotzdem relativ groß.

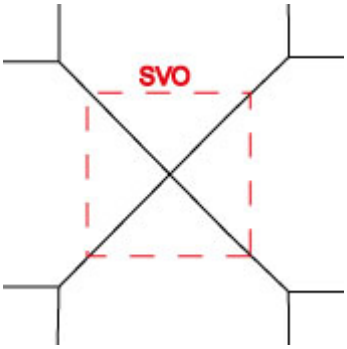


Abbildung 3-6: Entfernen von Features unter Berücksichtigung topologischer Beziehungen

Nach der Entfernung eines zu kleinen Features werden aufgrund der topologischen Beziehungen die Nachbargeometrieobjekte nachgezogen, ohne Lücken entstehen zu lassen. Elemente werden im Zuge der Abfrage verändert oder komplett aus dem Datenbestand entfernt. Topologieänderungen werden dabei direkt auf dem Ausgangsdatenbestand im Zuge der Datenbankabfrage durchgeführt. Daher die Bezeichnung der „On-the-fly“ Topologieänderungen.

3.2.3) Strategien für den Vergleich mit dem SVO-Kriterium

Das Smallest-Visible-Object (SVO) dient als Kriterium, um das kleinste am Bildschirm dargestellte Feature zu definieren. Zu kleine Features werden im Zuge der Abfrage aus dem Datenbestand entfernt. Das SVO ist demnach ein Maß für den Grad der Generalisierung im Datenbestand. Was bedeutet es, dass ein Feature kleiner als das SVO-Kriterium ist?

Für den Vergleich des SVO-Kriteriums mit einem Feature wird im Prototyp die Diagonale der geometrische Ausdehnung des Features herangezogen. Mit der geometrischen Ausdehnung wird hier die maximale Ausdehnung in X- und Y-Richtung bezeichnet. Ist demnach die Diagonale der geometrischen Ausdehnung kleiner als der skalare Wert des SVO, dann ist das SVO-Kriterium nicht erfüllt. Folglich muss dieses Feature aus dem Datenbestand entfernt, und die topologischen Nachbarn angepasst werden. Die geometrische Ausdehnung der Objekte wird deshalb für die Überprüfung des Kriteriums herangezogen, weil diese relativ einfach zu bestimmen ist. Für den Vergleich der geometrischen Ausdehnung mit dem SVO Kriterium gibt es unterschiedliche Strategien.

Vergleich von Seitenlängen mit dem SVO-Kriterium

In der folgenden Abbildung wird ein Polygon mit einer geometrischen Länge von 140.52m und einer Breite von 84.04m dargestellt. Übersteigt das SVO-Kriterium die Länge der kürzesten Seite, so wird das Polygon aus dem Datenbestand entfernt.

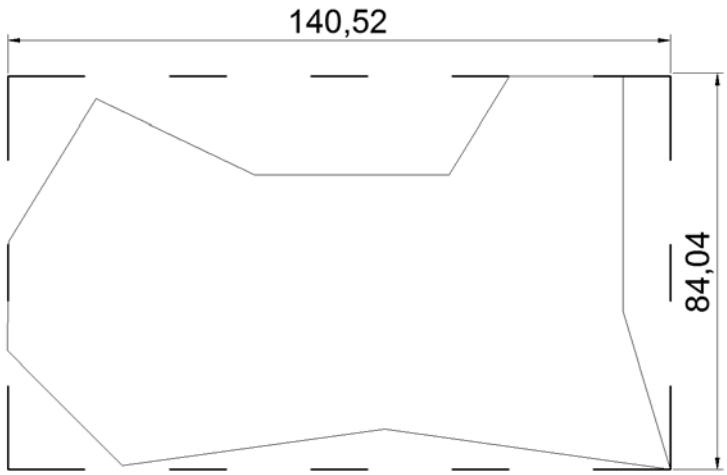


Abbildung 3-7: SVO Prüfung über Seitenlängen

Trotz der Einfachheit des Vergleichs der Seitenlängen mit dem SVO-Kriterium, kann es zu ungünstigen geometrischen Konstellationen kommen, bei welchen diese Variante einen Nachteil mit sich bringt.

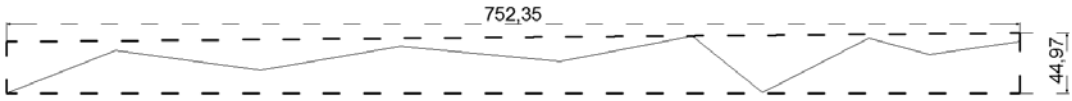


Abbildung 3-8: Nachteile für einen Vergleich von Seitenlängen

Wie in der obigen Abbildung ersichtlich, kann abhängig vom Datenbestand, die Länge und die Breite der Ausdehnung in einem eher ähnlichen oder auch sehr unterschiedlichen Wertebereich liegen. Ist ein Feature um einiges länger als breit bzw. umgekehrt, so würden Features mit einer länglichen Ausdehnung aus dem Datenbestand entfernt werden, obwohl deren visuelle Ausprägung im Vergleich zum gesamten Datenbestand relativ markant ist. Das Ergebnis wäre eine zu starke Generalisierung der Geodaten, welche ein Betrachter als unnatürlich einstuft. Im obigen Beispiel würde ein SVO Kriterium von 45m ausreichen, um ein Feature der Länge 752.35m zu entfernen.

Vergleich der Diagonale mit dem SVO-Kriterium

Um dem Nachteil aus dem vorigen Kapitel entgegenzuwirken, wird im umgesetzten Prototyp mit der Diagonale der Ausdehnung gearbeitet. Features mit länglicher Ausdehnung werden demnach erst entfernt wenn das SVO-Kriterium größer als der Wert der Diagonale der Objektausdehnung ist. Dieser Ansatz bringt für viele Fälle ein gutes Ergebnis. Trotzdem können geometrische Konstellationen auftreten, in welchen eine weitere Optimierung eine Verbesserung erbringen würde.

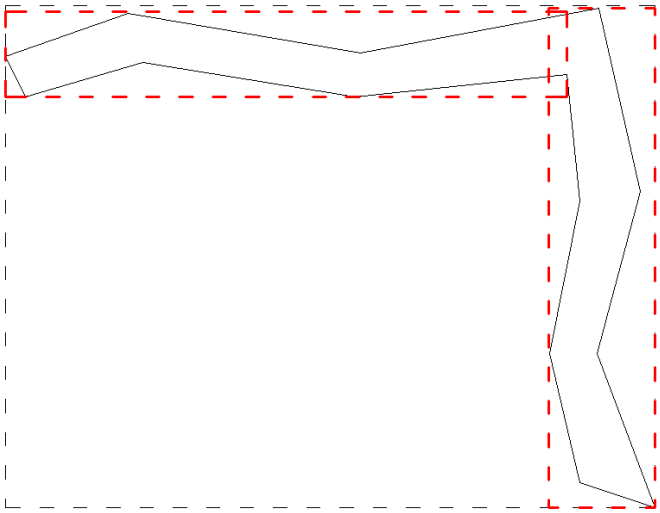


Abbildung 3-9: Ungünstige Konstellation für den Vergleich der Diagonale

In diesem Fall ist die räumliche Ausdehnung des Objektes relativ klein im Vergleich zur Fläche der Ausdehnung. Eine Aufteilung des Features in Teilobjekte würde demnach eine Zerlegung der geometrischen Ausdehnung in Einzelbereiche ergeben und damit einer besseren Ausgangssituation für den Vergleich mit dem SVO-Kriterium führen. Der verbesserten Ausgangssituation steht der erhöhte Rechenaufwand gegenüber. Derartige Fälle müssten erst erkannt bzw. in der Folge aufgelöst werden. Für die Erkennung dieser ungünstigen Situationen kann die Verwendung sogenannter *Shape Indizes* angedacht werden. Hierbei handelt es sich um Kennwerte, die die Form eines planaren Objektes beschreiben. Einen sehr guten Überblick über mögliche formbeschreibende Indizes findet man in (Kidner 1996). Besonders zu erwähnen ist hierbei das Kompaktheitsmaß (*Compactness ratio*) bzw. Dünnheitsmaß (*Thinning ratio*). Es wird berechnet durch die Formel:

$$T = 4 * \pi(\frac{A}{p^2})$$

, wobei A dem Flächeninhalt des Polygons und p dem Umfang des Polygons entspricht.

Nach dieser Berechnung hat ein Kreis eine maximale Kompaktheit von 1. Generell kann man sagen, dass Objekte mit höherem *Thinning ratio* eine kompaktere Ausprägung haben als Objekte mit kleinem *Thinning ratio*. Demnach sind Objekte mit niedrigem *Thinning ratio* ein Indiz dafür, dass es sich hierbei um ein unregelmäßiges Polygon handelt, für das man beim Vergleich mit dem SVO-Kriterium eine genauere Prüfung durchführen sollte. Die Kompaktheit eines Features wird durch das Verhältnis des Flächeninhalts zum Umfang eines Polygons beschrieben. Um einen derartigen Problemfall zu finden, gibt es auch einen einfacheren Lösungsansatz. Aus beiden Seitenlängen der Ausdehnung wird der Maximalwert ermittelt und das Quadrat dieses Wertes mit dem Flächenausmaß des Features verglichen. Man ermittelt diese Verhältniszahl durch:

$$V = \frac{A}{\max(l_{bbox}, b_{bbox})^2}$$

Je kleiner die Verhältniszahl ist, desto größer weicht die Fläche des Features von der ermittelten Fläche der maximalen Objektausdehnung ab.

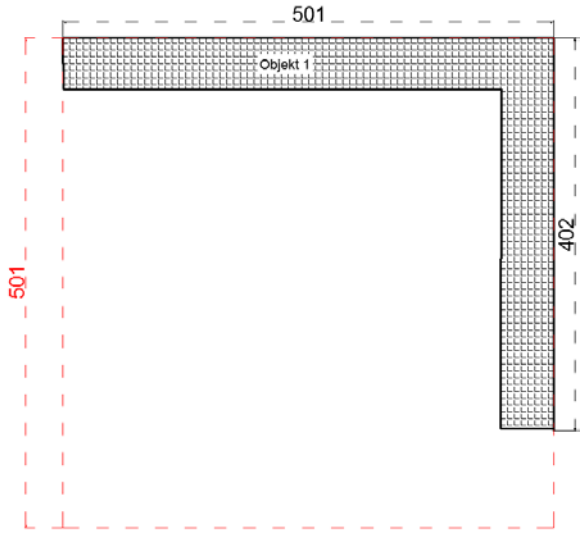


Abbildung 3-10: Suche nach schmalen Polygonfeatures mithilfe einer Verhältniszahl

In obigem Beispiel hat das dargestellte Feature eine geometrische Ausdehnung von 501m x 402m. Die längere Seitenlänge wird für die Berechnung verwendet. Als Ergebnis erhalten wir einen Verhältniswert von

$$V = \frac{44941}{501^2} = \sim 0{,}18 = \ 18\%$$

Die Objektfläche deckt demnach nur rund 18% der ermittelten Fläche auf Basis der längsten Seitenlänge ab. Ist dieser Wert unterhalb einer definierten Schranke, sollte man

eine Zerlegung des Polygonfeatures für den Vergleich mit dem SVO-Kriterium in Betracht ziehen.

Generalisierung über thematische Regeln

Martijn Meijers spricht in seiner Master Thesis (Meijers 2006) eine weitere interessante Möglichkeit für die Wahl des SVO-Kriteriums an. Dabei führt dieser den Begriff *Importance* bzw. *Importance-Level* ein, wobei die Wichtigkeit eines Features darüber entscheidet, ob es im aktuellen Darstellungsmaßstab angezeigt werden muss. Der *Importance Level* kann dabei vom Anwender entsprechend festgelegter Geschäftsregeln (*Business rules*) definiert oder eben wie zuvor beschrieben von der räumlichen Ausdehnung eines Features abhängig gemacht werden.

Dabei könnte man zum Beispiel wichtige Verkehrsrouten oder zentrale, bedeutende Gebäude in einem Datenbestand durch einen hohen *Importance Level* kennzeichnen, sodass diese auch noch bei einem sehr kleinen Maßstab sichtbar sind. Wichtig für die Verwendung eines *Importance Levels* ist dabei die Zuordnung zwischen dem jeweiligen Level und dem aktuellen Darstellungsmaßstab. Abhängig vom aktiven Darstellungsmaßstab muss entschieden werden, ob ein Feature erhalten bleiben muss oder entfernt werden kann.

3.2.4) Größe des Smallest-Visible-Object

Das Smallest-Visible-Object definiert den Grad der Generalisierung, welche im Zuge der Datenbankabfrage durchgeführt wird. Wie wird nun die Größe des SVO-Kriteriums festgelegt? Für die Definition gibt es grundsätzlich zwei Möglichkeiten:

- Definition als Anteil des dargestellten Grafikfensters in Bildschirmeinheiten:

$$SVO = Länge\ des\ Grafikfensters * \frac{Anteil}{100}$$

  Die Länge des Grafikfensters kann in Pixel oder Millimeter ermittelt werden. Die sich aus der obigen Formel ergebende Größe des SVO-Kriteriums, wird in Modelleinheiten umgerechnet und in der Abfrage berücksichtigt.

- Definition als Anteil der dargestellten Ausdehnung in Modelleinheiten:

$$SVO = Länge\ der\ dargestellten\ Ausdehnung * \frac{Anteil}{100}$$

In diesem Fall liegt die Größe des SVO-Kriteriums bereits in Modelleinheiten vor und kann direkt in der Abfrage berücksichtigt werden.

Umrechnung von Bildschirmeinheiten zu Modelleinheiten

Die Größe des SVO-Kriteriums muss für die Datenbankabfrage in Modelleinheiten vorliegen. Dies stellt allerdings eine Herausforderung dar, da die Ermittlung eines korrekten Maßstabes nicht einfach möglich ist. Dies soll anhand eines Beispiels demonstriert werden. Verwendet wird ein Bildschirm der Größe 16.4 Zoll mit den Abmessungen 363 mm x 204 mm. Die Angabe der 16.4 Zoll bezieht sich dabei auf die Länge der Bildschirmdiagonale. Betrieben wird der Bildschirm mit einer Auflösung von 1920x1080 Pixel. Durch den Vergleich der Bildschirmdiagonale in Zoll mit der Diagonale in Pixel erhält man die Anzahl der Pixel pro Zoll bzw. durch Umrechnung die Anzahl der Pixel pro Millimeter.

$$diag_{Pixel} = \sqrt{Länge_{Pixel}{}^2 + Breite_{Pixel}{}^2}$$

$$PixelPerInch = \frac{diag_{Pixel}}{diag_{In}}, mit\ diag_{In} = 16.4\ In$$

$$PixelPerInch = 134.3236 \frac{Pixel}{In} = 5.2883 \frac{Pixel}{mm}$$

Ein dargestelltes Fenster mit 1000 Pixel bei einem Maßstab von 1:100.000 entspricht einer Länge von

$$Länge_{Bildschirm} = 1000\ Pixel * \frac{1}{5.2883 \frac{Pixel}{Millimeter}} = 189.1mm$$

$$Länge_{Modell} = 189.1mm * 100000 = 18910000mm = 18910m$$

in Modelleinheiten. Ein SVO-Kriterium mit einem Anteil von 0.1% der Fensterausdehnung ergibt somit eine Länge von 18.91 Meter. Leider ist es in vielen Betriebssystemen nicht möglich, die physikalischen Abmessungen des Bildschirms per Software exakt

auszulesen. Deshalb ist bei der Ermittlung und Anzeige von berechneten Maßstäben am Bildschirm immer eine gewisse Ungenauigkeit enthalten.

Maßstabsermittlung bei geographischen Koordinatensystemen

Zusätzliche Vorsicht ist bei der Darstellung von Daten in geographischen Koordinatensystemen geboten. Diese können mittels einfacher Projektionen in der Ebene dargestellt werden. Jedoch handelt es sich hierbei um die Darstellung von geographischen Längen und geographischen Breiten. Für die Ermittlung der Distanz zweier Punkte darf nicht einfach die euklidische Distanz verwendet werden. Es muss die Länge zwischen den beiden Punkten auf dem Großkreis ermittelt werden. Im umgesetzten Prototyp wurde die Ermittlung der geographischen Länge mit Hilfe der oft verwendeten Haversine Formel (Smart 1960) ermittelt. Dabei wird der kürzeste Abstand zwischen zwei Punkten auf einer Kugel ermittelt. Bei der Umsetzung ist darauf zu achten, dass für die Distanz zwischen den Punkten (-180 lon; 0 lat) und (180 lon, 0 lat) die Äquatorlänge und nicht die Streckenlänge 0.0 Meter ermittelt wird. Zu berücksichtigen ist, dass Projektionssysteme immer gewissen Verzerrungen unterworfen sind. Es herrscht daher nicht überall Längentreue. Im Prototyp wird bei der Längenberechnung berücksichtigt in welchen Breiten sich der Datenbestand befindet. Es handelt sich nur um eine Verbesserung der Berechnung. Eine bestimmte Ungenauigkeit ist weiterhin vorhanden.

3.3) Topologische Beziehungen

Die berücksichtigten geometrischen Grundobjekte sind Punkt, Linienzug (Polylinie) und Polygon. Damit nun im Zuge der Abfrage Topologieänderungen durchgeführt werden können, müssen die topologischen Beziehungen der Geometriearten bekannt sein.

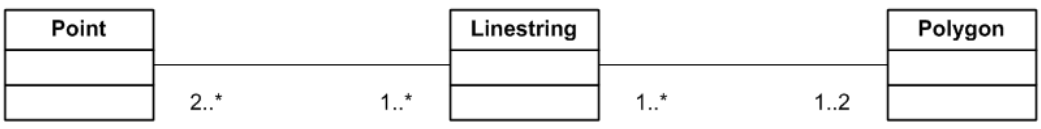


Abbildung 3-11: Topologische Beziehungen zwischen den Geometrieobjekten

Ein Punkt muss Teil eines oder mehrerer Linienzüge sein. Ein Linienzug besteht zumindest aus 2 oder mehreren Punkten und ist Bestandteil von 1 oder 2 Polygonen. Ein

Linienzug verbindet zwei Knoten miteinander, wobei sich ein Knoten dadurch auszeichnen, dass zumindest 3 Linienzüge von diesem starten. Der umgesetzte Vereinfachungsprozess verwendet die vorliegenden topologischen Abhängigkeiten, um die Generalisierung durchzuführen. Dabei kann durchaus vorkommen, dass im Datenbestand implizit mehrere untereinander, unabhängige Topologien vorhanden sind. Für die Vereinfachung werden jedoch nur Geometrieobjekte mit einer direkten Beziehung zum aktuellen Feature berücksichtigt.

3.4) Durchgeführte Vereinfachungen

Für die Lösung der Problemstellung wurde lediglich eine kleine Menge an Generalisierungsoperatoren umgesetzt. Dabei wurde bewusst darauf geachtet, dass die Anwendung des Operators in kurzer Zeit möglich ist und eine hohe Datenreduktion durch die Anwendung des Operators ermöglicht wird.

Linienteile mit dem Douglas-Peucker Algorithmus entfernen

Linienzüge werden über den bekannten und einfach umsetzbaren Douglas-Peucker Algorithmus (Douglas & Peucker 1973) vereinfacht. Ausgangsbasis ist ein aus mehreren Liniensegmenten bestehender Linienzug:

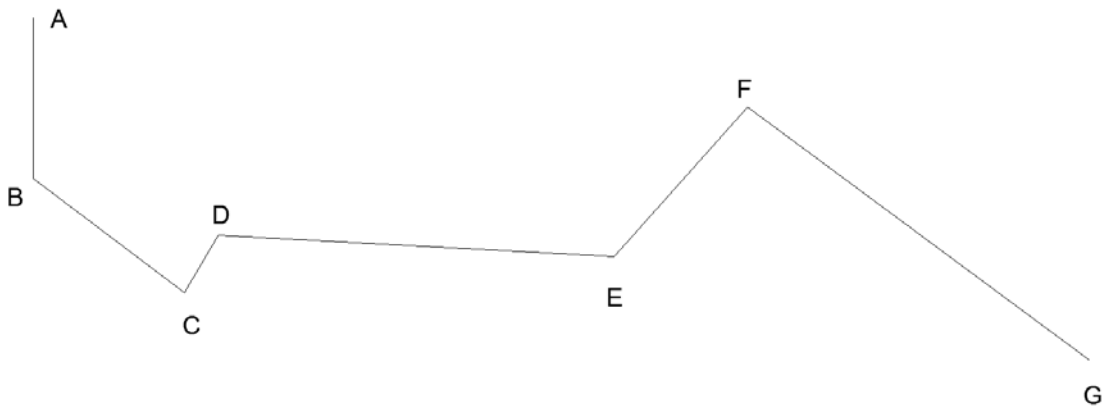


Abbildung 3-12: Beispiel Linienvereinfachung

Für jeden Stützpunkt des Linienzuges wird die Lotdistanz ermittelt und zu diesem Punkt gespeichert. D.h., es wird eine virtuelle Linie zwischen Stützpunkt A und Stützpunkt G gezogen. Dabei wird die Lotdistanz aller Zwischenpunkte zu dieser virtuellen Linie ermittelt. Der Punkt mit der größten Lotdistanz wird ausgewählt und dient als Ausgangsbasis für die weitere Berechnung.

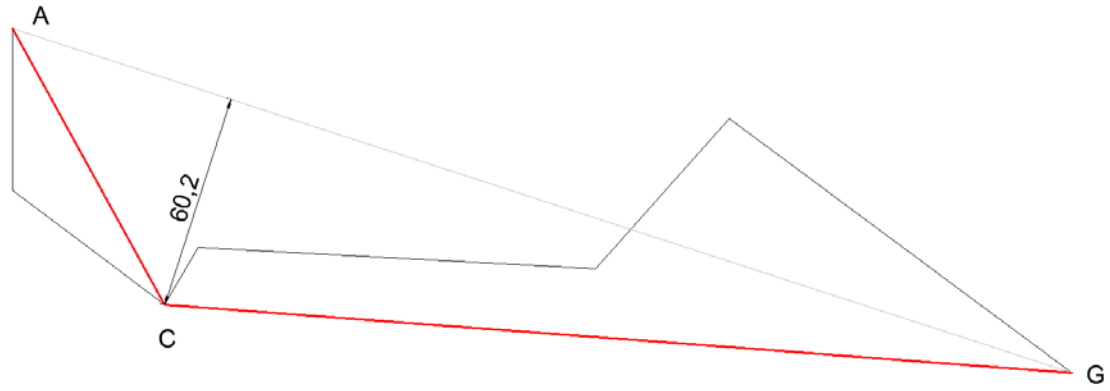
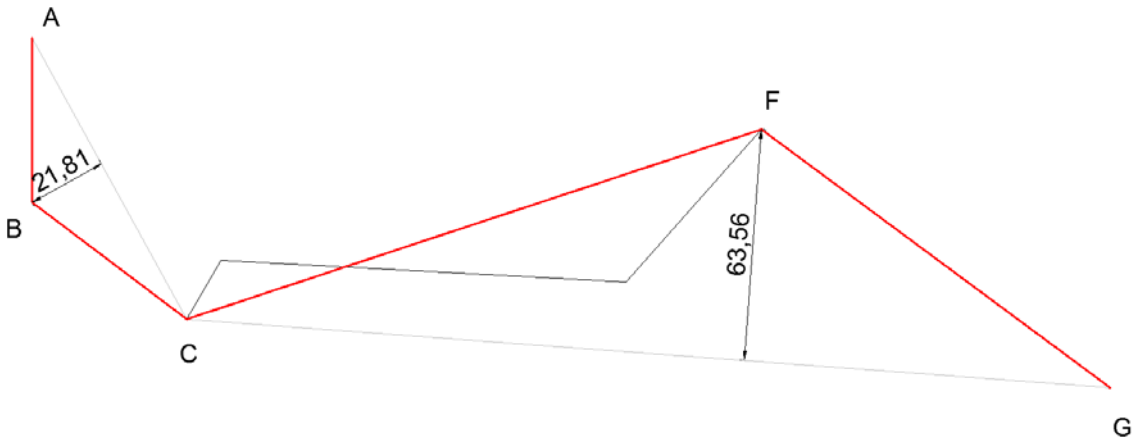
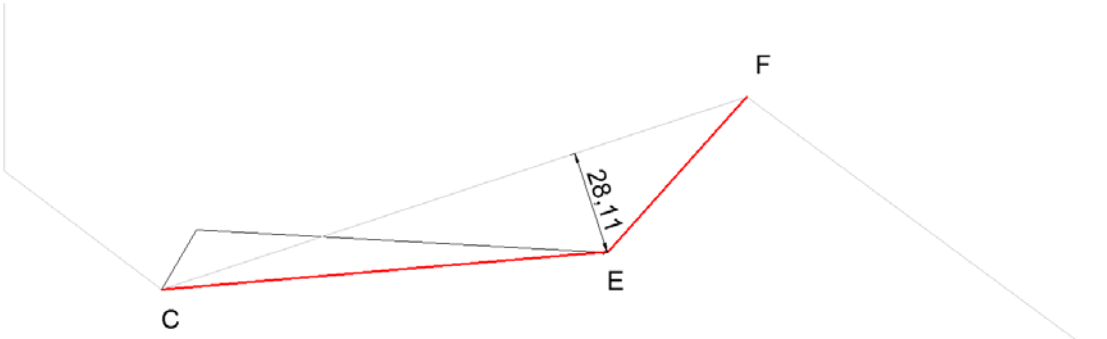


Abbildung 3-13: Douglas-Peucker Linienvereinfachung

Der Stützpunkt C erzielte mit 60.2 die größte Lotdistanz aller Stützpunkte zwischen Anfangs- und Endpunkt. Nun wird vom Anfangspunkt des Linienzuges bis zu diesem Punkt eine virtuelle Linie gezogen und vom Stützpunkt C bis zum Endpunkt. In der Folge wird für beide virtuellen Linien der Vorgang wiederholt. Dies erfolgt solange, bis für jeden Stützpunkt eine Lotdistanz ermittelt wurde.





