CSCI 585 - Assignment 3 Rubrics

Geospatial data handling

Format of upload not as .zip (-0.5 points)

Selfies:
No points. But deduct for the following reasons
- Note: The students can take selfies with half-covered faces or with their USC IDs or a part of their hand etc. in the picture. It's to prove that they have gone to the location and taken a selfie. Excuse if even a few are just pictures and not selfies.
- -2 from total if 0-8 selfies are submitted.
- -1 from total if 8-12 selfies are submitted.

KML File (+1 POINT)
- It should have 13 different GPS locations (1 should be the home location and the rest can be anywhere). If there aren't 13 locations, **deduct 0.25 for each missing location**.
- ***(For Non-DEN students) There should 3 libraries, 3 restaurants/coffee places, 3 waterworks (eg. fountains), 3 department buidings and should be placed in four KML 'folders' - 'libraries', 'eateries', 'waterworks' and 'department buildings' and additionally one point for home location. (-0.4 if no folders)***
- Should have a convex hull mapped (The convex hull should contain all 13 points and a few should be on the boundary; in other words, it is always a convex polygon, comprised of a subset of the pointset input to its calculation). Incorrect -**0.25**
- Should have the 4 nearest neighbors from their home location Incorrect -**0.25**
    - Should create 4 line segments (one for each nearest_neighbor) in .kml file: line(home,nearest_neighbor) if Incorrect -**0.25**
- If the convex hull and nearest neighbors are split into two separate files, it's fine.

If the convex hull or the nearest neighbor is missing -**0.5 points each**.

Screen Grabs from Q3 and Q5 (+1 point)
- Google Earth/Map Screenshot of sampled locations +0.33 points
    - if there are fewer than 13 locations - 0.2 points
- Convex Hull +0.33 point
- Nearest Neighbors line +0.34 point

Q4) (No marks for this, as it refers to installing software)
You have to install one of the Postgres GUI app to run the queries, follow the instructions as mentioned in the assignment. https://bytes.usc.edu/cs585/f23-Da-taaa/hw/HW3/index.html

Q5)

(If you are not sure about the correctness of queries, please run them)

NOTE:Table creation commands: if they use Postgres and directly specify points in their queries, no need to have table creation commands, in any other case if table creation commands missing -0.5 point

1. compute the convex hull (+1 points)

- Can call ST_CONVEXHULL (in Postgres) to generate a convex hull for points.
- If only table creation and data insertion commands are present without the rest of the query -0.5 point
- if convex hull generated is not convex (i.e. there are points outside the boundary) -0.25 point

2. Nearest Neighbors (+1 points)

- Can call ST_Distance (in Postgres) to order points based on their distance.
- If only table creation and data insertion commands are present without the rest of the query -0.75 point
- if incorrect -0.5 point

Sample Queries

```
-- Query 1 from part 5
-- Polygon from 13 points
-- Version 1 (Without creating a table)

SELECT ST_AsText(ST_ConvexHull(
ST_GeomFromText('MULTIPOINT(-118.28999161637675 34.019925784781684,
                            -118.28907072246174 34.02005301965949,
                            ...
                            -118.27947853663389 34.02810266419148)')
));
-- Query 1 from part 5
-- Polygon from 13 points
-- Version 2 (Creating a table named hw3)

-- To create a Table

CREATE TABLE HW3 (
    id SERIAL PRIMARY KEY,
    geom geometry(Point, 4326)
);
```

```
-- Inserting Datapoints

INSERT INTO hw3 (geom) VALUES
    (ST_SetSRID(ST_MakePoint(-118.29070783589493, 34.02232558908887), 4326)),
     ...
    (ST_SetSRID(ST_MakePoint(-118.28689932588098, 34.020384007138766), 4326)),
    (ST_SetSRID(ST_MakePoint(-118.27947853663389, 34.02810266419148), 4326));

--Computing Convex Hull

SELECT ST_AsKML(ST_ConvexHull(ST_Collect(geom))) as kml
FROM hw3;
```

```
-- Query 2 from part 5
-- 4 Nearest Neighbours

SELECT id, ST_Distance(geom, ST_SetSRID(ST_MakePoint(-118.27947853663389,
34.02810266419148), 4326)), ST_AsKML(geom) AS kml
FROM hw3
WHERE id != 13
ORDER BY ST_Distance(geom, ST_SetSRID(ST_MakePoint(-118.27947853663389,
34.02810266419148), 4326))
LIMIT 4;
```

