Connecting the Geospatial Dots with Raku

by Brian Duggan

bduggan

FOSDEM 2025

- Brian Duggan
- Logistics Engineer at Instacart
- We have a bottom-up geospatial culture.

rough outline

- About Raku
- Geospatial Data with Nominatim and Overpass
- Geospatial Visualization with Leaflet and Deck.gl
- Geospatial Analysis with DuckDB and GEOS
- Conclusion

```
say "hello, world!";
```

Raku philosophy

There's Always More Than One Way To Do It
...But Some Ways Are More Equal Than Others.

Easy things should be trivial;
Hard things should be easy;
Impossible things should be (merely) hard.

In Raku we seek to support developers without getting in their way, by giving them clean, efficient, and robust tools that adapt easily to however they prefer to think about problems and code solutions.

(Damian Conway)

Portrait of Gerhardus Mercator 1576, Franz Hogenberg

Nominatim

```
use WebService::Nominatim 'nom';

say nom.search: "Université Libre de Bruxelles";
```

```
[{addresstype => amenity, boundingbox => [50.8096525 50.8159744 4.3777200
4.3851645], category => amenity, display_name => Université libre de Bruxelles
(Campus du Solbosch), Avenue Brillat-Savarin - Brillat-Savarinlaan, Petite
Suisse - Klein-Zwitserland, Ixelles - Elsene, Brussel-Hoofdstad - Bruxelles-
Capitale, Région de Bruxelles-Capitale - Brussels Hoofdstedelijk Gewest, 1050,
België / Belgique / Belgien, importance => 6.219958800636328e-05, lat =>
50.81347185, licence => Data © OpenStreetMap contributors, ODbL 1.0.
http://osm.org/copyright, lon => 4.381235729203142, name => Université libre
de Bruxelles (Campus du Solbosch), osm_id => 13699100, osm_type => relation,
place_id => 96092240, place_rank => 30, type => university}]
```

https://nominatim.org

Nominatim

Blog Documentation Cookbook Downloads Support Us

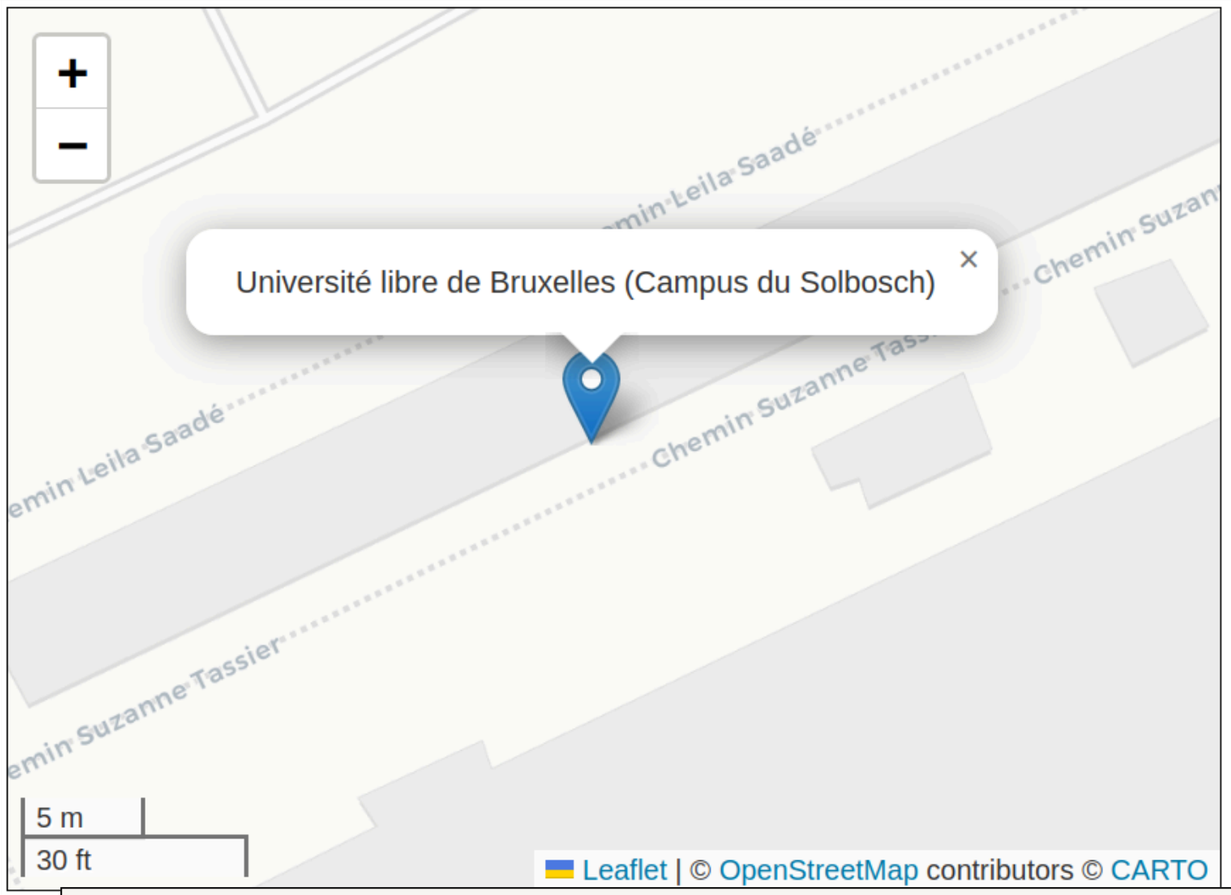
Open-source geocoding with OpenStreetMap data

Nominatim uses OpenStreetMap data to find locations on Earth by name and address (geocoding). It can also do the reverse, find an address for any location on the planet.

- `nom.search:` is equivalent to `nom.search(...)`

Nominatim + Leaflet

```
use WebService::Nominatim 'nom';
use Map::Leaflet 'map';

my $name = "Université Libre de Bruxelles";
for nom.search: $name {
  map.add-marker: .<lat lon>, .<name>;
}

map.show;
```



https://leafletjs.org

Leaflet

an open-source JavaScript library
for mobile-friendly interactive maps

Overview Tutorials Docs Download Plugins Blog

May 18, 2023 — Leaflet 1.9.4 has been released!

