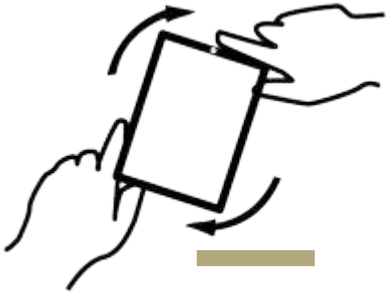
Adding rotation to Leaflet.draw

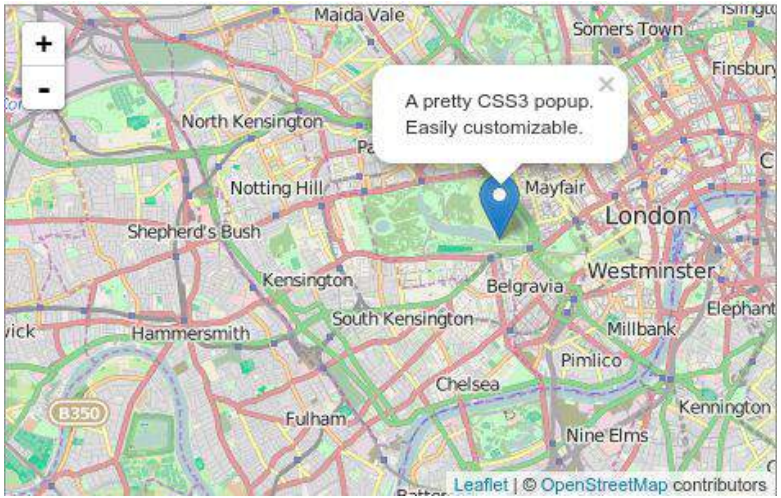


FOSS4G Belgium 2015
Jan De Moerloose Geosparc

GEOSPARC

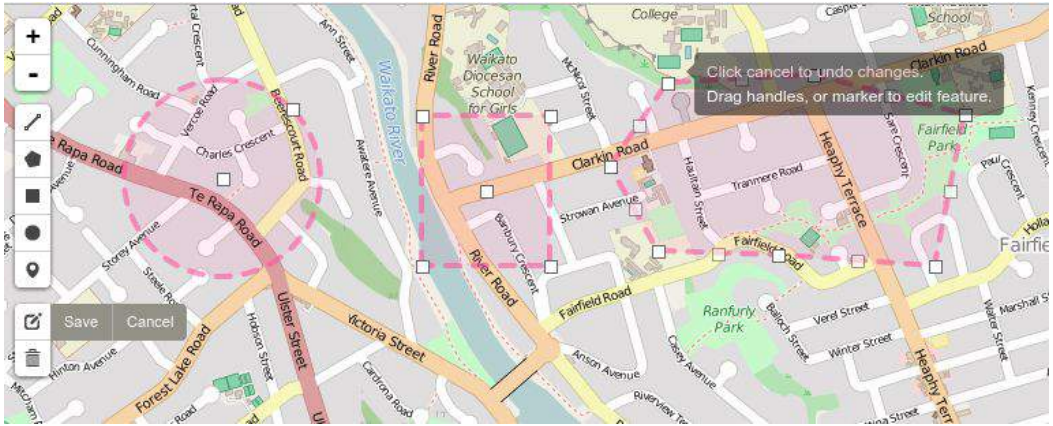
Leaflet

- JavaScript library for interactive maps
- Popular alternative to OpenLayers
- Mobile-friendly
- Small and simple
- Lots of plugins



