What is coming in PostGIS 3.1

PARAGON CORPORATION

[(https://www.paragoncorporation.com)](https://www.paragoncorporation.com)

Regina Obe

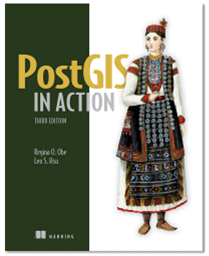
PostGIS

[(https://postgis.net)](https://postgis.net)

PostGIS In Action 3rd Edition available for Purchase

- Get a copy of 2nd edition with purchase of 3rd edition
- Covers PostGIS 3 and 3.1
- https://www.manning.com/books/postgis-in-action-third-edition (https://www.manning.com/books/postgis-in-action-third-edition)
- 16 of 17 chapters completed.
- https://www.postgis.us (https://www.postgis.us) find code and data here

- Live Book https://livebook.manning.com/book/postgis-in-action-third-edition (https://livebook.manning.com/book/postgis-in-action-third-edition)



Safe Harbor Statement

Some of the following mentions are forward looking statements and intended to outline the direction of PostGIS development.
They are not a commitment to deliver any code or functionality and should not be relied upon in making life altering decisions. The development, release, and timing of any features or functionality described is subject to change without warning.

What is coming in PostGIS 3.1?

Manual: https://postgis.net/docs/manual-dev/ (https://postgis.net/docs/manual-dev/)

Watch out for 3.1.0alpha3 coming this week.

Where to get it

- PostGIS Docker - PG13 - postgis-master https://github.com/postgis/docker-postgis/tree/master/13-master (https://github.com/postgis/docker-postgis/tree/master/13-master)
- Debian / Ubuntu - https://apt.postgresql.org (https://apt.postgresql.org) (currently at 3.1.0alpha2)
- Yum (CentOS, Redhat EL, Scientific Linux): https://yum.postgresql.org (https://yum.postgresql.org) (has 3.1.0alpha2)
- Windows: https://postgis.net/windows_downloads/ (https://postgis.net/windows_downloads/) (has fresh updates for PostgreSQL 11,12,13 with GEOS 3.9, GDAL 3.2, Proj 7.1.1)(Experimental Builds section - builds on each commit)

New Functions and overloads - Non-GEOS related

- Gridding functions - ST_HexagonGrid , ST_SquareGrid,(in 3.1.0alpha1)
- ST_MaximumInscribedCircle (in 3.1.0alpa2)
- ST_TileEnvelope - optional margin parameter
- ST_Force3* functions can now take a measure. (in 3.1.0alpha2)

Enhancements - Non-GEOS related

- Cast geojson to geometry - for implicit geojson ingestion
- ST_ClusterKMeans now works with 3D geometries
- postgis_topology: topology.GetRingEdges now implemented in C (should be much faster)
- Avoid de-Toasting -- means many spatial joins particularly on big geometries will be much faster
- postgis_topology: ST_GetFaceGeometry provides more info about corruption
- ST_Simplify - speed improvements
- Wagyu now at 0.5.0 should be faster MVT processing
- More tweaking of function costs to better take advantage of parallelism (3.1.0alpha2)
- Too many things much faster to talk about but I didn't benchmark. I'm just taking other people's word for it.
- Textual output functions like ST_AsText, ST_AsGeoJSON etc, 5-100x faster

Only for GEOS >= 3.9 users, rest of you are out of luck

- Several functions can now work in fixed precision with extra arg gridsize: `ST_Difference`, `ST_Intersection`, `ST_Subdivide`, `ST_SymDifference`, `ST_UnaryUnion`, `ST_Union`
- Improved robustness of `ST_Intersection`, `ST_Union` and other functions if running GEOS 3.9.0. Should result in fewer `Topology Exception` errors when doing ST_Intersection and ST_Union. Using a magic called OverlayNG introduced in 3.9 (boring name but I was not consulted)

From PostGIS 3+ on minor version dropped from Lib

For developers who need to test both versions in same PostgreSQL version build like so. Will make the libs end in postgis-3.1.{ext} instead of postgis-3.{ext}

```
./configure --with-library-minor-version
```