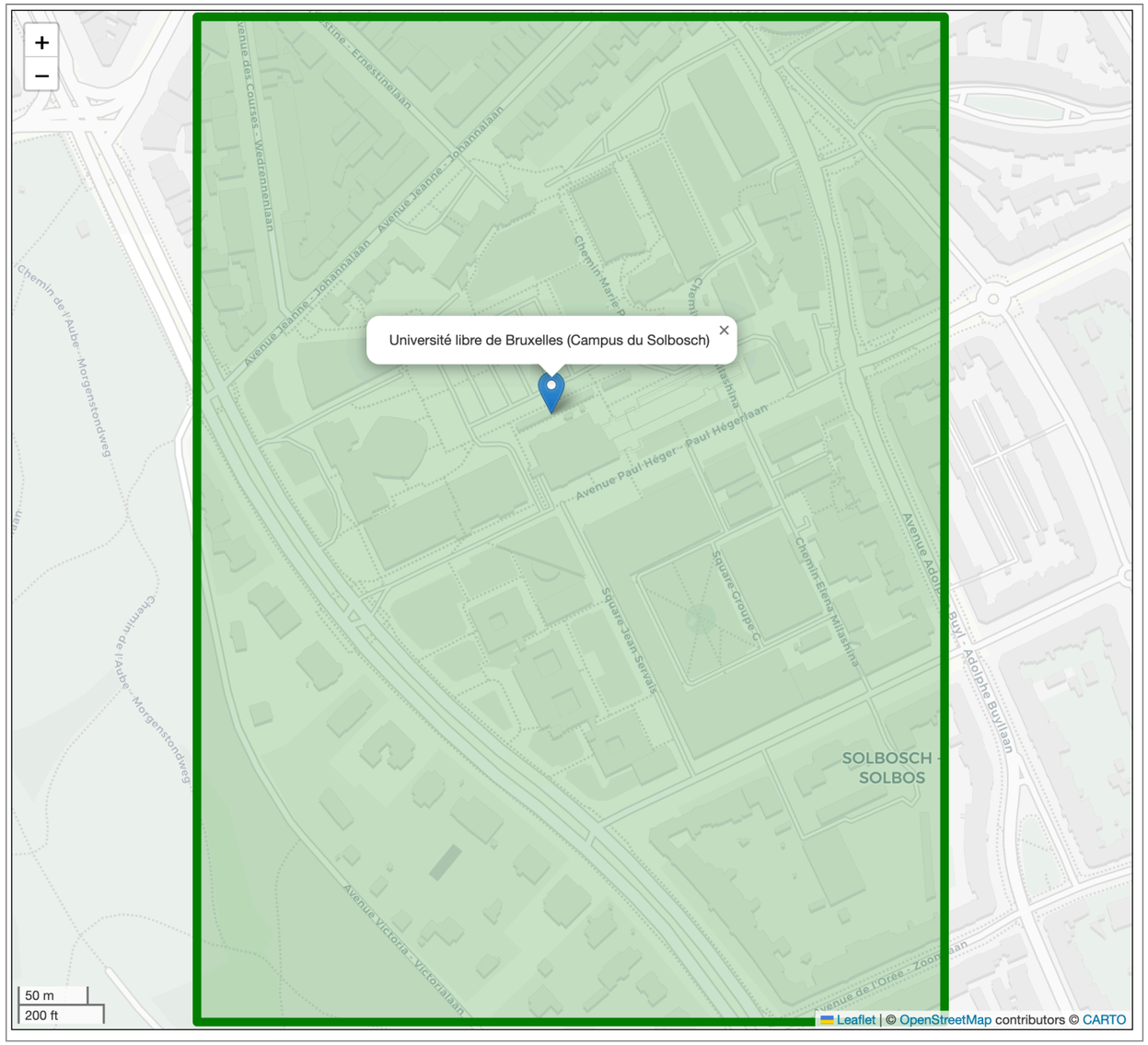
Leaflet is the leading open-source JavaScript library for mobile-friendly interactive maps. Weighing just about 42 KB of JS, it has all the mapping features most developers ever need.

Leaflet is designed with *simplicity*, *performance* and *usability* in mind. It works efficiently across all major desktop and mobile platforms, can be extended with lots of plugins, has a beautiful, easy to use and well-documented API and a simple, readable source code that is a joy to contribute to.

- The topic variable, $_ , is the default for loops and blocks
- And the default invocant (before a .)
- .<lat lon> retrieves hash slice

Rectangles

```
use WebService::Nominatim 'nom';
use Map::Leaflet 'map';

my $name = "Université Libre de Bruxelles";
for nom.search: $name {
  map.add-marker: .<lat lon>, .<name>;
  map.create-rectangle:
    bounds => .<boundingbox>[ [0,2], [1,3] ],
    :stroke,
    :color<green>,
    :weight(8)
}

map.show;
```



Slice the bounding box into coordinates!

```
my @bbox = <0 10 20 30>;

say @bbox[0,1,2,3];         # 0 10 20 30
say @bbox[0,3,1,2];         # 0 30 10 20
say @bbox[ [0,2], [1,3] ]; # (0 20) (10 30)
```

- Brackets [...] retrieve a list slice
- :color<green> makes a Pair , color => green
- :stroke makes stroke => True

Leaflet classes

Classes in Map::Leaflet

```
class Map::Leaflet::Rectangle is Map::Leaflet::Polyline { ... }

class Map::Leaflet::Polyline is Map::Leaflet::Path { ... }

class Map::Leaflet::Path is Map::Leaflet::InteractiveLayer {
  has Bool $.stroke;
  has Str $.color;
  has Numeric $.weight;
  ...
}

class Map::Leaflet::InteractiveLayer is Map::Leaflet::Layer { ... }

class Map::Leaflet::Layer { ... }
```

Classes in leaflet.js

Rectangle

Creation

Factory	Description
`L.rectangle(<LatLngBounds> latLngBounds, <Polyline options> options?)`	

Options

▶ Options inherited from Polyline

▼ Options inherited from Path

Option	Type	Default	Description
`stroke`	`Boolean`	`true`	Whether to draw stroke along the path. Set it to `false` to disable borders on polygons or circles.
`color`	`String`	`'#3388ff'`	Stroke color
`weight`	`Number`	`3`	Stroke width in pixels
`opacity`	`Number`	`1.0`	Stroke opacity

▶ Options inherited from Interactive layer

▶ Options inherited from Layer

```
my $layer = Map::Leaflet::Rectangle.new(
    bounds => .<boundingbox>[ [0,2], [1,3] ],
    :stroke,
    :color<green>,
    :weight(8)
```

