PGDAY 2019

Partitionnions selon le couple latitude/longitude

cnes
CENTRE NATIONAL D'ÉTUDES SPATIALES

Démonstrateur de partitionnement selon le couple lat/lon

- Le CNES en quelques mots
- L'algorithme healpix
- L'implémentation sous postgreSQL
- Les apports des différentes version

Démonstrateur de partitionnement selon le couple lat/lon

- Le CNES en quelques mots
- L'algorithme healpix
- L'implémentation sous postgreSQL
- Les apports des différentes version

Le CNES en 4 questions

Qui ? Agence spatiale française

Depuis quand ? Depuis 1961

Dans quel but ? Développer la politique et les activités spatiales de la France

Avec qui ? Les ministères de la Recherche, de la Défense et de l'Environnement, les autres agences spatiales dont l'ESA et la NASA, les industriels, les laboratoires...

CHIFFRES CLE DU CNES

2,400
employés

€2,4 B
budget

2ème
budget mondial

80%
revient à l'industrie

CST

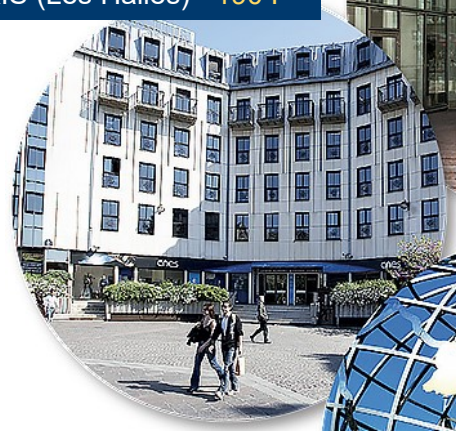
LES CENTRES DU CNES

4 centres / 2400 employés

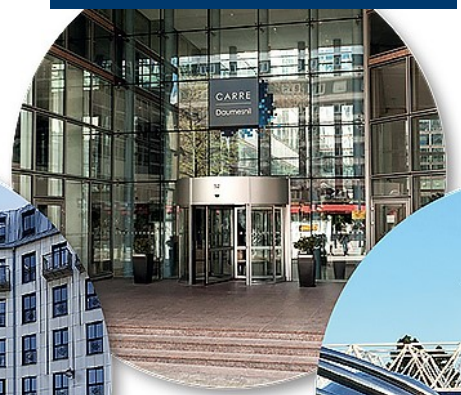
Siège

- Stratégie,
- Relations internationales,
- Administration

1 PARIS (Les Halles) - 190 P



2 PARIS (Daumesnil) - 210 P



Lanceurs

- Etude, conception, développement des systèmes de lancement (Ariane, Soyouz, Vega,)
- Préparation du futur

TOULOUSE - 1720 P 3

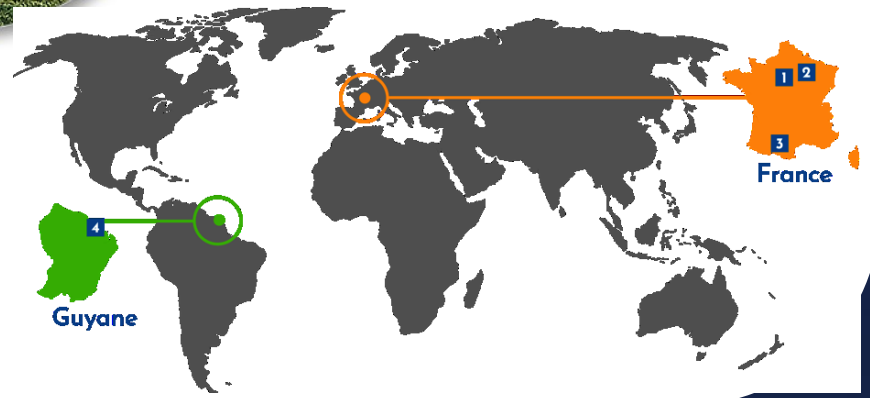
Systèmes Orbitaux

- Etude, conception, développement et contrôle des systèmes orbitaux
- Numérique et exploitation des données
- Préparation du futur
- Aire sur l'Adour: centre d'opérations de ballons

4 GUYANE– 280 P

Port spatial de l'Europe

- Ariane
- Soyouz
- Vega



CST

LES PARTENARIATS DU CNES

Un grand réseau de partenaires internationaux

Le CNES développe des accords de partenariat avec un grand nombre de pays non européens
(principaux acteurs de l'Espace et nouveaux arrivants)