How you install extensions in PostGIS 3+

This is running in psql. If in pgAdmin just manually reconnect to your gisdb. Example is gisdb but do for any spatial databases you have.

```
CREATE DATABASE gisdb;
ALTER DATABASE gisdb SET search_path=public,postgis,tiger,contrib;
\c gisdb
CREATE SCHEMA IF NOT EXISTS postgis;
CREATE SCHEMA IF NOT EXISTS contrib;
CREATE EXTENSION postgis SCHEMA postgis;
CREATE EXTENSION postgis_raster SCHEMA postgis; -- used to be part of postgis extension
CREATE EXTENSION postgis_sfcgal SCHEMA postgis; -- much of the functionality like dealing
with 2d TINS and Polyhedralsurfaces now supported in PostGIS proper
CREATE EXTENSION postgis_topology;
CREATE EXTENSION fuzzystrmatch SCHEMA contrib;
CREATE EXTENSION address_standardizer SCHEMA contrib; #for standardizing addresses
CREATE EXTENSION postgis_tiger_geocoder; -- needs postgis and fuzzystrmatch, can use addr
ess_standardizer
```

How you upgrade from PostGIS 3.0 to 3.1

```
SELECT postgis_extensions_upgrade();
```

How you upgrade from PostGIS 2.* to 3.1

1. First install the binaries - different for each platform
2. Connect to your database
3. Run these sql commands

```
ALTER EXTENSION postgis UPDATE; -- if running pre PostGIS 2.5

SELECT postgis_extensions_upgrade(); -- this unpackages raster (if already on 3.0, no nee
d to run twice)

SELECT postgis_extensions_upgrade(); -- do again to repackage raster
```

Check your installation

```sql
SELECT postgis_full_version();
```

```
screen
POSTGIS="3.1.0dev 3.1.0alpha2-159-g9c6431748" [EXTENSION]
PGSQL="130" GEOS="3.9.0-CAPI-1.14.0" SFCGAL="1.3.8"
PROJ="7.1.1"
GDAL="GDAL 3.2.0, released 2020/10/26"
LIBXML="2.9.9" LIBJSON="0.12" LIBPROTOBUF="1.2.1" WAGYU="0.5.0 (Internal)" TOPOL
OGY RASTER
```

Plumbing change in 3.1, but mostly one of packaging, no change in install process

- Drop sfcgal (cgal binding) from postgis-3.so and spin-off as postgis_sfcgal-3.{ext}

```
markdown
In 3.0 (before):
  EXTENSION         LIB ext=so,dylib,dll,whatever
  postgis        -> postgis.{ext}
  postgis_raster -> postgis_raster-3.{ext}
  postgis_sfcgal -> postgis-3.{ext}
  postgis_topology -> postgis_topology-3.{ext}
  postgis_tiger_geocoder -> none
  address_standardizer -> address_standardizer-3.{ext}

In 3.1 (after):

  EXTENSION         LIB  ext=so,dylib,dll,whatever
  postgis        -> postgis.{ext}
  postgis_raster -> postgis_raster-3.{ext}
  postgis_sfcgal -> postgis_sfcgal-3.{ext}
  postgis_topology -> postgis_topology-3.{ext}
  postgis_tiger_geocoder -> none
  address_standardizer -> address_standardizer-3.{ext}
```

Requirements changes

- PostgreSQL 9.6+ (PostGIS 3.0 had minimum 9.5)
- Bump minimum protobuf-c requirement to 1.1.0 to enable MVT (too many complaints with lower versions)
- Proj 5.0+, PostGIS 3.0 required Proj 4.9 or higher

PostGIS 3.1 ST_Union Fixed Precision

```sql
CREATE TABLE union_counties(name text, geom geometry);
-- 2 min 37 secs
INSERT INTO union_counties(name, geom)
SELECT 'original', ST_Union(the_geom)
FROM county;

-- 22 secs 983 ms - 0.005 degree fixed precision
INSERT INTO union_counties(name, geom)
SELECT '0.005', ST_Union(the_geom, 0.005)
FROM county;

-- 1 min - 0.001 degree fixed precision
INSERT INTO union_counties(name, geom)
SELECT '0.001', ST_Union(the_geom, 0.001)
FROM county;
```

```
SELECT name, ST_NPoints(geom), ST_IsValid(geom)
FROM union_counties;
```

```
screen
   name    | st_npoints | st_isvalid
-----------+------------+------------
 original  |     106587 | t
 0.005     |      44554 | t
 0.001     |      74666 | t
 0.01      |      32497 | t
(4 rows)
```