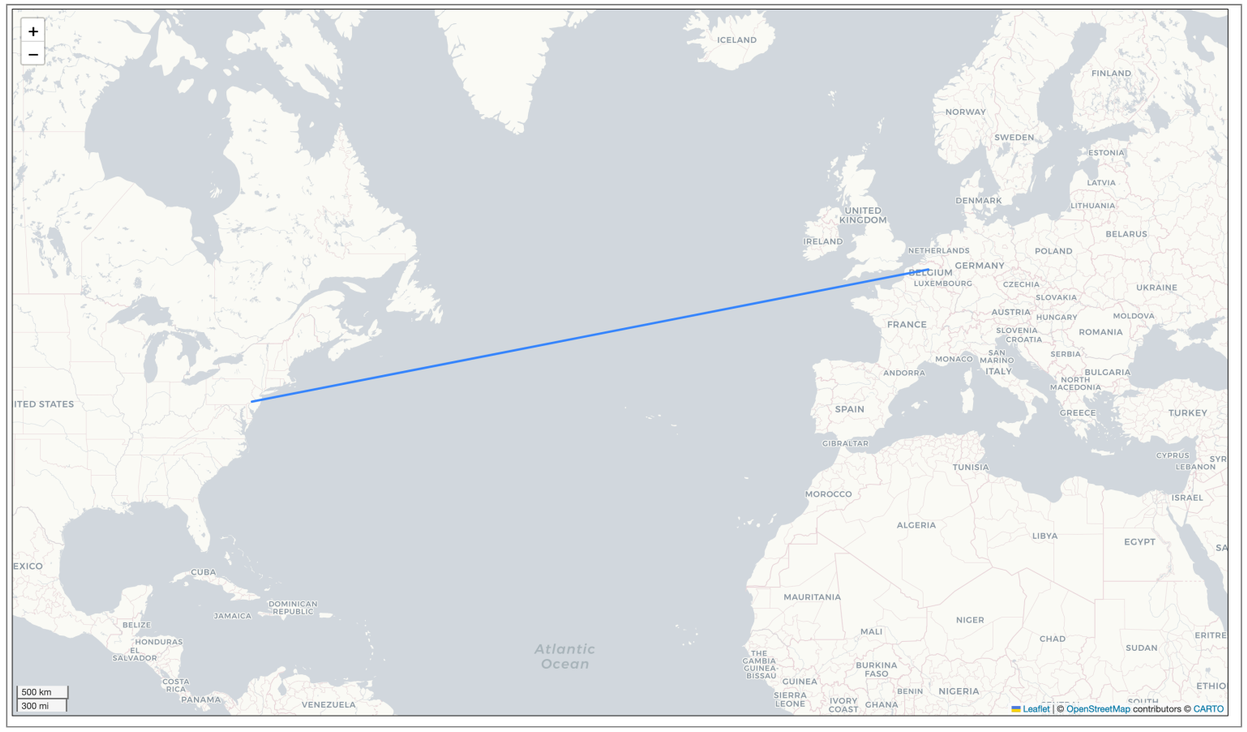
```
map.create-rectangle:
    bounds => .<boundingbox>[ [0,2], [1,3] ],
    :stroke,
    :color<green>,
    :weight(8)
```

- Gradual typing can enforce constraints.
- is is for class inheritance

LLMs

By the way -- LLMs can also generate simple geojson.

```
use Map::Leaflet 'm';
use LLM::DWIM;

m.add-geojson: dwim q:to/AI/;
  Create a GeoJSON LineString that goes
  from Philadelphia to Brussels
  @JSON
  AI

m.show;
```



LLM::DWIM is "Do What I Mean". It builds on many other modules that can interface with OpenAI, Gemini, LLaMA and other LLMs.

There is a library of macros that can be expanded (like @JSON)

- Use heredocs with the q:to/.../ syntax

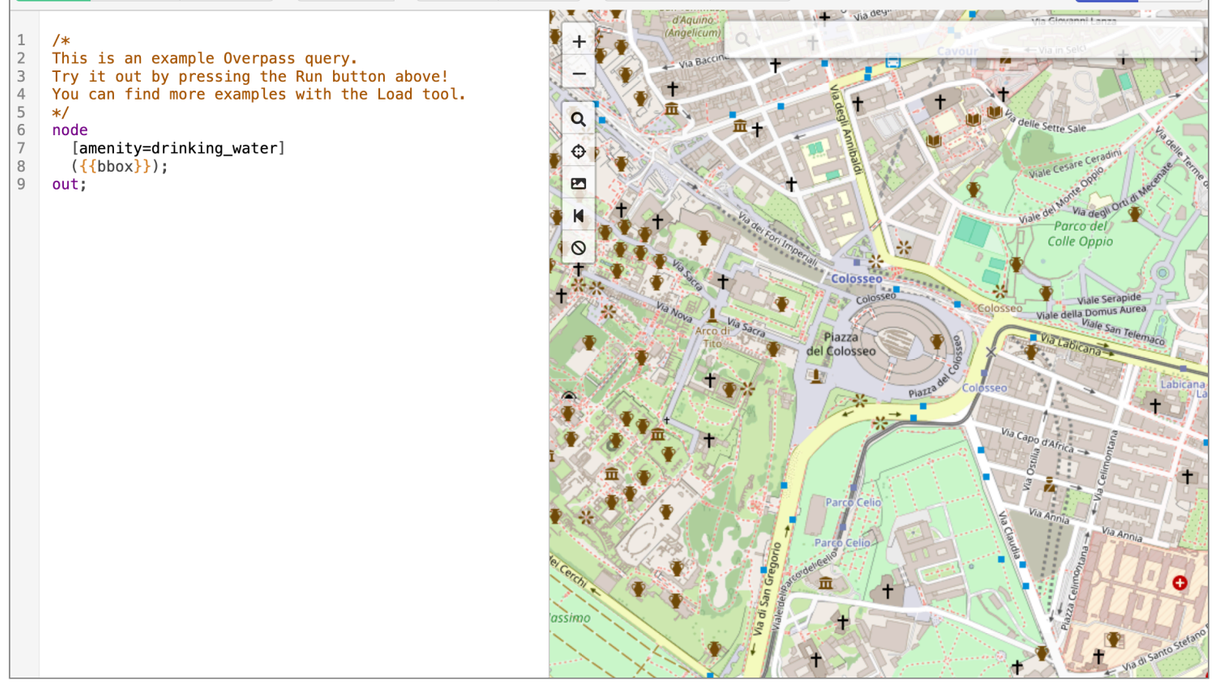
Overpass

Overpass is a read-only API for getting OSM data.

See https://wiki.openstreetmap.org/wiki/Overpass_API

```
use WebService::Overpass 'op';
my $bbox = join ',', 41.884435, 12.484138, 41.895569, 12.49982
op.statements = qq:to/OVERPASS/;
node
  [amenity=drinking_water]
  ({$bbox});
out;
OVERPASS

say op.execute: :json; # or :xml
```

https://overpass-turbo.eu/



Harder challenge

Given a multipolygon, how can we search for all OSM nodes inside it?

We want to build an overpass query like this:

```
  (
    node(poly("10  20  11  22 ... "));
    node(poly("100 200 111 222 ... "));
  )
```

But we have GeoJSON like this:

```
{ "type" : "MultiPolygon",
   "Coordinates" :
   [
     [   # polygon 1
       [ [ 20, 10 ], [22, 11] ... [ 20,10 ] ], # outer ring
         ... # other rings
     ],
     [   # polygon 2
       [ [ 200, 100 ], [ 222, 111 ] ...     ], # outer ring
     ]
   ]
}
```

We need to extract the rings, reverse the coordinates, and flatten the lists.

