Going spatial with PostGIS

Hugo Mercier – Oslandia

OSLANDIA

Oslandia

- Open source GIS experts
  - Training
  - Consulting
  - Support
  - Development
- Techs
  - PostGIS
  - QGIS
  - Mapserver, ...

OSLANDIA

Outline

PostGIS basics

Topology

Rasters

Point cloud

3D

PostGIS

Geographical Information System extension to PostgreSQL

Storage and analysis

OSLANDIA

Spatial RDBMS

Vector data (geometry):

- Points, lines, polygons, volumes

Raster data:

- Georeferenced array of pixels

SQL Queries on

- Attributes
- Geometries / rasters

OSLANDIA

PostGIS project : standards

International standards

Specifications

- OGC SFS (Simple Feature for SQL)
- ISO SQL/MM part 3

What's in :

- Geometry types
- Spatial functions prototypes

PostGIS 2.0

- Released in 2012
- Brings :
  - Rasters
  - Topology
- Other extensions
  - PgRouting
  - Point cloud

PostGIS Geometries

0D : Points

1D : Linestrings

2D : Polygons

3D : PolyhedralSurfaces (2.0)

... and functions to manipulate them

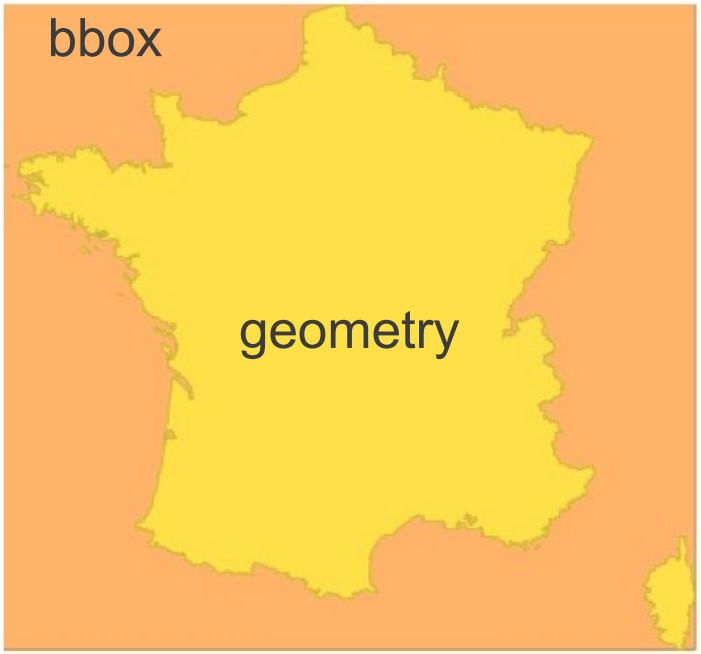
- ST_MakePoint, ST_MakeLine
- ST_PointN, ST_ExteriorRing, etc.

Spatial index

Better spatial filter performance

Geometry approximation with bbox

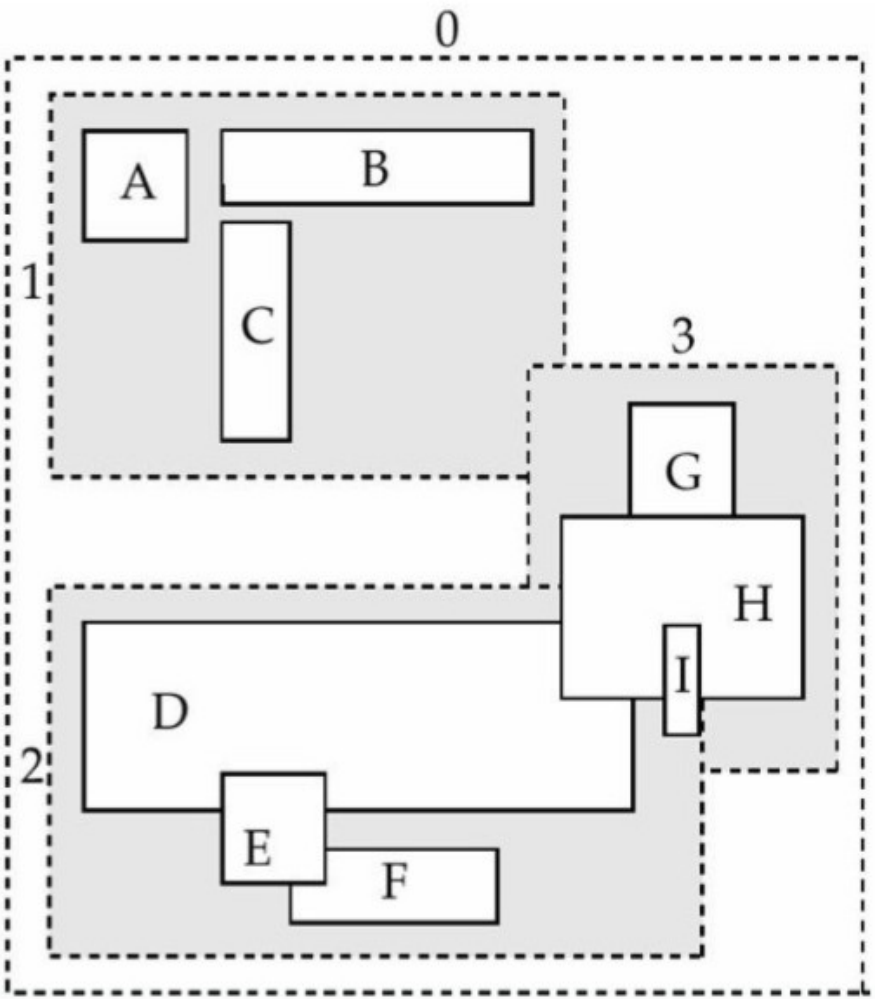
Spatial index creation :

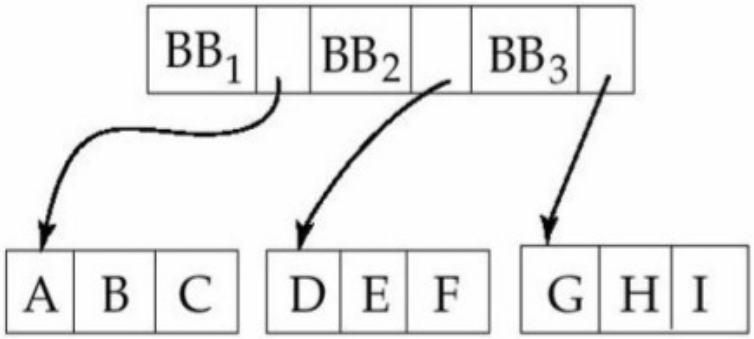



```
CREATE INDEX index_name ON table_name
USING GIST(geom_column_name);
```

OSLANDIA

Spatial index : R-tree



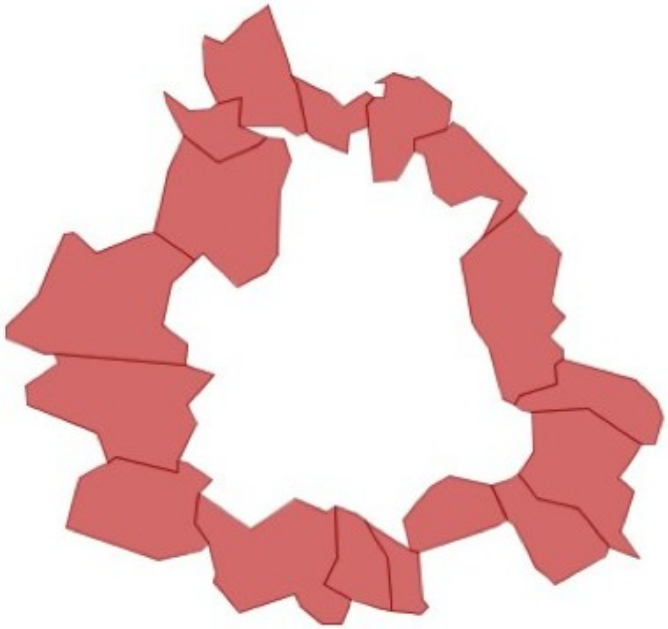


Bounding boxes are grouped in regions of the index

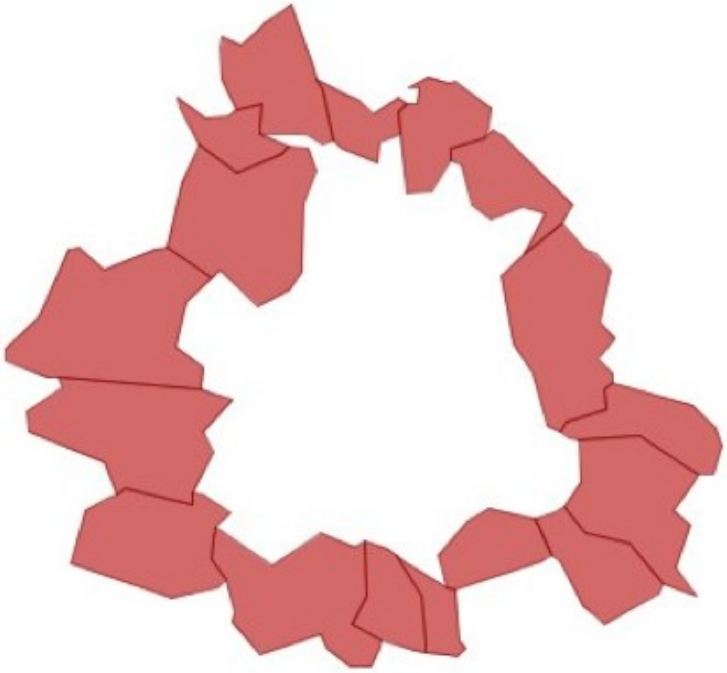
OSLANDIA

Spatial index

```
SELECT
  c1.name
FROM
  cities c1, cities c2
WHERE
  c2.name = 'Toulouse'
  AND ST_Touches(c1.geom, c2.geom);
```



