Lab: Creating Line Mashups

GEOG 319/658 **Exercise #8** **FALL 2014**

Creating Line Mashups

Due: Monday, November 10

Download the code for Chapter 14 of the Peterson text from http://maps.unomaha.edu/cloud/code.html

A. A simple flow map

Edit the code for Figure 14.2 (use file 14_02_Flow_Map in the folder A-lines) to solve the problem described below.

Assume that you wish to portray the transport of goods via truck between Chicago and three cities, and that the total value of goods transported is as follows:

Chicago-Minneapolis: 5,000
Chicago-Omaha: 1,000
Chicago-Detroit: 10,000

1. Document the code to indicate which pairs of cities are handled by each section of the code.
2. Specify strokeWeights that produce lines directly proportional to the above data values.
3. Change the color to a shade of red for all flows.
4. Specify an opacity value of 0.5 for all flows.
5. Change the variable names to reflect that you are dealing with trucking flows rather than airline data.
6. Document the sections of code that cause the flow lines to actually appear on the map (as opposed to defining the coordinates making up each flow line).
7. Add code that will permit arrows to be shown at the end of lines so that the direction of the flows is clear. (Here you can use Google to search for code that will allow you to do this in the Google Maps API.)
8. Document the code that you have added.

B: Mercator vs Geodesic Projection—Flight trip and distance

Run the application 14_03_b_Great_Circle -_input (in the folder A-lines) so that you become familiar with its “look and feel” and what it accomplishes.

Modify the code for 14_03_b_Great_Circle so that it will allow a user to enter multiple points and draw the great circle routes associated with each pair of points. The user should also be able to click on a button called New Trip that allows a new set of points to be specified. Rather than showing Origin, latitude Destination, and longitude Heading at the bottom of the application, the user should see boxes for the Number of Stops and the total Distance. Finally, you also will want to change the heading found at the top of the application.

This can be accomplished as follows:

1. In the “var” section of the code, add a declaration for “markers”, which will be used to store an array of markers to indicate the origin and stops.
2. Before the code for the listener, add code that will create a new array for “markers”.
3. Modify the “addLocation” function as follows:

a. Delete the following code

```
if (clickcount == 1) {
    addOrigin(event);
                    }
if (clickcount == 2) {
    addDestination(event);
 }
```

b. Replace this code with code that will do the following

1) Add a marker at the location. The “title” of the marker should appear as # followed by the “clickcount” when the user hovers over the marker.

2) Add the marker to the markers array (use the “push” method).

3) Add the location to the two polylines (this code can be found in the AddOrigin function). Just move the code you need to your function.

4) Update trip stops and distance. You will call the following function, which we will explain in class

```
function updateStopsAndDistance() {
   var path = poly.getPath();
   var pathSize = path.getLength();
   var geodesicDist =   google.maps.geometry.spherical.computeLength(path);
   document.getElementById('distance').value = (geodesicDist/1600).toFixed(2);
   document.getElementById('stops').value = pathSize-1;
   }
```

4. Delete the functions “addOrigin” and “addDestination”.

5. Retain the “clearPaths” function, but rename it “newTrip”. Add to this function code that will accomplish the following:

a. Remove the markers from the map.
To accomplish this, create a loop that will go through each marker and set it to null (e.g., use markers[i].setMap(null);)

b. Remove the markers from the marker array

```
while (markers.length>0) {
    markers.pop();
  }
```

c. Reset the clickcount to 0

6. In the “body” section of the code do the following:

a. Replace the original heading with one more appropriate for your application.

b. After the heading, add code that will create a button for creating a new trip.

c. Modify the code for boxes at the bottom of the application so that the Number of Stops and Distance is specified.

7. Now document all sections of your code and format the code with appropriate indentations so that it is easy to read.

C. Using Google's Visualization API to Create an Elevation Profile

Run the application 14_10_Cross-section (in the folder A-lines) so that you become familiar with its "look and feel" and what it accomplishes.

Modify the code for 14_03_b_Great_Circle -_input so that it will allow a user to enter a series of points for which an elevation profile is desired. Include two buttons at the top of your application that allow the user to Show the Elevation profile and create a New Profile. This can be accomplished by modifying the code for 14_10_Cross-section as follows:

1. In the "var" declaration section, delete the line associated with the infowindow.

2. Also in 'var" declaration section, delete the lines of code associated with Mt. Whitney, Death Valley etc. since we want to allow the user to enter any set of points.

3. Modify the "initialize" function as follows:
   a. Center the map on 36.339722 N and -117.467778 W.
   b. Retain the section of the function that sets the map options and creates a map object.
   c. After the code for section b, add code that will allow you to specify a polyline. This will involve setting up the options for the path and creating a polyline object.
   d. Add a listener for a click event.
   e. Move the code from the "drawPath" function that allows you to "Create a new chart in the elevation_chart DIV."
   f. Retain the code that allows you to "Create an ElevationService".
4. Add a function called "addlatLng" that is called when a click event occurs. This function will use the "getPath" method to get the path array associated with the polylin and the "push" method to add the new stop to the path. The path is an MCV array so the new stop is automatically displayed.

5. Modify the "drawPath" function as follows:
   a. Delete the code to "Create a new chart…".
   b. Modify the code for "var path…" so that "path" reflects the path that the user has specified. Here you will use the following statement:

```
var path = polyline.getPath().getArray();
```

   c. Retain the rest of the function.

6. Modify the "plotElevation" function as follows:
   a. Delete the section to "Display a polyline of the elevation path, as this has already been done.

7. Add a function called "clearPath" to remove the path drawn on the map and clear the associated column chart when the user clicks on the button for the "New Profile".
   a. First, we need to remove the existing path from the polyline. This is accomplished with the following statements.

```
var path=polyline.getPath();
path.clear();
```

   b. Then we clear the columnchart with the following statements:

```
document.getElementById('elevation_chart').style.display = 'none'
chart.clearChart();
```

8. In the “body” section, add code for the buttons for Show Elevation and New Profile. Also include a message to the user that they need to click on the map to create a profile.

9. Now document all sections of your code and format the code with appropriate indentations so that it is easy to read.

Create a folder for exercise 8 on the webhosting service. Load all files related to parts A, B, and C into this folder. Post the resulting URLs to the following Google Worksheet . Hand in a hard copy version for all three parts of the exercise.