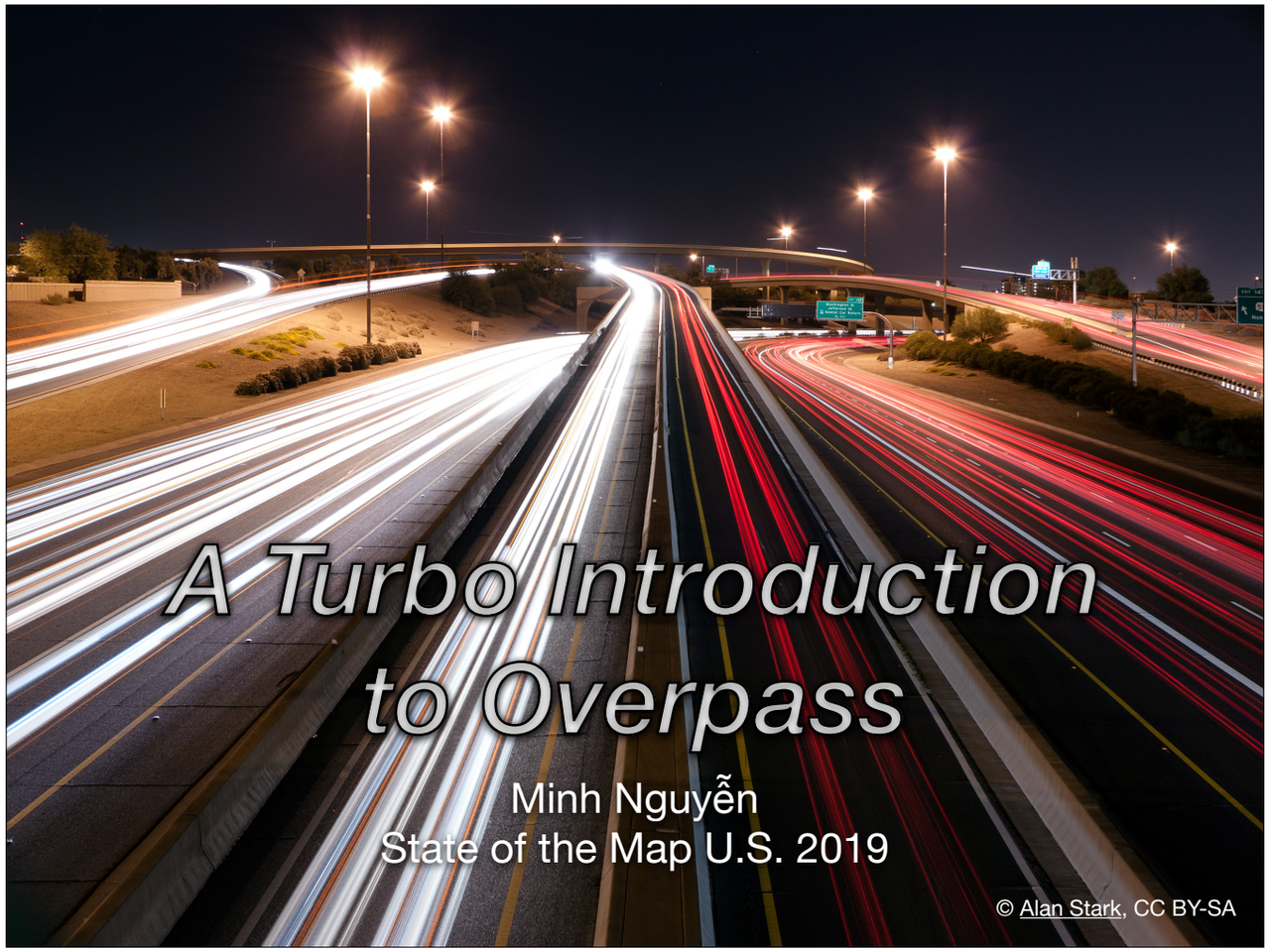
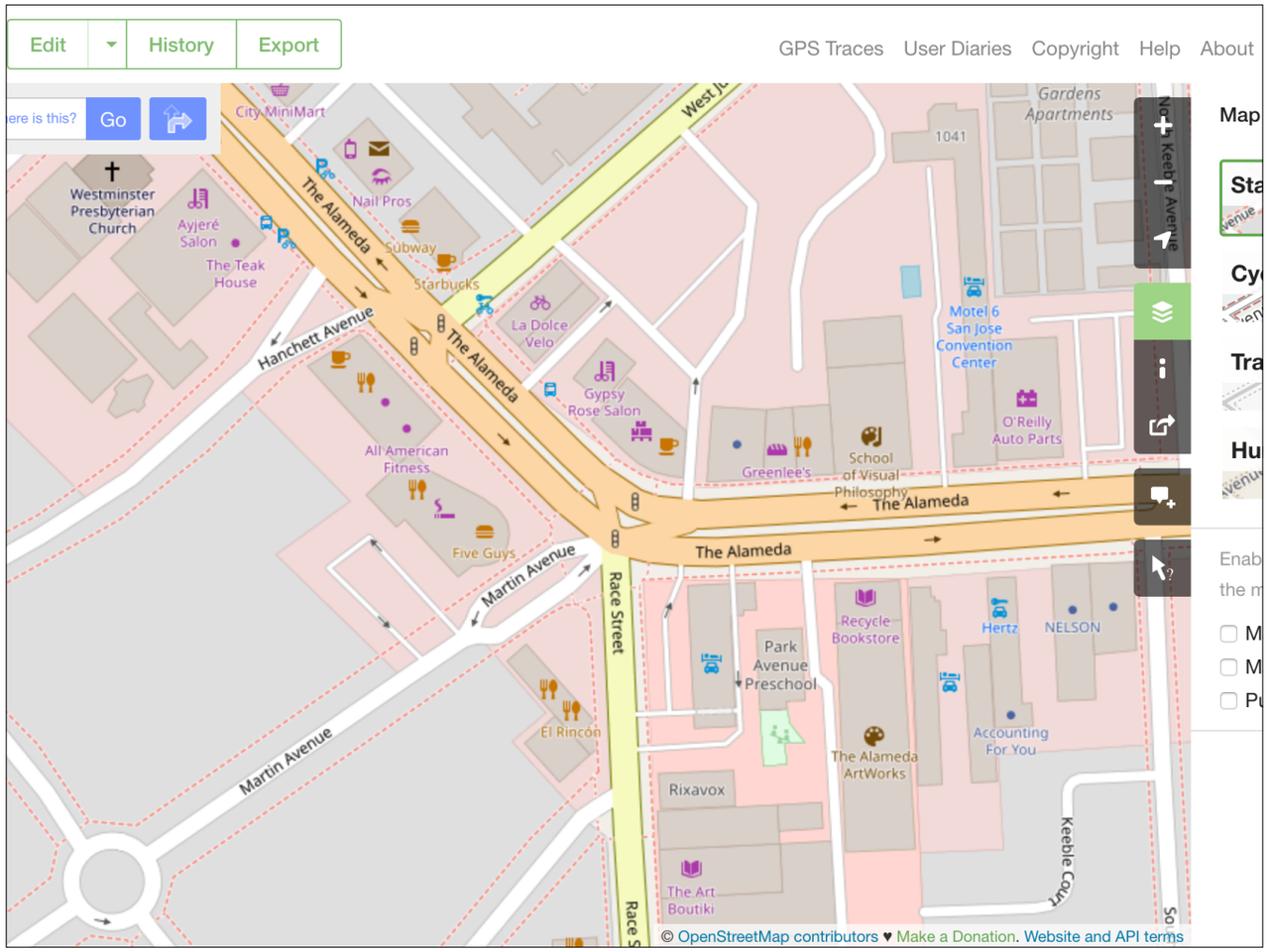
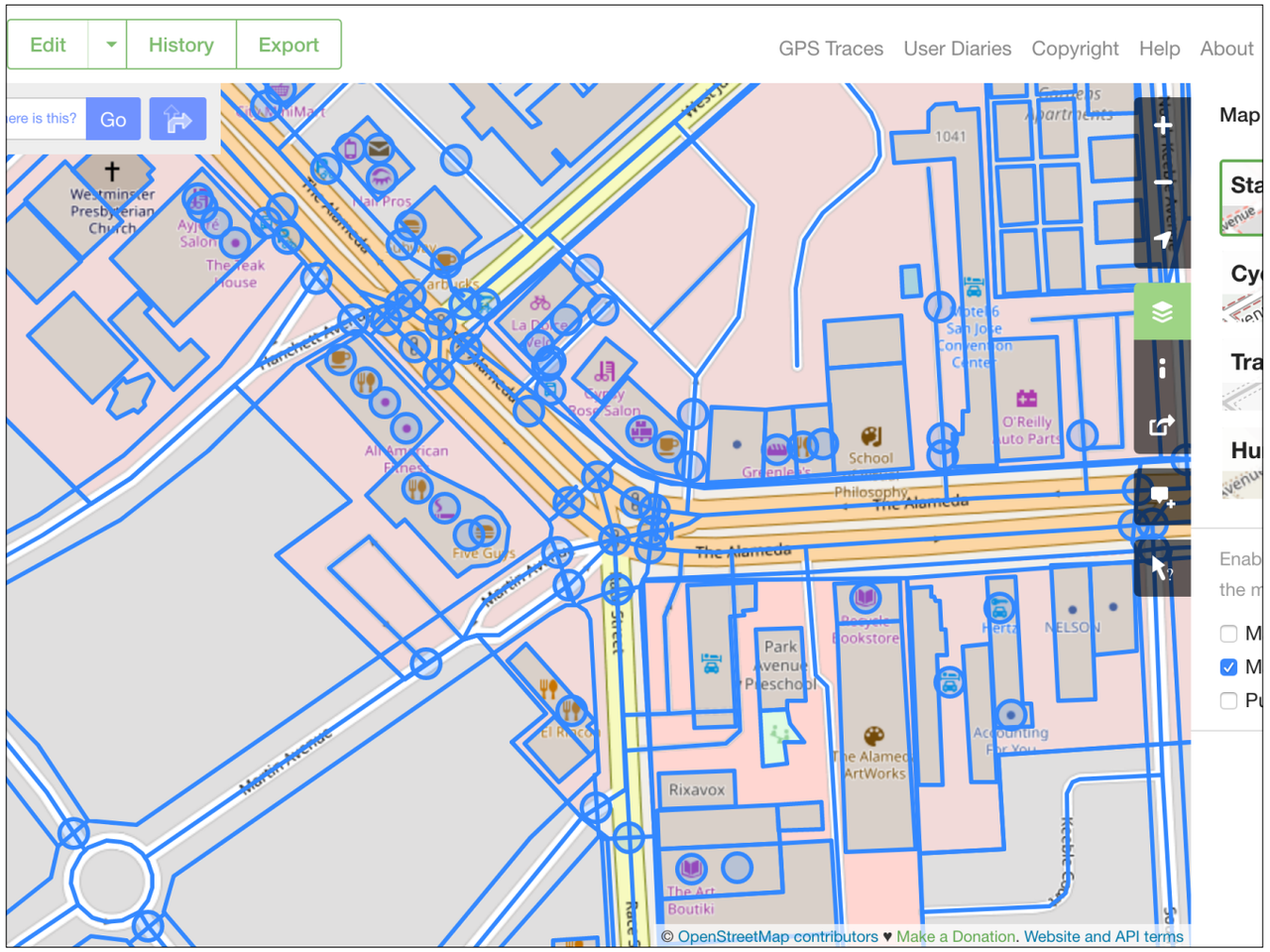


