HANDS-ON WORKSHOP TOOLS FOR SPATIAL DATA VISUALIZATION: Using selected Open Source tools and Open Data to visualize your own spatial data

Barend Köbben

Version 1.0
December 12, 2012

Contents

Key points

This is the exercise description for the hands-on workshop "Concepts and tools for Spatial Data Visualization ", part of the SENSE PhD day, organized by the SENSE Research Cluster XIII.

! → In many cases during exercises, you will have to type code (HTML, Python, JavaScript or MapServer configuration code). It's very easy to make mistakes in such code. Some code (e.g., HTML and MapServer map files) are not case–sensitive, but others (e.g. JavaScript) is: the variable `mySomething` is different from the variable `MySomeThing`! Also take care of the *special character* (→) in the code examples we provide:

```
here is some code that should all be typed on 1 line in your→
 file but is divided over 2 lines in our example...
```

→ this character means you **should not** type a `return` or `enter`→ in this place. The line should be typed **without interruption**, the move to the next line in our example is only because it would not fit otherwise.

Typing the code of longer listings is not always necessary: For longer code fragments, we provide a text file in the `filefragments` folder in the exercise data. In this folder you will find code fragments from these exercises in text files with the same names as the listing title.

There are several software tools that can help you: Use a text–editor that is more intelligent than your basic text editor, e.g. on MacOS use *TextWrangler*, on Windows *Notepad++*, which is (at the time of writing) available in ITC on the `P:\` drive. This will provide you with line numbers, automatic highlighting of recognised HTML and JavaScript keywords, etcetera.

For a web–browser, FireFox (with the add-on FireBug installed), or Chrome and its debugger are good choices. This gives you useful error messages, code views of HTML, CSS and a JavaScript console, network traffic monitoring, etc...

1 Open Data: using OpenStreetMap

First we'll show you a prime source of free maps and data on the web: OpenStreetMap.

> **Note:** The OpenStreetMap Project, based at openstreetmap.org, is the worldwide mapping effort that includes more than 400,000 volunteers around the globe. OpenStreetMap is an initiative to create and provide free geographic data, such as street maps, to anyone.

There are many ways in which you can access the OpenStreetMap: as a simple webmapping service (not unlike Google and Bing Maps, but based on truly free non-proprietary data on a non-commercial website), as a webservice in various gis–viewers and as a database service, providing the actual vector data in raw form.

Task 1 : Visit http://www.openstreetmap.org/ using a web browser. Try to find the ITC building (it's just North–West of the main station in downtown Enschede, a town in the East of The Netherlands)... •

The OpenStreetMap site itself uses the OpenLayers Javascript API, just as we will do later ourselves. The icons you see in the map are the default Graphics User Interface (GUI) of OpenLayers. They offer the following interactivity:

- You can *pan* using the arrow icons, or by dragging the map;
- You can *zoom in* using the + icon, or shift–drag a zoom box in the map;
- You can *zoom out* using the – icon;

Task 2 : Try setting up the map in such a way, that it starts zoomed in on the **Aamsveen Nature Reserve**... •

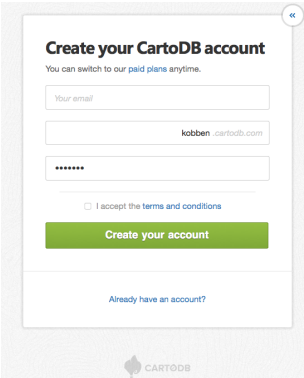
> **Note:** The Aamsveen Nature Reserve is a 130 ha large area just South-East of Enschede in the Netherlands. It comprises of the last remnants of a very large peat area which in 1976 was turned into a protected nature reserve. Stichting Het Overijssels Landschap is responsible for the maintenance and protection. See http://www.landschapoverijssel.nl/aamsveen for more information (in Dutch).

