Custom OpenStreetMap Rendering – OpenTrackMap Experience

Radek Bartoň
Department of Computer Graphics and Multimedia
Faculty of Information Technology, Brno University of Technology
`ibarton fit.vutbr.cz`

Keywords: OpenStreetMap, mapnik, osm2pgsql, PosgreSQL, mapping, rendering, hiking, hiking tracks

Abstract

After 5 years of its existence, the OpenSteetMap [1] is becoming to be an important and valuable source of a geographic data for all people on the world. Although initially targeted to provide a map of cities for routing services, it can be exploited to other and often unexpected purposes. Such an utilization is an effort to map a network of hiking tracks of the Czech Tourist Club [2].

To support and apply this endeavour, the OpenTrackMap [3] project was started. Its aim is to primarily provide a customized rendering style for Mapnik renderer which emphasizes map features important to tourists and displays a layer with hiking tracks. This article presents obstacles which such project must face and it can be used as a tutorial for other projects of similar type.

Motivation and Background

Small portable devices equipped with GPS chips and high bandwidth Internet connection are more frequently becoming our attendants on trips to nature. Although they are sometimes more powerful than personal computers used in near past, they have not enough processing power to interactively render vector geographic data of larger areas with good visual quality.

For this reason Web-based raster geographic data services such as TMS [4], WMS [5,6] or Slippy Map [7] combined with geocoding [8] and routing services are more suitable for this platform. Rendering requests are sent to a server and rendered tiles are then transferred back to a client as images using HTTP protocol. This transfers heavy load of data rendering to distant servers.

Although servers like [9] provides hiking maps of Czech Republic, they are almost impossible to use on the portable devices with an open source software due to licensing or technical restrictions. In the field of the open source, the OpenSteetMap is only available source of geographic data of the entire planet. On the other hand, portion of already mapped tracks of CTC hiking tracks network in the OpenStreetMap is only partial. This implies need of publicly accessible and regularly updated service which displays hiking tracks from the OpenSteetMap for Czech Republic. It should provide fast enough feedback for mappers at home and it could also be used in a terrain.

First project to develop fully open source portable device with GPS capabilities is Openmoko [10]. So a primary target of the OpenTrackMap project is to provide maps for a TangoGPS application on the Openmoko platform.

Similar Projects

Before getting down to start developing the OpenTrackMap, proper research of existing projects with same scope was made. Most similar of them is an openstreetmap.cz [11] with its display of hill shading and Czech hiking tracks layer. Unfortunately, it is not regularly updated a it does not provide a baked layer with all map features together. Moreover, its rendering style is not high quality.

Projects like a Freemap Slovakia [12] or an OSMC Reit- und Wanderkarte [13] are more mature but they covers different countries while projects [14,15,16,17] are also customized OpenStreetMap renderings but aimed to different audience.

Requested Features

With respect to objectives of the project, following list with requested features can be written up:

- Display hiking tracks of the CTC network with their color, that is use red, green, blue and yellow lines for them. Properly display two to four parallel tracks using dashed lines of appropriate colors.
- Distinguish between regular, local, learning, skiing and horse tracks using shields with established symbols [18,19,20].
- Overlay a layer with contour lines generated from publicly available elevation data.
- Add a layer with hill shading also extracted from elevation data.
- Display additional point objects attractive for hikers (i. e. guideposts, castles, peaks, pubs, etc.) with appropriate icons that are not included in a default OpenStreetMap rendering style.
- Provide both single TMS service for each layer and a complete TMS service with all layers baked into a single layer.
- Setup automatic data synchronization from a main OpenStreetMap data repository with reasonable update interval and delay.

Available Renderes

From the list of known renderers of OpenSteetMap data, only three are more interesting.

Mapnik

The Mapnik [21] is the most features and also the most used renderer available. Basically, it is a C++ library with Python bindings generator accompanied with rendering scripts using the library. Appearance of the rendered map is specified in custom XML style document [22] but a CSS like styling syntax can be also used and then compiled to the default XML style with a Cascadenik tool [23]. There is also a simple GUI application for Mapnik styles preview called a Mapnik Viewer [24]. On-demand tile rendering is possible with mod_tile [25] or TileLite [26].

Possible input vector formats of the Mapnik are following:

- ESRI shapefile.
- OSM XML format [27].
- PostGIS database.
- Oracle Spatial database.
- SQLite database.
- And many others through OGR library [28] like GRASS vector format, GPX or KML.

Input raster formats supported are:

- TIFF image.
- And many others through GDAL library [29] like GRASS raster format, GeoTIFF or JPEG2000.

It is optional to choose between two rendering libraries [30]: Anti-Grain Geometry and Cairo. This implies possible output formats:

- PNG raster image (AGG, Cairo).
- JPEG raster image (AGG).
- SVG raster image (Cairo).
- PDF document (Cairo).
- PostScript document (Cairo).

Osmarender

Osmarender [31] is all in all an XSLT template, an XML styling document and a Perl script that converts the OSM XML format to a SVG vector image. It is used for distributed rendering in a Tiles@home [32] project, but it has not so much features like the Mapnik and for the OpenTrackMap is quite insufficient regarding to the requested features list.

Kosmos

This software [33] is just simple GUI to style and render the OpenStreetMap data on a desktop. It is written in C# for Windows and thus it needs .NET Framework to run and it is not much compatible with Mono implementation of it. This means that the Mapnik is only option for renderer for the OpenTrackMap.

Source of Elevation Data

To display contour lines and hill shading layer in the map, elevation height map data are needed. Well-known and free source of elevation data for nearly whole world is lidar data obtained from NASA's SRTM project [34]. Resolution of its grid is 3 arc seconds, which is approximately 90 meters in the area of Czech Republic, while it is claimed that their vertical accuracy is 10 meters. There are many servers run by U. S. government organizations providing this dataset. They differ in quality of a data post-processing, that is gap filling of missing data. For the OpenTrackMap project, just USGS [35] and GLCF [36] data was tried.

In the time of the project creation a new source of elevation data was introduced. It is an ASTER GDEM [37] stereoscopic data from joint project of the NASA and the Japanese Ministry of Economy, Trade, and Industry (METI). Its resolution is 1 arc second and expected vertical accuracy is 7 to 14 meters. Although it is not yet clear whether this data can be used in open source projects like the OpenTrackMap is, some experiments with the data was made to prove that they can be useful.

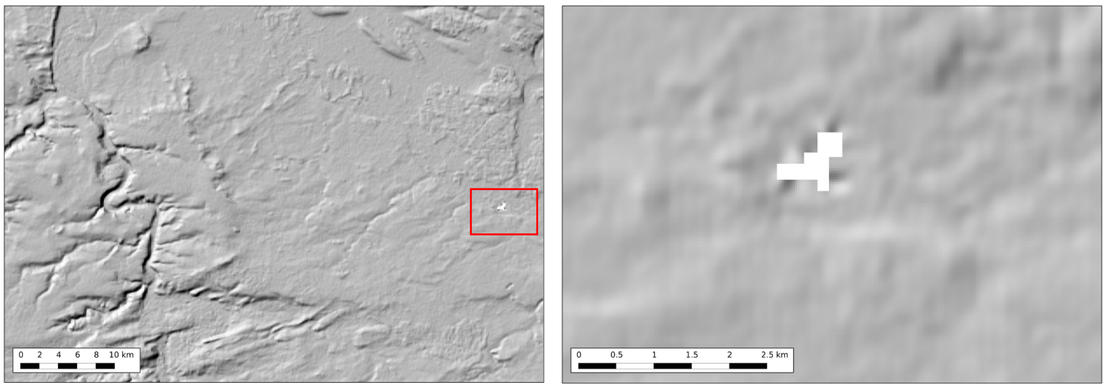

Figure 1: Hill shading map of central Bohemia near capital city Prague generated using the USGS SRTM data in WGS84 projection. The red rectangular area in the left part of the figure is an extent of the cut-out right part. The completely white area is caused by missing data.