Hi I'm Minh Nguyen. Hope you've all enjoyed the conference so far. You're going to see lots of photos of roads because I'm a roadgeek, but this talk isn't really about roads, it's about overpasses.

I've been mapping for a number of years, and Overpass is my favorite tool for understanding OSM. I'm no expert at Overpass, just a practitioner who learned on the fly through ample trial and error and help from others. Hopefully by the end of this talk, you'll learn just enough that you'll always keep an Overpass tab or two open in your browser.



So what is Overpass? We often say that OSM is not just a map but also a database.



There’s a lot of data, more than meets the eye.

Connecticut	[.osm.pbf]	(27.4 MB)	[.shp.zip]	[.osm.bz2]
Delaware	[.osm.pbf]	(10.5 MB)	[.shp.zip]	[.osm.bz2]
District of Columbia	[.osm.pbf]	(15.8 MB)	[.shp.zip]	[.osm.bz2]
Florida	[.osm.pbf]	(220 MB)	[.shp.zip]	[.osm.bz2]
Georgia (US State)	[.osm.pbf]	(175 MB)	[.shp.zip]	[.osm.bz2]
Hawaii	[.osm.pbf]	(11.9 MB)	[.shp.zip]	[.osm.bz2]
Idaho	[.osm.pbf]	(62 MB)	[.shp.zip]	[.osm.bz2]
Illinois	[.osm.pbf]	(166 MB)	[.shp.zip]	[.osm.bz2]
Indiana	[.osm.pbf]	(69 MB)	[.shp.zip]	[.osm.bz2]
Iowa	[.osm.pbf]	(84 MB)	[.shp.zip]	[.osm.bz2]
Kansas	[.osm.pbf]	(58 MB)	[.shp.zip]	[.osm.bz2]
Kentucky	[.osm.pbf]	(104 MB)	[.shp.zip]	[.osm.bz2]
Louisiana	[.osm.pbf]	(99 MB)	[.shp.zip]	[.osm.bz2]
Maine	[.osm.pbf]	(43.8 MB)	[.shp.zip]	[.osm.bz2]
Maryland	[.osm.pbf]	(132 MB)	[.shp.zip]	[.osm.bz2]
Massachusetts	[.osm.pbf]	(235 MB)	[.shp.zip]	[.osm.bz2]
Michigan	[.osm.pbf]	(128 MB)	[.shp.zip]	[.osm.bz2]
Minnesota	[.osm.pbf]	(169 MB)	[.shp.zip]	[.osm.bz2]
Mississippi	[.osm.pbf]	(68 MB)	[.shp.zip]	[.osm.bz2]

So much of it that there are dedicated services to cutting the OSM planet down into smaller “extracts”, so that people can load just their area of interest into a GIS tool. But what if you don’t need all the data in a given area, just the data that meets some criteria?

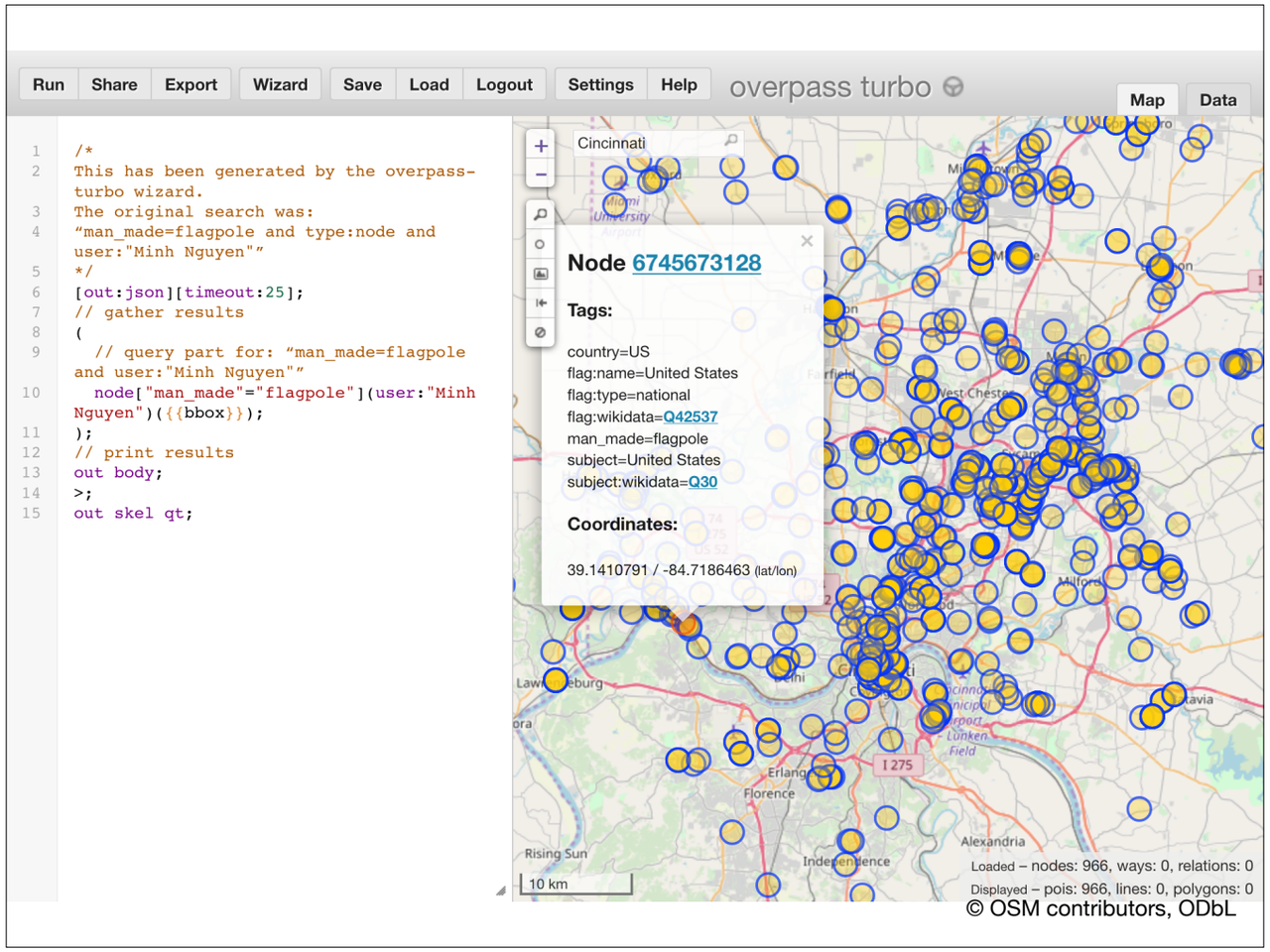
Overpass API Query Form

```
[out:json][timeout:25];
(
  way[~"turn"~".*"](user:"Minh Nguyen");
);
out body;
>;
out skel qt;
```

Query

Overpass API Convert Form

Every database has a way to query it. OSM has the Overpass API. It’s like a search engine for metadata. It’s also quite intimidating – this is what it looks like, with an example of the cryptic query language, OverpassQL.