Using the OpenStreetMap site as described above is fine for casual map browsing, but what if we want to have more control over the

content and looks of our map? There are many ways to reach that goal, in this workshop we will use two quite different solutions:

1. Using an on-line map creation and data visualization application, called **CartoDB**;
2. Creating our own webpage with interactive maps, using the OpenLayers javascript API, and using OpenStreetMap as a reference. We will also add some new data and using QGIS, a free Open Source GIS.

2 The CartoDB web application



CartoDB is an interactive web site that you can use to visualize your spatial data on the web. On their website (http://cartodb.com) they describe it as follows: "CartoDB is a product of Vizzuality, a data visualization consulting company, and was first created to meet the companyńs own internal needs to map large and complex geospatial data sets. While using the platform for various internal projects we realized its full usefulness and potential and decided to make it open source".

On the site there is else a small intro movie: http://vimeo.com/vizzuality/introducing-cartodb

CartoDB is free to use for testing and small projects (using up to 5 data tables and 5Mb of data storage), but you will need to sign up for it:

Task 3 : Go to the website and click the "Sign up now" button. Put in your details and create your account. •

2.1 Creating a data table

We will first have to import some data into the system. We will use vegetation data for the Aamsveen Nature Reserve. The data can be found in your data directory in a file `vegemap.geojson`, and the data directory also includes a small document with *metadata* for this dataset.

Task 4 : Click the "Create you first table" button. Click "select a file" and browse to the file `vegemap.geojson`. The file is imported into the PostgreSQL/PostGIS database on the CartoDB servers... •

> **Note:** PostgeSQL/PostGIS is a popular Open Source *spatial database back–end* that stores the spatial data using the Open Geospatial Consortium Simple Features specifications. As a platform, the object–relational DBMS PostgreSQL is a solid DBMS that has a reasonably gentle learning curve, yet is very appropriate for advanced database applications, and its documentation is very transparent. PostGIS in addition, is the leading open standards implementation of spatial vector management, and enjoys a lively and supportive user/developer community.

You can now use this data in several ways. Because it is stored in a database, you can use the tools to do selections, editing and even spatial analysis. If you are familiar with the SQL query language, you can use the small "SQL" tab in the right-hand side to directly work with the data in SQL.

2.2 Creating a map

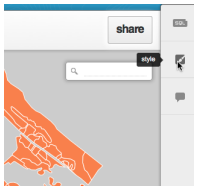
But of course, you can also show the data as a map:

Task 5 : Click the "MAP VIEW" tab. You see the data shown on top of a general background map. You can change which background map to use with the various icons at the top. The layers called "MapBox" are visualization made by the CartoDB people on the basis of OpenStreetMap data.

Now try to zoom out so that the whole Aamsveen dataset is shown, on a a neutral grey background (instead of a map)... •

Note that if you click anywhere in your map, a Info Window will pop up with the attribute data of the polygon you clicked on. You can even edit the spatial data (in a limited way), but we will not use that now.

2.3 Changing the map visualization



You can click on the "Style" icon (on the right-hand side of the screen) to change the way the data is mapped. There is a "Style Wizard" that you can use to generate three basic types of thematic maps:

1. **Simple:** This is a simple type where all elements (in our case polygons) are mapped the same. You can use the menus below to set Polygon Fill and Stroke colors, as well as their transparencies. You can also choose to label the elements with the content of one of the attribute fields;
2. **Choropleth:** A "choropleth map" depicts relative ratio values in different colour *values*. Colour values for cartographers are *shades* of a colour: E.g. ranging from white to black, or from light red to dark red.
3. **Bubbles:** This is what cartographers call a "proportional point symbol map", where you proportionally scale symbols (in this case circles) with some numerical attribute.

