IGN INSTITUT NATIONAL DE L'INFORMATION GÉOGRAPHIQUE ET FORESTIÈRE

MATIS COGIT

THALES

A POSTGRES SERVER FOR POINT CLOUDS STORAGE AND PROCESSING

PRESENTED AT POSTGRESL PARIS SESSION 6.

HTTPS://GITHUB.COM/REMI-C/POSTGRES_DAY_2014_10_REMIC

RÉMI CURA Remi dot Cura … gmail

JULIEN PERRET – NICOLAS PAPARODITIS – GILDAS LE MEUR

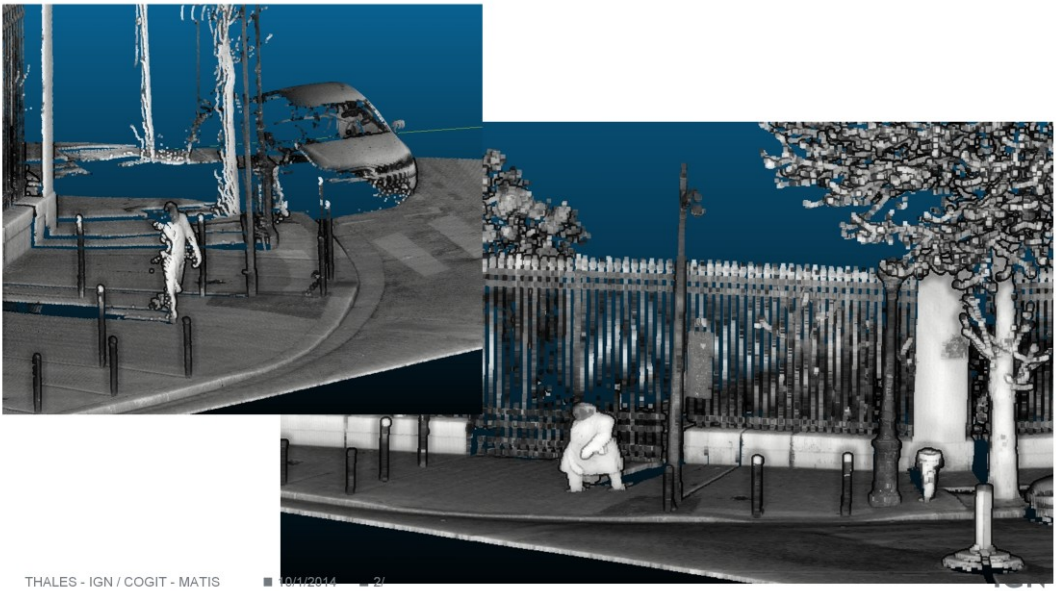
ign.fr

This presentation aims at

- introducing point cloud data, a new type of geo-spatial data
- demonstrating that using a point cloud server is better than using a file system.
- demonstrating that a point cloud server can be easily made and is efficient
- demonstrating some advanced capabilities of this point cloud server

INTRODUCTION TO POINT CLOUDS

- NEW KIND OF HUMAN: POINT CLOUD HUMAN!



THALES - IGN / COGIT - MATIS ■ 10/1/2014 – 2/

Here is an example of a terrestrial high density pointcloud acquired by a vehicle in Paris street.
The grey level is proportional to the intensity of the returned laser impulsion

CONTENTS

We will introduce you to Point Cloud data.
Then we show that such data is better used inside a DBMS rather than in a file system

Storing the data in DBMS allows fast operations, like different types of querying

We can also directly process data in base.

Last, such point cloud server can be integrated into a complex architecture with other servers.

WARNING

- **A FAIR WARNING :**
  - Except PointCloud extension : **research tools**.
    - **NOT** for production, **NOT** reliable
  - All **open source** (here , here , here), including nice data set

Thanks to my colleagues!

efficient storing/querying/Loading in postgres	Prototype
In base processing	Proof of concept
Visualization	
LOD	Proof of concept
Streaming	Prototype
Complex architecture	
Point cloud streaming	Prototype
Interactive road side modeling	Prototype
Batch computing	Prototype

IGN

All the work we present is open source.

It showcases advanced usages, that are mostly proof of concept or prototype.

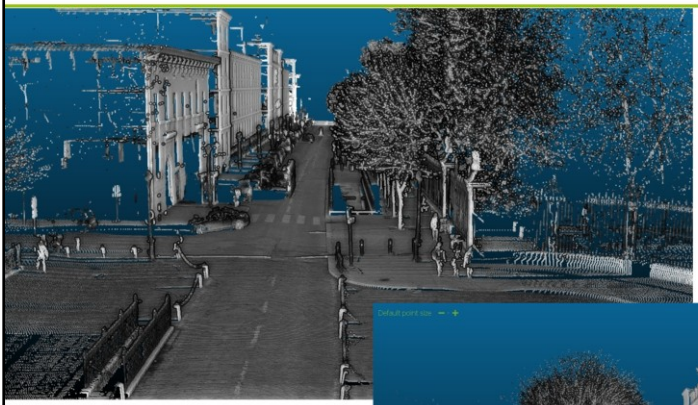
This examples demonstrate that a point cloud server can work and give orders of magnitude, but are absolutely not suited for production usage.

Be careful if you re-use code.

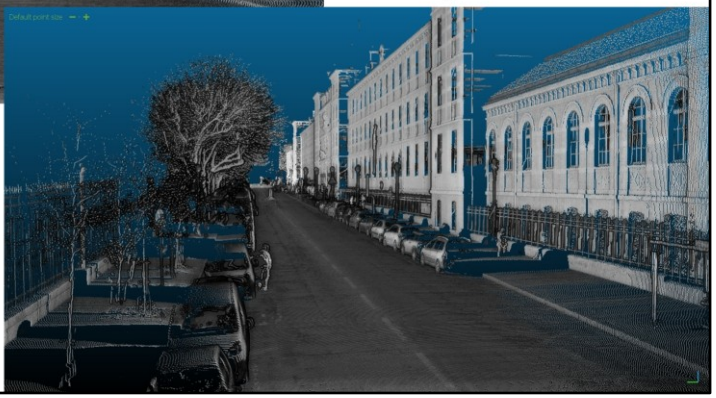
INTRODUCTION TO POINT CLOUDS

- WHAT ARE POINT CLOUDS
- ORDER OF MAGNITUDE

INTRODUCTION TO POINT CLOUDS



Riegl reflectance



Riegl reflectance

THALES - IGN / COGIT - MATIS ■ 10/1/2014

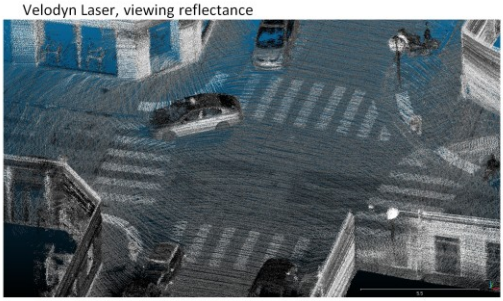
12 millions points, with the color proportional to intensity of reflected light.
Note that the lighting is artificial and only helps to understand the scene.