CST

LES MISSIONS DU CNES

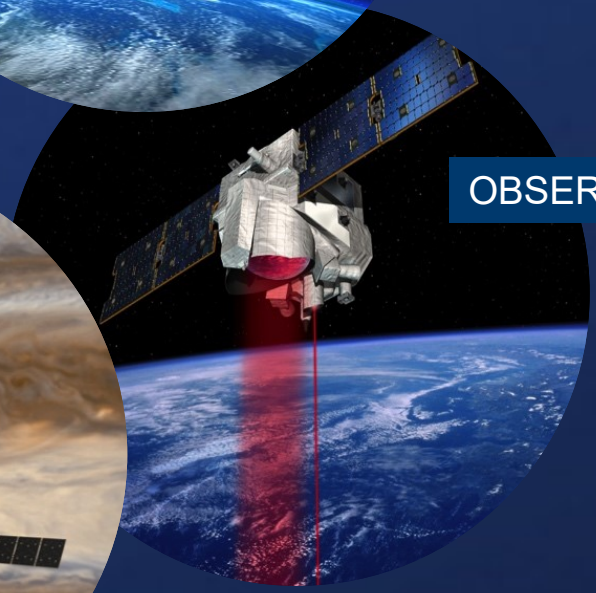
5 Domaines d'intervention

ARIANE

DEFENSE

TELECOMMUNICATIONS

SCIENCES



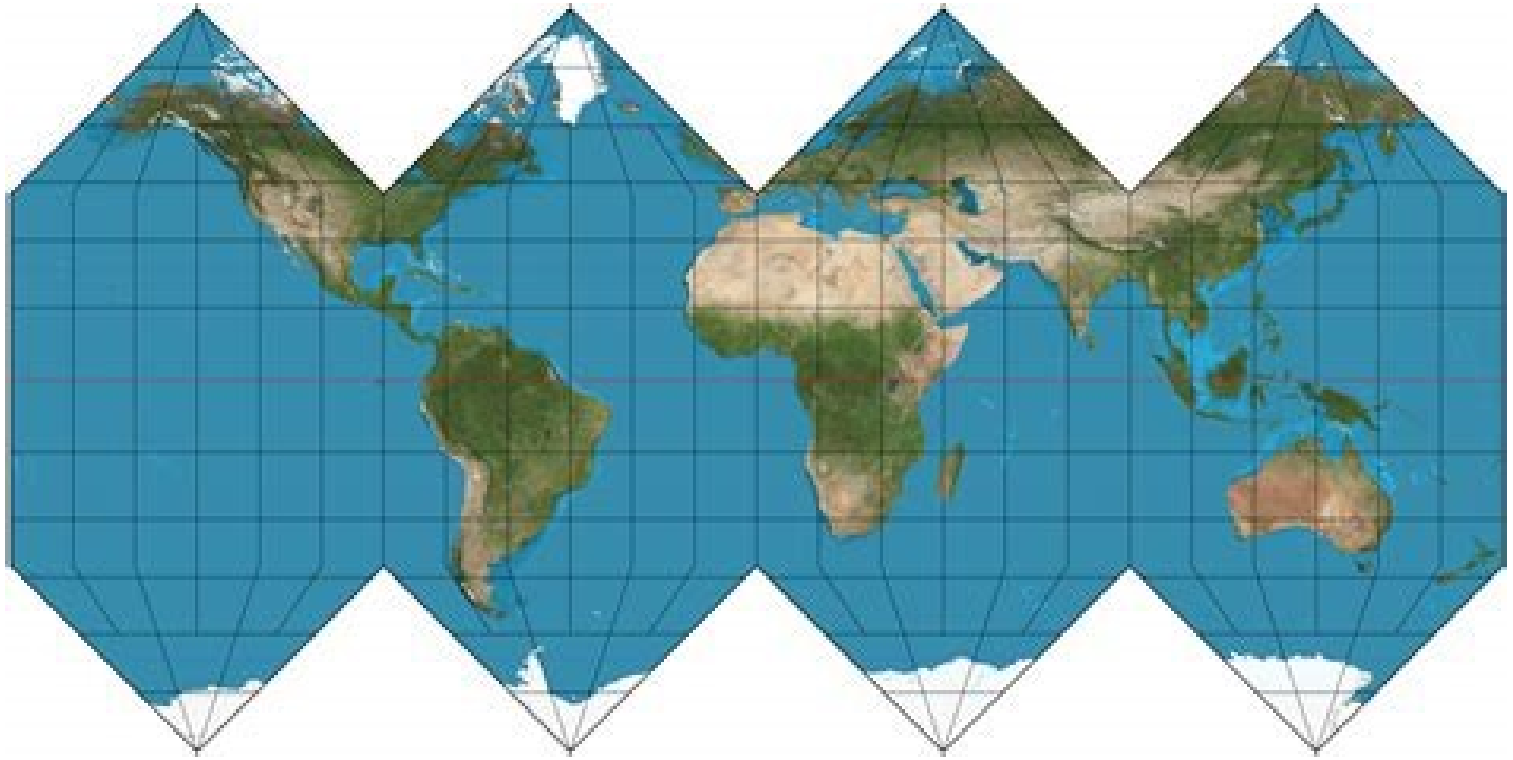
OBSERVATION

Démonstrateur de partitionnement selon le couple lat/lon

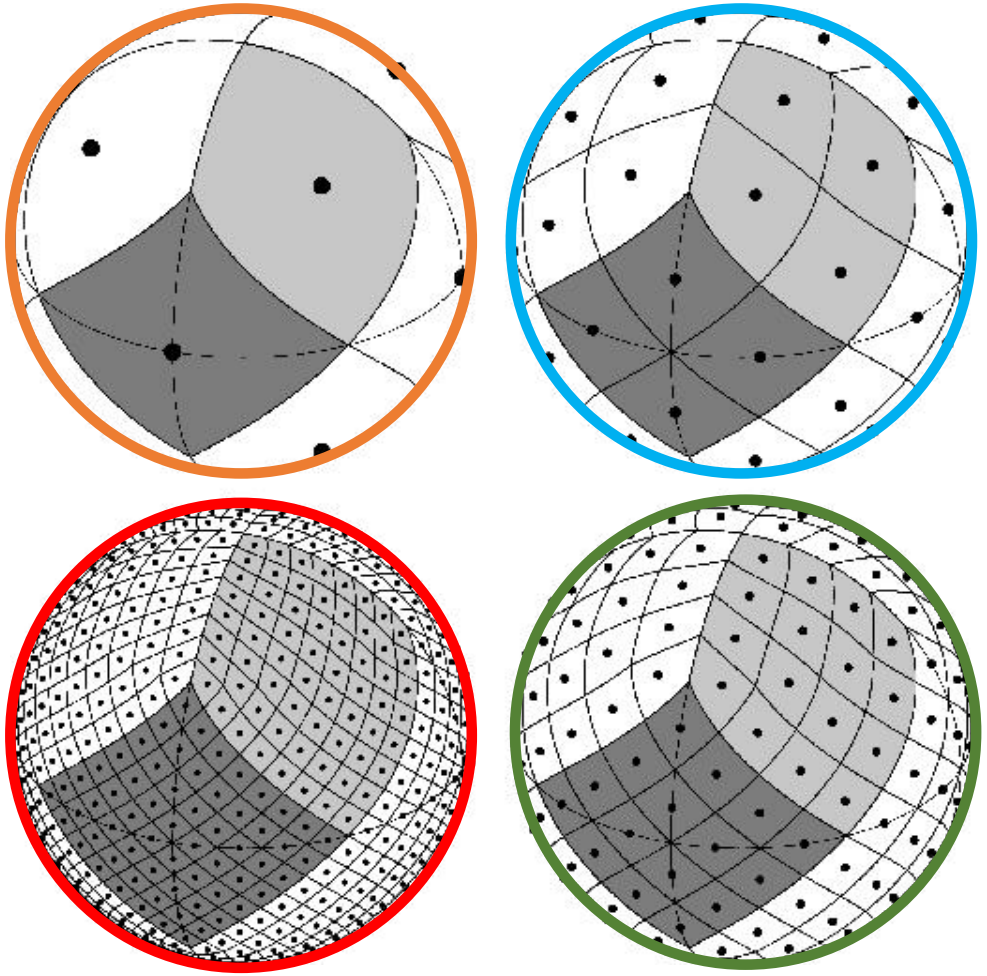
- Le CNES en quelques mots
- L'algorithme healpix
- L'implémentation sous postgreSQL
- Les apports des différentes version

Healpix

- Healpix est un algorithme permettant de découper une sphère en plusieurs pixels de surface identique et attribuant une valeur entière à chacun de ces pixels.



Healpix



N_{side}	N_{pix}
1	12
2	48
4	192
8	768
16	3072
32	12288
64	49152
128	196608
256	786432
512	3145728
1024	12582912
2048	50331648
4096	201326592

N_{side} est une puissance de 2

Le nombre de pixels est défini par la formule :

$$N_{pix} = 12 \times N_{side}^{2}$$

Démonstrateur de partitionnement selon le couple lat/lon

- Le CNES en quelques mots
- L'algorithme healpix
- L'implémentation sous postgreSQL
- Les apports des différentes version

Principe de fonctionnement

- Pour chaque point dans une table contenant un couple lat/lon, une colonne est ajoutée pour persister la valeur pixval (correspondant à la valeur de l'algorithme healpix) dudit point (via trigger).

