How To Build Tile Server Using OpenStreetMap tools and data

Andrii Mishkovskyi

March 2011

Outline

What will you learn?

- Cartography basics
- How to render maps with Mapnik
- How to build basic map service with Flask

First of all

- Grab the CD or flash with all the data from me
- Use materials at content.mishkovskyi.net/pycon2011
- Or use handouts

Outline

Terms

Map Graphic representation of the geographical setting

Cartography Science and practice of making maps

GIS A system that collects, analyzes, manages or processes in any other way data linked to location.

Applications of GIS

- Geography
- Cartography
- Navigation
- Search engines
- Remote sensing, land surveying, urban planning ...

Characteristics of maps

- Reduction

Characteristics of maps

- Reduction $\Longrightarrow$ Scale

Characteristics of maps

- Reduction $\Longrightarrow$ Scale
- Transformation

Characteristics of maps

- Reduction $\Longrightarrow$ Scale
- Transformation $\Longrightarrow$ Map projection

Characteristics of maps

- Reduction $\Longrightarrow$ Scale
- Transformation $\Longrightarrow$ Map projection
- Abstractions

Characteristics of maps

- Reduction $\Longrightarrow$ Scale
- Transformation $\Longrightarrow$ Map projection
- Abstractions $\Longrightarrow$ Symbolism

Classification of maps

- By scale – small-scale world, large-scale your neighborhood
- By function – general reference, thematic, charts
- By subject matter – cadastre, plans

Outline

Projection

Conversion of spherical information into 2d pane

Characteristics of projection

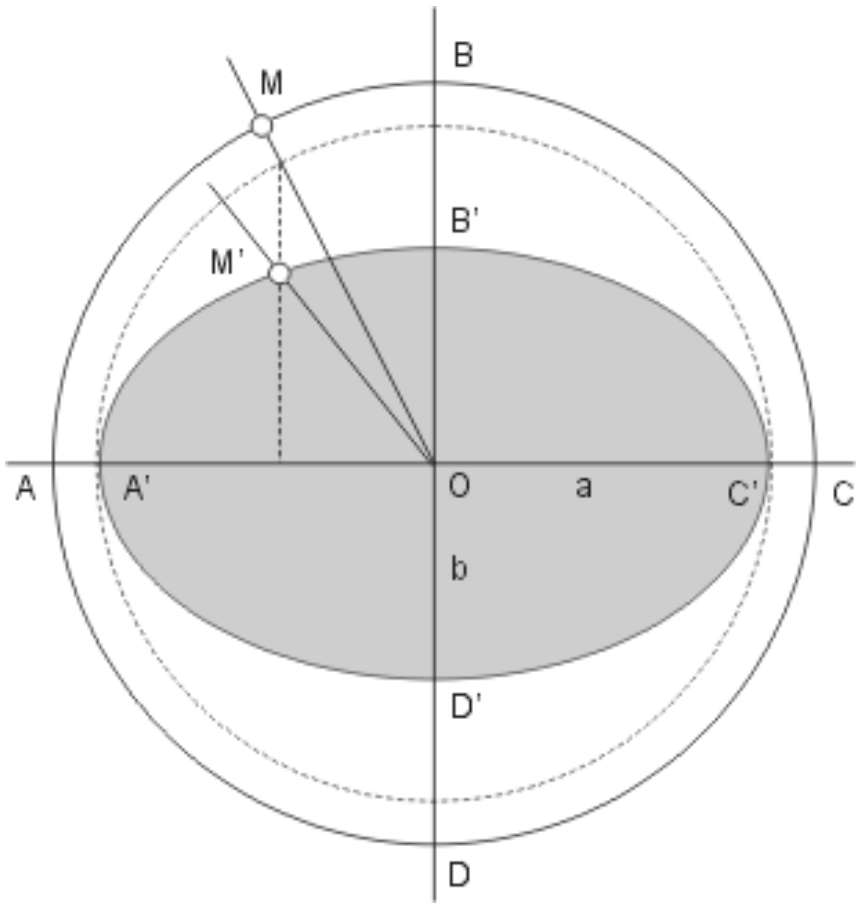
- Linear distortion (scale factor)
- Area distortion
- Angle distortion
- Distance distortion