The figures 1. to 3. show examples of the hill shading generated from different elevation maps. Although the USGS (figure 1.) and the GLCF SRTM (figure 2.) data seem pretty much the same, the ASTER GDEM (figure 3.) data contains disturbing artifacts of artificial edges of a splotch shape. This is probably caused by the fact that the ASTER GDEM data was filled in with the SRTM data in areas of missing data occluded by clouds and that there may be some constant value offset between those sources. There are also some missing data

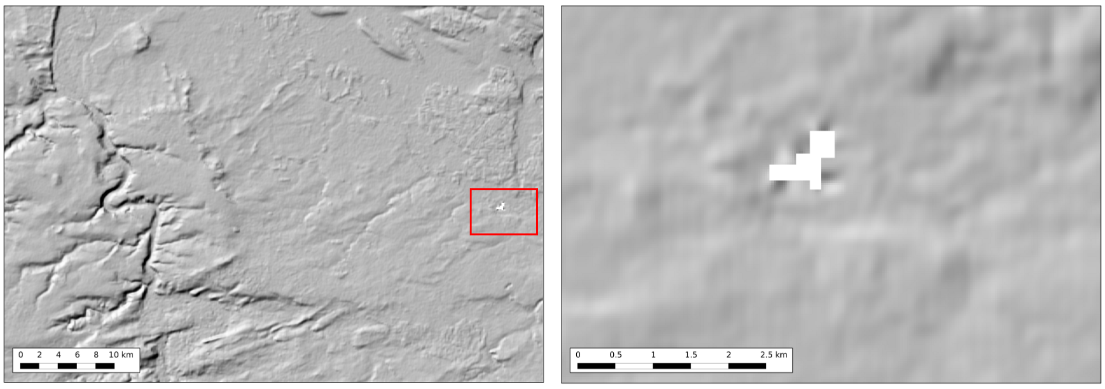

Figure 2: Hill shading map of central Bohemia near capital city Prague generated using the GLCF SRTM data in WGS84 projection. The red rectangular area in the left part of the figure is an extent of the cut-out right part. The completely white area is caused by missing data.

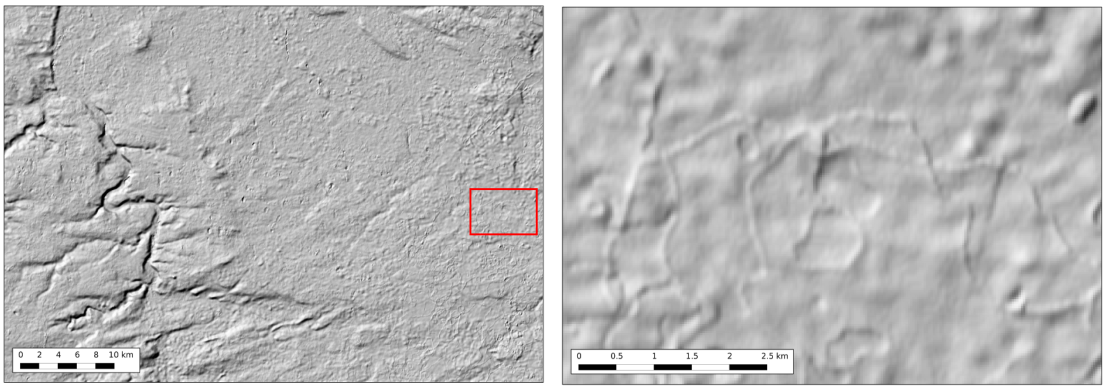

Figure 3: Hill shading map of central Bohemia near capital city Prague generated using the ASTER GDEM data in WGS84 projection. There are many visible artifacts of a splotch shape. The red rectangular area in the left part of the figure is an extent of the cut-out right part and shows those artifacts in detail.

in the SRTM sources but they are few and does not mind so much. They produce small but dense clusters of contour lines as is shown on the figure 4.

Conclusion of this investigation is that though the ASTER GDEM data was very promising at the beginning, they turned out to be inapplicable in practice while there is no significant difference among foreign sources of the SRTM data.

Infractructure

A complete infrastructure of the OpenStreetMap project is quite extensive as a figure 5. shows. Fortunately, only its subset, marked by a red border on the image, is needed to be setupped to run a regularly updated tile service.

At the beginning there is a source of whole planet data in a database dump form in a compressed OSM XML format. It is imported to a PostgreSQL [39] database with a PostGIS [40]

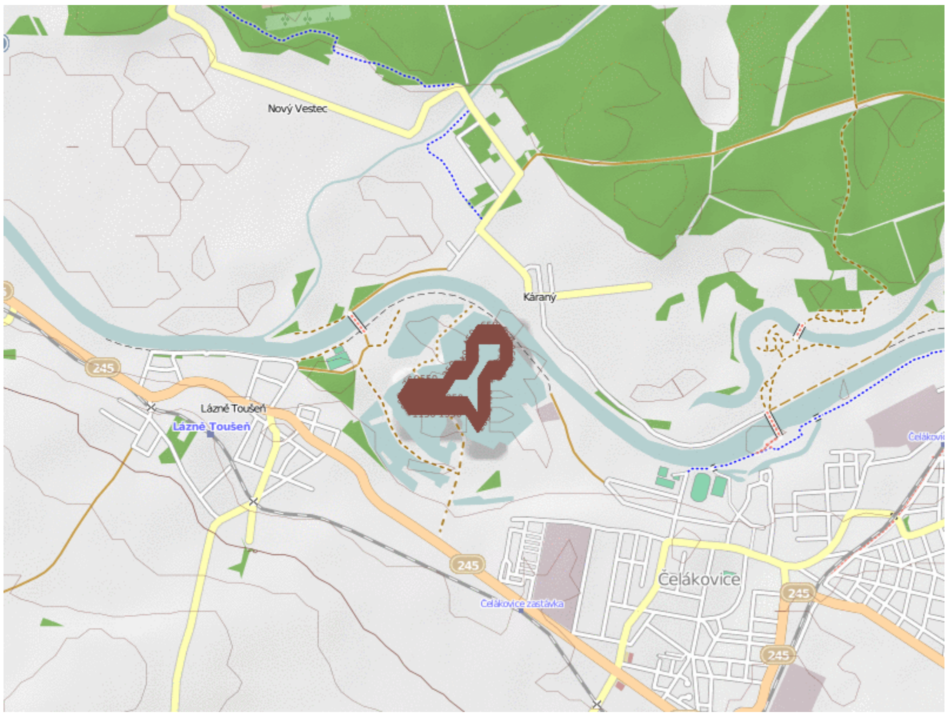

Figure 4: A detail of a dense cluster of contour lines generated from the SRTM data on a place of the missing data.

