CMPS 161 Prog1 Writeup

User Documentation:

The user will receive a directory containing an html file called “driver.html” and a subdirectory called “/prog” which contains the necessary JavaScript Files needed to run the wind map.

(Image shows what the given directory should look like)

If the user does not already have this directory, he or she can retrieve it at:

https://people.ucsc.edu/~ydchoe/CMPS%20161/prog1/WindMap/

To start the program, the user must open the html file on a web browser (i.e. Chrome, Firefox, Safari, Edge, etc). Please note that internet connection is required in order to use the D3.js library and Google Map API. Once opened, a static Google Map and heat map will be rendered on screen. Buttons below the map can be used to toggle different features (heat map, station arrows, arrow plot, streamlines) on and off.

(Image 1: Heatmap, Image 2: Station Arrows, Image 3: Arrow Plot, Image 4: Streamlines)

Technical Write-up:

Introduction to the Problem:

The assignment requested for a “wind map based on wind sensor readings from [meteorological stations] within a geographical region of interest”. A region that interested me was the Bay Area. My data is made up of 14 weather stations that were listed on the Bay Area Air Quality Management District (BAAQMD) website. The timeframe I had chosen was 3AM February 02, 2019. I picked this particular timeframe because I was emailed a high wind advisory by Public Affairs at UCSC. The BAAQMD provided station names, station geo locations, wind speeds, and wind directions.
Features:

The user is given a 5 color heatmap that represents wind speed (bluish means low, reddish means high). Along with the heatmap, the user is given station arrows, an arrow plot, and streamlines. Size and color of the arrows correlate to its wind speed, while the direction of the arrow head points to the direction of the wind. The streamlines help show the general path of the wind.

What Was Used to Represent Data:

I decided to use HTML, JavaScript, D3.js (Data Driven Documents), and a Google Maps API to represent my data. JavaScript will provide most of the calculations and data structure needed for the program, while D3.js will provide the user with a visual representation of the data (i.e. Heatmap, Station Arrows, Arrow Plot, Streamlines). The Google Maps API also provides the user with visual representation of the geographical region of interest, but its main function is to help convert between HTML pixel coordinates and GPS coordinates in order to accurately interpolate wind data.

Spatial Extent and Resolution:

The grid data is 100x100 in resolution and it is visually represented on a SVG element of 700x700 pixels. Underneath the SVG element lies a Google Map that has a resolution of 700x700 pixels. The map is centered at latitude 37.93997352043745 and longitude -122.21933390178572 with a map zoom factor of 9. This gives a map of the Bay Area. For reference the boundaries for the map are: North-East Corner (38.6941920402164, -121.25803019084822), South-West Corner (37.177934816171096, -121.25803019084822).

Generating Grid Data:

```javascript
class StationInfo
{
    constructor(name, latitude, longitude, speed, direction)
    {
        this.stationName = name; // Name of the wind station
        this.coordinates = [parseFloat(latitude), parseFloat(longitude)]; // Coordinates of the wind station (in Decimal)
        this.windSpeed = parseFloat(speed); // Speed of the wind (in MPH)
        this.windDirection = parseFloat(direction); // Direction of the wind (in degrees)
        this.xPos = null;
        this.yPos = null;
    }
}

// 2017 Air Monitoring Network Plan (July 1, 2018) Document
stationArray.push(
    new StationInfo(
        "Bethel Island",
        38.004311,
        -121.641918,
        23,
        129));
```

(Image above shows what initializing a StationInfo Object looks like)