Sans index: temps = 150 ms



Avec index: temps = 30 ms

I/O functions

WKB/WKT (OGC)

```
LINESTRING(0 0,10 10)
0102000000020000000000000000000000000000
0000000000000000000000024400000000000000
002440
```

Import / export tools

- Shp2pgsql, pgsql2shp (ESRI shapefiles)

GeoJSON, GML, KML, SVG X3D

- ST_GeomFrom[WKB,WKT,GeoJSON,GML...]
- ST_As[WKB,WKT,GeoJSON,GML,...]

Relationship and measurements

ST_Length, ST_Area

ST_Distance

ST_Intersects, ST_Covers, ST_Contains

Operators on bboxes :

- A && B : A intersects B
- A &&& B : the same in 3D
- A << B : A to the left of B
- A ~ B : A contains B
- ...

OSLANDIA

Processing

ST_Buffer(geom, radius)

ST_Intersection(geomA, geomB)

ST_Union(geom set)

ST_Transform(geom, SRID) : reprojection

Processing

ST_ConvexHull

ST_Simplify : Douglas-Peucker algorithm

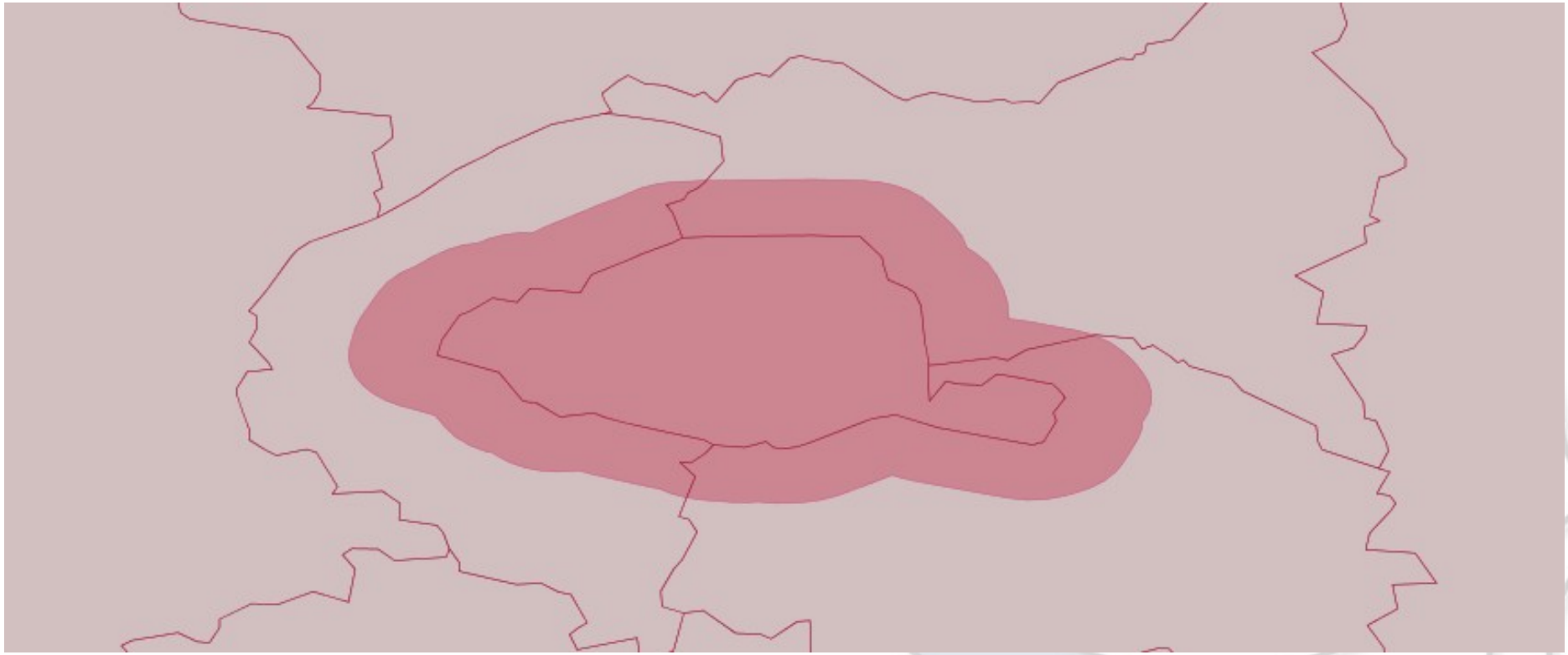
ST_Difference

ST_Split

ST_Snap

PostGIS functions : buffer

```
SELECT ST_Buffer(the_geom, 2500)
FROM dept
WHERE code_dept='75';
```



PostGIS functions : intersection

```
SELECT nom_dept
FROM dept
WHERE ST_Intersects(the_geom,
  (SELECT ST_Buffer(the_geom, 2500)
   FROM dept WHERE code_dept='75')
);
```

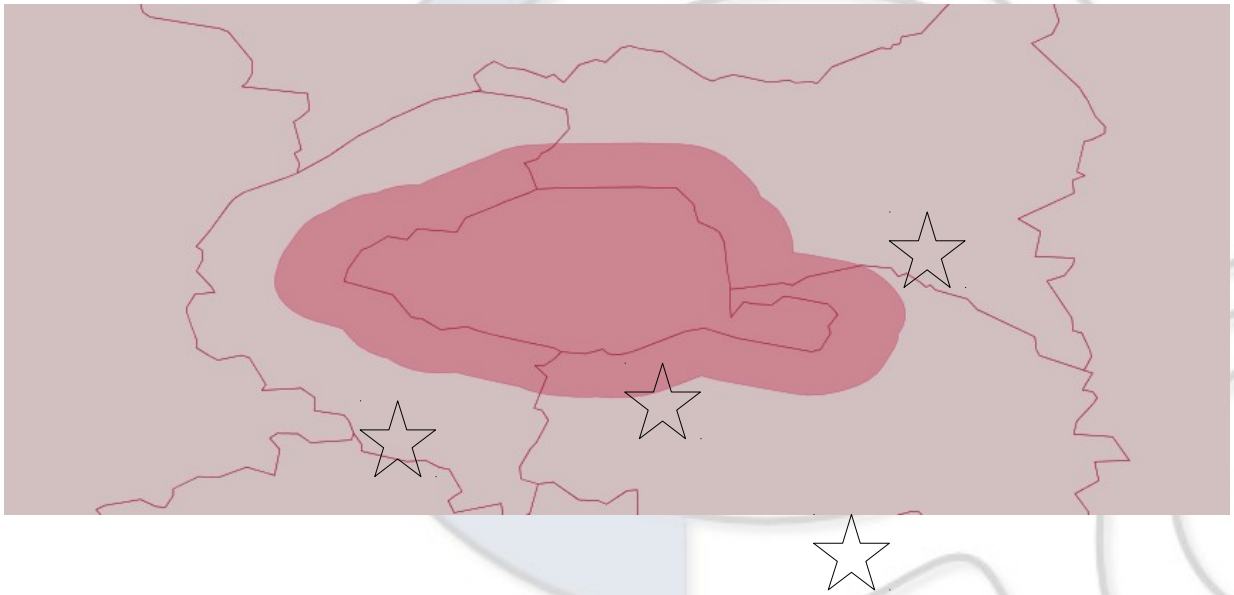
Results :

PARIS

HAUTS-DE-SEINE

SEINE-SAINT-DENIS

VAL-DE-MARNE



OSLANDIA

PostGIS functions : geometry aggregate

```
SELECT ST_Union(geom)
FROM commune
GROUP BY code_dept;
```

Geometries
for communes

Les communes de France

Geometries for departments
built from communes

Les communes de France
fusionnées par département

PostGIS functions : ST_Distance

```
SELECT code_dept, round(
  ST_Distance(ST_Centroid(the_geom),
  (SELECT ST_Centroid(the_geom)
   FROM dept WHERE code_dept='75')) / 1000)
AS distance
FROM dept ORDER BY distance LIMIT 4;
```

Results:

```
75|0
92|7
93|12
94|13
```

OSLANDIA

Topology

Beware of the spaghetti monster !