PostGIS 3.1 ST_Subdivide Fixed Precision

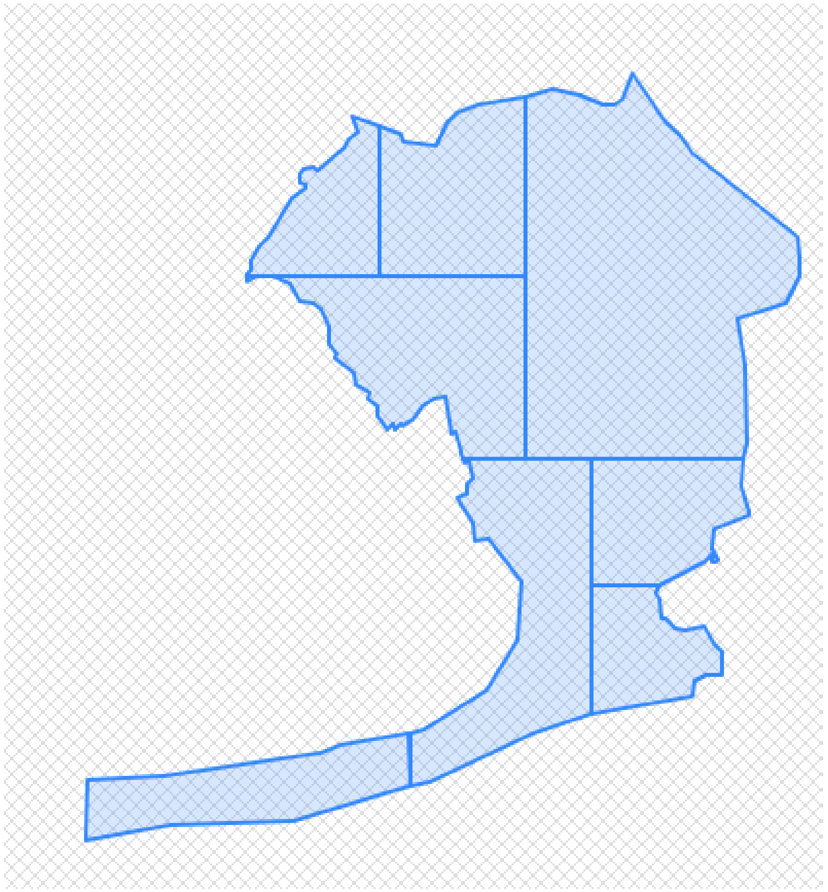
No Precision - 9 pieces - Before

```
SELECT sb.ord, sb.geom
FROM (SELECT geom,
        ST_NPoints(geom) AS npoints
      FROM ch11.boroughs WHERE boroname = 'Queens') AS ref
      , ST_SubDivide(ref.geom,200) WITH ORDINALITY AS sb(geom,ord);
```

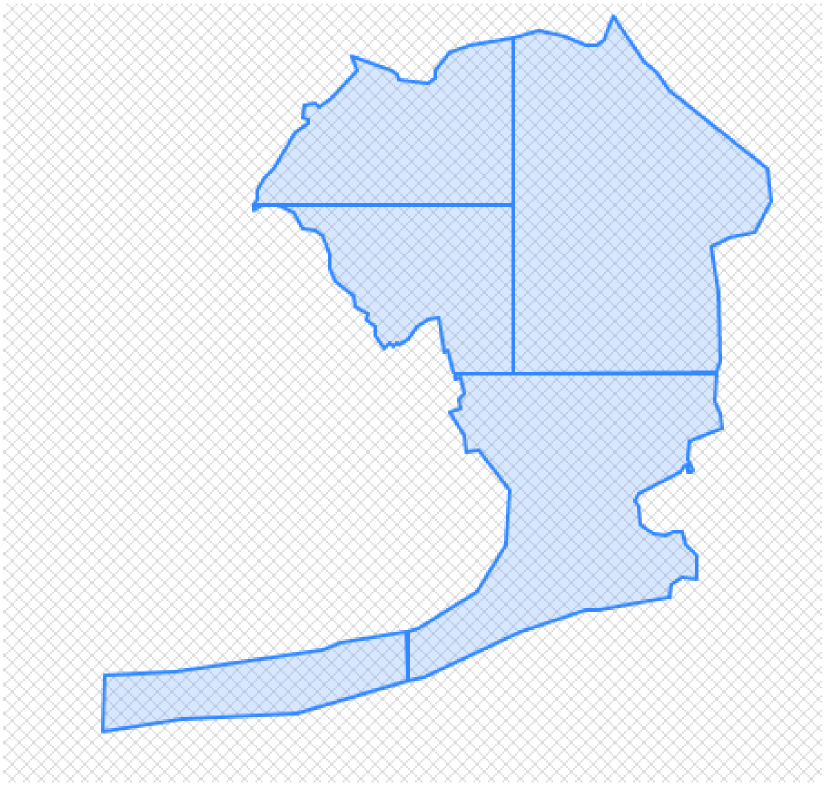
Fixed Precision - 50 ft gridsize (srid is 2263 NY State Plane ft), 5 pieces

```
SELECT sb.ord, sb.geom
FROM (SELECT geom,
        ST_NPoints(geom) AS npoints
      FROM ch11.boroughs WHERE boroname = 'Queens') AS ref
      , ST_SubDivide(ref.geom,200,50) WITH ORDINALITY AS sb(geom,ord);
```

