Towards a more readable Openstreetmap based world map for westerners

Sven Geggus

Motivation

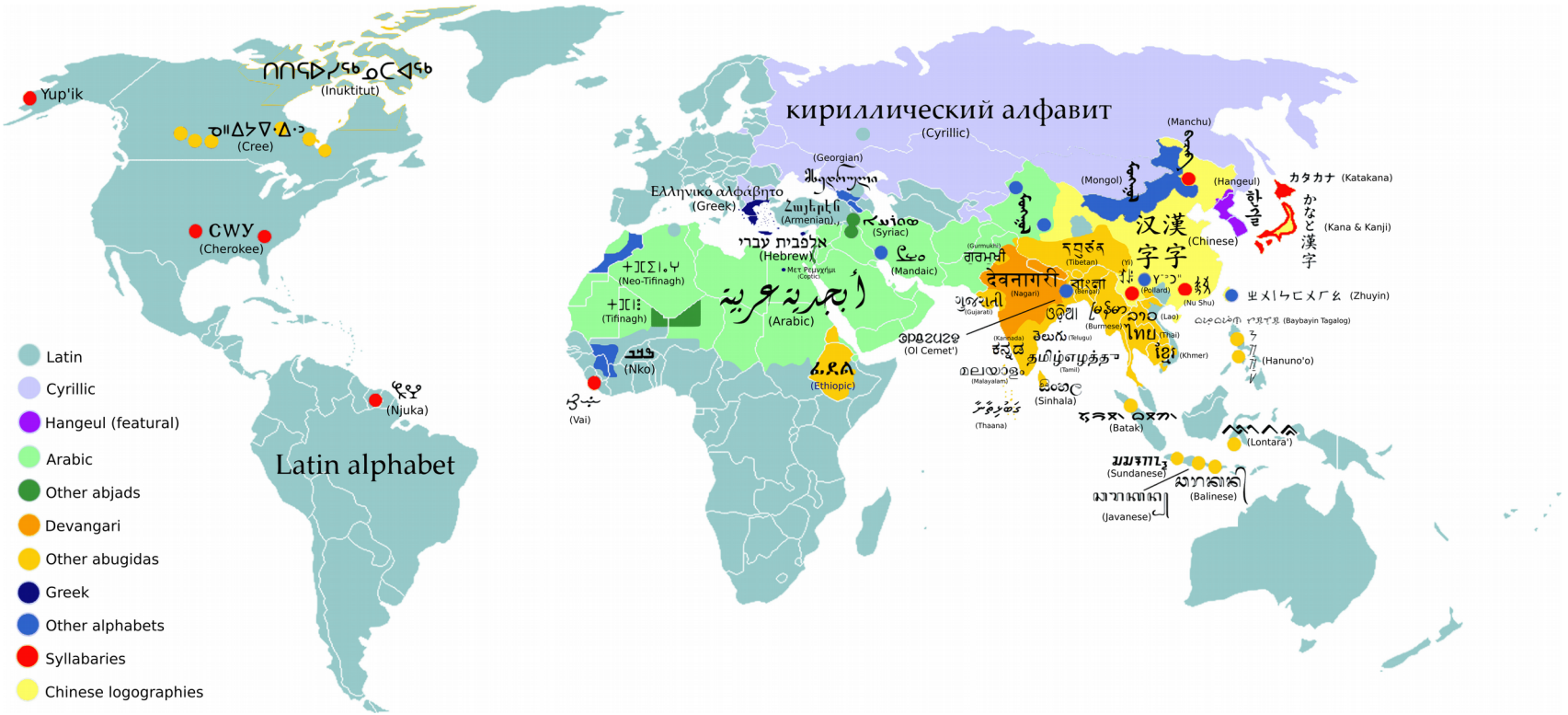
- Looking at areas where Latin script is not the norm Openstreetmap based maps are mostly unreadable for westerners.
- The reason for this is our rule, that names of geographical objects are to be acquired in local language.
- In contrast to conventional spatial dataset Openstreetmap data does often contain additional localized data, which is unfortunately not yet often used when rendering maps.