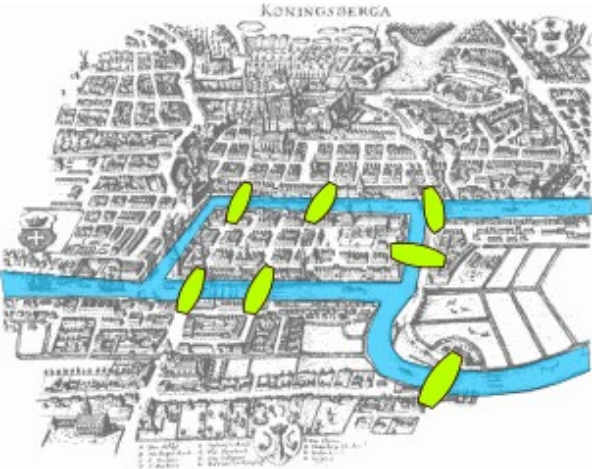
OSLANDIA

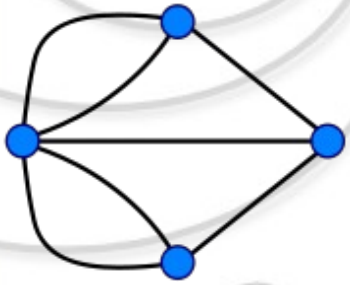
Topology - Graphs

Explicit relations between objects

Graph representation

Node / edge / face





Topology

TopoGeometry Datatype

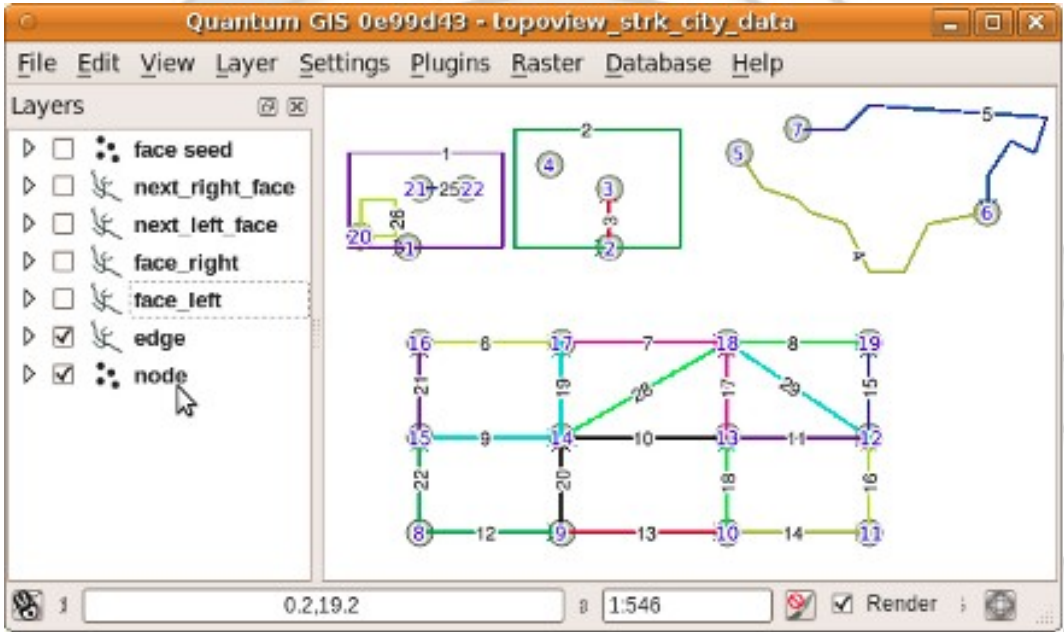
Uses schemas

- «topology» for functions and others
- Each topology in its own schema

Full SQL/MM topology support

Integrated in 2.0

- Sandro Santilli
- Toscane Region



OSLANDIA

Topology

Reduced storage size

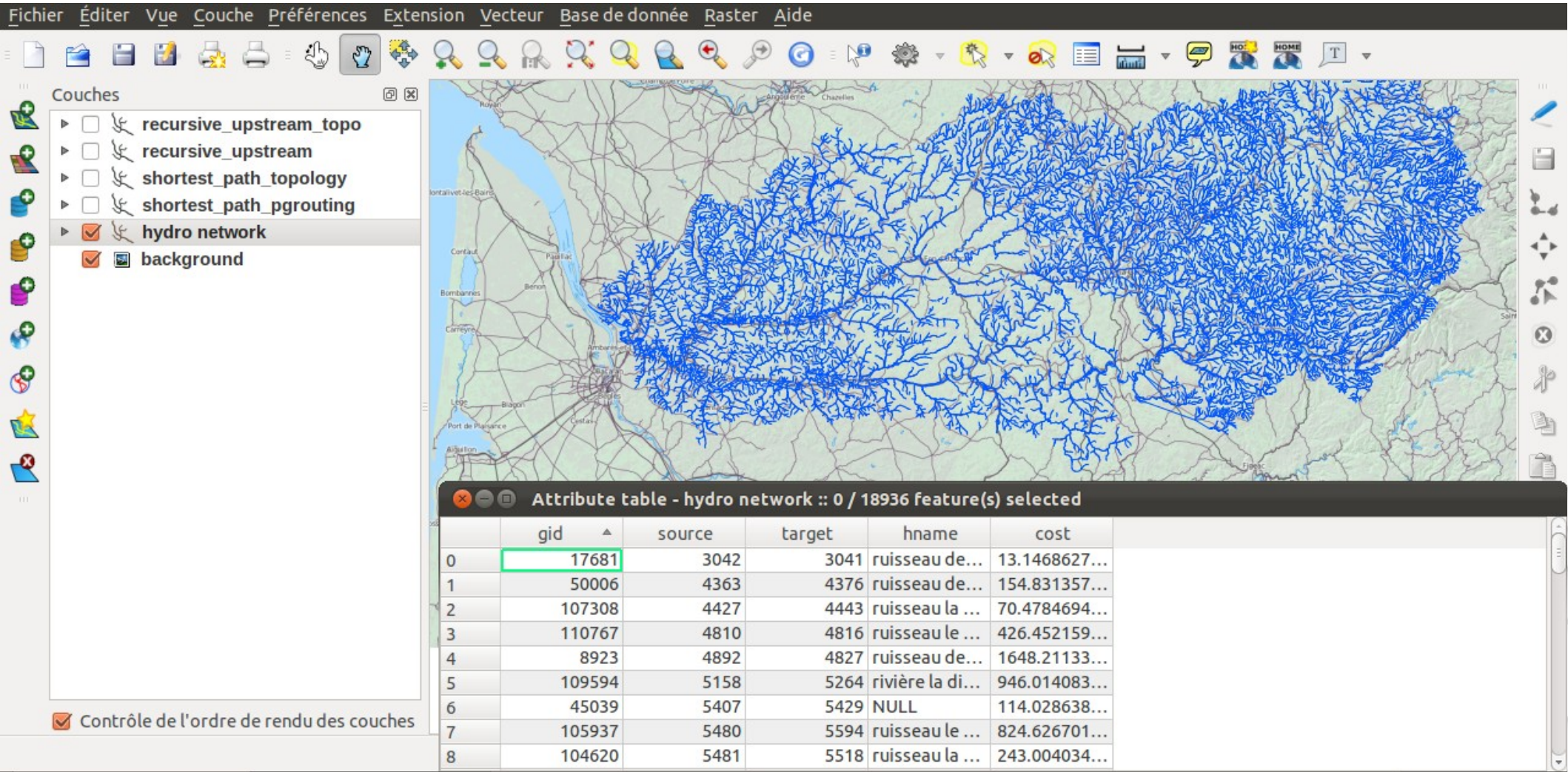
Explicit spatial relationships

Without topology !

OSLANDIA

Topology use case

Table name : **tr**



	gid	source	target	hname	cost
0	17681	3042	3041	ruisseau de...	13.1468627...
1	50006	4363	4376	ruisseau de...	154.831357...
2	107308	4427	4443	ruisseau la ...	70.4784694...
3	110767	4810	4816	ruisseau le ...	426.452159...
4	8923	4892	4827	ruisseau de...	1648.21133...
5	109594	5158	5264	rivière la di...	946.014083...
6	45039	5407	5429	NULL	114.028638...
7	105937	5480	5594	ruisseau le ...	824.626701...
8	104620	5481	5518	ruisseau la ...	243.004034...