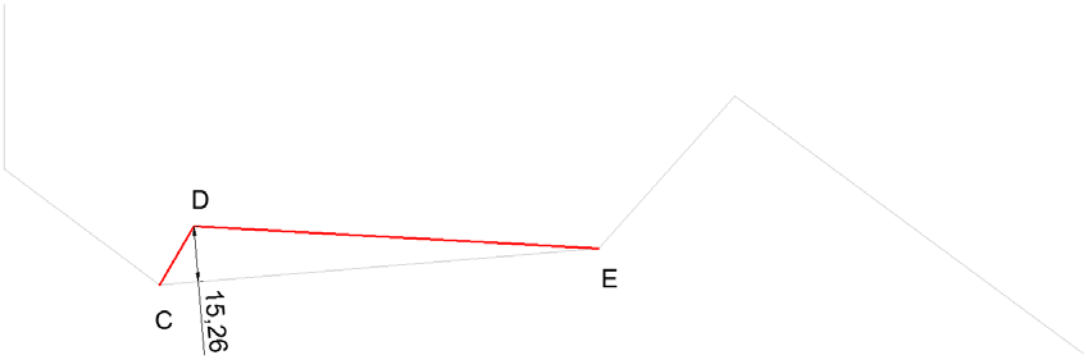


Abbildung 3-14: Fortführung der Douglas-Peucker Vereinfachung

Nach Ermittlung der Lotdistanzen erhält man für die Punkte die folgende Tabelle:

Stützpunkt	Lotdistanz
A	max. Distanz
B	21,81
C	60,2
D	15.26
E	28,11
F	63,56
G	max. Distanz

Tabelle 3-2: Lotdistanzen der Linienzugstützpunkte

Hat das SVO-Kriterium den Wert 40, so werden entsprechend der obigen Tabelle die folgenden Stützpunkte selektiert und daraus ein neuer Linienzug gebildet:

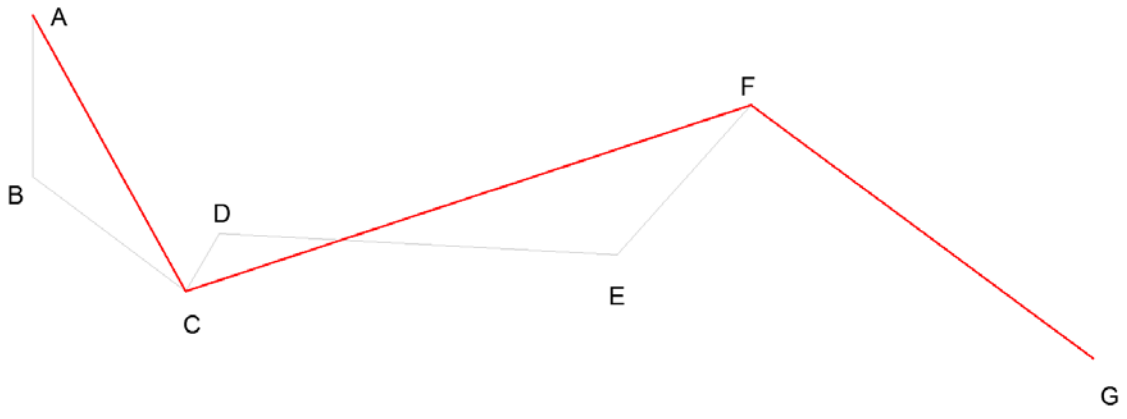


Abbildung 3-15: Vereinfachter Linienzug bei SVO = 40

Polygon entfernen mit Topologieänderung

Werden Polygone gelöscht, so müssen die angrenzenden Polygone und deren Linienzüge angepasst werden.

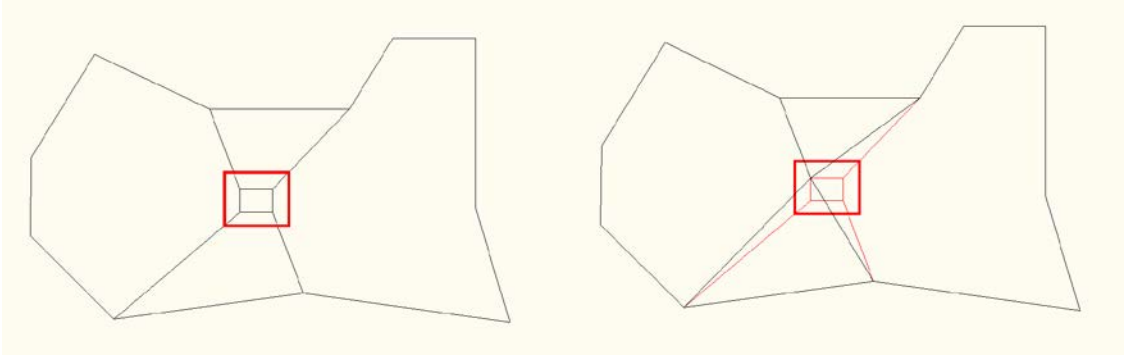


Abbildung 3-16: Polygon entfernen mit Topologieänderung

Polygon entfernen ohne Topologieänderungen

Alleinstehende Polygone können einfach aus dem Datenbestand entfernt werden, da keine Topologieänderungen nachgezogen werden müssen. Diese Vereinfachung wurde hier extra gelistet, weil diese im Prototyp extra speziell berücksichtigt wurde. Das Entfernen freistehender Polygone kann ohne Topologieänderungen durchgeführt werden.

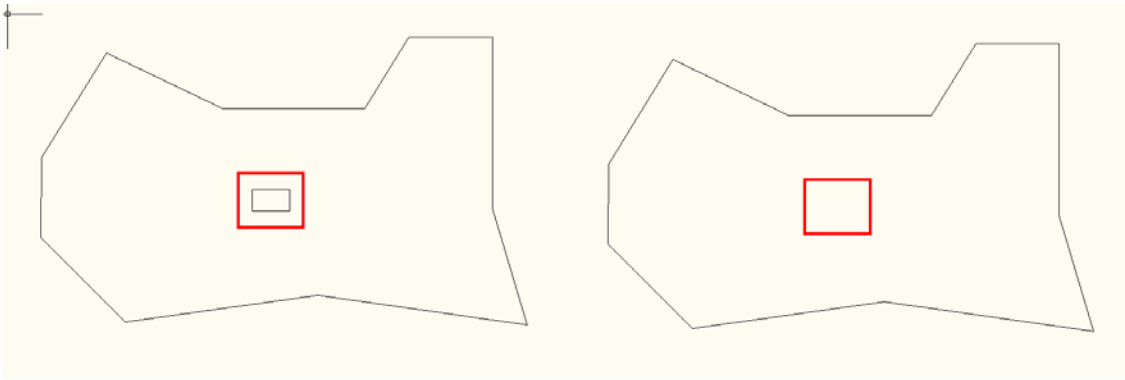


Abbildung 3-17: Polygon entfernen ohne Topologieänderung

3.5) Bereinigung von Ausgangsdaten

Für eine gute Qualität der Generalisierung und für die Erreichung einer guten Abfrageperformance kommt es auf gut aufbereitete Ausgangsdaten an. Daten mit Geometriefehlern oder Fehlern in der Topologie können zu Problemen im Zuge der Generalisierung führen

3.5.1) Fehlerhaftes Entfernen von Features

Sind die topologischen Beziehungen einzelner Geometrieobjekte nicht bekannt, so kann es schnell zu Lücken im Ergebnis kommen, da aufgrund des SVO-Kriteriums Geometrieobjekte einfach entfernt werden, ohne die Änderungen in benachbarte Objekte nachzuziehen. Dies ist aufgrund der fehlenden topologischen Beziehungen nicht möglich.

3.5.2) Generalisierungsfehler aufgrund falscher Topologie

Für die Anwendung des Douglas-Peucker Algorithmus zur Vereinfachung von Linienzügen ist es enorm wichtig, dass Linienzüge immer zwei Knoten miteinander verbinden. Ein Knoten ist dabei ein Stützpunkt an dem mehr als 2 Linienzüge starten.

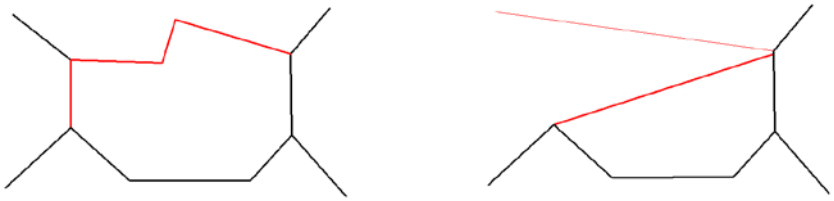


Abbildung 3-18: Abbildung eines fehlerhaften Linienzuges; rechts: Abbildung des vereinfachten Linienzuges

Ein korrekter Linienzug muss somit an jedem Knoten aufgebrochen sein:

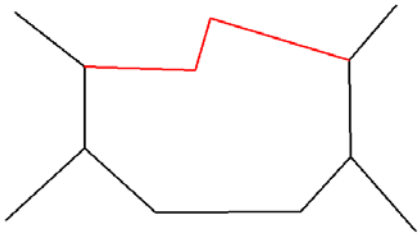


Abbildung 3-19: korrekter Linienzug

3.6) Wahrnehmung durch das menschliche Auge

Das menschliche Auge nimmt die Umgebung entsprechend definierter Gesetzmäßigkeiten wahr. In der Gestaltpsychologie beschäftigt man sich mit diesen Zusammenhängen (siehe Palmer 1999). Das menschliche Auge folgt demnach einem Schema, welches auch für das effiziente Anzeigen von Geodaten ausgenutzt werden soll. Ab einem bestimmten Maßstab können nicht mehr alle Details aufgenommen und korrekt unterschieden werden. Wahrgenommen wird das Bild trotzdem als vollständig und konform. D.h., für die Darstellung der Detailobjekte ist keine exakte Generalisierung der Objekte

notwendig. Auch Topologie relevante Aspekte zwischen Features müssen nicht zwingend eingehalten werden. Ziel muss „nur“ ein wohlgeformtes Abbild des Ausgangsdatenbestandes sein. Ist für den Anwender ein bestimmter Detailbereich interessant, so wird dieser das Abfragefenster ändern und damit den Darstellungsmaßstab, sodass die Details besser zu erkennen sind. Aufgrund des veränderten Darstellungsmaßstabes ergibt sich jedoch ein neues Bild und andere Details können wieder vernachlässigt werden. Das Ziel, welches durch die Vereinfachung verfolgt wird, ist keine exakte Generalisierung (inkl. Einhaltung aller geometrischen, topologischen und thematischen Rahmenbedingungen), jedoch exakt genug, um für den aktuellen Darstellungsmaßstab ein sinnvolles, natürliches Bild zu erzeugen.

4. Umsetzung des Viewing-Service

4.1) Ablaufdiagramm

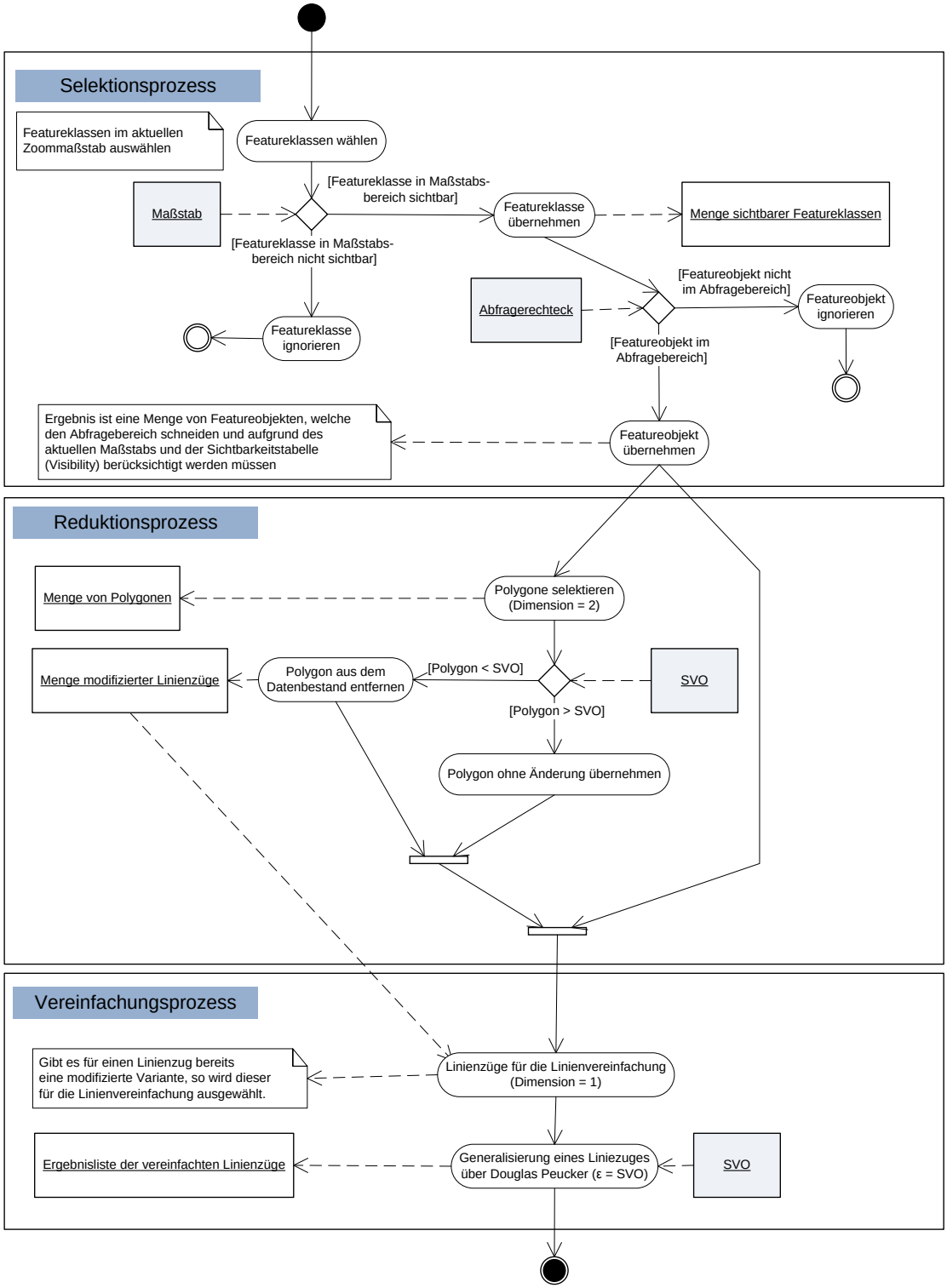


Abbildung 4-1: Arbeitsschritte zur Erzeugung einer generalisierten Sicht auf die Daten

Die Ermittlung des Abfrageergebnisses unter Berücksichtigung des SVO-Kriteriums erfolgt in 3 aufeinanderfolgenden Schritten:

- Selektionsprozess: Die darzustellenden Features werden ausgewählt.
- Reduktionsprozess: Polygonfeatures, die das SVO-Kriterium nicht erfüllen werden entfernt.
- Vereinfachungsprozess: Linienzüge werden durch den Douglas-Peucker Algorithmus vereinfacht

4.1.1) Selektionsprozess

Ziel dieses Arbeitsschrittes ist die Auswahl der relevanten Features aus dem gesamten Datenbestand. Dabei kommt sowohl der räumliche als auch der thematische Filter zum Einsatz. Als Eingabeparameter für den Selektionsprozess werden das Abfragerechteck und der aktuelle Maßstab übergeben. Bei der Umsetzung im Prototyp wird die Aufteilung von Featureklassen in Sichtbarkeitsbereiche berücksichtigt, damit auch größere Datenmengen verarbeitet werden können. Betrachtet man die österreichische digitale Katastermappe (DKM[11]), so findet man eine Aufteilung aller vorhandenen Grenzlinien in die Featureklassen:

- Staatsgrenze
- Landesgrenze
- Vermessungsbezirksgrenze
- Gerichtsbezirksgrenze
- politische Gemeindegrenze
- Katastralgemeindegrenze
- Grundstücksgrenze
- Gebäudegrenze (Hausgrenze)
- Gebäudegrenze aus Luftbildauswertung
- Nutzungsgrenze
- sonstige Linie

Je nach Wichtigkeit der Grenze, ist eine Featureklasse in einem entsprechenden Maßstabsbereich sichtbar. Hochrangige Grenzen, wie Staatsgrenze oder Landesgrenze, sind

[11] http://www.bev.gv.at

noch bei einem sehr kleinen Darstellungsmaßstab interessant, wo hingegen bei Grundstücksgrenzen oder Nutzungsgrenzen aufgrund der Datendichte keine sinnvolle Information ableitbar ist. In Kapitel 3.6 wurde bereits auf den Begriff des *Importance Level* (vgl. Meijers 2006) eingegangen, was im Falle der DKM eine Aufteilung der Grenzlinien nach *Importance Level* entspricht und diese einem Maßstabsbereich zugeordnet werden. Da die Untersuchung auf eine Verarbeitung größerer Datenmengen abzielt, ist die thematische Filterung über Maßstabsbereiche sinnvoll und ermöglicht eine effiziente Datenreduktion, sofern die Anzahl der Features bei Featureklassen höherer Priorität kleiner ist, als bei Featureklassen mit einer niedrigeren Priorität. Featureklassen mit hoher Priorität haben somit einen relativen kleinen Anteil an der gesamten Objektanzahl.

Beim vorliegenden Testdatenbestand handelt es sich um administrative Grenzen unterschiedlicher Priorität in Form von Polygonfeatureklassen. Durch einen thematischen Primärfilter würde die Datenmenge drastisch reduziert werden, bevor anschließend über den räumlichen Sekundärfilter nur mehr jene Featureobjekte gewählt werden, welche sich im Interessensgebiet befinden. Diese funktionieren dann effizient, wenn ein räumlicher Index pro Featureklasse oder Maßstabsbereich verfügbar ist. Der räumliche Index ist jedoch nur für die gesamte Tabelle verfügbar, wodurch die Ausführung unter Umständen länger dauert, wenn die Anzahl der Features sehr groß wird. Hier könnte eine Partitionierung von Tabellen und der Einsatz von räumlichen Indizes pro Partition Vorteile bringen (siehe nächstes Kapitel). Ergebnis des Selektionsprozesses ist eine Menge von Features, welche dem Reduktionsprozess, dem 2. Schritt in der Prozesskette, zugeführt werden.

Partitionierung von Tabellen

In Oracle gibt es das Konzept des *Partitioning* bzw. der *partitioned Indizes*. Dabei wird physikalisch der Inhalt der Tabelle anhand von semantischen Regeln abgelegt, wodurch Zugriffe auf das Dateisystem eingespart werden können. Eine Tabelle kann demnach physikalisch auf Basis unterschiedlicher Methoden gespeichert werden:

- *List partitioning*: physikalische Gruppierung von Datensätzen nach Werteliste
- *Range partitioning*: physikalische Gruppierung von Datensätzen nach Bereichen
- *Hash partitioning*: physikalische Gruppierung von Datensätzen nach Hash-Werten
- …

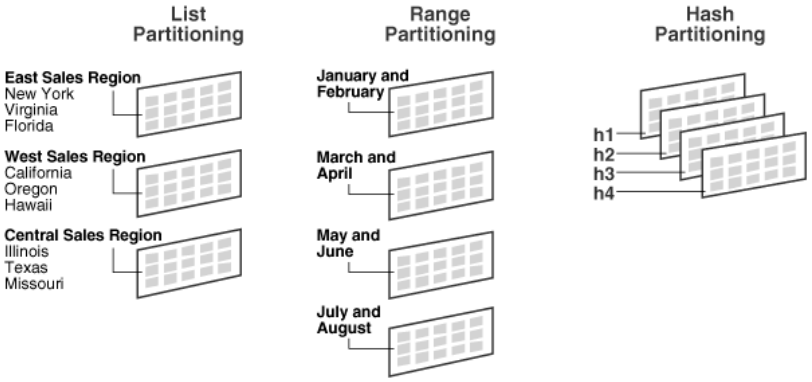


Abbildung 4-2: Partitionierung in Oracle (Bild: Cyran 2002)

Eine vollständige Liste der Partitionierungsmethoden ist in der Oracle Dokumentation[12] der aktuellen Version enthalten. Wird nun eine Abfrage gegen eine partitionierte Tabelle abgesetzt, so können je nach Abfrage einige Partitionen einer partitionierten Tabelle sofort ausgeschlossen werden. Die Datensätze in diesen ausgeschlossenen Partitionen müssen nicht mehr eingelesen und geprüft werden.

Partitionierte Tabellen können auch indiziert werden. Dafür gibt es in Oracle drei verschieden Ansätze, die je nach Einsatzgebiet verwendet werden können:

- *Local Partitioned Indices*: Pro Partition wird ein eigener Index erzeugt. Der Index enthält nur die Datensätze der lokalen Partition.

[12] http://www.oracle.com/technetwork/database/options/partitioning/index.html

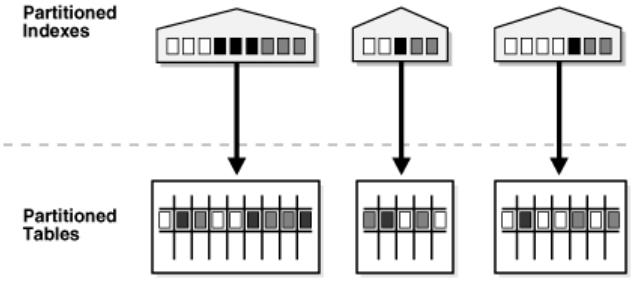


Abbildung 4-3: Local Partitioned Index (Bild: Cyran 2002)

- *Global Partitioned Indices*: Die Partitionierung des Index ist unabhängig von der Tabellenpartitionierung

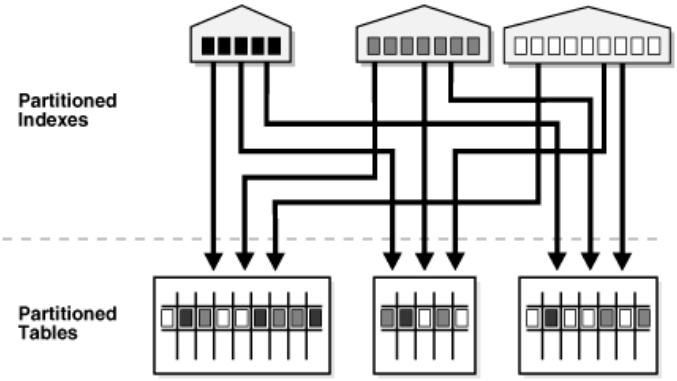


Abbildung 4-4: Global Partitioned Index (Bild: Cyran 2002)

- *Global Non-Partitioned Indizes*: Der Index geht über alle vorhandenen Partitionen. Dies entspricht dem Verhalten einer nicht partitionierten Tabelle.

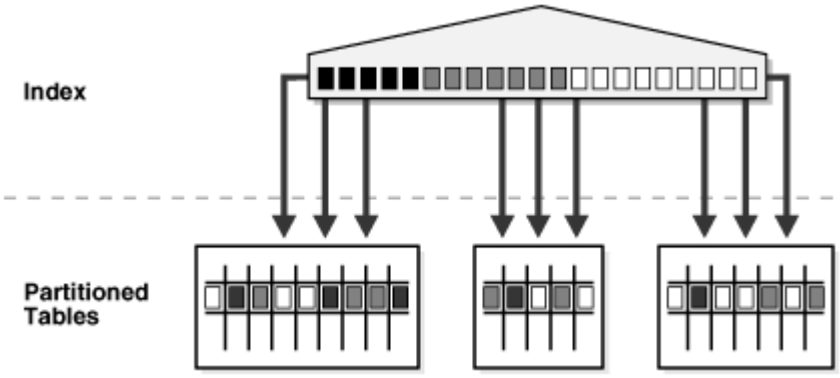


Abbildung 4-5: Global Non-Partitioned Index (Bild: Cyran 2002)

Im Testdatensatz könnte man eine Tabelle nach dem Namen von Featureklassen oder der Featureklassen-ID partitionieren. Auf diesen Partitionen wird anschließend ein lokaler partitionierter Index angelegt. Wird in der Folge eine Abfrage ausgeführt, so werden über den thematischen Primärfilter jene Partitionen ausgeschlossen, welche nicht dem Filterkriterium entsprechen. Über den lokalen räumlichen Index kann anschließend der

Sekundärfilter angewendet werden. Da der räumliche Index nur lokal pro Partition erzeugt wurde, ist dieser kleiner und effizienter als über die gesamte Tabelle.

Im umgesetzten Prototyp wurde diese Möglichkeit jedoch nicht weiter berücksichtigt, da für die Partitionierung von Tabellen die hochpreisige Oracle Enterprise Edition notwendig ist. Als Systemvoraussetzungen für die vorliegende Untersuchung wird jedoch die Oracle Standard Edition verwendet, welche diese Option nicht anbietet. In der Testimplementierung wird demnach der räumliche Filter mit dem thematischen Filter kombiniert, wonach aufgrund der fehlenden Partitionierungsfunktion immer der räumliche Index über die gesamte Tabelle betrachtet werden muss.

