Web Maps

Barry Rowlingson

August 12, 2009

Javascript web mapping

There is great demand these days for maps on web pages. Providers such as Google and Yahoo! have mapping systems that allow users to add data onto their base maps for web pages.

However there are also mapping frameworks written in Javascript that run within web browsers. They can then fetch map data locally or over the web for display. All that's necessary to make the map public is to put it on a public web server. This provides a very low bar to publishing spatial data.

The big map data companies restrict use of their map data. As an alternative, the OpenStreetMap project has been sourcing freely usable data and encouraging the collection and creation of data by anyone with a GPS or GPS-enabled phone. A very high quality map database with a wide coverage has been created this way. This data can be combined with open source javascript mapping tools to create a freely usable and re-usable map system.

Spatial data in R

The `sp` package contains functions for handling spatial data along the lines of the Open Geospatial Consortium Simple Features specification. This covers point, line, and area features (including holes and islands). Each feature can have a database-style record of attributes. Spatial data can also have a coordinate reference system to enable projections and conversions between reference systems.

This package will make use of the `sp` package objects for spatial data.

State data

For this section we will create a simple map from the USA state locations and data provided with R. First, we need the `webmaps` package – this also loads the `sp` package:

```
> require(webmaps)
```

Then we make a `SpatialPointsDataFrame` from the state data:

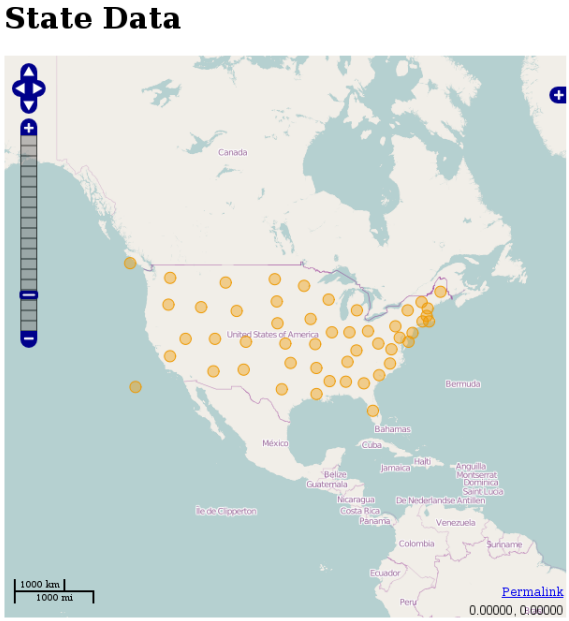
```
> state=data.frame(state.x77)
> state$Name = rownames(state)
> coordinates(state) = cbind(state.center$x,state.center$y)
```

And now we can make a map

```
> osmMap(layer(state,"States"),title="State␣Data",outputDir="./states1")
```

```
[1] "./states1/index.html"
```

Now open that file in your web browser and you should see a map. You can click on the points to get the state data:



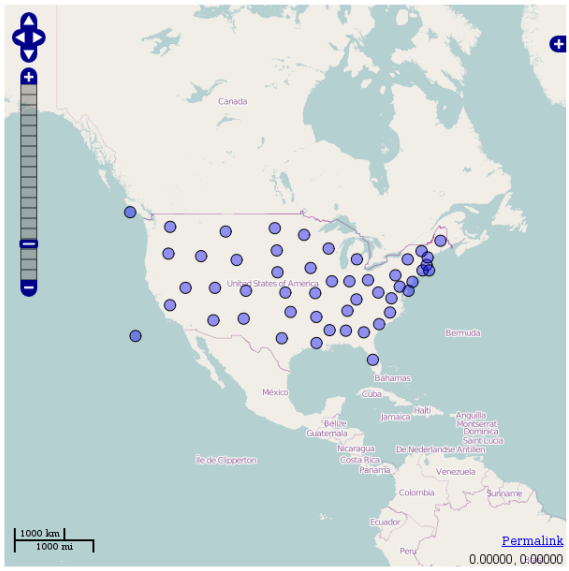
This map is produced using the OpenLayers mapping toolkit and OpenStreetMap base layer. At upper-left are the zoom and navigation controls (although you can also drag and use a mouse wheel). Upper-right is a control to let you switch layers on and off. Bottom-left is a scale bar, and bottom right are your mouse coordinates and a bookmarkable 'permalink' - this lets you zoom and shift your map, and then get a web link you can use to preserve the state of the map.