- Lors d'une recherche de points contenus dans un polygone, l'ensemble des pixvals de ce polygone sont déterminés afin de ségréguer les partitions à prendre en compte.

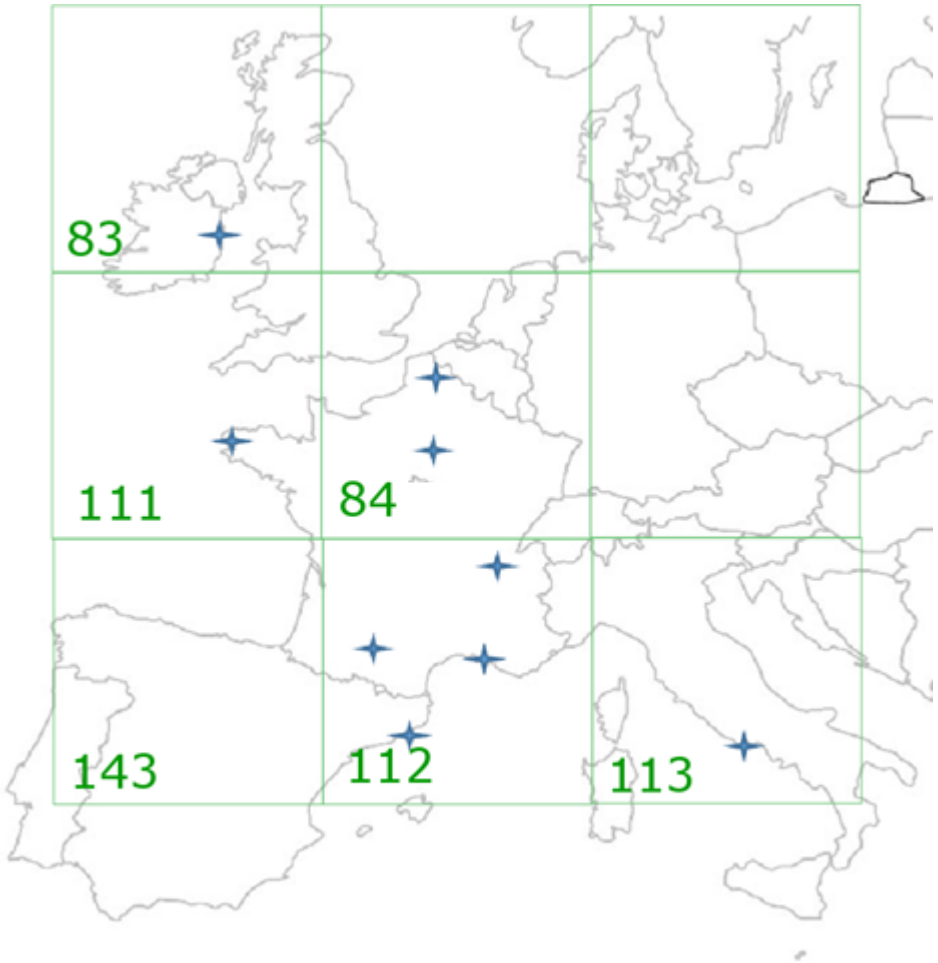
Fonction hp_ang2pix

- Fonction permettant d'associer la valeur pixval aux coordonnées en entrées (à appliquer et persister pour chaque ligne de la table)

Lyon : Pixval = 112

```
create or replace function hp_ang2pix(i_lat float, i_long float) returns bigint as $$
import numpy as np
import healpy as hp
return hp.ang2pix(8, 0.5 * np.pi - np.deg2rad(i_lat), np.deg2rad(i_long))
$$ language plpythonu IMMUTABLE;
```

Exemple simple : La table villes




```
postgres=# update villes set pixval = hp_ang2pix(lat, lon);

postgres=# select ville, lat, lon, pixval from villes ;

    ville   |    lat     |    lon     | pixval
 -----------+------------+------------+--------
  Brest     | 48.3905283 | -4.4860088 |  111
  Lyon      | 45.6963425 |   4.735948 |  112
  Toulouse  | 43.6044622 |  1.4442469 |  112
  Marseille | 43.2961743 |  5.3699525 |  112
  Lille     | 50.6305089 |  3.0706414 |   84
  Barcelone | 41.3828939 |  2.1774322 |  112
  Dublin    | 53.3497645 | -6.2602732 |   83
  Romes     |  40.521874 | 19.6685638 |  113
  Paris     |  48.862725 |  2.2875920 |   84
```

