Web Mapping with OpenStreetMap and Leaflet

Introduction

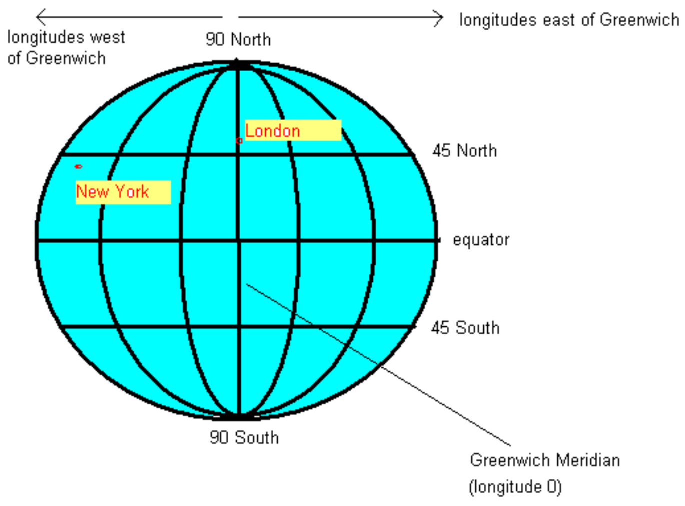
Websites and smartphone apps which show maps are very common these days. Many such sites use commercial mapping providers such as Google Maps and the like. However, such providers place restrictions on their users and the maps are often generic and do not show information for specialised users such as walkers and cyclists.

OpenStreetMap is a project to produce free, editable maps of the entire world. Users can contribute their own mapping data and the data can be used for free by anyone; see the OpenStreetMap site for more details. Users typically survey a road or path with a GPS device, such as a smartphone, and then draw the road or path on top of their GPS trace using editing software. The fact that the data is free means that developers can use it for their own pruposes, for instance, create their own maps or develop routing applications.

Leaflet is an open-source JavaScript mapping library which offers similar functionality to commercial web mapping services such as Google Maps. It allows you to embed a "slippy" map into a web page. However, unlike these other services, Leaflet can be used to display maps from ***a whole range*** of map providers, including, but not restricted to, OpenStreetMap.

Latitude and Longitude

In order to understand location-based applications, it is important to understand the coordinate system used on the earth. The most common coordinate system uses ***latitude and longitude***. Latitude is a measure of how far north or south you are: the equator is at 0 degrees, while the North Pole is at 90 degrees North, we are at about 50 and Spain is at about 40. Longitude is a measure of how far east or west you are: 0 degrees of longitude is referred to as the ***Prime Meridian*** (or ***Greenwich Meridian***) and passes through Greenwich, London. By contrast Germany is located between approximately 7 degrees and 15 degrees East, while New York is at 74 degrees West and the west coast of North America at approximately 120 degrees West.


longitudes west of Greenwich
longitudes east of Greenwich
90 North
London
45 North
New York
equator
45 South
90 South
Greenwich Meridian (longitude 0)

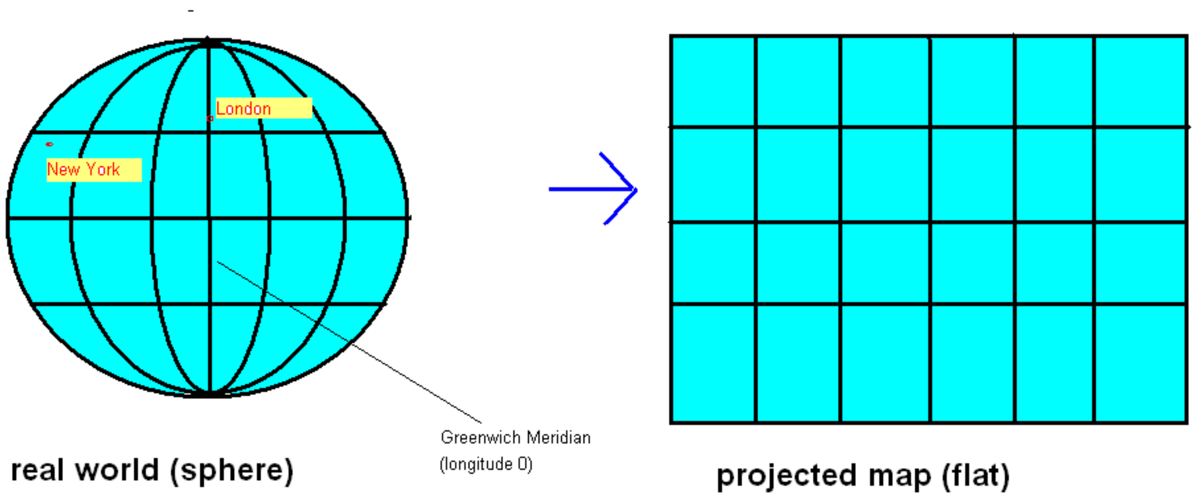
So a given point on the earth can be defined via its latitude and longitude. We are at, approximately, 50.9 North (latitude) and 1.4 West (longitude). By convention, latitudes north of the equator and longitudes east of Greenwich are treated as positive, so we can also define our position as **longitude -1.4, latitude +50.9**.

