
PostGIS is your new bicycle
Come and be wowed by the power of free alternatives to costly desktop GIS

PostGIS is:

- an extension that enables a PostgreSQL database to store, query and transform geographic data, like points, lines and polygons.
- free, as in "free beer" and as in "free speech."
- compatible with free desktop software like Quantum GIS and uDig.
- integrated with the Python programming language and its Django web framework.
- your new bicycle.

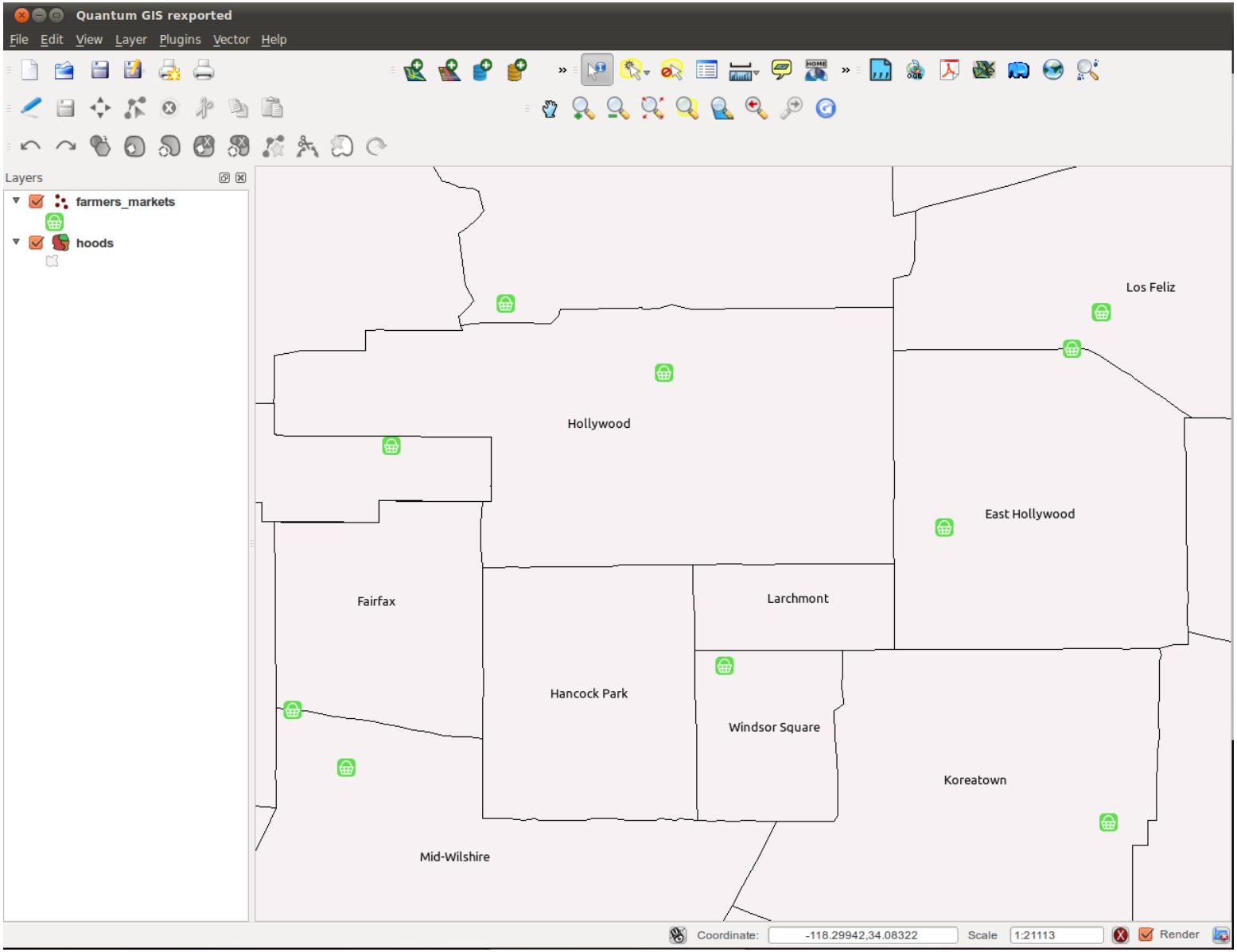
In other words, SQL like this...

```
db=# SELECT id, name FROM farmers_markets;

 id | name
----+----------------------------
  1 | Alhambra
  4 | Huntington Park
 16 | La Verne
 24 | Chinatown
 25 | Little Tokyo/Arts District
 27 | Atwater Village
 30 | Studio City
 34 | Glendale
 40 | Bellflower
 45 | Lawndale
 ...
```