Localized objects contained in Openstreetmap data

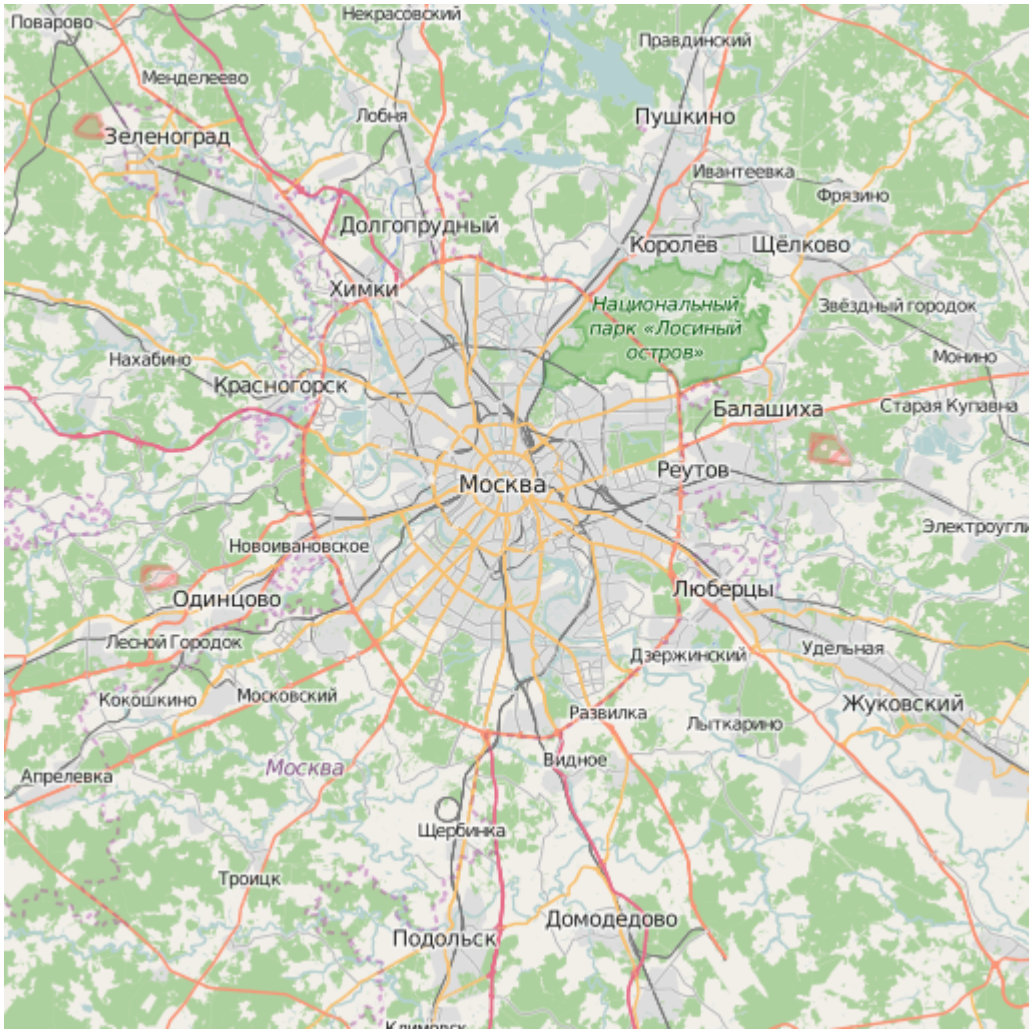
name	=> Deutschland
...	
int_name	=> Deutschland
name:de	=> Deutschland
name:en	=> Germany
name:ar	=> ألمانيا
name:ja	=> ドイツ
name:ru	=> Германия
name:he	=> גרמניה

name	=> ישראל
...	
int_name	=> Israel
name:de	=> Israel
name:en	=> Israel
name:ar	=> إسرائيل
name:ja	=> イスラエル
name:ru	=> Израиль
name:he	=> ישראל

