An Algorithm Based Methodology for the Creation of a Regularly Updated Global Online Map Derived From Volunteered Geographic Information

Marcus Goetz, Johannes Lauer, Michael Auer
Chair of GIScience, Institute of Geography
University of Heidelberg
Heidelberg, Germany
{m.goetz, jlauer, auer}@uni-heidelberg.de

Abstract— **Global online maps are an important tool and data sets such for such maps are normally provided by commercial providers or public authorities. Nevertheless, the ever expanding trend of collaboratively collected geodata by hobbyists, namely Volunteered Geographic Information (VGI), increases regarding both data quantity and quality. Therefore, VGI can be considered as a real alternative data source for the provision of a global online map service, similar to those provided by Google Maps or Bing Maps. Therefore, an online map service needs to be created, whereby relevant data comes from a regularly updated database containing VGI. Due to the dynamic and fast changing nature of VGI sources, the workflow for processing VGI data needs to be automated on a regular base. The particular innovation of the here presented approach is that after an initial data import all required processing steps for transforming VGI data into a map-optimized data structure, is done internally with SQL database functions. That is, the processing is purely based on database technology and no additional software is required. The developed system uses standards and open-source software and is publicly available at www.osm-wms.de. The data can be consumed by either using a user adaptable standardized WMS, or via a high-performance web map application with partly pre-rendered map tiles. With the here presented approach, an regularly updated map based on open data can be provided.**

Keywords-Open Geospatial Consortium; OpenStreetMap; PostgreSQL; Volunteered Geographic Information; Web Map Service

I. INTRODUCTION

Maps are an important tool for diverse planning activities such as route planning, urban planning or agricultural planning. Moreover, services such as Google Maps or Bing Maps additionally push the usage of online maps, thus global online maps are omnipresent in professional and public areas. The before mentioned services are based on data collected by commercial data providers such as Teleatlas or Navteq.

Trying to push and evolve the spirit of the Web 2.0 approach, i.e., collaboratively providing and sharing information over the web by a broad mass, a new kind of geographic data source has arised during the last couple of years. This data source, namely called Volunteered Geographic Information (VGI), describes an ever expanding group of users, which collects geographic data in a voluntary and collaborative manner [1]. Thereby, different users with different levels of skills create spatial data by either performing personal measurements via GPS etc. or by tracing publicly available aerial images such as those provided by Bing. Afterwards, this data is uploaded to a Web 2.0 community and shared with other users, which are also allowed to reedit existing data or to use existing data at no charge. Additionally, also other geo-referenced information such as geo-referenced pictures or place locations can be considered as VGI. There are many different communities and portals sharing and collecting VGI, and there is an enormous potential arising from six billion humans acting as remote sensors [2]. The OpenStreetMap (OSM) community can be considered as the most prominent example of such a VGI community. With more than 400,000 registered users [3], i.e., more than 400,000 potential contributors, the OSM community grew rapidly considering the available data (Cf. chapter 3 for more details on the data structure). General statements about the quality of the OSM data (regarding both accuracy and completeness) from a global perspective are hard to tell, because both amount and quality vary between different regions. Nevertheless, diverse evaluations performed by Zielstra & Zipf [4], Haklay [5], Neis et. al. [6] or Ludwig et. al. [7] have proven that, especially in urban regions, OSM is able to compete against or even surpass data provided by commercial providers or governmental authorities. This is also the main motivation for providing an online map service similar to those of Google Maps or Bing Maps, whereat the data is purely based on VGI data from OSM. Therefore, the main contribution of this paper is the presentation of a workflow for creating a regularly updated database with VGI for the automated provision of a global online map, whereby the main processing steps are performed inside the database. The key characteristic of the presented approach is the innovative processing methodology for processing fast changing and dynamic geodata from VGI sources.

The rest of this paper is organized as follows: First, there is a brief overview about related work, followed by an introduction to OSM. Afterwards, a workflow for the provision of a regularly updated VGI database for map services is presented. Thereafter, a short introduction of the system architecture is given. Concluding, the developed methodology for materializing database views is presented and the workflow is demonstrated. Finally, a summary of the presented work is provided, and future work on this urging topic is discussed.

 ISBN: 978-1-61208-178-6

II. RELATED WORK

Online maps, no matter whether they are focused on urban regions or on a global perspective, are omnipresent in the internet and it seems as if they have displaced ordinary paper maps. Some of the most famous examples for global online maps are Google Maps [8] or Bing Maps [9], whereby these are both based on data provided by commercial data providers such as Navteq, Tele Atlas, etc.

In contrast to those "commercial maps", there is a global online map available on the OpenStreetMap project page [10], whereby this map is purely based on collaboratively and voluntarily collected geodata (i.e., VGI). The OSM map illustrates different map features such as streets, naturals (e.g., forests or water areas), Point-of-Interests (POIs), rails, waterways etc, thus provides a detailed overview about both urban and rural areas. As stated above, diverse research approaches demonstrated that OSM is able to compete against commercial data providers, thus a map based on VGI from OSM can also be utilized for official processes such as urban planning. However, the architecture of the OSM project page is not based on Open Geospatial Consortium standards such as Web Map Services (WMS). It is not possible to set a user style (like SLD – Styled Layer Descriptor for WMS). The Mapnik renderer, which is used to produce the OSM-tiles, is configurable on server side but not on client side. They only provide server side rendered tiles in discrete zoom levels (the WMS is flexible and has continuous zoom levels). Further, there's no option to get a feature info and there's no option (without requesting the OSM-API) for getting the features themselves.

