HW3: Geospatial data handling

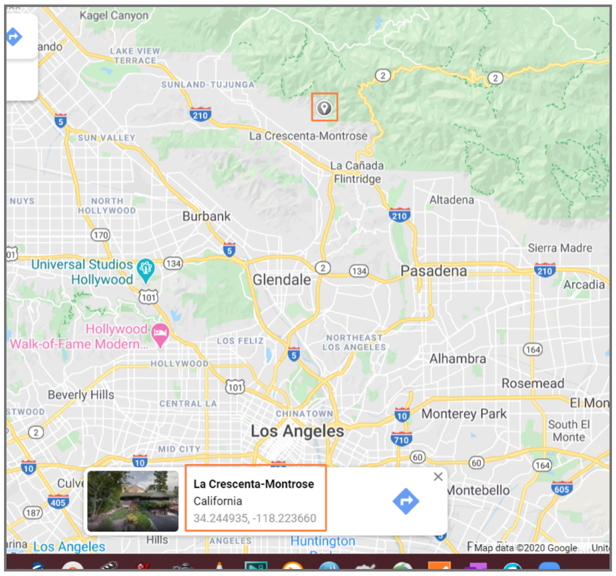
Total points: 5

In this homework, you are going to work with **spatial data** - you will create (generate/sample) some 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 talk to a TA/grader and/or post on Piazza if you are unable to proceed at any point!

1. You need to collect latitude,longitude pairs (ie. spatial coordinates) for **15 locations**, in the USC campus (UPC). Ideally you'd do this by walking around (and using your phone's GPS) to get the (long,lat) values, **but on account of #StayAtHome, you would do this virtually, ie. using Google Maps (https://www.google.com/maps)** - if you long-click on any location (with your left-mouse-button, press for about 1 sec, release), you can see the coordinates for the location, eg:



You need to sample 5 points, in 3 categories of your choice - libraries, cafes, waterworks, etc. Make a note of each location's name/label, category, coordinates.

2. Now that you have 15 coordinates and their labels and categories, 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), under three groups/categories [which will become 'expand/collapse' tree items, in the map software (eg. Earth) that you will use to visualize the data].

Here (https://developers.google.com/kml/documentation/kml_tut#placemarks) is more detail. The .kml file with the 15 placemarks in 3 categories is going to be your starter file, for doing visualizations and queries. Here (data/starter_kml.xml) is a .kml skeleton to get you started (just download, rename and edit it to put in your coords and labels). NOTE - keep your name and category strings to be 15 characters or less (including spaces). Here (data/starter_kml.txt) is the same .kml skeleton in .txt format, if you'd like to RMB save it instead and rename the file extension from .txt to .xml (or you can copy and paste the XML text on the screen into a text file and rename it with a .kml extension). 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']! Also, the 'KML Samples' file from this (https://developers.google.com/kml/documentation/kml_tut) page is a good one to study, to learn how data is structured in the KML format - be sure to load this into Google Earth (#3 below) and look at how the KML structure is interpreted (rendered) by Earth.

You are going to use Google Earth to visualize the data in your KML file (see #3 below). FYI, as a quick check, you can also visualize KML using this (http://display-kml.appspot.com/) 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 Google Earth (https://www.google.com/earth/download/ge/agree.html) on your laptop, install it, bring it up. Load your .kml file into it - that should show you your sampled locations, on Google Earth's globe :) Take a snapshot (screengrab) of this, for submitting.

4. Install Oracle 11g+Oracle Spatial, or Postgres+PostGIS on your laptop, and browse the docs for the spatial functions.

Here (BigSQL/index.html) is my page that walks you through installing a packaged version of Postgres. If you get a 'Problem running the post-install step. Installation may not complete correctly.' error, follow the steps in here (https://webkul.com/blog/postgresql-windows-installation-problem-running-post-install-step-installation-may-not-complete-correctly/) to fix the issue.