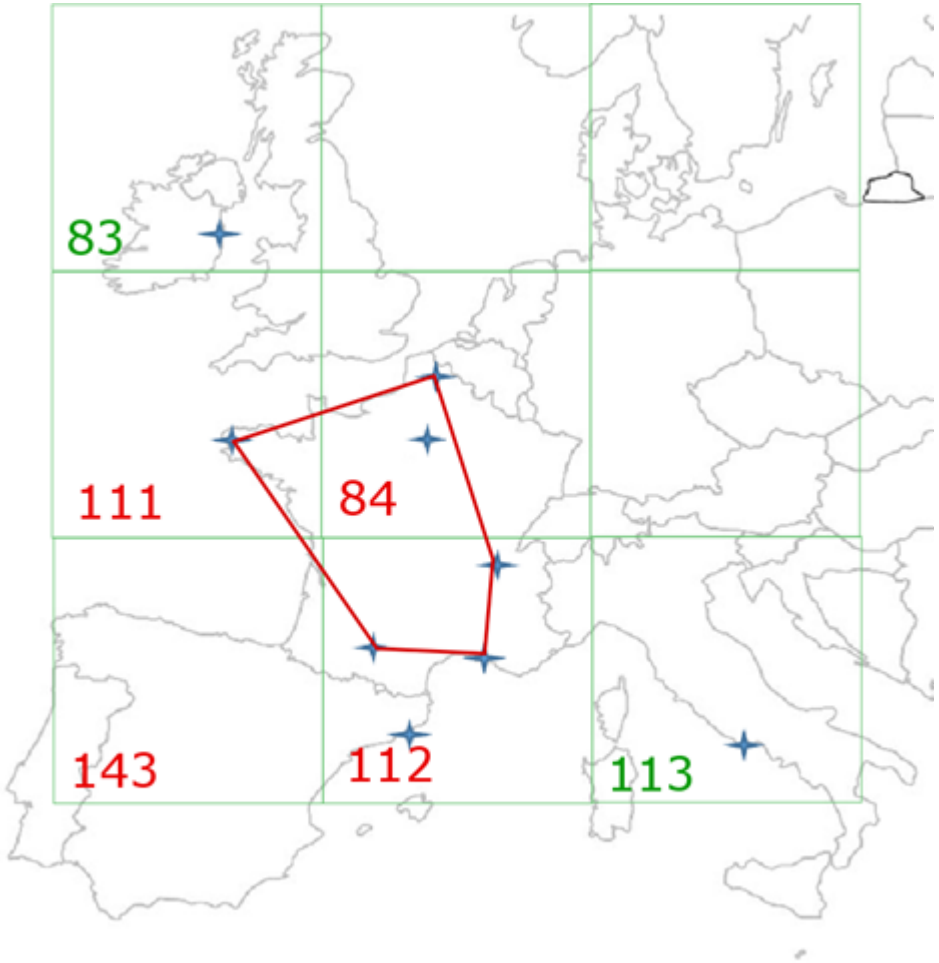
Fonction hp_query_polygon_agg

- Fonction permettant de trouver l'ensemble des pixval pour un polygone postgis donnée.

France : Pixval = {84,111,112,143}

```
create or replace function hp_query_polygon_agg(polygon text) returns bigint[] as $$
            # calculs de transformation d'angles (une dizaine de lignes)
            ...
return hp.query_polygon(8, np.array(pt_cart), inclusive=True, fact=64)
except:
    plpy.warning("Error on healpy module, return all healpix possible values.")
    return np.asarray(range(769))
$$ language plpythonu IMMUTABLE;
```

- Trouver les pixels appartenant à un polygone :



Brest – Lille – Lyon – Marseille – Toulouse – Brest

```
select hp_query_polygon_agg('POLYGON((48.3905283 -
4.4860088, 50.6305089 3.0706414, 45.6963425
4.735948,43.2961743 5.3699525, 43.6044622
1.4442469,48.3905283 -4.4860088))');

  hp_query_polygon_agg
-----------------------
  {84,111,112,143}
```

- Trouver le pixel correspondant à un point de type geometry (postgis):