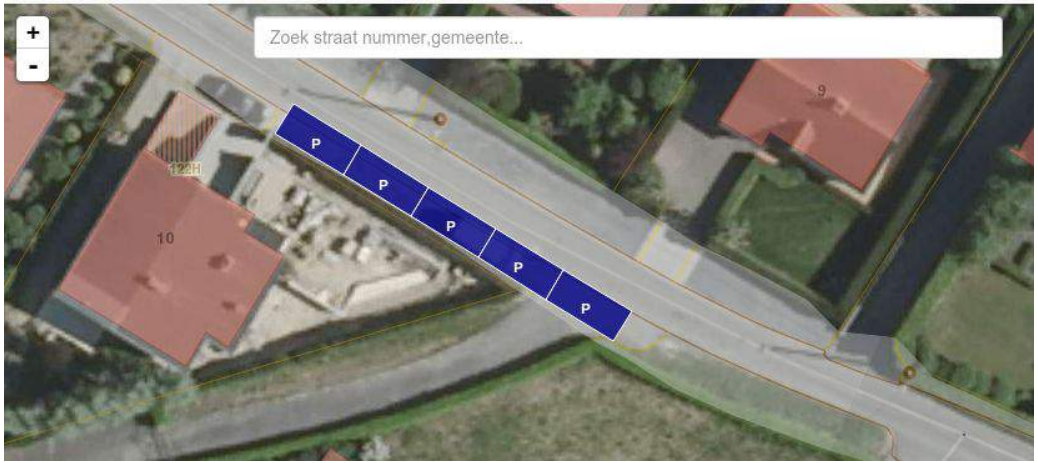
Leaflet.draw

- Leaflet plugin
- For drawing geometries
- Supports drawing point/line/polygon shapes
- Supports dragging shapes
- Supports vertex editing



Why rotation ?

- Next thing to do after dragging
- Sounded easy, just one more affine transformation :-)
- We needed it, for this:

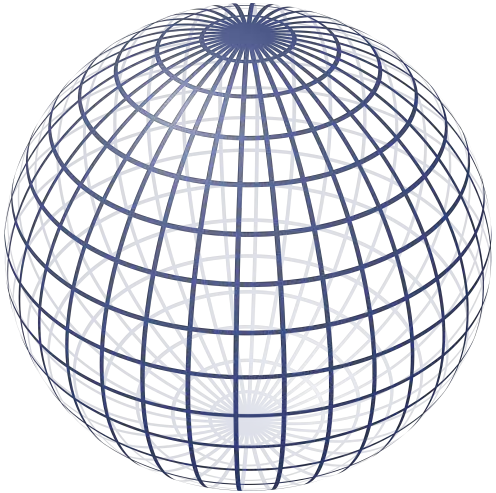


Philosophical question: what does it mean to rotate shapes on a map ?

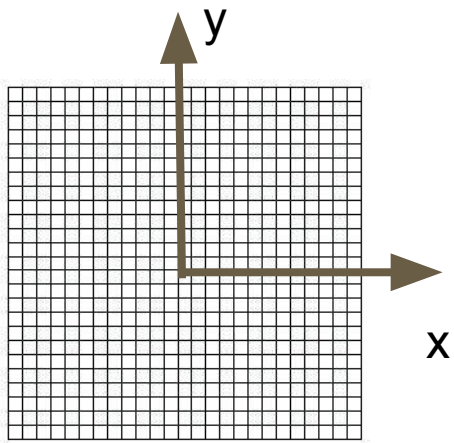
- Maps are 2D, but earth is 3D
- Dragging/translation and rotation are well-defined for flatlanders
- ...but not so much on the earth's surface !
- How to drag along a straight line ? How to choose the rotation axis ?
- But wait...
- ...it all makes sense on the 2D map !
- ...or does it ?