Below are Yiqi Zhong's instructions/steps, for Mac users [be sure to turn on PostGIS, using this command: CREATE EXTENSION postgis;]. Note that you'd be working in the 'console', which includes input, output and result panes: https://www.jetbrains.com/help/datagrip/database-console.html (https://www.jetbrains.com/help/datagrip/database-console.html)

Step 1. Download the Postgres.app from https://postgresapp.com/ (https://postgresapp.com/)

Step 2. Follow the instruction on its website configure the port number(super easy, almost need to do nothing)

Step 3. Download DataGrip from https://www.jetbrains.com/datagrip/ (https://www.jetbrains.com/datagrip/) (Nice database IDE and free for students email)

Step 4. Follow the lead on https://www.jetbrains.com/help/datagrip/connecting-to-a-database.html#connect-to-postgresql-database.(Very (https://www.jetbrains.com/help/datagrip/connecting-to-a-database.html#connect-to-postgresql-database.(Very) clear instruction)

4 (alt). You can also use MySQL (https://dev.mysql.com/doc/refman/5.7/en/spatial-extensions.html) if you want, or even sqlite (http://www.bostongis.com/PrinterFriendly.aspx?content_name=spatialite_tut01); if you are familiar with using SQL Server (https://docs.microsoft.com/en-us/sql/relational-databases/spatial/spatial-data-sql-server), that can also help you do the homework. Even QGIS (https://gis.stackexchange.com/questions/38937/how-to-connect-to-postgres-with-qgis) can be used to do the HW.

4 (alt alt). IF YOU ARE FEELING ADVENTUROUS: as an alternative to installing Oracle or Postgres (or MySQL or sqlite or SQL Server...) on your machine, you can use Postgres or Oracle on the AWS cloud platform (ie. without installing anything on your laptop!) - eg. see this (https://aws.amazon.com/free/?) page, and this (https://aws.amazon.com/rds/postgresql/) one. **Be sure to not leave your DB instance running, when you aren't working on the hw!**

4 (alt alt alt). Last but not least, do feel free to use GCP for this! Here are some relevant resources:

* https://cloud.google.com/sql/docs/postgres/quickstart (https://cloud.google.com/sql/docs/postgres/quickstart)

* https://medium.com/google-cloud/postgres-is-incredibly-awesome-c54353b88655 (https://medium.com/google-cloud/postgres-is-incredibly-awesome-c54353b88655)

* https://cloudplatform.googleblog.com/2017/03/Cloud-SQL-for-PostgreSQL-managed-PostgreSQL-for-your-mobile-and-geospatial-applications-in-Google-Cloud.html (https://cloudplatform.googleblog.com/2017/03/Cloud-SQL-for-PostgreSQL-managed-PostgreSQL-for-your-mobile-and-geospatial-applications-in-Google-Cloud.html)

* https://cloudplatform.googleblog.com/2017/08/Cloud-SQL-for-PostgreSQL-updated-with-new-extensions.html (https://cloudplatform.googleblog.com/2017/08/Cloud-SQL-for-PostgreSQL-updated-with-new-extensions.html)

5. You will use the spatial db software to execute the following two spatial queries that you'll write:

· **compute the convex hull** for your 15 points [a convex hull (http://mathworld.wolfram.com/ConvexHull.html) for a set of 2D points is the smallest convex polygon that contains the point set]. If you use Oracle, see this (https://docs.oracle.com/cd/A97630_01/appdev.920/a96630/sdo_aggr.htm) page; if you decide to use Postgres, read this (http://postgis.net/docs/ST_ConvexHull.html) and this (http://stackoverflow.com/questions/10461179/k-nearest-neighbor-query-in-postgis) instead. 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 Google Earth, 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 (https://www.quora.com/What-are-the-real-life-applications-of-convex-hulls) 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 four nearest neighbors** of one of your 15 locations [look up the spatial function of your DB, to do this]. Use the query's results, to create four line segments in your .kml file: line(loc,neighbor1), line(loc,neighbor2), line(loc,neighbor3), line(loc,neighbor4). Verify this looks correct, using Google Earth, take a snapshot.