Q6) A .html file or CodePen/jsfiddle link (+1 Point)

- If there are less than 13 locations -**0.25 Points**
- If there are no locations -**1 Point**
- If the coordinates mentioned in the html file/code link is not matching with the text/sql file's co ordinates, this hints they have copied the .js code from external sources -**0.5 Points**
- Check if localstorage is not used (points are plotted directly (hard-coded in addMarker) ,without being stored or received) -**0.5 Points**

Sample .js code

```
console.log("Hola, all!");
alert("My DB Homework!");
var d = {"k1":[{"a":34.018725, "b":-118.286611},
{"a":34.021743, "b":-118.282786},
{"a":34.020057, "b":-118.283693},
{"a":34.019211, "b":-118.28555},
```

```
{"a":34.021276, "b":-118.283993},
{"a":34.01876, "b":-118.284367},
{"a":34.020069, "b":-118.289994},
{"a":34.022894, "b":-118.287091},
{"a":34.020159, "b":-118.283309},
{"a":34.01872, "b":-118.282454},
{"a":34.019516, "b":-118.289478},
{"a":34.021963, "b":-118.284722},
{"a":34.022128, "b":-118.286126},]};
```

localStorage.setItem("myData",JSON.stringify(d));

```
// you'd need to modify the above, to store all your points
var dataStored = JSON.parse(localStorage.getItem("myData"));
// verify that we fetched our data correctly
console.log(dataStored);
// we can iterate through our array [of points], like so:
var a = dataStored.k1; // 'a' will contain our array
initMap();
for(var indx in a) {
addMarker(a[indx].a,a[indx].b);
```

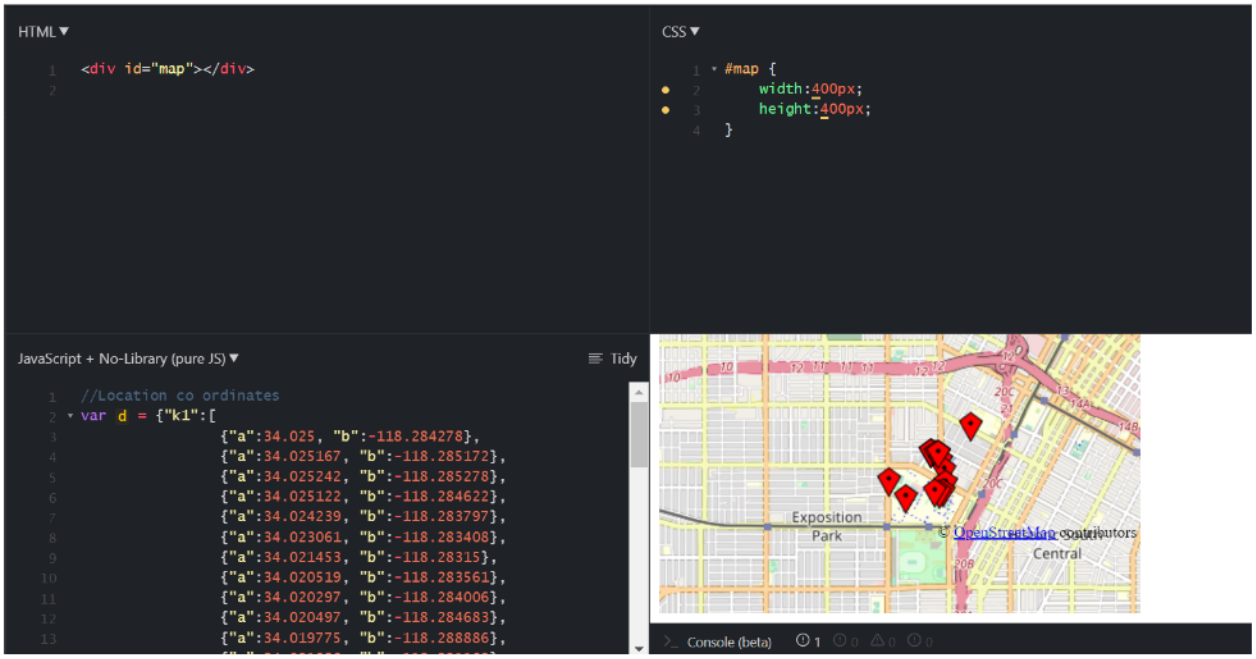
initMap Sample Code

```
function initMap() {
        map = new OpenLayers.Map('map');
        basemap = new OpenLayers.Layer.OSM("Simple OSM Map");
        map.addLayer(basemap);
        markers = new OpenLayers.Layer.Markers("Markers");
        map.addLayer(markers);
        map.setCenter(
           new OpenLayers.LonLat(34.018725,-118.286611).transform(
              new OpenLayers.Projection("EPSG:4326"),
              map.getProjectionObject()
           ), 12
        );
}// initMap()
```

