Master Thesis

im Rahmen des

Universitätslehrganges „Geographical Information Science & Systems“

(UNIGIS MSc) am Interfakultären Fachbereich für GeoInformatik (Z_GIS)

der Paris Lodron-Universität Salzburg

zum Thema

Vector Tiles als Datenbasis für OGC-Darstellungsdienste

WMS und WMTS

vorgelegt von

Dipl. Ing. (FH) Gerd Jünger

104577, UNIGIS MSc Jahrgang 2016

Zur Erlangung des Grades

„Master of Science (Geographical Information Science & Systems) – MSc(GIS)”

Halle (Saale), 21.11.2018

Eigenständigkeitserklärung

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

Halle (Saale), 21.11.2018

Ort, Datum

Unterschrift

Kurzfassung

Mit der Zunahme des Erfordernisses Geodaten mit einer hohen clientseitigen Flexibilität performant über das Internet zugänglich zu machen, wächst das Interesse an effizienten und vielseitig nutzbaren Datenaustauschformaten. Rasterdaten, die zur Darstellung von Geodaten weit verbreitet sind, erfordern einen hohen Speicherbedarf sowie einen höheren Datendurchsatz und sind in ihrer Flexibilität eingeschränkt. Originäre Vektordaten, z. B. über WFS bereitgestellt, bieten eine hohe clientseitige Flexibilität, die entsprechenden Dienste sind jedoch oft sehr langsam. An dieser Stelle bieten sich Vector Tiles zur Verteilung von Geodaten an, da sie die Vorteile von Raster- und Vektordaten verbinden können.

Da Vector Tiles in den verbreiteten Desktop Clients bisher nicht oder nur unzureichend implementiert sind, entstand die Idee zu untersuchen ob es möglich und sinnvoll ist, Geodaten in Vector Tiles zur direkten Nutzung anzubieten und parallel eine Beschleunigung von OGC Darstellungsdiensten zu erreichen, indem diese Vector Tiles als deren Datenquelle eingesetzt werden. Im Rahmen der detaillierten Recherchen zur Verwirklichung dieser Idee wurde eine umfassende, strukturierte Grundlagendarstellung zu Vector Tiles erarbeitet und aktuelle Software zu ihrer Verwendung ermittelt. Basierend auf den daraus gewonnenen Erkenntnissen erfolgte die Umsetzung von zwei Szenarien eines WMS basierend auf Vector Tiles. Ein Szenario nimmt den Umweg über mit *TileServer GL* gerenderte Raster Tiles und wird über *Mapproxy* bereitgestellt. Die zweite Lösung nutzt die direkte Einbindung von Vector Tiles in *MapServer* über die *GDAL*-Bibliothek. Diese ist aktuell jedoch nur sehr eingeschränkt umgesetzt. Die Performance der beiden Szenarien wurde in Lasttests mit der von zwei herkömmlichen WMS basierend auf *Shape*-Dateien verglichen.

Im Ergebnis wurde festgestellt, dass die Performance der WMS mit der untersuchten Software nicht eindeutig verbessert werden konnte. Beim der Verwendung von *TileServer GL* und *Mapproxy* entscheidet die Schwerpunktsetzung bei der Interpretation der Ergebnisse über die Beantwortung der Frage. Bei dem Szenario mit *MapServer* verhinderten die aktuell noch unzureichende Einbindung der *GDAL*-Bibliothek und deren eingeschränkte Funktionalität einen effektiven Zugriff auf die Daten. Darüber hinaus ergeben sich beim ersten Szenario entscheidende Nachteile, wie der Wegfall von Attributinformationen. Mit Blick auf die aufgetretenen Probleme und festgestellten Einschränkungen, verbunden mit der aktuell stark voranschreitenden Entwicklung bei Vector Tiles ist ein produktiver Einsatz der untersuchten Lösungsansätze nicht empfehlenswert. Es ist davon auszugehen, dass nach Entwicklung eines offenen Standards für Vector Tiles mittelfristig praktikablere und performantere Lösungen zu deren Einbindung als Datengrundlage in einen Darstellungsdienst möglich werden. Diese Richtung ist in einem ersten Schritt anhand der direkten Einbindung von Vector Tiles in *MapServer* abzulesen.

Abstract

With the increase in the need to make geodata accessible via internet with a high client-side flexibility, interest in efficient and versatile data exchange formats is growing. Raster data, which are widely used for the representation of geodata, requires a high storage capacity and a higher data throughput and are limited in their flexibility. Original vector data, e.g. provided via WFS, offering a high client-side flexibility. The services are, however, often very slow. At this point, Vector Tiles can be used to distribute geodata because they can combine the advantages of raster and vector data.

Since Vector Tiles are not or only insufficiently implemented in popular desktop clients, the idea was born to investigate whether it is possible and reasonable to offer geodata in Vector Tiles for direct use and to achieve an acceleration of OGC display services in parallel by using these Vector Tiles as data source. As part of the detailed research for the realization of this idea, a comprehensive, structured basic presentation of Vector Tiles was developed and current software for their use was determined. Based on the lessons learnt two scenarios of a WMS based on Vector Tiles were implemented. One scenario takes the detour via raster tiles rendered with *TileServer GL* and is provided via *Mapproxy*. The second solution uses direct integration of Vector Tiles into *MapServer* via *GDAL* library. However, this is currently only implemented to a very limited extent. The performance of the two scenarios was compared in load tests with two conventional WMS based on *shape* files.

As a result, it was determined that the performance of WMS could not be clearly increased with the examined software. When using *TileServer GL* and *Mapproxy* the focus of the interpretation of the results decides on the answer to the question. In the scenario with *MapServer* the currently insufficient integration of *GDAL* library and its limited functionality prevented an effective access to the data. In addition, the first scenario has decisive disadvantages, such as the omission of attribute information. With regard to the problems that have occurred and the limitations that have been established, combined with the current strong progress in the development of Vector Tiles, a productive use of the investigated solution approaches is not recommendable. It can be assumed that after the development of an open standard for Vector Tiles, more practicable and better performing solutions will be published. This will serve as a data basis for a presentation service in the medium term. The direct integration of Vector Tiles into *MapServer* can be seen as first step in this direction.

Inhalt

Abbildungen

Tabellen

Abkürzungen

BKG	Bundesamt für Kartographie und Geodäsie
BLOB	Binary Large Object
BSD	Berkeley Software Distribution
CDB	OGC Common DataBase
CRS	coordinate reference system
CSW	Catalogue Services for the Web
DGIWG	Defense Geospatial Information Working Group
DLM1000	Digitales Landschaftsmodell 1:1 000 000
DLM250	Digitales Landschaftsmodell 1:250 000
ECON	eC Object Notation
EPSG	European Petroleum Survey Group Geodesy
FME	Feature Manipulation Engine
GDAL	Geospatial Data Abstraction Library
GeoJSON	Geografic JSON
GEOS	Geometry Engine Open Source
glTF	GL Transmission format
GML	Geography Markup Language
GNU	Softwareprojekt für unixähnliches Betriebssystem - "GNU’s Not Unix"
GPL	General Public License
GPU	Graphics Processing Unit, Grafikprozessor
I3S	Indexed 3D Scene Layer (ESRI)
ISC	Internet Systems Consortium
JSON	JavaScript Object Notation
JSONP	JSON mit Padding
KVP	Key Value Pair
LGPL	Lesser General Public License
LOD	Level of detail
MIME	Multipurpose Internet Mail Extensions
MIT	Massachusetts Institute of Technology
MS4W	MapServer 4 Windows
MVT	Mapbox Vector Tiles
NoSQL	Not only SQL Datenbank
OGC	Open Geospatial Consortium
OGR	OGR Simple Features Library
OSciM	OpenScienceMap
OSGeo	Open Source Geospatial Foundation
OWC-6	OGC Web Services, Phase 6
PBF	Protocol Buffer
REST	Representational State Transfer
SDK	Software Development Kit
SLD	Styled Layer Descriptor
SQL	Structured Query Language
TMS	Tile Map Service
TOML	Tom's Obvious, Minimal Language
TopoJSON	Topografic JSON
UMS	Unified Map Service
URL	Uniform Resource Locator
URN	Uniform Resource Name
UTF	Unicode Transformation Format
UTM32	Universal Transverse Mercator, Meridianzone 32
WCS	Web Coverage Service
WFS	Web Feature Service
WGS 84	World Geodetic System 1984 (EPSG 4326)
WKB	Well-known Binary
WMS	Web Map Service
WMS-C	WMS Tile Caching
WMTS	Web Map Tile Service
XML	Extensible Markup Language
XYZ	Indizierungsschema für Kacheln, z.B. von OpenStreetMap und Google Maps verwendet
YAML	vereinfachte Auszeichnungssprache: YAML Ain't Markup Language

1 Einleitung

Das Internet erlangt eine immer stärkere Bedeutung für die Bereitstellung räumlicher Daten (Yang 2005). Über die OGC-Darstellungsdienste Web Map Service (WMS) (de la Beaujardiere 2006) und Web Map Tile Service (WMTS) (Masó, Pomakis, et al. 2010) bereitgestellte Karten nehmen dabei eine wichtige Stellung ein. Sie sind bereits seit vielen Jahren offizieller Standard und werden aus diesem Grund von einer Vielzahl von Clients unterstützt.

Eine Vielzahl geografischer Daten liegt im Original in Vektorformaten vor. Bei deren Bereitstellung über einen WMS müssen diese bei jeder Nutzeranfrage in Bilddaten umgewandelt werden, was eine hohe Serverlast nach sich zieht (Shekhar et al. 2001). Um die Serverlast zu reduzieren und die Antwortzeiten auf Anfragen an den Service zu beschleunigen besteht die Möglichkeit, über WMTS vorprozessierte Rasterkacheln zur Verfügung zu stellen, die sowohl server- als auch clientseitig gecacht werden können (Kalberer 2017b). Durch die Vorprozessierung entsteht jedoch der Nachteil, dass der Rechen- sowie der Speicheraufwand, vor allem bei gleichzeitigem Angebot mehrerer Stile und Projektionen, sehr hoch werden (Blanc et al. 2016) und die Darstellung starr ist. Der Vorteil von Vektordaten als Grundlage für einen WMS liegt bei deren geringerem Speicherbedarf.

An moderne Darstellungsdienste bzw. die für diese bereitgestellten Daten ergeben sich folgende Anforderungen:

1. niedriger Speicherbedarf
2. schnelle Antwortzeiten
3. geringe Serverlast
4. Flexibilität in der Anwendung
5. Verfügbarkeit von Attributinformationen

Angesichts dieser Erfordernisse und der möglichen Verbindung der Vorteile von Vektordaten und Raster Tiles lohnt sich ein Blick auf Vector Tiles als alternative Datengrundlage für Darstellungsdienste. Vector Tiles werden im Internet bereits vielfältig zur Bereitstellung geografischer Daten verwendet (z. B. OpenStreetMap, Google Maps, Bing Maps u. a.). Analog zu WMTS werden die Daten für vordefinierte Maßstäbe vorprozessiert und in einer pyramidenähnlichen Kachelstruktur abgelegt (Mapbox 2016; Blasby & Hocevar 2016a). Dabei sind Vector Tiles in der Regel kleiner als die vergleichbaren Bildkacheln, woraus ein schnellerer Datentransfer mit geringerer Auslastung der Bandbreite resultiert (GeoServer Contributors 2018). Im Gegensatz zu WMTS sind mit einem Satz von Vector Tiles mit clientseitigem Rendern beliebig viele Darstellungsvarianten möglich (Blasby & Hocevar 2016a; Roth et al. 2016). Durch das clientseitige Rendering wird die Serverlast erheblich reduziert (Sloup & Přidal 2016). Jedoch scheitert die direkte Nutzung von Vector Tiles derzeit häufig noch an deren mangelnder Unterstüzuung durch verbreitete GIS-Clients (Sloup & Přidal 2016; Cavazzi 2018). Diese müssen demnach erst auf die Erfordernisse von Vector Tiles erweitert werden (Defense Geospatial Information Working Group 2016). Erschwert wird deren Weiterentwicklung dadurch, dass aktuell kein offener Standard existiert (Ingensand et al. 2016).

Aus der Überlegung, einerseits die Vorteile von Vector Tiles bei der Bereitstellung von Geodaten nutzen zu können, andererseits aber auch weiter herkömmliche OGC Darstellungsdienste anzubieten, entstand der Forschungsschwerpunkt dieser Arbeit. Es soll untersucht werden, ob es möglich ist, Vector Tiles als Datengrundlage für diese Webservices zu nutzen (Abbildung 1). Aufgrund deren Datenformat und -struktur (Ablage) ist zu erwarten, dass der Zugriff auf Daten in Vector Tiles effizienter erfolgt, als bei der Nutzung

originärer Vektordaten. Diese Überlegung basiert darauf, dass die Geometrien in Vector Tiles der Zoomstufe entsprechend bereits generalisiert sind und das *Clippen* großer Features bei Anfragen an den Service entfällt oder vereinfacht wird. Die daraus resultierende Reduktion der Zugriffszeiten wäre der größte Vorteil bei der Nutzung vorprozessierter Vektorkacheln als Datengrundlage für einen WMS. Darüber hinaus werden Vector Tiles in Zukunft immer größere Verbreitung bei der direkten Nutzung von Geodaten in Clients erlangen. Eine gleichzeitige Nutzung von Vector Tiles für die direkte Bereitstellung und als Datengrundlage für WMS würde eine doppelte Datenhaltung mit der Gefahr von Redundanz und dem erhöhten Aufwand für die Datenaufbereitung vermeiden.

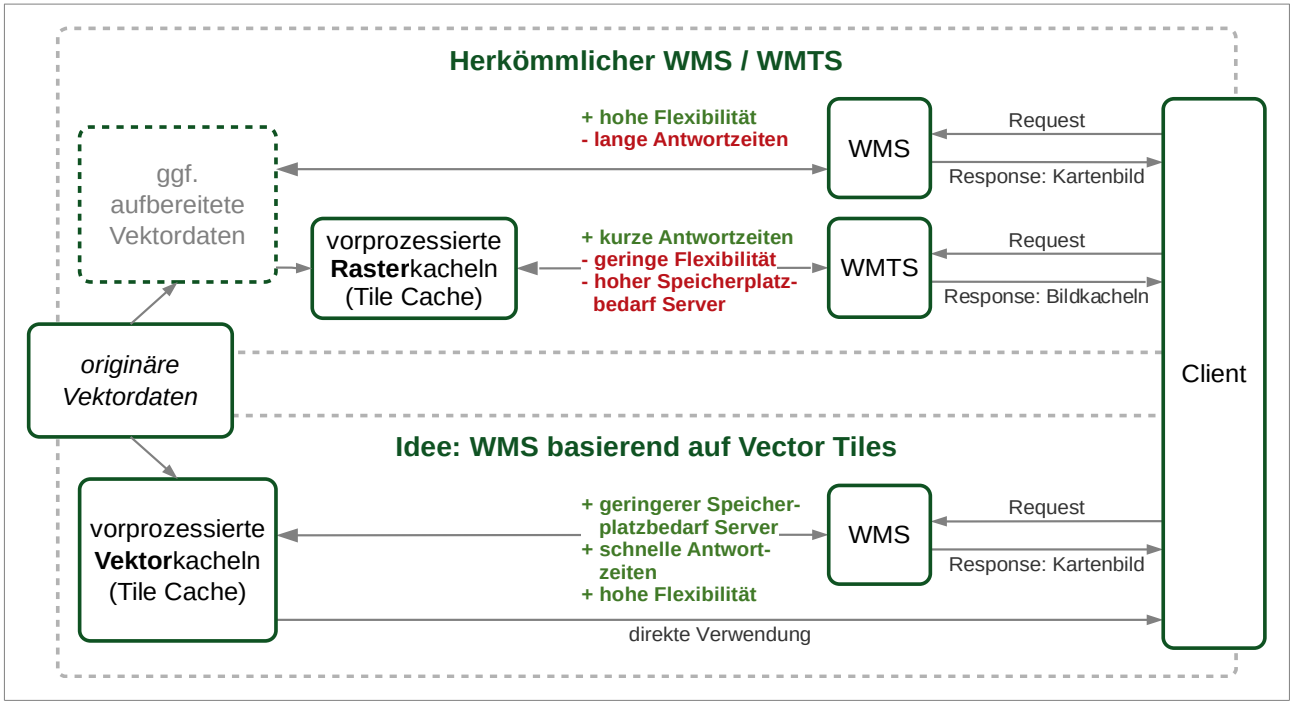


Abbildung 1 Gegenüberstellung von herkömmlichen OGC Darstellungsdiensten und der Idee eines WMS basierend auf Vector Tiles

1.1 Forschungsziel

Das zentrale Ziel der Arbeit ist es, eine Schnittstelle zu finden, mit welcher sich Vector Tiles als Datengrundlage für die OGC-Darstellungsdienste WMS und ggf. WMTS einsetzen lassen. Dabei soll der Schwerpunkt bzw. das Hauptziel auf der Verwendung für einen WMS liegen. Dies liegt vor allem darin begründet, dass WMS über eine höhere clientseitige Flexibilität und damit einen weiteren Einsatzbereich verfügt als WMTS.

Zur Feststellung der Umsetzbarkeit wurden folgende Untersuchungen durchgeführt:

1. Welche Konzepte und möglichst offene Datenformate stehen zur Speicherung von Daten in Vector Tiles zur Verfügung?
2. Welche aktiv entwickelte Software (möglichst *Open Source*) steht zur Generierung und Verarbeitung von Vector Tiles zur Verfügung und welche Methoden nutzt diese?
3. Auf welchem Weg können Vector Tiles in Geodatenservern als Grundlagendaten eingebunden werden?
4. Wie werden Attributinformationen in Vector Tiles gespeichert und auf welchem Weg können die in den Vector Tiles enthaltenen Attributinformationen in Darstellungsdiensten verfügbar gemacht werden?

Im Rahmen dieser Arbeit soll, sofern eine technische Umsetzung der in Abbildung 1 dargestellten Idee eines Darstellungsdienstes basierend auf Vector Tiles möglich ist, weiterhin geklärt werden, ob und wie die Performance eines WMS bzw. WMTS auf diesem Weg gegenüber einer klassischen (nicht aufbereiteten) Vektordatenquelle verbessert werden kann. Als Nebenprodukt dieser Arbeit entsteht so im Optimalfall ein Workflow, in welchem alle Schritte von der Aufbereitung der Originaldaten zu Vector Tiles bis zu deren Einbindung in einen Darstellungsdienst nachvollzogen und beispielhaft technisch umgesetzt werden.

Mit der Arbeit sollen die derzeit verfügbaren Tools und Methoden zur Aufbereitung und Darstellung von Geodaten aus Vector Tiles mit Blick auf das zentrale Ziel dieser Arbeit untersucht werden. Dabei sind die Antworten auf die Fragen 1. und 2. (s. o.) voraussichtlich durch Recherche von Literatur und Software ableitbar. Zusätzlich sollen sie durch eigene Untersuchungen untermauert bzw. vertieft werden, z. B. wie sich unterschiedliche Vector Tile Formate auf die enthaltenen Geometrien auswirken. Zur Beantwortung der Fragen 3. und 4. ist es erforderlich, basierend auf Spezifikationen und Dokumentationen zu den recherchierten Datenformaten und der gefundenen Software, Wege für eine praktische Machbarkeit der theoretisch erarbeiteten Ideen zu untersuchen. Hier ist entsprechend der Voruntersuchungen absehbar, dass derzeit keine *fertige* Lösung für eine direkte Einbindung von Vectot Tiles in einen Darstellungsdienst zu erwarten ist.

1.2 Vorgehensweise

In einem ersten Schritt werden vorhandene Standards, *de facto* Standards und Spezifikationen zu Vector Tiles sowie die verfügbaren Datenformate und Speicherkonzepte analysiert. In einem zweiten Schritt erfolgt eine Recherche nach Software (Server, Caching Software, Renderer, Konvertierungsprogramme), um basierend auf diesen Ergebnissen Produkte zu ermitteln, die geeignet sind, Vector Tiles zu erzeugen und als Datengrundlage für Darstellungsdienste bereitzustellen. Teilaspekte sollen bei beiden Schritten die Verbreitung (Datenformate), Verfügbarkeit (*Freeware*, *Open Source*) und die Aktualität (und Aktivität der Entwicklergemeinschaft) darstellen, um zu vermeiden, dass auslaufende oder nicht weiter entwickelte Formate und Produkte verwendet werden. Recherchen zu Formaten, Software usw. bzw. deren Aktualisierung erfolgten bis 31.08.2018. Spätere Neuerungen werden im Rahmen dieser Arbeit nicht berücksichtigt.

Um eine Abschätzung von Rechen- und Speicheraufwand zu ermöglichen, werden nach Festlegung auf geeignete Vector Tile-Formate der Zeitaufwand für das Erstellen der Kacheln sowie der benötigte Speicherplatz (ggf. mit dem Vergleich unterschiedlicher Datenformate) berechnet und sowohl untereinander verglichen als auch zusätzlich mit dem Aufwand für Rasterkacheln mit gleichartigem Inhalten gegenübergestellt.

Wenn eine Software-Architektur ermittelt werden kann, die es ermöglicht Vector Tiles als Datengrundlage für eine OGC-Darstellungsdienst einzusetzen, werden Performancetests durchgeführt. Dabei wird ein WMS mit einer originären Datenquelle und mit vergleichbarer Symbolisierung bei identischer Hardware zu Vergleichszwecken herangezogen.

1.3 Nicht behandelte Themen und unberücksichtigte Inhalte

Eine tiefergehende Untersuchung der Verbreitung von Desktop-GIS-Systemen mit Blick auf deren Fähigkeit, Vector Tiles oder OGC-Darstellungsdienste (v. a. WMTS direkt) einzubinden, erfolgt nicht. Nachfolgend werden ausschließlich die Clients ArcGIS Pro 2.2, QGIS 2.18 und QGIS 3.2 beispielhaft für Untersuchungen herangezogen.

Im Rahmen dieser Arbeit werden Vector Tiles nur mit Blick auf 2D-Daten und für einen Zeitschnitt untersucht. Erfordernisse, die sich gegebenenfalls aus der Nutzung von 3D-Daten und sogenannte *Moving Features* (Cavazzi 2018) ergeben würden, bleiben unberücksichtigt. An ausgewählten Stellen werden jedoch Besonderheiten von 3D-Tile-Formaten der Vollständigkeit halber erwähnt.

Verschiedene *Caching*-Techniken werden z. B. bei Loechel & Schmid (2012), dort jedoch für Rasterdaten, dargestellt. Im Rahmen der hier vorliegenden Arbeit findet keine tiefergehende Untersuchung unterschiedlicher *Caching*-Mechanismen (mit Blick auf Vector Tiles) statt. Es werden lediglich die unterschiedlichen Ablageformen (Verzeichnisstruktur, Datenbanken usw.) vorgestellt, da sie für den Zugriff auf die gespeicherten Vector Tiles von Bedeutung sind.

Erfordernisse mit Blick auf die Topologie beim Erstellen von Vector Tiles sowie mögliche Probleme beim Rendern bei fehlerhaften Topologien werden aufgeführt. Damit verbunden findet eine Darstellung von Besonderheiten statt, die sich mit Blick auf die Symbolisierung und Beschriftung bei der Darstellung von Daten aus Vector Tiles im Vergleich zu originären Vektordaten ergeben. Es werden jedoch bei beiden Sachverhalten keine Lösungsvorschläge für potenzielle Problemfälle recherchiert oder erarbeitet.

2 Stand von Forschung und Technik

2.1 Forschungsarbeiten

Eine wesentliche Grundlage zum Einsatz von Vector Tiles bildet die Arbeit von Antoniou et al. (2009). Dort wird beschrieben, dass über die Koordination aller beteiligten Teile einer Client-Server-Architektur eine effiziente Möglichkeit zur Übertragung von Vektoren geschaffen werden kann. Dabei wird gezeigt, wie die von Rasterdaten bekannte, kachelbasierte Methode implementiert werden kann, um die Besonderheiten der Vektordatenübertragung zu lösen. Wichtige Grundlagen werden in den *OGC Testbeds 12* und *13* herausgearbeitet (Balog & Houtmeyers 2017a; Cavazzi 2018). In deren Folge wird jedoch deutlich, dass dringend ein offizieller Standard für Vector Tiles erarbeitet werden muss, welcher mit der Veröffentlichung einer Spezifikation für einen *Unified Map Service* (UMS) einhergehen sollte (Cavazzi 2018). Ingensand et al. (2016) untersuchen mit Blick auf das *Swiss Federal Geoportal*[1] die Möglichkeit zur Bereitstellung von Vector Tiles unter Nutzbarkeit der verfügbaren Infrastruktur und Services (hier WMTS). Die Vector Tiles sollen hierbei jedoch nicht, wie in der vorliegenden Arbeit, als Datengrundlage für die vorhandenen Dienste verwendet werden. Die Autoren stellen in diesem Dokument überblicksmäßig Datenformate und zu berücksichtigende Fragen beim Einsatz von Vector Tiles vor. Diese werden ebenfalls von Yu et al. (2017) dargestellt und basierend auf den Grundlagen ein vollständiges Paket (Server und Client) entwickelt und getestet, um bestmögliche Performance mit Blick auf Ladegeschwindigkeit und Zuordnung zu erreichen.

Viele Forschungsarbeiten zielen auf die Erhöhung der Übertragungsgeschwindigkeit von Geodaten ab. Hier lag vor der zunehmenden Verbreitung von Vector Tiles der Schwerpunkt im Bereich Vektordaten auf der serverseitigen Generalisierung von Daten mit Begriffen wie *multi-resolution* oder *multiple representations* (Bertolotto & Egenhofer 2001; Yang 2005; Weibel & Dutton 2005; Du et al. 2008; Kozioł et al. 2014). In der Android Applikation *OpenScienceMap*[2] ist dieser Ansatz umgesetzt (Schmid et al. 2013; Dufilie & Grinstein 2014). Allgemein mit der Verbesserung der Performance bei der Nutzung von Vector Tiles beschäftigen sich u. a. Yu et al. (2017). Shang (2015) schlägt eine *Cloud Server Architektur* vor, um die Übertragung von Vector Tiles und die Skalierbarkeit des Dienstes zu verbessern. Einen Ansatz basierend auf einer verteilten *Not only SQL Datenbankumgebung (NoSQL)* für Kartenservices mit hohem Datenvolumen und niedriger Latenz stellen Wan et al. (2016) vor. Ein *virtual globe* basiertes Vektordatenmodell wird von Zhou et al. (2015) vorgeschlagen, um große Mengen mehrmaßstäblicher Vektordaten verwalten, visualisieren und analysieren zu können. Eine auf drei Schritten basierende Methode zur Kompression von Vektorkartendaten entwickelten Yang & Li (2009). Ein Framework zur Verbesserung der Leistungsfähigkeit von Vektorkartendiensten unter Berücksichtigung spezifischer Formate für Vektordaten und Symbologie, Vector-Tiling, räumliche Indexdienste und Generalisierung für Multi-Scale-Daten hat Gaffuri (2012) entwickelt und anhand eines Prototyps erfolgreich getestet. Die Berücksichtigung des geometrischen Zusammenhangs von räumlichen Daten bei Datenanforderungen eines Nutzers an Webdienste sehen Corcoran et al. (2012) als wesentliche Grundlage zur Reduzierung der Komplexität und Verbesserung der Performance. Eine weitere Möglichkeit zur Verbesserung der Performance bei der Übertragung von Vektordaten wurde von Corcoran et al. (2011) mit der progressiven Übertragung von Daten untersucht, wobei der Fokus darauf liegt, dass diese kombiniert view- und maßstabsbasiert erfolgt. View-basiert bedeutet hierbei, dass beim Verschieben des Kartenausschnittes nur die neu hinzukommenden Inhalte und maßstabsbasiert, dass

1 https://map.geo.admin.ch
2 http://www.opensciencemap.org

detailliertere Daten nicht komplett, sondern nur die Verfeinerungen gegenüber der vorherigen Zoomstufe nachgeladen werden (Taraldsvik 2012).

In weiteren Arbeiten werden potenzielle Probleme der Topologie von Features, die bei der Generalisierung entstehen können, untersucht und Algorithmen zur Lösung bzw. Vermeidung abgeleitet (Du et al. 2008). Speziell für Vector Tiles wurde analysiert, wie topologische Metadaten über Kachelgrenzen hinweg erhalten werden können (Antoniou et al. 2009; Nordan 2012). Li et al. (2017) stellen ein Vector Tile Datenmodell vor, welches die Beziehungen zwischen geografischen Merkmalen, Symboldarstellungen und Kartenwiedergaben berücksichtigt und mit dessen Hilfe visuelle Diskontinuitäten an Kachelgrenzen vermieden werden können.

Speziell für die Erfordernisse des Frameworks *Weave*[1] beschreiben Dufilie & Grinstein (2014) mit den sogenannten *Feathered Tiles* einen Algorithmus, welcher abweichend von regelmäßigen Kachelpyramiden, die Daten in unregelmäßige Kacheln mit gleicher Größe (in Bytes) unter Berücksichtigung der Bedeutung eines Knotens als dritte Dimension neben den X- und Y-Koordinaten aufteilt.

Zur Messung und Bewertung der Leistung eines Webdienstes liegen Arbeiten zu WMTS, WMS, Web Map Service-Caching (WMS-C) und Web Feature Service (WFS) vor (Jurk 2010; Loechel & Schmid 2012; Cibulka 2013; Giuliani et al. 2013; Guan et al. 2014). Mit speziellem Blick auf die Einhaltung der INSPIRE-Vorgaben testen Seip und Bill (2016) unterschiedliche Werkzeuge zum Messen verschiedener Parameter, welche die Leistungsfähigkeit von Webdiensten wiedergeben.

2.2 Stand der Technik

In Anlehnung an die Ausführungen bei Wikipedia (2018) erfolgte die Entwicklung und Implementierung von Vector Tiles wie nachfolgend beschrieben:

Nach Tomlinson (1990) wurde bereits in den 1960er Jahren im *Canada Information System (CGIS)* eine Segmentierung großer Karten in gleichmäßige Teile als *Morton Matrix* (Morton 1996) vorgenommen, um leistungsschwächeren Rechnern einen Stück-für-Stück-Zugriff auf umfangreiche Kartendaten zu ermöglichen. Dies kann als eine erste Form der Kachelung geografischer Vektordaten angesehen werden. In nachfolgenden Jahren wurden gekachelte Vektordatenstrukturen im *Wetlands Analytical Mapping System* (WAMS) (Pywell & Niedzwiadek 1980) sowie in *DELTAMAP* (Reed 1986) implementiert. Unter Anderem mit dem Einsatz in Google Maps im Jahr 2010 (Google Inc. 2010) und bei OpenStreetMap im Jahr 2013 (OpenStreetMap 2018b) erhalten Vector Tiles im Internet in den letzten Jahren zunehmende Bedeutung. Trotz dieser recht langen Entwicklungszeit existiert bisher kein offizieller Standard zum Kacheln und Bereitstellen von Vektordaten (Cavazzi 2018). Nordan (2012) beschreibt für Vector Tiles einen üblichen Weg bei deren Entwicklung und Einführung, wie für WMS und WMTS:

Phase 1: unterschiedliche Innovationen
Phase 2: verbreitete Übernahme der Technologie, welche zu *de facto* Standards führt
Phase 3: ein formeller Standardisierungsprozess

Für 2012 sieht Nordan noch Phase 1 als aktuell an. Mit Stand der hier vorliegenden Arbeit existieren bereits *de facto* Standards (OSGeo Wiki 2012a; Mapbox 2016; Google Inc. 2017b; Schwartz 2018; OpenStreetMap 2018a) und der formelle Standardisierungsprozess ist mit *OGC Testbed 12 und 13* (Balog & Houtmeyers 2017a; Cavazzi 2018) aktuell in der Entstehung begriffen. Aus dieser Situation resultiert derzeit eine Vielzahl

1 http://www.iweave.com/index.html

an Konzepten zur Nutzung von Vector Tiles. Diese werden in Kap. 4.2 detailliert beschrieben, um darauf basierend Ableitungen für deren Nutzung im Kontext der Zielstellung dieser Arbeit zu ermöglichen.

Die OGC-Darstellungsdienste WMS und WMTS liegen aktuell mit Stand Mai 2006 (WMS 1.3.0) bzw. April 2010 (WMTS 1.0.0) vor. Von Cavazzi (2018) wird vorgeschlagen, diese Standards nicht mit eigenständigen Spezifikationen weiterzuentwickeln, sondern diese Dienste mit WFS und dem Web Coverage Service (WCS) und unter Berücksichtigung der Erfordernisse zur Nutzung von Vector Tiles zum UMS zusammenzufassen. Mit Version 2.3 (Stand 04.05.2018) haben Vector Tiles mit dem *MVT*-Treiber und der Erweiterung des *MBTiles*-Treibers in die GDAL-Bibliothek (Kap. 5.11) Einzug gehalten. Für *GeoPackage* (Kap. 4.6.2.2) wurde am 28.06.2018 eine *Vector Tiles Extension* zur Implementierung von Vector Tiles veröffentlicht. Die beiden letzten Informationen verdeutlichen einmal mehr, dass im Bereich Vector Tiles aktuell eine sehr aktive Entwicklung stattfindet.

3 Grundlagen

3.1 OGC Darstellungsdienste

Alle in den beiden nachfolgenden Kapiteln gemachten Ausführungen basieren, sofern nicht anders gekennzeichnet, auf der *OpenGIS® Web Map Server Implementation Specification* (de la Beaujardiere 2006) (Kap. 3.1.1) und dem *OpenGIS® Web Map Tile Service Implementation Standard* (Masó, Pomakis, et al. 2010) (Kap. 3.1.2).

3.1.1 Web Map Service (WMS)

Ein *Web Map Service* (WMS) stellt fertig gerenderte Kartenbilder geografischer Daten als Bild (z. B. PNG, JPG) oder auch als Vektorgrafik (SVG oder WebCGM) zur Verfügung, übermittelt jedoch keine Kartendaten im eigentlichen Sinn. Die Parameter eines WMS können über einen *GetCapabilities-Request* abgefragt und anhand dieser ein *GetMap-Request* für einen Kartenausschnitt erstellt werden, welcher das aus diesen Angaben resultierende Bild zurückgibt. Die Symbolisierung wird entweder durch vom WMS vorgegebene *LAYERS* in Kombination mit *STYLES* oder, bei SLD-fähigen WMS, durch benutzerdefinierte Angaben mit Hilfe von *Styled Layer Descriptoren (SLD)* im XML-Format (Lupp 2007) festgelegt. Das *Styling* kann dem WMS entweder über ein SLD-Dokument oder in einer POST-Anfrage als *StyledLayerDescriptor*-Element übermittelt werden. Das vom WMS zurückgegebene Bild kann in jedem CRS gerendert sein, welches der Dienst unterstützt (siehe *Capabilities-Response*). Optional kann ein WMS über *GetFeatureInfo*-Anfragen Attributdaten zu den dargestellten Daten zur Verfügung stellen.

Da die Kartenbilder eines WMS bei Anfrage gerendert werden, bietet sich ein WMS u. a. für dynamische also sich regelmäßig ändernde Daten an. Dieses wird jedoch mit einer geringeren Performance bzw. hoher Serverlast erkauft. Dieser Nachteil von WMS wurde versucht durch WMS-C zu reduzieren, womit ein WMS in die Lage versetzt werden soll, Bildkacheln zwischenzuspeichern oder sogar vorgenerierte Bilder anzubieten (OSGeo Wiki 2012b). WMS-C wurde durch WMTS ersetzt, wird jedoch noch von einigen Anwendungen, z. B. *GeoServer* (Kap. 5.5), weiterhin unterstützt.

3.1.2 Web Map Tile Service (WMTS)

Der *Web Map Tile Service* (WMTS) wurde entwickelt, um gekachelte Kartendaten schnell zurückgeben zu können. Die Spezifikation schlägt zu diesem Zweck vor, vorgerenderte Kacheln in einem *Cache* abzulegen und somit bei einer Anfrage an den WMTS keine Geo-Verarbeitung oder Bildbearbeitung mehr zu benötigen. Darüber hinaus ist das Rendern von noch nicht im *Cache* vorhandenen Bildkacheln *on the fly* möglich.

Wie beim WMS sind auch beim WMTS die Informationen zum Service über eine *GetCapabilities*-Anfrage abrufbar. Aus dem *Response* können die vorhandenen Layer, Stile, Rückgabeformate und Tilematrixsets inklusive Projektionen abgelesen werden. Anhand dieser Infos werden die Kacheln per *GetTile* abgerufen (anlog zu *GetMap* beim WMS). Darüber hinaus kann ein WMTS optional über *GetFeatureInfo* mit einem *FeatureInfo-Response* Attributdaten der dargestellten Daten verfügbar machen. Ein WMTS gibt mit jeder Kachel eine vorgefertigte Kombination eines einzelnen Layers bzw. einer Gruppe von Layern mit einer Symbolisierung in einem CRS zurück. Werden diese Kacheln in einem Bildformat angefordert, welches Transparenz unterstützt (z. B. PNG), können durch Überlagerung im Client verschiedene Kachelsets miteinander kombiniert werden.

3.2 Beispieldaten

Für Testzwecke im Rahmen dieser Arbeit werden zwei freie Geodatensätze des BKG verwendet: das *Digitale Landschaftsmodell 1:1.000.000* (DLM1000) und das *Digitale Landschaftsmodell 1:250.000 (*DLM250). Beide sind über die Internetseite des *Dienstleistungszentrums des Bundes für Geoinformation und Geodäsie*[1] (nachfolgend synonym mit *BKG* bezeichnet) verfügbar.

In den *Digitalen Landschaftsmodellen* werden die topografischen Objekte als Vektordaten beschrieben, wobei die Objektauswahl und die Art, wie die Objekte zu bilden sind, im ATKIS-Objektartenkatalog festgelegt sind (Bundesamt für Kartographie und Geodäsie 2018). Nachfolgend werden die wichtigsten Informationen zu den beiden Beispieldatensätzen, abgeleitet aus den Daten bzw. übernommen von der Internetseite des BKG, aufgelistet:

- ***Digitales Landschaftsmodell 1 : 1.000.000***
  - Stand: 19.03.2018
  - 28 Geometrielayer (6 Punkt, 9 Linie, 13 Fläche)
  - 258.453 Features (41.128 Punkt, 204.014 Linie, 13.311 Fläche)
  - Generalisierung entsprechend *Signaturenkatalog (SK1000) der Topographischen Karte 1:500.000 (TK500)*
  - Genauigkeit für die wichtigsten punkt- und linienförmigen Objekte ±250 m
- ***Digitales Landschaftsmodell 1 : 250.000***
  - Stand: 19.03.2018
  - 35 Geometrielayer (8 Punkt, 11 Linie, 16 Fläche)
  - 751.318 Features (76.317 Punkt, 547.180 Linie, 127.821 Fläche)
  - Generalisierung entsprechend *Signaturenkatalog (SK250) der Digitalen Topographischen Karte 1:250.000 (DTK250)*
  - Genauigkeit für die wichtigsten punkt- und linienförmigen Objekte ±100 m

Eine detaillierte Auflistung aller Layer ist in Anlage 1 beigefügt. Die Daten wurden im *Shape*-Format im CRS Universal Transverse Mercator, Meridianzone 32 (UTM32) (EPSG 25832) als aktuellem Referenzsystem der deutschen Landesvermessung übernommen. In den Layern sind neben weiteren Attributen folgende wesentliche Attribute enthalten:

- OBJART = Kennung nach ATKIS-Objektartenkatalog
- OBJART_TXT = Objektart nach ATKIS-Objektartenkatalog
- OBJID = eindeutiger Objektidentifikator
- NAM = Name des Objektes, z. B. Ortsnamen
- BEZ... = Bezeichnungen

Die genaue Beschreibung aller Layer inklusive der Attributdaten ist in den Dokumentationen der Datensätze DLM1000[2] und DLM250[3] online abrufbar.

Im bisherigen Workflow werden die originären Vektordaten beim *BKG* in einer *ESRI File Geodatabase* aufbereitet als Datenquelle für den WMS verwendet. Mit Blick auf die in dieser Arbeit geplanten Untersuchungen wurde festgestellt, dass in *GeoServer* theoretisch eine *ESRI File Geodatabase* über GDAL/OGR als Datenquelle genutzt werden könnte, wie es mit *MapServer*[4] möglich ist. Es konnte jedoch keine

1 http://www.geodatenzentrum.de/geodaten/gdz_rahmen.gdz_div
2 http://sg.geodatenzentrum.de/web_download/dlm/dlm1000/dlm1000.pdf
3 http://sg.geodatenzentrum.de/web_download/dlm/dlm250/dlm250.pdf
4 https://mapserver.org/input/vector/filegdb.html

entsprechende Anleitung zur Einbindung recherchiert werden. Die einzigen beiden Hinweise im Internet[1,2] sprechen gegen eine Umsetzbarkeit, so dass hier die Daten des DLM im *Shape*-Format verwendet werden, um beide Programme einsetzen zu können.

3.3 Softwarelizenzen

Für die Beachtung von Rechten bei deren Nutzung, ist die Lizensierung von Software von entscheidender Bedeutung. *Open Source*- und *Freeware*-Projekte sind bei gleicher Eignung anderen Produkten vorzuziehen. Aus diesem Grund wurde zusätzlich die Lizenzinformation bei allen Sofwareprodukten (vgl. Kap. 5) dokumentiert. Kurze, aussagekräftige Informationen zu den einzelnen Lizenzen können der umfangreichen Liste auf GNU.org[3] entnommen werden. Tabelle 1 führt die in Kap. 5 recherchierten Lizenzen mit den offiziellen Links und Erläuterungen in deutscher Sprache auf *Wikipedia* auf. Alle aufgeführten Lizenzen sind *free* oder *permissive free*.

Tabelle 1 Liste recherchierter Softwarelizenzen

Lizenz	**Typ**	**offizieller Link**	**Wikipedia**
Apache Software License	permissive free software license	http://www.apache.org/licenses/LICENSE-2.0.html	https://de.wikipedia.org/wiki/Apache-Lizenz
BSD 3-Clause	permissive free software license	https://opensource.org/licenses/BSD-3-Clause	https://de.wikipedia.org/wiki/BSD-Lizenz
GNU GPL 3	free software license	https://www.gnu.org/licenses/gpl-3.0.en.html	https://de.wikipedia.org/wiki/GNU_General_Public_License
ISC	permissive free software license	http://www.isc.org/downloads/software-support-policy/isc-license/	https://de.wikipedia.org/wiki/ISC-Lizenz
LGPL 2.1	free software license	https://www.gnu.org/licenses/old-licenses/lgpl-2.1.en.html	https://de.wikipedia.org/wiki/GNU_Lesser_General_Public_License
MIT	permissive free software license	https://opensource.org/licenses/mit-license.php	https://de.wikipedia.org/wiki/MIT-Lizenz

1 http://osgeo-org.1560.x6.nabble.com/ESRI-File-Geodatabase-Support-td3790742.html
2 https://gis.stackexchange.com/questions/246453/add-file-personal-geodatabse-to-geoserver
3 https://www.gnu.org/licenses/license-list.html

4 Vector Tiles

4.1 Allgemeine Beschreibung

Cavazzi (2018 p. 8) definiert ein *Tile*, basierend auf der WMTS-Spezifikation (Masó, Pomakis, et al. 2010), wie folgt:

> *„a rectangular representation of geographic data, often part of a set of such elements, covering a spatially contiguous extent which can be uniquely defined by a pair of indices for the column and row along with an identifier for the tile matrix“*
>
> ***übersetzt:***
>
> *„eine rechteckige Darstellung von Geodaten, die oft Teil eines Satzes solcher Elemente ist und eine räumlich zusammenhängende Ausdehnung abdeckt, die eindeutig durch ein Paar von Indizes für die Spalte und Zeile zusammen mit einem Identifikator für die Kachelmatrix definiert werden kann“*

Hinter Vector Tiles steht demnach das von Rasterkacheln bekannte Konzept, geografische Daten anhand von pyramidenartig aufgebauten Gittern aufzusplitten (Abbildung 2). Nach Cavazzi (2018) können in einem Vector Tile ein oder mehrere Layer und die Attributinformationen der enthaltenen Features gespeichert werden. Daten in Vector Tiles werden, um späteren Darstellungsproblemen vorzubeugen, häufig mit einem *buffer* um die eigentliche Ausdehnung des Tiles erzeugt.

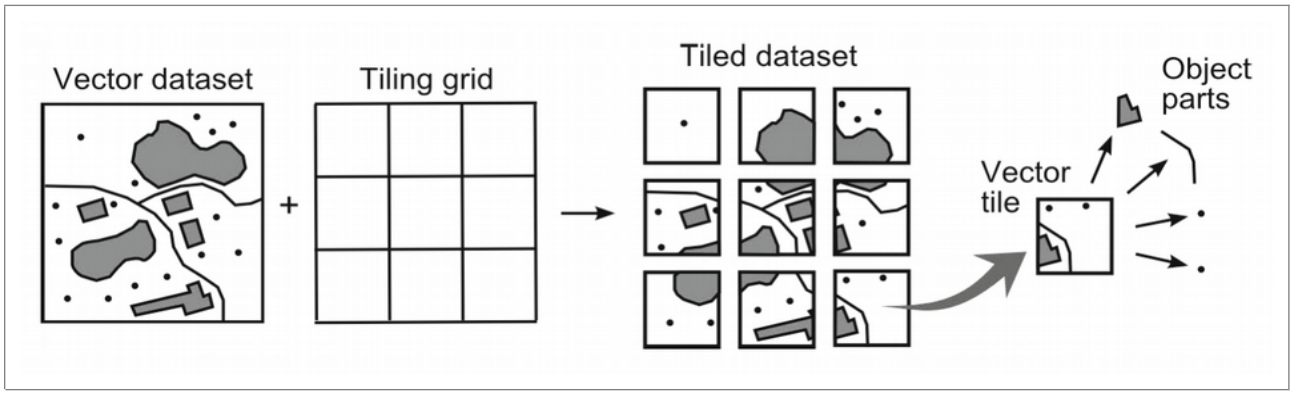


Abbildung 2 Prinzip des Kachelns von Vektor Daten (aus: (Gaffuri 2012))

Ordnance Survey (2018) listet tabellarisch wesentliche Unterschiede zwischen Vektor- und Rasterdaten auf. Bei Cavazzi (2018) werden diese erweitert und mit Blick auf die Vor- und Nachteile von Vector Tiles spezifiziert:

- ***Vorteile:***
  - Das Rendering erfolgt beim Client, so sind beliebige Kartenstile ohne Konfigurationsänderung des Servers und Neuberechnung der Kacheln möglich.
  - In der Regel kleiner als vergleichbares Raster Tile.
  - Da Vektordaten beim Client verfügbar, kann die Kartenauflösung ohne Erhöhung der Bandbreite erhöht werden.
  - Der Client hat Zugriff auf die tatsächlichen Feature-Informationen (Attribute (Kap. 4.4.2.2) und Geometrie).
  - Sie können in beliebige CRS projiziert werden, ohne dass Symbole und Beschriftungen verzerrt werden.

- ***Nachteile:***
  - Die geografischen Daten müssen (beim Client; Anm. des Autors) möglicherweise vorverarbeitet werden, um sie darzustellen.
  - Grafikkonflikte bzw. Informationsverluste können entlang von Kachelgrenzen auftreten.

Neben der individuellen Festlegung der Darstellung von Vector Tiles beim Client können serverseitig zusätzlich zu den Vector Tiles Informationen zu deren Symbolisierung bereitgestellt werden. So wird dem Client die Darstellung der Daten mit vorgefertigten Stilen bzw. die Verwendung einheitlicher Stile bei unterschiedlichen Clients erleichtert. Beim Generieren der Tiles können verschiedene Generalisierungsmechanismen zum Einsatz kommen, um den Inhalt der Zoomstufe entsprechend anzupassen (Abbildung 3).

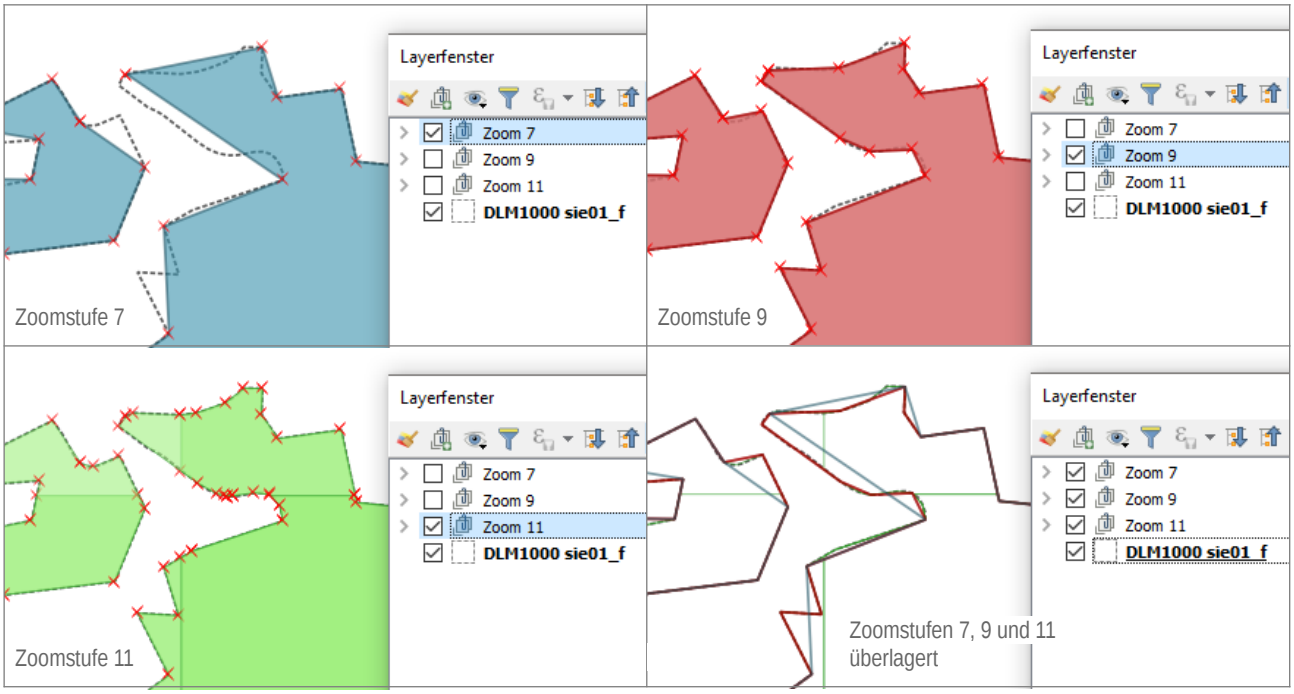


Abbildung 3 Generalisierung von Flächenumrissen niedriger Zoomstufen beim Erstellen von Vector Tiles im *GeoJSON*-Format mit *GeoServer* (EPSG 4326)
Kartengrundlage: Siedlungsgrenzen aus DLM1000 (© GeoBasis-DE / BKG 2018); Darstellung in *QGIS* 2.18

4.2 Begrifflichkeiten

4.2.1 Ansätze

Wie in Kap. 2.1 aufgeführt, existieren neben der ***Erzeugung klassischer Tiles***, wie sie Cavazzi (2018) beschreibt, unterschiedliche ***Ansätze***, um den Datentransfer bei Vektordaten zu verbessern. So beschreiben Corcoran & Mooney (2011) den Ansatz, beim Zoomen in eine Karte vom Server nur die Vertices einer Linie oder eines Polygons nachzuladen, die in der gröberen Auflösung der vorherigen Zoomstufe fehlen (*progressive transmission*). Mit mehreren Darstellungen eines Features, deren Genauigkeit über Generalisierung angepasst ist (*multi-resolution polygons*), kann die Anzahl der zu übertragenden Vertices je nach Zoomstufe reduziert werden (Yang 2005; Balog & Houtmeyers 2017b). Die Defense Geospatial Information Working Group (DGIWG) (2016) stellt einen Lösungsansatz für eine *GeoPackage* Extension mit Hilfe einer Metadatentabelle vor, in welcher Datentabellen beschrieben werden, die identische Daten, jedoch mit unterschiedlichen Zoomstufen (*Level of detail* = LOD) enthalten.

4.2.2 Konzepte

Für das klassische *Tiling* (Kap. 4.1) in Form von Kachelpyramiden (Kap. 4.3.1) kann nach render- und feature-basierten ***Konzepten*** unterschieden werden (Cavazzi 2018). Render-basierte Lösungen sind für die Visualisierung optimiert. Deswegen kommen hier Datenformate zum Einsatz, die rechentechnisch einfach zu verarbeiten sind und wenig Speicherplatz beanspruchen. Dabei werden jedoch z. B. Genauigkeitsverluste durch die Umwandlung in Bildschirmkoordinaten hingenommen (vgl. Kap. 4.5.1.1). Bei feature-basierten Konzepten liegt der Schwerpunkt auf Beibehaltung geografischer Koordinaten und Topologie und damit der Möglichkeit, kachelübergreifende Features korrekt zusammenzufügen und geografische Auswertungen durchzuführen.

4.2.3 Schemata

Cavazzi (2018) unterscheidet ***zwei Tiling Schemata***: ein *reguläres* (bei ESRI (2018) als *flaches Schema* bezeichnet) und ein *irreguläres, gewichtetes Schema*. Aus ***regulärem Tiling*** resultiert eine Kachelpyramide, in der alle Kacheln identisch groß sind und alle Kacheln, oft auch die Kacheln ohne Inhalte, erzeugt werden (Abbildung 5). In einem ***irregulären Schema*** wird die Kachelgröße z. B. anhand der Anzahl enthaltener Vertices festgelegt (Abbildung 4). Bei ESRI wird diese Aufteilung als *indiziertes Schema* bezeichnet und basiert auf einem Feature-Dichte-Index (ESRI 2018).

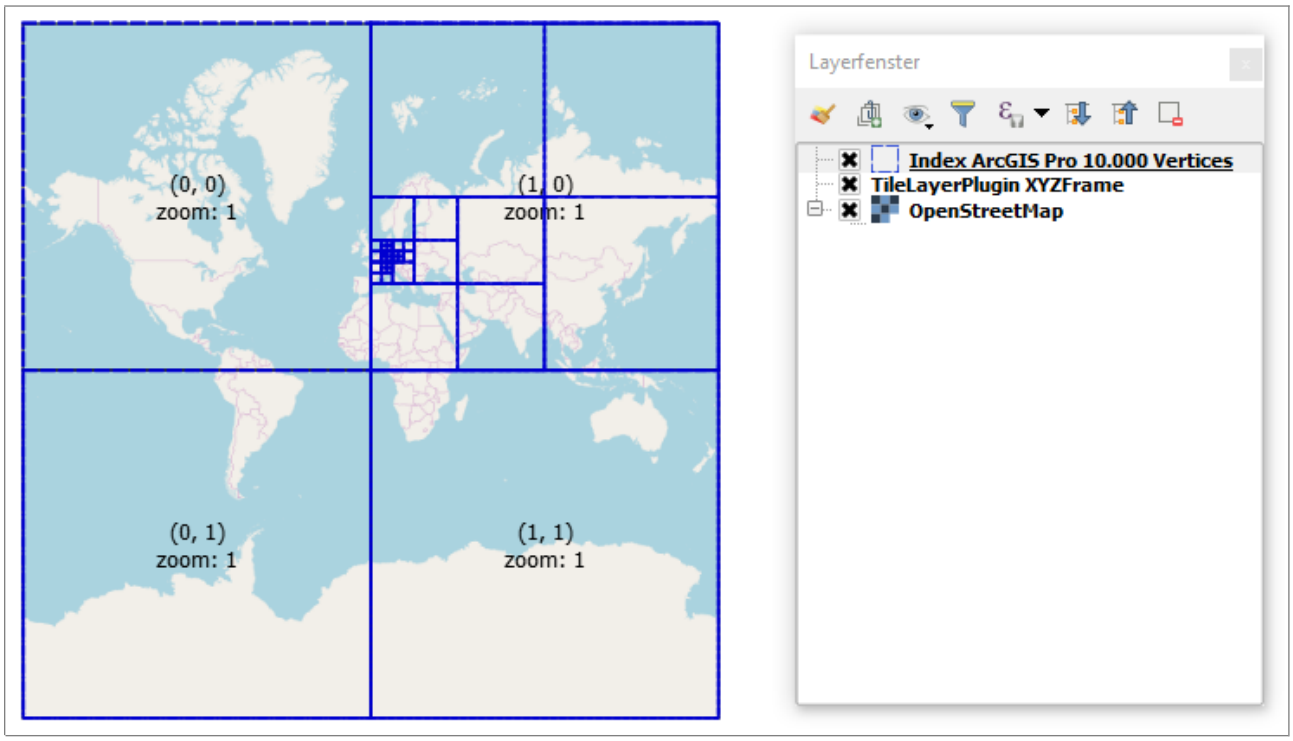


Abbildung 4 Quadtree erstellt mit dem Tool *Create Vector Tile Index* in ArcGIS Pro 2.1.2
basierend auf Layer sie01_f.shp aus DLM1000 des BKG (© GeoBasis-DE / BKG 2018)
Hintergrundkarte: © OpenStreetMap Mitwirkende CC-BY-SA
Einstellungen: Package for ArcGIS Online | Bing Maps | Google Maps, Maximum Vertex Count: 10.000

Im Bereich der offenen Standards werden derzeit v. a. reguläre Schemata eingesetzt. Eine Zwischenstellung zwischen regulärem und irregulärem Schema nimmt das in *ECERE GNOSIS*[1] implementierte Schema

1 http://ecere.ca/gnosis/

(*GNOSIS global tiling scheme*[1]) ein. Bei diesem Schema wird ein globales Kachelnetz definiert, welches an den Polen horizontal weniger Kacheln aufweist (Cavazzi 2018). Ein irreguläres Schema, welches eine angepasste räumliche Unterteilung ermöglicht, verwendet *Cesium® 3D Tiles™* (Cozzi 2015). Ein weiteres Tiling-Schema wird im *OGC Common DataBase Standard* (CDB) beschrieben. Hier wird die Welt in geodätische Tiles, welche durch Längen- und Breitengrad begrenzt sind, unterteilt (Reed 2017). Ausgewählte Tiling-Schemata werden in Kap. 4.3.2 detaillierter vorgestellt.

Die aus den beschriebenen Schemata resultierenden Kacheln werden in unterschiedliche Datenformate überführt. Diese werden in Kap. 4.5 beschrieben. Die Speicherung der einzelnen Kacheln erfolgt unabhängig vom Datenformat entweder in Verzeichnisbäumen, Datenbanken oder Packages/Archiven, auf welche in Kap. 4.6 näher eingegangen wird.

4.2.4 Metatiles

Das Metatile-Konzept stammt aus dem Bereich der Raster Tiles. Hierbei werden mehrere Kacheln zu größeren Metakacheln zusammengefasst. Der Hauptzweck des Einsatzes von Metatiles besteht darin, die Effizienz beim Rendern zu erhöhen, die Platzierung von Beschriftungen und Symbolen zu vereinfachen, bzw. dabei entstehende Konflikte zu reduzieren und die Speichereffizienz zu erhöhen (OpenStreetMap 2015; Tonnhofer 2017; GeoWebCache Contributors 2018). Diese Vorteile werden jedoch mit größeren Datenmengen pro Metatile (im Vergleich zu einzelnen Tiles) und damit Performanceverlusten bei deren Übertragung erkauft.

