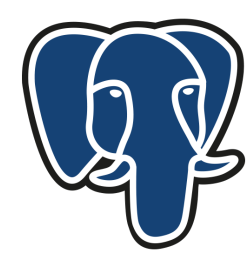
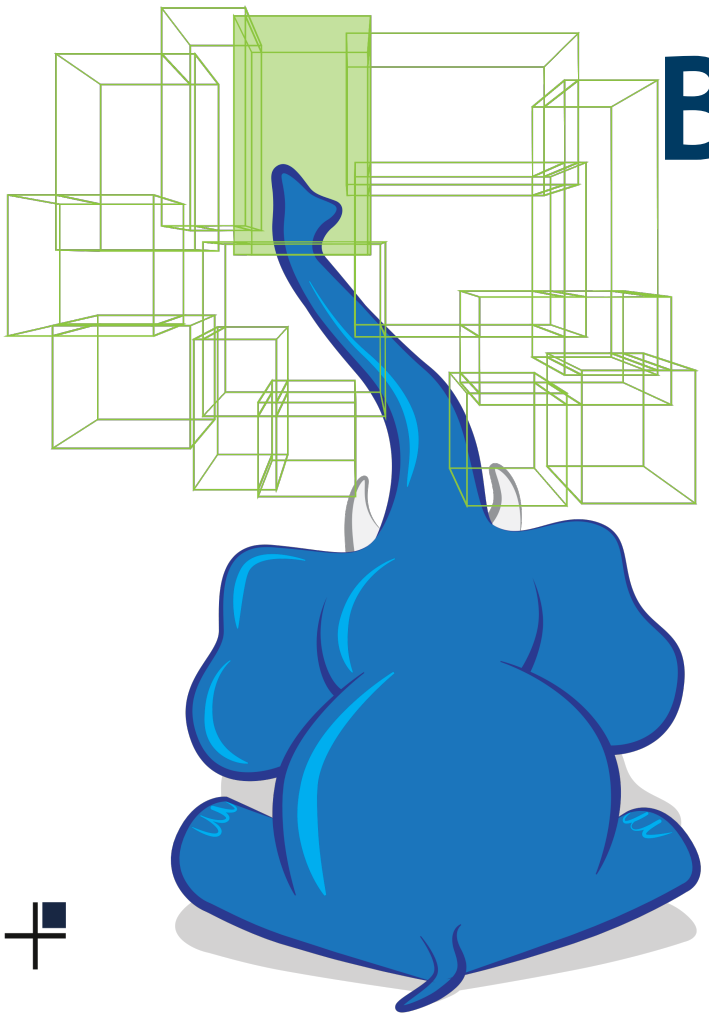


September, 22nd 2016
Lyon - France

POSTGRESQL SESSIONS - #8



BRIN indexes on geospatial databases

2ndQuadrant
Professional PostgreSQL

DALIBO
L'expertise PostgreSQL

J. Rouhaud - julien.rouhaud@dalibo.com
G. Broccolo - giuseppe.broccolo@2ndquadrant.it

2ndQuadrant
Professional PostgreSQL

DALIBO
L'expertise PostgreSQL

Who we are?

Giuseppe Broccolo

2ndQuadrant
Professional PostgreSQL

@giubro
gbroccolo
gbroccolo7
gemini___81

Julien Rouhaud

DALIBO
L'expertise PostgreSQL

@rjuju123
rjuju
rjuju

PostgreSQL index type

- Aka **Access Methods**
- Can add user defined new access methods
  - Fully supported since 9.6 (thanks to postgrespro)
    - CREATE ACCESS METHOD (avoir catalog update)
    - Generic WAL interface (crash safe)
- Some external access methods available
  - Bloom (postgrespro)
  - RUM (postgrespro)

Btree

- Most famous access method
- Balanced tree
- Keys are sorted
- Only handle “standard” operators (=, <, <=, >, >=)
- Index and Index Only Scan
- Unique constraints