Topology

Geometry table = spaghetti

Custom attribute-based topology :

source, target (and cost)

```
select * from tr limit 10;
```

au sortie

rtie de données | Expliquer (Explain) | Messages | Historique

gid integer	source integer	target integer	hname character varying(127)	cost double precision	geom geometry(MultiLin
17681	3042	3041	ruisseau de chaize	13.146862743	01050000206A0
50006	4363	4376	ruisseau de villevalei	154.83135760	01050000206A0
107308	4427	4443	ruisseau la méouzette	70.478469414	01050000206A0

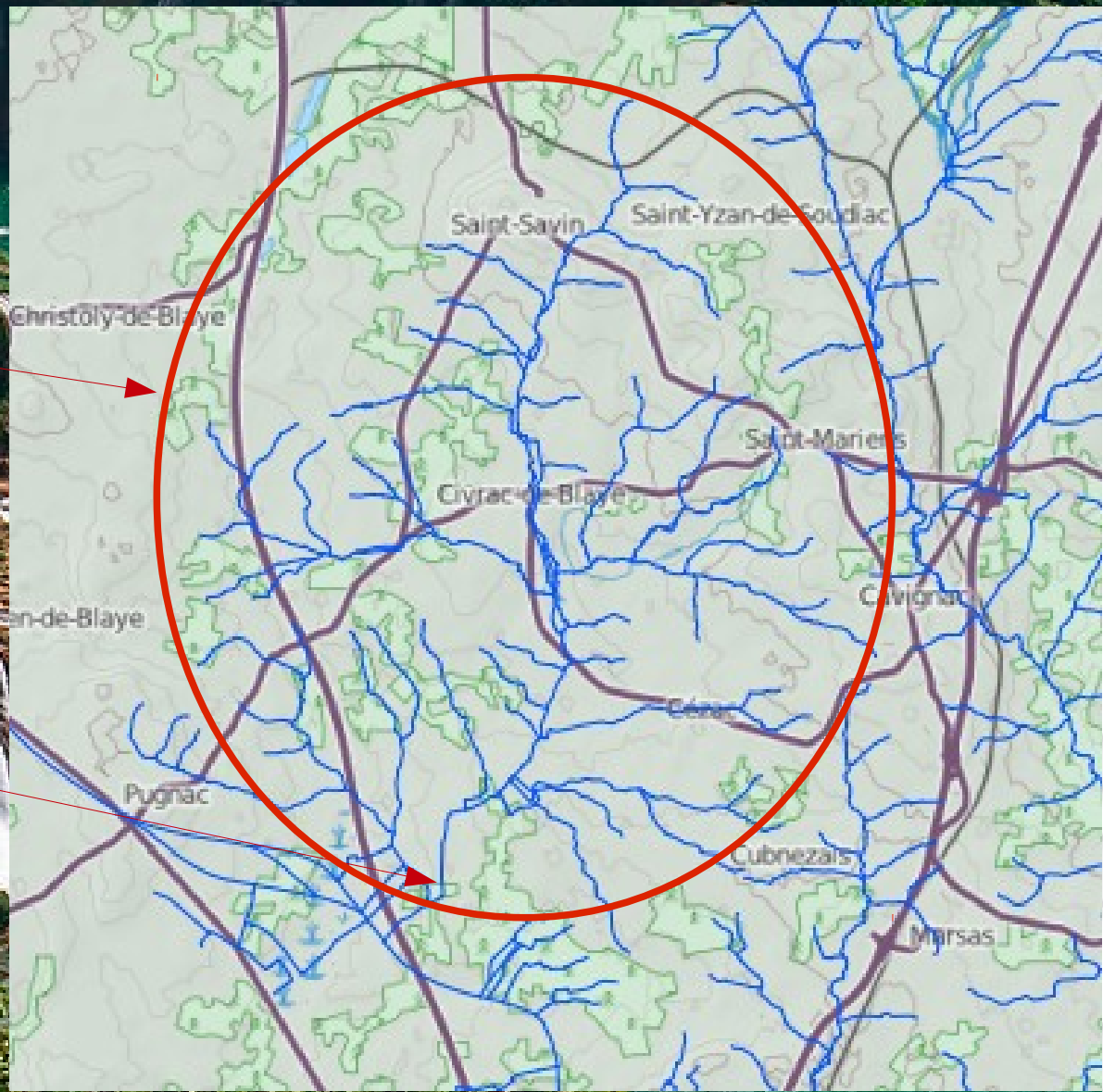
Or build a PostGIS topology based on geom

OSLANDIA

Find upstream

We want all connected edges upstream

Starting edge



Touching upstream edges

```
/* == Find all upstream edges == */

-- our starting edge : 31913
select gid, source, target, hname, cost from tr where gid = 31913;
/*
  gid  | source | target |       hname       |       cost
-------+--------+--------+-------------------+------------------
 31913 |  20850 |  21413 | ruisseau le moron | 2666.05230179502
*/

-- our starting edge and all upstream touching edges
select gid, source, target, hname, cost from tr where gid = 31913
union all
select gid, source, target, hname, cost from tr where target = 20850

/*
  gid  | source | target |          hname          |       cost
-------+--------+--------+-------------------------+------------------
 31913 |  20850 |  21413 | ruisseau le moron       | 2666.05230179502
 33855 |  20735 |  20850 | ruisseau de la marzelle | 807.256330186324
 32477 |  20845 |  20850 | ruisseau le moron       | 59.7117241419599
(3 rows)
*/
```

OSLANDIA

```
create table
        rec_res as
with recursive
        search_graph(gid, source, depth, path, length, cycle) as (
                select
                        g.gid, g.source, 1 as depth, ARRAY[g.gid] as path
                        , cost, false as cycle
                from
                        tr as g
                where
                        gid = 31913
                union all
                select
                        g.gid
                        , g.source
                        , sg.depth + 1 as depth
                        , path || g.gid as path
                        , sg.length + g.cost as length
                        , g.gid = ANY(path) as cycle
                from
                        tr as g
                join
                        search_graph as sg
                on
                        sg.source = g.target
                where
                        not cycle
                )
```

1

2

Recursive CTE

```
select
        sg.*
        , tr.geom
from
        search_graph as sg
join
        tr
on
        sg.gid = tr.gid
limit 1000;
```

3

gid integer	source integer	depth integer	path integer[]	length double precision	cycle boolean	geom geometry(MultiLineString,2154)
31913	20850	1	{31913}	2666.0523017	f	01050000206A08000001000
33855	20735	2	{31913,	3473.3086319	f	01050000206A08000001000
32477	20845	2	{31913,	2725.7640259	f	01050000206A08000001000
33854	19909	3	{31913,	7183.7295195	f	01050000206A08000001000

OSLANDIA

1 : init

```
select
        g.gid, g.source, 1 as depth, ARRAY[g.gid] as path
        , cost, false as cycle
from
        tr as g
where
        gid = 31913
```

OSLANDIA

2 : recursive part

```
select
        g.gid
        , g.source
        , sg.depth + 1 as depth
        , path || g.gid as path
        , sg.length + g.cost as length
        , g.gid = ANY(path) as cycle
from
        tr as g
join
        search_graph as sg
on
        sg.source = g.target
where
        not cycle
```

Stack the gid to the path for this record

Sum up the cost (it's the length here)

