HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

HTW Chur
Institut für Informations- und
Kommunikationstechnologien

MSE | MASTER OF SCIENCE
IN ENGINEERING

OSM Inspector reloaded

Project thesis report
January 2014

Author:
Lukas Toggenburger

Project partner:
Geofabrik GmbH, Karlsruhe

Examiner:
Prof. Stefan Keller, HSR

Second Examiner:
Prof. Martin Studer, HTW

This project thesis report was written in the course of the Master Study Programme Master of Science in Engineering (MSE).

Abstract

OpenStreetMap is a collaboratively maintained geodatabase. Its content is edited by an open community, similar to Wikipedia. Roads, buildings and postal addresses can be entered among other features. Currently, the database and available editors provide very limited possibilities to easily detect erratic address entries such as typing errors. To mitigate this situation, the german OSM consulting company Geofabrik GmbH created a tool called "OSM Inspector", which lets mappers see potential errors in postal address data and other categories as well. The process of computing the necessary data to visualize postal addresses proved to be complex and time-consuming: Parts of the planet file are loaded into a local PostGIS database to be processed, exported, loaded into another remote PostGIS database and be displayed using UMN MapServer. A substantial reduction of execution time and conversion to an object-oriented architecture to provide a basis for future development are addressed as specific goals. Thus, software which simplifies this process by directly converting OSM files to SQLite output processible by the UMN MapServer is presented. The software is written in C++11 and uses libosmium to read OSM files and GDAL/OGR to write SQLite output.

Abstract (Deutsch)

OpenStreetMap ist eine gemeinschaftlich gepflegte Geodatenbank. Ihr Inhalt wird von einer offenen Community editiert, ähnlich wie bei Wikipedia. Unter anderem können Strassen, Häuser und postalische Adressen eingepflegt werden. Derzeit bieten die Datenbank und die verfügbaren Editoren nur geringe Möglichkeiten, auf einfache Weise fehlerhafte Einträge wie beispielsweise Tippfehler aufzuspüren. Um diese Situation zu Verbessern hat die deutsche OSM Consulting Firma Geofabrik GmbH ein Werkzeug namens "OSM Inspector" erstellt, welches Mappern potenzielle Fehler in Adress-Daten und in weiteren Kategorien anzeigen kann. Der Prozess um die Adress-Daten für die Visualisierung aufzubereiten erwies sich jedoch als kompliziert und zeitaufwändig: Teile der Planet Datei werden in eine lokale PostGIS-Datenbank geladen um sie zu verarbeiten, zu exportieren und in eine zweite PostGIS-Datenbank zu laden um schlussendlich durch einen UMN MapServer angezeigt zu werden. Ziele der vorliegenden Arbeit sind zum einen eine deutliche Reduktion der Rechenzeit und zum andern das Überführen in eine objekt-orientierte Architektur als Grundlage für zukünftige Funktionserweiterungen. Es wird eine Software präsentiert, welche die Verarbeitung vereinfacht, indem die OSM Dateien direkt als SQLite Output geschrieben werden, welcher vom UMN MapServer verarbeitet werden kann. Die Software ist in C++11 geschrieben und verwendet libosmium um OSM Dateien zu lesen und GDAL/OGR um den SQLite Output zu schreiben.

Contents

Part I

Technical report

Chapter 1

Introduction

1.1 Problem statement

1.1.1 OpenStreetMap

OpenStreetMap (or short: *OSM*) is a database containing geographical datasets such as ways, buildings and points of interests. It is created by volunteers, called *mappers*, collecting data on the ground, often using GPS devices or tracing contours from aerial imagery. Sometimes data from official sources such as land surveying offices is included too.

OpenStreetMap was created with the intention of creating a source of geodata that is available to anyone with very few legal restrictions: Earlier the data was distributed under CC-BY-SA[1], later the licence was changed to ODbL[2]. Due to this, existing data can sometimes not be added to the OpenStreetMap database because of legal (not technical) reasons.

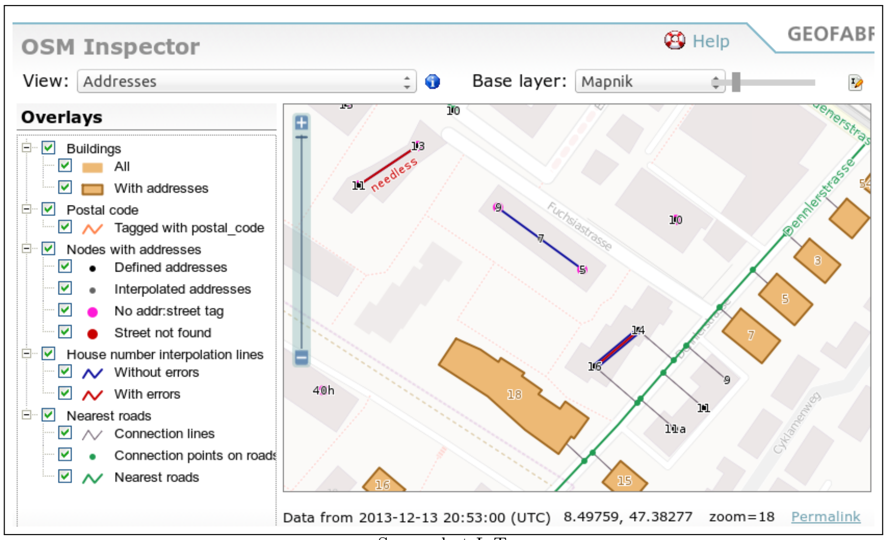
Such datasets could for example be street names and adjacent house numbers including coordinates, which often are registered at (and maybe even published from) governmental bodies. If automatical imports from such sources are not an option, one can alternatively collect this information on the ground. This can however be a tedious and error prone work.

1.1.2 OSM Inspector

OSM Inspector (or short: *OSMI*) is the name of a free, web-based service to spot potential errors in the OpenStreetMap database. It is developed and maintained by Geofabrik GmbH[3], a german company offering services around OpenStreetMap. Its purpose is to support mappers in improving the quality of the OpenStreetMap database.

The output of the OSMI are map overlays (also called *views*), which can be displayed on top of other maps and indicate problematic regions. Over a dozen different thematic overlays, such as geometry, tagging, routing or addresses, are provided. The work at hand will focus exclusively on the Addresses view. Each view has a number of overlays itself, which can be seperately activated or deactivated. The overlays are available through a web interface (shown in Figure 1.1) or as WMS or TMS.

The Addresses view is currently computed for Europe only. The reason is that calculating the necessary data takes over 21 hours for this limited area already.



View: Addresses

Base layer: Mapnik

Data from 2013-12-13 20:53:00 (UTC) 8.49759, 47.38277 zoom=18 Permalink

Screenshot L.T.

Figure 1.1: Screenshot of the *OSM Inspector* web page displaying the *Addresses* view with all its overlays activated

1.1.3 Background: Tagging scheme for addresses

Buildings and streets are some of the possible objects in the OpenStreetMap database. Giving a building an address in the real-world usually means assigning it a street and a housenumber. To make addresses machine-readable such an assignment needs to be made in the OSM world too. This is the purpose of address tagging schemes. There are two general philosophies regarding address tagging, which both have assets and limitations.

Karlsruhe Schema The simpler philosophy is called *Karlsruhe scheme*.[4, 5] Each building (OSM node or way) is tagged with `addr:street=` and `addr:housenumber=` with the assumption that such a building belongs to the closest street that has its `name=` set to exactly that what was written in the streets `addr:street=`. Optionally, other tags such as `addr:city=`, `addr:postcode=` or `addr:country=` can be set as well.

This scheme is easy to grasp and tag but it is hard to apply changes when for example a street name needs to be changed. It also adds redundant information.

Relations The second philosophy is to tag addresses using relations. For this purpose there exist two very similar types of relations which are tagged with `type=street` or `type=associatedStreet`.[6, 7] Further, both relations use a tag `name=` to denote the name of the street. Buildings are added to the relation using the role `address` (or optionally role `house` when using `type=associatedStreet` relations). House numbers are still tagged using `addr:housenumber=`.

This scheme is possibly a bit more organized but may be harder to understand for novice mappers. It is also more complex to process in software programs.

Address interpolation From a time when high-resoluation aerial imagery was not available to be used in OpenStreetMap stems the concept of address interpolation[4]. Given a series of apartment buildings in a row, one only needs to know and tag the address and location of the first and last building, while the addresses and locations of the other can be interpolated. To indicate an address interpolation an OSM way is tagged with `addr:interpolation=` and spanned between two nodes that are tagged using the Karlsruhe scheme. The key `addr:interpolation=` can have the following values:

- `all`: Increase house numbers in steps of one.
- `even`/`odd`: Increase house numbers in steps of two. Use even/odd values only.
- a number: Increase house numbers in steps of this number.
- `alphabetic`: Increase the letter part of the house number, for example 8a to 8g.

1.2 Goals

It is the goal of this project to overhaul the Addresses view of the OSM Inspector. The calculation of the underlying data shall be sped up so that updates can be delivered more often or the covered region can be expanded.

It should however still be possible to do an update in less than a day on the existing hardware at the Geofabrik.

Where possible the functional range shall be expanded according to the needs of the OSM community. Even if there is not enough time to do so, the software shall be built in a way that allows easy adding of new functionalities.

This leads to the two major goals of this thesis:

Goal 1: Speed up calculations to allow for quicker updates.

Goal 2: Object-oriented software architecture to provide basis for future development.

The original task description can be found in the appendix (see section C on page 56).

1.3 Approach

The development of the software happened in two phases:

Phase 1: Gather requirements, get to know the Osmium framework, bring things to run in a quick fashion with basic functionalities, do benchmarks and find critical spots

Phase 2: Refactor the code, implement further functionalities, bugfixes and performance improvements

To learn about the needs of the OSM community an email was sent to the swiss and german OSM mailing lists *talk-ch*[8] and *talk-de*[9]. The results are collected in section 6.4 on page 32.

