

# PostGIS



## Analyses spatiales avancées

PostgreSQL Sessions, Paris, 25 sept 2014

# Plan de la présentation

- PostgreSQL comme serveur de traitements géo
- Spécificités d'un serveur de traitement
- Fonctionnalités et réglages liés à ces traitements
- Préparation/nettoyage des données
- Exemples d'utilisation:
  - Recalage de points GPS sur un réseau routier et intégration des points dans le réseau
  - Fermeture de courbes de niveau
  - Itération pour le parcours de réseau hydrologique
  - Animation des Iterations avec Qgis Time Manager

- Utilisation des Common Table Expressions (CTE) à la place de sous requêtes
- Plus facile à écrire
- Plus clair à relire
- Exemple:

```
With matable as (
    Select id, geom
    From communes
    Where code_dept = '75'
) select ...
From matable ...
```

# PostGIS comme serveur de traitement

- Accès « exclusif » à la base
  - Peu de connexions
  - Plus de mémoire disponible pour une seule requête
  - Augmenter WORK\_MEM (sort, ORDER BY)
    - Au niveau de postgresql.conf
    - Avant une requête
- set work\_mem to '100MB';**
- **⚠** Mémoire prise pour chaque opération de sort

# PostGIS comme serveur de traitement

- Table temporaire  
`create temp table ...;`
- Stockée en mémoire si espace suffisant
- UNLOGGED TABLE (PG 9.1+)  
`create unlogged table...;`
- Pas d'écriture de WAL => **beaucoup plus rapide** que des tables classiques
- **⚠️** Pas de protection en cas de crash => restaurer les données avant traitement
- **⚠️** Index unlogged également (depuis 9.3, support des index GiST)

# Préparation des données

- Vérification de la validité des données

```
select st_IsValidReason(geom)  
from matable  
where not st_isValid(geom);
```

Ring Self-intersection[609041.352413901 6871106.22630108]

Ring Self-intersection[664822.900948696 6818304.66749848]

Ring Self-intersection[671185.256427205 6870811.75940178]

...

- ! Résultats indéterminés en cas de données invalides

# Préparation des données

- Réduction de la précision des coordonnées
  - Adapter à la précision des données
  - Moins de sources d'erreurs
  - Optimisation lors d'échange textuel
  - `st_snapToGrid(geom, précision):`  
**update** matable  
**set** geom = `st_snapToGrid(geom, 1);`

MULTIPOLYGON(((632715.29999983 6880137.99999852, 632718.29999983 6880239.69999852,  
MULTIPOLYGON(((632715 6880138, 632718 6880240, 632773 6880258,

- Ex: Couche PLU, format GeoJSON: 34 Mo vs 70 Mo
-  Données potentiellement invalides

# Préparation des données

```
update matable set geom =  
    st_collectionExtract(  
        st_makeValid(  
            st_snapToGrid(geom, 1)  
        ), 3  
    );
```

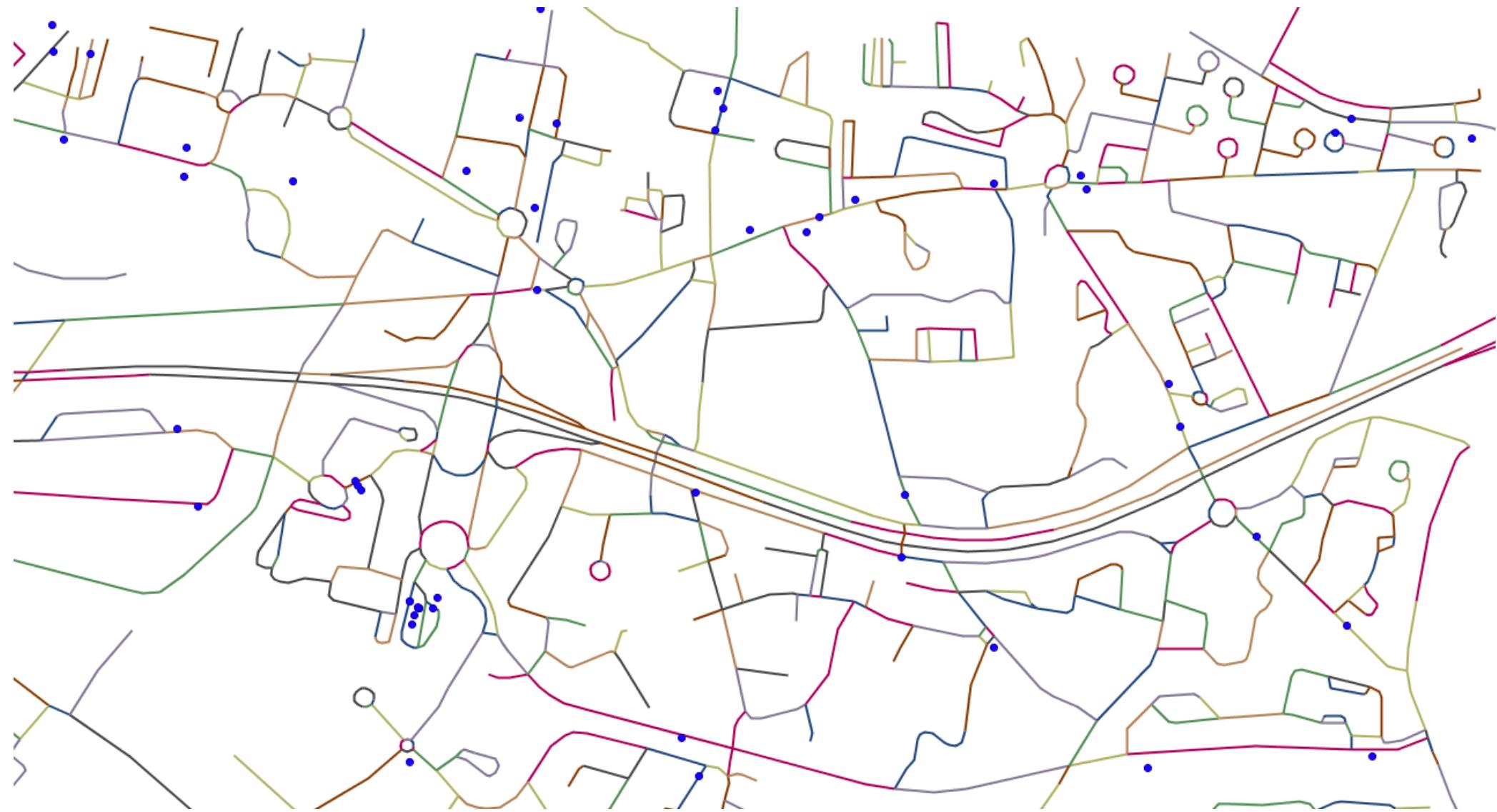
- Contrôle après nettoyage:
- Comparaison des surfaces des polygones avant/après nettoyage

## Traitements avancés: calage de points GPS sur un réseau

- Calage de points GPS sur réseau routier + intégration des nœuds dans le réseau
- But: obtenir une réseau permettant le calcul d'itinéraires multimodaux:
- Données:
  - réseau routier
  - Points GPS des arrêts de bus
- Etapes:
  - Analyse des données => choix des outils/fonctions PostGIS
  - Recalage de points: fonctions de référencement linéaire
  - Intégration de points dans un réseau existant
  - Intégration en une seule requête



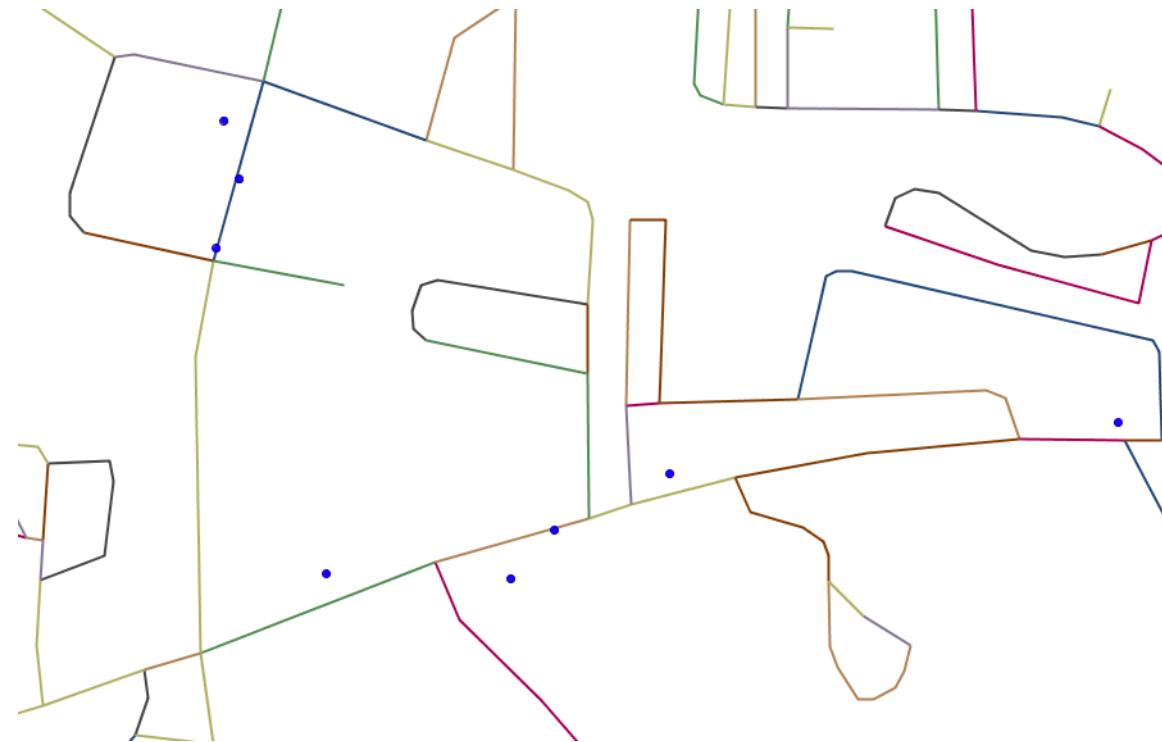
# Traitements avancés: calage de points GPS sur un réseau





# calage de points GPS sur un réseau

- Analyse des données:
- Faible erreur de localisation des points par rapport aux lignes => chaque point est recalé sur la ligne la plus proche



- ! Plusieurs points sur la même ligne => plusieurs sommets à insérer dans une ligne



# calage de points GPS sur un réseau

- Données routières topologiques (connectées à chaque intersection)
  - shp2pgsql -SID -g geom -W latin1 Streets.shp rues | psql pgsql
  - Données GPS: fichier CSV: id, x, y, nom\_arret
    - Création d'une table
    - Commande COPY table FROM 'fichier.csv'
    - Fabrication des géométries (st\_makePoint(X, Y))