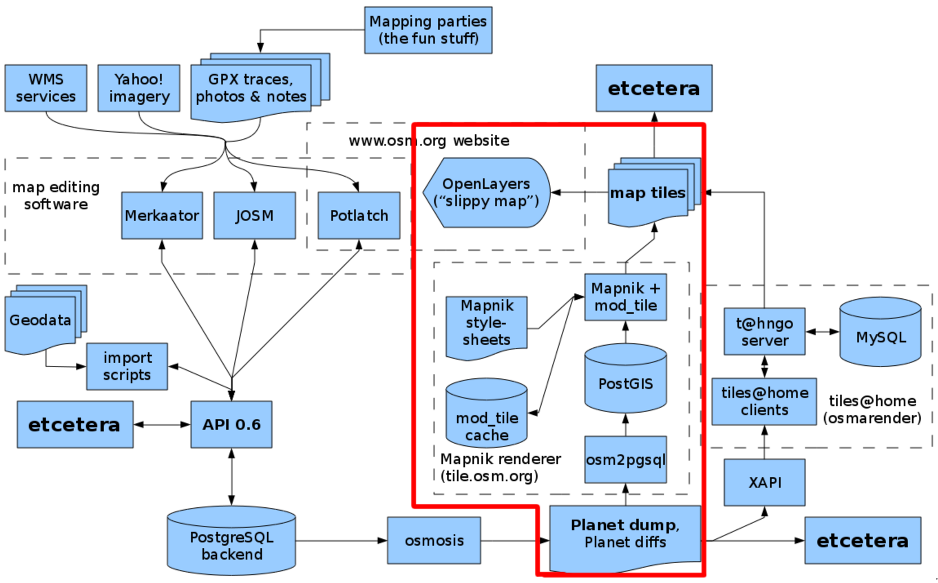

Figure 5: Scheme of the OpenStreetMap infrastructure. Source of image [38]

extension using a osm2pgsql [41] tool. An Apache Web server with a mod_tile [42] module is running on the server machine. This module directs tile requests to a tile cache on a server's disk storage if tile is in the cache is present and up-to-date or to a resident daemon which renders the tile using the Mapnik library if it is not. A JavaScript AJAX-based [43] web interface called an OpenLayers [44] inlined in a HTML document is responsible for the tiled map display to a user though a web browser while mobile or desktop applications are using the tile requests directly using the SlippyMap [7] scheme.

Raster and Vector Data Storage

Vector Data

First step to establish the OpenSteetMap based tile server is to prepare the vector data for rendering. Though the Mapnik supports rendering from the OSM XML files directly, it is more efficient and customizable to import the OSM XML data to the PostgreSQL database with the PostGIS extension. The only way how to do this is to use the *osm2pgsql* because an another tool for the OpenSteetMap data processing an *osmosis* [45] exports to the PostGIS database using a different schema.

Let us suppose that there is a fresh installation of a PostgreSQL 8.4, a PostGIS 1.4.0 and a SVN version of the *osm2pgsql*. Then the following steps will prepare the data for the Mapnik rendering. They are relevant to Ubuntu Karmic Koala desktop but they should be more or less the same for other systems. See [46,47] for the details.

First we create a new database with a PL/pgSQL and PostGIS support:

```
$ su postgres
$ createuser osmuser
$ createdb -E UTF8 -O osmuser osm
$ createlang plpgsql osm
$ psql -d osm -f /usr/share/postgres/8.4/contrib/postgis.sql # if PostGIS compiled from source
$ psql -d osm -f /usr/share/postgres-8.4-postgis/lwpostgis.sql # if installed from Ubuntu package
$ echo "ALTER TABLE geometry_columns OWNER TO osmuser; ALTER TABLE spatial_ref_sys OWNER TO osmuser;" \
    | psql -d osm
```

The *osm2pgsql* operates in two different modes. First, the default one, uses a memory efficient database schema but it does not support further changes in data using this tool. Second is a slim mode which adds additional tables that are needed for incremental updates support. Slim mode heavily utilizes array types so we need to include an *initarray* PostgreSQL module from contrib package:

```
$ psql -d osm -f /usr/share/postgresql/8.4/contrib/_init.sql
```

In the *osm2pgsql* package is a 900913.sql script which has to be imported to the database to include a Google projection:

```
$ cd osm2pgsql/
$ psql -d osm -f 900913.sql
```

Now the database is prepared for the import of the data downloaded from

http://planet.openstreetmap.org/planet-latest.osm.bz2.

osm2pgsql imports only tags that are truly necessary for rendering. Which one they are, it is specified in a *default.style* file. Following code is a diff file against SVN version to include tags needed for hiking tracks rendering:

```
Index: default.style
===================================================================
--- default.style (revision 18005)
```

```
+++ default.style (working copy)
@@ -86,18 +86,37 @@

 # If you're interested in bicycle routes, you may want the following fields
 # To make these work you need slim mode or the necessary data won't be remembered.
-#way lcn_ref text linear
-#way rcn_ref text linear
-#way ncn_ref text linear
-#way lcn text linear
-#way rcn text linear
-#way ncn text linear
-#way lwn_ref text linear
-#way rwn_ref text linear
-#way nwn_ref text linear
-#way lwn text linear
-#way rwn text linear
-#way nwn text linear
-#way route_pref_color text linear
-#way route_name text linear
+way lcn_ref text linear
+way rcn_ref text linear
+way ncn_ref text linear
+way lcn text linear
+way rcn text linear
+way ncn text linear
+way lwn_ref text linear
+way rwn_ref text linear
+way nwn_ref text linear
+way lwn text linear
+way rwn text linear
+way nwn text linear
+way route_pref_color text linear
+way route_name text linear

+# Czech style hiking tracks.
+way kct_yellow text linear
+way kct_red text linear
+way kct_green text linear
+way kct_blue text linear
+
+# Slovak style hiking tracks.
+way marked_trail text linear
+way marked_trail_yellow text linear
+way marked_trail_red text linear
+way marked_trail_green text linear
+way marked_trail_blue text linear
+
+# International hiking tracks.
+way network text linear
+way iwn text linear
+
+# Signposts.
+node information text nocache
```

And the actual import commands are:

```
$ cd osmosis/
$ bzcat planet-lastest.osm.bz2 | ./osmosis --read-xml file=- --bounding-box left="12.10" \
    right="18.87" top="51.32" bottom="48.27" --write-xml file=../czech_republic.osm.gz
$ cd osm2pgsql/
$ ./osm2pgsql -s -m -d osm ../czech_republic.osm.gz
```

The *osmosis* command here serves for bounding the data to a region of the Czech Republic. Although the *osm2pgsql* can bound the data too, this feature is quite slow comparing to the *osmosis* way. The *osmosis* can read compressed OSM XML files directly but there is a bug in the implementation. The *osm2pgsql* has to be run from its directory, the *-s* option enables the slim mode and the *-m* option specifies that a Merkaartor projection is used.

Hill Shading

To create raster hill shading layer in GeoTIFF format from the USGS SRTM elevation data downloaded in a zipped HGT format, this simple script should be sufficient:

```
for TILE in N*E*.hgt.zip; do
  yes | ./srtm_generate_hdr.sh ${TILE}
  rm -f "${TILE%%.hgt.zip}.bil" "${TILE%%.hgt.zip}.hdr" "${TILE%%.hgt.zip}.prj" "${TILE%%.hgt.zip}.hgt"
done
gdal_merge.py -v -o srtm.tif -ul_lr 12.10 51.06 18.87 48.55 N*E*.tif
gdalwarp -of GTiff -co "TILED=YES" -co "BIGTIFF=YES" -srcnodata 32767 -dstnodata 0 -t_srs "+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 +nadgris=@null +no_defs" \
  -rcs -order 3 -tr 30 30 -wt Float32 -ot Float32 -wo SAMPLE_STEPS=100 -multi srtm.tif warped.tif
hillshade warped.tif hillshade.tif -s 0.2
```

It first converts the HDRs to the GeoTIFFs, merges them to a single big GeoTIFF, projects it to a Merkaartor projection and then generates a hill shade GeoTIFF image.
The *srtm_generate_hdr.sh* script can be downloaded from [49], the *hillshade* program belongs to a *demtools* [50] package and the *gdal_merge.py* and the *gdalwarp* are part of GDAL library [29]. Intensity of the hill shading can be controlled with a *-s* parameter of the *hillshade* command, lower values means higher intensity. *-co "BIGTIFF=YES"* argument and BigTIFF support in GDAL are needed if size of GeoTIFF files exceeds 4 GB.

