Räumliche Partitionierung von OpenStreetMap-Daten mit dem Ziel der Qualitätssicherung

Michael Reichert

Masterarbeit

am

Karlsruher Institut für Technologie (KIT)

Fakultät für Bauingenieur-, Geo- und Umweltwissenschaften
Geodätisches Institut (GIK)

Erstgutachter: Prof. Dr. rer. nat. Martin Breunig
Zweitgutachter: Prof. Dr.-Ing. Stefan Hinz
Betreuer: Dr.-Ing. Norbert Rösch

eingereicht am 23. Januar 2017
Selbstständigkeitserklärung


Karlsruhe, den 23. Januar 2017
Schreibweisen und Konventionen

Klassennamen sind in dieser Arbeit in serifenloser Schrift gesetzt und werden in allen Programmen, die im Rahmen dieser Arbeit geschrieben wurden, in CamelCase geschrieben.

Methodennamen sind ebenfalls in serifenloser Schrift gesetzt. Meistens werden aus Gründen der Lesbarkeit oft nur die Typen der Argumente, nicht ihre Namen angegeben. Für weitere Details zu den Methoden sei auf die im Anhang befindliche Klassendokumentation der einzelnen Programme verwiesen.

Auch für Klassenattribute wird serifenlose Schrift verwendet.

Die Namen von Datenbanktabellen und ihre Typen sind in serifenloser Schrift gesetzt.

SQL-Befehle und Befehle, die auf der Kommandozeile eingegeben werden müssen, sind in nichtproportionaler Schrift gesetzt. Variable Teile dieser Befehle sind *kursiv gesetzt*. Kommandozeilenbefehle beginnen mit einem Dollarzeichen.

Wird der Name eines Programms, einer Softwarebibliothek oder ein anderer besonderer Fachbegriff in einem Kapitel erstmals erwähnt, ist er *kursiv* gesetzt. Alle weiteren Verwendungen des Begriffs sind in dem Kapitel nicht mehr kursiv gesetzt, um den Text ruhiger wirken zu lassen.

Für OpenStreetMap wird an vielen Stellen die Abkürzung OSM verwendet.

Lizenz

Diese Arbeit ist verfügbar unter den Bedingungen der Lizenz Creative Commons Namensnennung Weitergabe unter gleichen Bedingungen 2.0 oder neuer. Hiervon ausgenommen sind:

- Die Klassendokumentation unterliegt den Bedingungen des jeweiligen Programms. Hierfür sei auf die Datei COPYING.md, die dem jeweiligen Programm beiliegt, verwiesen.

- Die Karten im Textteil dieser Arbeit verwenden ausschließlich die Daten des OpenStreetMap-Projekts. Die Daten unterliegen den Bedingungen der Open Database License 1.0. Die Kartengrafik einschließlich aller Overlays unterliegt den Bedingungen der Lizenz Creative Commons Namensnennung Weitergabe unter gleichen Bedingungen 2.0

- Da das originale Titelblatt nicht unter einer freien Lizenz verfügbar ist, ist es in dieser Ausgabe durch ein alternatives Titelblatt ersetzt worden, das unter derselben Lizenz wie die restliche Arbeit steht.
# Inhaltsverzeichnis

## Abbildungsverzeichnis

## Tabellenverzeichnis

## Listings

### 1. Technische Grundlagen und Motivation

1.1. Geschichte und Beweggründe von OpenStreetMap ........................................ 11
1.2. OpenStreetMap-Datenmodell ........................................................................... 12
   1.2.1. Tags ........................................................................................................ 12
   1.2.2. Nodes ..................................................................................................... 13
   1.2.3. Ways ...................................................................................................... 13
   1.2.4. Flächen .................................................................................................. 14
   1.2.5. Relationen ............................................................................................... 14
   1.2.6. Versionsgeschichte .................................................................................. 18
   1.2.7. Anderungssätze ...................................................................................... 18
1.3. Schnittstellen des OpenStreetMap-Projekts ..................................................... 19
1.4. Tiles ................................................................................................................ 20
1.5. Vektortiles ...................................................................................................... 21
1.6. Gebräuchliche Software zur Verarbeitung von OSM-Daten ................................ 23
1.7. Qualitätssicherung bei OpenStreetMap ........................................................... 26
   1.7.1. Keep Right ............................................................................................. 29
   1.7.2. Osmose .................................................................................................... 29
   1.7.3. OpenStreetMap Inspector ......................................................................... 31
   1.7.4. MapRoulette ........................................................................................... 33
   1.7.5. To-Fix ...................................................................................................... 34
1.8. Probleme bestehender Qualitätssicherungswerkzeuge ...................................... 34

### 2. Architektur

2.1. Datenbankserver ............................................................................................ 37
   2.1.1. Komponenten und Funktionsweise ......................................................... 37
   2.1.2. Warum werden die OSM-Daten in einer Datenbank gespeichert? .......... 37
2.2. Prozessierungsserver ..................................................................................... 39
2.3. Renderingserver ............................................................................................ 39

### 3. Cerepso

3.1. Gründe für die Entwicklung von Cerepso ......................................................... 40
3.2. Datenbankschema und Indexe ....................................................................... 42
<table>
<thead>
<tr>
<th>Inhaltsverzeichnis</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.3. Optimierung des Datenbankzugriffs</td>
</tr>
<tr>
<td>3.4. Funktionsweise</td>
</tr>
<tr>
<td>3.4.1. Initialer Import</td>
</tr>
<tr>
<td>3.4.2. Diff-Import</td>
</tr>
<tr>
<td>3.5. Tile Expiry</td>
</tr>
<tr>
<td>3.5.1. Prinzip der Tile Expiry und derzeit vorhandene Software</td>
</tr>
<tr>
<td>3.5.2. Implementierung in Cerepso und Herausforderungen</td>
</tr>
<tr>
<td>4. Partitionierung</td>
</tr>
<tr>
<td>4.1. Das Partitionierungsverfahren</td>
</tr>
<tr>
<td>4.2. Wechselwirkungen zwischen Inhalt und Häufigkeit</td>
</tr>
<tr>
<td>4.3. Größe der Tiles</td>
</tr>
<tr>
<td>4.4. Erzeugung eines Vektortiles</td>
</tr>
<tr>
<td>4.5. Herausforderungen bei der Partitionierung in kleine Teile</td>
</tr>
<tr>
<td>4.6. Kompromisse und Lösungen</td>
</tr>
<tr>
<td>4.6.1. Unvollständige Referenzen</td>
</tr>
<tr>
<td>4.6.2. Nodes ohne Tags</td>
</tr>
<tr>
<td>4.6.3. Performanceeinbußen durch vollständige Relationen</td>
</tr>
<tr>
<td>4.7. Validierung unvollständiger Relationen</td>
</tr>
<tr>
<td>4.7.1. Multipolygone</td>
</tr>
<tr>
<td>4.7.2. Administrative Grenzen</td>
</tr>
<tr>
<td>4.7.3. Routenrelationen</td>
</tr>
<tr>
<td>4.7.4. Abbiegebeschränkungen</td>
</tr>
<tr>
<td>5. Cerepso2vt</td>
</tr>
<tr>
<td>5.1. Programmablauf und Architektur</td>
</tr>
<tr>
<td>5.2. Die Erzeugung eines Vektortiles</td>
</tr>
<tr>
<td>5.3. Benutzung</td>
</tr>
<tr>
<td>5.4. Hstore- und Array-Parser</td>
</tr>
<tr>
<td>5.4.1. PostgresParser als Basis des Array- und Hstore-Parsers</td>
</tr>
<tr>
<td>5.4.2. Hstore-Parser</td>
</tr>
<tr>
<td>5.4.3. Array-Parser</td>
</tr>
<tr>
<td>6. Ausblick</td>
</tr>
<tr>
<td>6.1. Cerepso</td>
</tr>
<tr>
<td>6.2. Partitionierung</td>
</tr>
<tr>
<td>6.3. Cerepso2vt</td>
</tr>
<tr>
<td>6.4. Prozessierungs- und Renderingserver</td>
</tr>
</tbody>
</table>

A. Beispiel einer Buslinie nach dem Public-Transport-Schema 2 | 86 |

B. Versionsgeschichte eines Ways, die verborgene Version enthält | 88 |

C. OpenStreetMap Change XML | 89 |
Abbildungsverzeichnis

1.2. Topologische Änderung ohne geometrische Änderung ........................................ 20
1.3. Adressen der Tiles auf Zoomstufe 8 ....................................................................... 21
1.4. Adressierungsschema der Knoten eines Quadtrees ............................................... 22
1.5. Rastertiles und Vektortiles im Vergleich ............................................................... 22
1.6. Beispiel zweier unverbundener Straßen ................................................................ 28
1.7. Benutzeroberfläche von Keep Right ...................................................................... 30
1.8. Benutzeroberfläche von Osmose .......................................................................... 31
1.9. Ablaufschema OpenStreetMap Inspector .............................................................. 32
1.10. Benutzeroberfläche des OpenStreetMap Inspectors ............................................. 33
1.11. Zeiträume der Sichtbarkeit von Fehlern beim OpenStreetMap Inspector ............... 35

2.1. Entworfene Architektur für ein Qualitätssicherungswerkzeug mit stündlichen Updates ................................................................................ 36

3.1. UML-Klassendiagramm von Cerepso ohne Tile Expiry .......................................... 45
3.2. Speicherverbrauch beim Import eines Planetdumps ............................................. 46
3.3. UML-Klassendiagramm von Cerepso mit Fokus auf die Tile Expiry ......................... 50
3.4. Erforderlichkeit, Ways bei der Tile-Expiry zu berücksichtigen ................................. 51

4.1. Resultierende Partitionierung im Bereich der Elbe- und Wesermündung ................. 54
4.2. Markierungen von Tiles auf Zoomstufe 12 als veraltet im Nordosten Deutschlands .............................................................................................................. 58
4.3. Markierungen von Tiles auf Zoomstufe 12 als veraltet in Nordostfrankreich und Südwestdeutschland ......................................................... 59
4.4. Markierungen von Tiles auf Zoomstufe 12 als veraltet ohne Berücksichtigung von Relationen .................................................... 61
4.5. Markierungen auf Zoomstufe 12 als veraltet in Kalifornien ................................... 62
4.6. Markierungen von Tiles als veraltet im mittleren Lateinamerika auf Zoomstufe 12 .............................................................................................................. 62
4.7. zeitliche Entwicklung der Tile-Expiry im Lauf von 24 Stunden ................................ 64
4.8. Beispiel für die Bildung einer falschen Geometrie eines Ways, dessen Nodes nicht vollständig vorhanden sind ........................................... 67
4.9. sich überschneidende Ringe eines Multipolygons .................................................. 69
4.10. Überlappung eines inneren und äußeren Ringes eines Multipolygons ................... 69
4.11. nicht geschlossener innerer Ring eines Multipolygons ........................................... 70
4.12. Unmöglichkeit der Suche von Verstößen gegen die Hierarchie-Regel zweier administrativer Grenzen, wenn die Schnittmenge zweier ihrer Ringe ein LineString ist ................................................. 71

5.1. vereinfachtes UML-Klassendiagramm von Cerepso2vt ...................................... 75
5.2. UML-Klassendiagramm der Bibliothek pg-array-hstore-parser .......................... 80
Tabellenverzeichnis

1.1. gemeinsame Spalten der Geometrie-Tabellen von osm2pgsql-Datenbanken 24
1.2. Felder, die bei osm2pgsql nur in einem Teil der Geometrie-Tabellen verfügbar sind 25
1.3. gemeinsame Spalten der Slim-Tabellen von osm2pgsql-Datenbanken 25
1.4. Felder, die bei osm2pgsql nur in einem Teil der Slim-Tabellen verfügbar sind 25

2.1. gemeinsame Felder aller Spalten von Datenbanken, die mit Cerepso importiert wurden 42
2.2. Felder, die nur in einem Teil der Tabellen von mit Cerepso importierten Datenbanken verfügbar sind 43
## Listings

1.1. Beispiel für einen Node in der OSM-XML-Darstellung ........................................ 13
1.2. Beispiel für einen Way in der OSM-XML-Darstellung ........................................ 14
1.3. Beispiel für eine Relation vom Typ Multipolygon in der OSM-XML-Darstellung .......... 15
1.4. Beispiel für die Metadaten eines Änderungssatzes in der OSM-XML-Darstellung ........ 18
1.5. Beispiel für einen qualitativ schlechten Import .................................................. 28

4.1. alphabetisch sortierte Ausgabe von *dense_tiles* für die Zoomstufe 9 .................. 55
4.2. sortierte Ausgabe von *dense_tiles* für die Zoomstufe 10 .................................... 55
4.3. Inhalt von Beispieldatei 1 (Eingabe für *comm* in Listing 4.5) .......................... 56
4.4. Inhalt von Beispieldatei 2 (Eingabe für *comm* in Listing 4.5) .......................... 56
4.5. Beispielhafte Ausgabe von *comm* ...................................................................... 56
4.6. SQL-Abfrage zur Erstellung einer Heatmap der Tile-Expiry ................................. 59
4.7. SQL-Abfragen, die prinzipiell zur Erstellung eines Vektortiles notwendig wären .......................................................... 65

5.1. Die wichtigsten Teile der Klasse ArrayParser ....................................................... 82
5.2. Die Klasse TypeConversion .................................................................................. 82
1. Technische Grundlagen und Motivation

1.1. Geschichte und Beweggründe von OpenStreetMap

Das OpenStreetMap-Projekt (abgekürzt OSM) wurde von Steve Coast im Sommer 2004 in London gegründet. Ziel war es, Daten für eine freie Karte zu sammeln. Die damals verfügbaren Karten waren entweder teuer oder waren nicht unter einen freien Lizenzvertrag für beliebige Verwendung verfügbar. Hinter OpenStreetMap steht also eine ähnliche Motivation wie hinter freier Software – der Wunsch, damit tun zu dürfen, was man will [21].


Hinter dem OpenStreetMap-Projekt steht seit 2006 die OpenStreetMap Foundation, welche die Server betreibt und seit dem Wechsel von der Lizenz Creative Commons Namensnennung Weitergabe unter gleichen Bedingungen 2.0 zur Open Database License 1.0 (ODbL) im September 2012 auch die Rechte an den Daten hält und sie unter den Bedingungen der ODbL jedem zur Verfügung stellt [12, 21, 121].

OpenStreetMap hat zwar, anders als die Wikipedia, keine Relevanzkriterien, aber erfasst nur Dinge, die vor Ort beobachtbar sind, und keine direkt personenbezogenen Daten wie Klingelschilder. Ausnahmen von dieser Regel existieren nur für die Dinge, die man für eine Karte wichtig erachtet, wie z. B. administrative Grenzen [21, 120, 122].

OpenStreetMap fing vielerorts bei Null an, nur in einigen Gegenden wurde in frühen Jahren durch den Import von Daten aus Fremdquellen eine gleichmäßige Grundabdeckung erreicht. In den anderen Gegenden (v. a. in Europa) hat sich eine schlagkräftige Community gebildet, die die Daten manuell vor Ort erfasst und deren Schwerpunkt im deutschsprachigen Raum liegt (von Projektbeginn bis Dezember 2011 hatten 31 Prozent aller Benutzer überwiegend in Deutschland, Österreich oder der Schweiz Daten erfasst [76]).


In England hatte OpenStreetMap im Jahr 2010 in urbanen Gebieten bereits ein längeres Wegenetz als der Datensatz Meridian 2 der Ordnance Survey (britische Vermessungsverwaltung) [47]. Einem Vergleich von Neis et. al. aus dem Jahr 2011 zufolge hatte OSM zwar ein neun Prozent kleineres Kfz-Straßennetz als der Multinet-Datensatz von TomTom, aber das gesamte Netz war 27 Prozent, das für Fußgänger geeignete Wegenetz 31 Prozent
Technische Grundlagen und Motivation

Außerhalb von Ballungsräumen und touristisch stark frequentierten Gebieten war OpenStreetMap doch ziemlich lückenhaft (TomTom hatte dort ein bis zu 80 Prozent längeres Netz) \(^7\). Das Projekt hat seine Wurzeln in der Freie-Software-Bewegung. Freie Daten zu Schaffen, war eine logische Schlussfolgerung. Die Community organisiert sich auf Mailinglisten, in Foren, dem OpenStreetMap-Wiki und weiteren Diskussionskanälen, u. a. diversen IRC-Kanälen \(^4\), \(^8\), \(^5\). Für Ruhm und Ansehen einer Person in der Community zählen daher nicht seine wissenschaftlichen Veröffentlichungen, sondern was derjenige praktisch als Mapper und/oder Software-Entwickler geleistet hat. Wissenschaftliche Untersuchungen haben sich in der Vergangenheit meist mit der Datenquantität und -qualität sowie der Herkunft und Aktivität der Mitwirkenden beschäftigt. Technische Aspekte werden selten im Detail beleuchtet oder nur in Form studentischer Abschlussarbeiten. Letztere haben kein besonders großes Ansehen, da viele ihrer Verfasser nach Abschluss der Arbeit das Interesse am Thema verlieren.

Viele Quellen dieser Arbeit sind daher nicht wissenschaftliche Veröffentlichungen, sondern meist Beiträge in den Diskussionskanälen der Community sowie die Dokumentation und in einigen Fällen auch der Quellcode veröffentlichter freier Software. Hinzu kommt an einigen Stellen die Erfahrung des Autors als Mitwirkender, welche sich leider schwerlich zitieren lässt.

1.2. OpenStreetMap-Datenmodell

Bei OSM gibt es drei verschiedene Objektarten – Nodes, Ways und Relationen. All diese Objektarten können, müssen aber nicht, Tags haben.


Das Datenmodell folgt dem KISS-Prinzip (Keep it simple, stupid) und ist speziell auf die Bedürfnisse eines Community-Projekts angepasst. Es gibt nur sehr wenige feste Vorgaben, wie bestimmte Dinge zu erfassen sind. Die verschiedenen Objektarten (Nodes, Ways, Relationen) orientieren sich nicht an gebräuchlichen GIS-Standards. Die Einstiegshürde soll für neue Mapper und Entwickler möglichst niedrig sein, denn ohne (neue) Mapper und Entwickler von Drittapplikationen hat das Projekt keine Zukunft \(^2\).

1.2.1. Tags

Mittels Tags wird festgelegt, was das jeweilige OSM-Objekt repräsentiert. Bei Tags handelt es sich nicht um Kategorien, wie sie bei anderen Web 2.0-Diensten üblich sind. Tags

\(^1\) Internet Relay Chat
sind bei OpenStreetMap Key-Value-Paare, die frei wählbar sind. Key und Value dürfen jeweils bis zu 256 Zeichen lang sein [105].

Es gibt keine festen, verbindlichen Regeln, welche Tags zu verwenden sind. Da jedoch unzählige Anwendungen OSM-Daten nutzen und eine uneinheitliche Erfassung der weiteren Verarbeitung nicht förderlich ist, haben sich gewisse Regeln herausgebildet. Gebräuchliche Tags sind in der Liste Map Features im OSM-Wiki dokumentiert [63]. Dennoch gilt bei OpenStreetMap der Grundsatz Any tags you like, d. h. man kann selbst beliebige Tags definieren [16]. Das Datenmodell ist an dieser Stelle absichtlich sehr flexibel gehalten, damit frühe Entwurfsentscheidungen die Entwicklung des Projekts nicht negativ beeinflussen und das Projekt agil bleibt. Beispiele für gebräuchliche Tags sind:

- highway = residential für Wohnstraßen
-  shop = supermarket für einen Supermarkt
-  name = Kaiserstraße für ein Objekt mit dem Namen „Kaiserstraße“

1.2.2. Nodes

Punkte werden in OSM als Nodes bezeichnet. Sie sind die einzigen Objekte in OSM, welche Koordinaten haben. Sie repräsentieren entweder ein punktförmiges Objekt (z. B. einen Baum oder einen Briefkasten) oder dienen als Stützpunkt für einen Way. In letzterem Fall haben sie meist keine Tags. Listing 1.1 zeigt einen als Node erfassten Supermarkt in der OSM-XML-Repräsentation.


```xml
<osm version="0.6" generator="CGImap 0.5.8 (26682 thorn-03.openstreetmap.org)"
  copyright="OpenStreetMap and contributors" attribution="http://www.openstreetmap.org/copyright" license="http://opendatacommons.org/licenses/odbl/1-0/">
  <node id="4400868292" visible="true" version="1" changeset="42144099" timestamp="2016-09-14T08:55:21Z" user="wegavision" uid="564585" lat="49.4365025" lon="8.6790851">
    <tag k="name" v="Rewe"/>
    <tag k="opening_hours" v="Mo-Sa 08:00-24:00"/>
    <tag k="shop" v="supermarket"/>
    <tag k="wheelchair" v="yes"/>
  </node>
</osm>
```

1.2.3. Ways


---

Listing 1.2: Beispiel für einen Way in der OSM-XML-Darstellung (gekürzt) [2]

```
<osm>
  <way id="148588529" visible="true" version="5" changeset="22986547" timestamp="2014-06-17T14:57:04Z" user="Uli Maier" uid="729531">
    <nd ref="24956331"/>
    <nd ref="1426415783"/>
    <nd ref="1428967683"/>
    <nd ref="1426415811"/>
    <nd ref="26974332"/>
    <nd ref="1426415812"/>
    <nd ref="26974333"/>
    <tag k="access" v="destination"/>
    <tag k="bicycle" v="yes"/>
    <tag k="foot" v="yes"/>
    <tag k="highway" v="service"/>
    <tag k="name" v="Besoldgasse"/>
    <tag k="width" v="2.4"/>
  </way>
</osm>
```

1.2.4. Flächen

Flächen gibt es im OpenStreetMap mit der aktuellen API 0.6 nicht. Flächen, die nur aus einem äußeren Ring bestehen, der maximal 2000 Punkte hat, werden meist als geschlossene Ways modelliert (erster und letzter Node sind identisch). Alle anderen Flächen werden als Relationen mit dem Tag type=multipolygon modelliert. Ob es sich wirklich um eine Fläche und nicht um eine ringförmige Struktur (z. B. Ringstraße) handelt, kann nur anhand der Tags ermittelt werden. Zwar gibt es mit area=yes ein Tag zur Kennzeichnung von Flächen, dieses wird aber nur dort verwendet, wo es nicht eindeutig aus anderen Tags hervorgeht, z. B. linienförmige vs. flächenförmige Bahnsteige, Zufahrtswege vs. Hofflächen.