Coordinate systems and projections

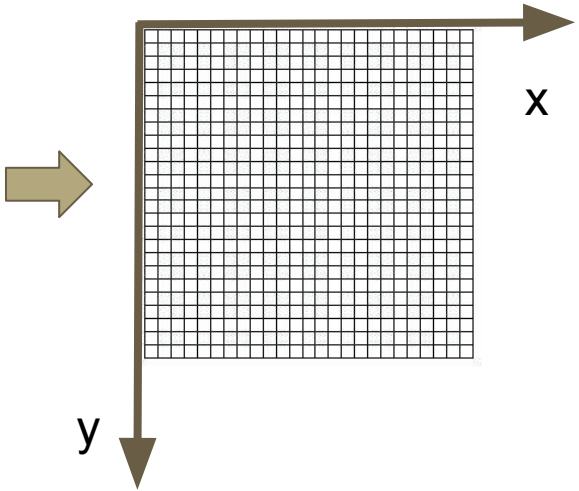
World coordinates
(Lat/Lon)



Projected
coordinates

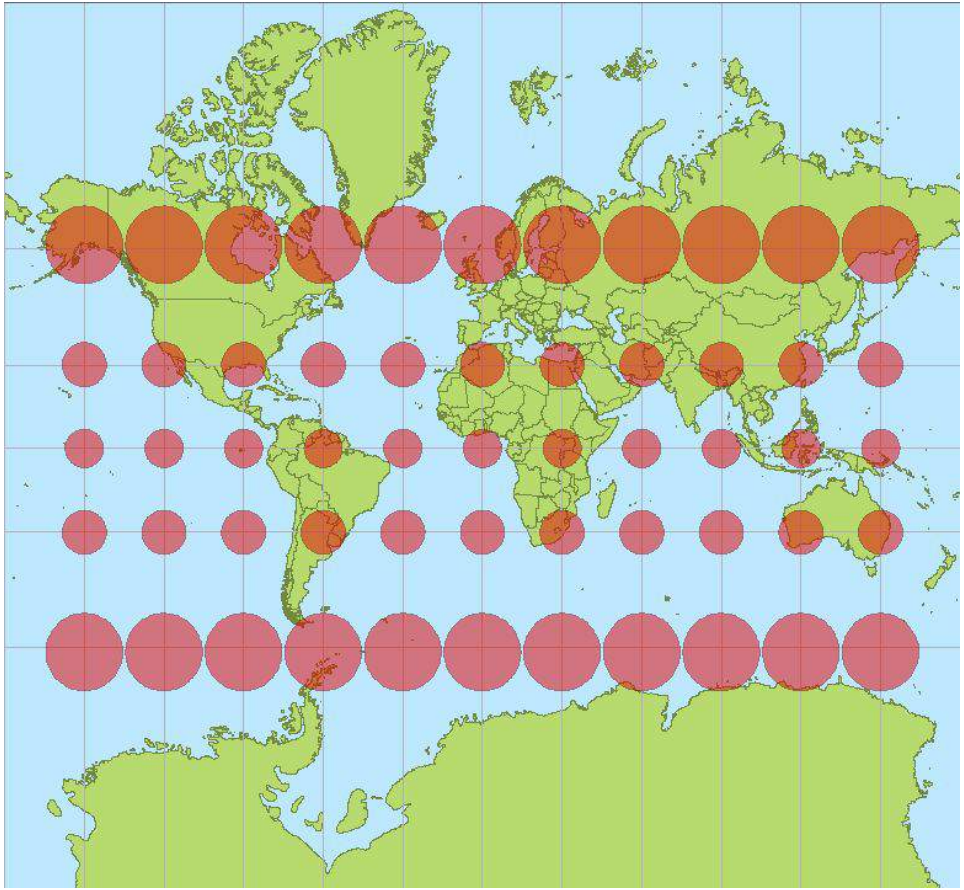


Screen or
map container or
image coordinates



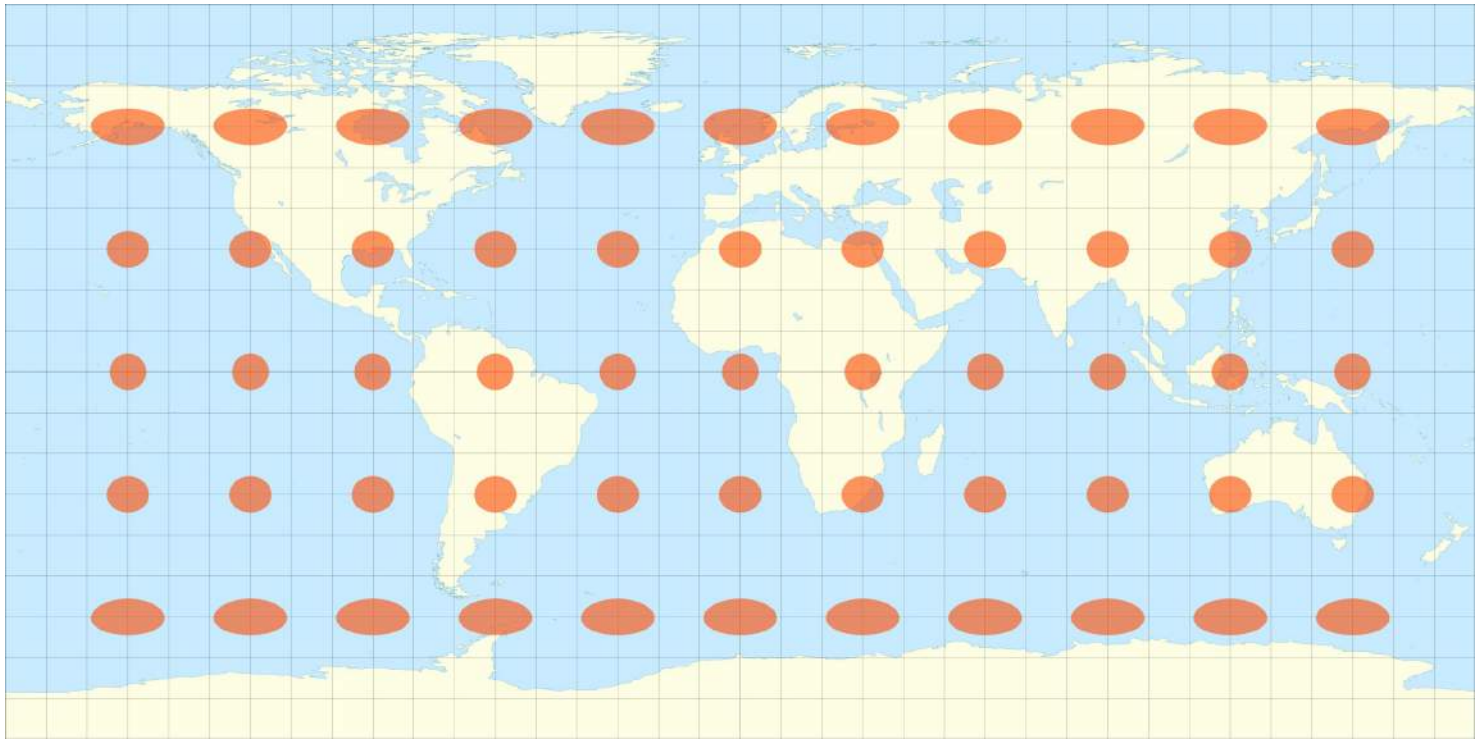
Example: Web Mercator Projection

- Conformal projection: preserves shape if you look at small areas (circles stay circles)
- As a consequence: scale grows when moving to higher latitudes (same circle on the sphere becomes bigger on the map)
- Which means that dragging a shape to the north makes it smaller in reality !



Example: Plate Carée Projection

- Simply use Longitude/latitude as x/y
- Not a conformal projection: circles become ellipses
- Which means rotating a (projected) shape on the map changes its form/shape in reality !