Task 6 : Experiment with the different map types. Try to understand the possibilities and limitations of the Style Wizard interface... •

Note that you cannot use the Wizard to map the attribute "name", the one holding the names of the simplified vegetation classes. This is because the wizard only allows mapping of numerical values, as they are the only ones suitable for the map types "choropleth" and "bubbles". But fortunately, we can also use the more powerful CartoCSS styling: Advanced styling with CartoCSS uses the concept of Cascading Style Sheets (a W3C standard for styling in webpages) to style the maps. The CartoCSS language is an easy, flexible, and powerful way to making a better looking map. If you know how to use CSS to style websites, you already know how to use CartoCSS. We will show an example of its use in the next task:

Task 7 : First use the Style Wizard to map the attribute "class_id3" using the "Choropleth" style, using the maximum seven colours (called "buckets" in CartoDB). Now click the "CartoCSS" tab. •

You now see how the pre-defined styles in the Style Wizard are also in the end stored as CartoCSS. And now you can edit the style to make it more appropriate for our goals:

Task 8 : We are going to create a so-called "chorochromatic map", a map that uses different colors for different nominal classes. Nominal classes are attributes that differ only in quality, or name, but cannot be ordered, or numerically expressed. So the first thing to do is change the remark in the first line from "choropleth visualization" to "chorochromatic visualization".

The next few lines define the **vegemap** class. CSS is a hierarchical styling system, so all items of class #vegemap will use this class. They therefore will have a white line-color (#FFF means full red, full green and full blue, which together renders as white), the lines will be fully opaque, with a width of 1 pixel, and the polygon fill will be slightly transparent (because its opacity has been set at 80 percent).

The next lines add to the general #vegemap class by adding rules for more specific subclasses, specified by the expression $class\_id3 <=$

111. Whenever a polygon is encountered that complies to this expression, it will be given a polygon fill of colour #B10026, which is a mix of darkish red and light blue. We can now change such a class by editing the text. Make sure to edit it such that it looks like the Listing below, and check the results in the map. •

Listing 1: The CartoCSS after a first edit

```
/** chorochromatic map visualization */

#vegemap{
  line-color: #FFF;
  line-opacity: 1;
  line-width: 1;
  polygon-opacity: 0.8;
}
#vegemap [ name = 'Molinia'] {
   polygon-fill: red;
}
```

You now should see a map with most polygons in grey, and some (the ones with attribute 'name' being 'Molinia') in red. The reason the others are grey is because for those attributes,no polygon fill colour was specified, and therefore a default was used. You can change that by specifying a "polygon-fill" in the general #vegemap class. Note also we have specified the Molinia fill colour as "red". CSS has a lot of pre-set colour names that actually are translated to RGB colors. If we would have specified #FF0000 we would have seen exactly the same result.

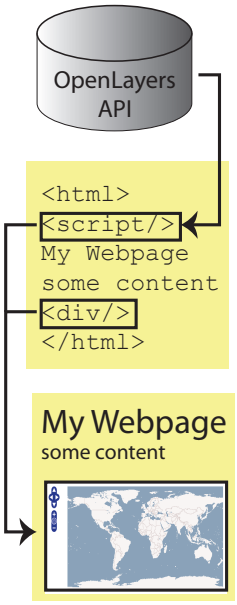
Task 9 : Now finish the map by adding subclasses for all vegetation names. Take some time to create classes with colours that are easy to distinguish... •

---

3 The OpenLayers API

OpenLayers makes it easy to put a dynamic map in any web page. It can display map tiles and markers loaded from any source.

> **Note:** OpenLayers is a pure JavaScript library for displaying map data in most modern web browsers, with no server-side dependencies. OpenLayers implements a JavaScript API (Application Programming Interface) for building rich web-based geographic applications, similar to the Google Maps, with one important difference: OpenLayers is Free Software, developed for and by the Open Source software community based at `http://openlayers.org/`. OpenLayers is written **in object-oriented JavaScript**, using components from *Prototype.js* and the *Rico* library. In these exercises, we will only show the basic building blocks, and how to employ them. Those wanting to go further, should check out the development pages and the examples at the website.



The latest version of the OpenLayers script library is always available on the OpenLayers website. You can "install" the API by including a link to the Javascript files in your own HTML webpages and then call the methods and properties of the library using simple JavaScript functions. Using the Openlayers API is done by creating *webpages* (using HTML) that include Javascript ***script***; this code makes calls to the API methods to create the necessary map object and connect that to an HTML *placeholder*. Mostly we use an HTML `<div>` element as a placeholder.