Extend native access methods

- Access method use operator classes (opclass)
  - `CREATE INDEX idx_name USING method ON tbl (col opclass_name);`
- Define
  - operators for the needed types
  - support functions depending on the access method
- Can be extended by third part extensions

BRIN - Overview

- Block Range Index
  - S. Riggs, Á Herrera, H. Linnakangas in 9.5
- “Summarized” index
  - Split the table in **ranges**
    - A range is a set of pages (blocks)
    - By default 128 pages
    - Overloaded with the pages_per_range parameter
  - Really small index, faster to create

```
CREATE INDEX ON table USING BRIN (col) WITH (pages_per_range = 64)
```

BRIN - How data are stored

- For each **range**, computes a summary of the blocks
- Two kind of opclass:
  - **minmax** : for numerical values. Key values are added follwing sorting criteria + sorting operators
  - **inclusion** : for more exotic data. Key values are added following inclusion criteria + inclusion operators
- Extensibility
  - Provide support functions + operator

http://www.postgresql.org/docs/9.5/static/brin-extensibility.html

BRIN - internal overview

Index BRIN
pages_per_range = '16'
Correlation = 1

BRIN

Range	Min	Max	allnulls	hasnulls
0	1	3616	F	F
1	3617	7232	F	F
2	7233	10000	F	F

Heap	Table
Blk 0	1 2 ...
Blk 1	...
Blk 14	... 3295 ...
Blk 15	... 3615 3616
Blk 16	3617 ...
Blk 31	... 7232
Blk 32	7233 ...
Blk 43	... 9883 ...
Blk 44	... 10000

BRIN - Search the index

- Finding the matching rows
  - Scan the whole index for potentially matching ranges
  - Scan and recheck all these table blocks to keep only matching rows
- Can be seen as partial/enhanced sequential scan
  - Requires that rows are well distributed to avoid scanning too much table blocks
  - Useless on random data

BRIN - tips

- Tuning **`pages_per_range`**
  - Bigger value
  - Smaller index
  - More false positive
  - More table blocks to recheck
- Need to be tuned depending on queries
  - Less selective queries can usually afford bigger **`pages_per_range`**

BRIN - More tips

- New blocks (not in existing ranges) are not summarized
  - Perform **`VACUUM`**
  - Or call **`brin_summarize_new_value()`**
- Brin will never “shrinks” summarized data
  - If you update / delete boundary data, need to **`REINDEX`**

BRIN for PostGIS

- Original idea and POC by Giuseppe Broccolo
- C implementation and debugging in OSGEO Code Sprint in Paris
  - Bug in PostGIS fixed
  - Thanks a lot to Ronan Dunklau
- Code cleanup and some improvements since
- Committed in PostGIS repo on July, 31st (**`2b3c01b`**)

BRIN for PostGIS - 2

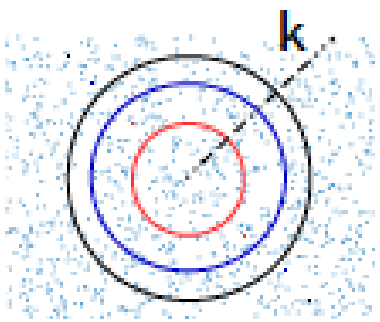
- Use PostGIS infrastructure for storing bounding box
  - Some helper functions and casts added
- 3 opclass
  - 2D (default), 3D, 4D **geometry**
  - geography
  - **box2d/box3d** (cross-operators defined in the opfamily)
- Storage datatype: float-precision (same as GiST)
  - gidx (3D, 4D **geometry**)
  - box2df (2D **geometry**, **geography**)

BRIN for PostGIS - 3

- Use BRIN inclusion infrastructure
  - **brin_inclusion_add_value()** has been redefined
  - Inclusion infrastructure doesn't handle different datatype
  - Redefining it allow some optimizations
- Operators: &&, @, ~ (2D), &&& (3D, 4D)
- No kNN support, since BRIN doesn't handle kNN

BRIN for PostGIS - What's next?