To sum it up

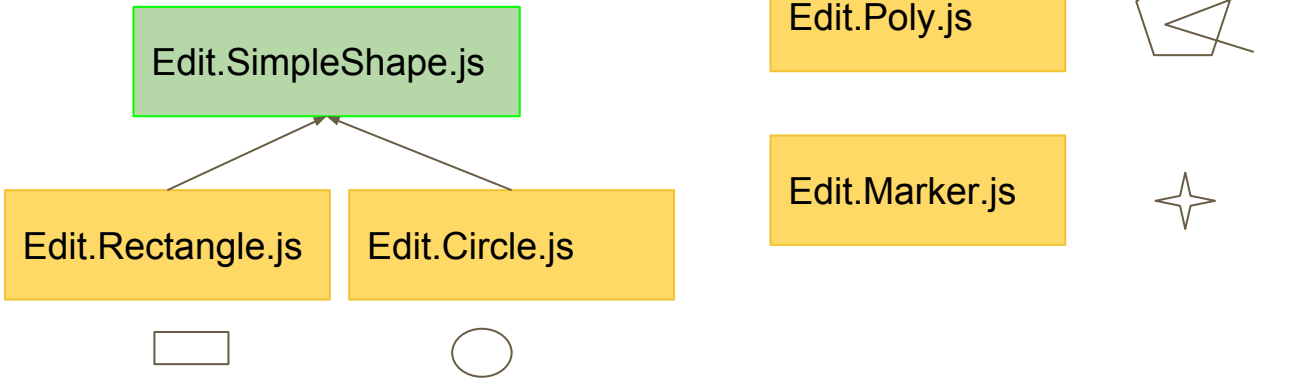
While transformations should happen in projected space to not confuse the end user...

This only makes sense when:

1. The projection is conformal (rotation)
2. The geometries are sufficiently small as to neglect the projection error (say less than 1km)

Editing class hierarchy

- Code split up between drawing (creating new geometries) and editing (updating existing)
- Editing classes:



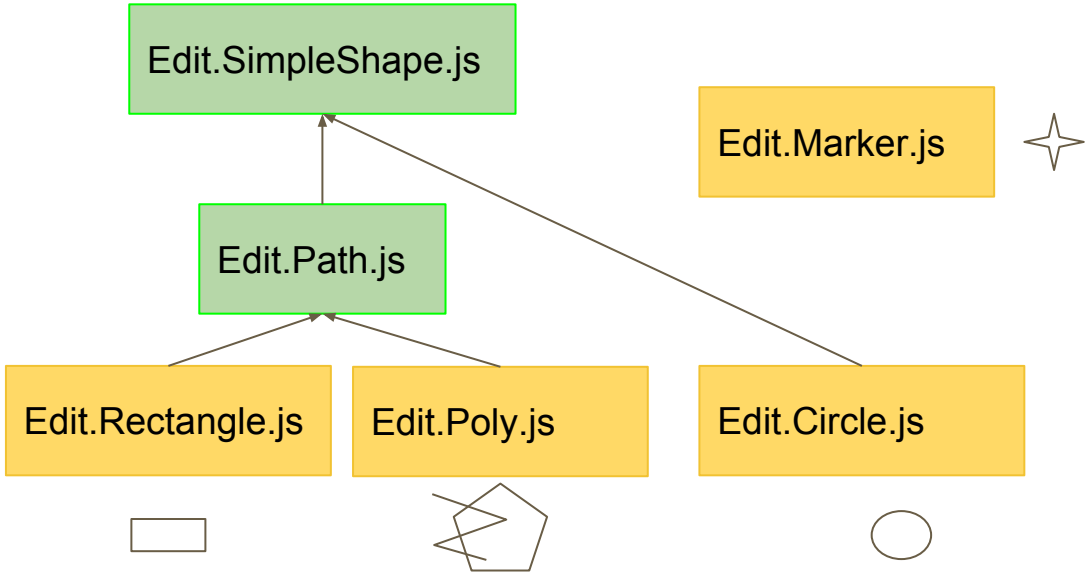
Problem #1

- Rectangle/Polygon/Polyline are all path-based
- Rectangle is just a special kind of Polygon
- Which suggests equal treatment for transformations
- But current code treats Rectangle differently (because of its own specific resize operation)