4.1.2) Reduktionsprozess

Da im Testszenario Polygonfeatureklassen berücksichtigt werden, bezieht sich der Reduktionsprozess auf die Reduktion der Datenmenge durch Entfernen von Polygonen.
Für jedes Polygon, welches im Zuge des Selektionsprozesses gewählt wurde, wird überprüft, ob die geometrische Ausdehnung des Features markant genug ist, um im Ergebnis aufzuscheinen. Überprüft wird dies über das bereits mehrfach erwähnte SVO-Kriterium. Ist die Diagonale der räumlichen Polygonausdehnung kleiner dem Smallest-Visible-Object, so muss das Polygon entfernt werden und die Geometrie der topologischen Nachbarn entsprechend nachgebessert werden. Alleinstehende Polygone sind im Prototyp speziell behandelt, da das Entfernen dieser durch einfache Filterung im Zuge der Datenbankabfrage möglich ist.

4.1.3) Vereinfachungsprozess

Der Vereinfachungsprozess bezieht sich im umgesetzten Prototypen lediglich auf die Vereinfachung von Linienzügen mit Hilfe des Douglas-Peucker Algorithmus. In (Galanda 2003) oder in (Li 2007) findet man einen guten Überblick über mögliche weitere Generalisierungs- bzw. Vereinfachungsoperatoren. Die Adaption der Geometrie von Features unter Berücksichtigung topologischer und thematischer Regeln ist ein teurer Prozess in Hinblick auf die Performance. Deshalb wird im Viewing-Service nur die Linienvereinfachung nach Douglas-Peucker durchgeführt, welche nach einer Vorbe-

rechnung und Ablage von Lotdistanzen zu jedem Stützpunkt nur wenig Rechenzeit benötigt. Ein großer Vorteil ist dabei, dass bei einer korrekten Ausgangstopologie im Normalfall keine Topologieänderungen notwendig sind.

4.2) Datenbankmodell in Oracle 11g

Wie bereits in Kapitel 3.3 beschrieben, müssen topologische Beziehungen zwischen Geometrieobjekten bekannt sein. Ohne bekannte topologische Beziehungen kann keine sinnvolle Generalisierung durchgeführt werden. Mögliche Auswirkungen einer Generalisierung ohne Berücksichtigung der Topologie sind überlappende Geometrieobjekte oder Lücken im Ergebnis der Abfrage. Je nach Ausgangsdatenbestand kann die Topologie bereits vorliegen oder diese muss erst in einem Vorverarbeitungsschritt ermittelt werden. Neben der Information zur Geometrie eines Objektes werden die topologischen Beziehungen in folgendem Datenbankmodell abgebildet:

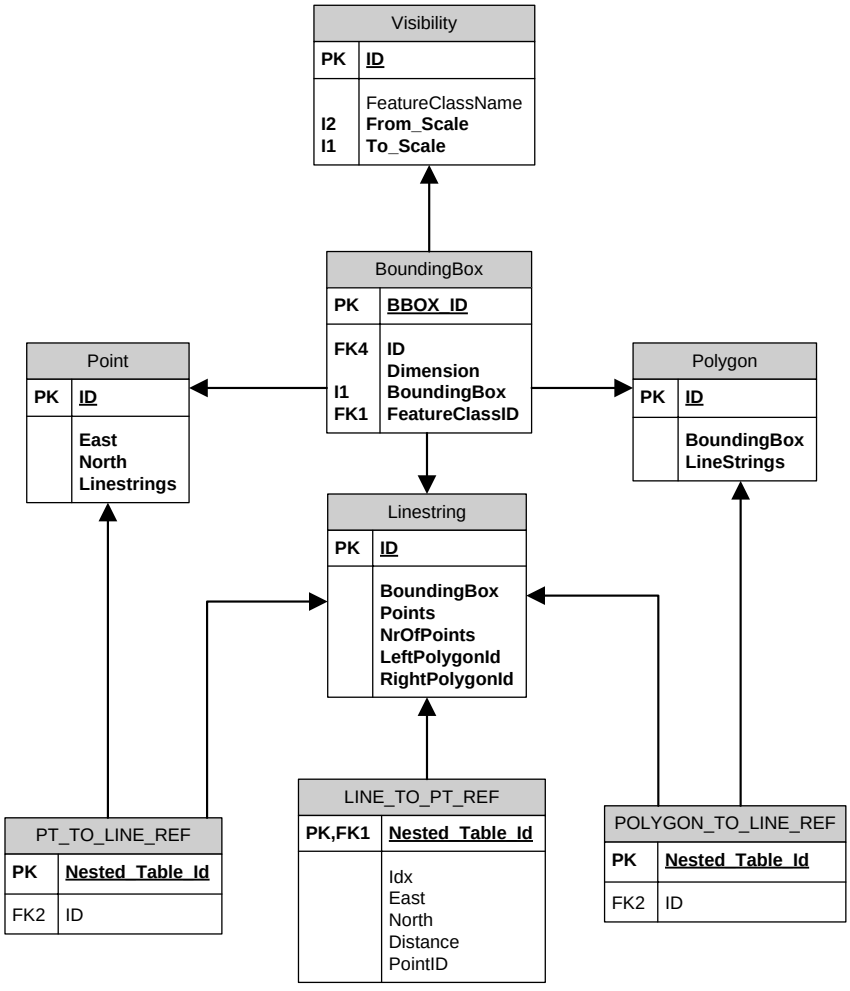


Abbildung 4-6: Datenbankmodell des Prototyps

Die Sichtbarkeitssteuerung von Featureklassen wird über die Tabelle *Visibility* vorgenommen. D.h., zu jeder Featureklasse wird der Maßstabsbereich gespeichert, in welchem die Features der Featureklasse sichtbar sind.

Jedes Feature wird mit seiner Ausdehnung in der Tabelle *BoundingBox* erfasst. Auf dieser Tabelle, im Speziellen auf die Spalte *BoundingBox*, welche den Datentyp *SDO_GEOMETRY* besitzt, wird nach dem Datenimport ein räumlicher Index angelegt. (Oracle Spatial Domain Index (vgl. Murray 2001)). Der räumliche Index ermöglicht sehr effiziente räumliche Abfragen auf die Datensätze der Tabelle *BoundingBox*. Jeder Datensatz besitzt durch die Spalte *Dimension* die Information über die Geometrieart, welcher dieser repräsentiert. Über die Spalte *ID* gelangt man zu den Datensätzen der Tabellen *Point*, *Linestring* und *Polygon*. Diese enthalten die Geometrie- und Topologieinformation der Features. In der *Point* Tabelle werden jedoch nur Knoten, sprich Punkte an denen mehr als 2 Linienzüge enden, gespeichert. Linienzüge und deren Stützpunkte sind in der Tabelle *Linestring* abgelegt. Topologische Beziehungen zwischen den Geometrieobjekten sind nicht über gewöhnliche Foreign Keys modelliert sondern über sogenannte *Nested Tables* (vgl. Arora u.a. 2005, Kapitel „Design Considerations for Oracle Objects“).
Ziel ist die Abbildung der topologischen Beziehungen zwischen Punkt, Linienzug und Polygon. Dabei müssen unter anderem in der Tabelle *Linestring* die Stützpunkte zu einem Linienzug bzw. in der Tabelle *Polygon* die Linienzüge zu einem Polygon gespeichert werden. Diese *One-to-Many* Beziehung könnte über Foreign Keys gelöst werden, jedoch bieten *Nested Tables* den Vorteil, dass diese direkt über eine PL/SQL Prozedur dereferenziert und verwendet werden können ohne vorher einen Join auf eine zweite Tabelle durchzuführen. Des Weiteren sind *Nested Tables* indiziert, sodass ein Zugriff auf die *Nested Table* effizient möglich ist.

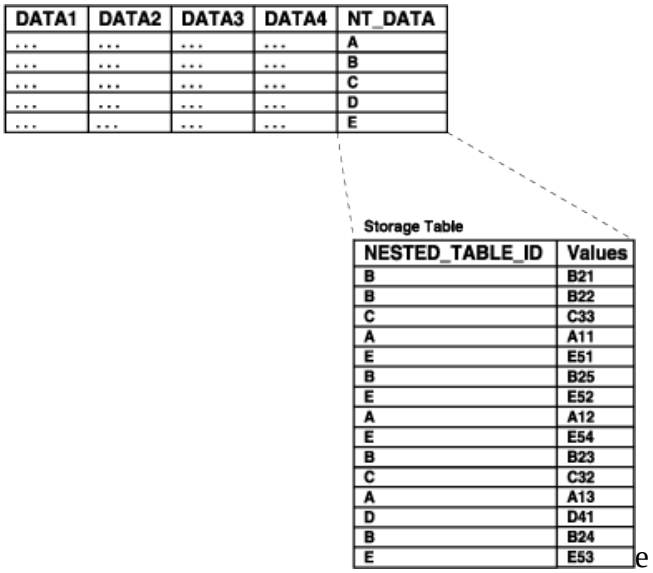


Abbildung 4-7: Nested Table in Oracle (Bild: Arora u.a. 2005)

In Oracle gibt es die Möglichkeit eine Collection als Spaltendatentyp zu verwenden. Dabei gibt es die drei Varianten *VARRAY*, *Nested Table* oder *Associative Array*, wobei die *Nested Table* für diesen Anwendungszweck am besten geeignet ist. Abgelegt werden alle Collections in einer *Nested Table*. Für die gesamte Tabelle *Linestring* gibt es eine *Nested Table*, in welcher alle Stützpunktreferenzen aller Linienzüge enthalten sind. Über eine Abfrage auf die *Linestring* Tabelle, können die dazugehörigen Stützpunkte des Linienzuges aus der *Nested Table* ausgelesen werden. Ist die gesamte Stützpunktanzahl sehr groß, so kann eine Abfrage der Stützpunkte viel Zeit in Anspruch nehmen, da ohne Index auf der *Nested Table* immer die gesamte Tabelle durchsucht werden muss. Aus diesem Grund ist das Anlegen eines Index auf die *Nested Table* sinnvoll. Oracle bietet jedoch zusätzlich die Konfigurationsmöglichkeit der *Nested Table* als *Index-Organized-Table (IOT)*. Dabei werden die Datenbankzeilen der *Nested Table* gruppiert nach Linienzug. Dabei stehen die Stützpunkte eines Linienzuges immer hintereinander in der Tabelle und können effizienter ausgelesen werden. Folglich muss nicht mehr die gesamte *Nested Table* betrachtet werden und der Zugriff ist effizienter.

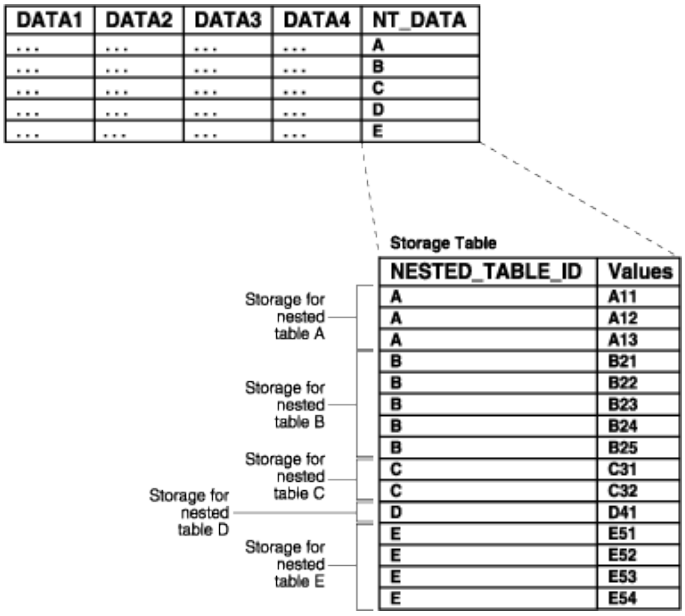


Abbildung 4-8: Nested Table als Index-organized-table (Bild: Arora u.a. 2005)

Nested Tables können in SQL Abfragen mit der Spaltenfunktion *TABLE* sehr einfach in Tabellen umgewandelt und in der Folge in weiteren Joins verwendet werden.

```
// selects all linestring id's of the polygon with the id = 10442751

select polygonlines.id
  from polygon, table(polygon.linestrings) polygonlines
 where polygon.id = 10442751;
```

Quellcode 4-1: Beispielabfrage mit einer Nested Table

Angelegt werden die Tabellen *Point*, *Linestring* und *Polygon* inklusiver der *Index-organized Nested Tables* über folgendes DDL Skript:

```
/*Point table*/
CREATE TABLE "POINT"
  (  "ID" NUMBER(38,0) NOT NULL ENABLE,
     "EAST" FLOAT(126),
     "NORTH" FLOAT(126),
     "LINESTRINGS" ID_TABLE" ,
     CONSTRAINT "POINT_ID_PK" PRIMARY KEY ("ID")
  )
  NESTED TABLE "LINESTRINGS" STORE AS "PT_TO_LINE_REF"
 (( PRIMARY KEY ("NESTED_TABLE_ID", "IDX") ENABLE)
 ORGANIZATION INDEX OVERFLOW
  ) RETURN AS VALUE;

/*Linestring table*/
CREATE TABLE "LINESTRING"
   (  "ID" NUMBER NOT NULL ENABLE,
      "BBOX" "MDSYS"."SDO_GEOMETRY" ,
      "POINTS" VIEWSERV."POINTARRAY" ,
      "LEFTPOLYGON" NUMBER,
      "RIGHTPOLYGON" NUMBER,
      "NROFPOINTS" NUMBER,
   CONSTRAINT "LINESTRING_PK" PRIMARY KEY ("ID")
   )
 NESTED TABLE "POINTS" STORE AS "LINE_TO_PT_REF"
 (( PRIMARY KEY ("NESTED_TABLE_ID", "IDX") ENABLE)
 ORGANIZATION INDEX ) RETURN AS VALUE;

/* Polygon table*/
CREATE TABLE "POLYGON"
   (  "ID" NUMBER(38,0) NOT NULL ENABLE,
      "BBOX" "MDSYS"."SDO_GEOMETRY" ,
```

```
      "LINESTRINGS" "ID_TABLE" ,
      "ISISLAND" NUMBER,
      CONSTRAINT "POLYGON_PK" PRIMARY KEY ("ID")
   )
 NESTED TABLE "LINESTRINGS" STORE AS "POLYGON_TO_LINE_REF"
 (( PRIMARY KEY ("NESTED_TABLE_ID", "IDX") ENABLE)
 ORGANIZATION INDEX OVERFLOW ) RETURN AS VALUE;
```

Quellcode 4-2: DDL Statements zur Erzeugung der Datentabellen

Wie im obigen DDL Statement ersichtlich wird für die Spalte *POINTS* eine *Nested Table* erzeugt, wobei die Stützpunkte der Linienzüge in der *Nested Table LINE_TO_PT_REF* abgespeichert sind. Die *Nested Table* kann nur in Verbindung mit der Parent-Table über ein SQL Statement angesprochen werden. Zur Ablage der Stützpunkte eines Linienzuges werden die *User Defined Types POINTARRAY* und *POINTTYPE* verwendet, welche folgendermaßen definiert sind:

```
CREATE OR REPLACE TYPE POINTTYPE
    AS OBJECT
    (
        IDX NUMBER ,
        POINTID NUMBER ,
        EAST FLOAT ,
        NORTH FLOAT ,
        DISTANCE FLOAT
    );

CREATE OR REPLACE TYPE POINTARRAY
    IS TABLE OF POINTTYPE ;
```

Quellcode 4-3: Oracle User Defined Types POINTTYPE und POINTARRAY

Neben einer Punktnummer und den Koordinaten des Stützpunktes wird für die Unterstützung der Douglas-Peucker Linienvereinfachung noch die Reihenfolge der Punkte durch die Eigenschaft *Idx* und die Lotdistanz pro Punkt gespeichert. Mit diesen zusätzlichen Informationen zu jedem Stützpunkt, lässt sich effizient eine vereinfachte Variante eines Linienzuges erzeugen.

Die Abbildung der Linienreferenzen der Polygone erfolgt durch Ablage der eindeutigen Linien-ID‘s:

```
CREATE OR REPLACE TYPE IDTYPE
    AS OBJECT
    (
        IDX NUMBER ,
        ID NUMBER
    )
;

CREATE OR REPLACE TYPE ID_TABLE
    IS TABLE OF IDTYPE ;
```

Quellcode 4-4: Oracle User defined Type zur Ablage von ID Collections

Die Spalte IDX wird für die Erzeugung der *Index-organized-Table* verwendet. IDX identifiziert in diesem Fall ein Polygon, sodass alle Linienzüge eines Polygons effizient abgefragt werden können.

Die Tabellen *Visibility* und *BoundingBox* werden über das folgende DDL-Statement angelegt:

```
/* Visibility table*/
CREATE TABLE "VISIBILITY"
   (  "ID" NUMBER NOT NULL ENABLE,
      "FEATURECLASSNAME" VARCHAR2(200),
      "FROM_SCALE" NUMBER,
      "TO_SCALE" NUMBER,
      CONSTRAINT "VISIBILITY_ID" PRIMARY KEY ("ID")
   ) ;

CREATE INDEX VIEWSERV."VISIBILITY_TO_SCALE_IDX" ON "VISIBILITY" ("TO_SCALE")
  COMPUTE STATISTICS
 ;

CREATE INDEX VIEWSERV."VISIBILITY_FROM_SCALE_IDX" ON "VISIBILITY" ("FROM_SCALE")
  COMPUTE STATISTICS
   ;

create or replace type BBOX_TABLE as table of bbox_row;

/* Bounding box table */

CREATE TABLE "BoundingBox"
   (  "ID" NUMBER NOT NULL ENABLE,
      "DIM" NUMBER(38,0),
      "BBOX" "MDSYS"."SDO_GEOMETRY" ,
      "BBOX_ID" NUMBER,
      "FCN_ID" NUMBER,
      CONSTRAINT "BBOX_PK" PRIMARY KEY ("BBOX_ID")
   );
```

Quellcode 4-5: Erstellung der Tabellen Visibility und BoundingBox

4.3) Verwendung von Indizes

Für die Nutzung des Viewing-Service ist es wesentlich, dass entsprechende Indizes in der Datenbank erzeugt wurden:

- B-Tree Index auf die Spalten *From_Scale*, *To_Scale* der Tabelle *Visibility*
- B-Tree Index auf die *ID* Spalte jeder Tabelle
- B-Tree Index auf die Spalte *FeatureClassID* in der Tabelle *BoundingBox*
- Räumlicher R-Tree Index auf die Geometriespalte *BoundingBox* der Tabelle *BoundingBox*.
- Verwendung von *Index organized tables* zur Speicherung der Beziehungen zwischen Punkt, Linienzug und Polygon

Durch die Verwendung dieser Indizes wird ein sehr effizienter Zugriff auf die Geodaten ermöglicht.

4.4) Beschreibung der PL/SQL Packages

Der gesamte Prozess wird in einem PL/SQL Package namens Viewing-Service umgesetzt. Die Beschreibung der einzelnen PL/SQL Prozeduren, welche gemeinsam verwendet werden um das Abfrageergebnis zu ermitteln, wird in diesem Kapitel mit Pseudocode beschrieben. Die wesentlichen Teile sind dadurch leichter zu verstehen sein. Methoden, deren Inhalt über den Methodennamen erkenntlich ist, wurden nicht näher beschrieben (z.B. DetermineExtents). Der Quellcode des gesamten PLSQL Packages ist im Anhang verfügbar.

```
/*
Description:
       Function produces a list of reduced and generalized linestrings depending on a
       query window, a smallest visible object and the actual scale
IN:
       x1 - xmin of the bounding box (left top)
       y1 – ymax of the bounding box
       x2 – xmax of the bounding box ( right bottom)
       y2 – ymin of the bounding box
       SVO – smallest visible object
       Scale – actual scale
RETURNS:
       List of linestrings
*/
function GetGeneralizedData(x1    in double,
                            y1    in double,
                            x2    in double,
                            y2    in double,
                            SVO   in double,
                            scale in double)
                            return resultLines is
begin
     // first step: Selection process
     ListOfFeatures := SelectionProcess(x1, y1, x2, y2,scale);

     // second step: Reduction process
     // the modified points and lines are produced through the topology modification
     // they are used in the SimplificationProcess to create new Linestrings
     ModifiedLines = ReductionProcess((select Polygons from ListOfFeatures), SVO,
                                      ModifiedPoints);

     // third step: Simplification process
     return SimplificationProcess((select Linestrings from ListOfFeatures) minus
                                  DeletedLines, ModifiedLines,ModifiedPoints,SVO)
end GetGeneralizedData
```

Quellcode 4-6: Funktion GetGeneralizedData

Die Ergebnislinien werden durch Ausführen des Selektions-, Reduktions- und Vereinfachungsprozesses ermittelt.

4.4.1) Selektionsprozess

Der Selektionsprozess ermittelt mit Hilfe des thematischen und räumlichen Filters die zu bearbeitenden Features:

```
/*
Description:
       Functions selects features from the database who fulfill
        a spatial and thematic filter criteria.
IN:
       x1 - xmin of the bounding box (left top)
       y1 – ymax of the bounding box
       x2 – xmax of the bounding box ( right bottom)
       y2 – ymin of the bounding box
       scale – actual scale
RETURNS:
       List of features
*/
function SelectionProcess(x1    in double,
                          y1    in double,
                          x2    in double,
                          y2    in double,
                          scale in double) return ListOfFeatures
begin
      // get list of feature classes according to the visibility table
      // => thematic filter
      ListOfFeatureClasses := DetermineListOfFeatureClassesFromScale(scale);

      // select the features in the given query rectangle
      // => spatial filter
      foreach Feature in DetermineFeaturesInSelectionArea(x1,y1,x2,y2)

        // important are only those who's featureclass is in the
        //  list of necessary featureclasses
        //  => apply thematic filter
        if( DetermineFeatureClassOfFeature(Feature) in ListOfFeatureClasses)
               ListOfFeatures.Add(Feature);
        end if
      end for

      return ListOfFeatures;
end SelectionProcess;
```

Quellcode 4-7: Funktion SelectionProcess

4.4.2) Reduktionsprozess

Der Reduktionsprozess ist dafür verantwortlich zu kleine Polygonfeatures aus dem Datenbestand zu entfernen.

```
/*
Description:
       Function removes polygons from the dataset which are too small for
       the actual scale.
IN:
       ListOfPolygons – selected polygons from the database query
       SVO – smallest visible object
        ModifiedPoints – through topology modifications modified points are produced
        DeletedLines – removed linestrings through topology modifications
RETURNS:
       ModifiedLines – through topology modifications modified linestrings are produced

*/
function ReductionProcess (ListOfPolygons in Collection,
                           SVO in double,
                           ModifiedPoints out Collection
```

```
                          DeletedLines out Collection) return ModifiedLines is
begin

        foreach Polygon in ListOfPolygons
                // check for polygons which are too small for the actual scale
                if( IsFeatureToSmall(Polygon, SVO))
                        // if too small => remove it and update topology
                        ModifiedLines += RemovePolygonAndUpdateTopology(Polygon,
                                                                        ModifiedPoints,
                                                                        DeletedLines)
                end if;
        end for

        return ModifiedLines;
end ReductionProcess

/*
Description:
        Function checks whether a given feature is too small for the actual scale
IN:
        Feature – can be a point, linestring or polygon
        SVO – smallest visible object
RETURNS:
        True if the feature is too small, otherwise false
*/
function IsFeatureToSmall(feature in Feature,
                          SVO in double) Return Boolean is
begin
        BoundingBox := DetermineExtents(feature);

        Diagonal := DetermineDiagonal(BoundingBox);

        If(Diagonal is smaller as SVO)
                Return true;
        Else
                Return false;
        End if;
end IsFeatureToSmall;

/*
Description:
        Function removes a polygon from the dataset and corrects the topology
IN:
        polygon – polygon to remove
        ModifiedPoints - through topology modifications modified points are produced
        DeletedLines – removed linestrings through topology modifications
RETURNS:
        ModifiedLines - through topology modifications modified linestrings are produced

*/
function RemovePolygonAndUpdateTopology (polygon in Feature,
                                         ModifiedPoints in out Collection,
                                         DeletedLines in out Collection)
                                         return ModifiedLines is
begin

        if(IsPolygonUnattached(polygon))
                // unattached polygons are removed without topology modifications
                DeletedLines += polygon.Linestrings;
        Else
                // remove polygon and correct the topology
                ModifiedLines += RemovePolygonAndModifyTopology(polygon,
                                                                DeletedLines,
                                                                ModifiedPoints);
        End if

        Return ModifiedLines;
end RemovePolygonAndUpdateTopology;

/*
Description:
        Function removes a polygon and modifies the topology
IN:
        Polygon – polygon to remove
        DeletedLines – removed linestrings through topology modifications
        ModifiedPoints – modified points through topology modifications
```

```
RETURNS:
       ModifiedLines – modified linestring through topology modifications
*/
function RemovePolygonAndModifyTopology (polygon in Polygon,
                                         DeletedLines in out Collection
                                         ModifiedPoints in out Collection)
                                         return ModifiedLines is
begin
       PreservingPointId := -1;

       // for each linestring of the polygon
       foreach ActualLine in polygon.Linestrings
            // the first point of the polygon is the preserving point
            // here it is also possible to create a better point, maybe the
            // center of  gravity
            // but the tests shown, the it isn't necessary because the SVO is small
            // enough that a user wouldn't recognize a change if the center of gravity
            // is used

            // GetCopyOfPoint also consideres the modified points that the case where
            // points get modified twice is considered
            if (PreservingPointId = -1 then
               PreservingPointId := ActualLine.StartPointId;
               PreservingPoint    := GetCopyOfPoint(ModifiedPoints,PreservingPointId);
            end if;

              // remove linestring and modify topology
              ModifiedLines += RemoveLinestringAndModifyTopology(ActualLine,
                                                                 PreservingPoint,
                                                                 ModifiedPoints,
                                                                  RemovedLines);
       end for;
end RemovePolygonAndMofifyTopology;

/*
Description:
       Function removes the linestring from the dataset and modifies the topology
IN:
       ActualLine – actual linestring to remove
       PreservingPoint – first point of the polygon preserves in the dataset
*/
function RemoveLinestringAndModifyTopology(ActualLine in Linestring,
                                           PreservingPoint in Point,
                                           ModifiedPoints in out Collection,
                                           RemovedLines in out Collection)
                                           Return ModifiedLines is
begin

  LineStartPointID := ActualLine.StartPoint;
  LineEndPointID  := ActualLine.EndPoint;

  if (LineStartPointID = PreservingPoint.ID AND
           PointAlreadyRemoved.Exists(LineEndPointID) = FALSE) then

     LineStartPointID:= -1; //ignore start point because it's the preserving point
     PointIDToRemove := LineEndPointID;

  elsif (LineEndPointID = startpoint_id AND
           PointAlreadyRemoved.Exists(LinenStartPointID) = FALSE) then

     LineEndPointID := -1; //ignore end point because it's the preserving point
     PointIDToRemove := LineStartPointID;

  else
       if (PointAlreadyRemoved.Exists(LinenStartPointID) = FALSE) then
           PointIDToRemove := LinenStartPointID;
        else
           PointIDToRemove := LineEndPointID;
        end if;
   end if;

   if (PointIDToRemove != PreservingPoint.ID) then

     // determine the linestring references from the actual point
     foreach NeighborLine in DetermineNeighborLines(PointIDToRemove)
            // copy linestring
```

```
            CopiedNeighborLine := GetCopyOfLineString(NeighborLine.ID);

            // redefine point in topology
            RedefinePoint(CopiedNeighborLine,
                          ActualLine.id,
                          PointIDToRemove,
                          PreservingPoint,
                          ModifiedLines,
                          ModifiedPoints);

            ModifiedLines.Add(CopiedNeighborLine);
     end loop;

     PointAlreadyRemoved (PointIDToRemove) := PointIDToRemove;
   end if;
  end if;

  // add actual line to the removed lines
  RemovedLines.Add(ActualLine);
  Return ModifiedLines;
End RemoveLinestringAndModifyTopology;

/*
Description:
       RedefinePoint is the function where the topology modifications take place.
       Point to Linestring references and Linestring to Point references are modified.
IN:
       PreservingLine – line that preserves in the dataset
       LineIDToRemove – id of the line which gets removed
       PointIDToRemove – id of the point which gets removed
       ModifiedLines – modified lines through the topology modifications
       ModifiedPoint – modified points through the topology modifications
*/
procedure RedefinePoint (PreservingLine in Linestring,
                                    LineIDToRemove in Number,
                                    PointIDToRemove in Number,
                                    PreservingPoint in Point,
                                    ModifiedLines in out Collection,
                                    ModifiedPoints in out Collection)
Begin
    // first step: correct point references of the linestring

    // the removed Point Id is changed with the preserving point id
    ModifiedLines(PreservingLine.ID).Points(PointIDToRemove).ID := PreservingPoint.id;
    // Changes are made to the copied linestring

    // second step: correct linestring references of the point
    If(ModifiedPoints(PointToPreserve.ID).Linestrings.Contains(PreservingLine.ID)
       = true)
        ModifiedPoints(PointToPreserve.ID).Linestrings(LineIDToRemove).ID =
             PreservingLine.ID;
    Else
        ModifiedPoints(PointToPreserve.ID).Linestrings.Add(PreservingLine.ID);
    End If;
End RedefinePoint;
```