- Handle more operators for 3D and 4D (with SFCGAL lib)
- kNN? Would require new infrastructure in PostgreSQL




```
ORDER BY geoms <-> point
LIMIT N
```

get blocks gradually included within the k parameter, as already made in GiST with single tuples

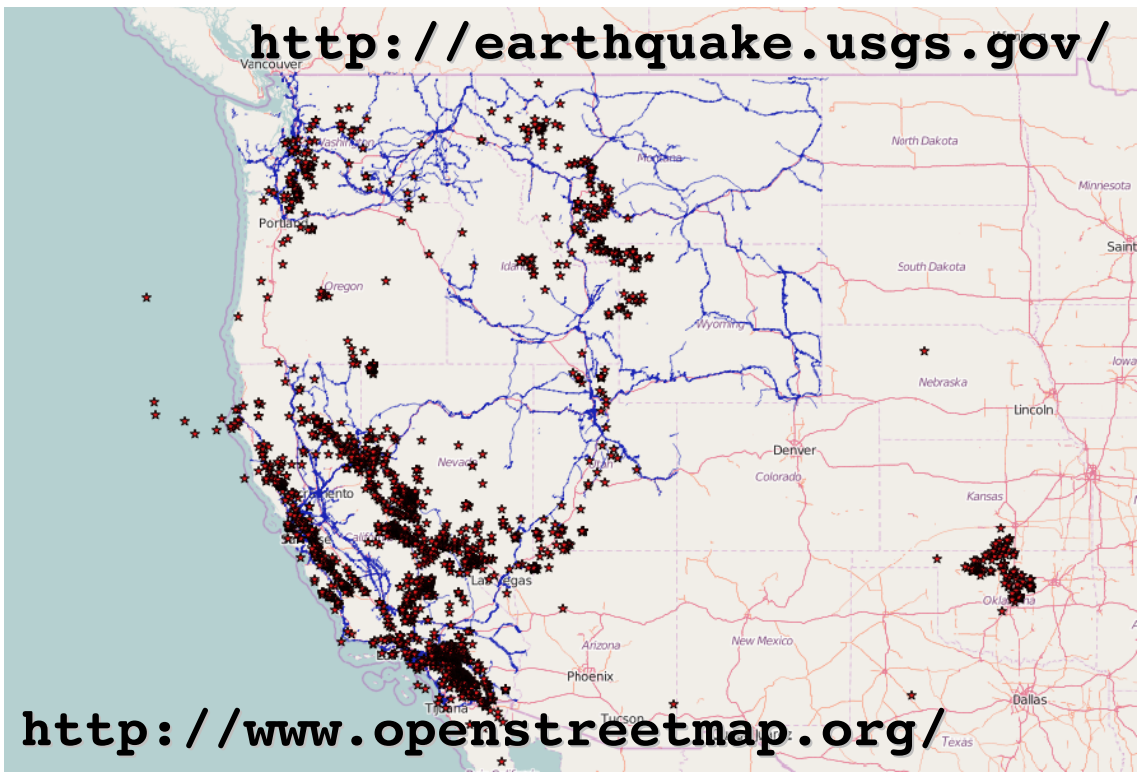
Benchmarks:

- OSM data
  - Speed up searches of lines intersercting the buffer of a set of points
- LiDAR dataset
  - Speed up searches of 3D points with X,Y coordinates included inside a 2D polygon

2ndQuadrant
Professional PostgreSQL

DALIBO
L'expertise PostgreSQL

Earthquakes in the USA in 2016



~100k lat/lon WGS84 points (earthquakes in the world)

~1M EPSG 102008 linestrings (railways in the west side)

Which railway was close (<10km) to an earthquake epicenter?