Um nicht komplette Metatiles übertragen zu müssen, berechnet z. B. *mod_tile*[2], ein System, welches auf *OpenStreetMap*-Servern eingesetzt wird, nicht ein großes Bild, sondern hängt die einzelnen Kacheln als Satz aneinander und kann sie so intern trennen und einzeln bereitstellen (OpenStreetMap 2015). Andere Konzepte, die große Zahl einzelner zu speichernder Tiles zu reduzieren, werden z. B. bei *MBTiles* (Kap. 4.6.2.1) oder den *multi-level tile pyramids* der *GNOSIS data stores* (Kap. 4.6.2.3) umgesetzt. Vorschläge zur Vermeidung von Konflikten beim Rendern von Vector Tiles werden in Kap. 4.4.2 vorgestellt.

4.3 Tiling

4.3.1 Kachelpyramiden

Die Idee und die Grundlagen für das Aufsplitten geografischer Daten in *Tile Sets* wurden nicht speziell für Vector Tiles entwickelt. Dies liegt auch darin begründet, dass bisher kein offener und verbreiter Standard für Vector Tiles existiert (Ingensand et al. 2016). Aus diesem Grund wird beim Kacheln (reguläres Tiling; Anm. des Autors) von Vektordaten oft auf Standards für Raster Tiles zurückgegriffen (Ingensand et al. 2015). Diese definieren Regeln, anhand derer geografische Daten in einer Pyramide von Kacheln (in dem Fall Bildkacheln) in mehreren Zoom-Leveln aufgeteilt und dargestellt werden (OpenStreetMap 2018c). Die Aufteilung bzw. Indizierung erfolgt beim regulären *Tiling* in der Form von Quadtrees, d. h. eine Kachel wird zur nächsten Zoomstufe in vier weitere Kacheln unterteilt (Abbildung 5).

Infolgedessen entstehen Tile Sets mit einer quadratisch anwachsenden Anzahl von Tiles von einer Zoomstufe zur nächsten. Werden bei Zoomstufe 2 insgesamt 16 Kacheln erzeugt, sind es bei Zoomstufe 10 bereits über 1 Million und bei Zoomstufe 20 rund 1.100 Milliarden Kacheln.

1 http://docs.opengeospatial.org/per/17-041.html#_global_gnosis_tiling_scheme_adapted_to_polar_regions
2 https://github.com/openstreetmap/mod_tile

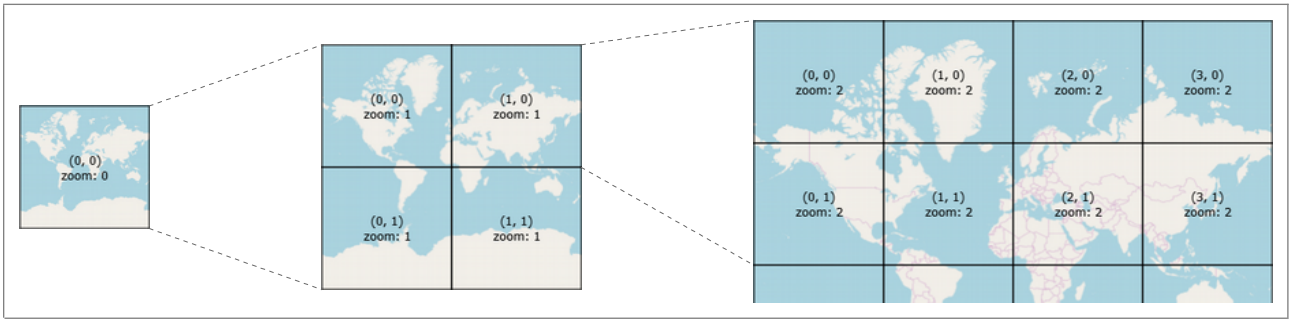


Abbildung 5 Reguläre Kachelung am Beispiel von *OpenStreetMap*
Kachelkoordinaten: (x, y) und z = *zoom*
Karte: © *OpenStreetMap* Mitwirkende CC-BY-SA; Gitter: *TileLayerPlugin* QGIS 2.18.17

Das bereits erwähnte irreguläre Tiling erfolgt in Bezug auf die räumliche Verteilung der Daten, z. B. bei den *Vector Tile Packages* bei *ArcGIS Pro*, basierend auf einem Kachelindex, der anhand einer maximalen Anzahl von Vertices pro Kachel errechnet wird (Abbildung 4). Hierbei kann die Anzahl der einzelnen Tiles und damit die Datenmenge drastisch reduziert werden. Beim in Abbildung 4 gezeigten Layer *sie01_f.shp* des DLM1000 erzeugt *ArcGIS Pro* mit einer regulären Kachelung 82.367 Tiles in 16 Zoomstufen, während beim Verwenden eines Index (berechnet mit maximal 1.000 Vertices pro Kachel) 267 Kacheln in 12 Zoomstufen resultieren.

4.3.2 *Tiling* Schemata

4.3.2.1 Entwicklung

Mit der *Tile Map Service Specification* (TMS)[1] wurde 2006 von einer losen Gemeinschaft unter dem Dach der *Open Source Geospatial Foundation* (OSGeo) ein erster offener Standard zur Art und Weise entwickelt, wie Clients Kartenkacheln anfordern und Server ihre (Geo)Datenbestände beschreiben (Masó, Pomakis, et al. 2010; OSGeo Wiki 2012a).

Parallel existierten zu diesem Zeitpunkt andere kartenkachelähnliche Implementationen, z. B. *Google Maps* und *NASA OnEarth* (Masó, Pons, et al. 2010). Auf dieser Situation basierend skizzieren Masó et al. (2010) die weitere Entwicklung wie folgt (Auszüge):

- 2007 erhielt die OGC WMS-Revisionsarbeitsgruppe eine Änderungsanforderung, um eine Unterstützung von Kacheln als Teil des WMS-Schnittstellenstandards aufzunehmen, die Gruppe beschloss jedoch, einen separaten Standard zu definieren, den Web Map Tile Service (WMTS).
- Von Oktober 2008 bis Juni 2009 wurden in OWC-6[2] Interoperabilitätsexperimenten vier unabhängige WMTS-Entwicklungen getestet.
- Der daraus erarbeitete Standard wurde im Dezember 2009 als OGC-Standard[3] genehmigt und im April 2010 veröffentlicht.

1 http://wiki.osgeo.org/wiki/Tile_Map_Service_Specification
2 http://www.opengeospatial.org/projects/initiatives/ows-6
3 http://www.opengeospatial.org/standards/wmts#schemas

4.3.2.2 Tile Map Service (TMS)

Die TMS-Spezifikation liegt in der Version 1.0 vor, wird jedoch nicht weiterentwickelt (OSGeo Wiki 2012a). TMS wird zwar im Wiki der *Open Source Geospatial Foundation* (OSGeo)[1] veröffentlicht, ist jedoch kein offizieller Standard der Foundation und wird nicht als Projekt der OSGeo anerkannt (OSGeo Wiki 2012a). Der Zugriff auf die Ressourcen eines TMS erfolgt über eine REST-Schnittstelle, also mit Hilfe einer URL. In drei Stufen werden unterschiedliche Informationen zum TMS im XML-Format zurückgegeben:

1. ***Root Resource***: Verfügbare Services, Auflistung von <TileMapService>-Elementen
2. ***TileMapService Resource***: Verfügbare Karten und Projektionen in der <TileMaps>-Auflistung
3. ***TileMap Resource***: Beschreibung von Ausdehnung (<BoundingBox>), Ursprung (<Origin>), Kachelformat (<TileFormat>) und Auflistung verfügbarer Kachelsets bzw. Zoomstufen (<TileSets>-Auflistung) einer Karte

Der Zugriff auf die Kacheln erfolgt über die beim <TileSet>-Element angegebene URL (*href*-Attribut), erweitert durch die X-Kachel-Koordinate als Unterordner und die Y-Kachel-Koordinate als Dateiname mit der Erweiterung, welche im <TileFormat>-Element als *extension* angegeben ist. Nach GeoWebCache Contributors (2018) gilt TMS als Vorläufer für den OGC-WMTS-Standard. Er wird aktuell neben WMTS als Quasistandard weiterverwendet, z. B. in den Applikationen *GeoWebCache*[2] und *TileCache*[3].

In der TMS-Spezifikation wird vergleichbar zur XYZ-Indizierung (z.B. von *OpenStreetMap* und *Google Maps* verwendet, Kap. 4.3.2.4) das Schema der Kachelung beschrieben. Der Ursprung des Kachelkoordinatensystems (<Origin>) wird in Koordinaten des CRS definiert. Die Kacheln werden pro *Tile Set* von diesem Ursprung aus, jeweils ganzzahlig, mit 0 beginnend, jeweils in X- und Y-Richtung nummeriert (Abbildung 6).

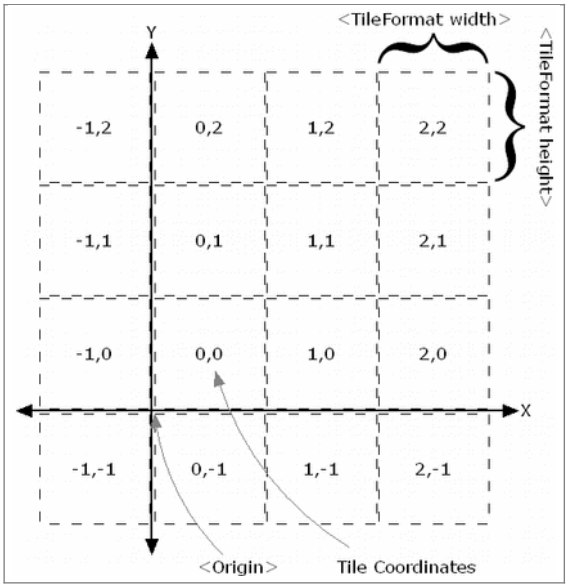


Abbildung 6 *TileMap Diagram* des TMS (Quelle: OSGeo[4])

Der wesentliche Unterschied zwischen TMS und XYZ besteht darin, dass der Ursprung des Kachelkoordinatensystems bei TMS in der unteren linken Ecke liegt. Somit unterscheiden sich die Indizes der beiden Systeme im Y-Wert. Dieser kann einfach umgerechnet werden (Code 1, Kap. 4.3.2.4).

1 http://wiki.osgeo.org/wiki/Main_Page
2 http://geowebcache.org/docs/current/services/tms.html
3 http://tilecache.org/
4 http://wiki.osgeo.org/wiki/File:Tms.png

4.3.2.3 Web Map Tile Service (WMTS)

Der *Web Map Tile Service* (WMTS) Implementation Standard bietet eine standardbasierte Lösung für die Bereitstellung digitaler Karten mit vordefinierten Bildkacheln (Masó, Pomakis, et al. 2010). Entsprechend des Standards ist die Darstellung des Raumes in einem gekachelten Layer von einem eigenständigen Satz von Parametern abhängig, welche in *Tile Matrix Sets* definiert werden. Jedes *Tile Matrix Set* enthält ein oder mehrere *Tile Matrizes.* Diese beschreiben:

- den Maßstab der Kacheln in Bezug auf eine standardisierte Rendering-Pixelgröße von 0,28 x 0,28 mm,
- die Breite und Höhe der Tiles in Pixel,
- die obere linke Ecke der Bounding Box (umgebendes Rechteck) der *Tile Matrix* (die CRS-Koordinate der oberen linken Ecke des oberen linken Pixels des oberen linken Tiles) sowie
- die Breite und Höhe der *Tile Matrix* (Anzahl der Tiles).

Laut WMTS-Spezifikation sind auch rechteckige Kacheln (siehe da Abschnitt 4.11 und Anhang D.2) verwendbar. Der Ursprung des Kachelkoordinatensystems ist beim WMTS in der linken oberen (nordwestlichen) Ecke. Somit entspricht die Bezeichnung der Kacheln des in vielen Quellen als XYZ bezeichnete Schemas dem WMTS-Standard. In Annex E der WMTS-Spezifikation werden vier bekannte *Tile Matrix Sets* beschrieben (E.1 bis E.4). Diese können jeweils über einen URN identifiziert werden. Im Anhang (Anlage 2) sind diese vier *Tile Matrix Sets* aufgelistet, wobei deren Unterschiede mit Beispielkacheln bildlich dargestellt werden.

4.3.2.4 Weitere verbreitete Schemata

Neben TMS und WMTS existieren noch weitere *Tiling*-Schemata und de facto Standards. Die bekanntesten sind *Slippy map tilenames* für OpenStreetMap Karten (OpenStreetMap 2018a), die *Google Maps API* (Google Inc. 2017b) sowie das *Bing Maps Tile System* (Schwartz 2018) welche sich jedoch grundsätzlich gleichen. Balkenbush (2017) beschreibt in seinen Ausführungen prägnant die Unterschiede und Bedeutung der einzelnen Schemata.

Alle drei Schemata erlauben nur die Projektion nach EPSG 3857 (syn. EPSG 900913 = WGS 84/Pseudo Mercator oder Web Mercator; deprecated EPSG 3785). Diese entspricht dem vordefinierten *Matrix Set* entsprechend der WMTS-Spezifikation mit der URN urn:ogc:def:wkss:OGC:1.0:GoogleMapsCompatible. Darüber hinaus liefern diese Services ausschließlich Bildkacheln in der Größe 256x256 Pixel im PNG-Format (Google Inc. 2017b; Schwartz 2018; OpenStreetMap 2018a). Sie sind damit starrer in der Anwendung als TMS und WMTS.

Die hier aufgeführten Schemata weisen zwei unterschiedliche Tile Indizes auf. Das von OpenStreetMap und Google Maps verwendete Schema wird im Internet häufig als XYZ-Schema bezeichnet (Přidal 2008; MacWright 2013; Balkenbush 2017; OpenLayers 2018). Bing Maps verwendet zur Optimierung der Indizierung und Speicherung seiner Tiles *Quadtree keys* (Schwartz 2018).

Das XYZ-Schema gleicht dem Indizierungsschema von TMS, hat jedoch den Ursprung der Kachelkoordinaten in der linken oberen Ecke des Koordinatensystems, d. h. die Y-Werte der Kacheln nehmen entgegengesetzt zu TMS von oben nach unten bzw. von Nord nach Süd zu. Die Zuordnung der Kachelkoordinaten entspricht somit den Vorgaben nach WMTS-Standard. Die Umrechnung der Y-Werte zwischen XYZ und TMS erfolgt entsprechend den in Code 1 aufgeführten Formeln nach Přidal (2008) und Aitchison (2011).

Code 1 Umrechnung von Y-Kachelkoordinaten zwischen XYZ und TMS

```
# aus Přidal (2008):
y = (2^zoom - 1) - y
# aus Aitchison (2011):
y = (1>>zoom) - y - 1
# dabei sind:
#    zoom = Zoomstufe oder LOD
#    y = Kachelkoordinate im XYZ oder TMS-System
# Hinweis:
# Term 2^zoom und die bitweise Verschiebung 1<<zoom liefern dasselbe Ergebnis
```

Bei den *Quadtree keys* von Bing Maps werden die Kacheln nicht über zweidimensionale XY-Koordinaten, sondern über eine eindimensionale Ziffernfolge angesprochen, die aus der Verschachtelung der Binärwerte der XY-Koordinaten erzeugt werden. Deren Bezeichnung basiert auf demselben System wie bei OpenStreetMap und Google Maps, d. h. die Nummerierung beginnt in der oberen linken Ecke des Kachelkoordinatensystems. Die Umrechnung der Kachelkoordinaten zwischen *Bing Quadtree Keys* und XYZ zeigt Code 2.

Code 2 Umrechnung von XYZ-Kachelkoordinaten in Quadtree Keys (aus Schwartz, 2018)

```
# XYZ-Koordinaten in binärer Schreibweise (führende Nullen einfügen, so
# dass Anzahl der Ziffern der Binärzahl dem Zoomfaktor entspricht)
z = 4₁₀
y = 7₁₀ = 0111₂
x = 3₁₀ = 1100₂
# Verschachtelung von y und x (jeweils von links nach rechts, y zuerst)
quadkey = 01111010₂ = 1322₄
# die Konvertierung von Quadkeys in XYZ-Koordinaten erfolgt auf dem
# umgekehrten Weg:
#    1. Konvertierung in Binärzahl
#    2. Aufsplitten in y und x
#    3. Umwandeln von y und x in Dezimalzahlen
```

4.3.2.5 Umrechnung von Bildschirm in Realweltkoordinaten

In *Tile Sets* werden unterschiedliche Koordinatensysteme und Einheiten verwendet. Durch die Umrechnung zwischen den einzelnen Parametern kann z. B. von einem Bildschirmpixel auf die Realweltkoordinaten rückgeschlossen werden. Přidal (2008) beschreibt die Umwandlung von geodätischen Koordinaten zu projizierten Koordinaten in der *Mercator Projektion* (EPSG:3857), die Zoomstufen und die sich daraus ergebenden Tile Indizes. In der Programmiersprache *Python* werden auf dieser Seite die in der Anwendung *Maptiler*[1] genutzten Formeln veröffentlicht. Sie ermöglichen u. a. die Berechnung von Kachelgrenzen, die Auflösung von Kacheln (Meter pro Pixel je Zoom Level) sowie verschiedene Umrechnungen, wie Pixel- in Kachelkoordinaten, Pixelkoordinaten in Meter, Meter in geografische Koordinaten und unterschiedliche Indizierungsschemata für Kacheln[2]. In der *Google Maps API* (Google Inc. 2017b) wird mit *JavaScript*-Beispielen die Umrechnung zwischen den unterschiedlichen Koordinatensystemen beschrieben[3]. Vergleichbare Berechnungen zeigt Schwartz (2018) in der Programmiersprache C#.

1 http://www.maptiler.com
2 http://www.maptiler.org/google-maps-coordinates-tile-bounds-projection/globalmaptiles.py
3 https://developers.google.com/maps/documentation/javascript/examples/map-coordinates

4.4 Besonderheiten von Vector Tiles

4.4.1 Generalisierung und Topologie

Ein wesentlicher Bestandteil bei der Erstellung von Vector Tiles ist die unterschiedlich starke Generalisierung, entsprechend dem LOD. Die Generalisierung dient einerseits einer übersichtlicheren Visualisierung und andererseits der Reduzierung der Datenmenge (Longley et al. 2005). Bei McMaster & Shea (1992) werden eine Vielzahl von Generalisierungsregeln beschrieben. Weibel und Dutton (2005) greifen die Regeln bzw. Operatoren auf, die auf alle gängigen Datentypen zutreffen und heben die Operatoren hervor, welche topologische Transformationen erfordern. Über diese Auflistung hinaus kann es auch bei der Vereinfachung von Linien oder Polygonen bei sogenannten *degenearate cases* (Kim & Kim 2006) und/oder durch eine fehlerhafte Implementierung von Generalisierungsalgorithmen (Ramer 1972; Douglas & Peucker 1973; Visvalingam & Whyatt 1992) geschehen, dass Topologiefehler auftreten, wie das Beispiel in Abbildung 1 zeigt.

Bei rein render-basierten Anwendungen sind kleine Topologiefehler, wie sie in Abbildung 11 erkennbar sind, vernachlässigbar, sofern es sich um Grenzen zwischen unterschiedlichen Features handelt, da sie aufgrund der begrenzten Pixelzahl bei der Bildschirmdarstellung nicht sichtbar werden. Auf diesen Daten basierende Auswertungen unterliegen, aufgrund der reduzierten Genauigkeit der Koordinaten, zwangsläufig einer gewissen Ungenauigkeit. Dagegen stellt die Beibehaltung der Topologie bei feature-basierten Vector Tile-Formaten einen wesentlichen Aspekt dar. Weibel und Dutton (2005) skizzieren die Anforderungen an ein digitales Generalisierungssystem. Anhand der Ausführungen von Li et al. (2017) wird deutlich, welche Darstellungsfehler beim Rendern entstehen können, wenn Feature-Fragemente aufgrund topologischer Fehler clientseitig nicht als ein Objekt erkannt bzw. dargestellt werden.

4.4.2 Rendering

Das direkte Rendering von Vector Tiles in Clients ist im Rahmen dieser Arbeit nur randlich von Bedeutung, da diese als Datengrundlage für die Darstellungsdienste WMS und evtl. WMTS, welche dem Client fertige Kartenbilder übermitteln, dienen sollen. Das Rendern soll demnach von diesen Diensten übernommen werden. Die Daten müssen jedoch in einer Form verfügbar sein, die es diesen Diensten ermöglicht, sie fehlerfrei zu verarbeiten. Aus diesem Grund sei an dieser Stelle auf Besonderheiten von Vector Tiles gegenüber „herkömmlichen“ Vektordatenquellen hingewiesen:

- Linien- und flächenhafte Features können sich in Fragmenten über mehrere Tiles erstrecken.
- Beim *Tiling* müssen bei abgeschnittenen Polygonen entlang der Kachelgrenzen „künstliche“ Polygonabschnitte eingefügt werden, um die entstehenden Fragmente auch als Polygone speichern zu können (Antoniou et al. 2009).
- Vector Tiles werden, um Renderproblemen vorzubeugen, bei fast allen recherchierten Formaten mit einem Puffer um das eigentlich Tile angelegt. Das bedeutet, dass es beim Anzeigen von zwei benachbarten Tiles zu Überlappungen kommt (Abbildung 7).
- Attributdaten werden bei Features, die sich über Tile-Grenzen hinweg erstrecken, entweder in jedem betroffenen Tile (und damit redundant) gespeichert oder es muss eine andere Lösung (Kap. 4.4.2.2) angewendet werden. Dies bedeutet jedoch, dass der Client die Möglichkeit haben muss, die Attributdaten jedem Fragment eines Features zuordnen zu können.

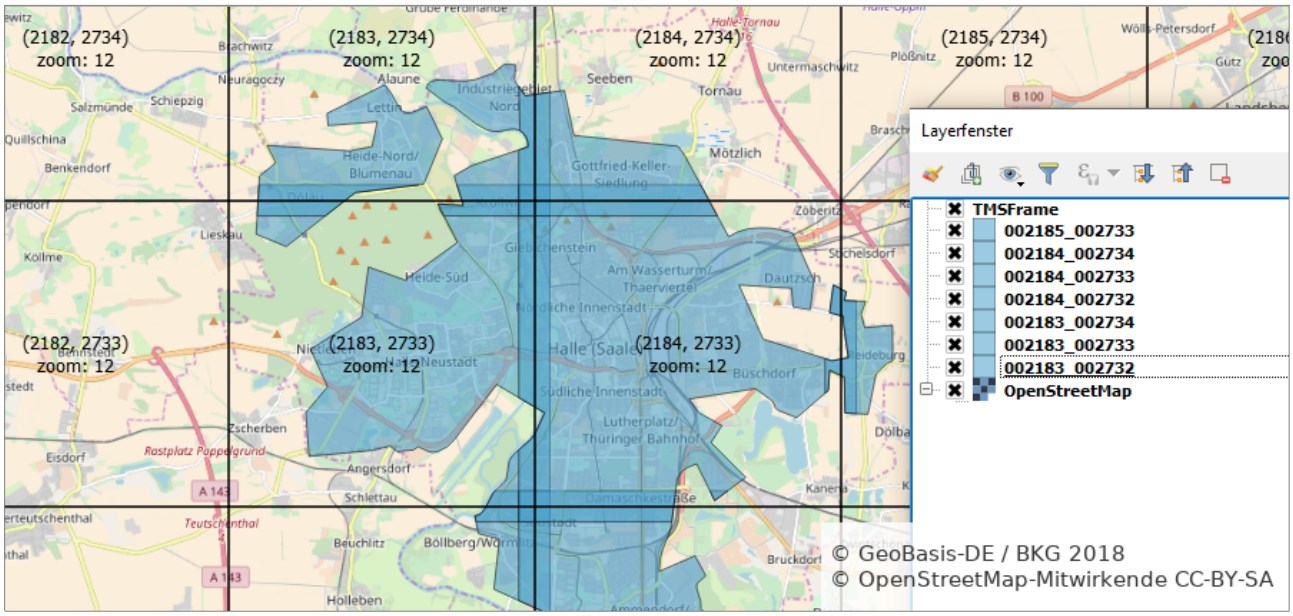


Abbildung 7 Tile-Gitter ({x}, {y}) Zoomstufe 12 nach TMS und *GeoJSON*-Tiles der Siedlungsgrenzen, erzeugt mit *GeoWebCache*
Quellen: Siedlungsgrenzen aus DLM1000 (© GeoBasis-DE / BKG 2018); Hintergrundkarte © *OpenStreetMap* Mitwirkende CC-BY-SA; Tile Gitter erstellt mit *TileLayerPlugin* in *QGIS* 2.18.17

Zur Formatierung bei der Darstellung von Vector Tiles gibt es unterschiedliche Ansätze. In den Tiles selbst werden nur die geografischen Informationen und, in den meisten Fällen, die Attributdaten mitgeliefert. Die Zuweisung von Stilen erfolgt standardmäßig clientseitig, wie beispielsweise auch bei der Nutzung eines WFS. Darüber hinaus besteht die Möglichkeit, dass vom Server Stilvorgaben für den Client in separaten Dateien mitgeliefert werden.

4.4.2.1 Zusammenfügen von Tiles

Für die Nutzung von Vector Tiles als Datengrundlage für WMS müssen die einzelnen Features entweder vorher, vereinigt (*merge* bzw. *dissolve)* werden oder die den WMS bereitstellende Serversoftware ist in der Lage, dies vor dem Rendern selbst zu tun. Nur so können Konflikte, z. B. durch die Beschriftung jedes Fragmentes eines Features oder Darstellungsprobleme an den Tile-Grenzen, und Überlappungen vermieden werden.

Nordan (2012) entwickelt Ansätze zur Erstellung (*construction*) von Vector Tiles und dem Zusammenfügen von Features (*concatenation*) aus diesen und geht auf bestehende Probleme ein. Wesentlich beim Zusammenfügen ist, dass die Features als zusammengehörig erkannt werden. Zu diesem Zweck greift der Autor die Idee von Antoniou et al. (2009) auf, eindeutige Schlüssel für jedes Feature in den Tiles mitzuspeichern. Darüber hinaus ist es für feature-basierte Anwendungen wesentlich, beim Zusammenfügen eine Information zur Vollständigkeit eines Features zu erhalten (Nordan 2012). Um Tiles, in denen sich ein Feature fortsetzt, effizient ermitteln zu können, schlägt Nordan (2012) einen Ansatz vor, bei dem Feature Segmente in einem Tile mit einer Information darüber versehen werden, in welchen benachbarten Tiles sie sich fortsetzen. Damit wird verhindert, dass alle Tiles einer Zoomstufe nach Fragmenten eines Features durchsucht werden müssen.

4.4.2.2 Attributdaten

Zur Übernahme von Attributdaten in Vector Tiles beschreibt Cavazzi (2018) drei Möglichkeiten und führt jeweils Vor- und Nachteile mit Blick auf Features auf, die über Kachelgrenzen hinausgehen:

1. In jeder Kachel werden alle Attributinformationen des Features gespeichert.
2. Die Attributinformationen des Features werden nur in einer Kachel gespeichert und alle anderen Kacheln mit Fragmenten des Features enthalten Verweise auf diese Kachel.
3. Die Attributinformationen werden nicht in den Kacheln, sondern in einem separaten Webservice gespeichert.

Variante 1 ist einfach überschaubar und interpretierbar, da jeweils nur die aktuelle Kachel analysiert werden muss, um Attributinformationen zu den enthaltenen Features auszulesen. Der Nachteil ist, dass die Attributdaten auf diesem Weg redundant gespeichert werden. Für Attributänderungen großflächiger Features bedeutet das eine Aktualisierung einer Vielzahl von Kacheln in allen Zoomstufen. Außerdem erhöht sich der Speicheraufwand durch die mehrfache Speicherung identischer Informationen. Für ***Variante 2*** bietet sich die einmalige Speicherung der Attributinformationen in der Kachel der niedrigsten Zoomstufe an, in der das betreffende Feature erstmalig vorkommt (Balog & Houtmeyers 2017b). In allen anderen Kacheln, die das Feature enthalten, wird ein Verweis auf diese sogenannte Ankerkachel gespeichert . Somit muss zur Abfrage der Attribute eines Features nur die jeweilige Ankerkachel nachgeladen werden. Detailliertere Informationen zu ***Variante 3*** sind der o. g. Quelle nicht zu entnehmen, jedoch ist ableitbar, dass zusätzliche Anfragen, vergleichbar einem *GetFeatureInfo-Request* bei WMS und WMTS, erforderlich sind, um Attributinformationen zu einzelnen Features zu erhalten.

4.5 Datenformate

Zur Speicherung von Vector Tiles existiert aktuell eine Vielzahl von Datenformaten bzw. Spezifikationen. Teilweise werden vorhandene Datenformate direkt genutzt. In einigen Fällen werden diese für die speziellen Erfordernisse von Vector Tiles erweitert. Grundsätzlich kann zwischen textbasierten, für Menschen lesbare und binären Datenformaten unterschieden werden. Taraldsvik (2012) stellt anschaulich die Reduzierung des Speicherbedarfs bei der Nutzung binärer Formate gegenüber Textformaten dar. Darüber hinaus werden unterschiedliche Konzepte eingesetzt, um Daten effizient zu speichern und damit auch die zu übertragende Menge zu reduzieren. Die Entscheidung für ein Dateiformat ist dem Einsatzzweck der Vector Tiles entsprechend zu treffen, z. B. Schwerpunkt auf schnelle Darstellung oder auf Genauigkeit und Erhaltung der Topologie.

4.5.1 Google Protocol Buffer

Protocol Buffers (auch *protobuf* genannt; Abkürzung PBF) sind ein binäres, plattform- und sprachenunabhängiges Datenformat zur Serialisierung strukturierter Daten, welches *Google* seit 2001 entwickelt (Google Inc. 2017a). Es wurde 2008 als *Open Source* Projekt veröffentlicht (Teufel 2008) und liegt aktuell in der Proto3-Syntax vor. Laut Google LLC (2017) sind *Protocol Buffers* nun:

> *„... Google's lingua franca for data ...“.*

Cogo (2014) beschreibt, wie die Serialisierung bei der Speicherung von Daten im *Protocol Buffer*-Format stattfindet und zeigt dessen Vorteile hinsichtlich der Performance mit Blick auf Speicherplatz und Zugriffszeiten gegenüber anderen Datenformaten. So ist der Speicherbedarf weniger als halb so groß als beim *JSON*-Format mit dem geringsten Speicherbedarf und beträgt weniger als zwei Drittel als das *XML*-Format mit dem geringsten Speicherplatz. Vergleichbar ist die Reduktion der gemessenen Zeit (Erstellung, Serialisierung und

Deseriealisierung). Sie beträgt gegenüber dem kleinsten *JSON*-Format etwas mehr als $^2/_3$ und dem kleinsten *XML*-Format ca. $^1/_5$. Google nennt Werte von 3- bis 10mal kleiner und 20- bis 100mal schneller als *XML*.

Das *protocol buffers*-Format ist im *Developer Guide*[1] beschrieben. Zu jeder unterstützten Sprache existiert eine API Referenz[2] und ist über *GitHub*[3] frei verfügbar. Die Definition der zu serialisierenden Informationen erfolgt in *proto*-Dateien über *message*-Typen. Mit Hilfe des *protocol buffer* Compilers werden basierend auf der *proto*-Datei für die gewünschte Sprache Zugriffsklassen generiert. Damit besteht für jede der unterstützten Sprachen einfacher Zugriff auf jedes Feld der *messages* und die Methoden zum Serialisieren und Analysieren der Daten. Bei Mapbox (2016) wird anschaulich beschrieben, wie speichereffizient die Verschlüsselung von Attributen erfolgt. Innerhalb einer Datei werden alle *keys* und *values*, im Falle von Geodaten aller Features (vgl. KVP bei *GeoJSON*, Kap. 4.5.2.1), in einem Index gespeichert und beim Feature auf diesen Index mit Hilfe von *tags* in Form von *integer*-Werten verwiesen.

Aufgrund der guten Performance, des offenen Standards und der Unabhängigkeit von Programmiersprachen bieten sich *protocol buffers* für die Speicherung von Vector Tiles und Implementierung in entsprechende Lösungen an und werden zum Beispiel von *Mapbox* für deren Vector Tiles Format verwendet (Kap. 4.5.1.1). Die serialisierten Daten werden in *pbf*-Dateien abgelegt.

4.5.1.1 Mapbox Vector Tiles

Die Firma *Mapbox* hat in einem offenen Standard[4] ein eigenes Vector Tile Format zur eindeutigen Identifizierung mit **.mvt*-Erweiterung, definiert (es wird jedoch auch häufig mit **.pbf* als Erweiterung verwendet). *Mapbox Vector Tiles* (MVT) werden im *Protocol Buffers Format* von Google (Kap. 4.5.1) verschlüsselt. Die Spezifikation liegt zum Stand dieser Arbeit in Version 2.1 (Stand 19.01.2016) vor. Die Basis-*proto*-Datei[5] verdeutlicht die Regeln zur Definition von Geometrien in *Messages*.

Die *Mapbox Vector Tile Spezifikation* (Mapbox 2016) beschreibt, wie die Daten im *mvt*-Format gespeichert werden, aber nicht das *Clippen* zum Erzeugen von Vector Tiles. *Mapbox* selbst schneidet die Geometrien mit einem *Buffer* um die Vector Tiles, so dass übergreifende Feature-Teile benachbarter Tiles einander überlappen (vergleiche Abbildung 7), aber auch andere Ansätze sind in diesem Datenformat möglich. Dieses erschwert eine allgemeingültige Nutzung der Tiles ohne Wissen um deren Herkunft. Ebenso behandelt die Spezifikation nicht, wie Vector Tiles gespeichert, angefordert und verteilt werden können und hält keine Lösungen für Probleme bereit, die aus der Vereinfachung der Geometrien entstehen. Als Referenz für *Mapbox Vector Tiles* dienen die *Web Mercator*-Projektion (EPSG 3857) und das *XYZ*-Tilingschema (Kap. 4.3.2.4), jedoch können *Mapbox Vector Tiles* für jede Projektion und jedes Tilingschema eingesetzt werden.

Um eine effiziente Speicherung geografischer Daten zu ermöglichen, werden die Realweltkoordinaten der Geometrien in ganzzahlige Vector Tile Gitter-Koordinaten, ausgehend von der linken oberen Ecke des Gitters (beginnend bei Tile 0,0 jeder Zoomstufe) konvertiert und dabei gerundet. Koordinaten von Linien und Polygone werden dabei vom ersten Vertex ausgehend relativ angegeben. Durch Umwandlung der Realwelt- in Bildschirmkoordinaten gehen die ursprünglichen Geometrien verloren (rote Linie in Abbildung 8). Damit sind *Mapbox Vector Tiles* ein Format, welches zwar Vektordaten beinhaltet, aber speziell zum Rendern von Kacheln, also z. B. für Darstellungsdienste, entwickelt wurde (Skowron 2017b). Wie diese Ausfüh-

1 https://developers.google.com/protocol-buffers/docs/overview
2 https://developers.google.com/protocol-buffers/docs/reference/overview
3 https://github.com/google/protobuf
4 https://github.com/mapbox/vector-tile-spec
5 https://github.com/mapbox/vector-tile-spec/blob/master/2.1/vector_tile.proto

rungen und die Informationen in Kap. 4.5.1 verdeutlichen, stellen *Mapbox Vector Tiles* ein performantes Datenformat dar. Durch das *simple coordinate snapping* werden unter Umständen jedoch Topologien aufgebrochen (Cavazzi 2018). Attributinformationen werden innerhalb eines Tiles zwar effizient, unter Minimierung der Redundanz von Attributinformationen gespeichert, dafür sind jedoch komplexere Strukturen beim Erzeugen (Server) und Rendern (Client) der Kacheln erforderlich (Cavazzi 2018). Bei Tile-übergreifenden Features werden deren Attribute jedoch in jedem Tile wiederholt (Erhöhung des Speicheraufwandes und Redundanz).

Nach Kalberer (2017b; ca. Min. 4:30) sind *Mapbox Vector Tiles* seit der Übernahme von *ESRI* in deren *Vector Tile Packages* (Kap. 4.6.3)zu einem de facto Standard für Vector Tiles geworden. Verstärkt wird diese Situation noch dadurch, dass seit Version 2.3 der *GDAL*-Bibliothek das *MVT*-Format eingebunden ist[1] und der *MBTiles*-Treiber auch Datenbanken mit Vector Tiles unterstützt[2].

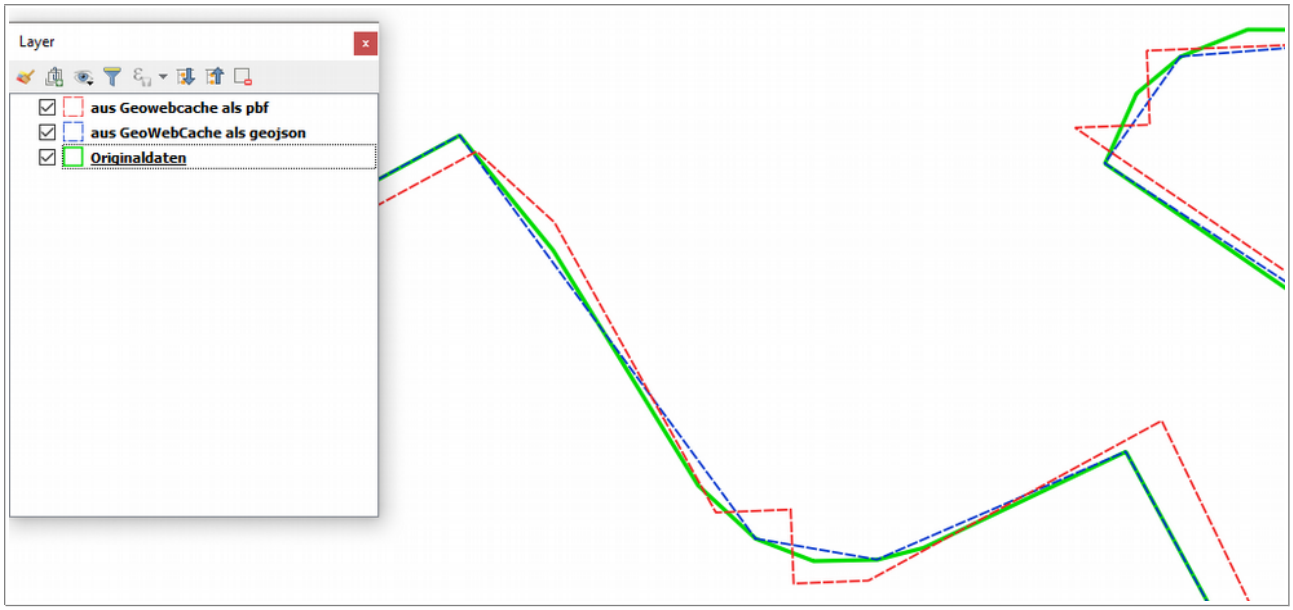


Abbildung 8 Abweichung von Polygonen in unterschiedlichen Vector Tile Formaten von den Originaldaten
Die blaue Linie (*GeoJSON*) ist zwar gegenüber den Originaldaten generalisiert, jedoch stimmen die noch vorhandenen Vertices mit denen der Originaldaten überein. Für Vector Tiles im *MVT*-Format (rote Linie) werden die Koordinaten in Bildschirmkoordinaten umgerechnet, mit Verschiebung der Knoten als Resultat.
Datenquelle Originaldaten: Siedlungsgrenzen aus *DLM1000* (© GeoBasis-DE / BKG 2018)

4.5.1.2 Mapnik Vector Tiles

Im *OpenStreetMap Wiki*[3] und in der API-Referenz des Tileservers *TileStache*[4] werden *Mapnik Vector Tiles* als eigenes Format aufgeführt. Entsprechend der Dokumentation von *Node Mapnik 3.6*[5] stellt *mapnik.VectorTile* eine Klasse für einen Kachelgenerator dar, welcher Tiles entsprechend der *Mapbox Vector Tile* Spezifikation erstellt. Darüber hinaus wird die Bezeichnung für eine *Mapnik-Implementierung*[6] der *Mapbox Vector Tile Spezifikation* geführt.

1 https://www.gdal.org/drv_mvt.html
2 https://www.gdal.org/frmt_mbtiles.html
3 https://wiki.openstreetmap.org/wiki/Vector_tiles
4 http://tilestache.org/doc/TileStache.Goodies.VecTiles.mvt.html
5 http://mapnik.org/documentation/node-mapnik/3.6/
6 https://github.com/mapbox/mapnik-vector-tile

Camper (2016) klärt das Durcheinander der Begriffe mit wenigen Worten:

> *„Mapnik Vector Tiles was simply an earlier name for Mapbox Vector Tiles. They have updated the spec to change the name, but the format and data structure is otherwise the same.“*
>
> ***übersetzt:***
>
> *„Mapnik Vector Tiles war einfach ein früherer Name für Mapbox Vector Tiles. Sie haben die Spezifikation aktualisiert, um den Namen zu ändern, aber das Format und die Datenstruktur sind ansonsten gleich.“*

4.5.1.3 OpenScienceMap-PBF (OSciM-PBF)

Ein weiteres binäres Format, welches auf *Protocol Buffers* basiert, ist *OpenScienceMap-PBF* (OSciM-PBF). Es wurde in Zusammenhang mit der für Android konzipierten Anwendung *OpenScienceMap* als wissenschaftliches Projekt entwickelt (Schmid 2013). Der Autor beschreibt das Projekt so:

> *„OpenScienceMap provides free and open maps for Android with the fastest and 100% pure vector maps around. OpenScienceMap consists of a map data server, vector data and a super-fast client.“*
>
> ***übersetzt:***
>
> *„OpenScienceMap bietet kostenlose und offene Karten für Android mit den schnellsten und 100% reinen Vektorkarten. OpenScienceMap besteht aus einem Kartendatenserver, Vektordaten und einem superschnellen Client.“*

Das Projekt *VTM*[1] wird von *Mapsforge* weiterentwickelt und unterstützt neben anderen Vector Tile Formaten auch weiter *OpenScienceMap-PBF*. Eine andere Anwendung für dieses Dateiformat konnte nicht recherchiert werden.

4.5.1.4 Spaten

Mit dem Geodatenformat *Spaten*[2,3] existiert seit 2017 ein neues binäres Geodatenformat, welches flexibel sein soll und den Fokus auf sehr schnelle Serialisierung/Deserialisierung setzt. Hintergrund für die Entwicklung des neuen Formates ist die Beschleunigung und Vereinfachung bei der Verwendung von *OpenStreetMap*-Daten in Vector Tiles (Skowron 2017a; Skowron 2018). Die Definition erfolgt, wie im Target-Format *Protocol Buffers*, im *proto*-Format[4]. Bisher liegt *Spaten* in Version 0 vor und wird ausschließlich in der noch in Entwicklung befindlichen Anwendung *Grandine*[5] eingesetzt.

4.5.2 JavaScript Object Notation (JSON)

JSON.org beschreibt die *JavaScript Object Notation* (JSON) wie folgt:

> *„... ist ein schlankes Datenaustauschformat, das für Menschen einfach zu lesen und zu schreiben und für Maschinen einfach zu parsen (Analysieren von Datenstrukturen) und zu generieren ist.“*

JSON wird aktuell in zwei konkurrierenden Spezifikationen (ECMA-404[6] und RFC 8259[7]) definiert (ECMA 2017). Aufgrund seiner Eigenschaften und der Flexibilität eignet es sich sehr gut zur kompakten, strukturierten Verwaltung und zum Austausch von Daten. *JSON* ist für die einfache Analyse der beschrie-

1 https://github.com/mapsforge/vtm
2 https://thomas.skowron.eu/spaten/
3 https://github.com/thomersch/grandine/tree/master/fileformat
4 https://github.com/thomersch/grandine/blob/master/fileformat/fileformat.proto
5 https://github.com/thomersch/grandine
6 http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf
7 https://tools.ietf.org/pdf/rfc8259.pdf

benen Daten mit *JavaScript* entwickelt worden (Taraldsvik 2012). Aus diesem Grund basieren auch einige für Vector Tiles verwendete oder speziell für Vector Tiles konzipierte Geodatenformate auf *JSON*. Die verbreitetsten sind die nachfolgend detaillierter beschriebenen Formate *Geografic JSON* (GeoJSON; Kap. 4.5.2.1) und *Topology JSON* (TopoJSON; Kap. 4.5.2.2).

Über diese beiden Formate hinaus sind noch weitere *JSON*-Formate für Vector Tiles in Nutzung oder für deren Nutzung denkbar. Das ***Kothic JSON*** Vector Tiles Format, welches von der Kartenrendering-Engine *Kothic JS*[1] verwendet wird und dem *GeoJSON*-Format ähnlich ist, basiert auf *JSONP* (Praliaskouski et al. 2017). Die Realweltkoordinaten werden in *Spherical Mercator Integers* umgewandelt, was den Speicherbedarf, damit jedoch auch die Genauigkeit reduziert, weswegen *Kothic JSON* als renderbasiertes Format eingestuft werden kann. Darüber hinaus verfügt es über ein Feld *reprpoint* (*representative point*), in welchem ein Koordinatenpaar innerhalb eines Polygons gespeichert werden kann, um an diesem Punkt ein Symbol oder eine Beschriftung zu zeichnen (Praliaskouski et al. 2017).

Der Vollständigkeit halber sollen hier auch *JSONB* und *BSON* als binäre Varianten des *JSON*-Formates aufgeführt werden. *JSONB* wird, wie auch *JSON*, als Datentyp in dem objektrelationalen Datenbankmanagementsystem *PostgreSQL*[2] akzeptiert. Es bietet gegenüber *JSON* eine größere Effizienz bei der Verarbeitung der gespeicherten Daten, da es aufgrund des Binärformates nicht *reparst* werden muss (The PostgreSQL Global Development Group 2018). *BSON* wird aus demselben Grund in der *NoSQL*-Datenbank *MongoDB*[3] verwendet. Mit diesen binären *JSON*-Formaten könnten in Vector Tiles Vorteile von *JSON* und *Protocol Buffers* kombiniert werden. Jedoch werden beide beim *OGC-Testbed 13* (Cavazzi 2018) nicht aufgeführt, so dass nicht mit einer Berücksichtigung bei der Erarbeitung eines offenen Standards zu rechnen ist. In der API-Referenz von *TileStache* (Kap. 15.1) wird auf binäre Formate von *GeoJSON* und *ArcJSON* verwiesen[4].

4.5.2.1 Geografic JSON (GeoJSON)

Butler et al. (2016) beschreiben *Geografic JSON* (GeoJSON) als ein Austauschformat für Geodaten, welches auf *JSON* basiert. Die derzeitige Spezifikation (RFC 7946[5]) kennzeichnet *GeoJSON* als *proposed standard*, wurde im August 2016 veröffentlicht und ersetzt die erste Spezifikation aus dem Jahr 2008 (GeoJSON.org 2018). Somit ist *GeoJSON* kein OGC-Standard, hat jedoch den Vorteil, dass es im Zusammenhang mit Webanwendungen weit verbreitet ist, da derartig strukturierte Daten leicht zu analysieren und in eine *JavaScript*-Anwendung zu integrieren sind (Cavazzi 2018). Als Dateierweiterungen werden von der Spezifikation *.json (z. B. bei *Tilestache* verwendet) oder *.geojson (z. B. bei *GeoServer* verwendet) vorgegeben.

In Version 1.0 der *GeoJSON Format Spezifikation* (Butler et al. 2008) war die Verwendung beliebiger CRS vorgesehen. Diese Option wurde in der aktuellen Version der Spezifikation (Butler et al. 2016) gestrichen, da nach Meinung der Autoren Verarbeitungssoftware im Allgemeinen keinen Zugriff auf CRS-Datenbanken besitzt und daraus Interoperabilitätsprobleme entstanden. Aus diesem Grund ist die Angabe von Koordinaten aktuell auf *WGS 84* (EPSG 4326) beschränkt. Diese Vorgabe wird jedoch aufgeweicht durch die Möglichkeit, bei Absprache aller Beteiligten, auch andere CRS zu verwalten. In diesem Sinne gibt z. B. auch *GeoServer* die Koordinaten in *GeoJSON*-Kacheln aktuell im CRS des Tilesets aus.

1 https://github.com/kothic/kothic-js
2 https://www.postgresql.org
3 https://www.mongodb.com
4 http://tilestache.org/doc/#vector-provider
5 https://tools.ietf.org/html/rfc7946

GeoJSON unterstützt die Geometrietypen *Point*, *MultiPoint*, *LineString*, *MultiLineString*, *Polygon* und *MultiPolygon* die alle auf dem fundamentalen Konstrukt *Position* aufbauen und zu *GeometryCollection* zusammengefasst werden können (Butler et al. 2016). Attributinformationen können in einer *properties*-Auflistung in Form von *Key Value Pairs* (KVP) verwaltet werden (Abbildung 9). Die Attributdaten von Tile-übergreifenden Features werden hierbei redundant in jedem Tile, welches Teile des Features enthält, gespeichert. So werden, im Gegensatz zur Verwendung von *tags* bei *Protocol Buffers* (Kap. 4.5.1), identische Inhalte in Textform wiederholt, z. B. die *keys* bei mehreren Features pro *geojson*-Datei. Zusätzlich werden die Realweltkoordianten als Dezimalzahlen übernommen, statt sie auf *Integer*-Bildschirmkoordinaten zu runden. Diese beiden Unterschiede sind wesentlich verantwortlich für den höheren Speicherplatzbedarf von *GeoJSON*, den Taraldsvik (2012) im Vergleich zu den dort untersuchten Binärformaten feststellt.

Abbildung 9 Attribute desselben Features in einer *Shape*-Datei (links, in *QGIS* 3.0) und in *GeoJSON* (rechts)
Quelle: Siedlungsgrenzen aus DLM250 (© GeoBasis-DE / BKG 2018)

In Vector Tiles im *GeoJSON*-Format können Features unterschiedlicher Feature-Klassen gespeichert werden. Die jeweilige Feature-Klasse kann über ein Attributfeld definiert sein. *GeoServer* nutzt für diesen Zweck das Feld *ID*, dessen Werte jeweils mit dem Namen des Eingangslayers beginnen und, durch einen Punkt getrennt, von einer fortlaufenden Nummer gefolgt werden (Kap. 5.5). *GeoJSON* ist ein feature-basiertes Format (Cavazzi 2018). Alle Informationen, einschließlich der Geometrien, werden unverändert übernommen. Generalisierungen bei Vector Tiles niedrigeren Zoomstufen sind davon ausgeschlossen, wobei auch hier die verbleibenden Vertices denen der Originaldaten entsprechen (Abbildung 8).

Die *Performance*-Vergleiche von Taraldsvik (2012) zeigen, dass *GeoJSON* mit 8 Bit *Integer*-Koordinaten vom Speicherplatz, der Abfragedauer und der Verarbeitungszeit teils deutlich hinter dem vergleichbaren Binärformat zurückliegt, aber bei der Dauer von Serverprozessen in der beschriebenen Architektur überlegen ist. Diese Ausführungen machen deutlich, dass mit Blick auf das Nutzungsziel bei der Einschätzung der Eignung alle relevanten Parameter zu berücksichtigen sind.

4.5.2.2 Topology JSON (TopoJSON)

Topology JSON (TopoJSON) ist eine Erweiterung von *GeoJSON*, welche es ermöglicht, Geometrien unter Beibehaltung der Topologie zu speichern (Bostock & Metcalf 2017). *TopoJSON* stellt ein kompakteres Datenformat dar als *GeoJSON* (Ross 2014; Cavazzi 2018). Es ist ebenso leicht zu *parsen* wie *GeoJSON*, hat jedoch den Nachteil, dass es aktuell nur wenig unterstützt wird (Cavazzi 2018).

Geometrien werden bei *TopoJSON*, in sogenannte *arcs* zerlegt und ohne Überlappungen (Wiederholungen von Abschnitten) gespeichert. Bei den einzelnen Features wird, statt der Angabe von Koordinaten wie bei *GeoJSON*, auf die verwendeten *arcs* verwiesen. Doppelte Geometrieninformationen werden so vermieden, was den Speicherbedarf reduziert. Darüber hinaus kann am Beginn einer *TopoJSON*-Datei ein *transform*-Element mit den Elementen *scale* und *translate* mit jeweils zwei Zahlenwerten eingefügt werden. Über diese Werte lassen sich die bei den *arcs* angegebenen Koordinatenwerte in Realweltkoordinaten transformieren. Auf diese Weise können bei den *arcs* speichersparende *integer*-Koordinaten vergeben werden, aus denen die Realweltkoordinaten zu errechnen sind (Code 3). Sobald ein *transform*-Element angegeben ist, wird pro *arc* jeweils nur die erste Position mit absoluten Werten gespeichert. Alle weiteren Positionen eines *arcs* werden in relativer Position zur vorherigen Position notiert (delta-codiert) (Bostock & Metcalf 2017).

Code 3 Funktion *transformPoint* zum Umrechnen von *TopoJSON*-Positionen zu Realweltkoordinaten

```
# aus Bostock und Metcalf (2017)
function transformPoint(topology, position) {
  position = position.slice();
  position[0] = position[0] * topology.transform.scale[0] +
                topology.transform.translate[0],
  position[1] = position[1] * topology.transform.scale[1] +
                topology.transform.translate[1]
  return position;
}

# Transform-Angaben aus einer TopoJSON-Datei = topology:
"transform":{
  "scale":[39135.75847656249,-39135.75847656249],
  "translate":[0.0,1.0018754169999998E7]
  }

# gespeicherte TopoJSON-Position: 202.1,137.5
# errechnete Realwelt-Koordinaten: 7909336.788113279229,4637587.379472655625
```

Aufgrund der Aufsplittung von Linien und Polygonen in *arcs* sind bei der Nutzung von Vector Tiles im *TopoJSON*-Format clientseitig erweiterte Mechanismen beim Zusammenfügen der Tiles erforderlich als z. B. bei *GeoJSON* mit *echten* bzw. zusammenhängenden Geometrien. Aufgrund der Genauigkeit der Geometrieinformationen (entweder Realweltkoordinaten oder *Integer* mit Transformationsinformation) kann *TopoJSON* als feature-basiertes Format angesehen werden.

Attribute werden bei Tile-übergreifenden Features redundant in jedem Tile gespeichert, welches Teile des Features enthält. Nach Cavazzi (2018) ist *TopoJSON* starr in der Anwendung und nicht flexibel genug, um alle Variationen geografischer Informationen, insbesondere Projektionen sowie alternative Kachelgrößen und -abmessungen, zu unterstützen.

4.5.2.3 UTFGrid

Das Format *UTFGrid* von *Mapbox* wurde zur schnellen Darstellung von Interaktivitätsdaten in Browsern, also für das Überlagern von Raster Tiles in *MBTiles*-Datenbanken (Kap. 4.6.2.1) entwickelt (MacWright et al. 2014). Ein auf Pixeln basierendes Kartenbild eignet nicht zum direkten Identifizieren von einzelnen Features. Hier wäre die Umrechnung der Pixelkoordinaten in Realweltkoordinaten und, basierend auf diesen, eine erneute Anfrage an den Kartendienst erforderlich (z. B. *GetFeatureInfo* eines WMS). Im Gegensatz dazu werden die Attributinformationen zu den Raster Tiles mit *UTFGrid* einfach und schnell lesbar in der-

selben *MBTiles*-Datenbank analgo zu den Pixelkoordinaten der Tiles mitgespeichert. Somit ist mit *UTFGrid* ohne vorherige Entschlüsselung eines Bildformates ein effizientes Hinzufügen von Interaktivität zu Kartenkacheln möglich (Käfer, 2011).

UTFGrid ist als *JSON*-Format zwar textbasiert, stellt jedoch gerasterte Daten dar. Analog zu Pixeln einer Bilddatei werden hierbei die enthaltenen Features mit UTF Textzeichen, ab Zeichen 32 und ohne die Zeichen 34 und 92, beginnend von oben links in einer Kachel kodiert. Jedes neue Feature der Kachel erhält fortlaufend ein neues UTF Zeichen zugewiesen. Diesen Textzeichen werden über ein Schlüssel-Array Attribute zugeordnet. Das zugrunde liegende Verfahren wird in der *UTFGrid*-Spezifikation[1] beschrieben. Über *JavaScript* können im Client die im *UTFGrid* enthaltenen Attribute den Pixeln der sichtbaren Kartenkachel zugeordnet werden. Die Datenübertragung ist aufgrund des schlanken Formates sehr schnell (Branigan 2013).

UTFGrid wird von einigen Servern als Ausgabeformat unterstützt. Seit 2013 hat keine Weiterentwicklung stattgefunden und auf der Internetseite von *Mapbox* ist kein Hinweis auf *UTFGrid* zu finden. In der aktuellen *MBTiles*-Spezifikation (Version 1.3) werden *UTFGrids* weiter unterstützt, sie sind jedoch nicht zwingend vorgeschrieben. Für *MBTiles*-Datenbanken mit Vector Tiles hat *UTFGrid* keine Bedeutung, da die Attributinformationen der Features in den Tiles enthalten sind und nicht separat verwaltet oder übertragen werden müssen.

4.5.2.4 eC Object Notation (ECON) und GeoECON

Die *eC Object Notation* (ECON)[2] ist ein textbasiertes Datenaustauschformat, welches von der *ECERE Corporation* entwickelt wurde. *ECON* ist als Obermenge von *JSON* definiert und enthält extra Features, die direkten Zugriff auf die *eC object instantiation* syntax, eine an *C* angelehnte objektorientierte Programmiersprache und Teil des *ECERE SDK*, erlauben (ECERE Corporation 2016).

GeoECON wurde als Nebenprodukt bei der Entwicklung der *GNOSIS-Tiles API*[3] realisiert und stellt eine textliche Repräsentation der *VectorFeatureCollection* in *eC* dar (Cavazzi 2018). *GeoECON* bietet die Möglichkeit, geografische Objekte im *ECON*-Format zu speichern. Koordinatenangaben werden in Dezimalgrad in *WGS 84* (EPSG 4326) gespeichert. Attribute werden nicht direkt in *GeoECON Vector Tiles* abgelegt, sondern in einer separaten Datei *attribs.econ*. Die Darstellung in dieser Datei ist textlich, die Struktur jedoch relational, vergleichbar den Tabellen in der *SQLite*-Datenbank eines *GNOSIS data stores* (Kap. 4.6.2.3) (Cavazzi 2018). Die Speicherung des räumlichen Index der Features erfolgt ebenfalls im *ECON*-Format.

Das *GeoECON*-Format bietet nach Cavazzi (2018) einige interessante Vorteile gegenüber *GeoJSON*, z. B.:

- Unterstützung für versteckte Kantensegmente in Polygonen.
- Es wird nicht erwartet, dass Polygone den ersten Punkt wiederholen.

4.5.3 GNOSIS compact vector tiles

GNOSIS map tiles stellen eine offene Spezifikation für eine kompakte, binäre Repräsentation geografischer Daten dar (ECERE Corporation 2017). Sie werden mit der *gmt*-Dateierweiterung gespeichert. *GNOSIS compact vector tiles* sind eine Form der *GNOSIS map tiles*. Die Kachelung erfolgt nach dem *Global GNOSIS tiling scheme*[4].