Chapter 2

State of the art

2.1 Current implementation of the OSM Inspector

The current implementation of the OSM inspector updates the Addresses view daily. A data flow diagram for the current implementation of the OSM Inspector is shown in Figure 2.1. Basis for updates is an up-to-date planet file, containing the whole OpenStreetMap database. It is split up into several continents (files in PBF format), which are used for other purposes than the Addresses view as well. The PBF file for Europe is further split up into nodes, ways and tags (in textual wkbhex format) and imported into a local PostGIS database. The necessary information to be displayed is calculated using PostGIS functions such as `ST_Distance` and saved to dedicated tables (see section 7.1 on page 37 for details). These tables are then dumped, uploaded and reimported into another remote PostGIS server located at a hosting company. Finally, the overlays are rendered and served using UMN MapServer.[10]

The current implementation of the GUI has checkboxes for the following (descriptions partly taken from the GUI):

- Buildings
  - All buildings (OSM ways tagged with `building=yes`) with or without address; This is currently disabled due to moderate usefulness.
  - Buildings for which an address was assigned
- OSM ways with tagged with `postal_code=`
- Nodes with addresses

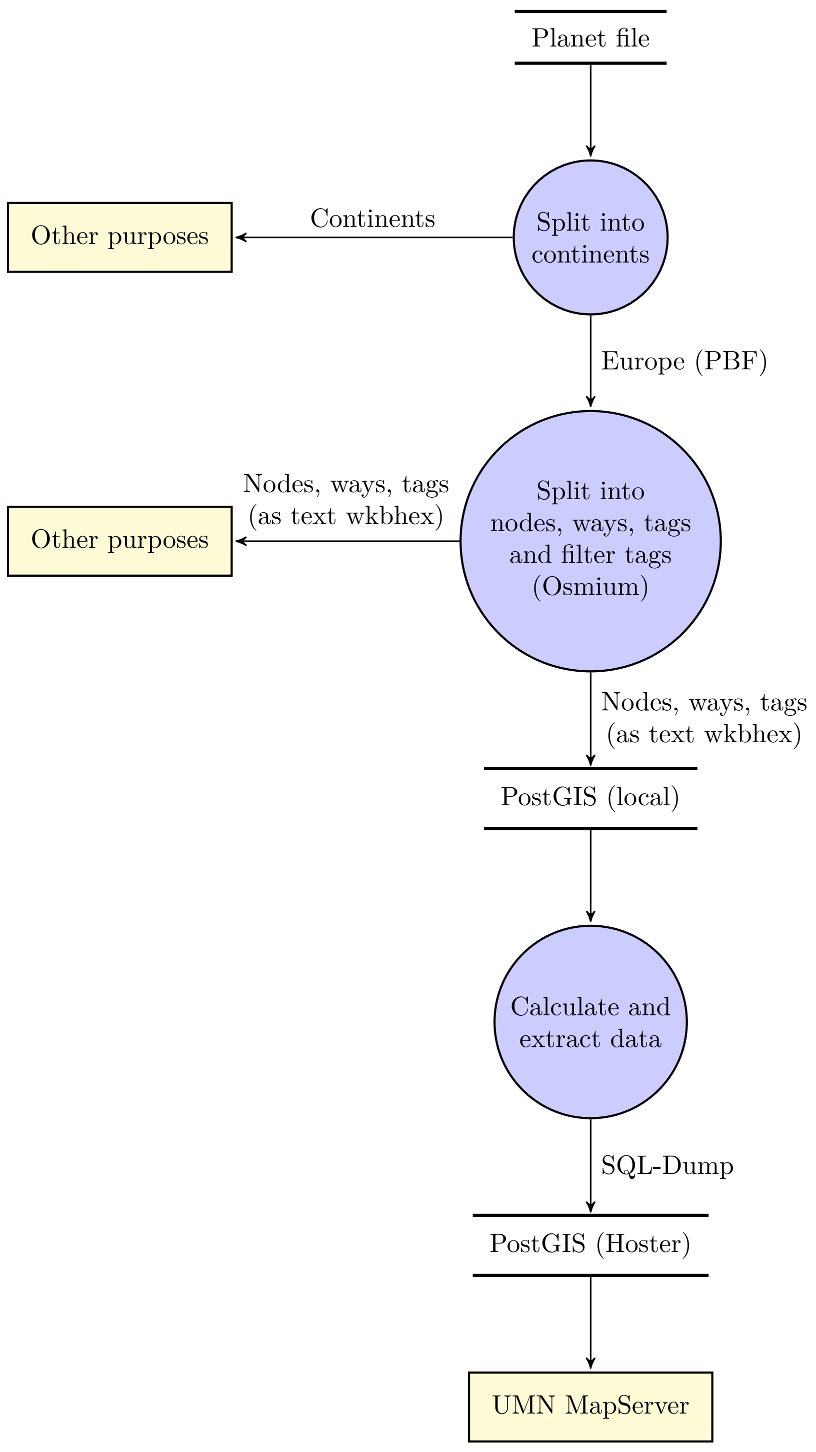

Planet file
Split into continents
Continents
Other purposes
Europe (PBF)
Split into nodes, ways, tags and filter tags (Osmium)
Nodes, ways, tags (as text wkbhex)
Other purposes
Nodes, ways, tags (as text wkbhex)
PostGIS (local)
Calculate and extract data
SQL-Dump
PostGIS (Hoster)
UMN MapServer

Figure 2.1: Data flow diagram for the current implementation of the OSM Inspector

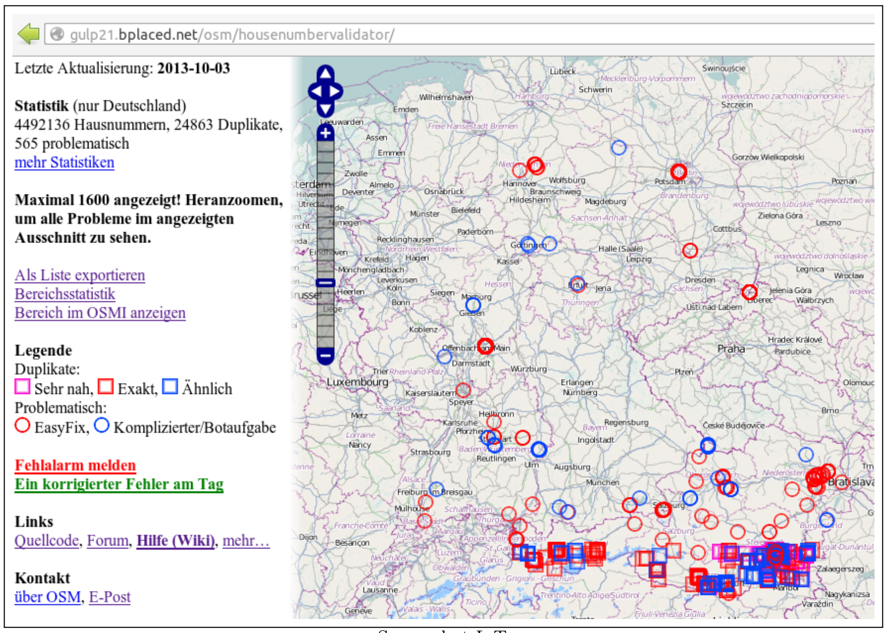
  - Defined addresses (tagged using the “Karlsruhe scheme” (see section 1.1.3 on page 10)
  - Interpolated addresses (virtual nodes whose values and positions were not mapped directly but calculated from interpolation lines; see section 1.1.3 on page 11) These nodes are not contained in the OSM database.
  - Nodes with some keys `addr:*=` set but `addr:street=` missing
  - Nodes with a defined address for which a matching street couldn’t be found in a square of about 1 km (0.01° in latitude and longitude)
- House number interpolation lines
  - Without errors: Lines between house number nodes to denote interpolation of not tagged house numbers between those nodes
  - With errors: Interpolation lines for which an inconsistent tagging was used
- Nearest roads
  - Connection lines: Lines connecting a road or building with an address to the nearest point on the road given in the `addr:street=` tag. These lines are not containted in the OSM database.
  - Connection points on roads: For every node or building with an address this is the nearest point on the corresponding road. These points are not in the OSM database.
  - Nearest roads: All roads which have an assigned node or building. These roads (together with all other roads) can typically also be seen on the base map, but appear more distinct when activated on the overlay.

2.2 Other web services

There are multiple other services along the OSM Inspector providing information about possible errors of postal address data in OSM.

2.2.1 housenumbervalidator

housenumbervalidator[11] is a web service displaying possibly duplicate, incomplete and broken address entries. The service covers the area of Germany and Austria. Its software is written in C++ and available from [12]. It is licensed under the GPLv3+ licence.


gulp21.bplaced.net/osm/housenumbervalidator/

Letzte Aktualisierung: **2013-10-03**

Statistik (nur Deutschland)
4492136 Hausnummern, 24863 Duplikate, 565 problematisch
mehr Statistiken

Maximal 1600 angezeigt! Heranzoomen, um alle Probleme im angezeigten Ausschnitt zu sehen.

Als Liste exportieren
Bereichsstatistik
Bereich im OSMI anzeigen

Legende
Duplikate:
☐ Sehr nah, ☐ Exakt, ☐ Ähnlich
Problematisch:
○ EasyFix, ○ Komplizierter/Botaufgabe

Fehlalarm melden
Ein korrigierter Fehler am Tag

Links
Quellcode, Forum, **Hilfe (Wiki)**, mehr…

Kontakt
über OSM, E-Post

Screenshot L.T.

Figure 2.2: Screenshot of the *housenumbervalidator* website

[13] lists the following defects that can be detected by the housenumbervalidator service:

- Duplicates
  - Close duplicates: Two objects have identical address entries and lie next to each other.
  - Exact duplicates: Two objects have identical address entries and identical values for the following keys: `amenity=`, `shop=`, `tourism=`
  - Possible duplicate: Two objects have identical address entries but one of the entries lacks some address data.
- Incomplete entries: The object has one of the following tags set, but another of them is missing: `addr:housenumber=`, `addr:street=`, `addr:postcode=`, `addr:country=`. Entries from relations are not considered.
- Broken entries
  - `addr:country=` isn't built of two letters.
  - The first letter of the `addr:city=` or `addr:street=` value is not a capital letter.
  - `addr:city=` looks like a street name.

  - `addr:postcode=` is not a number with a valid number of digits (Germany: 5, Austria: 4).
  - `addr:housenumber=` looks like a street name.
  - `addr:housename=` looks like a house number.
  - `addr:postcode=` contains `fix` or `unkn`.
  - `addr:street=` ends with `tr.`.
  - `addr:housenumber=` contains `<`, `..`, `?`, `fix` or `unkn`.

Objects with the following tags are being ignored:

- `fixme=*`
- `note=*`
- `street_lamp=*`
- `power=sub_station`

According to [13] the intended update interval is one day. However on 2013-10-18, according to the information displayed on the website, the last update was more than two weeks ago.

The website allows its users to report false-positive errors.

Users can enter their e-mail address on the website to receive a daily e-mail containing an address entry to check and correct. This is called *Ein korrigierter Fehler am Tag* (one corrected error per day).

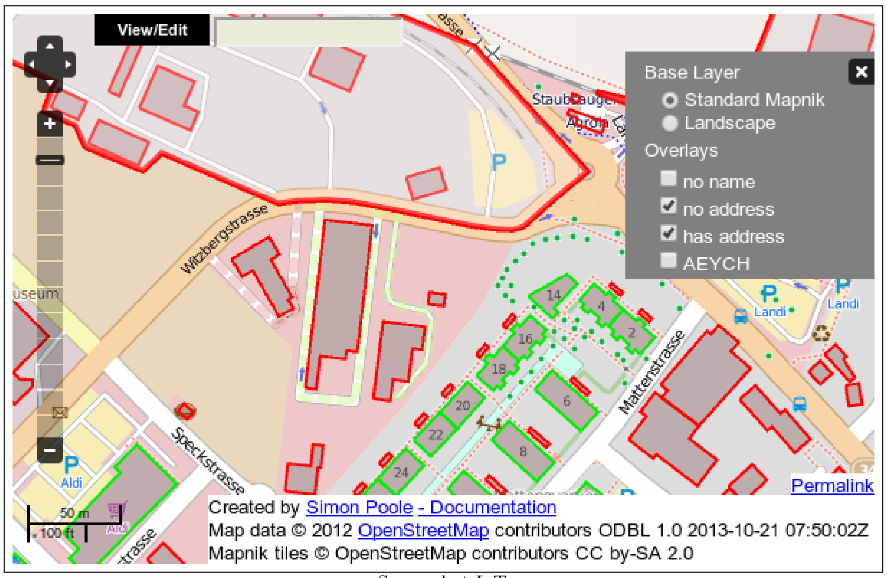
2.2.2 Simon Poole's Experimental QA site

Simon Poole is Chairman of the *OpenStreetMap Foundation* (OSMF).[14] His QA site[15] features four different worldwide map overlays:

1. No address
2. Has address
3. No name
4. AEYCH

A description of them is given in [16]:

No address The *No address* overlay highlights buildings and areas tagged with `landuse=residential` with missing address data.