The points on the map are drawn using the default OpenLayers colours. You can change the appearance by passing an `lstyle` object using the `style` parameter:

```
> osmMap(layer(state,"States",lstyle(fillColor="blue",strokeColor="black")),
+        title="State␣Data",outputDir="./states2")
```

```
[1] "./states2/index.html"
```

State Data

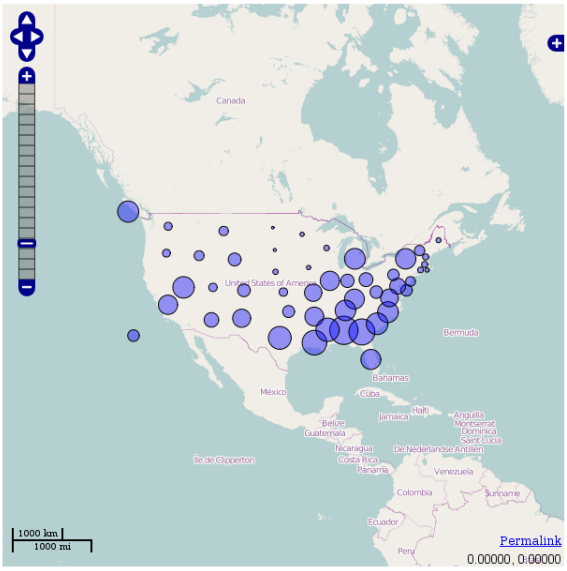


OpenLayers has a simple 'attribute replacement' system for styling features from attributes. If we want to set the point radius according to the murder rate, we just set the point radius style attribute like this:

```
> osmMap(layer(state,"States",
+               lstyle(fillColor="blue",strokeColor="black",pointRadius="${Murder}")),
+         title="State Data",outputDir="./states3")
```

```
[1] "./states3/index.html"
```

State Data



Note that you can't set a style value to an arithmetic expression of an attribute. If you want the radius to show the population you'll have to create an appropriately scaled population value first (I chose the murder rate here as an example since its values are good for pixel sizes). Another way of doing this is to edit the generated Javascript - see the OpenLayers docs for more information.

John Snow cholera data

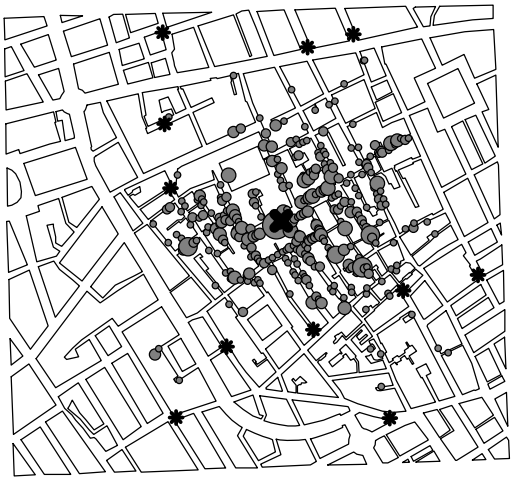
This is a famous data set from the 1854 map of cholera cases in London. The data used here come from the `snow_files.zip` downloaded from the web site for the book Applied Spatial Data Analysis with R by Bivand, Pebesma, and Gómez-Rubio, `http://asdar-book.org/`.

The data come as four shapefile sets - the building outlines, death locations, the location of the suspected Broad Street pump, and the locations of other water pumps in the area. We can read these in using `readOGR` from `rgdal`, and approximate the plot on page 107:

```
> require(rgdal)
> buildings = readOGR("./JohnSnow","buildings")
```

```
OGR data source with driver: ESRI Shapefile
Source: "./JohnSnow", layer: "buildings"
with  158  rows and  2  columns
Feature type: wkbPolygon with 2 dimensions
```

```
> deaths = readOGR("./JohnSnow","deaths",verbose=FALSE)
> pumps = readOGR("./JohnSnow","nb_pump",verbose=FALSE)
> brdst = readOGR("./JohnSnow","b_pump",verbose=FALSE)
> plot(buildings)
> points(deaths,cex=sqrt(deaths@data$Num_Cases/2),
+        col="#000000",bg="#808080",pch=21)
> points(pumps,pch=8,lwd=3,cex=1)
> points(brdst,pch=4,lwd=8,cex=1.5)
```



But this lacks context - we want to see these points on a modern map of London. The first thing we need to do is project the data to the WGS84 coordinate system used by web mapping packages.