1 https://github.com/mapbox/utfgrid-spec
2 http://ec-lang.org/econ/
3 http://docs.opengeospatial.org/per/17-041.html#TilesAPI
4 http://docs.opengeospatial.org/per/17-041.html#_global_gno sis_tiling_scheme_adapted_to_polar_regions

Cavazzi (2018) beschreibt *GNOSIS compact vector tiles* detailliert. Nach dessen Auffassung liegt dieses Format zwischen feature- und render-basierten Lösungen. Die Realweltkoordinaten werden in 16-Bit-Integerwerte von -32.767 bis 32.767 innerhalb eines Tiles von Süd und West beginnend umgewandelt. Damit ist der Ansatz vergleichbar dem von *Mapbox Vector Tiles* (Kap. 4.5.1.1), jedoch mit höherer Genauigkeit (~0,015 m pro *Integer*-Wert bei 1 km Kachelausdehnung im Vergleich zu ~3,9 m bei einer 256x256 Kachel im mvt-Format). Der Speicherbedarf reduziert sich gegenüber *double*-Realweltkoordinaten (64bit) auf ein Viertel, was die Übertragungsgeschwindigkeit erhöht. Render-freundlich sind *GNOSIS compact vector tiles* zusätzlich, da Polygone in Dreiecke aufgesplittet (*pre-triangulated*) und damit *GPU ready*[1] gespeichert werden.

GNOSIS compact vector tiles können in Verzeichnissen oder in sogenannten *GNOSIS data stores* (Kap. 4.6.2.3) abgelegt werden. Im ersten Fall werden Attribute und räumliche Indizes in **.econ*-Dateien und im zweiten Fall in einer *SQLite*-Datenbank in einer relationalen Tabellenstruktur gespeichert. In *GNOSIS compact vector tiles* werden alle unterschiedlichen Vertices eines Tiles in einem Array gespeichert. Auf diese Koordinatenwerte wird über deren Index von den Features aus zugegriffen. Von mehreren Features genutzte Vertices werden so nicht wiederholt. Über *flags* wird definiert, ob ein Vertex Teil einer Tile-Grenze, und damit beim Rendern auszublenden ist. Darüber hinaus besteht die Möglichkeit Mittellinien für die Beschriftung von Polygonen zu definieren.

4.5.4 Geography Markup Language (GML)

Portele (2012) beschreibt die *Geography Markup Language* (GML) im offenen OGC-Standard als:

> *„... an XML grammar written in XML Schema for the description of application schemas as well as the transport and storage of geographic information.“*
>
> ***übersetzt:***
>
> *„... eine in XML-Schema geschriebene XML-Grammatik zur Beschreibung von Anwendungsschemata sowie zum Transport und zur Speicherung geografischer Informationen.“*

Der *default*-Wert des optionalen *outputFormat*-Parameters eines WFS ist *application/gml+xml*, weswegen jedes WFS *GML* unterstützen sollte (Portele 2012). Unter anderem aus diesem Grund ist *GML* ein weit verbreitetes Format und bietet sich deswegen grundsätzlich auch zur Speicherung von Vector Tiles an.

Der große Vorteil von *GML* besteht darin, dass alle Arten von Geometrien unterstützt werden, also im Gegensatz zu anderen Formaten auch Kurven, mit denen Geometrien genauer und effizienter als mit herkömmlichen Polylinien oder Polygonen dargestellt werden können (Cavazzi 2018). Da *GML* textbasiert ist, ist es vergleichsweise langsam zu verarbeiten (Zhang et al. 2007). Ein wesentlicher Nachteil von *GML* besteht im höheren Speicherbedarf, auch im Vergleich zu anderen textbasierten Formaten (Taraldsvik 2012; Cavazzi 2018). Der Vorteil der genaueren Geometriespeicherung mit Kurven wird darüber hinaus relativiert, da nach Cavazzi (2018) bisher keine Lösung zum *Clippen* von Kurven bei der Erstellung von Vector Tiles existiert. Dort wird als Lösung vorgeschlagen, die Kurven entweder komplett in jedem betroffenen Vector Tile zu verwalten und vom Client Duplikate entfernen zu lassen, was jedoch redundante Daten nach sich zieht, oder die Kurven vor dem *Clippen* in Linien oder Polygone umzuwandeln. Letzteres würde jedoch deren Vorteil der Genauigkeit, zumindest mit Blick auf feature-basierten Lösungen, zunichte machen.

1 http://docs.opengeospatial.org/per/17-041.html#Comparison

Bei den im Rahmen dieser Arbeit untersuchten Lösungen konnte kein Ansatz recherchiert werden, der *GML* als Speicherformat für Vector Tiles einsetzt, was sicher mit der geringen Performance zu begründen ist, die dem derzeit überwiegenden Einsatz von Vector Tiles in möglichst schnellen Webanwendungen widerspricht.

4.5.5 Weitere Formate

4.5.5.1 Mapsforge Binary Map File Format

Ein weiteres kompaktes, binäres Format für Vector Tiles stellt das *Mapsforge Binary Map File Format*[1] dar. Es wurde speziell für Hardware mit begrenzten Ressourcen, wie Mobiltelefone entwickelt (mapsforge 2018). Die Koordinaten werden als *Integer*-Werte in sogenannten *Mikrograden* (Dezimalgrad * 10^6) gespeichert. Das *Mapsforge Binary Map File Format* basiert auf einer Datei, welche sogenannte *Subfiles* für jeden Zoomlevel enthält. In diesen *Subfiles* sind die einzelnen Vector Tiles gespeichert. Die Spezifikation liegt in Version 0.9.0 vom Dezember 2017 vor. Seit 2012 ist dies die zweite veröffentlichte Aktualisierung.

4.5.5.2 Shape-Format

Das *Shape*-Format wird im Zusammenhang mit Vector Tiles kaum genannt. Der Vollständigkeit halber sei jedoch auf einen Ansatz in Kombination mit CDB-Datenbanken (Kap. 4.6.2.5) verwiesen, welchen Balog & Houtmeyers (2017a) wie folgt beschreiben:

> *„One of the core CDB concepts is the use of a tiled representation of the Earth with multiple levels of detail. Each tile can link to data, such as a Shapefile dataset consisting of vector features.“*
>
> ***übersetzt:***
>
> *„Eines der Kernkonzepte von CDB ist die Verwendung einer gekachelten Darstellung der Erde mit mehreren Detaillierungsgraden. Jede Kachel kann auf Daten verweisen, wie einen Shapefile-Datensatz, der aus Vektormerkmalen besteht.*

Da mit den oben beschriebenen Formaten kompaktere und flexiblere Möglichkeiten zum Speichern von Vector Tiles zur Verfügung stehen (vgl. auch „7 Gründe, die gegen Shapefiles sprechen“ aus GISpunkt HSR (2017)[2]), ist nicht davon auszugehen, dass *Shape*-Dateien bei dieser Thematik Verbreitung finden.

4.5.5.3 3D-Formate

Neben den beschriebenen Formaten für zweidimensionale Vector Tiles existieren noch Lösungen, die speziell für die Erfordernisse von 3D-Daten entwickelt wurden. So wurde für *Cesium 3D Tiles*[3] das *GL Transmission format* (glTF) ein binäres Format übernommen (Cavazzi 2018). Die *Indexed 3D Scene Layer* (I3S) und *Scene Layer Package Format* Spezifikation sind *OGC-Community Standards* (Reed & Belayneh 2017). I3S ist das Format, welches *ESRI* als Format für *Scene Layer* einsetzt. Sowohl *Cesium 3D Tiles* als auch *I3S* unterstützen alle CRS und beide Formate sind sowohl feature- als auch render-basiert (Cavazzi 2018).

1 https://github.com/mapsforge/mapsforge/blob/master/docs/Specification-Binary-Map-File.md
2 https://giswiki.hsr.ch/Shapefile
3 https://github.com/AnalyticalGraphicsInc/3d-tiles

4.6 Ablagestrukturen und -formate

Zum *Cachen* und Verteilen von Vector Tiles existieren grundsätzlich zwei Ablagekonzepte (Cavazzi 2018): So werden z. B. Vector Tiles, die über REST abgerufen werden sollen, in Verzeichnisstrukturen (Kap. 4.6.1) abgelegt. Die andere Möglichkeit ist, sie zusammen mit Metadaten in Datenbankstrukturen (Kap. 4.6.2) abzuspeichern, was einerseits die Weitergabe in nur einer Datei und andererseits eine effektive Indizierung ermöglicht. Als Weiterführung der Ablage in Verzeichnisstrukturen können die sogenannten *Packages* gesehen werden, welche ESRI verwendet. In diesen werden Vector Tiles oder I3S-Daten verwaltet (Kap. 4.6.3) und parallel alle Metainformationen mitgeliefert.

4.6.1 Verzeichnisstrukturen

Vector Tiles innerhalb von Verzeichnisstrukturen abzuspeichern ist eine aus dem Bereich der Raster Tiles bekannte und einfach nachzuvollziehende Lösung. In der Regel werden die Ordner und Dateien über die x, y und z-Koordinaten der Tiles bezeichnet und sind über diese einfach direkt abrufbar (Code 4).

Code 4 Beispiel-URL zum Zugriff auf Vector Tiles nach TMS-Kachelungsschema

```
# Beispielpfad für eine GeoJSON-Kachel der Layers sie01_f
# im Geoserver Arbeitsbereich DLM250, berechnet im CRS EPSG 4326,
# 'Bildformat' geojson
http://localhost:8080/geoserver/gwc/service/tms/1.0.0/DLM250:sie01_f@EPSG
%3A4326@geojson/{z}/{x}/{-y}.geojson
```

Wie in Kap. 4.3.1 beschrieben, wächst mit jeder Zoomstufe die Anzahl der Tiles bei regulärem Tiling um das Vierfache. Dies resultiert bei Ablage der Tiles in einzelne Dateien in eine entsprechend großen Anzahl mit dem Risiko eines *Overhead* des Dateisystems (Cavazzi 2018). Weiterhin stellt sich die Frage nach der Speicherung der Attributdaten. Bei fast allen im Rahmen dieser Arbeit untersuchten Lösungen, die eine Verzeichnisstruktur nutzen, werden diese direkt in den Tiles und damit bei Tile-übergreifenden Features redundant gespeichert, was aufgrund der größeren Datenmenge Performance-Einbußen bei der Übertragung und Nachteile beim Aktualisieren des Datenbestandes bedeutet: Es müssen immer alle Tiles aller Zoomstufen eines Features aktualisiert werden. Vor allem bei großflächigen Features (z. B. Russland) bedeutet dies bei Veränderungen der Attribute eines Features eine große Anzahl zu aktualisierender Kacheln. Eine andere Lösung ist im *GNOSIS map server* implementiert. Hier können die Kacheln im *GeoECON*-Format in einer Verzeichnisstruktur gespeichert werden, wogegen die Attribute in eine separate **.econ*-Datei ausgelagert werden (Cavazzi 2018).

Mit Version 2.3 enthält die *GDAL*-Bibliothek den sogenannten *MVT*-Treiber, mit dem der Zugriff auf *Mapbox Vector Tiles* in XYZ-Verzeichnisstrukturen möglich ist (vgl. Kap. 5.11). Die Beschreibung der Tiles entnimmt der Treiber den Metadaten der *TileJSON-Datei*, die im Stammverzeichnis abgelegt ist.

4.6.2 Datenbanken

Die Speicherung von Vector Tiles in Datenbanken hat verschiedene Vorteile. Sie ermöglichen eine Indizierung der Daten und eine einfache separate Verwaltung der Attributdaten ohne Redundanzen. Darüber hinaus vereinfachen sie die Weitergabe von Tiles innerhalb einer Datenbankdatei (Container). Sie erhöhen jedoch auch den Aufwand beim Zugriff auf die einzelnen Tiles im Vergleich zum direkten Zugriff in Verzeichnisstrukturen, da ein Extraktionsmechanismus erforderlich ist (Cavazzi 2018). Zusätzlich zu den nachfolgend aufgeführten Spezialformen von *SQLite*-Datenbanken, wie *MBTiles* (Kap. 4.6.2.1) und

GeoPackage (Kap. 4.6.2.2) wird bei Gesteira (2017) auf die Speicherung von *GeoJSON*-Vector Tiles in einer *SQLite*-Datenbank hingewiesen (Kap. 5.12).

4.6.2.1 MBTiles

MBTiles[1] bezeichnen eine Spezifikation für *SQLite*-Datenbanken zum Speichern, direkten Verwenden und effizienten Übertragen von gekachelten Geodaten (Fischer et al. 2018). *MBTiles*-Dateien, werden auch als *Tilesets* bezeichnet und stellen einen Container für Tiles, Indizes und Metadaten dar. Die *Mapbox-Vector-Tiles* (Kap. 4.5.1.1) werden als *Binary Large Objec*t (BLOB) in der Tabelle *tiles* abgelegt (mit Feldern für die Kachelkoordinaten `zoom_level=z, tile_column=x und tile_row=y`), wobei die Festlegung der Koordinaten dem *TMS*-Schema folgt (Fischer et al. 2018). Außerdem besteht die Einschränkung, dass nur *Web Mercator*-Koordinaten (EPSG 3857) unterstützt werden.

Der Zugriff auf eine *MBTiles*-Datenbank kann lokal erfolgen, z. B. mit dem *Vector Tiles Reader* Plugin für *QGIS* (Kap. 5.13) oder über einen Tileserver (z.B. *TileServer PHP*[2]), welcher *MBTiles* interpretieren und die enthaltenen Vector Tiles dem Client zur Verfügung stellen kann. *UTFGrids* (Kap. 4.5.2.3) zur Speicherung von Interaktionsdaten können zusätzlich in *MBTiles* verwaltet werden.

Seit *GDAL* Version 2.3 unterstützt der enthaltene *MBTiles*-Treiber auch Datenbanken, welche Vector Tiles enthalten[3] (vgl. Kap. 5.11).

4.6.2.2 GeoPackage

Der *OGC GeoPackage Encoding Standard*[4] wurde am 12.02.2014 mit Version 1.0 veröffentlicht und liegt aktuell in Version 1.2 (Veröffentlichung 25.08.2017) vor. Laut OGC (2017) beschreibt der Standard einen Satz von Konventionen zum Speichern von Vektordaten, Kachelsätzen von Rasterdaten und Erweiterungen in einer *SQLite* Datenbank.

Bis vor Kurzem war das Speichern von Vector Tiles in einer *GeoPackage*-Datenbank entsprechend der Spezifikation nicht möglich. Es existierten lediglich Vorgaben zum Speichern von Raster Tiles, die in dieser Form für Vector Tiles übertragbar wären. In Rahmen des OGC *Testbed 12* wurden Möglichkeiten zur Implementierung von Vector Tiles in *GeoPackage* diskutiert (Balog & Houtmeyers 2017b). Beim *OGC-CIO StandardsTracker*[5] wurden am 27.10.2017 diese Vorschläge vorerst mit dem Hinweis abgelehnt , dass sie nicht umsetzbar seien, bis sich das *OGC* sich für einen Ansatz (feature- oder render-basiert) für Vector Tiles entschieden habe (vgl. *GitHub Repository opengeospatial/geopackage*[6]). Mit Stand 28.06.2018 wurde auf der *OGC*-Internetseite[7] eine *Vector Tiles Extension*[8] veröffentlicht, welche Regeln und Anforderungen dafür definiert, wie *Mapbox Vector Tiles* (Kap. 4.5.1.1) in eine *GeoPackage*-Datenbank implementiert werden können.

1 https://github.com/mapbox/mbtiles-spec
2 https://github.com/klokantech/tileserver-php
3 https://www.gdal.org/frmt_mbtiles.html
4 http://www.opengeospatial.org/standards/geopackage
5 http://ogc.standardstracker.org/show_request.cgi?id=427
6 https://github.com/opengeospatial/geopackage/issues/271
7 http://www.geopackage.org
8 https://github.com/jyutzler/geopackage-vector-tiles/blob/master/spec/1_vector_tiles.adoc

4.6.2.3 GNOSIS data store

Im *GNOSIS data store* können alle Arten von Geodaten offline gespeichert werden (Cavazzi 2018). Entsprechend der Beispiele bei *GNOSIS Map Server*[9] besteht ein *GNOSIS data store* aus der Datei *layerInfo.econ*, in welcher die enthaltenen Layer beschrieben werden. Sämtliche Attribute und räumliche Indizes (als *R-Tree*) sind in der *SQLite*-Datenbank *attributes.sqlite* abgelegt. Die einzelnen Tiles sind in *tile pyramids archive files* (*.gmt) in Form von *Ecere-Archiven* gespeichert. Diese m*ulti-level tile pyramids* sollen eine Balance herstellen zwischen Dateianzahl und Dateigröße, weswegen mehrere Zoomstufen in einem Archiv zusammengefasst werden (Cavazzi 2018).

Zur Speicherung der Attribute, wird in der *SQLite*-Datenbank für jedes Textfeld eine separate Indextabelle („*STRINGS_...*“) angelegt, in welcher jeder in diesem Feld auftretende Wert einmal gespeichert wird. In der *Attributes*-Tabelle werden dann in den jeweiligen Feldern nur die *integer*-Werte der Indizes dieser Einträge gespeichert. In den *GNOSIS data stores* (nicht in den Vector Tiles!) werden ausschließlich Koordinaten im *WGS 84* (EPSG 4326) als 32 Bit *Integer*-Werte, errechnet aus den Dezimalgradangaben multipliziert mit 1E7, also auf 7 Dezimalstellen genau gespeichert (Cavazzi 2018).

4.6.2.4 PostgreSQL mit PostGIS-Erweiterung

PostgreSQL und mit der Erweiterung *PostGIS* eignet sich aufgrund der Unterstützung offener Formate wie z. B *GML* und *GeoJSON* grundsätzlich zur Speicherung von Vector Tiles. Taraldsvik (2012) beschreibt einen Ansatz unter Verwendung von *PostGIS* zum Vergleich unterschiedlicher Formate und weist darauf hin, dass *PostGIS* wichtige Methoden zur Generalisierung von Geometrien mitbringt (vgl. auch Kap. 5.3).

4.6.2.5 OGC Common DataBase (CDB)

Die *OGC Common DataBase* (CDB) Spezifikation definiert ein standardisiertes Modell und eine einheitliche Struktur für versionierbare virtuelle Darstellungen der Erde und ist vorrangig für Echtzeitsimulationsanwendungen ausgelegt (Reed 2017). Ein Vorteil von *CDB* liegt in der Möglichkeit Raster- und Vektordaten zu verwalten, was die Effizienz für Visualisierungs- und Analyseoperationen erhöht (Cavazzi 2018).

Entsprechend der *CDB*-Spezifikation sollten geografische Standorte mit *WGS 84* Koordinaten ausgedrückt werden (Reed 2017). Unter anderem aus diesem Grund sieht Cavazzi (2018) CDB nicht als exklusive Basis für einen künftigen Standard sondern als Idee für einen möglichen Ansatz, um Daten in einer Datenbank zu strukturieren, die Vector Tiles bereitstellen kann.

4.6.2.6 Not only SQL Datenbanken (NoSQL)

Die beim Tiling entstehende große Anzahl von Einzeldateien stellt Speichersysteme vor große Herausforderungen. Aus diesem Grund ist die Ablage der Tiles in verteilten Speichern ein denkbarer Ansatz, um die Effizienz beim Zugriff auf diese Daten zu verbessern. Zur schnellen Bereitstellung verteilter Daten für Clients können hierbei *Not only SQL Datenbanken* (NoSQL) eingesetzt werden. Basierend auf dem *BSON*-Format (Kap. 4.5.2) könnte hier *MongoDB* als dokumentbasierte *NoSQL*-Datenbank eingesetzt werden. Wan et al. (2016) beschreiben einen Framework unter Einsatz der spaltenbasierten *NoSQL*-Datenbank *HBase*.

9 http://maps.ecere.com

4.6.3 ESRI Vector Tile Packages

Mit sogenannten *Packages* versucht ESRI die Vorteile von Verzeichnisstrukturen (schnellerer, direkter Zugriff auf die Tiles (Přidal 2016)) und das kompaktere Format (nur eine Datei inklusive aller Metadaten, Indizes usw. und damit einfachere Weitergabe (Cavazzi 2018)) zu kombinieren. Für *3D-Scene Layer* wurde das als *OGC*-Community *Standard* veröffentlichte *Scene Layer Package Format* und für Vector Tiles das proprietäre *Vector Tiles Package Format* entwickelt. Als Datenformat für die einzelnen Tiles in *Vector Tiles Packages* nutzt *ESRI* das *Mapbox Vector Tile* Format (Kap. 4.5.1.1). Das Erstellen und die Verwendung von Vector Tiles im proprietären *Vector Tiles Package Format* wird in Kap. 5.14 detailliert beschrieben.

4.7 Metadaten

Standardisierte Metadaten, wie sie z. B. im *GetCapabilities-Response* eines *WMS* enthalten sind, möglichen eine automatisierte Verarbeitung der Inhalte und Nutzung der vom Service angebotenen Daten. Außerdem kann beispielsweise bei einer Anfrage an den Service über die Einschränkung der geografische Ausdehnung und die Angabe eines CRS die Verarbeitung der Daten beschleunigt werden.

Im Einsatzbereich von Vector Tiles gibt es viele unterschiedliche Ansätze Metadaten bereitzustellen. Dem *WMTS* vergleichbar werden beim *TMS* der Service selbst und die verfügbaren Daten mit Hilfe von *Root Resource*, *TileMapService Resource* und *TileMap Resource* beschrieben (Kap. 4.3.2.2). Mit der *TileJSON-Spezifikation* von *Mapbox*[1] wird versucht, einen Standard für die Darstellung von Metadaten zu erstellen, um Clients mit diesen Informationen bei der Konfiguration und beim Browsen zu unterstützen (Käfer et al. 2018). Bei Cavazzi (2018) wird das *Vector Tiles Client-Plugin* (Kap. 5.13) vorgestellt, welches *Mapbox Vector Tiles* mit Hilfe der Definitionen in *TileJSON* lesen kann. Identisch erfolgt der Zugriff auf Tiles mit dem *Vector Tiles Reader QGIS-Plugin* (Kap. 5.13). Ausgewählte Metadaten (Layer, Datentyp, Zoomlevel, Überdeckung) werden in *GNOSIS data stores* (Kap. 4.6.2.3) in der Datei *layerinfo.econ* gespeichert. In *MBTiles*-Datenbanken (Kap. 4.6.2.1) werden die Metadaten in der Tabelle *metadata* als *JSON*-Repräsentation verwaltet. Ebenso bietet *GeoPackage* (Kap. 4.6.2.2) die Möglichkeit, in zwei Tabellen Metadaten bereitzustellen und mit den in der Datenbank enthaltenen Daten zu verknüpfen. Wobei hier durch die Spezifikation keine Erfordernis und keine feste Hierarchie zum Bereitstellen von Metadaten vorgegeben sind (OGC 2017). Für *Cesium 3D Tiles* (Kap. 4.5.5.3) werden Tilsets in *tileset.json*-Dateien definiert. In *Scene Layer Packages*, welche der Verwaltung von *Indexed 3D Scene Layern* (Kap. 4.5.5.3) dienen, werden Metadaten in der Datei *Metadata.json* abgelegt.

1 https://github.com/mapbox/tilejson-spec

5 Software

Im Rahmen der vorliegenden Arbeit wurden insgesamt 30 Softwarepakete, die im Zusammenhang mit Vector Tiles stehen, recherchiert. Gesucht wurden dabei Lösungen, die Vector Tiles generieren, bereitstellen, serverseitig in Bitmaps rendern und als WMS veröffentlichen können. Da sich schnell abzeichnete, dass die direkte Verwendung von Vector Tiles als Datenquelle für einen WMS nicht möglich ist, wurden zusätzlich Wege gesucht, wie ein WMTS als Datenquelle für einen WMS genutzt werden kann.

Wesentliche Voraussetzung für die weitere Berücksichtigung dieser Programme und Bibliotheken stellten die potenzielle Verwendbarkeit bei der Umsetzung der Fragestellung dieser Arbeit, eine möglichst einfache Installation und eine aktive Entwicklung dar. Der letzte Punkt soll gewährleisten, dass bei mehreren potenziell geeigneten Lösungen diejenige weiterverfolgt wird, die mit hoher Wahrscheinlichkeit längerfristig weiterentwickelt wird und damit verfügbar bleibt. Die Angabe wurde aus den Informationen zu Aktualität, *Releases* u. a. abgeleitet, die beim jeweiligen *Repository* auf *GitHub* verfügbar sind. Diese entsprechen einheitlich dem ***Stand vom 21.08.2018***, um eine Vergleichbarkeit der Angaben zu gewährleisten. Eine komplette Übersicht ist der Arbeitstabelle[1] im GitHub-Repository zu dieser Masterthesis[2] zu entnehmen. Basierend auf den ermittelten Informationen wurden 14 Programme und Bibliotheken in die nachfolgenden, detaillierteren Ausführungen einbezogen (Tabelle 3). Die Ausschlusskriterien für die verbleibenden Programme und Bibliotheken sind in der o. g. Arbeitstabelle aufgeführt.

Software zum Rendern von Bildern (Raster Tiles) aus Vektordaten (Vektor Tiles) hat bei der vorliegenden Arbeit dann eine Bedeutung, wenn sie serverseitig eingesetzt wird und Bildkacheln erzeugt, welche dann direkt (vgl. *Mapnik* (Kap. 5.10) und *Mapproxy* (Kap. 5.8)) oder über einen WMTS als Datenquelle für einen WMS genutzt werden können. Die Clients *ArcGIS Pro* und *ArcMap* (Kap. 5.14) sowie *QGIS* (Kap. 5.13) wurden bei den Recherchen untersucht, um die Aussage zu untermauern bzw. zu widerlegen, dass diese verbreitetsten GIS-Clients nicht oder nur eingeschränkt zur direkten Nutzung von Vector Tiles befähigt sind.

5.1 TileStache

TileStache[3] ist eine in *Python* programmierte Serveranwendung zur Bereitstellung von Kartenkacheln, die 2010 entwickelt wurde und der *BSD 3-Lizenz* (Kap. 3.3) unterliegt. *TileStache* liegt in Version 1.51.10 mit einem Alter von weniger als einem Monat vor. Auf *GitHub* ist das Projekt erst seit 2016 vorhanden, dort darüber hinaus nicht aktuell. Von einer regelmäßigen Weiterentwicklung kann aufgrund der vollständigen *history* des Projektes bei *pypi.org*[4] ausgegangen werden.

Nach Balog & Houtmeyers (2017b) war *TileStache* eine der ersten Implementierungen von Vector Tiles im *GeoJSON*-Format. Neben *GeoJSON*-Vector Tiles werden u. a. auch *MVT* und *TopoJSON* unterstützt. Die Konfiguration von *TileStache* erfolgt im *JSON*-Format. Bei der Definition der Eingangslayer sind unterschiedliche Datenquellen pro Zoomlevel definierbar oder es wird eine einzige Datenquelle für alle Zoomlevel angegeben[5]. Mit dem *VecTiles*-Package[6] wird ein Provider implementiert, der Vector Tiles in den genannten drei Formaten bereitstellt. Dieser versorgt außerdem *Mapnik* (Kap. 5.10) mit der *VecTiles*

1 https://github.com/GjueAtGit/VTs_datasource_ogc/tree/master/src
2 https://github.com/GjueAtGit/VTs_datasource_ogc
3 http://tilestache.org/
4 https://pypi.org/project/TileStache/#history
5 http://mattmakesmaps.com/blog/2013/10/09/tilestache-rendering-topojson/
6 http://tilestache.org/doc/TileStache.Goodies.VecTiles.html

Datasource, welche Tiles im CRS EPSG 3857 (*Spherical Mercator*) ohne die Nutzung einer lokalen *PostGIS*-Datenbank bereitstellt (Migurski 2018). In der *TileStache* API sind die CRS EPSG 3857 und EPSG 4326 beschrieben, grundsätzlich können in *TileStache* jedoch auch eigene Projektionen definiert werden[1]. Diese sind jedoch über die beschriebene Anbindung von *Mapnik* nicht verarbeitbar. Mit der *decode*()-Funktion[2] ist es möglich, *GeoJSON*-Dateien in eine *Well-known binary* (WKB) Repräsentation umzuwandeln und anschließend direkt an *mapnik.PythonDatasource.wkb_features()*[3] zu übergeben.

Aus der *TileStache API* geht nicht hervor, ob die erstellten Tiles in *GeoJSON* und *TopoJSON* ausschließlich menschenlesbare Repräsentationen der deckungsgleichen Tiles im *MVT*-Format sind oder genauere Informationen, z. B. die originalen Realweltkoordinaten statt Bildschirmkoordinaten, enthalten.

5.2 t-rex

Der Tileserver *t-rex*[4] ist eine relativ junge, in *Rust*[5] programmierte Anwendung zum Erzeugen und Bereitstellen von Vector Tiles aus *PostGIS*-Datenbanken, in welchen er die Layer automatisch erkennt, und *GDAL*-Vektorformaten (Kalberer 2017a). *T-rex* unterliegt der *MIT-Lizenz* (Kap. 3.3) und ist aktuell in Version 0.9.0 verfügbar. Seit 2016 wurden regelmäßig neue Releases veröffentlicht.

T-rex verfügt über zwei eingebaute *tile grids* (EPSG 3857 und EPSG 4326), es können aber auch eigene Gitter definiert werden (Kalberer 2018). Als Ausgabeformat stehen ausschließlich *Mapbox Vector Tiles* (*MVT*) zur Verfügung, die in einem Dateisystem-Cache abgelegt werden. Bei der Konfiguration von Layern bietet *t-rex* im Experimentalstatus die Möglichkeit der Einbettung von Stilen entsprechend der *Mapbox Style Specification* im *TOML*-Format[6]. Die so definierten Stile werden von *t-rex* im *JSON*-Format bereitgestellt.

Im Softwarepaket sind ein eingebetteter Webserver und der *t-rex Vector Tiles Viewer* zur Darstellung der Daten enthalten. Das *Styling* der Tiles kann direkt im Browser mit dem ebenfalls integrierten *Maputnik*[7] entsprechend der *Mapbox Style Specification*[8] erfolgen. Für zukünftige Versionen sind u. a. *Clipping* und Vereinfachung von *GDAL*-Vektorlayern und eine Performance-Optimierung für große Eingangsgeometrien geplant (Kalberer 2017a). Aktuell werden nur Geometrien aus *PostGIS*-Layern generalisiert: Linien mit *Douglas-Peucker*, Polygone mit *SnapToGrid* (Kalberer schriftlich 17.09.2018).

T-rex kann, neben dem Bereitstellen von Vector Tiles *on the fly* mit dem Kommando *generate* einen Tile *Cache* in einer Verzeichnisstruktur erzeugen. Basierend auf diesem *Cache* können die Vector Tiles mit dem Tool *mbutil*[9] von *Mapbox* in *MBTiles*-Datenbanken verpackt werden. Die Konfiguration von *t-rex* erfolgt in der einfach nachvollziehbaren *TOML*-Syntax[10]. Die Konfigurationseinstellungen können mit dem Kommando *genconfig* aus einem Eingangsdatensatz berechnet werden.

1 http://tilestache.org/doc/#projections
2 http://tilestache.org/doc/TileStache.Goodies.VecTiles.geojson.html
3 https://mapnik.org/docs/v2.2.0/api/python/mapnik.PythonDatasource-class.html
4 https://t-rex.tileserver.ch
5 https://www.rust-lang.org
6 https://pka.github.io/mapbox-gl-style-spec/
7 https://maputnik.github.io
8 https://www.mapbox.com/mapbox-gl-js/style-spec/
9 https://github.com/mapbox/mbutil
10 https://github.com/toml-lang/toml

5.3 PostGIS ST_AsMVT

Aus einer *PostGIS*-Datenbank heraus können seit Version 2.4 (veröffentlicht 30.09.2017) *Mapbox Vector Tiles* (Kap. 4.5.1.1) mit der Funktion *ST_AsMVT*[1] mit *SQL* direkt aus Vektor Layern erstellt werden. Die Funktion *ST_AsMVT* gibt eine *Mapbox Vector Tile* Darstellung einer Menge von Zeilen eines Vektorlayers zurück. Hierbei werden die Geometrien mit der Funktion *ST_AsMVTGeom* in den Koordinatenraum der Vector Tiles transformiert.

Beim Erstellen von Vector Tiles mit *ST_AsMVT* ist es möglich, die Geometrien entsprechend der Zoomstufe zu generalisieren. Dabei können z. B. die *PostGIS*-Funktionen *ST_Simplify* (Vereinfachung mit *Douglas-Peucker-Algorithmus*), *ST_SimplyPreserveTopology* (Vereinfachung mit *Douglas-Peucker-Algorithmus* unter Bewahrung der Topologie), *ST_SnapToGrid* (Punktfang an einem regulären Gitter), *ST_SimplifyVW* (Vereinfachung mit *Visvalingam-Whyatt-Algorithmus*) oder *ST_SetEffectiveArea* (Vereinfachung unter Berücksichtigung der effektiven Fläche eines Knotens) eingesetzt werden. Ein Beispiel mit Vereinfachung unter Nutzung von *ST_Simplify()* zeigt Code 5. Das daraus resultierende Ergebnis, einem Vector Tile ohne Vereinfachung gegenübergestellt, ist in Abbildung 10 erkennbar.

Code 5 Beispielcode zum Erstellen eines Vector Tiles (Ausdehnung *ST_MakeBox2D()*) in PostGIS mit der Funktion *ST_AsMVT* mit Geometrievereinfachung (*ST_Simplify()*)
basiert auf *PostGIS* 2.5 Dokumentation und de la Torre (2017)

```
SELECT ST_AsMVT(tile, 'Layername', 4096, 'geom') FROM (
  SELECT qbounds.id, qbounds.von,
         ST_Simplify(ST_AsMVTGeom(qbounds.geom,
            ST_MakeBox2D(ST_Point(1252051, 6731468),
            ST_Point(1408711, 6888128)), 4096, 256, true), 2500***, true
         ) AS geom
  FROM (
   SELECT *
   FROM example_layer
   WHERE geom &&
         ST_MakeEnvelope(1252051, 6731468, 1408711, 6888128, 3857)) AS qbounds
) AS tile;

# *** Toleranz: ergibt sich in produktiver Anwendung aus dem Zoomlevel,
#               Beispiel für EPSG 3857 hier[2]
```

Entsprechend der Dokumentation können mehrere *ST_As MVT*-Aufrufe zu Vector Tiles mit mehreren Layern verknüpft werden. Nach Harrtell (2016) erfolgt dies mit `ST_AsMVT(...) || ST_AsMVT(...)`, da eine Kachel, entsprechend der *Mapbox Vector Tiles Spezifikation*[3], aus einer Sammlung von Layern unterhalb der Wurzelebene besteht (siehe dazu auch folgende Diskussion[4]). In SQL kann dies mit *UNION ALL* umgesetzt werden (Code 6).

1 https://postgis.net/docs/manual-2.5/ST_AsMVT.html
2 https://github.com/openstreetmap/mapnik-stylesheets/blob/master/zoom-to-scale.txt
3 https://www.mapbox.com/vector-tiles/specification/
4 http://osgeo-org.1560.x6.nabble.com/ST-AsMVT-td5286323.html

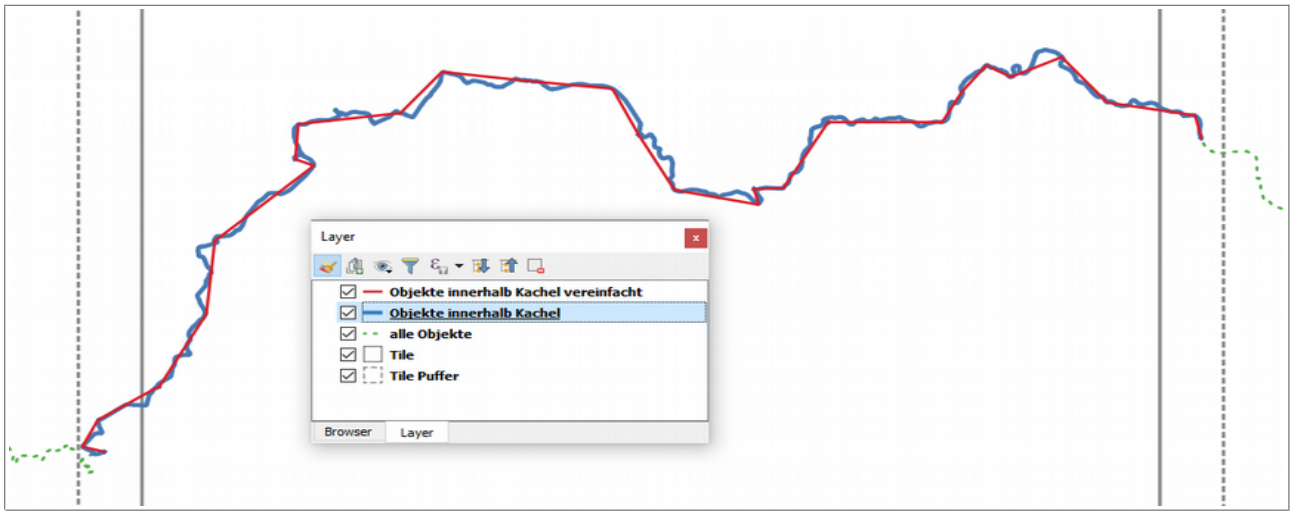


Abbildung 10 Ergebnisanzeige von Linien in *QGIS* 3.2, erstellt mit der *PostGIS*-Funktion *ST_AsMVT()*
Gegenüberstellung von Originaldaten, Objekten innerhalb der vorgegebenen Kachel und diese Objekte mit vereinfachter Geometrie (vgl. Code 5)
Quelle: *Verkehrswege* aus *DLM1000* (© GeoBasis-DE / BKG 2018)

Code 6 Verknüpfung von zwei Layern in einem Vector Tile

```
# SELECT-Anweisung basierend auf Code 5

SELECT ST_AsMVT(tile, 'Layername 1', 4096, 'geom') FROM (...) AS tile;
UNION ALL
SELECT ST_AsMVT(tile, 'Layername 2', 4096, 'geom') FROM (...) AS tile;
```

Allegri (2017) demonstriert ein Beispiel, in welchem mit Hilfe von *Python* Vector Tiles aus einer *PostGIS*-Datenbank generiert und in einer Verzeichnisstruktur abgelegt werden, die dann z. B. via REST mit `/{z}/{x}/{y}.pbf` abrufbar wären. Bei einer Anwendung zum Vorprozessieren von Vector Tiles eines kompletten Datensatzes (z. B. DLM1000, Kap. 3.2) in eine Verzeichnisstruktur oder ein *MBTiles*-File müsste eine vergleichbare Routine entwickelt werden. Der Vector Tile Server *postserve*[1] nutzt *ST_AsMVT* zur *on the fly* Generierung von *Mapbox Vector Tiles*. Ein Blick in den Quellcode der Datei *server.py* im *GitHub Repository* von *postserve* zeigt, dass hier keine Vereinfachung der Geometrien implementiert ist. Die Integration von *ST_AsMVT* in *t-rex* (Kap. 5.2) ist geplant[2] (Kalberer 2017a).

5.4 *tessera* Tileserver

Tessera[3] ist ein auf *Tilelive*[4] basierender Tile Server. Er ist in *JavaScript* programmiert, wurde 2014 erstmals veröffentlicht, seitdem aktiv weiterentwickelt und liegt aktuell in Version 0.13.0 vor. *Tessera* unterliegt der *BSD 2 Lizenz* (*tilelive* der *BSD 3 Lizenz;* Kap. 3.3). Über *node.js*[5] kann *tessera* auch auf Servern eingesetzt werden.

1 https://github.com/openmaptiles/postserve
2 https://github.com/t-rex-tileserver/t-rex/projects/1
3 https://github.com/mojodna/tessera
4 https://github.com/mapbox/tilelive
5 https://nodejs.org

Entsprechend des *tessera* und des *tilelive GitHub Repository* sind für eine Erstellung von Vector Tiles, die anschließend über *tessera* bereitgestellt werden, zusätzliche *tilelive*-Module erforderlich. Bei der Nutzung von *tessera* müssen die erforderlichen *tilelive* Module mit den Abhängigkeiten entsprechend der eigenen Konfiguration manuell installiert werden (Fitzsimmons 2018). Sie werden danach automatisch erkannt und geladen. Wenn die recht kurze Dokumentation richtig interpretiert wurde, werden mit *tilelive* Vector Tiles im EPSG 3857 erzeugt und als Tiling Schema wird *XYZ* unterstützt.

Über das Modul *tilelive-mapnik*[1] können mit *node-mapnik* Raster Tiles von einer *Mapnik XML*-Datei gerendert werden. Nach Martinelli & Roth (2015 p. 49) ist *tessera* auf 50 gleichzeitige Nutzeranfragen limitiert.

5.5 GeoServer und GeoWebCache

GeoServer[2] ist ein in *Java* programmierter Webserver, mit dem Karten und Daten aus einer Vielzahl von Formaten über standardbasierte Schnittstellen wie WMS bereitgestellt werden können (GeoServer Contributors 2018). In *GeoServer* ist *GeoWebCache*[3] als Kachelserver implementiert, welcher als Proxy zwischen *GeoServer* und dem Client fungiert (Bühner et al. 2018). *GeoServer* liegt aktuell in Version 2.13.2 vor und wird, genau wie *GeoWebCache* (aktuell Version 1.12.5), von einer großen, aktiven Community regelmäßig weiterentwickelt (Cavazzi 2018). Nach deren Angaben wird *GeoServer* in einem 6-monatigen Zyklus fortgeschrieben, mit einer Herausgabe von neuen Releases aller zwei Monate[4]. *GeoServer* und *GeoWebCache* unterliegen der *GNU GPL 3-Lizenz* (Kap. 3.3).

GeoServer ist über *Java*-Bibliotheken einfach erweiterbar. Bühner et al. (2018) beschreiben *GeoServer* dahingehend wie folgt:

> *„Der GeoServer ist ein offener, **Java**-basierter Server, der es ermöglicht Geodaten auf Basis der Standards des Open Geospatial Consortium (OGC) (insb. WMS und WFS) anzuzeigen und zu editieren. Eine besondere Stärke des GeoServers ist die Flexibilität, mit der er sich um zusätzliche Funktionalität erweitern lässt.“*

GeoWebCache ist ebenfalls in *Java* programmiert und kann *Tiles* von Kartendaten aus unterschiedlichen Quellen *cachen*. Er implementiert verschiedene Service-Schnittstellen (u. a. WMS-C, WMTS, TMS), um die Lieferung von Kartenbildern zu beschleunigen und zu optimieren und kann (WMTS-)Kacheln auch neu kombinieren, um mit normalen WMS-Clients zu arbeiten (GeoWebCache Contributors 2018). *GeoWebCache* kann als *standalone*-Version[5] auch unabhängig von *GeoServer* mit anderen WMS-kompatiblen Servern betrieben werden (GeoWebCache Contributors 2018). Das Cachen von Kacheln kann o*n the fly* beim ersten Aufruf eines Kartenausschnitts oder durch Vorprozessierung aller *Tiles* aller Zoomstufen erfolgen. Die Ablage der Kacheln folgt dem *TMS-Schema* (Kap. 4.3.2.2) in Verzeichnissen entsprechend der Eingangslayer, des CRS und der Zoomstufen. Die einzelnen Kacheldateien sind mit deren X- und Y-Koordinaten benannt (Code 7). Zum effektiven Zugriff werden von *GeoWebCache* innerhalb der Verzeichnisse der Zoomstufen Unterordner angelegt (in Code 7 `/1_2/`). Deren Bezeichnung wird mit der *FilePathGenerator*-Klasse[6] berechnet. Aus diesem Grund ist die Verzeichnisstruktur nicht zum externen Zugriff, sondern nur zur internen Nutzung gedacht und in der Anleitung auch nicht dokumentiert (Smith 2014).

1 https://github.com/mapbox/tilelive-mapnik
2 http://geoserver.org
3 http://geowebcache.org
4 http://geoserver.org/roadmap/
5 http://geowebcache.org/docs/current/introduction/whichgwc.html
6 https://github.com/GeoWebCache/geowebcache/blob/master/geowebcache/core/src/main/java/org/geowebcache/storage/blobstore/file/FilePathGenerator.java

Code 7 Beispielpfad eines Kachelcaches in *GeoWebCache* im TMS-Schema

```
# Beispielpfad für eine GeoJSON-Kachel der Layers ne_10m_admin_0_countries
# im GeoServer Arbeitsbereich Natural_Earth, berechnet im CRS EPSG 900913,
# Zoomstufe 5, mit den Kachelkoordinaten X=8 und Y=17
../Natural_Earth_ne_10m_admin_0_countries/EPSG_900913_05/1_2/08_17.geojson
```

Vector Tiles können als Ausgabeformat von *GeoServer* über die *Vector Tiles Extension*[1] hinzugefügt werden. Nach deren Installation sind zusätzlich zu den standardmäßig angebotenen Bilddateiformaten die Vector Tile-Formate *Mapbox Vector (MVT)*, *GeoJSON* und *TopoJSON* verfügbar. Sie können über die TMS-Schnittstelle abgerufen werden. Mit Blick auf *Mapbox Vector Tiles* (Kap. 4.5.1.1) hält sich diese Erweiterung jedoch nicht an die Empfehlung der entsprechenden Spezifikation, die Tiles mit der Dateinamenserweiterung *mvt* zu versehen, sondern verwendet *pbf*. Darüber hinaus wird auch nicht der in der Spezifikation empfohlene *MIME-Typ application/vnd.mapbox-vector-tile* sondern stattdessen *application/x-protobuf;type=mapbox-vector* genutzt. Vector Tiles im *MVT*-Format kann *GeoServer* ebenfalls mit der Erweiterung *gs-mvt*[2] über einen WMS-Request oder als *XYZ*-Tiles bereitstellen, jedoch werden diese nur *live*, ohne Nutzung eines Caches, erzeugt (Henneberger 2015). Analog beschreibt Cavazzi (2018) auch eine Erweiterung des WFS in *GeoServer* um eine Vector Tile Komponente[3].

Laut Cavazzi (2018; Kap. 6.3.7) nutzt *GeoServer* keine Generalisierungstechnik beim Erstellen der Vector Tiles. Dem widerspricht die Aussage von Henneberger (2015), nach welcher *GeoServer* den *Douglas-Peucker Algorithmus* (Douglas & Peucker 1973) zum Generalisieren der Geometrien anwendet. Dies bestätigen auch das Beispiel von Blasby und Hocevar (2016a; Folie 14) und eigene Untersuchungen mit Vergleichen von *GeoJSON* Vector Tiles, die in unterschiedlichen Zoomstufen erzeugt wurden (Abbildung 3).

Cavazzi (2018) bezeichnet *GeoServer* mit Blick auf Vector Tiles als eine feature-basierte Lösung. Diese Aussage kann nach eigenen Untersuchungen so allgemein ebenfalls nicht bestätigt werden, da mit dem *GeoJSON*- (Kap. 14.5.2.1) und dem *TopoJSON*-Format (Kap. 4.5.2.2) feature-basierte Formate und mit *Mapbox Vector Tiles* (Kap. 4.5.1.1) ein render-basiertes Format erzeugt werden. In Abbildung 8 wird die Auswirkung von mit *GeoServer* erzeugten *GeoJSON*- und *Mapbox Vector Tiles* auf die Genauigkeit von Polygonen visualisiert.

Mit Hilfe der *Vector Tiles Extension* können mit *GeoServer*, wie schon beschrieben, Vector Tiles im *TopoJSON*-Format zur Verfügung gestellt werden. Entsprechend der Spezifikation dieses Formates (Bostock & Metcalf 2017) müssten sich z. B. benachbarte Staaten für die Grenzen einen oder mehrere *arcs* teilen und deren Topologie müsste erhalten bleiben (Kap. 4.5.2.2). Dies ist jedoch nicht der Fall, wie der Test mit Daten aus *NaturalEarth* zeigt (Code 8 und Abbildung 11). Hier werden für den identischen Polygonabschnitt nicht nur zwei *arcs* erzeugt, was der Spezifikation widerspricht, diese enthalten im Resultat auch noch unterschiedliche Vertices, da bei der Generalisierung der Polygone der beiden Staaten an unterschiedlichen Stellen begonnen wird. In den Originaldaten verläuft die hier gezeigte Grenze zwischen Deutschland und Österreich absolut identisch, also mit denselben Vertices. Sogenannte *degenerate cases* (Kim & Kim 2006) können somit als Fehlerquelle ausgeschlossen werden.

1 http://docs.geoserver.org/latest/en/user/extensions/vectortiles/index.html
2 https://github.com/emplexed/gs-mvt
3 http://docs.opengeospatial.org/per/17-041.html#WFS_for_Vector_Tiling

Code 8 Auszug aus einem Vector Tile imTopoJSON-Format
(Feature *Austria* verweist auf nur einen *arc*, was identische *arcs* mit unterschiedlichen Nachbarländern ausschließt. *Feature* Germany belegt dieses, da auf *Arc 348* von *Austria* nicht verwiesen wird)

```
... {"type":"Polygon",...
 "properties":{... "NAME":"Austria", ...},
 "arcs":[[348]]},
...
{"type":"MultiPolygon",...
"properties":{... "NAME":"Germany", ...},
"arcs":[[[1239]],[[1240]],[[1241]],[[1242]],[[1243]],[[1244]],[[1245]],
[[1246]],[[1247]],[[1248]],[[1249]],[[1250]],[[1251]],[[1252]],[[1253]],
[[1254]],[[1255]],[[1256]],[[1257]],[[1258]],[[1259]],[[1260]]]}, ...
```

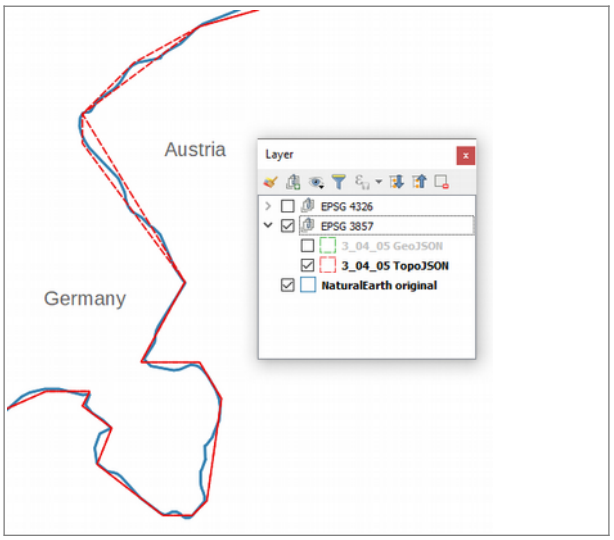


Abbildung 11 Topologiefehler (rote Linien) in Vector Tiles im *TopoJSON*-Format, erstellt mit *GeoServer*, Zoomstufe 3 EPSG 3857
Quelle: Layer *Admin 0 - Countries* www.naturalearthdata.com/

5.6 MapServer und MapCache

MapServer (auch *UMN MapServer*)[1] ist eine in C und C++ programmierte Software zum Bereitstellen von OGC-konformen Diensten. Die Weiterentwicklung erfolgt regelmäßig von einer großen, aktiven Community. Aktuell liegt Version 7.2.0 vor. *Mapserver* unterliegt der *MIT-Lizenz* (Kap. 3.3). Seit Version 6 sind *Caching*-Funktionen für Kacheln über das *MapCache*-Projekt[2] (aktuell Version 1.6.1) in *MapServer* implementiert (MapServer Contributors 2018).

Dank der Einbindung der *Proj.4-Bibliothek*[3] ist *MapServer* in der Lage, zwischen den in dieser Bibliothek enthaltenen Projektionen *on the fly* umzuprojizieren. Im sogenannten *Mapfile*[4] (Erweiterung **.map*) erfolgt in *MapServer* die Beschreibung bzw. Konfiguration eines Kartenprojektes in Textform über in der Dokumentation vorgegebene Parameter. Entsprechend der Dokumentation unterstützt *MapServer GDAL* und müsste bei Einbindung von Version 2.3 *Mapbox Vector Tiles* in Verzeichnissen und als *MBTiles* lesen und schreiben können. Im Rahmen der vorliegenden Arbeit ist v. a. die Möglichkeit Vector Tiles zu lesen interessant. Basierend auf der Dokumentation von *MapServer* wurde versucht, auf Vector Tiles in einer *MBTiles*-Datenbank zuzugreifen. Die Ergebnisse zeigen, dass *MapServer* grundsätzlich in der Lage ist, die Daten aus *Mapbox*

1 http://mapserver.org/
2 https://github.com/mapserver/mapcache
3 https://proj4.org
4 https://www.mapserver.org/mapfile

Vector Tiles zu lesen und anzuzeigen. Mit *MapManager*[1] als Tool zum Bearbeiten des *Mapfiles* wurde dies in ersten Tests beispielhaft umgesetzt (Abbildung 12). Jedoch bieten die beiden *GDAL*-Treiber aktuell noch keine *Merge* bzw. *Dissolve*-Option zum Zusammenführen kachelübergreifender Features. Darüber hinaus konnte mit den im *Mapfile* unterstützten Parametern[2] keine Verbindung zu den Optionen *CLIP* und *ZOOM_LEVEL_AUTO* des *MBTiles*-Treibers der *GDAL*-Bibliothek hergestellt werden. Damit werden dessen Standardeinstellungen übernommen. Dies bedeutet, dass immer alle *Buffer* abgeschnitten und unabhängig von der Zoomstufe in *MapServer* die Tiles der höchsten Zoomstufe aus der *MBTiles*-Datenbank geladen werden. Der Code des *Mapfiles*, mit welchem die Darstellung in Abbildung 12 erzeugt wurde, ist im GitHub-Repository zu dieser Arbeit[3] hinterlegt. Mit der Nutzung des *MVT*- oder *MBTiles*-Treibers der *GDAL*-Bibliothek als Eingangslayer stehen *MapServer* ausschließlich Vector Tiles im CRS EPSG 3857 zur Verfügung. Da *MapServer* jedoch durch die Verwendung der *Proj.4*-Bibliothek *on the fly* umprojizieren kann, wäre das kein wesentlicher Nachteil. Über den WMS-Endpoint könnten basierend auf einer Eingangsprojektion beliebige Augabeprojektionen bereitgestellt werden (OSGeoLive Contributors 2018).

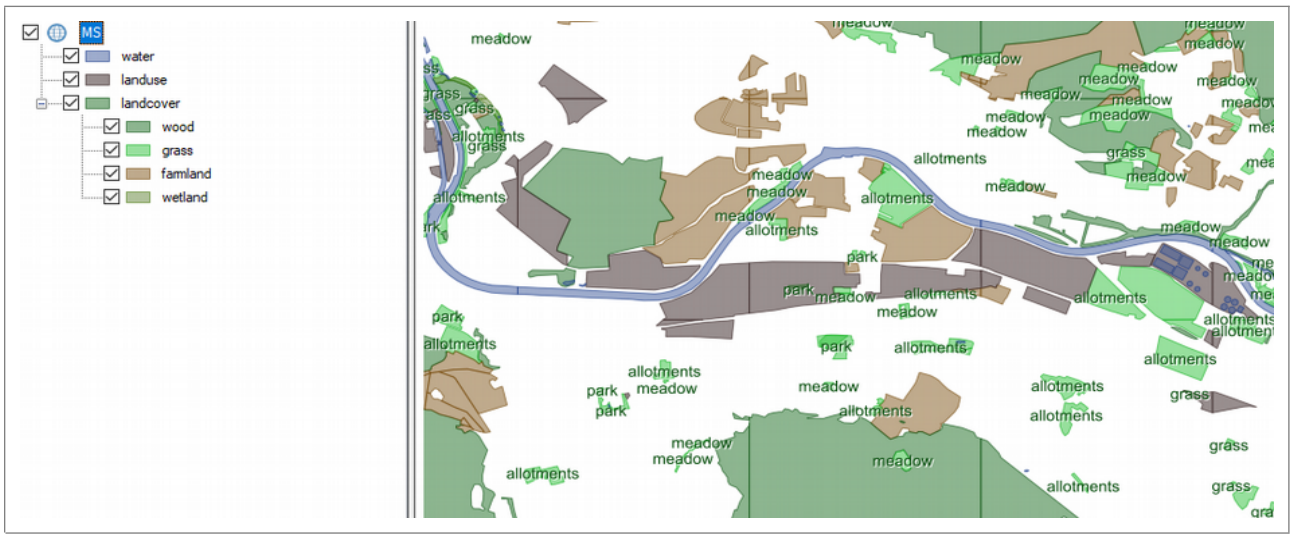


Abbildung 12 Vector Tiles als Datenquelle in MapServer
Datenquelle: Vector Tiles *Zurich*[4] von *OpenMapTiles.com*
erstellt mit *MapManager*

Das Erzeugen von Vector Tiles aus Vektordatenquellen ist mit *MapServer* ab Version 7.2 mit implementierter aktueller GDAL-Bibliothek ebenfalls möglich[5,6]. Darüber hinaus kann *MapServer* einen WMTS als Datenquelle für einen WMS nutzen[7,8] und somit als Proxy eingesetzt werden.

1 https://github.com/DMS-Aus/MapManager
2 https://www.mapserver.org/de/mapfile/
3 https://github.com/GjueAtGit/VTs_datasource_ogc/tree/master/mapserver
4 https://openmaptiles.com/downloads/europe/switzerland/zurich/
5 https://github.com/mapserver/mapserver/blob/branch-7-2/HISTORY.TXT
6 https://github.com/sdlime/mvt-demo
7 https://mapserver.org/mapcache/sources.html#wmts-sources
8 https://gis.stackexchange.com/questions/265679/mapserver-map-file-reading-wmts-and-serving-wms

5.7 TileServer GL

TileServer GL[1] ist ein im Jahr 2016 veröffentlichter *Open Source* Mapserver, welcher in *JavaScript* programmiert ist und Vector Tiles im *MVT*-Format in *MBTiles*-Datenbanken bereitstellt. Darüber hinaus kann *TileServer GL* mit der *Mapbox GL Native engine* (Kap. 5.9) serverseitig Raster Tiles rendern (Sloup & Přidal 2016). *TileServer GL* liegt aktuell in Version 2.3.1 vor und wurde seit der Veröffentlichung aktiv weiterentwickelt. Die nachfolgenden Angaben sind dem Vortrag *Hosting vector tile maps on your own server*[2] von Sloup and Přidal (2016) von der *FOSS4G* in Bonn und der *TileServer GL Documentation*[3] entnommen.

Die Konfiguration von *TileServer GL* erfolgt über *JSON*-Dateien. Standardmäßig werden von *TileServer GL* vorgenerierte Vector Tiles im *MBTiles*-Format (Kap. 4.6.2.1) und zusätzlich Stile, Schriften und Symbole genutzt und bereitgestellt. Darüber hinaus können auch dynamisch erzeugte Vector Tiles externer Server (z. B. *GeoServer*) eingebunden werden (Sloup & Přidal 2016; Min. 5:30 & 11:20). *TileServer GL* ist außerdem in der Lage, Vector Tiles mit Raster Tiles oder Vector Tiles aus anderen Quellen zu kombinieren und als Rasterbilder zu rendern. Die Tiles können über *TileJSON*, WMTS oder als *XYZ-Tiles* abgerufen werden.