1.2.5. Relationen

Relationen (engl. relations) werden für Objekte und Zusammenhänge verwendet, die sich nicht einfach mit Nodes, Ways und Tags modellieren lassen. Relationen können Tags und eine geordnete Liste an Mitgliedern (engl. members) haben. Die Mitglieder haben Rollen (engl. roles), welche Strings sind (Strings mit der Länge 0 sind möglich). Das Tag type gibt den Relationstyp an. Gebräuchliche Typen sind multipolygon, boundary, route und turn_restriction. Diese Aufzählung ist nicht abschließend.

**Multipolygone**  Relationen mit dem Tag type=multipolygon für Flächen, die nicht durch einen einigen Outer-Ring modelliert werden können. Die Mitglieder einer Multipolygon-Relation sind alle äußeren und inneren Ringe, also die Ways, die diese repräsentieren. Äußere Ringe haben die Rolle outer, innere Ringe die Rolle inner. Ein Ring kann aus einem oder mehreren Ways bestehen. Die Tags, die die Eigenschaften der Fläche
Technische Grundlagen und Motivation

Listing 1.3: Beispiel für eine Relation vom Typ Multipolygon in der OSM-XML-Darstellung (gekürzt) [3]

```xml
<osm>
  <relation id="5489215" visible="true" version="1" changeset="33838993" timestamp="2015-09-06T18:20:47Z" user="mikecc" uid="14034">
    <member type="way" ref="133745599" role="outer"/>
    <member type="way" ref="369409185" role="inner"/>
    <tag k="landuse" v="forest"/>
    <tag k="leaf_cycle" v="semi_deciduous"/>
    <tag k="leaf_type" v="mixed"/>
    <tag k="name" v="Herrenwald"/>
    <tag k="type" v="multipolygon"/>
  </relation>
</osm>
```

beschreiben, hängen am Relationsobjekt [105][107]. Insgesamt gibt es etwa 2,5 Millionen Multipolygon-Relationen in OpenStreetMap [118].


Im OSM-Wiki findet sich die Beschreibung, wann ein Multipolygon gültig ist. Im Prinzip entspricht das der Simple-Features-Spezifikation der OGC. Darüber hinaus ist es bei OSM jedoch erlaubt, dass zwei innere Ringe sich nicht nur in einem Punkt berühren, sondern eine gemeinsame Kante haben dürfen [107].


**Grenzen** Relationen mit dem Tag type=boundary werden für administrative Polygone, Schutz- und Sperrgebiete verwendet. Sie sind wie Multipolygonrelationen aufgebaut. Für Administrative Grenzen gibt es den Key admin_level mit den Werten 1 bis 11. Gebräuchlich sind in Deutschland 2 für die Grenze der Bundesrepublik, 4 für die Länder, 6 für Landkreise und höhere Werte für Gemeinden und deren Ortsteile [112]. Es ist
zwar nicht im OpenStreetMap-Wiki dokumentiert, aber einige Mapper fordern, dass Grenzrelationen streng hierarchisch und möglichst überlappendungsfrei (ausgenommen Kondomiaten) sein müssen. Das hieße, eine Fläche des Admin-Levels $n$ muss alle Flächen mit Admin-Level $> n$, die mit ihr eine nicht leere Schnittmenge haben, vollständig umgeben. Ein Landkreis muss eine Gemeinde vollständig umgeben, ein Staat muss seine Provinzen/Bundesländer vollständig umgeben. Sie dürfen nicht aus ihm „herausragen“.

**Routenrelationen** Relationen mit dem Tag `type = route` repräsentieren eine Route, z. B. einen markierten Wanderweg, den Verlauf einer Bundesstraße oder eine Buslinie. Routenrelationen referenzieren alle Ways, die Teil der Route sind. Wenn die Route Verzweigungen oder alternative Wege hat, werden auch diese alternativen Varianten in die Relation mit aufgenommen. Die Mitglieder haben normalerweise keine Rolle (d. h. eine leere Rolle). Wenn ein Way jedoch nur in Richtung des Ways benutzt wird (beispielsweise eine Route über eine Straße mit getrennten Richtungsfahrbahnen), erhält der Way die Rolle `forward`. Wird er ausschließlich entgegen seiner Richtung benutzt,

---

3 gemeinsam verwaltete Gebiete

erhält er die Rolle backward. Die Mitglieder einer Routenrelation sind nicht geordnet, auch wenn auf Seiten der OpenStreetMap-Datenbank Relationen Listen und keine Sets sind.

Abbildung 1.1 zeigt ein fiktives Beispiel einer solchen Routenrelation. Die Ways 14 und 15 werden nur in ihrer eigenen Richtung (d. h. vom ersten zum letzten Node, nicht umgekehrt) befahren und haben daher die Rolle forward. Der Weg über Node 3 wird von ein oder mehreren Kursen in beide Richtungen bedient. Es ist aber nicht erkennbar, ob diese Kurse nur von Node 1 bis 4 oder die gesamte Strecke fahren.


Im PTv2-Schema ist die Reihenfolge der Mitglieder relevant. Am Anfang der Mitgliederliste stehen die bedienten Haltestellen. Dies kann entweder die Halteposition (ein Node auf dem Fahrweg) mit der Rolle stop oder die Plattform mit der Rolle platform oder beides sein. Wenn an einer Haltstelle nur die Halteposition oder nur die Plattform erfasst ist, wird auch nur diese in die Relation aufgenommen. Das Schema verlangt nicht, dass beide Objekte erfasst sein müssen. Existieren sie aber, müssen sie in die Relation aufgenommen werden. Die Liste der Haltestellen muss geordnet sein.

Der Haltestellenliste folgt die Liste der befahrenen Ways, auch diese muss geordnet sein. Pro Variante wird eine Routenrelation erfasst, d. h. die Liste der befahrenen Ways muss einen durchgehenden, lückenlosen Linestring ergeben. Wenn das Fahrzeug Stichfahrten macht (z. B. Fahrten über Kopfbahnhöfe), sind die Ways im Bahnhofsbereich doppelt in der Mitgliederliste vorhanden, weil sie zweimal befahren werden. Im Anhang A ist die PTv2-Modellierung der in Abbildung 1.1 gezeigten Buslinie abgedruckt.

Abbiegebeschränkungen Abbiegebeschränkungen werden als Relationen mit dem Tag type = restriction modelliert. Das Tag restriction gibt an, ob es sich um ein Verbot oder ein Gebot handelt. Beginnt der Wert des Tags mit no_, ist es ein Verbot. Beginnt er mit only_, ist es ein Gebot. Bei Geboten gibt die Relation an, welches das einzige erlaubte Abbiegemanöver an der Kreuzung ist. Abbiegebeschränkungs-Relationen haben mindestens drei Rollen:

- einen Way mit der Rolle from, der den Way repräsentiert, von dem das Fahrzeug kommt
- einen Node mit der Rolle via, der den Node der Kreuzung repräsentiert
- einen Way mit der Rolle to, der den Way repräsentiert, in den das Fahrzeug nicht hineinfahren darf (Verbot) bzw. hineinfahren muss (Gebot)

4bei einfachen Bushaltestellen oft die Position des Haltestellenschilds
1. Technische Grundlagen und Motivation

1.2.6. Versionsgeschichte


1.2.7. Änderungssätze


Listing 1.4: Beispiel für die Metadaten eines Änderungssatzes in OSM-XML-Darstellung [5]

```xml
<osm version="0.6" generator="CGImap 0.5.8 (1192 thorn-01.openstreetmap.org)"
    copyright="OpenStreetMap and contributors" attribution="http://www.openstreetmap.org/copyright"
    license="http://opendatacommons.org/licenses/odbl/1-0/">
    <changeset id="43080643" created_at="2016-10-22T12:52:27Z" closed_at="2016-10-22T12:52:28Z" open="false" user="fred_ka" uid="190617" min_lat="49.4726510" min_lon="8.4694250" max_lat="49.4729292" max_lon="8.4696162"
        comments_count="0">
        <tag k="comment" v="added store info"/>
        <tag k="created_by" v="JOSM/1.5 (10786 en)"/>
        <tag k="source" v="visit on site"/>
    </changeset>
</osm>
```
Da die Benutzer selbst festlegen, was sie in einem Änderungssatz zusammenfassen und es nur die Beschränkung gibt, maximal 50 000 Objekte in einem Änderungssatz hochzuladen [17], gibt es auch Änderungssätze, die sehr groß sind (die halbe Welt umspannen), keinen Kommentar haben oder deren Änderungen nichts miteinander zu tun haben [105].

1.3. Schnittstellen des OpenStreetMap-Projekts

Planet Der Planet, auch Planetdump genannt, ist ein wöchentlich erstelltes Abbild der OSM-Datenbank, das alle nicht gelöschten Objekte in ihrer aktuellen Version enthält. Er ist im OSM-eigenen (BZIP2-komprimierten) XML-Format und als Protocol Buffers erhältlich [40, 95, 105].

Dritte, wie die Geofabrik GmbH oder Mapzen, bieten regionale Extrakte des Planets an, da die Verarbeitung des gesamten Planet mit derzeit 33 GB gewisse Anforderungen an die Hardware stellt [95].

Neben dem Planetdump, der nur die neuste Version aller nicht gelöschten Objekte enthält, stellt die OpenStreetMap Foundation auch noch einen Full History Planet Dump bereit, welcher alle Versionen aller Objekte (auch gelöschte Objekte) enthält. Desweiteren existieren noch Dumps aller Änderungssätze im XML-Format (siehe Listing [1.4]), aller Änderungssatz-Diskussionen (Änderungssätze können von anderen Benutzern kommentiert werden [27]) und aller Fehlerberichte (Notes) [40].

Diffs Zum Planetdump werden minütliche, stündliche und tägliches Updates im OSM-Change-XML-Format (siehe Anhang GZIP-komprimiert veröffentlicht. Die Diffs enthalten die neuen Versionen von OSM-Objekten, die seit dem letzten Diff geändert wurden [96].


API Die OpenStreetMap-API (oft einfach nur die API) ist die älteste und offizielle Schnittstelle von OpenStreetMap. Sie bestimmt das Datenformat und ist heutzutage nur noch für Editoren gedacht, also Anwendungen zum Bearbeiten von OpenStreetMap-Daten. Seit April 2009 ist die Version 0.6 im Einsatz [17]. Es gibt Pläne für eine Version 0.7 und erste Arbeiten daran [15], jedoch befriedigt die API 0.6 die meisten Bedürfnisse und hat anders als ihre Vorgängerversionen eine ausreichende Reife. Die API selbst wird in dieser Arbeit nicht genutzt.
1. Technische Grundlagen und Motivation

Abbildung 1.2.: Topologische Änderung ohne geometrische Änderung – der Way referenziert nach der Änderung (rechtes Bild) auch den Node 2, welcher selbst aber nicht bearbeitet wurde

**Overpass-API** Die Overpass-API ist eine zwar nicht offizielle, von der OpenStreetMap Foundation betriebene, aber sehr gebräuchliche Schnittstelle, über die kleine Auszüge aus dem OSM-Daten bezogen werden können. Die Schnittstelle hat eine eigene Abfragesprache, über die thematische und räumliche Abfragen möglich sind. Sie erspart Datennutzern, die eine einfache Spezialkarte im Web betreiben wollen (z. B. eine Karte aller Hotels [73]), den Betrieb einer eigenen PostGIS-Datenbank [93]. Stattdessen stellt der Client die Abfragen an die Overpass-API. Sie ist in dieser Arbeit nicht von weiterer Relevanz.

1.4. Tiles


Da die Tiles vom Server zwischengespeichert werden, sind sie nur in einigen festen Maßstäben verfügbar. Diese Maßstäbe werden meist Zoomstufen (engl. zoom levels) genannt. Der kleinste Maßstab ist die Zoomstufe 0, auf der die Welt aus einem einzigen Tile besteht. Mit jeder höheren Zoomstufe verdoppelt sich die Anzahl der Kacheln in x- und y-Richtung. Die Zoomstufe \( n \) hat also \( 2^{2n} \) Tiles [110].

Damit auch an den Rändern der Tiles Beschriftungen gerendert werden können, wird nicht nur der Inhalt des Tiles, sondern auch ein Buffer um das Tile herum abgefragt. Um das Verhältnis von Nutzfläche zur abgefragten Fläche zu verbessern, arbeiten viele Tileserver mit Meta-Tiles. Ein Meta-Tile wird gemeinsam gerendert und besteht meistens aus 8 × 8 Tiles [70].


Bing Maps verwendet zur Adressierung der Tiles sogenannte Quadkeys. Dabei handelt es sich um die Adressen, die bei Quadtrees verwendet werden. Durch jedes Aufteilen des Raumes wird ein Bit hinten an die Adresse angehängt. Der Raum wird abwechselnd in y- und x-Richtung aufgeteilt. Abbildung 1.4 zeigt, wie ein Tile der Zoomstufe 0 in vier Tile der Zoomstufe 1 (00₂, 01₂, 10₂ und 11₂) aufgeteilt wird. Beim erneuten Aufteilen dieser Tiles, verlängert sich die Adresse erneut um zwei Ziffern. Die Zoomstufe eines solchen Quadkeys ist $l \over 2$ (l sei die Länge des Quadkeys).

Um die Darstellung eines Quadkeys zu verkürzen, ist bei Bing Maps die Darstellung als Zahl zur Basis 4 gebräuchlich, d. h. aus 1110₂ wird 32₄ [110].

1.5. **Vektortiles**

Bei Vektortiles wird zwischen dem gerasterten Tile und der Datenquelle (meist einer PostGIS-Datenbank) eine Zwischenschicht eingeführt. Anders als der Name es vielleicht vermuten lässt, handelt es sich bei Vektortiles nicht um Vektorgrafiken (z. B. SVG, welches
1. Technische Grundlagen und Motivation

Abbildung 1.4.: Adressierungsschema der Knoten eines Quadtrees

mittlerweile die gebräuchlichen Webbrowser unterstützen), sondern um Vektordaten ohne Angaben, wie diese darzustellen sind.

Die Nutzung von Vektortiles in der Webkartographie kann auf zwei verschiedene Weisen erfolgen. Entweder erhält der Client statt einer Rastergrafik Vektordaten und das Rendering erfolgt clientseitig oder die Vektortiles werden serverseitig abgelegt und on-the-fly gerendert (vgl. Abbildung 1.5). Im ersten Fall kann der Client entscheiden, mit welchem Kartenstil er die Vektordaten rendern möchte und somit direkt Einfluss auf das Erscheinungsbild (Featureselektion, Renderingreihenfolge, Farben, Linienstärken, ...) nehmen. Im zweiten Fall erfolgt das Rendering serverseitig, jedoch wird der Tile-Cache flexibler. Es ist dann möglich, mit einem Server mehrere Kartenstile auszuliefern, da nicht mehr für jeden auszuliefernden Kartenstil die Rastertiles, sondern die rohen Vektordaten vorgehalten werden [36, 64, 69]. Bei serverseitigem Rendering müssen die

Abbildung 1.5.: Rastertiles und Vektortiles im Vergleich
Clients nicht angepasst werden und die Hardware-Anforderungen an die Clients sind niedriger. Damit das clientseitige Renderung von Vektortiles performant ist, müssen in kleinen Maßstäben die Geometrien vereinfacht werden, damit die Dateigröße der Tiles nicht zu groß und die Anzahl der Stützpunkte nicht zu hoch ist [68].


Zur Qualitätssicherung eignen sich diese für die Kartographie gedachten Formate nicht oder nur bedingt. Sie alle haben gemeinsam, dass sie nicht mit den originären OSM-Objekten, sondern davon abgeleiteten Geometrien arbeiten. Die Vektortiles enthalten Punkte, Linestrings und Polygone. Linestrings und Polygone haben Listen von Koordinatenpaaren anstatt Nodes zu referenzieren, was das Rendering vereinfacht. Es kann daher nicht geprüft werden, ob zwei Ways denselben Node referenzieren, da die Linestrings in diesen Formaten für jeden Punkt dessen Koordinaten und nicht die ID des Punkts speichern. Damit zwei Ways in OSM jedoch als verbunden gelten, müssen sie denselben Node referenzieren.

Mapbox bietet zwar mit den QA-Tiles spezielle Vektortiles kostenlos für Qualitätssicherungszwecke an, diese sind jedoch für viele Fälle nicht ausreichend. Es handelt sich dabei um GeoJSON, dessen Geometrien nicht vereinfacht wurden und das die unveränderten OSM-Tags enthält. Sie enthalten keine Relationen und statt der Node-Referenzen enthalten die Ways die Koordinaten ihrer Stützpunkte. Die QA-Tiles werden als mbtile-Pakete zum Download angeboten. mbtile ist ein von Mapbox entworfenes Austauschformat, welches die einzelnen Tiles als Binary Large Objects (Blobs) in einer SQLite-Datenbank enthält. Die Blobs enthalten GeoJSON [46].

1.6. Gebräuchliche Software zur Verarbeitung von OSM-Daten


**PostgreSQL** ist ein freies Datenbankmanagementsystem, das sich im OpenStreetMap-Umfeld als Datenbankmanagementsystem durchgesetzt hat. Fast alle Software, die Daten in eine Datenbank schreibt oder aus einer Datenbank liest, nutzt PostgreSQL. Grund dafür ist nicht nur PostgreSQL selbst, sondern auch die Verfügbarkeit der mächtigen Erweiterung PostGIS, welche Datentypen und Funktionen für die Verarbeitung räumlicher Informationen bereitstellt. In dieser Arbeit kommt die PostgreSQL-Erweiterung hstore zum Einsatz, die den gleichnamigen Datentyp, einen Key-Value-Speicher bereitstellt.
PostgreSQL wird sowohl im Datenbank-Backend der OpenStreetMap-API als auch von Tileservern, Geocoding-Diensten wie Nominatim u. v. m. verwendet [53,105,113].

osm2pgsql ist ein ursprünglich in C geschriebenes und mittlerweile in C++ portiertes Programm zum Import von OSM-Daten in eine PostgreSQL-Datenbank mit der PostGIS-Erweiterung. Das Datenbankschema besteht aus vier Tabellen:

- **planet_osm_point** enthält punktförmige Features
- **planet_osm_line** enthält linienförmige Features
- **planet_osm_roads** enthält eine Auswahl an Objekten aus planet_osm_line, um das Rendern von Tiles auf niedrigen Zoomstufen zu beschleunigen. Dabei handelt es sich um wichtige Straßen (Landstraßen und höher) sowie Eisenbahngleise. Die Auswahl der Objekte in dieser Tabelle ist hart kodiert und orientiert sich an einem mehrere Jahre alten Zustand des Kartenstils OSM Mapnik.
- **planet_osm_polygon** enthält flächenförmige Features.


