# OpenStreetMap over WMS

**Přemysl Vohnout, Jáchym Čepický**
Czech Centrum for Science and Society, Help Service Remote Sensing
vohnout@ccss.cz, jachym@bnhelp.cz

**Keywords:** WMS, OpenStreetMap, Mapserver

## Abstract

*This paper discuss the issues which we faced, while preparing WMS server with OpenStreetMap data of whole Europe. This article is divided into three sections. First is about mandatory applications which are required for working WMS service with OpenStreetMap data. Second is focused on tuning up PostgreSQL. Third is focused on rendering time improvement of layers.*

## Prerequisites

There is need of data first of all. Data can be acquired from several repositories where are made live dumps from OSM databases. There is possibility to choose data by geographic and country distribution. You can download for example shapefiles, country borders or "osm dump". This data can be obtained from http://downloads.cloudmade.com or http://geofabrik.de, ... We used data from geofabrik.de.

We are using MapServer for rendering data from PostgreSQL to WMS. There are some requirements which are required to keep for best result of maps.

- MapServer of version 5.4 and higher. This is required due to change of labels rendering machine.
- Application osm2pgsql. This is a little bit still development application.
- Postgresql of version 8.0 and higher with PostGIS. I recommend 8.3 with PostGIS 1.3.5 and higher. There was some bug in 1.3.3 which was returning bad bounding box.
- Optional mapserver-utils – this project contains mapfiles for advanced drawing of layers from OpenStreetMap. You needn't to use this mapfiles but it's recommended. This can be obtained from http://code.google.com/p/mapserver-utils.

Because MapServer 5.4 and PostGIS 1.3.5 is very recent user will probably don't find it in repository of almost every distribution of Linux if you are not using some kind of rolling-updates distribution (like Gentoo). Here is small howto create your own deb packages in debian. First of all you will need pbuilder application. This application will make "fake" image of system. After small tweaking of config file you can start making debian packages.

```
aptitude install pbuilder
pbuilder create --distribution lenny
```

(lenny is last branch of debian, etch is previous)

```
EDITOR /etc/pbuilderrc
```

and put there something like this:

```
MIRRORSITE=http://ftp.cz.debian.org/debian/
OTHERMIRROR="deb http://www.backports.org/debian lenny-backports main | \
 deb file:///var/cache/pbuilder/result ./"
HOOKDIR=/etc/pbuilder/hook.d
BINDMOUNTS="/var/cache/pbuilder/result /home /usr/src"
EXTRAPACKAGES="debian-backports-keyring proj=4.6.1-5~bpo50+1 proj-bin libproj0 libproj-dev"
```

- Download package control files from `svn://svn.debian.org/svn/pkg-grass/packages/<package>/trunk` somewhere where you can find them easily (for example /usr/local/src).
- Download tarballs to directory from previous point.
- Last step is to create package by

```
pdebuild --use-pdebuild-internal
```

If it is everything ok you will have package ready to install.

### PostgreSQL part

PostgreSQL is configured for secure and reliable handling of data by default, but if you want to handle big amounts of data it's better to do some tweaks. You should think about your hardware in the beginning for better performance of your database management system. Memory is faster than HDD but it is not possible to buy lot of memories because there is limited number of memory slots. SSD disks are slower but their size is still not so big (max 128GB) and they are more expensive. Most used scenario is disks connected into RAID. RAID1 at least but I prefer RAID5. RAID5 is slower when writing data to disks but faster when reading data from disks.

Software tweaking of PostgreSQL can be done in config file which you can find in CLUSTER-DIR/postgresql.conf.

- **shared_buffers** – this parameter defines amount of memory in which PostgreSQL will hold requests. This should be set to thousands (one buffer is 8kB). Some sources writes one quarter of memory another writes set this to 256MB (32768).
- **maintenance_work_mem** – specifies amount of memory which is used for maintenance operations like vacuum, create index ... This should be set to bigger amount than default. Very usable if you are using autovacuum operation.
- **effective_cache_size** – this parameter has impact on planning operation of PostgreSQL. Bigger amount will trigger using index scans, lower will use sequential scans.
- **checkpoint_segments** – If you will see this in your log "checkpoints are occurring too frequently (29 seconds apart)" set this parameter bigger then 3.

### Getting data to PostgreSQL

Data from osm format are put into PostgreSQL by osm2pgsql software. Before user can use this software he must have spatial enabled database and must have existing Mercator projection (or "Google projection") in /usr/share/proj/epsg and spatial_ref_sys table. This projection was added to recent version of EPSG.