Before (no precision)



After (50 ft grid)

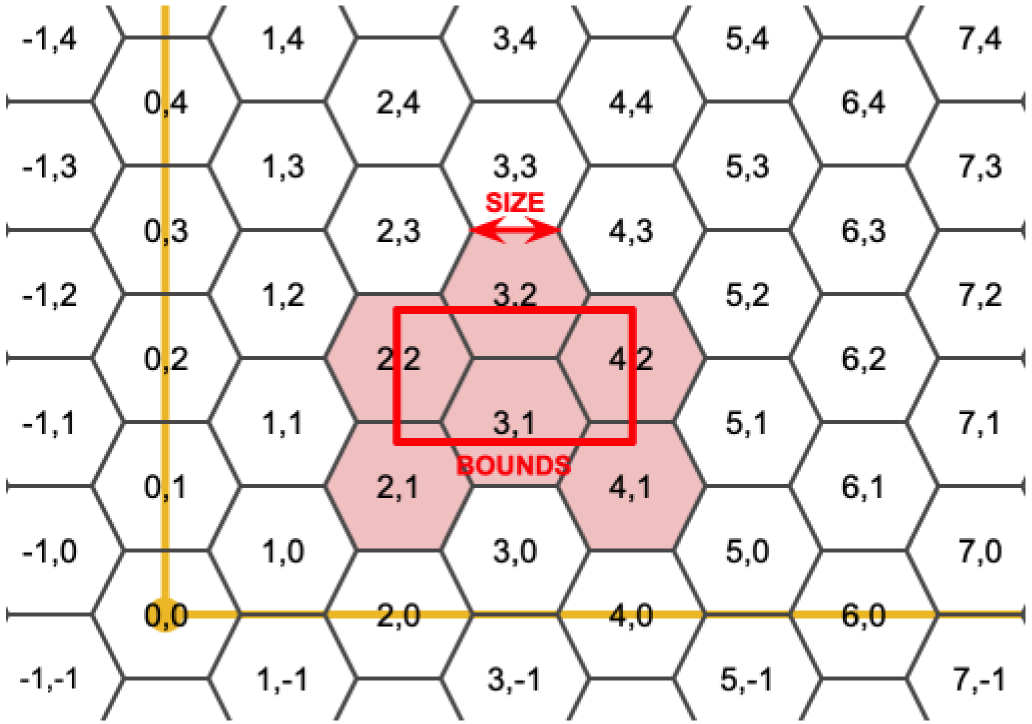


PostGIS 3.1 functions ST_HexagonGrid

https://postgis.net/docs/manual-dev/ST_HexagonGrid.html
(https://postgis.net/docs/manual-dev/ST_HexagonGrid.html)

1. ST_HexagonGrid creates a grid of the bounding box of geometry passed to it
2. Need ST_Intersects to filter out the hexagons that don't intersect the geometry

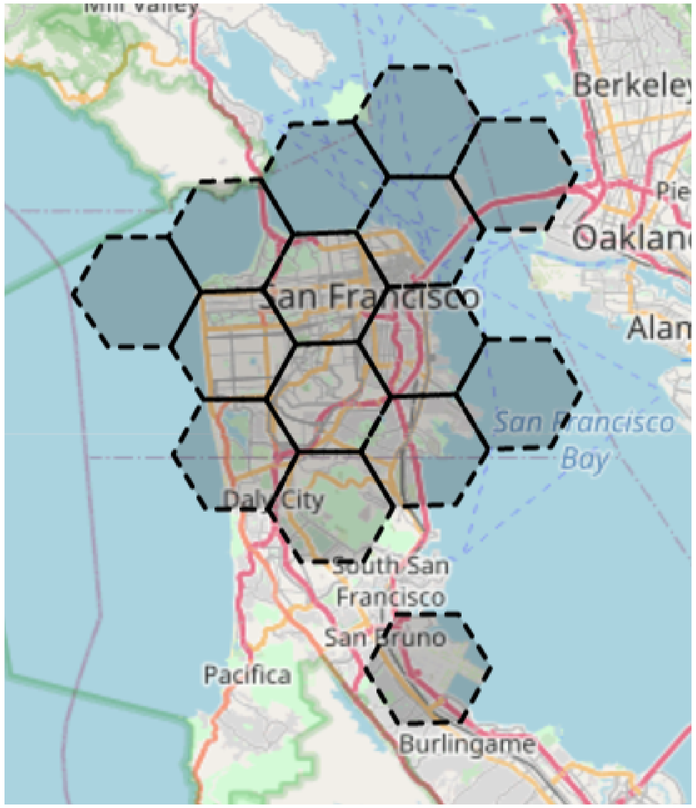
3. The size is length of an edge on the hexagon