Nevertheless, sometimes it is required to operate an own online map service (e.g., for personal map styling or personal requirements). In achieving this goal, it is necessary to import VGI data from OSM into a database and to provide access to this database for a map service. Generally, there are two tools, which can be utilized for OSM data import, namely osm2pgsql [11] and OSMOSIS [12]. The former one converts OSM data into a format that can be loaded into a PostgreSQL database and is often used in combination with the map renderer mapnik. However, osm2pgsql does not import all available data, but does some kind of preselection according to keys and values. In contrast, OSMOSIS performs a whole data import, so that every kind of information is available in the database. Hence an import with OSMOSIS is more comprehensive, thus afterwards the application is more adjustable to personal requirements and desires, which makes the application more flexible and adaptable.

Another possibility is the processing of the OSM data with an ordinary programming language. In the past, our OSM-WMS was based on such a processing file, but the rapidly increasing OSM data resulted in long processing times. Actually, the processing of an OSM Europe file took about 14 days on a workstation, which is not acceptable.

The tool OSM-in-a-Box provides an out-of-the-box application set for the automated provision of a free world map [13]. The solution is based on a PostgreSQL/PostGIS database, a Geoserver with additional OGC conform Web-Feature-Service (WFS) and Web-Map-Service on top. For a fast map provision, the solution also contains a GeoWebCache, so that map-tiles are rendered beforehand. This solution allows a fast and easy-to-install out-of-the-box solution, but it is not adjustable to personal requirements and desires.

When investigating related research, it became apparent, that there are little publicly available solutions for a personalized provision of a global online map. On the one hand, the available solutions are too specialized and not generalizable for individual requirements. On the other hand, the processing and computation mechanisms inside the solutions are hidden and not described, so it cannot be said, how the solutions exactly work (e.g., the online map of the OSM project).

To complete, it should also be mentioned that there is of course different work on low level data model formats (like that of OSM, Cf. next Section) available. However, the data format of OSM has been designed with a special focus on VGI purpose, thus scientific models such as [14] or [15] do not suite the requirements of crowdsourced geodata. That is, such models will not be discussed in detail within this paper.

III. DATA SOURCES AND THE DATA STRUCTURE OF OSM

As mentioned above, the developed map service shall be purely based on VGI data. Since OSM is one of the most popular VGI projects, it is assumed that it is also the best suitable data source for this intention.

By mid of September 2011, inside the OSM database there were nearly 1,200,000,000 tagged points, whereby every point describes a distinct geographic location with distinct latitude and longitude values. These points can be furthermore combined into ways (currently nearly 110,000,000), whereby these can be either closed (i.e., an area) or non-closed (i.e., a line). Being able to map complex geometries such as polygons with holes etc., there is furthermore the concept of relations inside OSM. A relation (currently about 1,100,000) is a collection of different ways, nodes or relations, whereby these so called relation members belong together to some extent. Relations can be especially utilized for mapping complex polygonal geometries, whereby one or more outer elements contain several inner elements (e.g., a closed outer ring describing an area with several closed inner rings describing holes in the area). For adding different semantic information on top of those geometries, OSM adapts a concept of open key-value pairs. This concept allows that OSM users can tag their geometries (single nodes, ways or relations) with different key-value pairs, whereby the key describes a distinct information domain or condition and the value describes the corresponding information or information refinement. For example, a way with the key *highway* and the value *residential* generally describes (according to the key) a street for vehicles and/or humans and additionally specifies (according to the value), that this street is a residential street inside a city. Thereby, the amount of key-value pairs for an element is not limited, because additional information can be attached by further key-value pairs (e.g., tagging the before mentioned way with key *maxspeed* and value *30* for

 ISBN: 978-1-61208-178-6

describing that the speed limit of this street segment is 30). Similar to streets, this key-value schema can also be utilized for mapping natural areas such as forests and seas, for mapping a Point-of-Interest (POI) such as an ATM or letter box, or for mapping the outer shape of buildings. The key-value pair concept in OSM is very flexible, because there are no limitations for the keys and values. There are indeed diverse best-practices and recommendations for mapping distinct map features such as the keys *amenity*, *boundary*, *building*, *natural*, *place*, *waterway* (refer to the OSM wiki [16] for more information) with corresponding values, but in general a user is able to add keys and values however liked. A complete list of all currently used keys is available at Tagwatch [17].

IV. SYSTEM ARCHITECTURE AND PROCESSING WORKFLOW

The developed system architecture can be generally characterized as a classic 3-tier architecture [18] as depicted in Fig. 1. At the bottom there is a data tier (blue layer) with different data sources and data types combined in a database. On top of that, there is the processing tier (orange), which processes raw data of the data tier and stores the processed data in a database. On top, there is the presentation tier (green) representing graphical user interface (GUI), which allows a user to consume and interact with the provided data. Since the aim of this work is the development of an online map, the GUI consists of a HTML webpage, which can be accessed all over the world with an ordinary web browser or by utilizing standardized OGC web services (e.g., WMS).