...can now do this...

```
db=# SELECT id, ST_AsText(point), name FROM farmers_markets;

 id | point                                | name
----+--------------------------------------+------------------
  1 | POINT(-118.12355 34.094109)          | Alhambra
  4 | POINT(-118.205872 33.972744)         | Huntington Park
 16 | POINT(-117.769776 34.101508)         | La Verne
 24 | POINT(-118.239494 34.064465)         | Chinatown
 25 | POINT(-118.243767 34.052551)         | Little Tokyo/Arts
 27 | POINT(-118.260452 34.118412)         | Atwater Village
 30 | POINT(-118.39347 34.143716)          | Studio City
 34 | POINT(-118.25473 34.147186)          | Glendale
 40 | POINT(-118.1304073 33.88270792)      | Bellflower
 45 | POINT(-118.352387 33.89829352)       | Lawndale
```

...and this...

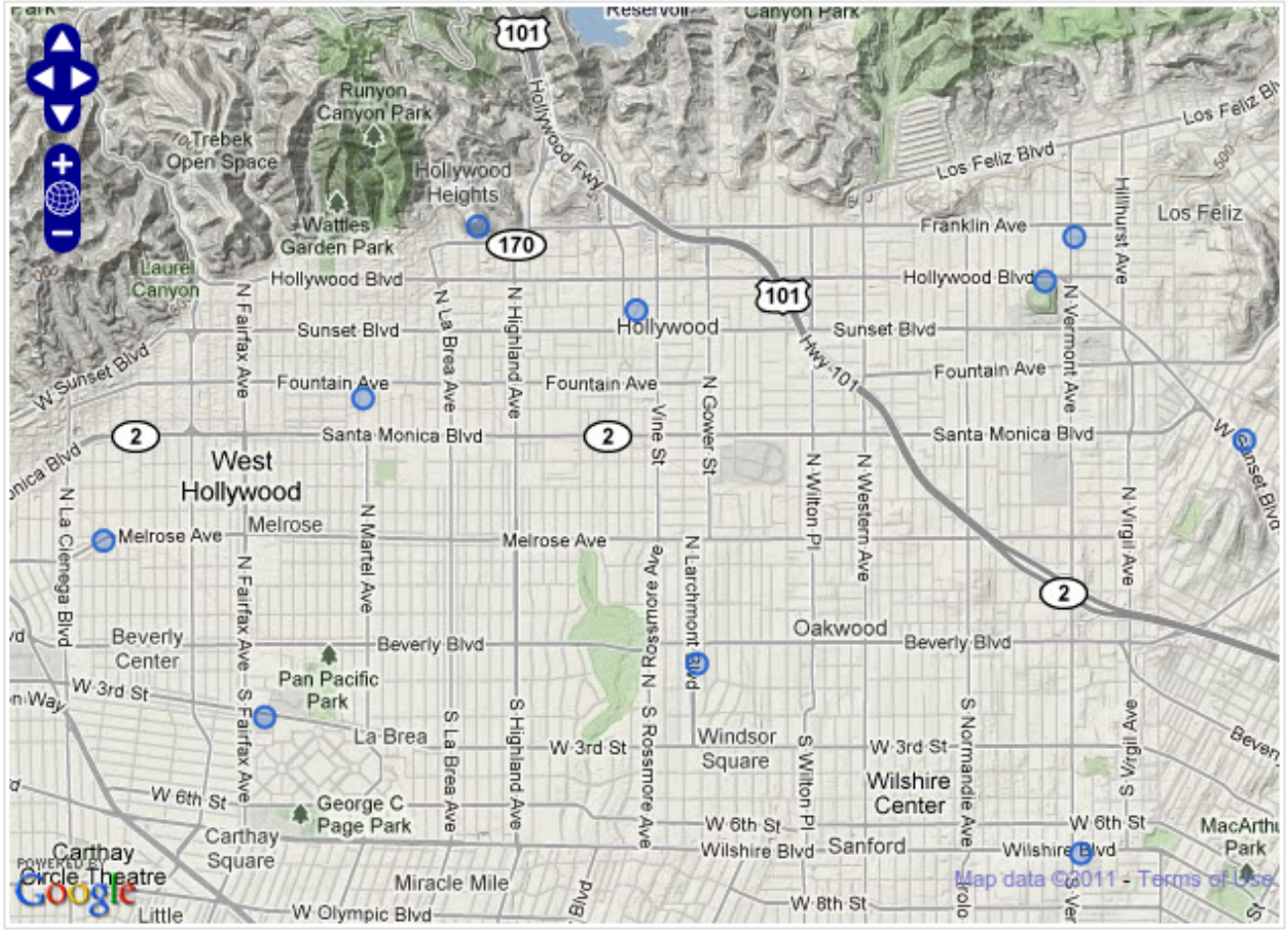

Quantum GIS rexported
File Edit View Layer Plugins Vector Help
Layers
farmers_markets
hoods
Los Feliz
Hollywood
East Hollywood
Fairfax
Larchmont
Hancock Park
Windsor Square
Koreatown
Mid-Wilshire
Coordinate:
-118.29942,34.08322
Scale
1:21113
Render