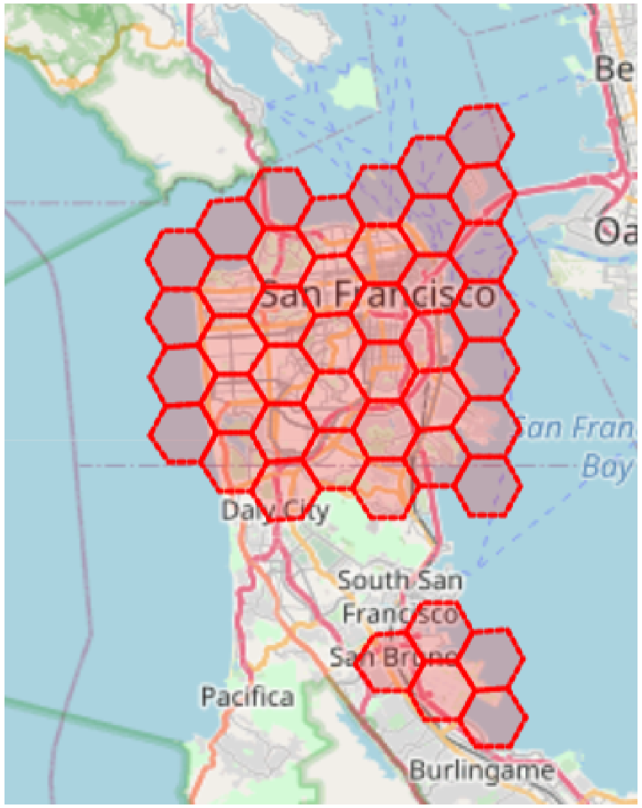
In this case we are using Northern CA State Plane feet.

```
SELECT grid.i, grid.j, ST_Union(grid.geom) AS geom
FROM ch11.cities AS c
    INNER JOIN ST_HexagonGrid(10000, c.geom) AS grid ON ST_Intersects(c.geom, grid.geom)
    WHERE c.city = 'SAN FRANCISCO'
GROUP BY grid.i, grid.j, grid.geom;
```