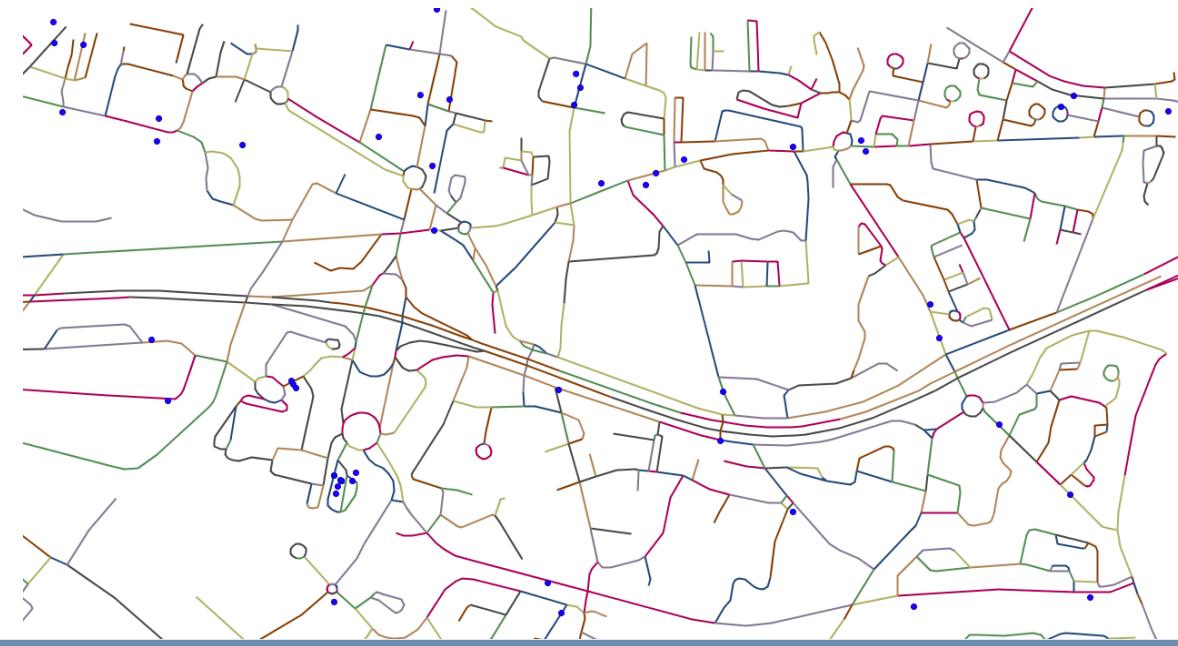
arrets	
id	bigint
id_tisseo	varchar
x	double precision
y	double precision
commune	varchar
nom_commercial	varchar
nom_physique	varchar
nom_hastus	varchar
geom	geometry
gid	int

rues	
gid	int
link_id	numeric(131089)
st_name	varchar(80)
feat_id	numeric(131089)
st_langcd	varchar(3)
num_stnames	double precision
st_nm_pref	varchar(2)
st_typ_bef	varchar(30)
st_nm_base	varchar(35)
st_nm_suff	varchar(2)
st_typ_aft	varchar(30)
st_typ_att	varchar(1)
addr_type	varchar(1)
l_refaddr	varchar(10)



# calage de points GPS sur un réseau

- Les outils nécessaires:
  - Trouver la rue la plus proche de chaque point
  - Coller un point sur une ligne
  - Découper une ligne suivant des points
  - Insérer des points dans une ligne: deux approches:
    - Fusionner des segments de ligne
    - Ordonner les sommets de la ligne et recréer la ligne



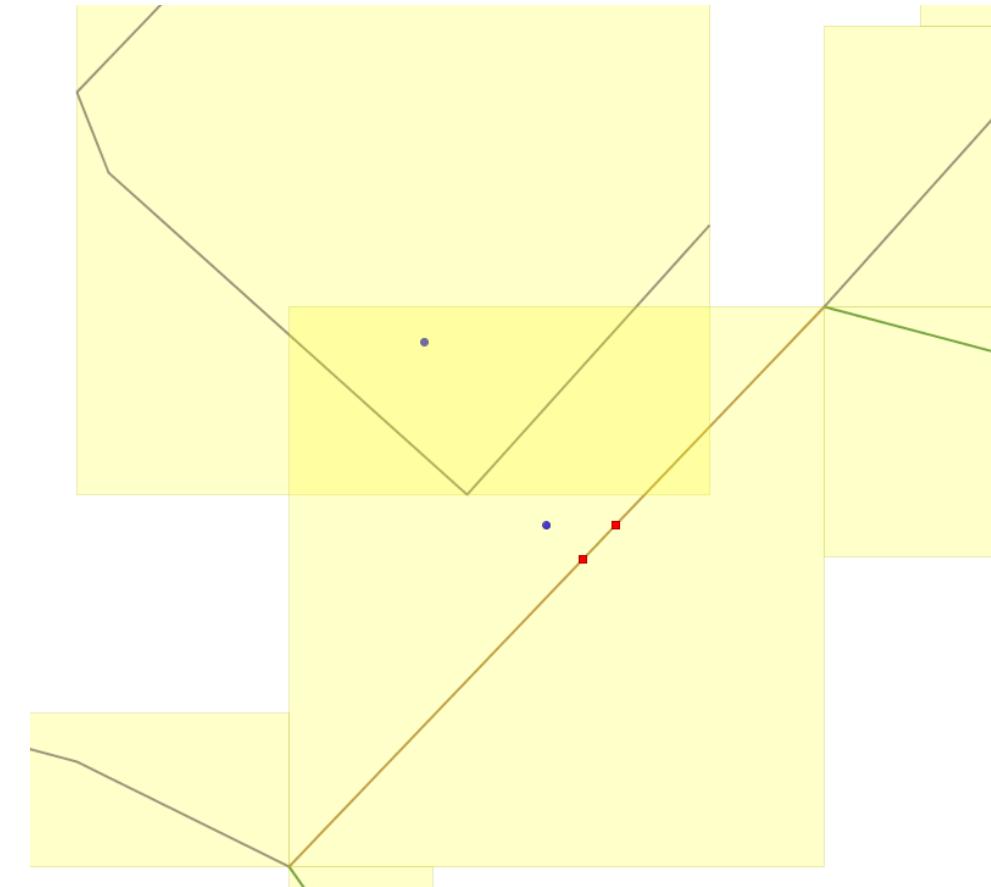
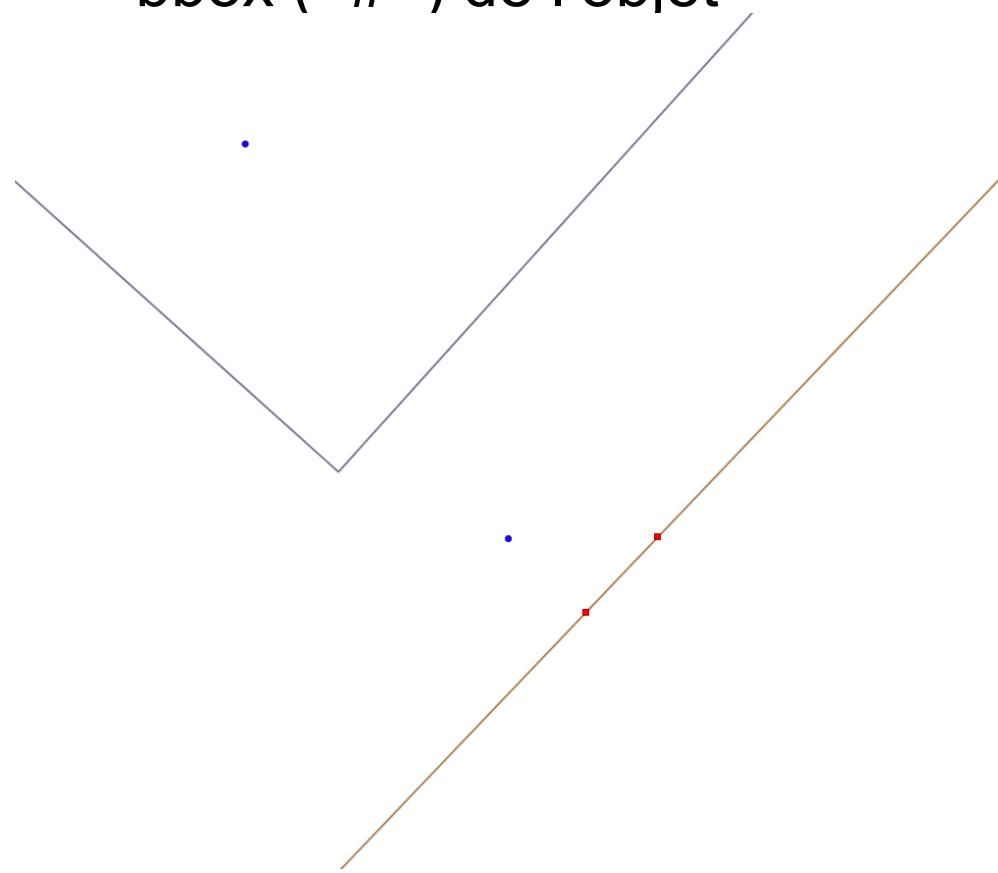


# calage de points GPS sur un réseau

- Les fonctions PostGIS disponibles:
  - Trouver les rues les plus proches de chaque point
    - Index KNN, `st_distance(point, ligne)`
  - Coller un point sur une ligne => trouver l'abscisse du point sur la ligne
    - `St_lineLocatePoint(ligne, point)` => abscisse du point sur la ligne
  - Découper une ligne suivant des abscisses
    - `St_lineSubstring(ligne, abs1, abs2)` => LINESTRING
  - Insérer des points dans une ligne: deux approches:
    - Fusionner des segments de ligne
      - `st_lineMerge(Collection(ligne))` => LINESTRING
    - Ordonner les sommets de la ligne et recréer la ligne
      - `st_dumpPoints(ligne)` => (POINT, index)

# calage de points GPS sur un réseau: requêtes pas-à-pas

- Trouver les rues les plus proches de chaque point
- Index KNN (plus proches voisins) avec opérateurs  $<->$  et  $<\#>$
- **! Problème:** point le plus proche du centroid de la bbox ( $<->$ ) ou de la bbox ( $<\#>$ ) de l'objet



# calage de points GPS sur un réseau: requêtes pas-à-pas

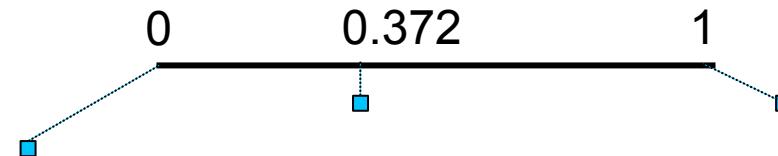
- Trouver les rues les plus proches de chaque point:
- => Trouver n plus proches voisins et classer par st\_distance() la plus petite.
- ☹ Nécessite d'estimer n (ici: 10)

```
SELECT a.gid AS arret_gid,  
unnest(ARRAY(SELECT r.gid  
              FROM rues r  
              ORDER BY r.geom <-> a.geom  
              LIMIT 10)  
        ) AS rues_gid,  
a.geom AS arret_geom  
FROM arrets a;
```