I manually take each station’s data from BAAQMD and assign them to a JS class object called StationInfo. Each class contains the station’s name, geo location (in decimal degrees), wind speed, wind direction, and pixel coordinates relative to the SVG element. Then each StationInfo object is added to an array called stationArray. If the user needs to add more stations, this is the place to do it.
In `main.js` I initialize a Google Map element and assign it to a variable called `map`. Here the user can pick a center location and a zoom factor. This would be the place to change the geographical location of the map. Google Maps API allows users to create an overlay to add custom graphic elements. The overlay also allows users to convert between pixel coordinates and GPS coordinates. Using GPS coordinates instead of pixel coordinates allows for a more accurate interpolation of data.

```javascript
var gridData = new GridData(100, 700);
var projection = overlay.getProjection();
var translatedRes = (gridData.resolution)/2;

for(let i = 0; i < gridData.gridPoints.length; i++)
{
    for(let j = 0; j < gridData.gridPoints[i].length; j++)
    {
        /*
        * I am translating pixel coordinates to GPS coordinates and storing them into each GridPoint.
        * I then use Shepard's algorithm to interpolate each grid point's lat and lng using the
        * 14 weather stations we have gathered
        * /
        var mapCoord = projection.fromDivPixelToLatLng(new google.maps.Point(
        gridData.gridPoints[i][j].mapCoord[0] = mapCoord.x();
        gridData.gridPoints[i][j].mapCoord[1] = mapCoord.y();
        shepardAlgorithm(stationArray, gridData.gridPoints[i][j]);
    }
}
```

Next I initialize the data grid and convert each pixel coordinate to its relative GPS coordinates on the Google Map. Using Shepard’s Interpolation and the data from the 14 weather stations, I interpolate the wind speed and direction for each grid point. The formula is as follows:

\[
   w(x) = \sum_{i=1}^{N} w_i(x) u_i, \quad \text{if } d(x,x_i) \neq 0 \text{ for all } i,
   
   = \sum_{i=1}^{N} w_i(x) u_i, \quad \text{if } d(x,x_i) = 0 \text{ for some } i,
\]

where

\[
   w_i(x) = \frac{1}{d(x,x_i)^p}
\]

(\text{Image above is from Wikipedia - “Inverse Distance Weighting”})

At this point, I have all the information I need in order to generate a heatmap, station arrows, and a arrow plot.
Generating Streamlines:

In order to generate streamlines I need the values for every 10th grid point. Using Euler’s integration, we can find the next and previous streamline point for a nth grid point. The formula is as follows:

\[ P_{k+1} = P_k \pm hV \]

To keep the streamlines smooth the \( h \) constant must be kept small. For my program I set the \( h \) value to 0.001. In order to find the \( V \) (vector) for \( P_{k+1} \), I simply use Shepard’s Interpolation for that point. The values are stored into an array that is part of a StreamLine object.

Displaying Data Using D3

D3 allows a user to easily manipulate html elements, such as SVG, based on data. For the heatmap, I drew a grid of squares and filled in the appropriate color using the wind speed data I interpolated earlier. Areas that are more blue indicate low wind speeds while areas that are more red indicate high wind speeds. Each wind station arrow is a line extending from its origin with its angle corresponding to a given wind direction. Each line is terminated with a triangle shape that serves as an arrow head. The length and color of each arrow corresponds with the wind speed data. A similar process is done for the arrow plot using data that was interpolated earlier. The streamline is just an array of points connected by a continuous line.

Observing the Data:

Because I am only using 14 weather stations, it is important to note that the wind map is probably only accurate where the stations are bunched up together, mostly around the center. This is apparent when changing the map type from a road map to a terrain map. With the streamlines on, we can see that the wind path is trying to avoid mountainous areas near Concord and Walnut Creek. However the streamlines makes no attempt to avoid mountainous areas around Santa Rosa. As there aren’t any wind stations near the area being used in my program, the wind map has no way knowing that a mountain is blocking wind.
Limitations and Shortcomings:

I did not generate the streamlines in the most efficient manner. The more weather stations the user adds the longer it will take to generate the streamlines. This is because the program is re-interpolating with Shepard’s algorithm for every vector during Euler’s integration. Using bilinear interpolation with the existing wind data that the program created earlier would have kept the streamline runtime to \( O(N) \) instead of \( O(N^N) \) with the input being the number of weather stations. Although using Shepard’s algorithm will result in a more accurate streamline, it will take much more time.