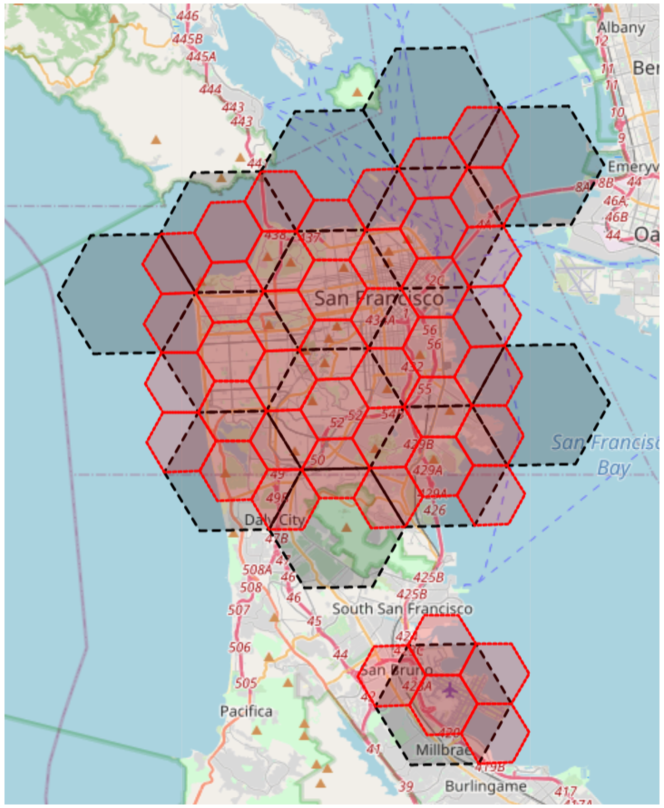
10,000 feet edge size



5,000 feet edge size



Hexagons cut at the middle, not able to nest a hexagon completely in another



PostGIS 3.1 functions ST_SquareGrid

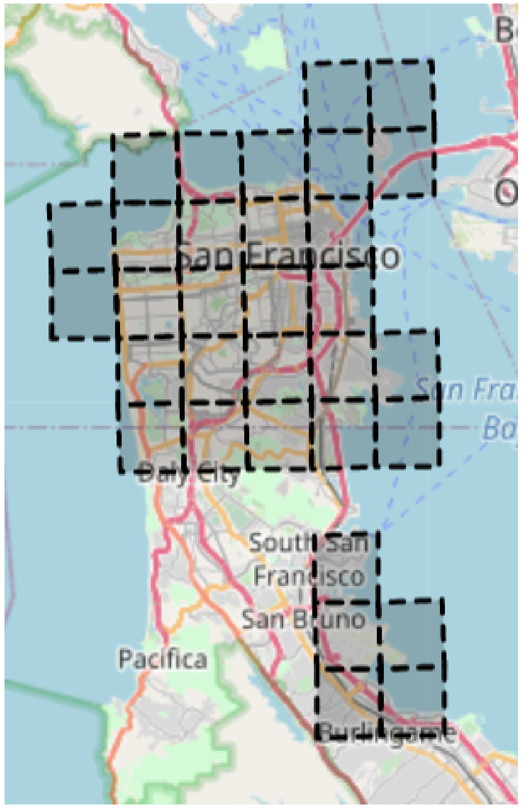
https://postgis.net/docs/manual-dev/ST_SquareGrid.html
(https://postgis.net/docs/manual-dev/ST_SquareGrid.html)

1. ST_SquareGrid creates a grid of the bounding box of geometry passed to it
2. Need ST_Intersects to filter out the squares that don't intersect the geometry
3. The size: 10,000 is length of an edge in measure of the units of the spatial reference system, in our case it would be California State Plane feet

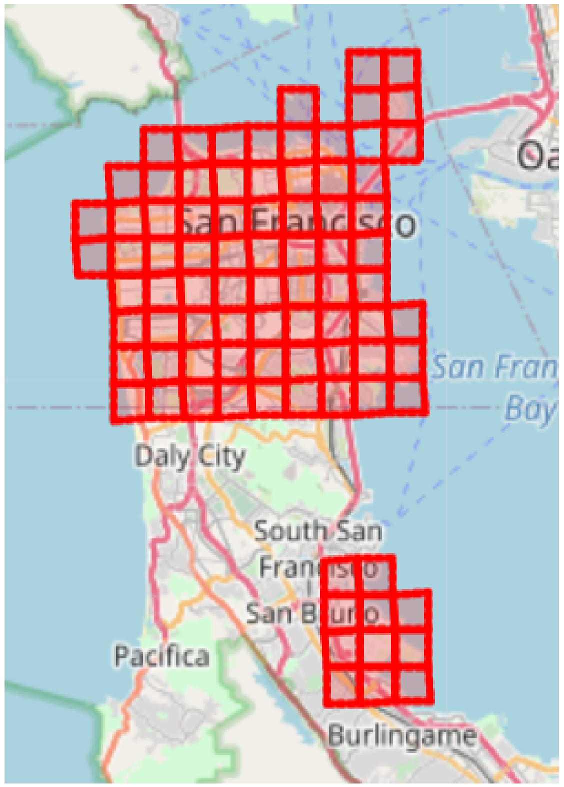
PostGIS 3.1 ST_SquareGrid San Francisco

```
SELECT grid.i, grid.j, ST_Union(grid.geom) AS geom
FROM ch11.cities AS c
    INNER JOIN ST_SquareGrid(10000, c.geom) AS grid ON ST_Intersects(c.geom, grid.geom)
    WHERE c.city = 'SAN FRANCISCO'
GROUP BY grid.i, grid.j, grid.geom;
```

