Smrender

A Rule-based Renderer for OSM Data

Bernhard R. Fischer

May 16, 2012

Contents

1. Release Notes

This document describes the current (2012/03/01) version of *Smrender* which is tagged as version 1.0 and corresponds to the internal SVN revision number 1187. Unfortunately, this documentation is not complete yet, but I will continue to work on it.
The basic concept of *Smrender* will not change with future versions but there are several constructions which will probably change. This is most notably the format of the actions and their parameters. They just evolved at random during development without any structure. It will be changed to have a well-formed structure also that parts of the code will be rewritten, to be more than just a prototype.
Smrender contains several functions which are experimental. Those functions are namely the auto-rotation and the polygon-size dependent captions (see Section 5.2.1).

2. Name

Smrender is a universal rule-based rendering engine for OSM data. Because smrender is a very generic and flexible OSM processing engine, it may be used for different tasks such as data filtering or data modification.

3. Synopsis

smrender [OPTIONS] window

4. Description

Smrender reads an OSM file and applies a set of rules to this input data to create an output image. The input is an OSM file and a second file containing the rule set. The output (currently) is a PNG image having the desired resolution and density and probably additional output files. The latter is explained later.
The primary goal of *Smrender* is to create a sea chart which is well-suited for print-out on paper. Nevertheless, it is a universal rendering engine and may be used for different tasks.
The input file should be an OSM/XML file as defined by the OSM standard. The file is required to be well-formed in that sense because *Smrender* itself does no XML validation, thus the rendering process might fail if the file is not well-formed. The data should also be complete. This means that it should contain all nodes to which is referred by the ways. Smrender will remove nodes from ways which are missing.
The rules are also defined in OSM format (see Section 5). The rules are applied iteratively in a loop depending on their *version*. Within the loop, *Smrender* always applies first all way rules and then all node rules of the same version. All rules of the same version are applied in the order of their *id*.
Smrender renders an area which is specified by the window. It is a compound argument as defined below.

```
<window> := <lat>:<lon>:<size>
<size>   := <scale> | <length>'d' | <length>'m'
```

lat and lon set the center coordinates in latitude and longitude in degrees in WGS84 reference system. Although it can be any valid coordinate, it is suggested to choose an "even" value rounded to 10 minutes (e.g. 43.666667 which is 43° 40').

Length defines the length of the mean latitude (parallel) in degrees if 'd' is appended or in nautical miles if 'm' is appended. Alternatively, the scale of the chart can be specified. Smrender calculates the size of the area to meet the scale. Obviously, this depends on the size of the output image. The height (the length of the mean longitude) is calculated automatically by *Smrender* in such a way that the output image is projected correctly using Mercartor projection. The height depends on the size of the output image (page format).

4.1. Options

-b color

This option allows to set a background color. The color may be define either as a color preset or an HTML-style definition (see 5.2.1). The default color is white.

-d density

Set the density of the output image. The default value is 300 which is typically used for print-outs in good quality.

-f Filter data while loading the input file.

With this option a bounding box is used to load just those nodes and ways which are within the selection. The bounding box is 10% larger then the area which resumes from the selected window. This option is useful if huge input data sets are used, such as the planet file.

Please note that this option works only correctly if the nodes and ways are stored in that order in the input file (first all nodes and then all ways).

-g d[:t[:s]]

This option defines the distance d of the grid lines in minutes. The border of the chart (latitude and longitude axis) also depends on this setting. As it is usual for sea charts, there is a major and a minor axis scale, called ticks and subticks. The ticks are defined by t and the subticks are defined by s in minutes. For a correct result, t as well as d should be integral multiples of s.

Note that *Smrender* internally uses a precision of hundreds of a minute while doing the grid calculation. Thus, the smallest granularity for the grid paramters is 0.01 minutes.

-G Do not generate grid nodes/ways. Smrender actually does not render the grid directly onto the output image but rather generates regular OSM nodes and ways. These objects are then rendered in a way as it is defined by the regular rule set. All ways of the grid are tagged with grid=*. Part of the grid are also labels on the border showing degrees in latitude an longitude. The labels are nodes which are tagged with grid=text and the tag name=* containing the value.

