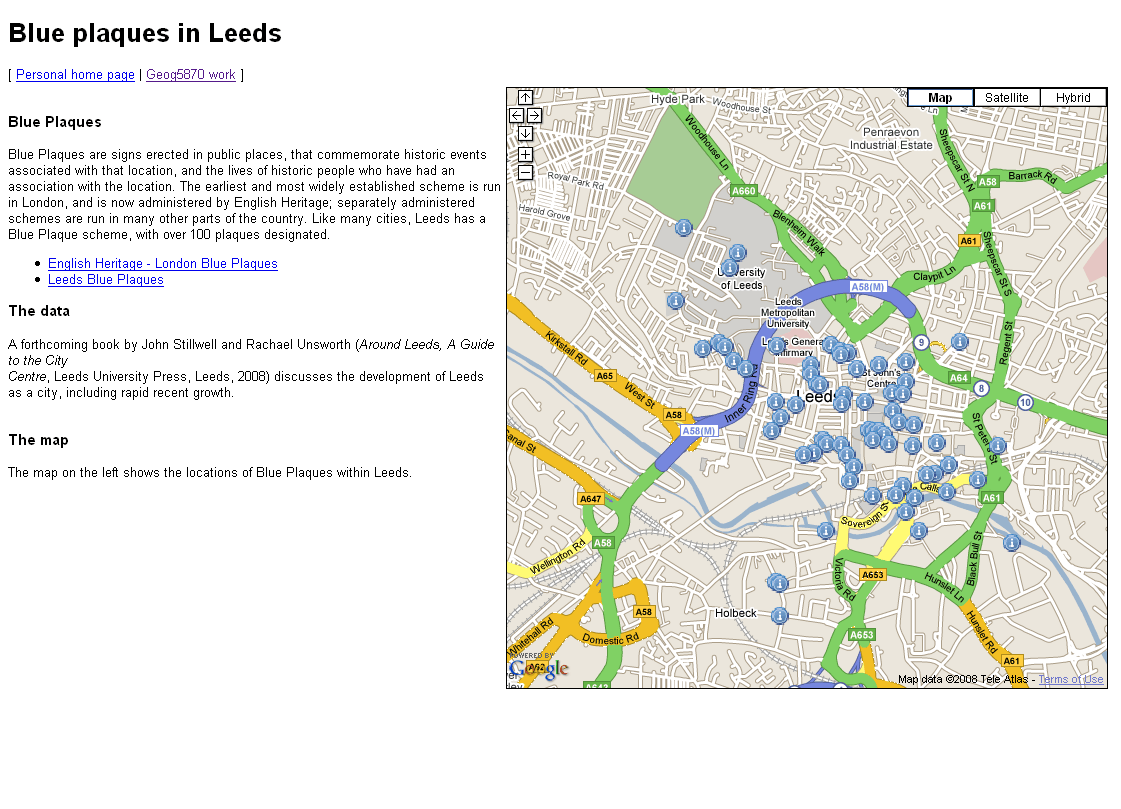
**Introduction**

So far, we have looked at creating maps and adding markers and infowindows to maps in Google Maps. This week, we look at various types of overlay.

**Contents**

1. [A recap of earlier work on Google Maps](https://vlebb.leeds.ac.uk/bbcswebdav/users/geo6odw/geog58705871_2010-11/wk8/geog5870_wk8_1.html)
2. [Creating and adding lines and polygons](https://vlebb.leeds.ac.uk/bbcswebdav/users/geo6odw/geog58705871_2010-11/wk8/geog5870_wk8_2.html)
3. [Adding information to the polygons](https://vlebb.leeds.ac.uk/bbcswebdav/users/geo6odw/geog58705871_2010-11/wk8/geog5870_wk8_3.html)
4. [Additional overlay types](https://vlebb.leeds.ac.uk/bbcswebdav/users/geo6odw/geog58705871_2010-11/wk8/geog5870_wk8_4.html)
5. [Using a raster overlay](https://vlebb.leeds.ac.uk/bbcswebdav/users/geo6odw/geog58705871_2010-11/wk8/geog5870_wk8_4.html)
6. [Summary](https://vlebb.leeds.ac.uk/bbcswebdav/users/geo6odw/geog58705871_2010-11/wk8/geog5870_wk8_5.html)
7. [Practical](https://vlebb.leeds.ac.uk/bbcswebdav/users/geo6odw/geog58705871_2010-11/wk8/geog5870_wk8_prac.html)
8. **A recap of earlier work**
9. In the first lecture on GoogleMaps, we started by looking at the history and general features of Google Maps. The service was introduced  in 2005, and has grown rapidly in popularity since then.  The basic map data, and the framework for placing markers on the maps, use a Mercator projection, based on the WGS84 datum, the same datum used by many applications with a worldwide scope, including the Global Positioning System.  GoogleMaps supports a number of map views, including Map (the default view) view, Satellite view, Terrain view, Street view and Traffic view. Traffic view is a derivative of the default Map view, in which a number of major roads have their shading and line-style varied according to real-time traffic flow data.  Street view is also derived from Map view: in selected cities some streets are highlighted - indicating that street view data are available - and clicking on such a street leads to a pop-up window being created that shows an eye-level view at that point, which is pan-able through 360°.
10. **Generating bespoke maps**
11. There are a number of ways of generating bespoke maps using Google Maps. A simple approach is to create a URL that calls a GoogleMaps server and includes a parameter that sets the map centre.  It is also possible to give a map a URL as a parameter. If that URL references a file containing KML format marker data, then those markers will be drawn on the map created by Google Maps. Both these approaches are fairly quick and easy, but are of limited use, although they may be an acceptable solution for illustrating the location of something in an ad hoc fashion, especially if one does not have access to a web server on which one can create and make available files.
12. A far more powerful and flexible way of creating bespoke maps is to use the GoogleMaps API. Google Maps is based around a Javascript interface that draws map data from a server, and presents it on screen with various control functions. By using the API, you can extend the Javascript map interface in many ways, adding your own data as overlays and modifying the appearance of the map.
13. [](https://vlebb.leeds.ac.uk/bbcswebdav/users/geo6odw/geog58705871_2010-11/wk8/wk8_files/fig_1.gif)  
    Figure 1: Presenting information about   
    Blue Plaques in Leeds using Google Maps
14. In the second lecture, we looked at ways of extending the basic map in order to be able to present more useful information. This was demonstrated by adding some simple markers to the map, with pop-up information windows. At the same time, the growing size of the html page associated with this led to a need to clean up the code, and to separate out elements on a functional basis: individual files were created and used to contain the basic html code, the map loading logic, and a data file.
15. The practical session associated with the second lecture focussed on a more substantive task: creating a map with a set of markers, and with pop-up information linked to each marker. The markers related to the locations of Blue Plaques in and around central Leeds. The task presented two major problems that are characteristic of similar real-world tasks: first, extracting the data from a spreadsheet in a usable format, and second, converting the co-ordinate data into the required format.
16. The resulting page should have looked something like the shown in Figure 1, although it is possible to have included much richer background information, and to have presented the map in a more appealing manner.

**Adding overlays**

Markers are an example of one sort of overlay that can be placed on top of the background map in Google Maps. As we have seen in last week's material, it is possible to alter the marker image ; one implication of this is that different marker styles can be used to convey additonal information about the location (e.g. markers of different colours could be used to chow different types of location). There are a variety of alternative overlay s that can also be used in Google Maps, and we will look at some of these this week.

**Creating and adding lines and polygons**

The Blue Plaques practical task, and most of the examples that we studied earlier focussed on using markers to represent information. These are ideal for point level information, but not so good for area level information.  An obvious thing to want to do is to draw lines and polygons on the map. Figure 2 shows sample code for a map which will be drawn with a set of overlay lines. Three files are illustrated.

The first of these is the HTML file which will contain our map. The HEAD section contains links to a number of other files:

* The Google Maps JavaScript API
* The JSCoord library used last week
* A JavaScript data file
* A JavaScript map setup file
* Two style sheets - a general one (intended for general page appearance) and a more map specific one

The body of the HTML is much the same as in previous examples, although as we will be drawing a map of the UK in this example, I have made the map\_canvas division long and thin.

The second file shown is an excerpt of a data file, showing the general structure. This was converted from CSV exported from an Excel spreadsheet; all values have been placed in quotes. The data are taken from the 2001 Census Special Migration Statistics, and detail migration flows from different districts of the UK to Leeds. The 'origin' value is simply a district number, and the 'flow' value is the number of persons recorded in the Census who had changed their usual residential address in the year prior to the census. For each data value, the flow shows the number of people who lived in Leeds at the time of the Census, and in the origin district one year before the Census.

The third file shown in the map set up file. This sets up a map in the same way as done in the previous examples. It then iterates through a data set, again as in previolus examples with markers. However, in this case we are doing something different with the data: we are drawing lines on the map.