INTRODUCTION TO POINT CLOUDS

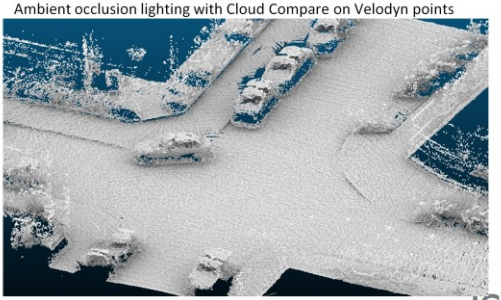
Riegl reflectance

Riegl reflectance

Velodyn Laser, viewing reflectance



Ambient occlusion lighting with Cloud Compare on Velodyn points



IGN

Top illustrations are again colored based on intensity of return.

Bottom illustrations are from a different lidar less precise, but acquiring much more points.

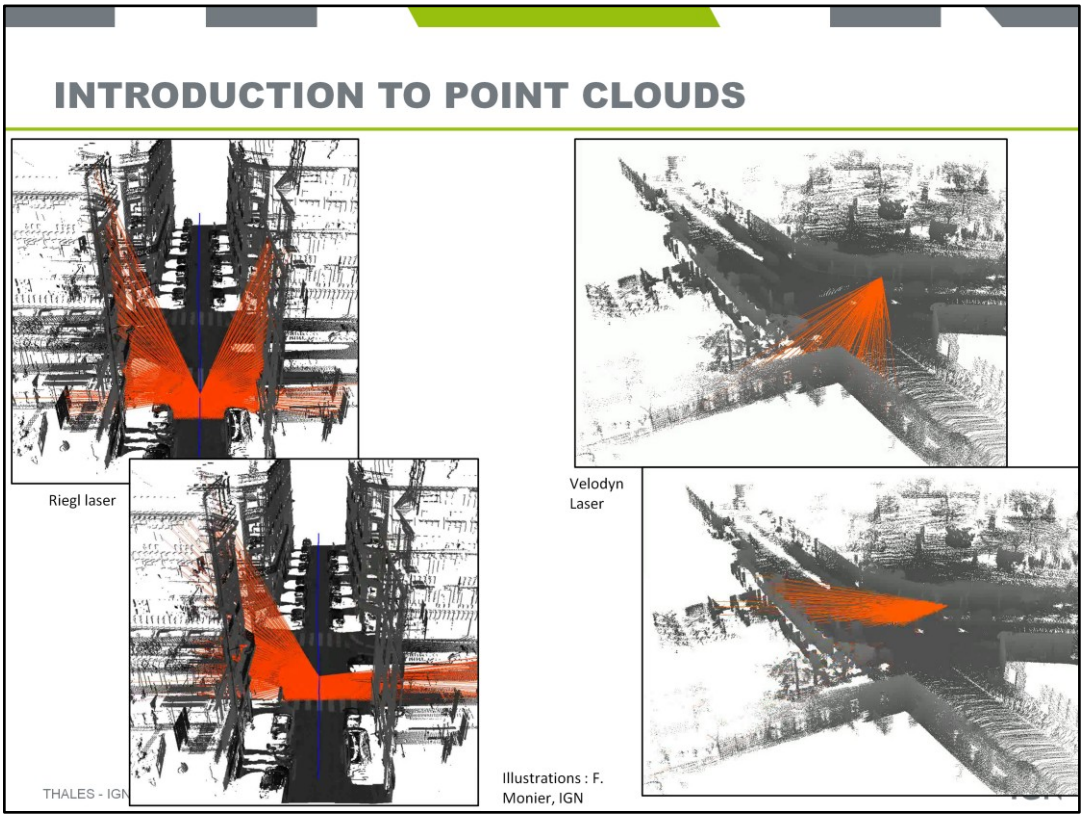
The bottom right illustration is only the geometry of the points (3D), with a lighting imitating ambient occlusion for a better understanding of the scene.

INTRODUCTION TO POINT CLOUDS

- **What are Point Clouds?**
- **A set of unordered 3D points with attributes resulting from a sensing operation.**
  - Unordered: don't know who the neighbors are .
  - Attributes: ex: intensity of returning light, class id, angle …
  - Sensing: physical sensing of reality, not like a vector point representing the position of a tree (no semantic).
- **Mostly from :**
  - Active sensors (laser time of flight based):
    - Terrestrial tripod
    - Terrestrial vehicle (cars/robot)
    - Aerial vehicle (plan/drone)
  - Passive/mixed : image (stereovision). RGBZ device (Kinect)

It is important to understand three things:

- point clouds are a big pack of unordered 3D points.
- each point may have several attributes.
- points are result the of sensing , they are a measure.



Left is an example of a Riegl lidar device:

When the vehicle moves forward along the blue line, it sweeps with laser around him in a plane orthogonal to direction.

This has several important consequences, like the presence of occlusion : if something stops the laser (like a car), we have no information for what is behind.

Right is an example of a Velodyn lidar device:

This laser is on top of a pole and rotating on himself.

It is like a lighthouse, but way faster.