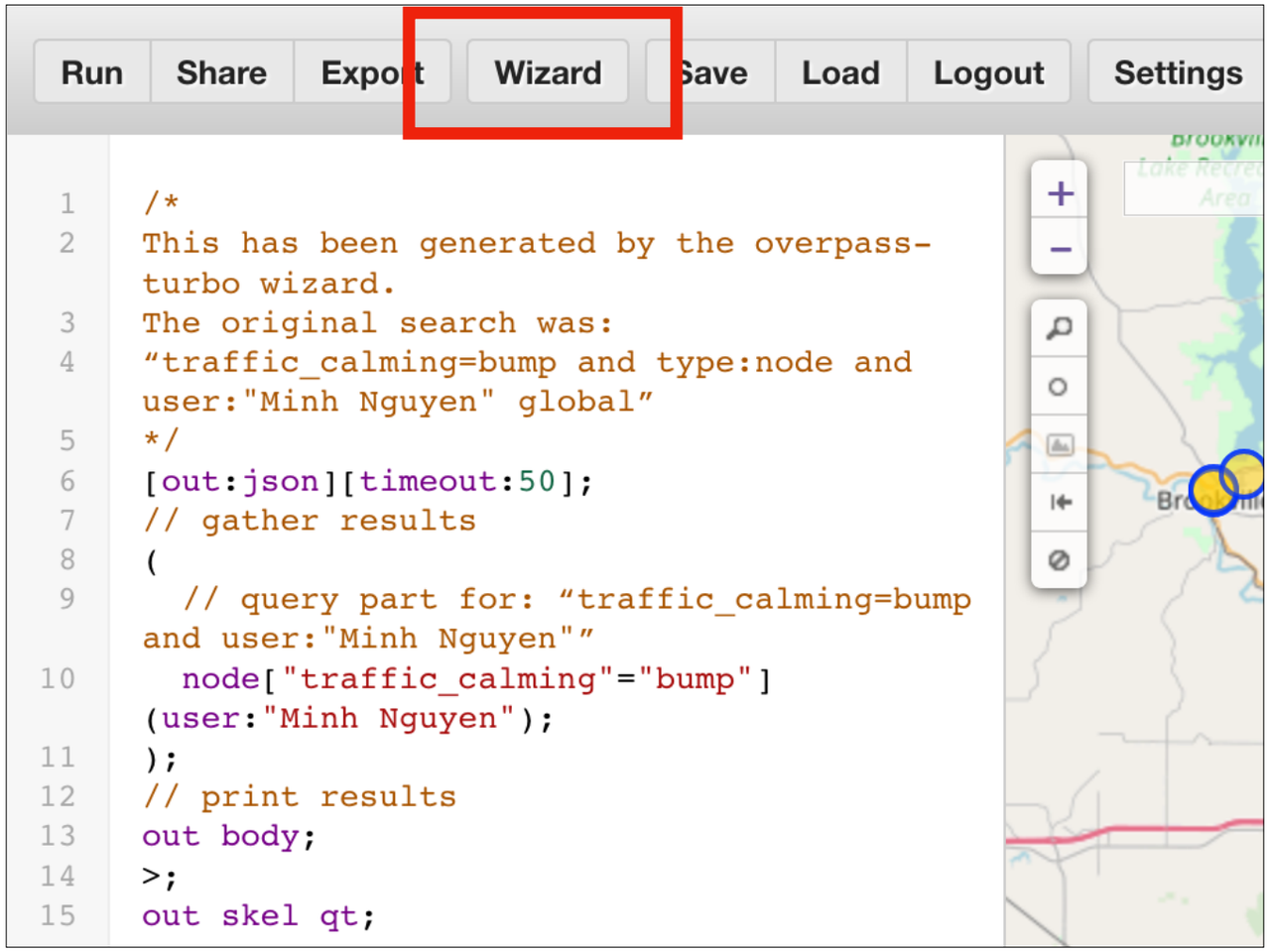


Overpass Turbo makes it a lot easier to create an Overpass query and make use of the results. By default, it shows results on an interactive map, and clicking a feature inspects all its tags.

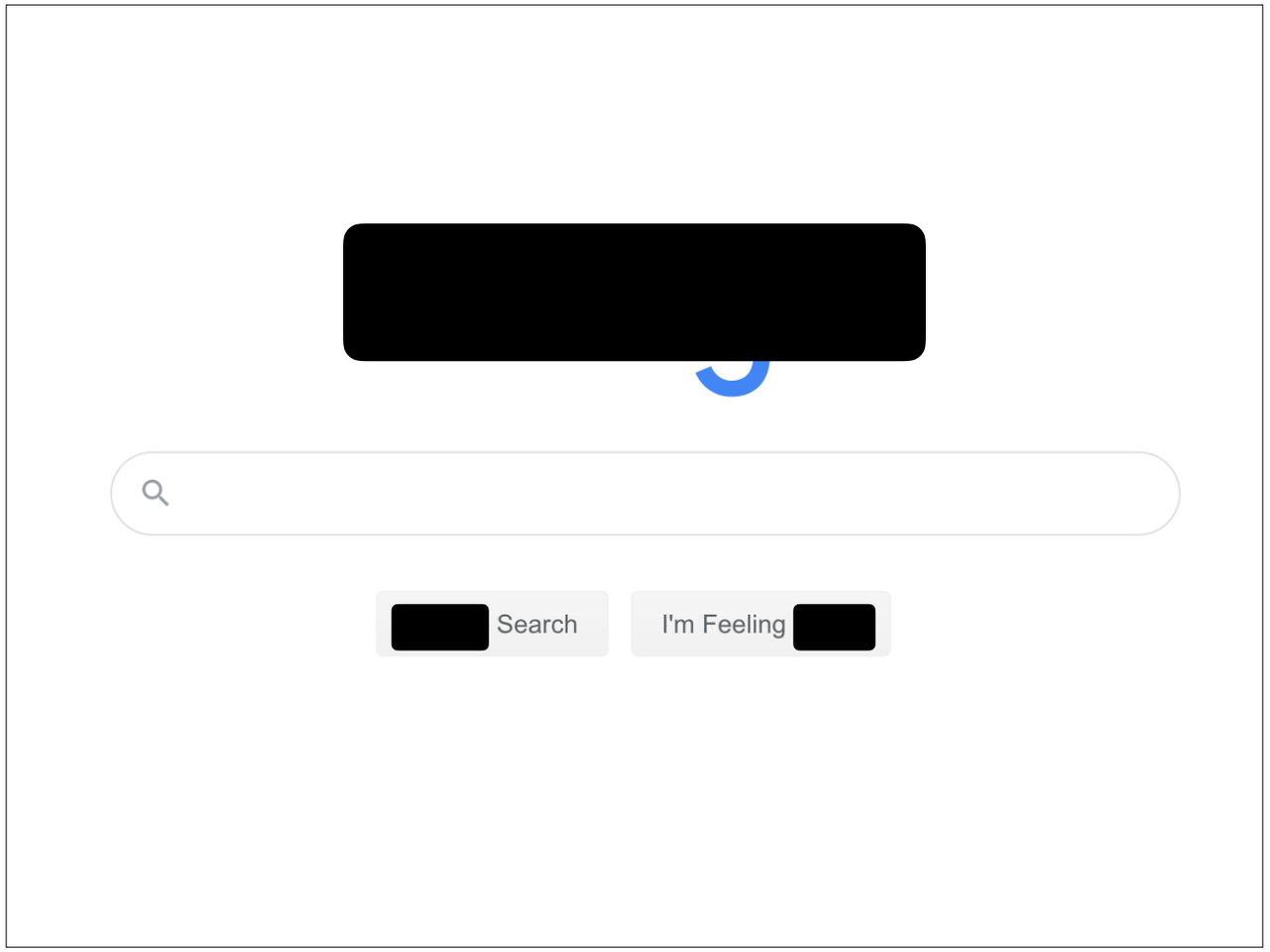
overpass-turbo.eu

Here’s the URL, please check out Overpass Turbo.

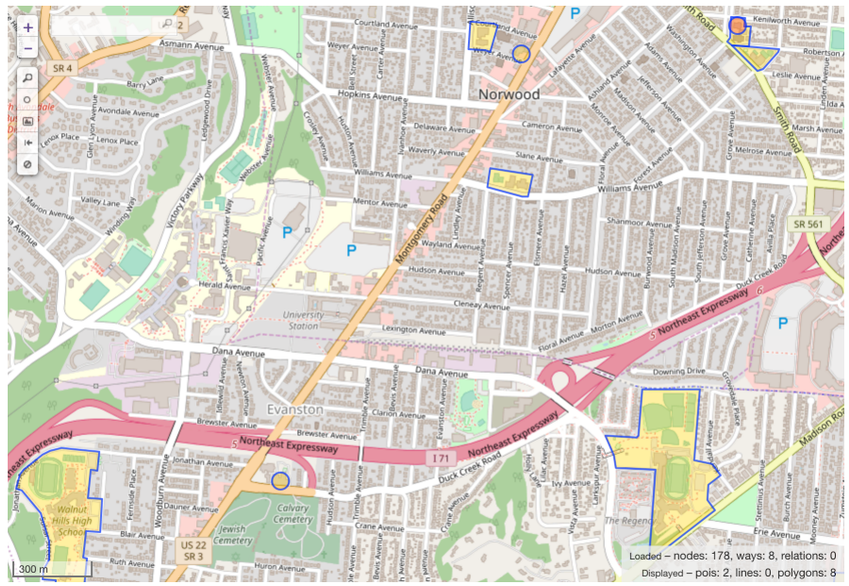
Before I walk you through OverpassQL, I'm going to walk you through a simpler "wizard" syntax that Overpass Turbo uses to generate OverpassQL. Yes, code that generates code.



To use the wizard, click the toolbar button, enter your wizard query, and click Build and Run.

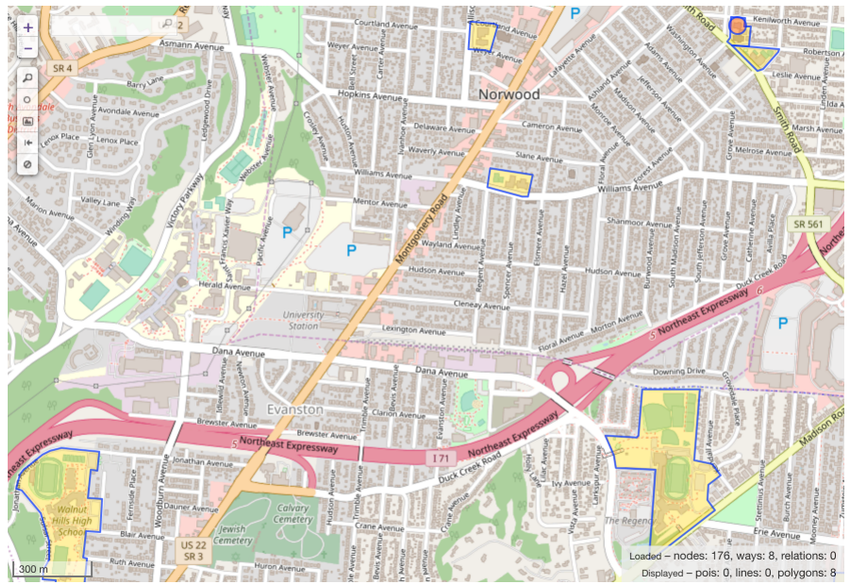


Think of the wizard syntax as similar to a search engine whose map service shall not be named.