If the record gid is already in the path, we have a cycle

Join result set from previous iteration to connected upstream edges

Do not take elements which make a cycle

3 : Get results

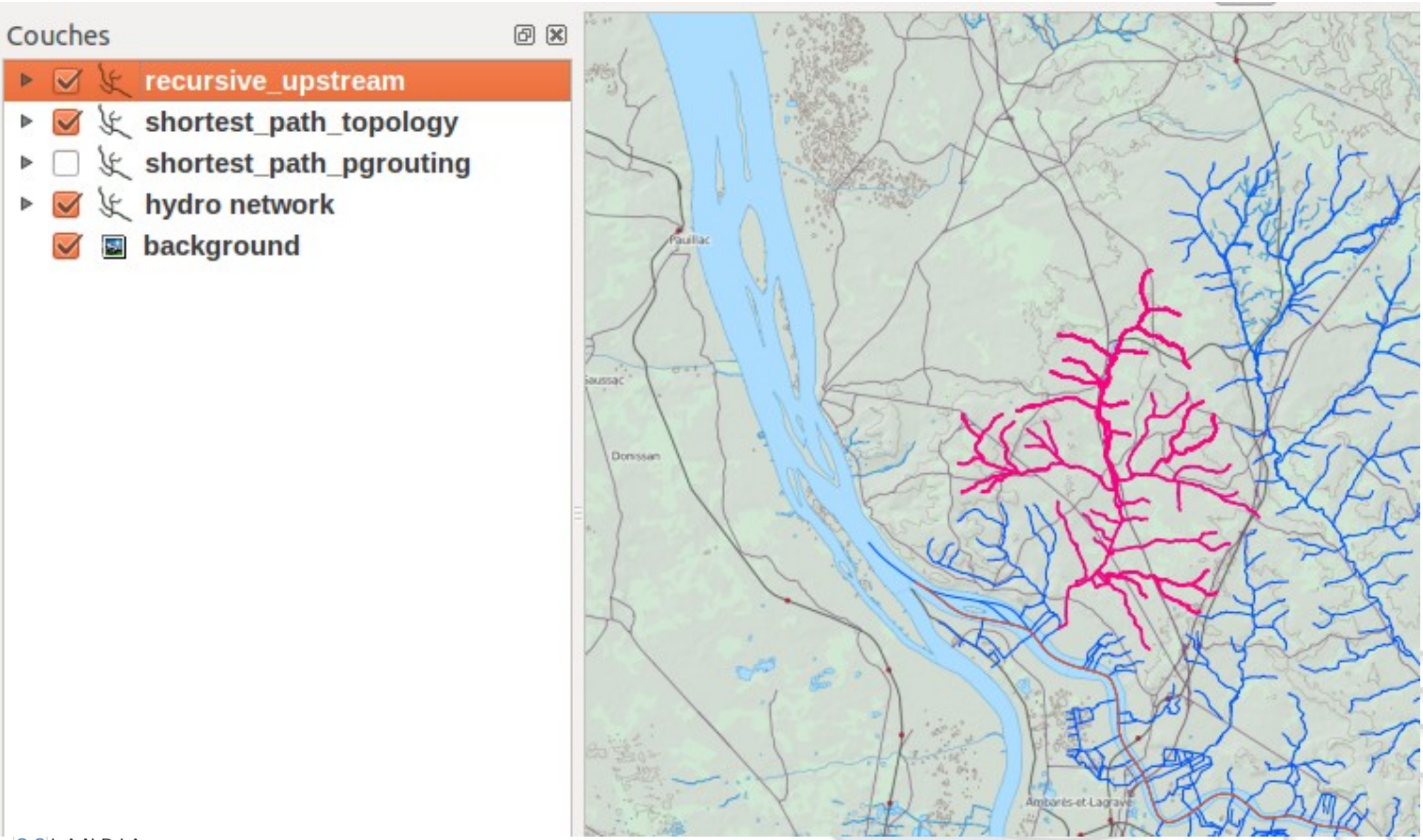
```
select
        sg.*
        , tr.geom
from
        search_graph as sg
join
        tr
on
        sg.gid = tr.gid
limit 1000;
```

Join CTE results to original table to get geometries

Better limit recursive queries to avoid unfinite loops

gid integer	source integer	depth integer	path integer[]	length double precision	cycle boolean	geom geometry(MultiLineString,2154)
31913	20850	1	{31913}	2666.0523017	f	01050000206A08000001000
33855	20735	2	{31913,	3473.3086319	f	01050000206A08000001000
32477	20845	2	{31913,	2725.7640259	f	01050000206A08000001000
33854	19909	3	{31913,	7183.7295195	f	01050000206A08000001000

OSLANDIA


Couches
recursive_upstream
shortest_path_topology
shortest_path_pgrouting
hydro network
background
Pauillac
Donissan
Ambarès-et-Lagrave

```sql
-- Create a topology
SELECT topology.CreateTopology('hydro', 2154);
-- 1

-- we put the postgis topology features for hydro network in another table
CREATE TABLE tr_topo (gid integer);

-- Add a layer
SELECT topology.AddTopoGeometryColumn('hydro', 'public',
        'tr_topo', 'topogeom', 'MULTILINESTRING');
-- 1

-- Populate the layer and the topology from tr geometry features
INSERT into tr_topo (gid, topogeom)
        SELECT gid, topology.toTopoGeom(geom, 'hydro', 1) FROM tr;
```

- Schémas (3)
  - hydro
    - Collationnements (0)
    - Domaines (0)
    - Configurations FTS (0)
    - Dictionnaires FTS (0)
    - Analyseurs FTS (0)
    - Modèles FTS (0)
    - Fonctions (0)
    - Séquences (5)
    - Tables (4)
      - edge_data
      - face
      - node
      - relation
    - Fonctions trigger (0)
    - Types (0)
    - Vues (1)
      - edge

```sql
select * from hydro.edge limit 10;
```

neau sortie

ortie de données | Expliquer (Explain) | Messages | Historique

	edge_id integer	start_node integer	end_node integer	next_left_edge integer	next_right_edge integer	left_face integer	right_face integer	geom geometry(LineStrin
1	175256	190369	190361	175230	-175243	0	0	01020000206A08
2	167356	183762	181917	166725	167356	0	0	01020000206A08

```sql
select * from tr_topo limit 10;
```

au sortie

rtie de données | Expliquer (Explain) | Messages

	gid integer	topogeom topology.topogeometry
	116768	(1,1,163704,2)
	116767	(1,1,163705,2)

… and on PostGIS topology

1

```
create table
        rec_res2 as
with recursive
        search_graph(edge_id, start_node, depth, path, length, cycle) as (
                select
                        g.edge_id, g.start_node, 1 as depth, ARRAY[g.edge_id] as path
                        , st_length(geom) as length, false as cycle
                from
                        hydro.edge as g
                where
                        edge_id = 173832
```

2

```
                union all
                select
                        g.edge_id
                        , g.start_node
                        , sg.depth + 1 as depth
                        , path || g.edge_id as path
                        , sg.length + st_length(g.geom) as length
                        , g.edge_id = ANY(path) as cycle
                from
                        hydro.edge as g
                join
                        search_graph as sg
                on
                        sg.start_node = g.end_node
                where
                        not cycle
                )
```

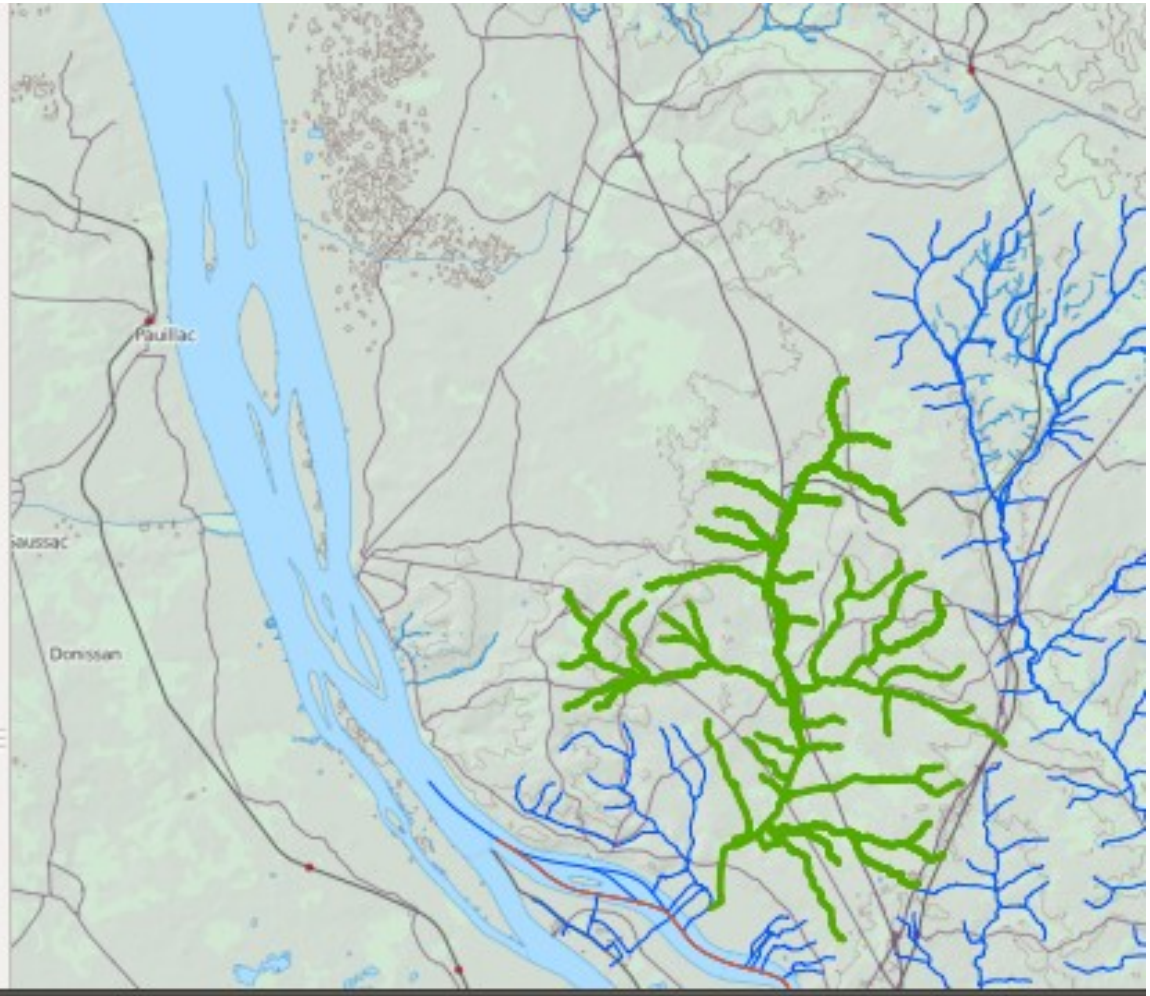
3

```
select
        sg.*
        , edge.geom as geom
from
        search_graph as sg
join
        hydro.edge as edge
on
        sg.edge_id = edge.edge_id
limit 1000;
```