Earthquakes in the USA in 2016

GiST

```
=# CREATE INDEX idx_rl_gist ON westrailways USING gist
-# (ST_Buffer(GEOGRAPHY(ST_Transform(geom), 4326), 10000));

=# CREATE INDEX idx_eq_brin ON worldearthquakes USING gist (coord);
```

BRIN

```
=# CREATE INDEX idx_rl_brin ON westrailways USING brin
-# (ST_Buffer(GEOGRAPHY(ST_Trasform(geom), 4326), 10000))
-# WITH (pages_per_range=10);

=# CREATE INDEX idx_eq_brin ON worldearthquakes USING brin (coord)
-# WITH (pages_per_range=10);
```

BRIN/GiST Sizes	~1/100
BRIN/GiST Times	~1/200

Earthquakes in the USA in 2016

```
=# WITH us_eq AS (
-#     SELECT coord FROM world
-#     WHERE coord && 'BOX2D(-126.90 49.73, -65.83 24.73)'::box2d
-# ) SELECT * FROM railways r, us_eq u
-#   WHERE ST_Buffer(GEOGRAPHY(ST_Transform(geom), 4326), 10000) && u.coord;
```

without indexes: ~60s

with GiST: ~20ms

with BRIN: ~400ms

BRIN: 20X slower than GiST - 150X faster than full scan

2ndQuadrant
Professional PostgreSQL

DALIBO
L'expertise PostgreSQL

Relational approach to LiDAR data with PG

http://www.slideshare.net/GiuseppeBroccolo/gbroccolo-foss4-geugeodbindex

- PostgreSQL9.3 + **pg_pointcloud** + PostGIS
- GiST indexing in PostGIS, achieved performances:
  - RAM 16GB, 1 billion of points (~80GB)
  - index size ~O(table size)
  - Index was used:
    - up to ~300M points in bbox inclusion searches
    - up to ~10M points in kNN searches

LiDAR size: ~O($10^9 \div 10^{11}$) → few % can be properly indexed!

The LiDAR dataset: the ahn2 project

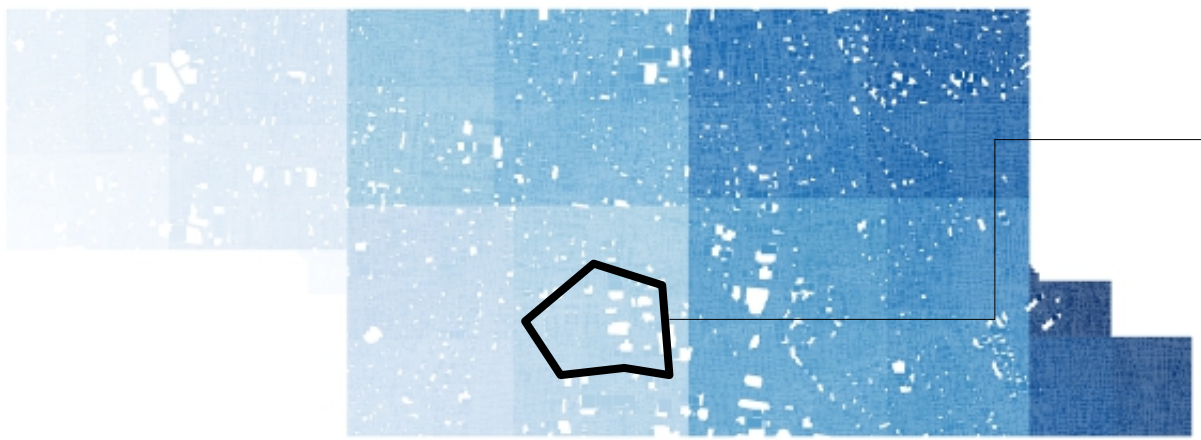
Geodan (thanks to Tom Van Tilburg)