Types of projections by surface

- Conical
- Cylindrical
- Conformal
-

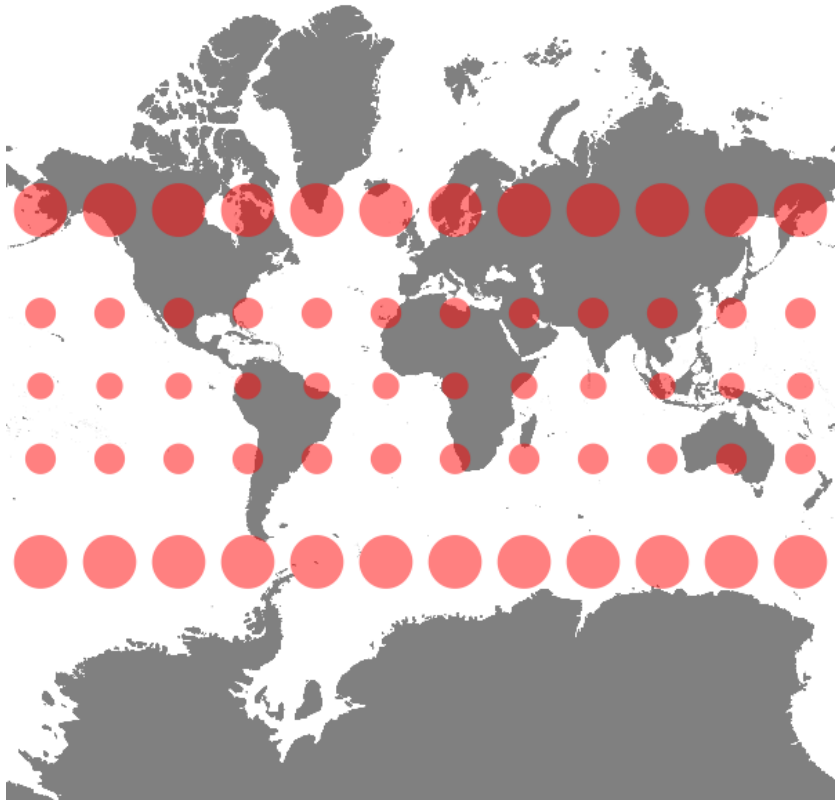
Tissot indicatrix

Representation of map distortions



Mercator projection

Tissot's indicatrix



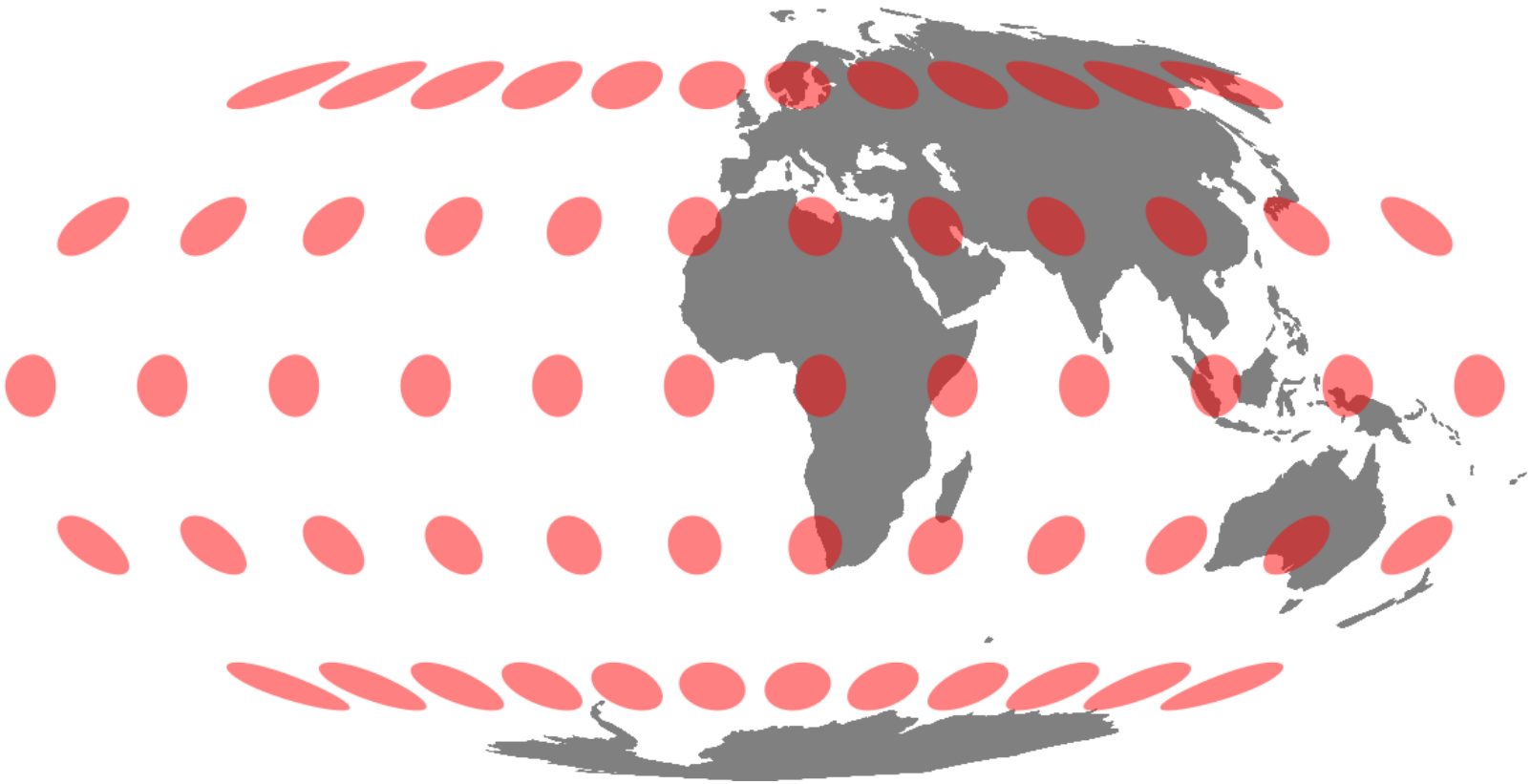
Mercator projection

Description

- Conformal cylindrical projection
- No angle distortion
- Major area and linear distortion around poles
- Historically used for navigation

Mollweide Projection

Tissot's indicatrix



Mollweide Projection

Description

- Pseudocylindrical equal-area projection
- Serious linear and angle distortion
- Useful where area size is important (global distributions, visual area size comparison, etc)

Outline

Classes of data

- Physical (Topographical – elevations, terrain & water objects)
- Cultural
- Human-made

Sources of data

- Mapping companies (TeleAtlas, Navteq, Ordnance Survey)
- Public sources (OpenStreetMap, Natural Earth)
- Government institutions (NASA, local government)
- Personal data (GPS tracks, social networks)

Collecting data

- Surveying
- GPS tracks
- Tracing aerial imagery

Storing data

- RDBMS (PostgreSQL + PostGIS, MySQL Spatial, Oracle Spatial, Spatialite)
- Non-relational databases (Neo4j, MongoDB, CouchDB)
- Flat files (WKB, WKT, GeoJSON, KML, GML)

Simple Feature acess: SQL

- Full set of querying functions
- Tons of useful features for on-the-fly projecting etc.
- Almost fully supported by PostGIS

Other standards

- Well-known text/binary
- Keyhole Markup Language
- Geography Markup Language
- Esri Shapefiles

How OSM handles data

Special XML format used in

- Official API
- Official plant extracts
- Unofficial eXtended API (XAPI)

OSM data primitives

Node Stands for one geospatial point.

Way

Relation

OSM data primitives

Node

Way Collection of nodes. Can be used to represent such geometric features as lines (roads, borders, etc) and polygons (parks, buildings, etc.).