# calage de points GPS sur un réseau

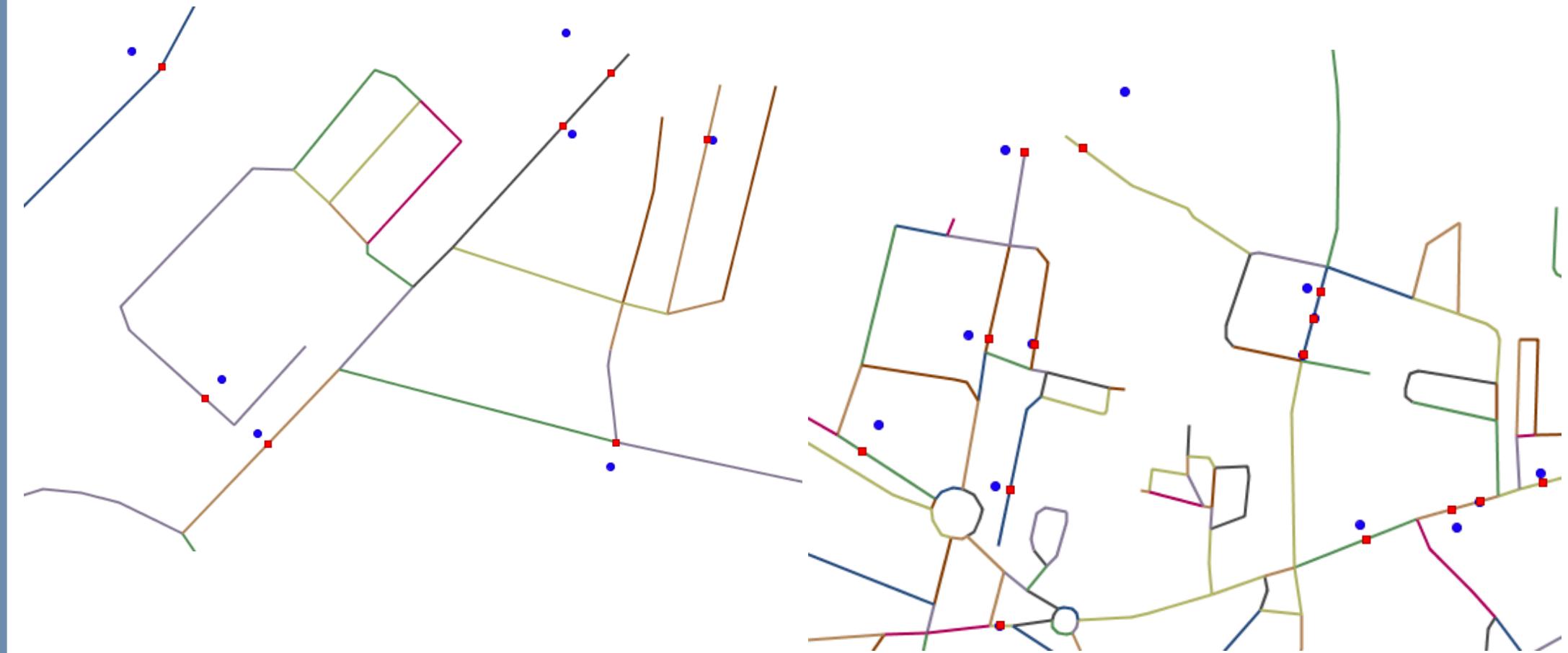
- Coller un point sur une ligne => trouver l'abscisse du point sur la ligne
  - `ST_lineLocatePoint(ligne, point)` => abscisse du point sur la ligne après projection du point sur la ligne



```
select t.arret_gid, t.rues_gid,  
       ST_lineLocatePoint(r.geom, t.arret_geom) as locus,  
       ST_lineInterpolatePoint(r.geom,  
       ST_lineLocatePoint(r.geom, t.arret_geom)) as geom  
from plusproches t  
      join rues r on t.rues_gid = r.gid;
```



## calage de points GPS sur un réseau





# calage de points GPS sur un réseau

- Découper une ligne suivant des abscisses:
  - `St_lineSubstring(ligne, abs1, abs2)` => LINESTRING
- **⚠** (*Cas à prendre en compte: plusieurs points par ligne*)

... **SELECT**

```
t1.gid,  
t1.locus,  
t2.locus,  
st_lineSubstring(t1.geom, t1.locus, t2.locus) as geom  
FROM points t1  
  JOIN points t2 ON t1.gid = t2.gid  
WHERE t2.rn = t1.rn + 1;
```



## calage de points GPS sur un réseau

- plusieurs points par ligne:
  - Découper la ligne entre 0, [abs1, abs2, ... absn] et 1
- self-join sur la table des points, après avoir généré les points d'abscisse 0 (début de ligne) et 1 (fin de ligne)



# calage de points GPS sur un réseau

```
...with locus as (
    SELECT s.gid, p.locus, s.geom
    FROM rues s, pts_proj p
    WHERE s.gid = p.rues_gid
    UNION SELECT s.gid, 0 AS locus, s.geom
    FROM rues s JOIN pts_proj p ON s.gid = p.rues_gid
    UNION SELECT s.gid, 1 AS locus, s.geom
    FROM rues s JOIN pts_proj p ON s.gid = p.rues_gid
), points as (
    SELECT gid, locus, row_number() OVER (PARTITION BY gid ORDER BY
        locus) AS rn, geom
    FROM locus) ...
```



# calage de points GPS sur un réseau

- Insérer des points dans une ligne:
  - Découpage de la ligne avec les points projetés
  - Fusion des lignes: `st_lineMerge(Collection(ligne)) => LINESTRING`

```
...substr as (
  SELECT t1.gid, t1.locus, t2.locus,
         st_lineSubstring(t1.geom, t1.locus, t2.locus) as geom
    FROM points t1 JOIN points t2 ON t1.gid = t2.gid
   WHERE t2.rn = t1.rn + 1
) select s.gid, st_lineMerge(st_collect(s.geom)) as geom
  from substr s group by s.gid;
```

# calage de points GPS sur un réseau

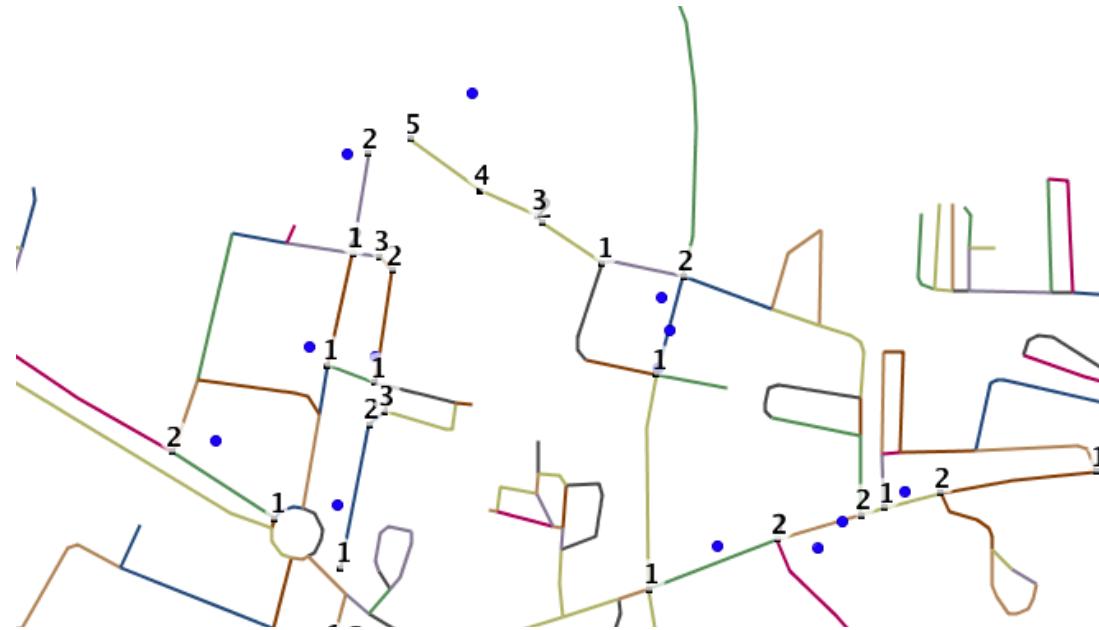
```

with plusproches as (
    SELECT a.gid AS arret_gid, unnest(ARRAY(
        SELECT r.gid
        FROM rues r ORDER BY r.geom <->
        a.geom
        LIMIT 10)) AS rues_gid,
        a.geom AS arret_geom
    FROM arrets a
), dist as (
    SELECT p.arret_gid, rues_gid,
    p.arret_geom,
    row_number() OVER (PARTITION BY
    p.arret_gid ORDER BY
    st_distance(p.arret_geom, r.geom)) as rn
    FROM plusproches p JOIN rues r ON
    p.rues_gid = r.gid
), newpoints as (
    SELECT t.arret_gid, t.rues_gid,
    ST_lineLocatePoint(r.geom,
    t.arret_geom) AS locus,
    r.geom AS geom
    FROM dist t JOIN rues r ON t.rues_gid =
    r.gid
    WHERE rn = 1
), locus as (
    SELECT n.rues_gid, n.locus, n.geom
    FROM newpoints n
    UNION SELECT distinct n.rues_gid,
    0 AS locus, n.geom
    FROM newpoints n
    , points as (
        SELECT rues_gid, locus,
        row_number() OVER (PARTITION BY rues_gid
        ORDER BY locus) AS rn,geom
        FROM locus
    ), substr as (
        SELECT t1.rues_gid, t1.locus, t2.locus,
        st_lineSubstring(t1.geom,t1.locus,t2.locus)
        as geom
        FROM points t1
        JOIN points t2 ON t1.rues_gid =
        t2.rues_gid
        WHERE t2.rn = t1.rn + 1
    ) select s.rues_gid,
    st_lineMerge(st_collect(s.geom)) as geom
    from substr s
    group by s.rues_gid;
)