Creating spatial enabled database:

```
createdb osm
createlang plpgsql osm
psql -f /usr/share/postgresql-8.3-postgis/lwpostgis.sql osm
psql -f /usr/share/postgresql-8.3-postgis/spatial_ref_sys.sql osm
```

After this log into PostgreSQL using osm database and add 'google projection' into spatial_ref_sys table:

```
psql -d osm

INSERT INTO spatial_ref_sys (srid, auth_name, auth_srid, srtext, proj4text)
VALUES (900913, 'spatialreference.org', 900913,
 'PROJCS["unnamed",GEOGCS["unnamed ellipse",DATUM["unknown",
  SPHEROID["unnamed",6378137,0]],PRIMEM["Greenwich",0],
  UNIT["degree",0.0174532925199433]],PROJECTION["Mercator_2SP"],
  PARAMETER["standard_parallel_1",0],PARAMETER["central_meridian",0],
  PARAMETER["false_easting",0],PARAMETER["false_northing",0],
  UNIT["Meter",1],
  EXTENSION["PROJ4","+proj=merc +a=6378137 +b=6378137 +lat_ts=0.0
  +lon_0=0.0 +x_0=0.0 +y_0=0 +k=1.0 +units=m +nadgrids=@null +wktext  +no_defs"]]',
  '+proj=merc +a=6378137 +b=6378137 +lat_ts=0.0 +lon_0=0.0 +x_0=0.0 +y_0=0
  +k=1.0 +units=m +nadgrids=@null +wktext  +no_defs');
```

And last step before inserting data into database is to create 'google projection' in /usr/share/proj/epsg. Add this line

```
<900913> +proj=merc +a=6378137 +b=6378137 +lat_ts=0.0 +lon_0=0.0 +x_0=0.0 +y_0=0 +k=1.0 +units=m
+nadgrids=@null +no_defs
```

For faster queries it's recommended to put projections which are used in the beginning of the epsg file. It's possible to start importing data into database now. This is done with osm2pgsql application which have some parameters. *-d* will specify database, *-p* – prefix of created tables, *-C* define amount of space for caching. If you run out of memory try turning on *-s*

```
osm2pgsl -d osm -p osm -C 2048 <file_with_osm_data>
```

At [1] is written some after-importing operations which makes better output from database.

### Speeding up rendering time

Mapfiles from *mapserver-utils* are nice but they are not usable for big amount of data like whole Europe. If you don't have some very powerful grid and want to see map before your coffee get cold, you must do some optimizations for small scales (from $\infty$ to $1:500000$). For example osm_line table has 9.5M records and in scale 1:5 000 000 are shown these layers:

coastline, borders, cities with more than 100k citizens and highways (motorways). If you are doing select of highways from osm_line after using spatial index there are 5.8M of records left and after apply of where clause (highway='motorway') there are only 50k of records so biggest optimization for small optimization is to do materialized view. Materialized views are views which are created physically on disks and act like real tables. Normal views are only "links" to tables so if user is doing some queries on normal views than the query is done on all data in those tables. Example will be done for motorways:

```
DROP TABLE IF EXISTS osm_highway_motorways;
DELETE FROM geometry_columns WHERE f_table_name='osm_highway_motorways';
CREATE table osm_highway_motorways AS select way, osm_id ,highway,ref, name, z_order from osm_line
 where highway = 'motorway' order by z_order;

CREATE INDEX osm_highway_motorways_index
 ON osm_highway_motorways
 USING gist
 (way);

CREATE INDEX osm_highway_motorways_pkey
 ON osm_highway_motorways
 USING btree
 (osm_id);

INSERT INTO geometry_columns (f_table_catalog, f_table_schema, f_table_name, f_geometry_column,
 coord_dimension, srid, "type") VALUES (' ', 'public', 'osm_highway_motorways', 'way', 2, 900913,
 'LINESTRING');

GRANT SELECT on osm_highway_motorways to "www-data";
```

First two lines are used for clearing things done before. Third line is creating materialized view. 4th and 5th (if you are calculating only semi-colons) line are creating indexes. Indexes are good that it makes tree structure upon data and this make searching through data faster. User has 3 possibilities in PostGIS with indexes. B-Tree, R-Tree and GiST. B-tree is only for one dimension data. R-Tree is only for multidimensional data. GiST is combining these two indexes in one. This is reason why this index is used in most cases. 6th line inserts geometry into geometry_columns table. Insert geometries into geometry_columns table is needed because some GIS clients is looking for geometries only into this table. Last line grants select privileges to user which is used in mapfile to access database.

Next possibility of optimization is to use CLUSTER command. This command will sort data in table by selected index. This is very useful with use of mapserver because it is doing spatial query first.

Another optimization can be done on client side. OpenLayers allows to make tiles. This mean that it cuts map window into small parts which are faster to render. This can be set by *SingleTile: false*. This parameter will cut whole map area into several tiles (depends on size of tile) and makes query for every tile separately. But be careful about partial labels so setup gutter in OpenLayers. Gutter will expand tile for specified amount of pixels and it will draw labels behind borders of tile.

### External resources

1. Rendering OpenStreetMap Data with MapServer, online
   http://trac.osgeo.org/mapserver/wiki/RenderingOsmData