**wk8\_leeds\_sms\_eg1.html**

<!DOCTYPE html>

<html>

<head>

<script type="text/javascript" src="http://maps.google.com/maps/api/js?sensor=false"></script>

<script type="text/javascript" src="../jscoord-1.1.1/jscoord-1.1.1.js"></script>

<script type="text/javascript" src="wk8\_leeds\_sms\_eg1\_data.js"></script>

<script type="text/javascript" src="wk8\_leeds\_sms\_eg1\_mapsetup.js"></script>

<link rel="stylesheet" media="all" href="../../../style/general.css" type="text/css">

<link rel="stylesheet" media="all" href="../../../style/google\_maps\_style.css" type="text/css">

<title>Google Maps examples - lines</title>

</head>

<body onload="initialize()">

<h1>Google Maps examples - lines</h1>

<p>[ <a href="../index.html">Personal home page</a> | <a href="index.html">Geog5870 work</a> ]</p>

<div class="gmap" id="map\_canvas" style="height: 800px; width: 600px"></div>

<p>This maps shows migratiof flows from districts of the UK to Leeds, as recorded in the 2001 Census.</p>

<ul>

<li>Source: 2001 Census: Special Migration Statistics

<li>Census output is Crown copyright and is reproduced with the permission of the Controller of HMSO and the Queen's Printer for Scotland.

</ul>

</body>

</html>

**wk8\_leeds\_sms\_eg1\_data.js**

var os\_flows = [

{

"origin":"62",

"easting":"418659.9155",

"northing":"279492.2417",

"flow":"75"

},

...

]

**wk8\_leeds\_sms\_eg1\_mapsetup.js**

var map; // The map object

var myCentreLat = 53.825740;

var myCentreLng = -1.506015;

var initialZoom = 6;

function initialize() {

/\*

\* In this example, lines are from each origin to a known destination point which is also

\* the map centre

\*/

var myDest = new google.maps.LatLng(myCentreLat,myCentreLng);

var myOptions = {

zoom: initialZoom,

center: myDest,

mapTypeId: google.maps.MapTypeId.ROADMAP

};

map = new google.maps.Map(document.getElementById("map\_canvas"),myOptions);

var info = "";

for (id in os\_flows) {

// Convert co-ords

var osPt = new OSRef(os\_flows[id].easting,os\_flows[id].northing);

var llPt = osPt.toLatLng(osPt);

llPt.OSGB36ToWGS84();

var myOrig = new google.maps.LatLng(llPt.lat,llPt.lng);

/\*

\* Construct a flow line

\*

\* In this example, lines are from each origin to a known destination point

\* Destination is hard coded here for sake of simplifying code

\*/

var flowLine = new google.maps.Polyline({

path: [myOrig,myDest],

strokeColor: "#FF0000",

strokeOpacity: 1.0,

strokeWeight: 2

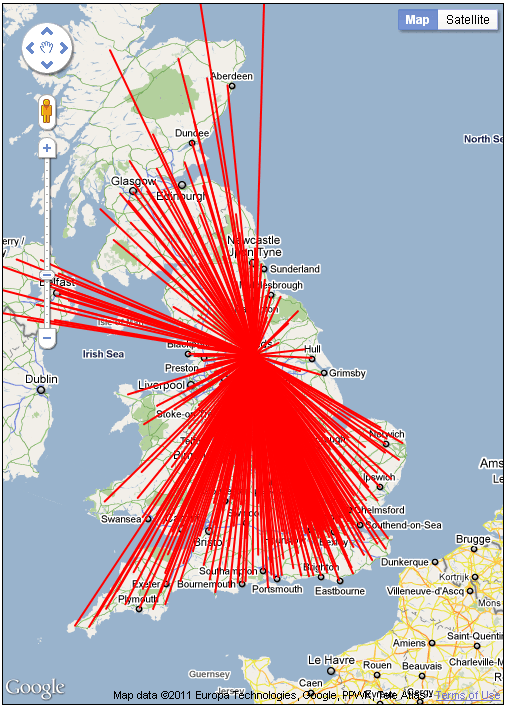
});

flowLine.setMap(map);

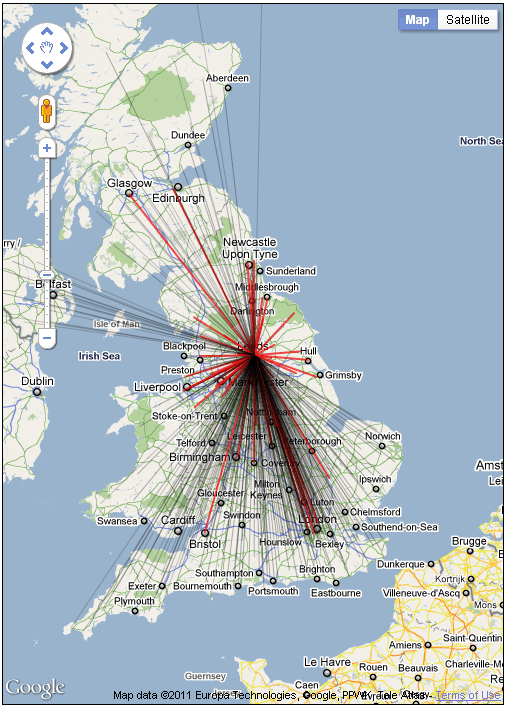
}

}

**Figure 2:  Files for the line-drawing example**



a) Ouput with initial line styles



b) Output with revised line styles

Figure 3: Output from line-drawing example

Figure 3a shows the map generated by this example.

The lin that have been added are called *polylines* in the Google Maps API. polyline is defined by a series of points; this map uses the simplest of such lines, which are defined by only a start and an end point. Let us return to the code shown in the map setup script in Figure 2. The loop 'for (id in os\_flows) {}' iterates through the data array, carry out its body section once for each data point. The first task - as with the Blue Plaques example from last week - is to convert the location data from OS grid references to lat-lng data points. In this example, we construct a LatLng object called myOrig from each point we read. Note that despite the fact that the eastings and northings are quoted in the data file, and might thus be thought of as strings, the latlng constructor is happy to treat them as numeric values.

Next we draw the line. There are two stages to this. Firstly, we contstruct a polyline object, using 'new google.maps.Polyline()'. This requires one paramter - an object which lists line co-ordinates and the various options. The most important of these are the line co-ordinates. These are given in the path property of the polyline options object. The path is an array, containing a set of points, each of which is a google.maps.LatLng() object. We use two points: the myOrig point that has just been created (as converted from the OS grid references), and a second point myDest. As this map shows flows to one location only we define the destination once, and use the same point each time in our path.

The other characteristics for a polyline are the colour, opacity and width of the line. In this initial example, we have drawn all lines with the same characteristics: opaque red lines of weight 2.

The second thing that we need to do is add the line that has now been defined to the map. The is done in the statement: 'flowLine.setMap(map);'. We use the setmap() method of the generic google.maps.Polyline() object to attach the line to the map, giving the name of the map object to which we wish to attach the line.

As you can see from Figure 3a, the resulting map contains a mass of lines, which does not tell us much apart from the fact that - like most large cities, especially those with large universities, Leeds draws migrants from almost everywhere in the country.

We can alter the setup program to change the line style for flows of different size. Figure 4 shows excerpts from the code of a revised example. Some changes have been made within the initialize() function. Firstly, we have added three new variables, myColor, myOpacity and myWeight; as we intend to alter these characteristics for each line.

We have also added a set of statements into the for loop that cycles through each data point. We have a series of if {} statements, which ste up the line characteristics. These are structured so that large flows which meet all the 'if' criteria (i.e. they're greater than 0, they're also greater than 10 etc...) will over-write the previous values for myColor etc.

The lines are set to be drawn in black (but thin, and fairly transparent) is the flow is small, and in red (and thicker, and more opaque) is the flow is larger. This stage is clearly in the arena of basic map design - one needs to know something about the nature of the data being mapped and about the range of possible values, and then try out various different line styles.

The output using this revised version is shown in Figure 3b. The map is clearer, and it is possible to distinguish those origins from which the major flows are drawn, although it is still quite crowded - a common result in maps of this type. Further improvements might be made with changes to the line styles, and we might also decide to only draw those lines that represent flows above a certain threshold (say, 50 persons).