INTRODUCTION TO POINT CLOUDS

- EXAMPLE : AERIAL AND TERRESTRIAL POINT CLOUDS

Non constant very high density

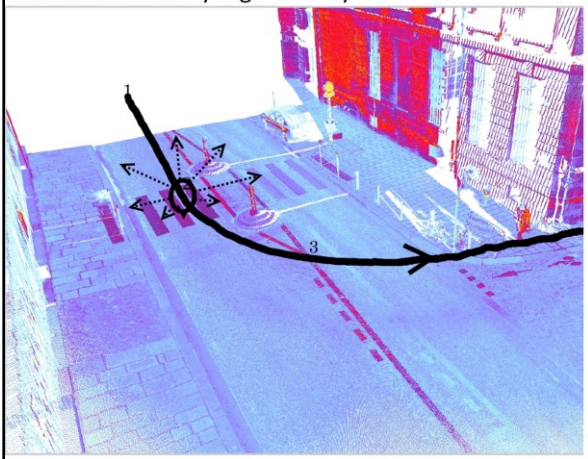


Image from J. Demantké Thesis, 2013

Almost constant low density

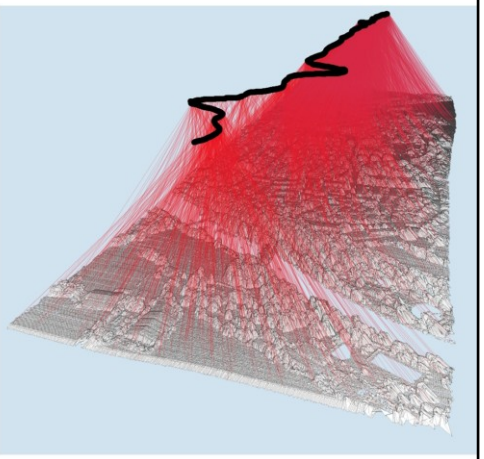


Image from J. Demantké Thesis, 2013

THALES - IGN / COGIT - MATIS ■ 10/1/2014 ■ 10/

IGN

The main differences between terrestrial lidar (left) and aerial lidar (right) are density of points, and geometrical complexity of scene.

INTRODUCTION TO POINT CLOUDS

- **WHY SO MANY PEOPLE USE IT?**
  - 3D from images is an ill posed problem. (i.e.: given a pixel in image space, what is its position and size in real world)
  - Very good precision (close and long range)
  - Reliable (night/bright day/shadows…)
  - Devices are affordable-ish (common on small robots, tested on small drones)
  - Complements very well images.

We live in a 3D world, and 3D is often essential for fast understanding of something.

Yet, it is difficult to get 3D information. One can draw it (CAD), reconstruct it from images, or get it from Lidar data.

Out of this 3 possibilities, only the Lidar data can scale well _and_ be robust and precise.

Combining Lidar and images allows to benefit from the best of the 2 worlds: lidar provides accurate geometry, image provides accurate radiometric information (i.e. colors).

INTRODUCTION TO POINT CLOUDS

- ORDER OF MAGNITUDE
- Point clouds are cool, but very BIG
  - Current devices: 1 **M**illion points/**sec**, 12+ attributes. Ouch !
  - 1 hour: several **B**illions points.
  - French mapping agency (IGN): 100's of data sets, aerial + terrestrial, several type of lasers.
  - Can't load much more than 20 Millions points in memory
    => can't process much more than 10 Millions points at a time.
  - So much data: working on its own copy of it is not an option.
  - All data must be centralized on specific storage solution.

Using point cloud one can easily step into the "big data" realm.

That is, when you have to use special strategies because using one computer won't suffice.

Big data is not a showstopper per se, but has to be deal with using dedicated tools and methods.

WHY USE A DATABASE MANAGEMENT SYSTEM ?

- WHAT PEOPLE DO WITH POINT CLOUDS?
- USING A FILE SYSTEM SOLUTION
- USING A DBMS SOLUTION

WHY USE A DBMS?

- **WHAT PEOPLE DO WITH POINT CLOUDS? (SEE [OOSTEROM, 2014] IN "RESOURCES")**
  - Create data
  - Query data based on
    localization
    time
    attributes
  - Mix data set
  - Convert data
  - Process data
  - Visualize data
  - Update data

IGN

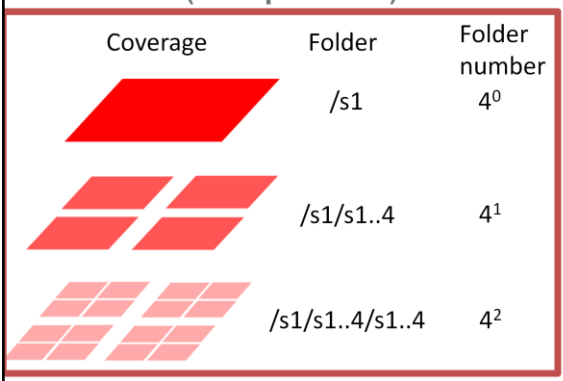
The requirement are very similar to other geospatial data:

Load the data, efficiently use it, process it, visualize it, write it.

The needs are a brief summary taken from (oosterom,2014)

WHY USE A DBMS?

- ILLUSTRATION OF QUAD TREE IN FILE SYSTEM
- Typically, point clouds are cut in small files inside a hierarchy of folders (ex : quad tree).



Level	folder	cum_folder
0	1	1
1	4	5
2	16	21
3	64	85
4	2.6e+02	3.4e+02
5	1e+03	1.4e+03
6	4.1e+03	5.5e+03
7	1.6e+04	2.2e+04
8	6.6e+04	8.7e+04
9	2.6e+05	3.5e+05
10	1e+06	1.4e+06
11	4.2e+06	5.6e+06
12	1.7e+07	2.2e+07

Greater than max. folder number on NTFS and EXT file systems

Is it enough? 100k pts/file
$8*10^6 * 10^5 = 10^{11}$ = **800 Billions** pts.
For **one** laser : 3.6 Billions/h.
Working day (8h) : 30 Billions
=> **Only 30 days of work for 1 laser!**

This kind of quad tree solution is very common in problems where data can be easily partitioned.

The main issue is that such file system partitioning allows to use only one key for partitioning : either a spatial key as illustrated here, or an attribute key (for instance the precise moment of acquisition of a point).

This problem is similar to trying to order its personal photo collection : you can order it by themes (sport/vacation/…), __OR__ order it by date , but you can't mix efficiently several orderings. To solve this problem one can use dedicated software … that use a database engine.

Similar problem should lead to similar solution.

WHY USE A DBMS?

File system limitations

▪ Create data	Only 1 user at a time
▪ Get data based on localization time attributes	Need to choose a File structure adapted to one and only one query type (analogue to : how to sort personal photos)
▪ Mix data sets	Very difficult
▪ Convert data	OK
▪ Process data	User need to manually create buffers of points
▪ Visualize data	OK
▪ Update data	Only 1 user at a time

The most severe limitations are efficient query (localization, time, attributes, functions) and mixing data set.

WHY USE A DBMS?

USING A **FILE SYSTEM**

- **Everything can be done, but it would amount to redeveloping a minimal DBMS system !**
- **No security of the data**
- **No concurrency**
- **Need a different solution for every kind of geo-data (points, raster, vector)**

- **Comparison to raster world:**
  - Who uses a pure file system solution for data over 100's of To?

We use security in the broad sense: protection against errors for example.

In the G.I.S world data base have been common for dozen of year.

Incorporating a new spatial data type (point cloud) with the other spatial data makes then lot's of sense.

WHY USE A DBMS?