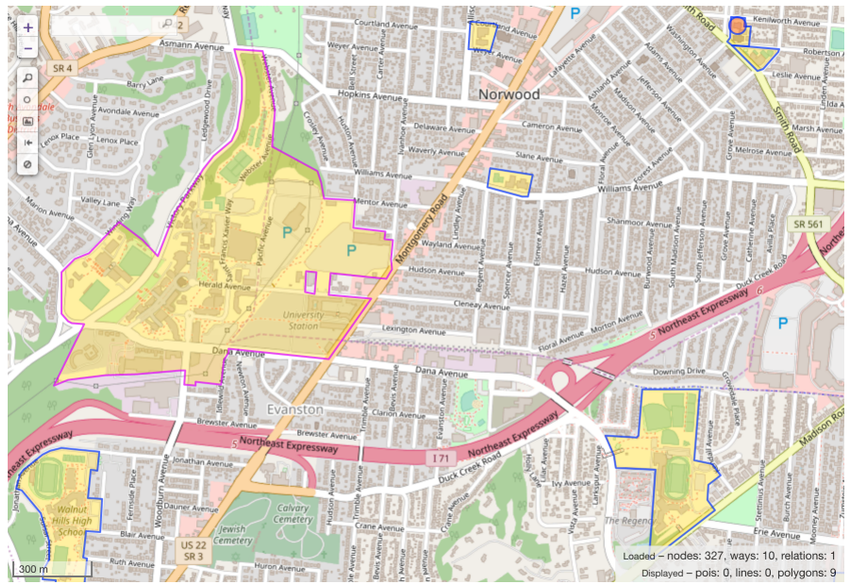
amenity=school

At its core is the key=value syntax that mappers often use in conversation. This query finds schools that are tagged amenity=school.



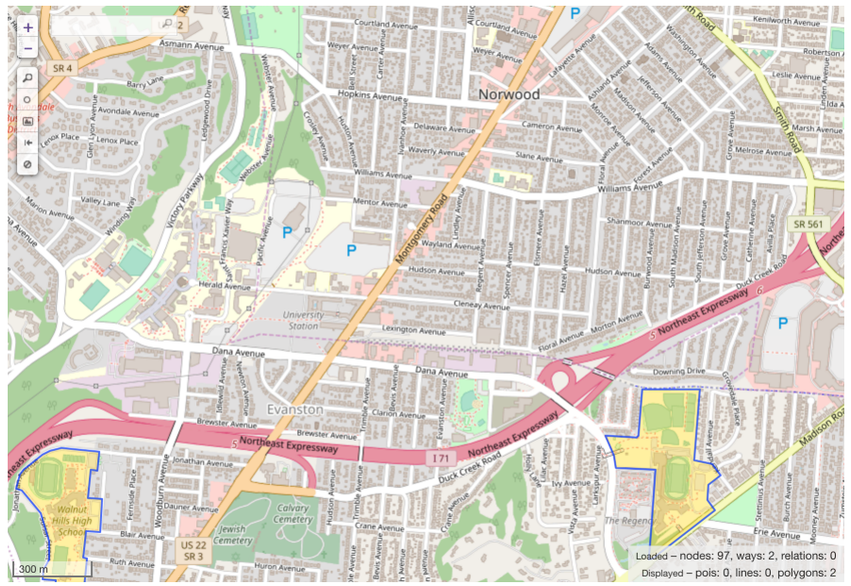
amenity=school
and type:way

This query filters the results down to just school campuses that have been mapped as closed ways. Notice the “and” operator as well as the “type:” operator. You can set the type to node, way, or relation. Or you can omit it if you don’t care or don’t know.



(amenity=school or amenity=university) and (type:way or type:relation)

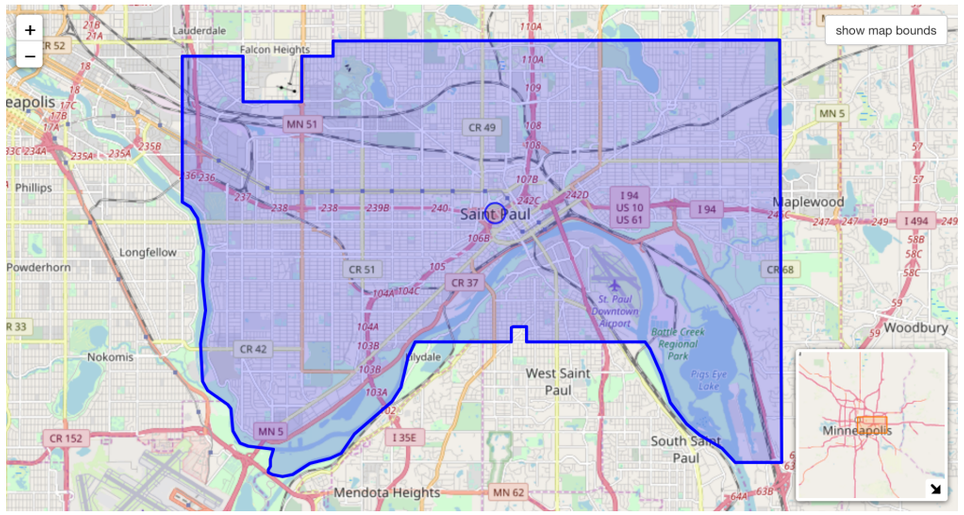
This query expands the results to include university campuses, even if they’re mapped as multipolygon relations.



amenity=school
and name~"High School"

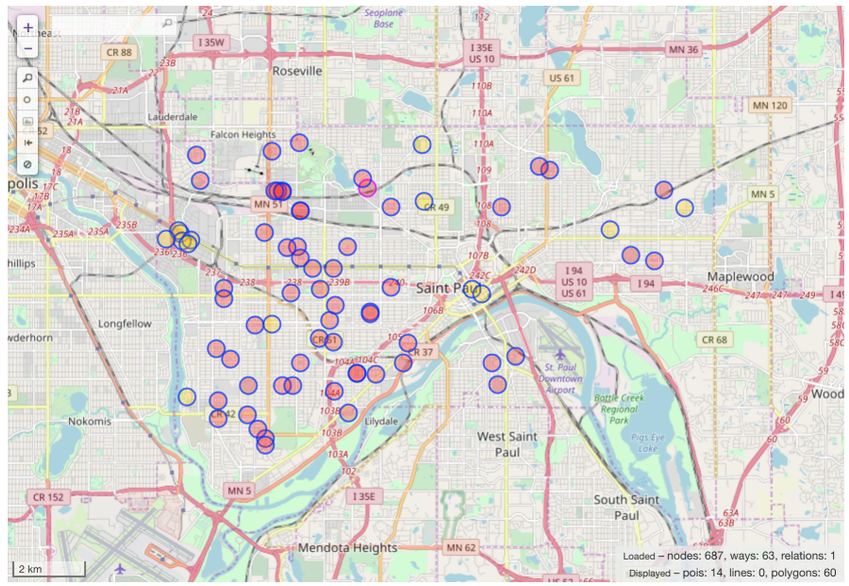
Regular expressions

Sometimes you want to match a key but don’t know the exact value. This query matches schools with “High School” in the name. Notice how it says name tilde, not name equals. The tilde means the subsequent string is a regular expression. Even if you aren’t comfortable writing a regular expression, this operator is pretty handy for efficiently listing out multiple values or matching a substring.



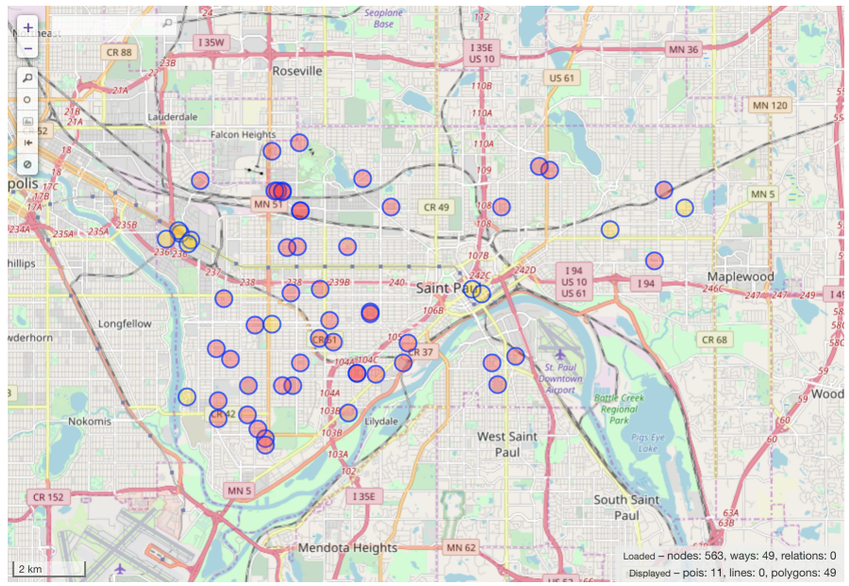
Saint Paul, Minnesota

Up to now, the queries have all defaulted to the current map viewport, which you can change at any time. But what if you want to search in a non-rectangular area? For example, what if you want to search in St. Paul but not in W. St. Paul or S. St. Paul?



amenity=school
in "Saint Paul, Minnesota"

That’s what the “in” operator is for. Overpass Turbo searches Nominatim, the search engine on osm.org, gets the first place returned by Nominatim, and limits the query’s results to within that place’s boundary.



amenity=school and operator!=*
in "Saint Paul, Minnesota"

Wildcards are also allowed. With the not-equals sign and asterisk, this query gives us schools that aren’t tagged with operators. Perhaps an opportunity for some armchair school district tagging this weekend.