The files used in these two examples are available here:

* [wk8\_leeds\_sms\_eg1.html](https://vlebb.leeds.ac.uk/bbcswebdav/users/geo6odw/geog58705871_2010-11/wk8/wk8_examples/line_examples/wk8_leeds_sms_eg1.html)
* [wk8\_leeds\_sms\_eg1.mapsetup.js](https://vlebb.leeds.ac.uk/bbcswebdav/users/geo6odw/geog58705871_2010-11/wk8/wk8_examples/line_examples/wk8_leeds_sms_eg1_mapsetup.js)
* [wk8\_leeds\_sms\_eg1\_data.js](https://vlebb.leeds.ac.uk/bbcswebdav/users/geo6odw/geog58705871_2010-11/wk8/wk8_examples/line_examples/wk8_leeds_sms_eg1_data.js)
* [wk8\_leeds\_sms\_eg1\_data\_test.js](https://vlebb.leeds.ac.uk/bbcswebdav/users/geo6odw/geog58705871_2010-11/wk8/wk8_examples/line_examples/wk8_leeds_sms_eg1_data_test.js) [a truncated data file for initial testing]
* [wk8\_leeds\_sms\_eg2.html](https://vlebb.leeds.ac.uk/bbcswebdav/users/geo6odw/geog58705871_2010-11/wk8/wk8_examples/line_examples/wk8_leeds_sms_eg2.html)
* [wk8\_leeds\_sms\_eg2\_mapsetup.js](https://vlebb.leeds.ac.uk/bbcswebdav/users/geo6odw/geog58705871_2010-11/wk8/wk8_examples/line_examples/wk8_leeds_sms_eg2_mapsetup.js) [uses same data file as eg1]

If you use copies of these, you may need to alter the relative paths in the HEAD section to the JSCoord library and the CSS files.

As with any map, the framing page should give some sort of explanatory key when lines are shown in different colours to convey some quantitative meaning.

excerpts from **wk8\_leeds\_sms\_eg2\_mapsetup.js**

...

var myColor;

var myOpacity;

var myColor;

for (id in os\_flows) {

...

if (os\_flows[id].flow > 0) {

myColor = "#000000";

myOpacity = 0.2;

myWeight = 1;

}

if (os\_flows[id].flow > 10) {

myColor = "#000000";

myOpacity = 0.3;

myWeight = 1;

}

if (os\_flows[id].flow > 100) {

myColor = "#EE0000";

myOpacity = 0.7;

myWeight = 2;

}

if (os\_flows[id].flow > 1000) {

myColor = "#EE0000";

myOpacity = 0.9;

myWeight = 3;

}

var flowLine = new google.maps.Polyline({

path: [myOrig,myDest],

strokeColor: myColor,

strokeOpacity: myOpacity,

strokeWeight: myWeight

});

flowLine.setMap(map);

}

**Figure 4: Modifications to line drawing map setup file**

As described above, a polyline is a line made up of a number of segments (in the above example, only one segment per line) defined as a set of co-ordinates. The step from this to a *polygon* is straightforward: a polygon is a line which returns to its initial starting point. Polygons have the additional characteristic beyond a line in that the y can have a fill colour (and variable fill opacity) as well as line characteristics.

Figure 5 shows the code of a set of files which illustrate the construction of basic polygons. The structure of these files is similar to those that we used before, but we will run through their contents.

The HTML file *wk8\_poly\_eg1.html* is illustrated first. It is very similar to the file use in the line drawing above, differing only in the names of the files loaded, and the on-page text.

The second file is a data file. It defines two polygons with some basic details, and a boundary property which contains an array of grid reference points.

**wk8\_poly\_eg1.html**

<!DOCTYPE html>

<html>

<script type="text/javascript" src="http://maps.google.com/maps/api/js?sensor=false"></script>

<script type="text/javascript" src="../jscoord-1.1.1/jscoord-1.1.1.js"></script>

<script type="text/javascript" src="wk8\_poly\_eg1\_data.js"></script>

<script type="text/javascript" src="wk8\_poly\_eg1\_setup.js"></script>

<link rel="stylesheet" media="all" href="../../../style/general.css" type="text/css">

<link rel="stylesheet" media="all" href="../../../style/google\_maps\_style.css" type="text/css">

<title>Google Maps examples - polygons</title>

</head>

<body onload="initialize()">

<h1>Google Maps examples - polygons</h1>

<p>[ <a href="../index.html">Personal home page</a> | <a href="index.html">Geog5870 work</a> ]</p>

<div class="gmap" id="map\_canvas" style="height: 300px; width: 400px"></div>

<p>This example looks at the process of adding a polygon</p>

</body></html>

**wk8\_poly\_eg1\_data.js**

/\*

\* A set of data which will be used to draw polygons

\*/

var os\_polydata = [

{

'title': 'poly1',

'description': 'poly1',

'value': 27,

'boundary': [

{'easting':430000,'northing': 433000},

{'easting':430000,'northing': 434000},

{'easting':431000,'northing': 434000},

{'easting':431000,'northing': 433000}]

},

{

'title': 'poly2',

'description': 'poly2',

'value': 40,

'boundary': [

{'easting':428000,'northing': 433000},

{'easting':428000,'northing': 434000},

{'easting':429000,'northing': 434000},

{'easting':429000,'northing': 433000}]

}

];

**wk8\_poly\_eg1\_setup.js**

var map; // The map object

var myCentreLat = 53.81;

var myCentreLng = -1.52;

var initialZoom = 12;

function initialize() {

var myCentre = new google.maps.LatLng(myCentreLat,myCentreLng);

var myOptions = {

zoom: initialZoom,

center: myCentre,

mapTypeId: google.maps.MapTypeId.ROADMAP

};

map = new google.maps.Map(document.getElementById("map\_canvas"),myOptions);

for (id in os\_polydata) {

var polyPath = []; // An empty array

/\*

\* Read through points

\*/

var thisBoundary = os\_polydata[id].boundary;

for (pt in thisBoundary) {

var osPt = new OSRef(thisBoundary[pt].easting,thisBoundary[pt].northing);

var llPt = osPt.toLatLng(osPt);

llPt.OSGB36ToWGS84();

var myLatLng = new google.maps.LatLng(llPt.lat,llPt.lng);

polyPath.push(myLatLng);

}

/\*

\* Construct the polygon

\*/

myPoly = new google.maps.Polygon({

paths: polyPath,

strokeColor: "#FF0000",

strokeOpacity: 0.8,

strokeWeight: 3,

fillColor: "#FF0000",

fillOpacity: 0.35

});

myPoly.setMap(map);

}

}

**Figure 5: Creating example polygons**

The final file illustrated in the Figure is *wk8\_poly\_eg1.js*.  Again, it is broadly similar to the previous examples. Instead of drawing lines or creating markers, the main loop in the initialize() function will create some polygons. For each iteration of the loop, the following happens:

* We create the variable polyPath as an empty array. This will contain the path that defines the polygon.
* We copy the boundary property of the current object being read from the data array. (We could work directly on os\_polydata[id].boundary below; this stage is done to make the code easier to read).
* We start a new loop (nested inside the main loop that is processing the data file) to read through the boundary object.
* For each point, we follow the previously illustrated process of conversion from OS grid reference to WGS84 Latlng.
* We add the new point to our path. This is done using polyPath.push(*point*). push() is a method attached to the generic Array object, and it adds the argument (in our case, a point) to the end of the array
* Having finished processing all of the boundary points, we construct a new google.maps.Polygon() object. This takes a number of parameters. The stroke parameters refer to the boundary line, whilst the fill parameters refer to the middle of the polygon.
* We add the polygon to the map using the setmap() method of the Polygon object (in the same way as we did with the lines).

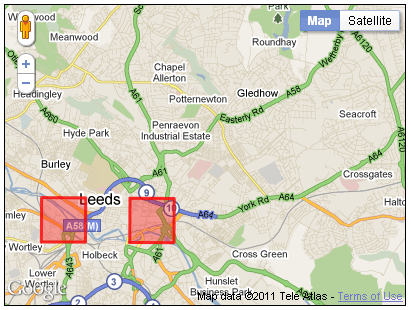


Figure 6: Adding polygons to the map

The map generated in this example is shown in Figure 6. It can be seen that the polygons are simple squares, with a solid red outline and semi-opaque fill.

**Summary**

On this page, we have seen examples of how we can add lines and polygons to a map. The lines example showed how a reasonably large number of lines can be added, whilst the polygon example used a simpler case. In the case of polygons, we can generate complex polygons as well as these very simple examples - you may note in Figure 5 that the boundary data was added to the polygon property paths rather than path as was the case with the polyline. With polygons, we can add multiple paths, and thus have polygon objects with more than one component; we can also define polygons with 'holes' in them.

On the next page , we shall look at how we can add information to polygons.

**Adding information to polygons**

The previous section illustrated the use of polygons by adding two simple polygons to the map.  However, the polygons provided no extra functionality, and were both shaded in the same way.  By shading polygons in different colours, we can add thematic meaning to the map, as we did with differential shading of lines.  We can also present further information by adding information to be included on a pop-up window. Figure 7 shows the code of a revised polygon example; this uses the same data file as the previous example.