addMarker Sample Code

```
function addMarker(latitude, longitude) {
    var lonLat = new OpenLayers.LonLat(longitude, latitude)
      .transform(
        new OpenLayers.Projection("EPSG:4326"), // transform from WGS 1984
        map.getProjectionObject() // to Spherical Mercator Projection
    );
    var point = new OpenLayers.Marker(lonLat);
    markers.addMarker(point);
    map.setCenter(lonLat, 12); // second arg is zoom level
    //console.log(latitude + ", " + longitude);
}// addMarker()
```

The code submitted may not be exactly the same, however the logic should be similar, that they are able to visualize the coordinates on the map.
The visualization would look something like this (this is the view from a jsfiddle link)



Q7)

The division of the marks is below, refer to the explanation below each division and samples at the end:

1. The screenshot showing the spirograph (0.25 point) (Cap sum of deductions at - 0.25)

- Either the screenshot from ArcGIS Online or Equator Studios is acceptable. **(-0.1)**

- The size of the spirograph shouldn't be **very large** compared to USC, it should be small enough to understand that the spirograph is centered at Tommy Trojan. Any spirograph that is larger than USC and **encompasses the entirety of USC, a lot more surrounding area like Expo Park etc is usually not acceptable**, anything bigger than that, crossing Mid-city, Koreatown is definitely not acceptable. The point is to ensure that effort to scale down was put in. **(-0.1)**
- There **must be four loops / petals in the spirograph**, anything extra / less will not be accepted. **(-0.1)**
- The center point needs to be at Tommy Trojan (check if roughly near it), it **need not be** labeled or plotted. (-0.1)

2. Spirograph point generation code (0.25 point)

- Code can be in any coding language.
- Below is an example python script as a reference.
- If you are doubtful about the code generating coordinates required, please run the code in your local machine / online any compiler like https://www.codechef.com/ide
- Parameters in the code **should be**: R=36, r=9, a=30. They can be scaled down by a constant factor, to make the spirograph smaller.
- **-0.1 for each wrong value to R / r / a, capped at -0.2.**
- If the spirograph in the attached screenshot is wrong, then most probably the point generation code generates the wrong points, you can verify by uploading the KML as described below, if you get a similar wrong result like the screengrab, then the code will also get a **deduction of -0.2**.

3. The resulting .kml file ("spiro.kml") (0.25 point)

- The KML file should contain the spirograph coordinates **(-0.25)**
- If you are doubtful whether the syntax is correct, please verify if it is correct by uploading it to ArcGIS online. Select Add on top left > Add layer from file > Upload the KML file > Select Keep original features. Check if the spirograph is displayed, if not that means the KML file is not right (https://www.arcgis.com/home/webmap/viewer.html?useExisting=1)
- It is acceptable if the spirograph is above the ground by some distance.