```

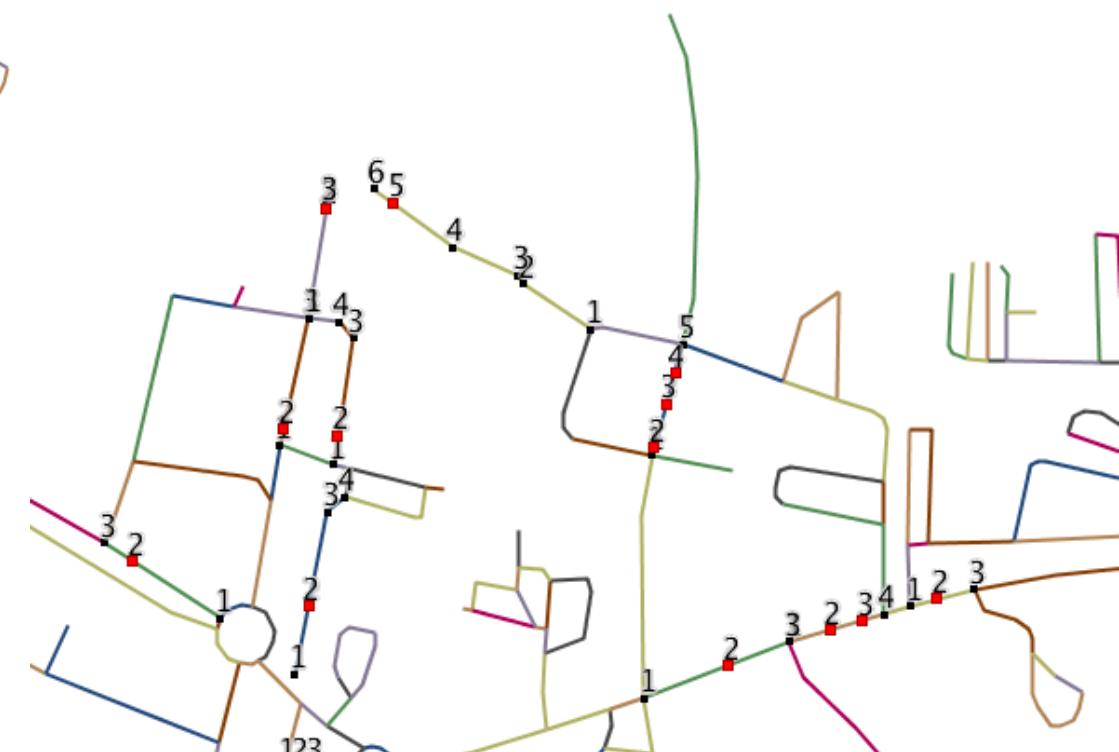


# calage de points GPS sur un réseau



Avant

Après



## Traitements avancés: exemple 2

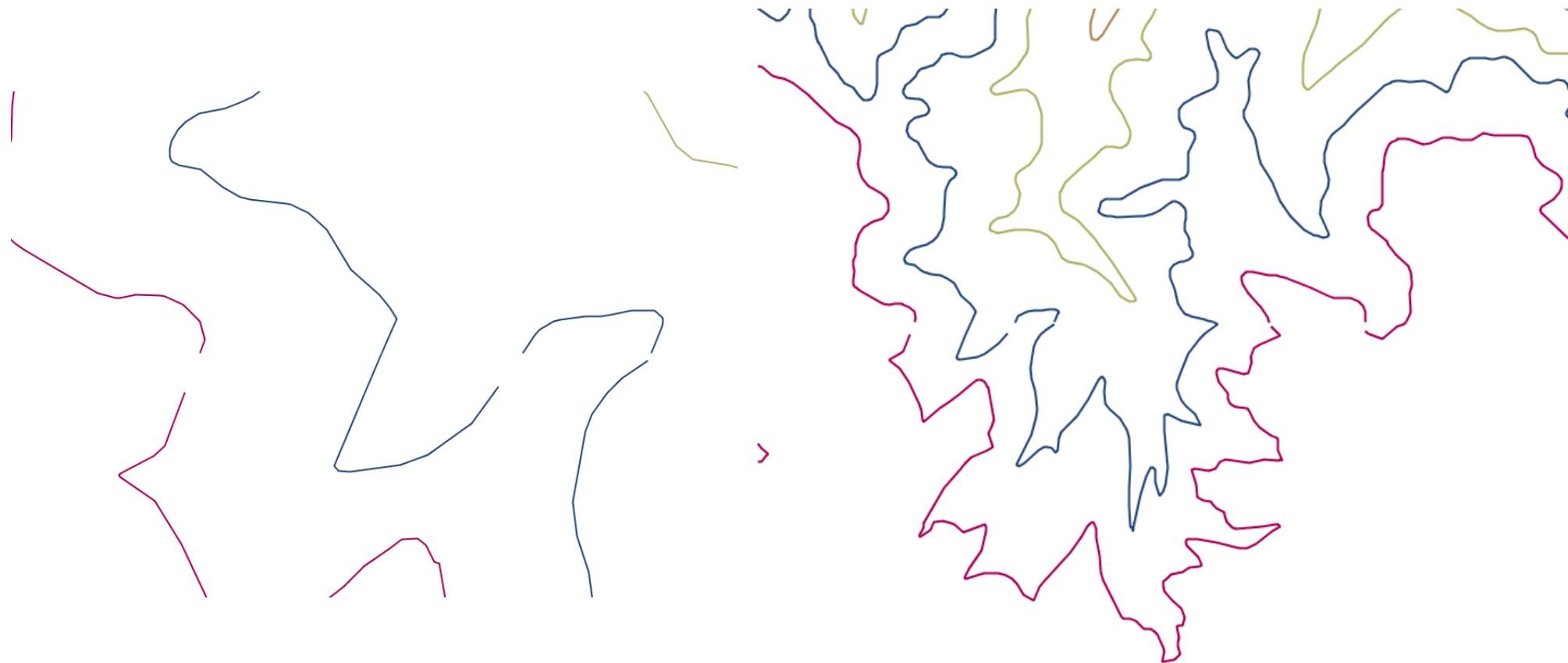
- Nettoyage de données: Fermeture et jointure de courbes de niveau

contour	
gid	int
elevint	int
geom	geometry



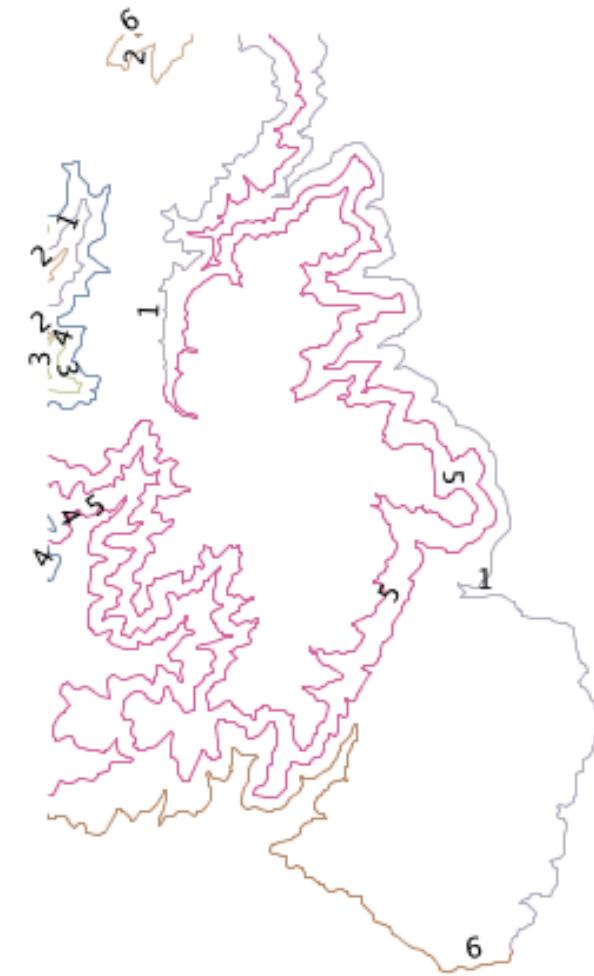
# Fermeture de courbes

- Analyse des données:
  - Ouverture des courbes aux bords de la carte
  - Trous entre segments



# Fermeture de courbes

- LINESTRING et MULTILINESTRING
- Ordre quelconque des segments dans les multilinestrings



## Fermeture de courbes

- `ST_lineMerge(multilinestring) => Linestring (si connectée)`
- `ST_makeLine(line1, line2)` pour connecter des segments consécutifs
- `WITH RECURSIVE` pour parcourir les tronçons de ligne
- Processus par étapes:
  - Traitements différents suivants les cas rencontrés
  - Contrôle des données aux différentes étapes
  - Optimisation possibles sur les tables tmp/unlogged
  - tables tmp fusionnées à la fin du processus

# Fermeture de courbes

contour	
gid	int
elevint	int
geom	geometry

① →  
Merge Linestring  
Merge Linestring

contour_t1	
gid	int
elevint	int
path	int
geom	geometry

② ↓  
Lignes fermées ou  
touchant un bord,  
suppression des lignes  
dans contour\_t1

contour_t2	
gid	int
elevint	int
path	int
geom	geometry

③ →  
Segments les plus  
proches de chaque ligne

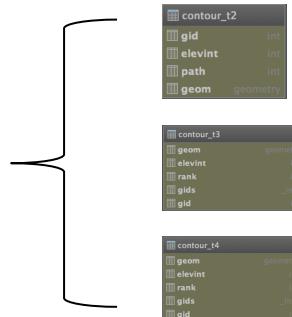
closest	
elevent	int
gid	int
path	int
closest	int
geom	geometry
dist	double precision
r	bigint

④ ↓  
Itération 1: pour chaque ligne touchant  
un bord, fusion (st\_makeShortestLine)  
avec le segment le plus proche

contour_t3	
geom	geometry
elevent	int
rank	int
gids	_int4
gid	int

⑤ ↓  
Suppression des lignes  
dans contour\_t1  
Itération 2: fermeture de  
toutes les lignes  
restantes (brute force)

new_contour	
elevint	int
geom	geometry



⑥ ←  
Union des tables

contour_t4	
geom	geometry
elevent	int
rank	int
gids	_int4
gid	int

# Fermeture de courbes

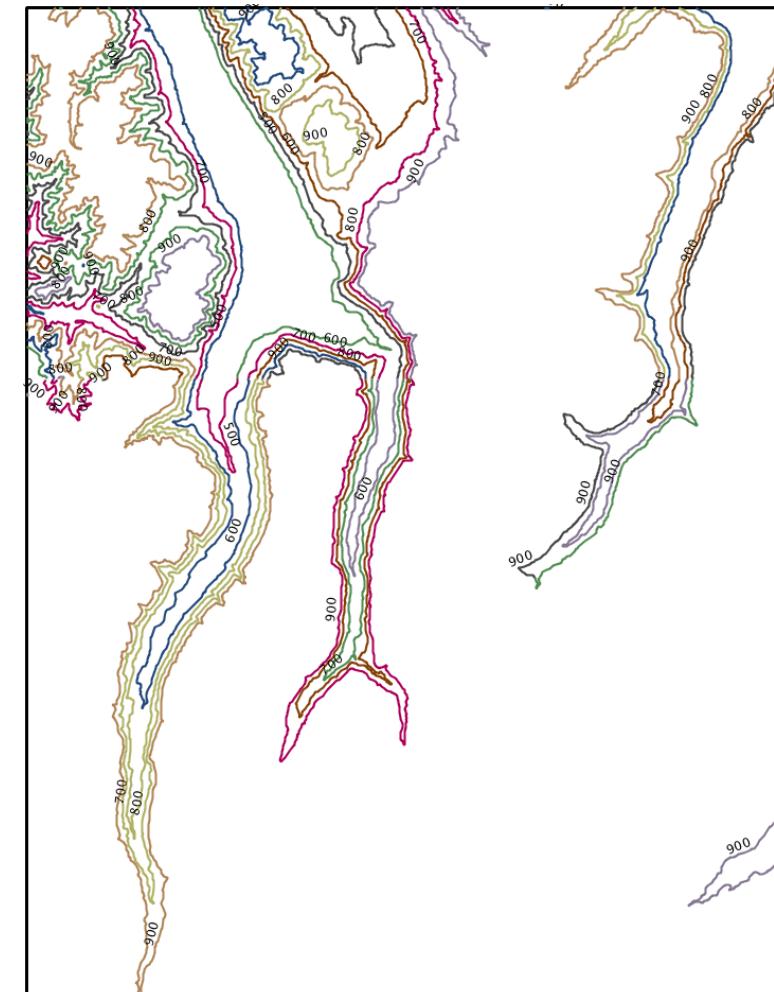
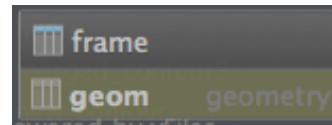
- Fonction `st_makeShortestLine(line1, line2)`

```
create or replace function st_makeShortestLine(g1 geometry, g2 geometry) returns geometry as $$  
with dist as (  
    select st_distance(st_endPoint($1), st_endPoint($2)) as dee,  
          st_distance(st_endPoint($1), st_startPoint($2)) as des,  
          st_distance(st_startPoint($1), st_endPoint($2)) as dse,  
          st_distance(st_startPoint($1), st_startPoint($2)) as ds  
) select case  
    when des = 0 or (dee < des and dee < dse and dee < ds) then st_makeline($1, st_reverse($2))  
    when des = 0 or (des < dee and des < dse and des < ds) then st_makeline($1, $2)  
    when ds = 0 or (dss < dee and dss < dse and dss < des) then st_makeline(st_reverse($1), $2)  
    else st_makeline($2, $1) end  
from dist;  
$$ language SQL;
```