This example adds a function for adding polygons to the map that is similar to the function used to add multiple markers. The function is called addPoly(), and it takes as its parameters an array of LatLng points (polyPath), some text to be used for an info window, a line colour and a fill colour. Again, we have to use a callback function to make sure that we get distinct infowindows for each polygon, however the function used is slightly different to that used in the markers example (and thus should really have been given a different name!). Infowindows can be attached to markers, or to any given LatLng point on the map; they cannot be attached in a general manner to a polygon: we have to say *where* we want them to pop up.

We can do this by setting the position parameter of the infowindow object. Here we have used the first point in our polygon (polyPath[0]), although a more appealing placement would probably require us to iterate through all points and calculate a centroid. Given that the position is set, we do not need to supply a marker reference to the callback function.

**wk8\_poly\_eg2\_setup.js**

var map; // The map object

var myPoly;

var myCentreLat = 53.81;

var myCentreLng = -1.52;

var initialZoom = 12;

// infocallback for info windows

function infoCallback(infowindow) {

return function() {

infowindow.open(map);

};

}

/\*

\* addPoly

\*/

function addPoly(polyPath,myInfo,line\_colour,fill\_colour) {

/\*

\* Construct the polygon

\*/

myPoly = new google.maps.Polygon({

paths: polyPath,

strokeColor: line\_colour,

strokeOpacity: 0.8,

strokeWeight: 3,

fillColor: fill\_colour,

fillOpacity: 0.35

});

var infowindow = new google.maps.InfoWindow({

content: myInfo,

position: polyPath[0]

});

google.maps.event.addListener(myPoly, 'click', infoCallback(infowindow));

myPoly.setMap(map);

}

function initialize() {

var myCentre = new google.maps.LatLng(myCentreLat,myCentreLng);

var myOptions = {

zoom: initialZoom,

center: myCentre,

mapTypeId: google.maps.MapTypeId.ROADMAP

};

map = new google.maps.Map(document.getElementById("map\_canvas"),myOptions);

for (id in os\_polydata) {

var polyPath = []; // An empty array

/\*

\* Read through points

\*/

var thisBoundary = os\_polydata[id].boundary;

for (pt in thisBoundary) {

var osPt = new OSRef(thisBoundary[pt].easting,thisBoundary[pt].northing);

var llPt = osPt.toLatLng(osPt);

llPt.OSGB36ToWGS84();

var myLatLng = new google.maps.LatLng(llPt.lat,llPt.lng);

polyPath.push(myLatLng);

}

var fillShade = 256\*(os\_polydata[id].value/100);

var fillShadeHex = parseInt(fillShade).toString(16);

var fillColour = '#'+fillShadeHex+fillShadeHex+fillShadeHex;

var info = "<div><h1>" + os\_polydata[id].title

+ "</h1><p>" + os\_polydata[id].description

+ "</p></div>";

addPoly(polyPath,info,"#000000",fillColour);

}

}

**Figure 7: Setup file for second polygon example**

The setup file also illustrates the dynamic generation of a fill shade from a value read from the data file (in this case, os\_polydata[id].value). First , we convert the value into a number between 0 and 256. The example above assumes that all values are in the range 0 to 100. We then convert this to a hexadecimal number: we use the builtin JavaScript function parseInt() to convert fillShade to an integer, and then run the method toString(16) on the result of parseInt(). The method toString() converts a number to a string. It can take an optional radix (base) value as a parameter; hexadecimal is base 16, so using toString(16) converts a numeric value to a string containing a hexadecimal version of that number. Finally, we construct a string called fillColour as a standard Google Maps colour reference: a '#' plus three hex values for red, green and blue. Here we have used the same value for red, green and blue, so we will get a variable shade of grey.

Having calculated a fill colour, we call our addPoly() function for each polygon.

The code in Figure 7 is relatively simple - it creates only shades of grey, and as mentioned, it assumes that the attribute term will vary between 0 and 100.  Some basic error correction for values outside this range is included.  The variable *fillValue* is used as a term to determine the shading level, and is iitially set as a copy of the raw data value.  If this value is less than 0 or greater than 100, it is modified to be 0 or 100 respectively. However, a better approach would be to have a preparatory loop that reads through the data file and determines the range of the shading variable, and then sets the shading accordingly.

**Increasing the number of polygons**

The above example used two polygons only.  It is possible to add more polygons, and also possible to add much more complex polygons than simple squares.  Look at Figure 8; it contains a larger data file, that contains boundary definitions for five wards in central and north Leeds. It is loaded by an HTML and map setup file that are identical to those used in the previous example apart from a) the designation of the appropriate data file and b) the initial zoom level specified.

**wk8\_poly\_eg3\_data.js**

var os\_polydata = [

{

'title': 'Kirkstall',

'description': '08DAFR',

'value': 27,

'boundary': [

{'easting':424937.406702199,'northing': 437355.594025358},

{'easting':425253.500142493,'northing': 437616.812329601},

{'easting':425343.187182577,'northing': 437517.593897509},

{'easting':425753.812206959,'northing': 437609.499945594},

{'easting':426193.406191368,'northing': 437497.90647349},

{'easting':426149.000431327,'northing': 437284.687145292},

{'easting':426434.000111593,'northing': 436914.812200947},

{'easting':426354.499823518,'northing': 436667.999528717},

{'easting':426469.593327626,'northing': 436420.281640487},

{'easting':426607.312111754,'northing': 436494.812456556},

{'easting':426763.000047899,'northing': 436317.593896391},

{'easting':427104.000240217,'northing': 436471.312680534},

{'easting':427125.687536237,'northing': 436375.312680445},

{'easting':427039.812848157,'northing': 436340.687144413},

{'easting':427186.000112293,'northing': 436297.499944372},

{'easting':427056.906480173,'northing': 436077.094184167},

{'easting':427119.687920231,'northing': 435708.093735823},

{'easting':427820.687600884,'northing': 435260.500263407},

{'easting':428024.592625074,'northing': 434796.093734974},

{'easting':428505.687281522,'northing': 434549.687590744},

{'easting':428096.906481141,'northing': 434024.187174255},

{'easting':427612.593392690,'northing': 434195.312934414},

{'easting':427315.093744413,'northing': 434180.687142401},

{'easting':427323.906288421,'northing': 434343.187750552},

{'easting':427155.999984265,'northing': 434714.500390898},

{'easting':426998.499568118,'northing': 434630.40643882},

{'easting':426748.812527886,'northing': 434694.593830879},

{'easting':426432.844015591,'northing': 435168.656679321},

{'easting':426276.312303446,'northing': 435118.312743274},

{'easting':426101.093615282,'northing': 435268.406567414},

{'easting':425949.781231142,'northing': 435610.500391732},

{'easting':426075.812079259,'northing': 435800.499495909},

{'easting':426049.687791235,'northing': 435984.687400081},

{'easting':425899.312367095,'northing': 436006.593832101},

{'easting':425253.812462493,'northing': 436409.000232476},

{'easting':424902.094062166,'northing': 436409.093416476},

{'easting':424774.906094047,'northing': 436665.499944715},

{'easting':424453.312749748,'northing': 436835.093800873},

{'easting':424557.812973845,'northing': 437049.499945073},

{'easting':424937.406702199,'northing': 437355.594025358}]

},

{

'title': 'Weetwood',

'description': '08DAGG',

'value': 12,

'boundary': [

{'easting':428588.310769599,'northing': 436488.17488855},

{'easting':427825.078512888,'northing': 436514.798888575},

{'easting':427480.093936567,'northing': 436298.187048373},

{'easting':427080.406256195,'northing': 436298.687784373},

{'easting':427104.000240217,'northing': 436471.312680534},

{'easting':426763.000047899,'northing': 436317.593896391},

{'easting':426607.312111754,'northing': 436494.812456556},

{'easting':426500.312303654,'northing': 436394.500392462},

{'easting':426354.499823518,'northing': 436667.999528717},

{'easting':426434.000111593,'northing': 436914.812200947},

{'easting':426149.000431327,'northing': 437284.687145292},

{'easting':426193.406191368,'northing': 437497.90647349},

{'easting':425753.812206959,'northing': 437609.499945594},

{'easting':425296.000238533,'northing': 437540.21917753},

{'easting':425253.500142493,'northing': 437616.812329601},

{'easting':425355.000046588,'northing': 437751.500073726},

{'easting':425279.398126517,'northing': 437987.673385946},

{'easting':425116.593390365,'northing': 438137.312554086},

{'easting':425330.406638565,'northing': 438450.719018377},

{'easting':425620.094190834,'northing': 438671.094058583},

{'easting':425642.186990855,'northing': 439044.906282931},

{'easting':425905.313007100,'northing': 439419.78141928},

{'easting':426246.593775418,'northing': 438970.593578862},

{'easting':426331.187439497,'northing': 438930.280746824},

{'easting':426329.906415496,'northing': 439020.905770909},

{'easting':426453.093615610,'northing': 439048.187178934},

{'easting':426837.093615968,'northing': 438700.905770611},

{'easting':426888.312048016,'northing': 439131.291947011},

{'easting':427088.000240202,'northing': 438942.593322836},

{'easting':427312.812272411,'northing': 438946.594090839},

{'easting':427545.687280628,'northing': 438691.093802601},

{'easting':427525.000432609,'northing': 438599.812394516},

{'easting':427944.093936999,'northing': 438520.812842443},

{'easting':428393.406705417,'northing': 437860.187433828},

{'easting':428799.312113795,'northing': 437770.313001744},

{'easting':429095.405810071,'northing': 437457.312041452},

{'easting':429461.999858413,'northing': 437295.094057301},

{'easting':429220.593906188,'northing': 437129.312553147},

{'easting':428915.500273904,'northing': 436631.218472683},

{'easting':428588.310769599,'northing': 436488.17488855}]