Quellcode 4-8: Funktion ReductionProcess inkl. Unterprogramme

4.4.3) Vereinfachungsprozess

Der Vereinfachungsprozess generalisiert alle Linienzüge mit dem Douglas-Peucker Algorithmus. Dabei sind die Lotdistanzen bereits in der Datenbank abgelegt und müssen nur mehr ausgewertet werden. Dadurch ist die Erzeugung des generalisierten Linienzugs effizient möglich.

```
/*
Description:
        Function simplifies the given linestrings and return them to the caller
IN:
        LineFeatures – list of linestrings
        ModifiedLines – modified lines through topology modifications
        ModifiedPoints – modified points through topology modifications
        SVO – smallest visible object
RETURNS:
        List of simplified linestrings
*/
Function SimplificationProcess (LineFeatures in Collection,
                                ModifiedLines in Collection,
                                SVO in double)
                                Return Resultlines is
Begin
        Foreach LineFeature in LineFeatures
                // if the linestring was modified through topology modifications, we have
                // to consider the modified linestring
                If(LineFeature is in ModifiedLines)
                        ResultLines += Simplify(ModifiedLines(Linie.Id),ModifiedPoints,
                                        SVO);
                Else
                        ResultLines += Simplify(Linie, ModifiedPoints, SVO);
                End if;
        End For;

        Return ResultLines;
End SimplificationProcess;

/*
Description:
        Function simplifies the given linestring
IN:
        Line – line to simplify
        ModifiedPoints- modified points through topology modifications
        SVO – smallest visible object
RETURNS:
        Simplified linestring
*/
Function Simplify(Line in Linestring,
                  ModifiedPoints in Collection,
                  SVO in double)
                  return Linestring is
Begin

        // Consider point modifications through topology changes
        // The function GetPoint searches for a modified point object
        // if there is one the startnode or the endnode gets modified
        // Note: this is only necessary for startnode and endnode
        // The other points of the line are not modified through the topology changes
        StartNode:= GetPoint(ModifiedPoints,Line.Points[First]);
        EndNode:= GetPoint(ModifiedPoints,Line.Points[Last]);

        Line.Points[First] := StartNode;
        Line.Points[Last] := EndNode;

        // already simple linestrings are not simplified through Douglas-Peucker because
        // they are already simple and the algorithm is an overhead for the process.
        If(Line.Points.Count < 5)
                Return Line;
        End if;
```

```
        foreach Point in Line.Points
                If(Point.Distance > SVO)
                        NewPointList.Add(Point);
                End If;
        End For;

        Return new Line(NewPointList);
End Simplify;

/*
Description:
        Functions returns a point from a given id. If the point was modified through
        topology modifications, the modified point is returned
IN:
        ModifiedPoints – modified points through topology modifications
        Id – id of the point
RETURNS:
        The point with the given Id
*/
Function GetPoint(ModifiedPoints in Collection,
                  Id in Number)
                  Returns Point
Begin
        If(ModifiedPoints.Contains(Id))
                Return ModifiedPoints(Id);
        Else
                Return (Select Point from Points where Points.Id = Id);
End GetPoint;
```

Quellcode 4-9: Funktion SimplificationProcess inkl. Linienvereinfachung

4.5) Zugriff über ODP.NET

Die Verwendung des PL/SQL Packages erfolgt über die Oracle ODP.NET Anwendungsschnittstelle. Diese ermöglicht neben dem Absetzen einfacher SQL-Statements auch den Aufruf von *Stored Procedures*. Demnach kann auch das Viewing-Service über die ODP.NET Schnittstelle genutzt werden. Bevor der Client das PL/SQL Paket nutzen kann muss für die User Defined Types:

- *SDO_Geometry*
- *SDO_Dim_Array*
- *SDO_Dim_Element*
- *SDO_Elem_Info_Array*
- *SDO_Ordinate_Array*
- *SDO_Point_Type*
- *POINTTYPE*
- *POINTARRAY*
- *IDTYPE*
- *ID_TABLE*

eine Wrapperklasse in C# erstellt werden. Dies erfolgt über einen von Oracle bereitgestellten Assistenten. Das Ergebnis sind C#-Klassen, welche die *User Defined Types*

repräsentieren und am Client verwendet werden können. Das Ergebnis einer Abfrage kann anschließend in Instanzen dieser Wrapperklassen gespeichert werden.

Der Aufruf des Viewing-Service und das Auslesen des Ergebnisses ist über einen einfachen, kurzen Quellcodeabschnitt möglich:

```
using (OracleCommand cmd = mConnection.CreateCommand())
{
  // specify command type as stored procedure
  cmd.CommandText = OraConnection.SchemaName+".VIEWINGSERVICE.GetGeneralizedData
  cmd.CommandType = CommandType.StoredProcedure;
  cmd.BindByName = true;

  // Add parameters for the query
  cmd.Parameters.Add("x1", Oracle.DataAccess.Client.OracleDbType.Double,
    querybox.LeftTop.East, System.Data.ParameterDirection.Input);
  cmd.Parameters.Add("y1", Oracle.DataAccess.Client.OracleDbType.Double,
    querybox.LeftTop.North, System.Data.ParameterDirection.Input);
  cmd.Parameters.Add("x2", Oracle.DataAccess.Client.OracleDbType.Double,
    querybox.RightBottom.East, System.Data.ParameterDirection.Input);
  cmd.Parameters.Add("y2", Oracle.DataAccess.Client.OracleDbType.Double,
    querybox.RightBottom.North, System.Data.ParameterDirection.Input);
  cmd.Parameters.Add("svo", Oracle.DataAccess.Client.OracleDbType.Double, svo,
    System.Data.ParameterDirection.Input);
  cmd.Parameters.Add("scale", Oracle.DataAccess.Client.OracleDbType.Double, scale,
    System.Data.ParameterDirection.Input);
  // Result cursor
  OracleParameter p_refcursor = new OracleParameter("result",OracleDbType.RefCursor);
  p_refcursor.Direction = ParameterDirection.ReturnValue;
  cmd.Parameters.Add(p_refcursor);

  cmd.ExecuteNonQuery();

  // open a DataReader to iterate over the resulting cursor
  using (OracleDataReader dr1 = ((OracleRefCursor)p_refcursor.Value).GetDataReader())
  {
     // while there is something to read
     while (dr1.Read())
     {
        rmdata.Graphic.Objects.Polyline pl = new rmdata.Graphic.Objects.Polyline();
        // read point array
        POINTARRAY array = (POINTARRAY)dr1[1];

        // create polyline for the viewer
        foreach (POINTTYPE pt in array.Value)
        {
          pl.Points.Add(new rmdata.Math.Point3d(pt.EAST, pt.NORTH, 0.0));
        }
        pl.SetId((int) (decimal)dr1[0]);

        // return the polyline to the viewer
        yield return pl;
     }
     p_refcursor.Dispose();
  }
}
```

Quellcode 4-10: Nutzung des Viewing-Service über einen C# Client

5. Test

5.1) Testumgebung

Für die Überprüfung der Umsetzbarkeit des Prototyps, die Durchführung der Testszenarios und die Auswertung der Messergebnisse, wurde eine Oracle Database 11g verwendet. Es handelt sich hierbei um die Release 11.2.0.1.0. Trotz der Verfügbarkeit der Oracle Enterprise Edition werden lediglich Features der Standard Edition verwendet, da die Standard Edition Voraussetzung für die vorliegende Arbeit ist. Der Oracle Datenbankserver befindet sich dabei im selben Gebäude wie der Client, welcher über räumliche Abfragen die Geometrie der Polygonfeatures erhält und im Datenviewer anzeigt. Verbunden durch ein Gigabitnetzwerk, kann die eigens in C# umgesetzte Konsolenanwendung Abfragen über die Oracle ODP.NET Schnittstelle an den Datenbankserver übermitteln. Der Client, auf welchem die Tests durchgeführt wurden, ist ein Intel CORE 2 QUAD 9550 mit 2.83 GHz mit 8 GB Arbeitsspeicher. Als Betriebssystem läuft auf dem Testrechner Microsoft Vista Business x64.

Neben der Protokollierung von Objektanzahlen und der Zeitmessung, wird das Ergebnis der Datenbankabfrage über einen Datenviewer angezeigt. Die Darstellung des Abfrageergebnisses in einem Viewer wurde deshalb in die Betrachtung hinzugezogen, da die darzustellende Objektanzahl doch wesentlichen Anteil an der Gesamtperformance des Prototypen hat. Dazu zählt neben der Ausführungszeit der Abfrage auch die Zeit, welche notwendig ist die retournierten Features auf dem Client darzustellen.

5.2) Testszenario

Konkret werden bei den Tests 3 Testszenarios ausgeführt:

5.2.1) Szenario „Generalisierung von Daten durch Veränderung des Maßstabes“

In einem ersten Testszenario liegt der Schwerpunkt der Untersuchung bei der Generalisierung von Geodaten im Zuge der Datenbankabfrage. Als Datenbestand für den Test wird dabei ein freier Datensatz der Geodatensammlung *Natural Earth* verwendet. Ausgewählt wurden dabei die Geodaten einer weltweiten. Der Datenbestand liegt im Koordinatensystem WGS84 (EPSG: 4326; siehe EPSG Datenbank[13]) vor. Da es sich um ein geographisches Koordinatensystem handelt, sind die Koordinatenangaben in geografischen Längen und geografischen Breiten.

Der Ausgangsdatenbestand liegt in Form einer Polygonfeatureklasse als Shape-Datei vor. Da der umgesetzte Client vorerst nur mit Linienzügen hantieren kann, musste der Ausgangsdatenbestand aufbereitet werden. Die ursprünglichen 521 Polygone wurden mit der Feature Manipulation Engine[14] (FME) zu 3994 Polylinien bzw. 532696 Einzelsegmenten aufgebrochen. Das Ergebnis einer Abfrage ist somit eine Menge von Linienzügen, die dann anschließend im Datenviewer angezeigt werden. Da es sich hierbei nur um die Anzeige und nicht um die Bearbeitung von Geodaten am Client handelt, reicht das Vorliegen von Linienzügen für diesen Testfall aus. Müssen im Ergebnis die Polygone bekannt sein, so muss entweder am Server oder anschließend am Client ein Zwischenschritt integriert werden, um die Linienzüge/-segmente zu Polygonen zu verschmelzen. Dieser Arbeitsschritt sollte jedoch recht einfach durchzuführen sein, da zu jedem Linienzug das linke und rechte Polygon bekannt ist.

Neben der Abfrage mit Hilfe des beschriebenen Konzepts, werden 2 weitere Abfragevarianten getestet, wobei zum einen die Polygone als Linienzüge gespeichert werden und zum anderen als einzelne Liniensegmente. In beiden Varianten handelt es sich um eine einfache, flache Linientabelle mit einer Geometriespalte in der Tabelle.

[13] http://www.epsg.org/
[14] http://www.safe.com/

Die Geometrie wird in Oracle für gewöhnlich mit Hilfe des *User Defined Datatype SDO_GEOMETRY* gespeichert. Es handelt sich dabei um einen speziellen Datentyp, welcher bei der Definition einer Tabelle verwendet werden kann. Im ersten Vergleichsfall werden ähnlich dem umgesetzten Prototyp die Linienzüge zwischen zwei Knoten gespeichert und im zweiten Vergleichsfall jedes Liniensegment als eigener Datensatz in der Linientabelle. Als Unterschied zum Prototyp werden keine weiteren Topologieinformationen zwischen den einzelnen Featureobjekten gespeichert. Da Ausgangsdaten oftmals in der unterschiedlichsten Form gespeichert sind, soll hier der Vergleich mit dem Viewing-Service angestellt werden. Für beide Vergleichsfälle wurden die Ausgangspolygone mit einer FME Workbench gesplittet, sodass man separate Linienzüge bzw. im zweiten Fall einzelne Liniensegmente erhält. Diese wurden anschließend mit FME direkt in eine neue Linientabelle in der Oracle Datenbank abgelegt.

Für jede der 3 Abfragevarianten werden dabei die folgenden Kennwerte ermittelt:

- Dauer für die Durchführung der Abfrage
- Dauer für die Durchführung der Abfrage inkl. Darstellung des Abfrageergebnisses im Datenviewer.
- Anzahl der Linienzüge im Abfrageergebnis
- Anzahl der Liniensegmente, aus welchen die Linienzüge bestehen
- Netzwerkverkehr
- Benötigter physikalischer Speicherplatz am Datenbankserver

Thesen

- Durch die Verwendung des Viewing-Service wird gegenüber den beiden anderen Umsetzungsvarianten eine kürzere Dauer für die Abfrage und Anzeige der Geodaten erreicht. Dies wird durch die Vereinfachung der Linienzüge erreicht.
- Durch die Verwendung des Prototyps ist die Netzwerkauslastung wesentlich geringer als bei den beiden anderen Varianten
- Die Umsetzung des Konzepts benötigt die Ablage der Topologie als Zusatz zur Geometrieinformation. Durch die Anreicherung des Polygondatensatzes mit Topologieinformationen soll eine Verbesserung bei der Abfrage der Daten erreicht

werden. Diese zusätzlichen Informationen benötigen dabei nicht wesentlich mehr Speicherplatz in der Datenbank.

Testdurchführung und Auswertung

Der gesamte Datenbestand liegt im Koordinatensystem WGS84 vor. Da es sich um einen weltweiten Staatendatensatz handelt, ist die Ausdehnung nahezu die gesamte Ausdehnung des Koordinatensystems: (-180 lon; 83.63 lat) - (180 lon; -90 lat).

In einen ersten Schritt wird die maximale Ausdehnung des Datenbestandes abgefragt. Das Abfragerechteck wird anschließend immer um 10 % in beiden Richtungen verkleinert, wobei das Zentrum des Abfragerechtecks gleich bleibt. Somit ergeben sich die folgende Abfragebereiche:

Schritt	Abfragerechteck in geographischen Längen und Breiten
1	(-177.39° lon; 83.63° lat) - (179.91° lon; -90° lat) = gesamte Ausdehnung
2	(-159.52° lon; 74.95° lat) - (162.04° lon; -81.32° lat)
3	(-127.37° lon; 59.33° lat) - (129.89° lon; -65.70° lat)
4	(-88.78° lon; 40.57° lat) - (91.30° lon; -46.94° lat)
5	(-52.76° lon; 23.07° lat) - (55.28° lon; -29.44° lat)
6	(-25.75° lon; 9.94° lat) - (28.27° lon; -16.31° lat)
7	(-9.55° lon; 2.07° lat) - (12.06° lon; -8.43° lat)

Tabelle 5-1: Abfragefenster für Szenario 1

Bei der Untersuchung wird ein SVO-Kriterium von 0.1% der Gesamtausdehnung verwendet. Dargestellt wird das Ergebnis in allen Fällen in einem Datenviewer unter Verwendung einer Rektangularprojektion. In der Literatur findet man diese auch unter der Bezeichnung Plattkarte (frz. Plate Carrée, vgl. Hake, Grünreich & Meng 2002). Diese Projektion ist für den Einsatz in heutigen GIS-Systemen sehr wichtig und wird daher aufgrund ihrer Einfachheit sehr oft verwendet um geographische Koordinaten darzustellen. Geographische Koordinaten werden direkt ohne Transformation in die Karte übertragen und können direkt von dieser abgelesen werden. Zu beachten ist jedoch, dass Längentreue nur entlang der Längskreise und der Schnittkreise vorherrscht. Die beiden Pole werden dabei mit gleicher Länge, wie der Äquator dargestellt. Für Längenberechnungen ist zu berücksichtigen, dass es sich hierbei um geographische Koordinaten han-

delt und dabei die Orthodrome, die Länge auf dem Großkreis, berechnet werden muss (vgl. Hake, Grünreich & Meng 2002).

Nr.	Maßstab	Viewing-Service			Ergebnis als Polylinien		Ergebnis als Einzelsegmente	
		# Polylinien	# Segmente	SVO (in m)	# Polylinien	# Segmente	# Polylinien	# Segmente
1	1:105929285	3783	9235	38459	3994	529026	532696	532696
2	1:105929285	3714	9404	38458	3699	511075	504944	504944
3	1:102360614	3388	8986	37163	2551	419806	366314	366314
4	1:55462249	1636	4498	20136	1038	190505	154290	154290
5	1:39945658	988	2805	14502	177	77078	59490	59490
6	1:17214333	232	1120	6249	31	25118	16132	16132
7	1:10213019	8	183	3707	7	3126	990	990

Abbildung 5-1: Vergleich der 3 Varianten bei unterschiedlichen Abfragebereichen

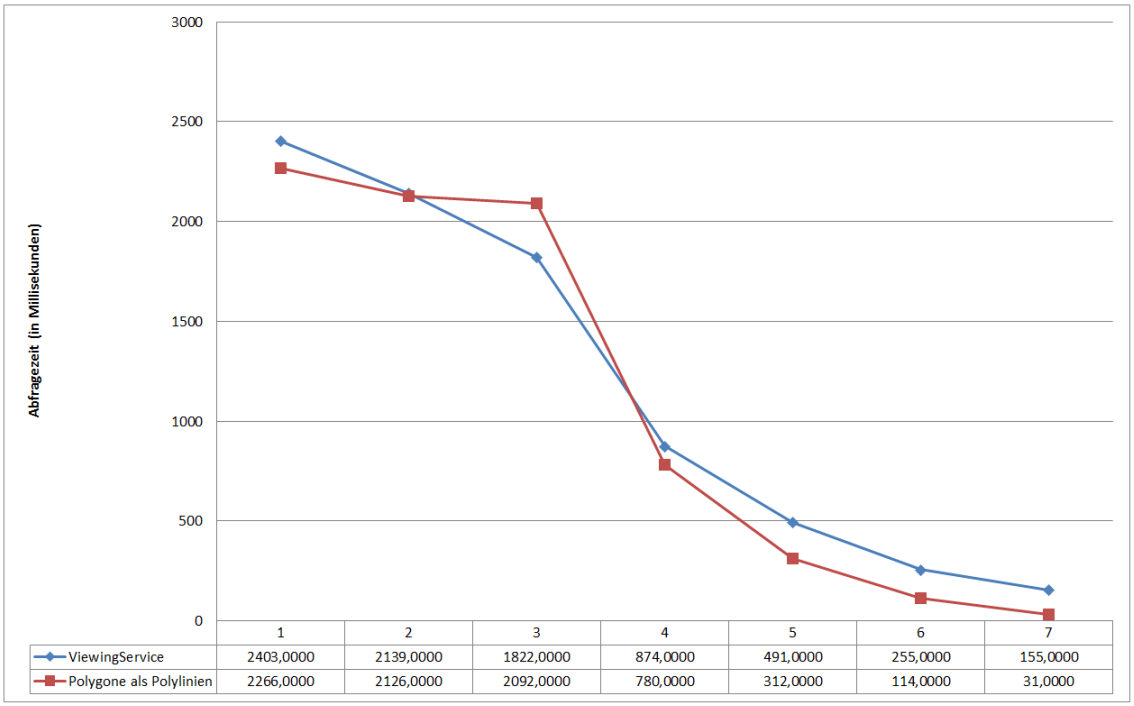
Wenn man nun die Anzahl der Features vergleicht, welche bei den einzelnen Abfragefenstern übertragen werden, sieht man sehr deutlich die Unterschiede der einzelnen Ansätze. Die Anzahl der übertragenen Polylinien ist bei den ersten beiden Varianten ähnlich. Unterschiede ergeben sich hier nur, da die Polylinien durch unterschiedliche Aufbereitungsmethoden erstellt wurden. Außerdem werden bei der Verwendung des Viewing-Service zu kleine Polygone aus dem Datenbestand entfernt, welche jedoch beim zweiten Ansatz enthalten sind. Bei der dritten Variante werden keine Polylinien gespeichert. Jedes Segment ist einzeln als Geometrie in der Tabelle abgelegt. Daher fällt hier der Unterschied zu den beiden ersten Varianten sehr deutlich aus.

Viel interessanter ist jedoch die Anzahl der Segmente, welche schlussendlich an den Client geschickt und im Datenviewer dargestellt werden müssen. Das Viewing-Service reduziert den Ausgangsdatenbestand anhand des SVO-Kriteriums. Somit müssen am Client wesentlich weniger Geometrieobjekte angezeigt werden. In Variante 2 gibt es keine Datenreduktion durch Generalisierung. Die Segmentanzahl ist demnach um einiges höher als beim Viewing-Service. Bei der dritten Variante gibt es keine Veränderung, da ohnehin nur Segmente gespeichert wurden.