Standardmäßig wird EPSG 3857 als CRS verwendet, es ist aber auch die Nutzung anderer CRS möglich. Eine Möglichkeit zur Umprojektion von Vector Tiles beim Rendern konnte nicht recherchiert werden, d. h die Vector Tiles müssen im CRS der Ausgabe generiert sein. *TileServer GL* ist ein reines *Rendering*-Tool ohne *Caching*-Mechanismus. Ein *Cachen* der generierten Raster Tiles ist jedoch z. B. mit *Mapproxy* (Kap. 5.8) möglich. Typischerweise läuft *TileServer GL* hinter einem Webserver wie *nginx* oder *apache*, mit welchem z. B. Restriktionen und Passwörter verwaltet werden können.

Das *Styling* erfolgt im *JSON*-Format entsprechend der *Mapbox Style Specification*[4]. Die Stile können in *Mapbox Studio*[5] entworfen werden. Diese Plattform ist jedoch nicht *Open Source* und erfordert den Upload der Daten auf den Server von *Mapbox* (vgl. Kap. 5.9). Es gibt jedoch *Open Source*-Editoren wie *Maputnik*[6] oder *style-editor*[7] mit vergleichbarer Funktionalität. Die Anzeige der entsprechend der Stildefinition gerenderten Vector und Raster Tiles ist über den Viewer von *TileServer GL* möglich. Dieser wird über den beim Starten des Servers angegebenen Port in einem Browser geladen. Daten aus *MBTiles*-Datenbanken werden im unteren Teil des Viewers angezeigt. Von diesen Tiles kann im Browser auch eine *GeoJSON*-Repräsentation der Tiles in Textform geladen werden. Bei Vector Tiles aus anderen Datenquellen ist dies nicht möglich.

5.8 Mapproxy

Mapproxy[8] ist ein Proxy für Geodaten, die er von vorhandenen Webdiensten zwischenspeichert, damit beschleunigt und darüber hinaus Werkzeuge zu deren Transformation und Optimierung bereitstellt (Tonnhofer 2017). *Mapproxy* liegt aktuell in Version 1.11.0 vor, wird aktiv federführend von der Firma *Omniscale*[9] und vielen Mitwirkenden als *Open Source*-Projekt entwickelt und unterliegt der *Apache Software License* Version 2.0 (Kap. 3.3).

1 http://tileserver.org
2 https://www.youtube.com/watch?v=rOg4VnSAnI4
3 https://tileserver.readthedocs.io/en/latest/
4 https://www.mapbox.com/mapbox-gl-js/style-spec/
5 https://www.mapbox.com/studio
6 https://maputnik.github.io
7 https://github.com/erikandre/mapbox-gl-style-editor
8 https://mapproxy.org
9 https://omniscale.de

Als Quellen kann *Mapproxy* u. a. WMS, WMTS, TMS, jeden *Tile Cache* und darüber hinaus *Mapserver*-(Kap. 5.6) und *Mapnik*-Konfigurationen (Kap. 5.10) direkt nutzen. Die unterschiedlichen Quellen können in einer URL unter Vereinheitlichung der Autorisierung, der Koordinatensysteme und mit Manipulation von Farben zu einem Dienst zusammengeführt werden. Die Konfiguration von *Mapproxy* erfolgt in *YAML*. Eine wesentliche Funktion ist die Umprojektion von Quelldaten. Bei dieser ist jedoch, wenn sie aus einer Rasterdatenquelle (z. B. WMTS oder XYZ) kommen, mit einer mehr oder weniger starken Verzerrung der Anzeige, insbesondere von Beschriftungen, zu rechnen (Abbildung 19). Über *Meta Tiles*[1] und *Meta Buffer*[2] kann auf der Seite von *Mapproxy* das Problem abgeschnittener und sich wiederholender Beschriftungen reduziert werden, indem das Programm bei einem Quell-WMS ein Bild pro Metatile anfordert und diese dann in einzelne Tiles zerschneidet (Tonnhofer 2017). Diese Funktion kommt jedoch beim geplanten Einsatzgebiet nicht zum Tragen, da als Datenquelle kein WMS dient sondern fertig gerenderte Raster Tiles angefordert werden.

Mit Blick auf die Fragestellung dieser Arbeit ist die Erstellung eines WMS mit *Mapproxy* basierend auf Daten eines WMTS von Bedeutung. Hierbei wird nach einer Anfrage an den WMS von *Mapproxy* aus dessen Tile-Cache, welcher mit den Raster Tiles des WMTS erzeugt wird, das in das Ziel-CRS projizierte und zusammengefügte Bild generiert. Mit dem Tool *mapproxy-seed*[3] können Kacheln zur Performancesteigerung vorab generiert werden. Hierbei ist es möglich, auch nur häufig benötigte Teilbereiche vorzugenerieren, fehlende Kacheln oder alte Kacheln neu zu erzeugen. Darüber hinaus besteht die Möglichkeit zur Bereinigung alter Kacheln.

5.9 Mapbox GL native

Mapbox GL native[4] ist eine Bibliothek zum Einbetten interaktiver und anpassbarer Karten in native Anwendungen. Sie ist in *C++* programmiert und unterliegt der *BSD-3 Lizenz* (Kap. 3.3). *Mapbox GL* verwendet *Stylesheets* entsprechend der *Mapbox Style Specification*[5] und rendert mit *OpenGL*[6] basierend auf diesen Stilen Vector Tiles nach der *Mapbox Vector Tile Spezification*[7] (Mapbox 2018). Im Gegensatz zu dem flexibleren Datenformat *Mapbox Vector Tiles* (Kap. 4.5.1.1) unterstützt *Mapbox GL* nur das CRS EPSG 3857[8]. So wie *Mapbox Vector Tiles* als Datenformat wird aktuell auch *Mapbox GL* (bzw. *Mapbox GL JS* in Webbrowsern) als Renderer häufig eingesetzt. Zu *Mapbox GL native* existieren SDKs für mehrere Plattformen, so dass die Bibliothek plattformübergreifend verwendet werden kann (Mapbox 2018).

Die *Mapbox Styles*[9] können in *Mapbox Studio*[10] entworfen werden. Jedoch muss dafür ein Account bei *Mapbox* vorliegen und die Quelldaten müssen auf den Server der Firma *Mapbox* hochgeladen werden. Die Publikation kann direkt über *Mapbox* erfolgen, ist ab 50.000 Webviews pro Monat jedoch kostenpflichtig. Es gibt alternativ auch *Open Source*-Editoren mit denen *Mapbox Styles* bearbeitet werden können (Kap. 5.7).

1 https://mapproxy.org/docs/1.11.0/labeling.html#meta-tiles
2 https://mapproxy.org/docs/1.11.0/labeling.html#meta-buffer
3 https://mapproxy.org/docs/nightly/seed.html
4 https://github.com/mapbox/mapbox-gl-native
5 https://www.mapbox.com/mapbox-gl-js/style-spec/
6 https://www.opengl.org
7 https://github.com/mapbox/vector-tile-spec
8 https://www.mapbox.com/help/define-projection/
9 https://www.mapbox.com/mapbox-gl-js/style-spec/
10 https://www.mapbox.com/mapbox-studio/

5.10 Mapnik

> *„Mapnik is an open source toolkit for developing mapping applications. At the core is a C++ shared library providing algorithms and patterns for spatial data access and visualization.“ (Mapnik Contributors 2018)*
>
> ***übersetzt:***
>
> *„Mapnik ist ein Open-Source-Toolkit für die Entwicklung von Mapping-Anwendungen. Kernstück ist eine C++ Shared Library, die Algorithmen und Muster für den Zugriff auf Geodaten und die Visualisierung bereitstellt.“*

Mapnik[1] liegt aktuell in Version 3.0.20 vor, wird seit 2005 aktiv entwickelt und unterliegt der *LGPL-2.1-Lizenz* (Kap. 3.3). Die Programmbibliothek ist in *C++* geschrieben, es sind aber auch Anbindungen für *node.js* (*Node Mapnik*[2]) und *Python*[3] (*Python Mapnik*[4]) verfügbar. Diese machen es möglich, verschiedene Vector Tile-Formate als Eingangsdaten in *Mapnik* zu verwenden. So weisen Migurski (2013) und Skowron (2017b; Min. 8:50) auf die Nutzung von *Mapbox Vector Tiles* und Gesteira (2017) auf die Einbindung von *GeoJSON* Vector Tiles aus einer *SQLite*-Datenbank in *Mapnik* hin. Mit der Klasse *VectorTile* in *Node Mapnik* wird der Zugriff auf Vector Tiles ermöglicht. In *Python* erfolgt der Zugriff auf eine Datenquelle über die Klasse *PythonDatasource*[5,6], wie es im *Wiki* des *GitHub Repository*[7] beispielhaft beschrieben wird. In *TileStache* (Kap. 5.1) wird diese Möglichkeit genutzt, um *Mapnik* über die *VecTiles Datasource Mapbox Vector Tiles* mit dem *VecTiles*-Package zur Verfügung zu stellen[8]. Migurski (2013) weist darauf hin, dass die *Python Datatsource* von *Mapnik* Eingangsdaten in der Projektion des finalen Renderings als WKB-Geometrien mit einfachen *dictionaries* mit Zeichenketten für Schlüssel erwartet.

Über die *Plugin Architecture*[9] verfügt *Mapnik* über Lesezugriff auf eine Vielzahl von Datenformaten. So können über die *GDAL*-Bibliothek viele Vektorformate gelesen werden. Ob der direkte Zugriff auf *Mapbox Vector Tiles* mit der aktuellen *Mapnik*-Version unter Einbindung der aktuellen Version der *GDAL*-Bibliothek (vgl. Kap. 5.11) möglich ist, konnte im Rahmen dieser Arbeit nicht ermittelt werden. Auch hier gelten die momentanen Einschränkungen zu den beiden Vector Tile-Treibern von *GDAL*, die bei *MapServer* (Kap. 5.6) dargestellt werden. Jedoch ermöglicht die *Mapnik Implementierung* der *Mapbox Vector Tile Spezifikation mapnik-vector-tile*[10] das Lesen von Vector Tiles im *MVT*-Format und damit ein Rendern von Raster Tiles aus diesen Daten. Ebenso ist diese Implementierung in der Lage, Geodaten aus unterschiedlichen Vektorquellformaten in Vector Tiles, die der *Mapbox Vector Tile Spezifikation* entsprechen, umzuwandeln. Als Ausgabeformate von *Mapnik* sind im Rahmen dieser Arbeit JPG und PNG von Bedeutung. Interessant ist, dass *Mapproxy*[11] (Kap. 5.8) *Mapnik* direkt, ohne einen WMS, aufrufen und damit die gerenderten Raster Tiles verwenden kann.

1 https://github.com/mapnik/mapnik
2 https://github.com/mapnik/node-mapnik
3 https://github.com/mapnik/mapnik/wiki/Python-Plugin
4 https://github.com/mapnik/python-mapnik
5 https://mapnik.org/docs/v2.2.0/api/python/mapnik.PythonDatasource-class.html
6 https://github.com/mapnik/python-mapnik/blob/master/mapnik/__init__.py
7 https://github.com/mapnik/mapnik/wiki/Python-Plugin#examples
8 http://tilestache.org/doc/TileStache.Goodies.VecTiles.mvt.html
9 https://github.com/mapnik/mapnik/wiki/pluginArchitecture
10 https://github.com/mapbox/mapnik-vector-tile
11 https://mapproxy.org/docs/nightly/sources.html#mapnik

5.11 Geospatial Data Abstraction Library (GDAL)

Die *Geospatial Data Abstraction Library* (GDAL) war ursprünglich eine Bibliothek zur Übersetzung von Rastergeodatenformaten. Die zugehörige *OGR Simple Features Library* (OGR) für Vektordaten wurde bereits in früheren Versionen innerhalb des *GDAL source tree* mitverwaltet und ist seit Version 2.0 enger integriert (OSGeo 2018). *GDAL* ist in *C++* geschrieben und unterliegt der *X/MIT-Lizenz* (Kap. 3.3).

Seit Version 2.3 von *GDAL* können mit dem *MVT*-Treiber[1] *Mapbox Vector Tiles* (Kap. 4.5.1.1) gelesen und geschrieben werden. Parallel wurde der *MBTiles*-Treiber[2] erweitert, so dass nun auch *MBTiles* Datenbanken mit Vector Tiles (Kap. 4.6.2.1) unterstützt werden. Vector Tiles können über den *MVT*-Treiber unkomprimiert, als *gzip*-komprimiert oder als Tileset gelesen und erstellt werden. Ein Schreibzugriff auf den *MVT*-Treiber erfordert, dass *GDAL* mit Unterstützung von *libsqlite3*[3] und *Geometry Engine Open Source (GEOS)*[4] erstellt wird. Für den Schreibzugriff des *MBTiles*-Treibers muss *GDAL* ebenfalls mit *GEOS* erstellt werden (OSGeo 2018). Standardmäßig nutzt der *MVT-Treiber* EPSG 3857 als CRS, es können jedoch auch andere CRS mit eigenen Tilesets festgelegt werden. Dagegen ist bei Verwendung des *MBTiles*-Treibers aufgrund der Spezifikation dieses Formates (Kap. 4.6.2.1) das CRS ausschließlich auf EPSG 3857 beschränkt.

Mit Stand vom 23.06.2018 liegt *GDAL* in Version 2.3.1 vor. In *QGIS* 2.18 und 3.2 ist GDAL Version 2.2.4 implementiert. *ArcGIS* 10.6 unterstützt *GDAL* Version 2.0.1. Somit ist in diesen beiden Clients die neue Funktionalität noch nicht anwendbar. Dagegen kann in *MapServer* Version 7.2.0 (Kap. 5.6) die aktuelle *GDAL*-Bibliothek eingebunden werden[5]. Die beiden beschriebenen Treiber von *GDAL* bieten *CLIP* als Option an, um den *Buffer* um Vector Tiles abzuschneiden (vgl. Abbildung 12). Eine *Merge*- bzw. *Dissolve*-Option zum Zusammenfügen von Features, welche Kachelgrenzen überschreiten, ist jedoch sowohl im *MBTiles*- als auch im *MVT*-Treiber in Version 2.3 noch nicht vorhanden.

5.12 Feature Manipulation Engine (FME)

Die *Feature Manipulation Engine* (FME)[6] ermöglicht u. a. die Transformation geografischer Daten unterschiedlicher Formate und Projektionen untereinander. Gesteira (2017) stellt in seinem Vortrag[7] einen Workflow vor, mit welchem unterschiedliche Eingangsdaten mit Hilfe von *FME* in Vector Tiles im *GeoJSON*-Format in einer *SQLite*-Datenbank gespeichert werden. Dabei werden eine Tabelle *map* mit dem Tileindex (*zoomlevel*, *tile_row*, *tile_column* und *tile_id*) und eine Tabelle *images* mit den Geometrien der Tiles erzeugt. Diese sind über das Feld *tile_id* untereinander verbunden. Im beschriebenen Beispiel werden die Vector Tiles anschließend mit *Mapnik* (Kap. 5.10) in Raster Tiles gerendert.

Für *FME* 2019 ist die Implementierung eines *Vector Tiles Writer* geplant[8]. Dieser ist mit dem Update der *GDAL*-Version in *FME* auf 2.3+ (Kap. 5.11) verfügbar und kann *Mapbox Vector Tiles* als einzelne Dateien und in *MBTiles*-Datenbanken ausgeben, *FME* 2018 kann dies nur mit Raster Tiles.

1 https://www.gdal.org/drv_mvt.html
2 https://www.gdal.org/frmt_mbtiles.html
3 https://packages.debian.org/sid/libsqlite3-0
4 https://trac.osgeo.org/geos/
5 https://www.gisinternals.com
6 https://www.safe.com
7 https://www.safe.com/presentation/creating-vector-tiles-for-background-maps-with-fme/
8 https://knowledge.safe.com/idea/47940/mapbox-vector-tiles-writer.html

5.13 QGIS

In den aktuellen *QGIS*-Versionen 3.2 und 2.18 ist das Darstellen von *Mapbox Vector Tiles* über das ***Vector Tiles Reader QGIS-Plugin***[1] möglich. Als Quelle dienen entweder ein Server, der über eine *TileJSON*-URL definiert ist, eine *MBTiles*-Datenbank oder ein lokales Verzeichnis, in welchem die Tiles in der Struktur `/{z}/{x}/{y}.pbf` abgelegt sind. Stildefinitionen im *Mapbox GL-Style* können über eine *JSON*-Datei verknüpft werden. Die Stile sind jedoch auch direkt in *QGIS*, wie bei anderen Vektorlayern üblich, anpassbar. Attributinformationen zu den Features sind ebenfalls abrufbar. Somit verdeutlicht diese Erweiterung die Möglichkeiten der clientseitigen Nutzung von Vector Tiles.

Mit der Option *Merge Tiles* werden überlappende Features eines Layers zu einem Feature zusammengeführt. Einerseits arbeitet diese Option noch nicht fehlerfrei, da vereinzelt Überlappungen vorhanden bleiben (Abbildung 14). Darüber hinaus werden auch überlappende Features innerhalb eines Tiles unabhängig von den Attributwerten zusammengeführt (vgl. Abbildung 13 ohne Option *Merge)*. Im Beispiel wird das kleinere Feature mit dem Wert *kindergarten* des Attributs *class* mit dem größeren, überlagernden Feature mit dem Wert *residential* zusammengeführt. Beim entstehenden neuen Feature wird dem Attribut *class fälschlich* der Wert *kindergarten* zugewiesen, obwohl die Umrisse *residential* entsprechen. Bei Eingangsdaten ohne überlappende Features würde dieses Problem nicht auftauchen, eine Option, *Merge* anhand von Attributwerten zu steuern, wäre hier jedoch von Vorteil. Aufgrund der noch nicht zuverlässigen *Merge*-Operation werden auch Beschriftungen von Features, die sich über mehrere Tiles erstrecken, wiederholt.

Aktuell unterstützt das *Vector Tiles Reader QGIS-Plugin* ausschließlich *Mapbox Vector Tiles* und aktualisiert nur eine Vector Tiles Quelle gleichzeitig. Das bedeutet, dass bei mehreren Vector Tile Quellen in einem *QGIS*-Projekt nur die zuletzt hinzugefügte Quelle aktualisiert wird.

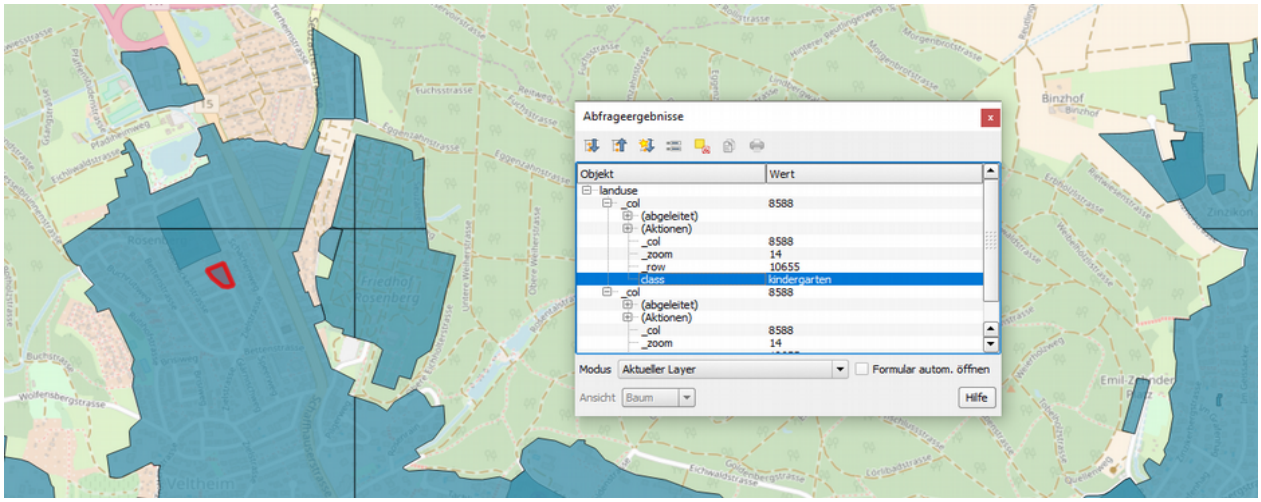


Abbildung 13 Überlagernde Features beim Layer *landcover* in Winterthur
Datengrundlage: Vector Tiles *Zurich*[2] von *OpenMapTiles.com*; Hintergrundkarte © *OpenStreetMap Mitwirkende CC-BY-SA;* Darstellung in *QGIS* 2.18 mit *Vector Tiles Reader Plugin*, Option *Clip*

1 https://github.com/geometalab/Vector-Tiles-Reader-QGIS-Plugin
2 https://openmaptiles.com/downloads/europe/switzerland/zurich/

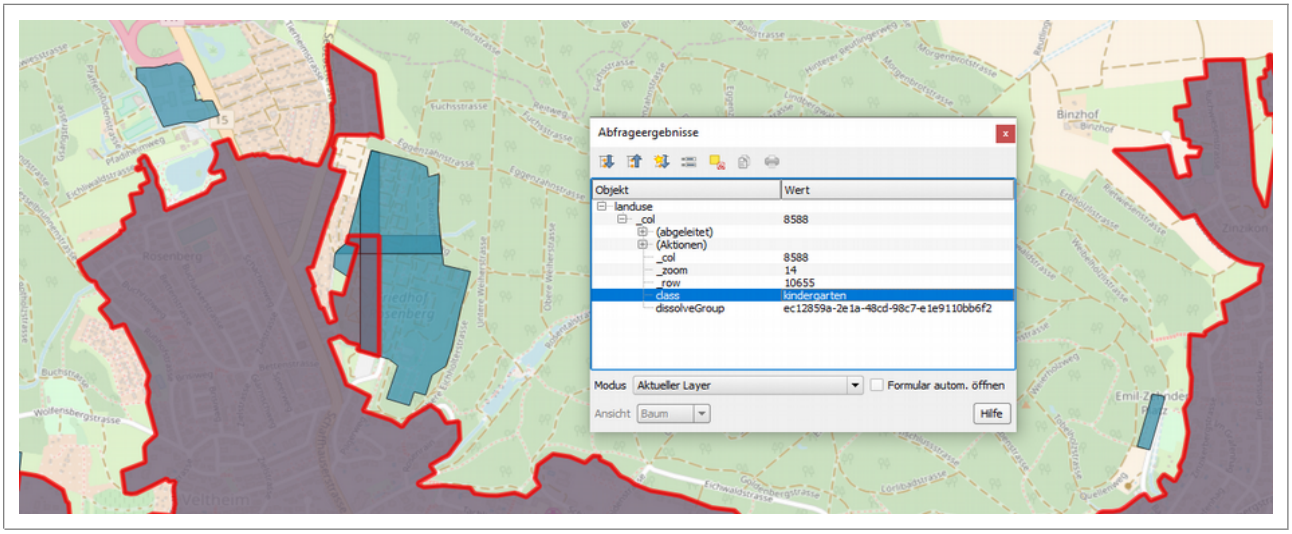


Abbildung 14 Erkennbare Kachelgrenzen in Winterthur beim Layer *landcover* durch Überlagerung von Features an Tile-Grenzen und fehlerhafte Attributierung nach *Merge*
Datengrundlage: Vector Tiles *Zurich* von *OpenMapTiles.com*; Hintergrundkarte © *OpenStreetMap Mitwirkende CC-BY-SA;* Darstellung in *QGIS* 2.18 mit *Vector Tiles Reader Plugin*, Optionen *Merge* und *Clip*

Mit dem ***Vector Tiles Client-Plugin*** für *QGIS* 2.18 wird bei Cavazzi (2018; Kap. 9) eine weitere *QGIS*-Erweiterung für Vector Tiles beschrieben. Diese wurde u. a. vom *Center for Spatial Information Science and Systems* (CSISS)[1] der *George Mason University* (GMU) entwickelt. Das *Vector Tiles Client-Plugin* kann Vector Tiles in den Formaten *GeoJSON*, *Mapbox Vector Tiles* und *GML* aus den Tiling Schemata *XYZ*, *TMS* und von erweiterten WFS einlesen. Derzeit bietet jedoch der WFS-Standard keine Möglichkeit, bestimmte Vector Tiles anzufordern, wie es z. B die bei Cavazzi (2018 p. 108) vorgeschlagenen zusätzlichen Parameter *tileId* und *tileScheme* für eine *GetFeature*-Operation erlauben würden.

Unterstützt werden grundsätzlich alle EPSG-Koordinatensysteme. Beim Wiedergeben der Tiles werden überlappende Teile von Features per *Dissolve* aufgelöst und somit die Kachelgrenzen eliminiert. Mit diesem Plugin könnten Vector Tiles mit der vollen Funktionalität von *QGIS* dargestellt werden und ein vollständiger Zugriff auf deren Attribute wird ermöglicht. Über die Angaben bei Cavazzi (2018) hinaus gehende Informationen zum *Vector Tiles Client-Plugin*, Hinweise zu einer möglichen Veröffentlichung sowie das Plugin selbst waren zum Zeitpunkt dieser Arbeit nicht verfügbar.

5.14 ArcGIS Pro

ArcGIS Pro unterstützt seit der Version 1.2 ausschließlich proprietäre Packages zum Speichern und Lesen von Vector Tiles (Williams & Punt 2017). Die Grundlagen zu diesem Speicherformat sind in Kap. 4.6.3 beschrieben. In *ArcMap* 10.6 ist keine Möglichkeit zur Einbindung von Vector Tiles implementiert. Auch die Anzeige der proprietären *Vector Tile Packages* wird nach Aussage von Williams und Punt (2017 p. 38) in *ArcMap* vermutlich auch zukünftig nicht unterstützt.

Zur Erstellung und Nutzung von Vector Tiles stehen in *ArcGIS Pro* 2.2 drei Tools in der Toolbox *Package* zur Verfügung:

- *Create Vector Tile Index*: Erstellen eines Gitter Indexes mit *Create Vector Tile Index*
- *Create Vector Tile Package*: *Erstellen* eines Vector Tile Packages (*.vtpk)
- *Extract Package*: Entpacken eines Packages in ein lokales Verzeichnis

1 http://csiss.gmu.edu/index.htm

Mit ***Create Vector Tile Index*** wird ein Gitter anhand einer frei wählbaren, maximal möglichen Anzahl von Knoten pro Kachel erzeugt. Anhand dieser Anzahl wird die Karte in unterschiedlich große Kacheln aufgeteilt. Einen daraus resultierenden *Quadtree* zeigt Abbildung 15. Beim Erstellen eines ***Vector Tile Packages*** bestehen die Optionen, einen vorher erstellten *Vector Tile Index* als Grundlage zu verwenden oder einen *flachen* Tile Index zu verwenden (Kap. 4.2.3 und 4.3.1). Wird ein vorher errechneter Index verwendet, ist es wesentlich, dass dieser mit genau den Inhalten erstellt wurde, die im zu erzeugenden *Package* in den Vector Tiles enthalten sein werden. Andernfalls resultieren unter Umständen sehr unterschiedliche Detailgrade pro Kachel (Abbildung 15), da *ArcGIS Pro* basierend auf der beim Erstellen des *Vector Tile Index* errechneten Anzahl der Vertices die Generalisierung pro Zoomstufe vornimmt.

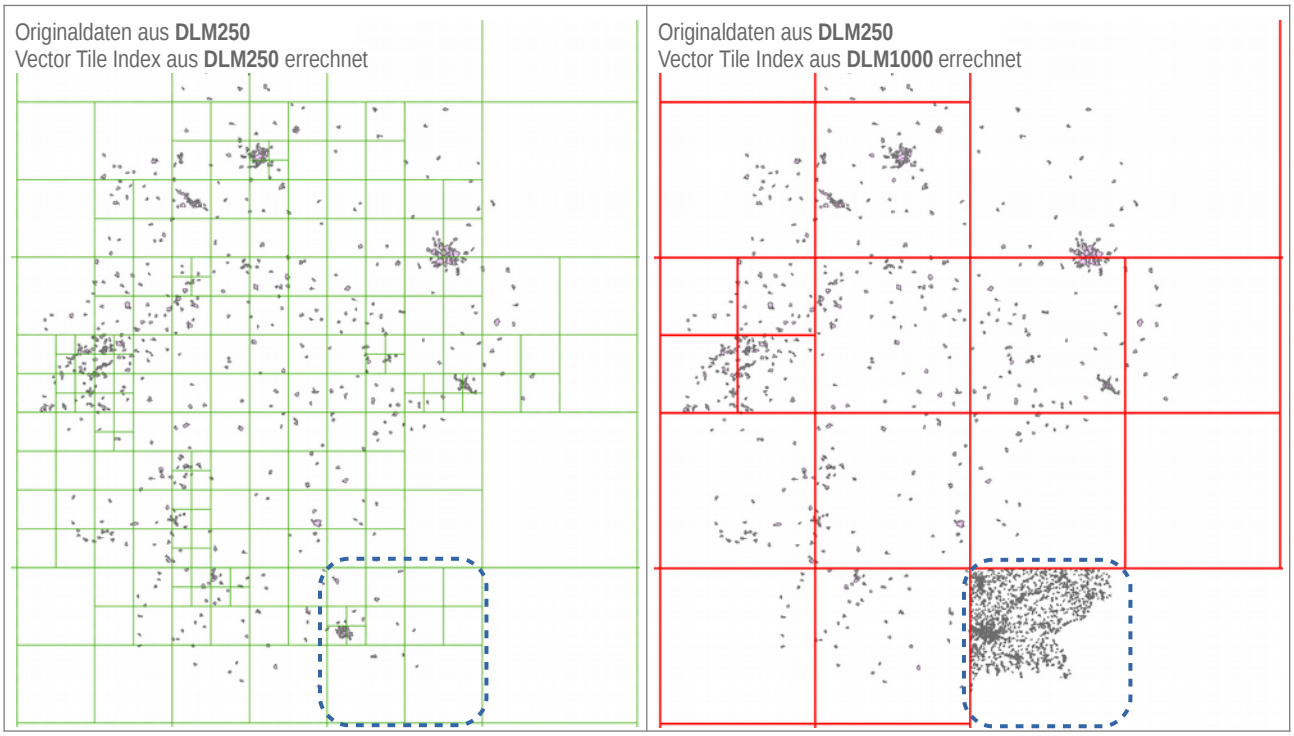


Abbildung 15 Resultat der Verwendung eines falschen Vector Tile Index (rechts) beim Erstellen von *Vector Tile Packages* mit *ArcGIS Pro 2.1.2*
Kartengrundlage: *Siedlungsgrenzen* aus *DLM250* (© GeoBasis-DE / BKG 2018)

Die Tiles werden im *Mapbox Vector Tile*-Format (Kap. 4.5.1.1) erzeugt und im Package in *bundles* gespeichert. Nach dem Extrahieren sind sie in einer Ordnerstruktur `/tile/{z}/{x}/{y}.pbf` abgelegt. Die Stildefinitionen werden aus der Ausgangskarte heraus übernommen und im Package im *JSON*-Format mitgeführt. In Version 2.2 von *ArcGIS Pro* ist bei Einbindung eines *Vector Tile Packages* noch keine Änderung der Stile möglich.

6 Rechercheergebnisse

6.1 Ergebnisse zu Datenformaten, Tiling-Schemata und Ablagestrukturen

Die Ergebnisse der Recherche nach Datenformaten, Tiling-Schemata und Ablagestrukturen von Vector Tiles aus Kap. 4 werden in Abbildung 16 grafisch dargestellt. Tabelle 2 listet die in Kap. 4.5 recherchierten 2D Vector Tile-Formate überblicksmäßig auf.

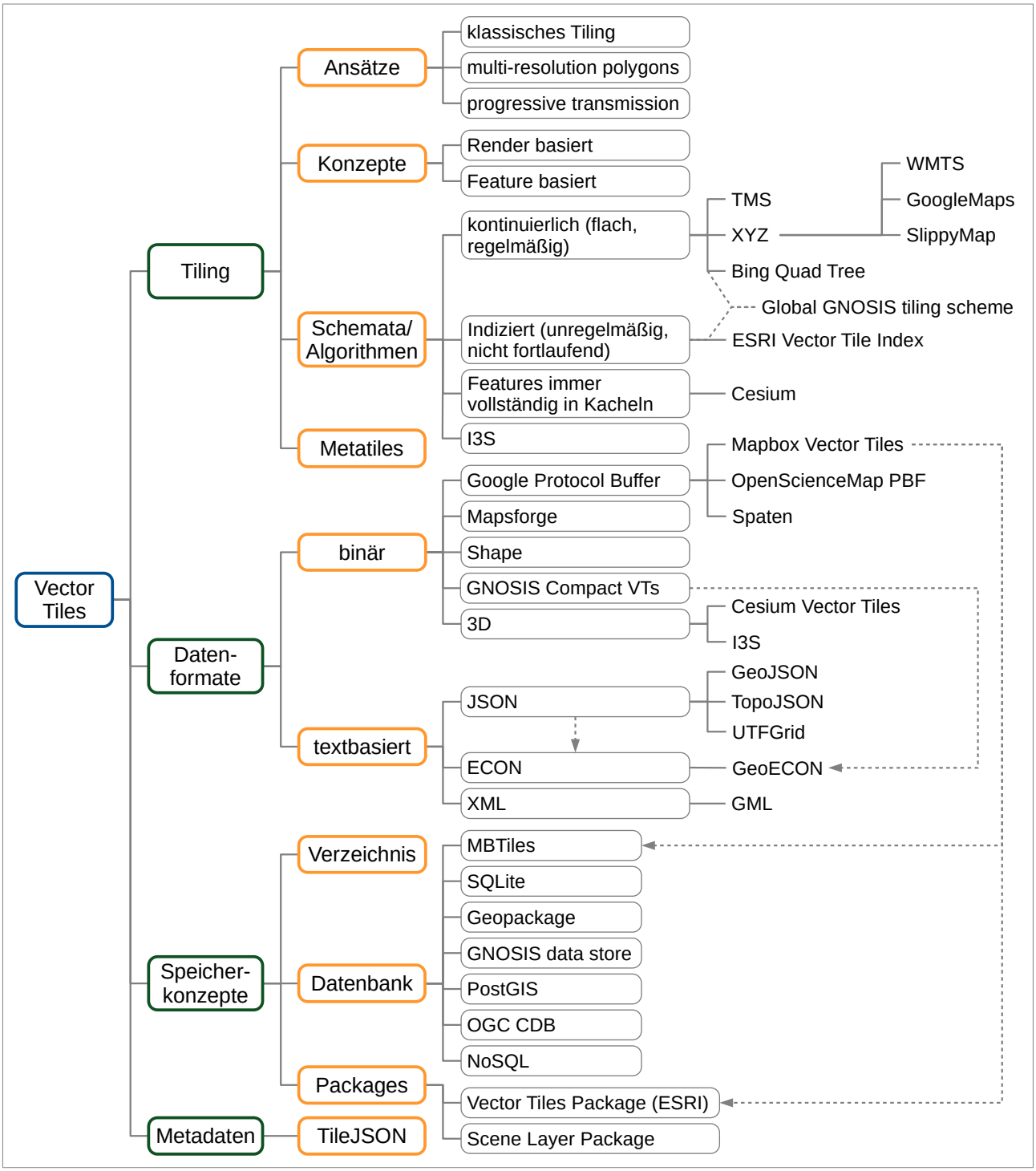


Abbildung 16 Schematische Darstellung von Vector Tile Datenformaten, Tiling-Schemata und Ablagestrukturen

Tabelle 2 Übersichtstabelle der recherchierten 2D-Vector Tile-Formate

Format	Erweite-rungen	Target- Format	CRS	Koordinaten	feature-basiert	render-basiert	kompakt
MapBox Vector Tiles	*.mvt, *.pbf	Protocol Buffers	v.a. EPSG 3587, aber alle möglich	Bildschirm		x	x
OSciM-PBF	*.vtm	Protocol Buffers	n.n.	Bildschirm		x	x
Spaten	*.spaten	Protocol Buffers	alle	Bildschirm (?)		x	x
GeoJSON	*.geojson; *.json	JSON	laut Spezifikation nur EPSG 4326, aber alle möglich	Realwelt	x		
TopoJSON	*.topojson; *.json	JSON	alle	Realwelt	x		(x)
UTF-Grid		JSON	n.n.	Bildschirm		(x)	x
GeoECON	*.econ	ECON -> JSON	EPSG 4326	Realwelt	x		
GNOSIS compact vector tiles	*.gmt	binär	EPSG 4326	16-Bit-Integer	x	x	(x)
GML	*.gml	XML	alle	Realwelt	x		
Mapsforge Binary Map File		binär	alle	Mikrograd		x	x
Shape	*.shp (*.dbf, ...)	-	alle	Realwelt	x		

Von den recherchierten Datenformaten besitzen aktuell nur *Mapbox Vector Tiles* (Kap. 4.5.1.1) und *GeoJSON* (Kap. 4.5.2.1) eine Bedeutung mit Blick auf die Fragestellung dieser Arbeit. Alle anderen Formate sind zu wenig verbreitet bzw. teilweise nur auf ein Softwareprodukt (z. B. *OSciM-PBF*) oder eine Funktion, die für die Zielstellung dieser Arbeit nicht relevant ist (z. B. *UTF-Grid*), beschränkt. *TopoJSON* (Kap. 4.5.2.2) und *GNOSIS Compact Vector Tiles* (Kap. 4.5.3) stellen, mit Blick auf die Beibehaltung der Topologie und einem guten Kompromiss von Genauigkeit der Koordinaten und Kompaktheit des Formates, interessante Ansätze dar, sind jedoch bisher kaum oder wie das Beispiel in Kap. 5.5 zeigt fehlerhaft implementiert. Das *Shape*- und das *GML*-Format sind bei herkömmlichen Geodaten weit verbreitet und theoretisch auch für die Speicherung von Vector Tiles geeignet, jedoch im Vergleich zu den anderen verfügbaren Formaten zu speicher- und/oder in der Verarbeitung zu rechenintensiv.

Allen Vector Tile-Formaten ist gemeinsam, dass bereits bei deren Erzeugung festgelegt wird, welche Layer grundsätzlich und in welchem Zoomlevel enthalten sind (Skowron 2017b). Demnach ist bereits bei der Generierung die Abwägung zwischen hoher Performance (nur ausgewählte Layer pro Tile bzw. Zoomstufe und damit geringere Datenmengen) und hoher Flexibilität bei der Nutzung der Vector Tiles (höhere Datenmengen) zu treffen. Dieselben Überlegungen sind bei der Auswahl der in die Vector Tiles übernommenen Attribute vorzunehmen.

Die Entwicklung neuer Spezifikationen kurz vor und im Zeitraum der Bearbeitung dieser Arbeit, z. B. Dateiformat *Spaten* (Kap. 4.5.1.4), die *Vector Tiles Extension*[1] für *GeoPackage* (Kap. 4.6.2.2) und die Implementierung von *Mapbox Vector Tiles* in die *GDAL*-Bibliothek (Kap. 5.11) zeigen, dass aktuell beim Thema Vector Tiles viel Entwicklungsarbeit stattfindet und in naher Zukunft mit weiteren Verbesserungen und Implementierungen zu rechnen ist. Mit Blick auf die weitere Verwendung sind derzeit *Mapbox Vector Tiles*, entweder in einer Ordnerstruktur oder gespeichert in *MBTiles*-Datenbanken (Kap. 4.6.2.1), das Datenformat für renderbasierte Lösungen und das am häufigsten unterstützte Vector Tiles Format. Für die Weitergabe von Daten mit unveränderter Geometrie in den höchsten Zoomstufen bieten sich derzeit Vector Tiles im

1 https://github.com/jyutzler/geopackage-vector-tiles/blob/master/spec/1_vector_tiles.adoc

GeoJSON-Format an. Mit Blick auf die Zielstellung dieser Arbeit konnte ein mögliches Szenario unter Verwendung von *GeoJSON*-Vector Tiles recherchiert werden (vgl. Kap. 7.1). Jedoch wäre hier keine direkte Nutzung als Datenquelle für einen WMS möglich, sondern ebenfalls der „Umweg“ über einen WMTS erforderlich, so dass keine Vorteile, sondern eher Nachteile gegenüber den kleineren und performanteren *Mapbox Vector Tiles* bestehen würden. Aus diesem Grund wird dieser Lösungsansatz hier nicht weiterverfolgt.

Als *Caching*-Mechanismen kommen mit Blick auf die recherchierte Software *MBTile*-Datenbanken und die Ablage von Einzeldateien in Ordnerstrukturen in Frage. Datenbanken ermöglichen im Vergleich zu Ordnerstrukturen eine effizientere Attributverwaltung und Indizierung der Tiles, erhöhen jedoch auch den Aufwand beim Zugriff auf diese, da Extraktionsmechanismen erforderlich sind (Cavazzi 2018). Sofern die eingesetzte Software nur einen der beiden Ablagemechanismen unterstützt, ist dies entscheidend für dessen Auswahl. Anderenfalls überwiegen vermutlich mit Blick auf den Einsatzzweck der Vector Tiles als Datengrundlage für Darstellungsdienste die Vorteile einer Ablage in einer Ordnerstruktur, da der schnelle, direkte Zugriff auf die Tiles wesentlich ist.

Bei den recherchierten Tiling-Schemata dominiert bei Anwendungen von Vector Tiles das *Google Tiling Scheme* (*XYZ*, Kap. 4.3.2.4) gefolgt von *TMS* (Kap. 4.3.2.2). Auch hier ist, wie schon bei den *Caching*-Mechanismen, die verwendbare Software entscheidend für das letztendlich verwendete Tiling-Schema. Bei der Verwendung von *XYZ* als Tiling-Schema beschränkt sich die Projektion der Daten in den Vector Tiles jedoch auf EPSG 3857. Das Tiling entsprechend WMTS-Standard (Kap. 4.3.2.3) hat zwar mit *XYZ* den Ursprung in der nordwestlichen Ecke des Kachelkoordinatensystems gemeinsam, ist jedoch flexibler in der Anwendung (z. B. beliebige CRS) und darüber hinaus OGC-Standard.

6.2 Ergebnisse der Softwarerecherche

Eine komplette Übersicht der im Zusammenhang dieser Arbeit recherchierten Softwarelösungen ist der Arbeitstabelle[1] im GitHub-Repository zu dieser Masterthesis zu entnehmen, von denen eine Auswahl in Kap. 5 einer detaillierteren Betrachtung unterzogen wurde. Eine zusammenfassende Übersicht dieser Programme gibt Tabelle 3.

Zum Erstellen von Vector Tiles in nichtproprietären Formaten existiert eine Vielzahl von Lösungen. In dieser Arbeit wurden *Tilestache*, *t-rex* , die Funktion *ST_AsMVT* von *PostGIS*, *tessera Tileserver*, *GeoServer*, *MapServer* und *FME* detaillierter untersucht. Bis auf *FME* können alle diese Programme *Mapbox Vector Tiles* (Kap. 4.5.1.1) erzeugen und in einer Verzeichnisstruktur oder *MBTiles*-Datenbank (Kap. 4.6.2.1) ablegen. Neben *FME* unterstützen auch *GeoServer* und *TileStache* Vector Tiles im *GeoJSON*-Format. Von den genannten Programmen können *TileStache*, *t-rex*, *GeoServer* und *FME* Vector Tiles mit beliebigen Projektionen speichern. Bei *tessera* und *ST_AsMVT* ist das CRS auf EPSG 3857 beschränkt. Bei *MapServer* wurde das Lesen von Vector Tiles über die *GDAL*-Bibliothek genauer betrachtet. Dabei wurden die beschriebenen Einschränkungen festgestellt, die einen produktiven Einsatz aktuell verhindern (Kap. 5.11). Ob das Erzeugen von Vector Tiles über die beiden Treiber von *GDAL* derzeit schon besser funktioniert als das Lesen wurde in dieser Arbeit aufgrund ausreichend vorhandener Alternativen nicht untersucht.

Die mit den genannten Programmen erzeugten Vector Tiles können zwar von einer Vielzahl von Webclients gelesen werden, aber eine direkte Verwendung als Datenquelle von Programmen, welche die *OGC*-Dienste WMS und WMTS bereitstellen können, funktioniert bislang nicht oder nur eingeschränkt. Grundsätzlich in die richtige Richtung geht die Erweiterung der *GDAL*-Bibliothek in der aktuellen Version um den *MVT*- und

1 https://github.com/GjueAtGit/VTs_datasource_ogc/tree/master/src

den *MBTiles*-Treiber. Bei beiden Treibern fehlt jedoch eine Option zum Zusammenfügen von Features, die über Kachelgrenzen hinweg reichen (*Merge* bzw. *Dissolve*). Bei der Symbolisierung würde das bedeuten, dass Polygone nur vollfarbig, ohne Transparenz und ohne Umrandung bzw. mit Umrandung in identischer Farbe wie die Füllfarbe dargestellt werden können. Andernfalls würden die Kachelgrenzen als horizontale und vertikale Unterbrechungen sichtbar (vgl. Abbildung 12). Beschriftungen würden sich dabei bei kachelübergreifenden Features in jeder Kachel wiederholen (Abbildung 17). In *MapServer*, mit welchem Versuche zum Lesen von Vector Tiles über die *GDAL*-Bibliothek unternommen wurden, fehlt bisher darüber hinaus die Möglichkeit, die Optionen anzusteuern, welche die beiden Treiber zur Auswahl des Zoomfaktors und zum *Clippen* der durch *Buffer* überlappenden Tiles bieten.

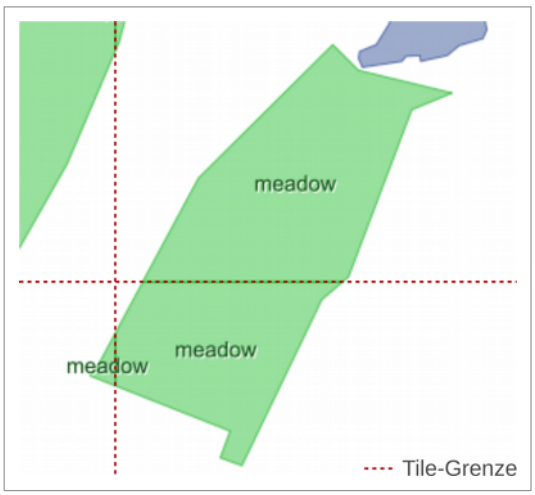


Abbildung 17 Wiederholte Beschriftung für ein Feature an Kachelgrenzen
Datengrundlage: Vector Tiles *Zurich* von *OpenMapTiles.com*

FME bietet nach Gesteira (2017) die Möglichkeit, *GeoJSON*-Vector Tiles in einer *SQLite*-Datenbank zu speichern. Im dort beschriebenen Beispiel werden diese im nächsten Schritt mit *Mapnik* in Raster Tiles gerendert. Einerseits konnte dieser Workflow im Rahmen dieser Arbeit nicht nachgestellt werden. Andererseits fallen die Vorteile des *GeoJSON*-Formates gegenüber *MVT* weg, wenn die Vector Tiles nicht direkt als Datenquelle für einen WMS genutzt werden können. Darüber hinaus müsste *FME* für eine Produktionsumgebung als Serverversion zur Verfügung stehen und wäre als einziges der hier vorgestellten Programme zum Erzeugen von Vector Tiles kostenpflichtig. Aus diesen Gründen und weil auch wieder der Umweg über Raster Tiles erforderlich ist, wird dieser Ansatz hier nicht weiter verfolgt.

TileStache ermöglicht die Generierung und Bereitstellung von Vector Tiles in unterschiedlichen Formaten und Projektionen. Interessant ist vor allem die direkte Verknüpfung zu *Mapnik* über die *VecTiles Datasource*. Diese ist jedoch auf das CRS EPSG 3857 beschränkt[1], so dass bei der Weiterverwendung der daraus gerenderten Raster Tiles die beschriebenen Qualitätsverluste (Abbildung 19) bei notwendigen Umprojektionen auftreten.

Um Raster Tiles in einer anderen als der *Sphärischen Mercator Projektion* (EPSG 3857) mit *Mapnik* zu rendern würde sich die Nutzung der *PythonDatasource* anbieten. Diese erfordert jedoch die Programmierung einer eigenen Schnittstelle, um *Mapnik* mit Vector Tiles zu versorgen, was den Rahmen dieser Arbeit übersteigt. Darüber hinaus ist *Mapnik* nicht in der Lage Daten umzuprojizieren, so dass für jede gewünschte Ausgabeprojektion ein Satz Vector Tiles in der entsprechenden Projektion als Eingangsdaten vorgehalten werden müsste. Im Gegensatz zu *Mapnik* kann *TileServer GL* laut Beschreibung Daten umprojizieren, so dass dieser sich unter Nutzung nur eines Eingangsdatensatzes anbietet, um Raster Tiles in unterschiedlicher

1 http://tilestache.org/doc/TileStache.Goodies.VecTiles.client.html#Datasource

Projektion zu rendern. Neben *TileStache* bietet sich hier *t-rex* zum Generieren der Vector Tiles an. Beide können Tiles im *MVT*-Format erzeugen, welches *TileServer GL* lesen kann. Darüber hinaus ist er in der Lage, diese entsprechenden Clients, aus *MBTiles*-Datenbanken heraus, zur direkten Nutzung bereitzustellen. Die mit *TileServer GL* oder *Mapnik* gerenderten Raster Tiles können mit Hilfe von *Mapproxy* als Datenquelle für einen WMS genutzt bzw. direkt als WMTS bereitgestellt werden.

Theoretisch ist es möglich, die *PostGIS-Funktion* von *ST_AsMVT* zum Erstellen von Vector Tiles zu nutzen. Diese erfordert jedoch Ausgangsdaten, die in einer *PostGIS*-Datenbank vorliegen. Darüber hinaus generiert die Funktion jeweils ein Tile, d. h. zum Vorprozessieren von Vector Tiles eines kompletten Datensatzes in eine Verzeichnisstruktur oder eine *MBTiles*-Datenbank müsste eine Routine (vgl. Allegri (2017)) entwickelt werden.

Die implementierten Algorithmen zum Generalisieren von Geometrien beim Erstellen der Vector Tiles konnten nicht in allen Fällen ermittelt werden. Die *PostGIS-Funktion ST_AsMVT* kann in Kombination mit den unterschiedlichen in *PostGIS* implementierten Generalisierungsfunktionen verwendet werden. *GeoServer* nutzt den *Douglas-Peucker* Algorithmus. Mit *t-rex* können nur Daten aus *PostGIS*-Datenbanken generalisiert werden, wobei für Linien ebenfalls der *Douglas-Peucker* Algorithmus und für Polygone *SnapToGrid* eingesetzt wird. Ob und welchen Generalisierungsmechanismus *TileStache* verwendet, konnte nicht abschließend festgestellt werden. Der Effekt, den eine ausschließliche Verschiebung der Realwelt-koordinaten auf Bildschirmkoordinaten nach sich zieht, wird in Kap. 7.3.1 beschrieben.

Die Recherche nach der Einbindung von Vector Tiles in Desktop-GIS Clients bestätigte die Vermutung, dass entweder keine oder bisher nur eine sehr eingeschränkte Unterstützung von Vector Tiles vorhanden ist. So bietet *ArcMap* 10.6 keine Unterstützung von Vector Tiles und eine Implementierung ist nicht geplant. *ArcGIS Pro* 2.2 unterstützt bisher ausschließlich die proprietären *Vector Tile Packages*, die wiederum in keiner anderen Software Anwendung fanden. *QGIS* 2.18 und 3.2 können über das *Vector Tiles Reader QGIS-Plugin* ausschließlich *Mapbox Vector Tiles* lesen. Die Funktion dieses Plugins ist jedoch noch eingeschränkt und die Darstellung mit den verwendeten Testdaten nicht fehlerfrei.

Aus den Ergebnissen der Softwarerecherche in Verbindung mit den Erkenntnissen zu Datenformaten wurden mögliche Szenarien für eine Nutzung von Vector Tiles als Datenquelle die OGC-Dienste WMS und WMTS abgeleitet (Kap. 7.1).

Tabelle 3 Zusammenfassende Darstellung der in Kap. 5 recherchierten Software

Software	Version	Alter	Vector Tiles erstellen			Caching/ Ablage	Vector Tile-Server	Vector Tiles einlesen	server-seitiges Rendern		OGC-Dienste kaska-dieren	Ausgabe WMS/ WMTS
		Stand 21.8.2018	Format	Schema	CRS (EPSG)				Bilder	CRS (EPSG)		
TileStache	1.51.10	21 Tage	MVT, GeoJSON, TopoJSON	XYZ	3857, 4326 und eigene	Verzeichnis, MBTiles, ...	x		in Mapnik			
t-rex	0.9.0	27 Tage	MVT	XYZ und TMS	3857, 4326, eigene	Verzeichnis, MBTiles, ...	x					
PostGIS ST_AsMVT	2.4.4	5 Monate	MVT	einzelne Tiles	3857 (?)							
tessera und tilelive	'0.13.0 6.0.0	2 Monate 8 Monate	MVT	XYZ TMS (?)	3857 (?)	Verzeichnis, MBTiles, ...	x		tilelive mapnik	3857 (?)		
GeoServer	2.13.2 1.12.5	28 Tage 0 Tage	MVT, GeoJSON, TopoJSON	TMS	beliebig	Verzeichnis, MBTiles, ...	x		x	beliebig	x	x
MapServer	7.2.0	28 Tage	(MVT)	XYZ	3857	Verzeichnis, MBTiles	?	(MVT)	x	beliebig	x	x
TileServer GL	2.3.1	8 Monate				kein Cache	x	MVT	x	beliebig		nur WMTS
Mapproxy	1.11.0	9 Monate									x	x
Mapbox GL native		5 Tage							x	3857 (?)		
Mapnik	3.0.20	4 Monate						TileStache, (GeoJSON)	x	beliebig		
GDAL	2.3.1	2 Monate	MVT		(3857)	Verzeichnis, MBTiles		MVT				
FME	2018	7 Monate	GeoJSON	?	beliebig	SQLite						
QGIS	2.18.23 3.2.2	4 Tage						Erweiterung				
ArcGIS Pro	2.2	2 Monate	MVT			Vector Tile Package		proprietär				

7 Szenarien

7.1 Abgeleitete Szenarien

Aus den Ergebnissen der Recherche zu Vector Tile Formaten und geeigneter Software (Kap. 6) ergeben sich unterschiedliche Szenarien zur Nutzung von Vector Tiles als Datenquelle für die OGC Darstellungsdienste WMS und WMTS. Die Übersicht in Abbildung 18 stellt die aus den gesammelten Erkenntnissen abgeleiteten theoretisch möglichen Kombinationen grafisch dar, wobei einige Einschränkungen bestehen.

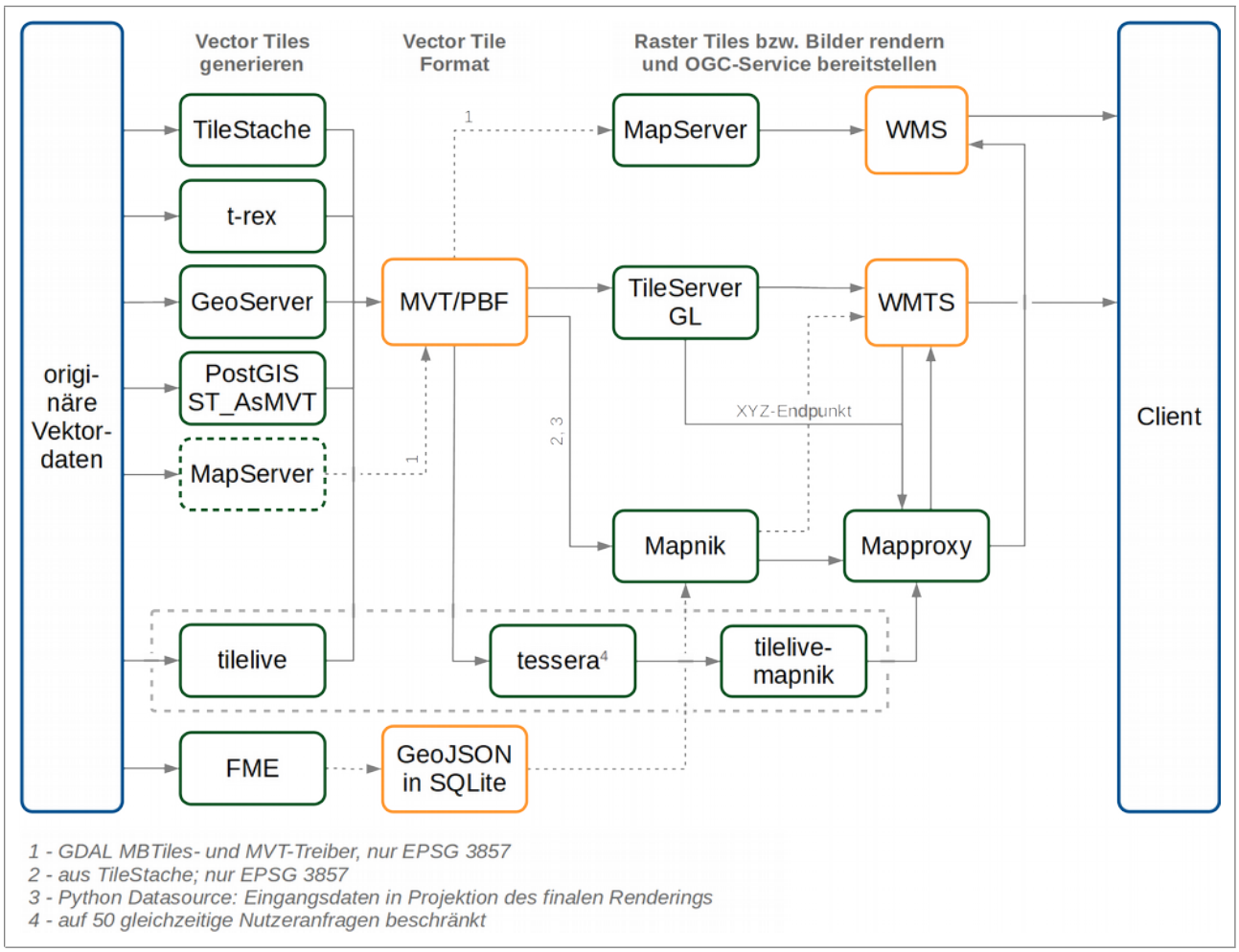


Abbildung 18 Entwurf möglicher Szenarien für die Nutzung von Vector Tiles als Datenquelle für WMS und WMTS

Aus diesen Kombinationen wurden vier im Kontext dieser Arbeit erfolgversprechend erscheinende Szenarien abgeleitet und deren Eignung nachfolgend detaillierter analysiert.

a) t-rex | TileStache | GeoServer → TileServer GL → **WMTS** → Mapproxy → **WMS**
b) TileStache → *nur EPSG 3857* → Mapnik → Mapproxy → **WMS**
c) t-rex | TileStache | GeoServer → MapServer (via GDAL) → **WMS**
d) tilelive → tessera → tilelive mapnik → Mapproxy → **WMS**

Bisher ist nur mit Szenario ***c)*** bei der Verwendung der *GDAL*-Bibliothek (Version ≥ 2.3.) in *MapServer* eine direkte Nutzung von Vector Tiles als Datenquelle für einen WMS umsetzbar. Die Einschränkungen durch eine fehlende *Merge*- bzw. *Dissolve*-Option beim *MVT*- und *MBTiles*-Treiber der *GDAL*-Bibliothek sowie

die fehlende Möglichkeit deren Optionen zum *Clippen* und zur Festlegung der Zoomstufe in *MapServer* zu nutzen, machen einen produktiven Einsatz derzeit jedoch nicht möglich.