Easy!

```
my @polys = $geojson<coordinates>[*;0].map: *».reverse[**];
my $query = join "\n", @polys.map: { qq[ node(poly:"$_"); ] };
```

Easy! (explained)

```
my @polys = $geojson<coordinates>[*;0].map: *».reverse[**];
my $query = join "\n", @polys.map: { qq[ node(poly:"$_"); ] };
```

ingredients

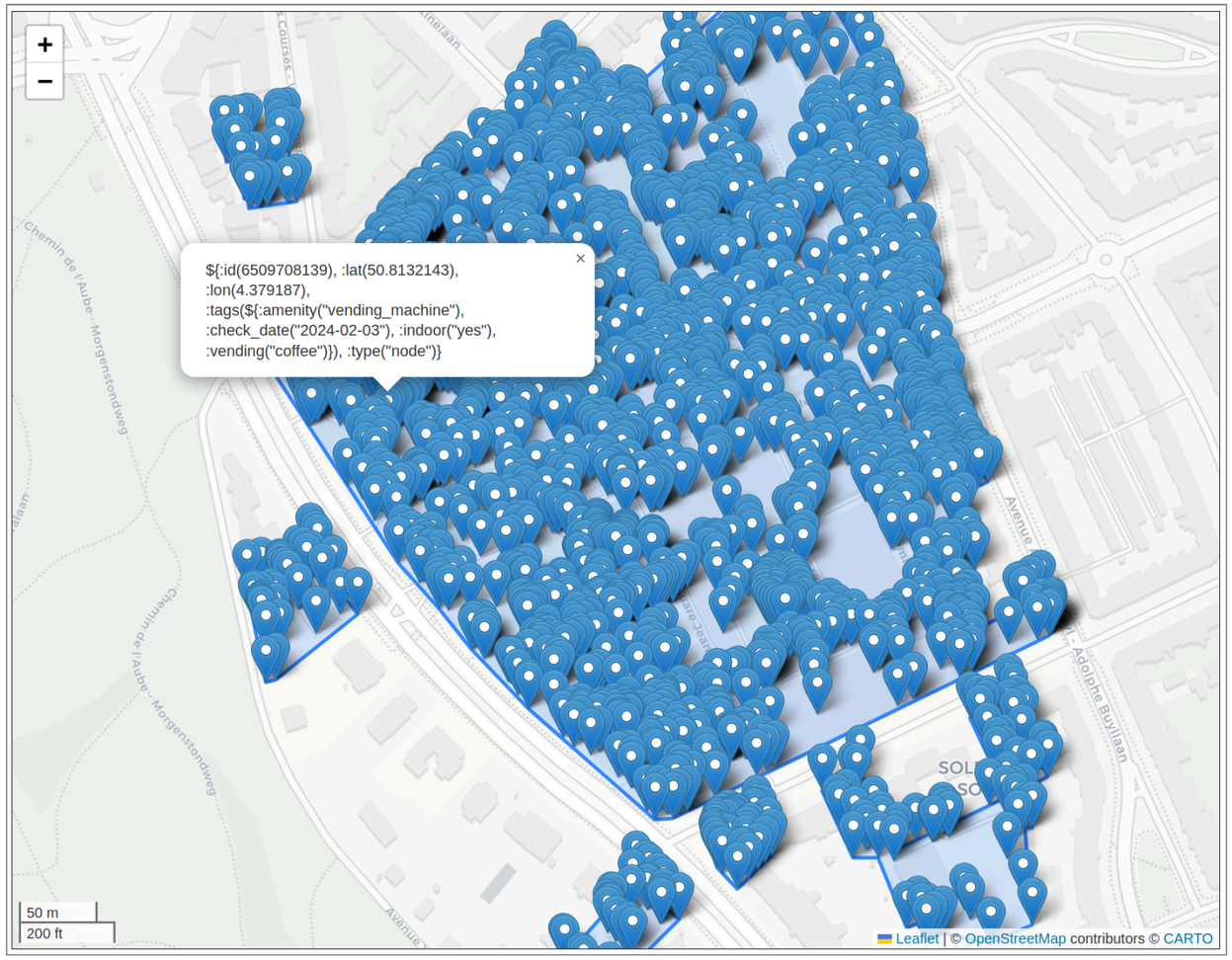
```
@coords[0];   # first row
@coords[*;0]; # first column
@coords[**];  # bulldoze (flatten completely)
```

```
@pairs».reverse;   # flip all pairs
<1 2 3>.map: * + 1 # Whatever in term position
qq[...];           # quote, using `[]` delimiters
```

- `[*;0]` extracts the first ring from each polygon. `*` is the Whatever operator
- `.map` runs the following block for each element
- The next `*` creates the block
- `»` is a hyper operator; like `map` , it applies a method to each item
- `reverse` to flip the lat/lon
- `[**]` A `HyperWhatever` in a list slice is a bulldozing operation (flatten completely)
- `qq` is a quoting construct
- `$_` is the implicit variable in the block for map

Complete code to find all OSM nodes at ULB

```
use WebService::Nominatim 'nom';
use WebService::Overpass 'op';
use Map::Leaflet 'map';

my $name = "Université Libre de Bruxelles";
my ($res) = nom.search: $name, :polygon_geojson;

my @polys = $res<geojson><coordinates>\
  [*;0].map: *».reverse[**];
my $q = join "\n", @polys.map: { qq[ node(poly:"$_"); ] };

op.statements = qq[ ( $q ); out body; ];
my $out = op.execute: :json;

$out<elements>.map: {
  map.add-marker: .<lat lon>, .raku;
}

map.add-geojson: $res<geojson>;
map.show;
```