The architecture itself is distributed, meaning that different system components such as the database or the webpage are located on different servers. This procedure allows for the distribution of work load onto different servers, thus is likely to increase the overall performance of the system. In particular, this distribution allows that a server can be configured according to its purpose (e.g., a database server can be equipped with much RAM and fast hard disks). The workflow for the data processing and data supply and the whole computation chain is depicted in Fig. 2.

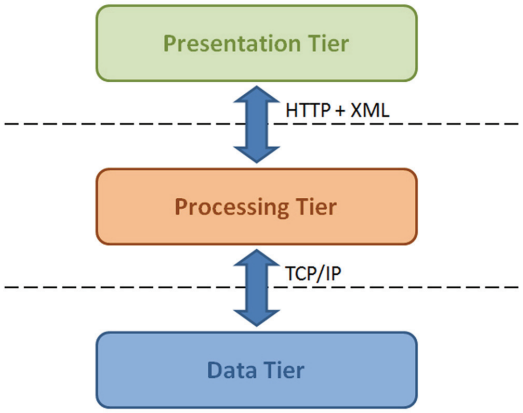

Figure 1. 3-tier architecture.

As described, an initial OSM planetfile (can be retrieved from [19]) import can be performed by using the processing tool OSMOSIS [12], which will be described in more detail in the next chapter.

A. OSMOSIS Import

OSMOSIS is a command line tool written in JAVA, which provides many functions to extract and convert OSM data. The tool is stream based, so there is always an input and an output stream. The tool is able to read almost all different OSM Formats such as XML, compressed XML, pbf etc., and furthermore it is capable to store the extracted OSM data into different data formats and data storage types (e.g., XML, pbf, mySQL, PostreSQL, diff-files, etc.). It also creates the former mentioned data schema that represents the whole OpenStreetMap Data. It is possible to create geometries for points and linestrings while importing the OSM data into a PostgreSQL/PostGIS relational database. Furthermore the tool can import diff-files (files that represent the changes between two OSM data sets). Within the here presented work, the OSMOSIS tool is used to create the database schema (namely called OSMOSIS simple schema) for PostgreSQL/PostGIS. Thereby, OSMOSIS separates the data into different tables such as nodes, ways etc. This allows a relational perspective on the OSM planetfile and with a few simple SQL queries, all data (i.e., key-value pairs or other information) for a distinct node, way or relation, can be retrieved. After the initial import, diverse indices are created on several database tables for increasing the performance of the database queries. The OSMOSIS database can be kept up-to-date by regularly performing diff-file updates. These diff-files are available for minutely changes, hourly changes or daily changes on the same webpage as the planetfile (and some mirror pages) and can be imported into the database by using the OSMOSIS parameter *--read-xml-change*. You can find further information about the tool at the OSM-Wiki pages [20]. Based on these OSMOSIS tables, different database views (i.e., virtual perspectives on the current data sets) have been created. These views were designed in a way, that they contain relevant data for a distinct information domain. For more information on the database views themselves, please refer to the next Section.

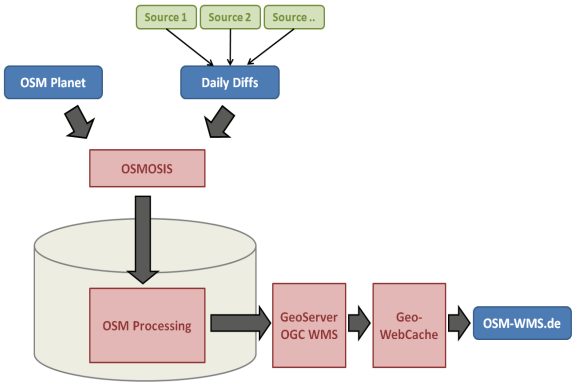

Figure 2. OSM data processing workflow

 ISBN: 978-1-61208-178-6

B. *View Creation*

As stated above, the developed system is based on an OSMOSIS database schema. That is, all VGI data from OSM is included in the database and separated into the tables *nodes*, *ways*, *way_nodes*, *relations*, and *relation_members*. The tags of the different elements are added to the corresponding data tuples by using a so called *hstore* column, which allows the storage of multiple key-value pairs in one single database row.

Depending on what type of element (i.e., what type of geometry) shall be included into the view, the selection of required OSMOSIS tables varies. So e.g., when creating a view, which contains single point-geometries (e.g., for POIs), the database view only considers the *nodes* table. In contrast, when concentrating on simple linestring geometries (e.g., streets or rails), the database view considers the *ways* tables. Since the OSMOSIS import is performed with activated linestring creation (i.e., OSMOSIS creates linestring geometries for all ways while importing the data), the table *way_nodes* is not required at all in the process. For more complex geometries such as the extraction of complex-shaped natural areas (e.g., forests or water areas) it is necessary to join different tables with each other (e.g., the *relation* table with the *way* table). Moreover, some geometries in OSM consist of several non-closed ways, whereby the collection of all of them results in one closed linestring. That is, when trying to create a polygon for such a relation, one must be aware of this issue, thus the corresponding view must be designed accordingly.