More wizardry

- amenity=school and user:"Marvelous Mapper"
- amenity=school newer:1day
- ~"(disused:|demolished:)?amenity"~"school"
- amenity=language_school global

The wizard supports several other operators, like filtering by who last touched a feature or when they touched it; or using regular expressions on the keys, not just the tags; or searching globally.

OK, we've looked at some examples of wizard queries, but wizard queries are fairly basic and you can't tweak them very much. Now I'm going to show you how these wizard queries translate to OverpassQL, the actual language for querying Overpass.

amenity=school

```
[out:json][timeout:25];
(
  node["amenity"="school"]
    ({{bbox}});
  way["amenity"="school"]
    ({{bbox}});
  relation["amenity"="school"]
    ({{bbox}});
);
out body;
>;
out skel qt;
```

Here’s that really basic amenity=school query again, and you can see why the wizard language exists – that’s a lot of gobbledygook for filtering on a single tag!

amenity=school

```
[out:json][timeout:25];
(
  node["amenity"="school"]({{bbox}});
  way["amenity"="school"]({{bbox}});
  relation["amenity"="school"]
    ({{bbox}});
);
out body;
>;
out skel qt;
```

Let’s break it down and see what each part means.

amenity=school

```
[out:json][timeout:25];
(
  node["amenity"="school"]({{bbox}});
  way["amenity"="school"]({{bbox}});
  relation["amenity"="school"]
    ({{bbox}});
);
out body;
>;
out skel qt;
```

The first part is the settings. Here it specifies the output format and timeout. Normally you can leave the settings alone.

amenity=school

```
[out:json][timeout:25];
(
  node["amenity"="school"]({{bbox}});
  way["amenity"="school"]({{bbox}});
  relation["amenity"="school"]
    ({{bbox}});
);
out body;
>;
out skel qt;
```

The wizard generated one line for each kind of element: node, way, and relation. This first line says, “Get me the nodes that are tagged amenity=school and that are located within this bounding box.” Notice how these lines are surrounded by parentheses. The parentheses group the lines, unioning their results into a single set. If not for the parentheses, Overpass would look for all the nodes, then throw away the results and look for all the ways, and so on.

amenity=school

```
[out:json][timeout:25];
(
  node["amenity"="school"]({{bbox}});
  way["amenity"="school"]({{bbox}});
  relation["amenity"="school"]
    ({{bbox}});
);
out body;
>;
out skel qt;
```

Finally, the wizard generates these lines that output the matching feature, then recurses down into the nodes that make up the feature and outputs those too. Depending on the query, this can be overkill and just outputting the body is enough. Remember, Overpass Turbo's wizard is generating all this code for you, so if it's overwhelming, don't fret, you don't have to muck with it normally.

amenity=school

```
[out:json][timeout:25];
(
  node["amenity"="school"]
    ({{bbox}});
  way["amenity"="school"]
    ({{bbox}});
  relation["amenity"="school"]
    ({{bbox}});
);
out body;
>;
out skel qt;
```

So now that we’ve dissected the simplest possible query, let’s see what other generated queries look like.

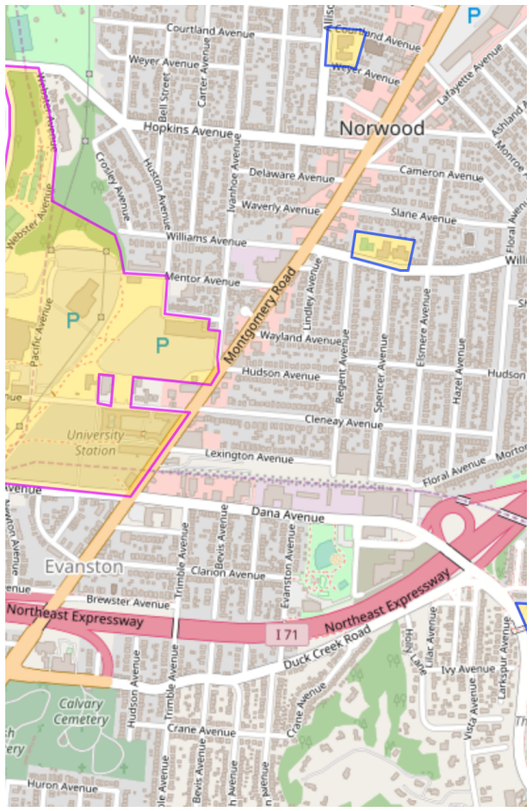
amenity=school and type:way

```
[out:json][timeout:25];
(
  way["amenity"="school"]
    ({{bbox}});
);
out body;
>;
out skel qt;
```

If we add type:way, the node and relation lines disappear and the query runs faster. So if you know you only care about one type of element, you should add the type operator.

(amenity=school or amenity=university) and (type:way or type:relation)

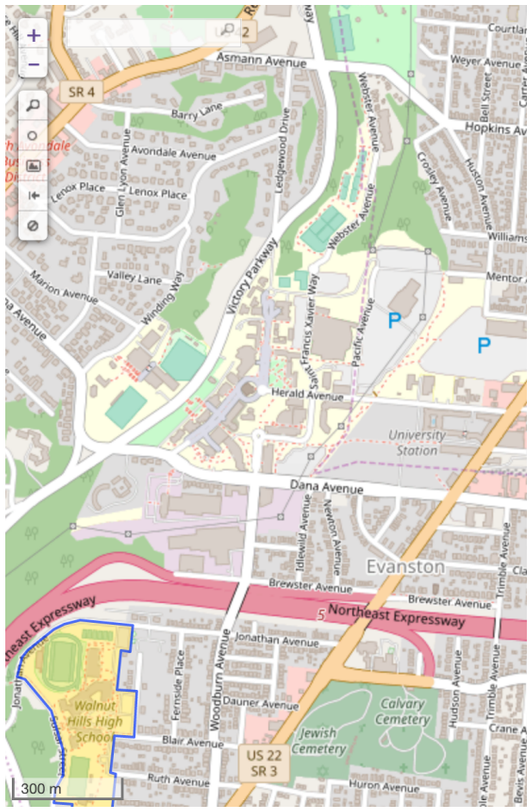
```
[out:json][timeout:25];
(
  way["amenity"="school"]
    ({{bbox}});
  relation["amenity"="school"]
    ({{bbox}});
  way["amenity"="university"]
    ({{bbox}});
  relation["amenity"="university"]
    ({{bbox}});
);
out body; >; out skel qt;
```



Here we see the “or” operator adds more lines to the query, which could potentially slow things down if there are a lot of “or”s.

amenity=school and name~"High School"

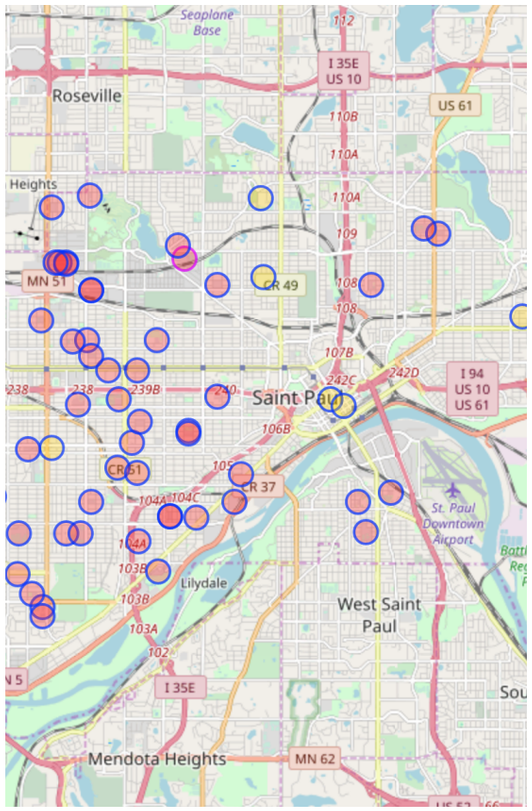
```
[out:json][timeout:25];
(
  node["amenity"="school"]
    ["name"~"High School"]({{bbox}});
  way["amenity"="school"]
    ["name"~"High School"]({{bbox}});
  relation["amenity"="school"]
    ["name"~"High School"]({{bbox}});
);
out body;
>;
out skel qt;
```



The tilde operator works the same way as the equals operator, but for regular expressions.

amenity=school in "Saint Paul, Minnesota"

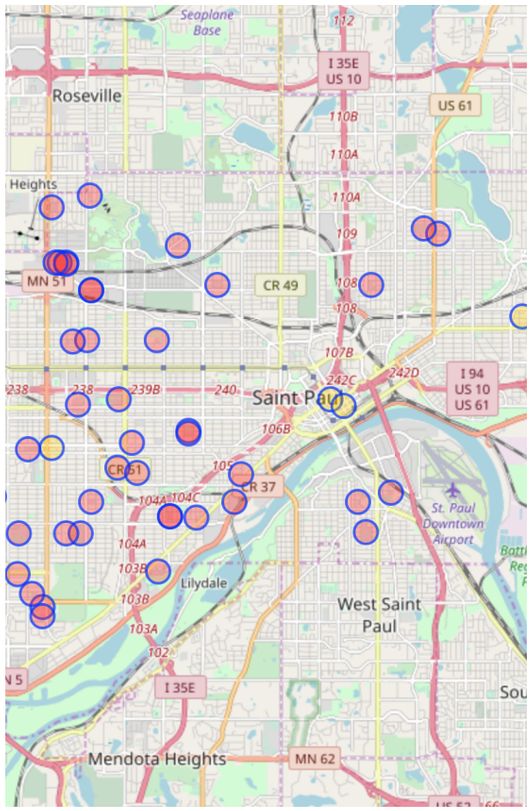
```
[out:json][timeout:25];
{{geocodeArea:Saint Paul, Minnesota}}
  ->.searchArea;
(
  node["amenity"="school"]
    (area.searchArea);
  way["amenity"="school"]
    (area.searchArea);
  relation["amenity"="school"]
    (area.searchArea);
);
out body; >; out skel qt;
```



Here's that "in" operator. Notice the line towards the top that says "geocodeArea". That's a special Overpass Turbo syntax that searches Nominatim. That syntax is replaced by the first Nominatim result's area ID, like a way or relation ID, and the arrow operator (hyphen/greater-than) saves the St. Paul area as "searchArea" so it can be referenced three times below.