},

{

'title': 'City and Holbeck',

'description': '08DAFH',

'value': 93,

'boundary': [

{'easting':429562.687730506,'northing': 431042.406691478},

{'easting':429717.312754650,'northing': 430761.999651217},

{'easting':429567.094002511,'northing': 430677.687587138},

{'easting':429214.593266182,'northing': 431303.999779722},

{'easting':429020.093682001,'northing': 431191.405859617},

{'easting':428678.781169683,'northing': 430801.218851254},

{'easting':427864.406256925,'northing': 430527.499554999},

{'easting':427553.905904635,'northing': 431239.312675662},

{'easting':427250.594032353,'northing': 431537.312035939},

{'easting':427716.499696787,'northing': 432110.187812473},

{'easting':427565.593840646,'northing': 432273.187108624},

{'easting':427419.312368510,'northing': 432894.000421202},

{'easting':427022.405872140,'northing': 433102.718245397},

{'easting':427928.593648984,'northing': 432889.499941198},

{'easting':427947.812081002,'northing': 433170.28074146},

{'easting':428078.405873124,'northing': 433261.187365544},

{'easting':428015.187185065,'northing': 433437.312293708},

{'easting':428082.312433128,'northing': 433560.812837823},

{'easting':428181.812465220,'northing': 433542.906149807},

{'easting':427973.999857027,'northing': 433779.406118027},

{'easting':427899.000048957,'northing': 434093.499686319},

{'easting':428096.906481141,'northing': 434024.187174255},

{'easting':428196.187377234,'northing': 434190.09360641},

{'easting':429037.499634017,'northing': 433532.999973798},

{'easting':429369.906418327,'northing': 433737.000229988},

{'easting':429461.312754412,'northing': 434090.093862316},

{'easting':429607.093490548,'northing': 434279.405862493},

{'easting':430017.400050930,'northing': 434417.500454621},

{'easting':430293.812467187,'northing': 434301.812006514},

{'easting':430468.406515350,'northing': 433935.281446172},

{'easting':431195.187444027,'northing': 433696.71914195},

{'easting':430802.499827661,'northing': 433344.594213622},

{'easting':430740.906227604,'northing': 433151.094053442},

{'easting':431137.406195973,'northing': 432875.000101185},

{'easting':431394.594036213,'northing': 432411.000100753},

{'easting':430918.312179769,'northing': 432293.187876643},

{'easting':430998.687987844,'northing': 432072.499492437},

{'easting':430451.948787334,'northing': 431733.817636122},

{'easting':430573.687027448,'northing': 431419.187491829},

{'easting':430482.093299363,'northing': 431150.093603578},

{'easting':430083.687666992,'northing': 431188.593955614},

{'easting':430056.593650966,'northing': 431070.093603504},

{'easting':429562.687730506,'northing': 431042.406691478}]

},

{

'title': 'University',

'description': '08DAGF',

'value': 45,

'boundary': [

{'easting':429531.219186477,'northing': 436038.312232131},

{'easting':430048.093426958,'northing': 435681.406247798},

{'easting':430425.312499310,'northing': 435045.405991206},

{'easting':430532.406515409,'northing': 435145.187623299},

{'easting':430613.999859485,'northing': 434975.499559141},

{'easting':431034.093811877,'northing': 435060.09424722},

{'easting':431190.406388022,'northing': 434975.187239141},

{'easting':431574.531316380,'northing': 435197.843751348},

{'easting':431881.999604666,'northing': 435142.312231296},

{'easting':431912.500468695,'northing': 434810.281254987},

{'easting':432177.187060941,'northing': 434623.406374813},

{'easting':431578.406132384,'northing': 434002.312486235},

{'easting':431335.777524158,'northing': 433927.770406165},

{'easting':431195.187444027,'northing': 433696.71914195},

{'easting':430468.406515350,'northing': 433935.281446172},

{'easting':430209.687795109,'northing': 434368.687398576},

{'easting':429791.499506720,'northing': 434387.188006593},

{'easting':429491.187954440,'northing': 434143.094054366},

{'easting':429369.906418327,'northing': 433737.000229988},

{'easting':429037.499634017,'northing': 433532.999973798},

{'easting':428210.312433247,'northing': 434182.281510402},

{'easting':428474.593521493,'northing': 434435.093798638},

{'easting':428884.187377874,'northing': 435353.499943493},

{'easting':429054.276850033,'northing': 435491.319079621},

{'easting':429119.187186093,'northing': 435970.312488068},

{'easting':429223.500018190,'northing': 435939.687720039},

{'easting':429345.406194304,'northing': 436164.906280249},

{'easting':429531.219186477,'northing': 436038.312232131}]

},

{

'title': 'Headingley',

'description': '08DAFN',

'value': 67,

'boundary': [

{'easting':428573.898993585,'northing': 434715.399462899},

{'easting':428505.687281522,'northing': 434549.687590744},

{'easting':428024.592625074,'northing': 434796.093734974},

{'easting':427820.687600884,'northing': 435260.500263407},

{'easting':427119.687920231,'northing': 435708.093735823},

{'easting':427056.906480173,'northing': 436077.094184167},

{'easting':427186.000112293,'northing': 436297.499944372},

{'easting':427545.312496628,'northing': 436310.187304384},

{'easting':427825.078512888,'northing': 436514.798888575},

{'easting':428732.687601733,'northing': 436505.687336566},

{'easting':428915.500273904,'northing': 436631.218472683},

{'easting':428999.785713982,'northing': 436842.75741688},

{'easting':429196.593394165,'northing': 436686.812456735},

{'easting':429106.999538082,'northing': 436578.187560634},

{'easting':429240.500466206,'northing': 436392.218920461},

{'easting':429444.312306396,'northing': 436317.09418439},

{'easting':429281.593586245,'northing': 435994.09488809},

{'easting':429119.187186093,'northing': 435970.312488068},

{'easting':429054.276850033,'northing': 435491.319079621},

{'easting':428943.187185929,'northing': 435455.312167588},

{'easting':428573.898993585,'northing': 434715.399462899}]

}

]

**Figure 8:  mapdata file for example 3**

The co-ordinates were derived in a fairly lengthy manual manner: the polygon data were derived from a set of polygon boundaries downloaded from [UKBorders](http://edina.ac.uk/ukborders/) (although fortunately, there are easier ways than this of converting polygon data). The MIF form presents the polygons as a series of easting, northing pairs, with one pair per line.  For a relatively small set of wards such as this, it was possible to then edit the resulting file manually, to prefix each line with "*{'easting':* " and to add other bits of text as required. However, this is time consuming and is clearly not a suitable approach for larger amounts of data. The other properties (the title, description, and fill colour) was added manually for each polygon.

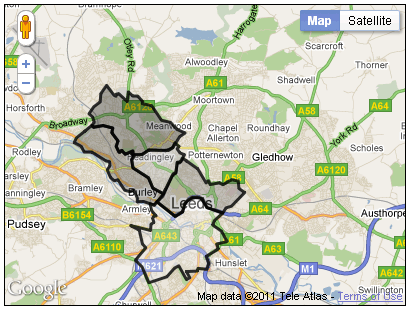


Figure 9:  Output of example 3

The output of this example is shown in Figure 9. For the sake of simplicity, the polygons have been shaded in various intensities of grey. It would of course be quite easy to work out different reg, green and blue values, in order to use colours.

Whilst polygons are useful, there are limits on how many can be added to a map before it becomes unwieldy (slow to load and respond to clicks).