# Fermeture de courbes

- Création du bord extérieur de la carte

```
create table frame as (
    select st_exteriorRing(st_setSRID(st_extent(geom)::geometry, 2193))
    as geom
from contour);
```



# Fermeture de courbes

- Fusion des lignes
  - Au sein d'une Multilinestring
  - Puis groupée par altitude (pour éviter les GeometryCollections)

```
CREATE TABLE contour_t1 AS (
    WITH dmp AS (
        SELECT c.gid, c.elevint, (st_dump(st_lineMerge(c.geom))).geom AS geom
        FROM contour c
    ), merge AS (
        SELECT d.elevint, st_lineMerge(st_collect(d.geom)) AS geom
        FROM dmp d
        GROUP BY d.elevint
    ) SELECT m.elevint, (st_dump(m.geom)).path[1] as path,
        (st_dump(m.geom)).geom
        FROM merge m
);
```

# Fermeture de courbes

- Segments les plus proches de chaque ligne

```
create table closest as (
  with tmp as (
    select s1.elevint, s1.gid, s1.path as path, s2.gid as closest,
    s2.geom, st_distance(st_collect(st_startpoint(s1.geom),
    st_endpoint(s1.geom)), st_collect(st_startpoint(s2.geom),
    st_endpoint(s2.geom))) as dist,
    row_number() over (partition by s1.gid order by
    st_distance(st_collect(st_startpoint(s1.geom), st_endpoint(s1.geom)),
    st_collect(st_startpoint(s2.geom), st_endpoint(s2.geom)))) as r
  from contour_t1 s1, contour_t1 s2
    where s1.elevint = s2.elevint
  and s1.gid <> s2.gid
) select distinct * from tmp
  where r < 3 and dist < 100
);
```

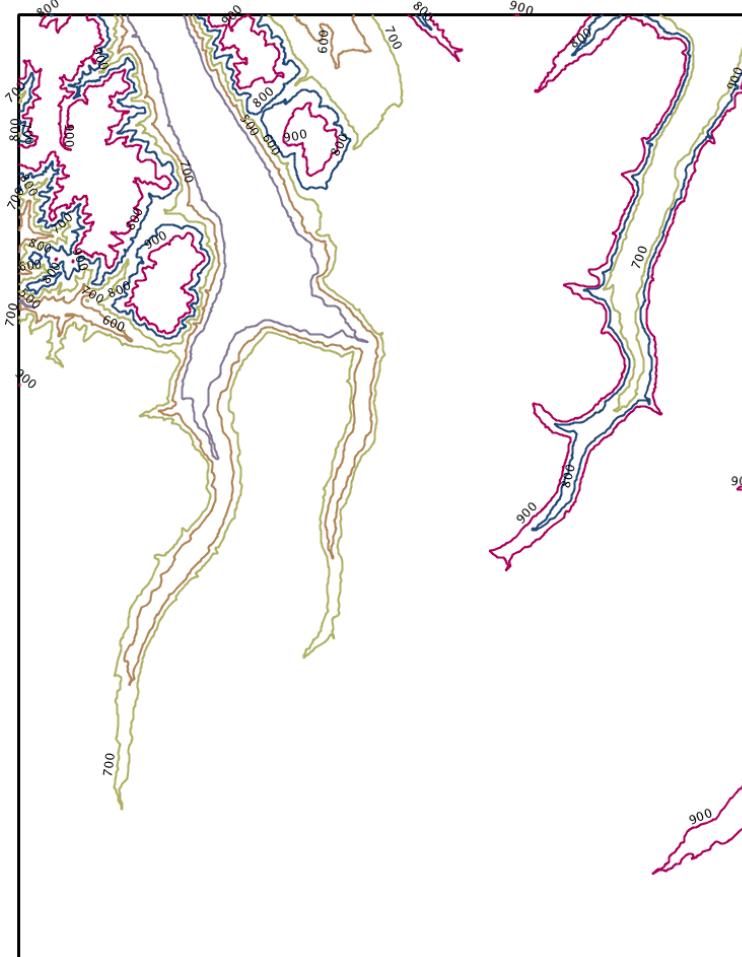
# Fermeture de courbes

- Comblement des trous entre courbes par  
`st_makeShortestLine`

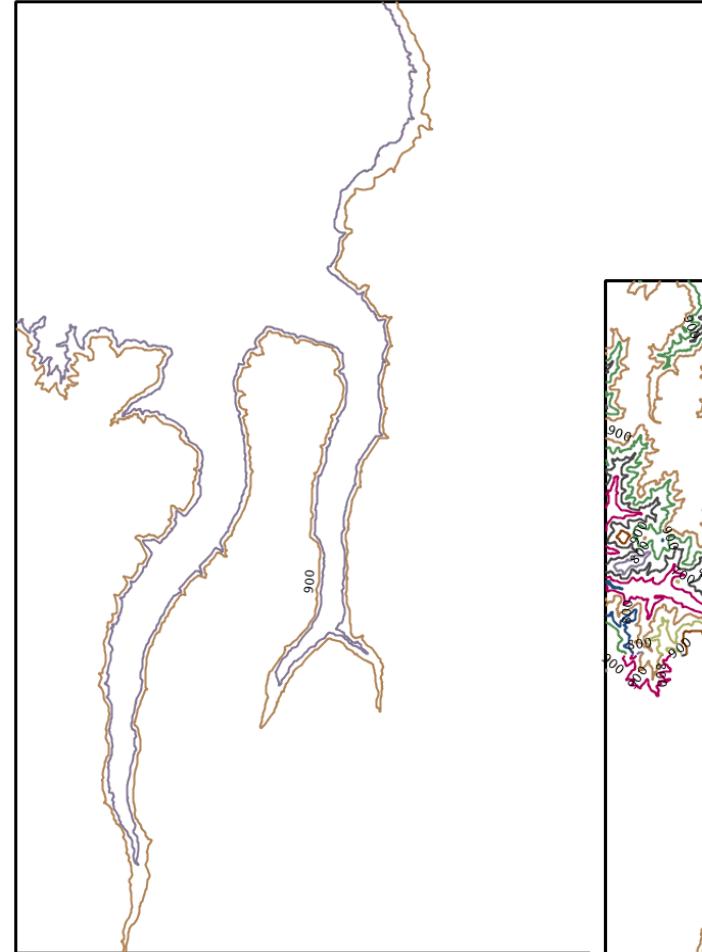
```
create table contour_t3 as (
with recursive tab as (
    select s.gid, s.elevint, t.closest, array[s.gid, t.closest] as gids,
    st_makeShortestLine(s.geom, t.geom) as geom, 1 as rank
    from contour_t1 s, closest t, frame f
    where s.gid = t.gid and s.elevint = t.elevint
        and (st_dwithin(st_startpoint(s.geom), f.geom, 0.001)
        or st_dwithin(st_endpoint(s.geom), f.geom, 0.001))
UNION ALL
    select tab.gid, tab.elevint, c.closest, gids || c.closest,
           st_makeShortestLine(tab.geom, c.geom) as geom, rank+1
    from closest c, tab
    where c.elevint = tab.elevint and c.gid = tab.closest
        and not (c.closest = any(gids)))
) select distinct on (t.geom) t.geom, t.elevint, t.rank, t.gids, t.gid
from tab t, frame f
where st_dwithin(st_startpoint(t.geom), f.geom, 0.001)
and st_dwithin(st_endpoint(t.geom), f.geom, 0.001)
```