OSLANDIA

Couches

- recursive_upstream_topo
- recursive_upstream
- shortest_path_topology
- shortest_path_pgrouting
- hydro network
- background



Attribute table - recursive_upstream_topo :: 0 / 478 feature(s) selected

	edge_id	start_node	depth	path	length	cycle
0	173832	189333	1	{173832}	2666.05230...	f
1	173452	189332	2	{173832,17...	3473.30863...	f

OSLANDIA

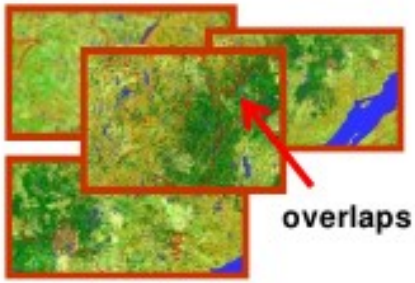
Rasters

OSLANDIA

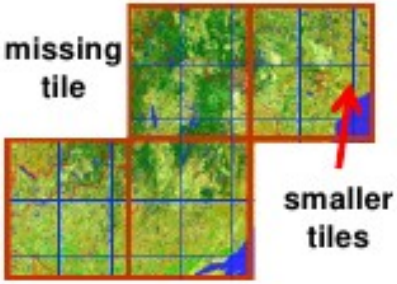
PostGIS Rasters

- Raster ~= bitmap picture
  - With a georeference
- Multiresolution, multiband, tile coverage
- New datatype
- Import/export (GDAL)
- Functions
  - Statistics, reprojection, edit, compute
  - Vector/raster functions

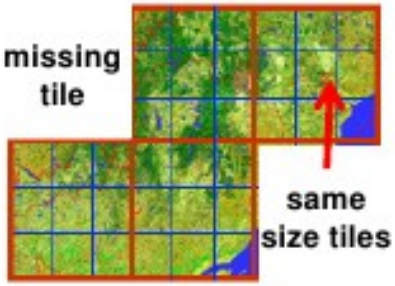
PostGIS Rasters



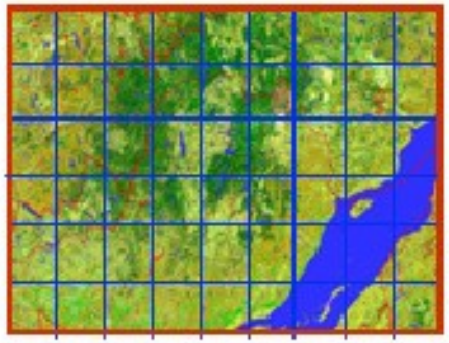
a) warehouse of untiled and unrelated images (4 images)



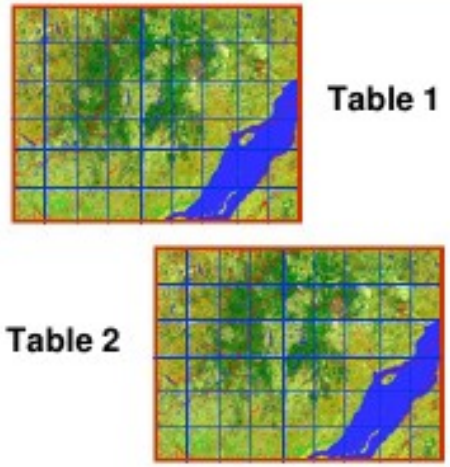
b) irregularly tiled raster coverage (36 tiles)



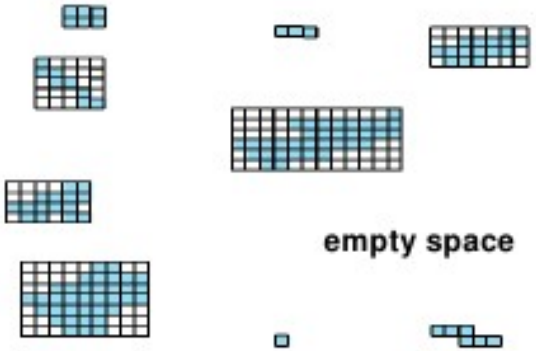
c) regularly tiled raster coverage (36 tiles)



d) rectangular regularly tiled raster coverage (54 tiles)



e) tiled images (2 tables of 54 tiles)



f) rasterized geometries coverage (9 lines in the table)

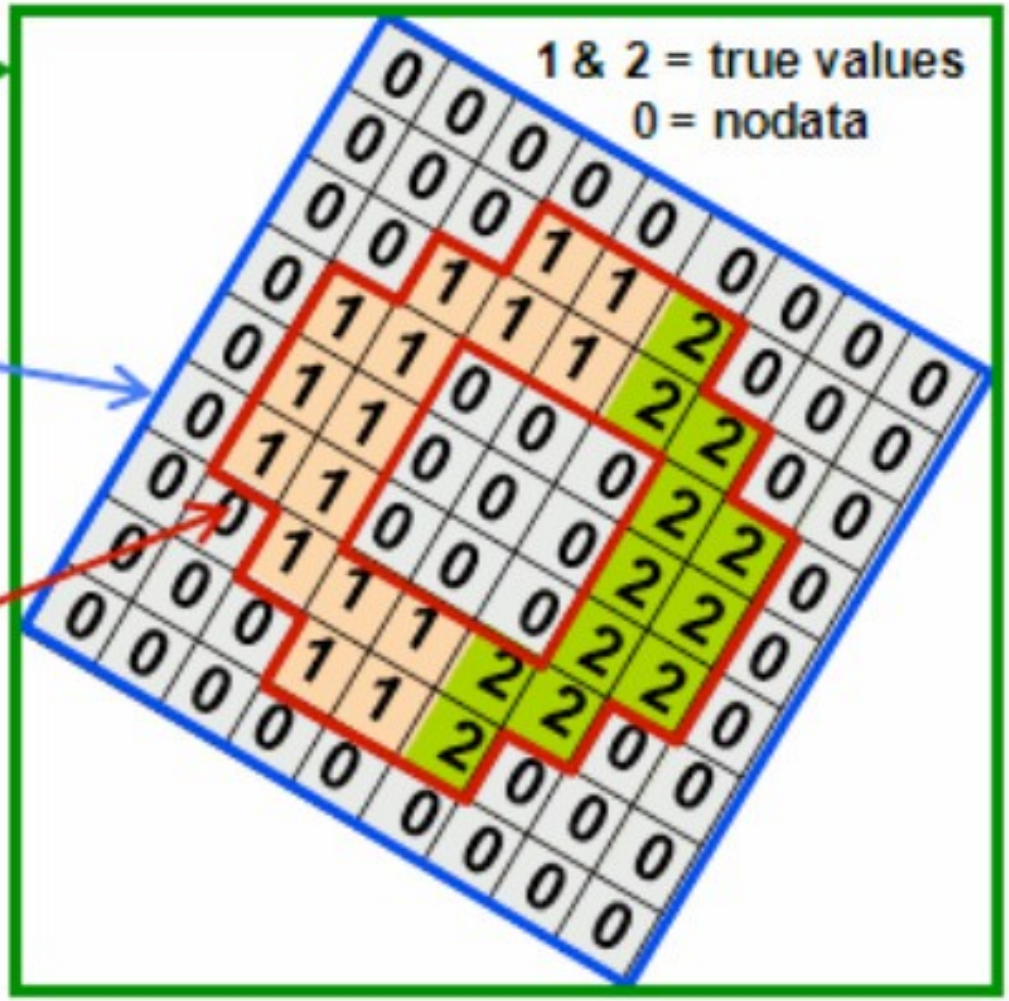
Rasters

Geometries ↔ rasters

ST_Envelope(raster) →

ST_ConvexHull(raster) →

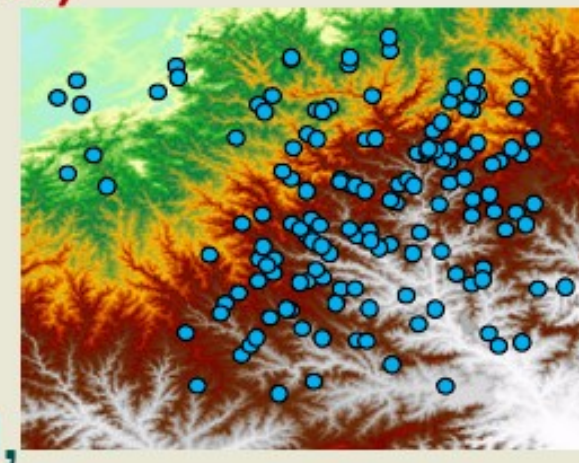
ST_Polygon(raster) →



Queries with rasters

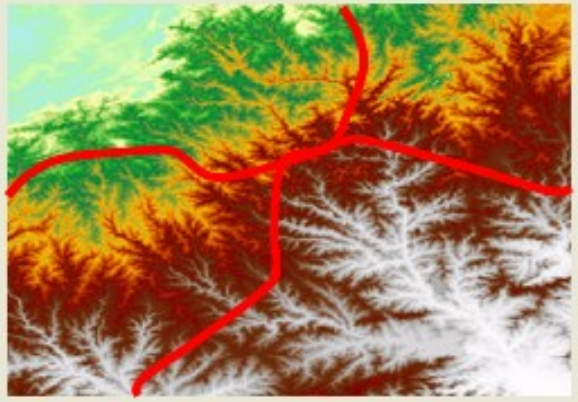
Extract ground elevation values for lidar points…