### Tabelle 1.1: gemeinsame Spalten der Geometrie-Tabellen von osm2pgsql-Datenbanken

<table>
<thead>
<tr>
<th>Feldname</th>
<th>Typ</th>
<th>Beschreibung</th>
</tr>
</thead>
<tbody>
<tr>
<td>osm_id</td>
<td>bigint</td>
<td>ID des OSM-Objekts</td>
</tr>
<tr>
<td>access</td>
<td>text</td>
<td>Wert des Tags des Objekts in OSM (Spaltenname = Key in OSM)</td>
</tr>
<tr>
<td>addr:housename</td>
<td>text</td>
<td></td>
</tr>
<tr>
<td>addr:housenumber</td>
<td>text</td>
<td></td>
</tr>
<tr>
<td>addr:interpolation</td>
<td>text</td>
<td></td>
</tr>
<tr>
<td>admin_level</td>
<td>text</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>text/real/int4/…</td>
<td>aus layer und anderen Tags berechnete Renderingreihenfolge</td>
</tr>
<tr>
<td>z_order</td>
<td>integer</td>
<td></td>
</tr>
</tbody>
</table>


- **planet_osm_nodes** enthält alle Nodes des importierten OSM-Datensatzes.
- **planet_osm_ways** enthält alle Ways des importierten OSM-Datensatzes.
- **planet_osm_relations** enthält alle Relationen des importieren OSM-Datensatzes.
1. Technische Grundlagen und Motivation

Tabelle 1.2.: Felder, die bei osm2pgsql nur in einem Teil der Geometrie-Tabellen verfügbar sind

<table>
<thead>
<tr>
<th>Feldname</th>
<th>Type</th>
<th>Tabellen</th>
<th>Beschreibung</th>
</tr>
</thead>
<tbody>
<tr>
<td>way</td>
<td>geometry(Point)</td>
<td>planet_osm_point</td>
<td>Geometrie</td>
</tr>
<tr>
<td>way</td>
<td>geometry(LineString)</td>
<td>planet_osm_line, planet_osm_roads</td>
<td>Geometrie</td>
</tr>
<tr>
<td>way</td>
<td>geometry(Geometry)</td>
<td>planet_osm_polygon</td>
<td>Geometrie</td>
</tr>
</tbody>
</table>

Tabelle 1.3.: gemeinsame Spalten der Slim-Tabellen von osm2pgsql-Datenbanken

<table>
<thead>
<tr>
<th>Feldname</th>
<th>Typ</th>
<th>Beschreibung</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>bigint</td>
<td>ID des OSM-Objekts</td>
</tr>
<tr>
<td>tags</td>
<td>text[]</td>
<td>Tags des OSM-Objekts, in folgender Reihenfolge: Key von Tag 1, Value von Tag 1, Key von Tag 2, Value von Tag 2, ...</td>
</tr>
</tbody>
</table>

Tabelle 1.4.: Felder, die bei osm2pgsql nur in einem Teil der Slim-Tabellen verfügbar sind

<table>
<thead>
<tr>
<th>Feldname</th>
<th>Type</th>
<th>Tabellen</th>
<th>Beschreibung</th>
</tr>
</thead>
<tbody>
<tr>
<td>lat</td>
<td>integer</td>
<td>planet_osm_nodes</td>
<td>geographische Breite in der Integer-Repräsentation</td>
</tr>
<tr>
<td>lon</td>
<td>integer</td>
<td>planet_osm_nodes</td>
<td>geographische Länge in der Integer-Repräsentation</td>
</tr>
<tr>
<td>nodes</td>
<td>bigint[]</td>
<td>planet_osm_ways</td>
<td>Nodes, die ein Way referenziert</td>
</tr>
<tr>
<td>parts</td>
<td>bigint</td>
<td>planet_osm_rels</td>
<td>IDs der referenzierten Objekte</td>
</tr>
<tr>
<td>members</td>
<td>text[]</td>
<td>planet_osm_rels</td>
<td>Typen, IDs und Rollen der referenzierten Objekte, z. B. w293249683,inner,w293249772,outer</td>
</tr>
</tbody>
</table>

Osmosis ist ein Java-Programm zum Lesen, Schreiben und Verändern (nach Tags filtern, räumliche Extrakte erzeugen, ...) von OSM-Daten. Es ist ein relativ vielseitiges Werkzeug, das auf keine Aufgabe spezialisiert ist. Zu seinen Funktionen gehört auch der Import von OSM-Daten in eine PostgreSQL-Datenbank. Dafür stehen drei Schemata zur Auswahl [90][91]:

Technische Grundlagen und Motivation


- Das pgsnapshot-Schema ersetzt dieses Schema. Tags werden nicht mehr in einer separaten Tabelle, sondern als hstore-Feld gespeichert [24] [32] [91].

Alle drei Schemata ermöglichen keinen direkten Zugriff auf die Ways anhand eines räumlichen Index. Stattdessen muss zuerst nach den Nodes gesucht werden und anschließend nach den Ways/Relationen, die diese Nodes referenzieren.

Osmium wird ab Version 2 auch libosmium genannt und ist eine C++-Bibliothek zum Lesen und Schreiben von OSM-Daten nach dem Streaming-Prinzip. Der Entwickler Jochen Topf legt Wert auf Schnelligkeit und geringen Speicherverbrauch. Bei Osmium werden die OSM-Objekte nicht auf dem Stack, sondern in sogenannten Buffern abgelegt, wodurch Platz gespart und der Zugriff beschleunigt wird. Um mit Osmium eine OSM-Datei einzulesen, wird eine Klasse implementiert, welche von der Klasse osmium::handler::Handler abgeleitet ist und deren Callbackmethoden implementiert:

- osmium::handler::Handler::void node(const osmium::Node&) verarbeitet Nodes.
- osmium::handler::Handler::void way(const osmium::Way&) verarbeitet Ways.

Osmium ruft die jeweils passende Callbackmethode des Handlers auf, sobald es ein neues Objekt aus der einzulesenden OSM-Datei gelesen hat.

Da ein Way nur Referenzen auf die Nodes, nicht aber deren Positionen selbst enthält, muss Osmium eine Map mit den IDs der Nodes als Schlüssel und der Position (Länge, Breite) als Wert im Speicher halten. Dafür bietet Osmium mehrere verschiedene Indexverfahren an. Einige sind für kleinere, andere für größere Extrakte bzw. den gesamten Planetdump geeignet. Die Auswahl kann zur Laufzeit erfolgen, Osmium stellt hierfür eine Factory bereit. [116] [117]

Im Rahmen dieser Arbeit wurden Lücken in der Dokumentation geschlossen und Jochen Topf als Pull-Request zur Übernahme angeboten. Für Details sei auf Anhang G verwiesen.

1.7. Qualitätssicherung bei OpenStreetMap

Änderungen eines Mappers werden sofort in die OSM-Datenbank übernommen, eine Sichtung gibt es bei OpenStreetMap nicht [31] [105]. Vorschläge zur Einführung eines Sichtungsverfahrens konnten sich in der OSM-Community nicht durchsetzen [92] [119]. OSM ist anders als die Wikipedia deutlich weniger von Vandalismus betroffen, da das Projekt weniger bekannt ist und zum Bearbeiten eine gültige E-Mailadresse erforderlich
Technische Grundlagen und Motivation

ist. Außerdem stellt sich das Problem des Zusammenführens konkurrierender Änderungen, wenn ein Neuling eine Änderung hochgeladen hat, vor ihrer Sichtung aber ein anderer Mapper das Objekt erneut bearbeitet hat. Zudem bindet ein Sichtungsverfahren Personal und wirkt sich vermutlich negativ auf die Motivation der Mitwirkenden aus, wenn diese ihre Änderung nicht zeitnah auf der Karte auf openstreetmap.org sehen.

Die OSM-API führt abgesehen von der Validierung des hochgeladenen XML-Dokuments und einfachen Prüfungen wie der maximalen Anzahl an Nodes eines Ways und der Gültigkeit von Koordinaten (keine Längengrade > ±180°, keine Breitengrade > ±90°) fast keine Datenprüfungen beim Hochladen von Änderungen durch. Da der Großteil der Informationen über Tags modelliert wird (Multipolygone sind kein eigener Datentyp, sie sind nur Relationen mit speziellen Tags), kann die API solche Prüfungen gar nicht durchführen. Täte sie das, würde sie die weitere Entwicklung von OSM beschränken. Wer weiß, ob wir das, was heute heutzutage als korrekt gilt, in fünf Jahren immer noch als korrekt gilt, geschweige denn Stand der Technik ist? Linien des öffentlichen Verkehrs wurden z. B. 2010 als Routenrelationen modelliert, die alle Ways, welche von den Fahrten dieser Linie benutzt werden, referenzierten. Mittlerweile wird für jeden einzelnen Kurs einer Linie des öffentlichen Verkehrs eine separate Routenrelation angelegt, deren Mitglieder (Haltepositionen und Plattformen ausgenommen) eine durchgehende Strecke vom Start zum Ziel ergeben [33]. Was damals aktuell war, ist heute veraltet.


Fehler können also leicht in den OpenStreetMap-Datenbestand gelangen. Die Prüfung der Daten erfolgt daher durch die Community mithilfe von Werkzeugen, die von Dritten entwickelt und betrieben werden. Dabei handelt es sich meist um Webseiten.

Aktive Mapper lassen sich oft mit WhoDidIt [124] und/oder Latest Changes on OpenStreetMap [103] die Änderungssätze in ihrer Gegend anzeigen und schauen sich die Änderungssätze genauer an, die in irgendeiner Art und Weise verdächtig aussehen, z. B. weil ein großes Gebiet editiert wurde, kein Änderungssatzkommentar eingegeben wurde oder der Mapper noch unerfahren ist. Selbst wenn sie keine Ortskenntnis haben, können sie damit offensichtliche Fehler wie falsche Tags, invalide Geometrien usw. finden.

Fehler erfahrener Mapper und fehlende Daten bleiben damit aber unentdeckt. Nach solchen Fehlern suchen Qualitätssicherungswerkzeuge. Sie sind entweder auf ein Thema spezialisiert oder suchen nach einer Vielzahl an Fehlern. Die entdeckten Fehler sind nur mögliche Fehler. Abbildung 1.6 zeigt ein Beispiel. Es kann sein, dass die beiden
Abbildung 1.6.: Beispiel zweier unverbundener Straßen

Straßen in der Realität wirklich verbunden sind, aber in OSM einen Abstand von nur wenigen Metern haben. Es ist aber auch möglich, dass das kurze Verbindungsstück nur ein Fußweg ist. Eine sichere Entscheidung ist nur mit aktuellen Luftbildern, oft aber auch nur vor Ort möglich. Würden aber stupide/automatisiert diese Straßen verbunden werden, wäre es falsch und würde zu unerwarteten Fehlern bei Routenberechnungen führen.

Erfassungsfehler und ungebräuchliche Tags sind häufig auch ein Indiz für Beiträge, die nicht den Richtlinien der Community entsprechen. Importe von Fremddaten müssen vorher diskutiert und dokumentiert werden, was eine Prüfung der Lizenz einschließt. Geschieht das nicht, fallen sie auf, wenn sie ungebräuchliche oder falsch geschriebene Tags verwenden oder anderweitig Daten anders als gewöhnlich erfassen. Listing 1.5 zeigt einen Node in Kalifornien, welcher durch großgeschriebene Keys (normalerweise bestehen Keys in OSM ausschließlich aus Kleinbuchstaben) auffällt, zudem sind die Koordinaten noch einmal als Tag erfasst, obwohl ein Node bei OSM schon Koordinaten hat. Automatische Reparaturen beseitigen solche Probleme zwar, verwischen die Spuren und potentielle Urheberrechtsverletzungen fallen nicht auf.

Listing 1.5: Beispiel für einen qualitativ schlechten Import

```xml
<node id="306079948" changeset="10328169" timestamp="2012-01-08T05:29:07Z" version="2" visible="true" user="Bryce C Nesbitt" uid="355242" lat="37.8838199" lon="-122.1422283">
  <tag k="ALAND" v="0"/>
  <tag k="AWATER" v="431202"/>
  <tag k="COUNTYFP" v="013"/>
  <tag k="ele" v="139"/>
  <tag k="landuse" v="reservoir"/>
  <tag k="latitude" v="37.8812992"/>
  <tag k="longitude" v="-122.1420097"/>
  <tag k="MTFCC" v="H2040"/>
  <tag k="name" v="Lafayette Reservoir"/>
  <tag k="Source" v="Tiger2009"/>
  <tag k="STATEFP" v="06"/>
  <tag k="Tiger:HYDROID" v="110169760987"/>
  <tag k="Tiger:MTFCC" v="H2040"/>
</node>
```
Die Behebung der Fehler bleibt deshalb stets Menschen überlassen, automatische Fehlerkorrekturen finden nur selten und unter strengen Auflagen statt. Viele Qualitätsicherungswerkzeuge stellen die gefundenen Fehler als Overlay auf einer Basiskarte dar.


\textbf{1.7.1. Keep Right}


Die Visualisierung der gefundenen Fehler erfolgt auf \url{http://keepright.at} mit der JavaScript-Bibliothek OpenLayers 2. Benutzer können dort einen Fehler als gelöst oder False Positive kennzeichnen. Diese Markierungen werden in einer MySQL-Datenbank verwaltet\footnote{Abbildung 1.7 zeigt einen Screenshot der Anwendung.}. Abbildung 1.7 zeigt einen Screenshot der Anwendung.

\textbf{1.7.2. Osmose}


Auch Osmose partitioniert die Welt. Die Partitionierung wird jedoch anhand administrativer Grenzen durchgeführt – meist an Staatsgrenzen, Länder mit einer großen
Abbildung 1.7.: Benutzerobfläche von Keep Right

[Image of a user interface for Keep Right]
Abbildung 1.8.: Benutzeroberfläche von Osmose

Datenmenge werden noch einmal in Bundesländer/Provinzen aufgeteilt. Für manche dieser Gebiete gibt es zusätzliche Prüfroutinen, die etwa an nationale Besonderheiten angepasst sind oder Vergleiche mit Open-Data-Datensätzen durchführen. Statt den Planet herunterzuladen und zu partitionieren, werden von Anbietern, die regionale Planet-Extrakte anbieten, die Extrakte bezogen und in die Datenbank importiert bzw. mittels den Python-Skripten direkt nach Fehlern durchsucht.


1.7.3. OpenStreetMap Inspector

Der OSMI wird von der Geofabrik GmbH entwickelt und betrieben und ist ähnlich modular aufgebaut wie Osmose. Anders als bei Osmose und Keep Right gibt es fast

Derzeit enthält der OSMI folgende Views:

- Geometry
- Tagging
- Places
- Highways
- Areas\footnote{bereitgestellt aus externer Quelle}

\footnote{bereitgestellt aus externer Quelle}

---

Abbildung 1.9.: Ablaufschema OpenStreetMap Inspector

Planetdump von planet.osm.org \quad Diiffs von planet.osm.org

lokaler Planet

\quad C++-Programme für

\quad Multipolygone \quad Taggingfehler \quad ...

\quad Shapefiles für

\quad Multipolygone \quad Taggingfehler \quad ...

Mapserver \quad WMS
1. Technische Grundlagen und Motivation

Abbildung 1.10.: Benutzeroberfläche des OpenStreetMap Inspectors

- Routing
- Addresses
- Water
- Public Transport (mehrere Views)

Abbildung 1.10. zeigt die Benutzeroberfläche des OpenStreetMap Inspectors.

1.7.4. MapRoulette


7 bereitgestellt aus externer Quelle
8 Entwicklung extern, Prozessierung durch Geofabrik
1.7.5. To-Fix

To-Fix ist eine „Mikro-Tasking-Plattform“ [62], die starke Ähnlichkeiten zu MapRoulette aufweist [38] und von der Firma Mapbox entwickelt und betrieben wird. Das Austauschformat für die Fehler einer Challenge ist CSV. Die Hauptnutzer sind Mapbox-Angestellte, die damit für das Routing relevante Fehler wie etwa unverbundene Straßen, aber auch andere Fehler korrigieren. Von den 363 799 Änderungssätzen in OSM, welche die Zeichenkette „#to-fix“ (Groß-/Kleinschreibung wurde ignoriert) enthalten, stammen 97,90 Prozent von 31 Benutzerkonten, die von Mapbox als Mitarbeiter benannt werden.9

1.8. Probleme bestehender Qualitätssicherungswerkzeuge

Keep Right, Osmose und der OSMI haben das Problem gemeinsam, dass sie aufgrund der großen Datenmenge, es nur mit Mühe schaffen, einen akzeptablen Aktualisierungszyklus zu erreichen. Da diese Werkzeuge nur einmal täglich ihre Daten aktualisieren können (das Beziehen von stündlichen Diffs nützt nichts, da die Prozessierung fast einen Tag dauert), kann es im ungünstigsten Fall bis zu zwei Tage dauern, bis eine Änderung eines Benutzers in OpenStreetMap von diesen Werkzeugen als fehlerhaft erkannt wird: An Tag 1 lädt kurz nach Mitternacht der Benutzer seine fehlerhafte Änderung zu OpenStreetMap hoch. Um Mitternacht wurde das tägliche Diff erstellt, das diese Dienste beziehen. Die Änderung des Benutzers wird also erst in dem Diff enthalten sein, das am Ende von Tag 1 erstellt wird. Bis die geänderten Daten dann prozessiert sind, dauert es aber noch fast einen Tag. Der Fehler erscheint also erst am Ende von Tag 2 auf den Websites dieser Dienste.

Viele Datennutzer, die ihren OSM-Datenbestand regelmäßig aktualisieren, beziehen jedoch auch die täglichen Diffs. Bis der Fehler aus ihrem Datenbestand verschwindet, vergeht also noch einmal ein weiterer Tag.


Die räumliche Partitionierung, wie sie bei Osmose und Keep Right verwendet wird, dient allein dazu, den Speicherbedarf beim Import in die SQL-Datenbank zu reduzieren, die zur Qualitätssicherung verwendeten SQL-Abfragen zu beschleunigen und – was aber höchstens bei Osmose erfolgt – die Verarbeitung zu parallelisieren. Dennoch erreichen Keep Right, Osmose und der OpenStreetMap Inspector keine akzeptablen Aktualisierungszyklen, da sie jeden Tag unnötig viele, sich nicht ändernde Daten verarbeiten.

9Changeset-Planet vom 22. August 2016 [6]. Messfehler 0,03 %
Im Rahmen dieser Masterarbeit soll ein Ansatz entworfen und die zentralen Teile davon implementiert werden, sodass nur noch die Teile des Datenbestandes prozessiert werden müssen, die sich geändert haben.

Während bei Osmose und Keep Right die Partitionierung dazu dient, mit schwächerer Hardware einen großen Planetdump zu verarbeiten, soll die Partitionierung, die in dieser Arbeit entworfen wird, unveränderte von veränderten Daten trennen. Nach der initialen Prozessierung aller Daten im Rahmen des Setups sollen nur noch die Gebiete prozessiert werden, die sich geändert haben. Die Partitionierung ist deutlich kleinteiliger als bei Osmose und Keep Right. Es soll das Prinzip eines Tileservers, der nur die Gebiete neu rendert, deren Daten sich geändert haben, auf ein Qualitätssicherungswerkzeug übertragen.

Da die Partitionierung deutlich kleinteiliger ist, sinken die Anforderungen an die Rechenknoten, die die Datenvalidierung durchführen. Es muss nicht mehr die ganze Welt bzw. ein ganzes Bundesland im Speicher gehalten werden, sondern nur noch ein Tile, das wenige Kilometer oder hundert Meter Kantenlänge hat. Die Partitionierung ermöglicht es darüber hinaus, die Verarbeitung zu parallelisieren – sei es in Form mehrerer paralleler Prozesse oder in Form mehrerer Rechenknoten.

Abbildung 1.11.: Zeiträume der Sichtbarkeit von Fehlern beim OpenStreetMap Inspector
2. Architektur

Abbildung 2.1.: Entworfene Architektur für ein Qualitätssicherungswerkzeug mit stündlichen Updates. Die schwarzen Teile wurden nicht im Rahmen dieser Arbeit implementiert bzw. haben keiner Anpassungen bedurft.

Dieses Kapitel gibt einen Überblick über die verwendeten Komponenten. Für Details sei auf die folgenden Kapitel verwiesen, in denen die Komponenten im Detail beschrieben und Gründe für die Designentscheidungen dargelegt werden. Abbildung 2.1 zeigt die entworfene Architektur, die aus mindestens drei Servern besteht.


Die Prozessierungsserver holen sich ein oder mehrere Jobs und berechnen alle Views für dieses Vektortile. Für jeden View entsteht eine Datei in einem Transportformat, z. B. ein Shapefile, welches auf den Renderingserver übertragen und dort in die Renderingsdatenbank (PostgreSQL mit PostGIS) importiert wird.

36
Der *Renderingserver* existiert schon im bisherigen OSMI ohne Vektortiles. Auf ihm bietet derzeit ein MapServer einen WMS (mit aktiviertem GetFeature-Request) an, jedoch sind Shapefiles (eins pro View) derzeit die Datenquelle. In der neuen Version ist eine PostgreSQL-Datenbank (mit PostGIS) die Datenquelle, da laufend Änderungen in die Renderingdatenbank eingefügt werden. Der Renderingserver sucht dazu kontinuierlich auf den Prozessierungsservern nach neuen Shapefiles und importiert diese in seine Renderingdatenbank.

Der Fokus der Arbeit lag auf dem Datenbankserver. Die Komponenten auf dem Prozessungs- und Renderingserver wurden nicht implementiert bzw. angepasst.