Alle anderen abgeleiteten Kombinationen erfordern den Umweg über Raster Tiles bzw. das serverseitige Rendern mit *Mapnik* in nur einer Projektion, die dann über *Mapproxy* als Datenquelle für einen WMS genutzt werden können. Hierbei ist das Problem, dass ein großer Vorteil eines WMS verloren geht. Dieser ist, je nach Definition, in der Lage Rasterbilder beliebiger Projektion und vom Nutzer ausgewählter Stile und Layerkombinationen *on the fly* aus Vektordaten zu rendern. Werden dagegen vorgerenderte Raster Tiles über *Mapproxy* für einen WMS umprojiziert, kommt es zu mehr oder weniger starken Verzerrungen bei Beschriftungen und unscharfer Darstellung der Geometrien (Abbildung 19). Dies kann dadurch umgangen werden, dass für jede gewünschte Zielprojektion und jeden gewünschten Stil die entsprechenden Rasterkacheln (vor)gerendert werden. Soll in diesem Fall für jede gewünschte Kombination ein *Cache* aufgebaut werden, kommt es aufgrund der großen Anzahl resultierender Raster Tiles schnell zu einem serverseitigen *overload*. Das Rendern von Raster Kacheln *on the fly*, ohne Anlage eines *Caches*, wäre hier eine denkbare Lösung. Jedoch ist hierbei mit einer hohen Serverlast durch wiederholtes Rendern derselben Tiles zu rechnen. Eine praktikable Möglichkeit stellt hier die Auswahl weniger Projektionen dar, von denen aus bei Umprojektion in das erforderliche Ziel-CRS die unvermeidbaren Verzerrungen auf ein Minimum reduziert werden können. Eine intelligente Auswahl des bestmöglichen Quell-CRS für ein gewünschtes Ziel-CRS durch die Software wäre eine denkbare Lösung (Thalheim, *BKG* schriftlich, 16.11.2018). Im Beispiel von Abbildung 19 könnte das die Umprojektion von EPSG 3857 auf EPSG 25833 getreffen. Darüber hinaus empfiehlt es sich, das Angebot eines derartig aufgebauten Dienstes auf wenige ausgewählte Layouts zu reduzieren.

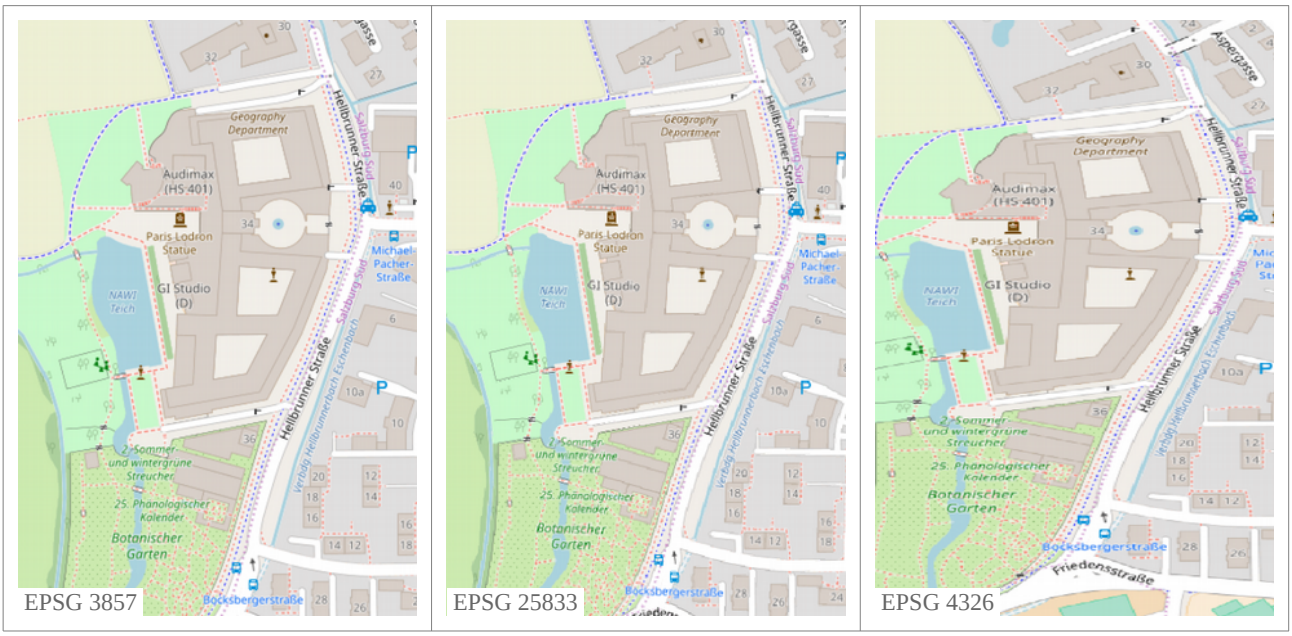


Abbildung 19 Gegenüberstellung von Rasterbildern unterschiedlicher Projektionen
Karte: © *OpenStreetMap* Mitwirkende CC-BY-SA; Ausgangsprojektion EPSG 3857

Von den verbleibenden drei Kombinationen unterliegen ***b)*** und ***d)*** der Einschränkung, dass nur Rasterkacheln im CRS EPSG 3857 gerendert werden können. Hier würde also das Problem starker Verzerrungen beim Veröffentlichen von Projektionen bestehen, die von EPSG 3857 stark abweichen, z. B. bei Umprojektion in EPSG 4326 (WGS 84; vgl. Abbildung 19). Bei Kombination ***d)*** kommt zusätzlich hinzu, dass nach Martinelli & Roth (2015 p. 49) *tessera* auf 50 gleichzeitige Nutzeranfragen limitiert und damit für den

produktiven Einsatz nur bedingt geeignet ist. Grundsätzlich wäre in Variante ***a)*** auch die Nutzung von *Mapnik* statt *TileServer GL* denkbar. Jedoch würde hier die bereits erwähnte Einschränkung von *Mapnik* hinzukommen, sodass keine Umprojektion beim Rendern der Raster Tiles möglich ist. Das würde bedeuten, dass nicht nur Raster Tiles in den im WMS benötigten Projektionen vorgehalten oder *on the fly* gerendert werden müssen, sondern auch die Generierung der zugrundeliegenden Vector Tiles in jeder gewünschten Projektion erforderlich ist. Das würde einen zusätzlichen erhöhten serverseitigen Rechenaufwand bedeuten. Der Speicheraufwand für die Vector Tiles würde sich entsprechend erhöhen, was aber bei dem zu erwartenden Speicherplatzbedarf dieser Daten keine wesentliche Rolle spielt.

Aus den beschriebenen Gründen bleibt als derzeit erfolgversprechendste Variante die Kombination ***a)*** aus *t-rex*, *TileStache* oder *GeoServer* zum Generieren der Vector Tiles und *TileServer GL* zum Rendern und Bereitstellen von Raster Tiles. Diese kann *Mapproxy* als Datenquelle nutzen, um einen WMS anzubieten. Die Vector Tiles könnten hierbei parallel von *TileServer GL* geeigneten Clients zur direkten Nutzung verfügbar gemacht werden. Der Nachteil der geringeren Flexibilität gegenüber einer direkten Einbindung von Vector Tiles bzw. originären Vektordaten in einen WMS bei dieser Konstellation wurde bereits beschrieben. Die Einschränkungen mit Blick auf die Generalisierung von Geometrien beim Erzeugen der Vector Tiles wurden ebenfalls dargelegt (Kap. 6.2). Es bleibt zu untersuchen, ob auf diesem Weg ein Performancegewinn gegenüber einem WMS unter Verwendung originärer Vektordaten zu verzeichnen ist und ob dieser so hoch ist, dass die beschriebenen Einschränkungen bis zur Verfügbarkeit besserer Lösungen in Kauf genommen werden können.

7.2 Umsetzung der Szenarien

7.2.1 Vector Tiles erzeugen

7.2.1.1 Generierung von Vector Tiles mit GeoServer

In einem ersten Versuch mit einem Layer (*sie01_f.shp* des DLM1000; vgl. Kap. 3.2) wurde *GeoServer* verwendet, um Vector Tiles aus *Shape*-Dateien zu erzeugen. Diese sind, wie mit Raster Tiles vergleichbar, über den TMS- oder WMTS- Endpunkt von *GeoWebCache* abrufbar. Die Einbindung in Clients oder Renderern erfolgt über *TileJSON*-Dateien, in welchen der Service beschrieben wird, der die Vector Tiles verfügbar macht. Die Beispiele der erzeugten *TileJSON*-Dateien für die Nutzung der beiden Endpunkte sind im *Github-Repository* zu dieser Arbeit[1] einsehbar. Die Variante für die Nutzung des WMTS-Endpunktes zeigt Code 9.

Durch Einbindung der *TileJSON*-Dateien wurde jeweils versucht, die Vector Tiles über die genannten Endpunkte in *QGIS* mit dem *Vector Tiles Reader Plugin* und mit *TileServer GL* anzuzeigen. Mit *QGIS* konnten die Vector Tiles nur über den WMTS-Endpunkt eingelesen werden. Bei Definition des TMS-Endpunktes innerhalb der *TileJSON*-Datei wurden die Daten nicht dargestellt (vgl. auch *Issues* im Repository des Plugins[2]). Hier scheint es einerseits Probleme zu geben, weil *vector_layers* noch kein offizieller Bestandteil der *TileJSON Specification (Version 2.2.0*[3]) ist und andererseits, weil *GeoServer* (Version 2.13) *TileJSON* zur Datenquellendefinition nicht eingebunden hat. Mit *TileServer GL* war eine Darstellung der Vector Tiles bei testweiser Definition beider Endpunkte nicht umsetzbar. Die versuchsweise Einbindung erfolgte mit Code 10, welcher die Stildefinition in Code 11 aufruft, die mit der Datenquellendefinition in Code 9 verknüpft ist.

1 https://github.com/GjueAtGit/VTs_datasource_ogc/tree/master/geoserver
2 https://github.com/geometalab/Vector-Tiles-Reader-QGIS-Plugin/issues/112
3 https://github.com/mapbox/tilejson-spec/tree/master/2.2.0

Code 9 *TileJSON* zur Definition einer Datenquelle für Vector Tiles

```
# geoserver_wmts.json in Code 11 aufgerufen
{"tilejson":"2.2.0",
  "tiles":[
    "http://localhost:7070/geoserver/gwc/service/wmts?
REQUEST=GetTile&SERVICE=WMTS&VERSION=1.0.0&LAYER=DLM:sie01_f&STYLE=&TILEMATRIX
=EPSG:900913:{z}&TILEMATRIXSET=EPSG:900913&FORMAT=application/x-
protobuf;type=mapbox-vector&TILECOL={x}&TILEROW={y}"
  ],
  "minzoom" : 0,
  "maxzoom" : 14,
  "center" : [10.5668, 51.14895, 8],
  "bounds" : [5.5888, 47.2718, 15.5448, 55.0261],
  "vector_layers":[
    {"id":"sie01_f",
     "geometry_type":"polygon",
     "minzoom":0,
     "maxzoom":14,
     "fields":{
      "LAND":"String", ... }
    }
  ]
}
```

Code 10 Ausschnitt aus der *config.json* von *TileServer GL* zur versuchsweisen Darstellung von mit *GeoServer* generierten Vector Tiles

```
# Ausschnitt aus config.json in TileServer GL
# zugehörige Stildefinition von DLM1000_GeoServer.style.json in Code 11
...
  "styles": {
    "GeoServer_Example": {
      "style": "DLM1000_GeoServer.style.json",
      "tilejson": {
        "type": "overlay",
        "bounds": [5.5888, 47.2718, 15.5448, 55.0261]
      }
    }
  }
...
```

Code 11 Stildefinition zur Darstellung von Vector Tiles aus *GeoServer* in *TileServer GL*

```
# DLM1000_GeoServer.style.json in Code 10 aufgerufen
{
  "version":8,
  "name": "DLM1000 (aus GeoServer)",
  "center": [
    10.306989, 51.020088
  ],
  "zoom": 8,
  "sources": {
    "dlm1000_src": {
      "type": "vector",
      "url": "http://localhost:7070/geoserver/www/geoserver_wmts.json" *
    }
  },
  "layers": [
    {
      "id": "sie01",
      "type": "fill",
      "source": "dlm1000_src",
      "source-layer": "sie01_f",
      "layout": {
        "visibility": "visible"
      },
      "paint": {
        "fill-color": "rgba(199, 15, 15, 0.67)",
        "fill-outline-color": "rgba(88, 5, 5, 1)"
      }
    }
  ]
}
# * in Code 9 dargestellt
```

Obwohl die Vector Tiles über den WMTS-Endpunkt von *GeoServer* in *QGIS* dargestellt werden können, wird beim Versuch des Aufrufs in *TileServer GL* die Fehlermeldung in Code 12 zurückgegeben und die Anzeige im Viewer von *TileServer GL* bleibt leer. Die direkte Einbindung der Datenquelle in Code 11 über den WMTS-Pfad (*tag tiles* in Code 9) führt zum selben Effekt. Das *Pre Seeding* aller Tiles ist mit *GeoWebCache* grundsätzlich möglich. Eine direkte Nutzung der Tiles über die Ordnerstruktur ist jedoch nicht vorgesehen, da diese nicht direkt dem Schema `/z/x/y.pbf` folgt, sondern zusätzliche Unterordner eingefügt und die Dateien nach Schema `x_y.pbf` benannt werden (vgl. Kap. 5.5). Aus diesen Gründen wurde *GeoServer* im Rahmen dieser Arbeit letztendlich nicht zum Erzeugen der Vector Tiles eingesetzt.

Code 12 Fehlermeldung beim Aufruf von Vector Tiles aus *GeoServer* in *TileServer GL*

```
mbgl: { class: 'Style',
        severity: 'ERROR',
        text: 'Failed to load source dlm_1000_src: 0 - The document is empty.'
      }
```

7.2.1.2 Generierung von Vector Tiles mit t-rex

In einem zweiten Versuch wurde versucht, Vector Tiles mit dem Tileserver *t-rex* (Kap. 5.2) zu erzeugen. Erste Aufrufe von *t-rex* über einen *Docker* Container schlugen fehl. Spätere Tests ergaben, dass die Kombination aus *Portmapping* im *Docker*-Kommando mit der Angabe des Parameters `port` in den Konfigurationseinstellungen von *t-rex* der Auslöser waren. Auch wenn beide Portangaben identisch vergeben wurden, also *Portmapping* `-p 7373:6767` und `port = 7373`, wurde der *Viewer* von *t-rex* im Browser nicht angezeigt. Erst nach Weglassen der zusätzlichen Angabe von `port` in der Konfigurationsdatei bzw. des *Portmappings* im *Docker* Kommando resultierte eine erfolgreiche Anzeige der Viewers und eine Nutzung der Vector Tiles (Code 13 und Code 14).

Code 13 Erfolgreiche Aufrufe von *t-rex* in einem *Docker*-Container

```
# Aufruf im Browser jeweils mit http://localhost:7373/
# Konfigurationseinstellungen in Docker-Kommando
docker run --rm --name=trex
           -v 'd:\UNIGIS\Masterarbeit\Docker\t-rex\data_in\:/var/data/in:ro'
           -v 'd:\UNIGIS\Masterarbeit\Docker\t-rex\data_out\:/var/data/out'
           -v 'd:\UNIGIS\Masterarbeit\Docker\t-rex\cache\:/var/cache/mvtcache'
           -p 6767:6767
           sourcepole/t-rex serve
           --bind 0.0.0.0
           --datasource ver01_l.shp
           --clip false
           --detect-geometry-types true
           --no-transform false
           --openbrowser false
           --simplify false
           --cache '\cache\'

# Aufruf mit Konfigurationseinstellungen in myconfig.toml (Code 14)
docker run --rm -p 7373:6767 --name=trex
           -v 'd:\UNIGIS\Masterarbeit\Docker\t-rex\data_in\:/var/data/in:ro'
           -v 'd:\UNIGIS\Masterarbeit\Docker\t-rex\data_out\:/var/data/out'
           sourcepole/t-rex serve
           --openbrowser=true --
           config=myconfig.toml
```

Code 14 Konfigurationsdatei für *t-rex* zum Aufruf aus Docker mit Code 13

```
[service.mvt]
viewer = true

[[datasource]]
name = "dlm1000_src"
path = "ver01_l.shp"

[grid]
predefined = "web_mercator"

[[tileset]]
name = "dlm1000"
attribution = "© GeoBasis-DE / BKG 2018"
extent = { minx = 5.613322, miny = 45.091784, maxx = 16.867215, maxy =
55.759054 }
minzoom = 0
maxzoom = 14

[[tileset.layer]]
name = "ver01_l"
datasource = "dlm1000_src"
table_name = "ver01_l"
geometry_type = "LINE"
simplify = false

[cache.file]
base = "<mein Pfad>/cache/"

[webserver]
bind = "0.0.0.0"
```

Aufgrund der ersten Fehlversuche bei der Nutzung von *t-rex* aus Docker wurde ein Installationsversuch von *t-rex* mit der zum Download angebotenen *msi*-Datei[1] unter *Windows 10* durchgeführt. Das so installierte Programm konnte über die Kommandozeile erfolgreich mit Code 15 gestartet werden. Die Konfigurationsdatei von *t-rex* wurde dafür entsprechend angepasst (Code 16). Anschließend lässt sich der Viewer von *t-rex* mit `http://localhost:7373` im Browser anzeigen.

Code 15 Start von *t-rex* aus einer Windows-Installation

```
# vorher bei o.g. Installation zu C:\Program Files\t-rex navigieren
.\t_rex serve --config d:\<mein Pfad>\data_in\myconfig.toml
              --openbrowser false
```

1 https://github.com/t-rex-tileserver/t-rex/releases/tag/v0.9.0

Code 16 Konfigurationsdatei *myconfig.toml* zum Start von *t-rex* aus einer Windows-Installation

```
[service.mvt]
viewer = true

[[datasource]]
name = "dlm1000_src"
path = "'d:/<mein Pfad>/data_in/ver01_l.shp"

[grid]
predefined = "web_mercator"

[[tileset]]
name = "dlm1000_tileset"
attribution = "© GeoBasis-DE / BKG 2018"
extent = { minx = 5.613322, miny = 45.091784, maxx = 16.867215, maxy =
55.759054 }
minzoom = 0
maxzoom = 14

[[tileset.layer]]
name = "ver01_l"
datasource = "dlm1000_src"
table_name = "ver01_l"
geometry_field = "geom"
geometry_type = "LINE"
simplify = false

[cache.file]
base = "d:/<mein Pfad>/cache/"

[webserver]
bind = "0.0.0.0"
port = 7373
```

Mit dem Aufruf von *t-rex* in Code 15 werden im lokalen *Cache*-Ordner, welcher in Code 16 unter `[cache.file]` definiert wurde, jeweils eine *JSON*-Datei zur Definition der Datenquelle und der Stile angelegt. Die Bezeichnung der Dateien richtet sich nach dem unter `[[tileset]]` definierten `name` (vgl. Code 16), im Beispiel *dlm1000_tileset.json* und *dlm1000_tileset.style.json*. Die Datei zur Stildefintion kann anschließend über den im Viewer von *t-rex* angezeigten Pfad, im gezeigten Beispiel in *TileServer GL* mit `http://localhost:7373/dlm1000.style.json` eingebunden werden (Code 17). Da *TileServer GL* in der Konfigurationsdatei keine http-Pfade zur Definition einer Stildefinition akzeptiert, ist es erforderlich, eine Kopie der von *t-rex* erzeugten Datei im *gemounteten* Datenverzeichnis von *TileServer GL* (Code 22) anzulegen. Diese Datei kann manuell nachbearbeitet werden, um die Darstellung den eigenen Wünschen anzupassen.

Code 17 Code zur Einbindung von mit *t*-rex erzeugten Vector Tiles

```
# Ausschnitt aus config.json in TileServer GL
# zugehörige Stildefinition von DLM1000_t_rex.style.json in Code 18
...
  "styles": {
    "t_rex_Example": {
      "style": "DLM1000_t_rex.style.json",
      "tilejson": {
        "type": "overlay",
        "bounds": [5.613322, 45.091784, 16.867215, 55.759054]
      }
    }
  }
...
```

Code 18 Stildefinition zur Darstellung von Vector Tiles aus *t-rex* in *TileServer GL*

```
# DLM1000_t_rex.style.json in Code 17 aufgerufen
{
  "version":8,
  "name":"DLM1000 (aus t-rex)",
  "sources":{
    "dlm1000_src":{
      "type": "vector",
      "url": "http://localhost:7373/dlm1000.json"
    }
  },
  "layers":[
    {
      "id": "sie01_f",
      "source": "dlm1000_src",
      "source-layer": "sie01_f",
      "type": "fill",
      "layout": {
        "visibility": "visible"
      },
      "paint": {
        "fill-color": "rgba(180, 15, 15, 0.5)",
        "fill-outline-color": "rgba(88, 5, 5, 1)"
      }
    }
  ]
}
```

Beim Starten von *TileServer GL* wird mit den in Code 17 und Code 18 gezeigten Einstellungen in der *PowerShell* die identische Fehlermeldung wie in Code 12 mit Vector Tiles aus *GeoServer* angezeigt. Trotz dieser Fehlermeldung werden die Daten im Viewer von *TileServer GL* korrekt mit der in Code 18 definierten Symbolik dargestellt. Dagegen bleibt die Anzeige bei Auswahl des *Raster Viewers* leer. Ein Grund für die unterschiedliche Anzeige konnte nicht ermittelt werden. Der Versuch, die gerenderten Raster Tiles über die im Viewer von *TileServer GL* angegebenen *endpoints* WMTS und XYZ in einem anderen Client anzuzeigen, blieb ebenfalls erfolglos.

Aus diesem Grund wurden die Vector Tiles mit der Funktion *generate* von *t-rex* in eine Verzeichnisstruktur vorgeneriert und anschließend mit dem Tool *mbutil* in eine *MBTiles*-Datenbank verpackt. Zu beachten ist

hier, dass die Ablage der Tiles in der Verzeichnisstruktur dem *XYZ*-Schema folgt, d.h. der Ursprung des Kachelkoordinatensystems ist im Nordwesten, wogegen die Bezeichnung der Tiles in der *MBTiles*-Datenbank entsprechend der Spezifikation dem *TMS*-Schema folgen muss (vgl. Kap. 4.6.2.1). So ist die Kachel mit Deutschland in Zoomstufe 3 mit `/3/4/2.pbf` aufrufbar, in der Datenbank jedoch als `z=3, x=4, y=5` gespeichert. Beim Verpacken der Tiles mit *mbutil* ist es erforderlich, den Parameter `--scheme=xyz` zu setzen, damit das Tool die Y-Kachelkoordinaten umrechnet. Irritierenderweise wird in den Metadaten der so erzeugten Datenbank als *Scheme* weiterhin *XYZ* angegeben. Die so erzeugte *MBTiles*-Datenbank wurde als Datenquelle für eine Stildefinition in *TileServer GL* eingebunden und damit war sowohl die Anzeige der Vector Tiles als auch das Rendern von Raster Tiles erfolgreich.

Als nächster Schritt wurde die Nutzung unterschiedlicher Projektionen getestet. Wie in Kap. 5.7 beschrieben, unterstützt *TileServer GL*, welcher die mit *t-rex* generierten Vector Tiles weiterverwenden soll, zwar unterschiedliche Koordinatensysteme, ist aber nicht in der Lage, eingehende Vector Tiles umzuprojizieren. Jedoch besteht laut der Dokumentation von *t-rex* die Möglichkeit, Vector Tiles in anderen CRS als EPSG 3857 zu generieren. In diese Richtung wird jedenfalls die Angabe von `[grid] predefined = "web_mercator" | "wgs84"` in der Dokumentation interpretiert. Nach dem Neustart von *t-rex mit* der entsprechend modifizierten *myconfig.toml* meldet die Konsole `Reprojecting geometry to SRID 4326`. Offensichtlich wird die Vorgabe akzeptiert und umgesetzt. Nach einem Neustart des Viewers von *t-rex* bleibt das Kartenfenster der vorhandenen Implementierungen jedoch leer. Aus diesem Grund wurden unterschiedliche CRS der Quelldaten mit den beiden verfügbaren *predefined*-Einstellungen von *t-rex* getestet. Hierbei sind die unterschiedlichen, in Tabelle 4 aufgelisteten Verhaltensweisen zu beobachten. In allen Fällen, bei denen `[grid] predefined = "wgs84"` eingestellt wurde, werden keine Daten in den *Cache* von *t-rex* geschrieben. Wird dagegen ein *Cache* mit dem *t-rex*-Befehl `generate` erzwungen, werden im *Cache* Vector Tiles abgelegt. Auffällig ist, dass in der Datei *metadata.json* im *root*-Verzeichnis der mit `[grid] predefined = "web_mercator" | "wgs84"` generierten Tiles weiter EPSG 3857 als CRS angegeben ist. Aus den oben beschriebenen Gründen war eine Anzeige der Tiles in der Verzeichnisstruktur in anderen Anwendungen nicht umsetzbar.

Um *MVT*-Vector Tiles zur Kontrolle der Inhalte in menschenlesbare Form zu bringen, konnte nur das Tool *vt2geojson*[1] recherchiert werden. Beim Start von *vt2geojson* ist neben dem Pfad der *pbf*-Datei auch die Angabe Tilesetkoordinaten für *z*, *x* und *y* erforderlich (Code 19). Das Tool interpretiert diese jedoch automatisch als Koordinaten eines Tilesets in EPSG 3857, so dass bei der Umwandlung für die Längengrade teilweise Angaben von über 180° resultieren (Beispiel: Original 5,7°, *MVT* 191,5°) und auch die Werte der Breitengrade sind entsprechend verschoben (Beispiel: Original 53,2°, *MVT* 72,2°). Somit ist eine Kontrolle der tatsächlich im Tile gespeicherten Geometrien nicht möglich. Eine weiterführende Recherche nach den Ursachen, die dazu führen, dass die Vector Tiles in EPSG 4326 in *TileServer GL* nicht angezeigt werden, übersteigt den Umfang dieser Arbeit. Ein Verpacken der Vector Tiles, welche im EPSG 4326 erzeugt wurden, in eine *MBTiles*-Datenbank ist aufgrund deren Beschränkung auf EPSG 3857 ebenfalls nicht möglich. Aus diesen Gründen wird im Folgenden beispielhaft ausschließlich mit in einer *MBTiles*-Datenbank verpackten Vector Tiles im CRS EPSG 3857 weitergearbeitet (in *t-rex:* `[grid] predefined = web_mercator"`).

1 https://github.com/mapbox/vt2geojson

Code 19 Beispielaufruf von *vt2geojson*

```
# mit "> D:/3_4_2.geojson" wird das Ergebis in eine Textdatei geschrieben
vt2geojson -l ver01_l D:/3_4_2.pbf -z 3 -x 4 -y 2 > D:/3_4_2.geojson
```

Tabelle 4 Verhalten von *t-rex* bei unterschiedlichen Projektionen der Quelldaten und Konfiguration des *grids*

CRS der Quelldaten	*grid predifined*	Anzeige	Anlage Cache	Fehlermeldung
EPSG 4326	web_mercator	in Projektion EPSG 3857	ja	keine
	wgs84	keine	nein	keine
EPSG 3857	web_mercator	in Projektion EPSG 3857	ja	keine
	wgs84	keine	nein	keine bei *Mapbox GL* und *OpenLayers*; Fehler 1 in Code 20 bei *X-Ray* und *Inspector*
EPSG 25832	web_mercator	in Projektion EPSG 3857	ja	keine
	wgs84	keine	nein	Fehler 2 in Code 20

Code 20 Fehlermeldungen von *t-rex* bei unterschiedlichen Projektionen der Quelldaten *grid predefined="wgs84"*

```
# Fehlermeldung 1
ERROR Unable to transform Extent { minx: -90.0, miny: -90.0, maxx: 0.0, maxy:
0.0 }: Invalid coordinate range while transforming points from EPSG:4326 to
EPSG:3857: None

# Fehlermeldung 2
ERROR Unable to transform Extent { minx: -90.0, miny: -28.125, maxx: -84.375,
maxy: -22.5 }: Invalid coordinate range while transforming points from
EPSG:4326 to +proj=utm +zone=32 +ellps=GRS80 +units=m +no_defs : None
```

Im nächsten Schritt wurde versucht, mehrere Layer in einen Satz von Vector Tiles zu verpacken. Dazu ist im ersten Schritt die Anpassung der Konfigurationsdatei von *t-rex* erforderlich (Code 21). Anschließend wurde *t-rex* gestartet und im Viewer kontrolliert, ob die Anzeige der Daten erfolgreich ist. Die über den Link im Infofenster des Viewers abrufbare, von *t-rex* erzeugte Stildefinition wurde als Grundlage genommen, um den Stil der Darstellung anzupassen und die so erzeugte *JSON*-Datei in den *styles*-Ordner von *TileServer GL* zu kopieren. Die Verknüpfung zur Datenquelle ist in der Stildefinition bereits korrekt enthalten. Die Stildefinitionsdatei muss noch in der *config.json* von *TileServer GL* eingebunden werden, bevor dieser (neu) gestartet wird. Die auf diese Weise mit *t-rex* erzeugten Vector Tiles mit mehreren Layern wurden, nach dem Verpacken in eine *MBTiles*-Datenbank mit *mbutil*, im Viewer von *TileServer GL* fehlerfrei dargestellt.

Code 21 Beispielkonfiguration für *t-rex* für zwei Layer

```
[service.mvt]
viewer = true

[[datasource]]
name = "sie01_src"
path = "d:/<mein Pfad>/data_in/sie01_f.shp"

[[datasource]]
name = "veg02_src"
path = "d:/<mein Pfad>/data_in/veg02_f.shp"

[grid]
predefined = "web_mercator"
#predefined = "wgs84"

[[tileset]]
name = "dlm1000_multi"
attribution = "© GeoBasis-DE / BKG 2018"
extent = { minx = 5.613322, miny = 45.091784, maxx = 16.867215, maxy =
55.759054 }
minzoom = 0
maxzoom = 14

[[tileset.layer]]
name = "sie01_f"
datasource = "sie01_src"
# Select all attributes of table:
table_name = "sie01_f"
#geometry_field = "geom"
geometry_type = "POLYGON"
simplify = false

[[tileset.layer]]
name = "veg02_f"
datasource = "veg02_src"
# Select all attributes of table:
table_name = "veg02_f"
#geometry_field = "geom"
geometry_type = "POLYGON"
simplify = false

[cache.file]
#base = "/var/cache/mvtcache"
base = "d:/<mein Pfad>/cache/"

[webserver]
bind = "0.0.0.0"
port = 7373

#[[webserver.static]]
#dir = "./public/"
#path = "/static"
```

7.2.2 TileServer GL

TileServer GL (Kap. 5.7) soll bei der Umsetzung des Szenarios *a)* (Kap. 7.1) zum Rendern von Raster Tiles aus Vector Tiles eingesetzt werden. Zur Installation steht im *GitHub-Repository* von *TileServer GL*[1] die Befehlszeile zur Installation eines *Docker*-Images und zum Laden in einen *Docker*-Container zur Verfügung (Code 22). Dieser wurde zur Nutzung unter Windows durch *mounten* des Benutzerordner (Verknüpfung des Benutzerordners des *Docker*-Containers mit einem Windowsordner) und *portmapping* auf Port 7171, da Standardport 8080 bereits anderweitig belegt ist, angepasst.

Code 22 *Docker*-Image von *TileServer GL* laden und Container starten

```
# Original entsprechend Anleitung
docker run --rm -it -v $(pwd):/data -p 8080:80 klokantech/tileserver-gl

# angepasster Code für Docker unter Windows
docker run --rm -it --name=tileserv -v ~/tileserver_gl:/data -p 7171:80
           klokantech/tileserver-gl
# oder speziell für Anwendungsfall der Arbeit
docker run --rm -it --name=tileserv -v 'D:/<mein Pfad>/docker_data/:/data'
           -p 7171:80 klokantech/tileserver-gl
# dabei: --name=tileserv -> Name für den Container, sonst automatische Vergabe
#        -p 7171:8080 = verschiebt Port 80 zum Zugriff auf TileServer GL zu
#                       7171 --> Aufruf im Browser mit http://localhost:7171
#        -v ~/tileserver_gl:/data -> verbindet (mounted) data-Verzeichnis des
#                                    Docker-Contaiers mit lokalem Ordner
#                                    <windows-benutzerordner>/tileserver_gl
#                                    alternativ 'D:/MeinPfad/' möglich
```

Das in Code 22 definierte lokale Verzeichnis, auf welches *gemounted* wird, muss vorher unter Windows manuell angelegt werden. Entsprechend der Anleitung sucht *TileServer GL* hier nach der Konfigurationsdatei *config.json* und lädt diese oder startet, falls sie nicht vorhanden ist, mit einer voreingestellten Standard-konfiguration. Eine Beispielkonfiguration wird in der Dokumentation von *TileServer GL*[2] gezeigt. Diese wurde auf die Erfordernisse dieser Arbeit angepasst (Code 23). Die Stildefinitionen erfolgen in den in Code 23 unter *styles* definierten Dateien im JSON-Format entspre-chend der *Mapbox Style Specification*[3]. Eine Beispieldefinition für die in Code 23 aufgeführten Stile zeigt Code 24.

1 https://github.com/klokantech/tileserver-gl
2 https://tileserver.readthedocs.io/en/latest/config.html
3 https://www.mapbox.com/mapbox-gl-js/style-spec/

Code 23 Definition in *config.json* für den Zugriff mit *TileServer GL* auf die mit *t-rex* erzeugten Vector Tiles des Layers *ver01_l* des *DLM1000* in drei unterschiedlichen Tilegrößen

```
{
  "options": {
    "paths": {
      "root": "",
      "fonts": "fonts",
      "sprites": "sprites",
      "styles": "styles",
      "mbtiles": ""
    },
    "domains": [
      "localhost:7171",
      "127.0.0.1:7171"
    ],
    "formatQuality": {
      "jpeg": 80,
      "webp": 90
    },
    "maxScaleFactor": 3,
    "maxSize": 2048,
    "pbfAlias": "pbf",
    "serveAllFonts": false,
    "serveStaticMaps": true
  },
  "styles": {
    "DLM1000_ver01_256px": {
      "style": "DLM1000_ver01_l_256px.style.json",
      "tilejson": {
        "type": "overlay",
        "bounds": [5.6, 45.0, 15.5, 55.2]
      }
    },
    "DLM1000_ver01_512px": {
      "style": "DLM1000_ver01_l_512px.style.json",
      "tilejson": {
        "type": "overlay",
        "bounds": [5.6, 45.0, 15.5, 55.2]
      }
    },
    "DLM1000_ver01_4096px": {
      "style": "DLM1000_ver01_l_4096px.style.json",
      "tilejson": {
        "type": "overlay",
        "bounds": [5.6, 45.0, 15.5, 55.2]
      }
    }
  }
}
```

Code 24 Beispiel Stildefinition für die Darstellung von Vector Tiles nach *Mapbox Style Specification*
Datei *DLM1000_ver01_l_256px.style.json* aus Code 23

```
{
  "version": 8,
  "name": "DLM1000 ver01_l EPSG 3857 256px (aus MBTiles-Datenbank)",
  "metadata": {
    ...
  },
  "sources": {
    "DLM_ver01_l_256px": {
      "type": "vector",
      "url": "mbtiles://dlm1000_ver01_l_256px.mbtiles"
    }
  },
  "glyphs": "http://localhost:7171/fonts/{fontstack}/{range}.pbf",
  "layers": [
    {
      "id": "ver01_l",
      "type": "line",
      "source": "DLM_ver01_l_256px",
      "source-layer": "ver01_l",
      "paint": {
        "line-color": "rgba(88, 88, 88, 1)"
      }
    }
  ],
  "id": "DLM_ver01_l_256"
}
```

Aus den Angaben in *wmts.xml* aus dem *GetCapabilities-Response* ist abzulesen, dass *TileServer GL* über diesen Endpunkt die *TileMatrixSets* EPSG 3857 und 4326 unterstützt. Bei der Einbindung von *MBTiles* mit EPSG 3857 ist beim entsprechenden Layer nur das *TileMatrixSet GoogleMapsCompatible* (EPSG 3857) verknüpft. Dies entspricht der Aussage, dass keine Umprojektion durch *TileServer GL* möglich ist. Bei den untersuchten Szenarien konnte jedoch aus den bei *t-rex* (Kap. 7.2.1.2) der WMTS-Endpunkt nicht mit WGS84 (EPSG 4326) getestet werden.

7.2.3 Mapproxy

Die Installation von *Mapproxy* wurde entsprechend der Anweisungen der Dokumentation[1] unter Windows durchgeführt. Als Erweiterungen wurden zusätzlich *PyProj*[2] (Version *pyproj-1.9.5.1-cp27-cp27m-win32.whl*) und *Shapely and GEOS*[3] (Version *Shapely-1.6.4.post1-cp27-cp27m-win32.whl*) installiert. Nach erfolgreicher Installation kann *Mapproxy* über die Eingabeaufforderung mit den Kommandos in Code 25 gestartet werden. Eine Beispielkonfiguration für die Nutzung einer von *TileServer GL* generierten *XYZ*-Datenquelle für einen WMS in *Mapproxy* ist in Code 26 dargestellt. Versuche der Einbindung des WMTS-Endpunktes von *TileServer GL* als Datenquelle für *Mapproxy* waren nicht erfolgreich, auch wenn dieser in anderen Clients, z. B. *QGIS* fehlerfrei dargestellt wurde.

1 https://mapproxy.org/docs/latest/install_windows.html
2 https://www.lfd.uci.edu/~gohlke/pythonlibs/#pyproj
3 https://www.lfd.uci.edu/~gohlke/pythonlibs/#shapely

Code 25 Kommandos zum Start von *Mapproxy* unter Windows

```
# Virtuellen Umgebung für Mapproxy
#   Erstellen eines Ordners für die virtuelle Umgebung (nur beim ersten Start)
md C:\mapproxy_venv\
# Anlegen der virtuellen Umgebung
#   dabei entspricht C:\Python27 dem Installationsordner von Python
#   und c:\mapproxy_venv einem vorher anzulegenden Lokalen Ordner für die
#   virtuelle Umgebung (mit md)
C:\Python27\python C:\Python27\Lib\site-packages\virtualenv.py c:\
mapproxy_venv
# aktivieren der virtuellen Umgebung
C:\mapproxy_venv\Scripts\activate.bat
# Anlegen eines Ordners für den Cache und Konfigurationsdateien
# im Beispiel im Ordner der virtuellen Umgebung
md C:\mapproxy_venv\my_data
# zum Script-Verzeichnis von Python wechseln (Speicherort mapproxy-util)
cd C:\Python27\Scripts
# Starten von Mapproxy mit einer Beispielkonfiguration (myconfig.yaml)
# mit -b (bind) wird hier die IP-Adresse mit freiem Port des
# Installationsrechners von Mapproxy im Netzwerk angegeben; dies ist
# erforderlich, wenn von einem anderen Rechner auf den Service zugegriffen
# werden soll, andernfalls funktioniert hier die Angabe von localhost:port
mapproxy-util serve-develop C:\mapproxy_venv\my_data\myconfig.yaml
                            -b 192.168.0.10:7575
# Zugriff auf den in myconfig.yaml konfigurierten Service
#   mit QGIS WMS/WMTS-Verbindung
http://192.168.0.10:7575/service
#   WMS GetCapabilities
http://192.168.0.10:7575/service?REQUEST=getcapabilities
```

Die in Code 26 gezeigte Beispielkonfiguration speichert die angeforderten Daten mit den Einstellungen unter `caches:` persistent, legt also einen *Cache* an. Das bedeutet, dass mit zunehmender Anzahl von Zoomstufen eine wachsende Zahl von Bildkacheln gespeichert wird. Dies kann bei mehreren Projektionen und Stilen zum bereits beschriebenen exponentiellen Anwachsen des Speicherbedarfs führen (Kap. 7.1). Das *Caching*-Verhalten von *Mapproxy* kann mit den Parametern `use_direct_from_level` und `disable_storage` gesteuert werden. Mit `use_direct_from_level` wird gesteuert, welche Zoomstufen gecachet und welche immer direkt von der Quelle angefordert werden. Wenn `disable_storage: true` festgelegt wird, speichert *Mapproxy* keine Kacheln für den betreffenden *Cache*. Beides hat zur Folge, dass der Speicherbedarf des Services reduziert wird, aber die betroffenen Anfragen vermutlich länger dauern als mit gefülltem *Cache*, da die entsprechenden Bilder erst bei Anfrage gerendert werden.

Code 26 Beispielkonfiguration in *myconfig.yaml* für die Einbindung von *XYZ-Tiles* aus *TileServer GL* (Code 23) in einen WMS in *Mapproxy*

```
services:
  wms:
    md:
      title: Layer ver01_l des DLM1000 BKG aus TileServer GL
      abstract: © GeoBasis-DE / BKG 2018; Zugriff auf XYZ-Endpunkt von TileServer GL.
    srs: ['EPSG:3857']

layers:
  - name: DLM1000_ver01_l_256px
    title: DLM1000 ver01_l BKG 256px
    sources: [mycache_xyz_256px]

caches:
  mycache_xyz_256px:
    grids: [GLOBAL_WEBMERCATOR]
    sources: [my_tile_source_256px]

grids:
  my_grid:
    base: GLOBAL_WEBMERCATOR
    srs: 'EPSG:3857'
    origin: 'nw'

sources:
  my_tile_source_256px:
    type: tile
    grid: my_grid
    url: http://localhost:7171/styles/DLM1000_ver01_256px/%(z)s/%(x)s/%(y)s.png
    coverage:
      bbox: [5.6, 45.0, 15.5, 55.2]
      srs: EPSG:4326
    transparent: true
```

7.2.4 MapServer und GDAL

Wie in Kap. 5.6, 5.11 und 6.2 beschrieben, können Vector Tiles aktuell von *MapServer* nur mit Einschränkungen als Datenquelle genutzt werden. Trotzdem sollte untersucht werden, ob bereits beim bisherigen technischen Stand ein Performancegewinn bei der direkten Nutzung von Vector Tiles als Datenquelle für einen WMS abzulesen ist. Darüber hinaus sollte *MapServer* als zweite häufig verbreitete Serversoftware Vergleichsdaten für Szenarien mit Shape-Dateien als Datenquelle liefern.

Aus diesem Grund wurden unterschiedliche Installationsversuche mit *MapServer* unternommen. Die Tests mit den *Docker Images mapserver/mapserver:latest*[1] und *camtocamp/mapserver:7.2*[2] verliefen erfolglos. Der enthaltene *Apache*-Server meldete Fehler, die *Logfiles*, auf die in der Fehlermeldung verwiesen wurde, blieben jedoch ohne Eintrag, so dass der Fehler nicht nachvollzogen und behoben werden konnte. Dieses Verhalten war identisch bei der Installation von *Docker* unter *Windows 10* und unter *Ubuntu 18.04*. Das Installationspaket *MS4W* (*MapServer 4 Windows*)[3] konnte zwar unter *Windows 10* erfolgreich installiert werden, liegt jedoch aktuell in Version 3.2.7 vor, welches *MapServer* 7.0.7 und *GDAL* 2.2.4 enthält und somit Vector Tiles nicht unterstützt.

1 https://store.docker.com/community/images/mapserver/mapserver
2 https://store.docker.com/community/images/camptocamp/mapserver
3 https://www.ms4w.com

Erfolgreich war schließlich die Installation von *Binaries* von *MapServer* und *GDAL*, welche auf der Internetseite *GISInternals*[1] bereitgestellt werden und die anschließende Verknüpfung mit einer vorhandenen Installation von *Apache* 2.4, wie sie auf der Internetseite von *OneGeology*[2] beschrieben wird. Die Anpassungen wurden entsprechend der Anleitung unter Verwendung eigener Pfade vorgenommen und *Apache* der Port 7979 zugewiesen, um potenzielle Komplikationen mit anderen Anwendungen bei Nutzung des Standardports 8080 vorzubeugen. Der WMS war danach über die URL `http://localhost:7979/cgi-bin/exemplars/dlm1000/wms` verfügbar. Entsprechend der Spezifikation von *MapServer* war anschließend noch die Definition der *Mapfiles* mit den Konfigurationseinstellungen des WMS und der Datenquellen erforderlich. Auf diese Weise konnte der Zugriff auf die Originaldaten im *Shape*-Format und auf Vector Tiles in einer *MBTiles*-Datenbank über `CONNECTIONTYPE OGR` realisiert werden. Der Versuch, mit *MapServer* auf Vector Tiles in einer Verzeichnisstruktur zuzugreifen, war nicht erfolgreich. Dabei war es gleichgültig, ob als Parameter bei `CONNECTION` das Verzeichnis mit den Unterordnern der Zoomstufen oder die ebenfalls in diesem Verzeichnis vorhandene *metadata.json* direkt angegeben wurde. Auch der parallele Versuch, mit *MapManager* (vgl. Kap. 5.6) Vector Tiles aus einer Verzeichnisstruktur aufzurufen, war erfolglos. Hier scheint die Implementierung des *GDAL*-Treibers noch nicht vollständig zu sein. Eine tiefer gehende Recherche nach den Ursachen wurde im Rahmen dieser Arbeit nicht durchgeführt. Die aus den Installations- und Konfigurationsversuchen entwickelten und schließlich im Folgenden verwendeten *Mapfiles* sind im GitHub-Repository dieser Arbeit[3] hinterlegt.

7.3 Performance-Vergleiche

Die Performance-Vergleiche wurden zur einfacheren Nachvollziehbarkeit basierend auf nur einem Eingangsdatensatz durchgeführt. Dazu wurde aus dem DLM1000 der Layer *Straßenverkehr* (*ver01_l.shp*, Geometrie Linie, 92.223 Features mit 973.976 Vertices) aufgrund der höchsten Featureanzahl ausgewählt (vgl. Kap. 3.2 und Anlage 1). Als Symbolik wurde jeweils eine einfache, durchgängige Linie in der Farbe *RGB 255 0 0* definiert. Somit sind die Angaben der unterschiedlichen Programme und Dateiformate vergleichbar.

7.3.1 Generierung und Speicherbedarf

Die Zeiten für die Generierung und der Speicherbedarf von Vector- und Raster Tiles sind abhängig von der Anzahl der Zoomstufen, Layer und Features, der Symbolisierung, dem Dateiformat, den Rechnerressourcen und anderen Parametern. Insoweit stellen die hier vorgestellten Werte Momentaufnahmen dar, bieten jedoch die Möglichkeit, verschiedene Konstellationen bei identischen Rahmenbedingungen in Relation zu einander zu betrachten.

In Tabelle 5 werden die unterschiedlichen Erstellungszeiten und resultierenden Dateigrößen für Vector Tiles dargestellt und mit Raster Tiles verglichen. Hierbei wird deutlich, dass bei identischem Datensatz und Zieldateiformat (*MVT/PBF*) Vector Tiles mit *t-rex* im Vergleich zu *GeoServer* schneller generiert werden. Der Speicherplatzbedarf ist jedoch bei den mit *t-rex* generierten Vector Tiles höher. Die Ursache ist in der unterschiedlichen Herangehensweise bei der Generalisierung zu suchen. Für *GeoServer* erfolgt in einem ersten Schritt eine Generalisierung bzw. Vereinfachung der Geometrien entsprechend der aktuellen Zoomstufe mit anschließender Eliminierung kleiner und redundanter Features (Blasby & Hocevar 2016b; Folien 14 und 15). Darüber hinaus werden die Vertices der Geometrien bei der Ausgabe von *Mapbox Vector Tiles* in ganz-

1 http://download.gisinternals.com/sdk/downloads/release-1900-x64-gdal-2-3-2-mapserver-7-2-1.zip
2 http://www.onegeology.org/wmsCookbook/4_4_3.html
3 https://github.com/GjueAtGit/VTs_datasource_ogc/blob/master/mapserver/masterthesis/README_de.md

zahligen Bildschirmkoordinaten gespeichert. Bei *t-rex* findet in der aktuellen Fassung eine echte Generalisierung von Geometrien nur dann statt, wenn diese aus *PostGIS*-Datenbanken bezogen werden (Kap. 5.2). Das ist bei den hier verwendeten Beispieldaten nicht der Fall. Demnach erfolgt eine Vereinfachung ausschließlich durch die Umwandlung der Realweltkoordinaten in Bildschirmkoordinaten. Die Dauer der Übertragung der mit *t-rex* erzeugten Vector Tiles aus einer Verzeichnisstruktur in eine *MBTiles*-Datenbank wird in Tabelle 5 nicht separat aufgeführt. Bei der hier aufgeführten Anzahl von Tiles betrug die Dauer mit der in Abbildung 24 als *Server* dargestellten Rechnerleistung nur ~1 Minute.

Tabelle 5 Gegenüberstellung der Dauer der Generierung, Anzahl und Größe von Vector- und Raster Tiles Layer *ver01_l* des DLM1000, Zoomstufen 0 bis 14, EPSG 3857

Software	**Dateiformat**	**Größe Tiles** [px]	**Zeit** [HH:MM]	**Anzahl VTs** (ohne leere Tiles)	**Speicherbedarf** Einzeln / MBTiles [MB]	**Min** [KB]	**Max** [KB]
t-rex	MVT/PBF	256	00:52	186.661	117 / 133	1	1.911
		512	00:47		123 / 139	1	1.991
		4.096*	00:55		262 / 163	1	2.424**
GeoServer	MVT/PBF	256	01:30	187.409	154	1	1.401
	PNG	256	02:17		854	2	88
	GeoJSON	256	01:38		718	1	16.545***
Original	*Shape*				*63*		

* *default*
** *Kachel z=3, x=4, y=2 (Ursprung NW), 92.223 Features*
*** *Kachel z=6, x=33, y=21 (Ursprung NW), 33.864 Features*

Eine Relation zwischen den hier verwendeten Beispieldaten des Layers Verkehr (*ver01_l*) und dem vollständigen Datensatz mit allen 28 Geometrielayern des DLM1000 ist aus Tabelle 6 ablesbar. Die Generierungszeit bei der Verwendung von *GeoServer* erhöht sich hierbei auf mehr als das 20fache und der Speicherplatzbedarf steigt fast auf das 4fache an. Exakt lassen sich die Werte nicht auf eine andere Anzahl von Zoomstufen übertragen, da die Generierung eines Vector Tiles in den kleineren Zoomstufen deutlich länger dauert als in den höheren Zoomstufen (28 Layer: Zoomstufe 7 ~9 Tiles/Minute; Zoomstufe 10 ~75 Tiles/Minute) und die Tiles im Durchschnitt kleiner werden (28 Layer: Zoomstufe 7 Ø 755 KB/Tile; Zoomstufe 10 Ø 37 KB/Tile). Jedoch kann anhand der Zahlen geschätzt werden, dass bei der hier verwendeten Hardwareleistung (Abbildung 24, PC *Server*) für die Zoomstufen 0 bis 14 (vgl. Tabelle 5) die Generierungszeit ca. 20 bis 30 Stunden beträgt und der Speicherplatz unter 1 GB liegt. Damit ist der Speicherbedarf bei heutigen Festplattenkontingenten für einen Satz Vector Tiles vernachlässigbar.

Als Vergleich mit Raster Tiles aus einer produktiven Umgebung können Beispielzahlen des *BKG* herangezogen werden (Thalheim, *BKG* 13.11.2018 schriftlich). Hier wurde das DLM1000 des *BKG* in den Zoomstufen 0 bis 14 (ca. 1 : 591 Mio. bis 1 : 36.000) im Kachelungsschema *GoogleMapsCompatible* in der aktuellen Symbolik des WMS des *BKG* gerendert. Die Dauer betrug ca. 45 Minuten und es resultierten 255.502 Kacheln mit einem Speicherbedarf von 1,33 GB. Mit diesen Werten wäre es problemlos möglich, auch andere Stile und Projektionen in den genannten Zoomstufen vorzuhalten. Das *BKG* bietet Hintergrundkarten über Darstellungsdienste in der Regel jedoch bis zum Maßstab 1:1.000 an, was beim hier genannten

Kachelungsschema fünf zusätzliche Zoomstufen mit einem exponentiellen Anstieg von Generierungszeit, Anzahl und Speicherplatzbedarf nach sich ziehen würde. Als Beispiele liegen die Werte für den *WebAtlas*[1] in Farbdarstellung vor. Diese belegen in EPSG 25832 einen Speicherplatz von ca. 139 GB und in EPSG 3857 ca. 234 GB. Allein die hierbei erforderliche Zusammenführung der von den Bundesländern vorprozessierten *Caches* benötigt mehrere Tage (Thalheim, *BKG* 14.11.2018 schriftlich). Diese Zahlen machen deutlich, wie aufwändig es sein kann, einen Kartendienst in mehreren Projektionen und Darstellungen vorzuhalten.

Tabelle 6 Gegenüberstellung der Dauer der Generierung, Anzahl und Größe von Vector Tiles mit unterschiedlicher Anzahl enthaltener Layer
DLM1000, Zoomstufen 7 bis 10, erzeugt mit *GeoServer*, Format MVT/PBF

Anzahl Layer	**Anzahl Vector Tiles**		**Zeit** [Min]	**Speicherbedarf** (ohne leere Tiles)		
	gesamt	ohne leere		**Tiles gesamt** [MB]	**Min** [KB]	**Max** [KB]
1	1.589	1.272	1	31	1	654
28	1.966	1.439	21	116	1	1.781

Der jeweilige Effekt bei der Erstellung von Vector Tiles im Vergleich zu den Ausgangsgeometrien im *Shape*-Format ist in Abbildung 20 und Tabelle 7 ablesbar. Die stärkste Generalisierung, die auch dem Effekt entspricht, der aufgrund der Beschreibung von Blasby und Hocevar (2016b) erwartet wurde, ist bei den mit *GeoServer* generierten *Mapbox Vector Tiles* zu erkennen. Bei dem vergleichbaren Tile im *GeoJSON*-Format ist der Effekt der Generalisierung geringer. Statt 738 Features wie beim *MVT*-, sind im *GeoJSON*-Format der gleichen Kachel 12.229 Features enthalten. Hinzu kommt das ineffizientere Speicherformat, so dass bei *GeoJSON* vor allem in den kleineren Zoomstufen um ein Vielfaches größere Vector Tiles resultieren als mit *MVT*. Infolgedessen steigt bei *GeoJSON* die Zugriffszeit entsprechend an. Dieser Effekt ist bei Yu et al. (2017 pp. 4–5; Fig. 4 & 5) beschrieben und anschaulich grafisch dargestellt.

Bei dem vergleichbaren, mit *t-rex* erzeugten Tile ist die Umwandlung in ganzzahlige Bildschirmkoordinaten anhand der resultierenden stufenförmigen Geometrien zu erkennen. Sehr kleine, bei dieser Zoomstufe nicht sinnvoll darstellbare Features wurden jedoch nicht entfernt. Es sind alle 92.223 Features und ebenfalls alle Vertices der Ausgangsdaten enthalten. Bei 973.976 enthaltenen Vertices in Zoomstufe 3 und einer maximal möglichen Anzahl von 65.536 unterschiedlichen Punkten in einer 256x256 Pixel großen Kachel bedeutet dies, dass viele identische Vertices und auch Features mit komplett identischen Geometrien übereinander liegen. Etwas abgeschwächt, aber vergleichbar ist diese Situation auch bei größeren Kacheln. Somit wird die Größe eines Tiles mit in dieser Zoomstufe nicht darstellbaren Informationen unnötig aufgebläht. Dieser Sachverhalt wird in Abbildung 21 visualisiert. Das mit *GeoServer* erzeugte *MVT*-Tile ist zwar datentechnisch sehr optimiert, gibt dagegen die Wirklichkeit des Layers *Straßenverkehr* nicht adäquat wieder. Im Beispiel wirken Ballungsräume straßenärmer, wogegen in ländlichen Regionen eine höhere Dichte von Verkehrswegen dargestellt wird. Das die beiden anderen Tiles für diese Zoomstufe zu viele Informationen enthalten, ist anhand der Darstellung in Abbildung 21 ebenfalls eindeutig erkennbar. Mit diesen Ausführungen wird, neben sinnvollen Generalisierungsmechanismen auch mit Blick auf die kartografische Darstellung, ein Teil

1 http://www.geodatenzentrum.de/geodaten/gdz_rahmen.gdz_div

der Problematik zur fehlenden Nachvollziehbarkeit der Generierung von Vector Tiles deutlich, wie sie in Kap. 4.5.1.1 recherchiert und beschrieben wurde.

Die zugehörigen Dateigrößen (Tabelle 7) der in Abbildung 21 dargestellten Tiles verdeutlichen den Einfluss auf die Performance und hier v. a mit Blick auf die Übertragung zum Client bzw. die Auswertungsdauer beim Rendern. Zusätzlich ist in Tabelle 7 (Zeile 1 und 2) der Größenunterschied von *MVT* und *GeoJSON* bei vergleichbarem Inhalt sehr eindrücklich ablesbar, welcher die Effektivität von *Protocol Buffers* als Speicherformat unterstreicht.