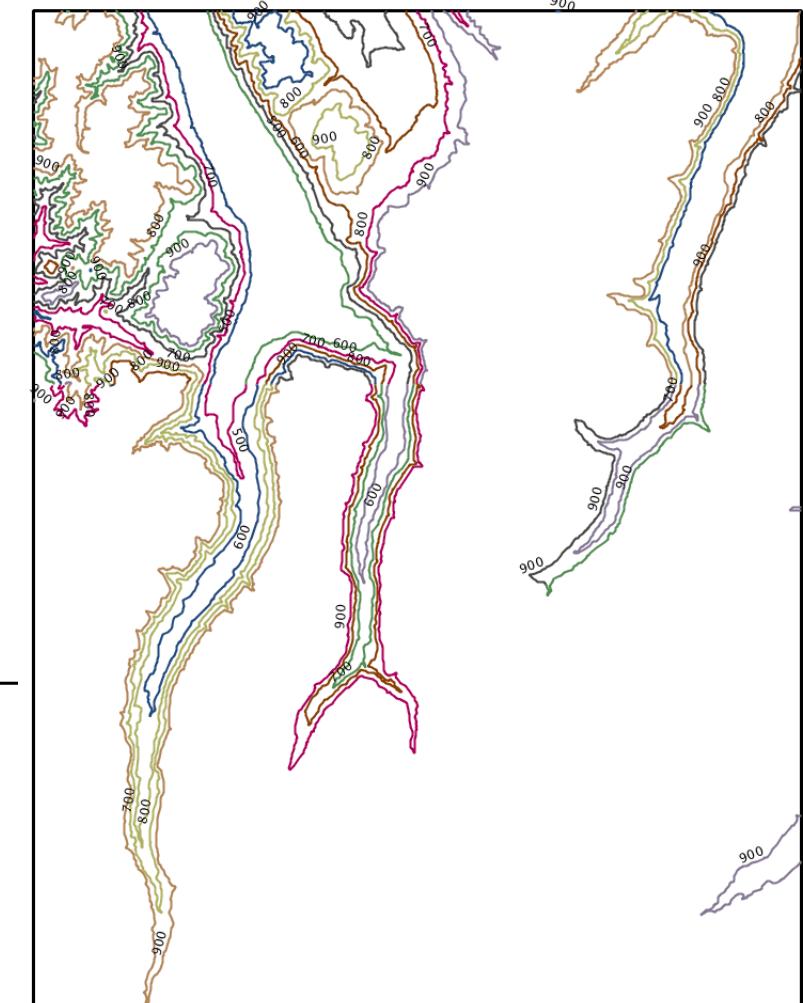
# Fermeture de courbes



contour\_t2



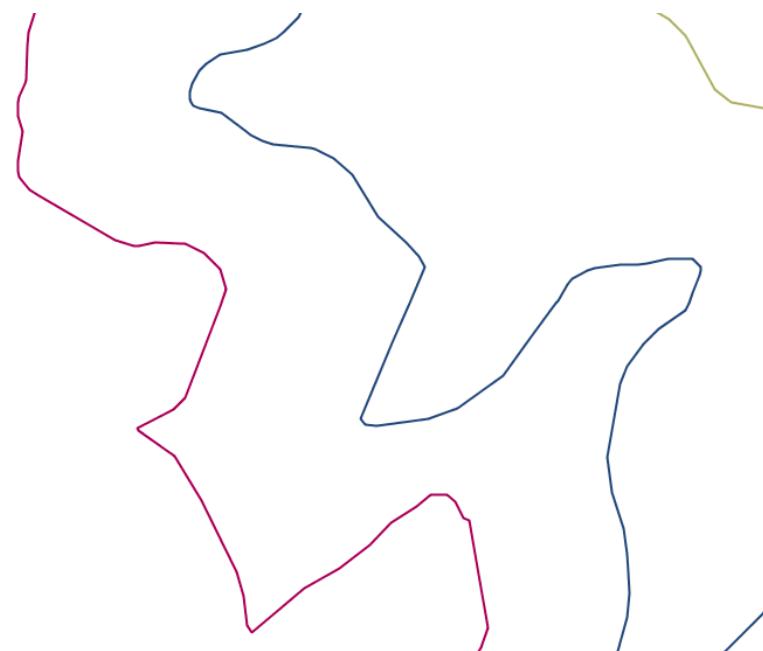
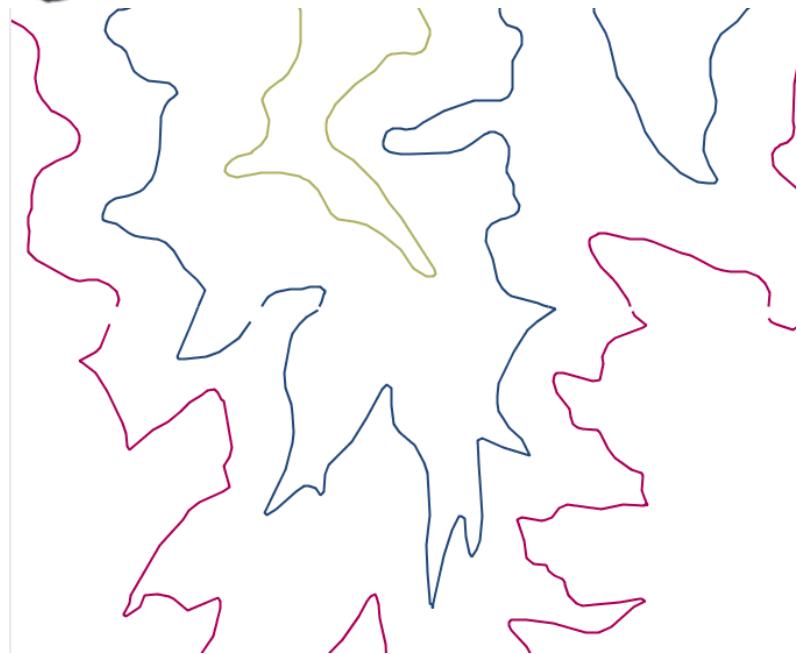
contour\_t3



new\_contour



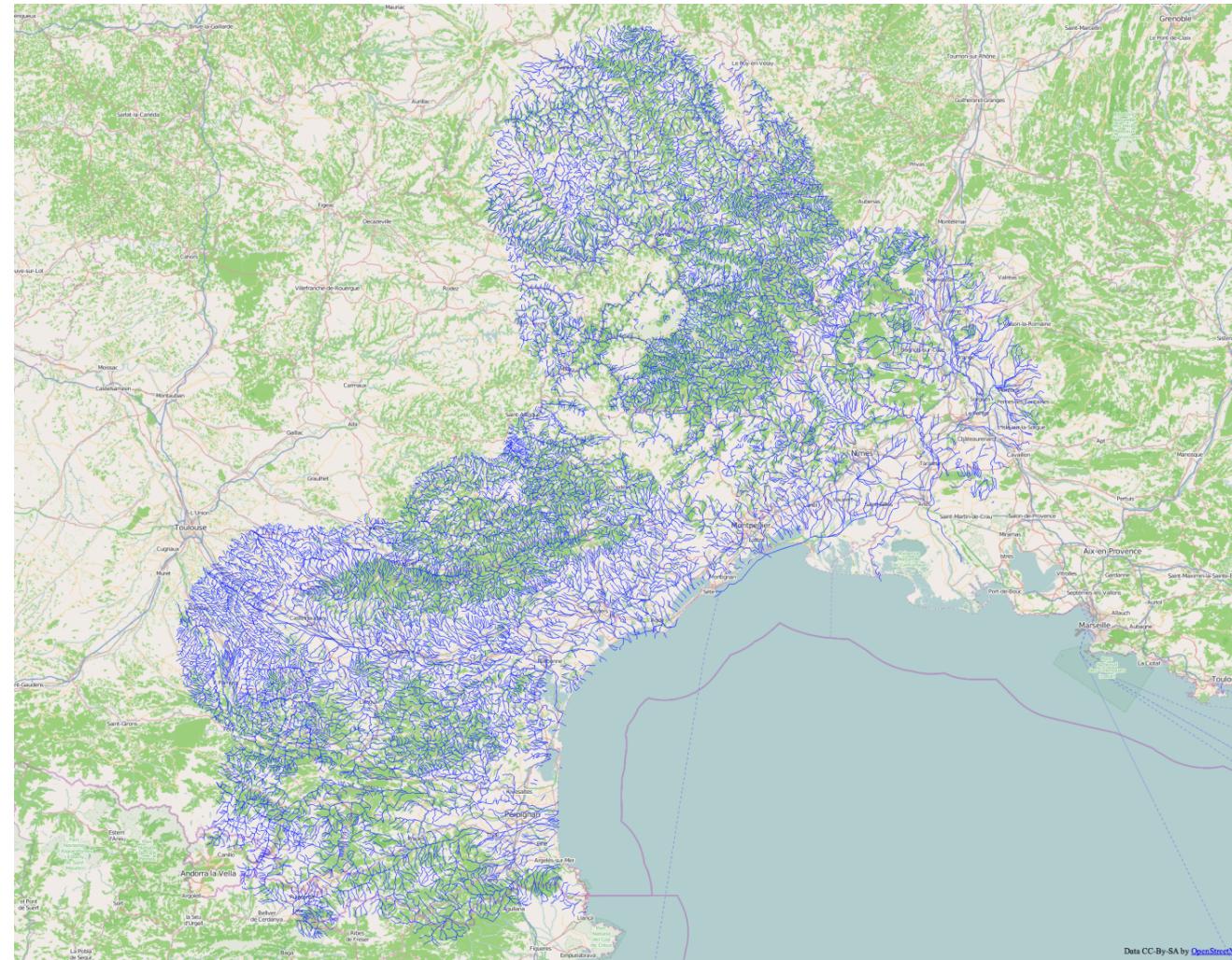
## Fermeture de courbes





# Parcours de graphe

- Itération avec WITH RECURSIVE : parcours d'un réseau hydrographique connecté

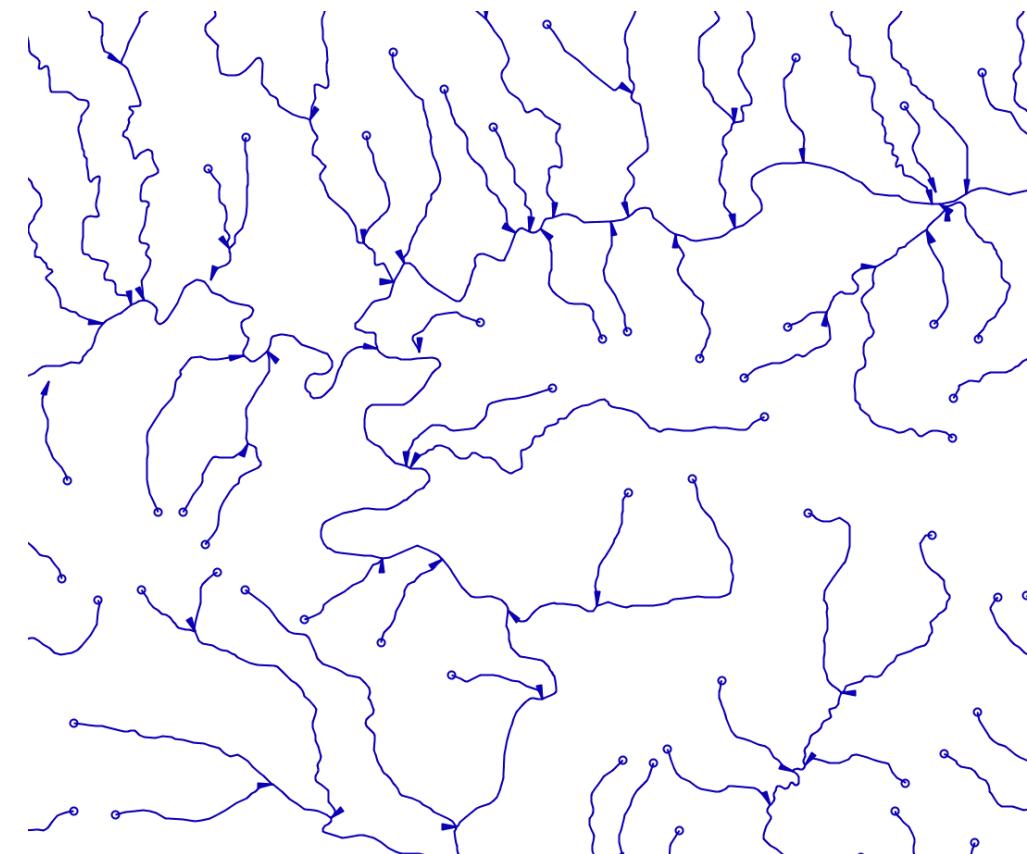




# Parcours de graphe

- Données IGN BD CARTHAGE®
- LINESTRING
- Tronçons de cours d'eau connectés et orientés

hydro	
gid	int
code_hydro	varchar(8)
classe	varchar(1)
toponyme	varchar(127)
candidat	varchar(127)
geom	geometry



# Parcours de graphe

- But: trouver les affluents d'un cours d'eau donné
- Utilisation de WITH RECURSIVE
- Permet de réutiliser le résultat d'une requête

```
WITH RECURSIVE t(n) AS (
    -- terme non récursif
    VALUES (1)
    UNION ALL
    -- terme récursif
    SELECT n+1 FROM t WHERE n < 100
) SELECT sum(n) FROM t;
```

# Parcours de graphe

```
WITH RECURSIVE t(n) AS (
    -- terme non récursif
    VALUES (1)
    UNION ALL
    -- terme non récursif
    SELECT n+1 FROM t WHERE n < 100
) SELECT sum(n) FROM t;
```