-i file

This option specifies the name of the input file. If this option is omitted, *Smrender* reads from stdin.

-l Output page has landscape format rather than portrait, which is default. This option is used only in conjunction with option **-P** if a literal page format is used.

-o file

Set the filename of the output image. If this option is omitted, stdout is used instead.

-P fmt|geom

Select the page format of the output image. The format can be set either named format fmt which has a specific dimension or as geometry geom which contains the width and height of the page in millimeters in the format widthxheight. Fmt currently supports the values A4 up to A0. If this option is omitted, A3 format is selected by default.

-M This option is mandatory if the input file is larger than the amount of memory available on the rendering system. If the system has enough memory this option can be omitted. Using memory mapping probably is a little bit slower but heavily depends on the operating system. On Linux kernel 2.6.32 there is no significant difference in speed.

In any case, files which are larger then 2 GBytes are only supported on 64 bit operating systems.

-r file

This option specifies the file name of the rules file. If it is omitted, *Smrender* expects the rules file to be named rules.osm.

-w file

This option specifies the file name of an output OSM file. Smrender will dump all nodes/ways to this file that have been selected by the import process and all nodes/ways which have been generated during the rendering process.

These nodes/ways include the objects generated for the grid and also the close coastline ways. Thus the output depends on the options **-f**, **-C**, and **-G**.

If this option is omitted, no output will be generated.

5. Ruleset Definition

The rule set is also defined in OSM format. It contains nodes and ways together with tags. Nodes are considered to be rules for rendering nodes and ways are rules applied to ways. Each object (node or way) has a list of tags. These tags represent patterns which are matched against the tags of the objects which are to be rendered. The values of a tag's key (k="...") and/or value (v="...") may be either just a string which is directly matched in a case-sensitive manner or a special match operation (see Section 5.1). The match operations can be used for the key as well as for the value. Each object has to have a special tag which defines the action that should be carried out in case of a match. This tag has the form _action_=*. The actions are described below in Section 5.2.
The match algorithm always applies all tags to match, and all of them have to match in order to execute the action. If just a single tag does not match, the node is skipped.

5.1. Match Operations

Basically there are the four different match operations *string compare*, *regular expression match*, *greater than*, and *less than*. Additionally, all of them may be *inverted*, or *excluded*.

- *String compare* is the most basic match operation. It does an exact case-sensitive string compare. The following tag matches all objects which own the tag seamark:light_character=*.

```
<tag k='seamark:light_character' v=''/>
```

The empty value v='' represents a wildcard match. It matches any string.

- *Regular expressions* are invoked by enclosing the expression within two slashes /.../ as it is usual in Perl and several other languages. The expression is interpreted as a POSIX extended regular expression (see manpages regex(3) and regex(7)). The following expression matches any object which is tagged with either highway=primary or highway=secondary.

```
<tag k='highway' v='/^(secondary|primary)$/'/>
```

- *Smrender* can interpret tag values as numerical values. Thus, it can do arithmetical comparisons which is *less than* ($<$) and *greater than* ($>$).

  The rule has to contain a number enclosed in square brackets. The direction of the brackets denotes the comparison operation; $[x]$ means that the value of the tag should be lower than x and $]x[$ matches if the tag value is greater than x. The tag value as well as x are always interpreted as decimal number with double precision.

  Square brackets are used deliberately instead of angle brackets to avoid possible conflicts with the XML format or buggy parsers.

  The following rule matches all objects whose seamark:light:range-value is greater than 7.5.

```
<tag k='seamark:light:range' v=']7.5['/>
```

- Match inversion is done by enclosing the match expression within exclamation marks. This inverts the match if and only if the expression matched. This means that if a tag would match the expression, the inversion would return "false" (no match). But it would not return "true" (match) if the expression would not match.

  The following expression would match all objects which have a tag with the key seamark:type but its value is neither landmark nor light_major.

```
<tag k='seamark:type' v='!/landmark|light_major/!'/>
```

- Match exclusion is denoted by enclosing the match expression within tildes. It does not match objects with certain tags. The following rule avoid matching of objects which have a tag seamark=*.

```
<tag k='~seamark~' v=''/>
```

5.2. Rule Actions

Smrender supports a view powerful actions which are carried out upon successful match. As already mentioned at the beginning of this Section, actions are defined simply with the tag _action_=*. The following example shows an action which places an image at the position of a node.

```
<tag k='_action_' v='img:icons/Light_Minor.png'/>
```

The basic format of an action is defined as follows.

```
<action>   := <type> ':' <definition>
<type>     := 'cap' | 'draw' | 'func' |
              'img' | 'img-auto' | 'mskfill' |
              'out' | 'settags'
```

The definition depends on the type of rule.