- USING A DBMS SOLUTION

- Allow concurrency / clusters of servers
- All geospatial data in the same place

- Efficient querying on localization & time & attributes & data set
- Proper management of metadata at the data set level
  + relational link to other data.
- Point clouds as a service : can be integrated in sophisticated client system (see end of this presentation).

Why use Postgres and not a NoSQL database?
Parallel computing on multi cluster possible (postgres-xc)
Full ACID warranty
Can use python/R/C/Java
All geo data in same place.

Without too much surprises using a DBMS solves problems that are hard to solve using a file system solution.

The price is that a DBMS is more complex to understand than a simpler solution.

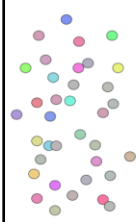
POINTCLOUD : AN EXTENSION FOR EFFICIENT STORING/QUERYING/LOADING IN POSTGRES

- STORING POINT CLOUDS IN DBMS : BLUE OR RED PILL?
- WHAT IS POINTCLOUD?
- EFFICIENT STORAGE
- FAST QUERYING
- FAST LOADING
- ORDER OF MAGNITUDE

All this section uses PointCloud, a postgres extension written by P.Ramsey, the founder of PostGIS.

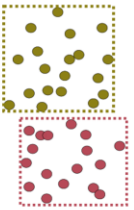
EFFICIENT STORING/QUERYING/LOADING IN POSTGRES

STORING POINT CLOUDS IN DBMS : 2 APPROACHES



1 point =

GPS_time (s)	X (m)	Y(m)	Z(m)	reflectance	(....)
54160.295	2068.230	20690.025	45.934	-9.4497	(....)



1 row = **1** point
Analogy : ST_Point

1 row = **N** points
Analogy : ST_MultiPoint

Billions of row	->	**manual** partitioning	Millions of row	->	1 table per dataset
Billions of row	->	**big** indexes	Millions of row	->	small indexes
Double/float storage	->	**wasted** storage	Custom bit width	->	efficient storage
1 row = 1 point	->	**no** compression	1 row = N point	->	compress by redundancy
Different laser	->	**no** compatibility	Different laser	->	1 family type
Different laser	->	**custom** code everywhere	Different laser	->	postgres extension

Usage : do we really need to get points 1 by 1?

THALES - IGN / COGIT - MATIS ■ 01/10/2014 ■ 20/ IGN

When storing billions of points in data base, we have to make a fundamental choice : do we store one point per row, or one group of point per row?

The only scalable and easy enough solution is to store **a group of points** per row.

This is the correct solution because we don't usually want to get only one point, but almost always a group of points.

EFFICIENT STORING/QUERYING/LOADING IN POSTGRES

- WHAT IS POINTCLOUD?
  - A Postgres extension created by P.Ramsey (founder of PostGIS) (see ref.).
  - Strong similarities to PostGIS (design, robustness, reliability, perfs) !
  - New types (PC_Point, PC_Patch) + cast to geom + functions

XML Schema

```
<pc:PointCloudSchema xmlns:pc=
"http://pointcloud.org/schemas/PC/1.1"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <pc:dimension>
    <pc:position>1</pc:position>
    <pc:size>4</pc:size>
    <pc:description>X coordinate as a long integer. You must
    use the
            scale and offset information of the header to
            determine the double value.</pc:description>
    <pc:name>X</pc:name>
    <pc:interpretation>int32_t</pc:interpretation>
    <pc:scale>0.01</pc:scale>
  </pc:dimension>
```

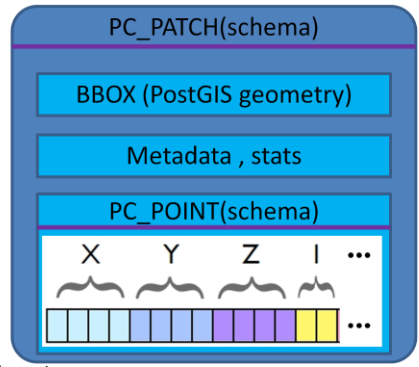


Illustration
from [Ramsey,2013]

IGN

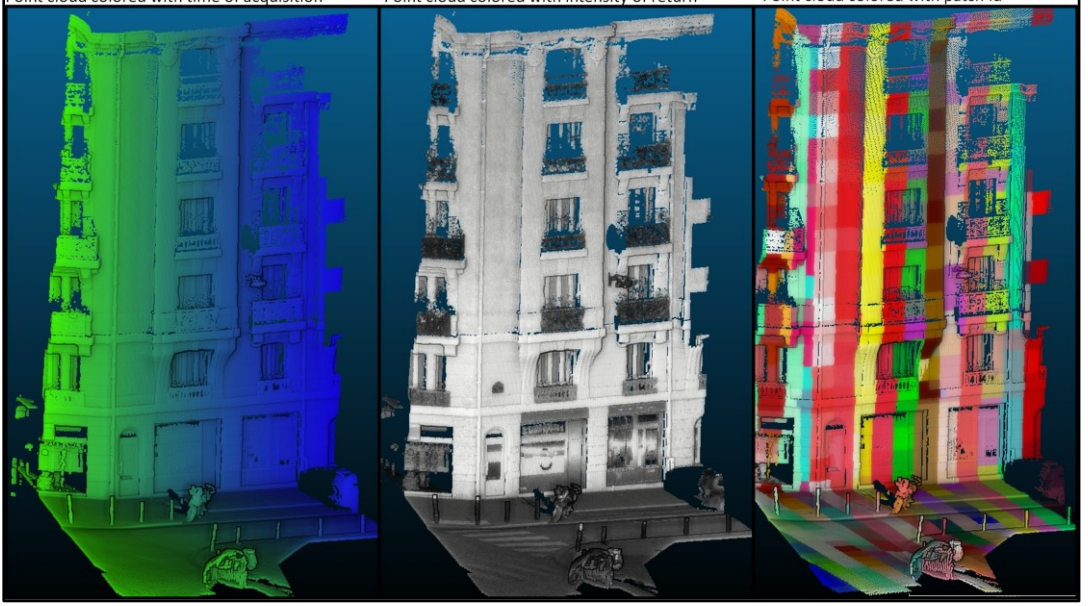
PointCloud is a really good base for point cloud in DBLS storage.
It is very stable and reliable, and overall reasonably efficient.

Using it is simple.
One define a point type with an XML schema. Each attribute of the point can be precisely described (size, interpretation, precision, scale).

Then we group points into patch. A patch is a group of points, plus metadata, like Bounding Box, stats, etc.

EFFICIENT STORING/QUERYING/LOADING IN POSTGRES

Point cloud colored with time of acquisition | Point cloud colored with intensity of return | Point cloud colored with patch id



Here are illustrations of a small piece of point cloud (about 1 million points), representing a façade and a portion of Paris street.