	1	2	3	4	5	6	7
ViewingService	2403,0000	2139,0000	1822,0000	874,0000	491,0000	255,0000	155,0000
Polygone als Polylinien	2266,0000	2126,0000	2092,0000	780,0000	312,0000	114,0000	31,0000
Polygone als Einzelsegmente	25259,5257	24214,4210	17439,7438	7506,0000	2195,0000	860,0000	119,0000

Abbildung 5-2: Abfragezeit der 3 Varianten im Vergleich

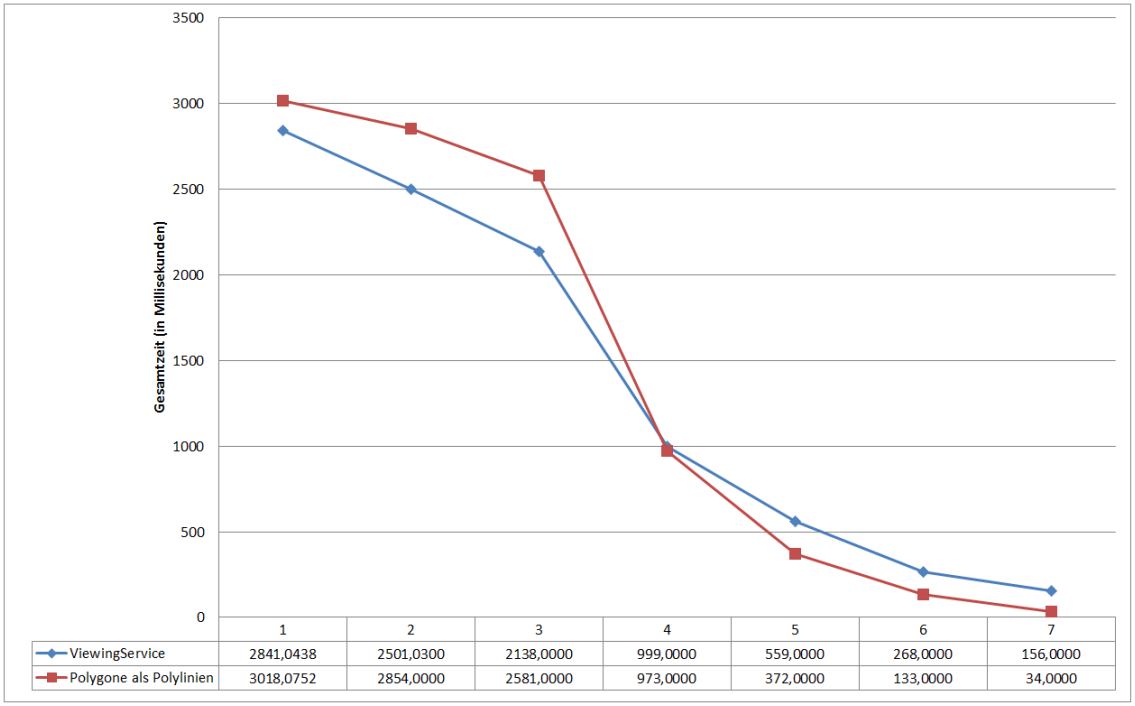
Wenn man nun die Abfragezeit betrachtet erkennt man, dass die Abfrage über das Viewing-Service bzw. über Variante 2 „Polygone als Polylinien“ in etwa gleich lang dauert. Die Abfragezeit startet beim ersten Schritt bei 2.5 Sekunden und wird bei verkleinerten Abfragefenstern gleichmäßig weniger. Hingegen bei Variante 3 „Polygone als Einzelsegmente“ zeigt sich der negative Effekt der großen Objektanzahl aufgrund der ineffizienten Datenhaltung.



	1	2	3	4	5	6	7
ViewingService	2403,0000	2139,0000	1822,0000	874,0000	491,0000	255,0000	155,0000
Polygone als Polylinien	2266,0000	2126,0000	2092,0000	780,0000	312,0000	114,0000	31,0000

Abbildung 5-3: Abfragezeit Viewing-Service + Variante 2

Betrachtet man nur das Viewing-Service und die Variante 2 separat so erkennt man, dass die Abfragezeit der Variante 2 bis auf einen Ausreißer etwas effizienter ist als jene des Viewing-Service. Dies würde der ersten formulierten These widersprechen, jedoch wurde hier noch nicht die Zeit berücksichtigt, welche für die Anzeige der Features benötigt wird.

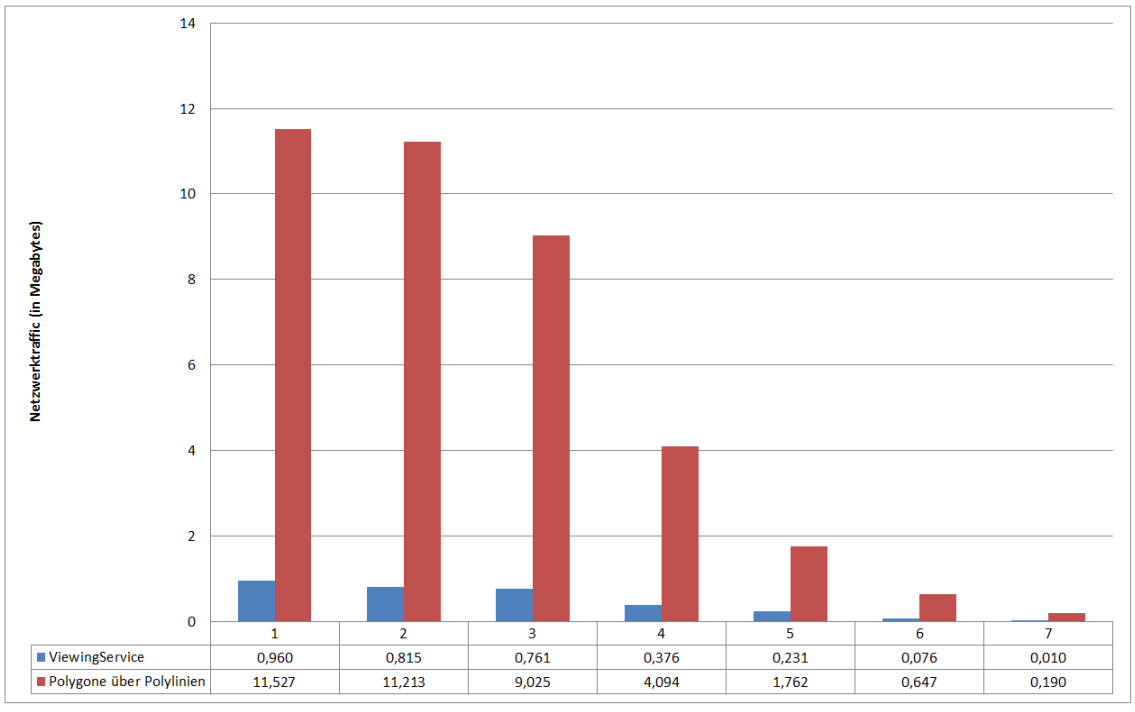


	1	2	3	4	5	6	7
ViewingService	2841,0438	2501,0300	2138,0000	999,0000	559,0000	268,0000	156,0000
Polygone als Polylinien	3018,0752	2854,0000	2581,0000	973,0000	372,0000	133,0000	34,0000

Abbildung 5-4: Gesamtzeit Viewing-Service und Variante 2

Bei der Betrachtung der Gesamtzeit (Abfragezeit + Darstellungszeit) sieht das Ergebnis besser zugunsten des Viewing-Service aus. Da am Client wesentlich weniger Segmente dargestellt werden müssen, liegt die Gesamtzeit für große Ausschnitte unter jener der zweiten Variante. Ab dem 4. Schritt ist das Übertragen einfacher Polylinien ohne Topologieinformation wieder effizienter als der Generalisierungsansatz.
Anzumerken ist hierbei, dass der verwendete Datenviewer besonders effizient bei der Verarbeitung von Polylinien ist. Die Stützpunktanzahl wirkt sich nur sehr wenig auf die Darstellungszeit aus. In einem anderen Datenviewer würde der Vergleich der Gesamtzeit wohl besser zugunsten des Viewing-Service ausfallen. Variante 3 wurde bei der Betrachtung der Gesamtzeit außen vor gelassen, da aufgrund der viel größeren Segmentanzahl und der bereits sehr großen Abfragezeit kein Vergleich zum Viewing-Service sinnvoll ist.

Es ist anzunehmen, dass der Netzwerkverkehr mit dem Oracle Server, welcher aufgrund einer Abfrage verursacht wird, mit der Anzahl der zu übertragenden Objekte korreliert. Für die ersten beiden Varianten wurde dies für die 7 Abfragebereiche dokumentiert.



	1	2	3	4	5	6	7
ViewingService	0,960	0,815	0,761	0,376	0,231	0,076	0,010
Polygone über Polylinien	11,527	11,213	9,025	4,094	1,762	0,647	0,190

Abbildung 5-5: Vergleich Netzwerkverkehr

Die gemessenen Werte bestätigen die Vermutung, dass mittels des Viewing-Service aufgrund geringerer Features um einiges an zu übertragenden Megabytes eingespart

werden kann. Für das Viewing-Service müssen lediglich 7-13% des Datenumfangs der Variante 2 übertragen werden.

In einem letzten Vergleich soll der benötigte physikalische Speicherplatz am Datenbankserver gegenübergestellt werden. Dabei wird der Speicherplatz aller zur Umsetzung der jeweiligen Variante benötigten Datenbankelemente (Tabellen, Indizes, Nested Tables, LOB Segmente, …) summiert und verglichen.

	Viewing-Service	Polygone aus Polylinien	Polygone aus Einzelsegmenten
Speicherverbrauch	54.125 MB	14.6875 MB	141.375 MB

Abbildung 5-6: Physikalischer Speicherverbrauch im Vergleich

Die These bzgl. des physikalischen Speicherverbrauchs kann aufgrund der gemessenen Werte nicht ganz gehalten werden. Zur Ablage der topologischen Beziehungen als Zusatzinformation zur Geometrie und den sich daraus ergebenden Speicherverbrauch durch Verwaltungsstrukturen, muss für den getesteten Datensatz etwas mehr als das 3.5fache des Speicherverbrauchs aus Variante 2 reserviert werden. Das Viewing-Service benötigt jedoch nur ungefähr ein Drittel des Speicherplatzes der Variante 3, in welcher jedes Segment einzeln als Geometrie abgelegt wurde.

Zusammenfassend konnten aufgrund der durchgeführten Tests die Thesen zur Dauer einer Abfrage bzw. zum Datentransfer übers Netzwerk belegt werden. Die Vorteile bei Abfragedauer und Datentransfer konnten nur durch Anreicherung von zusätzlichen Informationen am Server erreicht werden. Die zusätzlichen Informationen benötigen dementsprechend Speicherplatz am Datenbankserver. Der dafür notwendige Mehrbedarf hält sich allerdings in Grenzen.

5.2.2) Szenario „Maßstabsabhängige Filterung ganzer Featureklassen inkl. Generalisierung“

Neben der „On-the-fly“ Generalisierung von Geodaten ist die maßstabsabhängige Darstellung von Featureklassen eine zweite wesentliche Möglichkeit, um die Datenmenge zu reduzieren. Geodaten mit Detailinformation (z.B. Detailinformation zu Leitungen oder Kabeln) sind nur bis zu einem konfigurierten Zoommaßstab von Interesse. Navigiert der Anwender mit dem Datenviewer aus dem Datenbestand, können die Objekte der Detailfeatureklassen ignoriert werden. Neben dem Weglassen von Featureklassen soll aufgrund der Generalisierung von Daten bei einem Wechsel auf einen kleineren Maßstab eine derartige Datenreduktion erreicht werden, dass die Objektanzahl unter Berücksichtigung einer Schwankungsbreite konstant bleibt. Umgekehrt wird beim Wechsel auf einen größeren Maßstab zwar der Detailgrad der Anzeige erhöht, jedoch durch das verkleinerte Abfragefenster sollte sich die Objektanzahl nicht wesentlich vergrößern. Die Anzahl der Features, welche an den Client übertragen werden, ist von 3 Faktoren abhängig:

- Größe des Abfragebereiches
- Größe des SVO-Kriteriums
- Konfigurierte Maßstabsbereiche

Über die letzten beiden Faktoren konfiguriert ein GIS-Administrator, wie detailliert und wie effizient die Übertragung und Darstellung am Client sein soll. Je nach Fachdomäne bzw. konkreten Anforderungen an ein Vorhaben, muss der GIS-Administrator eine effiziente Konfiguration erzeugen.

Als Datensatz werden für die folgende Untersuchung die Staats-/Landes- und Bezirksgrenzen von Österreich und die Gemeinden Oberösterreichs herangezogen. Die Daten stammen vom freien Datensatz der *Global Administrative Areas*[15] und vom *RIS des Landes Oberösterreich*[16] , wobei hier ebenfalls die Geodaten im Koordinatensystem WGS84 (EPSG: 4326) vorliegen.

[15] http://www.gadm.org/
[16] http://doris.ooe.gv.at/

Thesen

- Aufgrund der maßstabsabhängigen Anzeige können größere Datenmengen in der Datenbank verwaltet und bei großen Maßstäben ohne eine längere Abfragezeit angezeigt werden.
- Der Wechsel des Maßstabes bewirkt eine Vergrößerung bzw. Verkleinerung der Karte. Aufgrund des SVO-Kriteriums werden Objekte generalisiert dargestellt bzw. generell aus dem Datenbestand entfernt. Durch die Generalisierung, der Reduktion und des veränderten Anzeigeausschnitts, bleibt die Objektanzahl in etwa konstant bzw. verringert sich beim Wechsel auf sehr große Darstellungs-maßstäbe, trotz der Darstellung detaillierterer Features.

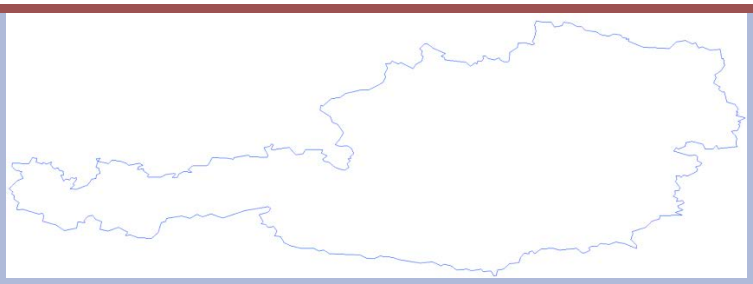
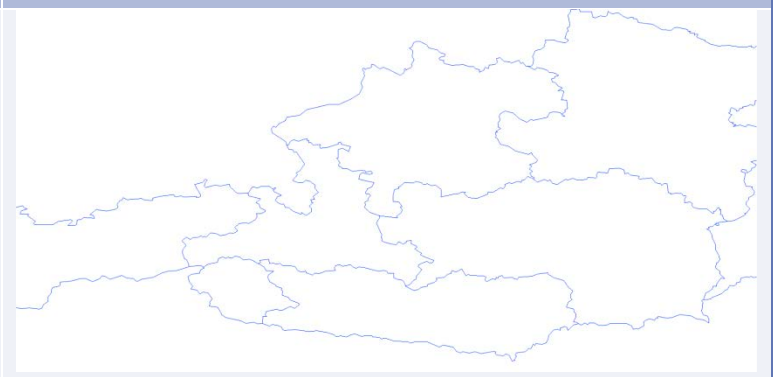
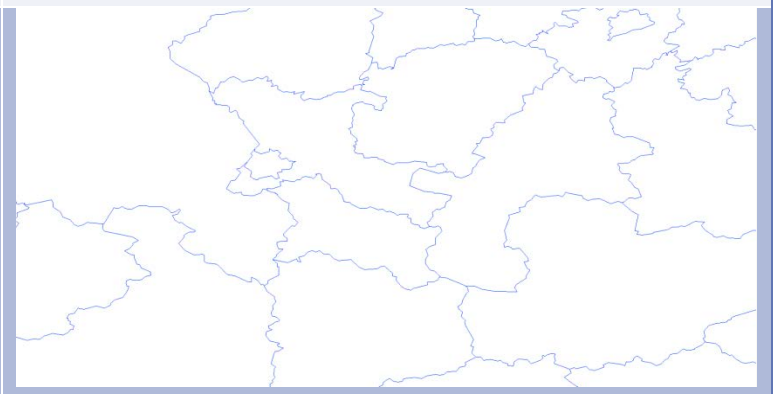
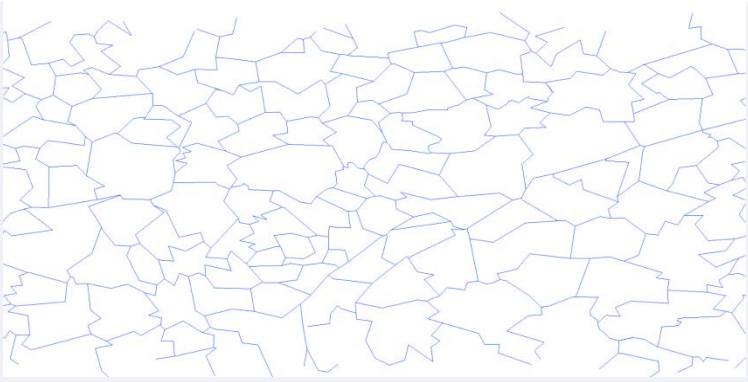
Testdurchführung und Auswertung

In einem ersten Test wird der Einfluss der maßstabsabhängigen Filterung auf die Dauer einer Abfrage untersucht. Dabei werden für die Polygonfeatureklassen die folgenden Maßstabsbereiche in der Datenbank konfiguriert:

Featureklasse	Sichtbarkeitsbereich	#Features	#Segmente
Österreich (Staatsgrenzen)	1500000 - unendlich	1	4175
Bundesländer (Staatsgrenzen + Landesgrenzen)	700000 – 1500000	9	10559
Bezirke (Staatsgrenzen + Landesgrenzen + Bezirks-grenzen)	200000 – 700000	99	31485
Gemeindegrenzen OÖ (Quelle DORIS, Land OÖ)	0 - 200000	2285	4312
Summe	-	**2394**	**50531**

Tabelle 5-2: Sichtbarkeitsbereiche für Szenario 2

Je nach aktuellem Darstellungsmaßstab wird nur ein Teilbereich des Datenbestandes aus der Datenbank selektiert. Für den Test wird für jeden Sichtbarkeitsbereich ein passender Darstellungsmaßstab gewählt und das Ergebnis dokumentiert:

Maßstab	# Segmente	Zeit (in ms)	Abfrageergebnis
1:1745039	390	305	
1:872654	1354	739	
1:289486	1541	402	
1:109792	988	428	

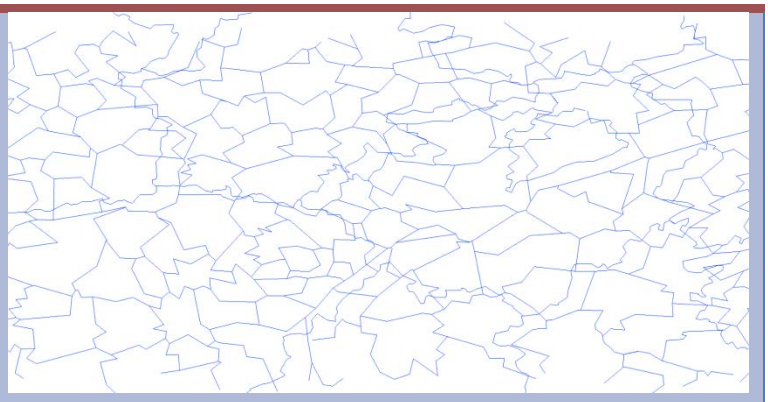
1: 109792 mit allen Feature-klassen	2792	2631	

Tabelle 5-3: Maßstabsabhängige Darstellung von Featureklassen

Anm.: Der verwendete Darstellungsmaßstab ergibt sich aufgrund der Größe des dargestellten Ausschnitts in Pixel. Entsprechend der dargestellten Pixel und des PPI-Wertes (Pixel per Inch) des Monitors wird der Maßstab für den aktuellen Ausschnitt errechnet.

Die Features, welche aufgrund des aktuellen Maßstabs nicht sichtbar sind, haben keinen Einfluss auf die Dauer der Abfrage. Da sich bei einem Wechsel auf einen kleineren Maßstab auch der sichtbare Bereich verkleinert, können mehr Features selektiert und dargestellt werden. Durch die Vorselektion der sichtbaren Featureklassen können größere Datenausschnitte bei sehr kleinen Maßstabsbereichen dargestellt werden ohne dass sich eine merkliche Veränderung bei der Abfragedauer widerspiegelt. Dies sieht man sehr deutlich in der letzten Zeile der obigen Tabelle. Hier wurden keine Maßstabsbereiche konfiguriert. Vom Viewing-Service müssen alle Geometrieobjekte der Featureklassen berücksichtigt werden. Die Objektanzahl und die Abfragedauer ist dementsprechend höher als bei den vorigen Schritten.

Anm.: Die Gemeindegrenzen aus Oberösterreich sind aus einem anderen Datenbestand, als die restlichen Verwaltungsgrenzen. Die Daten liegen daher in einer unterschiedlichen Generalisierungsstufe vor. Da im Testfall der Vergleich der Objektanzahl bzw. der Abfragedauer im Vordergrund stand, wurde auf eine Harmonisierung von Datenbeständen verzichtet.

In einem zweiten Test wird die Auswirkung der Generalisierung auf die Anzahl der zu selektierenden Features betrachtet. Dabei werden die Bezirke Österreichs mit unterschiedlichen Abfragefenstern aus der Datenbank selektiert und im Datenviewer angezeigt. Aufgrund eines veränderten Abfragefensters ergibt sich ein neuer Darstellungsmaßstab und somit zusätzlich unter Berücksichtigung des Maßstabes ein neues SVO-

Kriterium. Wird der Darstellungsmaßstab kleiner, so wird ein größerer Bereich abgefragt. Dies wiederum ergibt eine größere Objektanzahl, welche an den Client transportiert werden muss. Der Zuschuss an Objekten soll durch die Generalisierung der Features ausgeglichen werden.

Die vorliegenden Bezirkspolygone haben das Koordinatensystem WGS84 (EPGS: 4326). Ausgangsbasis für die Untersuchung ist die räumliche Ausdehnung der Polygonfeatureklasse: (9.53 lon ;49.02 lat) (17.16 lon; 46.37 lat). Die Ausdehnung wird dabei immer um 10% in beiden Richtungen ausgeweitet, was eine Verkleinerung des Maßstabes bewirkt und somit ein größerer Datenausschnitt sichtbar wird.

Der Ausgangsdatenbestand beinhaltet ohne Linienvereinfachung 31485 Einzelsegmente. Durch die Linienvereinfachung sollte sich die Segmentanzahl wesentlich reduzieren.

Nr.	Originalausdehnung	#Polylinien	#Segmente	Ladezeit (in ms)	Anzeigezeit (in ms)	SVO (in m)
1		599	3155	292	101	389
2	+ 10%	599	2907	236	42	439
3	+ 10%	599	2514	238	41	527
4	+ 10%	599	2011	229	40	685
5	+ 10%	599	1554	224	40	959
6	+ 10%	599	1118	230	40	1438
7	+ 10%	599	856	237	39	2300
8	+ 10%	599	704	250	23	3909
9	+ 10%	598	627	771	40	7034
10	+ 10%	594	608	594	40	13355

Tabelle 5-4: Auswirkung eines vergrößerten Abfragefensters auf die Objektanzahl und die Gesamtzeit

Aufgrund des kleineren Maßstabes und der daraus resultierenden Änderung des Smallest-Visible-Object ergibt sich die Generalisierung der Features. Wie man in der obigen Tabelle sehr gut erkennen kann, bleibt zwar die Polylinienanzahl gleich, jedoch müssen wesentlich weniger Segmente dargestellt werden. Aufgrund der geringeren Objektanzahl bleibt die Abfragezeit innerhalb einer gewissen Schwankungsbreite.

Wird der Darstellungsmaßstab vergrößert, so wird nun ein kleinerer Ausschnitt des Datenbestandes selektiert. Durch die daraus resultierende Verkleinerung des Smallest-Visible-Objects müssen mehr Details dargestellt werden. Dies soll durch den verkleinerten Darstellungsausschnitt kompensiert werden. Den Einfluss der veränderten Ausgangssituation (kleinerer Ausschnitt, kleinerer SVO-Wert) kann aufgrund der folgenden Tabelle nachvollzogen werden:

Nr.	Originalausdehnung	#Polylinien	#Segmente	Ladezeit (in ms)	Anzeigezeit (in ms)	SVO (in m)
1		599	3155	292	101	389
2	- 10%	588	3436	271	42	359
3	- 10%	466	3291	268	36	287
4	- 10%	226	2207	165	14	201
5	- 10%	81	1289	132	5	121
6	- 10%	26	710	73	1	60
7	- 10%	8	432	85	1	24

Tabelle 5-5: Auswirkungen eines verkleinerten Abfragefensters auf die Objektanzahl und die Gesamtzeit

Die Anzahl der zu ladenden Segmente bleibt bei den ersten beiden Testschritten noch recht hoch. Anschließend nimmt die Anzahl stark mit der Einschränkung des Abfragebereiches ab, was sich in der verkürzten Gesamtzeit niederschlägt. Trotz der Darstellung von detaillierteren Features wird aufgrund der geringeren Objektanzahl eine kürzere Gesamtzeit erreicht.

5.2.3) Szenario „Wahl des SVO-Kriteriums“

In den beiden obigen Tests wurde das Smallest-Visible-Object als Anteil der horizontalen Fensterausdehnung festgelegt. Wie bereits in Kapitel 3.2.4 beschrieben, kann das Smallest-Visible-Object als Anteil der Länge des dargestellten Bereiches in Modelleinheiten:

$$SVO = Länge\ in\ Modelleinheiten * \boldsymbol{Anteil}/100$$

oder als Anteil der Länge des dargestellten Bereichs in Bildschirmeinheiten (Millimeter oder Pixel):

$$SVO = Länge\ in\ Bildschirmeinheiten * \boldsymbol{Anteil}/100$$

angegeben sein.

Die Wahl des Anteils an der Ausdehnung ist vom GIS-Administrator frei definierbar. Je größer dieser Anteil ist, umso größer ist der Grad der Generalisierung.

Thesen

- Zu starke Generalisierung verändert das entstehende Bild derartig, dass ein Betrachter dieses als unnatürlich, ungenau oder auch falsch einstuft.
- Das SVO-Kriterium ist von der Ausdehnung des Grafikfensters in Pixel/Millimeter oder der Ausdehnung in Modelleinheiten abhängig. Die Höhe des Anteils ist dabei abhängig vom verwendeten Koordinatensystem.