Contour Lines

A generation and an import of the contour lines to the database in a vector form goes with several steps. First one, the HGT format conversion, is same as for the hill shading. Then a shapefile with the contours is generated using a *gdal_contour* tool from the GDAL library. Then it's imported with a *shp2pgsql* which is part of the PostGIS.

```
CREATE="1"
for TILE in N*E*.hgt.zip; do
  yes | ./srtm_generate_hdr.sh ${TILE}
  rm -f "${TILE%%.zip}" "${TILE%%.hgt.zip}.shp" "${TILE%%.hgt.zip}.shx" "${TILE%%.hgt.zip}.dbf"
  gdal_contour -i 10 -snodata -32767 -a height "${TILE%%.hgt.zip}.tif" "${TILE%%.hgt.zip}.shp"
  if [ "$CREATE" == "1" ]; then
    shp2pgsql -d -I -g way "${TILE%%.hgt.zip}" contours | psql -q osm
  else
    shp2pgsql -a -g way "${TILE%%.hgt.zip}" contours | psql -q osm
 fi;
 unset CREATE
 rm -f "${TILE%%.hgt.zip}.shp" "${TILE%%.hgt.zip}.shx" "${TILE%%.hgt.zip}.dbf"
 rm -f "${TILE%%.hgt.zip}.bil" "${TILE%%.hgt.zip}.hdr" "${TILE%%.hgt.zip}.prj"
done
```

It was tried that the import from the separate shapefiles generated from each elevation data tile is more efficient for rendering than merging the tiles followed by the generation and the import from a single big image. This is because first method produces shorter and segmented contour lines which are bounded on smaller spatial area.

Rendering

Mapnik Stylesheet Design

When designing the Mapnik XML stylesheet for the OpenTrackMap project, a default Open-SteetMap stylesheet from a OpenStreetMap SVN [51] was taken as a template and customizing modifications was applied into it.

Since the Czech Republic is an inland country, there is no need to render a coastline layers but default color of the map has to be set to white instead of light blue:

```
...
-<Map bgcolor="#b5d0d0" srs="&srs900913;">
+<Map bgcolor="#ffffff" srs="&srs900913;">
...
-<Layer name="world" status="on" srs="&srs900913;">
-    <StyleName>world</StyleName>
-    <Datasource>
-      <Parameter name="type">shape</Parameter>
-      <Parameter name="file">&world_boundaries;/shoreline_300</Parameter>
-    </Datasource>
-</Layer>
-<Layer name="coast-poly" status="on" srs="&srs900913;">
-    <StyleName>coast-poly</StyleName>
-    <Datasource>
-      <Parameter name="type">shape</Parameter>
-      <Parameter name="file">&world_boundaries;/processed_p</Parameter>
-    </Datasource>
-</Layer>
-<Layer name="builtup" status="on" srs="&srsmercator;">
-    <StyleName>builtup</StyleName>
-    <Datasource>
-      <Parameter name="type">shape</Parameter>
-      <Parameter name="file">&world_boundaries;/builtup_area</Parameter>
-    </Datasource>
-</Layer>
...
```

The contour lines imported to the database and the hill shading layer in a GeoTIFF file can be added using following style and layer directives:

```
...
+
+<Style name="shading">
+    <Rule>
+      &maxscale_zoom9;
+      &minscale_zoom18;
+      <RasterSymbolizer>
+        <CssParameter name="opacity">0.5</CssParameter>
+        <CssParameter name="mode">multiply</CssParameter>
+        <CssParameter name="scaling">bilinear8</CssParameter>
+      </RasterSymbolizer>
+    </Rule>
+</Style>
+
+
+<Style name="contours10">
+    <Rule>
+      &maxscale_zoom14;
+      &minscale_zoom18;
+      <LineSymbolizer>
+        <CssParameter name="stroke">#844c44</CssParameter>
+        <CssParameter name="stroke-width">0.3</CssParameter>
+      </LineSymbolizer>
+    </Rule>
+</Style>
...
+
+<Layer name="hillshade" status="on" srs="&srs900913;">
+    <StyleName>shading</StyleName>
+    <Datasource>
+      <Parameter name="type">gdal</Parameter>
+      <Parameter name="file">&hillshade;</Parameter>
+      <Parameter name="format">tiff</Parameter>
+    </Datasource>
+</Layer>
+
+
+<Layer name="contours10" status="on" srs="+proj=latlong +datum=WGS84">
+    <StyleName>contours10</StyleName>
+    <StyleName>contours-text10</StyleName>
+    <Datasource>
+      <Parameter name="table">(SELECT way,height FROM contours WHERE height::integer % 10 = 0
+                             AND height::integer % 50 != 0 AND height::integer % 100 != 0)
+                             as "contours10"</Parameter>
+      &latlon-datasource-settings;
+    </Datasource>
+</Layer>
...
```

Only the style and layer for the contour lines with an elevation dividable by 10 meters and displayed at higher zoom levels are showed in this article. Other styles and layers for the contour lines with the elevation dividable by 50 and 100 meters and displayed only at lower zoom levels can be found in a complete OpenTrackMap Mapnik stylesheet [52,53].

Hiking tracks of the CTC are marked with one of four colors: red, green, blue and yellow [18,19,20]. They are mostly mapped using relations and tagged with *kct_red*, *kct_green*, *kct_blue* and *kct_yellow* tags respectively. Some of the allowed values are *major*, *local*, *learning*, *ski* and *horse* [54] depending on the type of the track. Slovak hiking tracks are tagged with *marked_trail_<color>* counterparts of CTC-style tags and they are taken into account too. Since the hiking tracks of different colors can go in parallel on the same road, special care has to be taken to display them correctly and legibly.

The main problem about this is that road segments that are part of multiple hiking track relations of a different color has to be detected and rendered appropriately. The OpenTrackMap face this problem by copying segments of relations with *kct_<color>* tags to a new database table of lines a using Python script [55]. Disadvantage of this solution is that it is not much friendly to regularly performed incremental updates since whole table has to be recreated each time the data are updated. Better option is to modify *osm2pgsql* tool to do this automatically with each update and it is a space for further advancement.

The tracks are then rendered from the custom table with the copied segments and a default table without the relations with following styles and layers using dashed lines of intermittent colors:

```
...
+
+<Style name="red-green-track">
+  <Rule>
+    &maxscale_zoom13;
+    &minscale_zoom18;
+    <LineSymbolizer>
+      <CssParameter name="stroke">#dd0000</CssParameter>
+      <CssParameter name="stroke-opacity">0.7</CssParameter>
+      <CssParameter name="stroke-width">4</CssParameter>
+      <CssParameter name="stroke-dasharray">4,4</CssParameter>
+    </LineSymbolizer>
+    <LineSymbolizer>
+      <CssParameter name="stroke">#00dd00</CssParameter>
+      <CssParameter name="stroke-opacity">0.7</CssParameter>
+      <CssParameter name="stroke-width">4</CssParameter>
+      <CssParameter name="stroke-dasharray">0,4,4,0</CssParameter>
+    </LineSymbolizer>
+  </Rule>
+</Style>
...
+<Layer name="red-green-track" status="on" srs="&srs900913;">
+    <StyleName>red-green-track</StyleName>
+    <StyleName>red-shield</StyleName>
+    <StyleName>green-shield</StyleName>
+    <Datasource>
+      <Parameter name="table">((SELECT osm_id,way,route,name,ref,kct_red,kct_green,
+                              marked_trail_red,marked_trail_green,
+                              char_length(ref) AS length FROM &prefix;_line WHERE osm_id > 0 AND
+                           ((kct_yellow IS NULL) AND (marked_trail_yellow IS NULL)) AND
+                           ((kct_red IS NOT NULL) OR (marked_trail_red IS NOT NULL)) AND
+                           ((kct_green IS NOT NULL) OR (marked_trail_green IS NOT NULL)) AND
+                           ((kct_blue IS NULL) AND (marked_trail_blue IS NULL))) UNION
+                           (SELECT osm_id,way,route,name,ref,kct_red,kct_green,
+                            marked_trail_red,marked_trail_green,
+                              char_length(ref) AS length FROM &prefix;_track_rels WHERE
+                           ((kct_yellow IS NULL) AND (marked_trail_yellow IS NULL)) AND
+                           ((kct_red IS NOT NULL) OR (marked_trail_red IS NOT NULL)) AND
+                           ((kct_green IS NOT NULL) OR (marked_trail_green IS NOT NULL)) AND
+                           ((kct_blue IS NULL) AND (marked_trail_blue IS NULL)))) as
+                             red_green_track
+      </Parameter>
+      &datasource-settings;
+    </Datasource>
+</Layer>
...
```

This is the style and layer just for a red-green combination of the hiking track colors and only for higher zoom levels. In the full stylesheet [52,53], there are styles and layers for all combinations of four colors and for three ranges of the zoom levels, that is fifteen different styles with three rules each.