The OpenLayers API has two concepts which are important to understand in order to build your first map: *Map*, and *Layer*. An OpenLayers *Map* stores information about the default projection, extents, units, and so on of the map. Inside the map, data is displayed via *Layer*s. A *Layer* is a data source – information about how OpenLayers should request data and display it. We then uses the methods and properties of the API to change the content and behaviour of the map. In practice, all this means typing (and/or copying) HTML and JavaScript code.

4 Making an OpenLayers viewer for OpenStreetMap

In listing 2 you see the most basic example of using OpenLayers with the OpenStreetMap service.

Task 10 : Create an HTML page with the content of listing 2 and save it as a `osm.html`. You can type the code, but it is easier to copy it from the file we stored in the `filefragments` folder. Do **not** copy from this PDF file! Make sure you save this file as a new

file with the extension `.html`, *not* `.html.txt`!). View the result in a web browser. •

Listing 2: osm.html

```
<html><head>                                                    html header
   <title>OpenLayers Basic Single OSM Example</title>           html title
   <script src="http://openlayers.org/api/OpenLayers.js"></script>   include the API
   <script type="text/javascript">                              script for our map
   var myMap, myOSMLayer;                                       define map and layer object
   var myCenter = new OpenLayers.LonLat(                        define center
     254031,6254016                                             XY of Paris
     );
   function init() {                                            function triggered on load
     myMap = new OpenLayers.Map("mapDiv");                      create map object
     myOSMLayer = new OpenLayers.Layer.OSM("OSM Map");          create OSM layer
     myMap.addLayers([myOSMLayer]);                             add layer to map
     myMap.setCenter(myCenter,16);                              zoom to center
   }
   </script>
 </head>
 <body onload="init()">                                         run init script
 <div id="mapDiv"                                               map placeholder
    style="width:400px; height:400px;"></div>                   placeholder style
 </body></html>
```

The result should look like figure 1, showing the OpenStreetMap for the Porte Maillot area in Paris (France).

You can set up the OpenStreetMap to start at any place on the globe, by changing the coordinates that were used in the `myCenter` variable:

```
var myCenter = new OpenLayers.LonLat(254031,6254016);
```

But in order to find which coordinates to use to zoom to, it would be nice to have a knowledge of where (in coordinates) you are in the map. For that we will include a coordinate–readout line and a scale bar:

Task 11 : Add the following line in the script **just before** the line with the `myMap.SetCenter` command:

```
myMap.addControl(new OpenLayers.Control.MousePosition());→
```

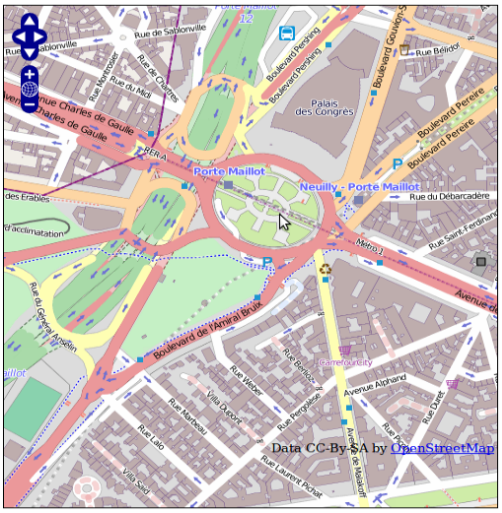


Figure 1: result of loading listing 2.

```
myMap.addControl(new OpenLayers.Control.ScaleLine());
```

Save the results as `osmPlusCoordinates.html`. Try out the result in the browser. •

The coordinates you see are X– and Y–coordinates in a Mercator projection on the spherical WGS84 datum. This is used nowadays by most popular public webmapping services (such as Google Maps, Bing Maps and OpenStreetMap). The projection is officially standardized as **EPSG code 3857**, and named "WGS 84 / Pseudo-Mercator". Unfortunately, lots of software uses instead the un-official EPSG code 900913 (chosen because it sort of spells "google"), that was introduced and has become popular before the official standard was set.