Testdurchführung und Auswertung

Zur Belegung der ersten These werden die Bezirke Österreichs mit unterschiedlichem SVO-Kriterium dargestellt:

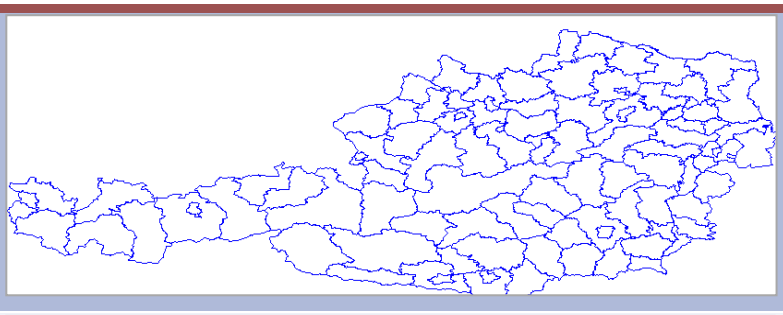
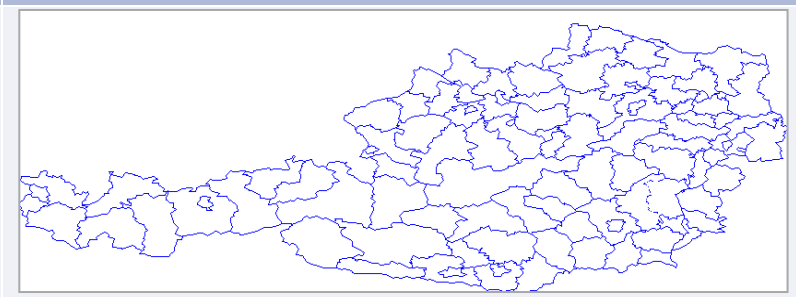
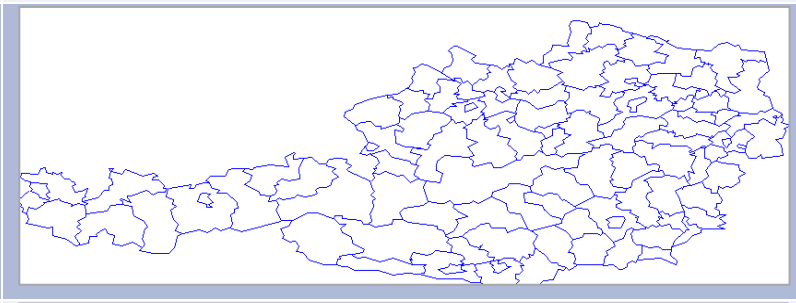
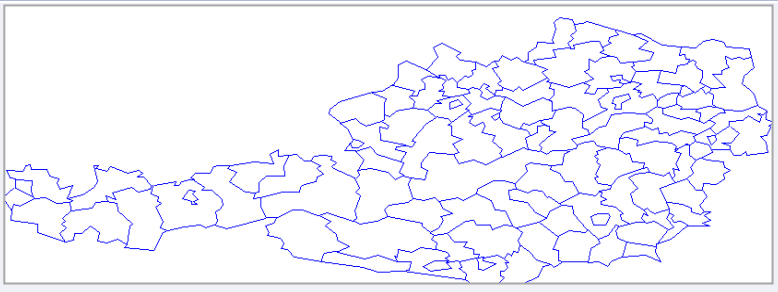
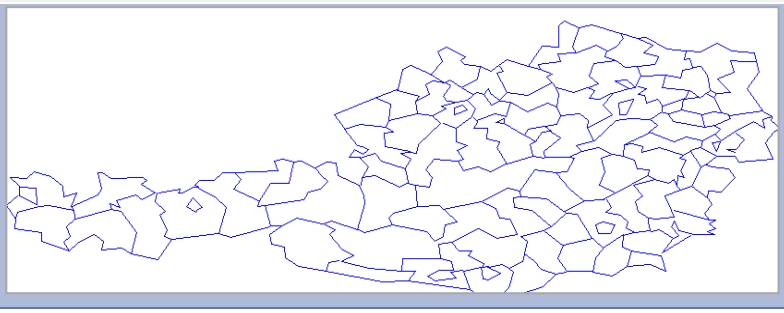
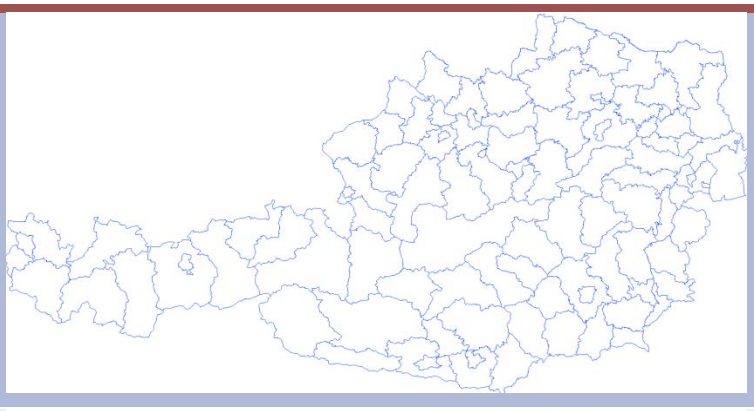
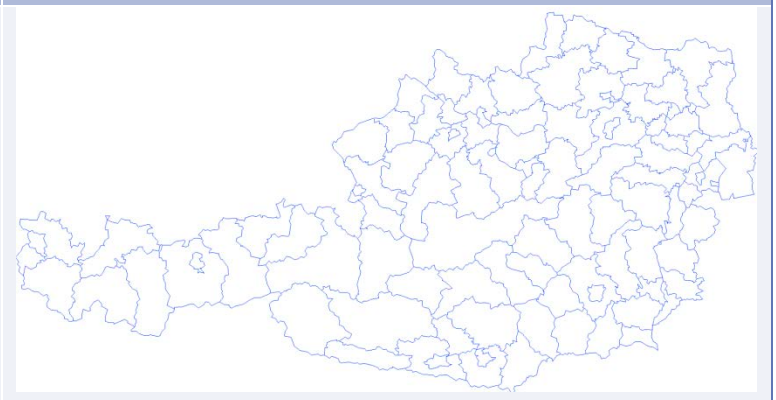
Anteil	#Segmente	Ergebnis
0.0% = keine Gene-ralisie-rung	31485	
0.1%	2915	
0.2%	1510	
0.3%	1029	
0.5%	641	

Tabelle 5-6: Vergleich von Abfragen mit unterschiedlichem SVO Kriterium (Koordinatensystem: EPGS:4326)

Je größer der gewählte Anteil, desto größer ist das SVO-Kriterium und desto größer ist der Grad der Generalisierung. In der obigen Abbildung erkennt man die sinkende Segmentanzahl bei einem gesteigerten SVO-Kriterium. Anteilswerte von 0.1-0.3% wirken dabei auf den Betrachter als natürlich. Unterschiede zum Originalbild lassen sich bei genauerer Betrachtung bei einem Anteil von 0.2-0.3% erkennen. Hingegen bei einem Anteil von 0.5% wirkt das Abbild kantig und unnatürlich auf den Betrachter. Für den aktuellen Darstellungsmaßstab, in diesem Fall 1:2.724.794, ist der Wert des SVO zu hoch gewählt. Für die Anzeige dieses Datenbestandes ist ein Anteil von 0.1% für die Bestimmung des SVO Wertes günstig. Anzumerken ist hierbei, dass die dargestellte Polygonfeatureklasse das Koordinatensystem WGS84 hat.

Kann für ein anderes Koordinatensystem derselbe Anteil für das SVO verwendet werden, oder muss hier ein anderer Anteil berücksichtigt werden? Als Vergleich wird die Featureklasse mit FME nach Lambert (EPSG: 31287) transformiert und die Ergebnisbilder nochmals erzeugt.

Anteil	#Segmente	Ergebnis
0.0%	31485	
0.1%	3541	

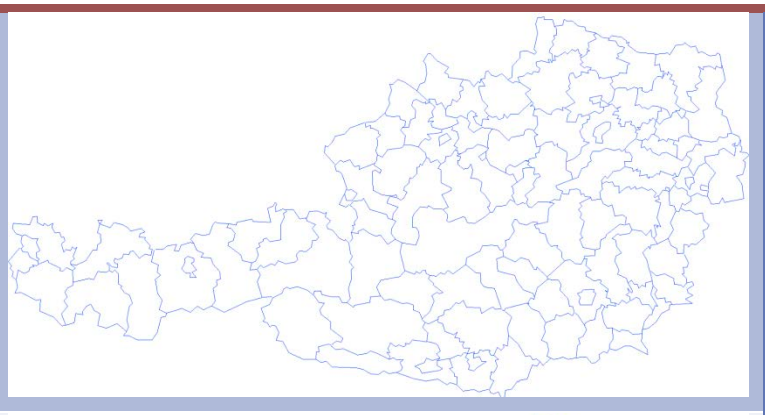
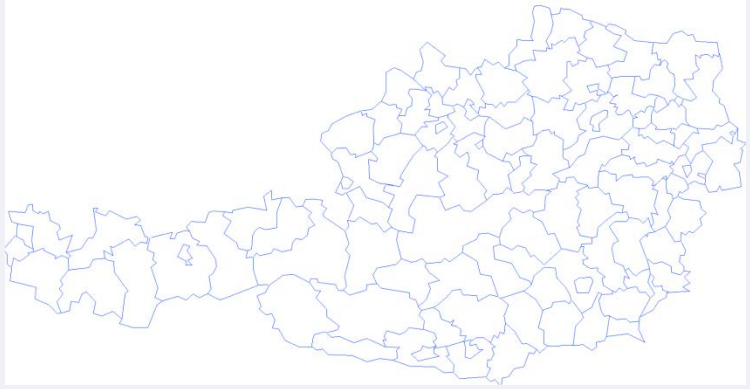
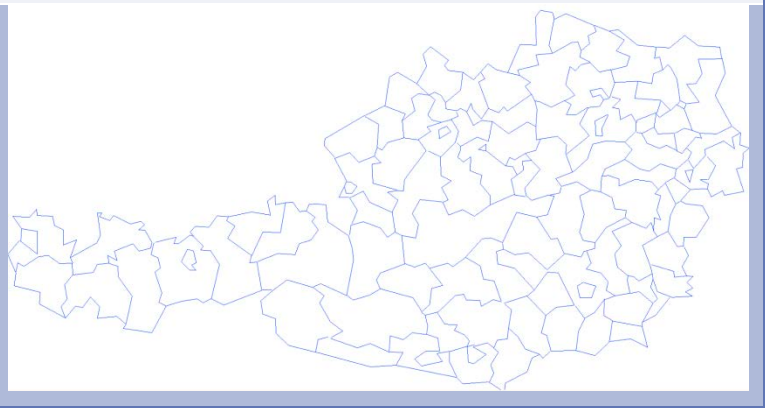
0.2%	1863	
0.3%	1275	
0.5%	758	

Tabelle 5-7: Vergleich von Abfragen mit unterschiedlichem SVO Kriterium (Koordinatensystem: EPGS:31287)

Man erkennt einen Unterschied in den gemessenen Werten und in den angezeigten Kartenausschnitten.

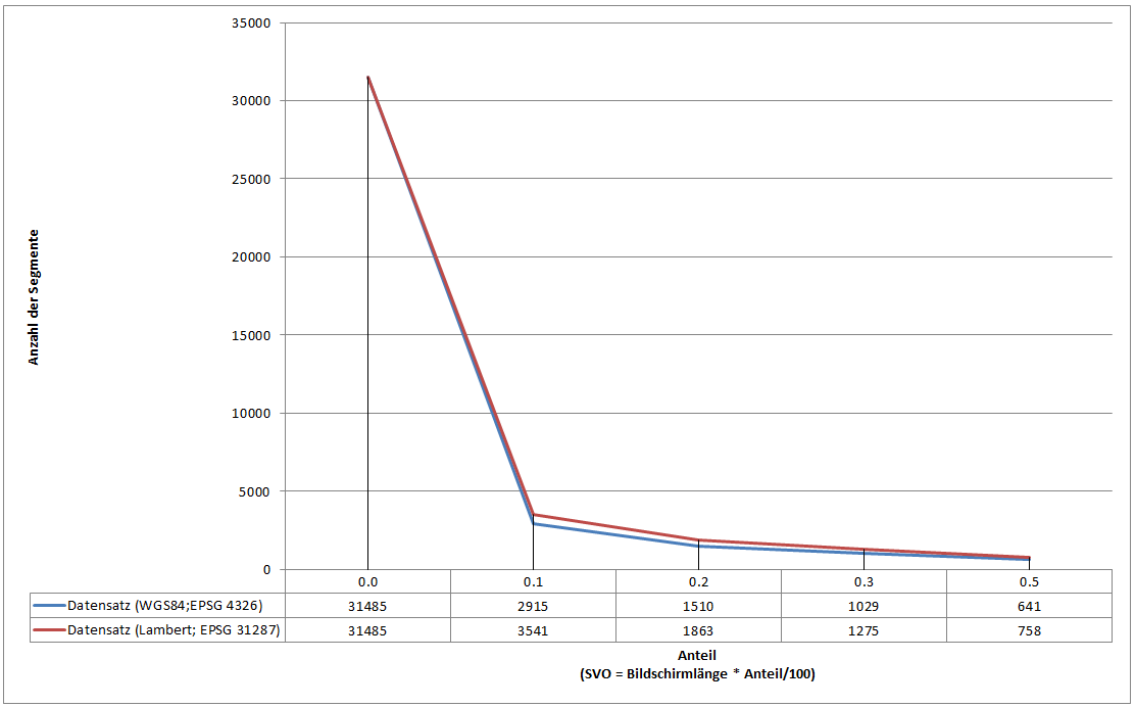


Abbildung 5-7: Anzahl der Segmente bei unterschiedlichem Koordinatensystem

Bei der Darstellung des Datensatzes in Lambert Austria ist aufgrund der Untersuchung ein Anteil von 0.2–0.3% empfehlenswert, wohingegen bei der Darstellung von WGS84-Koordinaten ein Wert von 0.1% ein sehr gutes Ergebnis liefert. Diese Unterschiede lassen sich aufgrund der Längen- und Flächenverzerrungen begründen, die jedem Projektionssystem unterworfen sind.

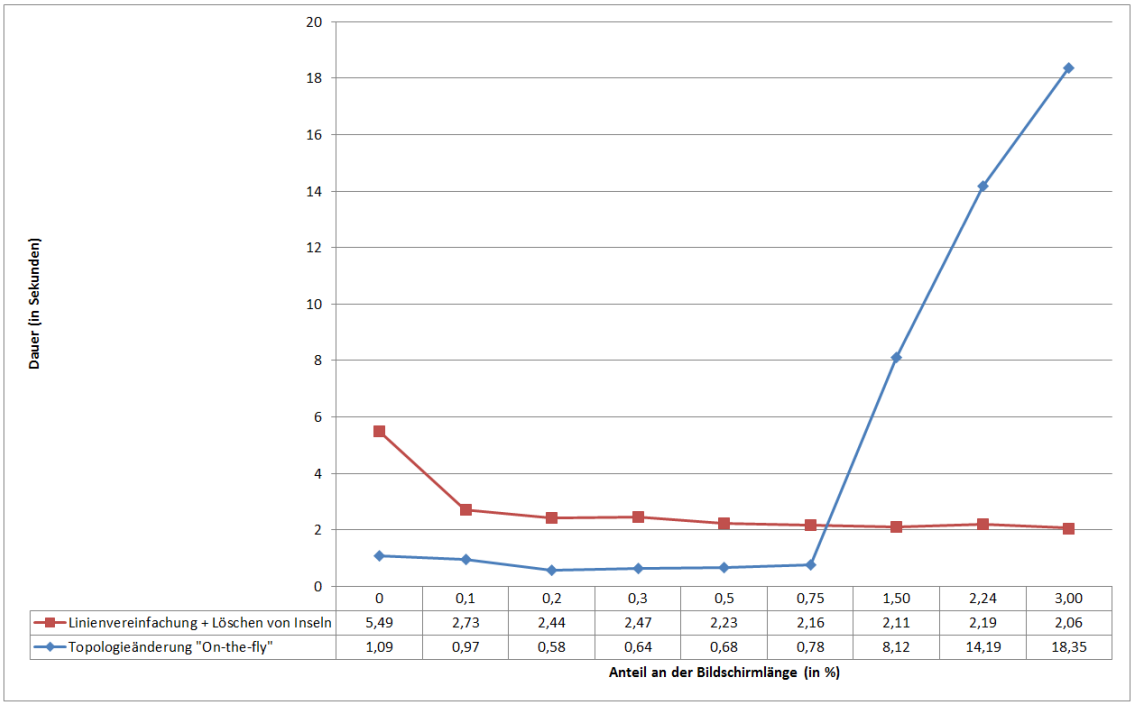
Das berücksichtigte SVO-Kriterium wird dabei aus der Fensterlänge ohne Berücksichtigung jeglicher Koordinatensystemverzerrungen abgeleitet. Da beim Koordinatensystem WGS84 eine größere Streckenverzerrung vorhanden ist, hat sich für den Lambert-Datensatz ein höherer Anteil zur Berechnung des SVO-Kriteriums als günstiger erwiesen.

Anm.: Das geographische Koordinatensystem WGS84 kann nur mittels einer Projektion auf der Ebene abgebildet werden. Durch eines der einfachsten Projektionssysteme wird das geographische Koordinatensystem in der Ebene bzw. auf dem Bildschirm abgebildet. Die geographischen Längen und Breiten werden als X- und Y-Werte im kartesischen System interpretiert und dargestellt. In der EPSG Datenbank des OGP Geometics Comittee gibt es dafür das Projektionssystem „WGS 84 / World Equidistant Cylindrical“ (EPSG: 32663).

5.3) Linienvereinfachung vs. Topologieänderungen

Durch das Weglassen von Featureklassen aufgrund von konfigurierten Maßstabsbereichen kann zum einen eine einfache und effiziente Datenreduktion erreicht und zum anderen durch die Auswertung der Maßstabsbereiche eine gute Performance erzielt werden. Neben der maßstabsabhängigen Filterung sind Linienvereinfachung durch Douglas-Peucker und „On-the-fly“ Topologieänderungen die beiden weiteren Methoden, um eine effizientere Übertragung zu gewährleisten.

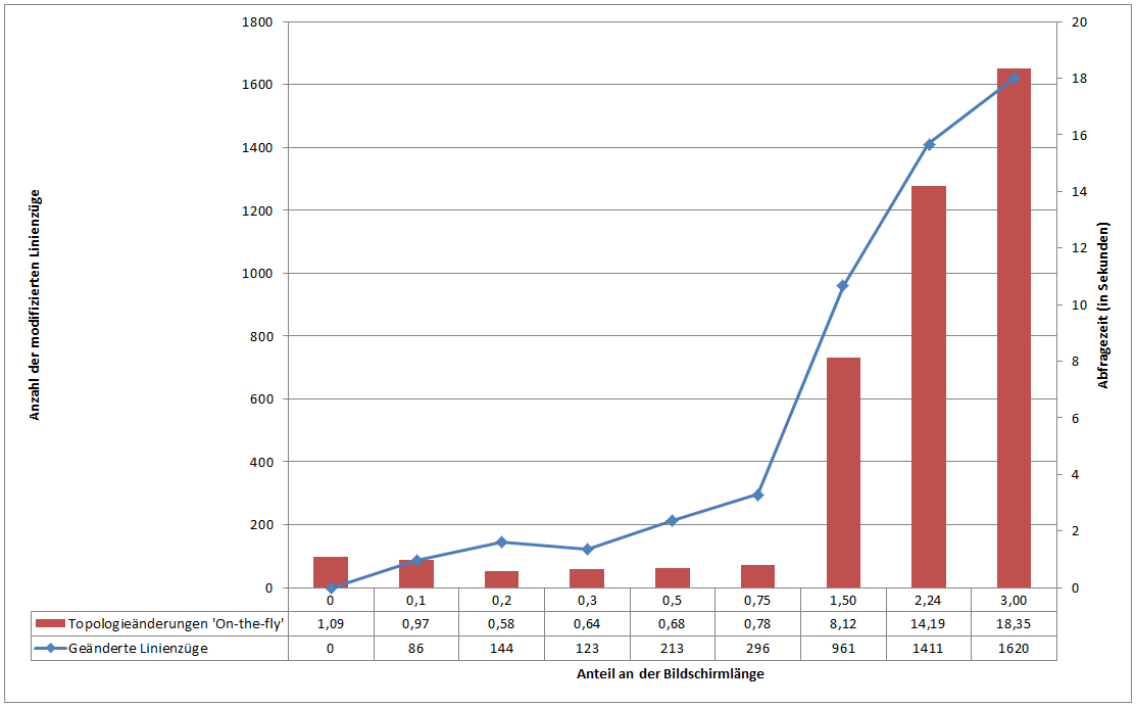
In der folgenden Untersuchung soll ein Vergleich der Linienvereinfachung mit den Topologieänderungen angestellt werden. Dazu wird der weltweite Staatendatensatz mit unterschiedlichen SVO Werten abgefragt. Hierbei wurde in der PL/SQL Prozedur die Zeit für die Linienvereinfachung und die Zeit der Topologieänderungen nach dem Entfernen eines Polygons separat protokolliert.
Inselflächen, welche als Ganzes entfernt werden können und keine Topologieänderungen nach sich ziehen, sind in der Zeit der Linienvereinfachung enthalten. Eine extra Protokollierung der Dauer würde hier das Ergebnis verfälschen. Die gemessenen Zeiten sind nicht mit den anderen Testszenarien vergleichbar, da diese über eine PL/SQL Testprozedur ausgewiesen wurden und der Overhead für die das Ausführen der Testprozedur nicht bekannt ist.



	0	0,1	0,2	0,3	0,5	0,75	1,50	2,24	3,00
Linienvereinfachung + Löschen von Inseln	5,49	2,73	2,44	2,47	2,23	2,16	2,11	2,19	2,06
Topologieänderung "On-the-fly"	1,09	0,97	0,58	0,64	0,68	0,78	8,12	14,19	18,35

Tabelle 5-8: Vergleich der Durchführungsdauer von Linienvereinfachungen und Topologieänderungen

Das obige Diagramm zeigt ein sehr interessantes Ergebnis. Die erste Messung wurde mit einem SVO-Kriterium gleich 0.0m erstellt, d.h., hier wurde keine Generalisierung des Datenbestandes durchgeführt. Die gemessene Gesamtzeit von rund 6.5 Sekunden entsteht durch den Overhead, welchen die Datenstrukturen und die durchlaufene PL/SQL Prozedur erzeugen. Wie bereits bei den Testszenarien beschrieben sind für die untersuchten Geodaten ein Anteil von 0.1-0.3% für die Berechnung des SVO-Kriteriums sinnvoll. In diesem Bereich sind sowohl die Linienvereinfachung (inkl. Eliminierung von Inselflächen) und die Topologieänderungen vom Zeitverhalten ähnlich. Die Linienvereinfachung beansprucht in etwa 2-3 Sekunden, wohingegen die Topologieänderungen unter einer Sekunde bleiben.



	0	0,1	0,2	0,3	0,5	0,75	1,50	2,24	3,00
Topologieänderungen 'On-the-fly'	1,09	0,97	0,58	0,64	0,68	0,78	8,12	14,19	18,35
Geänderte Linienzüge	0	86	144	123	213	296	961	1411	1620

Tabelle 5-9: Vergleich der Durchführungszeit von Topologieänderungen zur Anzahl der betroffenen Linienzüge

In der obigen Tabelle sieht man eine Gegenüberstellung der protokollierten Zeit der Topologieänderungen mit der Anzahl der modifizierten Linienzüge im Zuge der Topologieänderungen. Es ist gut zu erkennen, dass ab einem Anteil von 1-1,5 % wesentlich mehr Linienzüge modifiziert werden müssen, als bei kleineren SVO-Werten. Bei einem Bildschirm von 16.4 Zoll und einer Bildschirmlänge von 360 mm wäre dies ein SVO-Kriterium von 3.6-5.4 mm am Bildschirm. Bei größeren SVO-Werten ist die aktuelle Umsetzung im Prototyp nicht vorteilhaft. Die Abfragezeit steigt wesentlich schneller an, als die Steigung der zu ändernden Linienzüge. Wie man bereits bei den Testszenarien gesehen hat, können durch eine geschickte Kombination aus Abfragerechteck, konfigurierten Maßstabsbereich und Definition eines SVO-Kriteriums die notwendigen Topologieänderungen in Grenzen gehalten werden. Daher wurde dieser Anstieg der Ausführungszeit für Topologieänderungen im Zuge dieser Arbeit nicht weiter untersucht.

5.4) Problembereiche

Durch die Testläufe mit dem Prototyp konnten auch einige Schwachstellen identifiziert werden. Die Linienvereinfachung mit dem Douglas-Peucker Algorithmus erzielte sehr gute Ergebnisse. Im Zuge der Tests konnten jedoch 2 konkrete Fehlerfälle identifiziert werden:

- Überschneidungen bei geometrisch ungünstigen Situationen:

Abbildung 5-8: Fehler in der Linienvereinfachung aufgrund ungünstiger geometrischer Konstellationen

  Durch das Weglassen von Stützpunkten kann es zu Selbstschnitten kommen. Wenn für den aktuellen Darstellungsmaßstab der SVO-Wert zu groß ist, kann dies vom Betrachter visuell erkannt werden. Durch Verkleinern des SVO-Wertes verschwinden Selbstschnitte, welche von einem Betrachter erkannt werden können.