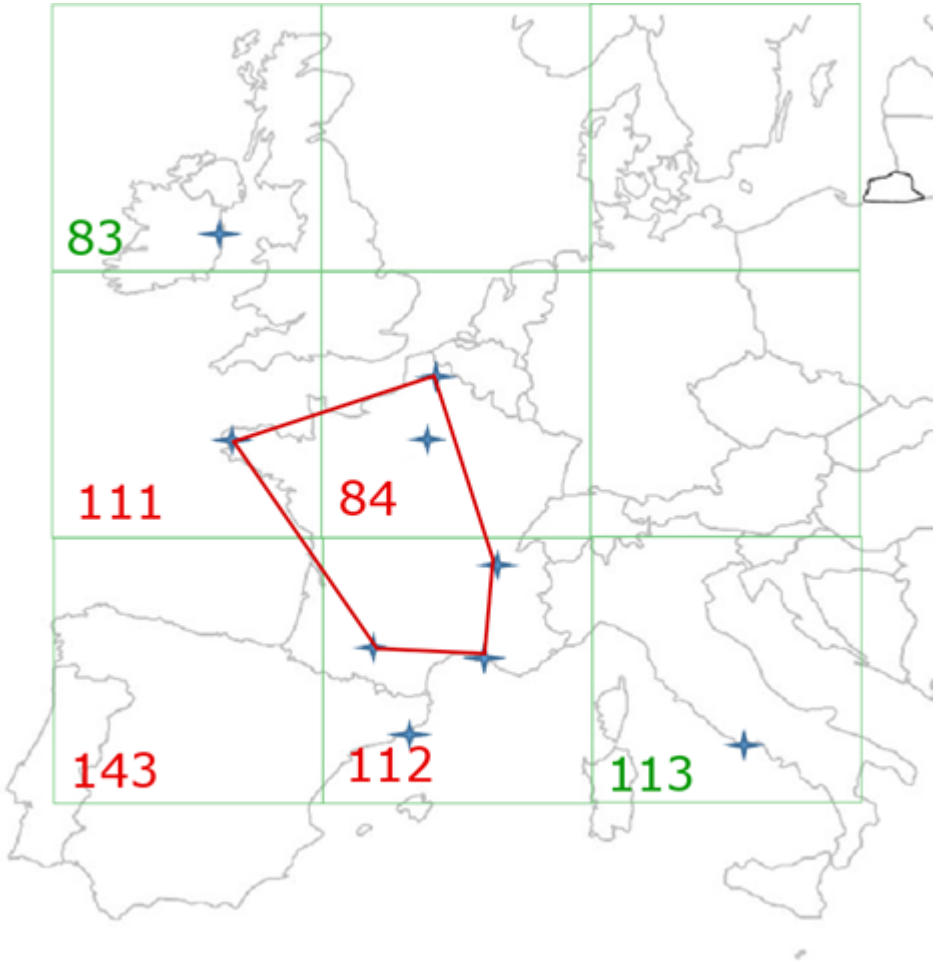
```
postgres=# SELECT ST_MakePoint(48.3905283,-4.4860088);

st_makepoint
--------------------------------------------
  0101000000CC54D2D4FC314840D745764AACF111C0
(1 row)

postgres=# select
hp_ang2pix(st_x('0101000000CC54D2D4FC314840D745764AACF111C0'::geometry),
st_y('0101000000CC54D2D4FC314840D745764AACF111C0'::geometry));

  hp_ang2pix
------------
         111
(1 row)
```

- Trouver les villes contenus dans les pixels correspondant à un polygone :




```
postgres=# select ville from villes where
hp_ang2pix(lat, lon) =
any(hp_query_polygon_agg('POLYGON((48.3905283 -
4.4860088, 50.6305089 3.0706414, 45.6963425
4.735948,43.2961743 5.3699525, 43.6044622
1.4442469,48.3905283 -4.4860088))') );

    ville
-----------
  Lyon
  Toulouse
  Marseille
  Lille
  Barcelone
  Paris
```