Depending on what type of geometry shall be created and what type of information shall be described, there is one or more database views required. For example, for a table containing all water elements of the world map, one would require a view for polygons created via one single closed way, one view for elements with several non-closed ways, and one view for polygons created via several (non-closed) ways whereby the resulting polygon consist of several non-overlapping elements.

The view itself is created via a SQL script, which iterates over a distinct OSMOSIS table (e.g., the *ways* table) and selects different attributes. Also by using *WHERE* conditions, it is possible to filter the results, so that for example only ways with special conditions or constraints are included in the materialized view.

Since database views are virtual, and for accessing them, they first need to be computed, access times on the views are not satisfying. Additionally, it is not possible to create indexes on views. For overcoming this issue, it has been decided to regularly materialize those views into real physical database tables (i.e., once the views are computed, their content is stored in a real database table). In doing so, all views are computed and all data tuples from the views are stored into physical tables. For further improving access times, several indexes for different table columns are created and stored in the database. These materialized tables are the data input source for the Geoserver WMS, which is used by the web-frontend.

However, this approach requires some further methodology for OSM updates, because an OSMOSIS update will not affect the materialized database views. That is, after an OSMOSIS update (i.e., a diff-file import), it is additionally required to redo the view materialization. This is achieved by recomputing the database view and storing the new view data into a physical database table. As long as the view is not computed successfully and all the data is stored in the materialized table, the system still utilizes the old tables. After the completion of the materialization process, the old tables are renamed (for backup possibilities), and the new ones are plugged into the system architecture.

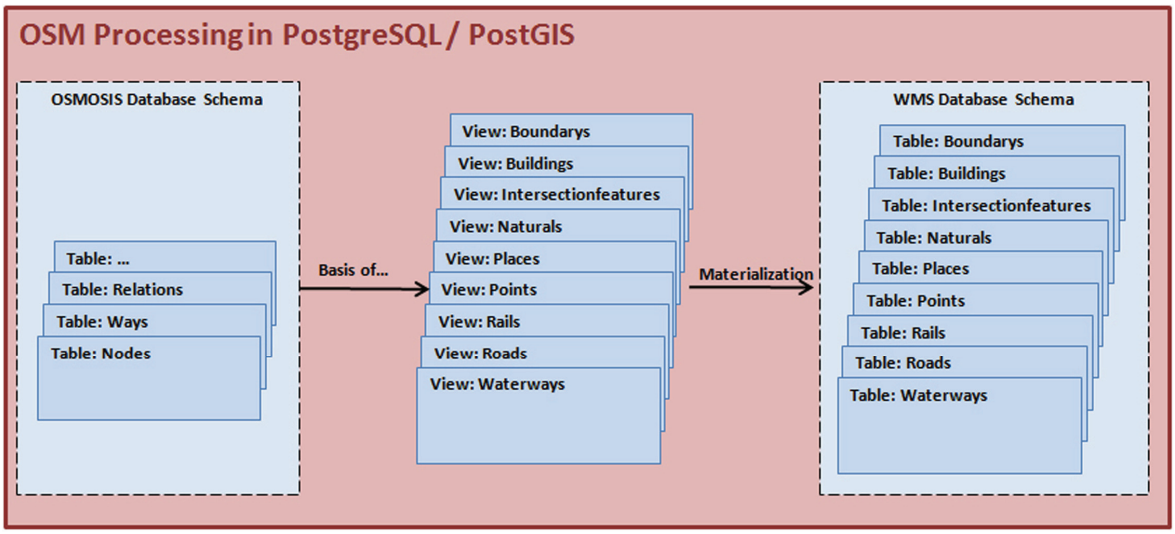

Figure 3. The Workflow of processing raw OSM data for a WMS

 ISBN: 978-1-61208-178-6

C. WMS and GeoWebCache

After importing, processing and storing the data, the created database tables are ready for the rendering part of the visualization pipeline. A very widespread method for visualizing web maps is the use of a rendering software, which on the one hand can interpret geographic data formats and on the other hand provides the results through a standardized web mapping interface. Such a standardized interface for rendering and delivering map images from geodata is described by the well known WMS Standard from the OGC. There are several open source and also commercial software tools, which implement this standard. In our case we are using the open source software Geoserver, which can be connected to the before mentioned database tables. With a standardized WMS Request (including different parameters like Bounding Box, Spatial Reference System etc.) it is possible do draw a map extent on the fly and deliver it to the client. The appearance of the resulting map image can be influenced by appending a user style in OGC SLD/SE or by predefined SLD styles on the server. This on-the-fly rendering of geodata provides a great flexibility with respect to the adaptation of maps to different user requirements, but is related to a high consumption of processing resources. Especially when rendering low zoom levels with many features, the rendering process is very slow. A dataset with such a big amount of features like the OSM dataset cannot be rendered on-the-fly in an acceptable period of time without reduction of the data by generalization or omitting certain feature types. This is particular the case if multiple users request map images from the server at the same time.