- Aktuell wird die Topologie in einer separaten Anwendung erzeugt und das Ergebnis der Ermittlung in das beschriebene Datenbankmodell übernommen. Die Topologie wurde für jede Polygonfeatureklasse gebildet, d.h., es gibt keine Informationen über topologische Beziehungen zwischen Featureklassen. Dies hat weitreichende Auswirkungen auf das gesamte Viewing-Service. Polygone werden im Prototyp nur über dessen geometrische Eigenschaften gegen das SVO-Kriterium geprüft und entfernt. Topologische Beziehungen zwischen Featureklasse spielen dabei keine Rolle. Des Weiteren verlaufen Linienzüge immer von Knoten zu Knoten innerhalb einer Polygonfeatureklasse. Auf Basis dieser Li-

nienzüge wird der Douglas-Peucker Algorithmus zur Vereinfachung des Linienzugs angewendet. Werden nun Linienzüge unterschiedlicher Polygonfeatureklassen im selben Maßstabsbereich generalisiert und angezeigt, kann es zu Fehlern in der Abbildung kommen, da die Linienzüge in jeder Featureklasse unterschiedliche Knoten haben können.

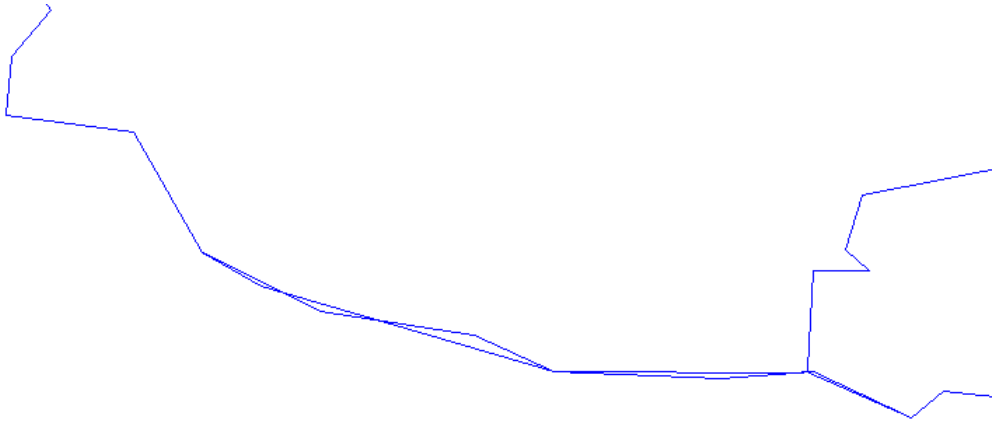


Abbildung 5-9: Fehler in der Linienvereinfachung aufgrund unterschiedlicher Knoten

Das Ergebnis sind Lücken im Datenbestand. Hier kann durch Modifikation am SVO-Kriterium das Ergebnis auf Kosten einer höheren Objektanzahl verbessert werden.

Ein weiterer Problembereich ist die stark ansteigende Durchführungszeit von „On-the-fly“ Topologieänderungen, wie dies in Kapitel 5.3 dokumentiert ist. Durch einen geringeren SVO-Wert, sprich 0.1-0.3 % der Bildschirmlänge, sollten sich Topologieänderungen in Grenzen halten, sofern nicht trotz der maßstabsabhängigen Sichtbarkeitsschaltung sehr viel Polygone vorhanden sind, welche das SVO-Kriterium nicht erfüllen. Bisher wurde dieser Anstieg der Ausführungszeit nicht näher untersucht, da anhand der untersuchten Datensätze (Administrationsgrenzen mit unterschiedlichem Detailgrad) ein höherer SVO-Wert ein zu schlechtes visuelles Ergebnis ergeben hätte und sich somit die „On-the-fly“ Topologieänderungen in Grenzen hielten. Hier könnte man versuchen mittels Tuning der PL/SQL Prozedur ein besseres Zeitverhalten bei vielen Topologieänderungen zu erreichen.

Ein generelles Problem der Douglas-Peucker Linienvereinfachung wurde von Brenner und Sester in (Brenner & Sester 2005) beschrieben. Durch die Linienvereinfachung

kann eine Degenerierung strukturierter Objekte entstehen. Die Geometrie von Features, welche zum Beispiel Häuser, Grundstücksflächen oder Featureobjekte mit parallelen Linienzügen repräsentiert, kann durch simple Linienvereinfachung deformiert werden. Ein Betrachter erkennt dadurch sofort einen Fehler im Datenbestand. Für die Lösung dieses und ähnlicher Generalisierungsprobleme, im Speziellen bei der Generalisierung von Gebäudegrundrissen, gibt es von Sester (Prof. Dr.-Ing habil. Sester 2000) einige interessante Ideen, welche jedoch für diese Arbeit nicht aufgegriffen wurden, da die Dauer des Generalisierungsprozesses im Zuge der Datenbankabfrage möglichst minimiert werden sollte. Zu einem gewissen Grad können derartige Fehlerfälle durch die Verkleinerung des SVO-Kriteriums und der Definition von Maßstabsbereiche gelöst werden. Speziell für Übersichtsdarstellungen ist der beschriebene Ansatz aber sehr gut geeignet. Features mit sehr großem Darstellungsmaßstab können aufgrund eines kleinen Abfragefensters mit voller Detailstufe dargestellt werden.

6. Resümee

Die Verarbeitung größerer Datenmengen stellt schon seit je her eine große Herausforderung für die IT-Branche dar. Durch immer besser werdenden Hardwarevoraussetzungen kann oftmals die Verarbeitung einer größeren Datenmenge ermöglicht werden. In vielen Fällen ist eine Investition in bessere Hardware keine adäquate Lösung für die Problemstellung. Dann liegt es daran sich über einen anderen, besseren Einsatz von Softwaremethoden Gedanken zu machen. Mit dieser Master Thesis wird ein Lösungsansatz beschrieben, welcher die Darstellung größerer Geodatenmengen auf einem Client ermöglicht. Der in der vorliegenden Arbeit beschriebene Ansatz versucht über eine Kombination aus intelligenter Selektion und Generalisierung von Geometrieobjekten für den aktuellen Darstellungsmaßstab eine Datenreduktion herbeizuführen, sodass die Gesamtdauer für Datenbankabfrage, Übertragung der Daten über ein Netzwerk und Darstellung der Geodaten in einem Datenviewer möglichst gering ist. Wie van Osteroom in (van Oosterom 1993) beschrieben hat, kann durch Speicherung generalisierter Ansichten eine effiziente Datenbankabfrage ausgeführt werden, bei welcher eine reduzierte Datenmenge an einem Client übertragen wird. Die Abbildung von Generalisierungsstufen in sogenannten *Reactive data structures* erfordert in der Datenbank einiges mehr an Speicherplatz, da jede Generalisierungsstufe in dieser Datenstruktur abgelegt ist. In der vorliegenden Arbeit wird dabei ein Mittelweg ins Auge gefasst. Die Basisdatenstruktur, welche eine flache Tabelle mit Geometrieinformation darstellt, wird mit Topologieinformationen und Lotdistanzen für die Douglas-Peucker Linienvereinfachung angereichert. Dadurch kann, abhängig vom verwendeten Darstellungsmaßstab und dem Grad der Generalisierung (SVO-Kriterium), eine vereinfachte Sicht auf die Geodaten erstellt und angezeigt werden.

Wie die durchgeführten Testläufe gezeigt haben, ist der beschriebene Ansatz besonders bei der Darstellung von Übersichtskarten sehr gut einsetzbar. Polygone mit vielen Stützpunkten können sehr effizient vereinfacht werden. Problematisch sind jedoch Polygonfeatureklassen, bei welchen die Objektstruktur möglichst erhalten bleiben muss, da diese durch die Linienvereinfachung degenerieren können. Dabei handelt es sich meist um die Darstellung von Detailinformationen in einem Datenbestand. Durch die Konfiguration von Maßstabsbereichen zur Sichtbarkeitssteuerung von Featureklassen kann die Datenmenge reduziert werden. Featureklassen mit Detailinformation werden

dann nur bei relativ großen Maßstäben (1:250 – 1:5000) dargestellt. Dies impliziert ein kleineres Abfragefenster für die Datenbankabfrage, woraufhin weniger Features dargestellt werden müssen. Diese können dann mit vollem Detailgrad, sprich ohne Generalisierung, dargestellt werde.

Zusammenfassend kann man sagen, dass mithilfe eines aufbereiteten Ausgangsdatensatzes, der geschickten Konfiguration von Sichtbarkeitsbereichen und der Generalisierung von aktuell sichtbaren Daten, auch sehr große Vektordatenmengen verarbeitet werden können.

6.1) Verbesserungsvorschläge

Der umgesetzte Prototyp veranschaulicht die Möglichkeiten einer Kombination aus intelligenter Datenselektion und Generalisierung von Geodaten. Dabei ist zu bemerken, dass die Geodaten immer direkt aus den Originaldaten im Oracle Datenbankmodell abgeleitet wurden. Dabei wurde weder auf Serverseite noch auf Clientseite mit einem Cache gearbeitet.
Ein Cache könnte für zukünftige Untersuchungen eine weitere Möglichkeit zu einer effizienteren Darstellung der Geodaten sein. Eine verbesserte Performance wird dabei auf Kosten eines höheren Speicherverbrauchs am Client bzw. Server erreicht, also eine Verschiebung in Richtung eines statischen, vorberechneten Ansatzes.

Die Vereinfachung von Linienzügen wird aktuell im Prototyp durch eine PL/SQL Prozedur durchgeführt. Da die Lotdistanzen bereits vorab berechnet wurden und nur mehr die Punktliste unter Berücksichtigung dieser Lotdistanzen zu bestimmen ist, könnte man diese PL/SQL Prozedur durch eine erweiterte SQL-Anweisung ersetzen. Dies würde zwei große Vorteile gegenüber der bisherigen Lösung bringen. Zum einen müssen die Punkte aus der *Nested Table* nicht mehr in eine neue Punktliste umkopiert werden und andererseits erspart man sich die Kontextwechsel zwischen PL/SQL Engine und SQL Engine der Oracle Datenbank. In Summe könnte dadurch eine bessere Abfragezeit erreicht werden.

Eine weitere Chance, um eine bessere Performance zu erlangen, ist der Einsatz von Partitionen auf die *BoundingBox* Tabelle. Hierbei sind Partitionen in unterschiedlichen Varianten denkbar:

- Partitionen nach Geometrieart (Punkt, Linienzug, Polygon)
- Partitionen nach Featureklassen
- Partitionen nach Maßstabsbereichen

Dabei ist wohl die Partitionierung nach Maßstabsbereichen die vielversprechendste Variante, da jede Abfrage immer für einen Darstellungsmaßstab ausgeführt wird. In dieser Arbeit wurde die Partitionierung nicht weiter untersucht, da diese eine Oracle Enterprise Edition erfordern würde.

Neben der Partitionierung bietet Oracle viele weitere Möglichkeiten des Tunings. Neben dem Tuning von SQL Abfragen, der Verbesserung des PL/SQL Quellcodes oder der Vermeidung von SQL-PL/SQL Kontextwechsel, kann die Oracle Datenbank selbst im Hinblick auf die physikalische Ablage beeinflusst werden. Eine Vorsortierung der Geodaten ist eine weitere Möglichkeit, welche einen Einfluss auf die Abfrageperformance haben könnte. Dadurch wird es möglich, dass räumlich nahe beieinanderliegende Features auch physikalisch Nahe gespeichert werden. Oracle benötigt dann weniger Dateizugriffe, um die notwendigen Geodaten in den Speicher einzulesen. Dies sind jedoch Möglichkeiten der Performanceverbesserung aus datenbanktechnischer Sicht. Da in der Thesis das beschriebene Viewing-Service untersucht werden sollte, wurde bei Performanceproblemen eher an Konzeptänderungen investiert, als an Tuningmöglichkeiten in der Datenbank.

6.2) Zukünftige Untersuchungen

Der umgesetzte Prototyp untersucht einen konkreten sehr speziellen Teil eines Services zur Anzeige von Geodaten. Für einen Einsatz in einem Softwareprodukt müssen noch einige weitere Fragestellungen untersucht werden:

- Wie können Featureklassen unterschiedlicher Geometrieart (Punkt, Linienzug, Polygon) in der Datenbank gespeichert werden und mit dem Viewing-Service abgefragt werden?
- Wie können domänenspezifische, topologische Beziehungen (zusätzlich zu Punkt-Linienzug-Polygon Beziehungen) zwischen Features berücksichtigt werden? Beispiel: ein Gebäude gehört zu einem Grundstück, welches zu einer Katastralgemeinde gehört.
- Die Ausgangsdaten (ESRI Shape-Dateien) wurden mittels einer eigenen in C# geschriebenen Komponente für das Oracle-Datenbankmodell aufbereitet. Dabei wurden die topologischen Beziehungen zwischen den Geometrieobjekten ermittelt und in der Datenbank gespeichert. Kann die Herstellung der Topologie und das Anfüllen des Datenmodells direkt in der Datenbank durchgeführt werden?
- Wie sieht der Workflow für Änderungen bzw. Bearbeitungen des Datenbestandes aus? Können Änderungen am Datenbestand direkt in den Datenbanktabellen durchgeführt werden oder muss es einen Differenzdatenimport geben?
- Eine Datenbereinigung wurde bisher vollends außer Acht gelassen. Fehler im Ausgangsdatenbestand führen zu Fehler in den topologischen Beziehungen und folglich zu Fehler im Zuge des Generalisierungsprozesses. Welche Datenqualität müssen Daten für die Nutzung des Viewing-Service besitzen und wie wird diese hergestellt?
- Eine Verwendung eigener SVO-Kriterien für Polygonreduktion und Linienvereinfachung ist zu testen. Der Grad der Generalisierung könnte damit noch besser gesteuert werden.
- …

Appendix A

Datenbankskript

```
/**************
Create Schema script for Viewing-Service
SGamperl
26.01.2011
**************/

/*
User defined types
*/

// ID Type to store id references in the nested tables
CREATE OR REPLACE TYPE VIEWSERV.IDTYPE
    AS OBJECT
    (
        IDX NUMBER ,
        ID NUMBER
    ) FINAL
;

// ID Table to store id references
CREATE OR REPLACE TYPE VIEWSERV.ID_TABLE
    IS TABLE OF IDTYPE ;

// integer array
CREATE OR REPLACE TYPE VIEWSERV.INTARRAY
    IS TABLE OF INTEGER ;

// point type to store the points of a linestring
CREATE OR REPLACE TYPE VIEWSERV.POINTTYPE
    AS OBJECT
    (
        IDX NUMBER , // point index
        POINTID NUMBER , // point id
        EAST FLOAT , // east value
        NORTH FLOAT , // north value
        DISTANCE FLOAT // distance for Douglas Peucker
    ) FINAL
;

// point array type for linestrings
CREATE OR REPLACE TYPE VIEWSERV.POINTARRAY
    IS TABLE OF POINTTYPE ;

// bounding box type to store selected id's within the query rectangle
create or replace type bbox_row as object
(
  ID   NUMBER, // id of the object
  DIM  INTEGER, // dimension (0 = point,1 = linestring, 2 = polygon)
  BBOX SDO_GEOMETRY // bounding box as MDSYS.SDO_GEOMETRY
);

// collection of type bbox_row
create or replace type BBOX_TABLE as table of bbox_row;

/* Visibility table*/
CREATE TABLE "VISIBILITY"
   (  "ID" NUMBER NOT NULL ENABLE,
      "FEATURECLASSNAME" VARCHAR2(200),
      "FROM_SCALE" NUMBER,
      "TO_SCALE" NUMBER,
      CONSTRAINT "VISIBILITY_ID" PRIMARY KEY ("ID")
   ) ;
```

```
CREATE INDEX VIEWSERV."VISIBILITY_TO_SCALE_IDX" ON "VISIBILITY" ("TO_SCALE")
  COMPUTE STATISTICS
 ;

CREATE INDEX VIEWSERV."VISIBILITY_FROM_SCALE_IDX" ON "VISIBILITY" ("FROM_SCALE")
  COMPUTE STATISTICS
   ;

create or replace type BBOX_TABLE as table of bbox_row;

/* Bounding box table */

CREATE TABLE "BoundingBox"
   (  "ID" NUMBER NOT NULL ENABLE,
      "DIM" NUMBER(38,0),
      "BBOX" "MDSYS"."SDO_GEOMETRY" ,
      "BBOX_ID" NUMBER,
      "FCN_ID" NUMBER,
      CONSTRAINT "BBOX_PK" PRIMARY KEY ("BBOX_ID")
   );

/*Point table*/
CREATE TABLE "POINT"
  (  "ID" NUMBER(38,0) NOT NULL ENABLE,
      "EAST" FLOAT(126),
      "NORTH" FLOAT(126),
      "LINESTRINGS" ID_TABLE" ,
      CONSTRAINT "POINT_ID_PK" PRIMARY KEY ("ID")
  )
  NESTED TABLE "LINESTRINGS" STORE AS "PT_TO_LINE_REF"
 (( PRIMARY KEY ("NESTED_TABLE_ID", "IDX") ENABLE)
 ORGANIZATION INDEX OVERFLOW
  ) RETURN AS VALUE;

/*Linestring table*/
CREATE TABLE "LINESTRING"
   (  "ID" NUMBER NOT NULL ENABLE,
      "BBOX" "MDSYS"."SDO_GEOMETRY" ,
      "POINTS" VIEWSERV."POINTARRAY" ,
      "LEFTPOLYGON" NUMBER,
      "RIGHTPOLYGON" NUMBER,
      "NROFPOINTS" NUMBER,
   CONSTRAINT "LINESTRING_PK" PRIMARY KEY ("ID")
   )
 NESTED TABLE "POINTS" STORE AS "LINE_TO_PT_REF"
 (( PRIMARY KEY ("NESTED_TABLE_ID", "IDX") ENABLE)
 ORGANIZATION INDEX ) RETURN AS VALUE;

/* Polygon table*/
CREATE TABLE "POLYGON"
   (  "ID" NUMBER(38,0) NOT NULL ENABLE,
      "BBOX" "MDSYS"."SDO_GEOMETRY" ,
      "LINESTRINGS" "ID_TABLE" ,
      "ISISLAND" NUMBER,
      CONSTRAINT "POLYGON_PK" PRIMARY KEY ("ID")
   )
 NESTED TABLE "LINESTRINGS" STORE AS "POLYGON_TO_LINE_REF"
 (( PRIMARY KEY ("NESTED_TABLE_ID", "IDX") ENABLE)
 ORGANIZATION INDEX OVERFLOW ) RETURN AS VALUE;

/*
Sequences
*/

CREATE SEQUENCE  VIEWSERV."LINEID_SEQ"  MINVALUE 1 MAXVALUE 1000000000 INCREMENT BY 1000
START WITH 10100001 CACHE 20 NOORDER  NOCYCLE ;
CREATE SEQUENCE  VIEWSERV."POLYID_SEQ"  MINVALUE 1 MAXVALUE 1000000000 INCREMENT BY 1000
START WITH 140001 CACHE 20 NOORDER  NOCYCLE ;
CREATE SEQUENCE  VIEWSERV."POINTID_SEQ"  MINVALUE 1 MAXVALUE 1000000000 INCREMENT BY
1000 START WITH 10100001 CACHE 20 NOORDER  NOCYCLE ;
CREATE SEQUENCE  VIEWSERV."BBOXID_SEQ"  MINVALUE 1 MAXVALUE 1000000000 INCREMENT BY 1
START WITH 9982 CACHE 20 NOORDER  NOCYCLE ;
```

Quellcode 6-1: Datenbankskript zur Erstellung der Tabellen und User Defined Types

```
/*
Linestrings (stored as polylines)
*/

CREATE TABLE "LINESTRINGNORMAL"
   (  "ID" NUMBER NOT NULL ENABLE,
      "BBOX" "MDSYS"."SDO_GEOMETRY" ,
       "PROJECTID" NUMBER,
      "POINTS" VIEWSERV."POINTARRAY" ,
      "NROFPOINTS" NUMBER,
      "FEATURECLASSNAMEID" NUMBER,
      CONSTRAINT "LINESTRINGNORMAL_PK" PRIMARY KEY ("ID")
   )
 NESTED TABLE "POINTS" STORE AS "LINE_TO_PT_NORMAL_REF"
 (( PRIMARY KEY ("NESTED_TABLE_ID", "IDX") ENABLE)
 ORGANIZATION INDEX ) RETURN AS VALUE;

CREATE INDEX "FEATURECLASSIDIDXNORMAL" ON "LINESTRINGNORMAL" ("FEATURECLASSNAMEID") ;


/*
Linestring detail table (stored as line segments)
*/

CREATE TABLE VIEWSERV."LINESTRINGDETAIL"
   (  "ID" NUMBER NOT NULL ENABLE,
      "BBOX" "MDSYS"."SDO_GEOMETRY" ,
       "PROJECTID" NUMBER,
      "POINTS" VIEWSERV."POINTARRAY" ,
      "NROFPOINTS" NUMBER,
      "FEATURECLASSNAMEID" NUMBER,
      CONSTRAINT "LINESTRINGDETAIL_PK" PRIMARY KEY ("ID")
   )
 NESTED TABLE "POINTS" STORE AS "LINE_TO_PT_DETAIL_REF"
 (( PRIMARY KEY ("NESTED_TABLE_ID", "IDX") ENABLE)
 ORGANIZATION INDEX ) RETURN AS VALUE;

CREATE INDEX VIEWSERV."FEATURECLASSIDIDXDETAIL" ON VIEWSERV."LINESTRINGDETAIL" ("FEA-
TURECLASSNAMEID") ;
```

Quellcode 6-2: Einfache Linientabellen für Testszenario 1

PLSQL Package Viewing-Service

```
create or replace package ViewingService is

  -- Author  : SGAMPERL
  -- Created : 05.07.2010 19:06:18
  -- Purpose : Viewing Service / Efficient transmission of geodata from the database
  --           to a client

  -- flag for trace statements
  debugOutput boolean := false;

  /************************************************/
  /* Public function and procedure declarations   */
  /************************************************/

  -- retrieve simplified linestring features in the given query rectangle
  -- depending on the given svo criteria and the actual scale
  function GetGeneralizedData(x1    in binary_float,
                              y1    in binary_float,
                              x2    in binary_float,
                              y2    in binary_float,
                              SVO   in binary_float,
                              scale binary_float) return Sys_Refcursor;

  /************************************************/
  /* Additional public functions because they are  */
  /* used inside of sql statements. the SQL engine */
  /* have to know these function. That's the reason*/
  /*  why they are public                          */
  /************************************************/

  -- checks whether a feature is to small
  function IsGeometryToSmall(bbox in mdsys.sdo_ordinate_array,
                             svo  in float) return varchar2 ;

  -- simplifies a given linestring
  function SelectLineString(lineid     number,
                            nrofpoints number,
                            points     pointarray,
                            svo        float) return pointarray;

  -- simplifies a list of linestring points
  function GetGeneralizedPoints(nrofpoints            integer,
                                points                pointarray,
                                smallestVisibleObject float) return pointarray;
  /************************************************/
  /* Utility functions to administrate the service */
  /************************************************/

  -- Get a free id range for imports of new datasets
  procedure GetFreeIdRange(dimension   number,
                           neededRange number,
                           startNumber out number,
                           endNumber   out number);

  -- Build the spatial index on the bounding box table
  -- and creates the metadata for the bounding box table
  procedure buildindex;
  -- Deletes the spatial index on the bounding box table
  procedure deleteindex;

end ViewingService;
```