### 2.1. Datenbankserver

#### 2.1.1. Komponenten und Funktionsweise


Nach dem Import eines Diffs auf dem Datenbankserver wird auf diesem *Cerepso2vt* gestartet. Es liest die Liste an veralteten Tiles ein (4) und erzeugt die nötigen, neu zu berechnenden Vektortiles (5). Die Vektortiles werden auf der Festplatte abgelegt (6).


#### 2.1.2. Warum werden die OSM-Daten in einer Datenbank gespeichert?

Die Entscheidung, ein Datenbankmanagementsystem (für Geoinformationssystem und im OpenStreetMap-Umfeld ist PostgreSQL gebräuchlich) für die Datenhaltung der OSM-
Daten zu nutzen, fiel relativ früh. Prinzipiell gibt es zwei Möglichkeiten Vektortiles zu erzeugen.


Beim der Verwendung einer Datenbank könnten durch entsprechend formulierter SQL-Abfragen auch thematische Tiles erzeugt werden. Wenn die Laufzeit der Abfragen sich dadurch erheblich verlängert, wird nur Prozessierungsaufwand von den einzelnen Rechenknoten (wovon beliebig viele eingesetzt werden können) zur nicht so einfach skalierbaren zentralen Datenbank verlagert. Das ist nicht im Sinne dieser Arbeit.

Ein unwichtiger Vorteil ist, dass die Datenbank auch noch für viele andere Zwecke als die Erzeugung von Vektortiles verwendet werden kann. Wenn ein passendes Datenbankschema verwendet wird, kann die Datenbank auch als Backend für einen Renderingserver verwendet werden, wenn dieser nur schwach ausgelastet ist.


Wenn ein Diff importiert wird, werden dessen Änderungen auf die einzelnen Vektortiles angewendet. Um zu ermitteln, aus welchem Vektortile ein gelöschtes Objekt entfernt werden soll, muss eine Index existieren, der für jedes OSM-Objekt speichert, in welchen Vektortiles es liegt.

Wird ein Node verschoben, der von einem Way referenziert wird, müssen alle Tiles, in denen dieser Way vorher lag bzw. danach liegen wird, aktualisiert werden. Auch alle Relationen, die Ways referenziern, welche diesen Node referenziern, müssen aktualisiert werden. Die Listen der Referenzen müssen daher effizient invers durchsuchbar sein (einer ID eines Nodes müssen alle Ways, die ihn referenziern, zugeordnet sein). Neben einem Index, der einem Objekt ein oder mehrere Tiles zuordnet, muss also noch ein zweiter Index mit der Zuordnung Objekt → referenzierende Objekte existieren. Wenn beide Indizes im Arbeitsspeicher gehalten werden, können damit schneller Vektortiles erzeugt werden, als es mit einer Datenbank möglich ist.

2.2. Prozessierungsserver


2.3. Renderingserver

Der Renderingserver existiert schon im bisherigen OSMI und muss nur an das Vektortile-Backend angepasst werden. Während beim OpenStreetMap Inspector bislang ein Shapefile pro View die Quelle der WMS-Dienste ist, die von MapServer auf diesem Server bereitgestellt werden, werden künftig mehrere PostgreSQL-Datenbanken die Quelle sein (13). Durch die Verwendung eines Datenbankmanagementsystems sind konkurrierende Zugriffe möglich – eine Voraussetzung für kontinuierliche Updates.

Der Renderingserver sucht kontinuierlich auf den Prozessierungsservern nach neuen Shapefiles, die zu importieren sind, und importiert diese in seine Datenbank (12). Er bietet einen WMS an (13), der im OSMI-Webinterface eingebunden ist.

3. *Cerepso – ein alternativer Importer für PostgreSQL-Datenbanken*

Das im Rahmen der Arbeit erstellte PostgreSQL-Importprogramm *Cerepso* (Langname: *Cerepso replaces osm2pgsql*) ist eine der Komponenten, die auf dem Datenbanksystem zum Einsatz kommen. Es importiert einen OSM-Planet in die Datenbank und wendet die von OSM bereitgestellten Diffs auf die Datenbank an. Als Datenbankmanagementsystem wird PostgreSQL eingesetzt, weil sich dieses im OpenStreetMap-Umfeld als stabil und zuverlässig erwiesen hat und es mit PostGIS eine gute Erweiterung für räumliche Daten gibt.

3.1. Gründe für die Entwicklung von Cerepso

Anfangs war geplant, für den Import der Datenbank und das Anwenden von Diffs auf *osm2pgsql* zurückzugreifen, welches als Importwerkzeug für die Datenbanken von Renderingservern und zur allgemeinen Datenanalyse verwendet wird. *osm2pgsql* hat jedoch einige Nachteile, die dazu geführt haben, mit Cerepso einen Ersatz zu entwickeln.


Tile abdeckt. Das ist mit osm2pgsql ohne Modifikationen nicht möglich. Die Tabelle planet_osm_rels hat keinen räumlichen Index und die Feature-Tabellen, welche räumliche Indizes haben, enthalten nicht alle Relationen.

Für die Slim-Tabelle planet_osm_nodes gibt es seit einiger Zeit eine Alternative. Statt die Nodes in einer PostgreSQL-Tabelle zu verwalten, können sie auch in einer Datei auf der Festplatte zwischengespeichert werden, dem sogenannten Flatnodes-File \[\text{[28]}\]. Da die Slim-Tabellen nicht als öffentliches Interface gedacht sind und schon eine der drei durch eine dateibasierte Lösung ersetzt wurde, kann davon ausgegangen werden, dass die anderen beiden auch eines Tages durch Dateien ersetzt werden. Paul Norman, einer der Maintainer von osm2pgsql, deutet das in einer Diskussion im Bugtracker von osm2pgsql und auf Mailinglisten so an \[\text{[77, 79]}\].


Das Datenbankschema ist für die Nutzung als Datenbankbackend eines Tileserver entworfen und eignet sich ansonsten nur für wenige andere Anwendungen gut (z. B. GIS-Analysen \[\text{[78]}\]). Für die hier beabsichtigte Erzeugung von Vektortiles ist das Datenbankschema nur eingeschränkt geeignet, hätte zu unsaubерem Code geführt und wäre nur eine Behelfslösung, weshalb von einer Erweiterung von osm2pgsql abgesehen und stattdessen ein eigener Importer für OSM-Daten in PostgreSQL entwickelt wurde. Das Datenbankschema, einige Designentscheidungen und die Optimierungen bezüglich des Datenbankzugriffs sind jedoch von osm2pgsql übernommen worden (siehe dazu Kapitel \[\text{3.3}\]).


\[^{1}\text{Der Typ der Geometriespalte ist Geometry.}\]
3.2. Datenbankschema und Indexe


Das Datenbankschema besteht aus vier Tabellen und Cerepso unterscheidet, wie auch schon osm2pgsql, nicht nach den Tags an den Objekten, d. h. es gibt keine getrennten Tabellen für die verschiedenen Objektklassen (z. B. Landnutzung vs. Gebäude).

- nodes für Nodes, die ein oder mehrere Tags haben
- untagged_nodes für Nodes ohne Tags
- ways für Ways
- relations für Relationen

Tabelle 3.1 zeigt die Felder, die in allen vier Tabellen verfügbar sind. Die übrigen Spalten sind nicht in allen Tabellen verfügbar und in Tabelle 3.2 aufgelistet.


Für den effizienten Zugriff werden Indexe auf die Geometriespalte(n) und die Spalte osm_id angelegt. Der Geometrie-Index ist für die Erstellung der Tiles erforderlich, da hierbei der Index die ST_Intersects-Abfrage beschleunigt. Der Index auf die ID-Spalte ist erforderlich, da beim Einfügen eines Ways alle Nodes des Ways abgefragt werden.

Tabelle 3.1.: gemeinsame Felder aller Spalten von Datenbanken, die mit Cerepso importiert wurden

<table>
<thead>
<tr>
<th>Feldname</th>
<th>Typ</th>
<th>Beschreibung</th>
</tr>
</thead>
<tbody>
<tr>
<td>osm_id</td>
<td>bigint</td>
<td>ID des OSM-Objekts</td>
</tr>
<tr>
<td>osm_user</td>
<td>text</td>
<td>Benutzername des letzten Bearbeiters</td>
</tr>
<tr>
<td>osm_uid</td>
<td>text</td>
<td>ID des letzten Bearbeiters</td>
</tr>
<tr>
<td>osm_version</td>
<td>int</td>
<td>Version des Objekts</td>
</tr>
<tr>
<td>osm_lastmodified</td>
<td>char(20)</td>
<td>Zeitstempel der letzten Bearbeitung in OSM</td>
</tr>
<tr>
<td>osm_changeset</td>
<td>bigint</td>
<td>ID des Änderungssatzes der letzten Bearbeitung</td>
</tr>
</tbody>
</table>
3. Cerepso

Tabelle 3.2.: Felder, die nur in einem Teil der Tabellen von mit Cerepso importierten Datenbanken verfügbar sind

<table>
<thead>
<tr>
<th>Feldname</th>
<th>Type</th>
<th>Tabellen</th>
<th>Beschreibung</th>
</tr>
</thead>
<tbody>
<tr>
<td>geom</td>
<td>Point(2)</td>
<td>nodes, untagged_nodes</td>
<td>Geometrie</td>
</tr>
<tr>
<td>geom</td>
<td>LineString(2)</td>
<td>ways</td>
<td>Geometrie</td>
</tr>
<tr>
<td>geom_points</td>
<td>MultiPoint(2)</td>
<td>relations</td>
<td>Geometrie aller Mitglieder, die Nodes sind</td>
</tr>
<tr>
<td>geom_lines</td>
<td>MultiLineString(2)</td>
<td>relations</td>
<td>Geometrie aller Mitglieder, die Ways sind</td>
</tr>
<tr>
<td>way_nodes</td>
<td>bigint[]</td>
<td>ways</td>
<td>IDs der Nodes des Ways</td>
</tr>
<tr>
<td>member_ids</td>
<td>bigint[]</td>
<td>relations</td>
<td>IDs der Mitglieder der Relation</td>
</tr>
<tr>
<td>member_types</td>
<td>char[]</td>
<td>relations</td>
<td>Typen der Mitglieder der Relation (n für Nodes, w für Ways, r für Relationen)</td>
</tr>
</tbody>
</table>

müssen, um die Geometrie erstellen zu können. Diese Abfrage erfolgt anhand der OSM-ID.

3.3. Optimierung des Datenbankzugriffs

Der naive Ansatz zum Einfügen von Daten in eine Datenbank ist die Verwendung des INSERT-Befehls. Um jedoch eine akzeptable Performance beim initialen Import zu haben, wird, wie es auch andere Importwerkzeuge tun und von der PostgreSQL-Dokumentation empfohlen wird [98], der COPY-Befehl verwendet.

Ein kleiner Versuch bestätigt den Performanceunterschied. Ein Import von 39709 Nodes per INSERT dauert abhängig von dem verwendeten Verfahren zur Erzeugung des Well-Known-Binary-Strings für die Geometrie-Spalte 5,3 bis 6,4 Sekunden. Wird COPY verwendet, dauert es nur noch 0,25 Sekunden[2].

Weil zum Import der COPY-Befehl verwendet werden muss, muss die C-Bibliothek libpq zum Zugriff auf die PostgreSQL-Datenbank verwendet werden. In der C++-Bibliothek libpqxx sind die dafür notwendigen Klassen als deprecated gekennzeichnet [100], außerdem ist die Entwicklung auch mehr oder weniger eingeschlafen [29], was gegen eine Verwendung von libpqxx spricht.

Wie auch osm2pgsql verwendet der Cerepso COPY FROM STDIN ..., d.h. nach Senden dieses Befehls muss pro einzufügendem Objekt eine Zeile als CSV an die Datenbank gesendet werden. Als Trennzeichen wird ein Tabulator verwendet.

Zeitweise wurde versucht, gepuffert zu senden (eine Zeile wird mit einem Newline beendet), osm2pgsql verwendet als Puffer einen String und prüft, nachdem eine neue Zeile in den Pufferstring eingefügt wurde, ob der Puffer größer als 1024 Byte ist. Ist der Schwellwert überschritten, wird der Inhalt des Puffers an die Datenbank gesendet und

[2] Intel Core i5-750, 16 GB RAM, Festplatte Seagate ST1000VX000 mit 7200 rpm SATA-600
3. Cerepso


Eine weitere Optimierungsmöglichkeit ist, die Protokollierung der Tabellen auszuschalten (CREATE UNLOGGED TABLE . . .anstelle von CREATE TABLE . . .). Das bringt zwar das Risiko mit sich, dass die Datenbank in einen inkonsistenten Zustand gerät, wenn während des Imports das System abstürzt (z. B. Stromausfall), in diesem Fall muss aber der Import ohnehin erneut durchgeführt werden. Die Verwendung einer Tabelle ohne Protokollierung beschleunigt das Programm um knapp die Hälfte; der Import von Baden-Württemberg dauert mit Protokollierung etwa 7:30 Minuten, ohne nur noch 4:10 Minuten[3]. Die Protokollierung kann ab PostgreSQL 9.5 nach dem initialen Import mittels ALTER TABLE name SET LOGGED eingeschaltet werden.


3.4. Funktionsweise


Die Architektur ist erweiterbar gehalten. Alle Datenbankzugriffe (d. h. Verwendung von Funktionen der Bibliothek libpq) sind in der Klasse PostgresTable zusammengefasst; Teile davon sind sogar in die Klasse, postgres_drivers::Table im der Bibliothek Cerepso-Postgres-Backend ausgelagert, da sie auch von Cerepso2vt (siehe Kapitel 5) genutzt werden. PostgresTable erb that postgres_drivers::Table. Es handelt sich dabei um den Einsatz des Wrapper-Patterns[42].

Die Klasse postgres_drivers::Table besitzt eine Referenz auf eine Instanz der Klasse postgres_drivers::Columns. postgres_drivers::Columns enthält die Spaltendefinitionen der Tabelle (Spaltennamen und Typen) und stellt einen Iterator[42] über die einzelnen Spalten bereit. Die Spaltenfunktionen sind in postgres_drivers::Columns derzeit noch im

---

[3]Intel Core i5-750, 16 GB RAM, Festplatte Seagate ST1000VX000 mit 7200 rpm SATA-600
Konstruktor hart kodiert. Wenn Cerepso in Zukunft flexibler gestaltet sein soll, müssen
die Spaltendefinitionen in eine Konfigurationsdatei ausgelagert und stattdessen aus
dieser Datei eingelesen werden.

Die vom Benutzer über Kommandozeilenparameter gesetzten Einstellungen werden
von der Klasse CerepsoConfig verwaltet, die von den meisten Klassen referenziert wird.
Nach dem Einlesen der Kommandozeilenparameter wird für jede der Tabellen je ein
Objekt der Klasse CerepsoColumns instanziert. Dem Konstruktor wird der Geometrietyp
der Tabelle übergeben. Hart kodierte Regeln im Konstruktor dieser Klasse legen dann
die Spalten der Tabelle fest.

Der Instanziierung von postgres_drivers::Columns folgt die Instanziierung von Postgre-
Table. Der Konstruktor der Klasse PostgresTable baut eine Verbindung zur Datenbank
auf, registriert die Prepared Statements und löscht, falls die Tabelle schon existiert und
ein Neuimport erfolgen soll, die existierende Tabelle.

Durch Scoping wird der Destruktor des Location-Handlers vor dem Destruktor der
PostgresTable-Instanzen aufgerufen. Das sorgt dafür, dass der vom Location-Handler be-
anspruchte Speicher freigegeben wird, bevor im Destruktor der PostgresTable-Instanzen
die Indizes erstellt und ggf. die Tabellen nach ST_GeoHash sortiert werden. Abbildung 3.2
zeigt den Speicherverbrauch Cerepos und aller PostgreSQL-Daemons zusammen. Des-
halb geht nach etwa vier Stunden der Speicherverbrauch stark zurück. Zu diesem
Zeitpunkt wurde der vom Location-Handler reservierte Speicher freigegeben. Die Daten
wurden mit dem Bash-Skript pgimporter/util/get-ram-usage.sh sekündlich ermittelt.

Abbildung 3.2.: Speicherverbrauch von Cerepso und aller PostgreSQL-Daemons beim
Import eines Planetdumps (Details zur Hardware und den
Einstellungen der PostgreSQL-Datenbank siehe Anhang [H])
Der weitere Ablauf unterscheidet sich zwischen den beiden Modi – dem initialen Import und dem Diff-Import. Der Standardmodus ist der initiale Import, der Diff-Import wird mit \texttt{--append} aktiviert.

3.4.1. Initialer Import

Bei einem initialen Import (bei osm2pgsql wird das Planet-Import genannt) werden die nötigen Datenbanktabellen angelegt und die Daten initial importiert. Zuerst initialisiert das Hauptprogramm den von Osmium bereitgestellten NodeLocationsForWays-Handler, der einen zur Laufzeit ausgewählten Index-Typ verwendet. Der NodeLocationsForWays-Handler ermöglicht beim Einlesen den Zugriff auf die Positionen der Nodes anhand ihrer ID, welche von einem Way referenziert werden. Die Auswahl des Indexverfahrens erfolgt zur Laufzeit, der Benutzer legt über den Komandozeilenparameter \texttt{--location-handler=index_type} fest, welches Indexverfahren verwendet werden soll.

In diesem Modus wird die zu importierende OSM-Datei zweimal eingelesen. Beim ersten Einlesen werden nur die Relationen gelesen und vom RelationCollector gespeichert. Dieser Durchlauf wird in der Osmium-Terminologie \texttt{Pass 1} genannt. Im darauf folgenden \texttt{Pass 2} werden Nodes und Ways aus der OSM-Datei gelesen und den Callback-Funktionen des ImportHandler und des RelationCollector übergeben. Deren im Folgenden beschriebenen Callback-Methoden werden von Osmium aufgerufen und fügen per \texttt{COPY} die Objekte in die Datenbank ein.

\texttt{ImportHandler::node (const osmium::Node&)} prüft erst, ob der Node eine gültige Position hat. Aus der Frühzeit von OpenStreetMap gibt es noch Nodes im Planet, die ungültige Koordinaten haben \cite{116}. Anschließend wird der Query-String mit der Methode \texttt{prepare_node_query(const osmium::Node&, std::string&)} der Klasse Postgres-Handler erstellt. Die Geometrie wird als WKB-HEX-String, der ASCII-Repräsentation des Well-Known-Binary-Formats, eingefügt; das erspart den Aufruf von PostGIS-Funktionen wie \texttt{ST_GeometryFromText}. Der WKB-HEX-String wird von der Methode \texttt{postgres_drivers::Table::send_line(std::string&)} an die Datenbank gesendet (die Verbindung ist zu diesem Zeitpunkt im \texttt{COPY}-Modus).

\texttt{ImportHandler::way (const osmium::Way&)} führt ähnliche Prüfungen auch zu Beginn durch. Wenn der Way weniger als zwei Nodes referenziert oder einer der von ihm referenzierten Nodes invalide Koordinaten hat, wird er nicht in die Datenbank geschrieben. Das ist keine optimale Lösung. Würde der Node jedoch ignoriert und der Way dennoch in die Datenbank importiert werden, würde die Datenbank eine falsche Geometrie des Ways enthalten.

Der weitere Ablauf ist ähnlich zu dem in der Methode \texttt{PostgresHandler::prepare_node_query(osmium::Node&, std::string&)}. Weil anders als bei der node-Methode der Code zur Erzeugung des Query-Strings beim Diff-Import nicht auch noch einmal aufgerufen wird, ist er nicht in die Oberklasse PostgresHandler ausgelagert worden. Sollte beim Erzeugen WKB-HEX-Strings eine Exception geworfen werden, wird ein leerer LineString in die Geometrie-Spalte der Tabelle \texttt{ways} geschrieben.
RelationCollector::complete_relation(const osmium::RelationMeta&) stellt wie
das way-Callback des ImportHandler einen String zusammen, der an die Datenbank
gesendet wird. Da die Geometrie einer Relation als GeometryCollection in die Datenbank
geschrieben wird, wird für jedes Mitglied der Relation, das ein Way oder Node ist, ein
GEOS-Objekt mit der GEOSFactory von Osmium erzeugt. Über die GEOS-Bibliothek wird
dann eine GeometryCollection gebildet und als WKB kodiert.

3.4.2. Diff-Import

Beim Diff-Import werden die in einer OSC-Datei enthaltenen Änderungen im OSM-
Datenbestand auf die Datenbank angewendet. Auch der Diff-Import erfolgt in zwei
Durchläufen. In Pass 1 werden alle Objekte, die in dem Diff enthalten sind und de-
en Version größer als 1 ist, gelöscht. In Pass 2 werden alle Objekte, die als neu oder
gändert gekennzeichnet sind, in die Datenbank eingefügt. Das Einfügen erfolgt aus
Performancegründen mittels COPY (vgl. Kapitel 3.3).