Projections

An important consideration when doing web mapping is that ***the earth is not flat*** (it's more or less a sphere) while ***maps are flat***. To display a curved surface on a flat piece of paper or computer screen, we need to do a ***projection*** and mathematically transform the latitude and longitude to coordinates suitable for representation on a flat surface. Why is this? Imagine any printed map of the earth. The map is equal width everywhere, from far northern areas such as Greenland or north Norway, to the equator. This does not match reality; since the earth is (more or less) a sphere, the circumference of the earth will be much greater at the equator than those far northern areas - indeed, at the poles, the circumference of the earth is zero!

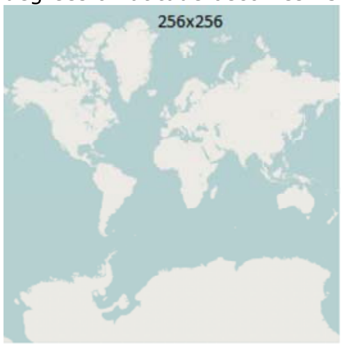
For this reason, latitude and longitude must be transformed to so called ***projected coordinates*** if we want to represent them on a flat surface, such as a computer screen. The details of exactly how this projection is done is out of scope of this unit, but it is something to be aware of if you aim to do more with web mapping. Leaflet makes it easy for us by doing the transformation automatically.

The most common projection used with web mapping is informally referred to as the "Google Projection" (more formally, a type of ***Spherical Mercator***), so called because Google Maps popularised it.

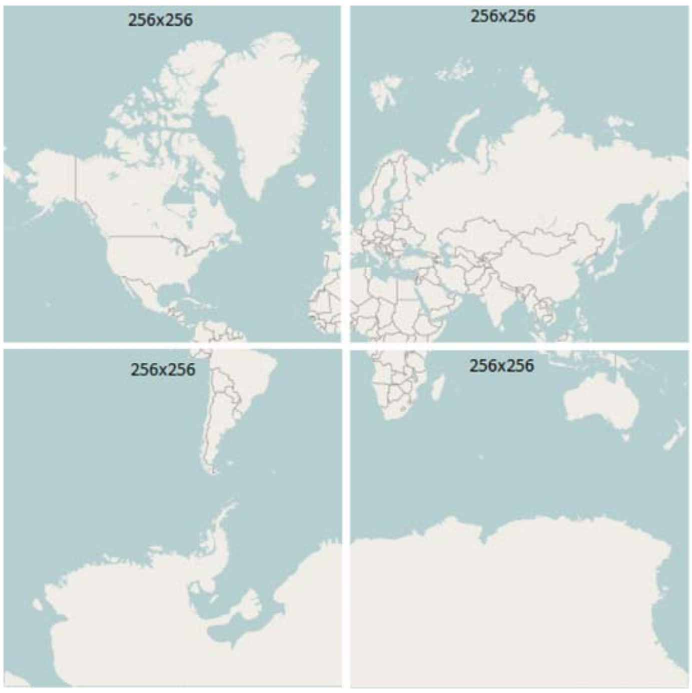


Details on the "Google Projection"

If you are interested, this is how the "Google Projection" works. It consists of a series of ***zoom levels***, with 0 the most zoomed out and successive levels progressively zoomed in. How does this work? Basically, zoom level 0 is defined as a flat map of the entire world, occupying 256x256 pixels, so that 360 degrees of longitude becomes 256 pixels and 180 degrees of latitude becomes 256 pixels, as shown below:



Each successive zoom level zooms in by a factor of 2 in both directions, so that at zoom level 1, there are four 256x256 pixel tiles, each covering a quarter of the earth (N of the equator and W of the Prime Meridian; N of the equator and E of the Prime Meridian; S of the equator and W of the Prime Meridian and S of the equator and E of the Prime Meridian):



With progressive zoom levels, we continue zooming in by a factor of 2, so that zoom level 2 has 16 tiles (4x4), zoom level 3 has 64 (8x8), and so on. Each tile has an x and y coordinate where x=0 represents the leftmost column of tiles, y=0 represents the topmost row of tiles, and the tile with x=0 y=0 represents the top left tile (x=1 y=0 represents the second tile on the top row, and so on)

Images from OpenStreetMap, (c) OSM contributors, licenced under CC-by-SA (not ODBL as they are old images)

Some examples

We will be using OpenStreetMap as the mapping provider.

Hello World in Leaflet

This is a basic example which creates a map:

```
<html>
<head>
<title>Leaflet Example</title>
<script type='text/javascript'
src='http://www.free-map.org.uk/osmuk/jslib/leaflet.js'></script>
<link rel='stylesheet' type='text/css'
href='http://www.free-map.org.uk/osmuk/jslib/leaflet.css' />
<script type='text/javascript'>

function init()
{
    var map = L.map ("map1");

    var attrib="Map data copyright OpenStreetMap contributors, Open Database Licence

    L.tileLayer
        ("http://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png",
            { attribution: attrib } ).addTo(map);

    map.setView([50.908,-1.4], 14);
}
</script>
</head>
<body onload="init()">
<h1>Leaflet Test</h1>
<div id="map1" style="width:800px; height:600px"> </div>
</body>
</html>
```

Note the following:

- We link in the Leaflet library (a local copy is on Freemap) as an external JavaScript file.
- We also link in the Leaflet CSS file; this is necessary ad Leaflet uses some advanced CSS features.
- Note how the HTML contains a ***div*** with an ID of ***map1***. This will hold our map.
- Moving onto the JavaScript init() function, where we write our custom code:
    - We first create a map object (the variable ***map***, an object of type ***L.Map*** created with the ***L.map()*** method), and associate it with ***div1***;
    - Next we set up a map ***layer***. Leaflet maps consist of several ***layers***. This allows us, for instance, to overlay, the locations of points of interest on top of an underlying map. Here, the layer is a ***TileLayer***, a layer of map tiles. The map tiles come from the server ***tile.openstreetmap.org***. The x, y and z in the URL relate to the "Google" tiling scheme, see above; z is the zoom level. Leaflet will automatically work out the correct values to place in here.
    - Having created the layer, we add it to the map (***addTo(map)***).
    - Finally, we set the latitude and longitude of the centre of the map, here latitude=50.908, longitude=-1.4. We pass the latitude and longitude in as a two-member ***array***.
    - Note how, when we set the centre of the map, we specify the ***zoom level*** (see discussion of "Google" tiling scheme, above). 13 and 14 are good ones to start with; try experimenting with different values.

Exercise 1

1. New York is at longitude 74 West, latitude 40.75 North (more or less). Change the example above so that it's centred on New York, at zoom level 13.
2. Find the latitude and longitude of your home town (see, for example, www.informationfreeway.org) and change the example so it's centred on your home town.

Adding features

Most web maps have some kind of ***overlay*** on the base map, for example a series of markers plotting the locations of pubs, cafes or other points of interest. We can even draw vector shapes (lines, polygons, circles) and add them to the map. This example creates a map and adds a feature to it:

```
<html>
<head>
<title>Leaflet Example</title>
<script type='text/javascript'
src='http://www.free-map.org.uk/osmuk/jslib/leaflet.js'></script>
<link rel='stylesheet' type='text/css'
href='http://www.free-map.org.uk/osmuk/jslib/leaflet.css' />
<script type='text/javascript'>

function init()
{
    var map = L.map ("map1");

    var attrib="Map data copyright OpenStreetMap contributors, Open Database Licence

    L.tileLayer
        ("http://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png",
            { attribution: attrib } ).addTo(map);

    var pos = [50.908, -1.4];
    map.setView(pos, 14);

    L.marker(pos).addTo(map);
}
</script>
</head>
<body onload="init()">
<h1>Leaflet Test</h1>
<div id="map1" style="width:800px; height:600px"> </div>
</body>
</html>
```

Hopefully this code is obvious. We simply create a marker at the specified position (***L.marker*** takes an array of two members, latitude and longitude), and add it to the map.

