12/25/2014

I calculated using basic trig and ratios that the distance between two lines of longitude given latitude can be represented by the equation: distance = diameter\_of\_earth \* cos(lat) / 360. The diameter of the earth is 40075 km at 0 deg lat, cos 0 = 1. The cos is proportionate to the radius and therefore the circumference of the earth.

12/26/2014

Integrated above formula into algorithm, and also calculated approximate elevations by finding elevations between intervals of each point. The results yielded: Algorithm: 160986, Google: 190305, which is about 0.4% difference. Considering the total is this accurate, I can assume that the general formula is correct. I do not understand what factors are influencing the google total.

This means that my algorithm is accurate enough to provide rough data. Next task will be to find data on OpenStreetMap for road speeds.

12/28/2014

Continued to experiment with Overpass API to extract OpenStreetMaps. Will need to use URLs to fill forms.

Formula for drag: *FD = cD 1/2 ρ v2 A   (2)*

*Engineer’s toolbox*

Query for OpenStreetMap:

var request = escape('(node["highway"="traffic\_signal"](36.4,-80,36.41,-79.99);way(36.4,-80,36.41,-79.99););(.\_;>;);out;');

var info = new XMLHttpRequest();

info.open(“GET”, “http://overpass-api.de/api/interpreter?data=”  
+info, true);

info.send();

var xml = info.responseXML;

//Sources from Stack Overflow and OpenStreetMap wiki; map data courtesy of OpenStreetMap.

I generated this code by searching the internet for examples on how pull xml data from OpenStreetMap onto javascript. I came across the Overpass API, which allowed me to do it. I initially did not know that the API was capable of doing what I wanted. After a couple hours of investigation, I discovered that it did exactly what I wanted; I just needed to find a way to get the information directly from the web into my script. Meanwhile, I was looking for ways to translate XML into javascript, and on Stack Overflow (where I go with my software questions as a default). The device which does this is XMLHttpRequest. I learned the usage and did some test with it. I discovered the url the submit button on the form for Overpass queries [here](http://overpass-api.de/query_form.html), and inside the top form I discovered where the data went after the form was submitted (relative url of action attribute of first form). I found the data name inside the form, and I assumed that it would be transferred as the same name to the server. I experimented with sending QL across, and it worked. It was immediately integrated into the code.

12/31/2014

Engine fuel efficiency is known as thermal efficiency, or the ratio of chemical energy in gasoline which is converted to mechanical energy in the engine. For most cars this is about 25%, meaning that 25% of the energy in the gasoline is converted into kinetic energy. <http://en.wikipedia.org/wiki/Thermal_efficiency>

Need to Know:

For each model:

* Fuel Consumption @ RPM
* Torque @ RPM
* Drag Coefficient
* Cross-Sectional Surface Area
* Number of wheels, width of wheels
* Mass

Constants:

* Density of Air
* Coefficients for Rolling Friction
* Energy Density of Gasoline
* Gravity (9.81 m/ss)

Equations:

* Drag
* Rolling Friction

The energy density of gasoline is 44.4 MJ/kg (Wikipedia). Density of air is 1.2754 kg/m3 (at sea level, 0degC). Force of Rolling Resistance is c\*m\*g, where c is coefficient, m is mass and g is gravity (9.81). Force of drag is ½\*c\*rho\*v2\*A, where c is coefficient (car “like Prius” is 0.26), rho is density of air (1.2754), v is velocity, and A is cross-sectional area.

Brake Specific Fuel Consumption (BSFC) is the ratio of work done (kWh) to the amount of fuel used (g).

By 1/15/15 I integrated Overpass data collection and geocoding. The geocoding is to match up roads in the overpass data to Google maps data. They are done by making a list of requests, and then having a single setInteral loop check if all the data has come in yet. When all the data is in, it exits to a joining function, which executes the compute() function if all the queries have come in.

These functions were initially unsuccessful because I did not take into account that the functions after several tenths of a second needed to access a value of a common iterator in the for loop, and therefore failed. An example: for(var i=0; i<5; i++){setTimeout(function(){console.log(i)},100\*i)}; The result was: (5) 5

Another related issue was that locally created setIntervals in loops were deleted, so they could not be terminated. for(var i=0; i<5; i++){var loop = setInterval(function(){if(<condition>){clearInterval(loop)};console.log(text)},100)}; The result of this was that the loops never exited. This was solved by having an array of requests which was built upon by a for loop, and then having a single setInterval loop check for completion, instead of doing everything in one loop on an individual basis.

