HW3: Geospatial data handling

Total points: 6

In this homework, you are going to work with **spatial data** - you will create (sample, and also, generate) data, visualize it, do queries on it, and visualize the query results.. Hope you have fun with this!

The exercise will give you a taste of working with spatial data, use of a spatial file format and spatial query functions, all of which are quite useful from a real-world (or job interview) perspective.

What you need to do is described below in sufficient, but not too much, detail - you'd need to do a bit of reading up and experimenting, to fill in the gaps. Please post on Piazza, or talk to a TA/CP, or me, if you are unable to proceed at any point!

1. You need to create (generate) latitude,longitude pairs (ie. spatial coordinates) for **12 locations** on the USC campus, centered at Tommy Trojan (TT) :) So you'll get TT's (long,lat), plus 11 more that are located around him.

How would you obtain (long,lat) spatial coordinates at a location? On account of COVID-19, you don't need to venture outside to gather the data (eg using your smartphone's GPS), instead, just read them off Google Maps, https://maps.google.com.

Make a simple table (write/type the data) of locations+names, 12 entries total.

2. Now that you have 12 coordinates and their label strings (ie. text descriptions such as "Tommy Trojan", "SAL", "Chipotle"..), you are going to create a KML file (.kml format, which is XML) out of them using a text editor. Specifically, each location will be a 'placemark' in your .kml file (with a label, and coords). Here is more detail. The .kml file with the 12 placemarks is going to be your file, for doing visualizations. Here is a .kml skeleton to get you started (just download, rename and edit it to put in your coords and labels). NOTE - keep your labels to be 15 characters or less (including spaces). Here is the same .kml skeleton in .txt format, if you'd like to RMB (right mouse button) save it instead and rename the file extension from .txt to .xml. NOTE too that in .kml, you specify (long,lat), instead of the expected (lat,long) [after all, longtitude is what corresponds to 'x', and latitude, to 'y']!

You are going to use Google Earth (GE) to visualize the data in your KML file (see #3 below). FYI, as a quick check, you can also visualize it using this page - simply copy and paste your KML data into the textbox on the left, and click 'Show it on the map' to have it be displayed on a map on the right :)

3. Download GE on your laptop, install it, bring it up. Load your .kml file into it - that should show you your sampled locations, on GE's globe :) Take a snapshot (screengrab) of this, for submitting.

4. Install Postgres+PostGIS on your laptop, and browse the docs for the spatial functions and search for basic tutorials (eg. this is a good one).

Windows users: please install Postgres v.13.2 from here, then install PostGIS after starting Postgres' Stack Builder UI. Mac users can do this instead.

UPDATE: Please install v.13.3 of Postgres, and v.3.1.2 of PostGIS.

After you install Postgres+PostGIS, here is how you can create a DB, create a table, insert data, query it.

5. You will use the spatial db software (PostGIS) to execute the following three spatial queries that you'll write:

• **compute the convex hull** for your 12 points [a convex hull for a set of 2D points is the smallest convex polygon that contains the point set]. Read this. Use the query's result polygon's coords, to create a polygon in your .kml file (edit the .kml file, add relevant XML to specify the KML polygon's coords). Load this into GE, visually verify that all your points are on/inside the convex hull, then take a screenshot. Note that even your data points happen to have a concave perimeter and/or happen to be self-intersecting, the convex hull, by definition, would be a tight, enclosing boundary (hull) that is a simple convex polygon. The convex hull is a very useful object - eg. see this discussion.. Note: be sure to specify your polygon's coords as '...-118,34 -118,34.1...' for example, and not '...-118, 34 -118, 34.1...' [in other words, do not separate long,lat with a space after the comma, ie it can't be long, lat].

• **compute the centroid** of the convex hull you just computed. Add a placemark in your .kml for the centroid (call it 'centroid'), load the .kml into GE, take a snapshot.

• **compute the distance** between your centroid, and TT. Also, in your .kml, draw a line between the centroid and TT, load into GE, take a snapshot.