Although the shapefiles have coordinate systems set in their `.prj` files, these seem to be slightly wrong, so the first thing we do is give them the correct projection string from the EPSG database distributed with the `sp` package:

```
> bng = CRS("+init=epsg:27700") # British Grid
> wgs84 = CRS("+init=epsg:4326") # WGS84
> proj4string(buildings) = bng
> proj4string(deaths) = bng
> proj4string(pumps) = bng
> proj4string(brdst) = bng
> buildings = spTransform(buildings, wgs84)
> deaths = spTransform(deaths,wgs84)
```

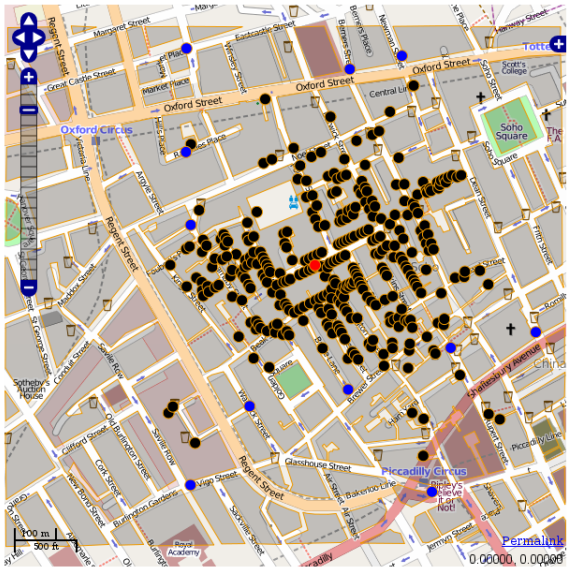
```
> pumps = spTransform(pumps,wgs84)
> brdst = spTransform(brdst,wgs84)
```

At this point the datasets are all correctly converted to WGS84, so we can create a web map with our data layers on:

```
> osmMap(layer(buildings,"Buildings",lstyle(fillOpacity=0.2,fillColor="black")),
+        layer(deaths,"Deaths",lstyle(fillColor="black",fillOpacity=1.0)),
+        layer(brdst,"BroadStPump",lstyle(fillColor="red",fillOpacity=1.0)),
+        layer(pumps,"Pumps",lstyle(fillColor="blue",fillOpacity=1.0)),
+        outputDir="./SnowMap1")
```

```
[1] "./SnowMap1/index.html"
```

map

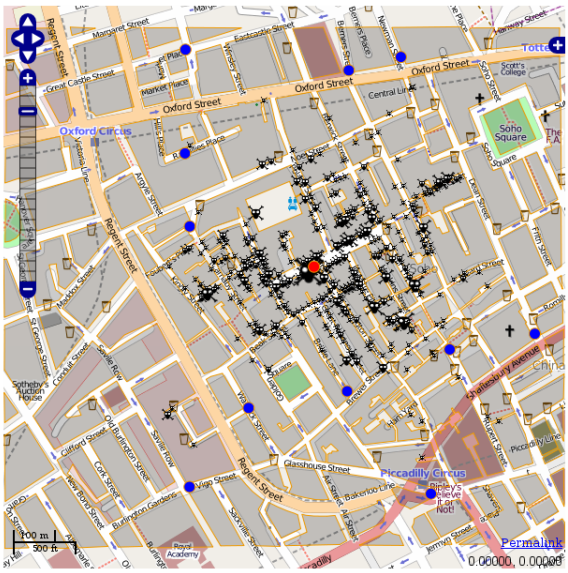


Now we'll use an external graphic scaled according to the number of cases. We'll create a new data column to get roughly the same scaling as when using the `cex=` parameter in the R example:

```
> deaths$size = 6 * sqrt(deaths$Num_Cases/2)
> osmMap(layer(buildings,"Buildings",lstyle(fillOpacity=0.2,fillColor="black")),
+        layer(deaths,"Deaths",lstyle(externalGraphic="skull.png",
+                                     pointRadius="${size}",
+                                     fillOpacity=1.0)),
+        layer(brdst,"BroadStPump",lstyle(fillColor="red",fillOpacity=1.0)),
+        layer(pumps,"Pumps",lstyle(fillColor="blue",fillOpacity=1.0)),
+        outputDir="./SnowMap2")
```

```
[1] "./SnowMap2/index.html"
```

map



Of course this map does not hold a candle to John Snow's original, but serves to illustrate some of the capabilities of OpenLayers.