Track type is visualized using shields with symbols placed on the lines:

```
...
+<Style name="red-shield">
+  <Rule>
+    <Filter>[kct_red]='ski'</Filter>
+    &maxscale_zoom13;
+    &minscale_zoom18;
+    <ShieldSymbolizer name="osm_id" face_name="DejaVu Sans Bold" size="0"
+      placement="line" file= "&symbols;/kct-ski-red.png" type="png" width="16"
+      height="16" min_distance="10" spacing="100"/>
+  </Rule>
+  <Rule>
+    <Filter>[kct_red]='horse'</Filter>
+    &maxscale_zoom13;
+    &minscale_zoom18;
+    <ShieldSymbolizer name="osm_id" face_name="DejaVu Sans Bold" size="0"
+      placement="line" file= "&symbols;/kct-horse-red.png" type="png" width="16"
+      height="16" min_distance="10" spacing="100"/>
+  </Rule>
+  <Rule>
+    <Filter>[kct_red]='local'</Filter>
+    &maxscale_zoom13;
+    &minscale_zoom18;
+    <ShieldSymbolizer name="osm_id" face_name="DejaVu Sans Bold" size="0"
+      placement="line" file= "&symbols;/kct-local-red.png" type="png" width="16"
+      height="16" min_distance="10" spacing="100"/>
+  </Rule>
+  <Rule>
+    <Filter>[kct_red]='learning'</Filter>
+    &maxscale_zoom13;
+    &minscale_zoom18;
+    <ShieldSymbolizer name="osm_id" face_name="DejaVu Sans Bold" size="0"
+      placement="line" file= "&symbols;/kct-learning-red.png" type="png" width="16"
+      height="16" min_distance="10" spacing="100"/>
+  </Rule>
+</Style>
...
```

An international hiking tracks rendering is easier to set up comparing to the local ones:

```
...
+
+<Style name="international-track">
+  <Rule>
+    <Filter>[route]='foot' and ([network]='e-road' or [network]='nwn' or [network]='iwn')</Filter>
+    &maxscale_zoom5;
+    &minscale_zoom8;
+    <LineSymbolizer>
+      <CssParameter name="stroke">#dd0000</CssParameter>
+      <CssParameter name="stroke-opacity">0.7</CssParameter>
+      <CssParameter name="stroke-width">2</CssParameter>
+    </LineSymbolizer>
+  </Rule>
+</Style>
...
+
+<Layer name="international-track" status="on" srs="&srs900913;">
+    <StyleName>international-track</StyleName>
+    <Datasource>
+      <Parameter name="table">
+        ((SELECT osm_id,way,route,name,ref,network,char_length(ref) AS length FROM &prefix;_line
+          WHERE osm_id > 0 AND
+         route='foot' AND (network='e-road' OR network='nwn' OR network='iwn')) UNION
+        (SELECT osm_id,way,route,name,ref,network,char_length(ref) AS length from &prefix;_track_rels
+         WHERE route='foot' AND (network='e-road' OR network='nwn' OR network='iwn')))
+         as international_tacks
+      </Parameter>
+      &datasource-settings;
+    </Datasource>
+</Layer>
...
```

Static Tile Rendering

For the static tiles rendering, a custom multithread version of a OpenStreetMap's *generate_tiles.py* Python script was written [56] because in early times of the OpenTrackMap project existence, no such script was available. The multithreading was introduced to this script as far as with changeset 17484 [57]. Rendering times for different zoom levels with the OpenTrackMap Mapnik stylesheet are discussed in a section "Experimental Results".

On-Demand Tile Rendering

A *mod_tile* [25] is an Apache module that allows us to serve, cache and render tiles with a Mapnik in several layers using different XML stylesheets. It communicates with standalone *renderd* daemon, that performs actual rendering, via sockets.

An Apache virtual host configuration for the Web server with the *mod_tile* and statically served tiles could look like this:

```
<VirtualHost *>
  # Virtual host configuration.
  DocumentRoot /mnt/data/OpenTrackMap/
  ServerName opentrackmap.no-ip.org

  # renderd configuration.
  LoadTileConfigFile /etc/apache2/renderd.conf
  ModTileRequestTimeout 60
  ModTileMaxLoadOld 2
  ModTileMaxLoadMissing 5
  ModTileRenderdSocketName /var/run/renderd/renderd.sock
  ModTileCacheDurationMax 604800
  ModTileCacheDurationDirty 900
  ModTileCacheDurationMinimum 10800
  ModTileCacheDurationMediumZoom 14 86400
  ModTileCacheDurationLowZoom 9 518400
  ModTileCacheLastModifiedFactor 0.20
  LogLevel debug

  # Location for index.html with OpenLayers interface.
  <Location />
    Allow From All
  </Location>

  # Location for statically served tiles.
  <Location /tiles/>
    Allow From All
    SetHandler DefaultHandler
  </Location>
</VirtualHost>
```

The *mod_tile* configuration comprehends a specification of a *renderd* configuration file and several options for tuning caching behavior. Next is a root location where an *index.html* file with the OpenLayers JavaScript interface is coded. The last location is directory with the statically rendered tiles.

The *renderd* configuration file with a single on-demand rendered layer accessible under a *http://opentrackmap.no-ip.org/default/* URI is following:

```
[renderd]
socketname=/var/run/renderd/renderd.sock
num_threads=2
tile_dir=/mnt/data/OpenTrackMap/
stats_file=/var/run/renderd/renderd.stats

[mapnik]
plugins_dir=/usr/lib/mapnik/input
font_dir=/usr/share/fonts/
font_dir_recurse=1

[default]
URI=/default/
XML=/mnt/data/OpenTrackMap/osm.xml
HOST=opentrackmap.no-ip.org
```

Here, the *socketname* option must match the *ModTileRenderdSocketName* option in the Apache virtual host configuration. To enable another layer, just the *[default]* section is needed to be copied.

Since the *mod_tile* is still quite a fresh project with its flaws, it is necessary to register a cron job that looks after renderd daemon state:

```
TEST=‘ps -A | grep -w renderd‘
if [[ -n $TEST ]]
then
  echo ‘date‘ ": OK" >> /var/log/apache2/renderd.log
else
  echo ‘date‘ ": KO" >> /var/log/apache2/renderd.log
  /etc/init.d/renderd restart
fi
```

Automatic Server Updates

The automatic server updates are necessary for a quick feedback during track editing. Mappers would like to see their edits in real render as soon as possible. Concerning a technical restrictions, a database updated once per hour seems to be a good compromise. The main problem with this is a question how to effectively import full and incremental data files of a bounded area of Czech Republic. Also remember that the *relations2lines.py* script has to be re-run each time the data are changed.

Full Update

There are three different possibilities for the full updates. A first is to import a planet file [48] with an *osm2pgsql* tool and use its *-b* option to specify the bounding box limits. See a *full_update_osm2pgsql.py* script [58] for details. A second is to bound data with an *osmosis* tool and then import the result with the *osm2pgsql*. This is placed in a *full_update_osmosis.py* script [59]. Finally, the third option is to use an already bounded OSM data from a publicly available service such as [60,61] are.