The left illustration shows the time of acquisition of each point with a green blue color gradient. (i.e. the vehicle was going left to right).

The middle illustration is colored with the intensity of return, along with special lighting to help visual understanding.

The third illustration shows how the point cloud have been cut into patches.

In this case the patches are 1m3 wide.

Here the number of points per patch (density) varies a lot (10's to 10k's).

EFFICIENT STORING/QUERYING/LOADING IN POSTGRES

Illustrations : Wikipedia JPEG

- EFFICIENT STORAGE
- Why is compression important?
- An example form image world :
  - Full HD Camera : 1920×1080 pixels × 25images × (1+1+1 octet) /sec -> **155 Mbyte/s**
  - Laser : 1 million points * 10 attributes * 2 (doubles) -> **20 Mbyte/s**

Yet nobody is speaking of big data regarding video!
WHY?
Because we know very well how to compress images.
Example: standard for professional of video : DNXHD

- Full HD : **18 Mbyte/s** (1/8) or **4.5 Mbyte/s** (1/30)

Compression is fundamental.

The more important part of it is not really solving the practical problem of the data size.

It is more that a good compression exploits a structure inherent to the data. Thus a good compression of the data is more about understanding the data.

EFFICIENT STORING/QUERYING/LOADING IN POSTGRES

- EFFICIENT STORAGE
- POINTCLOUD can **compress patches**.
  - Compression always is about exploiting similarities.
  - For a PointCloud patch, compression is attribute by attribute
  - Depending of the similarities, 3 methods are automatically used
    - Use bit mask : 10001 ; 10002; 10003 → mask=1000, data = 1,2,3
    - Use repetition: 10,10,10,10,10,10,10 → repetition=6, data = 10
    - Use lzip deflate algorithm →(dictionary, tree)
  - Example :
  - For this benchmark data , 12 Million points,
  - On disk : binary file on disk **600 Mbyte**
    - zipped Binary file on disk : **305Mbyte**
  - In base : 25 k patches, (Table=10)**+**(toast=290)**+**(index=5) =**305 Mbyte**

Generally the strategies used to compress point clouds are much less sophisticated than those used in image world.

For instance the sophistication of the JPEG2000 standard has no match in point cloud world.

EFFICIENT STORING/QUERYING/LOADING IN POSTGRES

- FAST SPATIAL QUERYING
  - We index bounding box of patches.
    CREATE INDEX ON patch USING GIST(patch::geometry)
  - Index on several S.R.S possible
    CREATE INDEX ON patch USING GIST(ST_Transform(patch::geometry,4326))
- FAST ATTRIBUTE QUERYING

  Example : precise time of acquisition
  - Use a function that compute range of an attribute in a patch
    rc_compute_range_for_a_patch(patch, text) :
    NUMRANGE(PC_PatchMin(patch, 'GPS_Time'),PC_PatchMax(patch, 'GPS_Time'),'[]');
  - GIST Index on this function
    CREATE INDEX ON patch USING
    GIST(rc_compute_range_for_a_patch(patch,'GPS_Time')

It is important to understand that with _one line of code_

- we deploy a powerful Rectangle-Tree on the data

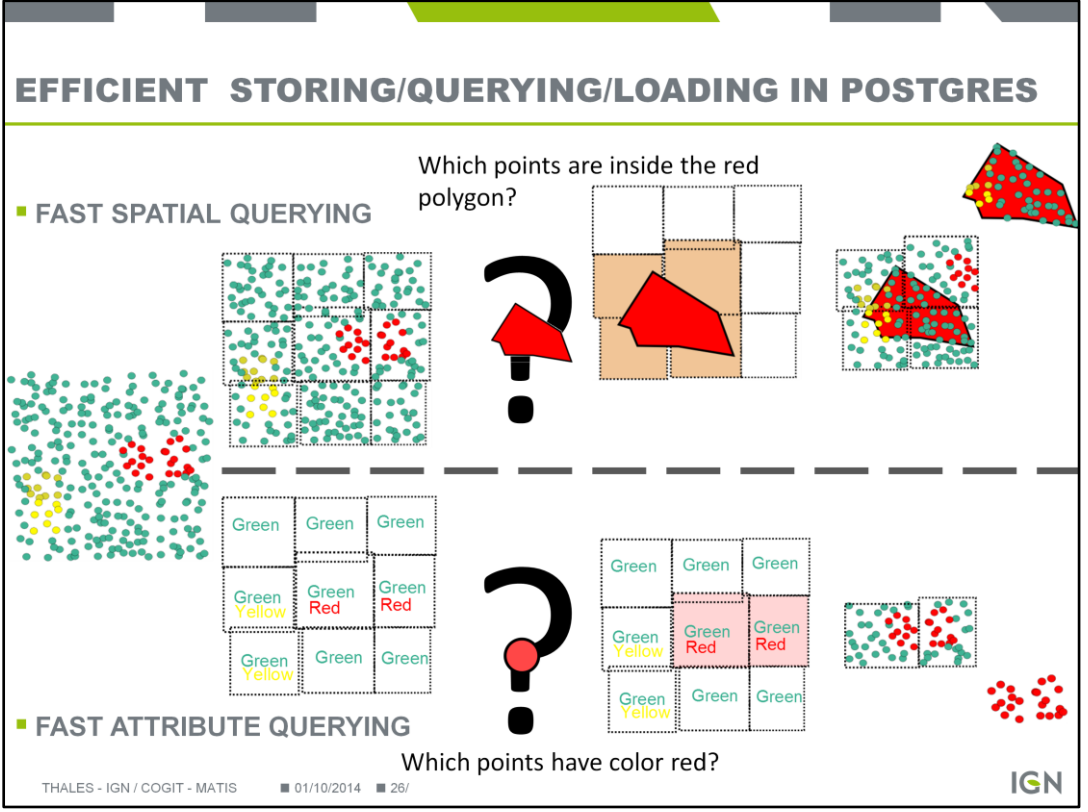
- we deploy automatic stats on the data to help decide if it is worth it to use R Tree or not.

The simplicity of index usage allows to use several indexes on different keys.

This indexes are automatically combined with multi criteria query.

There are several type of indexes , and several options to best use them. This is out of scope of this presentation.

Bottom line is : index creation is easy, and index are very effective.



The top part illustrates an efficient spatial query (ex. 1 ms, on the 5 Billion points database, 500k patches).

In a spatial query we want to find points that relate to a given spatial location.

This query is efficient because an automatic indexed first step uses bounding box of patches.
Because of this, only 4 patches need to be read, and only points of those patch need to be tested (compared to several Billions).
The bottom part shows an efficient attribute query. (same)

In an attribute query we look for points with given attribute values.