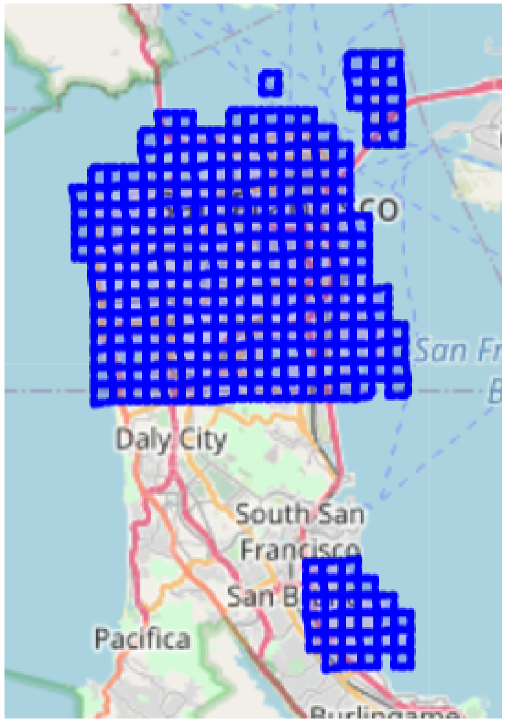
10,000 feet edge size



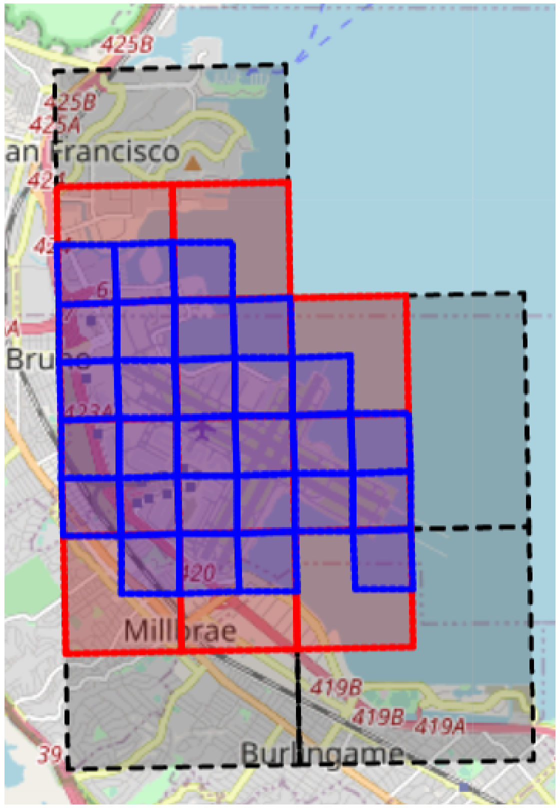
5,000 feet edge size



1,000 feet edge size



Square Grids are neatly divisible fit into smaller grids

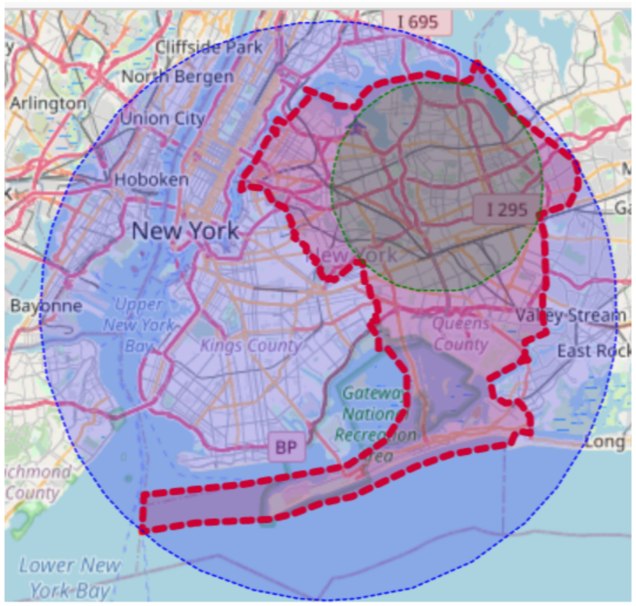


PostGIS 3.1: MaximumInscribedCircle - new vs. MinimumBoundingCircle

http://postgis.net/docs/manual-dev/ST_MaximumInscribedCircle.html
(http://postgis.net/docs/manual-dev/ST_MaximumInscribedCircle.html)
http://postgis.net/docs/ST_MinimumBoundingCircle.html
(http://postgis.net/docs/ST_MinimumBoundingCircle.html)

```
SELECT  ic.radius, ic.center, ic.nearest, ST_Buffer(ic.center, ic.radius) As geom
 FROM ch11.boroughs AS c, ST_MaximumInscribedCircle(c.geom) AS ic
 WHERE boroname = 'Brooklyn';

SELECT  ST_MinimumBoundingCircle(c.geom) AS geom
 FROM ch11.boroughs AS c
 WHERE boroname = 'Brooklyn';
```



PostGIS 3.0 enhanced GeoJSON support to now accept full features

very very old way of creating a feature collection - https://www.postgresonline.com/journal/archives/267-Creating-GeoJSON-Feature-Collections-with-JSON-and-PostGIS-functions.html (https://www.postgresonline.com/journal/archives/267-Creating-GeoJSON-Feature-Collections-with-JSON-and-PostGIS-functions.html) (painful)

New way

```
SELECT json_build_object('type', 'FeatureCollection', 'features',
    json_agg(ST_AsGeoJSON(c.*)::json) )
 FROM ch11.cities AS c
 -- (transform is just to convert to 2227 North CA stateplane feet)
 WHERE c.geom && ST_Transform(ST_MakeEnvelope(-122, 37.74, -121.5, 38,4326 ), 2227);
```

In [ ]:

```
%%sql
SELECT ST_AsGeoJSON(a.*)
 FROM ch09.airports AS a
    WHERE municipality ILIKE 'BOSTON%' LIMIT 3;
```