- SELECT pointID, ST_Value(rast, geom) elevation
  FROM lidar, srtm WHERE ST_Intersects(geom, rast)

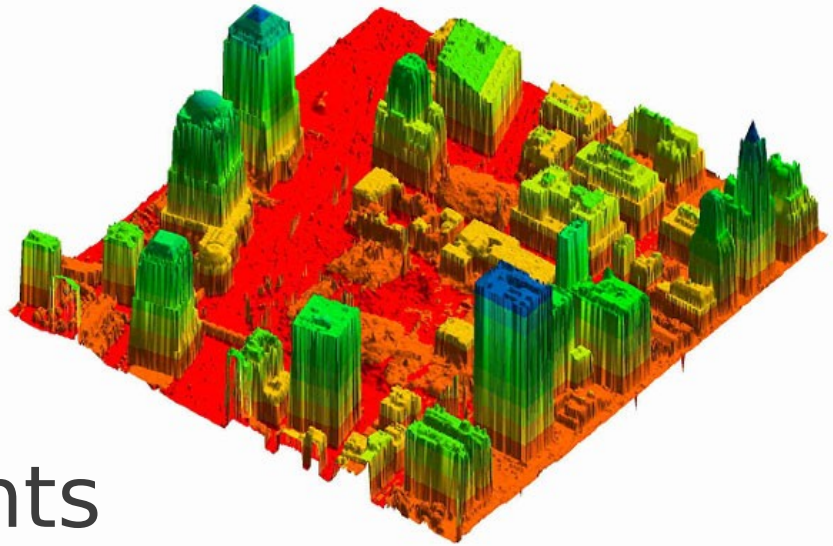


Intersect a road network to extract elevation values for each road segment

- SELECT roadID,
  (ST_Intersection(geom, rast)).geom road,
  (ST_Intersection(geom, rast)).val elevation
  FROM roadNetwork, srtm WHERE ST_Intersects(geom, rast)



Point cloud (independant / 2.1)



LIDAR data

As a PG extension

- And a PostGIS extension

Points and patches of points

Arbitrary number of dimensions

- and representations
- (Use of point cloud « schema »)

Compression storage

Link with PDAL

OSLANDIA

Point cloud

Spatial filtering

```
SELECT PC_AsText(PC_Explode(PC_Intersection(
      pa,
      'SRID=4326;POLYGON((-126.451 45.552, -126.42
47.55, -126.40 45.552, -126.451 45.552))'::geometry
)))
FROM patches WHERE id = 7;

             pc_astext
-----------------------------------------
 {"pcid":1,"pt":[-126.44,45.56,56,5]}
 {"pcid":1,"pt":[-126.43,45.57,57,5]}
 {"pcid":1,"pt":[-126.42,45.58,58,5]}
 {"pcid":1,"pt":[-126.41,45.59,59,5]}
```

OSLANDIA

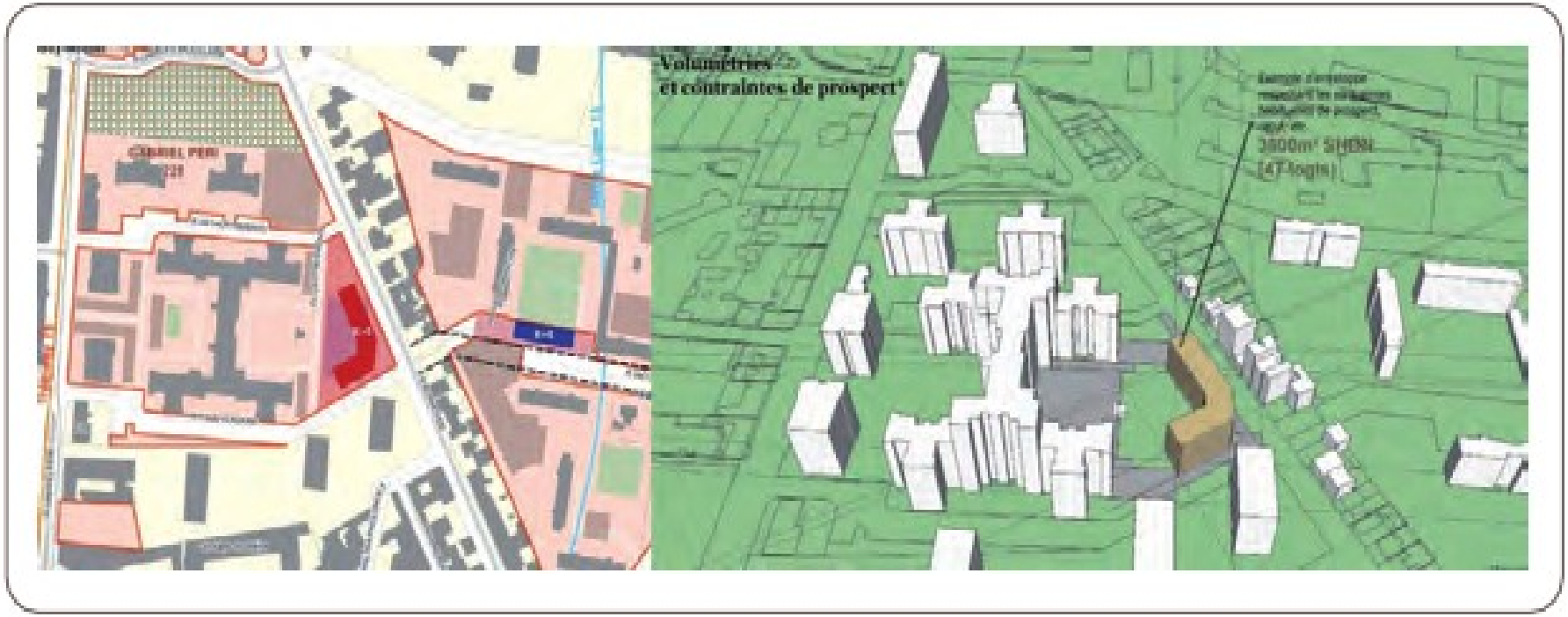
Let's go 3D !

OSLANDIA

PostGIS 3D

Partial FEDER funding

Focus on urban planning



OSLANDIA

PostGIS 3D

2D & 3D geometric computation

C++

Exact computation

Efficient, generic, extensible...

OSLANDIA

SFCGAL

GIS Layer on top of CGAL

Some operators :

- 3Dintersects
- 3Dintersection
- 3Dconvexhull
- Tesselation
- Straight skeletons
- Extrusion

1.0 release in July

www.sfcgal.org

SFCGAL

=

CGAL

+

CERTIFIED OGC® COMPLIANT

OSLANDIA

PostGIS 3D

SFCGAL is integrated into PostGIS 2.1

3D storage is in

SFCGAL provides :

- ST_3Dintersects
- ST_3Dintersection
- ST_Extrude (2D -> 3D)
- ST_3Dconvexhull
- ST_StraightSkeleton
- ST_Tesselate...