Note - it *is* OK to hardcode points, in the above queries! Or, you can create and use a table to store your 12 points in it, then write queries against the table.

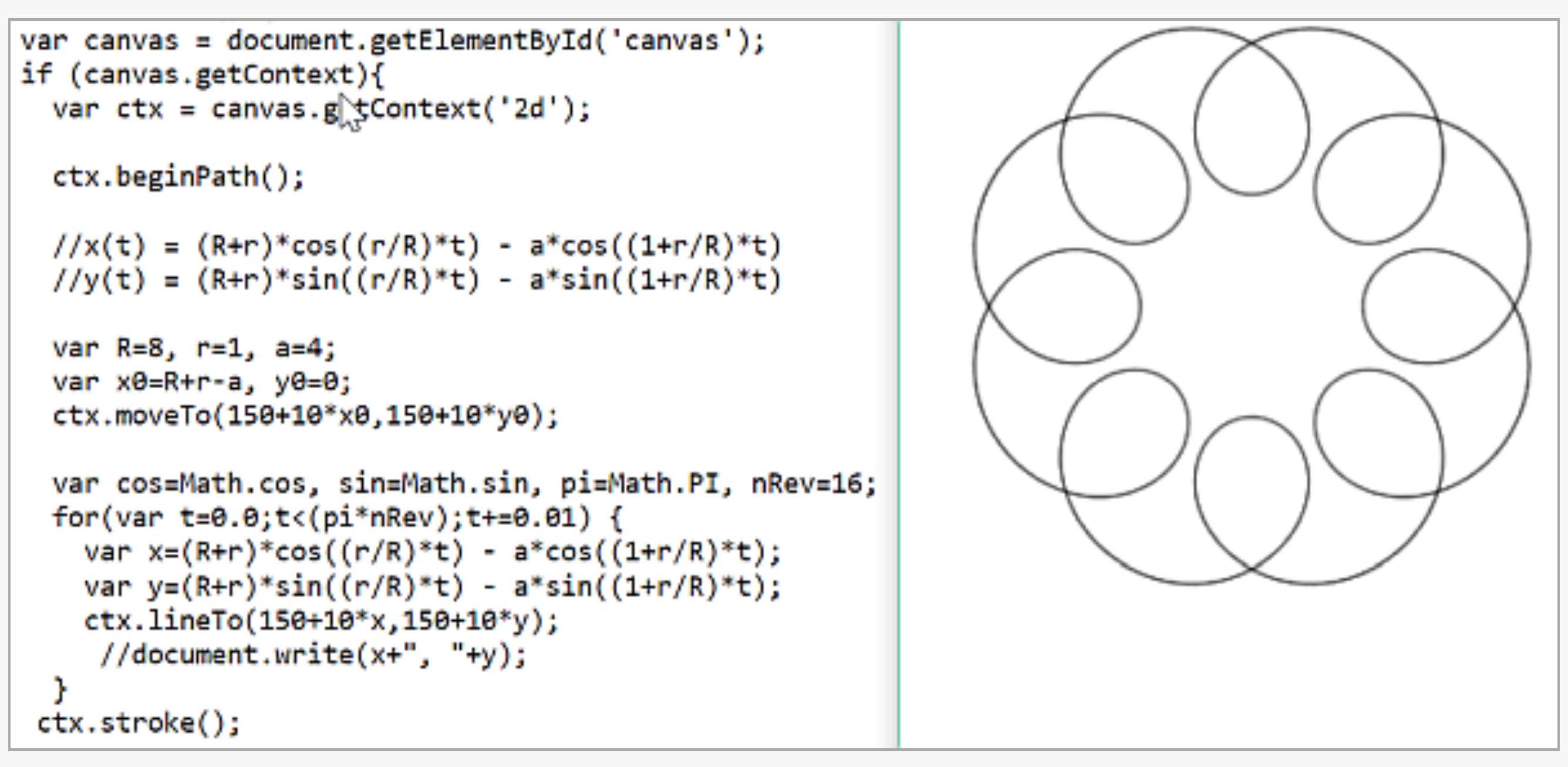
6. Using Tommy Trojan as the center, **compute** (don't/can't use GPS!) a set (sequence) of lat-long (ie. spatial) co-ordinates that lie along a pretty **Spirograph**™ curve :) Make sure the curve is 'small', ie. is confined to our campus (and doesn't expand out to Santa Monica beach, Mexico, etc.!).