Now you can change the line `myMap.setCenter(myCenter,16)` to set an alternative starting point (change `myCenter` variable) and zoom (from 0–18) for the map.

Task 12 : Try setting up the map in such a way, that it starts zoomed in on the **Aamsveen Nature Reserve**... •

---

5 Using QGIS to create KML data

5.1 Introduction of QGIS

QGIS (officially QuantumGIS, http://qgis.org) is an Open Source, stand-alone GIS client, programmed in C++ using the multi-platform Qt framework. You can use it to work with vector– and raster–files, databases or any open standard WMS or WFS–compliant server. A strong point of QGIS is its extensibility: you can add plug–ins that are written in either C++ or Python, and you can connect it to GRASS, a powerful GIS analysis tool.

Task 13 : ITC users can just double–click (or copy) the shortcut that they find at `P:\QuantumGIS\Quantum GIS` to their own computer.

Other users go to the URL http://qgis.org/ and download and install the latest release version (i.e. a stable version, not one in development) for their operating system. •

QGIS can load maps and data from a huge array of possible sources:

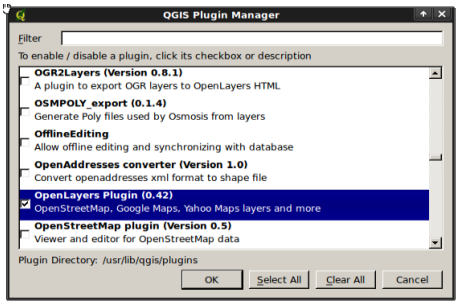
- online maps served as an OGC-compliant Web Map Service (WMS);
- online spatial data served as an OGC-compliant Web Feature Service (WFS) and Web Coverage Services (WCS);
- various other map services, such as OpenStreetMap, Google Maps, Bing Maps, etcetera;
- most vector formats supported by the OGR library, including ESRI shapefiles, MapInfo, KML, GPX and GML;
- raster formats supported by the GDAL library, such as digital elevation models, aerial photography or satellite imagery;
- spatially-enabled PostgreSQL tables using PostGIS and SpatiaLite, by means of a 'live' connection to such databases;
- locations and mapsets from GRASS (an open source GIS);

The list is in principle endless, because the functionality of QGIS can be extended by *plugins*. Plugins add functionality to QGIS, and they are usually made by others than the main QGIS developers.

Because QGIS is an Open Source software, anyone can add plugins, they can be programmed using C++ or Python.

5.2 Using QGIS to create your own data

QGIS is not limited to loading data from services and files, it also allows you to create data in many of these formats, and that is what we are going to do next. We will use an existing background map from the OpenStreetMap webmapping service to digitize some of our own data...



Task 14 : Start QGIS. To enable the use of OpenStreetMap, we will use a plug–in that offers that functionality.

First check if the OpenLayers plugin is already installed: Choose `Plugins > Manage Plugins...`, and type "openlayers" in the Filter box. If the plugin is available, it will be listed below. You can go on tho the next Task.

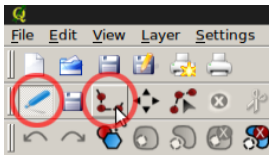
If not, open `Plugins > Fetch Python Plugins...`, go to the tab `Repositories` and check the list. Now go to the tab `Plugins` and find the one called "OpenLayers plugin" and enable it by making sure it is selected. If there are several versions, use the latest one. Press OK. •

Now let us use the plugin to have OpenStreetMap in the background:

Task 15 : Open the menu `Plugins` again. An item called `OpenLayers Plugin` should be available. This plugin offers access to many publicly available map services (but despite its confusing name it does *not* offer access to the OpenLayers Javascript API!) Choose `Add OpenStreetMap layer` from this submenu, and the OpenStreetMap will be opened, zoomed out on the whole world.