Quellcode 6-3: Package Specification Viewing-Service

```
create or replace package body ViewingService is
  -- line table type
  type line_table_type is table of linestring%rowtype index by pls_integer;
  -- point table type
  type point_table_type is table of point%rowtype index by pls_integer;

  -- line_table and point_table:
  -- these two tables hold the modified objects through one query
  line_table  line_table_type;
  point_table point_table_type;

  -- list of features who fullfill the spatial and thematic filter
  spatial_query_table bbox_table;

  /*********************************************************************/
  /* Helper procedure for debug outputs if the debugOutput flag is set */
  /*********************************************************************/
  procedure OutputWindow(msg varchar2) is
  begin
    if (debugOutput = true) then
      DBMS_OUTPUT.put_line(msg);
    end if;
  end;

  /*****************************************************************/
  /* GetGeneralizedData                                            */
  /*****************************************************************/
  function GetGeneralizedData(x1    in binary_float,
                              y1    in binary_float,
                              x2    in binary_float,
                              y2    in binary_float,
                              SVO   in binary_float,
                              scale in binary_float) return sys_refcursor is
  begin
    -- select the features
    spatial_query_table := SelectionProcess(x1, y1, x2, y2, scale);
    -- reduce the polygons
    ReductionProcess(SVO);
    -- simplify the linestrings
    return Simplificationprocess(SVO);

  end GetGeneralizedData;

  /*****************************************************************/
  /* SelectionProcess - selects the features from a given query    */
  /* rectangle and the actual scale                                */
  /*****************************************************************/
  function SelectionProcess(x1 in binary_float,
                            y1 in binary_float,
                            x2 in binary_float,
                            y2 in binary_float,
                            scale in binary_float) return bbox_table is
    createviewstmt varchar2(200);
     -- filter geometry
    geometry sdo_geometry;
    -- used srd
    srid_var integer;
  begin
    createviewstmt := 'create or replace view visiblefeatureclasses as       select v.id
        from visibility v
       where ' || scale || '  >= v.from_scale
         and ' || scale || ' <= v.to_scale';

    -- work around / future topic: find a better solution for the culture problem
    createviewstmt := replace(createviewstmt, ',', '.');
    execute immediate createviewstmt;

    -- find the spatial reference system identifier from the
    -- user_sdo_geom_metadata table
    select srid
      into srid_var
      from user_sdo_geom_metadata
     where table_name = 'BBOX';

    -- build rectancle geometry used for the query
    geometry := sdo_geometry(2003,
```

```
                            srid_var,
                            null,
                            sdo_elem_info_array(1, 1003, 3),
                            SDO_ordinate_array(x1, y1, x2, y2));

    -- retrieve all bounding boxes of the objects in the query area and of the
    -- right featureclass type
    select bbox_row(b.id, b.dim, b.bbox) bulk collect
      into spatial_query_table
      from bbox b, visiblefeatureclasses v
     where b.fcn_id = v.id
       and sdo_filter(b.bbox, geometry) = 'TRUE';

    return spatial_query_table;
  end SelectionProcess;

  /*****************************************************************/
  /* Reduction process - removes polygons which are too small      */
  /*****************************************************************/
  procedure ReductionProcess(SVO in binary_float) is
    nroffeatureaclasses integer;
    neighborline     linestring%rowtype;
    startpoint       point%rowtype;
    pointToRemove_id integer;
    linestart_id     integer;
    lineend_id       integer;
    startpoint_id    integer;
    type integer_array_type is table of integer index by pls_integer;
    pointAlreadyRemoved integer_array_type;
    removepolycountlines integer := 0;
  begin
    select count(*) into nroffeatureaclasses from visiblefeatureclasses;

    -- if there are featureclasses to consider
    if (nroffeatureaclasses > 0) then
      startpoint_id := -1;

      -- get all linestrings from polygons which have to be removed
      for actualLine in (select lines.*
                           from polygon p,
                                table(p.linestrings) l,
                                linestring lines,
                                (select s.id
                                   from table(spatial_query_table) s
                                  where s.dim = 2) s
                          where p.isisland = 0
                               and IsGeometryToSmall(p.bbox.SDO_ORDINATES, SVO) =
                                'TRUE'
                            and lines.id = l.id
                            and s.id = p.id
                         ) loop

        removepolycountlines := removepolycountlines + 1;
        linestart_id         := actualLine.points(actualLine.points.first)
                                .pointid;
        lineend_id           := actualLine.points(actualLine.points.last)
                                .pointid;

        -- set the preserving point
        if startpoint_id = -1 then
          startpoint_id := linestart_id;
          startpoint    := GetCopyOfPoint(startpoint_id);
          OutputWindow(CONCAT('PRESERVED POINT IS ', pointToRemove_id));
        end if;

        if (linestart_id = startpoint_id AND
           pointAlreadyRemoved.Exists(lineend_id) = FALSE) then
          linestart_id     := -1; -- ignore this point because it's the startpoint_id
which must remain in the dataset
          pointToRemove_id := lineend_id;
        elsif (lineend_id = startpoint_id AND
              pointAlreadyRemoved.Exists(linestart_id) = FALSE) then
          lineend_id       := -1; -- ignore this point because it's the startpoint_id
which must remain in the dataset
          pointToRemove_id := linestart_id;
        else
```

```
          if (pointAlreadyRemoved.Exists(linestart_id) = FALSE) then
            pointToRemove_id := linestart_id;
          else
            pointToRemove_id := lineend_id;
          end if;
        end if;

        if (pointToRemove_id != startpoint_id) then

          OutputWindow(CONCAT('DETERMINE NEIGHBOR LINES FOR POINT ID ',
                              pointToRemove_id));

          for neighborline_cursor in (select (lines.id)
                                        from Point,
                                             table(Point.linestrings) lines
                                       where (point.id = pointToRemove_id
                                             )
                                      ) loop
            OutputWindow(neighborline_cursor.id);

            -- copy linestring
            neighborline := GetCopyOfLineString(neighborline_cursor.id);

            -- redefine point
            RedefinePoint(neighborline,
                          actualLine.id,
                          pointToRemove_id,
                          startpoint);
          end loop;

          pointAlreadyRemoved(pointToRemove_id) := pointToRemove_id;
        end if;
      end loop;

      -- dbms_output.put_line(CONCAT('modified lines:', removepolycountlines));
    end if;
  end ReductionProcess;

  /*****************************************************************/
  /* SimplificationProcess - simplifies linestrings                */
  /*****************************************************************/
  function SimplificationProcess(SVO in float) return sys_refcursor is
    l_cursor sys_refcursor;
  begin

    open l_cursor for
      select l.id,SelectLineString(l.id, l.nrofpoints, l.points, svo)
      --l.leftpolygon,l.rightpolygon) -- for the moment not necessary
        from linestring l
       where l.id in
             (select s.id
                from table(spatial_query_table) s
               where s.dim = 1
              minus (select l.id
                       from polygon p, table(p.linestrings) l
                      where p.isisland = 1
                        and ViewingService.IsGeometryToSmall(p.bbox.SDO_ORDINATES,
                                                             SVO) = 'TRUE'));
    return l_cursor;
  end SimplificationProcess;

  /*****************************************************************/
  /* Determines whether a feature is too small for                 */
  /*****************************************************************/
  function IsGeometryToSmall(bbox in mdsys.sdo_ordinate_array,
                             svo  in float) return varchar2 is
    boxWidth  float; -- width of the bbox
    boxHeigth float; -- heigth of the bbox
    diagonal  float; -- diagonal of the bbox
  begin
     if (svo = 0.0) then
      return('FALSE');
    end if;

    boxWidth := bbox(3) - bbox(1);
    boxHeigth := bbox(4) - bbox(2);
```

```
    diagonal  := sqrt(boxWidth * boxWidth + boxHeigth * boxHeigth);

    -- if diagonal is too small
    if diagonal < svo then
      return('TRUE');
    end if;

    return('FALSE');
  end IsGeometryToSmall;

  /*********************************************************************/
  /* Gets the point with the specified ID                              */
  /* the returned point can be the original point or a modified point  */
  /*********************************************************************/
  function GetPoint(point_id integer) return point%rowtype is
    selectedpoint point%rowtype;
  begin
    -- if the point was modified, it gets returned from the point_table
    if (point_table.exists(point_id) = TRUE) then
      return point_table(point_id);
    else
      -- otherwise return the point from the point table
      select * into selectedpoint from point where point.id = point_id;
      return selectedpoint;
    end if;
  end GetPoint;

  /*********************************************************************/
  /* Creates a copy of the line with the id line_id, stores the line */
  /* in the associative array and returns it to the caller           */
  /* If the copy was already be made, the copied linestring is       */
  /* returned.                                                       */
  /*********************************************************************/
  function GetCopyOfLineString(line_id integer) return linestring%rowtype is
    copiedline linestring%rowtype;
  begin
    -- if line string copy already exists
    if (line_table.exists(line_id) = TRUE) then
      -- return the copy
      return line_table(line_id);
      null;
    else
      -- read the line string from the linestring table
      select *
        into copiedline
        from linestring
       where linestring.id = line_id;

      -- save a copy in the line_table
      line_table(line_id) := copiedline;
      return copiedline;
    end if;
  end GetCopyOfLineString;

  /*********************************************************************/
  /* Creates a copy of the point with the id point_id, stores the   */
  /* point in the associative array and returns it to the caller    */
  /* If the copy was already be made, the copied point is returned  */
  /*********************************************************************/
  function GetCopyOfPoint(point_id integer) return point%rowtype is
    copiedpoint point%rowtype;
  begin
    -- if the point is already in the point_table, return it from there
    if (point_table.exists(point_id) = TRUE) then
      return point_table(point_id);
    else
      -- otherwise make a copy from the point table
      select * into copiedpoint from point where point.id = point_id;
      -- save the copied point in the point_table
      point_table(point_id) := copiedpoint;
      -- and return the copy
      return copiedpoint;
    end if;
  end GetCopyofPoint;

  /*********************************************************************/
```

```
/* Redefine is a very essential method to redefine the topology of*/
/* one point and all its neighbor lines one point gets eliminated */
/* from the dataset => all referencing lines get connected to a   */
/* preserving point                                               */
/*****************************************************************/
procedure RedefinePoint(lineToPreserve    linestring%rowtype,
                        removedSegment_Id integer,
                        pointToRemove     integer,
                        pointToPreserve   point%rowtype) is
  v_index    BINARY_INTEGER;
  existsflag BOOLEAN;
begin
  -- first step: modify point references of the line

  -- replace pointToRemove from lineToPreserve.Points with pointToPreserve
  v_index := lineToPreserve.points.first;

  while v_index is not NULL loop

    -- if actual point is pointToRemove
    if (lineToPreserve.points(v_index).pointid = pointToRemove) then
      OutputWindow('LINE ID = ' || lineToPreserve.ID ||
                   ': EXISTING POINT REFERENCE ID= ' || pointToRemove ||
                   ' REPLACED WITH REFERENCE ID= ' || pointToPreserve.id);
      -- modify the copy in the in-memory line_table / not in the database table
      line_table(lineToPreserve.Id).points(v_index).pointid := pointToPreserve.id;
    end if;
    v_index := lineToPreserve.points.next(v_index);
  end loop;


  -- second step: modify line references of the point

  -- add lineToPreserver.ID to pointToPreserver.Lines
  v_index    := pointToPreserve.linestrings.first;
  existsflag := false;
  while v_index is not NULL loop
    -- if the actual line is the removedSegment_Id
    if (pointToPreserve.linestrings(v_index).id = removedSegment_Id) then
      OutputWindow('POINT ID = ' || pointToPreserve.ID ||
                   ': EXISTING LINE REFERENCE ID= ' || removedSegment_Id ||
                   ' REPLACED WITH REFERENCE ID= ' || lineToPreserve.Id);
      -- modify the copy in the in-memory point_table / not in the database table
      point_table(pointToPreserve.id).linestrings(v_index).id := lineToPreserve.Id;
      existsflag := true;
    end if;
    v_index := pointToPreserve.linestrings.next(v_index);
  end loop;

  -- if the line isn't already in the in the point_table
  if (existsflag = false) then
    OutputWindow('POINT ID = ' || pointToPreserve.ID ||
                 ': LINE REFERENCE ID= ' || lineToPreserve.Id ||
                 ' ADDED');
    point_table(pointToPreserve.id).linestrings.Extend(1);
    point_table(pointToPreserve.id).linestrings(pointToPreserve.linestrings.last + 1)
               := idtype(0,lineToPreserve.Id);
  end if;
  return;
end RedefinePoint;

/*****************************************************************/
/* SelectLineString - simplifies one linestring depending on a    */
/* given svo criteria
/*****************************************************************/
function SelectLineString(lineid     number,
                          nrofpoints number,
                          points     pointarray,
                          svo        float) return pointarray is
begin
  -- if there is a modified line => simplify the modified line
  if (line_table.Exists(lineid)) then
    return GetGeneralizedPoints(line_table(lineid).nrofpoints,
                                line_table(lineid).points,
                                svo);
  else -- else simplify the original line
```

```
    return GetGeneralizedPoints(nrofpoints, points, svo);
  end if;

end SelectLineString;

/*****************************************************************/
/* GetFreeIdRange                                                */
/*****************************************************************/
procedure GetFreeIdRange(dimension   number,
                         neededRange number,
                         startNumber out number,
                         endNumber   out number) is
begin
  if (dimension = 0) then
    execute immediate ('alter sequence pointid_seq increment by ' ||
                       neededRange);
    begin
      endNumber   := pointid_seq.nextval;
      startNumber := pointid_seq.currval - neededRange;
    end;
  elsif (dimension = 1) then
    execute immediate ('alter sequence lineid_seq increment by ' ||
                       neededRange);
    begin
      endNumber   := lineid_seq.nextval;
      startNumber := lineid_seq.currval - neededRange;
    end;
  elsif (dimension = 2) then
    execute immediate ('alter sequence polyid_seq increment by ' ||
                       neededRange);
    begin
      endNumber   := polyid_seq.nextval;
      startNumber := polyid_seq.currval - neededRange;
    end;
  else
    raise PROGRAM_ERROR;
  end if;
end GetFreeIdRange;

/*****************************************************************/
/* DeleteIndex                                                   */
/*****************************************************************/
procedure DeleteIndex is
begin
  begin
    Execute immediate 'DROP INDEX BBOX_SPATIAL_IDX';
  exception
    when others then
      dbms_output.put_line('BBOX_SPATIAL_IDX not found');
  end;

  DELETE FROM USER_SDO_GEOM_METADATA
   WHERE TABLE_NAME = 'BBOX'
     AND COLUMN_NAME = 'BBOX';
end deleteindex;

/*****************************************************************/
/* DeleteIndex                                                   */
/*****************************************************************/
procedure BuildIndex is
  polybox        sdo_geometry;
  dimensionArray sdo_dim_array;

begin
  select sdo_aggr_mbr(bbox) into polybox from polygon;
  if (polybox.SDO_SRID is NULL) then
    DBMS_output.put_line('No polygons there');
  else
    dimensionArray := SDO_DIM_ARRAY(SDO_DIM_ELEMENT('LONGITUDE',
                                                    polybox.SDO_ORDINATES(1),
                                                    polybox.SDO_ORDINATES(3),
                                                    0.5),
                                    SDO_DIM_ELEMENT('LATITUDE',
                                                    polybox.SDO_ORDINATES(2),
                                                    polybox.SDO_ORDINATES(4),
                                                    0.5));
```

```
      INSERT INTO USER_SDO_GEOM_METADATA
      VALUES
        ('BBOX', 'BBOX', dimensionArray, polybox.SDO_SRID);
      EXECUTE IMMEDIATE 'CREATE INDEX bbox_spatial_idx ON bbox(bbox) indextype is
mdsys.spatial_index';
    end if;
  end buildindex;

  /*****************************************************************/
  /* ConsiderPointChanges -  consider modified points in the       */
  /* simplification process                                        */
  /*****************************************************************/
  function ConsiderPointChanges(points in pointarray) return pointarray is
    tmpArray  pointarray;
    startnode point%rowtype;
    endnode   point%rowtype;
  begin
    return points;
    tmpArray := points;
    -- start and end node may have changed through topology modifications
    startnode := GetPoint(points(points.first).pointid);
    endnode   := GetPoint(points(points.last).pointid);

    tmpArray(points.first).east := startnode.east;
    tmpArray(points.first).north := startnode.north;
    tmpArray(points.last).east := endnode.east;
    tmpArray(points.last).north := endnode.north;

    return tmpArray;
  end ConsiderPointChanges;

  /*****************************************************************/
  /* GetGeneralizedPoints - simplifies a list of points            */
  /*****************************************************************/
  function GetGeneralizedPoints(nrofpoints           integer,
                                points               pointarray,
                                smallestVisibleObject float)
    return pointarray is
    type ptarray is table of pointtype index by pls_integer;
    tempArray ptarray;
    resPoints pointarray := pointarray();
    actualpointidx pls_integer;
  begin
    -- if we have already a simple line, just return it
    if (nrofpoints < 5) then
      return ConsiderPointChanges(points);
    end if;

    if (points.count > 0) then

      -- select all points where the distance is bigger than svo criteria
      actualpointidx := points.first;
      loop
        if (points(actualpointidx).distance > smallestVisibleObject) then
          tempArray(points(actualpointidx).idx) := points(actualpointidx);
        end if;
        exit when actualpointidx = points.last;
        actualpointidx := points.next(actualpointidx);
      end loop;

      actualpointidx := tempArray.first;
      FOR i in 1 .. tempArray.Count LOOP
        resPoints.extend(1);
        resPoints(i) := tempArray(actualpointidx);
        actualpointidx := tempArray.next(actualpointidx);
      end loop;
    end if;

    return ConsiderPointChanges(resPoints);
  end GetGeneralizedPoints;
end ViewingService;
```

Quellcode 6-4: Package Body Viewing-Service

Appendix B

FME Datenaufbereitungen

Für den Vergleich der unterschiedlichen Ablagevarianten aus Kapitel 5.2.1) wurden die Ausgangsdaten aufbereitet. Dies wurde mittels der Feature Manipulation Engine (FME) durchgeführt. Für die Aufteilung der Polygone in Polylinien bzw. Einzelsegmente und der Ablage dieser einer eigenen Oracle Tabelle wurde die folgende Workbench verwendet:

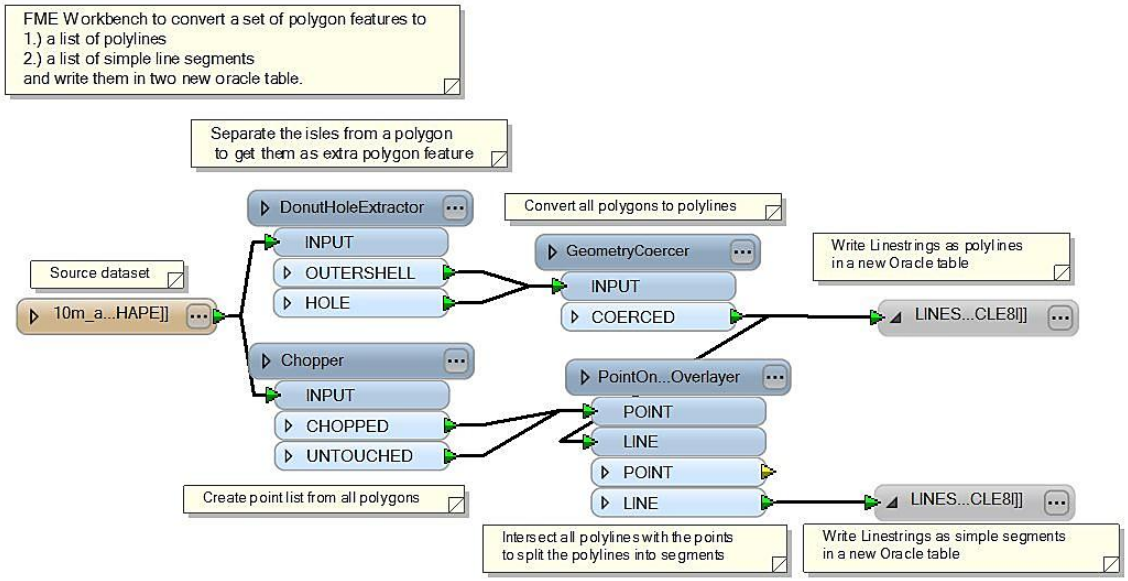


Abbildung 6-1: FME Workbench zur Aufteilung des Polygondatenbestandes

Für die Prüfung der Auswirkung des Koordinatensystems auf die Wahl des SVO-Kriteriums musste der Datenbestand von WGS84 (EPSG:4326) zu Lambert Austria (EPSG:31287) transformiert werden. Dies erfolgte ebenfalls mittels einer FME Workbench:

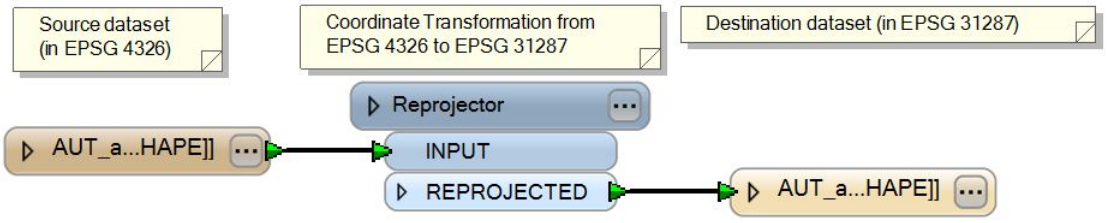


Abbildung 6-2: Koordinatentransformation mit FME (EPSG 4326 zu EPSG 31287)

Literaturverzeichnis

Arora, G., u.a. 2005. *Oracle Database Application Developer's Guide - Object-Relational Features 10g Release 1 (10.1): Design Considerations for Oracle Objects.* URL: http://download.oracle.com/docs/cd/B19306_01/appdev.102/b14260/toc.htm [Stand 2011-03-09].

Bartelme, N. 2005. *Geoinformatik: Modelle; Strukturen; Funktionen.* 4., vollständig überarbeitete Auflage. Berlin, Heidelberg: Springer-Verlag Berlin Heidelberg.

Booth, D., u.a. 2004. *Web Service Architecture*. URL: http://www.w3.org/TR/ws-arch/#service_oriented_architecture [Stand 2011-03-09].

Brenner, C. & Sester, M. 2005. Continuous Generalization for Small Mobile Display, in Taylor & Francis Group (Hg.): *ISPRS Book Series*. London, 33–41.

Burt, P. & Adelson, E. 1983. The Laplacian Pyramid as a Compact Image Code. *IEEE TRANSACTIONS ON COMMUNICATIONS*(31(4)), 532–540.

Clementini, E. 1996. A Model for Representing Topological Relationships between Complex Geometric Features in Spatial Databases. *A Model for Representing Topological Relationships between Complex Geometric Features in Spatial Databases*, Vol. 90, No. 1/4 (1996),121-136.

Cyran, M. 2002. *Oracle9i Database Concepts Release 2. Chapter Partitioned Tables and Indexes*. URL: http://download.oracle.com/docs/cd/B10500_01/server.920/a96524/c12parti.htm [Stand 2011-03-09].

DORIS, Land Oberösterreich 2008. *Gemeindegrenzen. Generalisierung (200 - 500m).* URL: http://www.doris.ooe.gv.at/index.asp?MenuID=4 [Stand 2011-03-12].

Douglas, D. & Peucker, T. 1973. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *The Canadian Cartographer* 1973(10), 112–122.

Galanda, M. 2003. *Automated polygon generalization in a multi agent system*. Zürich: Universität Zürich. Mathematisch-naturwissenschaftliche Fakultät.

Gao, W., Gong, J. & Li, Z. 2002. Thematic Knowledge for the Generalization of Land Use Data, in Maney Publishing (Hg.): *Cartographic Journal, The*, 245–252.

Hake, G., Grünreich, D. & Meng, L. 2002. *Kartographie: Visualisierung raum-zeitlicher Informationen.* 8., vollständig neu bearb. und erw. Aufl. Berlin: de Gruyter. (De-Gruyter-Lehrbuch).

Helle, D. & Tonnhofer, O. 2011. *MapProxy: MapProxy is an open source proxy for geospatial data*. URL: http://mapproxy.org/ [Stand 2011-03-20].

Hijmans, R., u.a. 2009. *Global Administrative Areas*. URL: http://www.gadm.org/formats [Stand 2011-03-12].

Hill, D. 2005. *Thin Client oder Smart Client: Architektur der Darstellungsschicht*. URL: http://msdn.microsoft.com/de-de/library/aa480039.aspx#ID0ERB [Stand 2011-03-09].

Kelso, N. & Patterson, D. 2009. *Natural Earth data*. URL: http://www.naturalearthdata.com/about/terms-of-use/ [Stand 2011-03-20].

Kidner, D. 1996. Geometric Signatures For Determining Polygon Equivalence During Multi-Scale GIS Update: Second Joint European Conference and Exhibition, Barcelona, Spain, [March 27 - 29] 1996 ; [V Congreso de la Asociación Española de

Sistemas de Información Geográfica]: *Geographical Information: From Research to Application through Cooperation.*, 241–244.
Li, Z. 2007. Algorithmic foundation of multi-scale spatial representation. *Algorithmic foundation of multi-scale spatial representation, Zhilin Li.* Online im Internet: URL: http://www.loc.gov/catdir/enhancements/fy0661/2006045503-d.html / http://www.gbv.de/dms/bowker/toc/9780849390722.pdf.
Li, Z. & Choi 2002. Topographic Map Generalization: Association of Road Elimination with Thematic Attributes, in Maney Publishing (Hg.): *Cartographic Journal, The*, 153–166.
Li, Z. & Openshaw, S. 1992. Algorithms for automated line generalization based on a natural principle of objective generalization. *International Journal of GIS*(6(5)), 373–389.
Li, Z. & Openshaw, S. 1993. A natural principle for objective generalisation of digital map data. *Cartography and Geographic Information Systems*(20(1)), 19–29.
Longley, Paul A. 2002. *Geographic information systems and science.* Repr. Chichester: Wiley. Online im Internet: URL: http://www.gbv.de/dms/bowker/toc/9780471495215.pdf.
Meijers, M. 2006. Implementation and testing of variable scale topological data structures. Master Thesis.
Murray, C. 2001. *Oracle® Spatial User's Guide and Reference: Indexing and Querying Spatial Data*. URL: http://download.oracle.com/docs/cd/B10501_01/appdev.920/a96630/sdo_index_query.htm [Stand 2011-03-09].
Open Geospatial Consortium 2006. *OpenGIS Web Map Service (WMS) Implementation Specification.* 0.3.0. Aufl. URL: http://www.opengeospatial.org/standards/wms [Stand 2011-03-09].
Open Geospatial Consortium 2010a. *OpenGIS Web Feature Service (WFS) Implementation Specification*. URL: http://www.opengeospatial.org/standards/wfs [Stand 2011-03-09].
Open Geospatial Consortium 2010b. *OpenGIS Web Map Tile Service Implementation Standard, Version 1.0.0*. URL: http://www.opengeospatial.org/standards/wmts [Stand 2011-03-09].
Palmer, S. 1999. *Vision Science: Photons to Phenomenology.*
Schütze, E. 2007. Stand der Technik und Potenziale von Smart Map Browsing im Webbrowser: am Beispiel der Freien WebMapping-Anwendung OpenLayers. Diplomarbeit. URL: http://www.smartmapbrowsing.org/diplomarbeit_EmanuelSchuetze.pdf [Stand 2011-03-10].
Sester, M. Prof. Dr.-Ing habil. 2000. Maßstabsabhängige Darstellungen in digitalen räumlichen Datenbeständen. Habilitationsschrift.
Smart, W. M. 1960. *Text-book on spherical astronomy.* [Repr.]. Cambridge: University Press.
van Oosterom, P. 1993. Reactive data structures for geographic information systems*:* Univ., Diss.--Leiden, 1990. *@Reactive data structures for geographic information systems, Petrus Johannes Maria van Oosterom.* Online im Internet: URL: http://www.gbv.de/dms/bowker/toc/9780198233206.pdf.
van Oosterom, P. 1995. The GAP-tree, an approach to "On-the-Fly" Map Generalization of an Area Partitioning. *GIS and Generalization: Methodology and Practice of GISDATA.*