Created by Simon Poole - Documentation
Map data © 2012 OpenStreetMap contributors ODBL 1.0 2013-10-21 07:50:02Z
Mapnik tiles © OpenStreetMap contributors CC by-SA 2.0

Screenshot L.T.

Figure 2.3: Screenshot of Simon Poole's *Experimental QA site*

Buildings are considered to have address data when at least one of the following tags is set:

- `addr:housenumber=`
- `addr:housename=`
- `addr:conscriptionnumber=`

Addresses for buildings are also recognized when address nodes lie inside or on the outline of a building. Furthermore, lines with `addr:interpolation=` that intersect buildings are recognized as being tagged.

`landuse=residential` areas are considered to not have address data, when there is not a single address node inside the area.

Major buildings without addresses are colored red. Minor buildings (`building=` set to `hut`, `garage`, `garages`, `rood`, `terrace`, `greenhouse` or `shed`) are colored orange. Minor buildings may not be highlighted anymore in the future.

`landuse=residential` areas without address data are colored red.

Has address The *Has address* overlay shows buildings and nodes with address information. The logic is similar to the one used in the *No address* overlay. If an address node inside or on the outline of a building is used to determine the address of a building, both the address node and the building will be highlighted.

Highlighted elements are colored green.

No name The *No name* overlay shows roads without a name.

AEYCH The *AEYCH* (Access equals yes considered harmful) overlay deals with problematic `access=` tags.

The denoted update interval is approximately five minutes.

2.2.3 Unvollständige Adressen in OSM map

An interesting approach to display erroneus address data was taken by OSM community member *ubahnverleih* in his *Unvollständige Adressen in OSM* map.[17] The map uses the Overpass API[18] to search for incomplete address entries. Results can therefore be presented in near-realtime and no server-side processing needs to be provided by the creator of the map. Instead, processing is outsourced to the Overpass server.

Two categories of incomplete addresses (nodes and ways) are shown[19]:

1. **Incomplete address:** `addr:housenumber=` is present, but one of the following is missing: `addr:street=`, `addr:postcode=` or `addr:city=`. Entries are displayed with a red pin.
2. **Missing street:** `addr:housenumber=` is present, but `addr:stret=` is missing. Entries are displayed with a blue pin.

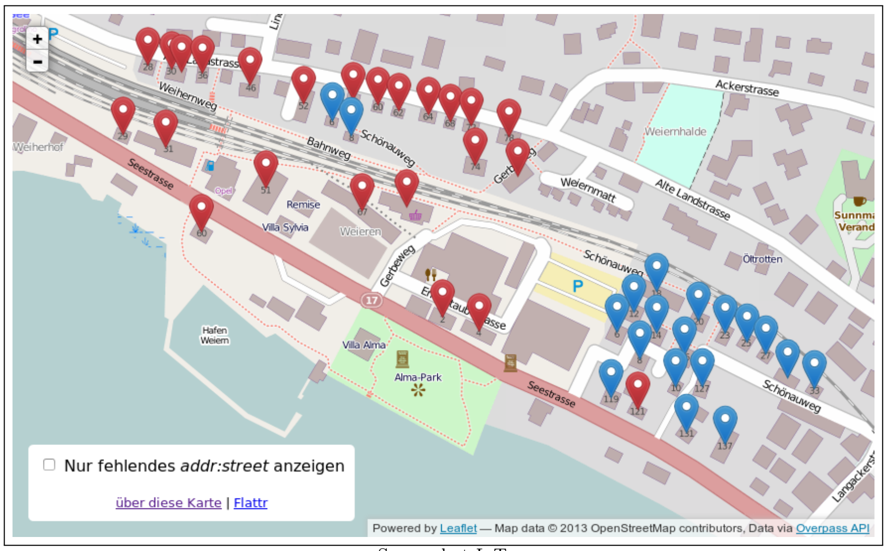
The source code of the software is available under the MIT license[20].

2.2.4 Osmose

Osmose (OpenStreetMap Oversight Search Engine) [21] is a general QA tool for OSM with a long list of checks of all kinds. It includes many checks relating to postal addresses as well.

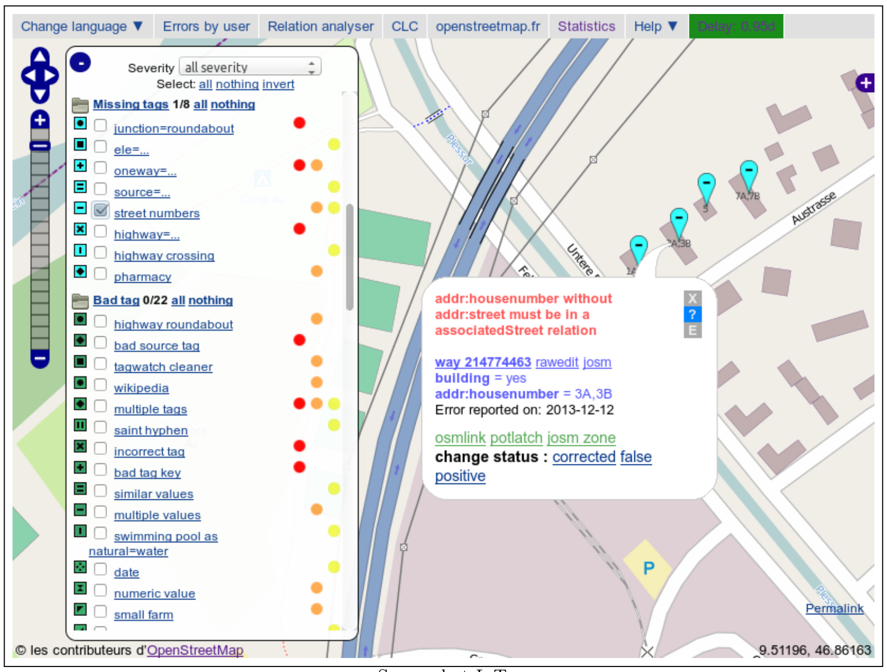
Sifting through the `analyser_osmosis_relation_associatedStreet.py` file of the Osmose source code [22] revealed the following detectable defects:

- Ways and nodes meeting all of the following conditions:
  - `addr:housenumber=` set,
  - `addr:street=` not set
  - not being member of a `associatedStreet` or `street` relation


Nur fehlendes *addr:street* anzeigen
über diese Karte | Flattr
Powered by Leaflet — Map data © 2013 OpenStreetMap contributors, Data via Overpass API
Seestrasse
Schönauweg
Ackerstrasse
Alte Landstrasse
Weihernweg
Bahnweg
Gerbeweg
Weiernmatt
Villa Sylvia
Villa Alma
Alma-Park
Weieren
Hafen Weiern

Screenshot L.T.

Figure 2.4: Screenshot of the *Unvollständige Adressen in OSM* map



addr:housenumber without addr:street must be in a associatedStreet relation

way 214774463 rawedit josm
building = yes
addr:housenumber = 3A,3B
Error reported on: 2013-12-12

osmlink potlatch josm zone
change status : corrected false positive

Screenshot L.T.

Figure 2.5: Screenshot of *Osmose*

- Relations with `type=associatedStreet` having no member with role `street`
- Relations with `type=associatedStreet` where the member with role

`street` has no `highway=` tag

- Ways and nodes in a `type=associatedStreet` relation without a role
- Nodes in a `type=associatedStreet` relation with `addr:housenumber=` not set
- Ways in a `type=associatedStreet` relation with role `house` but without `addr:housenumber=` or `addr:interpolation=` set
- Ways and nodes in a `type=associatedStreet` relation having the same value for `addr:housenumber=`
- Relations with `type=associatedStreet` containing more than one member with role `name`
- Different relations with `type=associatedStreet` having the same way as member.
- Ways and nodes in a `type=associatedStreet` relation in the `house` role that have a distance of at least 200 m to the associated road.

There are no clear indications of a specific update interval. There is however a link on the main page to a list of about 3000 sources, each with a date of the last generation and a link to a list of previous generation dates.[23] Recent entries show an update interval of approximately 2 days.

Osmose's source code is available under GPLv3 or later.[24]

2.2.5 Other resources related to quality assurance

The OpenStreetMap wiki has a long list of tools and services about quality assurance in general.[25]

There is a JOSM plugin which allows one to quickly set addresses for batches of buildings, increasing the house number for every house. It is called *House-NumberTaggingTool*. Its description can be found at [26].

OpenStreetMap community member *xylome* wrote a software called *xybot*.[27] It is a robot software which automatically fixes obvious typos of data in the OpenStreetMap database. This covers all kind of tags, including those from the Karlsruhe scheme, e.g. `add:housenumber=` will be corrected to `addr:housenumber=`. The rules concerning the Karlsruhe scheme can be found at [28].

2.2.6 Summary

Table 2.1 shows a summary of the previously described services. What is unique about the OSM inspector is its ability to visualize connections between addressed features and their belonging streets (*connection lines*). No other service does something similar. Compared to other services, the current implementation of the OSM Inspector neither has an especially broad area covered nor a quick update interval. It also is currently unable to recognize addresses in relations, detect duplicate address entries or to process feedback about false-positives. However, the connection line functionality provides the OSMI with the highest potential for a comprehensive service.

Table 2.1: Overview of services displaying possible errors in postal address data. blue: positive aspects for end-users, red: negative aspects for end-users

	housenumber-validator	**Simon Poole's QA site**	**Unvollständige Adressen in OSM**	**Osmose**	**Current OSMI**
Covered area	Germany, Austria	Worldwide	Worldwide	French-speaking countries [29]	Europe
Update interval	1 d	≈ 5 min	1 min	≈ 2 d [23]	≈ 1 d
Maintained by	gulp21	Simon Poole	ubahnverleih	Osmose maintainers	Geofabrik GmbH
Connection between address and street	No	No	No	No	Yes
Supports relations	No	Yes	No	Yes	No
Detects duplicates	Yes	No	No	Yes	No
Knows interpolation	No	Yes	No	Yes	Yes
Report false-positives	Yes	No	No	Yes	No
Programming language	C++	unknown	JS, Overpass	Python	SQL
Sourcecode available	Yes	No	Yes	Yes	on request
Soucecode licence	GPLv3+	-	MIT	GPLv3+	tbd

Chapter 3

Evaluation

3.1 Criteria for Goal 1

To quantify the execution speed of the new software, the elapsed (wall clock) time of `/usr/bin/time -v` will be measured. This value will be compared to the execution time of the current implementation.