The files used for these polygon examples are available here:

* Example 1
  + [wk8\_poly\_eg1.html](https://vlebb.leeds.ac.uk/bbcswebdav/users/geo6odw/geog58705871_2010-11/wk8/wk8_examples/poly_examples/wk8_poly_eg1.html)
  + [wk8\_poly\_eg1\_data.js](https://vlebb.leeds.ac.uk/bbcswebdav/users/geo6odw/geog58705871_2010-11/wk8/wk8_examples/poly_examples/wk8_poly_eg1_data.js)
  + [wk8\_poly\_eg1\_setup.js](https://vlebb.leeds.ac.uk/bbcswebdav/users/geo6odw/geog58705871_2010-11/wk8/wk8_examples/poly_examples/wk8_poly_eg1_setup.js)
* Example 2
  + [wk8\_poly\_eg2.html](https://vlebb.leeds.ac.uk/bbcswebdav/users/geo6odw/geog58705871_2010-11/wk8/wk8_examples/poly_examples/wk8_poly_eg2.html) [uses same data file as example 1]
  + [wk8\_poly\_eg2\_setup.js](https://vlebb.leeds.ac.uk/bbcswebdav/users/geo6odw/geog58705871_2010-11/wk8/wk8_examples/poly_examples/wk8_poly_eg2_setup.js)
* Example 3
  + [wk8\_poly\_eg3.html](https://vlebb.leeds.ac.uk/bbcswebdav/users/geo6odw/geog58705871_2010-11/wk8/wk8_examples/poly_examples/wk8_poly_eg3.html)
  + [wk8\_poly\_eg3\_data.js](https://vlebb.leeds.ac.uk/bbcswebdav/users/geo6odw/geog58705871_2010-11/wk8/wk8_examples/poly_examples/wk8_poly_eg3_data.js)
  + [wk8\_poly\_eg3\_setup.js](https://vlebb.leeds.ac.uk/bbcswebdav/users/geo6odw/geog58705871_2010-11/wk8/wk8_examples/poly_examples/wk8_poly_eg3_setup.js)

**Additional overlay types**

So far we have looked at three types of overlay: markers, lines and polygons. There are several other sorts of overlay supported by Google Maps, which we will consider briefly on this page.

**Circles**

A specialised form of polygon overlay is a circle. Circles can be added to a map given a centre and a radius (and line and fill colours); they are made available through the object [google.maps.Circle()](http://code.google.com/apis/maps/documentation/javascript/reference.html#Circle). Figure 10 shows an HTML document, a map setup file, and part of an associated data file. This map draws a set of markers representing (a subset of) secondary schools in the London Borough of Waltham Forest. Like most local authorities, distance from a school is one of the criteria used when selecting pupils for over-subscribed schools; each year data are published showing a 'cut-off' distance: the distance beyond which no pupils were accepted in the previous year's entry. A circle is thus a good way of illustrating the areas from which schools draw pupils (although of course, pupils aren't uniformly distributed in this area, and pupils may be accepted from outside this area on other selection criteria). The example also illustrates the dynamic modification of a web page using JavaScript, given some input data.

The first file shown is the containing HTML document. This is similar to those used before; the only element that might be worthy of initial note is the empty <OL> list element at the bottom of the page. The second file illustrated shows the structure of the data file; the first two entries are shown. Only the name of the school and a URL if available are included along with the location and cut-off distance, although clearly one could include a variety of other performance data. The locations have already been converted to LatLng references.

The third file illustrated is the map setup file. A lot of this repeats aspects that we have already seen in earlier examples, on which we do not need to dwell. In this example, the markers are constructed as part of the main loop, rather than through a specific 'addMarker' function. The loop has the following significant characteristics:

1. Prior to starting the loop, we declare an empty string variable: var schools='';. We will use this in due course...
2. For each entry that is read from the data file, we first create a marker, as normal.
3. We also generate some info text from the supplied school information, and create an infowindow object.
4. We append a string to the schools string: 'schools = schools + '<li>' + markers[id].title;'. This adds '<li>', which denotes an HTML list item, and the name of the school.
5. We create the circle object
6. We bind the circle to a marker.

The circle object declaration is similar to those that we have used for polygons. The options can include an explicit centre (as a LatLng object), but in this example we have only declared a radius (and colour information). Hence, we need the second stage - that of binding the circle to the marker, using the bindTo() method (an underlying method for objects in the maps API). The binding process is more complex than we necessarily need (we could just identify a centre position), but allows the circle visibility to be based on that of the marker. Note that we do not explicitly add the circle to the map in this example, as the marker to which it is bound has already been added.

Finally, in this example, we use a general JavaScript method to set the innerHTML property of the empty list that was added to our HTML page. Thus, without knowing in the HTML document how many schools we would use, or what they are called, we are able to update the text on the screen with a list of names.

**wk8\_lbwf\_schools.html**

<!DOCTYPE html>

<html>

<meta http-equiv="content-type" content="text/html; charset=utf-8" />

<script type="text/javascript"

src="http://maps.google.com/maps/api/js?sensor=false">

</script>

<script src="wk8\_lbwf\_schools\_data.js" type="text/javascript"></script>

<script src="wk8\_lbwf\_schools\_setup.js" type="text/javascript"></script>

<link rel="stylesheet" media="all" href="../../../style/general.css" type="text/css">

<link rel="stylesheet" media="all" href="../../../style/google\_maps\_style.css" type="text/css">

<title>School catchment areas</title>

</head>

<body onload="init()">

<h1>School catchment areas</h1>

<p>This map shows catchment areas for a selection of secondary schools in Waltham Forest, using cut off distances as applied for intake in September 2010.</p>

<div class="gmap" id="map\_canvas" style="width: 600px; height: 400px;"></div>

<h2>Notes</h2>

<ul>

<li>The colours of the map pins and the associated circles are randomly assigned. If any particular pin is too dark, then the page can be re-loaded in order to assign new colours

<li>Not all schools are currently included on the map

<li>Some schools (those which are not over-subscribed) do not have a distance cut-off

<li>The distance data is taken from <a href="http://www.babcockwf.co.uk/idocument/stream\_document.aspx?ObjectId=338373">Moving on - a guide to applying

for a secondary school place for September 2011</a> published by the London Borough of Waltham Forest

</ul>

<h2>Schools</h2>

<ol start="0" id=school\_list>

</ol>

</body>

</html>

**wk8\_lbwf\_schools\_data.js** (excerpt)

var markers = [

{

'latitude': 51.5567731,

'longitude': 0.0162396,

'url': 'http://www.buxtonschool.org.uk/',

'title': 'Buxton School',

'radius': 0

},

{

'latitude': 51.629601,

'longitude': -0.007813,

'url': 'http://www.chingford-school.co.uk/',

'title': 'Chingford Foundation School',

'radius': 4087

},

...

**wk8\_lbwf\_schools\_setup.js**

/\*

\* The starting position and initial zoom factor

\*

\* These could be determined using the supplied data...

\*/

var centreLat = 51.6;

var centreLong = -0.02;

var initialZoom = 11;

// Define the map object

var map;

/\* generate random colours

\*/

function genHex(){

colors = new Array(15);

colors[0]="0";

colors[1]="1";

colors[2]="2";

colors[3]="3";

colors[4]="4";

colors[5]="5";

colors[6]="6";

colors[7]="7";

colors[8]="8";

colors[9]="9";

colors[10]="a";

colors[11]="b";

colors[12]="c";

colors[13]="d";

colors[14]="e";

colors[15]="f";

digit = new Array(5);

color="";

for (i=0;i<6;i++){

digit[i]=colors[Math.round(Math.random()\*15)];

color = color+digit[i];

}

return color;

}

function init() {

function infoCallback(infowindow, marker) {

return function() {

infowindow.open(map, marker);

};

}

var latlng = new google.maps.LatLng(centreLat, centreLong);

var myOptions = {

zoom: initialZoom,

center: latlng,

mapTypeId: google.maps.MapTypeId.ROADMAP

};

var map = new google.maps.Map(document.getElementById("map\_canvas"),myOptions);

var schools = '';

for(id in markers) {

var contentString = '<div class="iw">'+

'<h1 class="iw\_h1">'+ markers[id].title+'</h1>'+

'<ul>'+

'<li>www: '+'<a href="' + markers[id].url+ '">'+ markers[id].title +'</a>'+

'</ul>'+

'</div>';

var infowindow = new google.maps.InfoWindow({content: contentString});

myColour=genHex();

myLatlng=new google.maps.LatLng(markers[id].latitude,markers[id].longitude);

var marker = new google.maps.Marker({

position: myLatlng,

map: map,

icon: 'http://chart.apis.google.com/chart?chst=d\_map\_pin\_letter&chld='+id+'|'+myColour+'|000000',

title: markers[id].title

});

schools = schools + '<li>' + markers[id].title;

google.maps.event.addListener(marker, 'click', infoCallback(infowindow, marker));

// Add a Circle overlay to the map.