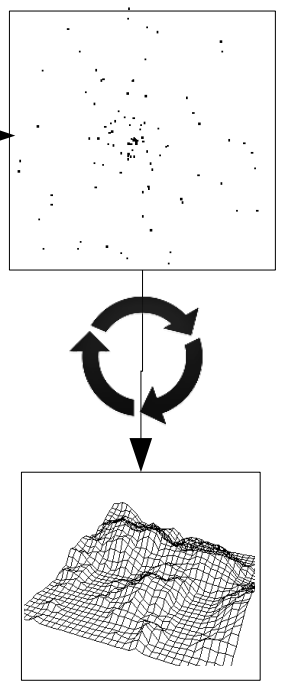
- 3D point cloud, coverage: almost the whole Netherlands
  - EPSG: 28992, ~8 points/m^2
- **1.6TB, ~250G points in ~560M patches (compression: ~10x)**
  - `.las` files imported through PDAL driver - `filter.chipper`
- available RAM: **16GB**
- Database based on `pg_pointcloud` extension
  - the point structure:

X	Y	Z	scan	LAS	time	RGB	chipper
32b	32b	32b	40b	16b	64b	48b	32b

the "indexed" part (can be converted to PostGIS datatype)

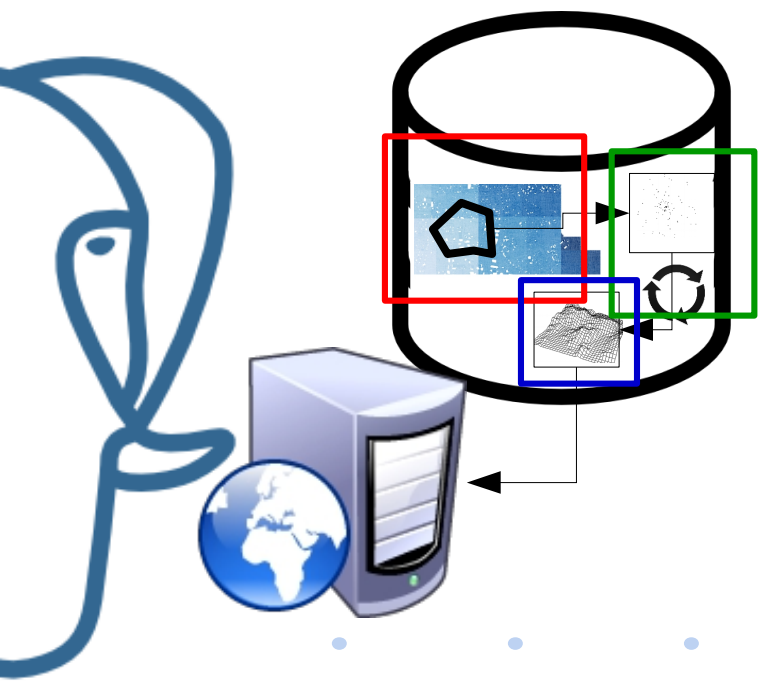
Typical searches on ahn2 - `x3d_viewer`





- Intersection with a polygon (*PostGIS*)
  - *the part that lasts longer - need indexes here!*
- Patch “explosion” + NN sorting (*pg_PointCloud+PostGIS*)
- *DEM: constrained* Delaunay Triang. (*SFCGAL*)

All in the DB & just with one query!



```
WITH patches AS (
 SELECT patches FROM ahn2
 WHERE patches && ST_GeomFromText('POLYGON(...)')
), points AS (
 SELECT ST_Explode(patches) AS points
 FROM patches
), sorted_points AS (
 SELECT points,
 ST_DumpPoints(ST_GeomFromText('POLYGON(...)'))).geom AS poly_pt
 FROM points ORDER BY points <#> poly_pt LIMIT 1;
), sel AS (
 SELECT points FROM sorted_points
 WHERE points && ST_GeomFromText('POLYGON(...)')
)
SELECT ST_Dump(ST_Triangulate2DZ(ST_Collect(points))) FROM sel;
```

patches && polygons - GiST performance

index building

GiST	2.5 days

index size

GiST	29GB

searches based on GiST

polygon size	timing
~O(10m)	~20ms
~O(100m)	~60ms
~O(1Km)	~3s
~O(10km)	hours

index not contained in RAM anymore

(~5G points → ~3%)