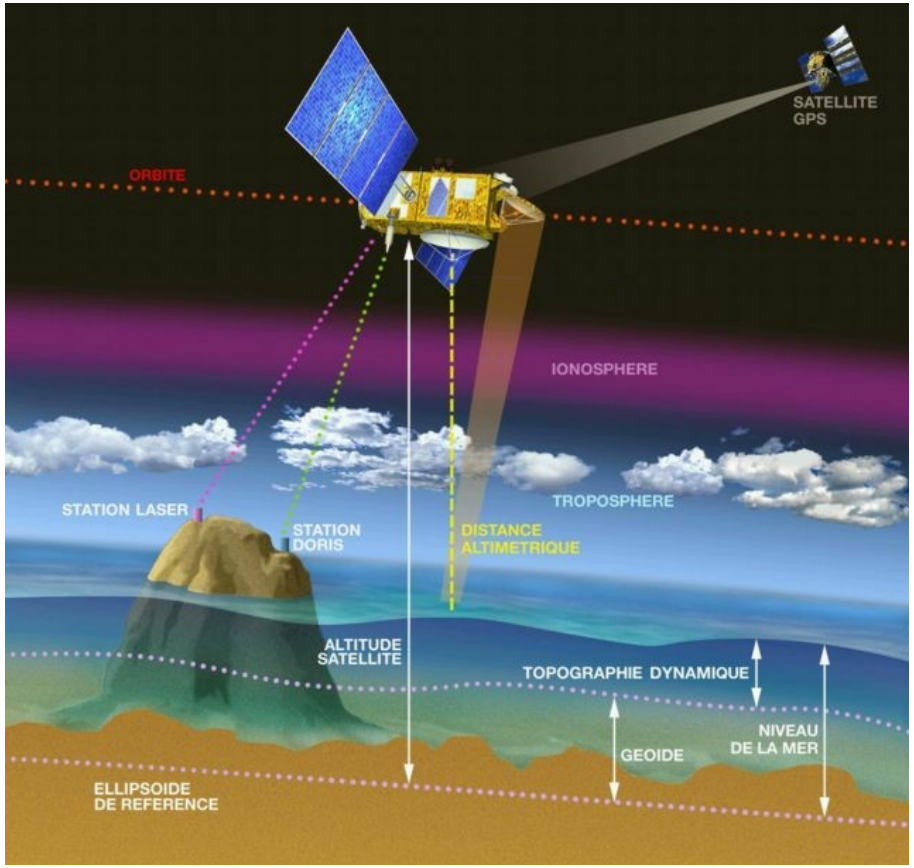
- Trouver les villes contenus dans un polygone :

```
postgres=# select ville from villes where hp_ang2pix(lat, lon) =
any(hp_query_polygon_agg('POLYGON((48.3905283 -4.4860088, 50.6305089
3.0706414, 45.6963425 4.735948,43.2961743 5.3699525, 43.6044622
1.4442469,48.3905283 -4.4860088))') )
and ST_setsrid(ST_MakePoint(lat, lon),4326) &&
ST_setsrid(ST_GeometryFromText('POLYGON((48.3905283 -4.4860088, 50.6305089
3.0706414, 45.6963425 4.735948,43.2961743 5.3699525, 43.6044622
1.4442469,48.3905283 -4.4860088))'), 4326) ;

    ville
-----------
  Lyon
  Toulouse
  Marseille
  Lille
  Paris
```

Exemple plus pratique : les mesures de capteurs

- Création du partitionnement :




```
postgres=# create table mesure (ts timestamp,
lat float, lon float, capteur varchar, val
double precision, pixval bigint) partition by
list (pixval);

for part in `seq 1 768 `
do
  psql -c "create table mesure_${part}
partition of mesure for values in ($part) ;"
done
```

- Insertion :

```
postgres=# insert into mesure values (now(), 48.3905283, -4.4860088,
'S1', 45.2, hp_ang2pix(48.3905283, -4.4860088));

postgres=# select lat, lon, capteur, val, pixval from mesure;
    lat     |    lon     | capteur | val  | pixval
------------+------------+---------+------+--------
 48.3905283 | -4.4860088 | S1      | 45.2 |    111
(1 row)

postgres=# select lat, lon, capteur, val, pixval from mesure_111;
    lat     |    lon     | capteur | val  | pixval
------------+------------+---------+------+--------
 48.3905283 | -4.4860088 | S1      | 45.2 |    111
(1 row)
```

- Dans un cas réel, on s'appuie bien souvent sur postgis et son type geometry ; le couple lat/lon est remplacé par une colonne de type geometry, c'est le trigger qui se charge d'extraire lat et lon pour calculer pixval

```
postgres # create table mesure_geom (ts timestamp, coord geometry, capteur varchar,
val double precision, pixval bigint) partition by list (pixval);

-- 8< --- Creation des partitions ---- 8< --

postgres # create index idx_coord on mesure_geom using gist (coord);

Postgres # \di
List of relations
 Schema |            Name             | Type  |     Owner     |      Table
--------+-----------------------------+-------+---------------+-----------------
 pgday  | mesure_geom_0_coord_idx     | index |  *********    | mesure_geom_0
 pgday  | mesure_geom_100_coord_idx   | index |  *********    | mesure_geom_100
 pgday  | mesure_geom_101_coord_idx   | index |  *********    | mesure_geom_101
 ---
 pgday  | mesure_geom_768_coord_idx   | index |  *********    | mesure_geom_768
 ---
 pgday  | mesure_geom_9_coord_idx     | index |  *********    | mesure_geom_9
```