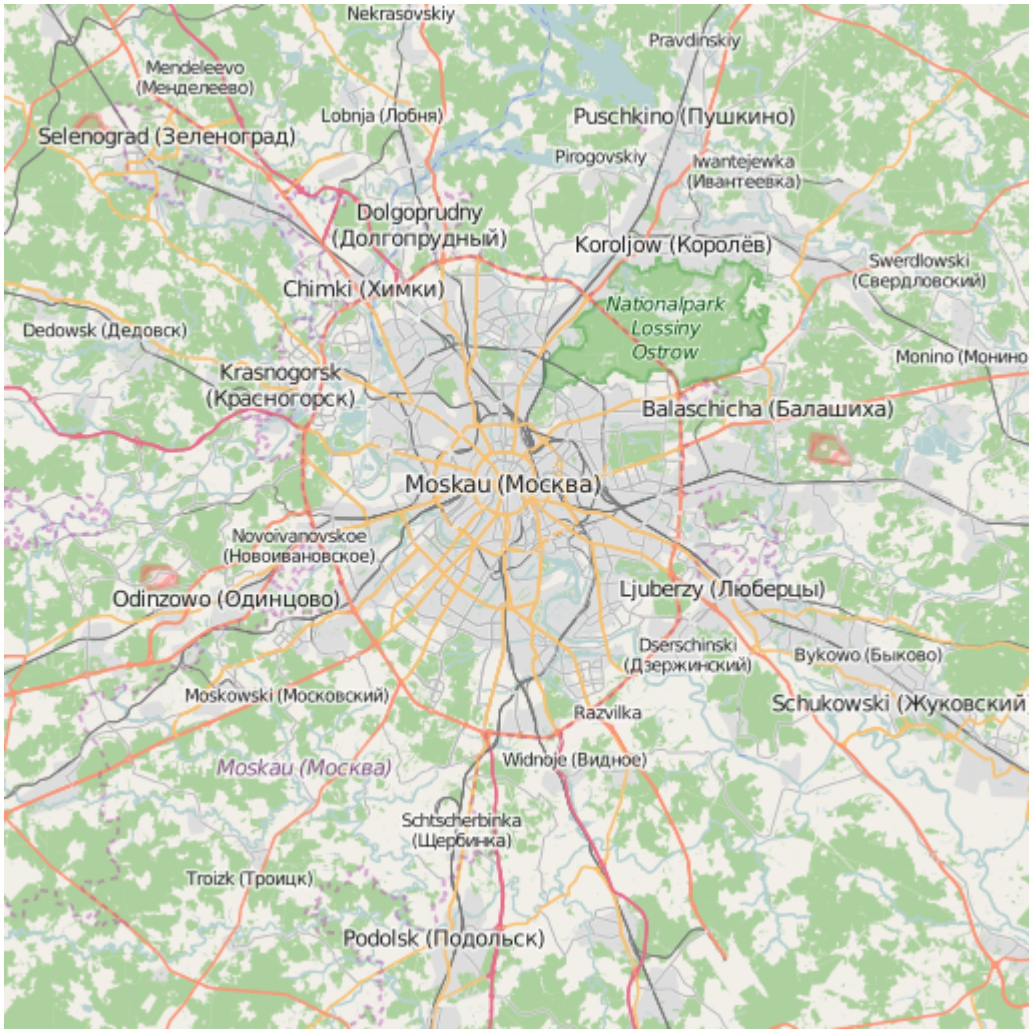
Writing systems of the world



Openstreetmap Carto Style (original)



Openstreetmap Carto Style (with German localization)



Fraunhofer IOSB

Main objective

- Making the map readable for westerners by using Latin script.
- Use localized data from Openstreetmap itself whenever possible.
- Use other localization methods like transcription or transliteration, if the objects from Openstreetmap do not contain localized data.

Approach using PostgreSQL „stored procedures“

Advantage:

- Localization is renderer independent. It is possible to use any render supporting PostgreSQL as a data-source (Mapnik, Mapserver, Geoserver, …).