5.2.1. Captions

The action type cap is used to place a caption. If the action is carried out in a node-rule, the caption is placed at the node's position with the specified properties. The formal definition looks like the following.

```
<action>       := 'cap:' <definition>
<definition>   := <font-def> ',' <size> ',' <alignment> ','
                  <color> ',' <rotation> ',' ['*']<key>
<alignment>    := <horiz> <vert>
<horiz>        := 'e' | 'w' | 'c'
<vert>         := 'n' | 's' | 'm'
<color>        := <col-preset> | <HTML-style color definition>
<col-preset>   := 'white' | 'black' | 'yellow' | 'blue' |
                  'magenta' | 'brown'
<rotation>     := 0 - 360 | <auto-rot>
<auto-rot>     := 'auto' [';' <color> [';' weight [';' phase ]]
<key>          := name of OSM key
```

Smrender currently uses libgd[1] for its drawing operations. Libgd may be compiled with fontconfig[2] support. Fontconfig support usually is enabled if the package libgd2-xpm is installed. *Smrender* will output a warning if fontconfig is not available.
If fontconfig is available, font-def is defined as specified by fontconfig (see fontconfig documentation). This is e.g. "serif:bold". If fontconfig is not available, font-def must be a full path to a TTF font file.
Size defines the size of the font in millimeters.
Alignment specifies the alignment of the caption in respect to its center point which is given by the coordinates of the node. There is a horizontal alignment (horiz) which could be either east, west, or center and a vertical alignment (vert) which is one of north, south, or middle.
Color defines the color in which the caption should be set. This is either a color preset as defined by col-preset or a color definition as used in HTML standard.
Rotation defines how the caption should be rotated. The angle is given as usual in trigonometrics which is degrees counterclockwise from 0 to 360 being 0 the regular left-to-write.[3] Alternatively, rotation may be set to "auto". This causes *Smrender* to try to find a rotation itself in such a way that it conflicts as little as possible which other objects that have been rendered already.
Smrender virtually rotates the caption from 0 to 360 degrees and samples the number of pixels of the given color (default is white) for each angle. It than chooses the angle with the highest number of pixels. If the angle is between 90 and 270 degrees, *Smrender* automatically flips the caption that it does not read upside down.
The *auto-rotation* has two further optional arguments. Weight is a decimal value between 0 and 1 (1 is default) which allows to weight the angles of 90 plus phase and 270 plus phase less than the others (a phase of 0 is default). This allows to e.g. prefer left-right angles above top-bottom angles. This makes sense because reading left-right is more easy than reading top-bottom.

[1] www.libgd.org
[2] fontconfig.org
[3] Please note that this is different to the angle definition of maritime navigation which is degrees clockwise from 0 to 360 being 0 upwards (North).

Finally, key specifies the key of the tag whose value should be printed. If a caption rule is applied to a node which does not have such a key, the rule simply does nothing. If the key is preceded by an asterisk '*', all letters are capitalized.

Captions on ways are handled a little bit different from captions on nodes. Actually, captions on ways (polylines) are not supported yet but captions on areas (close polygons) are supported very well, although it is very experimental.
Smrender will calculate the centroid and the area of the polygon. The caption is than placed at the position of the centroid[4] and the font size is chosen dependent on the square root of the area. Thus, larger polygons get larger captions and smaller ones get smaller captions.