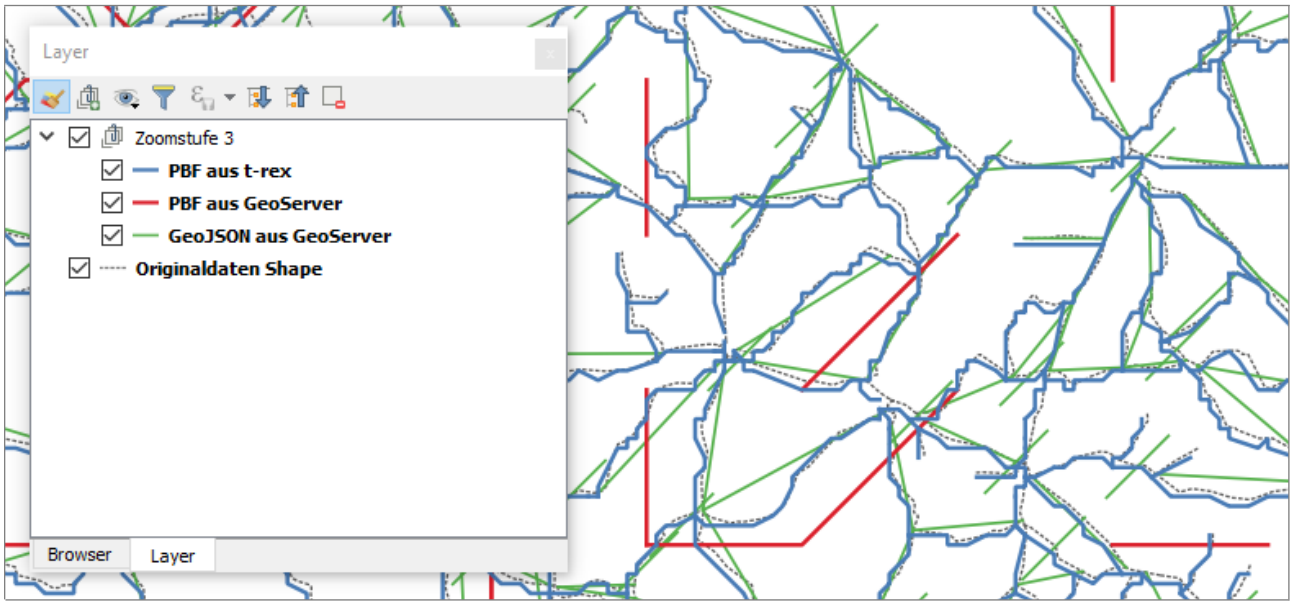


Abbildung 20 Unterschiedliche Generalisierung von Vector Tiles mit *GeoServer* und *t-rex* in den Dateiformaten *MVT/PBF* und *GeoJSON* bei identischen Eingangsdaten und gleicher Zoomstufe
Layer *ver01_l* DLM1000 (© GeoBasis-DE / BKG 2018); Vector Tile z=3, x=4; y=2; EPSG 3857

Tabelle 7 Vergleich von Dateigröße und Anzahl enthaltener Features in Vector Tiles identischer Ausdehnung mit unterschiedlicher Generierung (vgl. Abbildung 20)
Layer *ver01_l* des DLM1000, EPSG 3857, Tile z=3 x=4 y=2

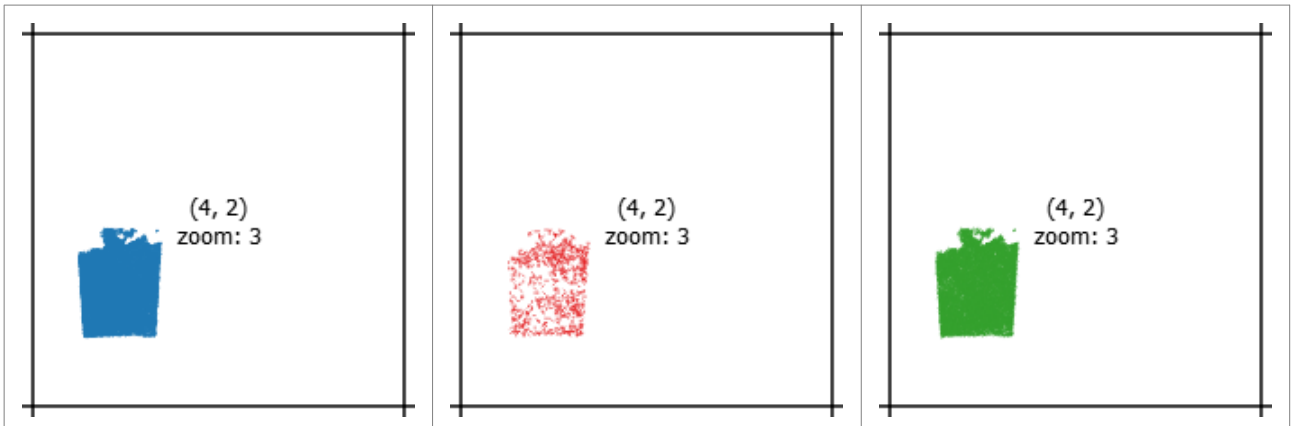
Software	**Dateiformat**	**Größe Tiles** [px]	**Speicherbedarf** [KB]	**Anzahl Features** (Original 92.223)	**Anzahl Vertices** (Original 973.976)
t-rex	MVT/PBF	256	1.874	92.223	973.976
		512	1.946		
		4.096 *(default)*	2.310 *(140.565 nach Umwandlung in GeoJSON)*		
GeoServer	MVT/PBF	256*	73 *(598 nach Umwandlung in GeoJSON)*	738	1.633
	GeoJSON	256*	6.715	12.229	24.463

** entnommen aus den Angaben der Eingabeaufforderung von Windows, vgl. Code 27*

Code 27 Informationen der Eingabeaufforderung von Windows beim *Seeding* von Vector Tiles mit *GeoServer/GeoWebCache*

```
# Mapbox Vector Tiles
RawKvp = {FORMAT=application/x-protobuf;type=mapbox-vector, STYLES=line,
WIDTH=256, HEIGHT=256, LAYERS=DLM:ver01_l_3857, EXCEPTIONS=SE_XML,
GWC_SEED_INTERCEPT=true, REQUEST=GetMap, SRS=EPSG:900913,
BBOX=1565430.3390625007,7200979.559687499,1878516.4068749994,7514065.627500001
, VERSION=1.1.1, SERVICE=WMS, TRANSPARENT=true}

# GeoJSON Vector Tiles
RawKvp = {FORMAT=application/json;type=geojson, STYLES=line, WIDTH=256,
HEIGHT=256, LAYERS=DLM:ver01_l_3857, EXCEPTIONS=SE_XML,
GWC_SEED_INTERCEPT=true, REQUEST=GetMap, SRS=EPSG:900913,
BBOX=0.0,5009377.085000001,2504688.5425000004,7514065.627500001,
VERSION=1.1.1, SERVICE=WMS, TRANSPARENT=true}
```



MVT/PBF aus *t-rex* * | *MVT/PBF* aus *GeoServer* | *GeoJSON* aus *GeoServer*

Abbildung 21 Darstellung unterschiedlich erzeugter Vector Tiles desselben Ausschnittes (vgl. Abbildung 20)
QGIS 2.18, Linienstärke 0,15 mm; Kachelgrenzen und -beschriftung mit *TileLayerPlugin*
** identisches Aussehen, ob mit Tiles der Größe 256 px, 512 px oder 4.096 px (vgl. oben)*
Kartengrundlage: Layer *ver01_l* des *DLM1000* (© GeoBasis-DE / BKG 2018)

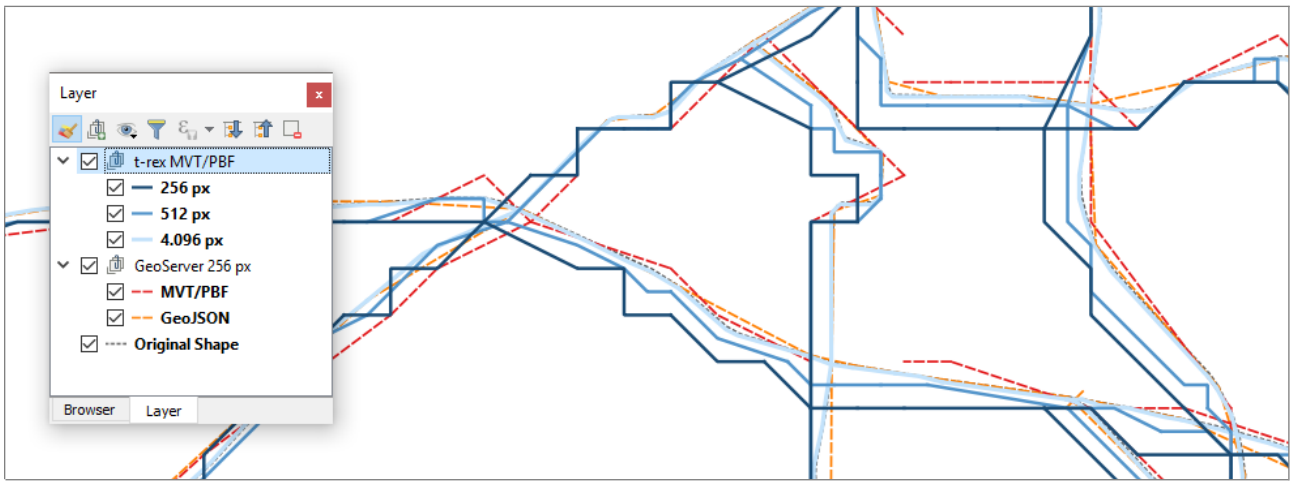


Abbildung 22 Unterschiedliche Generalisierung von mit *t-rex erzeugten* Vector Tiles bei unterschiedlichen Kachelgrößen im Vergleich mit Kacheln aus *GeoServer* und Originaldaten
Kartengrundlage: Layer *ver01_l* des *DLM1000* (© GeoBasis-DE / BKG 2018)
Vector Tile z=8, x=134; y=82; EPSG 3857

Neben der zwischen den genutzten Programmen unterschiedlichen Herangehensweise bei der Erzeugung von Vector Tiles kann das Ergebnis noch durch die Kachelgröße (in Pixeln) beeinflusst werden. *T-rex* verfügt über die Option, die Kachelgröße festzulegen. Der *default*-Wert ist 4.096 Pixel. Nachdem bei dieser Einstellung die Geometrie fast deckungsgleich mit denen der hier verwendeten Originaldaten verläuft (vgl. Abbildung 21), ist bei einer Vorgabe von 512 oder 256 Pixeln die Generalisierung in derselben Zoomstufe stärker. Jedoch ist die tatsächliche Reduktion des Speicherbedarfs aus den oben beschriebenen Gründen gering und resultiert ausschließlich aus den kleineren *Integer*-Werten der Pixelkoordinaten. Alle anderen Informationen sind, bei gleichem *XYZ*-Wert unabhängig von der Kachelgröße identisch. Hier wird von *t-rex* eine echte Reduktion der Information und damit ein resultierender Performancegewinn aktuell noch nicht umgesetzt. Diese ist für zukünftige Versionen jedoch geplant (Kap. 5.2).

Zusätzlich konnte die Aussage von Blanc et al. (2016 p. 16) belegt werden, nach welcher der Zeitaufwand zum Generieren von Vector Tiles geringer ist als bei vergleichbaren Raster Tiles (Tabelle 5, *GeoServer*). Dieses ist insofern zu erwarten, da bei Vector Tiles nur ein *Clippen* der Geometrien und ggf. eine Umwandlung in Bildschirmkoordinaten erforderlich ist, bei Raster Tiles dagegen ein komplettes Rendern der Bilder. In Anlehnung an Ingensand et al. (2016 p. 7; Fig. 3) wurden darüber hinaus die Dateigrößen von mit *GeoServer erzeugten* Vector- und Raster Tiles verglichen (Abbildung 23). Es ist deutlich ablesbar, dass in den niedrigeren Zoomstufen die Raster Tiles weniger Speicher beanspruchen, in den höheren Zoomstufen dagegen Vector Tiles kleiner sind. Die Größe der Einzeldateien ist bei Raster Tiles unabhängig von der Zoomstufe ausgeglichener, d. h. die Schwankungen sind sowohl zwischen den Zoomstufen als auch innerhalb dieser geringer. Bei den hier untersuchten Beispieldaten ist die Größe bei Zoomstufe 9 in etwa identisch. In Kombination mit der exponentiell wachsenden Anzahl einzelner Tiles in den höheren Zoomstufen erklärt sich so der in Summe geringere Speicherbedarf von Vector Tiles.

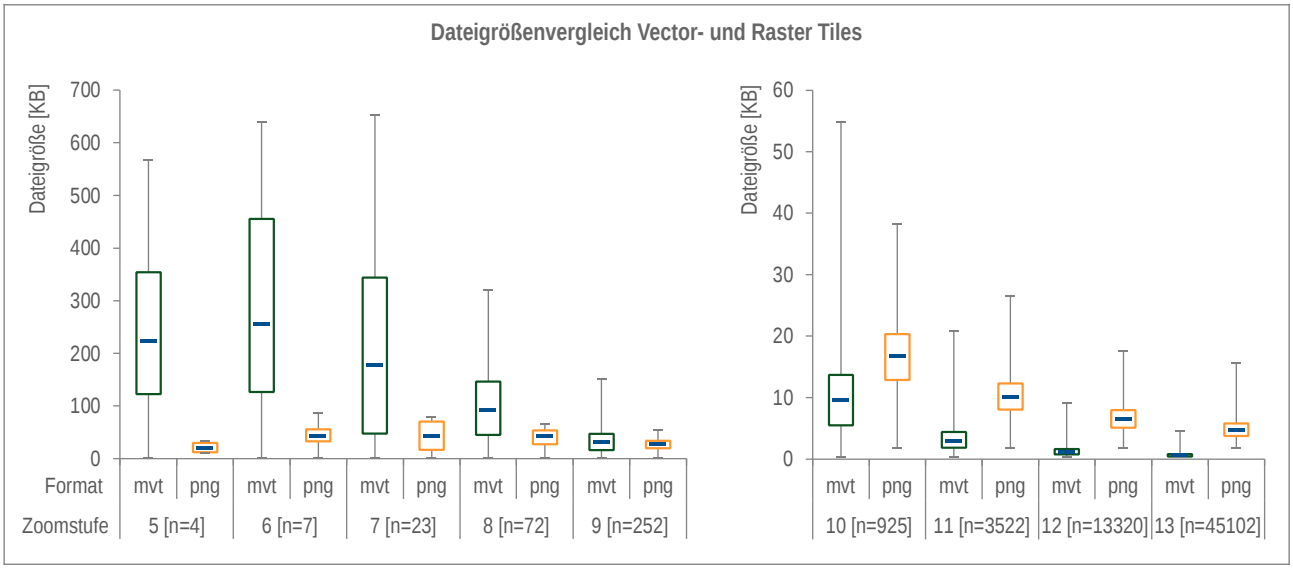


Abbildung 23 Boxplots zum Vergleich von Dateigrößen von Vector- und Raster Tiles der Zoomstufen 5 bis 13
Angegeben sind jeweils der Bereich, in welchem 50 % der Werte liegen, der Median und der jeweilige Minimal- und Maximalwert; erzeugt wurden die Tiles beider Formate mit *GeoServer*

7.3.2 Lasttests mit Apache JMeter™

Zum Ermitteln der Performance der in Kap. 7.1 (Abbildung 18) abgeleiteten und umgesetzten Szenarien wurde *Apache JMeter™*[1] Version 4 eingesetzt. Diese Software eignet sich, um Lasttests beim Zugriff auf einen Webservice von einem oder mehreren Nutzern zu simulieren. *JMeter* sendet *Requests* an den Service und misst verschiedene Parameter, wie z. B. Zugriffszeiten, Anzahl der Zugriffe und übertragene Datenmengen. Dabei ist das Programm sehr vielseitig konfigurierbar. Für die hier genutzte Konfiguration sind mindestens die *JMeter*-Plugins *HTTP Protocol Support*[2], *Java Components* und *Parameterized Controller & Set Variables Action*[3] erforderlich. Aus Gründen der Vergleichbarkeit war während der Tests auf dem Serverrechner nur die für den jeweiligen WMS erforderliche Software aktiv. Trotz allem können kurzzeitige systembedingte Schwankungen in der Auslastung der verfügbaren Rechnerressourcen (Abbildung 24 *Server*) nicht ausgeschlossen werden. Aus diesem Grund wurden pro Testszenario jeweils 10 Durchläufe gestartet, um den Einfluss von möglichen Ausreißern, welche aus Systemschwankungen resultieren, zu minimieren.

Basierend auf den Ergebnissen der Recherche zu Vector Tiles Formaten und Software in Kap. 7.2 wurden sechs Testszenarien (Quelldaten- und Softwarekombinationen) für *JMeter* entwickelt (Abbildung 24). Aus den Szenarien 1 bis 4 wurden in einem ersten Schritt neun unterschiedliche Varianten für Performance-Tests abgeleitet (Tabelle 8).

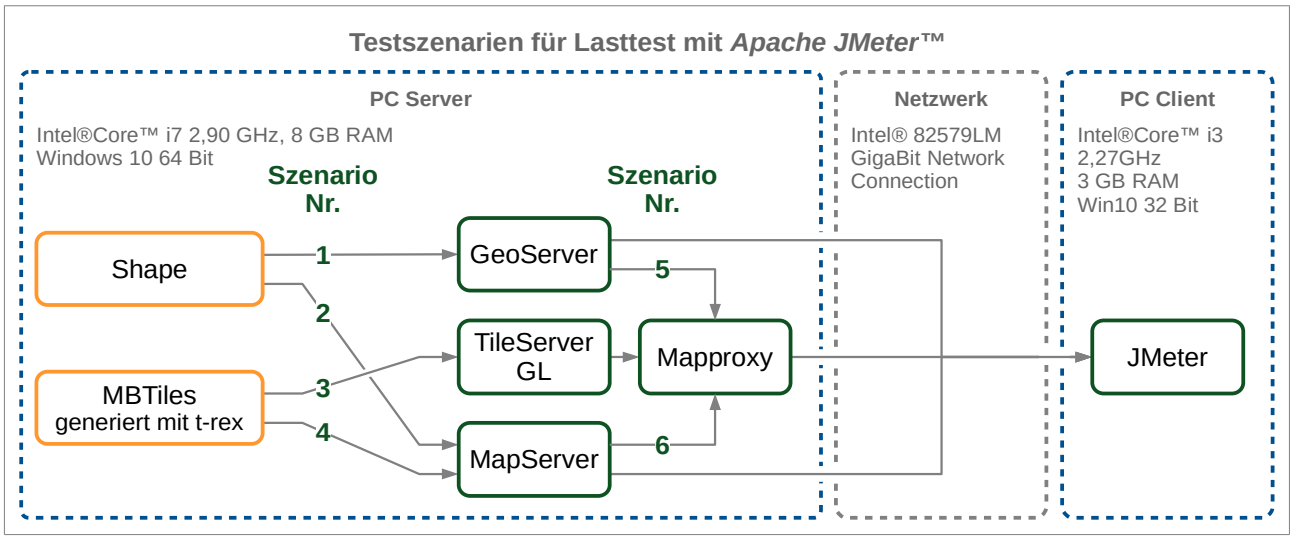


Abbildung 24 Testszenarien für Lasttests mit *Apache JMeter™*

Variante 1 und 2 nutzen *Shape*-Dateien als Quelldatenformat und die Varianten 3 bis 9 nutzen Vector Tiles als Datenquelle und unterscheiden sich in eingesetzter Software, den Größen der vorgenerierten Vector Tiles und der Nutzung des *Cache* von *Mapproxy*. Für jede der Varianten 1 bis 9 wurde ein einfacher Lasttest mit *JMeter* zur Ermittlung der *sample time* durchgeführt. Diese entspricht der *response time* der *INSPIRE*-Vorgaben und damit der Zeit von der Absendung einer Anfrage bis zu deren fertiger Bearbeitung. Dieser erste Test zeigt mögliche Unterschiede in der Reaktionszeit der Webdienste auf. Dafür wurde der Lasttest mit einem *Thread* (1 simulierter Nutzer) konfiguriert. Die hier genutzten Konfigurationen von *JMeter* (abrufbar im *GitHub Repository* dieser Arbeit[4]) basieren auf einem Lasttest des *BKG*, welcher für die einzelnen Varianten

1 http://jmeter.apache.org
2 http://jmeter.apache.org/usermanual/component_reference.html#HTTP_Request
3 https://jmeter-plugins.org/wiki/ParameterizedController/
4 https://github.com/GjueAtGit/VTs_datasource_ogc/tree/master/jmeter/

angepasst wurde. Jeder Test simuliert jeweils eine *GetCapabilities*-Anfrage gefolgt von *GetMap-Requests* in 15 Zoomstufen (0 bis 14) und zwei Bildformaten (*JPG* und *PNG*) im CRS EPSG 3857. Für jede Zoomstufe wird hierbei eine *bounding box* im Zentrum der Ausdehnung des Layers berechnet. Aus der beschriebenen Konfiguration resultieren somit pro Test 310 Einzelwerte bzw. 31 Mittelwerte aus 10 Durchläufen.

Tabelle 8 Varianten der mit *Apache JMeter* durchgeführten Lasttests
Pro Durchlauf ein *GetCapabilities*- gefolgt von *GetMap-Request* in 15 Zoomstufen und zwei Bildformaten (JPG und PNG) im CRS EPSG 3857

Umfang Test		**Datenquelle Service**		**Einstellungen**		**Szenario Nr.** (Abb. 24)	**Variante**
Threads	Loops	Software	Datenformat & ggf. Größe	Cache Raster Tiles	BBox		
1	10	GeoServer	Shape	-	Zentrum des Layers	1	**1**
		MapServer	Shape	-		2	**2**
			Vector Tiles 4096px	-		4	**3**
		Mapproxy	Vector Tiles 4096px	ohne		3	**4**
				gefüllt			**5**
			Vector Tiles 512px	ohne			**6**
				gefüllt			**7**
			Vector Tiles 256px	ohne			**8**
				gefüllt			**9**
5	10	GeoServer	Shape	-	zufällig	1	**10**
		Mapproxy	Vector Tiles 512px	leer		3	**11**
				gefüllt	Zentrum des Layers		**12**
			WMS GeoServer (Shape)			5	**13**
			WMS MapServer (Shape)			6	**14**

Die beschriebenen Tests wurden jeweils dreimal wiederholt, um sicherzustellen, dass die Ergebnisse pro Variante ähnlich sind und somit systembedingte Schwankungen ausgeschlossen werden können. Nach Sicherstellung dieses Punktes wurden jeweils die Ergebnisse des ersten Durchlaufs für die Auswertungen verwendet. Die Einzeldaten und die von *JMeter* ausgegebenen aggregierten Werte der Tests sind ebenfalls im *Github Repository* dieser Arbeit[1] hinterlegt. Die Zusammenfassung der Ergebnisse ist in Tabelle 9 ablesbar und in Abbildung 25 grafisch dargestellt. Die Werte in den Spalten *70 % Line* bis *99 % Line* in Tabelle 9 stellen dar, wie hoch der höchste Wert der *sample time* in einem Bereich von *x %* der niedrigsten Werte zu erwarten ist (= x%-Quantil). Die Boxplots mit dem Bereich in welchem 90 % der niedrigsten Werte liegen, dem Median sowie dem Maximum der *sample time*, sind für die einzelnen Zoomstufen in Anlage 3 aufgeführt. Der Wertebereich von 90 % der Werte wird hier aus den Vorgaben des *Technical Guidance for the implementation of INSPIRE View Services* (IOC Task Force for Network Services 2013) übernommen.

1 https://github.com/GjueAtGit/VTs_datasource_ogc/tree/master/jmeter/results

Tabelle 9 *Sample time* in Millisekunden von WMS der Varianten 1 bis 9 (Tabelle 8) in den Zoomstufen 0 bis 14

Variante	***Sample time*** [ms]							
	Mittelwert	Median	70% Line	90% Line	95% Line	99% Line	Min	Max
1: GeoServer Shapes	158	**44**	86	**594**	831	869	27	**912**
2: MapServer Shapes	178	**106**	134	**497**	602	680	40	**732**
3: MapServer Vector Tiles	8.446	**190**	1.407	**49.370**	56.430	57.752	95	**58.526**
4: Mapproxy (ohne Cache) 4096px	530	**289**	486	**1.338**	1.511	1.748	30	**3.160**
5: Mapproxy (Cache voll) 4096px	95	**97**	106	**124**	129	148	30	**179**
6: Mapproxy (ohne Cache) 512px	443	**301**	438	**884**	1.137	1.520	30	**3.357**
7: Mapproxy (Cache voll) 512px	95	**97**	104	**123**	129	148	31	**166**
8: Mapproxy (ohne Cache) 256px	441	**283**	501	**876**	1.230	1.513	30	**3.040**
9: Mapproxy (Cache voll) 256px	93	**95**	103	**114**	128	139	31	**163**

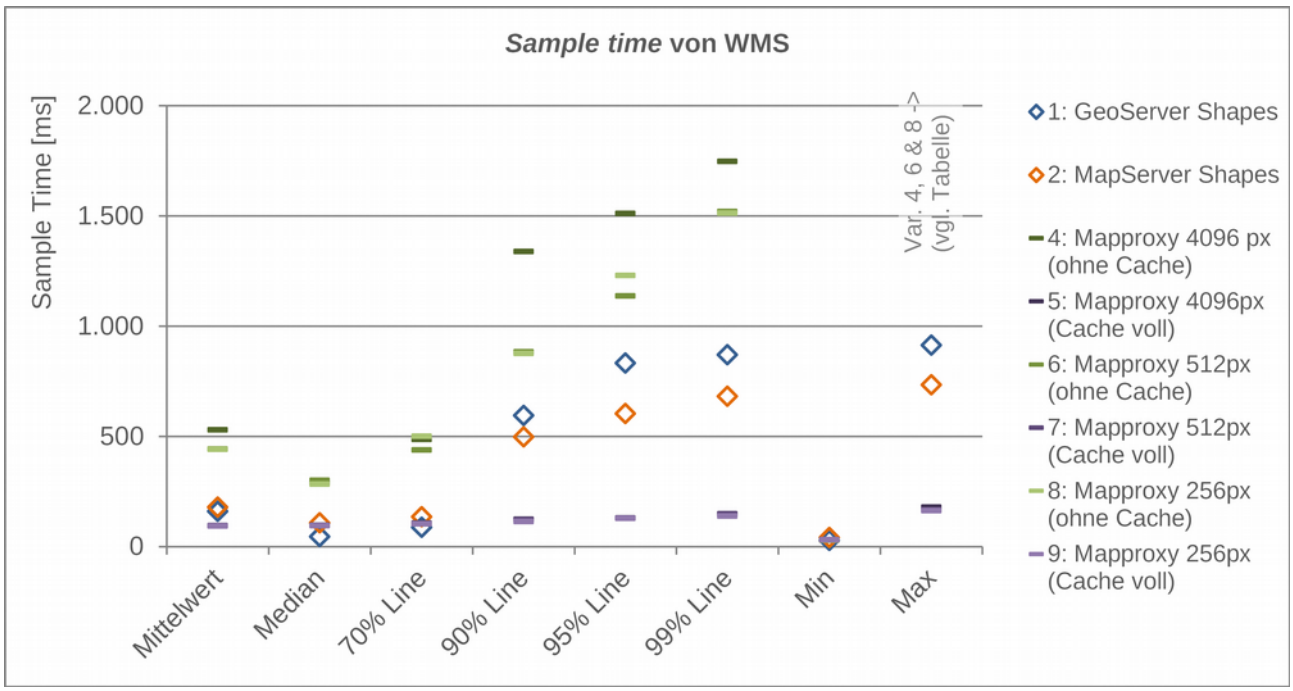


Abbildung 25 *Sample time* von WMS der Varianten 1, 2 und 4 bis 9 (Tabelle 8) in den Zoomstufen 0 bis 14
Variante 3 weicht von den anderen Varianten so stark nach oben ab, dass keine sinnvolle Darstellung möglich ist

Die Zahlen verdeutlichen, dass der WMS basierend auf Vector Tiles ohne vorhandenen *Cache* durchweg am langsamsten ist (Varianten 4, 6 und 8). Beim Zugriff auf den WMS basierend auf Vector Tiles mit gefülltem *Cache* (Varianten 5, 7 und 9) sind die Zugriffszeiten des 90 %-Quantils (Abbildung 26) höher als bei den beiden WMS basierend auf *Shape*-Dateien (Varianten 1 und 2). Jedoch sind der Median und das 70 %-Quantil bei Variante 1 niedriger als bei *Mapproxy* mit gefülltem *Cache* (Abbildung 27). Die Mehrzahl der Anfragen wird demnach bei dem herkömmlichen WMS aus *GeoServer* schneller beantwortet als mit *Mapproxy* und gefülltem Raster Tile *Cache*.

Die zugrundeliegenden Zahlen (vgl. Boxplots der Zoomstufen in Anlage 3) machen deutlich, dass in den Zoomstufen 0 bis 2 die *sample time* bei Anfragen an den WMS in den Varianten 5, 7 und 9 mit Vector Tiles als Datenquelle (*Mapproxy* mit gefülltem *Cache*) niedriger ist als bei Verwendung von *Shape*-Dateien,

sowohl über *GeoServer* als auch über *MapServer* (Variante 1 und 2). Bei Zoomstufe 3 beginnen sich die Werte der s*ample time* dieser fünf Varianten anzugleichen und in den Zoomstufen 4 bis 14 ist die *sample time* bei Anfragen an einen von *GeoServer* bereitgestellten WMS am niedrigsten.

Bei *GeoServer* und *MapServer* sind aufgrund der deutlich höheren *sample time* der Zoomstufen 0 bis 2 die Reaktionszeiten des WMS sehr ungleichmäßig. Dagegen resultieren bei den Varianten 5, 7 und 9 sehr gleichmäßige Reaktionszeiten über alle Zoomstufen. Innerhalb der Zoomstufen sind jeweils alle 10 Werte sehr gleichmäßig, mit niedrigen Standardabweichungen (vgl. Tabellen mit aggregierten Werten im GitHub Repository[1]). Eine nur durch Systemschwankungen erklärbare Ausnahme macht die Anforderung des Kartenbildes im JPG-Format in Zoomstufe 10 bei Variante 1. Hier weichen Maximalwert und Standardabweichung deutlich von allen anderen vergleichbaren Werten ab.

Sehr stark sind die Unterschiede zwischen den Zoomstufen auch bei den Varianten 4, 6 und 8 (*Mapproxy* ohne *Cache*) und auch die Schwankungen der Werte in den Zoomstufen (vgl. Boxplots Anlage 3 und Standardabweichungen) sind hoch. Die Unterschiede zwischen den Zoomstufen 5 bis 14 sind geringer als im Vergleich zu den niedrigeren Zoomstufen. Hier gleichen sich also die Zugriffszeiten einander an, schwanken aber insgesamt stärker als bei den vorher beschriebenen Varianten. Der beschriebene Verlauf der Werte ist z. B. im Vergleich der Mediane der *sample time* der jeweiligen WMS in den 15 Zoomstufen in Abbildung 28 ablesbar.

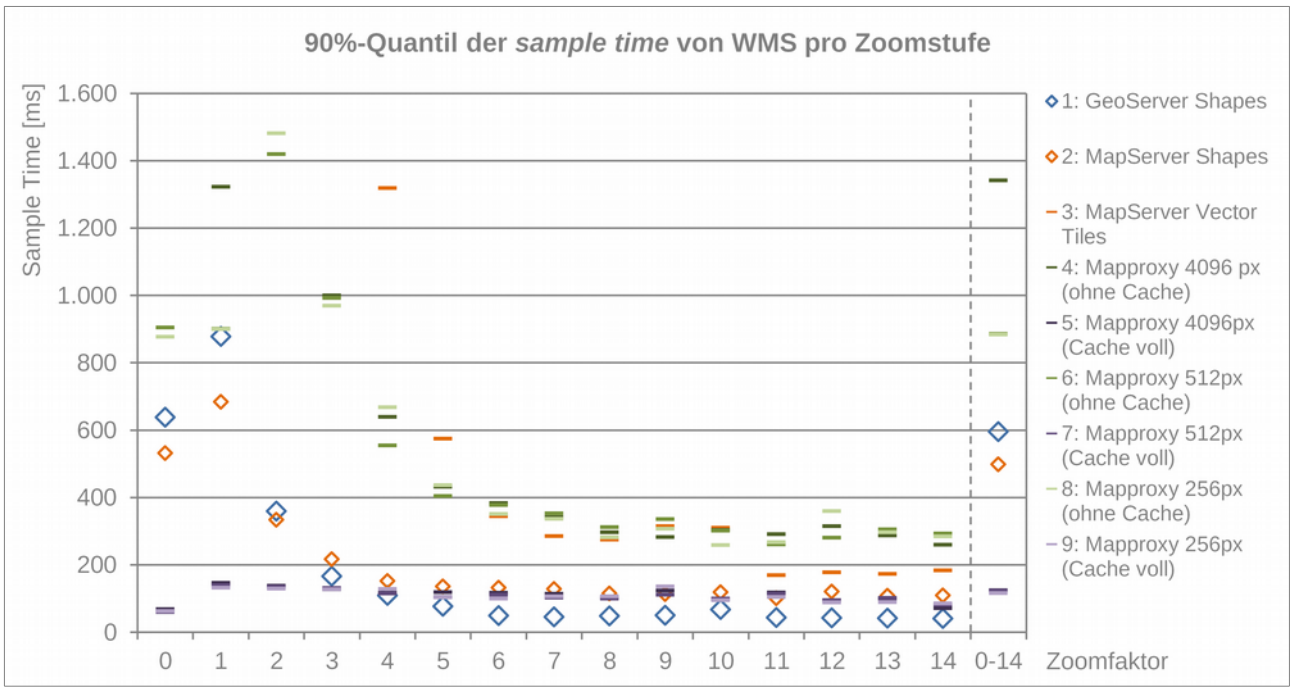


Abbildung 26 90 %-Quantil der *sample time* bei Anfragen an WMS der Varianten 1 bis 9 (Tabelle 8)

Eine Sonderstellung bei diesen ersten Tests nimmt Variante 3 ein, bei welcher der WMS in *MapServer* Daten von Vector Tiles aus einer *MBTiles*-Datenbank bezieht. Hier sind die Werte der *sample time* in den Zoomstufen 0 bis 5 um ein Vielfaches höher als die der anderen Varianten. Erst ab Zoomstufe 5 kommen die Werte in einen Bereich, der den Varianten 4, 6 und 8 vergleichbar ist. Sie liegen jedoch immer über den Werten der WMS basierend auf *Shape*-Dateien und *Mapproxy* mit gefülltem *Cache*. Die Ursache für die Extremwerte ist darin zu suchen, dass in der aktuellen Version die jeweilige Zoomstufe von *MapServer* noch nicht an den

1 https://github.com/GjueAtGit/VTs_datasource_ogc/tree/master/jmeter/results

GDAL-Treiber weitergegeben werden kann, so dass immer die Vector Tiles der niedrigsten Zoomstufe (123.561 Tiles in Zoom 14) durchsucht und z. B. in Zoom 0 bis 4 zu einer Bildkachel und in Zoom 5 zu 4 Bildkacheln gerendert werden. Dies zieht einen hohen serverseitigen Rechenaufwand nach sich. Die Vermutung, dass aus der direkten Nutzung von Vector Tiles als Datengrundlage für einen WMS mit *MapServer* die niedrigste *sample time* resultiert, konnte auch für die höheren Zoomstufen inkl. Zoom 14 nicht bestätigt werden. Die Werte liegen beim Median in den Zoomstufen 6 bis 14 jedoch unter denen der vergleichbaren WMS basierend auf Vector Tiles über *Mapproxy* mit leerem *Cache*.

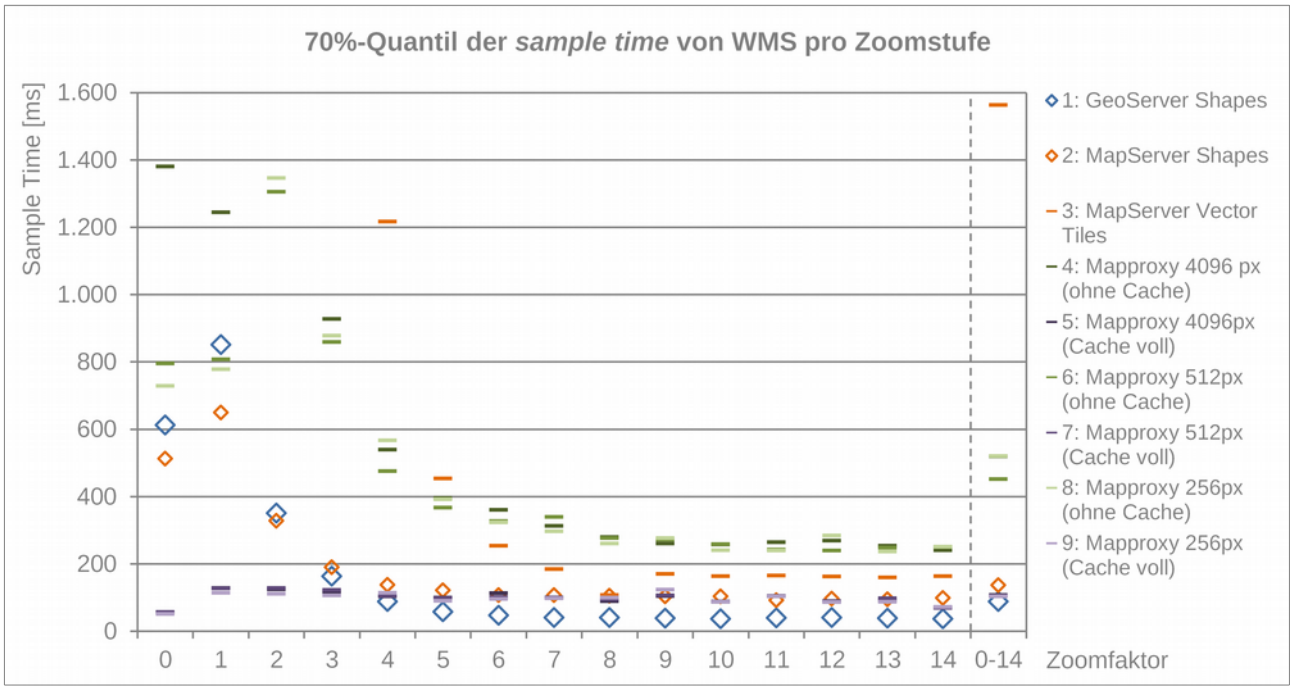


Abbildung 27 70%-Quantil der *sample time* bei Anfragen an WMS der Varianten 1 bis 9 (Tabelle 8)

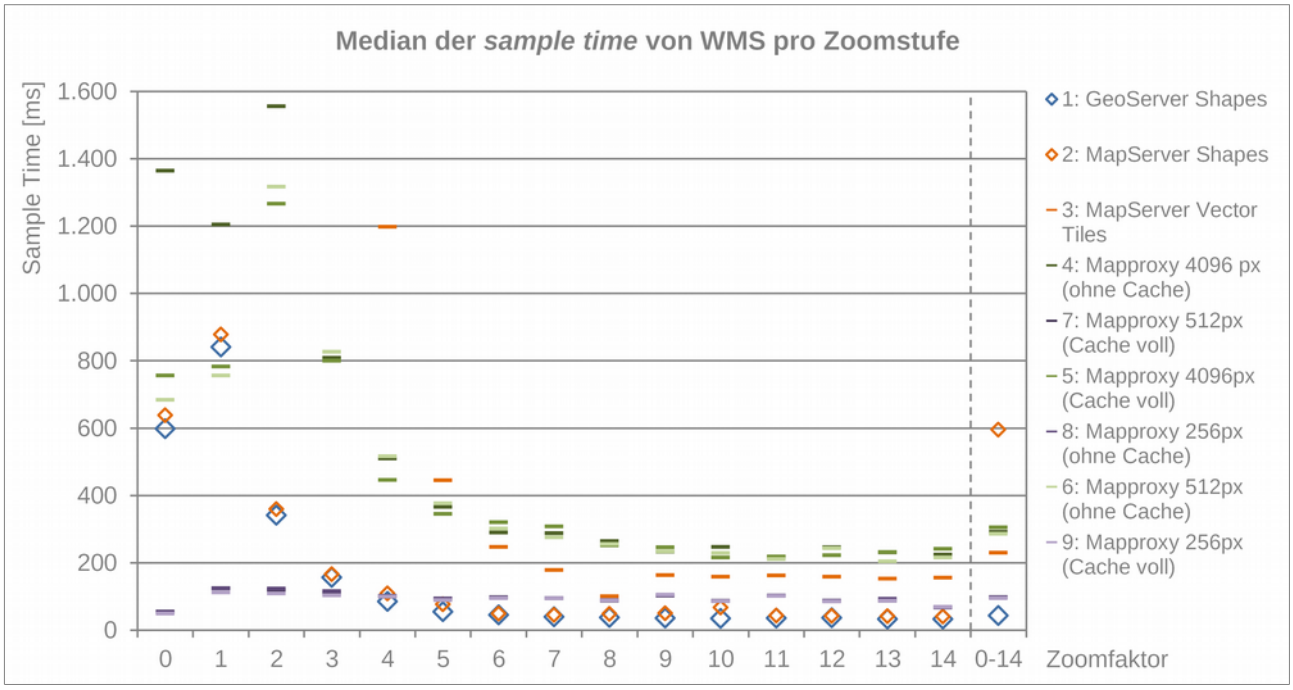


Abbildung 28 Median der *sample time* bei Anfragen an WMS der Varianten 1 bis 9 (Tabelle 8)

Basierend auf den Erkenntnissen der Lasttests mit den Varianten 1 bis 9 wurden fünf weiterführende Lasttests abgeleitet. Hierbei sollen mehrere Nutzer (*threads*) simuliert werden, um zu ermitteln, ob sich die Ergebnisse, welche bei den Varianten 1 bis 9 ermittelt wurden, bestätigen. In Kombination mit 10 *loops* ergeben sich bei 5 *threads*, 15 Zoomstufen und zwei Bildformaten pro Variante 1.550 Anfragen an den Service.

Darüber hinaus soll bei einem Teil der neuen Varianten die *bounding box der GetMap*-Anfrage nicht um den Mittelpunkt der Layerausdehnung, sondern zufällig generiert werden, um Vergleichsdaten zu einer echten produktiven Umgebung zu erhalten. Hierbei wurde der mögliche Bereich für den Zufallsgenerator so eingeschränkt, dass die resultierende *bounding box* immer vollständig im Areal der Ausdehnung des Layers liegt. Somit wird vermieden, dass in den niedrigen Zoomstufen fast leere Kartenausschnitte (*bounding box* direkt am Rand der Layerausdehnung) zurückgegeben werden. Statt 256 unterschiedlichen Rasterkacheln bei Variante 5, 7 und 9 sind nach einem deratigen Lasttest im *Cache* von *Mapproxy* über 6.200 Einzelkacheln abgelegt, was den höheren Aufwand für das Rendern bestätigt. Das Verhalten dieser Varianten ähnelt damit dem der Varianten 4, 6 und 8 ohne Nutzung des *Cache* von *Mapproxy*.

Dieser veränderte Lasttest mit zufälliger *bounding box* wurde für die Varianten 1 und 6 als neue Varianten 10 und 11 umgesetzt (Tabelle 8). Als Vergleich zu Variante 11 wurde aus Variante 7 die Variante 12 erstellt, die einen WMS basierend auf Vector Tiles mit bereits gefülltem Raster Tile Cache in *Mapproxy* und mehreren Nutzern simuliert. Zusätzlich zur *sample time* aus den ersten Tests wurden bei diesen Varianten noch der Durchsatz (*troughput*) und die übertragene Datenmenge ausgewertet. Die zusammenfassenden Ergebnisse sind in Tabelle 10 und Abbildung 29 dargestellt. Um zu belegen, inwieweit ein vorhandener *Cache* auch Anfragen an einen WMS basierend auf *Shape*-Dateien beschleunigt, wurden zusätzlich Tests zu den Varianten 13 und 14 durchgeführt, in denen *Mapproxy* jeweils einen WMS basierend auf Shape-Dateien als Datenquelle für einen WMS nutzt und einen *Cache* mit Rasterkacheln von 256x256 Pixeln Größe (*default*) erzeugt.

Tabelle 10 *Sample time* in Millisekunden, *throughput* und übermittelte Datenmenge pro Sekunde (*received*) von WMS der Varianten 10 bis 14 (Tabelle 8), 5 *treads*, 10 *loops*, Zoomstufen 0 bis 14

Variante	***Sample Time*** [ms]								**Through-put/s**	**Received** [KB/s]
	Mittel-wert	Median	70% Line	90% Line	95% Line	Min	Max	Standd.-abweich.		
10: GeoServer	500	105	210	1.951	2.635	35	3.355	812,18	9,57	579,86
11: Mapproxy	731	678	1.010	1.398	1.611	38	3.548	528,66	6,62	292,61
12: Mapproxy Cache	243	246	271	311	336	63	409	55,88	19,38	806,17
13: GeoServer Mapproxy	210	218	238	263	275	32	328	49,02	22,27	984,78
14: MapServer Mapproxy	215	223	242	268	280	41	372	49,35	21,83	776,92

Die Zahlen geben ein vergleichbares Bild wieder, wie es sich in den vorhergehenden Tests abgezeichnet hat. Sie verdeutlichen, dass die ausgeglichensten und, abgesehen vom Median und dem 70 %-Quantil, auch niedrigsten Werte der *sample time* bei Variante 12, 13 und 14, also unabhängig von der Datenquelle bei gefülltem Raster Tile Cache, zu verzeichnen sind. Der Durchsatz und die pro Sekunde übertragene Datenmenge sind bei diesen Varianten ebenfalls am höchsten. Alle drei Varianten gleichen sich in allen Werten stark. Ein gefüllter *Cache* erhöht demnach die Performance des WMS insgesamt, da das Rendern *on demand* entfällt. Nach den INSPIRE-Vorgaben wären dies die optimaleren Varianten. Jedoch ist auch hier wieder der Effekt zu beobachten, dass beim direkten Zugriff auf einen WMS aus GeoServer die Zugriffszeiten bei der Mehrzahl der Anfragen (70 %-Quantil) niedriger sind, als bei den Varianten mit Umweg über *Mapproxy* und

gefülltem *Cache*. Hier wären direkte Vergleiche von WMS und WMTS aus *GeoServer* für genauere Aussagen erforderlich.

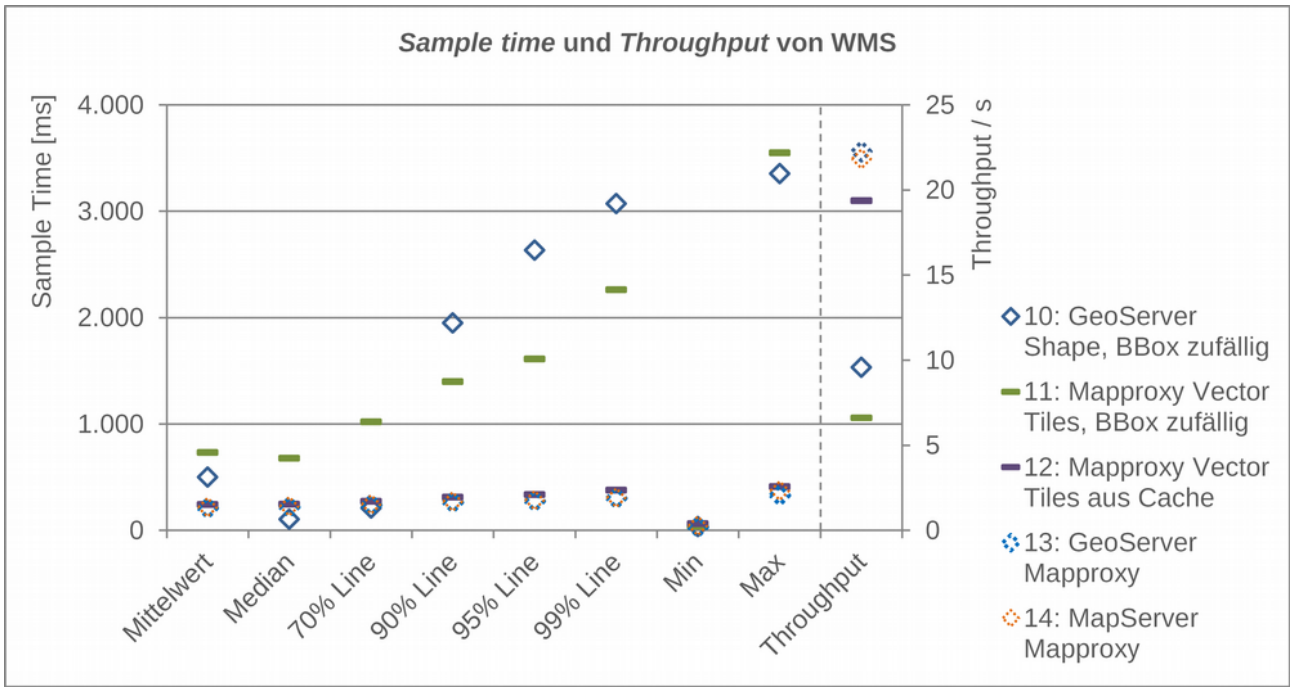


Abbildung 29 *Sample time* und *troughput* von WMS der Varianten 10 bis 14 (Tabelle 8) mit 5 threads in den Zoomstufen 0 bis 14

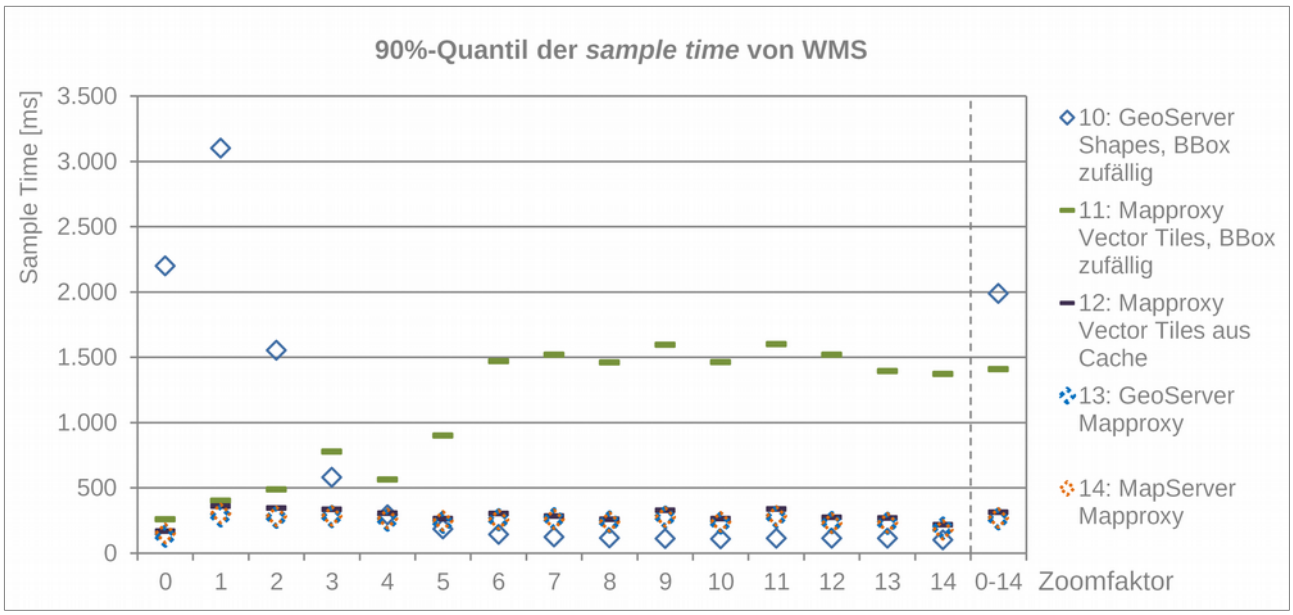


Abbildung 30 90 %-Quantil der *sample time* bei Anfragen an WMS der Varianten 10 bis 14 (Tabelle 8)

Bei der entsprechend den *INSPIRE*-Vorgaben entscheidenden 90 % Line (Abbildung 30) ist die maximale Zugriffszeit auf Daten über alle Zoomstufen bei Variante 11 mit Vector Tiles als Datengrundlage und zufälliger *bounding box* niedriger als bei Nutzung eines WMS basierend auf *Shape*-Dateien (Variante 10), also beide Male mit Rendern der Kartenbilder *on demand*. Dagegen sind der Mittelwert in Variante 10 etwas und der Median und das 70 %-Quantil der Zugriffszeit deutlich höher ist als in Variante 10. Bei den Varianten mit

Nutzung vorgerenderter Raster Tiles (12 bis 14) ist über alle Zoomstufen hinweg der Zugriff bei Betrachtung von 90 % aller Anfragen ca. 7-mal so schnell als bei Variante 10, wogegen auch hier die Werte von 70 %-Quantil (gleich schnell) und Median (~1/2-mal so schnell) ein anderes Bild vermitteln. Diese Unterschiede werden bedingt durch die deutlich höheren Zugriffszeiten in Variante 10 bei den Zoomstufen 0 bis 2, welche den Mittelwert und das 90 %-Quantil erhöhen, den Median und das 70 %-Quantil jedoch nicht beeinflussen (Abbildung 31 bis Abbildung 33). Im Umkehrschluss bedeutet dies, dass bei Variante 10 die Zugriffszeit bei der Mehrzahl der Anfragen am niedrigsten ist.

Analog zur *sample time* stellt sich die Situation im Vergleich von Durchsatz (*throughput*) und übertragener Datenmenge pro Sekunde (*received*) dar (Tabelle 10 und Abbildung 29). Hier sind die Werte bei vorgerenderten Raster Tiles unabhängig von der Datenquelle am höchsten und weisen somit die höchste Performance auf. Die Werte von Variante 10 sind höher als die von Variante 11, was wiederum den Aussagen zum Median der *sample time* entspricht. Dies betrifft sowohl die Werte über alle Zoomstufen in Tabelle 10 als auch die Einzelwerte der Zoomstufen[1].

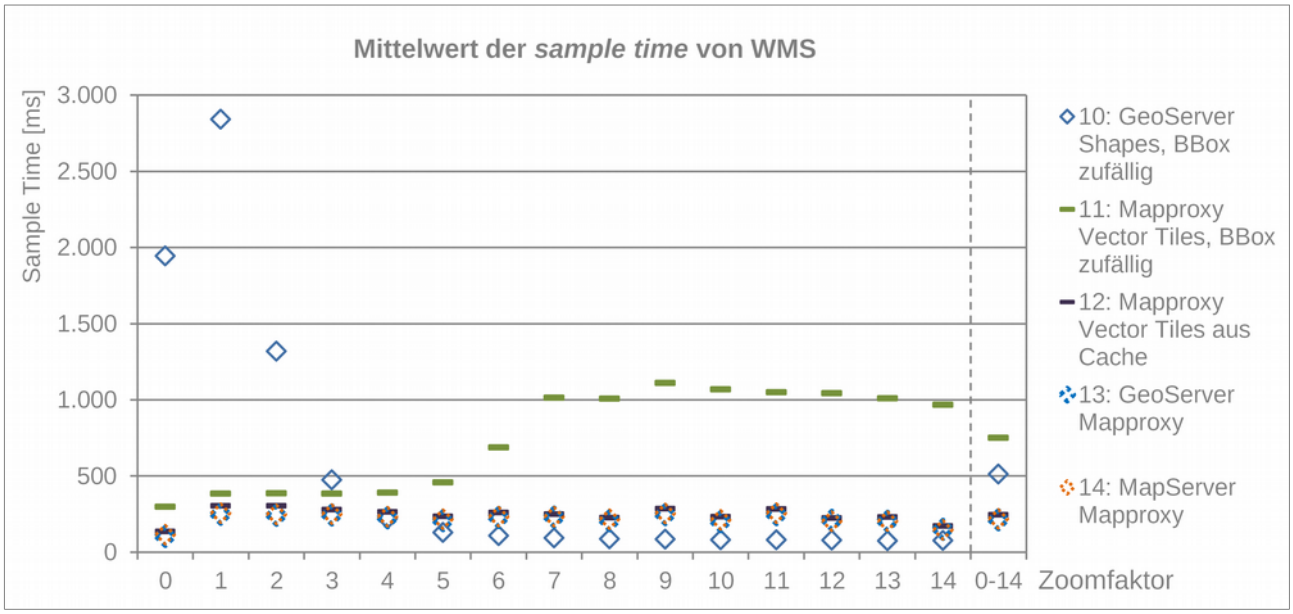


Abbildung 31 Mittelwert der *sample time* bei Anfragen an WMS der Varianten 10 bis 14 (Tabelle 8)

1 https://github.com/GjueAtGit/VTs_datasource_ogc/tree/master/jmeter/results

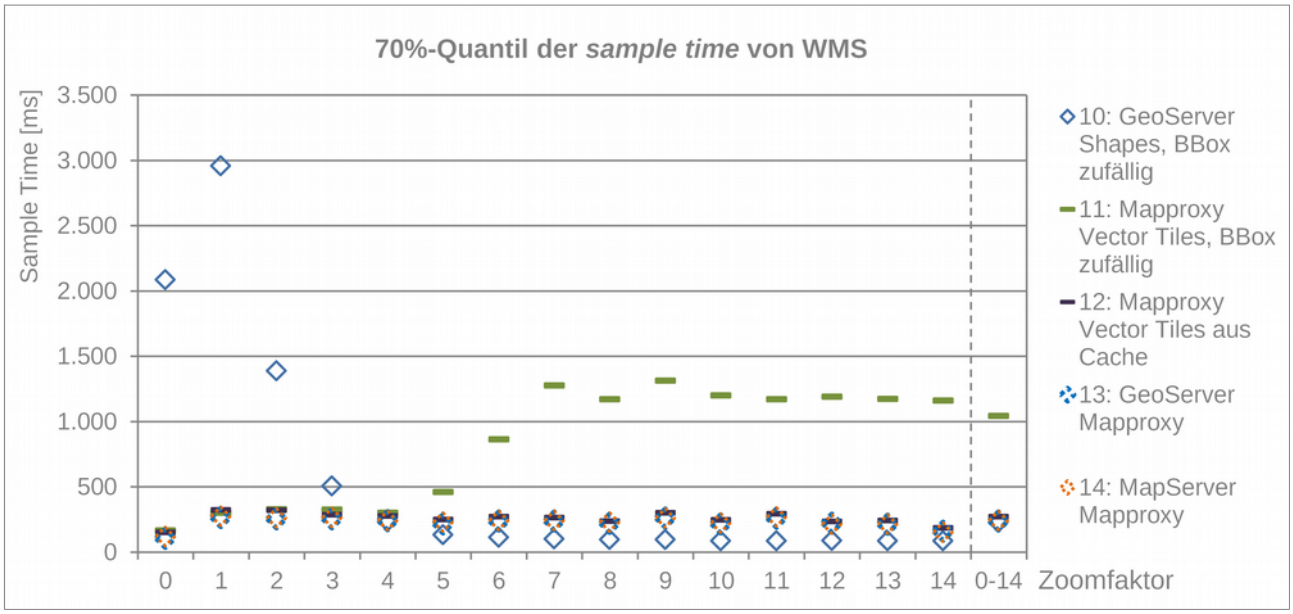


Abbildung 32 70 %-Quantil der *sample time* bei Anfragen an WMS der Varianten 10 bis 14 (Tabelle 8)

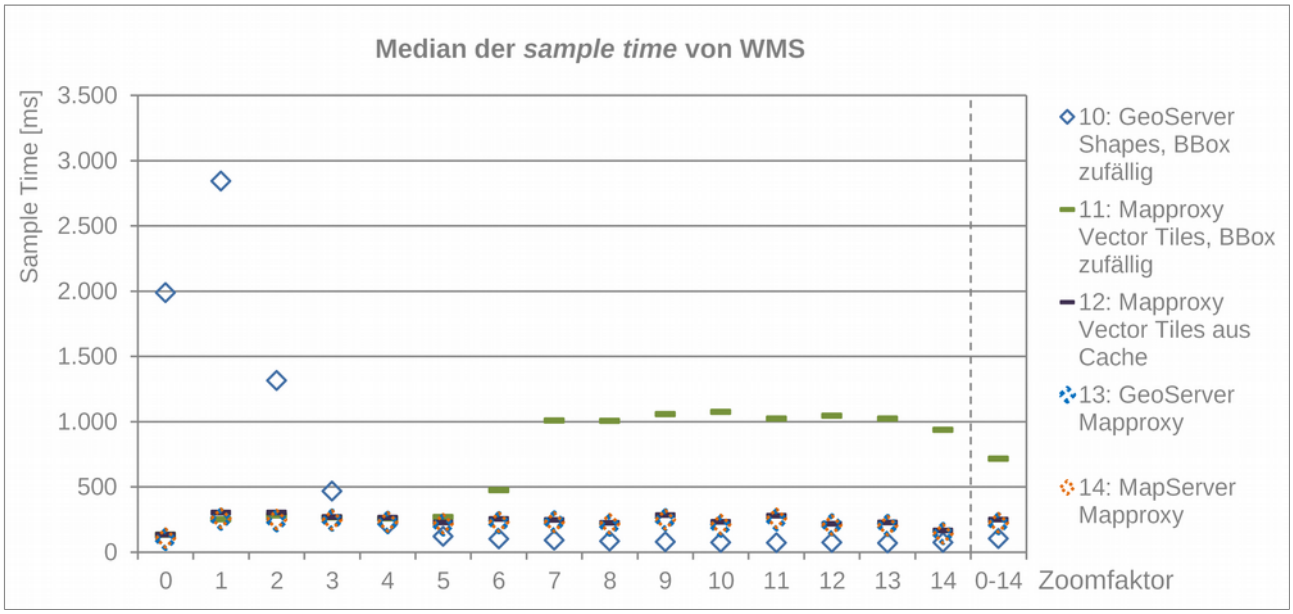


Abbildung 33 Mediane der *sample time* bei Anfragen an WMS derVarianten 10 bis 14 (Tabelle 8)

8 Bewertung und Diskussion

Die Ergebnisse der Recherche von Datenformaten und Software (Kap. 6) haben ergeben, dass es aktuell keine voll funktionsfähige Lösung zur direkten Nutzung von Vector Tiles als Datengrundlage für einen WMS gibt. In fast allen recherchierten, theoretisch möglichen Szenarien (Abbildung 18) ist der Umweg über Raster Tiles erforderlich. Die derzeit einzig mögliche direkte Einbindung von Vector Tiles in einen WMS mit *MapServer* über die *GDAL*-Bibliothek ist noch nicht praxistauglich umgesetzt. Das liegt vor allem daran, dass immer die Vector Tiles der höchsten Zoomstufe abgefragt werden, was besonders in den niedrigen Zoomstufen des WMS zu extrem langen Wartezeiten führt (Tabelle 9). Darüber hinaus ist das *Clippen* der *Buffer* um die Tiles nicht steuerbar und eine Funktion zum Zusammenführen von Tile-übergreifenden Features (*Merge* bzw. *Dissolve*) fehlt. Einschränkungen erfährt diese Kombination auch durch die Beschränkung von Vector Tiles in *MBTiles*-Datenbanken auf EPSG 3857. Die dahingehend flexibleren Tiles in einer Verzeichnisstruktur konnten mit den aktuellen Versionen von *MapServer* und *GDAL* nicht erfolgreich eingebunden werden.