Other parameters have an influence on the usefulness of the new implementation too, so these will be compared as well. Namely these are peak RAM usage and the size of the output which needs to be uploaded to the MapServer. It was however only a secondary objective to minimize those parameter. Peak RAM usage can be measured with `/usr/bin/time -v` (value "Maximum resident set size"). File size can be measured using `ls`.

3.2 Criteria for Goal 2

It can be expected that Geofabrik GmbH would like to extend the new software. It will be evaluated what changes are necessary to expand the content of the SQLite output file.

Chapter 4

Concept

4.1 Choice of software libraries

Geofabrik GmbH wished that libosmium[30] was to be used to process OSM data and SQLite output to be written. It is the successor of the Osmium framework[31]. The libosmium library is a header-only library written in C++11 by Jochen Topf and specifically crafted to process large blocks of OSM data (e.g. the whole planet file). It is distributed under the Boost Software License. A manual is available at [32].

The libosmium library can read OSM files in both PBF and XML (uncompressed, gzip2, bzip2 variants) format. OSM files contain nodes (points), ways (lines) and relations grouped and sorted in this order. The library can be used in such a way, that for every node/way/relation a callback function is called, which can decide what to do with these objects. To do so one or more Handlers, derived from `osmium::handler`, are to be implemented. In OSM files only nodes (but not ways or relations) carry coordinates. A way is simply defined by stating which nodes constitute a way (by referring to the nodes unique IDs). As a consequence, a callback function for a way would normally know only the way's tags and the IDs of its constituting nodes but not the nodes positions or tags. To circumvent this restriction libosmium provides a special handler object called `LocationHandler`, which does two things: Firstly, reading coordinates from all nodes and saving them in an index. Secondly, attaching the coordinates to the nodes of each way for which a callback function is called. Using `LocationHandler` is only needed when coordinates of way nodes are needed to be known inside the way callback function.

In order to write output files in a variety of formats, libosmium provides conversions from osmium objects to GDAL/OGR objects. GDAL/OGR is a C++ library to access vector data and can write to over 30 different file formats, including SQLite/Spatialite.[33, 34]

Alternative approaches to using libosmium or its predecessor and GDAL/OGR could have been:

- Loading OSM data into a spatial database such as PostGIS using e.g. osm2pgsql[35] or Imposm[36] and doing calculations in the DB (that is what has been done until now)
- Reading OSM data into a program of one's favourite programing language. The OpenStreetMap Wiki has an extensive list of libraries and frameworks[37].
- Writing to SQLite/Spatialite directly by using a SQLite library.

4.2 Finding closest point on street

In order to find the closest points on streets to draw connection lines for all addressed features, we can proceed as follows: In a first pass we gather the geometries and names of all named streets. We also save an estimated position of the street by averaging the latitude and longitude of its first and last node. In a second pass for every addresses feature we calculate the closest point by using the data from the first pass. Naturally, we only need to look at those entries whose names match the one from the addressed feature. We can do a further reduction by only considering the streets, which are close to the addressed feature by looking at the estimated street positions and applying thresholds to latitude and longitude. This leads to a selection of only a handful of streets which may contain the closest point. GDAL/OGR allows us to determine the minimum distance from addressed feature to street for every candidate street. Unfortunately, it is unable to report the *position* of the closest point. We therefore iterate over all nodes to find the closest one with the assumption that the closest node is attached to the closest segment. This assumption does hold for all cases, except pathological ones (see Figure 4.1). By analyzing the neighbouring segments of the closest node we can determine the closest point on the closest segment. This can be done using the dot product as described in [38].

An alternative to iterating all nodes of a street in order to find the closest node is described in section 9.2.5 on page 48.

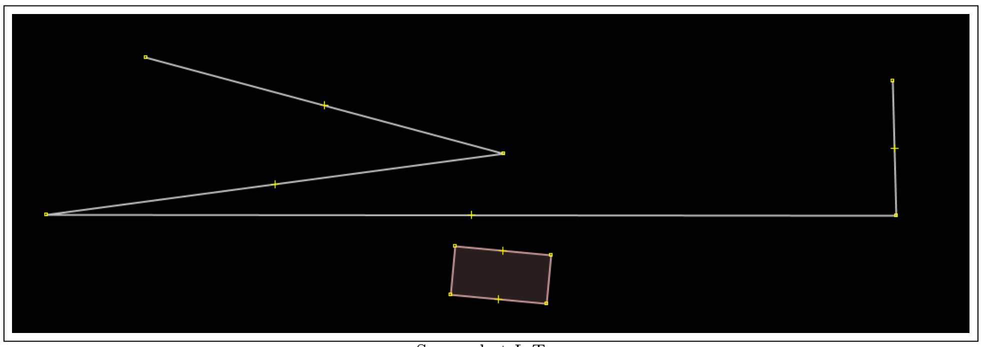

Screenshot L.T.

Figure 4.1: Pathological case in which the closest node of a building (north of it) is not attached to the closest segment (between closest node and building)

Chapter 5

Results

5.1 Achievement of goals

Both goals are addressed successfully:

Goal 1: It was possible to accelerate the execution of the software by a factor of five while providing the same functionalities (for a benchmark see section 9.1.1 on page 45).

Goal 2: Rewriting the software in C++ allows now to implement new functionalities in an object-oriented and possibly also multi-threaded manner (see section 9.1.2 on page 46). The libosmium and GDAL/OGR libraries allow a big number of input and output formats to be used with little effort. Advances in the development of the libosmium library and compilers in general will additionally speed up the software without further ado.

In a broader perspective, this software hopefully will display more address visualizations with less delay, therefore motivating more mappers to improve address data and finally lead to better and more complete OpenStreetMap data.

5.2 Outlook

Although the execution time of the new software is now five times as fast as previously, there still seems to be an issue with GDAL/OGR's SQLite driver, which could provide another massive speed boost when solved.

Generally, future work may focus on the following fronts:

- Provide more features
- Make execution faster
- Reduce output file size (helps to improve update intervals)
- Reduce peak RAM usage (reduces hardware costs / lets other things run in parallel)

There is another section in part II of this report, which elaborates the thopic of further development more deeply (see section 9.2 on page 46).

5.3 Acknowledgement

I thank the following persons for their support during the creation of this thesis:

Prof. Stefan Keller and Prof. Martin Studer for their supervision and inputs.

Frederik Ramm from Geofabrik GmbH for his useful and quick responses to my inquiries.

Jochen Topf for answering questions regarding libosmium.

Silvia Rohner for proofreading.

Part II

Software project documentation

Chapter 6

Requirements specifiation

6.1 Functional requirements

There are two basic sources for functional requirements. One source is the current implementation of the OSM Inspector. All existing functionality shall be available in the new implementation too. The other source is the input from the OSM community and from Geofabrik GmbH (see section 6.4 on page 32).

6.2 Non-functional requirements

The new implementation will run on the same hardware as the current implementation. The software shall be capable of calculating its output in less than 24 hours, given the whole planet file as input.

6.3 Data flow diagram

The new implementation will simplify the workflow of generating data to be displayed by the UMN MapServer. The data flow diagram for the new implementation is displayed in Figure 6.1 (compare with the diagram of the current implementation in Figure 2.1 on page 14).

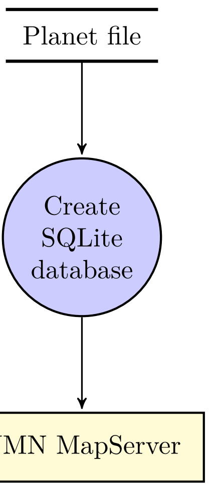


Figure 6.1: Data flow diagram for the new implementation of the Addresses view of the OSM Inspector

6.4 Ideas for new functionalities

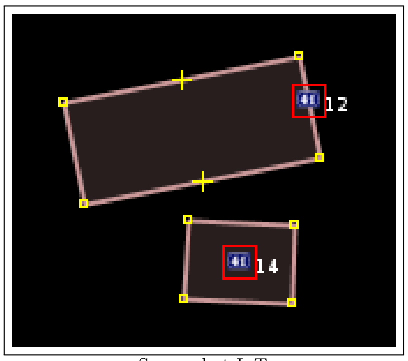
The following ideas for new functionalities stem from suggestions by the OSM community and Geofabrik GmbH.

6.4.1 Extend support of tagging methods

Addresses mapped with relations There exist two tagging schemes to tag addresses with relations: `type=street` and `type=associatedStreet` (see section 1.1.3 on page 11). These addresses should be recognized too by the OSM Inspector.

Respect tagging scheme with `addr:place=` The key `addr:place=` can be used to designate addresses when house numbers do not belong to a street but another territory such as village, island, hamlet, etc. If a nearby feature is tagged with `place=` and `name=`, a connection line shall be drawn.

Recognize addresses on/in buildings There are different philosophies on where to add a `addr:housenumber=` tag to a building. The easiest to parse is the one added to a `building=` way. Alternatively the key `addr:housenumber=` can be added to a node of the way or be placed inside the building (see Figure 6.2). The latter are more complex to process and were therefore not implemented in the previous version of the OSM Inspector. As a consequence, buildings tagged that way are not recognized as having an address.



Screenshot L.T.

Figure 6.2: House numbers tagged as node of a building and as a separate node inside a building

Draw building entries One proposition was to draw entries to buildings. These are tagged as nodes with either `entrance=` or (potentially less suitable) with `building=entrance`.

Recognize conscription numbers Some countries such as Slovakia or the Czech republic do not (only) use house numbers but so-called "conscription numbers".[39, 40] They work similarly to house numbers but are not ordered geographically. They are tagged with `addr:conscriptionnumber=`. As of January 2014 there are over 2 million tagged conscription numbers in the OpenStreetMap database.[41]

6.4.2 Detect more kinds of problematic entries

Mark missing postcode, city, country Missing entries for postcode (`addr:postcode=`), city (`addr:city=`) and maybe country (`addr:country=`) could be highlighted.

*** Highlight duplicate addresses** If more than one feature carries the same address (e.g. street name and house number) this indicates a possible error. Care must be taken not to produce false-positives: Legitimate cases are tagging a business and an apartment being located in the same building or neighbouring villages having both a separate street with a common name.

Highlight contradictory address entries There are several possibilities to add contradictory address information to features:

- Assigning one street using `addr:street=` and another using a relation

- Assigning one postcode using addr:postcode= and another using a polygon with postal_code= or boundary=postal_code
- ...

Such cases shall be highlighted.

*** Report "holes" in house numbers series / estimate missing house number positions** Sometimes there may be missing entries in a consecutive series of house numbers, e.g. 1, 2, 3, 5, 6. Such "holes" shall be highlighted. Additionally, the position of missing entries may be estimated. There surely will be legitimate cases of missing entries, therefore it should be possible to remove false-positives.

Detect overlapping buildings If two persons are mapping a building at around the same time, both specimens may end up in the OSM database. Usually, mappers won't notice this. Overlapping buildings as depicted in Figure 6.3 shall be detected. This kind of detection may better suit to the Geometry view of the OSM Inspector.