Anders als beim initialen Import sind beim Diff-Import während der Anwendung des
Diffs auch noch lesende Zugriffe auf die Datenbank erforderlich. Wenn in OpenStreetMap
die Tags eines Ways geändert wurden, seine Nodes aber nicht, ist nur der Way im Diff
enthalten, nicht aber die von ihm referenzierte Nodes. Wenn der Way in die Datenbank
eingefügt werden soll, müssen die Koordinaten jedes Nodes nachgeschlagen werden,
um die WKB-Geometrie für die Geometrie-Spalte der Tabelle ways erzeugen zu können.
Da die Ways erst eingefügt werden, nachdem alle Nodes eingefügt worden sind, hat die
Datenbankverbindung für die Node-Tabellen den COPY-Modus wieder verlassen und die
Änderungen wurden committet. Deshalb können im zweiten Durchlauf die Nodes der
Ways per SELECT abgefragt werden. Für den Import geänderter Relationen gilt dasselbe.
Die Geometrie-Spalte der Tabelle relations enthält die WKB-Geometrie aller Ways und
Nodes, die Mitglied der Relation sind.

Dieses Vorgehen ist möglich, da die OSC-Dateien nach Typen geordnet sind – erst alle
Änderungen an Nodes, dann die Änderungen der Ways und zuletzt die Relationen.

Relationen, die von dieser Relation referenziert werden, sind nicht enthalten. Da
derzeit kein Bedarf besteht, Relationen, die Relationen referenzieren, zu unterstützen,
wurde auf die Implementierung einer Rekursion verzichtet.

Sämtliche Änderungen, die im Rahmen eines Diff-Imports erfolgen, werden in einem
Transaktionsblock zusammengefasst, d. h. vor dem Initialisieren der Diff-Handler wird
für sämtliche Instanzen der Klasse PostgresTable die Methode PostgresTable::send_begin()
aufgerufen, welches ein BEGIN an die Datenbank sendet. Im Destruktor der Klasse
PostgresTable wird der Transaktionsblock mit COMMIT abgeschlossen. Ein Dirty Read
durch eine andere Applikation, die gleichzeitig auf die Datenbank lesend zugreift (z. B.
Cerepso2vt), wird dadurch vermieden.
3.5. Tile Expiry

3.5.1. Prinzip der Tile Expiry und derzeit vorhandene Software

Um feststellen zu können, welche Tiles nach dem Import eines Diffs neu gerendert werden müssen, werden auf vielen Tileserversn Tile Expiry Logs erstellt. Nach diesem Prinzip arbeitet auch der in dieser Arbeit entwickelte Datenbankserver. Wenn ein Node, ein Way oder eine Relation geändert wird, werden die Tiles als veraltet markiert, auf denen sich das Objekt in der alten Version befand und in der neuen Version befinde, dann diese Tiles müssen neu prozessiert werden. Diese Logs sind Textdateien, die pro neu zu berechnendem Tile eine Zeile in der Form \(Z/X/Y\) enthalten, z. B. 12/564/125 für das Tile \(x = 564, y = 125, z = 12\).


3.5.2. Implementierung in Cerepso und Herausforderungen


- ExpireTilesQuadtree ist die neue Implementierung, die im Folgenden genauer erläutert wird. (--expire-tiles=quadtree)
Mit --min-zoom=ZOOMSTufe und --max-zoom=ZOOMSTufe kann der Nutzer angeben, die Tiles welcher Zoomstufen in den Tile Expiry Logs enthalten sein sollen. Standardwert ist 9 als minimale und 15 als maximale Zoomstufe.

Beim Löschen aus und beim Einfügen eines Objekts in die Datenbank wird jeweils die für den Objekttyp passende Methode der Klasse ExpireTiles aufgerufen. Bei Nodes ist es die Methode ExpireTiles::expire_from_point(osmium::Location&), bei Ways ist es die Methode ExpireTiles::expire_from_coord_sequence(osmium::geom::CoordinateSequence*).

Bei Relationen werden von all deren Mitgliedern ebenfalls diese beiden Methoden aufgerufen. Um Cerepso einfach zu halten, werden Relationen, welche andere Relationen referenzieren, nicht unterstützt. Daher werden keine Tiles als veraltet markiert, wenn eine Relation, die nur andere Relationen referenziert, geändert wird. Die jeweilige Implementierung der Klasse ExpireTiles transformiert die geographischen Koordinaten in Web Mercator und wandelt sie in Tile-IDs um.
Im Folgenden wird die neue Quadtree-basierte Implementierung (die Klasse `ExpireTilesQuadtree`) genauer beschrieben. Kern der Quadtree-Implementierung ist ein Set (std::set), welches für jedes Tile, das als veraltet markiert werden soll, einen Eintrag enthält. Die Einträge sind vom Typ Integer, es handelt sich dabei um die Quadtree-Repräsentation der Tile-Adressen, die bei Bing-Maps verwendet wird (siehe Seite 21). Das Set enthält nur die Adressen in der höchsten Zoomstufe, für die die Tile Expiry berechnet werden soll. Die Tile-IDs der niedrigeren Zoomstufen werden erst beim Schreiben der Tile-Expiry-Datei berechnet. Um eine Quadtree-Adresse in der Zoomstufe $n$ in eine Quadtree-Adresse der Zoomstufe $n-d$ zu transformieren, ist ein Bitshift um $2 \cdot d$ nach rechts erforderlich. (vgl. Abschnitt 1.4 und insbesondere Abbildung 1.4)

Da ein Set jedes Element höchstens einmal enthält, wird kein Tile doppelt ausgegeben, auch wenn sich mehrere Geometrien in diesem Tile geändert haben und für jede geänderte Geometrie einer der Methoden der Klasse `ExpireTiles` aufgerufen wurde.

Das Tile, in dem ein veränderter Node liegt, als veraltet zu markieren, ist relativ einfach. Die Koordinaten des Nodes werden in Tile-IDs konvertiert und in das Set eingefügt, falls das Tile nicht schon wegen einer anderen Änderung als veraltet markiert wurde.

Bei Ways ist es nicht so einfach. Es genügt nicht, nur die Tiles, in denen ein Way Nodes hat, als veraltet zu markieren. Stattdessen müssen alle Tiles, in denen der Way liegt, als veraltet markiert werden. Abbildung 3.4 zeigt auf der linken Seite, wie ein Node eines Ways verschoben wurde. Wenn nur die Tiles neu gerendert werden, in denen die Nodes liegen, weil nur diese Tiles als veraltet markiert wurden, führt das zu unerwünschten Ergebnissen. Bei gerenderten Karten hängen z. B. Straßen in der Luft (rechte Seite). Durch das Verschieben des Nodes könnten auch Multipolygone invalide werden (z. B. könnten sich jetzt zwei Ringe kreuzen). Wenn die Tiles nicht als veraltet markiert werden, werden sie nicht prozessiert und der Fehler wird nicht gefunden.

Erst ab einem gewissen, hart kodierten, Schwellwert (derzeit 20 km) werden nur noch die Tiles als veraltet markiert, in denen der LineString Stützpunkte hat. Damit soll vermieden werden, dass bei sehr langen Segmenten zu viele Tiles als veraltet markiert werden. Die meisten langen Linien haben viel häufiger einen Stützpunkt – Hochspannungsleitungen z. B. alle paar hundert Meter einen Masten. Etwa ein bis fünf Mal im Jahr erzeugen Mapper versehentlich Polygone, die über einen ganzen Kontinent

Abbildung 3.4.: Wenn ein Node eines Ways verschoben wird, aber nur die Tiles als veraltet (grauer Hintergrund) markiert werden, in denen die Nodes liegen (links), führt das zu unerwünschten Ergebnissen, die gerade beim Rendering von Karten besonders auffällig sind (rechts).

4. Partitionierung

4.1. Das Partitionierungsverfahren

In der Online-Kartographie hat sich in den vergangenen zehn Jahren das Prinzip der Map Tiles (siehe Kapitel 1.4) durchgesetzt. Dieses Partitionierungsverfahren ist nicht datengetrieben, sondern teilt die Karte in gleich große quadratische Kacheln (Tiles) ein. Es ist für Entwickler recht einfach zu implementieren und bringt dennoch spürbare Performance-Verbesserungen, da sich die Daten bzw. die daraus erstellten Produkte (z. B. Kartentiles) leicht cachen lassen.

Die alleinige Verwendung von Tiles als Partitionierungsschema ist für den Zweck der Datenprozessierung nicht geeignet. Wenn man auf einer zu niedrigen Zoomstufe partitioniert, sind in Gebieten mit hoher Datendichte die Tiles zu groß und ihre Prozessierung dauert zu lange. In Gebieten mit niedriger Datendichte (dünn besiedelte und unbesiedelte Gebiete) sind die Tiles zu klein. Die Änderung eines Ways in solchen Gebieten in OpenStreetMap sorgt dafür, dass unzählige Tiles neu prozessiert werden müssen, da die Ways in diesen Gebieten relativ lang sind.


Daher fiel die Entscheidung, ein hybrides Verfahren zu verwenden, das zwar ein Tile-basiertes Verfahren ist, aber auch datengetriebene Elemente hat. Gegenüber der meisten eingesetzten Software verhält es sich wie ein reines Tile-basiertes Verfahren.

Grundsätzlich erfolgt die Partitionierung auf einer Zoomstufe $z_0$. Diese ist so gewählt, dass in Gebieten mit geringer Datendichte und einer großen projektionsbedingten Verzerrung keine zu kleinen Tiles entstehen. Gerade Kanada, Alaska und weite Teile Russlands haben sehr wenig Daten pro Quadratkilometer; ein Tile auf Zoomstufe $z_0$ deckt dort aber ein kleineres Gebiet als weitere südlich ab. Die Größe (Datenmenge) eines Tiles wird nur näherungsweise anhand der Anzahl der Nodes in diesem Tile bestimmt, anstatt
alle Objekte zu zählen. Hat ein Tile mehr als \( s \) Nodes, wird es in seine vier Untertiles aufgeteilt. Haben die vier Untertiles immer noch mehr als \( n \) Nodes, werden sie erneut aufgeteilt. Das Aufteilen endet an einer festgelegten Zoomstufe \( z_n \), die so gewählt ist, dass die entstehenden Tiles keine zu kleine Fläche abdecken. An dieser Stelle hat das Partitionierungsschema die Charakteristik eines linearen Quadtree \([25]\). Abbildung 4.1 zeigt die resultierende Partitionierung im Bereich der Elbe- und Wesermündung, sie wurde mit expiries2shp (siehe Anhang F) erstellt. Für die Details zur Wahl von \( z_0 \), \( z_n \) und \( s \) sei auf Kapitel 4.3 verwiesen.

Abbildung 4.1.: Resultierende Partitionierung im Bereich der Elbe- und Wesermündung mit \( z_0 = 9 \), \( z_n = 16 \), \( s = 5000 \); Datenbasis ist der Planetdump vom 29. August 2016

Diese Aufteilung ist statisch und passt sich nicht dynamisch an die Änderungen im Datenbestand an. Dadurch kann das Verfahren nach außen als (statisches) Tile-basiertes Verfahren auftreten, sich intern aber dennoch an die ungleiche räumliche Verteilung der Daten anpassen, was eine der Forderungen an räumliche Indizes ist \([25]\). In größeren zeitlichen Abschnitten muss mit dense_tiles von Frederik Ramm \([104]\) für alle Zoomstufen von \( z_0 \) bis \( z_n \) berechnet werden, wie viele Nodes die einzelnen Tiles enthalten:
1. Für die Zoomstufe $z_0$:

   $ ./dense_tiles -sac -z ZOOM_LEVEL PLANETFILE | sort > tile-sizes-z9.txt

   Die Ausgabe von dense_tiles erfolgt auf der Standardausgabe und dient als Eingabe für sort, das die Ausgabe alphabetisch sortiert. Die endgültige Ausgabe wird in eine Datei umgeleitet.


2. Für jede weitere Zoomstufe von $z_1$ bis $z_n$:

   $ ./dense_tiles -sac -M MAX_NODES -z ZOOM_LEVEL PLANETFILE | sort
   ↪ > tile-sizes-zZOOM_LEVEL.txt

   MAX_NODES ist die maximale Kapazität $s$ eines Tiles. Für die Zoomstufen $> z_0$ müssen nur die Tiles ermittelt werden, die mehr als $s$ Nodes haben, da nur sie weiter aufgeteilt werden müssen. Für jede Zoomstufe werden etwa 10 bis 15 Minuten benötigt, wovon etwa die Hälfte der Zeit allein für das Einlesen des Planetdumps benötigt wird und sich durch einen schnelleren Prozessor ein wenig, aber eher durch einen Prozessor mit mehreren Kernen um zwei bis vier Minuten senken ließe. Listing 4.2 zeigt die alphabetisch sortierte Ausgabe für Zoomstufe 10.

3. Das im Rahmen dieser Arbeit geschriebenen Python-Skript split-initial-tiles.py ausführen:

   $ python3 split-initial-tiles.py tile-sizes-z9.txt MAX_NODES | sort > z9.txt

   Es liest die in Schritt 1 erstellte Datei ein, welche für jedes Tile auf Zoomstufe $z$ angibt, wie viele Nodes es enthält. Enthält das Tile weniger als $s$ Nodes, wird es unverändert (jedoch ohne die Anzahl der Nodes) auf der Standardausgabe ausgegeben. Enthält es mehr als $s$ Nodes, werden seine vier Untertiles auf der Zoomstufe $z_1$ ausgegeben. Die Ausgabe muss mit sort alphabetisch sortiert werden. Sie hat dasselbe Format wie Tile-Expiry-Listen.

---

1 Intel Core i5-750, 16 GB RAM, Festplatte Seagate ST1000VX000 mit 7200 rpm SATA-600
4. Für jede weitere Zoomstufe von $z_1$ bis $z_n$ das im Rahmen dieser Arbeit geschriebene Python-Skript `next-level.py` ausführen:

```bash
$ python3 next-level.py zOOM_THIS.txt tile-sizes-zOOM_NEXT.txt ZOOM_NEXT
```

$ `MAXNODES | sort > zOOM_NEXT.txt`

`ZOOM_THIS` ist dabei $z_i$, `ZOOM_NEXT` ist dabei $z_{i+1}$. Auch die Ausgabe dieser Schritte wird jedes Mal alphabetisch sortiert.

Für jedes in der Datei `ZOOM_THIS.txt` aufgeführte Tile wird unverändert ausgegeben, wenn es ein Tile der Zoomstufe $z_i$ oder kleiner ist, da dann schon beim vorherigen Durchlauf geprüft wurde, dass das Tile weniger als $s$ Nodes enthält. Für die in der Datei `ZOOM_THIS.txt` aufgeführten Tiles der Zoomstufe $z_{i+1}$ wird in der Datei `tile-sizes-zOOM_NEXT.txt` geprüft, ob das Tile zu groß ist. Wenn es zu groß ist, werden seine vier Untertiles auf Zoomstufe $z_{i+2}$ ausgegeben. Andernfalls wird es unverändert ausgegeben.

Ein Durchlauf von `split-initial-tiles.py` bzw. `next-level.py` dauert je nach Zoomstufe 5 bis 15 Sekunden, der Speicherverbrauch ist sehr gering\(^2\). Da die Ausgabe jeweils alphabetisch sortiert wird, können die Dateien nach dem Prinzip des Streamings zeilenweise eingesehen werden und keine Daten müssen gepuffert werden.


### Listing 4.3: Inhalt von Beispieldatei 1

(Eingabe für `comm` in Listing 4.5)

```bash
$ cat datei1.txt
line only in 1
line in both
line in both, too
last line in 1
```

### Listing 4.4: Inhalt von Beispieldatei 2

(Eingabe für `comm` in Listing 4.5)

```bash
$ cat datei2.txt
line only in 2
line in both
line in both, too
last line in 2
```

### Listing 4.5: Beispielhafte Ausgabe von `comm`

```bash
$ comm datei1.txt datei2.txt
line only in 1
  line only in 2
line in both
  line in both, too
last line in 1
  last line in 2
```

\(^2\)Intel Core i5-750, 16 GB RAM, Festplatte Seagate ST1000VX000 mit 7200 rpm SATA-600
Da in den Expiry-Listen ein Tile auf allen Zoomstufen von \( z_0 \) bis \( z_n \) enthalten ist, wird mit comm das Tile herausgefiltert, das von Cerepso2vt erzeugt werden soll:

\[
\text{comm EXPIRY\_LOGFILE TILE\_LIST}
\]

EXPIRY\_LOGFILE ist die Expiry-Liste, die Cerepso ausgegeben hat. TILE\_LIST ist das Ergebnis der oben erwähnten Schritte 1 bis 4.

### 4.2. Wechselwirkungen zwischen Inhalt und Häufigkeit


Abbildung 4.2 zeigt, wie oft Tiles der Zoomstufe 12 in diesem 38 Tage langen Zeitraum als veraltet markiert worden wären. Es fällt auf, dass vor allem die Tiles in großen Städten als veraltet markiert worden wären. Vor allem das Tile 12/2200/1343, welches Berlin-
Abbildung 4.2.: Anzahl der Markierungen von Tiles auf Zoomstufe 12 als veraltet im Nordosten Deutschlands beim Import der stündlichen Diffs, Zeitraum 29.8.–5.10.2016
Abbildung 4.3.: Anzahl der Markierungen von Tiles auf Zoomstufe 12 als veraltet beim Import der stündlichen Diffs im Nordosten Frankreichs und Südwestdeutschland, Zeitraum 29.8.–5.10.2016

Mitte, Kreuzberg und Friedrichshain enthält, sticht heraus. Es wurde 382 Mal (d. h. durchschnittlich alle 2 bis 3 Stunden) markiert und gehört zu Top 5 Tiles weltweit, was die Anzahl der Markierungen als veraltet betrifft. Insgesamt wurden in diesem Zeitraum 3,464 Millionen Tiles auf Zoomstufe 12 als veraltet markiert. Wenn ein Tile mehrfach als veraltet markiert wurde, wird es mehrfach gezählt, weil jede Markierung eine Neuprozessierung auslöst. Wenn in dem Tile jedoch mehrere Objekte innerhalb einer Stunde editiert wurden, erfolgt nur eine Markierung.


Listing 4.6: SQL-Abrfrage zum Zusammenfassen von Polygonen, die mit expiries2shp aus den Tile-Expiry-Listen erzeugt wurden

```sql
SELECT count(geom), x, y, zoom, geom FROM expiry WHERE zoom = 12 GROUP BY x, y,
```

Betrachtungen der Daten außerhalb Deutschlands bestätigen diesen Eindruck. Die meisten Tiles, durch die eine Grenze eines souveränen Staats verläuft, haben sich geändert. Das fällt dort auf, wo die Staatsgrenze als seeseitiger Rand der 12-Meilen-Zone entlang der Küste verläuft. Im Meer gibt es meist wenig Daten, Änderungen sind dort selten.

Änderungen an Relationen sind daher vermutlich für den größten Teil der Markierungen als veraltet verantwortlich. Es gibt mehrere Möglichkeiten, dieses Problem zu lösen und dadurch die Anzahl der veralteten Tiles zu reduzieren:

- Durch Ignorieren der Relationen reduziert sich die Anzahl der Markierungen erheblich. Dieser Ansatz hat jedoch Nachteile. Wenn die Tags, die Rollen der Mitglieder oder die Mitgliederliste an sich geändert werden, ohne dass eine geometrische Änderung beteiligt ist, werden die Tiles der Relation nicht als veraltet markiert und die durch die Änderung eingeführten Fehler werden nicht entdeckt. Das Ergänzen eines Ways zu einer Multipolygon-Relation kann diese jedoch invalidieren, wenn sich dadurch z. B. zwei Ringe kreuzen oder nicht-tangential berühren. Zu den möglichen Fehlern einer Multipolygon-Relation sei auf Abschnitt 4 verwiesen.


- Der ideale Weg wäre, durch Vergleichen beider Relationen (sowohl die Tags als auch die Mitgliederliste und die resultierende Geometrie) die Tiles zu finden, die sich wirklich geändert haben. Es handelt sich jedoch um eine sehr geringe Zahl an zusätzlichen Fehlern, die damit zusätzlich entdeckt werden. Wenn eine Relation durch eine Bearbeitung eines Mappers invalide wird, ist es meist auch mit einer Änderung eines der Mitglieder verbunden. Eine perfekte Lösung brächte daher einen unnötig hohen Entwicklungsaufwand mit sich. Sie würde die Tile-Expiry-Implementierung komplexer machen und ist für die meisten Anwender nicht von Interesse. Deren Bedürfnisse sind mit einer 80-20-Lösung hinreichend befriedigt. Performance (schneller und speichersparsamer Diff-Import) ist ihnen wichtiger.
Abbildung 4.4.: Anzahl der Markierungen von Tiles auf Zoomstufe 12 als veraltet beim Import der stündlichen Diffs, Zeitraum 29.8.–5.10.2016, wenn keinerlei Relationen berücksichtigt werden

- Staatsgrenzen ändern sich seltener als Routenrelationen. Es liegt also nahe, Relationen prinzipiell zu berücksichtigen, aber Routenrelationen zu ignorieren.