This is fast because again we have an index on which patch contains which color.

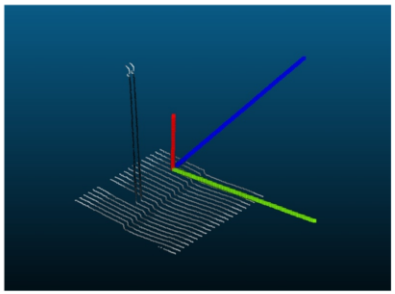
This attribute query is simplified for the illustration purpose, but in fact we can query on range. For instance let's say that every point has a number between 0 and 1 representing the intensity.

We can efficiently answer to the question ; "what points have intensity value between 0.34 and 0.59?".

EFFICIENT STORING/QUERYING/LOADING IN POSTGRES

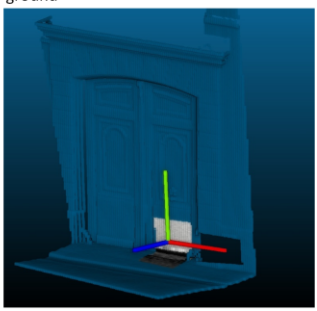
- BONUS : POSTGRES SUPER-COOL FEATURE :
- FAST FUNCTION QUERY : INDEX ON ND DESCRIPTORS
  - Example :
  - Define a function computing ND descriptors of a patch
    - example : **How vertical is the patch?** (based on I.C.A.)
  - Create an index on it

Verticality descriptor for a patch containing a pole



(**0.1** ,0.002,0.008)

Verticality descriptor for a patch half façade half ground



(**0.7** ,0.001,0.005)

IGN

Postgres has a very powerful functionality that is **index over function**.
We can leverage this in a point cloud server.

The point cloud is stored as a set of patches, each patch containing some points.

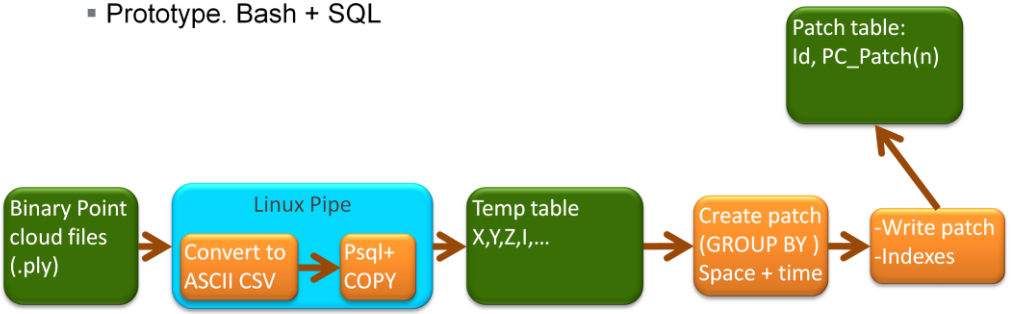
We can compute one or several descriptor for every patch. Then we can use these descriptors to query even more efficiently.
For instance, the illustrations shows a verticality descriptor.

When looking for points describing a street markings for instance (i.e. flat patch),

we can use the verticality descriptor to exclude all patches on façades (>50% gain).

EFFICIENT STORING/QUERYING/LOADING IN POSTGRES

- FAST LOADING IN POINTCLOUD
- For Simple test : use **PDAL** (GDAL for point clouds wannabe)
- An example of fast loading: project **PointCloud_in_db** (PROTOTYPE)
  - Parallel loading into a server : we can **load as fast as we acquire** data !
  - Prototype. Bash + SQL



THALES - IGN / COGIT - MATIS ■ 10/1/2014 ■ 28/ IGN

The point is to prove that efficient loading is possible.

The schema is simplified.

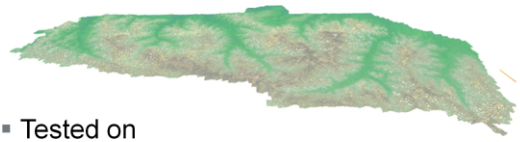
In fact PointCloud_in_db is a prototype and could be easily made several times faster.

(for instance, reading directly data in binary file, instead of casting it to ascii, then parsing the ascii…).

EFFICIENT STORING/QUERYING/LOADING IN POSTGRES

- ORDER OF MAGNITUDE

Vosges aerial acquisition, 3D rendering



Paris terrestrial acquisition 2013
Above view

- Tested on
  - T: terrestrial , 600 **Millions**, 22 Attributes . **12km** of streets in Paris
    - 2.1 Million rows
    - Table : 1.4 GByte | Toast : 29 Gbyte | indexes : 500 MBytes
    - Big variation in Density/patch
  - A: aerial, 5.2 **Billions** , 9 Attributes. 1300 km2 in Vosges, France
    - 580k rows
    - Table : 64MByte | Toast : 45 Mbyte | indexes : 85 MBytes
    - Density almost constant.