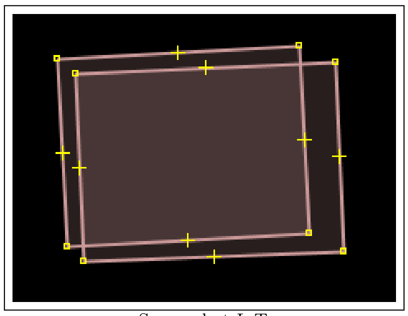

Screenshot L.T.

Figure 6.3: Overlapping buildings

Detect open buildings Buildings should always be tagged using *closed* OSM ways forming polygons. Due to a mapper's inattention such OSM ways may end up not being closed (see Figure 6.4). This is especially the case when buildings share nodes with other buildings and neighbouring walls should be drawn twice but are not. Such open buildings shall be detected. As with detecting overlapping buildings: This kind of detection may better suit to the Geometry view of the OSM Inspector.

Fuzzy search for street names If a street name given in addr:street= or a relation can not be found, a fuzzy search shall be conducted for similarly named streets.

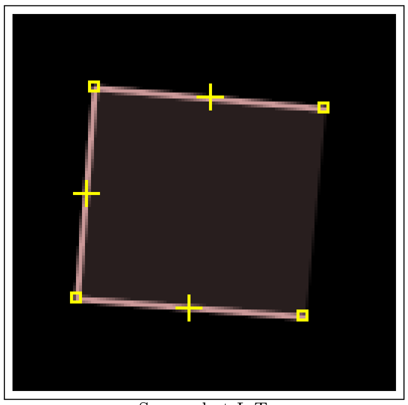

Screenshot L.T.

Figure 6.4: Open buildings

*** Heuristical trustworthiness** One idea was to calculate a quality value for every feature with an address heuristically. Two independant methods for doing so came up. As before, one would calculate for every addressed feature a connection line, i.e. the closest point on a street with the same name. For the first method, one counts the number of streets carrying another name that lie closer than the previously calculated closest point. For the second method, one looks at the connection line and counts the number of intersections with streets with other names. Values can be interpreted similarly for both methods: The higher the values, the bigger the chance that a wrong street name was assigned to a feature, e.g. to a building.

*** Visualize postcode areas and recognize isolated postcodes** One might be interested in seeing postcode areas. These can be tagged explicitly using closed OSM ways tagged with `boundary=postal_code` or `postal_code=`. Alternatively, features tagged with `addr:postcode=` could be analyzed and segmented using an appropriate area segmentation algorithm. [42] is an example of a postcode map. If there are isolated occurences of features tagged with `addr:postcode=` inside an area that is tagged with anoter postcode, this may indicate an erratic entry (or be a false-positive).

6.4.3 Other ideas

Process feedback about false-positives Some suggestions in this list would produce a noticeable amount of false-positives. These are marked with an asterisk (*). When implementing one or more of those checks, it should be considered to implement a mechanism to flag and subsequently ignore false-positives as well, e.g. by using a crowdsourcing technique.

Export buildings without an address For users en route it may be interesting to have a list of buildings without address tagging at hand. This could be implemented as downloadable GPX file or as Overpass query.

Functionalities implemented in other services Additionally to all these new suggestions it may make sense to add those functionalities that other services provide as well (see section 2.2 on page 15).

Chapter 7

Analysis

7.1 PostGIS tables

The current OSM Inspector implementation uses a number of different PostGIS tables to store everything related to the Addresses view (see also section 2.1 on page 13). The currently used tables are listed in Table 7.1.

Table 7.1: Overview of PostGIS table names used in the current implementation of the OSMI's Addresses view

Table	**Geometry type**
osmi_addresses_nodes_with_addresses	POINT
osmi_addresses_buildings	POLYGON
osmi_addresses_ways_with_addresses	POLYGON
osmi_addresses_interpolation	LINESTRING
osmi_addresses_nearest_roads	LINESTRING
osmi_addresses_connection_line	LINESTRING
osmi_addresses_nearest_points	POINT

osmi_addresses_nodes_with_addresses This table stores all OSM nodes that were assigned an address, i.e. all nodes with one of the following keys set:

- `addr:street=`
- `addr:housenumber=`
- `addr:postcode=`
- `addr:city=`
- `addr:full=`

osmi_addresses_buildings This table stores all buildings, i.e. all OSM ways tagged with `building=`.

osmi_addresses_ways_with_addresses This table stores all addressed buildings, i.e. all OSM ways that are tagged with `building=` and either `addr:street=` or `addr:housenumber=`.

osmi_addresses_interpolation This table stores all address interpolation lines, i.e. all OSM ways that are tagged with `addr:interpolation=`.

osmi_addresses_nearest_roads This table stores all streets (i.e. OSM ways with `highway=`) for which a nearby-lying addressed feature (OSM node or way) was found.

osmi_addresses_connection_line This table contains lines connecting addressed OSM nodes and ways to the nearest street with the same name.

osmi_addresses_nearest_points This table stores the intersection points of connection lines and nearest streets.

7.2 Correct use of coordinate systems

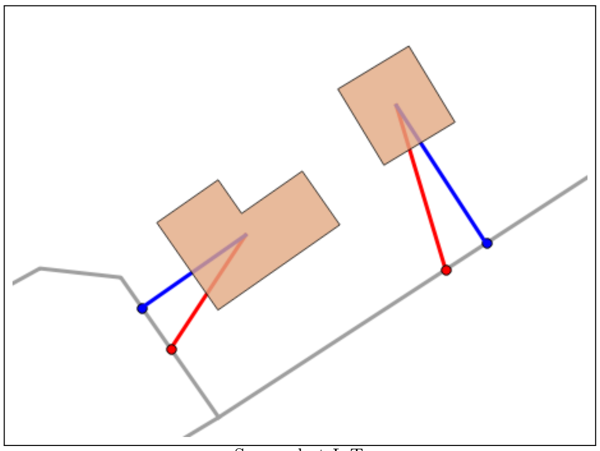
When computing closest points on streets for connection lines, care must be taken to do the calculation in the correct coordinate system. One can either choose to look for the closest point in reality but on the map it will look as a more distant point. Or one can choose to calculate a point which *looks* closest but isn't.[1] To achieve nicely displayed results and not confuse users we choose to do the latter.

The coordinate system used for the OpenStreetMap database is WGS84 (also known as EPSG:4326). The OSM Inspector's workflow converts all coordinates and displays it using "Web Mercator" (EPSG:3857). To achieve right angles at intersections of streets and connection lines, geometries of streets and buildings have to be converted to Web Mercator before calculating connection lines.[2] Figure 7.1 shows the results for calculations in both

[1]The reason is that a projection cannot be conformal and equidistant at the same time on a global scale.

[2]In the current implementatin of the OSM Inspector, after calculation of the nearest points, the results are converted back to WGS84, processed further and in the UMN MapServer again converted to Web Mercator for being displayed.

coordinate systems. The calculations were done using the dot product as described in [38].



Screenshot L.T.

Figure 7.1: Nearest points on streets computed in Web Mercator (blue) and WGS84 (red) for buildings in Switzerland displayed using Web Mercator projection.

Chapter 8

Implementation

8.1 Classes

A class diagram of the final implementation can be seen in Figure 8.1.

8.1.1 Class `FirstHandler`

The classes `FirstHandler` and `SecondHandler` are derived from the class `osmium::handler::Handler`. `FirstHandler` has two purposes:

- saving the IDs of the first and last node of every (postal address) interpolation line
- saving the geometry, way ID and approximate location of every way with tag `highway=` set in a `std::multimap` with the name of the street as key of the multimap

Saving those is necessary because in some cases we need information from *all* nodes or ways before we can compute our result. That is the case for interpolations and connection lines.

8.1.2 Class `SecondHandler`

The class `SecondHandler` iterates again over the whole planet file and saves all data to its belonging SQLite tables. It does so by either calling the implementations of `Writer` or intermediate classes which themselves call those implementations.

8.1.3 Class `Writer`

The class `Writer` is an abstract base class. The idea is to derive a class for every table that shall be written to the SQLite file. Its constructor creates an `OGRLayer`. If applicable, `Writer`'s pure virtual functions `feed_node()`, `feed_way()` and `feed_relation()` shall be implemented. They shall be called for every `osmium::Object` that `libosmium` executes a callback function for.

8.1.4 Class `ConnectionLinePreprocessor`

Not all desired outputs of the OSM Inspector fit the simple scheme that a `osmium::Object` leads to an entry in a single SQLite table: When evaluating connection lines, three tables are written: One for the connection line, one for the nearest point on the street and one for the nearest street itself. Because the same intermediate data is used, it would not make sense to create three different classes derived from `Writer`. Instead, to not break up the scheme "one `Writer`-derived class per output table", an intermediate class `ConnectionLinePreprocessor` was introduced. This intermediate class calculates everything necessary and then uses the classes `NearestPointsWriter`, `NearestRoadsWriter` and `ConnectionLineWriter` to write data to their tables.

8.1.5 Class `AltTagList`

The class `AltTagList` is a container class for storing key/value pairs (OSM tags). It is an alternative to `osmium::TagList` which was purposely made uncopyable.

8.1.6 Class `GeometryHelper`

The class `GeometryHelper` implements some helper functions like calculating the centroid of a way and doing coordinate transformations.