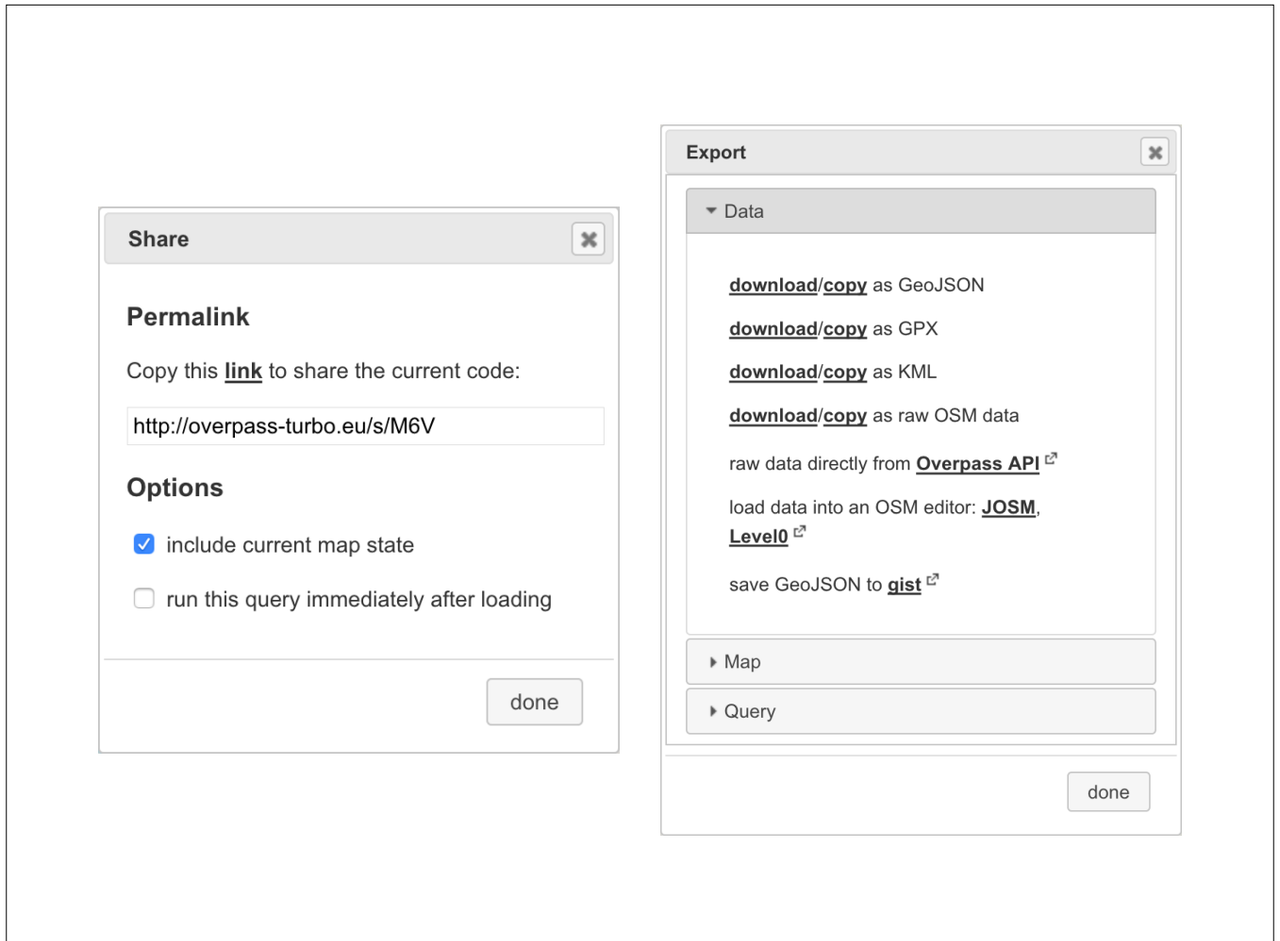
amenity=school and operator!=*

```
[out:json][timeout:25];
(
  node["amenity"="school"]
    ["operator"!~".*"]({{bbox}});
  way["amenity"="school"]
    ["operator"!~".*"]({{bbox}});
  relation["amenity"="school"]
    ["operator"!~".*"]({{bbox}});
);
out body; >; out skel qt;
```

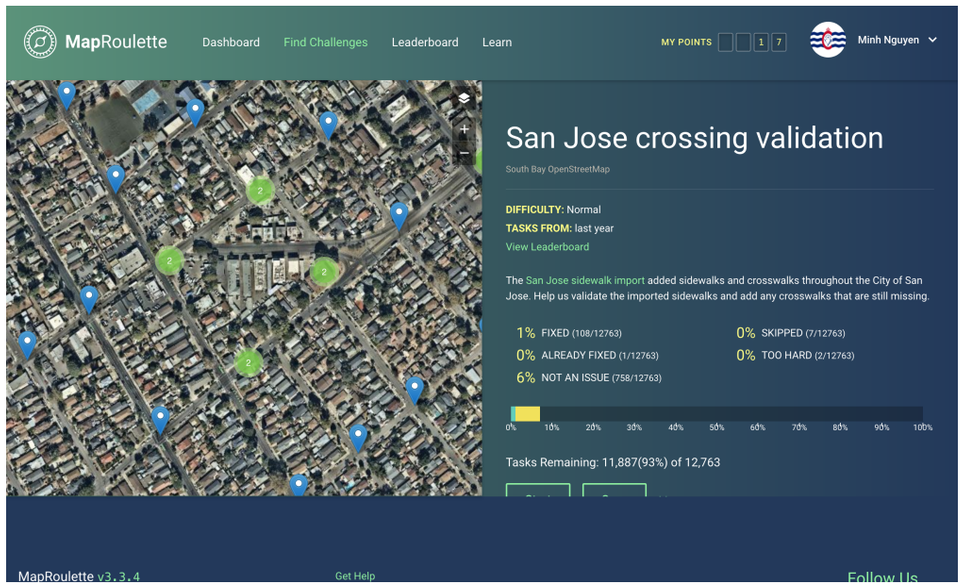


The wizard translates wildcards into regular expression operators in OverpassQL.

Now that I've bored you with all that programming minutiae, you might be wondering how to use the results beyond the interactive map that you get by default.

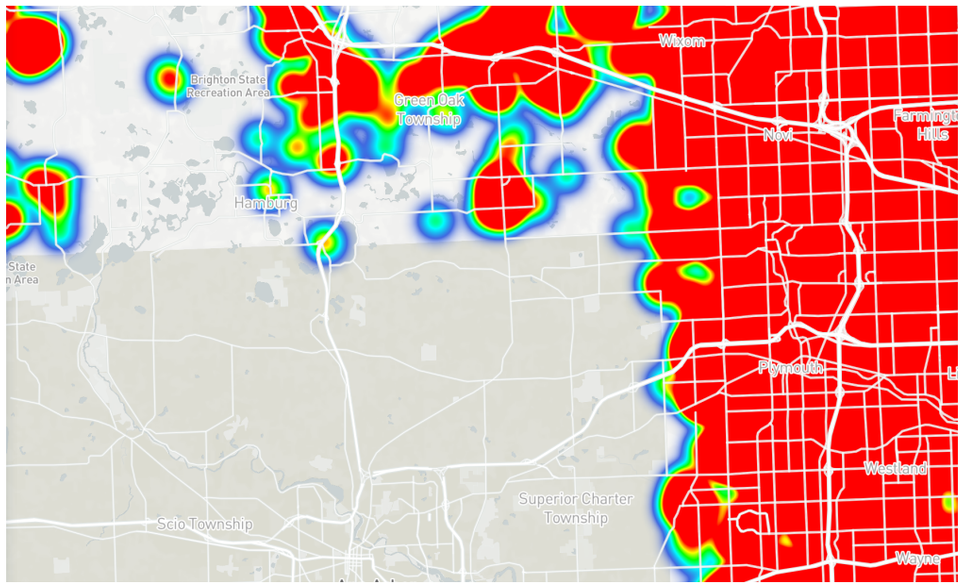


In the toolbar next to the Run button are Share and Export buttons. The Share button generates a permalink to the query. The Export button is a treasure trove. You can export the results in several formats like GeoJSON and KML, or you can generate a static map for sharing.