Create a new KML file with Spirograph curve points [see below], load it into GE, take a snapshot.

For the Spirograph curve point creation, use the following parametric equations (with R=8, r=1, a=4):

```
x(t) = (R+r)*cos((r/R)*t) - a*cos((1+r/R)*t)
y(t) = (R+r)*sin((r/R)*t) - a*sin((1+r/R)*t)
```

Using the above equations, loop through t from 0.00 to n*Pi (eg. 2*Pi; note that 'n' might need to be more than 2, for the curve to close on itself; and, t is in radians, not degrees), in steps of 0.01. That will give you the sequence of (x,y) points that make up the Spiro curve, which would/should look like the curve in the right side of the screengrab below, when R=8, r=1, a=4 (my JavaScript code for the point generation+plotting loop is on the left):



```
var canvas = document.getElementById('canvas');
if (canvas.getContext){
  var ctx = canvas.getContext('2d');

  ctx.beginPath();

  //x(t) = (R+r)*cos((r/R)*t) - a*cos((1+r/R)*t)
  //y(t) = (R+r)*sin((r/R)*t) - a*sin((1+r/R)*t)

  var R=8, r=1, a=4;
  var x0=R+r-a, y0=0;
  ctx.moveTo(150+10*x0,150+10*y0);

  var cos=Math.cos, sin=Math.sin, pi=Math.PI, nRev=16;
  for(var t=0.0;t<(pi*nRev);t+=0.01) {
    var x=(R+r)*cos((r/R)*t) - a*cos((1+r/R)*t);
    var y=(R+r)*sin((r/R)*t) - a*sin((1+r/R)*t);
    ctx.lineTo(150+10*x,150+10*y);
     //document.write(x+", "+y);
  }
 ctx.stroke();
```

Note - your figure MUST resemble the above, ie. it MUST have 8 loops.

In order to center the Spirograph at a given location [TT or other], you need to ADD each (x,y) curve point to the (lat,long) of the centering location - that will give you valid Spiro-based spatial coords for use in your .kml file. You can use any coding language you want, to generate (and visualize) the curve's coords: JavaScript, C/C++, Java, Python, SQL, MATLAB, Scala, Haskell, Ruby, R.. You can also use Excel, SAS, SPSS, JMP etc., for computing [and plotting, if you want to check the results visually] the Spirograph curve points.

Payoff - what you'll see is the Spirograph curve, superposed on the land imagery - pretty!

PS: Here is MUCH more on Spirograph (hypocycloid and epicycloid) curves if you are curious. Also, for fun, try changing any of R, r, a in the code for the equations above [you don't need to submit the results]!

Here is what you need to **submit** (**as a single .zip file**):

* your .kml file from step 5 above - with the placemarks, convex hull, centroid, and the line joining the centroid with TT (**1 point**)

* a text file (.txt or .sql) with your three queries from step 5 - table creation commands (if you use Postgres and directly specify points in your queries, you won't have table creation commands, in which case you wouldn't need to worry about this part), and the queries themselves (**3 points**)

* screengrabs from steps 3,5 (four screengrabs total) (**1 point**)

* your Spirograph point generation code, the resulting .kml file ("spiro.kml"), and screenshot, from step 6 (**1 point**)

HAVE FUN! From here on out, you know how to create custom overlays (via KML files containing vector symbols constructed from points, lines and polygons) on a map, and perform spatial queries on the underlying data :)

PS: If you liked Postgres, check this out :) https://github.com/alibaba/PolarDB-for-PostgreSQL