Disadvantage:

- Localization functions are only available if PostgreSQL is used as a data-source. It is impossible to localize other data sources like Shapefiles etc..

Implementation

- PL/pgSQL functions available are

```
osml10n_get_placename
osml10n_get_streetname
osml10n_get_name_without_brackets
```

- Despite the fact that this is currently used for German language only, the target language (available as a function parameter) might be any language using Latin script.
- A convenient way is to hide this functions behind database views. This will create virtual tables which will enable usage of existing cartographic styles with minimal changes (using another column for “name”).

Implementation

Making the decision which name to use:

- If data does contain the name of the target language (e.g. name:de in German language) use it.
- If name is written in Latin script, just use it.
- Otherwise use the international name (int_name)
- or English name (name:en) as an alternative
- If none of the above is true use transcription.

Transcription and transliteration

- Transliteration means a reversible, character by character conversion of other alphabets into Latin.
- Transcription shall allow a non mother-tongue speaker a reasonable pronunciation and is not necessarily reversible.

⇒ **Transcription is what we need!**

Classes of writing systems and transcription

- Alphabets (e.g. Latin, Greek, Cyrillic, Arabic, …)
  - Easy transcription
- Syllabaries (e.g. Kana)
  - Relatively easy transcription
- Logographic writing systems (e.g. Chinese)
  - Transcription is only possible by language
- Hybrid forms (e.g. Thai, Hangeul, …)
  - Easy transcription

Known problems of transcription

- Being a logographic alphabet, the transcription of Chinese characters has to be based on the place of the geographical objects. Using PostGIS we are able to determine the country where an object is located. This way it has been possible to implement a solution for this problem in Japan.
- Thailand usually uses Royal Thai General System of Transcription (RTGS) on road signs etc. Unfortunately the ICU library uses ISO11940 which is not widely used. I don't know of a FOSS library implementing RTGS.
- Some writing systems (Arabic, Hebrew) do not add all vocals to written words. Instead they are added by the reader. Transliteration is therefore often incomplete.
  Example: Transliteration of تهران (Teheran) using ICU gives „thrạn“

Current implementation of Transcription

- Usage of the free library *International Components for Unicode* (ICU) which is providing a *Any-Latin* transliteration.
- A PostgreSQL stored procedure (`osml10n_translit`) providing this function from ICU has been implemented.
- Place dependent use of other transcription libraries:
  - Currently transcription of Chinese characters (called kanji in Japanese) is performed by KAKASI library if the object to be rendered is located in Japan.
  - Extendable to other writing systems and countries
- Usage of other transcription libraries by writing system (not place) can be added easily.

Using different glyphs inside a single label

Issue:

- There is no single font which contains all glyphs in persistent good quality. In our multilingual map this is a problem in cases where local names are to appear in parenthesis, because renderers are usually unable to change a font inside a single label.
- As a compromise the current code will render local names in parenthesis only if labels use Latin, Greek or Cyrillic character sets.

Possible resolution:

- It might be possible to produce a free “best-of” font using good quality glyphs from various sources.
- It might be possible to enhance renderers to use different fonts based on the character set of the glyph which is about to get rendered.

Political problems in map localization

- Many regions of the world have been part of other countries in the past. Examples are former colonies or German settlement areas in eastern Europe.
- As an example even the smallest villages in Poland, Alsace and Lorraine have German names. However, often nobody knows if they are still in widespread use today. In a worst-case scenario, their usage will offend people.
- Du to the lack of an alternative we hope, that mappers will only acquire names which are still used (or use old_name respectively).
- Currently our map style is using a compromise:
  Always use the current local name in parenthesis.
  Example: Stettin (Szczecin)

Prospect and possible enhancements

- Technical solution of the problem to render glyphs from different writing systems in a single label.
- Addition of more and/or better suited libraries for transcription
- More fine grained distinction of transcription algorithms by place and/or writing system.
- Implement street abbreviation code for all common languages (we currently have German, English, Russian and Ukrainian).
- Add suggestions from the audience (especially from native speakers)

Code: https://github.com/giggls/mapnik-german-l10n

Please send pull requests :)