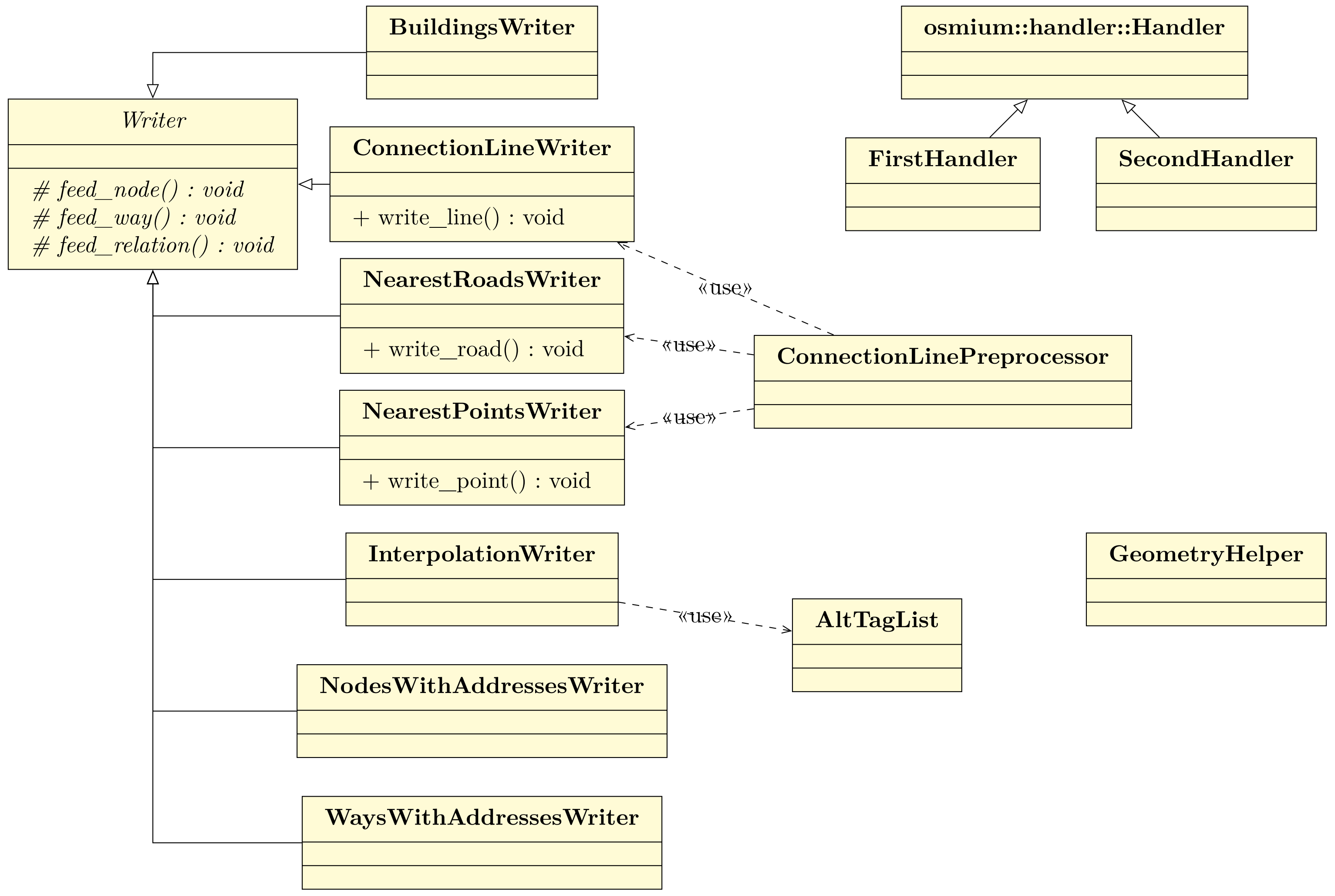


Figure 8.1: Class diagram

8.2 Testing

To continually test the code during development, the following procedure was used: A test situation containing roads, buildings and the like was created using the JOSM editor[43] and saved as OSM XML file. SQL queries were devised to make sure that contents in the tables of the SQLite output file are correct. To make sure that no errors slipped in, only the OSMI's source code or the test situation was changed at a time.

8.2.1 Testing framework

In order to simplify the tests a small command-line utility named `test_engine.pl` was built using the Perl programming language. It performs a single query to a SQLite file per execution. Using parameters a test can be formulated and performed. The parameters are:

- Path to an SQLite file
- Name or description of the test
- Query: Depending on the operator argument either an SQL query or an input argument
- Operator: Defines what to do with the query and is one of the following:
  - `=`
  - `inbbox`
  - `outofbbox`
- Expected result

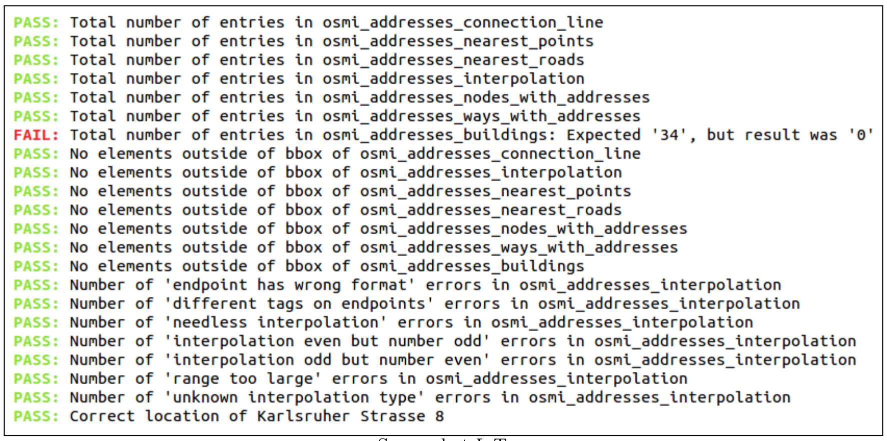
The most simple of the operators is "`=`". The belonging query could then look like this:

```
SELECT COUNT(*) FROM osmi_addresses_buildings
```

The syntax for the query parameter change when another operator (`inbbox` or `outofbbox`) is chosen. These operations count the number of features inside or outside of a given bounding box for a given table. A query could look like this:

```
osmi_addresses_nodes_with_addresses;8.7;8.8;47.2;47.3
```

A series of tests as calls to `test_engine.pl` was combined in a `bash` script called `run_tests.pl`, which provides a one stop shop for all tests.



```
PASS: Total number of entries in osmi_addresses_connection_line
PASS: Total number of entries in osmi_addresses_nearest_points
PASS: Total number of entries in osmi_addresses_nearest_roads
PASS: Total number of entries in osmi_addresses_interpolation
PASS: Total number of entries in osmi_addresses_nodes_with_addresses
PASS: Total number of entries in osmi_addresses_ways_with_addresses
FAIL: Total number of entries in osmi_addresses_buildings: Expected '34', but result was '0'
PASS: No elements outside of bbox of osmi_addresses_connection_line
PASS: No elements outside of bbox of osmi_addresses_interpolation
PASS: No elements outside of bbox of osmi_addresses_nearest_points
PASS: No elements outside of bbox of osmi_addresses_nearest_roads
PASS: No elements outside of bbox of osmi_addresses_nodes_with_addresses
PASS: No elements outside of bbox of osmi_addresses_ways_with_addresses
PASS: No elements outside of bbox of osmi_addresses_buildings
PASS: Number of 'endpoint has wrong format' errors in osmi_addresses_interpolation
PASS: Number of 'different tags on endpoints' errors in osmi_addresses_interpolation
PASS: Number of 'needless interpolation' errors in osmi_addresses_interpolation
PASS: Number of 'interpolation even but number odd' errors in osmi_addresses_interpolation
PASS: Number of 'interpolation odd but number even' errors in osmi_addresses_interpolation
PASS: Number of 'range too large' errors in osmi_addresses_interpolation
PASS: Number of 'unknown interpolation type' errors in osmi_addresses_interpolation
PASS: Correct location of Karlsruher Strasse 8
```

Screenshot L.T.

Figure 8.2: Screenshot of the output of `run_tests.pl` showing a failed test due to buildings not being written

8.2.2 Preparing the XML file containing a test situation

IDs of committed objects in the OpenStreetMap database are always positive. JOSM assigns temporary negative IDs to objects that have been created but have not yet been uploaded to the database. This is also true for saved OSM XML files. `libosmium` treats positive and negative IDs differently. But for the expected use case of processing planet files (containing committed data) all IDs will be positive. To not complicate things further a `bash` script called `makeidpositive.sh` was created to convert all object IDs to positive numbers. It does so by replacing all attributes `id` and `ref` containing negative values with their absolute value. The script needs to be executed every time after new objects were created using JOSM.

Chapter 9

Results and further development

9.1 Benchmarks

9.1.1 Goal 1

Table 9.1 shows a performance comparison between the current and the new implementation. The new implementation needs only half of the time to process the whole planet compared to the current implementation when processing Europe only.

Table 9.1: Performance comparison between the current and new OSM Inspector implementation for processing Geofabrik's PBF extract of Europe and the whole planet file

	Current implementation (Europe)	**New implementation (Europe)**	**New implementation (Planet)**
Processing time	25 h 6 min[1]	4 h 51 min[2]	10 h 35 min[3]
Output size	3.9 GB	12.9 GB (uncompr.) 3.9 GB (gzipped)	48 GB (uncompr.) 15 GB (gzipped)[4]
RAM usage	8 GB	80 GB	148 GB

[1]The import on the remote server takes additional 8 h 16 min, but processes data for other views as well.

[2]without applying compression

[3]without applying compression

[4]estimated

9.1.2 Goal 2

There are multiple ways in which one might want to expand the software. The simplest is probably to add fields to existing tables. Every table is written to by a dedicated implementation of the abstract class `Writer`. A list of fields (columns) for tables is assembled in each constructor for implementations of `Writer`. To add a field, only one additional line is needed.

To create a new table, a new implementation of `Writer` needs to be created (inside `SecondHandler`). If the content to be written can be created in one pass, it is easiest to implement the logic in the member functions `feed_node()`, `feed_way()` or `feed_relation()`. If two passes are needed, for example to have a temporary storage of some kind, a suitable container type can be created in `main()` and fed in `FirstHandler`. The constructors of `FirstHandler` and `SecondHandler` would need their argument list to be adjusted. If more than two passes were needed (see section 9.2.3 on page 47) another implementation of `osmium::handler::Handler` would need to be created.

Implementing more complex changes would of course need more adjustments. Then however, changes in the whole process (acquiring OSM data, general processing structure of libosmium, upload to and display by UMN MapServer) would probably also be needed. Generally we can say, the new implementation is ready for further updates with reasonable complexity while generating minimal overhead.

9.2 Approaches for further development

9.2.1 Speed up writes to the SQLite database file

About 80% of the time is spent executing the OSM way processing part of the software. Bearing the highest potential for an optimization, this part was analyzed further. During an analysis with `pmp`[5] it was revealed, that most of the time was used by SQLite to write features to disk. (Function `OGRLayer::CreateFeature()` used by `Writer::create_feature()`) For an exemplary dataset the deactivation of `create_feature()` brought down the execution time for the *complete* software from 73 s to 17 s. Writing to shapefiles instead takes 21 s, indicating there is a problem associated with writing SQLite output.

[5] `pmp` stands for *poor man's profiler* and is a relatively simple script to give results similar to those from a profiler software. It works by repeatedly applying `gdb` to create stack traces and using `awk` to see which functions took the most time to execute. For details see [44].

SQLite output is written through OGR's SQLite driver. When working with OGR datatypes (as we do) this has the advantage of being easily set up and changed to other output formats. Unfortunately it adds another layer of complexity when one tries to configure how the output shall be written. Using the command

```
CPLSetConfigOption("OGR_SQLITE_SYNCHRONOUS", "OFF");
```

in the C++ code, one should be able to avoid synchronizing every change safely to the storage medium and gain a noticeable speed boost.[45] Applying this has no effect on execution speed, therefore raising the suspicion that this setting gets overwritten somehow.

9.2.2 Implement multi-threading support

Some parts of `libosmium` such as the `LocationHandler` are already implemented in a multi-threaded manner using C++11's `std::thread`. One could try to implement further multi-threaded parts, e.g. when calling the different implementations of the `Writer` class in `SecondHandler`. Performance will be limited though, because new threads will need to be started for every OSM object. This is not the case when `libosmium` uses so-called "buffers".[46] These are data chunks that are larger than single OSM objects. Programming support for buffers will however need a major overhaul of most parts of the software and make it less clearly arranged.

9.2.3 Use more implementations of `Handler` to reduce peak RAM usage

As a general rule we can say: Increasing the number of implementations of `osmium::handler::Handler` may help to reduce peak RAM usage but will increase execution time when not saved elsewhere.

Currently every OSM way with `highway=` set will be saved (including its geometry) in a multimap using `name=` as the key of the multimap. One could do an additional pass and make sure that there are indeed addresses that need to be connected to each of these streets (street names) and only save geometries of those streets. For such a pass it won't be necessary to use `LocationHandler`.

As another (small) measure to reduce peak RAM usage, the two passes which manage address interpolations and connection lines could be separated, resulting in a total of 4 passes. Each functionality needs to gather data in a first pass and uses it in the second pass, but it is not necessary the datasets to be available at the same time.