var circle = new google.maps.Circle({

map: map,

fillColor: '#'+myColour,

fillOpacity: 0.2,

strokeWeight: 1,

radius: markers[id].radius

});

circle.bindTo('center', marker, 'position');

}

document.getElementById('school\_list').innerHTML = schools;

}

**Figure 10: Files used for google.maps.Circle() example**

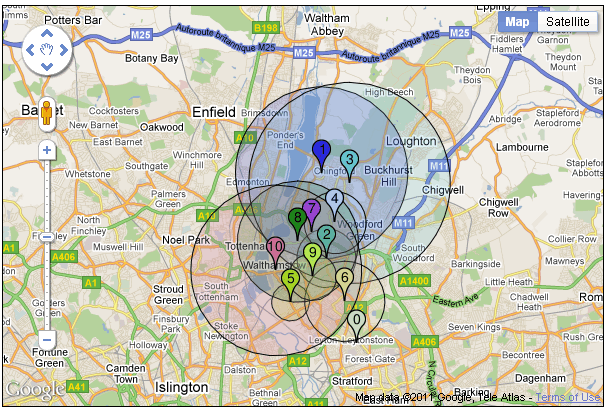


Figure 11: Output of circles map

Figure 11 shows the map produced by this example. The files used in this example are available here:

* wk8\_lbwf\_schools.html
* wk8\_lbwf\_schools\_data.js
* wk8\_lbwf\_schools\_setup.js

Circle overlays are useful for quickly and easily conveying quantitative information, by making the area proportionate to a known value of a variable at a given location. The area of a circle is *a=πr2*,so if the value to be mapped has value *a*, then the radius should be set to *r=√(a/π)*.

**Ground Overlay**

Another form of overlay is a ground overlay, which allows us to place an image over the map, which may be useful in some cases. Figure 12 shows the setup file for a map which uses a ground overlay. This is provided via [google.maps.GroundOverlay()](http://code.google.com/apis/maps/documentation/javascript/overlays.html#GroundOverlays). The setup file is loaded by an HTML page similar to those used in our other examples.

The map overlays a historic map of bomb damage in the London in the Second World War over the regular base map.

The code is relatively simple. Having done the usual map creation, we use google.maps.LatLngBounds() to create a bounding box. This is defined by two co-ordinates, the south-west and north-east points of a box; the points *must* be given in this order. It is up to the developer to work out the correct co-ordinates for the image to bve overlain!

We then create the overlay, supplying two parameters: the name of an image file (or a URL) and the bounding box that we have created. Finally, we add the image to the map using setMap() as in most of our previous examples.

wk8\_groundoverlay\_setup.js

/\*

\* The starting position and initial zoom factor

\*/

var centreLat = 51.5615;

var centreLong = -0.088;

var initialZoom = 15;

// Define the map object

var map;

/\*

\* init() function - this is where we start

\*/

function init() {

var latlng = new google.maps.LatLng(centreLat, centreLong);

var myOptions = {

zoom: initialZoom,

center: latlng,

mapTypeId: google.maps.MapTypeId.ROADMAP

};

var map = new google.maps.Map(document.getElementById("map\_canvas"),myOptions);

var imageBounds = new google.maps.LatLngBounds(

new google.maps.LatLng(51.55546,-0.09130),

new google.maps.LatLng(51.56590,-0.07980)

);

var myOverlay = new google.maps.GroundOverlay("bomb\_crop.jpg", imageBounds);

myOverlay.setMap(map);

}

**Figure 12: Using a ground overlay**

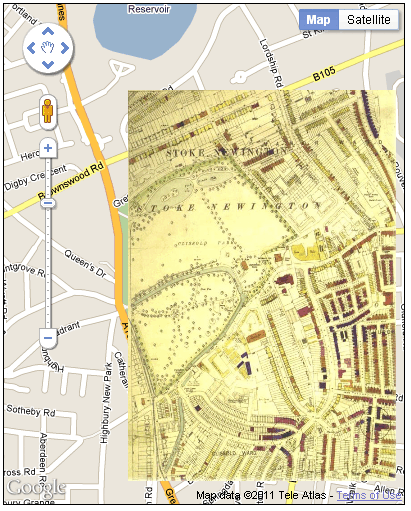


Figure 13: Bomb damage map

The results of this map are shown in Figure 13. The pattern of bomb damage can be easily seen, and the image remains present if the user switches to satellite view. Near the bottom right iof the ov the overlay image is a circle indicating a bomb impact location. A key attached to the original image noted colour coding of households, from total destruction (the darkest shades) to minor damage.

By default, the opacity of the overlay image cannot be altered, which limits it's usefulness although this page:

* <http://www.usnaviguide.com/v3maps/ProjectedOverlayTest.htm>

illustrates modifications which permit variation in opacity. However, it would also be possible to add a button to the HTML page which linked it's 'onclick' event to a JavaScript function that toggled the visibility of the overlay.

The files used in this example are:

* [bomb\_map.html](https://vlebb.leeds.ac.uk/bbcswebdav/users/geo6odw/geog58705871_2010-11/wk8/wk8_examples/overlay/bomb_map.html)
* [wk8\_groundoverlay\_setup.js](https://vlebb.leeds.ac.uk/bbcswebdav/users/geo6odw/geog58705871_2010-11/wk8/wk8_examples/overlay/wk8_groundoverlay_setup.js)
* [bomb\_crop.jpg](https://vlebb.leeds.ac.uk/bbcswebdav/users/geo6odw/geog58705871_2010-11/wk8/wk8_examples/overlay/bomb_crop.jpg)

**Using a raster overlay**

One of the earlier examples showed how ward boundaries could be added to map as polygon overlays.  However, only 5 generalised wards were used, and it was noted that this was close to the practical limit as to the number of polygons that could be added to a map.This is not ideal - clearly we may want to add many polygons to a map.

One approach is to create a raster image that can be used as an overlay layer, in which the raster image contains depictions of the polygons involved.  Using this approach, a large number of polygons can be rendered, although it may be necessary to break up the raster image into a number of tiles.

**Google Maps Creator**

Note: The version of Google Maps Creator used here produced code suitable for Google Maps API V2 (rather than V3, as in the rest of this module). Version 2 of the API has some major differences - I am investigating how easy it is to make the utility compatible with V3.

A useful tool is available from the [Centre for Advanced Spatial Analysis](http://www.casa.ucl.ac.uk/) (CASA) at UCL, which we will use for this example.  Google Maps Creator can be downloaded and installed on a PC (or Mac or Linux box), although administrator rights are required to install (in Windows at least) [ It appears that it can be installed as a regular user, but I experienced difficulties in running the software when doing so ].  The package can be downloaded from:

* <http://www.casa.ucl.ac.uk/software/gmapcreator.asp>

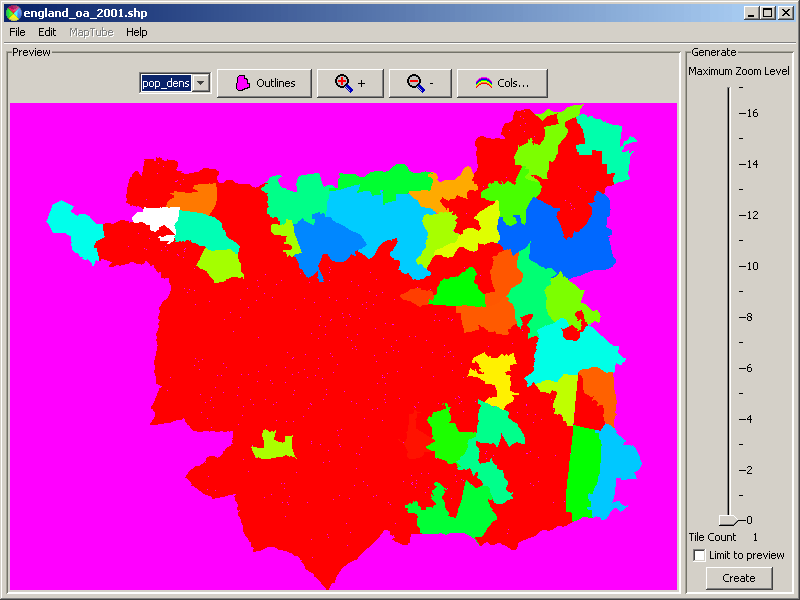
The package comes with help instructions, and so the process will be outlined here only in brief.

The tool takes as input a ESRI Shapefile, so it is necessary to prepare a coverage that you wish to use with ArcMap or equivalent.  Google Maps Creator allows you to select an attribute data item from the shapefile, and then use this to shade the map.  It is not possible to carry out any data transformations within Google Maps Creator, so it is necessary to ensure that any derived fields that are required have already been created in ArcMap.