Basierend auf den Recherchen wurden Softwarekonstellationen abgeleitet, die eine Nutzung von Vector Tiles als Datengrundlage zum Rendern von Raster Tiles und deren Verwendung für einen WMS ermöglichen. Mit diesen Erkenntnissen wurde ein Szenario praktisch umgesetzt. Hierbei werden mit *t-rex* aus *Shape*-Dateien Vector Tiles generiert und mit dem Tool *mbutil* in einer *MBTiles*-Datenbank abgelegt. Diese *MBTiles* werden von *TileServer GL* als Datenquelle zum Rendern von Raster Tiles genutzt, welche in *Mapproxy* zur Bereitstellung eines WMS eingebunden werden. Der beschriebenen Konstellation von *t-rex*, *TileServer GL* und *Mapproxy* wurde in Performancetests je ein WMS mit *Shape*-Dateien als Quelldaten in *GeoServer* und *MapServer* gegenübergestellt.

Die bei den Lasttests mit *Apache JMeter* ermittelten Zahlen (Kap. 7.3.2) lassen einen Vergleich der aus den beschriebenen Szenarien abgeleiteten Varianten zu (Tabelle 8). Dabei bestimmt die Schwerpunktsetzung bei der Interpretation der Werte das Ergebnis. Bei der Betrachtung der Zahlen ergeben sich aus Sicht des Autors zwei wesentliche Interpretationsmöglichkeiten bzw. Schwerpunktsetzungen:

1. Entscheidend ist der Maximalwert des Bereichs, in welchem 90 % der günstigsten Werte liegen (Minimum bis 90 % Quantil) - entspricht den *INSPIRE*-Vorgaben.
2. Entscheidend ist der Bereich, in dem die Mehrzahl der günstigsten Werte auftritt (hier z. B. 70 %-Quantil).

Anhand der im Rahmen dieser Arbeit ermittelten Werten kann abgelesen werden, dass z. B. im Vergleich von Variante 1 mit Variante 7 (Tabelle 8) die *sample time* des 90 %-Quantils 4,8-mal höher ist. Die daraus abgeleitete Aussage würde beinhalten, dass die Softwarekonstellation, auf welcher Variante 7 basiert, eine höhere Performance bietet als der vergleichbare WMS mit originären Vektordaten als Quelle (Variante 1). Dagegen beträgt der Faktor beim 75 %-Quantil nur noch 1,2 und schon beim 70 %-Quantil hat sich das Verhältnis der beiden Varianten umgekehrt und beim Median beträgt die *sample time* von Variante 7 sogar das Doppelte von Variante 1. Die Hälfte der Anfragen von Variante 1 ist also doppelt so schnell abgearbeitet als bei Variante 7, was die umgekehrte Schlussfolgerung erfordern würde. Entsprechend unterschiedlich kann mit den im Rahmen dieser Arbeit ermittelten Zahlen die Frage nach einem Performancegewinn bei der Nutzung von Vector Tiles als Datengrundlage für einen WMS beantwortet werden. Legt man die erste Interpretationsmöglichkeit als entscheidend fest, ergibt sich mit Vector Tiles als Datengrundlage in der Softwarekombination von *TileServer GL* und *Mapproxy* eine Beschleunigung der Zugriffe bei 90 % der Anfragen an den WMS. Dagegen ist bei über 70 % der Anfragen an den Webservice die Zugriffszeit über *GeoServer* basierend

auf *Shape*-Dateien niedriger. In der Praxis bedeutet dies, dass der Nutzer bei den kleineren Zoomstufen bei Variante 1 länger auf das Ergebnis warten muss, aber bei Anfragen in den Zoomstufen größer 4 die Darstellung der Karte schneller erfolgt.

Bei identischer Datengrundlage unterscheiden sich die Werte der Zugriffszeiten je nach eingesetzter Software. Die Unterschiede zwischen Variante 1 und 2, welche auf derselben *Shape*-Datenquelle basieren, machen deutlich, wie sehr sich unterschiedliche Software auf die Geschwindigkeit der Verarbeitung auswirken kann. Einerseits wurde hier zwar nicht bis ins Detail untersucht, wie die Performance durch softwareseitige Einstellungen verbessert werden kann. Andererseits zeigen die Werte der beiden Varianten jedoch, dass nicht abschließend zu klären ist, inwieweit die Gründe für die Unterschiede zwischen den Werten von Variante 1 und den Varianten 4 bis 9 eher softwareseitig zu suchen sind, als dass sie durch die unterschiedliche Datenquelle bedingt werden. Der Vergleich von Variante 2 und Variante 3, welche beide dieselbe Software mit unterschiedlicher Datenquelle nutzen, ist hier auch nicht aussagekräftig. Der Zugriff auf die Vector Tiles bei Variante 3 über die *GDAL*-Bibliothek in *MapServer* ist, wie beschrieben, derzeit nur unzureichend umgesetzt. Besser interpretierbare Werte sind hier zu erwarten, wenn Vector Tiles z. B. auch von *GeoServer* als Datenquelle genutzt werden können bzw. in *MapServer* besser eingebunden sind. Demnach muss die Antwort auf die zentrale Frage dieser Arbeit eher lauten: Mit der hier untersuchten Kombination von Eingangsdatenformat und Software ist eine Performancesteigerung je nach Schwerpunkt bei der Interpretation der Werte zu erwarten oder nicht zu erwarten.

Neben der Entscheidung für Schwerpunkt 1. oder 2. bei der Interpretation der Ergebnisse ist mit gleicher Auswirkung auf die Antwort die willkürliche Festlegung beim Lasttest zu sehen, bei jeder Zoomstufe dieselbe Anzahl von Anfragen an den Service zu schicken. Untersuchungen zum Nutzerverhalten bei Kartendiensten wurden im Rahmen dieser Arbeit nicht durchgeführt. Aus eigener Beobachtung des Autors kann jedoch davon ausgegangen werden, dass in den höheren Zoomstufen mehr Nutzeranfragen erfolgen, als in den niedrigeren. Dies ist z. B. bedingt durch Verschieben des Kartenausschnitts und damit Anfordern weiterer Kacheln in der aktuellen Zoomstufe, ohne noch einmal in eine niedrigere Zoomstufe zu wechseln. Die Folge wäre eine Verschiebung z. B. des 90 %-Quantils in Richtung der Zugriffszeiten von Zoomstufe 5 und größer und würde so eine Interpretation nach sich ziehen, die grundsätzlich den Schlussfolgerungen der aktuellen Zahlen bei Schwerpunktsetzung 2. entspricht.

Neben der Feststellung, dass im Rahmen dieser Arbeit nicht eindeutig zu klären war, ob die Unterschiede zwischen den einzelnen Varianten eher softwareseitig, als beim Quelldatenformat zu suchen sind, wäre beim Einsatz identischer Software und vergleichbarem Ergebnis zu untersuchen, inwieweit der Aufwand für das Zusammenfügen von Features aus mehreren Vector Tiles den vermuteten Vorteil durch unaufwändigeres *Clippen* von teilweise großen Features bei originären Vektordaten zunichte macht. Darüber hinaus wäre zu klären, inwieweit der Service die bereits stattgefundene Generalisierung von Vector Tiles in den kleineren Zoomstufen berücksichtigt und damit tatsächlich, wie in der Einleitung vermutet, eine Beschleunigung zu erwarten ist.

Mit *JMeter* wurde untersucht, wie sich ein vorhandener Raster Tile *Cache* auf die Zugriffsgeschwindigkeit des Services auswirkt. Diese ist bei ansonsten identischer Konfiguration des Dienstes erwartungsgemäß insgesamt niedriger und damit der Service schneller, als beim Rendern *on demand*. Mit dem Tool *mapproxy-seed* kann der *Cache* von *Mapproxy* gefüllt werden. Das Tool ermöglicht das *Cachen* der Kacheln für das komplette Gebiet oder für ausgewählte (z. B. häufig genutzte) Bereiche. Dabei und auch beim Generieren der Vector Tiles ist die Update-Häufigkeit der Daten zu berücksichtigen. In den Zoomstufen ≥ 5 ist ist die *sample time* bei direkten Anfragen an *GeoServer* basierend auf *Shape*-Dateien schneller als bei Einbindung

dieses Dienstes in *Mapproxy* und Anfragen basierend auf dessen vorgeneriertem *Cache*. Die Gründe für dieses unerwartete Verhalten wurden hier nicht näher untersucht.

Eine Einschätzung der mit den Lasttests ermittelten Werte im Vergleich zu den Vorgaben der INSPIRE-Richtlinie ist nur bedingt möglich. Grundsätzlich liegen alle Werte der *sample time* im Rahmen der Vorgabe des *Technical Guidance for the implementation of INSPIRE View Services* (IOC Task Force for Network Services 2013):

> *„For a 470 Kilobytes image (e.g. 800 x 600 pixels with a colour depth of 8 bits), the response time for sending the initial response to a Get Map Request to a view service shall be maximum 5 seconds in normal situation. ... Normal situation represents periods out of peak load. It is set at 90% of the time.“*
>
> ***übersetzt:***
>
> *„Für ein Bild mit 470 Kilobyte (z. B. 800 × 600 Pixel mit einer Farbtiefe von 8 Bit) beträgt die Antwortzeit für das Senden eines ersten Ergebnisses auf eine Get Map-Anfrage an einen Darstellungsdienst in einer normalen Situation höchstens 5 Sekunden. ... Mit einer normalen Situation ist ein Zeitraum ohne Spitzenbelastung gemeint. Eine normale Situation ist 90 % der Zeit gegeben.“*

Die untersuchten Softwarekonstellationen der Webservices sind jedoch nicht in einer echten Serverumgebung installiert und wurden auch nicht über das Internet abgerufen, sondern in einer Heimnetzumgebung mit gleichmäßig sehr hohen Übertragungsgeschwindigkeiten. Zudem wurde, in Anlehnung an die Vergleichstests beim *BKG*, nur ein 512 x 512 Pixel großer Kartenausschnitt angefragt. Testweise wurden zu Vergleichszwecken mit dem Lasttest von Variante 1 (*GeoServer* basierend auf *Shape*-Dateien) auch Durchläufe mit 1.024 x 1.024 und 2.048 x 2.048 Pixel großen *GetMap*-Anfragen durchgeführt. Hier erhöhte sich der Wert der 90% Line um ca. 55 % bzw. 105 %.

Über diese Einschränkung hinaus hinterfragen Seip & Bill (2016) die ausschließliche Verwendung von Ergebnissen aus *JMeter* kritisch:

> *„However, this reflects only the measurements of one tool (JMeter), thus there is no comparison questioning the accuracy of the results.“*
>
> ***übersetzt:***
>
> *„Dies spiegelt jedoch nur die Messungen eines Werkzeugs (JMeter) wider, so dass es keinen Vergleich gibt, der die Genauigkeit der Ergebnisse in Frage stellt.“*

Die Vorteile des Verpackens von Vector Tiles in *MBTiles*-Datenbanken liegen im kompakten Datenformat und in der Möglichkeit der Indizierung von Inhalten. Ersteres erleichtert die direkte Weitergabe von Daten und bietet somit dem Anbieter von Kartendiensten gute Potenziale zur Vereinfachung der erforderlichen Prozesse (Thalheim, *BGK* schriftlich 16.11.2018). Im Rahmen dieser Arbeit war es aus den in Kap. 7.2.1 dargestellten Gründen nicht möglich zu untersuchen, wie sich Vector Tiles in einer Dateistruktur im Vergleich zu in *MBTiles* verpackten Kacheln bei gleicher Softwarekombination auf die Performance auswirken. Cavazzi (2018) weist darauf hin, dass durch die zum Selektieren einzelner Tiles aus einer Datenbank erforderlichen Extraktionsmechanismen, Performanceeinbußen gegenüber dem direkten Zugriff von Kacheln in einer Verzeichnisstruktur zu erwarten sind. Bei direktem Zugriff auf Vector Tiles in einer Ordnerstruktur wäre somit eine Reduktion der Zugriffszeiten zu erwarten. Die Dauer von Abfrage von Attributinformationen wäre mit Blick auf die mögliche Indizierung in *MBTiles*-Datenbanken noch zu untersuchen. Weitere mögliche Fälle, bei denen Anfragen an den Dienst durch eine Indizierung beschleunigt würden, wären für eine abschließende Einschätzung ebenfalls zu recherchieren. Es ist jedoch davon auszugehen, dass die Ablage von Vector Tiles in Ordnern mit Blick auf das zentrale Ziel dieser Arbeit insgesamt performanter ist. Darüber hinaus wäre

diese Lösung mit Blick auf die derzeitige Beschränkung von *MBTiles* auf ein CRS flexibler. Das parallele Vorhalten der Vector Tiles in *MBTiles*-Datenbanken zur Weitergabe neben Daten in einer Verzeichnisstruktur wäre mit Blick auf die oben festgestellten Zeiten zum Verpacken mit *mbutil* und den zusätzlichen Speicherbedarf unaufwändig.

Eine intensive Untersuchung, inwieweit durch die Veränderung der Kombination oder von Parametern der verwendeten Software eine Verbesserung der Performance möglich ist, konnte aufgrund der großen Zahl eingesetzter Anwendungen nicht vorgenommen werden. Hier sind, wenn eine direkte, fehlerfreie Einbindung von Vector Tiles in einen WMS möglich ist, spezifische Analysen erforderlich. Bei den beschriebenen Tests wurden, wenn nicht anders dargestellt, die Standardeinstellungen der Software beibehalten. So legt *Mapproxy* im Cache in der Standardeinstellung 256 x 256 Pixel große Rastertiles ab. Im Vergleich zu größeren Kacheln erfordert dies mehr Einzelbilder bei einer *GetMap*-Anfrage an den WMS und damit für *Mapproxy* einen höheren Aufwand beim Zusammensetzen des Kartenbildes. Je nach Anwendungsfall könnte hier die Verwendung größerer Bildkacheln den Dienst zu beschleunigen.

Sehr unterschiedlich wird die Generierung von Vector Tiles gehandhabt. Die hier näher untersuchten Fälle mit *GeoServer* und *t-rex* liefern bei identischen Eingangsdaten und Vector Tile Format teilweise stark abweichende Ergebnisse (Kap. 7.3.1). Was die Darstellung der Geometrien beim Client betrifft sind diese Unterschiede mit Blick auf das zentrale Ziel dieser Arbeit unwesentlich. Dagegen sind die mit *t-rex* generierten, vom Inhalt her überladenen und damit vom Speicherbedarf viel zu großen Vector Tiles der kleinen Zoomstufen problematisch mit Blick auf die Performance des Dienstes. Zur Nachvollziehbarkeit fehlen klare Vorgaben für die Generierung bzw. entsprechende Informationen in den Tiles darüber, mit welchen Parametern sie erzeugt wurden. Die Dokumentationen der einzelnen Softwarelösungen (über die beiden genannten hinaus) sind dahingehend oft nicht ausreichend.

Die Generierungszeiten für Vector Tiles sind bei gleichem Zieldatenformat je nach verwendeter Software sehr unterschiedlich (Tabelle 5). Bei effizienterer Vereinfachung der Geometrien mit *t-rex* in zukünftigen Versionen ist es möglich, dass der Zeitaufwand beim reinen Schreiben der Vector Tiles in den kleinen Zoomstufen reduziert wird, was den Unterschied zur aktuellen Herangehensweise von *GeoServer* noch vergrößern könnte. Jedoch bleibt hier abzuwarten, inwieweit aufwändige Generalisierungsalgorithmen den Rechenaufwand bei *t-rex* wiederum erhöhen. Im Rahmen dieser Arbeit konnten Zeit- und Speicherplatzangaben ermittelt werden, die ausschließlich untereinander in Relation gestellt werden können. Die tatsächlich aufzuwendende Zeit und der Speicherplatzbedarf sind neben der eingesetzten Software stark von der Leistung der Hardware, der räumlichen Ausdehnung der Geometrien (*bounding box*), der Anzahl der zu erzeugenden Zoomstufen, den enthaltenen Layern und der Komplexität der eingesetzten Generalisierung abhängig (Kap. 7.3.1, Tabelle 6).

Der Speicherbedarf von Vector Tiles derselben geografischen Ausdehnung und Anzahl von Zoomstufen ist in Summe geringer als bei Raster Tiles. Beim hier umgesetzten Szenario mit *TileServer GL* und *Mapproxy* können die als Zwischenschritt erforderlichen Raster Tiles entweder *on demand*, dabei mit und ohne *Caching*, oder durch *Pre-Rendering* mit *mapproxy-seed* erzeugt werden. In den beiden Varianten mit Anlage eines *Cache* kommen beim Speicherplatzbedarf zu den Vector Tiles die Raster Kacheln hinzu, was diesen Vorteil zunichte macht. Die direkte Nutzung von mit *TileServer GL* gerenderten Bildkacheln über dessen WMTS- oder XYZ-Endpunkt ist theoretisch auch möglich, wurde hier jedoch nicht untersucht. Hierbei würde das serverseitige *Caching* entfallen, der serverseitige Renderaufwand dagegen ansteigen, da identische Kacheln bei jeder Anfrage erneut erzeugt werden. Im Rahmen der durchgeführten Lasttests wurden vergleichbare Zahlen ermittelt, indem der WMS aus *Mapproxy* einmal mit und einmal ohne *Caching* aufgerufen

wurde. Die Ergebnisse zeigen, dass die *sample time* beim Rendern *on demand* je nach Zoomstufe im Vergleich zum Laden der Raster Tiles aus dem *Cache* von *Mapproxy* bis auf das 25-fache ansteigt (vgl. Median bei Zoomstufe 0 von Variante 4 (*on demand*) = 1.365 ms und Variante 8 (aus dem *Cache*) = 55 ms). Insgesamt wird der WMS in der untersuchten Konfiguration ohne Verwendung persistenter Raster Tiles 3- bis 5-mal langsamer.

Der Umweg über von *TileServer GL* gerenderte Raster Tiles zur Nutzung von Vector Tiles für einen WMS enthält einen weiteren entscheidenden Nachteil: In den mit *t-rex* generierten Vector Tiles sind Attributinformationen zu den Features enthalten. Diese gehen jedoch beim Rendern der Raster Tiles mit *TileServer GL* verloren. Um Attributdaten abfragen zu können, bietet sich neben einem herkömmlichen WMS ein mit der Möglichkeit zu *GetFeatureInfo-Requests* konfigurierter WMTS basierend auf Vector-Datenformaten an. Der WMTS-Endpunkt von *TileServer GL* bietet aktuell jedoch keine Möglichkeit für einen *FeatureInfo-Response*. Um Attributinformationen aus Vector Tiles in einem WMS direkt nutzen zu können, wäre deren direkte Einbindung in einen WMS-fähigen Kartenserver erforderlich, wie in *MapServer* in Anfängen geschehen.

Ein wesentlicher Vorteil eines WMS gegenüber einem WMTS ist die höhere clientseitige Flexibilität (Stile, Projektionen) bei der Verwendung der Daten. Diese kann so mit dem hier umgesetzten Szenario der Nutzung von Vector Tiles aufgrund des Umweges über Raster Tiles nicht aufrechterhalten werden. Hier besteht zwar die Möglichkeit, die Vector Tiles in unterschiedlichen Layern bzw. Gruppen von Layern in Raster Tiles mit Transparenz zu rendern und über mehrere XYZ- bzw. WMTS-Endpunkte von *TileServer GL* anzubieten und diese dann in *Mapproxy* zu einem WMS zusammenzufassen. Ein wirklicher Vorteil gegenüber der direkten Nutzung eines WMS oder WMTS ist bei dieser Herangehensweise nicht zu erkennen und die Möglichkeit Attributdaten abzufragen fällt, wie bereits beschrieben, weg.

In dieser Arbeit wurden keine Performancevergleiche zwischen der oben beschriebenen Software-Architektur und einem herkömmlichen WMTS durchgeführt. Da bereits die Anfragen an den WMS aus *GeoServer* basierend auf *Shape*-Dateien in den meisten Zoomstufen mit einer niedrigeren *sample time* beantwortet werden, als bei der Verwendung des Szenarios mit *TileServer GL* und *Mapproxy* und die Nutzung eines *Cache* eine höhere Performance nach sich zieht (Loechel & Schmid 2012), kann davon ausgegangen werden, dass bei Nutzung eines WMTS (mit gefülltem Raster Tile *Cache*) aus *GeoServer* die Zugriffsgeschwindigkeit im Vergleich zu den hier untersuchten Varianten mit *TileServer GL* und *Mapproxy* sinkt. Dabei besteht zusätzlich die Möglichkeit, den Server mit optionaler *GetFeaturInfo*-Anfrage zu konfigurieren, was die Nutzbarkeit des Services verbessert. Hier erscheint auch ohne tiefer gehende Untersuchungen, ein WMTS basierend auf originären Vektordaten aktuell die sinnvollere Variante darzustellen.

Das Fehlen von Standards, junge oder auf einen Anwendungsfall ausgerichtete Software und teilweise unzureichende Dokumentationen machten es wiederholt unmöglich, die in Recherchen ermittelten, theoretisch möglichen Anwendungsfälle umzusetzen. So wurde festgestellt, dass von *GeoServer* über WMTS bereitgestellte Vector Tiles von *QGIS* über eine *TileJSON-Datei* gelesen werden können. Eine Einbindung des TMS-Endpunktes in diese Konfiguration war dagegen nicht erfolgreich. *TileServer GL* konnte in den Tests keinen der beiden Endpunkte nutzen. Eine nach Dokumentation von *t-rex* mögliche Ausgabe von Vector Tiles in einem anderen CRS oder sogar Umprojektion beim Generieren und deren anschließende Einbindung in *TileServer GL* konnte nicht realisiert werden. Versuchsweise mit *GeoServer* erzeugte *TopoJSON*-Vector Tiles wiesen fehlerhafte Geometrien auf, die der Beibehaltung von Topologien, dem zentralen Gedanken hinter diesem Datenformat, widersprechen.

Vector Tiles im *MVT*-Format wiesen bei identischen Ausgangsdaten, gleicher Projektion und Zoomstufe stark voneinander abweichende Inhalte (Lage und Anzahl der übernommenen Vertices, Anzahl der Geometrien) auf. Hier wird deutlich, dass es bisher keine Vorgaben dafür gibt, wie Generalisierung beim Erzeugen stattzufinden hat bzw. wenn ohne eindeutige Vorgabe generalisiert wurde, wie diese Prozesse nachvollziehbar in den Metadaten zu beschreiben sind. Mangels entsprechender Vorgaben könnte beispielsweise das Vector Tile `z = 4, x = 8, y = 5` (*XYZ*-Schema) im *GeoJSON*-Format theoretisch eine Kopie des kompletten DLM1000, im MVT-Format dieselben Daten, jedoch mit auf Bildschirmkoordinaten verschobenen Vertices oder nur wenige und auch mit Blick auf die Vertices stark generalisierte Features enthalten (vgl. Tabelle 5). Eine tiefer gehende Untersuchung, inwieweit bei der Generierung von Vector Tiles Einschränkungen bei der Übernahme von Features in kleineren Zoomstufen möglich sind, fand im Rahmen dieser Arbeit bei der eingesetzten Software nicht statt. Eine praxistaugliche Variante wäre vergleichbar mit der Steuerung der Darstellung von Features entsprechend des Maßstabes bei den verbreiteten Desktop-GIS-Systemen.

Der Schwerpunkt bei Vector Tiles liegt aufgrund der derzeitig häufigsten Anwendung in *Plugins* auf Internetseiten aktuell ganz klar beim EPSG 3857. Fehlerfrei konnte eine Umprojektion von Daten in ein anderes CRS beim Generieren von Vector Tiles nur mit *GeoServer* umgesetzt werden. Diese Vector Tiles waren aus den beschriebenen Gründen in den getesteten Szenarien jedoch nicht verwendbar. Somit konnten in dieser Arbeit, entgegen den Rechercheergebnissen, nur Szenarien mit EPSG 3857 erfolgreich umgesetzt werden. Ein wesentliches Kriterium ist auch die Beschränkung von *MBTiles*-Datenbanken auf EPSG 3857. Hier wäre eine größere Flexibilität, auch mit Blick auf andere Dateiformate von Vector Tiles wünschenswert. Der Ansatz von Gesteira (2017) und die Einbindung von Vector Tiles in *GeoPackage* zeigen, dass hier, abseits von *MBTiles*, auch andere Lösungen mit Vector Tiles in Datenbanken denkbar sind.

Alle in dieser Arbeit abgeleiteten Softwareszenarien können ausschließlich *Mapbox Vector Tiles* als Datengrundlage nutzen. Eine Verwendung von *GeoJSON* als Datenquelle ist derzeit noch kaum und in keinem der hier umgesetzten Szenarien möglich. Ein wesentlicher Gedanke bei dieser Arbeit, nämlich dieselbe Vector Tiles Datenquelle sowohl für exakte GIS-Auswertungen als auch als Datengrundlage für einen Darstellungsdienst zu verwenden, entfällt somit. Umgesetzt werden konnte dagegen die direkte Nutzung für clientseitiges Rendern über einen *TileJSON*-Endpunkt *von TileServer GL*. Durch die Beschränkung auf das *MVT*-Format entfiel die Möglichkeit, den Einfluss des Datei-Formates auf die Performance zu untersuchen. *Mapbox Vector Tiles* sind für die Darstellung optimiert, d. h. die geografischen Koordinaten der enthaltenen Geometrien werden in Bildschirmkoordinaten umgewandelt. Für die Nutzung in Darstellungsdiensten und damit das zentrale Ziel dieser Arbeit ist die damit verbundene Reduzierung der Genauigkeit jedoch irrelevant bzw. ausreichend.

Die vorausgegangenen Ausführungen können in der in Tabelle 11 aufgeführten Gegenüberstellung von erwarteten und erhofften Verbesserungen beim Einsatz von Vector Tiles als Datenquelle für WMS und WMTS und dem jeweiligen Ergebnis entsprechend der durchgeführten Untersuchungen zusammengefasst werden.

Tabelle 11 Darstellung erwarteter bzw. erhoffter Vorteile von Vector Tiles als Datenquelle gegenüber herkömmlichem WMS und WMTS und deren Umsetzungserfolg

erwartete bzw. erhoffte Vorteile	**erfolgreich bzw. umgesetzt?**
– schnellere Antwortzeiten als mit *Shape*-Dateien als Datenquelle	– je nach Schwerpunktsetzung bei der Interpretation der Ergebnisse – nicht bei Rendern der Raster Tiles *on demand*
– niedrigerer Speicherbedarf als Raster Tiles	– Vector Tiles grundsätzlich ja; jedoch ggf. zusätzlich Raster Tiles im *Cache* von *Mapproxy* erforderlich, dann keine Verbesserung
– flexiblere Anwendung als WMTS (Umprojektion, Stile)	– nein; identisch mit WMTS da Umweg über Raster Tiles erforderlich
– Verfügbarkeit von Attributinformationen	– sind über die mit *TileServer GL* gerenderten Raster Tiles nicht abrufbar → Nachteil gegenüber WMTS basierend auf originären Vektordaten (*GetFeatureInfo*) – Nutzung der Attribute über direkte Verwendung von Vector Tiles aus *TileServer GL* über *TileJSON*
– nur eine Vector Tiles Datenquelle für Darstellungsdienst und direkte Nutzung in Clients	teilweise: – mit *mbutil* erzeugte *MBTiles*-Datenbanken mit Vector Tiles können über die URL der *TileJSON*-Datei auch direkt von Clients genutzt werden → hier jedoch nur render-basiertes Datenformat. – Für feature-basierte Lösungen Vector Tiles in zusätzlichem Format (z. B. *GeoJSON*) erforderlich
– clientseitiges Rendering	– im untersuchten Kontext nicht vorgesehen, jedoch mit direkter Nutzung der Vector Tiles über *TileJSON*-URL möglich

9 Zusammenfassung und Ausblick

Als zentrale Ziel dieser Arbeit sollte herausgefunden werden, ob Vector Tiles als Datenquelle für einen OGC Darstellungsdienst eingesetzt werden können. Als wesentliche Grundlage fand eine detaillierte Recherche zu Vector Tiles statt, aus der als Ergebnis eine umfassende und strukturierten Übersicht zu Begrifflichkeiten, Tiling, Datenformaten, Ablagestrukturen und Besonderheiten von Vector Tiles erarbeitet wurde (Kap. 4 und Abbildung 16). In einem zweiten Schritt wurde Software recherchiert, die in der Lage ist, Vector Tiles zu generieren, in Kartenbilder zu rendern oder anderweitig zu verwenden. Aus den ermittelten Programmen und Bibliotheken wurden Softwarelösungen ausgeschlossen, die sich grundsätzlich nicht zur Umsetzung der Ziele dieser Arbeit eignen oder nur einschränkt nutzbar wären. Darunter fallen z. B. Anwendungen, die ausschließlich für das Rendern von Vector Tiles in Webclients entwickelt wurden. Die verbleibenden 14 Programme und Bibliotheken wurden mit Blick auf ihre Nutzbarkeit im Rahmen dieser Arbeit genauer analysiert.

Basierend auf den Erkenntnissen aus Grundlagen- und Softwarerecherche wurden mögliche Szenarien abgeleitet, die theoretisch eine Verwendung von Vector Tiles als Datengrundlage für einen WMS ermöglichen. Von diesen Szenarien wurden zwei beispielhaft umgesetzt und mit Lasttests mit dem Tool *Apache JMeter* auf deren Performance untersucht und mit zwei herkömmlichen WMS, basierend auf *Shape*-Dateien als Datenquelle, verglichen. Die Schwerpunktsetzung bei der Interpretation der Zahlen ist entscheidend dafür, ob ein Performancegewinn bei der Nutzung von Vector Tiles abgelesbar ist. Bei den verglichenen Szenarien ist, unter Berücksichtigung von 90 % der Anfragen an den jeweiligen Service, in den niedrigeren Zoomstufen (0 bis 4) der WMS basierend auf Vector Tiles unter Nutzung des *Cache* von *Mapproxy* performanter. Bei Zoomstufe 5 sind die beschriebenen Szenarien in etwa gleich schnell, wogegen in den Zoomstufen ≥ 6 der herkömmliche WMS schnellere Zugriffszeiten auf die Daten ermöglicht. Eine Änderung der Konfiguration der Lasttests würde, wie in Kap. 8 beschrieben, eine Verschiebung des Ergebnisses nach sich ziehen. Aus diesen Gründen kann die zentrale Frage dieser Arbeit erst nach Festlegung auf einen der beschriebenen Schwerpunkte eindeutig beantwortet werden. Darüber hinaus war mangels vorhandener Kombinationsmöglichkeiten von Software und Quelldatenformat nicht klärbar, ob die Datengrundlage oder die eingesetzte Software für die festgestellten Unterschiede verantwortlich sind.

Die oft nur auf ein Einsatzgebiet ausgerichtete und teilweise noch junge Software erschwerte die Ableitung funktionierender Softwarekonstellationen und ein Teil der theoretisch möglichen Funktionen oder Szenarien konnte nicht umgesetzt werden. Die Entwicklung im Bereich Vector Tiles wird aktuell sehr aktiv vorangetrieben, wie ein neues erst 2017 entwickeltes Vector Tile Format und im Zeitraum der Anfertigung dieser Arbeit hinzugekommene Funktionen und Erweiterungen verdeutlichen. So konnte mit der Einbindung der aktuellen *GDAL*-Bibliothek, welche *Mapbox Vector Tiles* unterstützt, in *MapServer* die direkte Verwendung von Vector Tiles in einen WMS beispielhaft nachgestellt werden. Diese funktioniert bisher nur eingeschränkt, zeigt aber die grundsätzliche Richtung bei der Verwendung von Vector Tiles auf, wie sie zukünftig auch für andere Anwendungen zu erwarten ist.

Als wesentlicher Grund für die aktuell oft noch nicht zuverlässig funktionierende Kommunikation zwischen Anwendungen kann das Fehlen eines offenen Standards für Vector Tiles gesehen werden. Deswegen ist die zeitnahe Entwicklung eines offenen und flexiblen Standards für Vector Tiles der wesentlichste Schritt für die weitere Entwicklung. Die Ausführungen des *OGC Testbed-13* (Cavazzi 2018) *z. B.* zum *OGC Vector tiling model* illustrieren, dass für entsprechende Entwicklung zahlreiche Ideen zur Verbesserung und Erweiterung der vorhandenen Quasistandards bei Übernahme in einen offenen Standard existieren. So sollte dieser

Standard mehrere Daten(austausch)formate erlauben, um einerseits performante Web- oder mobile Clients mit einfach strukturierten, kompakten Daten zu versorgen, aber auch komplexe Anwendungen mit erweiterten Informationen ermöglichen (Cavazzi 2018). Darüber hinaus empfehlen Cavazzi (2018) und Nordan (2012) die Verwendung derselben Standards für die Kachelung von Vektor- und Rasterdaten, um bei Kombination dieser Daten den Zugriff auf die Tiles zu vereinfachen. Im *OGC Testbed-13* (Seite 44) wird weiterhin empfohlen, im Zusammenhang mit der Entwicklung von Standards für Vector Tiles auch die Standards für vorhandene Webdienste zu einem UMS zusammenzuführen:

> *„A Unified Map Service regrouping most capabilities of WFS, WMS, WCS, WMTS and Catalogue Services for the Web (CSW) with shared semantics based on JSON & ECON ...“*
>
> ***übersetzt:***
>
> *„Ein Unified Map Service, der die meisten Funktionen von WFS, WMS, WCS, WCS, WMTS und Catalogue Services for the Web (CSW) mit gemeinsamer Semantik auf Basis von JSON & ECON vereint ...“*

Die meisten Anwendungen von Vector Tiles sind aktuell auf ein Datenformat (*MVT/PBF*) und das CRS EPSG 3857 beschränkt. Hiermit werden die Möglichkeiten, welche Vector Tiles bieten, bei weitem nicht ausgeschöpft. Aktuell beschränken sie sich meist auf den schnellen Datenzugriff für render-basierte Lösungen. Im Rahmen der Recherchen konnten jedoch weitere potenzielle Anwendungsgebiete von Vector Tiles ermittelt werden, die mit anderen Datenformaten besser umzusetzen sind. Diese sind jedoch bisher nur selten und teilweise auch fehlerhaft umgesetzt (z. B. *TopoJSON* in *GeoServer*). Eine größere Flexibilität würde auch die Erweiterung der *MBTiles*-Spezifikation auf andere CRS oder die Entwicklung eines offenen und weitergehenden Standards zur Ablage von Vector Tiles in Datenbanken nach sich ziehen. *GeoJSON* als feature-basiertes Vector Tile Format wurde bei den möglichen Szenarien nicht weiterverfolgt, da die Einschränkungen durch den aktuell erforderlichen Umweg über Raster Tiles so groß sind, dass die Vorteile des Formates nicht zum Tragen kommen würden, dagegen aufgrund des speicherintensiveren Formates eine geringere Performance als bei *MVT* zu erwarten wäre. Bei einer direkten Nutzung von Vector Tiles als Datenquelle für einen WMS und parallel direkter Nutzung durch Clients müsste dieser Punkt neu überdacht bzw. untersucht werden.

In Zukunft werden Vector Tiles voraussichtlich immer größere Verbreitung bei der direkten Nutzung von Geodaten in Clients erlangen. Spätestens mit diesem Schritt steigt auch der Bedarf an der Verwendung dieser Daten für exakte GIS-Auswertungen. Neben der beschriebenen Veränderung der Geometrien durch die Umwandlung in Bildschirmkoordinaten sind hierbei noch andere, von der Nutzung für reine Kartendarstellung abweichende, spezifische Probleme zu beachten. Die Ungenauigkeit durch die Transformation in Bildschirmkoordinaten könnte, je nach Anwendungsfall, teilweise in Kauf genommen werden. Jedoch stellt sich in diesem Zusammenhang z. B. die Frage, wie das Nachladen aller Tiles eines Features organisiert werden kann. Bei zusammenhängenden Features, wie einfachen Polygonen oder Linien, ist die Umsetzung noch gut vorstellbar. Wie stellt sich aber die Situation mit Features dar, die sich aus Multipolygonen oder -linien zusammensetzen? Nordan (2012) beschreibt dazu das Beispiel des Staates Portugal, dessen Fläche inklusive aller Inseln (auch in vom Festland weit entfernten Tiles) berechnet werden soll.

Als render-basierte Lösung (Darstellungsdienst) kann aktuell nur das Szenario aus *t-rex*, *TileServer GL* und WMS über *Mapproxy* zur Umsetzung empfohlen werden. Hierbei ist als einzige recherchiertes Szenario, zumindest theoretisch, die Verwendung unterschiedlicher CRS möglich. Dieses konnte im Rahmen dieser Arbeit jedoch nicht erfolgreich nachgestellt werden. Bei der dargestellten Konstellation findet für den WMS jedoch der Umweg über in *TileServer GL* gerenderte Raster Tiles statt. Die Attributinformationen der Features gehen hierbei verloren und die eigentliche Flexibilität eines WMS wird auf die Starrheit eines

WMTS, noch dazu ohne die Möglichkeit eines *GetFeatureInfo-Request* reduziert. Vor dem Hintergrund der nicht eindeutigen Performancesteigerung, den bestehenden Einschränkungen und der derzeit aktiven Weiterentwicklung im Bereich Vector Tiles ist es fraglich, ob der erforderliche Aufwand bei der Umstellung eines bestehenden WMS auf dieses Szenario gerechtfertigt ist. Ob eine Beschleunigung eines WMTS mit Vector Tiles erreicht werden kann, wurde in dieser Arbeit nicht separat untersucht. Mit dem Vorrendern der Raster Tiles wird, basierend auf originären Vektordaten (z. B. WMTS in *GeoServer*), derselbe Effekt erreicht wie beim Zugriff über *Mapproxy* mit gefülltem *Cache* im beschriebenen Szenario, nur das hier *GetFeaturInfo-Requests* möglich sind.

Die Ursachen für das unerwartete Verhalten der konfigurierten Dienste, wie eingesetzte Software, hoher Aufwand beim Zusammenfügen von Features aus Vector Tiles und die Berücksichtigung der bereits stattgefundenen Generalisierung, konnten im Rahmen dieser Arbeit nicht eindeutig geklärt werden. Zusammenfassend kann festgestellt werden, dass folgende Änderungen der aktuellen Rahmenbedingungen eine erfolgreiche Umsetzung des zentralen Ziels dieser Arbeit erwarten lassen:

- Entwicklung von offenen und flexiblen Standards für Datenformate von Vector Tiles,
- mit identischen Standards für das Tiling von Raster- und Vektordaten mit Unterstützung aller CRS,
- daraus folgend eine zeitnah zu erwartende direkte und effektive Einbindung aller in den neuen Standards vorgesehenen Optionen in die verbreiteten Anwendungen (z. B. *GeoServer* und *MapServer*) und damit vergleichbare Nutzbarkeit von Vector Tiles, wie bei herkömmlichen Vektordaten,
- ebenfalls damit verbunden die Verfügbarkeit von Attributinformationen und
- anschließend weitere Performancesteigerung durch Optimierung der Konfiguration der eingesetzten Software.

10 Quellenverzeichnis

Aitchison, Alastair. 2011. Converting TMS Tile Coordinates to Google/Bing/OSM Tile Coordinates. *Alastair Aitchison*. https://alastaira.wordpress.com/2011/07/06/converting-tms-tile-coordinates-to-googlebingosm-tile-coordinates/.

Allegri, Giovanni. 2017. Vector Tiles, PostGIS and OpenLayers. *Tantotanto*. https://medium.com/tantotanto/vector-tiles-postgis-and-openlayers-258a3b0ce4b6.

Antoniou, Vyron, Jeremy Morley und Mordechai (Muki) Haklay. 2009. Tiled Vectors: A Method for Vector Transmission over the Web. In *Web and Wireless Geographical Information Systems*, 56–71. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg. doi:10.1007/978-3-642-10601-9_5.

Balkenbush, Jason. 2017. Web Mapping - What Are the Differences between TMS, XYZ & WMTS? - Geographic Information Systems Stack Exchange. *Geographic Information Systems Stack Exchange*. https://gis.stackexchange.com/questions/132242/what-are-the-differences-between-tms-xyz-wmts.

Balog, Daniel und Robin Houtmeyers. 2017a. Testbed-12 Vector Tiling Implementation Engineering Report. 29 S. OGC. http://docs.opengeospatial.org/per/16-067r4.html.

Balog, Daniel und Robin Houtmeyers. 2017b. OGC Testbed-12 Vector Tiling Engineering Report. 38 S. OGC. http://docs.opengeospatial.org/per/16-068r4.html.

Bertolotto, Michela und Max J. Egenhofer. 2001. Progressive Transmission of Vector Map Data over the World Wide Web. *GeoInformatica* 5 (4): 345–373. doi:10.1023/A:1012745819426.

Blanc, Nicolas, Oliver Burkhard, Raphaël Burkhard, Sarah Composto, Michelle Fillekes, David Hanimann, Jens Ingensand, et al. 2016. Innovationsbericht GEOSummit 2016. 42 S. Bern. http://www.sogi.ch/fileadmin/redaktion/sogi2013/Innovationsbericht_GEOSummit_2016_final_d_f.pdf.

Blasby, David und Andreas Hocevar. 2016a. Vector Tiles with GeoServer and OpenLayers. presented at the FOSS4G North America, Raleigh, North Carolina. https://2016.foss4g-na.org/session/vector-tiles-geoserver-and-openlayers.html.

Blasby, David und Andreas Hocevar. 2016b. Vector Tiles with GeoServer and OpenLayers. presented at the FOSS4G 2016 Bonn. http://2016.foss4g.org/talks.html#154.

Bostock, Mike und Calvin Metcalf. 2017. Topojson: An Extension of GeoJSON That Encodes Topology!. https://github.com/topojson/topojson-specification.

Branigan, John. 2013. *UTF-Grid Is Unbelievably Totally Fast*. https://www.youtube.com/watch?v=JauVpmRC1d0.

Bühner, Nils, André Henn und Daniel Koch. 2018. GeoServer in action. In FOSSGIS 2018, 46 S. Bonn. https://terrestris.github.io/geoserver-in-action-ws/.

Bundesamt für Kartographie und Geodäsie. 2018. Open Data - Freie Daten und Dienste des BKG *Open Data - Freie Daten und Dienste des BKG*. http://www.geodatenzentrum.de/geodaten/gdz_rahmen.gdz_div.

Butler, Howard, Martin Daly, Allan Doyle, Sean Gillies, Stefan Hagen und Tim Schaub. 2016. The GeoJSON Format, 28 S. doi:10.17487/RFC7946.

Butler, Howard, Martin Daly, Allan Doyle, Sean Gillies, Tim Schaub und Christopher Schmidt. 2008. The GeoJSON Format Specification. http://geojson.org/geojson-spec.

Camper, Brett. 2016. Mapnik Vector Tiles vs Mapbox Vector Tiles · Issue #345 · Tangrams/Tangram. *GitHub*. https://github.com/tangrams/tangram/issues/345.

Cavazzi, Stefano. 2018. OGC Testbed-13: Vector Tiles Engineering Report. 157 S. OGC. http://docs.opengeospatial.org/per/17-041.html.

Cibulka, Dušan. 2013. Performance Testing of Web Map Services Tn Three Dimensions – X, Y, Scale. *Slovak Journal of Civil Engineering* 21 (1): 31–36. doi:10.2478/sjce-2013-0005.

Cogo, Vinicius Vielmo. 2014. Serializing Data with Protocol Buffers. Universidade de Lisboa. http://homepages.lasige.di.fc.ul.pt/~vielmo/notes/2014_02_12_smalltalk_protocol_buffers.pdf.

Corcoran, Padraig und Peter Mooney. 2011. Topologically Consistent Selective Progressive Transmission. In *Advancing Geoinformation Science for a Changing World*, 519–538. Lecture Notes in Geoinformation and Cartography. Springer, Berlin, Heidelberg. doi:10.1007/978-3-642-19789-5_26.

Corcoran, Padraig, Peter Mooney und Michela Bertolotto. 2012. Utilizing Geometric Coherence in the Computation of Map Transformations. *Computers & Geosciences* 47: 151–159.

Corcoran, Padraig, Peter Mooney, Michela Bertolotto und Adam Winstanley. 2011. View- and Scale-Based Progressive Transmission of Vector Data. In *Computational Science and Its Applications - ICCSA 2011*, 51–62. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg. doi:10.1007/978-3-642-21887-3_5.

Cozzi, Patrick. 2015. Introducing 3D Tiles | Cesium.Com. *The Cesium Blog*. https://cesium.com/blog/2015/08/10/introducing-3d-tiles/.

de la Beaujardiere, Jeff. 2006. Web Map Service | OGC Version 1.3.0. OGC. http://www.opengeospatial.org/standards/wms.

de la Torre, Rafa. 2017. MVT Generation: Mapnik vs PostGIS. https://carto.com/blog/inside/MVT-mapnik-vs-postgis/.

Defense Geospatial Information Working Group. 2016. Proposed Extension for Multi-Resolution Vector Data in OGC GeoPackage. presented at the DGIWG Technical Panel Meetings. https://portal.opengeospatial.org/files/?artifact_id=68934.

Douglas, David H und Thomas K Peucker. 1973. Algorithms for the Reduction of the Number of Points Required to Represent a Digitized Line or Its Caricature. *Cartographica: The International Journal for Geographic Information and Geovisualization* 10 (2): 112–122. doi:10.3138/FM57-6770-U75U-7727.

Du, Shihong, Qimin Qin, Qiao Wang und Haijian Ma. 2008. Evaluating Structural and Topological Consistency of Complex Regions with Broad Boundaries in Multi-Resolution Spatial Databases. *Information Sciences* 178 (1): 52–68. doi:10.1016/j.ins.2007.07.023.

Dufilie, Andrew und Georges Grinstein. 2014. Feathered Tiles with Uniform Payload Size for Progressive Transmission of Vector Data. In *Web and Wireless Geographical Information Systems*, 19–35. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg. doi:10.1007/978-3-642-55334-9_2.

ECERE Corporation. 2016. ECON - EC Object Notation (A JSON Superset). http://ec-lang.org/econ/.

ECERE Corporation. 2017. *GNOSIS Map Server @ Maps.Ecere.Com*. ECERE. http://maps.ecere.com/.

ECMA. 2017. *Standard ECMA-404*. http://www.ecma-international.org/publications/standards/Ecma-404.htm.

ESRI. 2018. Vektorkachelpaket Erstellen—Data Management (Toolbox) | ArcGIS Desktop. https://pro.arcgis.com/de/pro-app/tool-reference/data-management/create-vector-tile-package.htm.

Fischer, Eric, Konstantin Käfer, Tom MacWright, Justin Miller, Paul Norman, Blake Thompson und Will White. 2018. *Mbtiles-Spec: Specification Documents for the MBTiles Tileset Format*. En. Mapbox. https://github.com/mapbox/mbtiles-spec.

Fitzsimmons, Seth. 2018. *Tessera - A Tilelive-Based Tile Server.* JavaScript. https://github.com/mojodna/tessera.

Gaffuri, Julien. 2012. Toward Web Mapping with Vector Data. In *Geographic Information Science*, 87–101. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg. doi:10.1007/978-3-642-33024-7_7.

GeoJSON.org. 2018. GeoJSON. http://geojson.org/.

GeoServer Contributors. 2018. GeoServer Documentation. http://docs.geoserver.org/.

GeoWebCache Contributors. 2018. GeoWebCache 1.12.2 — GeoWebCache 1.12.2 User Manual. http://geowebcache.org/docs/current/.

Gesteira, Francisco Girón. 2017. Creating Vector Tiles for Background Maps with FME. presented at the FME International User Conference 2017, Vancouver. https://www.safe.com/presentation/creating-vector-tiles-for-background-maps-with-fme/.

GISpunkt HSR. 2017. Shapefile. Wiki. *Wiki GISpunkt HSR*. https://giswiki.hsr.ch/Shapefile.

Giuliani, Gregory, Alain Dubois und Pierre Lacroix. 2013. Testing OGC Web Feature and Coverage Service Performance: Towards Efficient Delivery of Geospatial Data. *Journal of Spatial Information Science* 2013 (7): 1–23. doi:10.5311/JOSIS.2013.7.112.

Google Inc. 2010. Under the Hood of Google Maps 5.0 for Android. *Official Google Blog*. https://googleblog.blogspot.com/2010/12/under-hood-of-google-maps-50-for.html.

Google Inc. 2017a. Frequently Asked Questions | Protocol Buffers. *Google Developers*. https://developers.google.com/protocol-buffers/docs/faq.

Google Inc. 2017b. Map and Tile Coordinates | Google Maps JavaScript API. *Google Maps API*. https://developers.google.com/maps/documentation/javascript/coordinates.

Google LLC. 2017. Protocol Buffer Developer Guide. *Google Developers*. https://developers.google.com/protocol-buffers/docs/pythontutorial.

Guan, Xuefeng, Bo Cheng, Aihong Song und Huayi Wu. 2014. Modeling Users' Behavior for Testing the Performance of a Web Map Tile Service. *Transactions in GIS* 18 (November): 109–125. doi:10.1111/tgis.12123.

Harrtell, Björn. 2016. [Postgis-Users]ST_AsMVT Returns an Error When Calculating Tile Bbox from x, y, z. https://lists.osgeo.org/pipermail/postgis-users/2016-November/041777.html.

Henneberger, Stefan. 2015. Geoserver – Rendering Binary Vector Tiles. *Salzburg Research Forschungsgesellschaft*. https://www.salzburgresearch.at/blog/geoserver-rendering-binary-vector-tiles/.

Ingensand, Jens, Marion Nappez, Olivier Ertz und Sarah Composto. 2016. Implementation of Tiled Vector Services: A Case Study. *GIScience 2016*, 9 S.

Ingensand, Jens, Marion Nappez, Cédric Moullet, Loïc Gasser und Sarah Composto. 2015. Vector Tiles for the Swiss Federal Geoportal. In *Proceedings of the 2015 FOSS4G Europe Conference*, 717–724. Como, Italy. http://geomatica.como.polimi.it/workbooks/n12/FOSS4G-eu15_submission_143.pdf.

IOC Task Force for Network Services. 2013. Technical Guidance for the Implementation of INSPIRE View Services v3.11. 115 S. INSPIRE. https://inspire.ec.europa.eu/documents/technical-guidance-implementation-inspire-view-services-1.

JSON.org. 2018. JSON. http://json.org/json-de.html.

Jurk, Thomas. 2010. Integration von Tiled Map Services in Geodateninfrastrukturen. Diplomarbeit, Dresden: HTW Dresden. https://geoinformatik.htw-dresden.de/2010/10/integration-von-tiled-map-services-in-gdi/.

Käfer, Konstantin. 2011. How Interactivity Works with UTFGrid. *Points of Interest - The Official Mapbox Blog*. https://blog.mapbox.com/how-interactivity-works-with-utfgrid-3b7d437f9ca9.

Käfer, Konstantin, Young Hahn und Tom MacWright. 2018. Tilejson-Spec: JSON Format for Describing Map Tilesets. Spezifikation. *Mapbox TileJSON 2.2.0*. https://github.com/mapbox/tilejson-spec.

Kalberer, Pirmin. 2017a. Von WMS zu WMTS zu Vektor-Tiles. presented at the FOSSGIS 2017, Passau. https://www.fossgis-konferenz.de/2017/programm/event.php?id=5233.

Kalberer, Pirmin. 2017b. T-Rex, a Vector Tile Server for Your Own Data. presented at the FOSS4G 2017, Boston. http://foss4g.guide/#.

Kalberer, Pirmin. 2018. T-Rex - Documentation. *T-Rex - Documentation*. http://t-rex.tileserver.ch/doc/.

Kim, Dae Hyun und Myoung-Jun Kim. 2006. An Extension of Polygon Clipping To Resolve Degenerate Cases. *Computer-Aided Design and Applications* 3 (1–4): 447–456. doi:10.1080/16864360.2006.10738483.

Kozioł, Krystian, Michał Lupa und Artur Krawczyk. 2014. The Extended Structure of Multi-Resolution Database. In *Beyond Databases, Architectures and Structures*, 435–443. Communications in Computer and Information Science. Springer, Cham. doi:10.1007/978-3-319-06932-6_42.

Li, Lin, Wei Hu, Haihong Zhu, You Li und Hang Zhang. 2017. Tiled Vector Data Model for the Geographical Features of Symbolized Maps. *PLOS ONE* 12 (5): 26 S. doi:10.1371/journal.pone.0176387.

Loechel, Alexander und Stephan Schmid. 2012. Verschiedene Caching-Techniken für Web Map Services. In *Angewandte Geoinformatik 2012*, 191–200. AGIT 2012 – Symposium und Fachmesse Angewandte Geoinformatik. Berlin: Wichmann. https://gispoint.de/gisopen-paper/584-verschiedene-caching-techniken-fuer-web-map-services.html.

Longley, Paul, Michael F. Goodchild, David J. Maguire und David W. Rhind. 2005. *Geographic Information Systems and Science*. John Wiley & Sons. 544 S.

Lupp, Markus. 2007. Styled Layer Descriptor | OGC. OGC. http://www.opengeospatial.org/standards/sld.

MacWright, Tom. 2013. The Difference between XYZ and TMS Tiles and How to Convert between Them. *Gist*. https://gist.github.com/tmcw/4954720.

MacWright, Tom, Will White, Konstantin Käfer, Justin Miller und Dane Springmeyer. 2014. *Utfgrid-Spec: Specification for UTFGrid, a Format for Rasterized Interaction Data*. En. Mapbox. https://github.com/mapbox/utfgrid-spec.

Mapbox. 2016. Mapbox Vector Tile Specification. *Mapbox*. https://www.mapbox.com/vector-tiles/specification.

Mapbox. 2018. *Mapbox-Gl-Native: Interactive, Thoroughly Customizable Maps in Native Android, IOS, MacOS, Node.Js and Qt Applications, Powered by Vector Tiles and OpenGL*. C++. Mapbox. https://github.com/mapbox/mapbox-gl-native.

Mapnik Contributors. 2018. *Mapnik* (version 3.0.20). C++. Mapnik. https://github.com/mapnik/mapnik.

MapServer Contributors. 2018. MapServer 7.2.0 Documentation. http://mapserver.org/de/documentation.html.

mapsforge. 2018. *Mapsforge: Vector Map Library and Writer - Running on Android and Desktop*. Java. https://github.com/mapsforge/mapsforge.

Martinelli, Lukas und Manuel Roth. 2015. *Vector Tiles from OpenStreetMap*. Student Research Project Thesis. Rapperswill. 78 S.

Masó, Joan, Keith Pomakis und Núria Julià. 2010. OpenGIS Web Map Tile Service Implementation Standard. OGC. file:///C:/Users/X/Downloads/07-057r7_Web_Map_Tile_Service_Standard.pdf.

Masó, Joan, Xavier Pons und R. Singh. 2010. OGC WMTS and OSGeo TMS Standards: Motivations, History and Differences. presented at the FOSS4G, Barcelona. http://2010.foss4g.org/presentations/3653.pdf.

McMaster, Robert B. und K. Stuart Shea. 1992. Generalization in Digital Cartography. 134 S. Washington D.C.: Association of American Geographers. http://geoinformatics.ntua.gr/courses/admcarto/lecture_notes/generalisation/bibliography/mcmaster_shea_1992.pdf.

Migurski, Michael. 2013. The Liberty of Postgreslessness: Tiled Vectors in Mapnik (Tecznotes). Blog. *Tecznotes*. http://mike.teczno.com/notes/postgreslessness-mapnik-vectiles.html.

Migurski, Michael. 2018. TileStache API. http://tilestache.org/doc/.

Morton, G. M. 1996. A Computer Oriented Geodetic Data Base; and a New Technique in File Sequencing. 9 S. IBM. http://domino.research.ibm.com/library/cyberdig.nsf/papers/0DABF9473B9C86D48525779800566A39/$File/Morton1966.pdf.

Nordan, Robert Patrick Victor. 2012. An Investigation of Potential Methods for Topology Preservation in Interactive Vector Tile Map Applications. Master Thesis, Trondheim. 97 S. https://brage.bibsys.no/xmlui/handle/11250/232116.

OGC. 2017. GeoPackage Encoding Standard | OGC. http://www.opengeospatial.org/standards/geopackage.

OpenLayers. 2018. OpenLayers Examples. https://openlayers.org/en/latest/examples/.

OpenStreetMap. 2015. Meta Tiles – OpenStreetMap Wiki. https://wiki.openstreetmap.org/wiki/Meta_tiles.

OpenStreetMap. 2018a. Standard Tile Layer – OpenStreetMap Wiki. https://wiki.openstreetmap.org/wiki/Standard_tile_layer.

OpenStreetMap. 2018b. Slippy Map Tilenames – OpenStreetMap Wiki. *Slippy Map Tilenames*. https://wiki.openstreetmap.org/wiki/Slippy_map_tilenames.

OpenStreetMap. 2018c. TMS – OpenStreetMap Wiki. https://wiki.openstreetmap.org/wiki/TMS.

Ordnance Survey. 2018. Raster and Vector Data | Support | Ordnance Survey. *Raster and Vector Data*. https://www.ordnancesurvey.co.uk/support/understanding-gis/raster-vector.html.