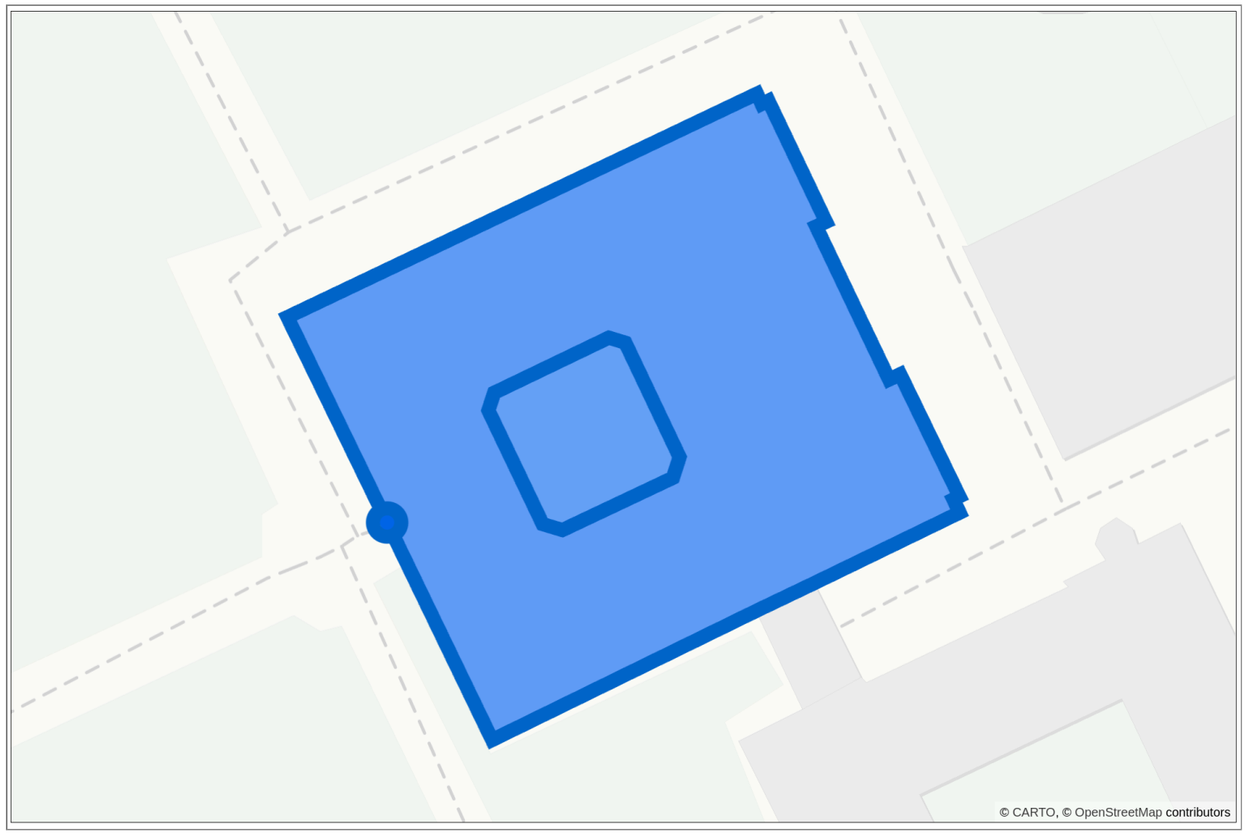


outline

- ✓ About Raku
- ✓ Geospatial Data with Nominatim and Overpass
- Geospatial Visualization with Leaflet and Deck.gl
- Geospatial Analysis with DuckDB and GEOS
- Conclusion

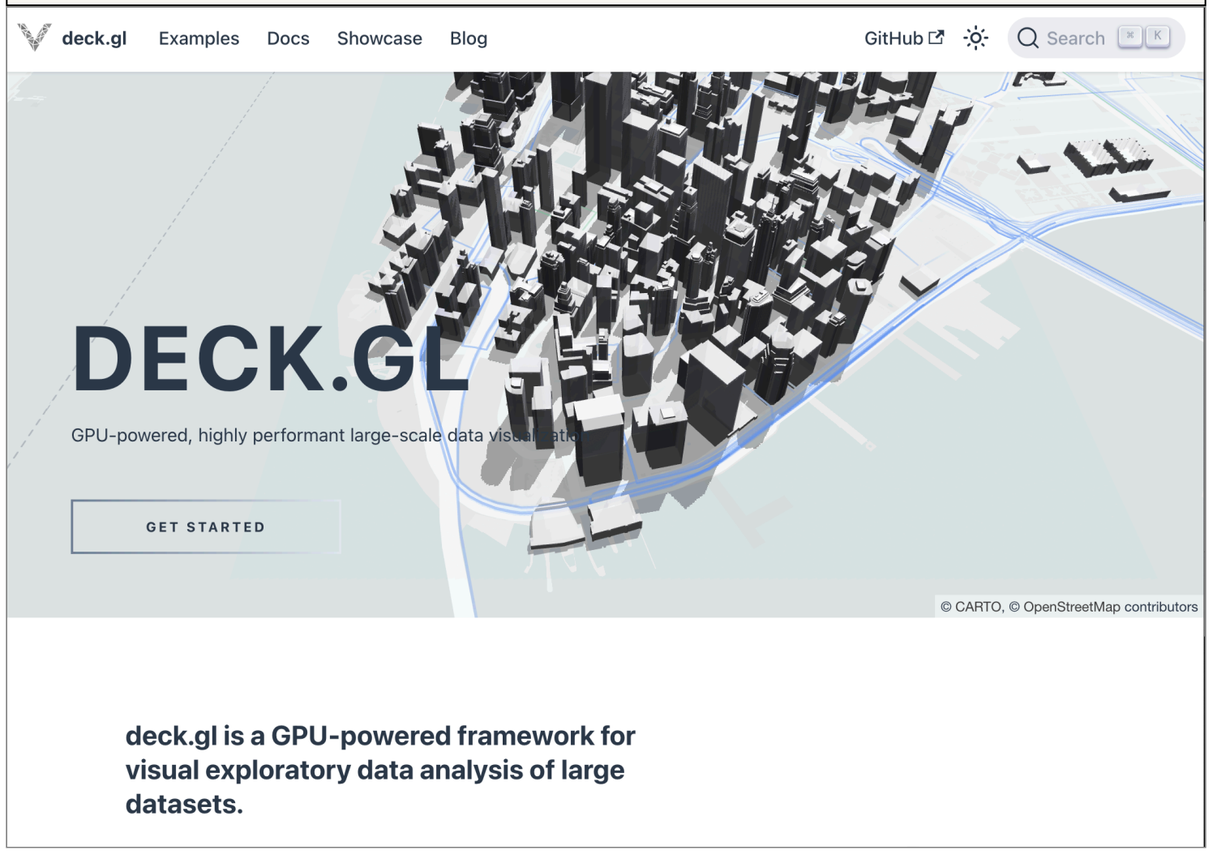
deck.gl

```
use Map::DeckGL 'deck';
deck.add-geojson: $geojson;
deck.show;
```