To avoid the problem of rendering time in worldwide datasets, it is possible to preprocess the map images in a regular grid of map tiles for several discrete zoom levels (scales). Assuming that the client always requests the map tiles in the same manner (size and origin) it is only necessary to process the tiles once and store the result on the server. That is, the tiles can be reused every time a new request from a user arrives, thus the usage of computation performance can be decreased. Such a tile caching can speeds up the performance of large scale map tiles from minutes to milliseconds. For caching map tiles there are also several open source solutions available like the TileCache from MetaCarta Labs [21] or GeoWebCache [22]. For providing the worldwide OSM map the GeoWebCache is used in a mixed application. The first 12 zoom levels from the whole world on one tile to the scale of approx. 1:200 000 (Manhattan on a tile) is preprocessed resulting in 5.5 million tiles of 256 by 256 pixels using 30 GB of disk space. Tiles in further zoom levels can be processed in an acceptable time so it isn't necessary to preprocess all the tiles. Nevertheless, since it is unlikely that all parts of the world (especially the sea) are of interest in every zoom level for the user, it has been decided, that all further tiles will be rendered on-the-fly on the first request. For all further requests for these distinct tiles, they will also be stored in the cache.

This demand driven approach of tile rendering saves a lot of storing and processing resources as each zoom level has 4 times the amount of tiles than the one before.

V. APPLICATION OF THE WORKFLOW FOR A GLOBAL DATASET

A. Hardware Resources

The developed framework is applied on a global dataset of OSM. In the real system architecture there are two different servers: one is a processing and database server, which is responsible for the processing of OSM VGI data and for the data management. It is a dedicated server with 35 GB RAM and an eight double-core processor with 2.9 GHz (each processor). There are four 400GB hard disc drives in a virtual RAID 0 assemblage for the database itself and an additional 500 GB hard disc for the operating system (OS). The other server contains an open-to-the-public Geoserver and a webpage representing the GUI for the user. When requesting different map data via either the web-frontend or the standardized OGC WMS interface, the server connects to the database server and retrieves the relevant data tuples (often with the help of a spatial database index). The WMS server is also a dedicated one with 12 GB RAM and a four double-core processor with 2.5 GHz (each processor). It has three hard discs, one of them with 1 TB and two with 512 GB each.

B. Issues on Data Transformation

The database server is set up with an initial OSMOSIS planet import. The WMS database contains 13 tables, whereby nine of them (namely *boundaries, buildings, intersectionfeatures, naturals, places, points, rails, roads, waterways*) are updated regularly. The other ones (o*sm_coastlines*, *sea_all*, *world*, *world_ocean*) contain data that hardly changes, thus we decided that a regular update for them is not required. The process of computing the raw OSM data (gathered from OSMOSIS) for the WMS is depicted in Fig. 3. For creating the nine updated tables, 13 database views have been created – Table 1 contains an overview of them. The column view describes the view name, the column table describes, which WMS table the view is for, the column OSM type describes whether the source of the corresponding geometry results from a node, a way or a relation, and the column geometry type describes how the geometry is created (e.g., by collecting several non-closed ways).

The SQL scripts for the different views are simple SQL scripts containing nested SELECT-statements, CASE-WHEN statements (these are used for deciding between different relevant OSM keys) and WHERE-statements. The geometries are created via SQL by using different PostGIS functions such as *ST_MakePolygon*. Fig. 4 depicts a quite simple SQL script for the database view *intersectionfeatures_v*. Intersectionfeatures are kind of crossing-points in the street-network, thus the geometry is a single point. Therefore, the SQL script iterates the nodes table and only selects these rows with relevant OSM key-value pairs (e.g., *highway = mini_roundabout*).

 ISBN: 978-1-61208-178-6

TABLE 1. DATABASE VIEWS WITH CORRESPONDING MATERIALIZED WMS TABLES AND OSMOSIS SIMPLE SCHEMA SOURCE TABLES

View	WMS table	OSMOSIS source tables	Geometry type
boundarys_oneout	boundaries	ways, relations	one closed linestring or relation with a closed linestring as outer way
boundarys_sevout	boundaries	relations	one relation with several distributed closed linestring
boundarys_sevout_nc	boundaries	relations	one relation with several non-closed linestrings
buildings_v	buildings	ways, relations	one closed way or relation with one outer closed way
intersectionfeatures_v	intersectionfeatures	nodes	one point geometry
naturals_oneout	naturals	ways, relations	one closed linestring or relation with a closed linestring as outer way
naturals_sevout	naturals	relations	one relation with several distributed closed linestring
naturals_sevout_nc	naturals	relations	one relation with several non-closed linestrings
places_v	places	nodes	one point
points_v	points	nodes	one point
rails_v	rails	ways	one way
roads_v	roads	ways	one way
waterways_v	waterways	ways	one way

```
CREATE OR REPLACE VIEW intersectionfeatures_v AS

SELECT
nodes.id AS osm_id,
(
   CASE WHEN defined(nodes.tags, 'junction') THEN nodes.tags -> 'junction'
   ELSE nodes.tags -> 'highway'
END
) AS type,
geom AS the_geom

FROM nodes

WHERE
defined(nodes.tags, 'junction')
   OR
   (nodes.tags -> 'highway' IN ('mini_roundabout', 'stop', 'give_way', 'traffic_signals', 'crossing', 'roundabout', 'motorway_junction'))
;
```