First solution turned out to be too slow comparing to the *osmosis* way - 16:13 h vs. 1:50 h. This is probably caused by a fact that a bounding algorithm implementation using memory operations written in Java is much faster than a PostGIS implementation of it. Using the data bounded by another developers has its drawbacks in higher delay to an official OpenSteetMap database and in a need of knowing an exact timestamp and bounding area parameters of the data. For this reasons, the second way is currently used in the OpenTrackMap project.

Incremental Updates

After a first full update is performed, another updates started with a regular interval can be incremental to the full one. The *osm2pgsql* supports this feature but data has to be imported in a so called slim mode which stores overhead information to additional tables. To bound changeset data more ways are also possible. Using the *osmosis* tool to bound a changeset of a larger area is not much efficient because the changeset is applied to an old full data first, a new full data are bounded and then a bounded changeset is determined by comparison with the old full data. On the other hand, this can be more optimal for smaller areas. An *incremental_updates_osm2pgsql.py* script [62], that is utilized in the OpenTrackMap project, uses direct changeset import with the *osm2pgsql* tool and with the *-b* option.

Practical Results

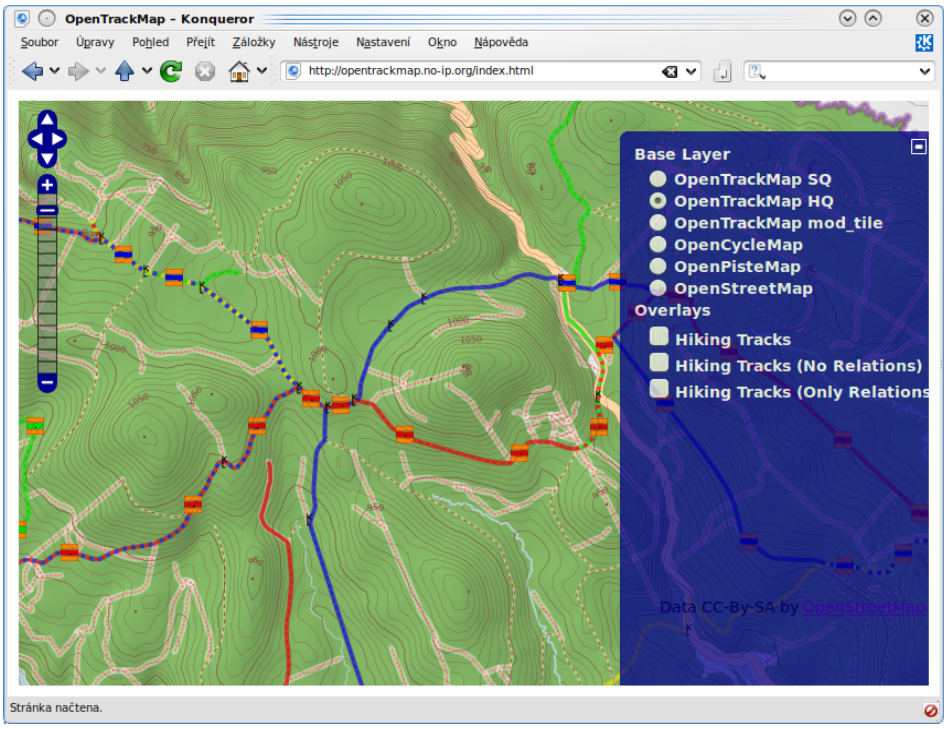

Figure 6: An OpenLayers based OpenTrackMap web interface. A baked layer with all features displayed.

Figure 6. shows an OpenTrackMap full-featured statically served layer in a web browser on a web page with an OpenLayers interface [44] while figure 7 shows a default OpenSteetMap layer as a base layer and a layer with just hiking tracks rendered on-demand using the mod_tile as an overlay.

Figure 8. shows the OpenTrackMap full and tracks-only layers in a TangoGPS application which can be run on an Openmoko Neo Freerunner mobile smartphone.

Finally, figure 9. presents the OpenTrackMap on a Sony Ericsson G705 cell phone with a Java Mobile support in a Mobile GMaps application [63]. For this application simple wrapper script that transforms Google URL scheme to SlippyMap scheme has to be written for the web server:

```
from mod_python import apache
from mod_python.util import FieldStorage

def handler(req):
    args = FieldStorage(req)
    if args.has_key("zoom") and args.has_key("x") and args.has_key("y"):
       req.internal_redirect("/tiles_hq/%d/%s/%s.png" % (17 - int(args["zoom"]),
         args["x"], args["y"]))
```

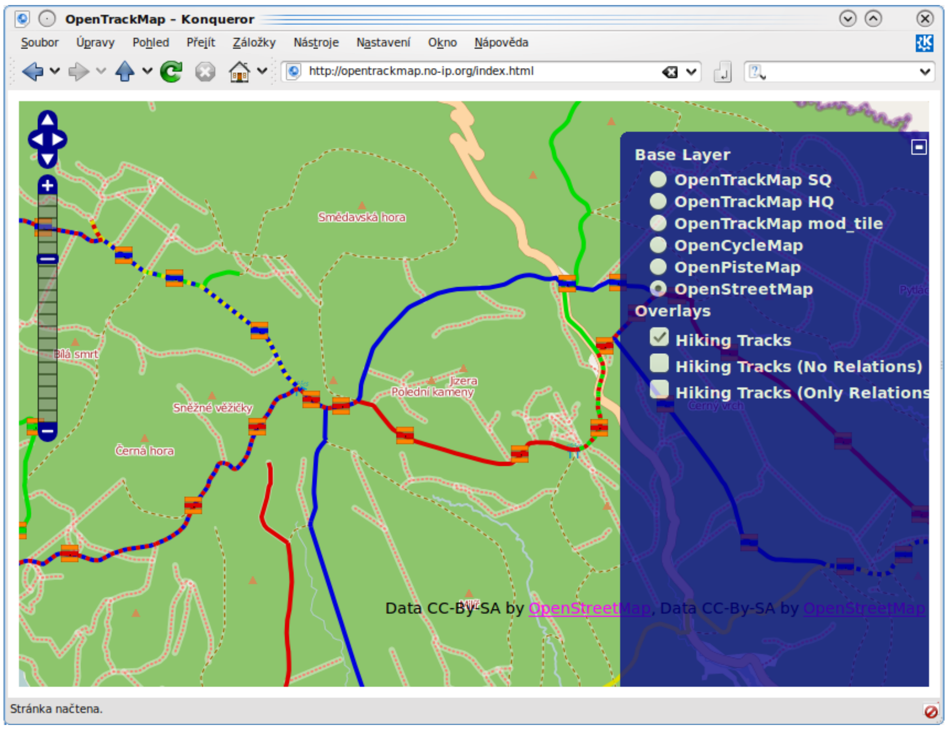
OpenTrackMap - Konqueror
Soubor Úpravy Pohled Přejít Záložky Nástroje Nastavení Okno Nápověda
http://opentrackmap.no-ip.org/index.html
Base Layer
OpenTrackMap SQ
OpenTrackMap HQ
OpenTrackMap mod_tile
OpenCycleMap
OpenPisteMap
OpenStreetMap
Overlays
Hiking Tracks
Hiking Tracks (No Relations)
Hiking Tracks (Only Relations
Smědavská hora
Bílá smrt
Sněžné věžičky
Černá hora
Jizera
Polední kameny
Data CC-By-SA by OpenStreetMap, Data CC-By-SA by OpenStreetMap
Stránka načtena.

Figure 7: An OpenLayers based OpenTrackMap web interface. A layer served from OpenStreetMap displayed with a hiking tracks layer overlay rendered on-demand.