On first loading Google Maps Creator, the **File / Open** command can be used to locate and load a shapefile.  This will be loaded and previewed on screen.  It is then necessary to choose a data item to use to shade the map.  By default, shading categories will be created between 0 and 1, regardless of the range of values that are actually observed in the selected variable.  The data ranges cannot be explored in Google Maps Creator, so it is important to know the ranges of observed values before starting.

The **Cols** button on the map interface can be used to change the shading categories, and also to assign category labels, and to change the colours used for shading.

On the right hand side of the map, a slider is used to select the maximum zoom level for which tiles will be generated.  The higher the zoom level, the more detail will be apparent as a user zooms into the map created.  However, the number of tiles that are created increases rapidly with an increase in zoom level, and so it is important to strike a balance between maximum zoom level and the number of tiles to be generated.

[](https://vlebb.leeds.ac.uk/bbcswebdav/users/geo6odw/geog58705871_2010-11/wk8/wk8_files/casa1.gif)  
Figure 14:  Using CASA Google Maps Creator

As an example, the shapefile containing Leeds Output Areas, with population density added as a field was used with Google Maps Creator.  This file was also used in the examples using MapServer in [week 3](https://vlebb.leeds.ac.uk/bbcswebdav/users/geo6odw/geog58705871_2010-11/wk3/index.html) of this module.  Figure 14 shows the state of the application when the shapefile has first been loaded, and population density selected as the shading variable.

A shading scheme was developed that used the same category boundaries and same colours as were used in the MapServer example with the same data set.

The maximum zoom level was set to 13: this is not the maximum level that can be zoomed to, but is a relatively small scale.  Experimentation is required in practice to help decide what the best setting for maximum zoom level will be for any chosen application.  You should not leave the setting at 0, as otherwise you will not be able to zoom into the map that is created at all.

The Google Maps Creator application will create an html page and all the associated files that are required to tile the shapefil on to a Google Map. The **Edit** menu should be used to set the correct API key for the use with the map (i.e. the API key you have registered for a given directory). When the **Create** button is pressed, the application will generate the html page and associated tile images in the same directory from which the original shapefile was read.

The html file will be called *shapefile\_name.html*, and there will be a directory *shapefile\_name-tiles* which contains the image files used for presenting on the map.  There will also be an image created featuring the UCL logo. These files should then be moved to an appropriate directory from which they can be served on the web.

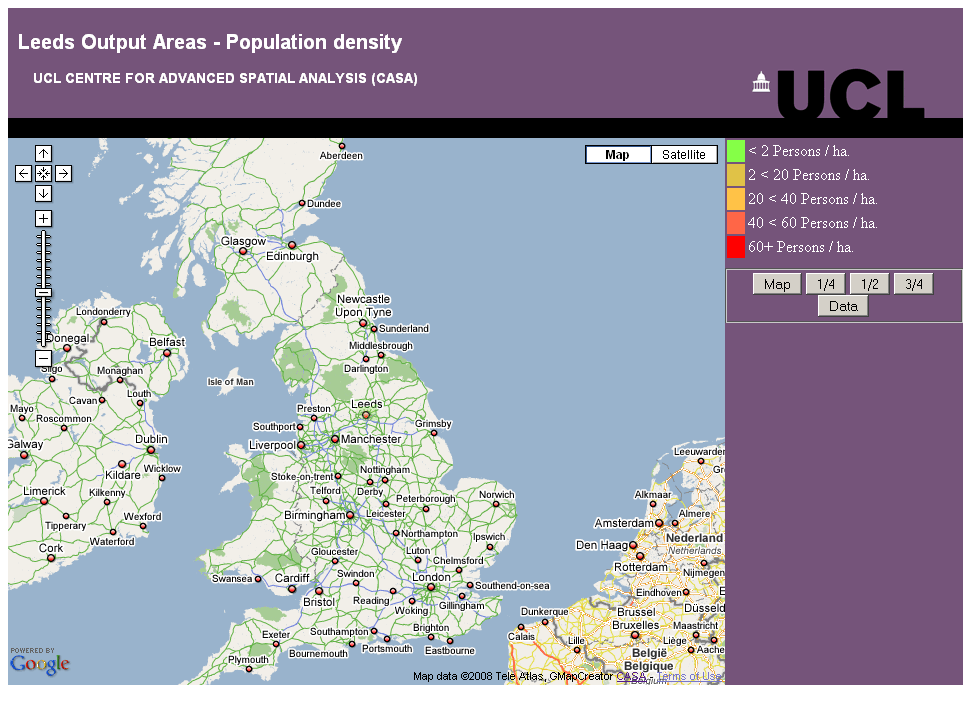
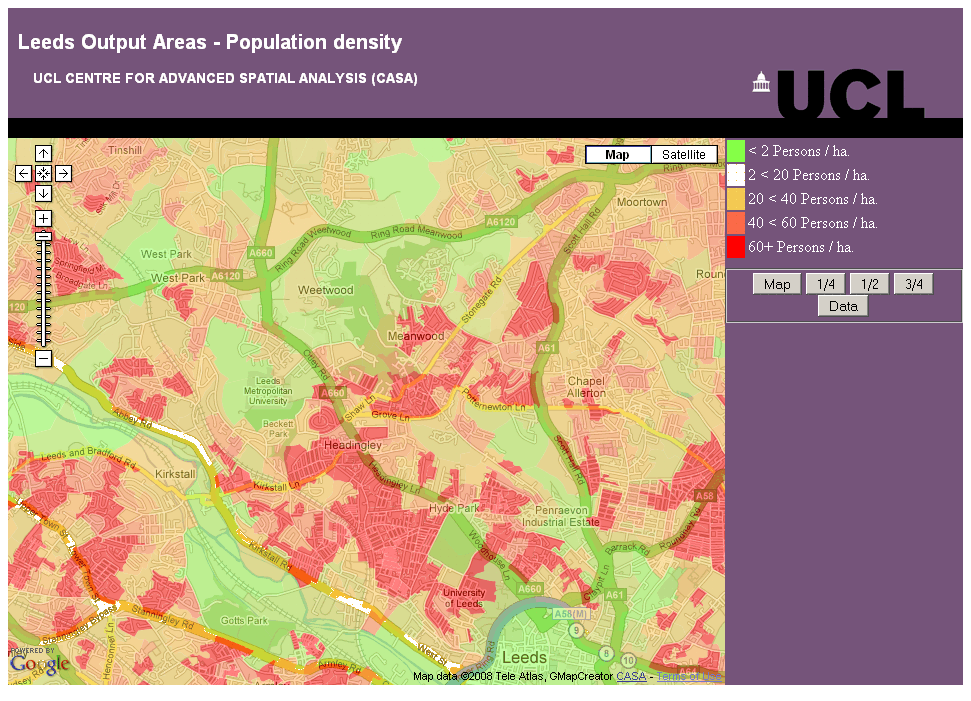
[](https://vlebb.leeds.ac.uk/bbcswebdav/users/geo6odw/geog58705871_2010-11/wk8/wk8_files/fig_14a.GIF)  
a) Initial appearance  
[](https://vlebb.leeds.ac.uk/bbcswebdav/users/geo6odw/geog58705871_2010-11/wk8/wk8_files/fig_14b.GIF)  
b) Zoomed in  
  
Figure 15:  Results of Google Maps Creator example

Figure 15 shows the results of using the Leeds OAs shapefile with the CASA Google Maps Creator.  The map initially appears zoomed out to cover most of Britain, and the Leeds raster layer is barely visible.  If the user zooms into Leeds, the raster overlay layer becomes more apparent.  Figure 15b shows the state of the map when it has been zoomed into the maximum extent.  This zoom level depends on the value selected for maximum zoom level in the Google Maps Creator application.

**Summary**

In previous work with Google Maps, we looked at the way in which markers can be created and added to a map. This week, we have looked at the process of creating and adding polygons. We started by creating some very simple polygons on a map, and then progressed on to making these polygons clickable, and to have them coloured in different ways.  
  
The initial examples used simple squares for the sake of illustration. This is not generally very useful, and so the next example used ward boundaries as an example. It was demonstrated how these can be added to a map, although the process of conversion from boundary data a given desktop GIS format to the form that we want to use is not terribly easy. A more siginficant problem is that only a small number of polygons can practically by added to a map.

An alternative method - to be illustrated in future material - is to use a KML layer. We can link to an external file of boundary and location data, and render it as an overlay on our map. A number of products including ArcGIS can export KML.

Another way around the problem is to convert the polygons into a raster image, and to use that as an overlay.  This is a useful approach, and we have looked at one tool which automates this procedure.  A disadvantage, compared to the 'real' polygons being rendered as map elements in the earlier examples is that the polygons are not clickable objects.

The final part of the lecture demonstrated the way in which a utility called CASA Google Maps Creator was used to create a detailed overlay; we shall look at the technical aspects of this next week.