Figure 4. SQL view definition for *intersectionfeatures_v*

The SQL script for the view *buildings_v* (Cf. Fig. 5) is a bit more complicated. Buildings can be either modeled via a single closed linestring or (in a more complex way) as a relation with one outer closed linestring and several inner closed linestrings (representing holes in the building). Generally, the SQL script iterates the *ways* table and only selects those tuples with a valid building tag in the corresponding *hstore* column (i.e., the column tags must contain a key *building* with any kind of value). Additionally, the SQL script also checks whether the way has enough points (it is not possible to create a polygon with a line with less than 4 points, start and end-point have to equal). Also the area of the polygon is calculated, because buildings with an area of zero shall not be included in the database (i.e., *ST_Area(ST_MakePolygon(linestring)) != 0*). For those tuples that pass the WHERE-statement (i.e., which are buildings), several key-value pairs (e.g., height, building:roof etc.) are collected and stored in view columns. For the geometry it needs to be figured out, whether the current way is an outer member of a relation, because in that case the polygon must be created by cutting out the inner holes of the polygon (this is realized with the CASE-WHEN-statement). If the current way is not a member of a relation, the polygon is created by simply using *ST_MakePolygon*. The SQL view definitions are similar to those described beforehand.

The geometries that are mapped with several non-closed ways (i.e., *boundarys_sevout_nc* and *naturals_sevout_nc*) are kind of a special case. Before the polygon can be created, it needs to be figured out, whether the collection of all those

 ISBN: 978-1-61208-178-6

linestrings leads to one closed linestring. This is realized with the PostGIS functions *ST_Collect* and *ST_Linemerge*. These functions form a polygon by sewing together several linestrings. If it is absolutely not possible to form one single linestring, the methods return a multi-linestring. However the SQL-script cannot deal with multi-linestrings, thus relations that lead to a multi-linestring are skipped during processing. If *ST_Collect* and *ST_Linemerge* result in one single linestring, a polygon can be created by using *ST_MakePolygon*. When creating a polygon with *ST_Collect* and *ST_Linemerge*, one strange effect has been discovered. For some geometries, *ST_Linemerge* did not result in a single linestring, although all linestrings shall be connected with each other (as could be seen when viewing the geometry in the well-know-text format WKT with the Method *AsText*). This issue has been solved by transforming the multi-linestring (resulting from *ST_Linemerge*) into WKT, then retransforming the WKT into a geometry (with *ST_GeomFromText*) and then performing again the *ST_Linemerge* method. This work-around was necessary because of a probable PostgreSQL rounding bug. It seems that the converting function (geometry to WKT) cuts the decimals of the coordinates, so that after the conversion the point matching works better. Additionally, this effect might also be caused by slight and minor inaccuracies within OSM. However, with this chain of several PostGIS functions it is possible to overcome the described problem.

```
CREATE OR REPLACE VIEW buildings_v AS

SELECT
ways.id AS osm_id,
(
   CASE WHEN defined(ways.tags,'name') THEN (ways.tags -> 'name')
   WHEN defined(ways.tags,'name:de') THEN (ways.tags -> 'name:de')
   WHEN defined(ways.tags,'ref') THEN (ways.tags -> 'ref')
   ELSE NULL
END
) AS name,
(
   CASE WHEN defined(ways.tags,'building:type') THEN (ways.tags -> 'building:type')
   WHEN defined(ways.tags,'building:use') THEN (ways.tags -> 'building:use')
   ELSE NULL
END
) AS type,
(
   CASE WHEN defined(ways.tags,'levels') THEN (ways.tags -> 'levels')
   ELSE NULL
END
) AS levels,
(
   CASE WHEN defined(ways.tags,'building:levels') THEN (ways.tags -> 'building:levels')
   ELSE NULL
END
) AS "building:levels",
...
(
   CASE WHEN EXISTS (SELECT 1 FROM relation_members WHERE member_id = ways.id AND LOWER(TRIM(member_role)) = 'outer')
   THEN
      (ST_MakePolygon(
         (linestring),
         ARRAY(
            SELECT linestring FROM ways AS w2
            WHERE w2.id IN (
               SELECT member_id
               FROM relation_members
               WHERE
                  LOWER(TRIM(member_role)) = 'inner' AND
                  member_type = 'W' AND
                  relation_id IN (SELECT relation_id FROM relation_members WHERE member_id = ways.id AND member_role = 'outer')) AND
                  ST_IsClosed(linestring) AND ST_NumPoints(linestring) > 3
         )
      ))
   ELSE ST_MakePolygon(linestring)
END
) AS the_geom

FROM
   ways

WHERE
   defined(ways.tags, 'building')
   AND
   ST_IsClosed(linestring)
   AND
   ST_NumPoints(linestring) > 3
   AND
   ST_Area(ST_MakePolygon(linestring)) != 0
;
```

Figure 5. Shortened SQL view definition of *buildings_v*

 ISBN: 978-1-61208-178-6

C. Results

As a result of the described workflow, an amount of 24 GB is processed and distributed in nine WMS optimized database tables (Cf. Table 2).