MapRoulette

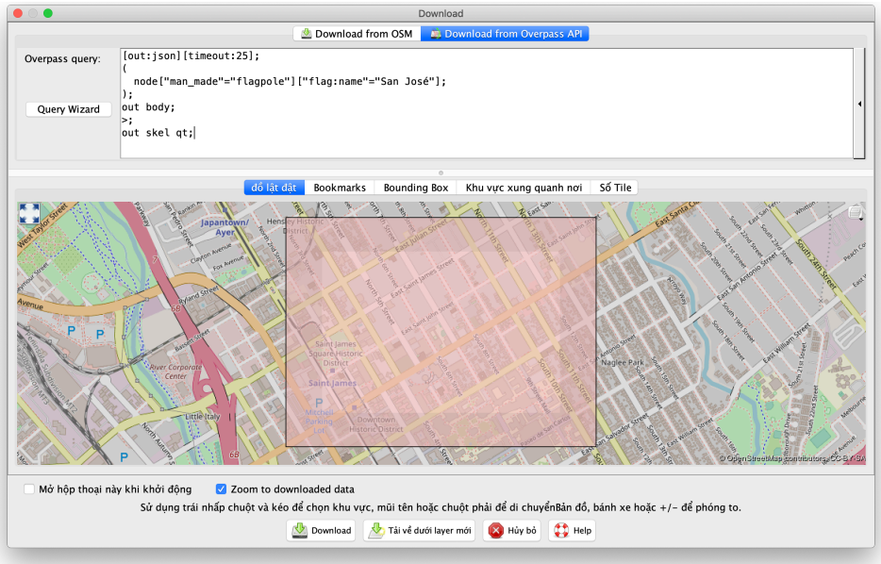
Exported Overpass results can be imported into several tools. For example, you can query Overpass for bad tagging, then export the results to MapRoulette, which makes it easy to coordinate a campaign to fix the bad tagging.



Mapbox Studio

tinyurl.com/sotmus18heatmaps2

At last year’s State of the Map U.S., I gave a workshop about building a heat map in Mapbox Studio using Overpass results. The workshop wasn’t recorded, but a written version is available at this URL.



JOSM

Expert Mode

Did you know JOSM has an “expert mode”? If you go in Preferences and enable expert mode, the Download dialog gains a tab where you can download features matching an Overpass query. This makes it easier to make systematic edits in an area of interest without ever leaving JOSM.

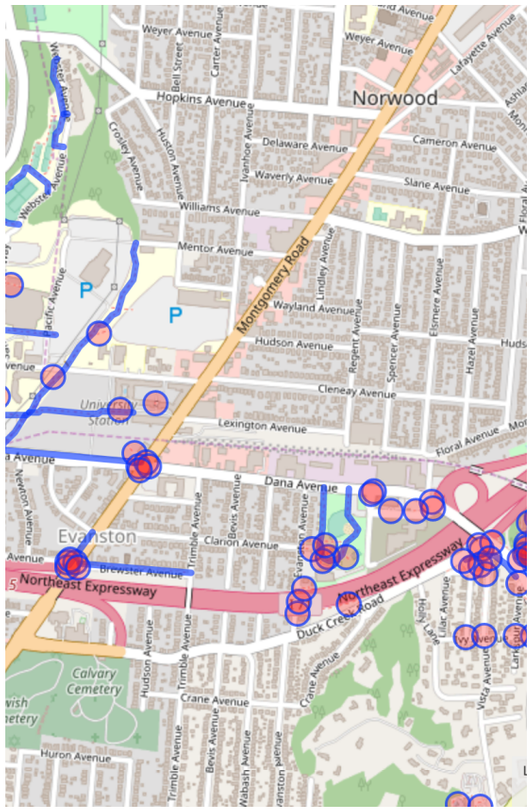
So far I've shown you what the wizard can generate, but the reason I showed you OverpassQL is that the language can do so much more.

Footways intersecting roadways

```
[out:json][timeout:25];

way["highway"!="footway"]
  ({{bbox}});
node(w);
way(bn)["highway"="footway"];

out body;
>;
out skel qt;
```



For example, recursion statements and filters let you drill down into a way’s nodes or a relation’s members or drill up into the ways containing a node. This query finds footways that intersect roadways, possibly candidates for crosswalk tagging.

Footways intersecting roadways

```
[out:json][timeout:25];

way["highway"!="footway"]
  ({{bbox}});
node(w);
way(bn)["highway"="footway"];

out body;
>;
out skel qt;
```

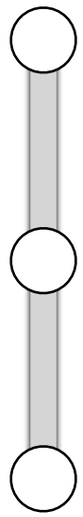
First it finds roadways

Footways intersecting roadways

```
[out:json][timeout:25];

way["highway"!="footway"]
  ({{bbox}});
node(w);
way(bn)["highway"="footway"];

out body;
>;
out skel qt;
```



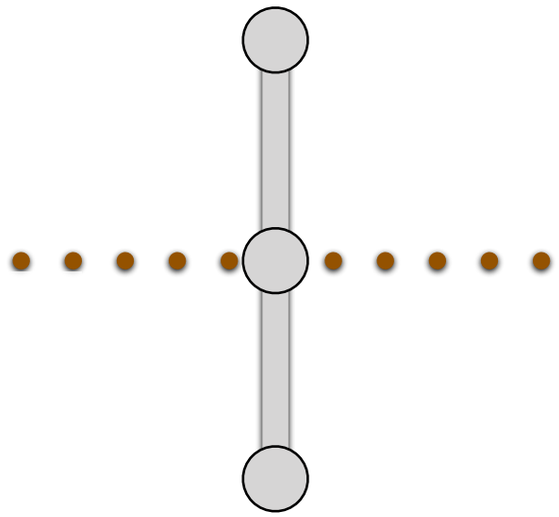
then recurses down to the nodes forming those ways

Footways intersecting roadways

```
[out:json][timeout:25];

way["highway"!="footway"]
  ({{bbox}});
node(w);
way(bn)["highway"="footway"];

out body;
>;
out skel qt;
```



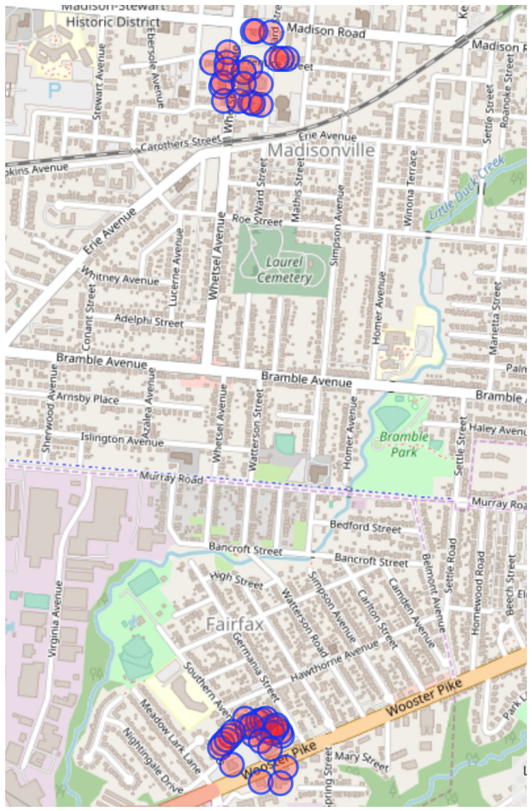
then recurses back up to the footways that incorporate those nodes.

Buildings near tornado sirens

```
[out:json][timeout:25];

node["emergency"="siren"]
  ({{bbox}});
way(around:100)["building"];

out body;
>;
out skel qt;
```



There’s an around filter for finding features within a certain radius of another set of features. This query finds buildings near tornado sirens.

Buildings near tornado sirens

```
[out:json][timeout:25];

node["emergency"="siren"]
  ({{bbox}});
way(around:100)["building"];

out body;
>;
out skel qt;
```

First it finds the siren nodes

Buildings near tornado sirens

```
[out:json][timeout:25];

node["emergency"="siren"]
  ({{bbox}});
way(around:100)["building"];

out body;
>;
out skel qt;
```

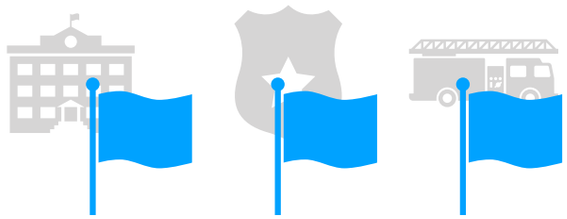
then finds the building ways within 100 meters of those nodes.

Flagpoles not in front of fire stations

```
[out:json][timeout:25];
node["man_made"="flagpole"]({{bbox}})
  ->.flagpoles;
(
  node["amenity"="fire_station"]
    ({{bbox}});
  way["amenity"="fire_station"]
    ({{bbox}});
);
node["man_made"="flagpole"]
  (around:100)->.firepoles;
(node.flagpoles; - node.firepoles;);
out body;
```

There isn't a "not near" filter, but we can achieve the same effect using a difference statement. This query finds flagpoles that are *not* in front of fire stations. First we find all the flagpoles and save it as the flagpoles set. Then we find the fire stations, then the flagpoles near those fire stations, saving that set too. Then we subtract one from the other, surrounded by parentheses.

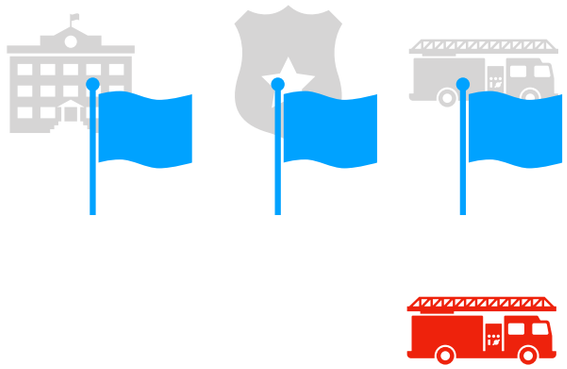
Flagpoles not in front of fire stations



```
[out:json][timeout:25];
node["man_made"="flagpole"]({{bbox}})
  ->.flagpoles;
(
  node["amenity"="fire_station"]
    ({{bbox}});
  way["amenity"="fire_station"]
    ({{bbox}});
);
node["man_made"="flagpole"]
  (around:100)->.firepoles;
(node.flagpoles; - node.firepoles;);
out body;
```


First we find all the flagpoles and save it as the flagpoles set.