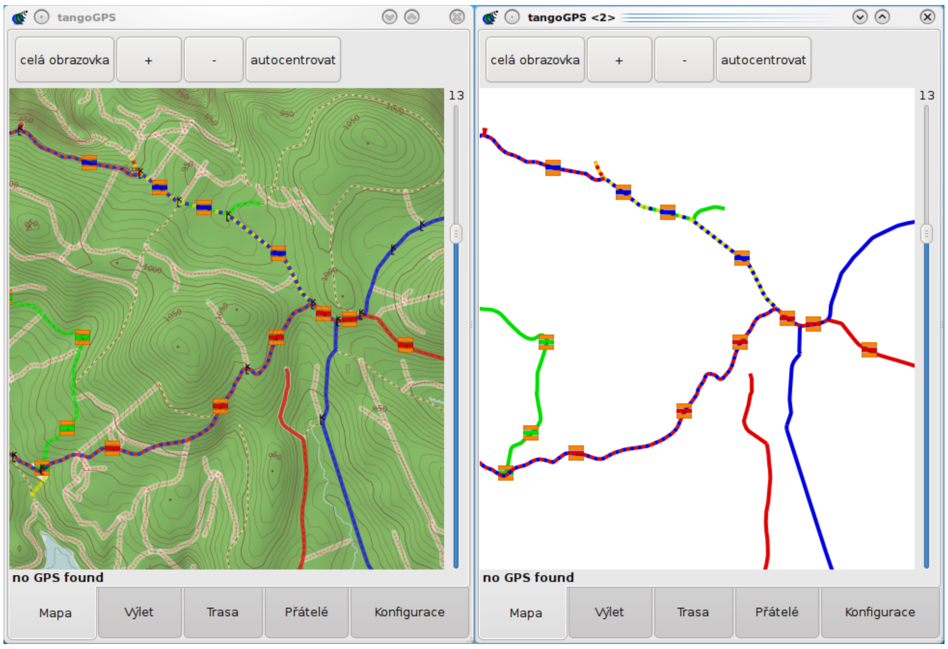

Figure 8: The full statically served layer (left) and the hiking tracks on-demand rendered layer in a TangoGPS application.

```
        return apache.OK
    else:
        return apache.HTTP_NOT_FOUND
```

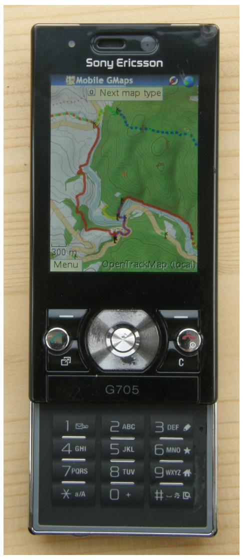

Figure 9: The OpenTrackMap on a mobile device Sony Ericsson G705 in a Mobile GMaps application.

Experimental Results

Few experiments with the *generate_tiles.py* script [56] was performed. Their results are on the figures 10., 11. and 12. While the first two show summarized rendering times (Y axis) in minutes for rendering of tiles up to certain zoom level (X axis) in a linear and a logarithmic scale, last figure displays average rendering time per tile in seconds depending on the maximal zoom level rendered.

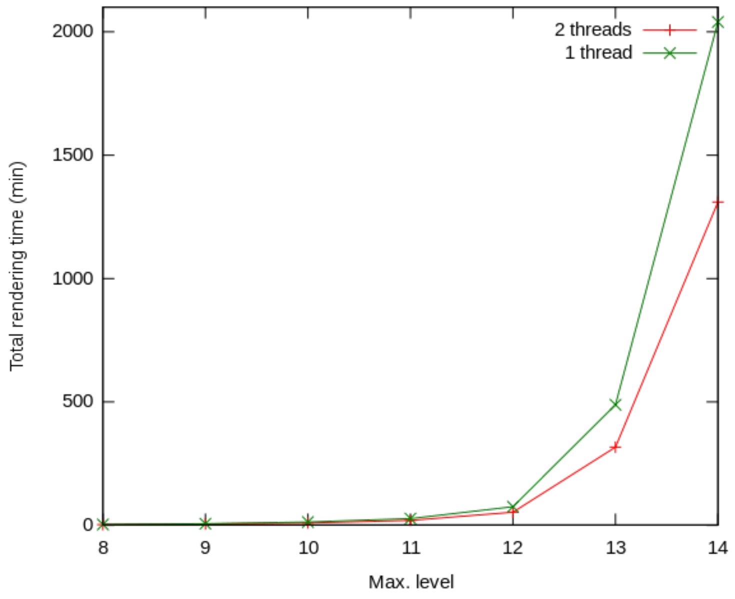

Figure 10: Rendering time in minutes dependence on a maximal zoom level rendered – a linear scale.

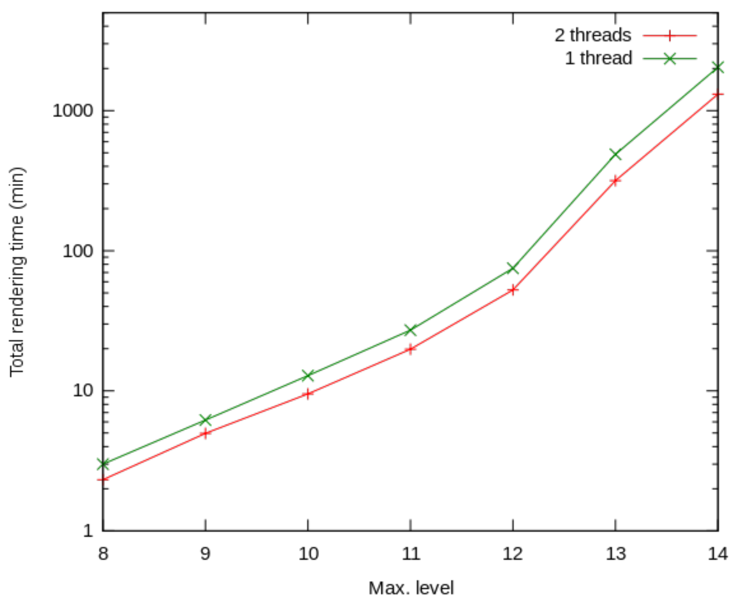

Figure 11: Rendering time in minutes dependence on a maximal zoom level rendered – a logarithmic scale.

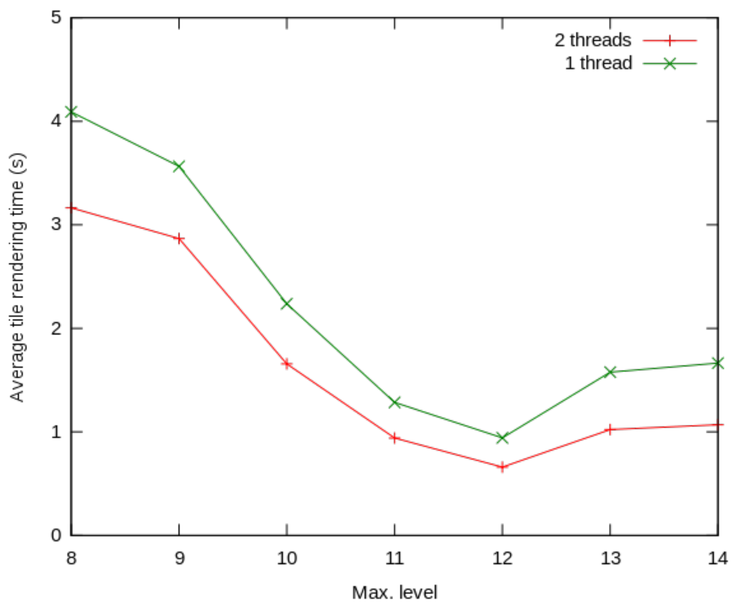

Figure 12: Average tile rendering time in seconds dependence on a maximal zoom level rendered.