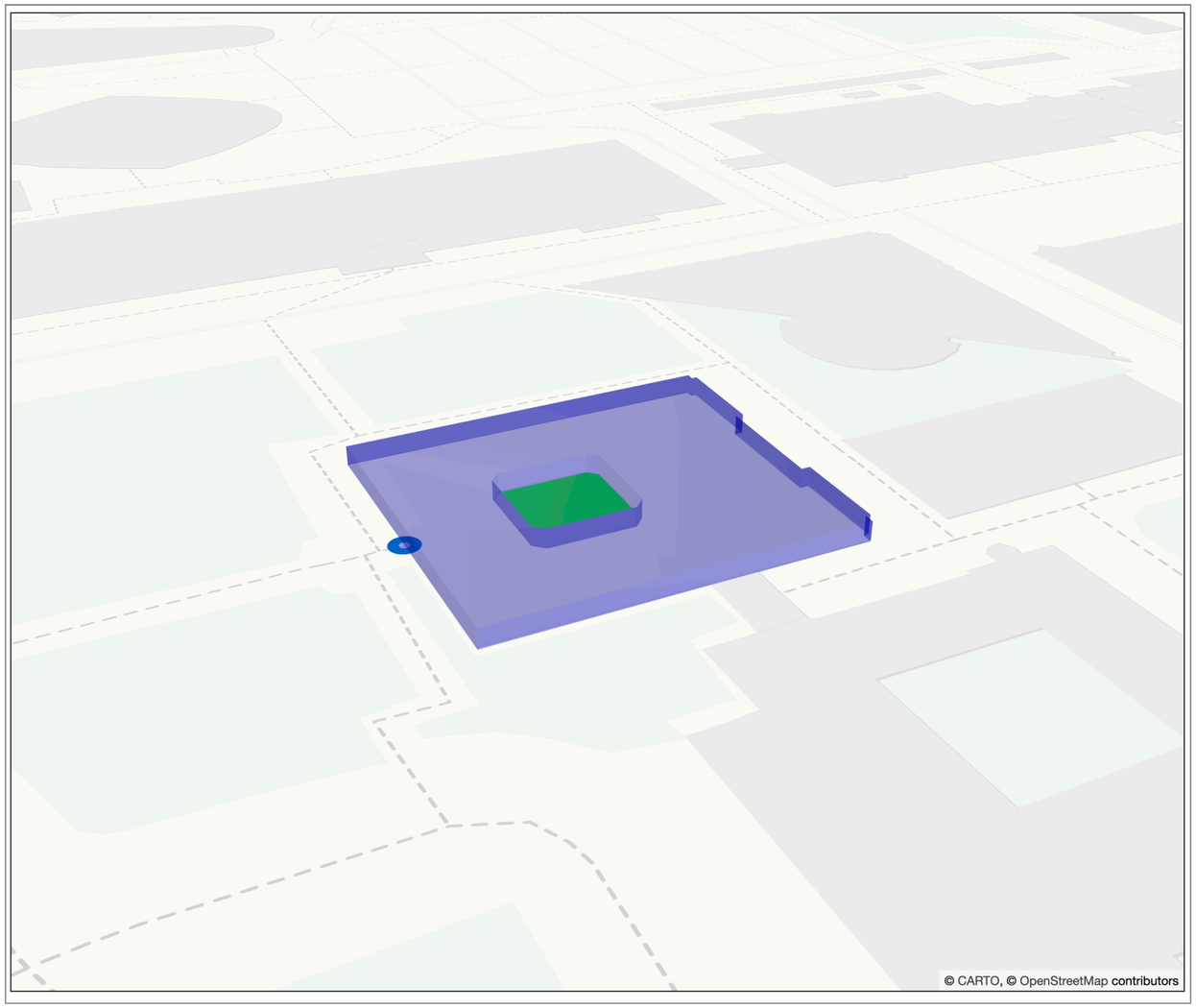
ULB Building AW (OSM relation 1316770)

https://deck.gl/



DeckGL fancier version

```
use Map::DeckGL 'deck';

deck.initialViewState<pitch> = 70;
deck.initialViewState<zoom> = 19;

deck.add-geojson: (slurp "building.geojson"),
  :extruded,
  :getElevation(
      f => 'f.properties["building:levels"] * 2'
  ),
  :getFillColor(
      f => q:to/JS/
        f.properties["leisure"] == "garden"
        ? [0, 200, 100] : [0, 0, 200, 100]
      JS
  );

deck.show;
```

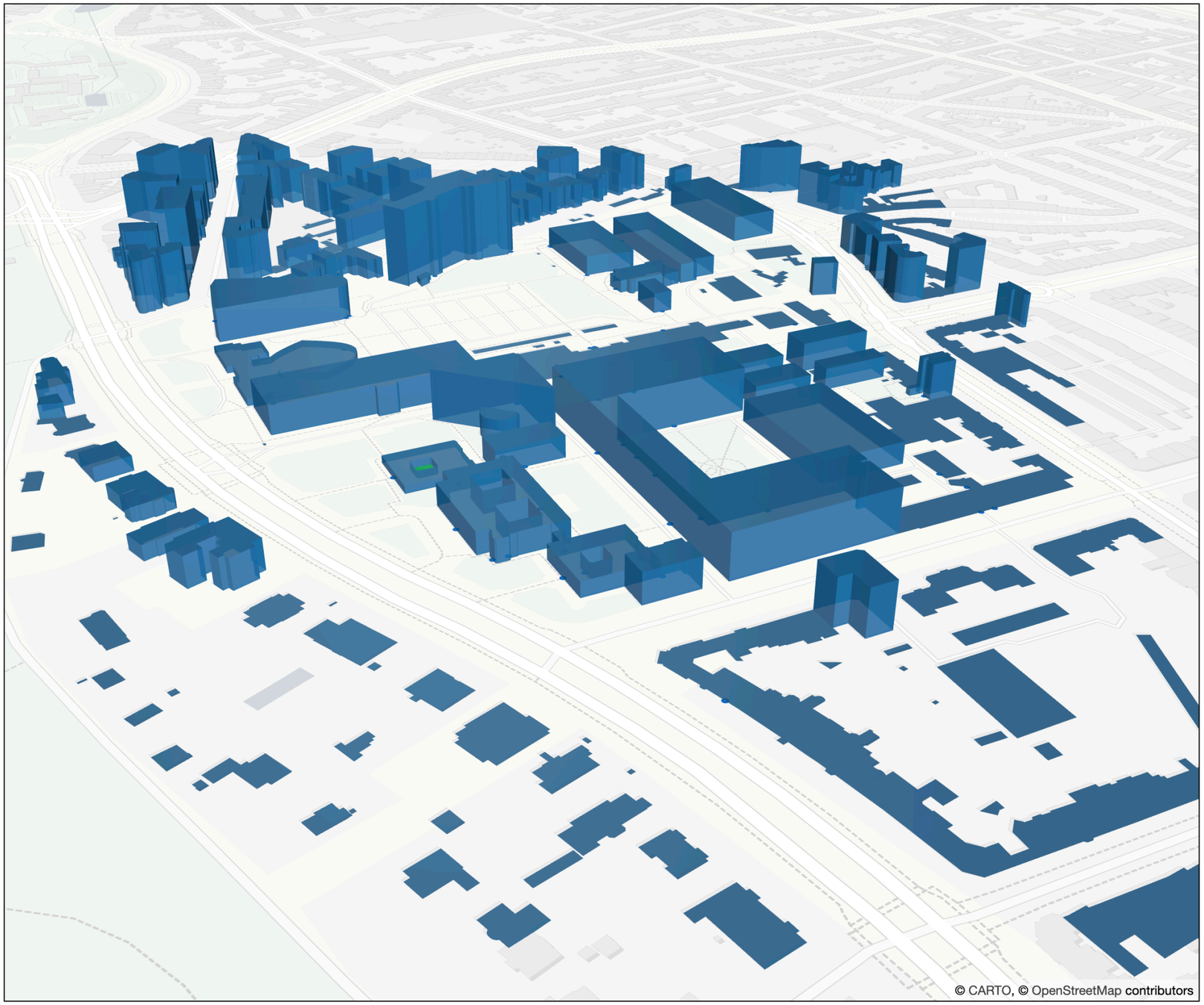



```
(geojson)
"properties": {
  "building:levels": "1",
 ...
   "leisure" : "garden"
```

Like Map::Leaflet, we also imitate the Deck class relationships.

This works fine on larger geojson files too.

Fancier version for a larger area



Deck.gl calculates heights and colors in javascript; this is a single geojson layer.