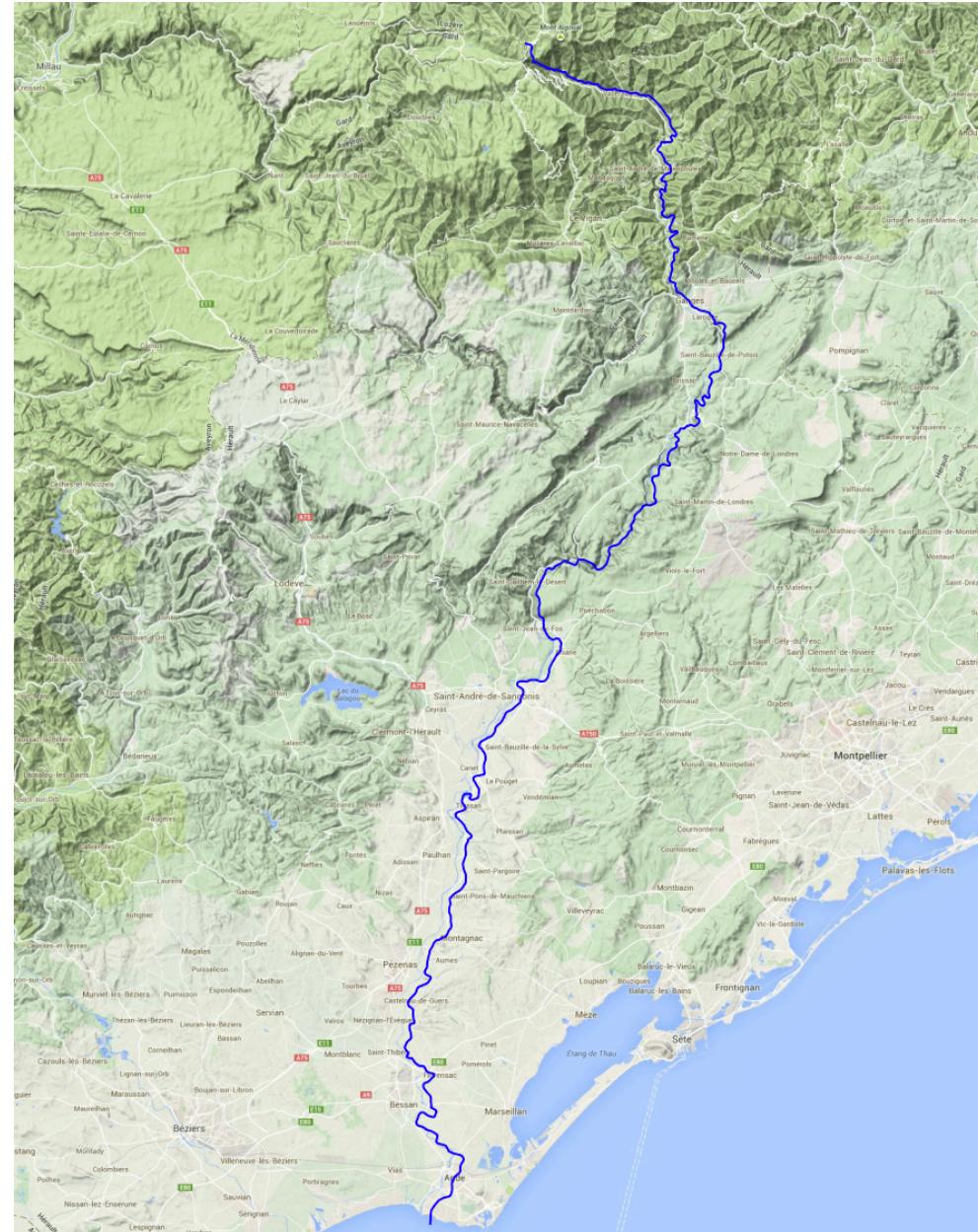
- Evaluation du terme non récursif
- Création d'une table de travail avec le résultat
- Tant que la table de travail n'est pas vide
  - Exécuter la partie récursive
  - Ajouter les résultats à la table de travail

# Parcours de graphe

```
WITH RECURSIVE troncon AS (
    -- partie non recursive: choix du troncon initial
    SELECT gid, ARRAY [gid] ids, geom, 1 AS iter
    FROM hydro
    WHERE gid = 8521
    UNION ALL
    -- partie recursive
    SELECT r.gid, ids || r.gid, r.geom, t.iter + 1
    FROM troncon t, hydro r
    WHERE NOT (r.gid = ANY (t.ids))
        -- condition sur l'affluent: son endpoint est connecté au segment en cours
        AND st_dwithin(st_endPoint(r.geom), t.geom, 1)
) SELECT t.gid, t.geom
FROM troncon t;
```



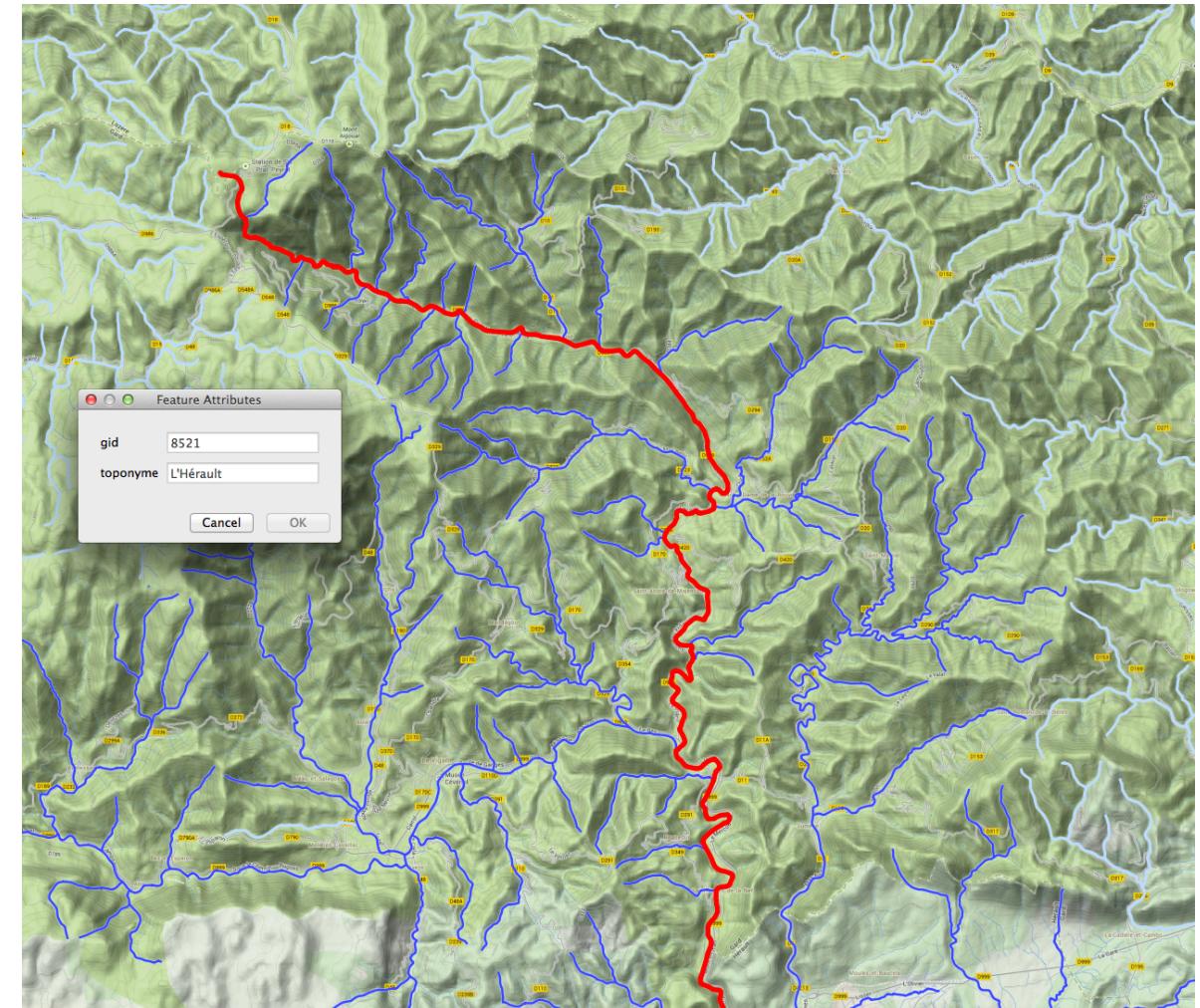
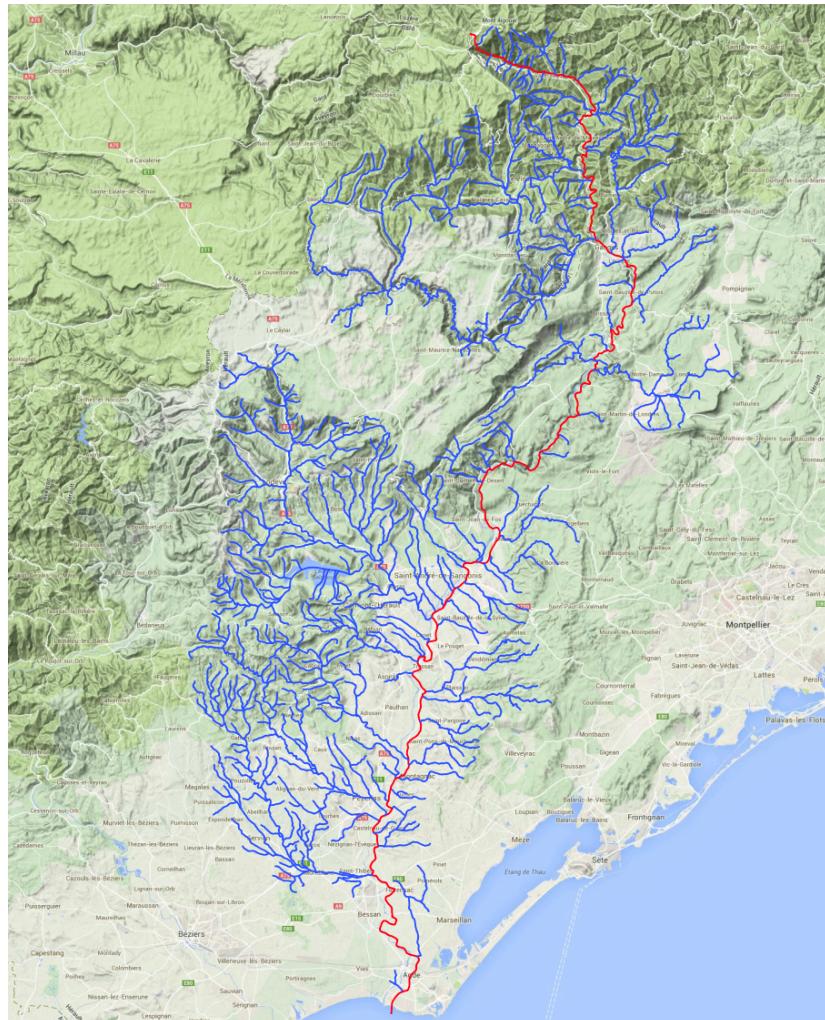
## Parcours de graphe