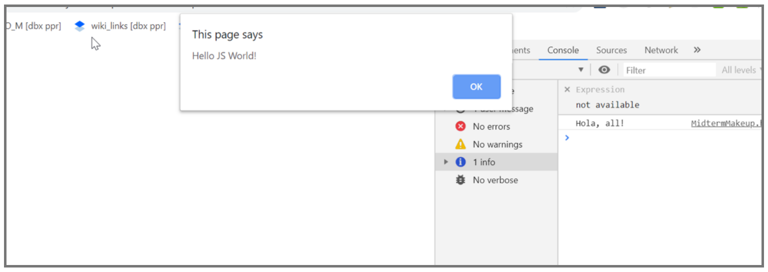
Note - it *is* OK to hardcode points, in the above queries! Or, you can create and use a table to store your 15 points in it, then write queries against the table. Note that you dont need a 'category' column, that was just for the .kml file.

6. Use OpenLayers (a JavaScript API) to visualize your location data. The idea is to store your 15 sampled points, via your web browser, in a browser cache area in your local machine, where the data would persist [even after you close the browser]; then you'd read back the stored values, and visualize them, using the OpenLayers API. To store and load points, you'll use 'HTML5 localStorage', which is a key-value based standard WWW API for storing data locally on your device. (https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage)

The API calls are in JavaScript, which you would run via an html page, like so:

```
<!DOCTYPE html>

<html>

<head>
<title>OL</title>
</head>

<body>
<script>
// your JS code
alert("Hello JS World!");
console.log("Hola, all!");
</script>
</body>

</html>
```

If you create an html file [OL.html] and run it (in Chrome, preferably), you'd see this (in Chrome, to see the console, use the three-dots dropdown menu on the top right: More tools -> Developer tools) :



This page says

Hello JS World!

OK

```
Expression
not available
Hola, all!
```

Now you're ready to store, retrieve, and plot your points!

The code in here (OL/OL.html) is to show you, roughly, how to do it; you'd need to do it 'for real' by modifying it :) EVEN IF you don't know JS or coding, don't worry - it's easy, and fun, to figure it out!

If you like, you can do the location visualization, using CodePen [https://codepen.io/ (https://codepen.io/)] or jsfiddle [https://jsfiddle.net/ (https://jsfiddle.net/)], and just submit a URL of your code - the grader would simply copy and paste your URL into their browser, and be able to see your code and the result. If you do this, submit a README file with your link, instead of submitting OL.html.

The html file linked above, can only be run from a webserver (not locally from your laptop) - you'll likely get a 'CORS' error if you run it locally. There are two ways to proceed. 1. If you have access to a web server, you can upload your completed html file to it, and submit its link. 2. A much simpler option is this: go to http://jsfiddle.net/7fd2g0y9/9/ (http://jsfiddle.net/7fd2g0y9/9/) [which is my fiddle, that contains the same starter code as in the OL.html file linked above], click on the 'Fork' button to create your own editable copy, write your code, run, save [you'd write->run->save many times; each time you save, the link's version # will update (mine is '/9/' above)], then submit JUST THE FINAL LINK, as a README. If you prefer CodePen over jsfiddle, do feel free to use it, and submit your Pen's link. Note - if you get an error related to OpenLayers, it is account of the API version I used, which is http://dev.openlayers.org/releases/OpenLayers-2.13.1/OpenLayers.js (http://dev.openlayers.org/releases/OpenLayers-2.13.1/OpenLayers.js) - switch it to https://openlayers.org/api/OpenLayers.js (https://openlayers.org/api/OpenLayers.js) instead.

7. Using Bovard as the center, **compute** a set (sequence) of lat-long (ie. spatial) co-ordinates that lie along a pretty Spirograph(TM) curve (https://www.google.com/search?q=Spirograph+curve&num=100&source=lnms&tbm=isch) :)

Create a new KML file with Spirograph curve points [see below], convert the KML to an ESRI 'shapefile', visualize the shapefile data using ArcGIS Online, and submit these four items (**as a separate, spiro.zip file**): your point generation code (see below), the resulting .kml file ("spiro.kml"), shapefile (this needs to be a .zip) and a screenshot ("spiro.jpg" or "spiro.png").