When implementing more functionalities, adding more implementations of `Handler` should be considered too. In order to minimize execution time, the `LocationHandler` should not be used when coordinates of nodes of OSM ways are not needed.

9.2.4 Reduce size of SQLite output by using compression

To reduce the upload time of the SQLite file to the MapServer it is desirable to shrink its size by using compression. A quick test with small SQLite files showed that for example 7-Zip provides stronger compression than gzip used in the benchmark but also takes more time.

Another approach is to let SQLite use compressed geometries.[47, 45] This works on polygon and linestring columns by applying a kind of lossy delta-compression: Only the first and last node are saved as 64-bit float values. All other node positions are referenced to by 32-bit float delta values.

It may be worth a try to combine both techniques, as they work differently and should be able to complement each other.

9.2.5 Use k-d trees to find closest node of ways

The new implementation uses a `std::multimap` to find all streets with a given name. Comparing the estimated position of these ways with the position of addressed features, only close streets are processed further. In a next step, out of these streets, every node of every street is checked of being the closest to an addressed feature. This is done for every addressed feature.

Processing speed could possibly be improved by saving every street geometry in a separate k-d tree. This would allow to find the closest nodes quicker. It would however also increase RAM usage and can therefore be considered a tradeoff between those two.

9.3 A selection of encountered bugs

9.3.1 `abs()` takes only integer arguments

In the code there was the following check:

```
if (abs(it->second.lat - node.lat()) < 0.01 &&
    abs(it->second.lon - node.lon()) < 0.01 )
```

The intention was to check if the approximated positions of a set of streets are close in both latitude and longitude to a given node.[6] Streets matching these criteria are then further examined and a connection line is drawn to the closest street. The result of this can be seen in Figure 9.1.

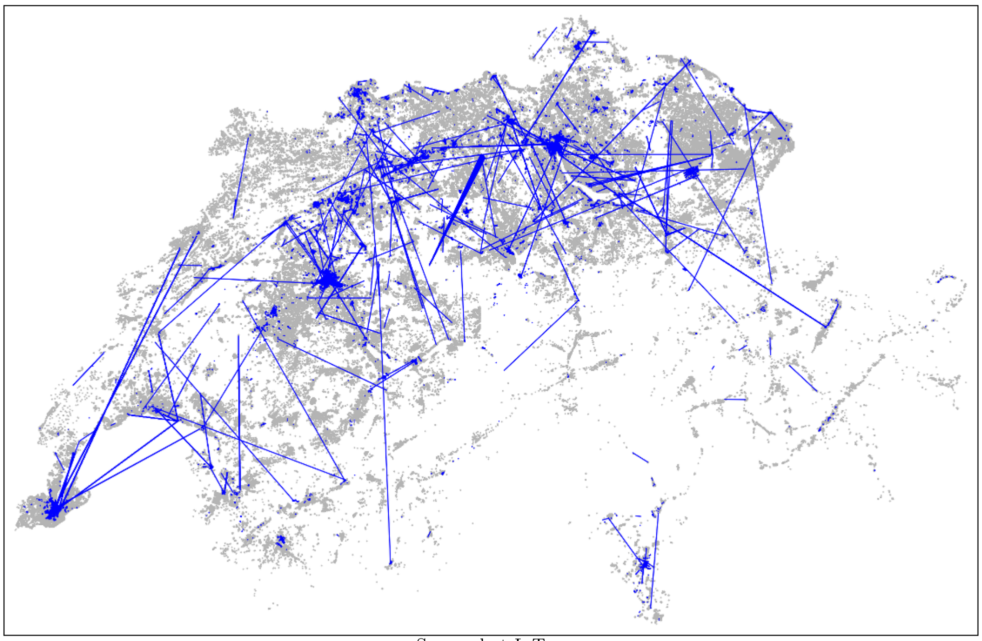

Screenshot L.T.

Figure 9.1: Erroneous overlong connection lines (blue: connection lines; grey: buildings)

As can be seen, connection lines would be drawn, even if they span all across Switzerland. This was not intended. The problem lies in the function `abs()` from `stdlib.h`: This function accepts only integer types (`int`, `long int`, `long long int`). With the latitude and longitude values both being of type `double` and their difference being close to zero, the argument for both `abs()` functions would be implicitly casted to an integer type and therefore rounded to zero, therefore making the `if` condition always true. This can be fixed by using `fabs()`, the floating point version of `abs()`.

9.3.2 Writing nearest roads multiple times

The table `osmi_addresses_nearest_roads` contains all streets as linestrings on which one or multiple connection line end. By mistake this table could receive the same linestring multiple times if there was more than one addressed feature connecting to the same street. This issue was resolved subsequently.

[6]An angular difference of 0.01° in longitude corresponds to about 750 m in Switzerland.

9.3.3 Multimap with type `char*` as key

In an old version of the software a `std::multimap` was used with data type `char*` as its key. It didn't work as expected because internally pointers were compared instead of C-strings. A custom comparison function needed to be defined as outlayed in [48]:

```
struct compare {
    bool operator()(char *left, char *right) const {
        return std::strcmp(left,right) < 0;
    }
};
```

This code was removed however, because it was not needed anymore.

Chapter 10

User manual

10.1 Compilation

10.1.1 Requirements

Due to `libosmium` being written in C++11, modern compiler versions are needed:

- GCC: 4.7.3 or newer
- clang: 3.2 or newer

10.1.2 Dependencies

Ubuntu and Debian systems need the following packages:

- `libboost-dev`
- `zlib1g-dev`
- `libbz2-dev`
- `libgdal1-dev`
- `libexpat1-dev`
- `libgeos++-dev`
- `libsparsehash-dev`
- `libprotobuf-dev`
- `protobuf-compiler`
- `libosmpbf-dev`

The header files for `libosmium` can be installed by typing:

```
make install
```

Compilation is not required since it is a header-only library.

Further information about dependencies of can be found on `libosmium`'s GitHub page [30]. There are no additional dependencies than those for `libosmium`.

10.1.3 Compiling

The directory `src/osmi` contains a `Makefile`. Calling

```
make
```

will compile a binary called `osmi` using GCC. To use the (slightly faster) clang compiler one can call:

```
CXX=clang++ make
```

10.2 Installation

The compiled executable `osmi` is a standalone application and needs no installation.

10.3 Usage

The software is able to read OSM input files in XML or PBF format. A file can be processed like this:

```
./osmi planet-latest.osm.pbf
```

By default a file called `out.sqlite` is written. If a second parameter is given, the name of the output file can be changed:

```
./osmi planet-latest.osm.pbf my-output-file.sqlite
```

An existing file will not be overwritten, the program will be aborted instead.

Appendix A

Glossary

JOSM	Java OpenStreetMap Editor
PBF	Protocolbuffer Binary Format: A binary alternative to the XML format
OSM	OpenStreetMap
OSMF	OpenStreetMap Foundation: The legal body to support OSM
QA	Quality assurance
TMS	Tile Map Service
WMS	Web Map Service

Appendix B

CD contents

```
/
├── flyer.pdf
├── personal_report.pdf
├── report.pdf
├── src/
│   ├── osmi/
│   │   ├── AltTagList.hpp
│   │   ├── BuildingsWriter.hpp
│   │   ├── ConnectionLinePreprocessor.hpp
│   │   ├── ConnectionLineWriter.hpp
│   │   ├── FirstHandler.hpp
│   │   ├── GeometryHelper.hpp
│   │   ├── InterpolationWriter.hpp
│   │   ├── main.cpp
│   │   ├── main.hpp
│   │   ├── Makefile
│   │   ├── NearestPointsWriter.hpp
│   │   ├── NearestRoadsWriter.hpp
│   │   ├── NodesWithAddressesWriter.hpp
│   │   ├── SecondHandler.hpp
│   │   ├── WaysWithAddressesWriter.hpp
│   │   └── Writer.hpp
│   └── test/
│       ├── makeidpositive.sh
│       ├── osmi-testzone.osm
│       ├── run_tests.sh
│       └── test_engine.pl
└── documents/
    ├── workjournal.pdf
    ├── minutes/
    └── status_reports/
```

Figure B.1: CD file structure

Appendix C

Task description

Semester

Semester: ☒ HS 2013 ☐ FS 20______

Vertiefungsprojekt

Folgendes Vertiefungsprojekt wird unter den genannten Bedingungen von: ***Lukas Toggenburger*** bearbeitet.

Projekttitel	OSM Inspector reloaded
Projektbetreuer/in	Prof. Stefan Keller, GeometaLab des IFS an der HSR Rapperswil
Datum Projektstarts	16. September 2013
Datum Projektende	27. Januar 2013
Projektziele und Projektbeschreibung	OpenStreetMap (OSM) ist ein Projekt, das frei nutzbare Geodaten mittels Crowdsourcing in einer Wiki-ähnlichen Datenbank sammelt. Das Projekt fordert explizit auch Amateure ohne GIS-Kentnisse zum Eintragen von Daten auf. Bei über 15‘000 Änderungssätzen pro Tag sind Fehl-Eintragungen praktisch unvermeidlich. Um potenzielle Fehler zur Kontrolle durch Menschen sichtbar zu machen, existieren verschiedene Werkzeuge. Ein bekanntes Projekt dafür ist "OSM Inspector" (OSMI) der Firma Geofabrik GmbH (http://tools.geofabrik.de/osmi/). Eine Web-Karte zeigt in unterschiedlichen Ansichten potenzielle Fehler an. Der Dienst kann auch via standardisierten Webservice für Rasterdaten (WMS) benutzt werden. Eine der Ansichten, die Adress-Ansicht für postalische Gebäude-Adressen und Hausnummern, wird derzeit nur für Europa berechnet. Der Grund dafür ist, dass die Berechnung der benötigten Daten bereits in diesem geografisch reduzierten Umfang über 21 Stunden dauert. Das Ziel des Projekts ist es, die Berechnung der Adress-Ansicht zu überarbeiten und zu beschleunigen. Die Einschränkung auf Europa soll aufgehoben werden. Aktualisierungen der Daten sollen mit derselben Hardware dennoch täglich oder noch häufiger durchgeführt werden können. Eine grobe Schätzung hat ergeben, dass sich bei der Ausweitung auf die gesamte Welt der rechnerische Aufwand ungefähr verdoppeln wird. Das Laden von Daten und Ausführen der Berechnungen in der derzeit vorliegenden Datenbank als Zwischenspeicher soll aufgehoben werden. Stattdessen soll mit angepassten Datenstrukturen die möglichst im Speicher gehalten werden, eine performantere Lösung implementiert werden.