5.2.2. Draw

The *draw* action is used to draw lines of various styles and fill polygons. The following shows the basic rule format.

```
<definition>  := <fillstyle> [':' <borderstyle>]
<fillstyle>   := <style>
<borderstyle> := <style>
<style>       := <color> [',' <width> [',' <pattern>]]
<pattern>     := 'solid' | 'dashed' | 'dotted' | 'transparent'
```

The action behaves a little bit different if it is a polyline (open way, e.g. a river) or a polygon (closed way, i.e. an area, e.g. a lake). In both cases the *fill style* is the dominant part. It is used for the background. The border style surrounds the background, e.g. with a thin line.
The width is given in millimeters. A width of 0 draws the thinnest possible line with a width of one pixel. If the rule is a applied to a polygon, the fill width obviously is ignored.
Smrender provides another more sophisticated action for filling of polygons. This is described in Section 5.2.5.

5.2.3. Calling Functions

Smrender has the ability to call user-defined library functions. This feature provides modularity and the flexibility to be extended on the fly without modifying the core. Thus, *Smrender* can be used for nearly every kind of rule-based OSM file processing. The library calls dlopen(3) and dlsym(3) are used to dynamically import those functions.
The basic rule format is defined in the following.

```
<definition>  := <function> ['@' <libstr> ['?' <param-str>]]
<libstr>      := 'NULL' | <library>
<library>     := path/name of shared library
<param-str>   := <av-pair> [',' <av-pair> [',' ...]]
<av-pair>     := <attr> '=' <val>
```

Function is the name of the function as it is exported by the shared object library. If library contains a '/', the path is resolved and the shared object loaded from that location. Otherwise the dynamic linker tries to find the library in the appropriate system directories.[5]

[4] This is similar to what *Osmarender* does.
[5] See dlopen(3) for details.

Beside importing the function, *Smrender* tries to import the optional functions function_ini() and function_fini().
The function is called on each match of an OSM node. Function_ini() is called once before the first match and, obviously, function_fini() is called onced after the last match.

```
void (*function_ini)(orule_t*);
int (*function)(osm_obj_t*);
void (*function_fini)(void);
```

Function() gets a pointer to the OSM object which matched. The object can be either a *node*, a *way*, or a *relation*.

```
typedef struct osm_obj
{
   // type of object: {OSM_NODE, OSM_WAY, OSM_REL}
   short type;
   // visibility: {0, 1}
   short vis;
   // OSM id
   int64_t id;
   // version, changeset, user id
   int ver, cs, uid;
   // Unix timestamp
   time_t tim;
   // number of tags
   short tag_cnt;
   // Pointer to tags
   struct otag *otag;
} osm_obj_t;
```

The type of object can be determined on examination of osm_obj_t.type. The variable may be set to either of OSM_NODE, OSM_WAY, or OSM_REL.[6] The object can than be type-casted to either a osm_node_t, a osm_way_t, or a osm_rel_t. All those OSM types are defined in osm_inplace.h.
The return value of the function controls the further behavior of *Smrender* while applying this same rule. A return value of 0 means no error. *Smrender* will call the function again at the next matching object. If the return value is greater than 0 it behaves similar but outputs a message in the log file. The message contains the return value. If a negative value is returned, *Smrender* immediately stops applying this rule, calls the _fini() function and processes the next rule.
The initialization function gets a orule_t parameter which contains the rule definition. The structure can be examined. The field orule_t.rule.type is always set to ACT_FUNC. Orule_t.rule.func.parm is of type char* and points to param-str which was set in the rules file. If no parameter was defined, it is set to NULL. Please note that if a param-str should be passed to an internal function, libstr *must* be set to "NULL".
Section B explains how to write own rendering functions more in detail.

Internal Functions may be called if the optional library is omitted as was depicted in the formal rule definition at the beginning of this Section. Currently, *Smrender* provides the following functions.

- act_poly_area()

  This function calculates the area of closed polygons in nautical square miles. It adds the tag smrender:area=* to the way. The value of the tag contains the area.

[6] Please note that OSM_REL is defined but not implemented yet.

- act_poly_centroid()

  This function calculates the centroid of a closed polygon. It then adds a new node at this position. The node will inherit all tags of the polygon. Additionally, the tag smrender:id:way=* is added whose value is set to the ID of the way, respectively.

- cat_poly()

  This function closes open polygons. To have really closed polygons is highly important because just such polygons can be filled with a background color.

  Polygons which are originally closed, such as the coastline or lakes are very often found as a set of open ways whose beginning and end share the same nodes. This is because different tags may be attached to different parts of the polygon. Furthermore, just partial data sets are used as input because typically just a small area out of the world's data is selected.