Relation

OSM data primitives

Node

Way

Relation Collection of other primitives (including relations themselves). Used for storing complex relationships between geospatial objects.

How OSM XML looks like

```
<osm version=''0.6''>
  <node></node>
  <way></way>
  <relation></relation>
</osm>
```

Nodes

```
<node id="16122" lat="12.1021" lon="-72.341">
  <tag k="tourism" v="hotel" />
  <tag k="name" v="Hyatt Regency" />
</node>
```

Ways

```
<way id="11231">
  <nd ref="17281" />
  <nd ref="17282" />
  <nd ref="17283" />
  <nd ref="17284" />
  <nd ref="17281" />
  <tag k="addr:housenumber" v="265" />
  <tag k="tourism" v="hotel" />
  <tag k="name" v="Hyatt Regency" />
  <tag k="building" v="yes" />
</way>
```

Relations

```
<relation id="8472">
  <tag k="type" v="associatedStreet" />
  <tag k="name" v="Peachtree Street" />
  <member type="way" ref="11231"
          role="house" />
  <member type="way" ref="314171"
          role="street" />
</relation>
```

Parsing OSM data

- Osmosis
- osm2pgsql
- imposm
- tons of little scripts

osm2pgsql

Imports data to PostgreSQL with the following table setup

planet_osm_point

planet_osm_line

planet_osm_polygon

osm2pgsql

planet_osm_point Stores features which are represented with single symbol, such as bus stops, hotels (if mapped as points), ATMs, etc.

planet_osm_line

planet_osm_polygon

osm2pgsql

planet_osm_point

planet_osm_line Almost 90% is take by different types of roads.

planet_osm_polygon

osm2pgsql

- planet_osm_point
- planet_osm_line
- planet_osm_polygon Buildings, parks, etc.

Fetching data from PostgreSQL

```
$ psql gis

gis=> SELECT ST_AsText(way)
FROM planet_osm_point
WHERE "name" LIKE 'Hyatt%';
                        st_astext
-----------------------------------------------------
 POINT(-9481012.96880667 3941017.3930227)
 POINT(-9399929.12076633 3983988.17286583)
 POINT(-9341209.61445808 4159288.58929291)
 POINT(-9027126.42457084 3774062.70553924)
(4 rows)
```

Limiting search by bounding box

```
SELECT ST_AsWKT(way) FROM planet_osm_point WHI
```

[language=SQL]

More silly examples

```
contains
buffer
```

[language=SQL]

What is Mapnik

Map rendering framework, written in C++ with built-in Python bindings.

Mapnik concepts

Symbolizer Drawing rule of given geometric feature

Rule

Style

Datasource

Layer

Map

Mapnik concepts

Symbolizer

Rule Collection of symbolizers for various data inputs.

Style

Datasource

Layer

Map

Mapnik concepts

Symbolizer

Rule

Style Collection of rules. Exists mostly of convenience for defining layers.

Datasource

Layer

Map

Mapnik concepts

Symbolizer

Rule

Style

Datasource Source of geospatial data.

Layer

Map

Mapnik concepts

- Symbolizer
  - Rule
  - Style
- Datasource
- Layer Link between datasource and styles
- Map

Mapnik concepts

- Symbolizer
  - Rule
  - Style
- Datasource
  - Layer
    - Map Abstraction over links between styles and layers

Outline

Rendering map

```
from mapnik import (PolygonSymbolizer, LineSym
                    Rule, Style, Color,
                    Layer, Shapefile,
                    Map, Image, render)
```

Rendering map

```
rule = Rule()
rule.symbols.append(PolygonSymbolizer(Color("g
rule.symbols.append(LineSymbolizer(Color("blac
```

Rendering map

```
style = Style()
style.rules.append(rule)
```

Rendering map

```
layer = Layer('world')
layer.datasource = Shapefile(file='coastlines/
layer.styles.append('world')
```

Rendering map

```
m = Map(800, 400)
m.background = Color('white')
m.append_style('world', style)
m.layers.append(layer)
```