Querying is fast! (1-2ms)
ASCII output : 100k pts/sec

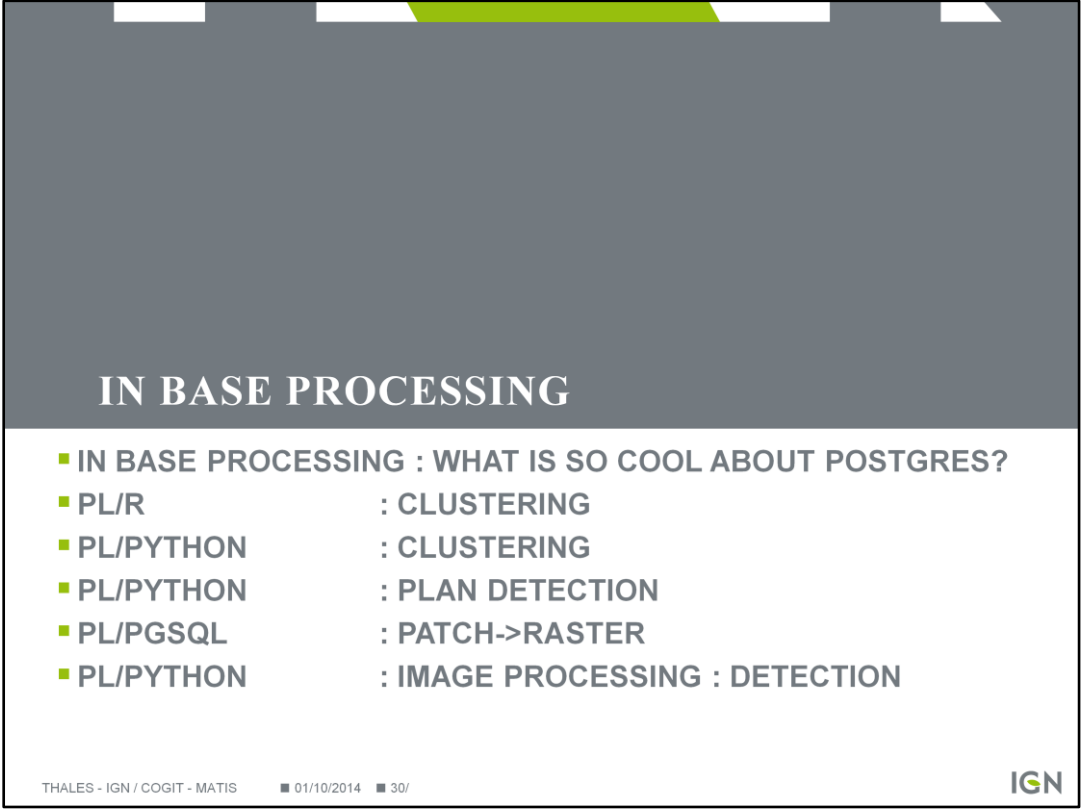
IGN

The left illustration is the Vosges dataset.

The second illustration is an overview of the Paris 6 data set.

The conclusion of this tests are

- It is possible to use a point cloud server with large amount of points
- Even with billions of points, querying (localization, attributes, function) is fast



The previous parts shows basics for a point cloud server :

- efficiently loading data.

- efficiently getting points based on several criteria (position, attributes, function).

We can use these to do in-base processing.

A classical **out-of-db workflow** would be : getting data, processing, then writing result on database.

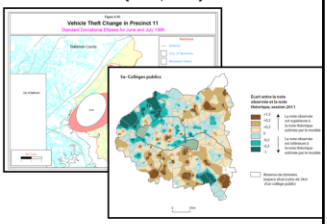
Instead, when doing **in-base processing**, the processing is directly done inside the point cloud server.

This avoid unnecessary data transfer, and greatly facilitate parallelism and automatism.

IN BASE PROCESSING

- IN BASE PROCESSING : WHAT IS SO COOL ABOUT POSTGRES?
- Very *easy and fast* for PROTOTYPING :
  - ~~C , C++~~
  - High level languages :
    - R
      - **Very** advanced stats
      - Advanced Fitting/patterns/clustering capabilities
      - Many geo modules (sp …)
      - Possibility of GUI in browser (shiny)
    - Python
      - Powerful script language
      - Powerful **O**bject **O**riented language
      - Fully integrated in most GIS (ArcGIS, GRASS, QGIS))
      - Can link to C/Cpp for perf (Cython)
    - Java
      - Less integrated in postgres

Illustrations from [Cura,2014]



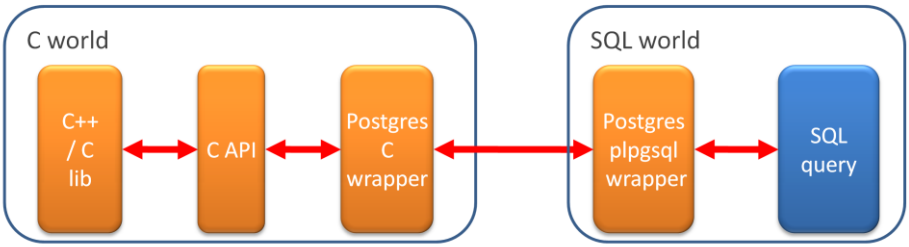
- Easy to learn/use
- IO to DBMS + GIS world.
- Central Repo.
- Fully portable
- Well established

Postgres offers the possibility to use many languages directly in base . Among these languages 2 are particularly useful for fast and powerful prototyping : R and Python.

In this presentation we demonstrate basic usage of both.

IN BASE PROCESSING

- IN BASE PROCESSING : WHAT IS SO COOL ABOUT POSTGRES?
- Easy to create a **postgres extension**
  - ~~High level languages~~
  - C , C++
    - Very efficient (memory/CPU)
    - Integrating into Postgres forces common meta API/data
    - This allows re-use / complex processing pipeline.



When prototyping is done, it is really easy to add a postgres C/C++ extension to have a better efficiency.

An important point of in base processing is that it forces a de facto common framework/API.

Thus it guarantees inter-operability, which maximize re-use and allow forming complex processing pipeline.

IN BASE PROCESSING

- PL/R CLUSTERING (PROOF OF CONCEPT)
- Unsupervised clustering.
- Input = 3D points, module= nnclust , method = minimum spanning tree
- Result : points in tree are separated from points on road

R classification, viewed from above. Color = classes

THALES - IGN / COGIT - MATIS ■ 01/10/2014 ■ 33/

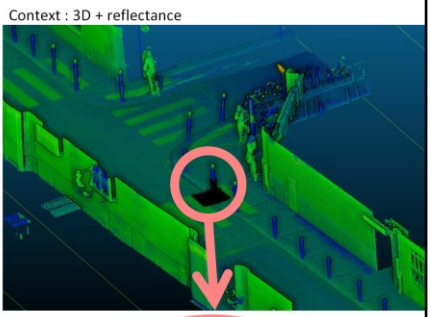
IGN

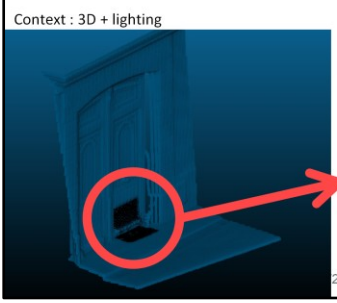
We demonstrate in-base R processing, using the nnclust R package.
We try to cluster points of a point cloud.

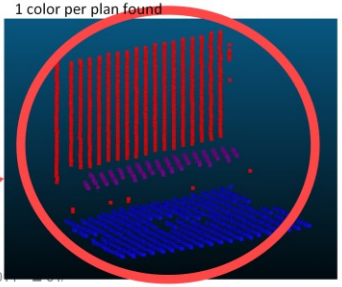
The illustration (viewed from above) shows that the R function has put different labels on points in tree and points on road.

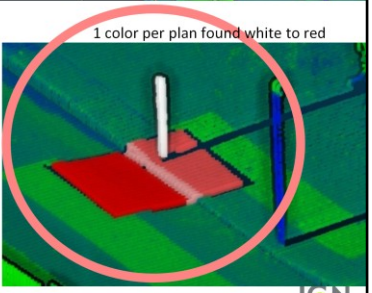
IN BASE PROCESSING

- **PL/PYTHON PLAN DETECTION (PROOF OF CONCEPT)**
- For list of points (double[ ])
  - Convert list to numpy array
  - Convert numpy array to python-pcl point cloud
  - Until no more plan is found
    - Find a plan using p-Ransac
    - Remove points of the plan from the cloud









This example using pl/python shows how to **detect plans** in a patch using python-pcl.