  This function finds all adjacent ways and closes them properly. The original data is not changed furthermore it creates and inserts new ways. Those new ways are tagged with all tags that where defined in the rule set plus the tag generator=smrender.

  As already mentioned, the typical application for this function is to close the coastline which will be open in most cases. The coastline is always tagged with natural=coastline.

- set_ccw() and set_cw()

  Those functions set the direction of a closed way to either clockwise ("cw") or counterclockwise ("ccw").

- refine_poly()

  This function smooths the edges of polygons. It may take the function arguments iteration and deviation. The first defines the number of loops of the iterative refinement process (default = 3). The latter defines the maximum deviation of the original polyline in meters (default = 50). This avoids too high distortion of the polylines.

- reverse_way()

  This function always reverses a way independent of its current direction.

Security Implications This feature basically allows any user to call arbitrary functions on the system. Thus, *Smrender* should **never ever** be installed with file modes SUID/GUID-root! This would be a potential security risk and might allow an attacker with access to your system to comprompise it.

5.2.4. Placing Images

Smrender allows to place images at the position of nodes. Basically it offers two different actions: img and img-auto.

```
<action>     := <img-act> ':' <file>
<img-act>    := 'img' | 'img-auto'
```

Both actions just take the single argument file which is a path to a PNG file. The image is placed directly at the position of the matching node without any modifications.
In case of img-auto, *Smrender* tries to find a rotation angle. This works as described in Section 5.2.1. This rotation function does not take the image into account except the size. The rotation test starts at direction East and rotates counterclockwise.

This action makes only sense if it is applied to asymmetric non-centered images, such as light flares.

5.2.5. Mask-based Filling

This action basically is used to fill polygons. This function works different from the polygon filling method described in Section 5.2.2.
It creates a temporary mask with all filled polygons which match the match criteria. The function takes care on the direction of the polygon, either if it is clockwise or counterclockwise. The portions filled are those on the *left side* of the directed edges.
This function specifically is useful on multi-polygons where some areas within should be excluded from filling. This function is more complex than the filling primitive draw (see Section 5.2.2), hence, it is slower in execution.
Rule set example:

```
<way>
   <tag k='seamark:type' v='depthcont'/>
   <tag k='seamark:depthcont' v='10'/>
   <tag k='_action_' v='mskfill:blue'/>
</way>
```

5.2.6. Output of OSM Data

With the action out it is possible to create an OSM file which contains all those objects which match.[7] *Smrender* will create one file for each action. This means that if the same file name is used in several out actions, the latter will overwrite the earlier ones. The action takes just one argument, the path to the file.

```
   <action>    := 'out:' <filename>
```

5.2.7. Adding Tags to Objects

The action settags allows to add an arbitrary number of OSM tags to an object. The tags have to be defined through an object within the rules file. This object may have no action tag. This template should have an id because the settags action needs to have a reference to it. To format simple looks like the following.

```
   <action>    := 'settags:' <id>
```

Please note that the object type of the rule must be the same as the object type of the template.

6. Signals

Smrender installs two signal handlers, one for SIGUSR1 and one for SIGINT.
If *Smrender* receives a USR1 signal during the process of reading OSM input data, it outputs some statistics about the current position of reading and data throughput. It may be used as progress indicator if huge files are used as input. If *Smrender* receives a INT signal during rendering, it immediately aborts rendering of further objects and saves the image in its current state and exits normally. If SIGINT is caught twice, *Smrender* exits immediately.

[7] This action is actually just a wrapper for the function act_output().

7. Extensions

As explained in Section 5.2.3, *Smrender* is able to call functions of shared objects through dynamic linking at runtime. Thus it is very easy to extend the core functionality of *Smrender*. Currently, it comes with one additional library which is libsmfilter.

7.1. Libsmfilter and Smfilter

Smfilter[8] is a preprocessor for Osmarender. It adds some sea chart specific virtual nodes and ways to simplify the rendering process. The functionality of smfilter is now integrated into *Smrender* with libsmfilter. As a result, smfilter is no longer supported. Libsmfilter exports two functions: vsector() and pchar(). The first is the replacement for smfilter the latter is a new function which generates combined strings for light descriptions.