Rendering map

```
m.zoom_to_box(layer.envelope())
image = Image(800, 400)
render(m, image)
with open('test.png', 'w') as image_file:
    image_file.write(image.tostring('png'))
```

Loading map file

```
from mapnik import (Map, load_map,
                    Image, render)

map = Map(600, 600)
load_map(map, "simple.xml")
```

Loading map file

```
m.zoom_to_box(layer.envelope())
image = Image(800, 400)
render(m, image)
with open('test.png', 'w') as image_file:
    image_file.write(image.tostring('png'))
```

Creating map file

```
<?xml version="1.0" encoding="UTF-8"?>
<Map 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 +nadgrids=@null
          +no_defs +over"
     bgcolor="#98bcda">

</Map>
```

Creating map file

```
<Style name="world">
  <Rule>
    <PolygonSymbolizer>
      <CssParameter name="fill">black
      </CssParameter>
    </PolygonSymbolizer>
    <LineSymbolizer>
      <CssParameter name="stroke">grey
      </CssParameter>
      <CssParameter name="stroke-width">0.1
      </CssParameter>
    </LineSymbolizer>
  </Rule>
</Style>
```

Creating map file

```
<Layer name="world"
       srs="+proj=longlat +ellps=WGS84
            +datum=WGS84 +no_defs">
  <StyleName>world</StyleName>
  <Datasource>
    <Parameter name="type">shape
    </Parameter>
    <Parameter name="file">coastlines/land
    </Parameter>
  </Datasource>
</Layer>
```

Outline

Hello world API

http://localhost:5000/map

```
from flask import Flask, Response
from utils import render
import mapnik

map = mapnik.Map(0, 0)
mapnik.load_map(map, "map.xml")
app = Flask(__name__)
```

Hello world API

http://localhost:5000/map

```
@app.route('/map')
def simple():
    bbox = mapnik.Envelope(
        mapnik.Coord(-20037508.34,
                     -20037508.34),
        mapnik.Coord(20037508.34,
                     20037508.34))
    return Response(
        render(map, bbox, 600, 600),
        content_type='image/png')
```

Hello world API

http://localhost:5000/map

```
def render(map, bbox, width, height):
    """Render a map within a given bounding bo
    map.resize(width, height)
    map.zoom_to_box(bbox)
    image = mapnik.Image(width, height)
    mapnik.render(map, image)
    return image.tostring('png')
```

Outline

Tile API

http://localhost:5000/0/0/0.png

```
from flask import Flask, Response
from utils import render, parse_coords
import mapnik

map = mapnik.Map(0, 0)
mapnik.load_map(map, "map.xml")
app = Flask(__name__)
```

Tile API

http://localhost:5000/0/0/0.png

```
@app.route('/<int:zoom>/<int:xtile>'
           '/<int:ytile>.png')
def tile(zoom, xtile, ytile, image_type):
    metasize = 1 if zoom < 2 else 2
    bbox = parse_coords(256, metasize,
                        zoom, xtile, ytile)
    return Response(
        render(map, bbox, 256, 256),
        content_type='image/png')
```

Tile API

http://localhost:5000/0/0/0.png

```
def parse_coords(tile_size, meta_size, zoom, x
    """Transform tile offsets into queriable
    coords = [(x, y), (x + meta_size, y + meta
    coords = [[offset * float(tile_size) for
              for coord in coords]
    coords = [pixel2latlon(coord, zoom, tile_s
    coords = [mapnik.Coord(*coord) for coord
    coords = [projection.forward(coord) for co
    return mapnik.Envelope(*coords)
```

Tile API

http://localhost:5000/0/0/0.png

```
def pixel2latlon(coord, zoom, tile_size=256):
    """Convert coordinate of pixel point to la
    x, y = coord
    tile_size = float(tile_size)
    e = tile_size * 2**(zoom - 1)
    g = ((e - y) * 2 * math.pi /
         (tile_size * 2**zoom))
    lat = ((x - e) /
           ((tile_size * 2**zoom) / 360.0))
    lon = math.degrees((2 * math.atan(math.exp
                       - (0.5 * math.pi))
    return lat, lon
```