There is an issue with street name syntax which makes matching up in some cases impossible. Example: North Peace Haven Road vs. Peace Haven Road North, Pine Valley Road North vs. Pine Valley Road, etc. Since roads with the same names with or without extra (“North”) have for the most part the same attributes, errors were thrown. This was solved by making three versions of names: (example) real: North Peace Haven Road, reversed: Peace Haven Road North, plain: Peace Haven Road. These three options covered all possible syntaxes in OpenStreetMap, and so far no errors have been caused by this.

I started gathering information on how OpenStreetMap data correlates to speed limit, but I have very little information as far as that goes. There are very limited speed limit datum, and it tends to be for one or two roads out of a thousand. I wrote a short script to collect data from thousands of road samples in var over and sort it. Results were recorded in an Excel spread sheet.

In assigning stop lights to points, I discovered that the Google and OpenStreetMaps data was off by about 20/10000s of a degree, which is significant. I applied a filter to allow all points within that radius of a stop light to have a stop light on them. This is imperfect, but after testing it appears to be reasonably good. Fine-tuning will have to be done.

The solution I used to find the average road speed was this: sort street data into categories of counties with sub-categories of road types. The median of each sub-category is used as the default value. If a speed is explicitly defined, that is used, but if not, the default is used. Mean was deemed unreliable because upon testing, I discovered that residential category had an average of 56 mph, which is unheard of.

Upon further inspection, I realized that map data contained in over was incorrect, thus throwing off the data derived from it. It likely got scrambled in the parse function, perhaps with a mistake I made in variable scope. I emailed Sayari Ghosh about it.

I analyzed the raw XML for a specific example and compared it to the over variable data, and my conclusion was that the elements did not exactly overlap when data was overwritten, by the design of my algorithm. This was solved with two concepts. The first was that I made each road name in over an array which was a collection the original types of data. I also split over up into different sectors based on the Overpass API query bounds, in order to localize the speeds for sections of road somewhat. In calculating the speed limits for a road, I changed the method from the median of each occurrence of a road name to the mean, considering the number of nodes in a segment of road.

In poking around a bit in the data, I noticed that the problem aforementioned was also caused by the Overpass API (perhaps) assigning close roads to roads that have a “ref” tag (like US 421). This was fixed by assigning by default the ref name to the name of a road if the ref tag existed. After fixing this, a new problem was seen, with road names coming out like “I 40 Business;US 421”, which did not exist in any of the geocoding data. Two road names were added to over, one for each of the strings split by the semicolon.

There is an issue with this method. The data in the over variable for roads like US 421 are invalid. I think this has to do with the above technique. The data was in general weird. An example from the console: (output\_01.png). However, the speeds variable had the correct information. The problem was that the conditional statement in the for loop assigning data to over was incorrect, and over was being overwritten constantly.

There was the finding of false street names in the geocode function. I added a local function which parses the xml to find the precise name by searching instead of guessing (like the old one). The leadup to this can be found in getNameTest.js. I originally only had it set to find the “street\_address” attribute, but expanded it to “route” to include highways. I also added a few conditional statements in the local function combo to format the road names to that of Overpass API (i.e. Business 40 to I 40 Business).

The issues outlined in output\_02.png and output\_03.png are caused by the following phenomenon: Overpass Data had either ref or name. I conducted a test (in output\_04.png) that demonstrates the problem: unused variables have the value hang around even when they are redefined. This was fixed by resetting all the variables in the for loop at the end to undefined. I now realize how sparse the data is. I will have to add more for the general region, or expand the area of the Overpass query to get a better database for speeds.

I redid the function for compiling averages to work, and also to reflect the number of nodes in a road. The final value is in m/s. I attempted to add it to each point in pts, but my initial attempt failed. It is commented off in the source code. I determined that there were three cases for assigning speed data: there is an explicitly defined speed limit, which is the first preference; there are other defined speed limits for the same road type; and finally there is a default value, which for the time being is 35 mph.

I added code which finds the amount of energy in Joules needed to move a vehicle along a path in the intersections object. I accounted for reversed roads, and added an exception if a point (rarely) does not have road data. As far as I know, this problem pertained to one point. I spent about an hour on 2/4/15 doing this. I still need to account for stoplights and signs, and acceleration.