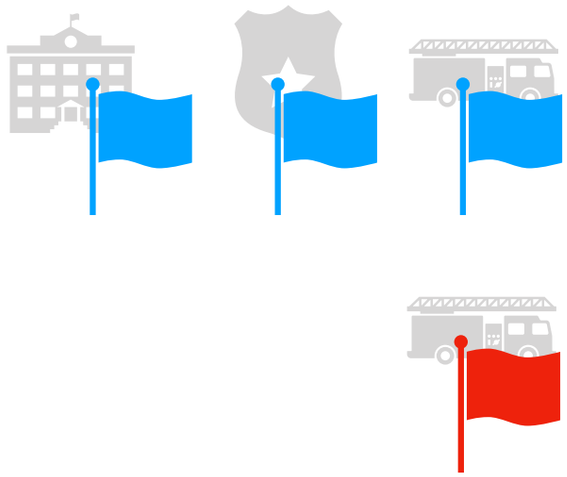
Flagpoles not in front of fire stations



```
[out:json][timeout:25];
node["man_made"="flagpole"]({{bbox}})
  ->.flagpoles;
(
  node["amenity"="fire_station"]
    ({{bbox}});
  way["amenity"="fire_station"]
    ({{bbox}});
);
node["man_made"="flagpole"]
  (around:100)->.firepoles;
(node.flagpoles; - node.firepoles;);
out body;
```


Then we find the fire stations

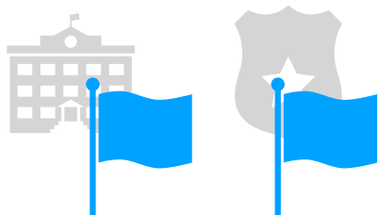
Flagpoles not in front of fire stations



```
[out:json][timeout:25];
node["man_made"="flagpole"]({{bbox}})
  ->.flagpoles;
(
  node["amenity"="fire_station"]
    ({{bbox}});
  way["amenity"="fire_station"]
    ({{bbox}});
);
node["man_made"="flagpole"]
  (around:100)->.firepoles;
(node.flagpoles; - node.firepoles;);
out body;
```


then the flagpoles near those fire stations, saving that set too.

Flagpoles not in front of fire stations



```
[out:json][timeout:25];
node["man_made"="flagpole"]({{bbox}})
  ->.flagpoles;
(
  node["amenity"="fire_station"]
    ({{bbox}});
  way["amenity"="fire_station"]
    ({{bbox}});
);
node["man_made"="flagpole"]
  (around:100)->.firepoles;
(node.flagpoles; - node.firepoles;);
out body;
```

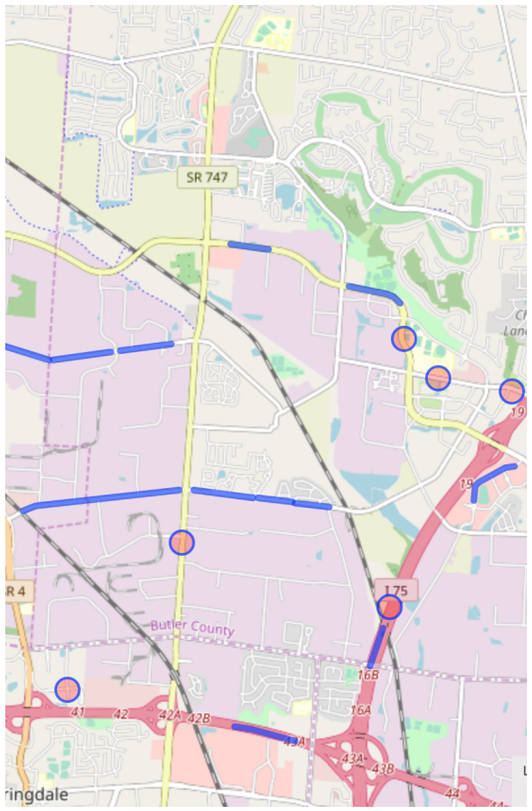

Then we subtract one from the other, surrounded by parentheses.

Long turn lanes

```
[out:json][timeout:25];

way[~"^turn"~".*"]({{bbox}})
  (if: length() > 300);

out body;
>;
out skel qt;
```



An “if” evaluator syntax lets you decide whether features should be returned on an individual basis. This query finds turn lane ways but only returns those that are over 300 meters long. (It turns out most of them are center turn lanes.)

Total length of bike infrastructure

```
[out:json][timeout:25];
(
  {{geocodeArea:Minneapolis}};
  {{geocodeArea:Saint Paul, Minnesota}};
)->.searchArea;

(
 way[~"cycleway"~"lane|track|opposite|shared_lane|shared|crossing|
opposite_lane|yes|share_busway"](area.searchArea);
 way["highway"="cycleway"](area.searchArea);
 way["bicycle"~"yes|permissive|designated"](area.searchArea);
);

make stats length=sum(length());
out;
```

490½ centerline miles

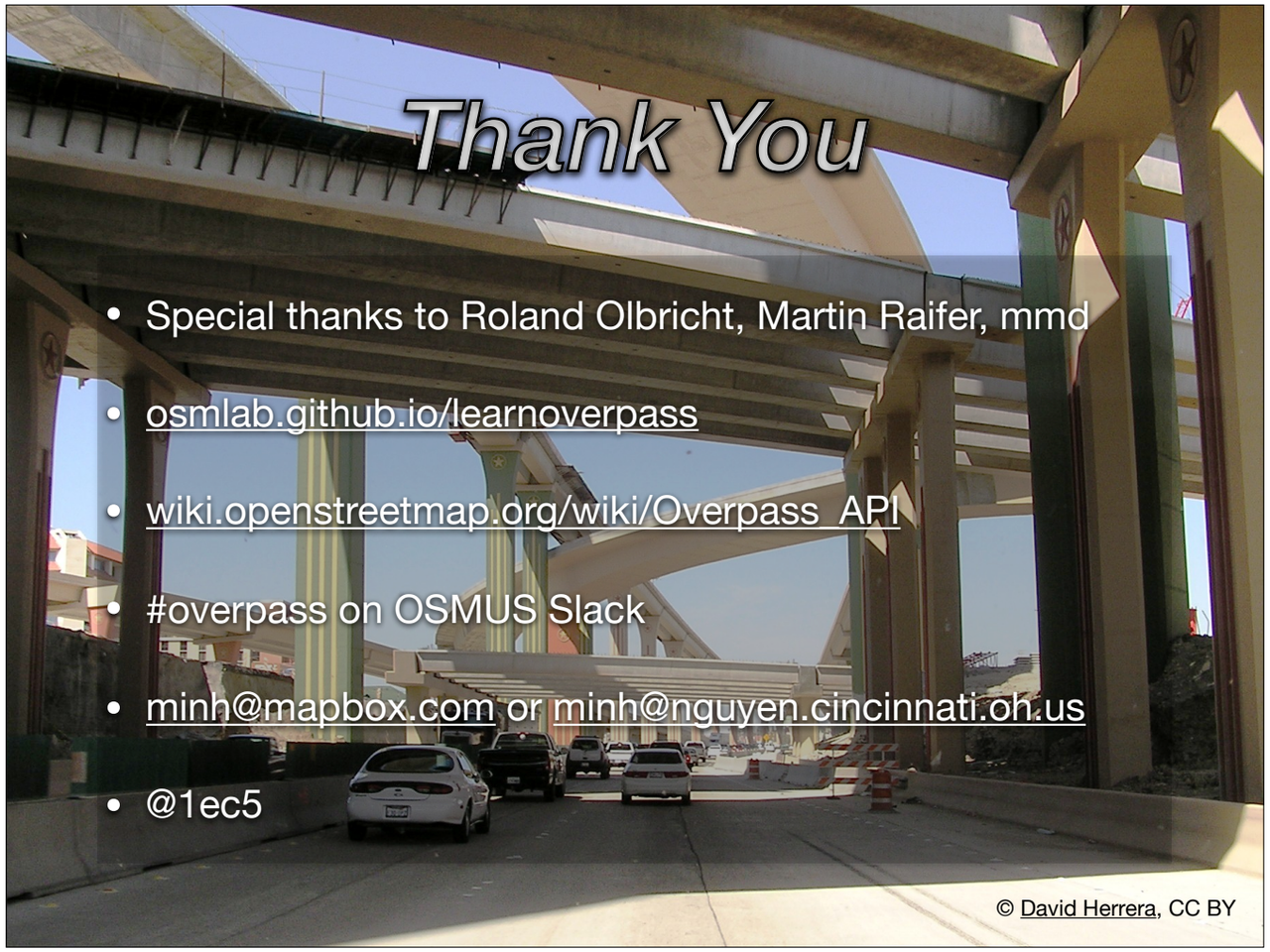
Overpass has some built-in statistical features that are handy for alternate outputs. For example, instead of outputting the bike lanes and sharrows themselves, how about summing up all their lengths and outputting that total in JSON format? Turns out there are just shy of 500 miles of bikeways mapped in the Twin Cities.

Performance

- Avoid overspecifying filters
- Increase timeout: `[timeout:60]` (1 minute)
- Increase memory: `[maxsize:1073741824]` (1 GB)
- Check the rate limiting status for your IP address: overpass-api.de/api/status

A final note about performance. As you play with Overpass, you'll quickly run into situations where it can't respond to your request in a reasonable amount of time. There are limits to ensure that no one user monopolizes the community service by accident. First, you should make sure your filters aren't overly qualified with too many details. Each additional filter creates more work for the server, not less. For example, if you want to find all the McDonald's, just search for name=McDonald's, not name=McDonald's and building=retail and amenity=fast_food and cuisine=burger.

If you know what you're doing, you can increase the timeout beyond a minute. But pay attention to rate limits – your IP address will be throttled if you run too many large queries in a short time span.



I think that's enough confusion for one day. Hopefully I've piqued your interest and you'll all try your hands at using Overpass in your day-to-day mapping. I want to extend a special thanks to Roland, Martin, mmd, and others who develop and maintain these tools. If you have any questions, please join OSMUS Slack and ask in the #overpass channel, or feel free to reach out to me directly. Thank you!