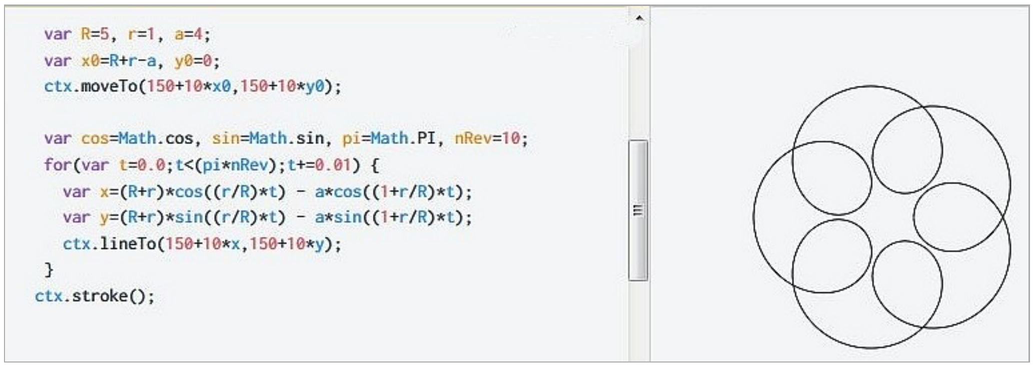
To convert your .kml into a shapefile, use this online converter: https://mygeodata.cloud/converter/kml-to-shp (https://mygeodata.cloud/converter/kml-to-shp) - the result will be a .zip [which is what we call 'shapefile'], which will contain within it, shape data (.shp), a relational table (.dbf), and other optional files (.shx, .prj, .cpg). Here (http://desktop.arcgis.com/en/arcmap/10.3/manage-data/shapefiles/what-is-a-shapefile.htm) is a page on shapefiles, and this (http://desktop.arcgis.com/en/arcmap/10.3/manage-data/shapefiles/shapefile-file-extensions.htm) talks about the various components (.shp, .dbf etc) of a shapefile.

Once you have your shapefile, you can upload it to ArcGIS' online map creator to view your Spirograph curve-shaped points. To do so, log on to ArcGIS [after creating a free 'public' or 'developer' account], at https://www.arcgis.com/ (https://www.arcgis.com/), then use the 'Map' tab - https://www.arcgis.com/home/webmap/viewer.html?useExisting=1 (https://www.arcgis.com/home/webmap/viewer.html?useExisting=1). Do 'Add -> Add Layer from File', and upload your shapefile .zip, you should see your data overlaid on a map. Here (spiro/ArcGIS_spiro.png) is a screenshot of roughly what to expect, and here (https://www.arcgis.com/home/webmap/viewer.html?webmap=2b8d27c576154fbe89f1140dfcab35df) is a live link.

For the Spirograph curve point creation, use the following parametric equations (with R=5, 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=5, r=1, a=4 (my JavaScript code for the point generation+plotting loop is on the left):



```
var R=5, 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=10;
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);
 }
ctx.stroke();
```

Note - your figure MUST resemble the above, ie. it MUST have 5 loops.

In order to center the Spirograph at a given location [Bovard 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 (https://docs.oracle.com/cd/B28359_01/server.111/b28285/sqlqr02.htm), 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 (https://www.google.com/search?q=Spirograph+curve&ie=utf-8&oe=utf-8) 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 in groups, convex hull and nearest-neighbor line segments (**1 point**)

* a text file (.txt or .sql) with your two 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 (1+1 = **2 points**)

* screengrabs from steps 3,5 (**0.5 point**)

* a .html file (with the OpenLayers code) from step 6, or a CodePen/jsfiddle link (**1 point**)

* your 'spiro.zip' file (see step 7, above) (**0.5 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, and its shapefile equivalent) on a map, and perform spatial queries on the underlying data :)