7.1.1. Generating Light Sectors with vsector()

This function is a full replacement for smfilter. Smfilter took several options[9] to adjust the rendering behavior. These are the options **-a**, **-b**, **-d**, and **-r** in particular. Libsmfilter now takes exactly the same parameters since it is just a port. The parameters must be fed to it within the rules file. This was commonly described in Section 5.2.3. In detail the format looks like the following.

```
<param-str>   := <a-v-pair>[',' <a-v-pair>[',' ...]]
<a-v-pair>    := <attribute> '=' <value>
<attribute>   := 'a' | 'b' | 'd' | 'r'
<value>       := decimal number
```

a This sets the maximum distance of arc nodes. Basically, the distance is scaled with the radius; the smaller the radius the smaller the distance and vice versa. With large radii the distance of nodes is limited to dist. The value is given in nautical miles.

b Leading and directional lights are rendered with a bearing line and a small arc at its end. deg sets the angle of this arc to one side which means it is drawn def degrees clockwise and def degress counterclockwise from the bearing line.

d This defines the arc divisor which is used to determine the distance of the arc nodes. Thus, the node distance equals the radius divided by div. (see also option -a).

r I light may not have any radius specified since the tag is optional. In such cases smfilter picks radius as default value.

This is an example for calling vsector() from the rule set.

```
<node>
   <tag k='seamark:type' v=''/>
   <tag
      k='_action_'
      v='func:vsector@./libsmfilter.so?a=0.05,d=20,r=0.5'
   />
</node>
```

[8] See http://www.abenteuerland.at/smfilter/
[9] See http://www.abenteuerland.at/smfilter/smfilter.html

A full description of the output produced by vsector() is found in the smfilter(1) man page[10] and in the OSM wiki.[11]

7.1.2. Compatibility to Smfilter

The function vsector() does exactly the same as the original smfilter tool. Thus, they are considered to be nearly 100% compatible. The functionality is exactly the same but the file structure will still be different because *Smrender* processes the OSM file in a different way than smfilter.
The following shows two exchangable command lines, the first for smfilter and the second for *Smrender*.

```
smfilter -a 0.05 -d 20 -r 0.5 < in.osm > out.osm
```

```
smrender -i in.osm -o /dev/null -M -G -w out.osm
```

The following rules file has to be used in conjunction with *Smrender* to be a replacement for smfilter. Of course, the file may be extended with other rendering rules.

```
<?xml version='1.0' encoding='UTF-8'?>
<osm version='0.6'>
   <node>
      <tag k='seamark:type' v=''/>
      <tag k='_action_'
           v='func:vsector@./libsmfilter.so?a=0.05,d=20,r=0.5'/>
   </node>
</osm>
```

7.1.3. Generating Light Description Strings

Pchar() generates a string which contains the characteristics of the light as it is used in official sea charts and the *List of Lights*. See Section P and P.16 in particular if the *Chart No. 1*.[12]
The function analyzes the tags of an object. If it contains valid *OpenSeamap* tags[13] it generates the light string and adds the new OSM tag seamark:ligh_character=* to the object. The value of the tag contains the string which may be rendered by a subsequent rule.

7.1.4. Generating Circles around Depth Soundings

Libsmfilter provides the function sounding() which generates small circles around depth soundings.

8. Examples

There is a very simple example for your first rendered map. Before, download and compile *Smrender* as explained in Section A.

[10] http://www.abenteuerland.at/smfilter/smfilter.html
[11] http://wiki.openstreetmap.org/wiki/OpenSeaMap/smfilter
[12] http://www.nauticalcharts.noaa.gov/mcd/chart1/ChartNo1.pdf
[13] See http://wiki.openstreetmap.org/wiki/OpenSeaMap/Lights_Data_Model.

From the same download URL get the seamap icons package (icons.tbz2) and extract it within your *Smrender* source directory.