The following example shows how to create circle, polyline (i.e. a line with multiple points) and polygon features:

```
<html>
<head>
<title>Leaflet Example</title>
<script type='text/javascript'
src='http://www.free-map.org.uk/osmuk/jslib/leaflet.js'></script>
<link rel='stylesheet' type='text/css'
href='http://www.free-map.org.uk/osmuk/jslib/leaflet.css' />
<script type='text/javascript'>

function init()
{
    var map = L.map ("map1");

    var attrib="Map data copyright OpenStreetMap contributors, Open Database Licence

    L.tileLayer
        ("http://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png",
            { attribution: attrib } ).addTo(map);

    map.setView([50.908, -1.4], 14);


    // In Leaflet 1.0+
     L.circle([50.9079, -1.4015], { radius:100}).addTo(map);


    // Saints stadium (football ground)
    L.polygon ( [
        [50.9063 , -1.3914 ] ,
        [50.9063 , -1.3905 ] ,
        [50.9053 , -1.3905 ] ,
        [50.9053 , -1.3914 ]
        ] ).addTo(map);

    // Route to railway station
    L.polyline ( [
        [50.9079, -1.4015] ,
        [50.9071, -1.4015],
        [50.9069, -1.4047],
        [50.9073, -1.4077],
        [50.9081, -1.4134]
        ]).addTo(map);

}
</script>
</head>
<body onload="init()">
<h1>Leaflet Test</h1>
<div id="map1" style="width:800px; height:600px"> </div>
</body>
</html>
```

Again, hopefully this should be mostly self-explanatory. The second parameter when creating a circle is the radius in metres. For polygons and polylines, we specify the shape using an array of points. Each point is a two-member array containing latitude and longitude.

Styling features

We can create custom styles for map features. For polygons, circles and polylines, we can set properties such as the colour and the opacity (opposite of transparency: 0 is completely transparent and 1 is completely opaque).

```
<html>
<head>
<script type='text/javascript'
src='http://www.free-map.org.uk/osmuk/jslib/leaflet.js'></script>
```

```
<link rel='stylesheet' type='text/css'
href='http://www.free-map.org.uk/osmuk/jslib/leaflet.css' />
<title>Leaflet Example</title>
<script type='text/javascript'>

function init()
{
    var map = L.map ("map1");

    var attrib="Map data copyright OpenStreetMap contributors, Open Database Licence

    L.tileLayer
        ("http://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png",
            { attribution: attrib } ).addTo(map);

    map.setView([50.908, -1.4], 14);


    //  in Leaflet 1.0+
     L.circle([50.9079, -1.4015], { radius:100, fillColor: 'blue',
                                 color: 'red', opacity: 0.5 }).addTo(map);


    // Saints stadium (football ground)
    L.polygon ( [
        [50.9063 , -1.3914 ] ,
        [50.9063 , -1.3905 ] ,
        [50.9053 , -1.3905 ] ,
        [50.9053 , -1.3914 ]
        ] ).addTo(map);

    // Route to railway station
    L.polyline ( [
        [50.9079, -1.4015] ,
        [50.9071, -1.4015],
        [50.9069, -1.4047],
        [50.9073, -1.4077],
        [50.9081, -1.4134]
        ]).addTo(map);
}
</script>
</head>
<body onload="init()">
<h1>Leaflet Test</h1>
<div id="map1" style="width:800px; height:600px"> </div>
</body>
</html>
```

In this example, we have set the circle's fill colour (interior colour) to blue, its outline colour to red, and the opacity to 0.5.