OSGeo. 2018. *GDAL: GDAL - Geospatial Data Abstraction Library* (version 2.3.1). C++. OSGeo. https://www.gdal.org/.

OSGeo Wiki. 2012a. Tile Map Service Specification. *Tile Map Service Specification*. http://wiki.osgeo.org/wiki/Tile_Map_Service_Specification#Servers.

OSGeo Wiki. 2012b. WMS Tile Caching - OSGeo. https://wiki.osgeo.org/wiki/WMS_Tile_Caching.

OSGeoLive Contributors. 2018. *OSGeoLive 12.0 Dokumentation* (version 12.0). OSGeo. https://live.osgeo.org.

Portele, Clemens. 2012. OGC® Geography Markup Language (GML) - Extended Schemas and Encoding Rules, Version 3.3.0, OGC Doc No. 10-129r1. Open Geospatial Consortium. https://portal.opengeospatial.org/files/?artifact_id=46568.

Praliaskouski, Darafei, Vladimir Agafonkin und Maksim Gurtovenko. 2017. *Kothic JS — a Full-Featured JavaScript Map Rendering Engine Using HTML5 Canvas*. JavaScript. Kothic Team. https://github.com/kothic/kothic-js.

Přidal, Petr. 2008. Tiles à La Google Maps: Coordinates, Tile Bounds and Projection - Conversion to EPSG:900913 (EPSG:3785) and EPSG:4326 (WGS84). *Tiles à La Google Maps: Coordinates, Tile Bounds and Projection*. http://www.maptiler.org/google-maps-coordinates-tile-bounds-projection/.

Přidal, Petr. 2016. *TileServer PHP: MapTiler and MBTiles Maps via WMTS*. PHP. Klokan Technologies GmbH. https://github.com/klokantech/tileserver-php.

Pywell, H. R. und H. A. Niedzwiadek. 1980. The Wetlands Analytical Mapping System: WAMS. In *Analytical Plotter Symposium and Workshop*, 261–270. Reston, VA. http://agris.fao.org/agris-search/search.do;jsessionid=DC22AF8AFFC333813C64379947DDF697?request_locale=es&recordID=US8260608&sourceQuery=&query=&sortField=&sortOrder=&agrovocString=&advQuery=&centerString=&enableField=.

Ramer, Urs. 1972. An Iterative Procedure for the Polygonal Approximation of Plane Curves. *Computer Graphics and Image Processing* 1 (3): 244–256. doi:10.1016/S0146-664X(72)80017-0.

Reed, C. N. 1986. DELTAMAP Just Another New GIS? In *Proceedings of the 3rd International Symposium on Spatial Data Handling*, 375–383. Williamsville NY.

Reed, Carl. 2017. CDB Common DataBase | OGC. http://www.opengeospatial.org/standards/cdb.

Reed, Carl und Tamrat Belayneh. 2017. OGC Indexed 3d Scene Layer (I3S) and Scene Layer Package Format Specification. Community Standard. http://docs.opengeospatial.org/cs/17-014r5/17-014r5.html.

Ross, Zev. 2014. Spatial Data on a Diet: Tips for File Size Reduction Using TopoJSON. *Technical Tidbits From Spatial Analysis & Data Science*. http://zevross.com/blog/2014/04/22/spatial-data-on-a-diet-tips-for-file-size-reduction-using-topojson/.

Roth, Manuel, Lukas Martinelli und Petr Přidal. 2016. Create Vector Tiles from OpenStreetMap. presented at the FOSS4G 2016, Bonn. https://frab.fossgis-konferenz.de/en/foss4g-2016/public/events/1144.

Schmid, Falko. 2013. *OpenScienceMap* (version 0.5). http://www.opensciencemap.org/?page_id=2.

Schmid, Falko, Hannes Janetzek, Michael Wladysiak und Bo Hu. 2013. OpenScienceMap: Open and Free Vector Maps for Low Bandwidth Applications. In *Proceedings of the 3rd ACM Symposium on Computing for Development*, 2. ACM DEV '13. New York, NY, USA: ACM. doi:10.1145/2442882.2442939.

Schwartz, Joe. 2018. Bing Maps Tile System. *Bing Maps Tile System*. https://msdn.microsoft.com/en-us/library/bb259689.aspx.

Seip, Christian und Ralf Bill. 2016. Evaluation and Monitoring of Service Quality: Discussing Ways to Meet INSPIRE Requirements. *Transactions in GIS* 20 (2): 163–181. doi:10.1111/tgis.12145.

Shang, Xiaohong. 2015. A Study on Efficient Vector Mapping With Vector Tiles Based on Cloud Server Architecture. Master Thesis, University of Calgary. doi:http://dx.doi.org/10.5072/PRISM/25046.

Shekhar, Shashi, Ranga Raju Vatsavai, Namita Sahay, Thomas E. Burk und Stephen Lime. 2001. WMS and GML Based Interoperable Web Mapping System. In *Proceedings of the ACM Workshop on Advances in Geographic Information Systems*. https://experts.umn.edu/en/publications/wms-and-gml-based-interoperable-web-mapping-system.

Skowron, Thomas. 2017a. Meine Eigene Karte: Überblick Über Rendering-Techniken Und Software. Passau. https://www.fossgis-konferenz.de/2017/programm/event.php?id=5183.

Skowron, Thomas. 2017b. Grandine: Vector Tiles, Summary March 2017. Blog. *Thomas Skowron*. https://thomas.skowron.eu/blog/grandine-summary-march-2017/.

Skowron, Thomas. 2018. Making Maps Without Database. https://2018.stateofthemap.org/2018/T092-Making_Maps_Without_Database/.

Sloup, Petr und Petr Přidal. 2016. Hosting Vector Tile Maps on Your Own Server. presented at the FOSS4G 2016, Bonn. https://frab.fossgis-konferenz.de/en/foss4g-2016/public/events/1288.

Smith, Kevin. 2014. GeoWebCache - WMTS Directory Structure - Geographic Information Systems Stack Exchange. *Geographic Information Systems | StackExchange*. https://gis.stackexchange.com/questions/77986/geowebcache-wmts-directory-structure.

Taraldsvik, Mats. 2012. The Future of Web-Based Maps: Can Vector Tiles and HTML5 Solve the Need for High-Performance Delivery of Maps on the Web? Trondheim: NTNU Trondheim. https://github.com/meastp/efficientvectortiles/raw/master/efficientvectortiles.pdf.

Teufel, Marc. 2008. Google Protocol Buffers: Mark-Set-Go! *JAXenter*. https://jaxenter.de/google-protocol-buffers-mark-set-go-9112.

The PostgreSQL Global Development Group. 2018. PostgreSQL 10.3 Documentation. https://www.postgresql.org/docs/current/static/.

Tomlinson, Roger F. 1990. Geographic Information Systems—a New Frontier. In *Introductory Readings In Geographic Information Systems*, 15–27. London: Taylor & Francis.

Tonnhofer, Oliver. 2017. MapProxy 1.11.0 Documentation. https://mapproxy.org/docs/1.11.0/index.html.

Visvalingam, Maheswari und James Duncan Whyatt. 1992. Line Generalisation by Repeated Elimination of the Smallest Area, 16 S.

Wan, Lin, Zhou Huang und Xia Peng. 2016. An Effective NoSQL-Based Vector Map Tile Management Approach. *ISPRS International Journal of Geo-Information* 5 (11): 25 S. doi:10.3390/ijgi5110215.

Weibel, R. und G. Dutton. 2005. Generalising Spatial Data and Dealing with Multiple Representations. In *Geographical Information Systems: Principles, Techniques, Management and Applications, Abridged*, 2nd ed., 125–155. Wiley-VCH..

Wikipedia. 2018. Vector Tiles. *Wikipedia*. https://en.wikipedia.org/w/index.php?title=Vector_tiles&oldid=830751249.

Williams, Craig und Edi Punt. 2017. Desktop Mapping: Creating Vector Tiles. presented at the 2017 Esri User Conference Technical Workshops, San Diego, California. http://proceedings.esri.com/library/userconf/proc17/tech-workshops.html.

Yang, Bisheng. 2005. A Multi-Resolution Model of Vector Map Data for Rapid Transmission over the Internet. *Computers & Geosciences* 31 (5): 569–578. doi:10.1016/j.cageo.2004.11.011.

Yang, Bisheng und Qingquan Li. 2009. Efficient Compression of Vector Data Map Based on a Clustering Model. *Geo-Spatial Information Science* 12 (1): 13–17. doi:10.1007/s11806-009-0181-5.

Yu, Eugene G., Liping Di, Md Rahman, Li Lin, Chen Zhang, Lei Hu, Ranjay Shrestha, Lingjun Kang, Junmei Tang und Guangyuan Yang. 2017. Performance Improvement on a Web Geospatial Service for the Remote Sensing Flood-Induced Crop Loss Assessment Web Application Using Vector Tiling. 6 S. doi:10.1109/Agro-Geoinformatics.2017.8047053.

Zhang, C., W. Li und T. Zhao. 2007. Geospatial Data Sharing Based on Geospatial Semantic Web Technologies. *Journal of Spatial Science* 52 (2): 35–49. doi:10.1080/14498596.2007.9635121.

Zhou, Mengyun, Jing Chen und Jianya Gong. 2015. A Virtual Globe-Based Vector Data Model: Quaternary Quadrangle Vector Tile Model. *International Journal of Digital Earth*, 1–37. doi:10.1080/17538947.2015.1016558.

11 Anlagen

Anlage 1 Übersicht Beispieldaten
Anlage 2 Vordefinierte *Tile Matrix Sets* nach WMTS-Standard
Anlage 3 Boxplots von Lasttests mit *Apache JMeter*

Anlage 1 Übersicht Beispieldaten

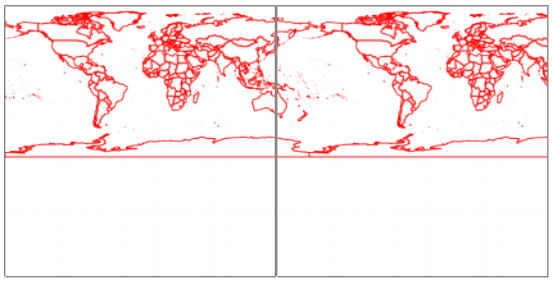
		DLM1000		DLM250	
		Layer	*Features*	*Layer*	*Features*
	gesamt	28	**258.453**	35	**751.318**
	Punkt	6	41.128	8	76.317
	Linie	9	204.014	11	547.180
	Fläche	13	13.311	16	127.821
Typ	***Layer***	***Beschreibung***			
Punkt	gew02_p	Besondere Gewässermerkmale		x	597
	rel01_p	Reliefformen		x	568
	rel02_p	Messdaten	x	2.739	x	10.046
	sie01_p	Ortslage	x	13.865	x	25.974
	sie03_p	Bauwerke und sonstige Einrichtungen	x	2.337	x	9.445
	sie04_p	Besondere Anlagen auf Siedlungsflächen	x	924	x	997
	sie05_p	Gebäude	x	2.252	x	12.858
	ver06_p	Verkehrsbauwerke und -anlagen	x	19.011	x	15.832
Linie	geb01_l	Verwaltungsgebiete	x	8.229	x	35.378
	gew01_l	Gewässer	x	49.918	x	73.137
	gew03_l	Gewässerachse	x	4.744	x	10.303
	rel01_l	Reliefformen	x	3.074	x	38.735
	sie03_l	Bauwerke und sonstige Einrichtungen			x	12.506
	sie04_l	Besondere Anlagen auf Siedlungsflächen	x	15	x	212
	ver01_l	Straßenverkehr	x	92.223	x	307.061
	ver02_l	Wege			x	10.667
	ver03_l	Bahnverkehr	x	23.917	x	22.126
	ver05_l	Schiffsverkehr	x	351	x	247
	ver06_l	Verkehrsbauwerke und -anlagen	x	21.543	x	36.808
Fläche	geb01_f	Verwaltungsgebiete	x	520	x	13.180
	geb02_f	Geographische Gebiete	x	269	x	1.170
	geb03_f	Schutzgebiete	x	998	x	2.569
	gew01_f	Gewässer	x	3.177	x	17.495
	gew02_f	Besondere Gewässermerkmale	x	228	x	498
	sie01_f	Ortslage	x	1.244	x	20.437
	sie02_f	Baulich geprägte Flächen	x	42	x	25.364
	sie03_f	Bauwerke und sonstige Einrichtungen			x	291
	sie04_f	Besondere Anlagen auf Siedlungsflächen			x	203
	veg01_f	Landwirtschaftliche Nutzfläche	x	587	x	1.883
	veg02_f	Forstwirtschaftliche Nutzfläche	x	5.365	x	35.994
	veg03_f	Vegetationsflächen	x	618	x	5.299
	veg04_f	Vegetationsmerkmal	x	198	x	3.034
	ver03_f	Bahnverkehr			x	152
	ver04_f	Flugverkehr	x	53	x	216
	ver06_f	Verkehrsbauwerke und -anlagen	x	12	x	36

Anlage 2 Vordefinierte *Tile Matrix Sets* nach WMTS-Standard

Nachfolgend werden die vier im WMTS-Standard aufgeführten *Tile Matrix Sets* aufgelistet und basierend auf diesen *Sets* Beispielkacheln der Zoomstufen 1 bzw. 2 gezeigt, die aus dem Layer *Admin 0 - Countries* von Natural Earth[1] in Geoserver erzeugt wurden. Die Hinweise zu den *Tile Matrix Sets* stammen aus dem WMTS-Standard.

GlobalCRS84Scale urn:ogc:def:wkss:OGC:1.0:GlobalCRS84Scale

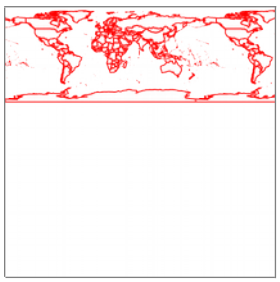
- für intuitive Repräsentation von Vektordaten
- gerundete Maßstäbe
- Maßstab nur akkurat am Äquator



Kacheln Zoomstufe 0 für Matrix Set *GlobalCRS84Scale*

GlobalCRS84Pixel urn:ogc:def:wkss:OGC:1.0:GlobalCRS84Pixel

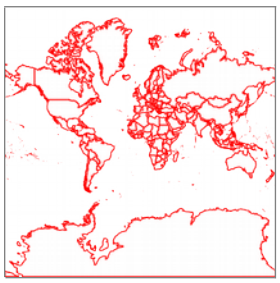
- für Rasterdaten
- gerundete Pixelgrößen,
- Angaben nur korrekt am Äquator



Kacheln Zoomstufe 0 für Matrix Set *GlobalCRS84Pixel*

GoogleMapsCompatible urn:ogc:def:wkss:OGC:1.0:GoogleMapsCompatible

- Web Mercator
- Maßstab nur akkurat am Äquator

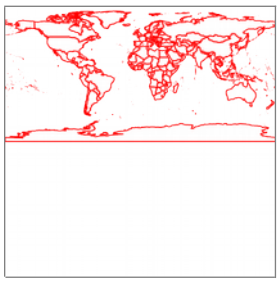


Kacheln Zoomstufe 0 für Matrix Set *GoogleMapsCompatible* (entspricht GridSet EPSG:900913 in GeoServer)

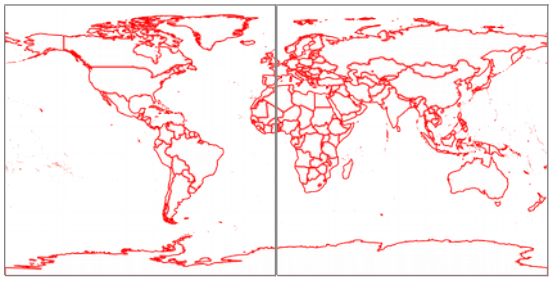
1 http://www.naturalearthdata.com/downloads/10m-cultural-vectors/

GoogleCRS84Quad ***urn:ogc:def:crs:OGC:1.3:CRS84***

- für Quadtree-Pyramiden im CRS84 (EPSG:4326)
- Maßstab nur akkurat am Äquator
- laut Standard Zoom 0 mit oben und unten 64 leere Pixel



Kacheln Zoomstufe 0 für Matrix Set *GoogleCRS84Quad*; Umsetzung durch Geoserver (Abstand oben und unten) anders als im Standard beschrieben

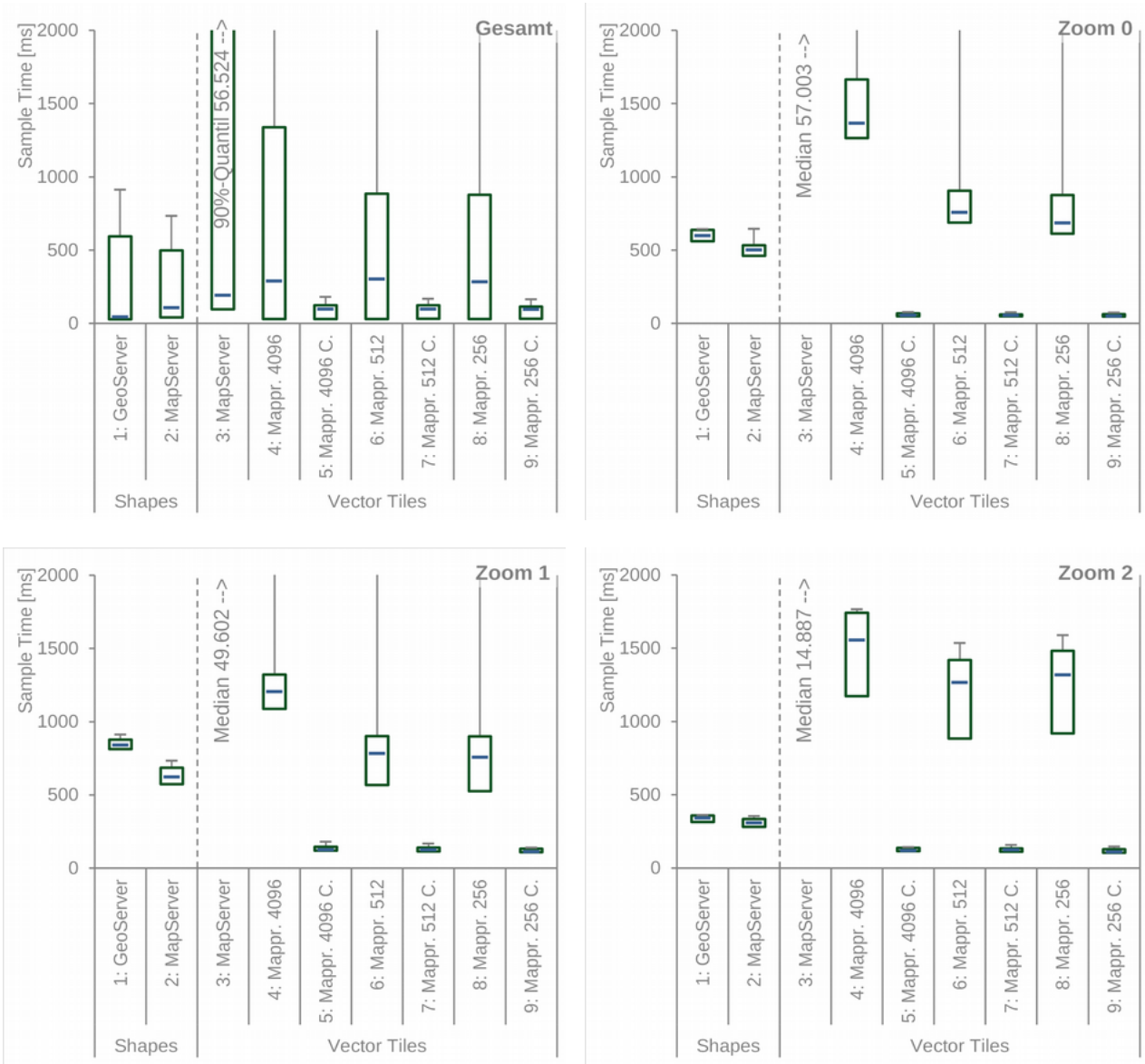


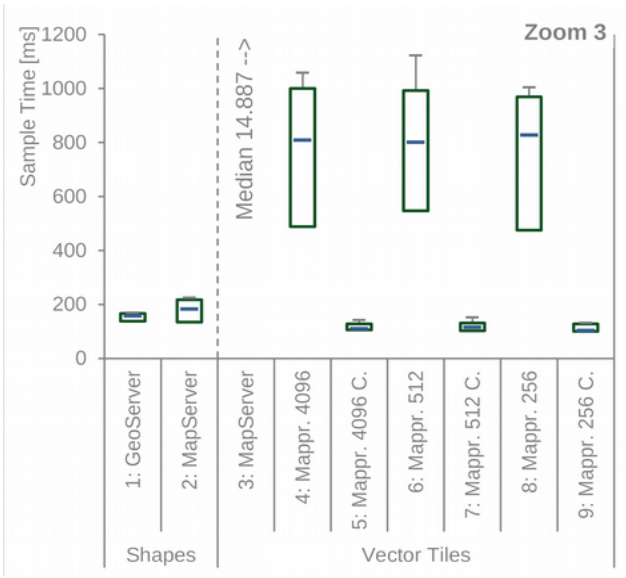
Kacheln Zoomstufe 1 für Matrix Set *GoogleCRS84Quad*

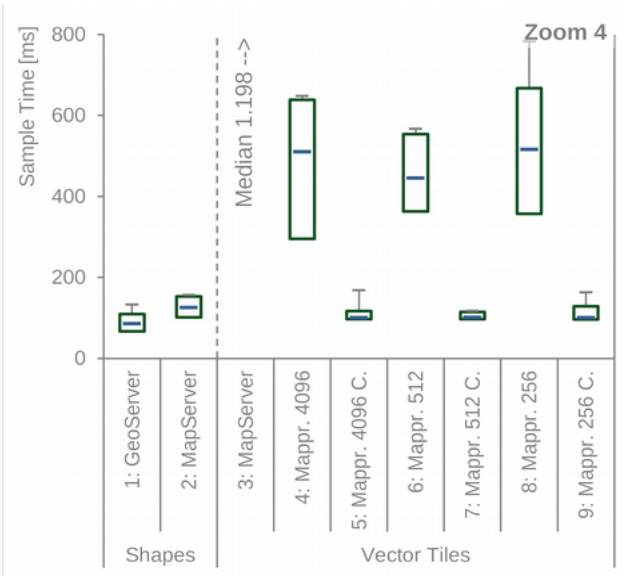
Anlage 3 Boxplots von Lasttests mit *Apache JMeter*

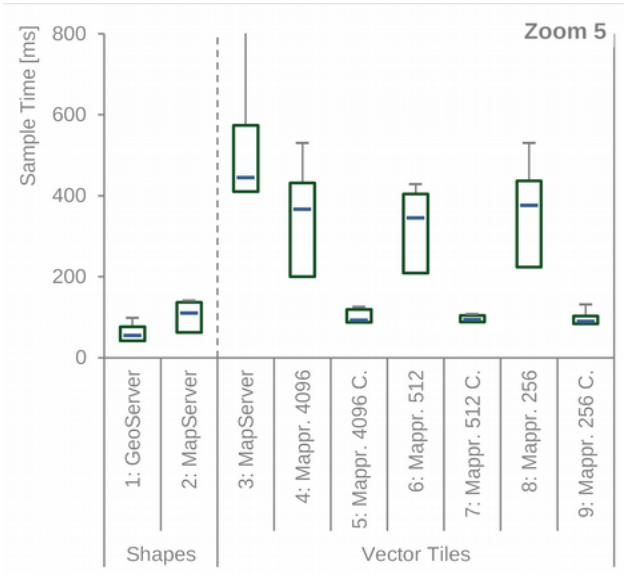
Darstellung von Wertebereich des 90%-Quantils (▭) - obere Grenze = 90 % Line, Median (─), Maximum (⊤) der *Sample Time* in Millisekunden. Bei jeder Variante (X-Achse) wird im Diagramm zur eindeutigen Zuordnung vor dem Doppelpunkt die entsprechende Nummer aus Tabelle 8 angegeben.

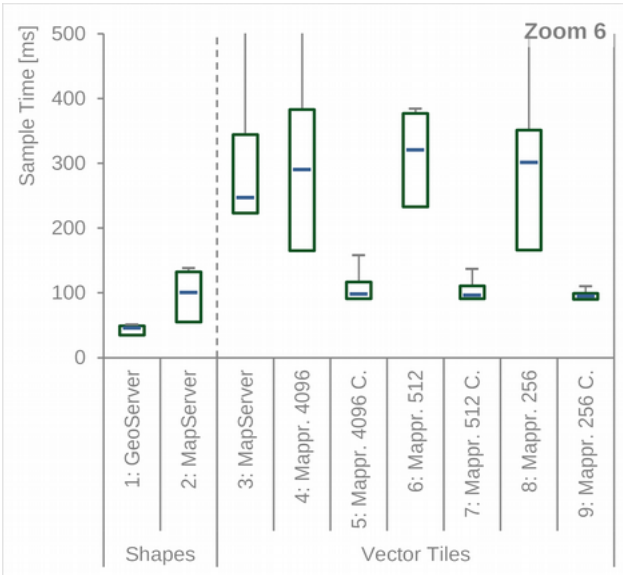
1. Boxplots *Gesamt* und einzelne Zoomstufen für Lasttest mit einem *Thread* - Varianten 1 bis 9

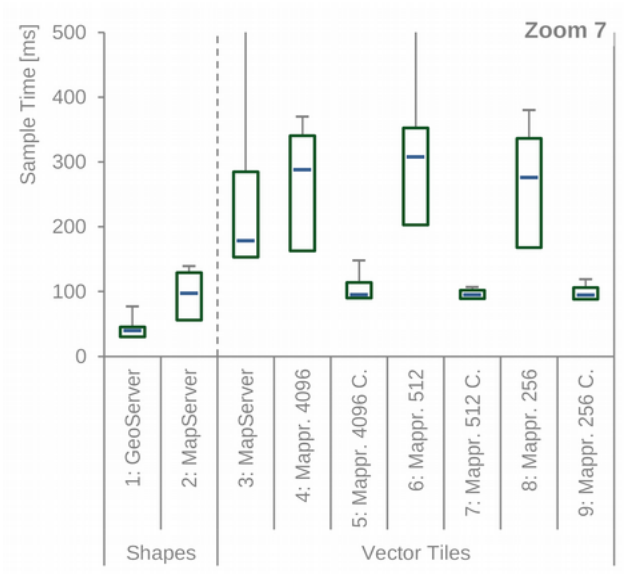


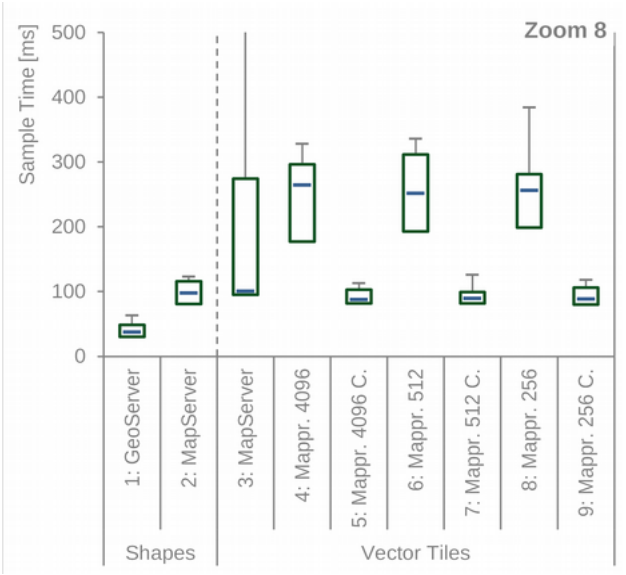

Zoom 3
Sample Time [ms]
1200
1000
800
600
400
200
0
Median 14.887 -->
1: GeoServer
2: MapServer
3: MapServer
4: Mappr. 4096
5: Mappr. 4096 C.
6: Mappr. 512
7: Mappr. 512 C.
8: Mappr. 256
9: Mappr. 256 C.
Shapes
Vector Tiles

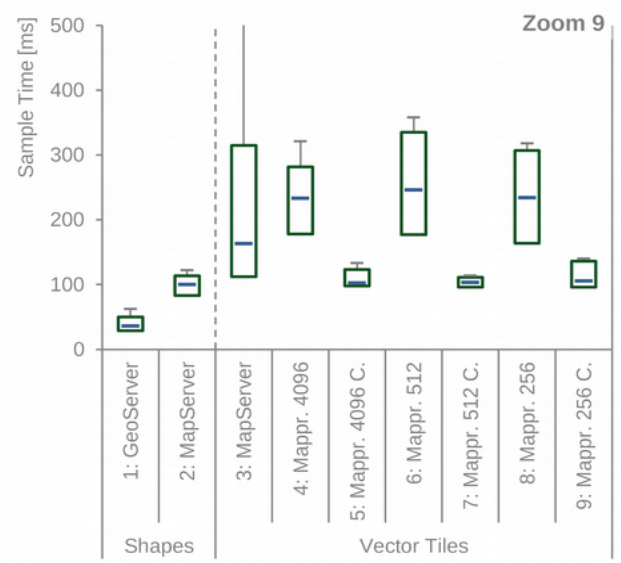

Zoom 4
Sample Time [ms]
800
600
400
200
0
Median 1.198 -->
1: GeoServer
2: MapServer
3: MapServer
4: Mappr. 4096
5: Mappr. 4096 C.
6: Mappr. 512
7: Mappr. 512 C.
8: Mappr. 256
9: Mappr. 256 C.
Shapes
Vector Tiles

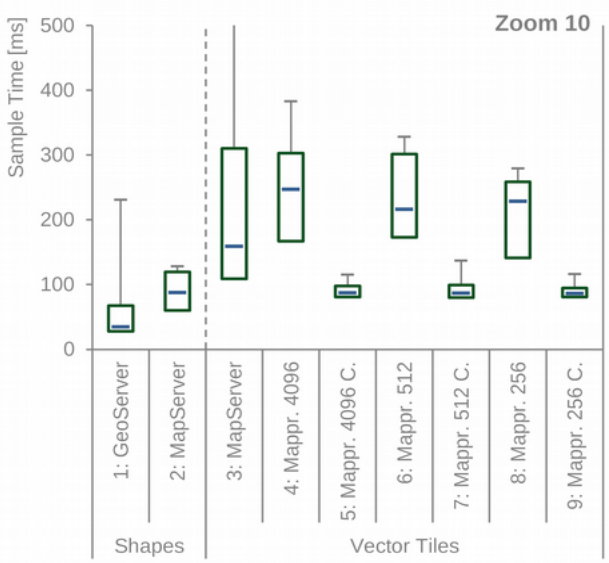

Zoom 5
Sample Time [ms]
800
600
400
200
0
1: GeoServer
2: MapServer
3: MapServer
4: Mappr. 4096
5: Mappr. 4096 C.
6: Mappr. 512
7: Mappr. 512 C.
8: Mappr. 256
9: Mappr. 256 C.
Shapes
Vector Tiles

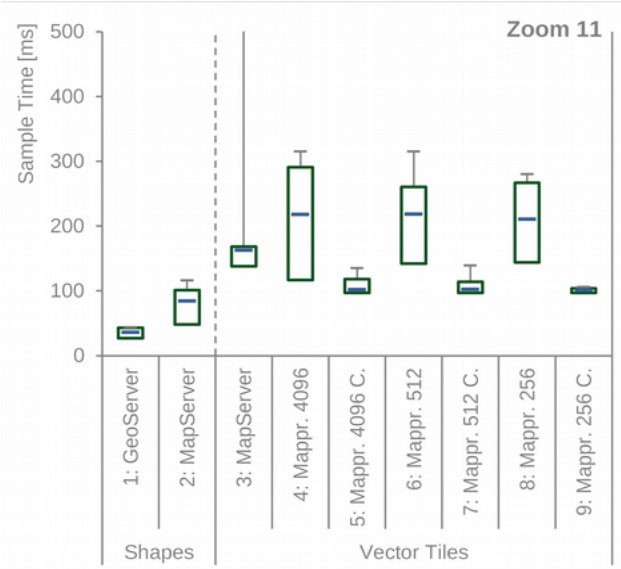

Zoom 6
Sample Time [ms]
500
400
300
200
100
0
1: GeoServer
2: MapServer
3: MapServer
4: Mappr. 4096
5: Mappr. 4096 C.
6: Mappr. 512
7: Mappr. 512 C.
8: Mappr. 256
9: Mappr. 256 C.
Shapes
Vector Tiles

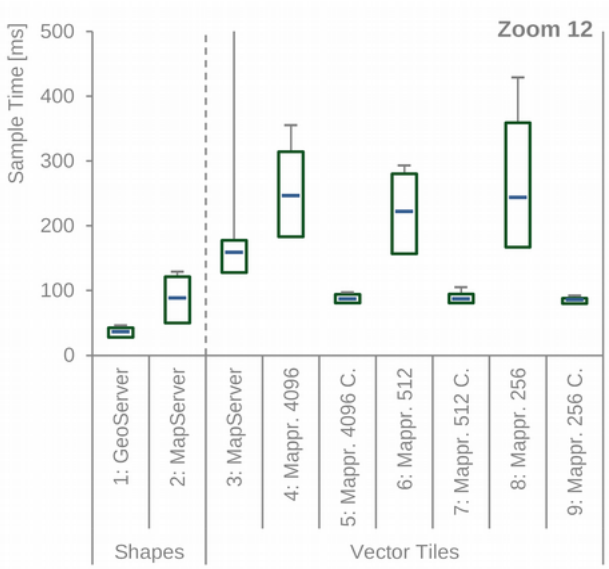

Zoom 7
Sample Time [ms]
500
400
300
200
100
0
1: GeoServer
2: MapServer
3: MapServer
4: Mappr. 4096
5: Mappr. 4096 C.
6: Mappr. 512
7: Mappr. 512 C.
8: Mappr. 256
9: Mappr. 256 C.
Shapes
Vector Tiles

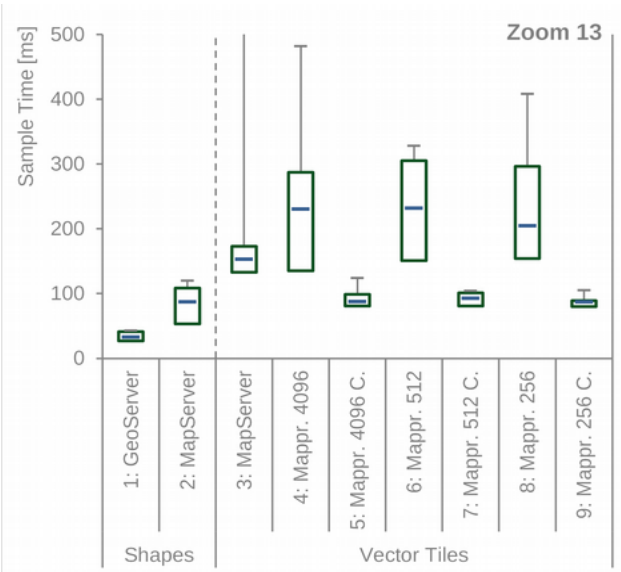

Zoom 8
Sample Time [ms]
500
400
300
200
100
0
1: GeoServer
2: MapServer
3: MapServer
4: Mappr. 4096
5: Mappr. 4096 C.
6: Mappr. 512
7: Mappr. 512 C.
8: Mappr. 256
9: Mappr. 256 C.
Shapes
Vector Tiles

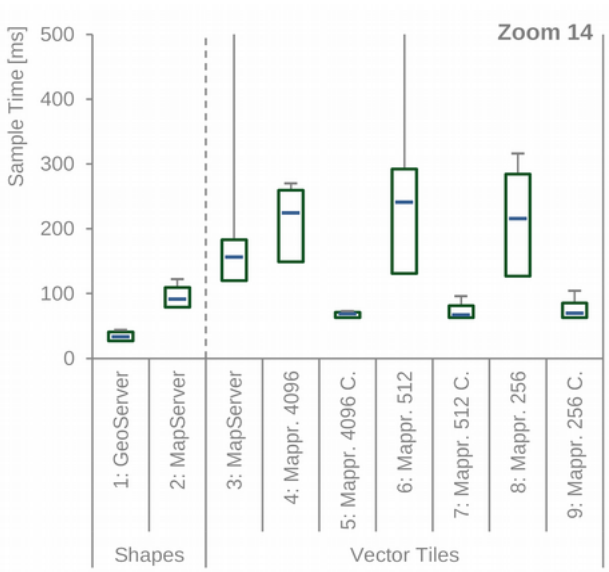

Zoom 9
Sample Time [ms]
500
400
300
200
100
0
1: GeoServer
2: MapServer
3: MapServer
4: Mappr. 4096
5: Mappr. 4096 C.
6: Mappr. 512
7: Mappr. 512 C.
8: Mappr. 256
9: Mappr. 256 C.
Shapes
Vector Tiles


Zoom 10
Sample Time [ms]
500
400
300
200
100
0
1: GeoServer
2: MapServer
3: MapServer
4: Mappr. 4096
5: Mappr. 4096 C.
6: Mappr. 512
7: Mappr. 512 C.
8: Mappr. 256
9: Mappr. 256 C.
Shapes
Vector Tiles

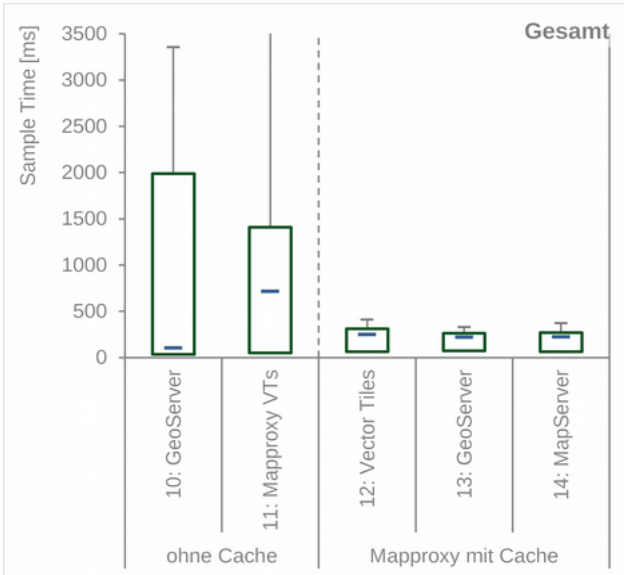

Zoom 11
Sample Time [ms]
500
400
300
200
100
0
1: GeoServer
2: MapServer
3: MapServer
4: Mappr. 4096
5: Mappr. 4096 C.
6: Mappr. 512
7: Mappr. 512 C.
8: Mappr. 256
9: Mappr. 256 C.
Shapes
Vector Tiles

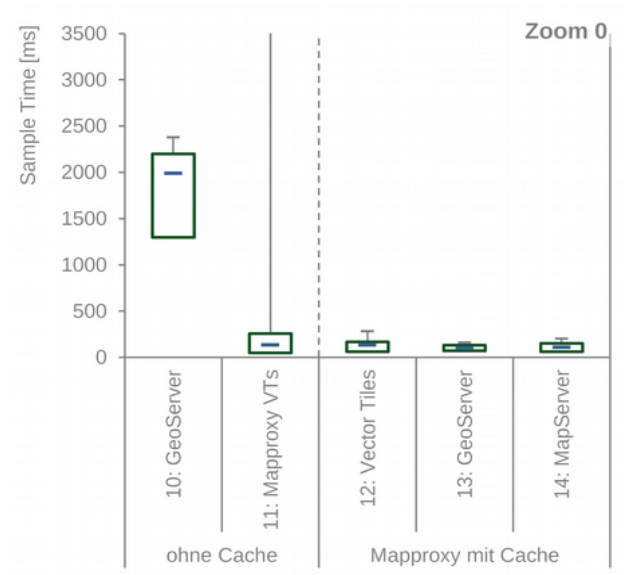

Zoom 12
Sample Time [ms]
500
400
300
200
100
0
1: GeoServer
2: MapServer
3: MapServer
4: Mappr. 4096
5: Mappr. 4096 C.
6: Mappr. 512
7: Mappr. 512 C.
8: Mappr. 256
9: Mappr. 256 C.
Shapes
Vector Tiles

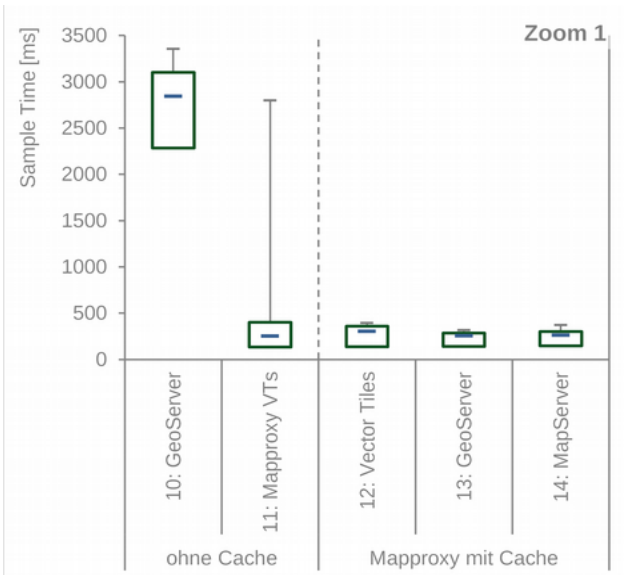

Zoom 13
Sample Time [ms]
500
400
300
200
100
0
1: GeoServer
2: MapServer
3: MapServer
4: Mappr. 4096
5: Mappr. 4096 C.
6: Mappr. 512
7: Mappr. 512 C.
8: Mappr. 256
9: Mappr. 256 C.
Shapes
Vector Tiles

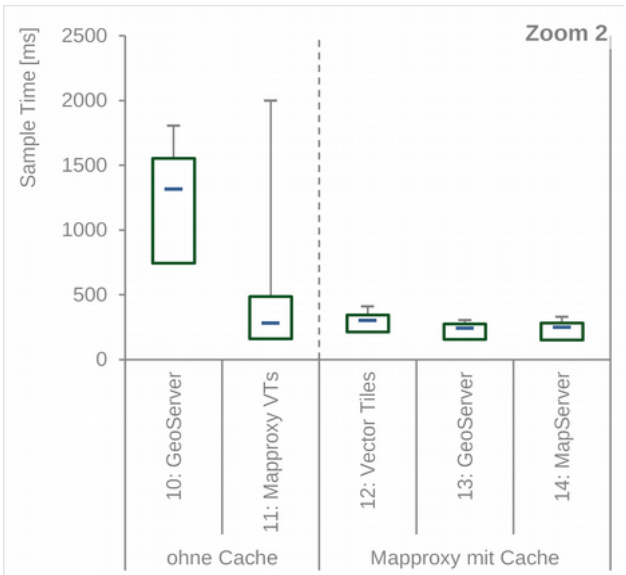

Zoom 14
Sample Time [ms]
500
400
300
200
100
0
1: GeoServer
2: MapServer
3: MapServer
4: Mappr. 4096
5: Mappr. 4096 C.
6: Mappr. 512
7: Mappr. 512 C.
8: Mappr. 256
9: Mappr. 256 C.
Shapes
Vector Tiles

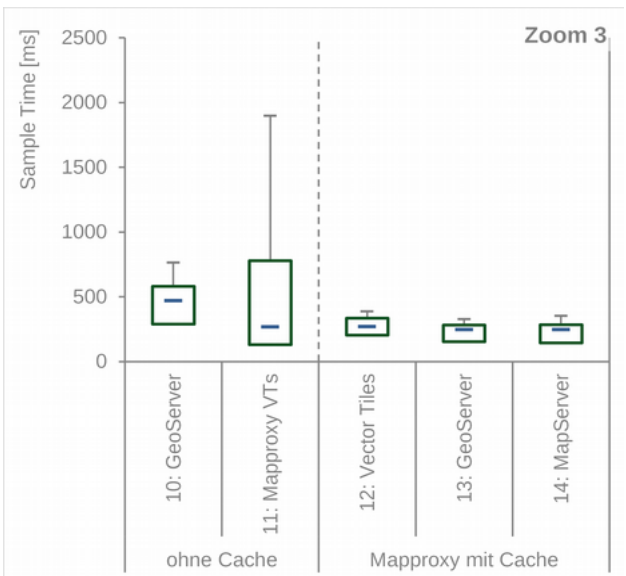
2. Boxplots *Gesamt* und einzelne Zoomstufen für Lasttest mit 5 *Threads* - *Varianten 10 bis 14*

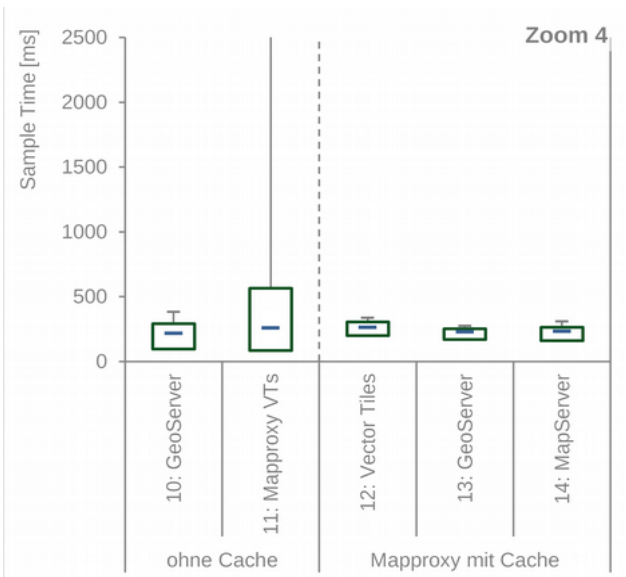


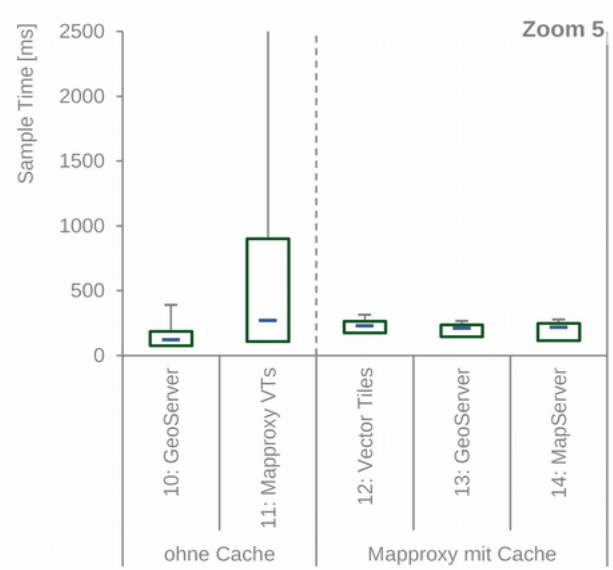


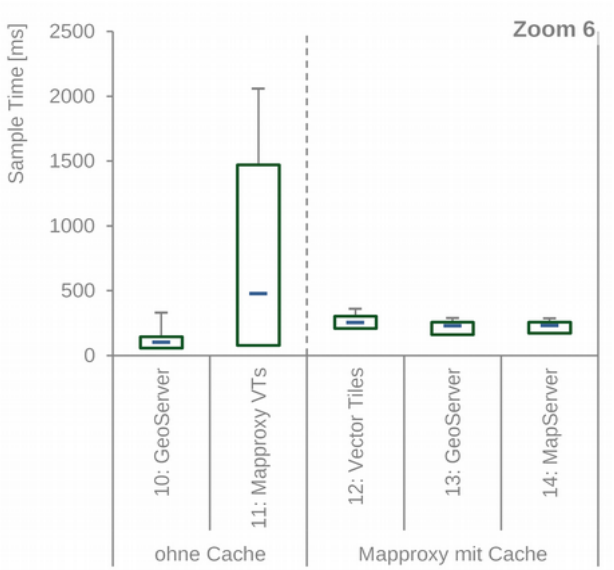


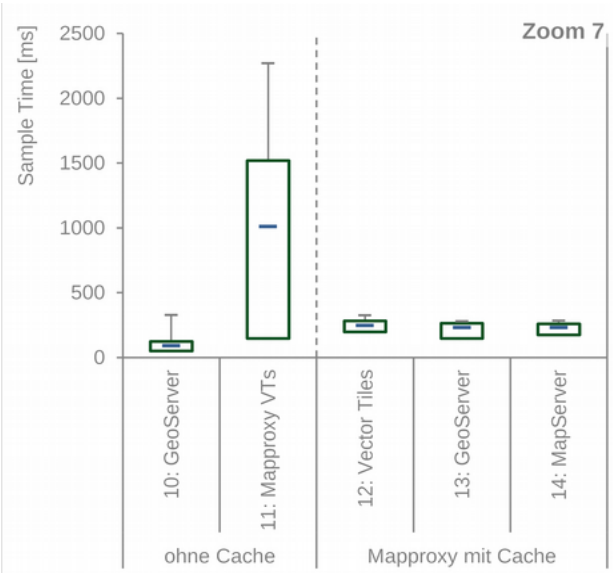


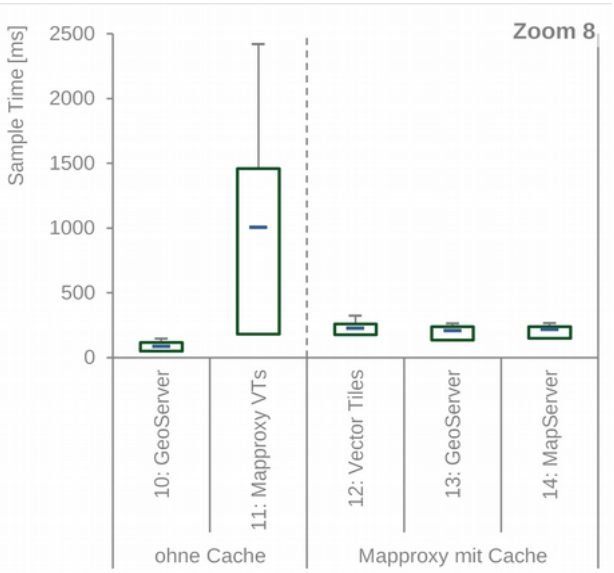


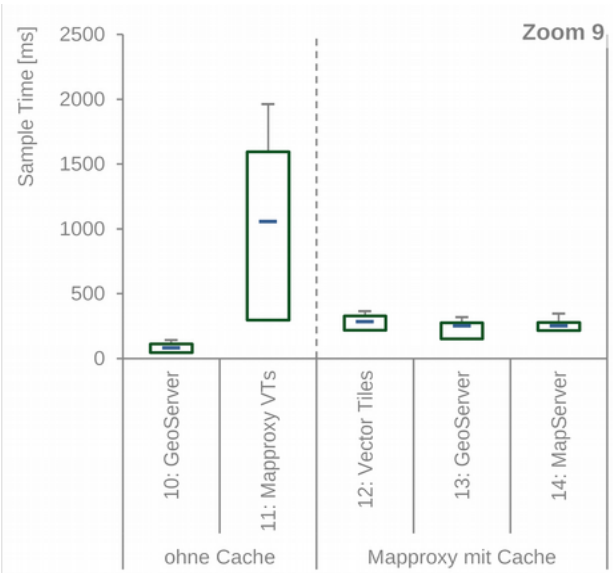


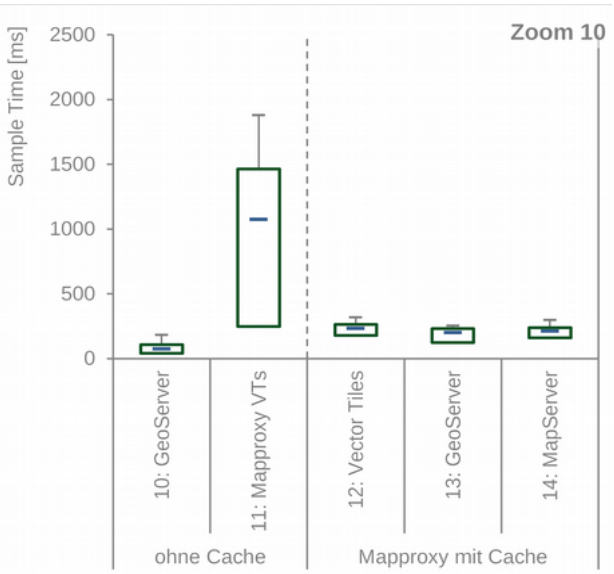

Zoom 5
Sample Time [ms]
2500
2000
1500
1000
500
0
10: GeoServer
11: Mapproxy VTs
12: Vector Tiles
13: GeoServer
14: MapServer
ohne Cache
Mapproxy mit Cache

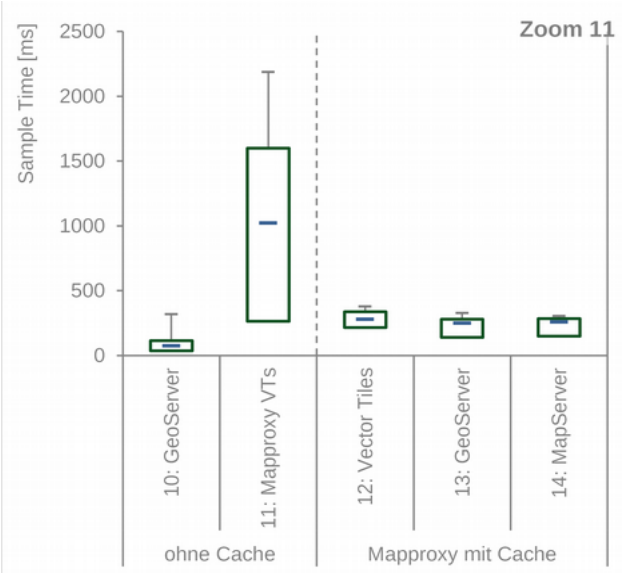

Zoom 6
Sample Time [ms]
2500
2000
1500
1000
500
0
10: GeoServer
11: Mapproxy VTs
12: Vector Tiles
13: GeoServer
14: MapServer
ohne Cache
Mapproxy mit Cache

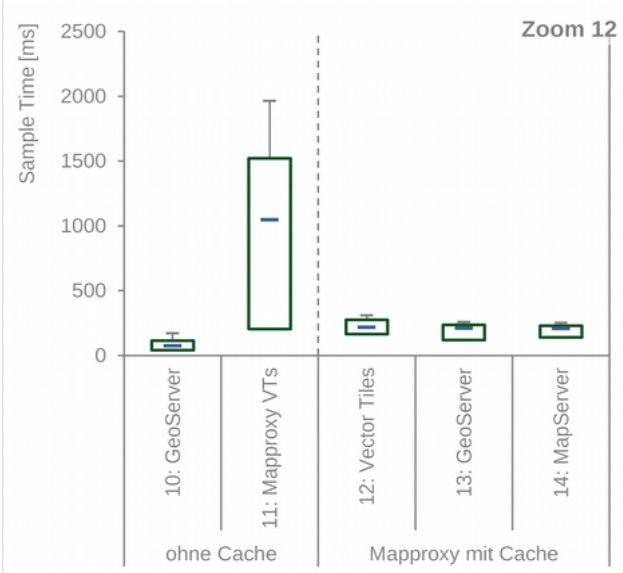

Zoom 7
Sample Time [ms]
2500
2000
1500
1000
500
0
10: GeoServer
11: Mapproxy VTs
12: Vector Tiles
13: GeoServer
14: MapServer
ohne Cache
Mapproxy mit Cache

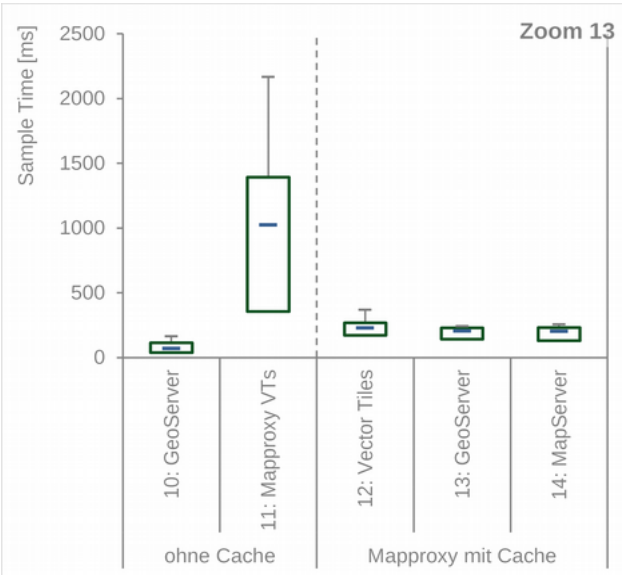

Zoom 8
Sample Time [ms]
2500
2000
1500
1000
500
0
10: GeoServer
11: Mapproxy VTs
12: Vector Tiles
13: GeoServer
14: MapServer
ohne Cache
Mapproxy mit Cache

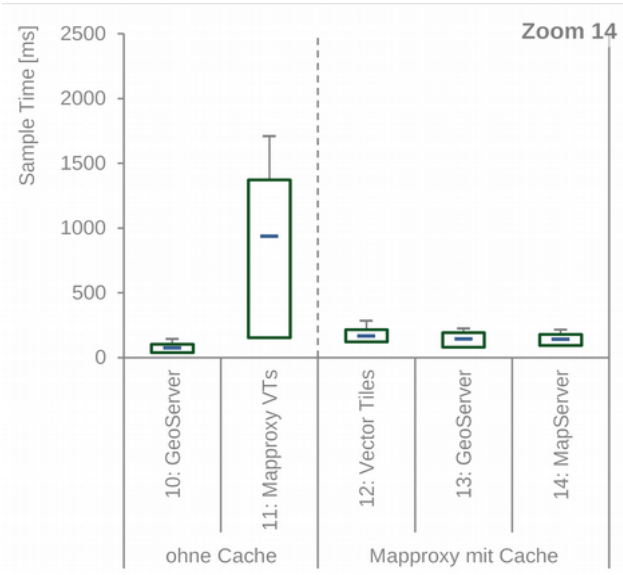

Zoom 9
Sample Time [ms]
2500
2000
1500
1000
500
0
10: GeoServer
11: Mapproxy VTs
12: Vector Tiles
13: GeoServer
14: MapServer
ohne Cache
Mapproxy mit Cache


Zoom 10
Sample Time [ms]
2500
2000
1500
1000
500
0
10: GeoServer
11: Mapproxy VTs
12: Vector Tiles
13: GeoServer
14: MapServer
ohne Cache
Mapproxy mit Cache


Zoom 11
Sample Time [ms]
2500
2000
1500
1000
500
0
10: GeoServer
11: Mapproxy VTs
12: Vector Tiles
13: GeoServer
14: MapServer
ohne Cache
Mapproxy mit Cache


Zoom 12
Sample Time [ms]
2500
2000
1500
1000
500
0
10: GeoServer
11: Mapproxy VTs
12: Vector Tiles
13: GeoServer
14: MapServer
ohne Cache
Mapproxy mit Cache


Zoom 13
Sample Time [ms]
2500
2000
1500
1000
500
0
10: GeoServer
11: Mapproxy VTs
12: Vector Tiles
13: GeoServer
14: MapServer
ohne Cache
Mapproxy mit Cache


Zoom 14
Sample Time [ms]
2500
2000
1500
1000
500
0
10: GeoServer
11: Mapproxy VTs
12: Vector Tiles
13: GeoServer
14: MapServer
ohne Cache
Mapproxy mit Cache