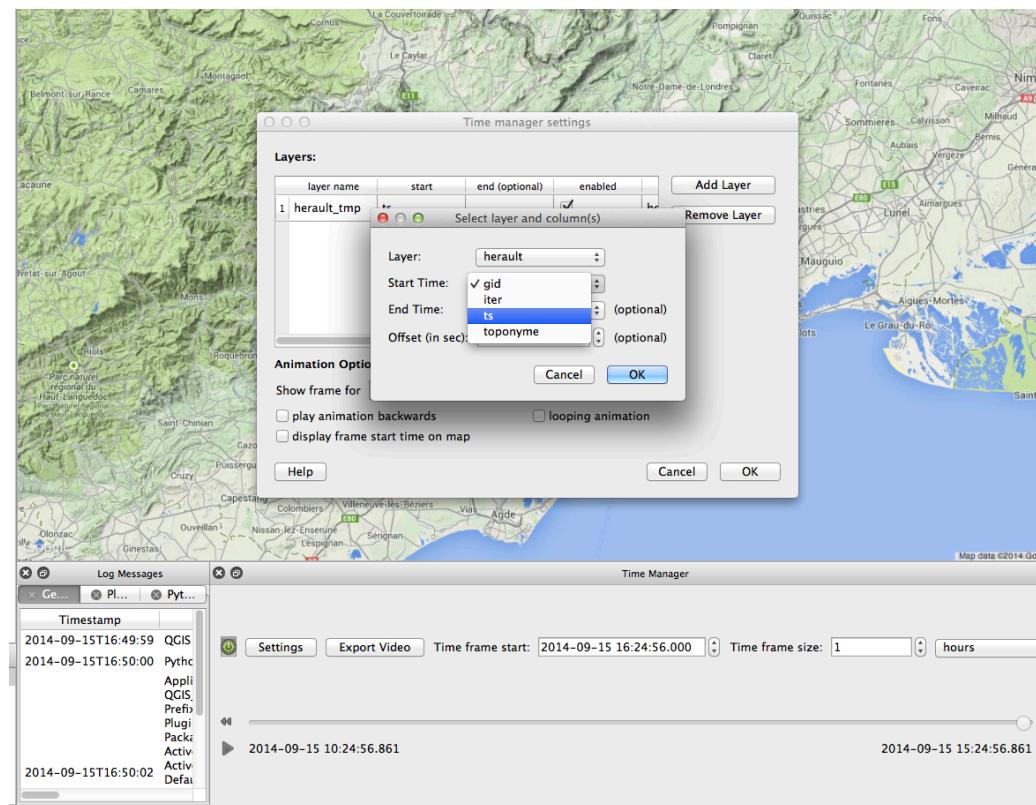
# Parcours de graphe

- Affluents de l'Hérault



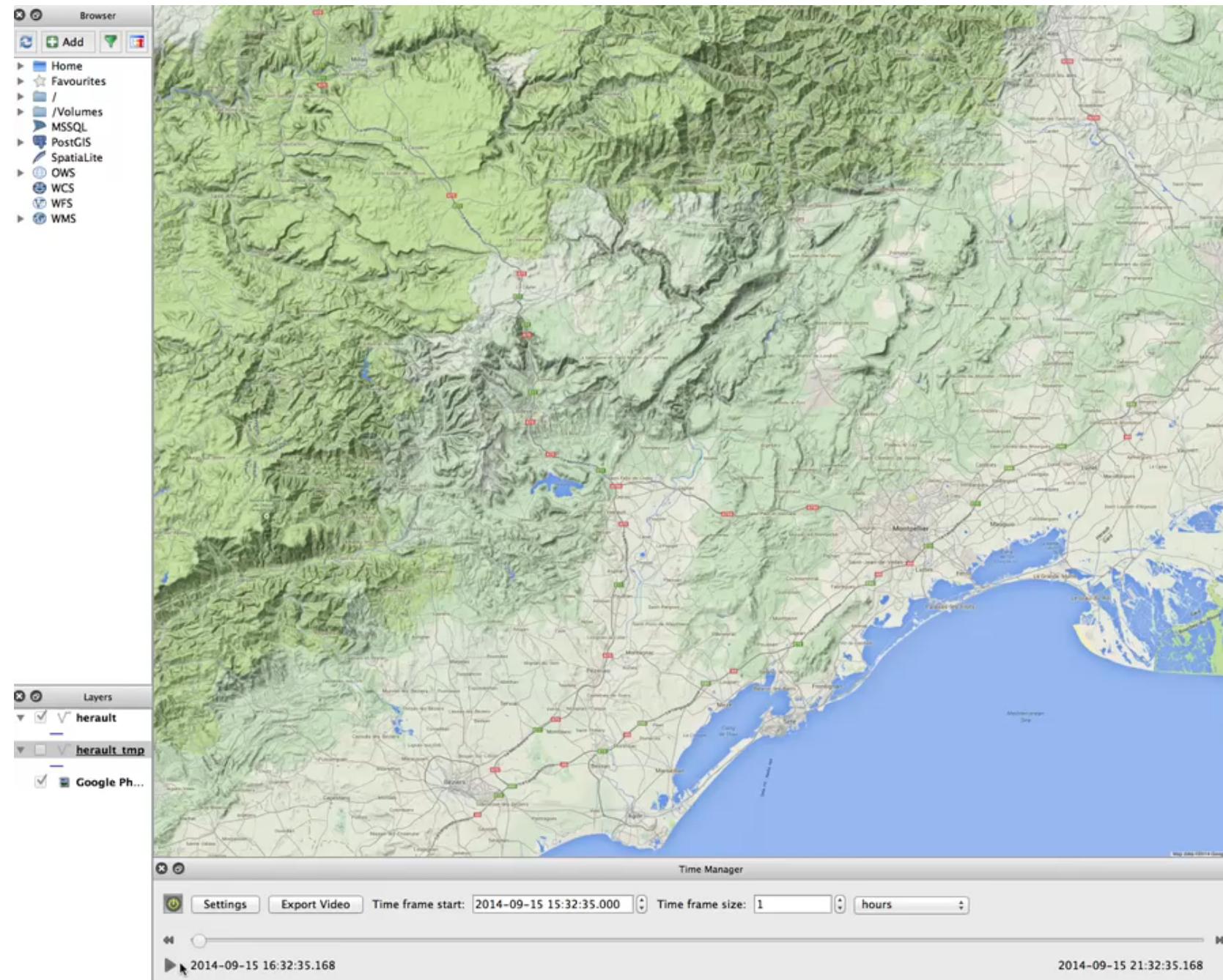
## Parcours de graphe

- Affichage temporel dans Qgis (Time Manager)
- Associer un timestamp à chaque étape du WITH RECURSIVE:
- `( INTERVAL '1 hour' * iter ) :: TIMESTAMP`
- Configurer la couche dans Time Manager





# Parcours de graphe





# Parcours de graphe

- <https://vimeo.com/107032218>

# Parcours de graphe

- Affichage cumulatif: conserver tous les tronçons identifiés aux étapes précédentes
- Duplication des lignes aux différentes étapes:

```
create table herault_tmp as (
  with tmp1 as (
    select max(iter)
    from herault
  ), tmp as (
    select distinct iter as m, max
    from herault, tmp1
    order by iter
  )select h.gid, h.iter, t.*,
  (now() - (INTERVAL '1 hour' * t.m))::TIMESTAMP as ts, h.geom
  from tmp t, herault h
  where t.m <= (t.max +1) - h.iter
);
```



# Parcours de graphe

Browser

- Home
- Favourites
- /
- /Volumes
- MSSQL
- PostGIS
- SpatialLite
- OWS
- WCS
- WFS
- WMS

Layers

- herault
- herault\_tmp
- Google Ph...

Time Manager

Settings Export Video Time frame start: 2014-09-15 08:24:56.000 Time frame size: 1 hours

2014-09-15 10:24:56.861 2014-09-15 15:24:56.861

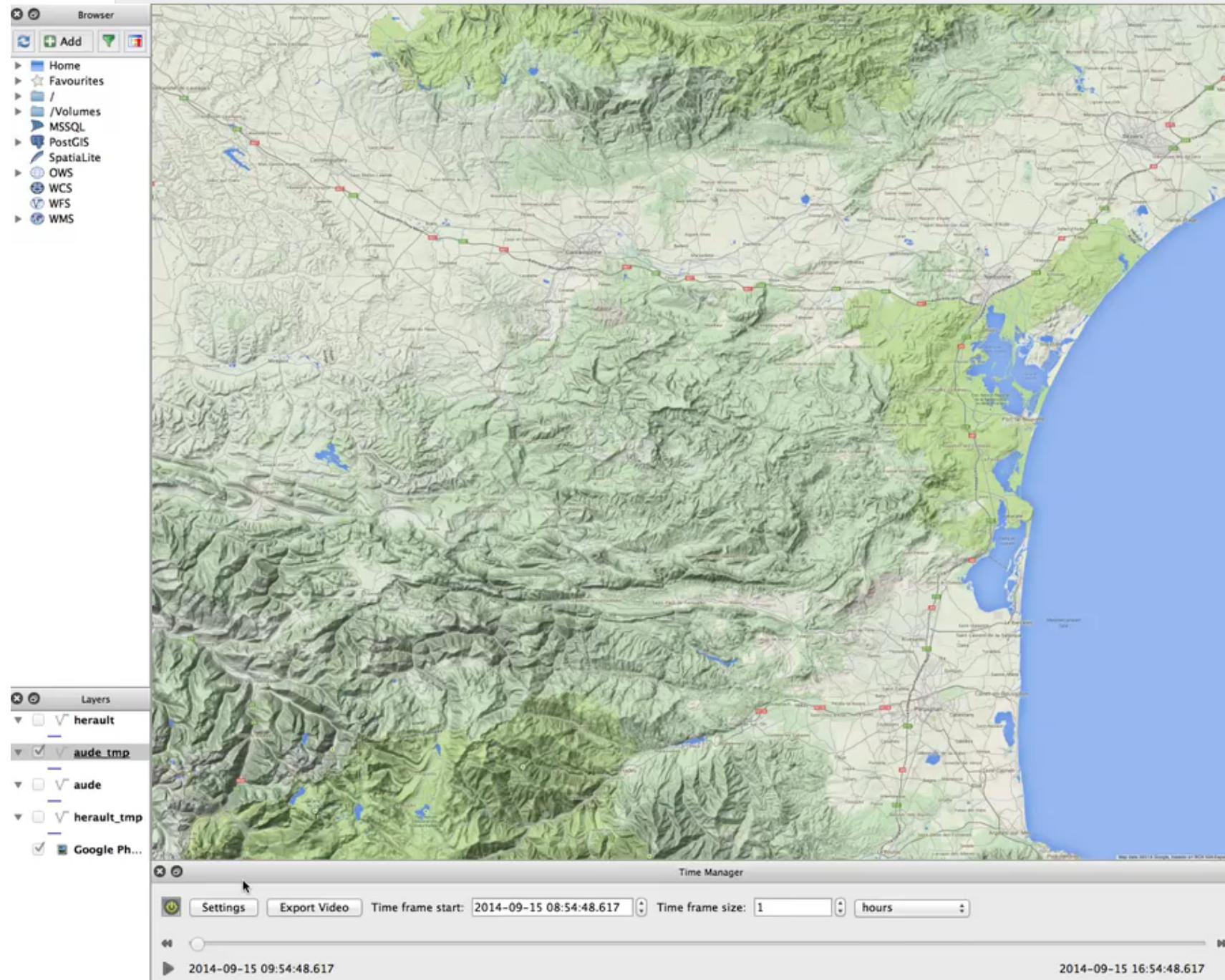


# Parcours de graphe

- <https://vimeo.com/107032219>



# Traitements avancés: ex 4





# Parcours de graphe

- <https://vimeo.com/107032217>



# Traitements avancés

Questions