Now we have to get some OSM data. We just use the Overpass API[14] to get a small window.

```
wget -O cr.osm \
   'http://www.overpass-api.de/api/xapi?map?bbox=15,43.8,15.4,44'
```

Now we can start *Smrender* by using the rule set rules.osm which comes with the *Smrender* packacge. The following command line renders the OSM file cr.osm to the output image cr.png having the dimension of an A4 landscape page. 43.88 (N) 15.213 (E) are the center coordinates and the scale is chosen to be 1:75000.

```
./smrender -i cr.osm -o cr.png -P A4 -l -M 43.88:15.213:75000
```

9. Files

The *Smrender* package contains all source C files and headers. A Makefile is provided to build *Smrender* (see Section A). It contains all sources for the smfilter library (see Section 7.1) in the directory libsmfilter/ and a skeleton libary in the directory libskel/ which may be used as a starting point for own functions (see Section B).
The package contains futhermore two different rule sets which may also be used as a basis for own rule sets. Those files are called rules.osm and rulesbig.osm.

10. Author

Smrender is written by Bernhard R. Fischer. The idea of the project was born in summer of 2010. The actual development started in October of 2011.

11. Copyright

[14] http://wiki.openstreetmap.org/wiki/Overpass_API.

A. Compiling and Installing

Smrender should be simple to compile. Currently, it depends on the GD graphics library[15] which requires its development package to be installed. Typically the package is named libgd2-xpm-dev and can easily be installed with your package manager. On FreeBSD it is found in ports/graphics/gd.
Download the most recent *Smrender* package from `http://www.abenteuerland.at/download/smrender/` and extract the package with tar xvfj smrender-r*xxxx*.tbz2. Change into the newly extracted directory. Then just type make and it will compile. Finally, there is the executable smrender and the libsmfilter.so. The latter is not mandatory for running *Smrender* since it may just be loaded dynamically by the rule set (see Section 7.1).
Please note that there is currently no install target for make.

Smrender is known to compile with gcc 4.x on Debian Linux (Lenny and Squeeze) and FreeBSD version 8.x. It should compile on most Unixoid plattforms without further troubles, maybe even on Windows with Cygwin.

B. Writing Own Rendering Functions

The *Smrender* package includes a skeleton library in the directory libskel/. It implements the library constructor and destructor, and the actual rule function together with its initialization and de-initialization functions.
The directory contains also a Makefile which shows how to compile the library.

Smrender exports several functions which may be called by the library. The following list shows the most imported ones. The prototypes are defined in smrender.h, smlog.h, or osm_inplace.h.

```
// Use smrender's standard logging. This function is defined in 'smlog.h' and
// works similar to syslog(3).
void log_msg(int, const char*, ...);

// Get an OSM object (OSM_NODE, OSM_WAY, OSM_REL) with the specifed id. This
// function returns a pointer to either an osm_node_t or osm_way_t or_osm_rel_t
// structure on success, or NULL on error.
void *get_object(int, int64_t);

// Add an OSM object to the memory. The function returns 0 on success,
// otherwise -1 is returned. Preexisting objects with the same id are simply
// overwritten.
int put_object(osm_obj_t*);

// These functions return unique ids for nodes and ways.
int64_t unique_node_id(void);
int64_t unique_way_id(void);

// Initialize an OSM object. The number of tags (type short) and the number of
// node references (type int) must be supplied. Currently, the functions always
// return a valid pointer to an object. The objects returned are just partially
// initialized (see 'osm_func.c').
osm_node_t *malloc_node(short);
osm_way_t *malloc_way(short, int);
```

[15] `http://www.boutell.com/gd/`

C. FAQ

This section covers some questions and answer which might arise.

C.1. Why is Smrender not written in C++?

On closer examination, the software architecture suggests an object-oriented programming language such as C++ but *Smrender* is written in C. The short answer is that C is always my first choice and the code was already too mature to switch to C++ without a high effort. The long answer is that *Smrender* is able to dynamically link libraries at runtime. Interfacing from C++ to a library written in C (currently) seems to be difficult (although not impossible).
But I still have in mind to rewrite *Smrender* in C++ when time comes.

D. Todo

- The simple Makefile will be replaced by a configuration by the GNU autotools.
- The GD library will be replaced by the more powerful *Cairo* library.[16]
- The action format will be replaced by a more flexible CSS-style format.
- Rendering of rotated (not North-up) maps.
- Dynamic rules; these are rules which are generated during the rendering itself and are applied subsequently.

[16] `http://cairographics.org/`