Navigate to the location you set for OpenLayers webpage you made earlier. •

We will create a very simple vector line dataset, that depicts a walking route. When creating new data, QGIS will adopt the current projection of the map window. Therefore we must make sure that the data is saved correctly projected, otherwise our new layer won't fit the OpenStreetMap base later on:



Task 16 : Choose the menu `Layer > New > New Shapefile→ Layer...` The New Layer dialog opens (see below). Make the following settings:

1. For `Type` choose `Line`;
2. Click `Specify CRS` and make sure you choose the Google Mercator (EPSG:900913). It can be found under `Projected → Coordinate Systems > Mercator`;
3. In the `New attribute` section, create one attribute of type `Text`, call it for example routeName. Click the `Add to → attribute list` button to actually add it;
4. Click OK to create the new file. Save it named "myRoutes.shp".

Now you can start adding lines for your route:

Click first the `Toggle Editing` button in the QGIS menubar, then the `Capture Line` button. Create a nice walking route from the Enschede main station to the ITC building. You can add points to the line by clicking in the map, undo them by using the `CTRL-Z` key or `Edit > Undo` menu. If you have finished a route, right-click, fill in the route name and press OK. Use the `Toggle Editing` button again to stop editing. You can change the visualisation of the line by right-clicking the layer name in the layers list, or choosing the `Layer > Properties` menu •

5.3 Using QGIS to save KML data

Note: KML is an XML notation for expressing geographic annotation and visualization within Internet–based, two–dimensional maps and three–dimensional Earth browsers. KML was developed for use with Google Earth, which was originally named Keyhole Earth Viewer. The KML file specifies a set of features (place marks, images, polygons, 3D models, textual descriptions, etc.), and locations are always expressed in longitude and a latitude. KML shares some of its structural grammar with GML. KML files are very often distributed in KMZ files, which are zipped versions of the file, with a `.kmz` extension.

KML is now an international standard of the Open Geospatial Consortium, and can be used in many geo-browsers and GIS software.

It is simple to save a QGIS layer to the Keyhole Markup Language format:

Task 17 : Right-click on the layer in the Layers list or choose

the menu `Layer > Save as...` In the Format menu, choose "Keyhole Markup Language (KML)", choose a place and name ("myRoutes.kml") and save the file by clicking OK. •

5.4 Testing and refining the KML

The KML that you have created can be tested in many environments. Most GIS software nowadays can import (and export) KML. But the most logical test platform is probably the software that the KML format was engineered for: *Google Earth*.

Task 18 : Start Google Earth. Choose the `File > Open...` menu and find the file 'myRoutes.kml' you created earlier. The file should be loaded and Google Earth zooms to the place on earth where the KML is placed. •

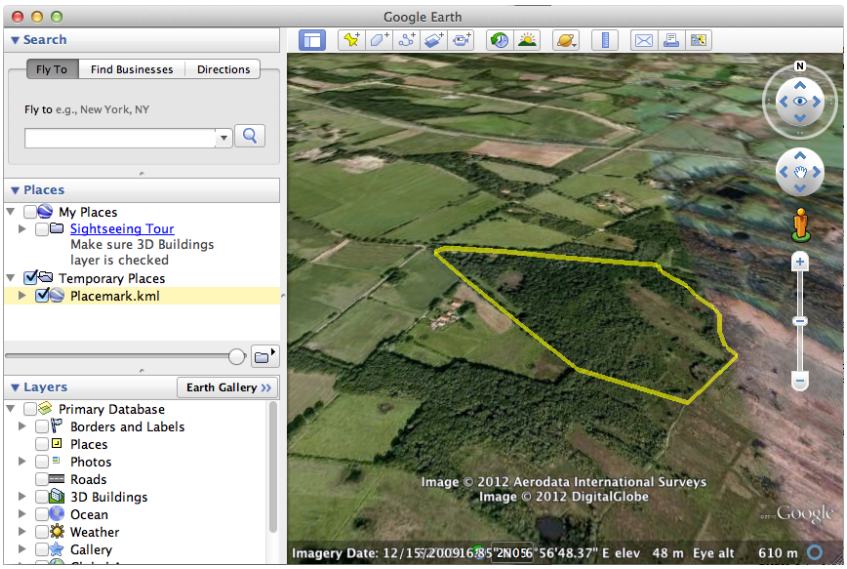


Figure 2: result of loading the route KML in Google Earth.