All tests was done on an Intel Core 2 Duo 3 GHz machine with a 2 GB of RAM and two processing cores using one and two rendering threads. A main finding about them is that a parallel speedup of the two-thread rendering is not approaching 2 but it is approximately 1.56. This is because the one-thread rendering is already partially parallel since the Mapnik rendering and a PostgreSQL database look-ups run in separate processes. It would be interesting to try how would the parallel rendering scale on computers with more cores or in a distributed environment.

The figure 12. also shows a rendering complexity of a displayed data on each zoom level. Levels 8 to 11 are slow for they render large areas of a hill shading layer. There is rising rendering time for levels 13 and 14 because on those levels more detailed data are being exposed.

Conclusion and Future Work

Although the OpenTrackMap project is existing only for four months, it has already fulfilled most of its resolutions. Yet, a lot of things can be done or improved. Namely they are the following:

- More interesting point objects of a different kind could be displayed.
- A full and stable support for the automatic hourly updates is not finished yet.
- Another source of the elevation data with a better void data fill could be searched for.
- A KML based raster image and/or vector data service (to be used in Google Earth) could be established.
- Rendering and data import process could be optimized.
- Overall quality of the service including the OpenLayers interface could be improved.

Only time and interest of OpenTrackMap's users can make those requests real.

References

1. OpenStreetMap Homepage
   http://www.openstreetmap.org
2. Czech Tourist Club Homepage
   http://www.klubturistu.cz/
3. OpenTrackMap Homepage
   http://opentrackmap.no-ip.org
4. Tile Map Service Specification
   http://wiki.osgeo.org/wiki/Tile_Map_Service_Specification
5. Wikipedia – Web Map Service
   http://en.wikipedia.org/wiki/Web_Map_Service
6. Open Geospatial Consorcium, Inc. – Web Map Service
   http://www.opengeospatial.org/standards/wms
7. OpenStreetMap – Slippy Map
   http://wiki.openstreetmap.org/wiki/Slippy_map
8. Wikipedia – Geocoding
   http://en.wikipedia.org/wiki/Geocoding

9. Mapy.cz (CZ)
   http://mapy.cz/

10. Openmoko
    http://wiki.openmoko.org/

11. openstreetmap.cz (CZ)
    http://openstreetmap.cz/

12. Freemap Slovakia
    http://www.freemap.sk/

13. OSMC Reit- und Wanderkarte
    http://topo.geofabrik.de

14. OpenCyclemap
    http://opencyclemap.org

15. http://beta.letuffe.org
    http://beta.letuffe.org

16. Freemap
    http://www.free-map.org.uk

17. OpenPisteMap
    http://openpistemap.org

18. Klub českých turistů – Pěší značení (CZ)
    http://www.klubturistu.cz/?oid=10191

19. Klub českých turistů – Lyžařské značení (CZ)
    http://www.klubturistu.cz/?oid=10192

20. Klub českých turistů – Cyklo značení (CZ)
    http://www.klubturistu.cz/?oid=10193

21. Mapnik
    http://mapnik.org/

22. Mapnik – Mapnik Configuration XML
    http://trac.mapnik.org/wiki/XMLConfigReference

23. Cascadenik
    http://code.google.com/p/mapnik-utils/wiki/Cascadenik

24. Mapnik – Mapnik Viewer
    http://trac.mapnik.org/wiki/MapnikViewer

25. OpenStreetMap – mod_tile
    http://wiki.openstreetmap.org/index.php/Mod_tile

26. TileLite
    http://bitbucket.org/springmeyer/tilelite/wiki/Home

27. OpenStreetMap – .osm
    http://wiki.openstreetmap.org/wiki/.osm

28. OGR Simple Feature Library
http://www.gdal.org/ogr/

29. GDAL – Geospatial Data Abstraction Library
http://www.gdal.org/

30. Mapnik – Mapnik Renderers
http://trac.mapnik.org/wiki/MapnikRenderers

31. OpenStreetMap – Osmarender
http://wiki.openstreetmap.org/wiki/Osmarender

32. tiles@home
http://tah.openstreetmap.org/

33. Kosmos Home
http://igorbrejc.net/kosmoshome

34. Shuttle Radar Tophgraphy Mission
http://www2.jpl.nasa.gov/srtm/

35. U. S. Geological Survey – Shuttle Radar Topography Mission
http://srtm.usgs.gov/

36. Global Land Cover Facility – Shuttle Radar Tophgraphy Mission
http://www.landcover.org/data/srtm/

37. ASTER Global Digital Elevation Model
http://www.gdem.aster.ersdac.or.jp/

38. OpenStreetMap – Component Overview
http://wiki.openstreetmap.org/wiki/Component_overview

39. PostgreSQL
http://www.postgresql.org/

40. PostGIS
http://postgis.refractions.net/

41. OpenStreetMap – osm2pgsql
http://wiki.openstreetmap.org/wiki/Osm2pgsql

42. OpenStreetMap – mod_tile
http://wiki.openstreetmap.org/index.php/Mod_tile

43. Wikipedia – AJAX
http://en.wikipedia.org/wiki/Ajax_(programming)

44. OpenLayers
http://openlayers.org/

45. OpenStreetMap – osmosis
http://wiki.openstreetmap.org/wiki/Osmosis

46. OpenStreetMap – Mapnik
http://wiki.openstreetmap.org/wiki/Mapnik

47. OpenStreetMap – Mapnik – PostGIS
http://wiki.openstreetmap.org/wiki/Mapnik/PostGIS

48. planet-lastest.osm.bz2
http://planet.openstreetmap.org/planet-latest.osm.bz2

49. Remote Sensing at ITC-irst – MPA Group – SRTM Shutte Radar Topography Mission
http://mpa.itc.it/rs/srtm/

50. GDAL-Based DEM Utilities
http://www.perrygeo.net/wordpress/?p=7

51. OpenStreetMap – osm.xml[1]

52. OpenTrackMap – osm.xml
http://blackhex.no-ip.org/browser/OpenTrackMap/mapnik/osm.xml

53. OpenTrackMap – osm.xml.diff
http://blackhex.no-ip.org/browser/OpenTrackMap/mapnik/osm.xml.diff

54. OpenStreetMap – WikiProject Czech Republic – Editing Standards and Conventions – Turistické značení[2]

55. OpenTrackMap – relations2lines.py
http://blackhex.no-ip.org/browser/OpenTrackMap/bin/relations2lines.py

56. OpenTrackMap – generate_tiles.py
http://blackhex.no-ip.org/browser/OpenTrackMap/mapnik/generate_tiles.py

57. OpenStreetMap – generate_tiles.py – Changeset 17484[3]

58. OpenTrackMap – full_update_osm2pgsql.py[4]

59. OpenTrackMap – full_update_osmosis.py
http://blackhex.no-ip.org/browser/OpenTrackMap/bin/full_update_osmosis.py

60. Geofabrik: Downloads
http://www.geofabrik.de/data/download.html

61. CloudMade: Downloads
http://downloads.cloudmade.com/

62. OpenTrackMap – incremental_update_osm2pgsql.py[5]

63. Mobile GMaps
http://www.mgmaps.com/

[1]http://trac.openstreetmap.org/browser/applications/rendering/mapnik/osm.xml?format=raw
[2]http://wiki.openstreetmap.org/wiki/WikiProject_Czech_Republic/Editing_Standards_and \
_Conventions#Turistick.C3.A9_zna.C4.8Den.C3.AD
[3]http://trac.openstreetmap.org/changeset/17484/applications/rendering/mapnik/generate_t \
iles.py
[4]http://blackhex.no-ip.org/browser/OpenTrackMap/bin/full_update_osm2pgsql.py
[5]http://blackhex.no-ip.org/browser/OpenTrackMap/bin/incremental_update_osm2pgsql.py