Abbildung 4.5.: Anzahl der Markierungen von Tiles auf Zoomstufe 12 als veraltet beim Import der stündlichen Diffs im mittleren Kalifornien, Zeitraum 29.8.–5.10.2016; links: mit Relationen, rechts: ohne Relationen

Abbildung 4.6.: Anzahl der Markierungen von Tiles auf Zoomstufe 12 als veraltet beim Import der stündlichen Diffs im Norden Südamerikas und Süden Mittelamerikas, Zeitraum 29.8.–5.10.2016; Legende siehe Abbildung 4.5


**4.3. Größe der Tiles**

Die Größe der Tiles muss so klein gewählt werden, dass ein Tile innerhalb einer akzeptablen Zeit verarbeitet werden kann. Andererseits muss das Tile so groß sein, dass es eine ausreichend große Umgebung enthält, denn bei einigen Abfragen ist die Umgebung des Objekts relevant. Wenn unverbundene, aber nahe bei einander liegende Straßenenden gesucht werden sollen, sollte zwischen zwei unverbundenen Enden möglichst keine Kante eines Tiles liegen. Daher müssen die Tiles sich ein wenig überlappen.

Abbildung 4.7 zeigt die Tiles, die zwischen dem 4. September 2016 05:00:01 UTC und dem 5. September 2016 05:00:00 UTC in Süddeutschland als veraltet markiert wurden. Wird eine zu niedrige Zoomstufe für die Partitionierung gewählt, werden in Gebieten mit hoher Datendichte, wie beispielsweise Mitteleuropa, unnötig viele Daten verarbeitet. Die Gesamtfläche aller zu prozessierenden Tiles ist bei einer Partitionierung auf Zoomstufe 14 um 75,4 Prozent geringer als bei einer Partitionierung auf Zoomstufe 12. Diese Ersparnis lässt sich dadurch erklären, dass die als veraltet markierten Rechtecke ein an die Geometrie angepasstes Raster darstellen. Je feiner aufgelöst das Raster ist (d. h. je höher die Zoomstufe), desto weniger Tiles werden als veraltet markiert.

Dennoch ist es keine gute Idee, einfach auf einer sehr hohen Zoomstufe zu partiotionieren. In unbesiedelten Gebieten befindet sich oft nur ein Way (z. B. ein Fluss) in einem Tile der Zoomstufe 12. Wenn die Partitionierung auf Zoomstufe 12 oder gar einer
4. Partitionierung

Abb. 4.7: Zeitliche Entwicklung der Tile-Expiry (Relationen nicht berücksichtigt) zwischen dem 5.9.2016 05:00:01 und 6.9.2016 05:00:00 UTC.

(a) Zoomstufe 12, 05:00–11:00
(b) Zoomstufe 12, 05:00–17:00
(c) Zoomstufe 12, 05:00–23:00
(d) Zoomstufe 12, 05:00–05:00
(e) Zoomstufe 14, 05:00–11:00
(f) Zoomstufe 14, 05:00–17:00
(g) Zoomstufe 14, 05:00–23:00
(h) Zoomstufe 14, 05:00–05:00

Aufgrund der auf den vorhergehenden Seiten dokumentierten Erkenntnissen fiel die Entscheidung, die Zoomstufe 9 als Baiss zu verwenden. Wählt man eine zu hohe Zoomstufe, sind die kleinsten Tiles zu klein. Als Schwellenwert wurde 5000 gewählt. In typischen mittel europäischen Städten haben bei geschlossener Bebauung, wenn alle Gebäude gemappt sind, Tiles auf Zoomstufe 15 etwa 10 000 Nodes. Solche Tiles als Vektortile zu erzeugen, kann wegen der vielen Daten zwei bis fünf Sekunden dauern. Tiles mit weniger Nodes beschleunigen daher nicht nur die Prozessierung, sondern auch die Erstellung der Vektortiles. Wenn ein Tile mehr als 5000 Nodes hat, wird es, wie in Kapitel 4.1 beschrieben, so oft aufgeteilt, bis die Anzahl der Nodes unter dem Schwellenwert ist. Für den Schwellenwert wurde 5000 gewählt.

### 4.4. Erzeugung eines Vektortiles

Ein Vektortile muss alle Objekte enthalten, die sich im Gebiet des Tiles befinden. Bei der Erzeugung wird dies durch eine ST_Intersects-Bedingung sichergestellt. Es wären also die in Listing 4.7 aufgelisteten SQL-Abfragen erforderlich. min_lon, min_lat usw. sind hierbei Platzhalter für die zwei Koordinatenpaare der Bounding-Box.

Listing 4.7: SQL-Abfragen, die prinziell zur Erstellung eines Vektortiles notwendig wären

```sql
SELECT tags, osm_id, osm_uid, osm_version, osm_lastmodified, osm_changeset, ST_X(geom), ST_Y(geom) FROM nodes WHERE ST_Intersects(geom, ST_MakeEnvelope(min_lon, min_lat, max_lon, max_lat, 4326));
SELECT tags, osm_id, osm_uid, osm_version, osm_lastmodified, osm_changeset, member_ids, member_types, member_roles FROM relations WHERE ST_Intersects(geom_points, ST_MakeEnvelope(min_lon, min_lat, max_lat, max_lat, 4326)) OR ST_Intersects(geom_lines, ST_MakeEnvelope(min_lon, min_lat, max_lon, max_lat, 4326));
```

65
4.5. Herausforderungen bei der Partitionierung in kleine Teile

Die in diesem Abschnitt aufgezeigten Probleme und Lösungen sind unabhängig vom Partitionierungsverfahren, solange die einzelnen kleinen Teile, die aus der Partitionierung hervorgehen, klein sind (wenige hundert Meter oder wenige Kilometer Länge und Breite). Sie treten auch auf, wenn statt Tiles bzw. einem Quadtree-Schema in Dreiecksmaschen oder Sechsecke partitioniert werden würde, denn jede Partitionierung führt zu mehr oder minder kleinen Teilen, die Kanten haben.

Die SQL-Abfragen aus Listing 4.7 lösen das Problem nicht. Stattdessen tun sich neue Herausforderungen auf:

1. Eine räumliche Abfrage benötigt einen räumlichen Index. Andernfalls führt die Datenbank einen *sequential Scan* durch (auch als *Full Table Scan* bekannt) und liest die gesamten Inhalte der Datenbank ein. Mit allen Metadaten (ausgenommen Benutzernamen) und Indexen belegt die Datenbank der OSM-Daten derzeit etwa 800 GB. Für sequentielles Einlesen ist kein Datenbankmanagementsystem erforderlich, diesen Overhead (Import, Indizes, Platzverbrauch) könnte dann eingespart werden.

2. Ein räumlicher Index auf die Geometriespalte der Tabelle *untagged_nodes* ist nicht zu empfehlen. Seine Erstellung benötigt mehrere Stunden und zahlreiche Gigabyte Platz. Dass das Ziel eigentlich auch ohne einen räumlichen Index auf diese Tabelle erreicht werden kann, wird im weiteren Verlauf dieser Arbeit gezeigt.


4. Partitionierung

Abbildung 4.8.: Beispiel für die Bildung einer falschen Geometrie eines Ways, dessen Nodes nicht vollständig vorhanden sind. Node 3 liegt außerhalb des Tiles. Wird er verworfen, ist die resultierende Geometrie (durchgezogene Linie) falsch.

4.6. Kompromisse und Lösungen

4.6.1. Unvollständige Referenzen

Um das in Abschnitt 4.5 Punkt 3 beschriebene Problem der unerfüllten Referenzen zu lösen, müssen Listen (oder vergleichbare Datenstrukturen) gepflegt werden, mit denen verwaltet wird, welche Objekte in der Menge der Ergebnisse enthalten sind. Bei jedem Objekt, das Referenzen auf andere Objekte haben kann (das sind Ways und Relationen), muss geprüft werden, ob diese Objekte schon in der Menge der Ergebnisse enthalten sind oder ob sie noch abgefragt werden müssen. Die fehlenden Objekte werden anschließend anhand ihrer ID abgefragt. Deshalb muss es für jede Datenbanktabelle einen Index auf die Spalte osm_id geben.

4.6.2. Nodes ohne Tags

Für das Problem der verwaisten Nodes (Abschnitt 4.5 Punkt 1 und 2) existiert zwar keine perfekte Lösung, aber eine praktikable. Wenn keine räumliche Abfrage auf die Tabelle untagged_nodes ausgeführt werden, werden die Nodes, die von Ways referenziert werden, aber keine Tags haben, als „fehlend“ erkannt, weil sie fast alle von Ways referenziert werden. Bei der räumlichen Abfrage der Tabelle ways fällt das auf, sodass sie später einzeln anhand ihrer ID abgefragt werden. Das resultierende Vektortile ist genauso gut mit Drittanwendungen verarbeitbar.

Art und Weise (ein Programm, das das gesamte Planetfile oder Extrakte davon einliest), nach verwaisten Nodes gesucht wird, genügt das.

Verwaiste Nodes können ein Indiz für einen Änderungssatz sein, der fragwürdig ist und genauer untersucht werden sollte. Das können Importe sein, die gegen die einschlägigen Community-Richtlinien verstoßen [19, 55], oder der Beitrag eines unerfahrenen Benutzers. Solche Änderungssätze fallen, der Erfahrung des Autors zufolge, meist auch anderweitig auf. Seien es Fehler anderer Art, wie invalide Multipolygone, Routingfehler usw., oder durch die Beobachtung der Editieraktivität durch andere Mapper.

### 4.6.3. Performanceeinbußen durch vollständige Relationen


Viele, wenn auch nicht alle Analysen an Relationen können auch durchgeführt werden, ohne dass alle Referenzen befriedigt sind. Das wird in Kapitel 4.7 gezeigt.

### 4.7. Validierung unvollständiger Relationen

#### 4.7.1. Multipolygone

Prinzipiell lassen sich auch mit insgesamt unvollständigen, aber lokal (im Gebiet des Tiles) vollständigen Relationen vom Typ Multipolygon fast alle nötigen Qualitätssicherungsanalysen durchführen:

- Wenn sich zwei Ways kreuzen und der Schnittpunkt im Gebiet des Tiles liegt, sind beide Ways im Vektortile vollständig enthalten, da die ST_Intersects-Funktion true zurückgibt (vgl. Abbildung 4.9).

- Wenn an einer Position zwei Nodes sind (Koordinaten identisch, aber ID verschieden) und die Position im Gebiet des Tiles liegt, sind beide Nodes im Vektortile enthalten.

- Wenn ein äußerer und ein innerer Ring sich berühren und die Berührung nicht tangential ist, ist es ein Fehler. Wenn der Abschnitt, auf dem der sowohl zum inneren als auch zum äußeren Ring gehört, ganz oder teilweise im Gebiet des Tiles liegt, sind alle beteiligen Ways vollständig im Tile enthalten (vgl. Abbildung 4.10).


4.7.2. Administrative Grenzen

Da administrative Grenzen denselben Regeln wie Multipolygone folgen, lassen sich die meisten Analysen auch für sie durchführen. Eine Prüfung ist jedoch nicht durchführbar: Ein administratives Polygon A mit Admin-Level n muss alle Gebiete mit Admin-Level
> n, die mit A eine gemeinsame Fläche haben, vollständig umgeben.\(^3\) Das heißt, ein Landkreis muss vollständig innerhalb eines Bundeslandes liegen. Diese Bedingung kann mit Vektortiles, in denen die Relationen unvollständig enthalten sind, nicht überprüft werden. Selbst wenn die Mitglieder der Grenzrelation Rollen (\textit{outer} oder \textit{inner}) hätten, kann nicht ermittelt werden, ob eine Seite eines Rings innen oder außen ist, solange der Ring nicht vollständig im Multipolygon enthalten ist.


Die Bedingung der vollständigen Bedeckung ist immer dann verletzt, wenn sich zwei Ringe (eigentlich zwei Ringstücke, da die Relationen nicht vollständig im Tile enthalten sind) A und B kreuzen und die Schnittmenge der beiden Ringstücke vom Typ Point oder MultiPoint ist. Dies ist mit einem Vektortile-basiertem Ansatz detektierbar. Für die Begründung sei auf den vorhergehenden Abschnitt zu Multipolygonen verwiesen.

Weitaus häufiger ist jedoch der Fall, dass die Schnittmenge zweier Ringstücke verschiedener administrativer Grenzrelationen vom Typ Linestring ist. In den allermeisten Fällen ist es kein Fehler, sondern die valide Realität. Das tritt bei administrativen Polygonen auf, die am Rand des Polygons eines höheren Admin- Levels liegen (z.B. Rheinland-Pfalz in Deutschland). Hier muss der Verlauf der beiden Ringe zueinander vor und nach dem gemeinsamen Abschnitt (der Schnittmenge vom Type LineString) untersucht werden. Abbildung 4.12 zeigt links den Fall, dass der gemeinsame Abschnitt 2–3 vollständig in einem einzigen Vektortile liegt. Wenn der zu A gehörige Abschnitt 1–2 links des zu B gehörigen Abschnitt 4–2 liegt, aber der zu A gehörige Abschnitt 3–6 \textit{rechts} des zu B gehörenden Abschnitts 3–5 liegt, ist die Bedingung der vollständigen Überdeckung

\(^3\)Es sei darauf hingewiesen, dass das keine offizielle und dokumentierte Anforderung ist (vgl. Seite 15).

\(^4\)In einem Vorverarbeitungsschritt müssen bei der Prozessierung des Tiles alle Ringstücke aller administrativen Grenzen gerichtet werden.
Abbildung 4.12: Unmöglichkeit der Suche von Verstößen gegen die Hierarchie-Regel zweier administrativer Grenzen, wenn die Schnittmenge zweier ihrer Ringe ein LineString ist

verletzt. Multipolygon- und Grenzrelationen haben in OpenStreetMap keine vorgeschriebene Drehrichtung. Das Innere von A kann also entweder die Sektoren I und II oder die Sektoren III und IV sein. Auf welcher der beiden Seiten das Innere von A liegt, ist egal. In beiden Fällen umgibt das Innere von A nicht das Innere von B. Dieses ist nämlich entweder I+IV oder II+III.


4.7.3. Routenrelationen

4. Partitionierung

Routenrelationen, die dem PTv2-Schema (vgl. Kapitel 1.2.5 und Anhang A) folgen, lassen sich nur sehr eingeschränkt validieren. Nach Lücken kann gesucht werden, denn für die Suche nach Lücken gilt dasselbe wie für Multipolygone.


Neben der Lückendetektion bleibt nur eine einfache Prüfung der Mitgliederliste möglich, für die die referenzierten Objekte nicht im Tile enthalten sein müssen. Es kann geprüft werden, ob alle Haltestellen am Anfang der Mitgliederliste aufgeführt sind und ob danach die Ways folgen.


4.7.4. Abbiegebeschränkungen

5. Cerepso2vt

Cerepso2vt ist die Komponente, welche aus der PostGIS-Datenbank mit den gesamten Daten eines OSM-Planedumps die Vektortiles erzeugt und Prozessierungjobs in die Datenbank einfügt. Das Programm hat zwei Betriebsmodi – entweder wird die ID des zu produzierenden Tiles als Kommandozeilenargument übergeben oder es wird Tile-Expiry-Logfile (siehe Abschnitts 3.5) eingesehen und die darin angegebenen Tiles erzeugt. Letzteres wird im Folgenden Batch-Modus genannt.

5.1. Programmablauf und Architektur

Abbildung 5.1 zeigt ein vereinfachtes UML-Klassendiagramm von Cerepso2vt. Für ein vollständiges UML-Diagramm sei auf Anhang E verwiesen.

Programmkonfiguration  Nach dem Start des Programms werden zuerst die vom Benutzer übergebenen Kommandozeilenparameter eingelesen und in Form von je einer Instanz der Klassen postgres_drivers::Config und VectorTileGeneratorConfig gespeichert. Die Informationen, die für die Verbindung mit der von Cerepso importierten Datenbank erforderlich sind, werden von postgres_drivers::Config verwaltet, weil diese Klasse von der Bibliothek Cerepso-Postgres-Backend bereitgestellt wird und dem Konstruktor ihrer Klasse postgres_drivers::Table übergeben wird.

Jedes Tile hat eine Bounding Box  Noch während des Einlesens wird im Batch-Modus das Tile-Expiry-Logfile eingesehen. Dies geschieht mit der statischen Fabrikmethode BoundingBox::read_tiles_list(const char*), welche einen Vektor mit Instanzen der Klasse BoundingBox zurückgibt.

Die Klasse BoundingBox speichert Informationen über das geographische Gebiet, das ein einzelnes Vektortile abdeckt. Zwecks schnellen Zugriffs speichert sie sowohl die geographische Länge und Breite (WGS84) des Tiles als auch dessen ID. Andere Klassen besitzen Attribute, die eine Referenz auf BoundingBox sind.

Verbindung zur Datenbank der OpenStreetMap-Daten  Anschließend werden die Spaltendefinitionen initialisiert, wozu die Klasse postgres_drivers::Columns verwendet wird. Um die Verbindung mit der Datenbank, das Senden der Abfragen und das Auswerten der Antworten der PostgreSQL-Datenbank auf die Abfragen kümmert sich die Klasse OSMDataTable. Von ihr werden vier Instanzen (für die vier Datenbanktabellen) instanziert. Diese Klasse ist von postgres_drivers::Table der Bibliothek Cerepso-Postgres-Backend abgeleitet und ergänzt sie um Attribute für die Koordinaten der Bounding
Box, die das Gebiet des Tiles beschreibt, und um eine Methode zur Ausführung von Prepared Statements, die alle Objekte eines Typs im Gebiet des Vektortiles abfragen. Die Koordinaten werden als ein Array von C-Strings gespeichert, da die Bibliothek libpq sie bei der Ausführung der Datenbankfragen als ein Array an C-Strings erwartet. Für jede Datenbanktabelle existiert eine Instanz von OSMDataTable. Im Batch-Modus werden die Instanzen für alle Tiles wiederverwendet und besitzen deshalb eine Methode set_bbox(BoundingBox&), mit der für jedes zu produzierende Vektortile die Bounding-Box geändert werden kann.


Für jede Instanz der Klasse VectorTile wird deren Methode generate_vectortile() aufgerufen, welche die entsprechenden Datenbankabfragen veranlasst und das VektorTile schreibt.

5.2. Die Erzeugung eines Vektortiles


Die Erzeugung des Vektortiles in der Klasse OSMVectorTileImpl::generate_vectortile() läuft in folgenden Schritten ab:

1. Nodes, die im Gebiet des Vektortiles einschließlich eines Puffers um das Tile liegen, mit der SQL-Abfrage 2 aus Listing 4.7 abfragen. Alle Nodes, die von dieser Abfrage zurückgegeben werden, werden einem LocationHandler übergeben.

2. Selbiges für Ways (Abfrage 3 aus Listing 4.7). Für jeden Way, den diese Abfrage zurückgibt, wird geprüft, ob jeder Node, den die Ways referenzieren, schon von der Abfrage 1 zurückgegeben wurde und seine Position also bekannt ist. Wenn er im LocationHandler fehlt, wird er in ein Set m_missing_nodes eingefügt. Die IDs aller Ways, die von dieser Abfrage zurückgegeben werden, werden in ein Set m_ways_got eingefügt.

3. Selbiges für Relationen (Abfrage 4 aus Listing 4.7). Es wird anschließend geprüft, ob die Mitglieder dieser Relationen schon von den ersten beiden Abfragen zurückgegeben wurden. Dazu wird nach der ID im Set m_ways_got bzw. im LocationHandler gesucht. Falls die ID dort nicht gefunden wurde, wird die ID entweder in m_missing_relations, m_missing_ways oder m_missing_nodes eingefügt.

Bei der Auswertung der oben genannten drei Abfragen werden die IDs aller Objekte, die in der Ergebnismenge enthalten sind, in ein Set des jeweiligen Objekttyps eingetragen. Folgende Sets werden dafür gepflegt:

- Nodes, die schon in der Ergebnismenge enthalten sind, werden vom Location-Handler verwaltet.
- IDs der Nodes, die noch fehlen, sind im Set `m_missing_nodes` enthalten.
- IDs der Ways, die in der Ergebnismenge enthalten sind, sind im Set `m_ways_got` enthalten.
- IDs der Ways, die noch fehlen, sind im Set `m_missing_ways` enthalten.
- IDs der Relationen, die in der Ergebnismenge enthalten sind, sind im Set `m_relations_got` enthalten.
- IDs der Relationen, die noch fehlen, sind im Set `m_missing_relations` enthalten.

Somit ergeben sich nacheinander folgende Schritte, um die Ergebnismenge zu vervollständigen:

1. Fehlende Relationen abfragen (d.h. solche, die von einer Relation aus der Ergebnismenge referenziert werden). In diesem Schritt kommen weitere fehlende Ways und Nodes in den Sets hinzu, die von den jetzt abgefragten Relationen direkt referenziert werden.
2. Fehlende Ways abfragen. In diesem Schritt kommen weitere fehlende Nodes im Set `m_missing_nodes` hinzu.
3. Fehlende Nodes abfragen.


\[1\] InputIterator und OutputIterator sind Templateparameter und müssen diverse Methode, die Iterator haben, bereitstellen. Diese Funktion ist, wie viele Funktionen der C++-Standardbibliothek generisch.
5.3. Benutzung

Ein einzelnes Vektortile wird erzeugt, wenn Cerepso2vt wie folgt aufgerufen wird:

\$vectortile-generator [Optionen] X Y Z AUSGABEDATEI

X, Y und Z sind dabei die drei Bestandteile der ID eines Tiles.