You can also use Google Earth to refine the visualization of the KML. You can right-click on the name of the KML file in the Google Earth 'Places' panel and use the `Get Info` menu to edit things like the description, the style (color and width of the line) the starting view, etcetera... If you made changes, do not forget to roght-click again and use the `Save Place As...` menu!

5.5 Alternative solutions for creating KML

In this section, we have used QGIS to create a vector layer and export it to a web–based format useable in OpenLayers. There are many alternatives for this. ESRI's ArcMAP for example can be used in a similar way: You can use the `File > Add Data > Add Basemap` menu to add OpenStreetMap as a layer, create new data using the ArcToolbox `Data Management Tools > Feature Class > Create` → `Feature Class`, edit it using the editor tools and export as KML using the ArcToolBox `Conversion Tools > Layer To KML...` The advantage of QGIS is that is a rather simple, Free and Open Source tool, whereas ArcMAP is licensed software. On the other hand, the ArcMAP KML format exporter is better in maintaining the styling of your symbolisation. Note that ArcMAP at present only exports KMZ files, and OpenLayers only imports KML, so you will need to unzip the files...

6 Combining OpenStreetMap with our own local overlay data

In the previous section we have created some local data (walking routes in the Aamsveen) that fit the OpenStreetMap web map. The last step now is to publish this overlay data on the webpage with the OpenLayers web-viewer we created in section 4.

6.1 Adding a KML layer

Let's add the KML file we created earlier and combine it with the OpenStreetMap:

Task 19 : Add the code in listing 3 to the html–file you made earlier (use the filefragments folder). Add it **after** the `myOSMLayer` definition. Now look for the line in the code that reads:

```
myMap.addLayer([myOSMLayer]);
```

To include the new layer in the `map` object **change** it to:

```
myMap.addLayers([myOSMLayer,myKML]);
```

Test it in the browser. •

Listing 3: kmlLayer.txt

```
myKML = new OpenLayers.Layer.Vector("KML route", {                  generic vector type
        strategies: [new OpenLayers.Strategy.Fixed()],              loads all features at once
        protocol: new OpenLayers.Protocol.HTTP({                    using web protocol
            url: "myRoutes.kml",                                    path to file
            format: new OpenLayers.Format.KML({                     use KML parser
                extractStyles: true,                                use KML styles
                extractAttributes: true                             use KML attributes
            })
        })
    })
```

Task 20 : Now we have more than one layer, we want to control the visibility of our overlay data. For that, add the line :

```
myMap.addControl(new OpenLayers.Control.LayerSwitcher()→
);
```

after the other `addControl` statements. Try it in the browser. •

A new control should have been added to the map: next to the *pan* and *zoom* tools, there now should be a little + icon in the *upper-right*. Clicking it will reveal a Layer Control, in which you can switch layers on and off.

For your final map, you could use this mechanism to have several routes, in different KML files (having different symbols and/or colours), in different layers in the OpenLayers map page...

7 Closing remarks and challenges

In these exercises, you have been using data and tools that can do so much more that what you we could do in the short time available. You should also keep in mind that, for simplicity and speed, we have used data that was prepared for the tasks given to you. In "real life" the process is usually not so smooth. You will have to use the tools more to achieve additional editing, preparation and maybe analyzing of your data.

As a challenge,we have included a bit more complicated task, for which you will have to puzzle a bit to get it done:

Task 21 : The task is to add to the CartoDB site you made earlier

some data about ground water levels. This data was collected from measurement tubes in the area in 2001. The data is provided as a comma-delimited text file (groundwaterPoints.csv). You have to add the data to your CartoDB tables and map it. •

The challenge here is that this data is not in a ready made GIS format, and also it is not in the latitude-longitude projection (on the WGS84 spheroid) that is used in the CartoDB system. You will need the QGIS software to make the data useable for the CartoDB system.

Happy puzzling!