An initial import of OpenStreetMap data takes about 8 hours. The daily OSM update (about 50 MB in a zip file) for the database (including the download) requires less than 3 hours. The amount of time required for an update is that high, because of the required search operations and linestring computations. Especially the search operations, which are not required for an initial import, have high computation costs. The processing inside the database (i.e., the table creation, view materialization, index creation) requires 20 hours. The subsequent preprocessing of the tiles for the first 12 zoom levels by the WMS for storage into the GeoWebCache takes about 6 hours. That is, in total a ready-to-run startup system (initial OSM import, database processing and tile preprocessing) can be realized in about one and a half day.

TABLE 2. QUANTITATIVE FIGURES OF DIFFERENT DATABASE TABLES (DATA FROM 2011-09-11)

Table	Number of tuples	Table size	Index size
boundaries	612,114	293 MB	61 MB
buildings	42,187,016	8,683 MB	3,935 MB
intersectionfeatures	740,166	51 MB	78 MB
naturals	9,843,763	3,833 MB	942 MB
places	2,245,932	168 MB	302 MB
points	13,563,055	976 MB	1,287 MB
rails	833,872	199 MB	82 MB
roads	44,178,883	10,487 MB	4,394 MB
waterways	5,215,237	2,005 MB	478 MB
Total	119,419,988	~26 GB	~11 GB

For keeping the system up-to-date, several methodologies have been invented to drive an automated update. The database server downloads every day the daily-diff files of OSM and imports them into the database. Once a week (at the weekend) the database processing is performed, resulting in new and updated WMS tables. It has been decided that a weekly update of the WMS tables is enough, although a daily update could be feasible (OSM update plus processing would require about 23 hours). Additionally, once in a week (after the database processing), new tiles (with new features) for the GeoWebCache are generated.

For demonstrating the scalability of the system architecture, different performance analyses for the cached WMS data have been conducted. A varying amount of concurrent threads simulate the concurrent usage of the WMS by different users. Thereby, the amount of concurrent threads can be interpreted in two different ways: on the one hand, for example 100 concurrent threads can be considered as one user requesting 100 tiles from the cached WMS. On the other hand, it can also be regarded as for example 10 users, whereby each of them concurrently requests 10 tiles. The analyses were conducted in the range between one concurrent thread and 1.000 concurrent threads, which ought to be realistic numbers for the operative system's usage. For the tiles it has been decided to utilize a medium zoom level and a resolution of 256 * 256 pixels, resulting in a filesize of 7kb for each individual tile. The results of the analyses are depicted in Fig. 7. Fig. 7 (a) depicts the average response time of the WMS in *ms* (the red, upper line), as well as the error rate (the blue, lower line), i.e. the percentage of WMS requests which could not be processed due to timeout issues (in the current system, a timeout occurs after 20 seconds processing time for an individual thread). Fig. 7 (b) visualizes the maximum response time in *ms* (the read, upper line), as well as the deviance of the response times of all requests in *ms* (the blue, lower line). These two diagrams show, that the current system is able to handle up to 500 concurrent threads with acceptable response times. For more than 500 concurrent threads, some requests receive a timeout (about *3.5%* timeout rate for 1.000 concurrent threads). With increasing amount of concurrent threads, also the maximum response time increases, converging an upper bound of about *12.700 ms*.

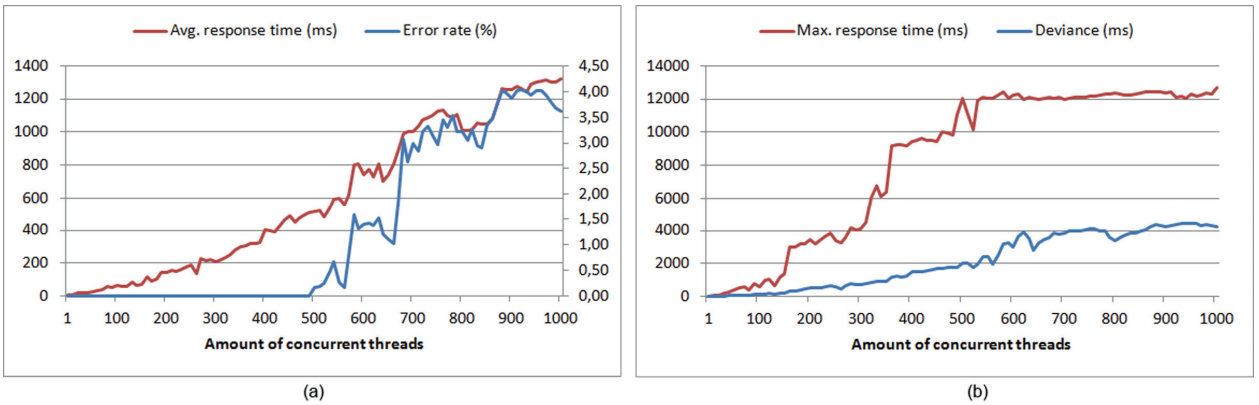

Figure 6. Average response times in *ms* and error rate in *percent* (a), deviance in *ms* and maximum response times ins *ms* (b) for the cached WMS with varying amount of concurrent threads

VI. CONCLUSIONS AND FUTURE WORK

Crowdsourced geoinformation from web community platforms can serve as a powerful and huge data source for online map services. Following a background literature review and investigation of existing web map services, an introduction to OpenStreetMap as one of the most popular examples of crowdsourced geodata is given. Afterwards a workflow for the automated processing of OSM data for the provision of a global online map has been described. See Fig. 7 for an exemplary excerpt of the map, showing the city of Heidelberg. The workflow has been textually described and a proof of concept was given. Additionally, quantitative figures about the current database, as well as qualitative performance analyses were provided.