2ndQuadrant
Professional PostgreSQL

DALIBO
L'expertise PostgreSQL

patches && polygons - BRIN performance

index building

BRIN	4 h

index size

BRIN	15MB

searches based on BRIN

polygon size	timing
~O(m)	~150s

2ndQuadrant
Professional PostgreSQL

DALIBO
L'expertise PostgreSQL

patches && polygons - BRIN performance

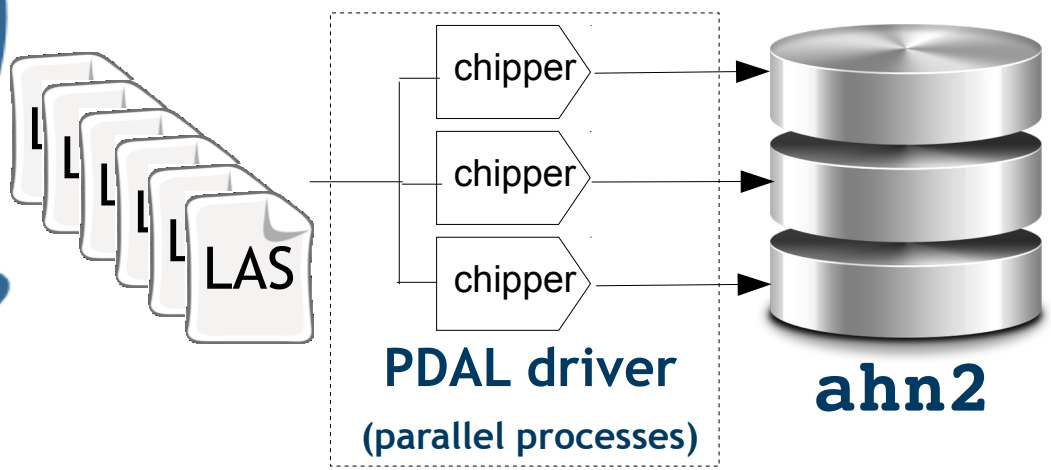
index building

BRIN	5 h

index size

BRIN	14MB

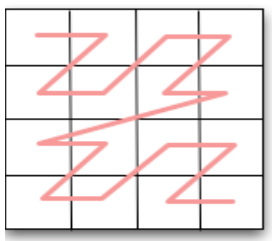
How data was inserted...



searches based on BRIN

polygon size	timing
~O(m)	~150s

Spatial sorting of a LiDAR dataset



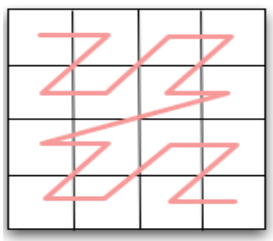
Morton order [http://geohash.org/]

```
CREATE INDEX patch_geohash ON ahn2
USING btree (ST_GeoHash(ST_Transform(Geometry(patch), 4326), 20));

CLUSTER ahn2 USING patch_geohash;
```

Need more than 2X table size free space

Spatial sorting of a LiDAR dataset

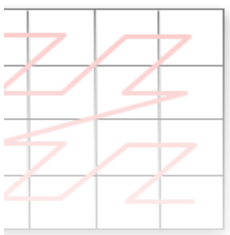


Morton order [http://geohash.org/]

```
CREATE TABLE ahn2_sorted AS
SELECT * FROM ahn2
ORDER BY ST_GeoHash(ST_Transform(Geometry(patch), 4326), 10)
COLLATE “C”;
```

Just need size for a second table

Spatial sorting of a LiDAR dataset



Morton order [http://geohash.org/]

45 hours

```
CREATE TABLE ahn2_sorted ON ahn2 AS
SELECT * FROM ahn2
ORDER BY ST_GeoHash(ST_Transform(Geometry(patch), 4326), 10)
COLLATE "C";
```