Solution

- Common treatment of all path-based shapes
- Implement a resize operation that works for all of them
- -> revisit the class hierarchy

New editing class hierarchy



New operations

- **Resize**: changing the bounds of a shape by dragging the boundary corners
- **Rotate**: rotating a shape by means of a rotate handle

On top of existing:

- Move: moving a shape
- Edge drag: moving the vertices of a polygon/polyline

Problem #2

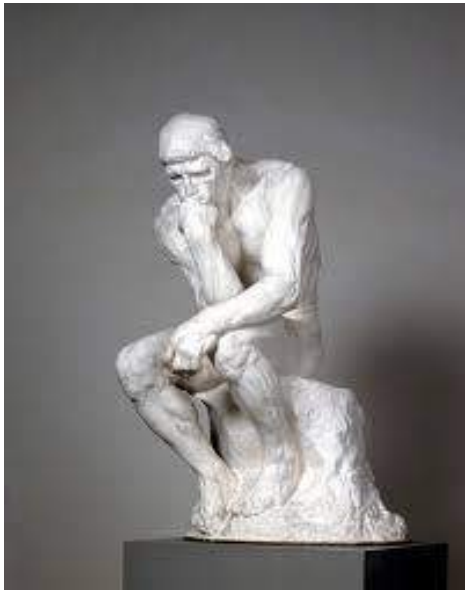
- All operations influence each other
- All operations use some form of affine transformation
- Currently operations have different execution paths

Solution

- Streamline the code for each transformation

    1. draw the markers (handles)
    2. register mouse event callbacks
    3. create the transformation (project/transform/unproject)
    4. transform the shape (and redraw)
    5. reposition all markers

Affine transformations



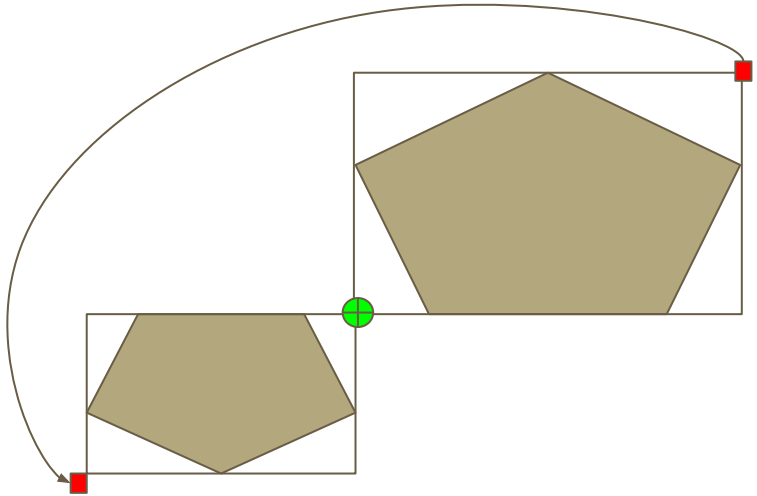
$$\begin{bmatrix} X_{scaled} \\ Y_{scaled} \\ 1 \end{bmatrix} = \begin{bmatrix} Scale_x & 0 & 0 \\ 0 & Scale_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} X_{translated} \\ Y_{translated} \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & D_x \\ 0 & 1 & D_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} X_{rotated} \\ Y_{rotated} \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix}$$