Wenn mehrere Vektortiles auf einmal erzeugt werden sollen, muss Cerepso2vt wie folgt aufgerufen werden:

\$vectortile-generator [Optionen] Tile-Expiry-Liste Ausgabeformat Ausgabeverzeichnis

Dann werden aus der Datei Tile-Expiry-Liste die IDs der zu erzeugenden Tiles eingelesen und die Tiles im Format Ausgabeformat in das Verzeichnis Ausgabeverzeichnis geschrieben. Die Dateinamen sind hart kodiert und folgen dem Schema Z_X_Y.suffix.

Dieser Modus wird im Folgenden *Batch-Modus* genannt.

Cerepso2vt hat folgende Optionen:

- **--help** gibt eine Hilfe aus und beendet das Programm.
- **--verbose** gibt die ID jedes Tiles aus, wenn es geschrieben wird
- **--database-name=DATABASE** sorgt dafür, dass die OSM-Daten aus der Datenbank DATENBANK gelesen werden
- **--jobs-database=DATABASE** aktiviert das Anlegen von Prozessierungsjobs in der Datenbank DATENBANK. Ohne dieses Argument werden keine Jobs angelegt.
- **--orphaned-nodes** Ohne dieses Argument wird das erzeugte Vektortile keine Nodes enthalten, die keine Tags haben und nicht von einem Way oder einer Relation referenziert werden. Wenn solche Nodes in den Vektortiles enthalten sein sollen, ist ein räumlicher Index auf die Datenbanktabelle untagged_nodes erforderlich. Für weitere Details dazu sei auf die Kapitel 4.5 und 4.6.2 verwiesen.
- **--recurse-relations** Wenn dieser Parameter übergeben wird, werden auch alle Relationen (ohne Mitglieder) in das Vektortile geschrieben, die von einer Relation referenziert werden, welche ein Way- oder Node-Mitglied im Gebiet des zu exportierenden Tiles hat.
- **--force-overwrite** Wenn das Programm nicht im Batch Modus läuft, erlaubt dieser Parameter das Überschreiben der Ausgabedatei, falls sie schon existiert.

Als Ausgabeformate stehen OSM-XML, PBF und die anderen von Osmium unterstützten Formate zur Verfügung. Das Ausgabeformat wird im Nicht-Batch-Modus automatisch durch Osmium anhand der Dateiendung erkannt. Im Batch-Modus muss die zu verwendende Dateiendung (ohne Punkt davor) übergeben werden, Osmium wählt anhand dieser Endung dann das Ausgabeformat aus.
5.4. Hstore- und Array-Parser

Die Datenbankfelder, die vom Typ Hstore oder Array (irgendeines Datentyps) sind, werden von der Bibliothek libpq als C-Strings zurückgegeben. Die Bibliothek libpq bietet selbst keine Funktionen, um sie zu parsen und daraus entsprechende C++-Objekte zu erzeugen. Es gibt für andere Programmiersprachen zwar Bibliotheken, meist jedoch für Skriptsprachen wie Javascript, Ruby, PHP und Python. Die Mehrzahl davon verwendet reguläre Ausdrücke, was für die Geschwindigkeit nicht vorteilhaft ist. Die Python-Bibliothek pghstore von Hong Minhee implementiert den Parser in C und ist somit nur ein Python-Binding für den C-Code. Es gäbe also eine in C geschriebene (und damit in C++ direkt nutzbare) Bibliothek, um Arrays zu parsen, aber kein Äquivalent zum Parsen von Hstore-Feldern. Um pghstore sinnvoll nutzen zu können, hätten die Python-spezifischen Code-Bestandteile entfernt werden müssen. Dennoch wäre das Ergebnis nicht besonders flexibel gewesen.

Die Implementierung eines Parsers für Hstore und Arrays wurde in eine separate Bibliothek ausgelagert, da es noch keine gibt, die den Vorstellungen des Autors bezüglich der Erweiterbarkeit (siehe folgende Absätze) entspricht.

5.4.1. PostgresParser als Basis des Array- und Hstore-Parsers


Mit der Methoden has_next() kann abgefragt werden, ob es noch ein weiteres Element in der zu parsenden Struktur (Array bzw. Hstore) gibt. Ein Aufruf von get_next() gibt ein Element aus der zu parsenden Struktur (d.h. ein Element des Arrays bzw. ein Schlüssel-Wert-Paar) zurück. Der Rückgabewert der Methode get_next() ist vom Typ TSingleElementType – das ist das Templateargument der Klasse PostgresParser<typename TSingleElementType>.

Das Attribut m_string_repr ist eine Referenz auf den String, der die String-Repräsentation der zu parsenden Struktur enthält. Diese wird beim Aufruf des Konstruktors übergeben. m_current_position ist vom Typ size_t und speichert, bis zu welcher Position der String schon gelesen ist.

5.2. Abbildung 5.2: UML-Klassendiagramm der Bibliothek pg-array-hstore-parser

5.4.2. Hstore-Parser

Die Klasse HStoreParser übergibt beim Erben von der Klasse PostgresParser die Klasse StringPair als Templateargument. StringPair ist ein Alias für std::pair<std::string, std::string>.

Das Attribut m_parse_progress vom Typ HStoreParts speichert, in welchem Teil der Hstore-Struktur sich der Parser gerade befindet. HStoreParts ist eine Enumeration mit fünf verschiedenen Werten:

- None (Parser befindet sich vor dem Beginn eines Schlüssels)
- Key (Parser befindet sich derzeit in einem Schlüssel)
- Separator (Parser befindet sich derzeit zwischen Key und Value)
- Value (Parser befindet sich derzeit in einem Wert)
- End (Parser befindet sich derzeit einem Wert und dem Ende bzw. dem nächsten Schlüssel)

5.4.3. Array-Parser


TypeConversion stellt eine mit to_output_format(std::string&) eine Methode zur Konvertierung von String in das gewünschte Ausgabeformat und ruft dazu die gleichnamige Methode der der konkreten Konvertierungsimplementierung (alias TypeConversionImpl) auf. Der Rückgabetyp von to_output_format(std::string&) ist TypeConversionImpl::output_type und wird somit von der verwendeten Implementierung festgelegt, was z.B. mit

```cpp
using output_type = int64_t;
```

im Falle von int64_t geschieht.

Mit mehreren Typedefs wird diese geschachtelte Template-Struktur für den Nutzer einfacher gehalten, damit der Quellcode von Anwendungen, die den ArrayParser nutzen, nicht mit unnötigen vielen Templateargument unleserlich gemacht wird:

```cpp
using StringConversion = TypeConversion<StringConversionImpl>;
using CharConversion = TypeConversion<CharConversionImpl>;
using Int64Conversion = TypeConversion<Int64ConversionImpl>;
```

Dadurch genügt ein

```cpp
pg_array.hstore.parser::ArrayParser<Int64Conversion> my_array_parser (
  array_string_repr);
```

zur Instanziierung eines Array-Parsers, der einen Array vom PostgreSQL-Typ Bigint parsst.

Listing 5.1 enthält die für das Verständnis der Architektur wichtigsten Teile der Klasse ArrayParser. Listing 5.2 enthält die relevanten Teile der Klasse TypeConversion. Aus Platz-
Listing 5.1: Die wichtigsten Teile der Klasse ArrayParser

template <typename TConversion>
class ArrayParser : public PostgresParser<typename TConversion::output_type> {
  protected:
    TConversion m_type_conversion;
    size_t m_max_length;
  public:
    ArrayParser(std::string& string_repr) : PostgresParser<typename TConversion::output_type>(string_repr) {
      m_max_length = string_repr.length();
    }

    // Has the parser reached the end of the hstore?
    bool has_next();

    // Return the next key value pair as a pair of strings.
    typename TConversion::output_type get_next() {
      std::string to_convert;
      // vollständige Implementierung siehe Anlage
      return m_type_conversion.to_output_format(to_convert);
    }
};

gründen sind die Dokumentation und weite Teile der Implementierung (teilweise sogar einige Attribute) nicht enthalten. Hierfür sei auf den beigelegten Quellcode verwiesen.

Mit Strategien, welche von einer abstrakten Strategie abgleitet sind und deren Methoden implementieren, kann hier nicht gearbeitet werden. Die abstrakte Strategie würde den Rückgabetyp der Methode to_output_format(std::string&) festlegen und der Bibliothek die gesamte Flexibilität entreißen. Deshalb wird auf Vererbung verzichtet, zur Kompilierzeit wird jedoch sichergestellt, dass die verwendeten Konvertierungsstrategien das erforderliche Interface haben.

Listing 5.2: Die Klasse TypeConversion

template <typename TypeConversionImpl>
class TypeConversion {
  TypeConversionImpl m_impl;

  public:
    using output_type = typename TypeConversionImpl::output_type;

    output_type to_output_format(std::string& str) {
      return m_impl.to_output_format(str);
    }

    output_type return_null_value() {
      return m_impl.return_null_value();
    }
};
Derzeit sind folgende Konvertierungsimplementierungen verfügbar:

- TypeConversion<CharConversionImpl> alias CharConversion interpretiert das Array-Element als String und gibt das erste Zeichen als char zurück.
- TypeConversion<Int64ConversionImpl> alias Int64Conversion konvertiert mit std::stoll den String in eine vorzeichenbehaftete 64-Bit Ganzzahl.
- TypeConversion<StringConversionImpl> alias StringConversion implementiert das Null Object Pattern und gibt den String unverändert als String zurück.

In Cerepso2vt selbst (nicht in der Bibliothek pg-array-hstore-parser) findet sich noch eine weitere Konvertierungsimplementierung, die TypeConversion<ItemTypeConversionImpl> alias ItemTypeConversion, welche osmium::item_type zurückgibt. Sie betrachtet nur das erste Zeichen des Array-Elements. Ist es ein n, wird osmium::item_type::node zurückgegeben. Ist es ein w, wird osmium::item_type::way und, wenn es ein r ist, wird osmium::item_type::relation zurückgegeben. Da diese Implementierung Osmium als Abhängigkeit hat und nur für Cerepso2vt benötigt wird, ist sie nicht in der Bibliothek enthalten. Dieses Pattern wird auch bei der GeometryFactory von Osmium verwendet und ist ihr nachempfunden.
6. Ausblick

6.1. Cerepso

Cerepso kann in Zukunft zu einem vollwertigen Ersatz von \textit{osm2pgsql} ausgebaut werden. Dafür fehlen ihm noch u. a. folgende Funktionen:


- **Flächen werden derzeit noch nicht unterstützt.** Als geschlossene Ways erfasste Flächen sind derzeit LineStrings, Multipolygon- und Grenzrelationen sind MultiLineStrings.

- **Derzeit werden alle Tags in einer hstore-Spalte gehalten.** \textit{osm2pgsql} kann jedoch auch einzelne Spalten für einzelne Tags anlegen (die hstore-Unterstützung in \textit{osm2pgsql} wurde erst später ergänzt).

- **\textit{osm2pgsql} kann bestimmte Tags verwerfen, sodass sie gar nicht erst in der Datenbank gehalten werden und Speicherplatz belegen.** Dies gilt insbesondere für Fremdreferenzen, die bei Datenimporten aus Fremdquellen in OpenStreetMap als Tags eingefügt werden, z.B. \textit{tiger:* = *}.

- **Bei \textit{osm2pgsql} kann der Nutzer darauf Einfluss nehmen, ob ein geschlossener Way als Fläche oder als Linie importiert wird.** Dies sowie die Modifikation von Tags beim Import (TagFilter) fehlen in Cerepso.

- **\textit{osm2pgsql} hat neben der Klasse \texttt{output\_pgsql\_t} noch weitere Ausgabemöglichkeiten (Datenbankschemata), z. B. für den Geocoding-Dienst Nominatim.**

- **Statt die Nodes ohne Tags in einer PostgreSQL-Tabelle zu speichern, könnte man sie auch in einer Datei speichern.** Bei einem Array-artigen Datenformat könnten dann auch große Extrakte und sogar der Planet mit nicht allzu viel Hauptspeicherbedarf importiert werden.

Es bleibt zu hoffen, dass sich die Community auch für den Importer, wenn sein Funktionsumfang zugenommen hat, zu interessieren beginnt. Neben Cerepso gibt es schon einen \textit{osm2pgsql}-Konkurrenten – \textit{imposm3}. \textit{imposm3} ist in der Programmiersprache Go geschrieben und speichert die für die Anwendung von Diffs notwendigen Informationen nicht in Datenbanktabellen, sondern, ähnlich wie die Flatnodes-Files von \textit{osm2pgsql
in Dateien. impros m3 hat ein zu osm2pgsql inkompatibles Datenbankschema, weshalb es kein Plug-in-Ersatz für osm2pgsql ist [56], [57]. Diesem Hindernis wurde bei der Entwicklung von Cerepso aus dem Weg gegangen; das Datenbankschema von Cerepso versucht möglichst kompatibel zu osm2pgsql zu sein. Schema-Kompatibilität ist nur eine Seite der Medaille. osm2pgsql wird auch zum Import der Datenbank von Nominatim verwendet. Selbst wenn Cerepso sich nicht durchsetzt, so werden sicherlich Teile seines Quellcodes und die ein oder andere Idee den Weg zu osm2pgsql finden.

6.2. Partitionierung

Das in dieser Arbeit entwickelte und implementierte Partitionierungsverfahren ist nur ein Anfang. Die verwendeten Parameter für die minimale und maximale Zoomstufe, auf denen partitioniert wird, sowie der Schwellenwert für die maximale Anzahl an Nodes pro Tile sind auf Basis von Gefühl und Erfahrungswerten geschätzt. Es bleibt zukünftigen Arbeiten überlassen, diese Parameter zu untersuchen und zu ermitteln, ob sie geeignet sind und welche Parameter optimal sind.

Das für das Partitionierungsverfahren verwendete Programm dense_tiles ist derzeit noch nicht so effizient, wie es sein könnte. Ein Durchlauf dauert etwa 15 Minuten, für jede Zoomstufe muss es neu gestartet werden. Könnte es alle Zoomstufen auf einmal ausrechnen, könnte man damit sechs Durchläufe (d. h. etwa 90 Minuten) sparen. Es wäre dann sogar denkbar, die Partitionierung alle ein bis zwei Tage neu auszurechnen, da die Neuberechnung nur noch 15 Minuten dauert.

6.3. Cerepso2vt

An den entscheidenden Stellen ist, wie im Kapitel 5.1 dokumentiert ist, Cerepso2vt dafür vorbereitet, weitere Ausgabeformate zu unterstützen. Das wohl interessanteste Format dürfte das Mapbox-Vector-Tile-Format sein, welches das populärste Format für Vektortiles im OpenStreetMap-Umfeld geworden ist.

6.4. Prozessierungs- und Renderingserver

Die Komponenten für den Prozessierungs- und Renderingserver wurden im Rahmen dieser Arbeit nicht implementiert. Es handelt sich überwiegend nur um einige einfache Skripte, die in Python und/oder Bash implementiert worden wären. Die Arbeit hat sich stattdessen auf den Datenbankimporter Cerepso und den Cerepso2vt konzentriert.

In das Anpassen der Programme, die die Datenvalidierung durchführen, kann jedoch noch etwas „Gehirnschmalz“ fließen. So müssen beispielsweise Programme, die bislang Multipolygone prüfen, neu geschrieben werden. Schließlich stehen zu dem Zeitpunkt, zu dem ein Multipolygon geprüft wird, nur die Mitglieder zur Verfügung, die im Gebiet des Vektortiles liegen.
A. Beispiel einer Buslinie nach dem Public-Transport-Schema 2

Die Ways mit Pfeilspitzen werden nur in Pfeilrichtung befahren. Aus Platzgründen sind nur die Kurse vom Rathaus zum Stadtpark abgedruckt.

<relation id="1">
  <member type="node" ref="1" role="stop"/>
  <member type="node" ref="10" role="platform"/>
  <member type="node" ref="13" role="stop"/>
  <member type="node" ref="14" role="platform"/>
  <member type="node" ref="6" role="stop"/>
  <member type="node" ref="7" role="platform"/>
  <member type="way" ref="10" role=""/>
  <member type="way" ref="11" role=""/>
  <member type="way" ref="13" role=""/>
  <member type="way" ref="14" role=""/>
  <tag k="from" v="Rathaus"/>
  <tag k="ref" v="42"/>
  <tag k="route" v="bus"/>
  <tag k="to" v="Stadtpark"/>
  <tag k="type" v="route"/>
  <tag k="via" v="Schlossallee"/>
</relation>

<relation id="2">
  <member type="node" ref="1" role="stop"/>
  <member type="node" ref="10" role="platform"/>
  <member type="node" ref="8" role="stop"/>
  <member type="node" ref="11" role="platform"/>
  <member type="node" ref="6" role="stop"/>
  <member type="node" ref="7" role="platform"/>
</relation>
A. Beispiel einer Buslinie nach dem Public-Transport-Schema

<relation>
  <member type="way" ref="10" role=""/>
  <member type="way" ref="21" role=""/>
  <member type="way" ref="20" role=""/>
  <member type="way" ref="13" role=""/>
  <member type="way" ref="14" role=""/>
  <tag k="from" v="Rathaus"/>
  <tag k="ref" v="42"/>
  <tag k="route" v="bus"/>
  <tag k="to" v="Stadtpark"/>
  <tag k="type" v="route"/>
  <tag k="via" v="Schule"/>
</relation>
B. Versionsgeschichte eines Ways, die verborgene Version enthält


```xml
<osm version="0.6" generator="OpenStreetMap server" copyright="OpenStreetMap and contributors" attribution="http://www.openstreetmap.org/copyright" license="http://opendatacommons.org/licenses/odbl/1-0/">
<way id="224314208" changeset="16414343" timestamp="2013-06-04T00:59:32Z" version="1" visible="true" user="MrJott" uid="12165">
  <nd ref="2331336866"/>
  <nd ref="2331336852"/>
  <nd ref="2331336854"/>
  <nd ref="2331336870"/>
  <nd ref="2331336866"/>
  <tag k="building" v="yes"/>
</way>
<way id="224314208" changeset="22096782" timestamp="2014-05-03T03:38:28Z" version="3" visible="true" user="pnorman redaction revert" uid="760215">
  <nd ref="2331336866"/>
  <nd ref="2331336852"/>
  <nd ref="2331336854"/>
  <nd ref="2331336870"/>
  <nd ref="2331336866"/>
  <tag k="building" v="yes"/>
</way>
<way id="224314208" changeset="38457115" timestamp="2016-04-10T18:00:35Z" version="4" visible="true" user="catweazle67" uid="1976209">
  <nd ref="2331336866"/>
  <nd ref="2331336852"/>
  <nd ref="2331336854"/>
  <nd ref="4112166152"/>
  <nd ref="4112166147"/>
  <nd ref="2331336866"/>
  <tag k="building" v="yes"/>
</way>
</osm>
```
C. OpenStreetMap Change XML

OpenStreetMap Change XML am Beispiel des Änderungssatzes 43080643[10] (gekürzt)

```xml
<osmChange version="0.6" generator="OpenStreetMap server" copyright="OpenStreetMap and contributors" attribution="http://www.openstreetmap.org/copyright" license="http://opendatacommons.org/licenses/odbl/1-0/">
  <create>
    <node id="4459088862" changeset="43080643" timestamp="2016-10-22T12:52:28Z" version="1" visible="true" user="fred_ka" uid="190617" lat="49.4727659" lon="8.4696162">
      <tag k="fax" v="0621 4406694"/>
      <tag k="health_facility:type" v="laboratory"/>
      <tag k="name" v="Julius Gehl Kieferorthopädisches Fachlabor"/>
      <tag k="phone" v="0621 4406687"/>
      <tag k="website" v="www.kfo-labor.com"/>
    </node>
  </create>
  <create>
    <node id="4459088863" changeset="43080643" timestamp="2016-10-22T12:52:28Z" version="1" visible="true" user="fred_ka" uid="190617" lat="49.4728862" lon="8.4694746">
      <tag k="amenity" v="cafe"/>
      <tag k="name" v="Perle"/>
      <tag k="opening_hours" v="Mo-Fr 10:30-18:30; Sa 10:00-19:00; Su 11:00-19:00"/>
      <tag k="wheelchair" v="no"/>
    </node>
  </create>
  <modify>
    <node id="2527712159" changeset="43080643" timestamp="2016-10-22T12:52:28Z" version="3" visible="true" user="fred_ka" uid="190617" lat="49.4726892" lon="8.4695871">
      <tag k="amenity" v="dentist"/>
      <tag k="name" v="Dr. Carola Wißmeier"/>
      <tag k="opening_hours" v="Mo 08:15-18:00; Tu 08:15-18:00; We 08:15-13:00; Th 08:15-19:00; Fr 08:15-13:00"/>
      <tag k="phone" v="0621 - 87 20 792"/>
      <tag k="website" v="www.zahnarzt-mannheim.com"/>
      <tag k="wheelchair" v="yes"/>
    </node>
  </modify>
</osmChange>
```
F. expiries2shp


Das Programm wurde unter https://github.com/Nakaner/expiries2shp unter den Bedingungen der GNU General Public License 3 oder neuer veröffentlicht.