...and this:

SOUTHERN CALIFORNIA

Farmers Markets

Explore your local farmers markets



All the old SQL still works

Use standard clauses like WHERE to filter results

```
db=# SELECT ST_AsText(point), name
     FROM farmers_markets
     WHERE occurence LIKE 'All Year';
```

```
 point                                          | name
------------------------------------------------+----------------
 POINT(-117.565130983637 33.8856669958338) | Corona
 POINT(-118.589684 34.17261)               | Woodland Hills
 POINT(-117.730286104982 34.0021365631974) | Chino Hills
 POINT(-117.435608 34.07401)               | Fontana
 POINT(-118.777999861395 34.2728219414777) | Simi Valley
 POINT(-118.395989283517 33.905558395832)  | El Segundo
 POINT(-118.756424172081 34.1569708326658) | Agoura Hills
 POINT(-118.290014983537 34.0115329958227) | Exposition Park
```

Loading data is a snap

Do you prefer command line? Great!

```
#Loading a shapefile with shp2pgsql command line loader:

$ shp2pgsql -D -s 4269 hoods/hoods.shp public.
neighborhoods_test > neighborhoods_test.sql

$ psql -d seismic_gis -f neighborhoods_test.sql
```

Would you rather use a GUI? Great!

```
#The QuantumGIS SPIT plugin makes shapefile loading really
easy.
```

Reprojections

PostGIS's magic SQL can change a geometry's projections on the fly

```
# Mercator (900913) transformed to NAD83 (4269)
db=# SELECT ST_Transform(
       ST_SetSRID(the_geom,900913),
       4269
     )

# TIGER/NAD83 transformed to UTM Zone 11N
db=# SELECT ST_Transform(
       ST_SetSRID(the_geom,4269),
       32611
     )
```

** As long as you import correctly, the ST_SetSRID function isn't necessary -- shown here for emphasis.

Spatial joins

Find all the points that fall within a polygon from another table

```
db=# SELECT farmers_markets.the_geom,
            farmers_markets.name as market_name,
            neighborhoods.name as hood_name
     FROM farmers_markets, neighborhoods
     WHERE ST_Contains(
             ST_Transform(neighborhoods.the_geom,4326),
             farmers_markets.the_geom
           )=True
     AND neighborhoods.name LIKE 'Long Beach';
```

Geometric unions

Merge multiple polygons together into one big shape

```
db=# SELECT
        ST_Union(the_geom),
        city_id
     FROM neighborhoods
     WHERE city_id = 264
     GROUP BY city_id;
```

Simplify geometries

You don't always need all that detail. PostGIS makes it easy to keep your data in its native resolution, but use only what you need.

```
# Simple
db=# SELECT ST_Simplify(the_geom, 0.005), state
     FROM state_borders
     WHERE state LIKE 'California';

# Simpler
db=# SELECT ST_Simplify(the_geom, 0.05), state
     FROM state_borders
     WHERE state LIKE 'California';

# Probably oversimplified
db=# SELECT ST_Simplify(the_geom, 0.5), state
     FROM state_borders
     WHERE state LIKE 'California';
```

Buffers

Query all the records within some distance of a point

```
db=# SELECT name, the_geom
     FROM farmers_markets
     WHERE ST_Within(
      ST_Transform(the_geom,32611), # UTM Zone 11N
      ST_Buffer(
        ST_Transform(
            ST_GeomFromText(
                'POINT(-118.255033 34.051166)',
                4326),
            32611),
        4023  # 2.5 mile radius (in meters)
      )
     ) = True
```

Geometric calculations

Figure out the area of a polygon

```
db=# SELECT ST_Area(
                ST_Transform(
                    ST_Union(the_geom),
                    32611
                )
            )
        FROM neighborhoods
        WHERE city_id = 264
        GROUP BY city_id;
```

* Yes, PostGIS automatically handles donuts when doing areas
** Also, be sure to convert to something like UTM to get meters

And...you can do it all in Python

GeoDjango provides easy wrappers on most PostGIS operations

```
# Query a record from the database
>> from mapping.neighborhoods import Neighborhood
>> dtla = Neighborhood.objects.get(name='Downtown')

# Mash it against a point table and return hits
>> from farmers_markets.models import FarmersMarket
>> obj_list = FarmersMarkets.objects.filter(
     point__intersects=dtla.polygon
   )

# Reproject a polygon
>> dtla.polygon.transform(900913)
```