The resize operation

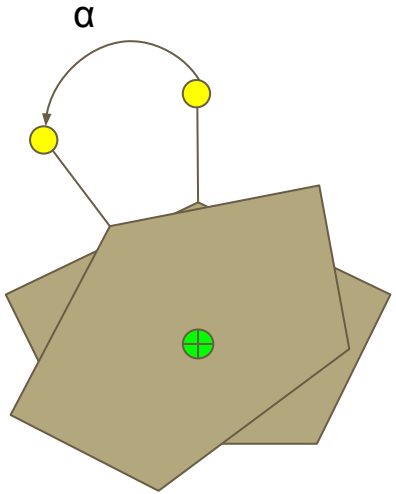
1. Translate opposite corner to origin
2. Scale sx/sy
3. Translate back



resize handle
opposite corner

The rotate operation

1. Translate center to origin
2. Rotate α
3. Translate back

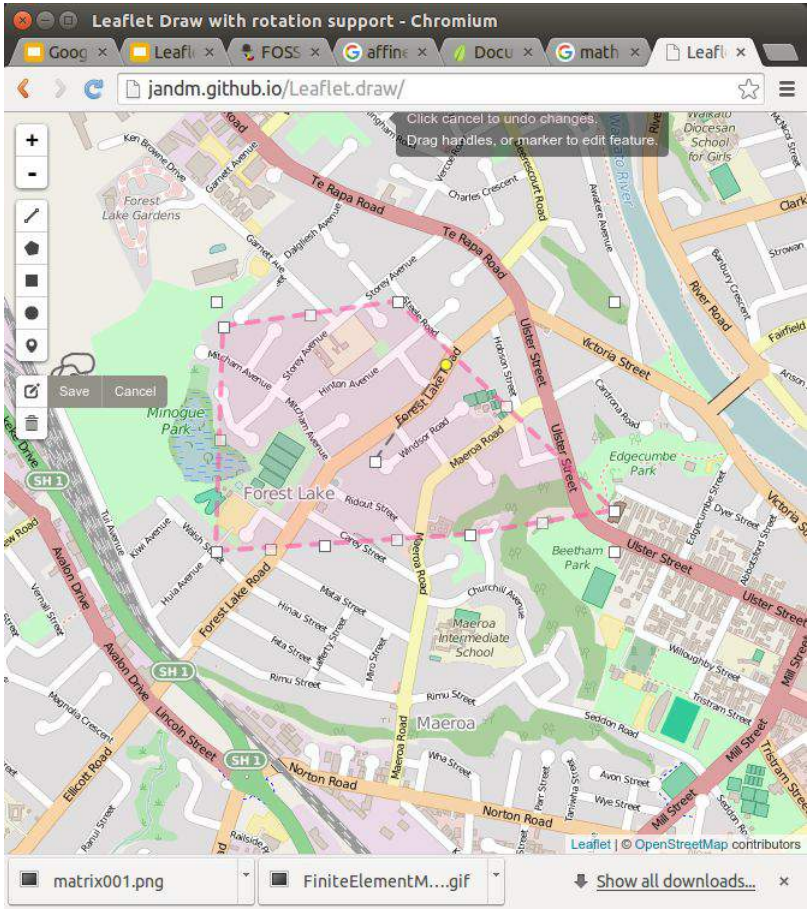


rotation handle
center

Helper classes

- AffineTransform.js: matrix transformations
- LineMarker.js: a fixed size line (for the rotation handle)
- MarkerExt.js: a displaced marker (for the rotation handle)

Demo time


Leaflet Draw with rotation support - Chromium
jandm.github.io/Leaflet.draw/
Click cancel to undo changes.
Drag handles, or marker to edit feature.
Save
Cancel
Te Rapa Road
Ulster Street
Victoria Street
River Road
Forest Lake
Minogue Park
Edgecumbe Park
Beetham Park
Maeroa Intermediate School
Maeroa
Forest Lake Road
Maeroa Road
Avalon Drive
Lincoln Street
Norton Road
Mill Street
Tristram Street
SH 1
Leaflet | © OpenStreetMap contributors
matrix001.png
FiniteElementM....gif
Show all downloads...

Questions ?

Code and demo page: https://github.com/jandm/Leaflet.draw

Mail me:

jan.demoerloose@geosparc.com