Popups

One commonly-encountered feature of web mapping is ***popups***, in which the user can click on a marker and be presented with additional information on that feature. These are easy to do in Leaflet: we simply call the ***bindPopup()*** method of the feature to attach a popup to that feature. ***bindPopup()*** takes one parameter, the text (you can include HTML tags) to appear in the popup. Here is an example. Note how you have to store the circle, polytlime and polygon in variables so that you can then call bindPopup():

```
<html>
<head>
<title>Leaflet Example</title>
<script type='text/javascript'
src='http://www.free-map.org.uk/osmuk/jslib/leaflet.js'></script>
<link rel='stylesheet' type='text/css'
href='http://www.free-map.org.uk/osmuk/jslib/leaflet.css' />
```

```
<script type='text/javascript'>

function init()
{
    var map = L.map ("map1");

    var attrib="Map data copyright OpenStreetMap contributors, Open Database Licence

    L.tileLayer
        ("http://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png",
            { attribution: attrib } ).addTo(map);

    map.setView([50.908, -1.4], 14);

    // Location of Solent University
    //In Leaflet 1.0+
    var solent = L.circle([50.9079, -1.4015],
        { radius: 100, fillColor: 'blue', color: 'red', opacity: 0.5 }
            ).addTo(map);



    // Saints stadium (football ground)
    var saintsStadium = L.polygon ( [
        [50.9063 , -1.3914 ] ,
        [50.9063 , -1.3905 ] ,
        [50.9053 , -1.3905 ] ,
        [50.9053 , -1.3914 ]
        ] ).addTo(map);

    // Route to railway station
    var routeToStation = L.polyline ( [
        [50.9079, -1.4015] ,
        [50.9071, -1.4015],
        [50.9069, -1.4047],
        [50.9073, -1.4077],
        [50.9081, -1.4134]
        ]).addTo(map);

    solent.bindPopup("Solent University");
    saintsStadium.bindPopup("Southampton FC football ground");
    routeToStation.bindPopup("Route from the university to the station");
}
</script>
</head>
<body onload="init()">
<h1>Leaflet Test</h1>
<div id="map1" style="width:800px; height:600px"> </div>
</body>
</html>
```

Events

In a mapping application, we commonly need to respond to user ***events***, for instance we might want something to happen if the user clicks on the map (such a display a new marker, for instance) or if the user finishes dragging the map to a new location (we might want to load markers from a server, for instance). It is easy to attach events in Leaflet, here is an example:

```
<html>
<head>
<title>Leaflet Example</title>
<script type='text/javascript'
src='http://www.free-map.org.uk/osmuk/jslib/leaflet.js'></script>
<link rel='stylesheet' type='text/css'
href='http://www.free-map.org.uk/osmuk/jslib/leaflet.css' />
<script type='text/javascript'>
```

```
function init()
{

    var map = L.map ("map1");

    var attrib="Map data copyright OpenStreetMap contributors, Open Database Licence

    L.tileLayer
        ("http://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png",
            { attribution: attrib } ).addTo(map);

    map.setView([50.908, -1.4], 14);
    map.on("click",onMapClick)
}

function onMapClick(e)
{
    // "e.latlng" is an object (of type L.LatLng) representing the mouse click
    // position
    // It has two properties, "lat" is the latitude and "lon" is the longitude.
    alert("You clicked at: " + e.latlng.lat + " " + e.latlng.lng);
}
</script>
</head>
<body onload="init()">
<h1>Leaflet Test</h1>
<div id="map1" style="width:800px; height:600px"> </div>
</body>
</html>
```

We use the ***on()*** method of the map to attach an event handler to the map. The ***on()*** method takes two parameters: the event type and the event handler function. A full list of event types can be found on the Leaflet website.

In the event-handling function itself (***onMapClick()***), we use the event object ***e*** to obtain details about the event (in this case, we are interested in the click position). The event object ***e*** is automatically passed to the event handler function by the Leaflet library. The event object has a ***latlng*** property, representing the position of the mouse click, which is an object of the type ***L.LatLng***. This in turn has two properties, ***lat*** and ***lng*** representing the actual latitude and longitude.

Exercise 2

1. Add a marker on your map from Exercise 1 on your home town.
2. Combine the marker and mouse click event examples, above, so that by clicking on the map, you add a marker to the map at that position. ***You will need to make your map a global variable, i.e. declare it outside of any function, so that the init() function and the mouse click handler can both access it.***e.g.:

```
var map;

function init()
{
    map = L.map("map1");
    // ... etc ...
}
```

3. Centre the map on your home town and draw a green circle with radius 5km there. Use informationfreeway.org to find the latitude and longitude of your home town. Add a popup to the circle, showing the name of the town.
4. Extend Question 1 to allow the user to specify details about the marker, using simple JavaScript prompt boxes, e.g:

```
var details=prompt("Please enter details");
```

to read in a type (e.g. pub, restaurant) and a description from the user, and use AJAX together with a server-side script to save the marker on the server. There is an ***annotations*** table in the ***dftitutorials*** database which you can add the details of the marker to.