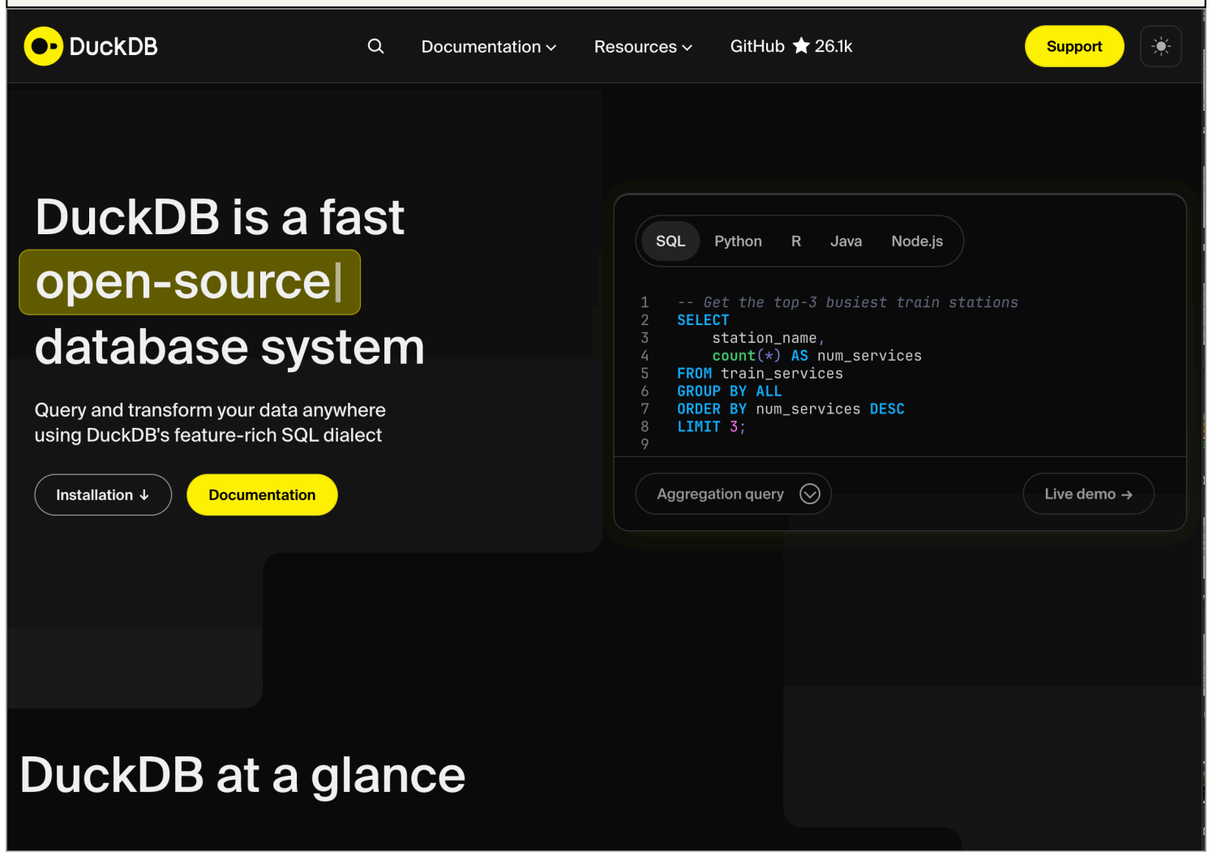
outline

- ✓ About Raku
- ✓ Geospatial Data with Nominatim and Overpass
- ✓ Geospatial Visualization with Leaflet and Deck.gl
- Geospatial Analysis with DuckDB and GEOS
- Conclusion

duckdb

```
use Duckie 'd';
my $res = d.query: q:to/SQL/
-- Get the top-3 busiest train stations
SELECT
    station_name,
        count(*) AS num_services
        FROM train_services
        GROUP BY ALL
        ORDER BY num_services DESC
        LIMIT 3;
SQL
say $res.rows;
```

https://duckdb.org/



duckdb

Find the areas of the top 10 buildings at ULB

```
use Duckie 'd';
my $res = d.query: q:to/SQL/;
load spatial;

SELECT
  (
    UNNEST(features)->'properties'
  )->'name' as name,
    ST_AREA_SPHEROID(
      ST_GEOMFROMGEOJSON(
          UNNEST(features)->'geometry'
      )
  ) as area_m2
FROM READ_JSON('campus.geojson')
ORDER by 2 desc
LIMIT 10
SQL

say "Building {.<name>}: {.<area_m2>.fmt('%.1f')} m²"
  for $res.rows;
```

```
Building "U": 13854.3 m²
Building "F1": 6462.2 m²
Building "H": 6136.5 m²
Building "L": 5180.3 m²
Building "J": 4888.1 m²
Building "C": 3821.8 m²
Building "R": 3723.4 m²
Building "E": 3681.3 m²
Building "AX": 3225.3 m²
Building "K": 2978.1 m²
```

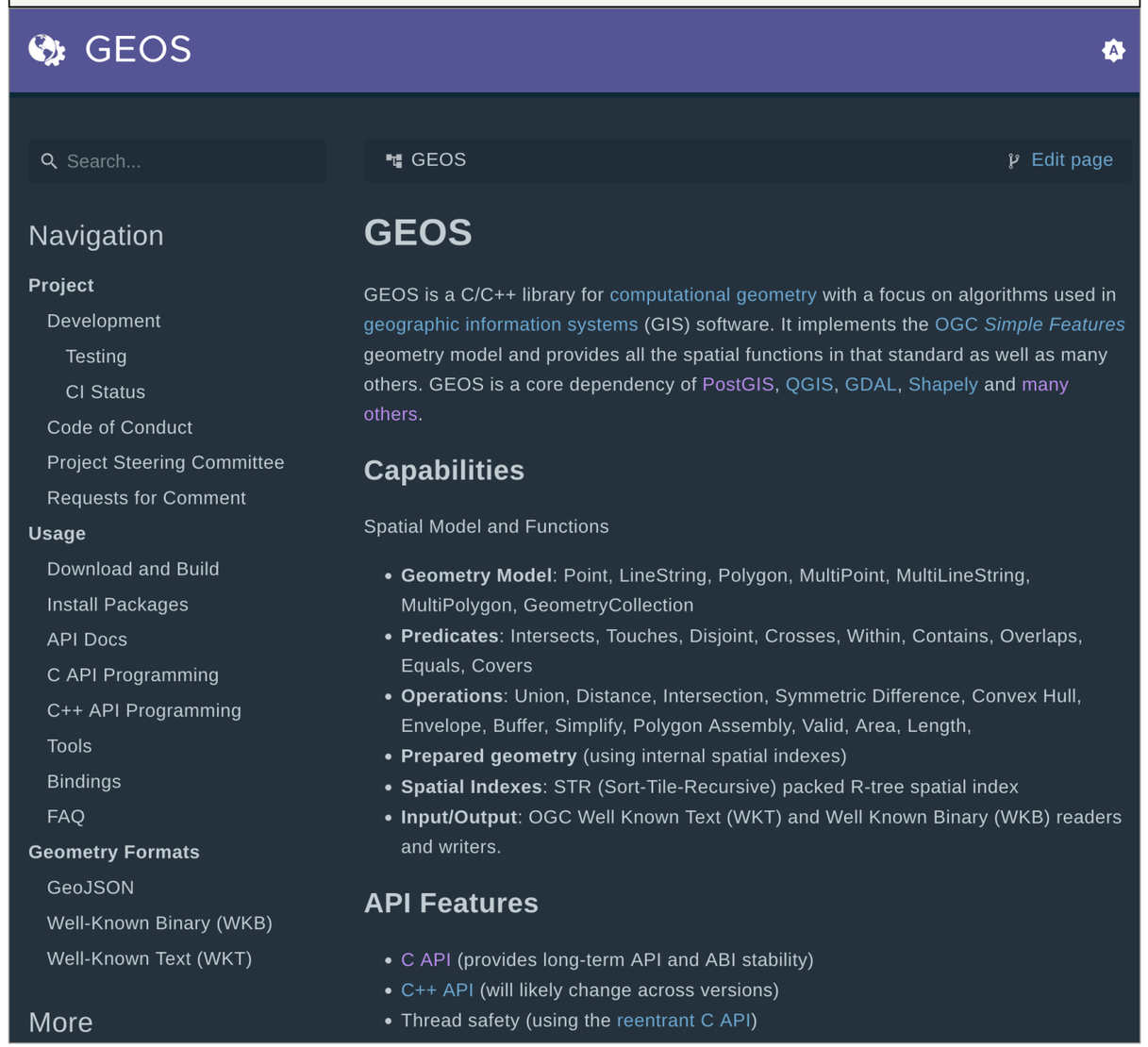
GEOS

```
use GEOS::Geometry;

my $point = GEOS::Geometry.from-wkt: 'POINT(4.367611 50.822694
say $point.geometry-type;
say $point.x;
```

```
Point
4.367611
```