PostGIS 3 ST_ASMVT faster and more parallelizable

If you are new to Mapbox Vector Tiles, try using https://github.com/CrunchyData/pg_tileserv (https://github.com/CrunchyData/pg_tileserv) (Crunchy Data pg_tileserv). pg_tileserv is a minimalist tile server written in Go that leverages PostGIS MVT functions. General concept behind it detailed - https://info.crunchydata.com/blog/dynamic-vector-tiles-from-postgis (https://info.crunchydata.com/blog/dynamic-vector-tiles-from-postgis)

1. Download the binary for your OS
2. Create a shell script # Nix

   ```
   export DATABASE_URL=postgresql://postgres:password@localhost/postgis_in_action
   pg_tileserv
   ```

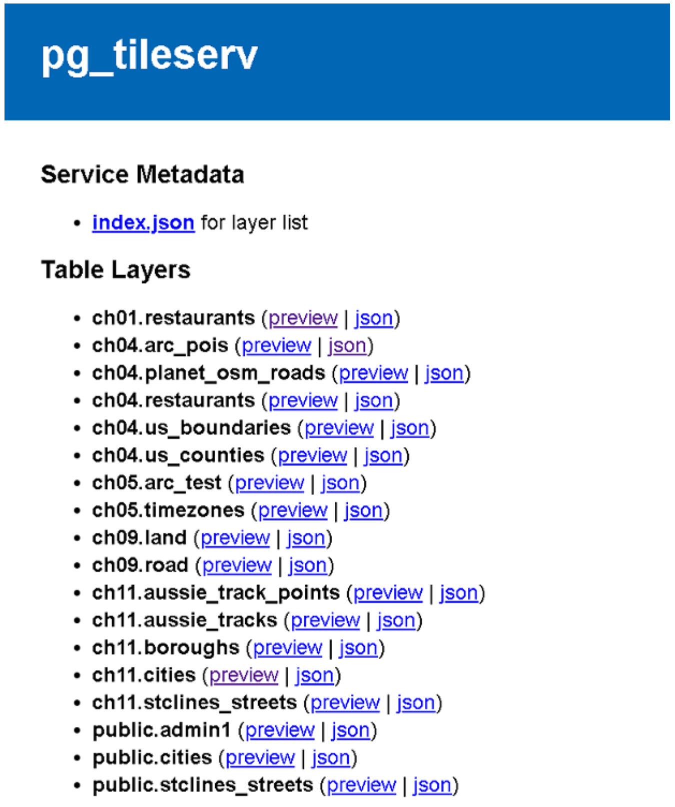
Windows

```
set DATABASE_URL=postgresql://postgres:password@localhost/postgis_in_action
pg_tileserv
```

1. Edit the packaged pg_tileserv.toml file if you want to change the port etc.

2. Browse to http://localhost:7800 (http://localhost:7800)

pg_tileserv catalog



pg_tileserv

Service Metadata

- **index.json** for layer list

Table Layers

- **ch01.restaurants** (preview | json)
- **ch04.arc_pois** (preview | json)
- **ch04.planet_osm_roads** (preview | json)
- **ch04.restaurants** (preview | json)
- **ch04.us_boundaries** (preview | json)
- **ch04.us_counties** (preview | json)
- **ch05.arc_test** (preview | json)
- **ch05.timezones** (preview | json)
- **ch09.land** (preview | json)
- **ch09.road** (preview | json)
- **ch11.aussie_track_points** (preview | json)
- **ch11.aussie_tracks** (preview | json)
- **ch11.boroughs** (preview | json)
- **ch11.cities** (preview | json)
- **ch11.stclines_streets** (preview | json)
- **public.admin1** (preview | json)
- **public.cities** (preview | json)
- **public.stclines_streets** (preview | json)

How do you use mapbox vector tiles for your maps?

In [ ]:

```python
from ipyleaflet import Map, VectorTileLayer, basemaps, LayersControl


# http://localhost:7800/ch04.us_counties/{z}/{x}/{y}.pbf #tile url for counties can use in leaflet
vlparcels = VectorTileLayer(name='Parcels', url='http://localhost:7800/staging.parcels/{z}/{x}/{y}.pbf')
vlff = VectorTileLayer(name='Fast Food', url='http://localhost:7800/ch01.restaurants/{z}/{x}/{y}.pbf')

m = Map(center=(42.38,-71.12), zoom = 15, basemap=basemaps.OpenStreetMap.BlackAndWhite)

m.add_layer(vlparcels)
m.add_layer(vlff)
m.add_control(LayersControl())
m
```

+
−
OpenStreetMap.BlackAndWhite
Parcels
Fast Food
Davis
West Somerville
North Cambridge Carhouse

```
In [ ]:
```