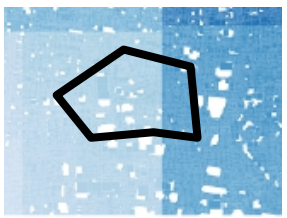
✓ Just need size for a second table

patches && polygons - BRIN performance

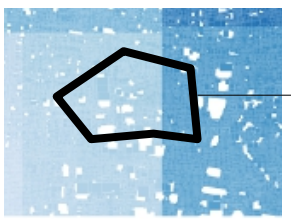
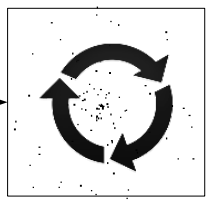
polygon size	**BRIN** timing	**GiST** timing
~O(10m)	~380ms	~20ms
~O(100m)	~400ms	~60ms
~O(1Km)	~2.7s	~3s
~O(10km)	~4.7s	hours

- After sorting: 150s→380ms
- GiST X20 faster than BRIN [r~O(10m)]
  - BRIN X200 faster than full scan
- BRIN X1000 faster than full scan [r~O(100m)]
- BRIN ~ GiST [r~O(1Km)]
- Just BRIN is used for searches r>1km

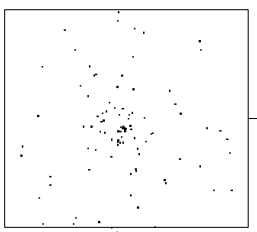
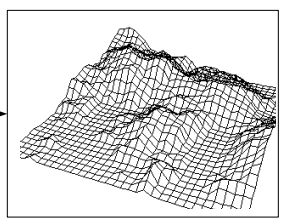
is the drop in performance acceptable?

- 

  BRIN searches = x20 GiST scan

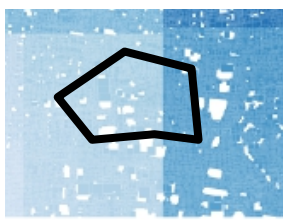
- 
  

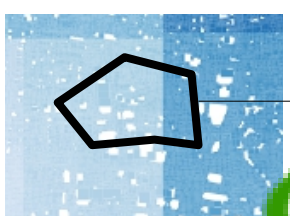
  patch explosion
  +
  NN sorting
  = x10÷x100 GiST scan

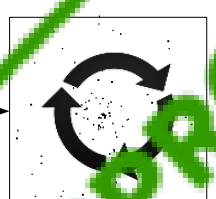
- 
  

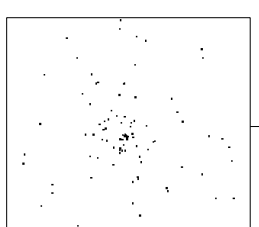
  constrained
  triangulation
  = x10÷x100 GiST scan

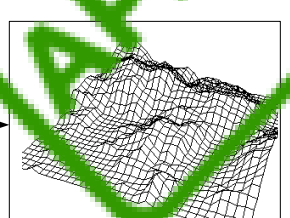
is the drop in performance acceptable?











- BRIN searches = x20 GiST scan
- patch explosion + NN sorting = x10÷x100 GiST scan
- constrained triangulation = x10÷x100 GiST scan

Conclusions

- BRINs can be successfully used in geo DB based on PostgreSQL
  - totally support PostGIS datatype, starting since PostGIS 2.3.0
  - easier indexes maintenance, low indexing time
  - less specific than GiST...but not to much!
- Really small indexes!
  - GiST performances drop as well as it cannot be totally contained in RAM
- Can PostgreSQL be used to manage LiDAR data?
  - Yes, at least for bbox inclusion searches
  - make sure that data has the same sequentiality of `.las`

2ndQuadrant
Professional PostgreSQL

DALIBO
L'expertise PostgreSQL

Creative Commons license