F.1. Benutzung

```
expiries2shp [Optionen] Eingabedateien Augabedatei
```

Augabedatei ist der Dateiname des zu erstellenden Shapefiles
Weitere Optionen:

- **--format=FORMAT** ändert das Ausgabeformat (standardmäßig „ESRI Shapefile“). Es muss die von GDAL verwendete Bezeichnung des Ausgabeformats verwendet werden.
- **--sequence=NUMMER** schreibt stets die Nummer NUMMER in die Spalte sequence des Shapefiles.
- **--ids** schreibt die x- und y-ID des Tiles in das Shapefile. Diese ergeben sich zwar aus den Koordinaten des Polygons, als separate Spalten ist der Zugriff darauf jedoch einfacher.
- **--verbose** gibt mehr Details zum Ablauf auf der Standardausgabe aus.
- **--min-zoom=ZOOM** legt die minimale Zoomstufe fest. Tiles mit einer niedrigeren Zoomstufe werden nicht in das Shapefile übernommen.
- **--max-zoom=ZOOM** dto., aber die maximale Zoomstufe

F.2. Anmerkungen zur Implementierung


Die Klasse Tile stellt Methoden zur Konvertierung der Tile-ID \((x, y \text{ und } \text{Zoom})\) in Web-Mercator-Koordinaten bereit. Die Methode \text{get_square()} ist neben dem Konstruktur die einzige öffentliche Methode. Sie gibt einen \text{Unique Pointer} auf eine Instanz von \text{OGRPolygon} zurück.

Alle übrige Funktionalität ist in Funktionen implementiert, die zu keiner Klasse gehören:

- Parsen der Tile-Expiry-Dateien mit \text{split()}
- Auflösen des Wildcard-Patterns in eine Liste von Dateinamen (Vektor vom Typ \text{String})
- Kommandozeilenargumente einlesen
- Verzeichnis und Dateiname aus einem Pfad extrahieren
G. Beiträge zur Dokumentation von Osmium

Im Rahmen dieser Arbeiten wurden Lücken in der Dokumentation der Osmium-Bibliothek geschlossen und diese Lücken dem Maintainer Jochen Topf als Pull-Requests auf der Github-Plattform zur Übernahme angeboten. Es handelte sich um folgende Pull-Requests:

- **Explain how writing files works** ([https://github.com/osmcode/libosmium-manual/pull/2](https://github.com/osmcode/libosmium-manual/pull/2)), mit der Commit-ID: 5a820e1
- **Document how to write to a buffer** ([https://github.com/osmcode/libosmium-manual/pull/3](https://github.com/osmcode/libosmium-manual/pull/3)), mit der Commit-ID: 2749bed
- **Add documentation of handlers** ([https://github.com/osmcode/libosmium-manual/pull/4](https://github.com/osmcode/libosmium-manual/pull/4)), mit der Commit-ID: 89c7ec4
- **Fix markup** ([https://github.com/osmcode/libosmium-manual/pull/5](https://github.com/osmcode/libosmium-manual/pull/5)), mit der Commit-ID: e9e3626
- **Document Collector and MultipolygonCollector** ([https://github.com/osmcode/libosmium-manual/pull/7](https://github.com/osmcode/libosmium-manual/pull/7)) mit den Commit-IDs f56347d, c7ecbd1 und 01c39b2
H. Technische Details zum Performance-Test von Cerepso

Für die Performancetests von Cerepso (Abbildung 3.2) wurde ein Server mit Ubuntu 16.04 und zwei Prozessoren vom Typ Intel Xeon E5-2623 v3 verwendet. Der Server verfügte über 256 GB RAM, der PostgreSQL-Tablespace befand sich auf SSDs.


shared_buffers = 10GB
temp_buffers = 100MB
work_mem = 1GB
maintenance_work_mem = 4GB
dynamic_shared_memory_type = posix
effective_io_concurrency = 6
fsync = off
checkpoint_timeout = 1h
log_min_duration_statement = 0
log_line_prefix = '%t [%p-%l] %q%u@%d ' 
log_timezone = 'localtime'
stats_temp_directory = '/var/run/postgresql/9.5-main.pg_stat_tmp'
avtovacuum = off

datestyle = 'iso, mdy'
timezone = 'localtime'
lc_messages = 'en_US.UTF-8'
lc_monetary = 'en_US.UTF-8'
lc_numeric = 'en_US.UTF-8'
lc_time = 'en_US.UTF-8'
default_text_search_config = 'pg_catalog.english'
Literatur


<table>
<thead>
<tr>
<th>Nr.</th>
<th>Literatur</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td>API v0.6. URL: <a href="https://wiki.openstreetmap.org/wiki/API_v0.6">https://wiki.openstreetmap.org/wiki/API_v0.6</a> (besucht am 16.11.2016).</td>
</tr>
</tbody>
</table>


[38] Martijn van Exel, Aaron Lidman u.a. What is this for? 2014. URL: https://github.com/osmlab/to-fix/issues/1 (besucht am 29.08.2016).


[60] Harald Klein und mueschel. *Keep Right*. 13. Nov. 2016. URL: http://keepright.at/report_map.php?zoom=13&lat=52.73012&lon=13.84159&layers=B0T&ch=0%5C%2C30%5C%2C40%5C%2C50%5C%2C70%5C%2C90%5C%2C100%5C%2C110%5C%2C120%5C%2C130%5C%2C150%5C%2C160%5C%2C170%5C%2C180%5C%2C191%5C%2C192%5C%2C193%5C%2C194%5C%2C195%5C%2C196%5C%2C197%5C%2C198%5C%2C201%5C%2C202%5C%2C203%5C%2C204%5C%2C205%5C%2C206%5C%2C207%5C%2C208%5C%2C210%5C%2C220%5C%2C231%5C%2C232%5C%2C237%5C%2C281%5C%2C282%5C%2C283%5C%2C284%5C%2C285%5C%2C291%5C%2C292%5C%2C293%5C%2C294%5C%2C295%5C%2C296%5C%2C297%5C%2C298%5C%2C299%5C%2C311%5C%2C312%5C%2C313%5C%2C320%5C%2C350%5C%2C370%5C%2C380%5C%2C401%5C%2C402%5C%2C411%5C%2C412%5C%2C413%5C%2C420%5C%2C60%5C%2C300%5C%2C360%5C%2C390&show_ign=1&show_tmpign=1 (besucht am 16.11.2016).


[73] Constantin Müller. Über die Unterkunftskarte. URL: http://unterkunftskarte.de/about.html (besucht am 15.11.2016).


[122] Welcome to Wikipedia users. 2016. URL: https://wiki.openstreetmap.org/wiki/Welcome_to_Wikipedia_users#We_don.27t_have_a_Notability_rule (besucht am 15.11.2016).


Klassen- und Methodenindex

C++-Standardbibliothek
std::pair<std::string, std::string>, 80
std::set, 51
std::stoll, 83

Cerepso
CerepsoColumns, 46
CerepsoConfig, 46
ExpireTiles, 49, 51
ExpireTiles::expire_from_coord_sequence(geos::geom::CoordinateSequence*), 50
ExpireTiles::expire_from_point(osmium::Location&), 50
ExpireTilesClassic, 49
ExpireTilesDummy, 49
ExpireTilesFactory, 49
ExpireTilesQuadtree, 49, 51
ImportHandler, 47, 48
ImportHandler::node(const osmium::Node&), 47
ImportHandler::way (const osmium::Way&), 47
PostgresHandler, 47
PostgresHandler::prepare_node_query(const osmium::Node&, std::string&), 47
PostgresHandler::prepare_node_query(osmium::Node&, std::string&), 47
PostgresTable, 44, 46, 48
PostgresTable::send_begin(), 48
RelationCollector, 47
RelationCollector::complete_relation(const osmium::RelationMeta&), 48

Cerepso-Postgres-Backend
postgres_drivers::Columns, 44, 46, 73
postgres_drivers::Config, 73
postgres_drivers::Table, 44, 73
postgres_drivers::Table::send_line(std::string&), 47

Cerepso2vt
BoundingBox, 73, 74
BoundingBox::read_tiles_list(const char*), 73
JobsDatabase, 74
OSMDataTable, 73, 74, 76
OSMDataTable::set_bbox(BoundingBox&), 74
OSMVectorTileImpl, 74, 81
OSMVectorTileImpl::clear(), 76
OSMVectorTileImpl::generate_vectortile(), 76
OSMVectorTileImpl::m_missing_nodes, 76, 77
OSMVectorTileImpl::m_missing_relations, 76, 77
OSMVectorTileImpl::m_missing_ways, 76, 77
OSMVectorTileImpl::m_relations_got, 77
OSMVectorTileImpl::m_ways_got, 76, 77
TVectorTileImpl, 74
TVectorTileImpl::generate_vectortile(), 74
VectorTile, 74, 76
VectorTile::generate_vectortile(), 74, 76
VectortileGeneratorConfig, 73

expiries2shp
   OutputLayer, 93, 94
   Tile, 93, 94
   Tile::get_square(), 94

GDAL
   OGRPolygon, 94

GEOS
   geos::geom::GEOSFactory, 48

libpq
   PQconnectdb(const char*), 74

osm2pgsql
   output_pgsql_t, 84
Osmium
   osmium::Area, 26
   osmium::GeometryFactory, 83
   osmium::handler::Handler, 26, 77
   osmium::handler::NodeLocationsForWays<index_type>, 47
   osmium::io::Writer, 77
   osmium::item_type, 83
   osmium::item_type::node, 83
   osmium::item_type::relation, 83
   osmium::item_type::way, 83
   osmium::memory::Buffer, 76
   osmium::Node, 26
   osmium::ObjectPointerCollection, 77
   osmium::OutputIterator, 77
   osmium::Relation, 77
   osmium::Way, 26

pg-array-hstore-parser
   ArrayParser, 10, 79, 81, 82
   CharConversion, 83
HStoreParser, 79, 80
HStoreParser::get_next(), 81
HStoreParser::m_key_buffer, 81
HStoreParser::m_parse_progress, 80
HStoreParser::m_value_buffer, 81
HStoreParts, 80
Int64Conversion, 83
ItemTypeConversion, 83
PostgresParser, 79, 80
PostgresParser::get_next(), 79
PostgresParser::has_next(), 79
PostgresParser::m_current_position, 79
PostgresParser::m_string_repr, 79
PostgresParser<typename TSingleElementType>, 79
StringConversion, 83
StringPair, 80
TConversion, 81
TypeConversion, 10, 81, 82
TypeConversion::to_output_format(std::string&), 81, 82
TypeConversion<CharConversionImpl>, 83
TypeConversion<Int64ConversionImpl>, 83
TypeConversion<ItemTypeConversionImpl>, 83
TypeConversion<StringConversionImpl>, 83
TypeConversionImpl, 81
TypeConversionImpl::output_type, 81
### Stichwort- und Softwareindex

<table>
<thead>
<tr>
<th>Stichwort/Software</th>
<th>Seiten</th>
</tr>
</thead>
<tbody>
<tr>
<td>C++-Standardbibliothek</td>
<td>51, 80</td>
</tr>
<tr>
<td>Cerepso</td>
<td>37, 40, 42, 44, 46, 51, 53, 65, 67</td>
</tr>
<tr>
<td>Cerepso-Postgres-Backend</td>
<td>44, 73, 75, 91</td>
</tr>
<tr>
<td>Cerepso2vt</td>
<td>37, 53, 73, 74, 76, 78, 81, 83</td>
</tr>
<tr>
<td>Batch-Modus</td>
<td>73</td>
</tr>
<tr>
<td>comm</td>
<td>56</td>
</tr>
<tr>
<td>Datenbankserver</td>
<td>36, 39</td>
</tr>
<tr>
<td>dense_tiles</td>
<td>10, 55</td>
</tr>
<tr>
<td>Diffs</td>
<td>24, 32, 34, 37</td>
</tr>
<tr>
<td>expires2shp</td>
<td>54, 93, 94</td>
</tr>
<tr>
<td>False Positive</td>
<td>29, 33</td>
</tr>
<tr>
<td>GDAL</td>
<td>39, 93, 94</td>
</tr>
<tr>
<td>ogr2ogr</td>
<td>93</td>
</tr>
<tr>
<td>ogrtindex</td>
<td>39</td>
</tr>
<tr>
<td>Generische Programmierung</td>
<td>74</td>
</tr>
<tr>
<td>GEOS</td>
<td>48</td>
</tr>
<tr>
<td>Importe</td>
<td>28</td>
</tr>
<tr>
<td>imposm3</td>
<td>84</td>
</tr>
<tr>
<td>Jobdatenbank</td>
<td>37</td>
</tr>
<tr>
<td>Keep Right</td>
<td>29, 31, 34, 35</td>
</tr>
<tr>
<td>Lange-Transaktion-Problem</td>
<td>34</td>
</tr>
<tr>
<td>Latest Changes on OpenStreetMap</td>
<td>27</td>
</tr>
<tr>
<td>libosmium, siehe Osmium</td>
<td></td>
</tr>
<tr>
<td>libpq</td>
<td>43, 44, 74, 79</td>
</tr>
<tr>
<td>libpqxx</td>
<td>43</td>
</tr>
<tr>
<td>MapRoulette</td>
<td>33, 34</td>
</tr>
<tr>
<td>Mikro-Tasking-Plattform</td>
<td>34</td>
</tr>
<tr>
<td>MySQL</td>
<td>29</td>
</tr>
<tr>
<td>NFS</td>
<td>39</td>
</tr>
<tr>
<td>Nodes</td>
<td>13</td>
</tr>
<tr>
<td>Nominatim</td>
<td>84, 85</td>
</tr>
<tr>
<td>OSM-API</td>
<td>18, 25, 27</td>
</tr>
<tr>
<td>osm2pgsql</td>
<td>24, 25, 37, 40, 42, 84</td>
</tr>
<tr>
<td>Flatnodes-File</td>
<td>41</td>
</tr>
<tr>
<td>Slim Tables</td>
<td>24</td>
</tr>
<tr>
<td>Slim-Tabellen</td>
<td>40, 41</td>
</tr>
<tr>
<td>OSMI, siehe OpenStreetMap Inspector</td>
<td></td>
</tr>
<tr>
<td>OSMI-Dispatcher</td>
<td>37, 39</td>
</tr>
<tr>
<td>OSMI-Worker</td>
<td>39</td>
</tr>
<tr>
<td>Osmium</td>
<td>26, 32, 38, 39, 44, 47, 48, 74</td>
</tr>
<tr>
<td>Callback</td>
<td>26</td>
</tr>
<tr>
<td>Handler</td>
<td>26</td>
</tr>
<tr>
<td>Osmose</td>
<td>29, 31, 34, 35</td>
</tr>
<tr>
<td>Osmosis</td>
<td>25, 26, 29</td>
</tr>
<tr>
<td>Overpass-API</td>
<td>29</td>
</tr>
<tr>
<td>Partitionierung</td>
<td>34, 35</td>
</tr>
<tr>
<td>pg-array-hstore-parser</td>
<td>10, 75, 79, 83</td>
</tr>
<tr>
<td>pghstore</td>
<td>79</td>
</tr>
<tr>
<td>PHP</td>
<td>29</td>
</tr>
<tr>
<td>Planetdump</td>
<td>35, 37</td>
</tr>
<tr>
<td>PostGIS</td>
<td>23, 25, 26, 29, 30, 37</td>
</tr>
<tr>
<td>PostgreSQL</td>
<td>23, 25, 26, 29, 30, 34, 36, 37</td>
</tr>
<tr>
<td>Hstore</td>
<td>23</td>
</tr>
<tr>
<td>Prepared Statements</td>
<td>44</td>
</tr>
<tr>
<td>Views</td>
<td>41</td>
</tr>
<tr>
<td>Prozessierungsserver</td>
<td>36, 37, 39</td>
</tr>
<tr>
<td>Python</td>
<td>29, 55</td>
</tr>
<tr>
<td>QA-Tiles</td>
<td>23</td>
</tr>
<tr>
<td>Qualitätssicherungswerkzeug</td>
<td>27, 34</td>
</tr>
<tr>
<td>Rechenknoten</td>
<td>35</td>
</tr>
<tr>
<td>Shapefile</td>
<td>32, 36, 39</td>
</tr>
<tr>
<td>Sichtung</td>
<td>26</td>
</tr>
<tr>
<td>Simple API for XML</td>
<td>29</td>
</tr>
</tbody>
</table>
sort, 55
SQLite, 32
Tags, 12
Tileindex, 39
To-Fix, 34
Vektortile, 37
Datenbanktabellen-, Datenbankfelder- und SQL-Befehlindex

ALTER TABLE name SET LOGGED,

BEGIN,

Cerepso,

   member_ids,
   member_roles,
   member_types,
   nodes,
   osm_changeset,
   osm_id,
   osm_lastmodified,
   osm_uid,
   osm_version,
   relations,
   tags,
   untagged_nodes,
   way_nodes,
   ways,

COMMIT,

COPY,

COPY FROM STDIN ..., 

CREATE TABLE ..., 

CREATE UNLOGGED TABLE ..., 

INSERT, 

osm2pgsql, 

   planet_osm_line, 
   planet_osm_nodes, 
   planet_osm_point, 
   planet_osm_polygon, 
   planet_osm_relations, 
   planet_osm_rels, 
   planet_osm_roads, 
   planet_osm_ways, 

Osmosis 

nodes,
Patternindex

Fabrikmethode, 73
Generische Programmierung, 79
Iterator, 44, 77
Null Object Pattern, 49, 83
Policy-based Design, 74, 76, 81
Strategy, 74, 81, 82
Wrapper, 44, 94
I. Inhalt der beiliegenden CD

Alle Pfade beziehen sich auf das Root-Verzeichnis der CD.

forked-git-repositories/ enthält im Rahmen dieser Arbeit geforkte Git-Repositories.


forked-git-repositories/osmium-contrib/ enthält im Unterverzeichnis dense_tiles die erweiterte Version des gleichnamigen Programms.

my-git-repositories/ enthält die im Rahmen dieser Arbeit erstellten Git-Repositories mit Programmquellcode.

my-git-repositories/Cerepso/ enthält den Datenbankimporter Cerepso.

my-git-repositories/Cerepso2vt/ enthält den Vektortile-Generator Cerepso2vt.

my-git-repositories/Cerepso-Postgres-Backend/ enthält die sowohl von Cerepso als auch von Cerepso2vt benötigten Wrapper für die Bibliothek libpq.

my-git-repositories/expiries2shp/ enthält das Hilfsprogramm expiries2shp.

my-git-repositories/expiry_list_filter/ enthält die beiden Python-Skript zum Filtern der Ausgabe von dense_tiles.


thesis/ enthält den Quellcode dieser Arbeit und ein PDF.


data/one-day-edits/ enthält die Expiry-Logfiles (Relationen nicht berücksichtigt), die durch den Import der stündlichen Diffs vom 5.9.2016 05:00:01 bis 6.9.2016 05:00:00 UTC erzeugt wurden. Außerdem enthält das Verzeichnis die daraus erstellten Shapefiles, welche Basis der Abbildung 4.7 sind.


ausgabe-z*-sorted.txt enthält die alphabetisch sortierte Ausgabe von dense_tiles.

z*.txt enthält die Ergebnisse nach jedem Durchlauf von split-initial-tiles.py und next-level.py. qt-z15.* sind die einzelnen Dateien des Shapefiles, welches aus z15.txt erstellt wurde und in Abbildung 4.1 abgebildet ist.
**data/expiry-lists/** enthält die Tile-Expiry-Listen für den Zeitraum 29.8.–5.10.2016. Das Unterverzeichnis `all` enthält die, die entstehen, wenn Änderungen an Relationen berücksichtigt werden. Das Unterverzeichnis `without-relations` enthält die, die entstehen, wenn Änderungen Relationen ignoriert werden.

**data/expiry-shapes/** enthält die aus den Tile-Expiry-Listen des Zeitraums 29.8.–5.10.2016 erstellten Shapefiles (nur die aggregierten, d.h. es gibt keine sich überlappenden Polygone). Das Unterverzeichnis `all` enthält die, die entstehen, wenn Änderungen an Relationen berücksichtigt werden. Das Unterverzeichnis `without-relations` enthält die, die entstehen, wenn Änderungen Relationen ignoriert werden. Die Shapefiles in diesen beiden Unterverzeichnissen dienten als Datengrundlage für die Abbildungen 4.2 bis 4.6.