Démonstrateur

- Jeu de test : 10 ans de données JASON dans une unique table
- Création de la structure
- Création d’une colonne pixval (valeur du healpix pour le couple lat/lon)
- Alimentation (en calculant pixval à postériori)
- Tests sur plusieurs versions de PG

- Exemple dans un contexte de forte volumétrie

Sur une table de 900 millions de lignes

```
postgres=# select count(*) from mesure_geom where coord &&
ST_GeometryFromText('POLYGON((0 0, 0 1,1 1, 2 1, 2 20, 1 3, 1 1, 10 0, 0 0))')
and pixval in (336,368,399,400);
  count
--------
  436650
(1 ligne)

Temps : 174,266 ms
```

Démonstrateur de partitionnement selon le couple lat/lon

- Le CNES en quelques mots
- L'algorithme healpix
- L'implémentation sous postgreSQL
- Les apports des différentes versions

Le dynamic pruning apporté en v11

- Le dynamic pruning apportée par la version 11, permet au moteur de construire son plan d'exécution en fonction du résultat d'une fonction (ici la fonction permettant le calcul des pixval).

```
postgres=# explain analyze select * from mesure_geom where pixval = any(hp_query_polygon_agg('POLYGON((45 1,55 2,45
3,40 4,45 1))') );
                                                    QUERY PLAN
-------------------------------------------------------------------------------------------------------------------
 Append  (cost=0.00..33.76 rows=9 width=850) (actual time=0.035..0.035 rows=0 loops=1)
   ->  Seq Scan on mesure_geom_12  (cost=0.00..11.24 rows=3 width=850) (actual time=0.025..0.025 rows=0 loops=1)
         Filter: (pixval = ANY ('{12,24,40}'::bigint[]))
   ->  Seq Scan on mesure_geom_24  (cost=0.00..11.24 rows=3 width=850) (actual time=0.004..0.004 rows=0 loops=1)
         Filter: (pixval = ANY ('{12,24,40}'::bigint[]))
   ->  Seq Scan on mesure_geom_40  (cost=0.00..11.24 rows=3 width=850) (actual time=0.005..0.005 rows=0 loops=1)
         Filter: (pixval = ANY ('{12,24,40}'::bigint[]))
 Planning Time: 216.087 ms
 Execution Time: 0.315 ms
(9 rows)

Time: 263,638 ms
```

⇨ On ne parcours que les trois partitions présentant potentiellement des données ; en v10, on aurait parcouru toutes les partitions

Amélioration apportée par postgreSQL v12

- Partionnement avec plus de 10000 partitions (Nside = 32)

- En V11

```
postgres=# explain analyze select * from mesure_geom where pixval = 39;
                                                        QUERY PLAN
---------------------------------------------------------------------------------------------------------------------------
 Append  (cost=0.00..11.13 rows=1 width=850) (actual time=0.022..0.022 rows=0 loops=1)
   ->  Seq Scan on mesure_geom_39  (cost=0.00..11.12 rows=1 width=850) (actual time=0.021..0.021 rows=0 loops=1)
         Filter: (pixval = 39)
 Planning Time: 1522.261 ms
 Execution Time: 0.310 ms
(5 lignes)
```

- En V12

```
postgres=# explain analyze select * from mesure_geom where pixval = 39;
                                                      QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------
 Seq Scan on mesure_geom_39  (cost=0.00..11.12 rows=1 width=850) (actual time=0.002..0.002 rows=0 loops=1)
   Filter: (pixval = 39)
 Planning Time: 0.641 ms
 Execution Time: 0.044 ms
(4 rows)
```

Conclusion

- Le principe de partitionnement repose sur une combinaison de 2 valeurs, l'algorithme healpix permet d'obtenir un integer pour un ensemble de points géométriquement proches.
- Le nombre de partitions à prendre en compte est important en fonction du besoin (recherche sur des zones bien limitées) car le partitionnement deviendra contre-productif si les recherches concernent un grand nombre de partitions.
- Ce démonstrateur a muri au fil des versions de postgreSQL car même si le principe n'a pas changé, avant la version 10, il n'apportait aucun gain.
- Un travail d'intégration avec le moteur postgreSQL pourrait, par exemple effectuer la création d'index multiples gist automatiquement basés sur ce mécanisme (qui dépasserait alors le partitionnement) afin d'en faciliter la maintenance.