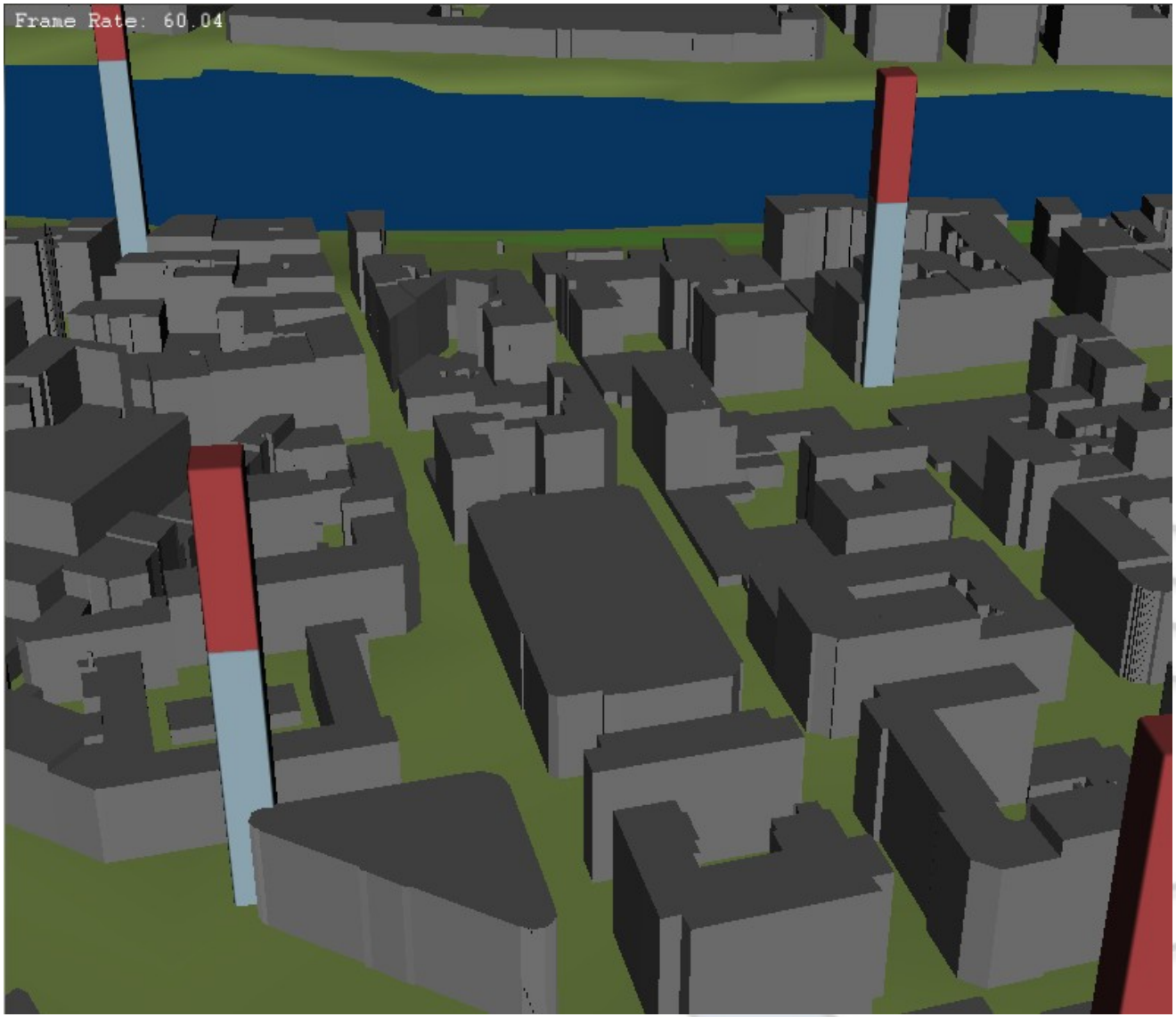
OSLANDIA

PostGIS 3D + QGIS

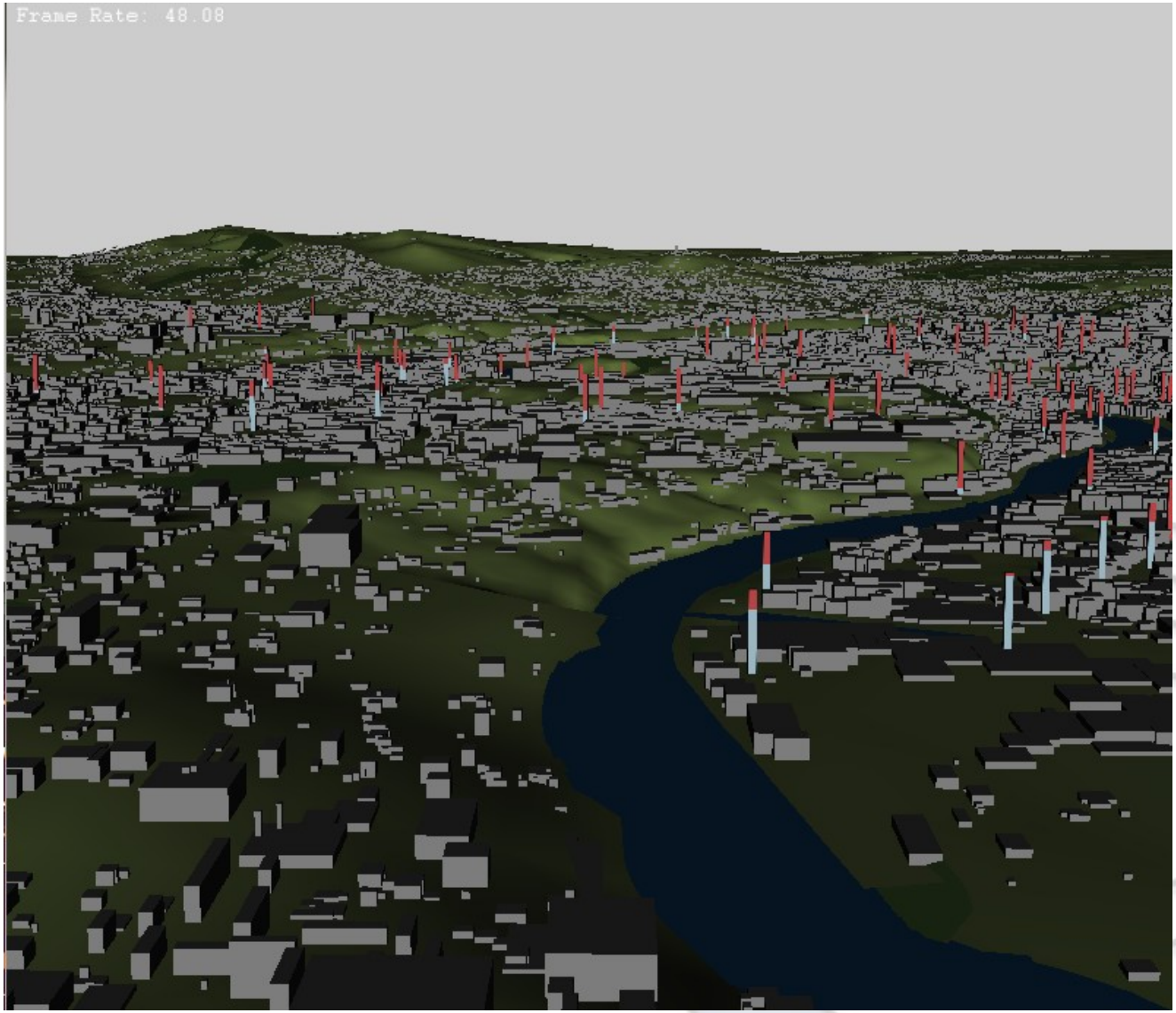
Visualisation project based on an OpenSceneGraph (OpenGL) stack

Horao

Plugin for QGIS



OSLANDIA



OSLANDIA

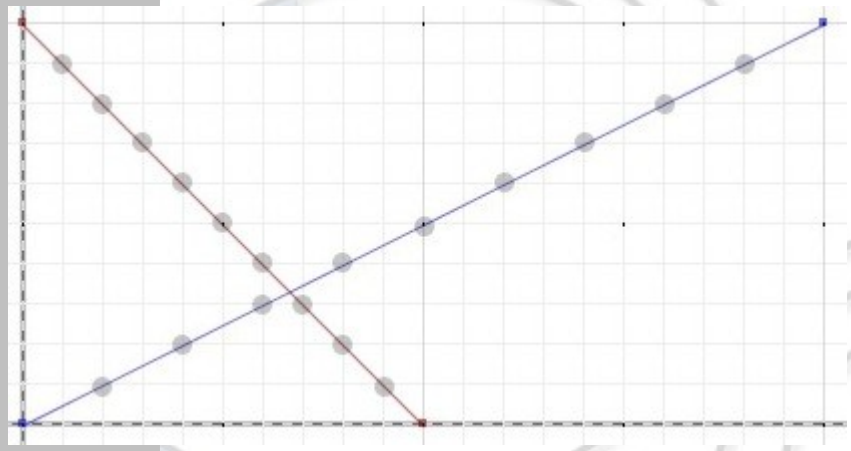
PostGIS 3D : next steps

CGAL : exact computations

New objects : exact geometries

- Try to avoid serialization
- (PostgreSQL patch)

```
SELECT
  ST_Intersects(
    ST_Intersection(
      'LINESTRING(0 0,2 1)'::geometry,
      'LINESTRING(1 0,0 1)'::geometry),
    'LINESTRING(0 0,2 1)'::geometry);
 st_intersects
---------------
 f
(1 row)
```



OSLANDIA

3D Next steps

More features from CGAL

- Alpha shapes
- 3D Minkowski sum
- ...

Better QGIS support (3D symbology)

Textures ?

Find €€€€€ to speed up development

Thanks !

hugo.mercier@oslandia.com

https://github.com/Oslandia

OSLANDIA

Also in PostGIS 2.1

- Additional raster/raster spatial relationships (ST_Contains, ST_Covers, ST_Touches, etc.)
- ST_Distance on curves
- N rasters ST_MapAlgebra
- Performance enhancements:
  - ST_Union, ST_DumpPoints

Coming soon ...

Raster test performances

TWKB

TopoJSON