Figure 7. WMS based OSM map with adapted User Style

As a future step, the different views of the processing chain could be refined and optimized, so that performance can be further increased. Additionally, by adding further OSM key-value pairs, other different types of map objects can be integrated into the online map. Although not directly connected with the work described in this paper, the improvement of the data quality inside OSM is also an important issue. More proper and qualitative data in OSM is likely to have a positive effect on the OSM-WMS itself, because the higher the quality and correctness of the data is, the more competitive is the OSM-WMS compared to commercial map providers such as Google Maps.

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers for providing valuable comments towards the improvement of this paper. Furthermore we would also like to thank all members of the Chair of GIScience for their proofreading and helpful hints. This research has been partially funded by the Klaus-Tschira Foundation (KTS) Heidelberg.

REFERENCES

[1] Goodchild, M.F. 2007. Citizens as sensors: the world of volunteered geography. GeoJournal, 69(4): p. 211-221.

[2] Goodchild, M.F. 2007. Citizens as Voluntary Sensors: Spatial Data Infrastructure in the World of Web 2.0. International Journal of Spatial Data Infrastructures Research, 2: p. 24-32.

[3] OSM. 2011. Stats - OpenStreetMap Wiki. http://wiki.openstreetmap.org/wiki/Statistics. (Accessed 09/11/2011).

[4] Zielstra, D. and A. Zipf. 2010. A Comparative Study of Proprietary Geodata and Volunteered Geographic Information for Germany. AGILE 2010. The 13th AGILE International Conference on Geographic Information Science: Guimarães, Portugal.

[5] Haklay, M. 2010. How good is volunteered geographical information? A comparative study of OpenStreetMap and Ordnance Survey datasets. Environment and Planning B-Planning & Design, 37(4): p. 682-703.

[6] Neis, P., D. Zielstra, A. Zipf, and A. Strunck. 2010. Empirische Untersuchungen zur Datenqualität von OpenStreetMap - Erfahrungen aus zwei Jahren Betrieb mehrerer OSM-Online-Dienste, AGIT 2010 Symposium für Angewandte Geoinformatik.: Salzburg. Austria.

[7] Ludwig, I., A. Voss, and M. Krause-Traudes. 2010. Wie gut ist OpenStreeetMap? Zur Methodik eines automatisierten objektbasierten Vergleichs der Straßennetze von OSM und Navteq in Deutschland. GIS.Science, 4: p. 148-158.

[8] Google. 2011. Google Maps. http://maps.google.de/ (Accessed: 11/14/2011)

[9] Bing. 2011. Bing Maps - Driving Directions, Traffic and Road Conditions. http://www.bing.com/maps/ (Accessed: 11/14/2011)

[10] OSM. 2011. OpenStreetMap. http://www.openstreetmap.org. (Accessed: 11/14/2011)

[11] Osm2pgsql. 2011. Osm2pgsql. http://wiki.openstreetmap.org/wiki/Osm2pgsql. (Accessed: 09/11/2011).

[12] Osmosis. 2011. Osmosis - OpenStreetMap Wiki. http://wiki.openstreetmap.org/wiki/Osmosis (Accessed: 09/11/2011).

[13] Hof, R., M. Huber, F. Renggli, and S. Keller. 2009. OpenStreetMap-in-a-Box - Geo-webservices for the free world map, Computer Science. HSR: Rapperswil, Suisse.

[14] Morehouse, S. 1992. The ARC/INFO geographic information system. Computers&Geosciences, 18(4): p. 435-441

[15] Hoel, E.G., Menon, S., and Morehouse, S. 2003. Building a Robust Relational Implementation of Topology. SSTD, 2750: p. 508-524

[16] OSM. 2011. OpenStreetMapWiki. (Accessed: 09/11/2011).

[17] Tagwatch. 2011. Tagwatch Planet-latest. http://tagwatch.stoecker.eu/Planet-latest/En/tags.html. (Accessed: 09/11/2011)

[18] Eckerson, W.W. 1995. Three Tier Client/Server Architecture: Achieving Scalability, Performance, and Efficiency in Client Server Applications. Open Information Systems, 10(3): p. 1-20.

[19] OSM. 2011. Planet OSM. http://planet.openstreetmap.org/ (Accessed 11/14/2011)

[20] OSMOSIS. 2011. OSMOSIS - OpenStreetMap Wiki. http://wiki.openstreetmap.org/wiki/Osmosis. (Accessed 09/11/2011).

[21] MetaCarta. 2011. Geographic Search and Reference Solutions - Meta Carta - At the Forefront of the GeoWeb. http://www.metacarta.com/. (Accessed 06/30/2011).

[22] GeoWebCache. 2011. GeoWebCache. http://geowebcache.org/. (Accessed 09/11/2011)

 ISBN: 978-1-61208-178-6