GEOS uses Raku's NativeCall interace to call the C API.

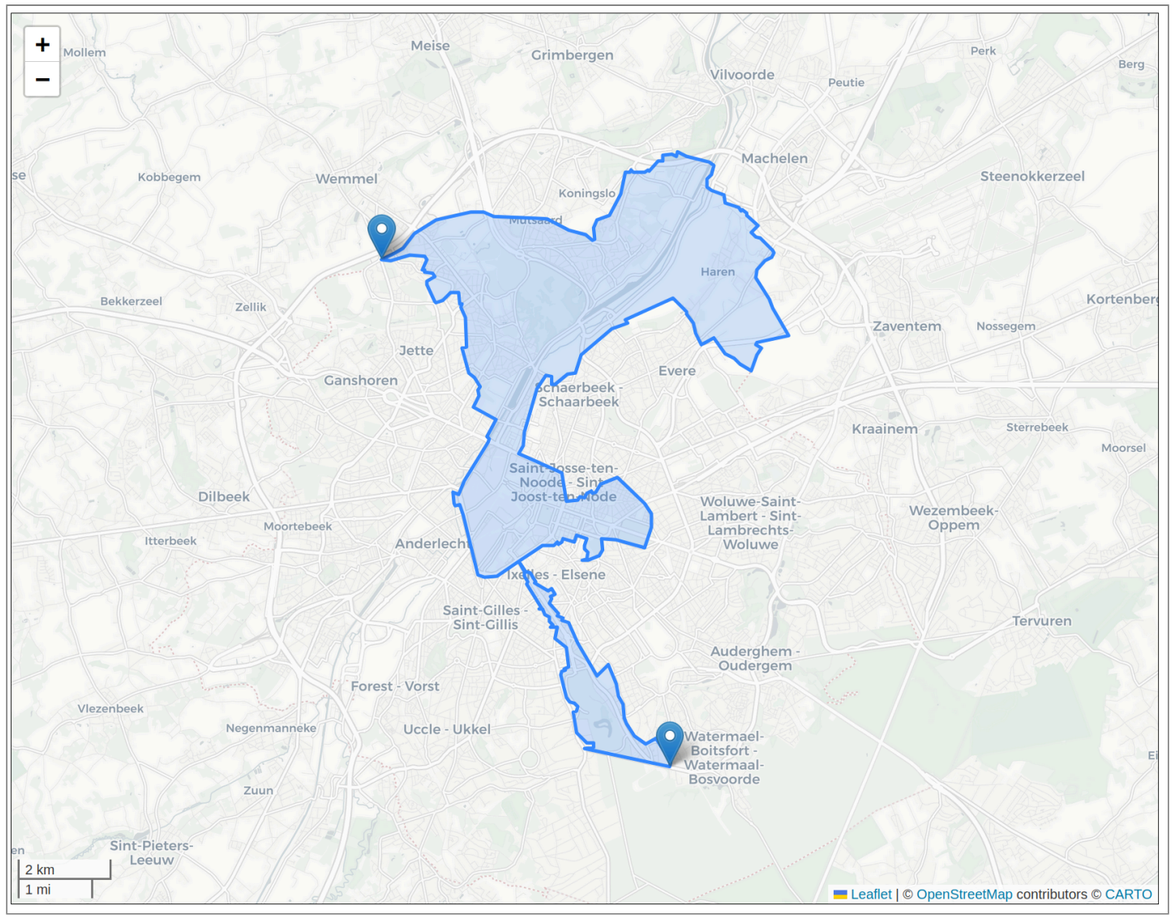
https://libgeos.org/



Thread-safe operations with GEOS

Find the two points farthest way from each other on the city border.

```
my ($res) = nom.search: "Brussels", :polygon_text;
my $geom = GEOS::Geometry.from-wkt($res<geotext>);
my @points = $geom.get-coordinates
  .map: { GEOS::Geometry.from-wkt("POINT($_[0,1])") };

my @node-pairs = @points.combinations(2);
my @distances = @node-pairs.hyper.map: -> ($a,$b) { $a.distance-to($b) };

my $max = @distances.max;
my $longest = @distances.first(:k, * eqv $max);
my $pair = @node-pairs[ $longest ];
```



Compute all distances between node pairs

```
@node-pairs.map: -> ($a,$b) { $a.distance-to($b) };
```

Compute all distances between node pairs in parallel

```
@node-pairs.hyper.map: -> ($a,$b) { $a.distance-to($b) };
```

- Use `hyper` to parallelize operations on a sequence.

conclusion

We have seen

- Some useful modules
- Gradual typing
- Hash and List slices
- Class Hierarchies
- Javascript generation
- NativeCall for C libraries
- `hyper` for parallelization

Other things we didn't see

- Many ways to run external processes!
- Many numeric types: rational, float, and native float
- Much more concurrency and asynchrony support
- Grammars

```
Thank You!
```

Website https://raku.org

Ecosystem https://raku.land

Installation https://rakubrew.org

Visit us at the Perl and Raku Stand!