	Ziel ist es, mittels der neusten Version des Osmium-Frameworks (http://osmcode.org/osmium/) einen Ersatz für die bestehende PostGIS-Datenbank zu erstellen. Als Eingabe für die neue Software soll wie bisher ein Planet-File im PBF Format dienen. Ausgabe soll nicht mehr ein PostGIS-Dump wie bis anhin sein, sondern ein SQLite / Shape-File. Auf dem Webserver sollen die Daten wie bisher mittels UMN MapServer als WMS zur Verfügung gestellt werden. Desweiteren soll bei der OSM-Community Feedback eingeholt werden, welche neuen Funktionen gewünscht werden. Sofern möglich werden diese Wünsche ebenfalls berücksichtigt.
Projektpartner	• Geofabrik GmbH, Karlsruhe, Deutschland • OSM Community
zu erwartende Ergebnisse	• Projektbericht • Software-Dokumentation • Persönlicher Bericht • Wissenschaftliches Paper • Flyer • Präsentation (nach Projektende) • Software, welche vom Projektpartner in die bestehende Infrastruktur eingebaut werden kann
zu erarbeitenden Kompetenzen (Fach-, Methoden- und Selbstkompetenzen)	• Der Student kennt die in OpenStreetMap verwendeten Datenstrukturen und kann OSM-Daten effizient verarbeiten. • Der Student kann Softwareprojekte erfolgreich durchführen und nach akzeptierten Standards verständlich dokumentieren. • Der Student kann Fragen zu IT-Architekturen bearbeiten. • Der Student kennt Grundzüge der C++11 Sprache.
Beschreibung der Leistungsbeurteilung	Gemäss HSR/HTW Richtlinien
ECTS-Credits	12

Appendix D

Declaration of authenticity

Ich erkläre hiermit,

- dass ich diese Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und erlaubten Hilfsmittel benutzt habe. Alle Stellen, die wörtlich oder sinngemäss aus Quellen entnommen wurden, habe ich als solche gekennzeichnet. Mir ist bekannt, dass anderenfalls die Hochschulleitung zum Entzug der aufgrund der Arbeit verliehenen Qualifikation oder des auf dieser Arbeit verliehenen Titels berechtigt ist.
- dass ich die Nutzungsrechte bezüglich dieser Arbeit für die Dauer meiner Studien an die HTW Chur abtrete und ihr damit das Recht einräume, das vorliegende Werk zu verwalten und zu nutzen und im Rahmen der eingeräumten Rechte dieses Werk auch Dritten zugänglich zu machen.
- dass ich ohne ausdrückliche Zustimmung von Auftraggeber und Referent keine Kopie dieses Berichts oder Teilen davon an Dritte weitergeben und die Arbeit oder Teile davon auch nicht veröffentlichen werde.

Chur,

Lukas Toggenburger

Appendix E

Bibliography

[1] Creative Commons. *Creative Commons Deed Attribution-ShareAlike 2.0 Generic*. URL: `http://creativecommons.org/licenses/by-sa/2.0/` (visited on 2013-08-09).

[2] Open Data Commons. *ODC Open Database License (ODbL) Summary*. URL: `http://opendatacommons.org/licenses/odbl/summary/` (visited on 2013-08-09).

[3] *Geofabrik GmbH website*. URL: `http://www.geofabrik.de` (visited on 2014-01-24).

[4] OpenStreetMap Wiki authors. *OpenStreetMap Wiki: Addresses*. 2014-01-14. URL: `http://wiki.openstreetmap.org/w/index.php?title=Addresses&oldid=981200` (visited on 2014-01-25).

[5] OpenStreetMap Wiki authors. *OpenStreetMap Wiki: Karlsruhe Schema*. 2013-11-06. URL: `http://wiki.openstreetmap.org/w/index.php?title=Karlsruhe_Schema&oldid=962717` (visited on 2014-01-25).

[6] OpenStreetMap Wiki authors. *OpenStreetMap Wiki: Relation:street*. 2013-11-06. URL: `http://wiki.openstreetmap.org/w/index.php?title=Relation:street&oldid=962709` (visited on 2014-01-21).

[7] OpenStreetMap Wiki authors. *OpenStreetMap Wiki: Relation: associatedStreet*. 2013-11-06. URL: `http://wiki.openstreetmap.org/w/index.php?title=Relation:associatedStreet&oldid=962710` (visited on 2014-01-21).

[8] *talk-ch mailling list*. URL: `http://lists.openstreetmap.ch/mailman/listinfo/talk-ch` (visited on 2014-01-24).

[9] *talk-de mailing list*. URL: `https://lists.openstreetmap.org/listinfo/talk-de` (visited on 2014-01-24).

[10] Regents of the University of Minnesota. *MapServer website*. URL: `http://mapserver.org/` (visited on 2014-01-24).

[11] gulp21. *housenumbervalidator*. URL: `http://gulp21.bplaced.net/osm/housenumbervalidator/` (visited on 2013-10-18).

[12] gulp21. *housenumbervalidator source code*. 2011-09-10. URL: `https://github.com/gulp21/housenumbervalidator` (visited on 2013-10-18).

[13] OpenStreetMap Wiki authors. *OpenStreetMap Wiki: User:Gulp21 / housenumbervalidator*. 2013-05-19. URL: `http://wiki.openstreetmap.org/w/index.php?title=User:Gulp21/housenumbervalidator&oldid=905738` (visited on 2013-10-18).

[14] *OpenStreetMap Foundation*. URL: `http://www.osmfoundation.org` (visited on 2014-01-24).

[15] Simon Poole. *Experimental QA site*. 2013-03-23. URL: `http://qa.poole.ch` (visited on 2013-10-18).

[16] OpenStreetMap Wiki authors. *OpenStreetMap Wiki: Noname and noaddress*. 2013-09-25. URL: `http://wiki.openstreetmap.org/w/index.php?title=Noname_and_noaddress&oldid=946764` (visited on 2013-10-21).

[17] ubahnverleih. *Unvollständige Adressen in OSM*. URL: `http://osm.lyrk.de/address/` (visited on 2013-10-21).

[18] Roland Olbricht. *Overpass API*. URL: `http://overpass-api.de` (visited on 2013-10-21).

[19] ubahnverleih. *About Unvollständige Adressen in OSM*. URL: `http://osm.lyrk.de/address/` (visited on 2013-10-21).

[20] ubahnverleih. *Unvollständige Adressen in OSM source code*. URL: `https://github.com/ubahnverleih/unvollst-ndige-adressen-in-OSM` (visited on 2013-10-21).

[21] Osmose maintainers. *Osmose*. URL: `http://osmose.openstreetmap.fr` (visited on 2013-10-22).

[22] Osmose maintainers. *Osmose: analyser_osmosis_relation_associatedStreet.py*. 2013-07-04. URL: `https://gitorious.org/osmose/backend/source/12153052d9065decc7fa64b1caec7940865e27a5:analysers/analyser_osmosis_relation_associatedStreet.py` (visited on 2013-10-22).

[23] Osmose maintainers. *Osmose – Last updates*. URL: `http://osmose.openstreetmap.fr/en/control/update` (visited on 2013-10-22).

[24] Osmose maintainers. *Osmose source code*. URL: `https://gitorious.org/osmose` (visited on 2013-10-22).

[25] OpenStreetMap Wiki authors. *OpenStreetMap Wiki: Quality assurance*. 2013-12-02. URL: `http://wiki.openstreetmap.org/w/index.php?title=Quality_assurance&oldid=970664` (visited on 2014-01-23).

[26] OpenStreetMap Wiki authors. *JOSM / Plugins / HouseNumberTaggingTool*. 2013-04-21. URL: http://wiki.openstreetmap.org/wiki/JOSM/Plugins/HouseNumberTaggingTool (visited on 2014-01-25).

[27] OpenStreetMap Wiki authors. *User:Xybot*. 2012-01-09. URL: http://wiki.openstreetmap.org/w/index.php?title=User:Xybot&oldid=721673 (visited on 2014-01-25).

[28] OpenStreetMap Wiki authors. *User FixKarlsruheSchema:Xybot*. 2009-05-08. URL: http://wiki.openstreetmap.org/w/index.php?title=User_FixKarlsruheSchema:Xybot&oldid=268053 (visited on 2014-01-25).

[29] Osmose maintainers. *Countries analyzed by Osmose*. URL: http://osmose.openstreetmap.fr/fr/api/0.2/meta/countries (visited on 2013-10-22).

[30] Jochen Topf. *GitHub – osmcode/libosmium*. URL: https://github.com/osmcode/libosmium (visited on 2014-01-19).

[31] Jochen Topf. *GitHub – joto/osmium*. URL: https://github.com/joto/osmium (visited on 2014-01-24).

[32] Jochen Topf. *GitHub – osmcode/libosmium-manual*. URL: https://github.com/osmcode/libosmium-manual (visited on 2014-01-24).

[33] Open Source Geospatial Foundation. *OGR Simple Feature Library*. URL: http://www.gdal.org/ogr/index.html (visited on 2014-01-24).

[34] Open Source Geospatial Foundation. *OGR Vector Formats*. URL: http://www.gdal.org/ogr/ogr_formats.html (visited on 2014-01-24).

[35] *GitHub – openstreetmap/osm2pgsql*. URL: https://github.com/openstreetmap/osm2pgsql (visited on 2014-01-24).

[36] Omniscale GmbH & Co. KG. *Imposm website*. URL: http://imposm.org (visited on 2014-01-24).

[37] OpenStreetMap Wiki authors. *OpenStreetMap Wiki: Frameworks*. 2014-01-06. URL: http://wiki.openstreetmap.org/wiki/Frameworks (visited on 2014-01-24).

[38] Joseph O'Rourke. *comp.graphics.algorithms FAQ*. URL: http://femto.cs.illinois.edu/faqs/cga-faq.html#S1.02 (visited on 2014-01-26).

[39] OpenStreetMap Wiki authors. *OpenStreetMap Wiki: Key: addr: conscriptionnumber*. 2013-05-19. URL: http://wiki.openstreetmap.org/w/index.php?title=Key:addr:conscriptionnumber&oldid=905706 (visited on 2014-01-21).

[40] Wikipedia contributors. *Wikipedia: Konskriptionsnummer*. 2013-12-10. URL: http://de.wikipedia.org/w/index.php?title=Konskriptionsnummer&oldid=125303201 (visited on 2014-01-21).

[41] Jochen Topf. *Taginfo: addr:conscriptionnumber.* URL: http://taginfo.openstreetmap.org/keys/addr%3Aconscriptionnumber (visited on 2014-01-21).

[42] *OSM Postcode Map.* URL: http://osm.wno-edv-service.de/plz/ (visited on 2014-01-21).

[43] *JOSM.* URL: http://josm.openstreetmap.de (visited on 2014-01-24).

[44] Domas Mituzas and Mark Callaghan. *http://poormansprofiler.org/.* URL: http://poormansprofiler.org/ (visited on 2014-01-23).

[45] *SQLite / Spatialite RDBMS.* URL: http://gdal.org/ogr/drv_sqlite.html (visited on 2014-01-22).

[46] Jochen Topf. *libosmium manual: Buffers.* URL: https://github.com/osmcode/libosmium-manual/blob/master/src/buffers.md (visited on 2014-01-22).

[47] *SpatiaLite: 3D and compressed geometries.* URL: http://www.gaia-gis.it/spatialite-2.4.0/SpatiaLite-Geometries-Addendum.pdf (visited on 2014-01-24).

[48] Rob. *Is it possible for std::multimap::equal_range to return incorrect results?* 2011-05-12. URL: http://stackoverflow.com/questions/5982760/is-it-possible-for-stdmultimapequal-range-to-return-incorrect-results/5983011#5983011 (visited on 2014-01-25).