4. **Shapefile (this needs to be a .zip) (+0.25 point)**

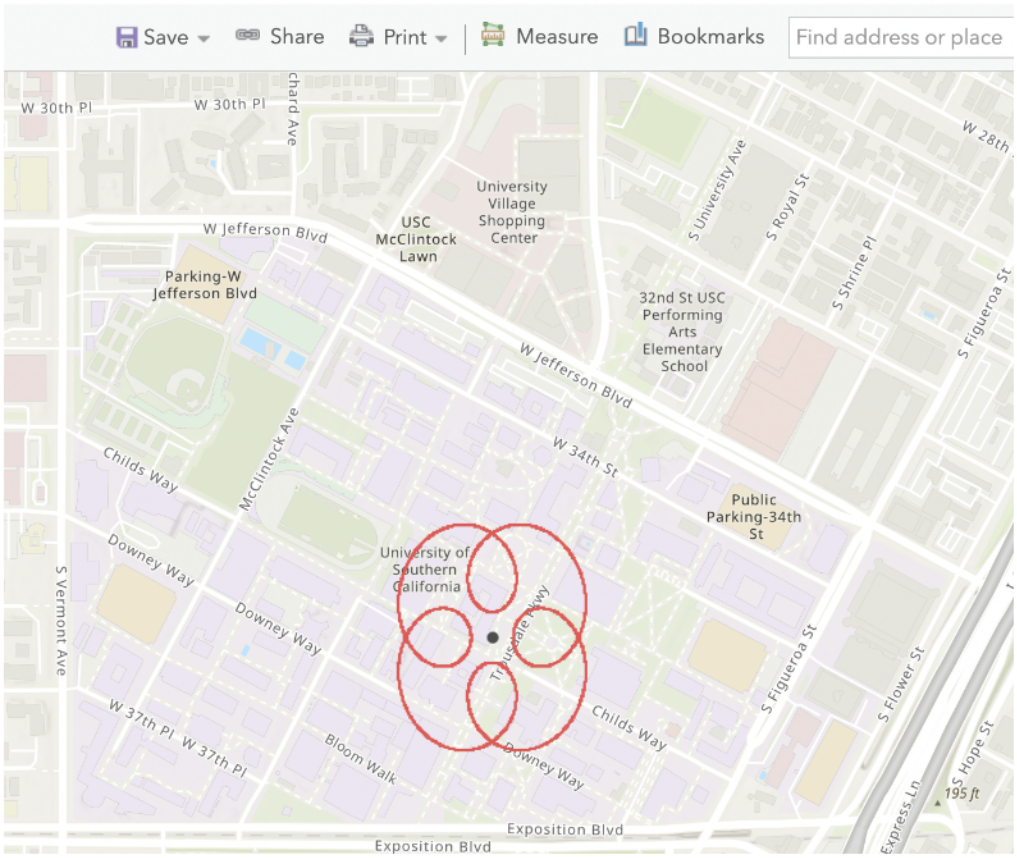
- Ensure that the zip file uploaded contains the .shp file and the .dbf file. **(-0.1)**
- Any other optional files **can be** present in zip generated.
- Though not mandatory, please verify if the zip file once uploaded to ArcGIS online using instructions in (3.) above gives the same spirograph as in the screengrab attached, if any errors occur deduct **-0.2** point.

Sample Python Spirograph point generation code:

```
import math
R, r, a = 36, 9, 30

tommy_x,tommy_y = -118.2854,34.0206
t = 0.0
n = 8
while t < n*math.pi:
   x = (R+r) * math.cos((r/R)*t) - a * math.cos((1+r/R)*t)
   y = (R+r) * math.sin((r/R)*t) - a * math.sin((1+r/R)*t)
   # Using 50,000 as scaling factor
   new_x = x/50000 + tommy_x
   new_y = y/50000 + tommy_y
   print(f"{new_x},{new_y},0")
   t = t + 0.01
```

Sample Spirograph screen grab from ArcGIS online (Notice how 4 loops / petals look like):



Sample KML file with some coordinates:

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2">
<Document>
   <visibility>1</visibility>
   <open>1</open>
   <Style id="WhiteLine">
       <LineStyle>
           <color>7f0000ff</color>
           <width>4</width>
       </LineStyle>
   </Style>

   <Placemark>
       <name>Tommy Trojan</name>
       <Point>
           <coordinates>
               -118.285383925603226,34.02050625861774
           </coordinates>
       </Point>
       <Style>
           <IconStyle>
               <Icon>
                <href>http://maps.google.com/mapfiles/kml/paddle/blu-circle.png</href>
               </Icon>
           </IconStyle>
           <LabelStyle>
               <color>ff00ffff</color>
               <Scale>10</Scale>
           </LabelStyle>
       </Style>
   </Placemark>

   <Placemark>
       <name>Spirograph</name>
       <visibility>1</visibility>
       <styleUrl>#WhiteLine</styleUrl>
       <LineString>
           <coordinates>
               -118.285484,34.020506
```

```
              ...
              -118.285484,34.020513
          </coordinates>
      </LineString>
   </Placemark>
</Document>
</kml>
```

****