The bottom middle and right illustrations use one color per plan detected (following orientation of normal) to get an idea of the result.

The important result is points have been correctly classified into plans.

Detecting plans is vital because they are **high level geometric primitive**.

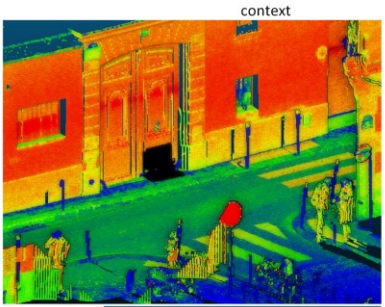
For instance instead of storing 1k points representing a plan, we can just store the equation of the plan (3 points) and its limits (varies).

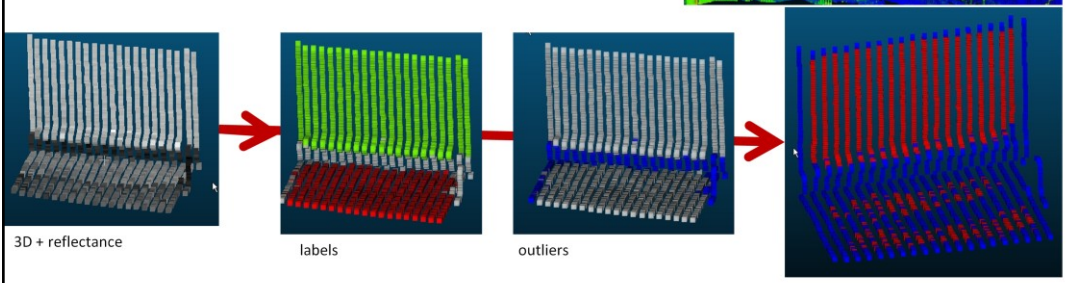
Using such geometric primitive will then allow to do very fast proximity and intersection test.
It also gives information about what things the points may be (**not a tree** for instance).

IN BASE PROCESSING

- **PL/PYTHON UNSUPERVISED CLUSTERING (PROOF OF CONCEPT)**

- For list of points + reflectance (double[ ])
  - Convert list to numpy array
  - (Compute normals using python-pcl)
  - Cluster using sklearn.cluster.DBSCAN
  - Join clustering label to point id





Here a pl/python proof of concept uses a totally different method to **try to group points into meaningful groups** (unsupervised clustering).

In opposite to the most common situation, the method doesn't know what to look for or how many objects it must find.

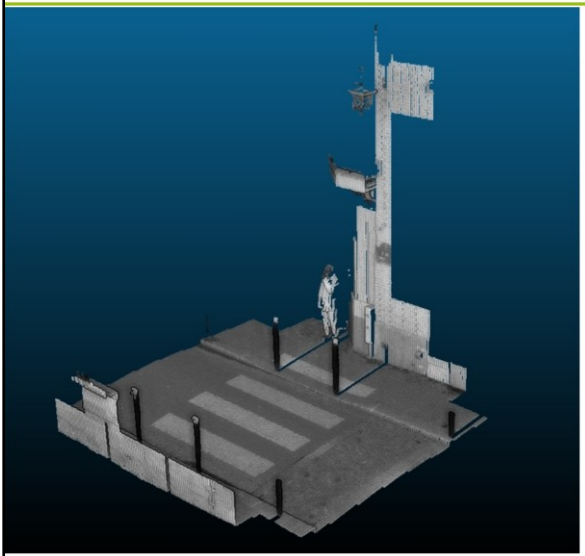
The DBSCAN implementation used here scales very badly (compute distances between every points),

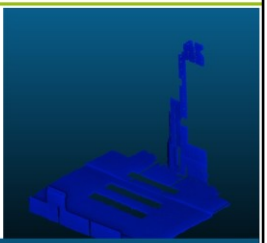
but is sufficient to exhibit interesting results.

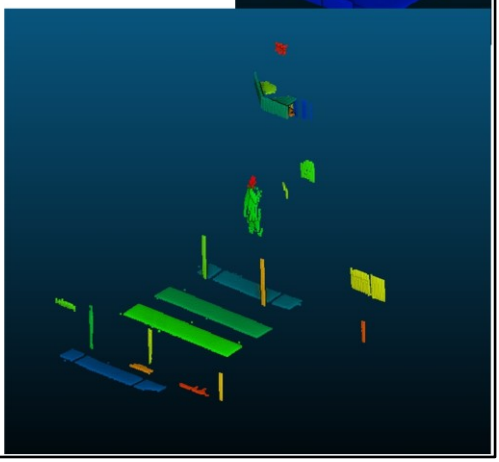
A byproduct of the method is a new label saying if the point is very meaningful to its group or not.

This can be used to robustify the process for example.

IN BASE PROCESSING







The result of the same method on a larger area are much more interesting.

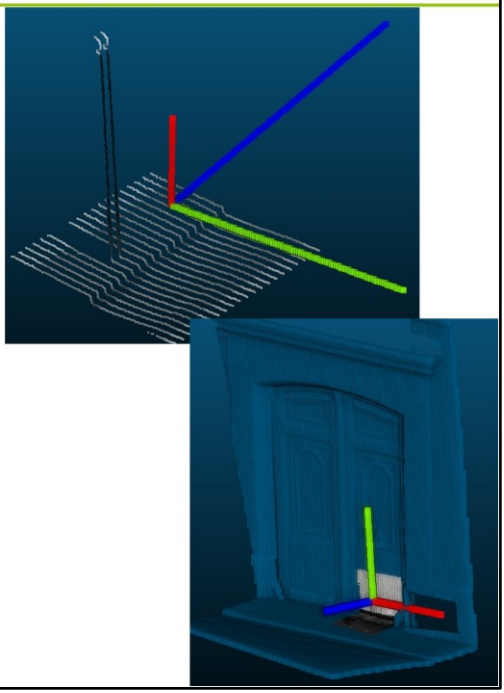
The method has automatically separated the points into objects.

These objects are quite close to ones a human would have chosen.

Yet some objects have not been correctly segmented (a zebra is missing for instance)

IN BASE PROCESSING

- PL/PYTHON DESCRIPTOR (PROOF OF CONCEPT)
- For list of points (double[ ])
  - Convert list to numpy array
  - Get Principal direction using sklearn.decomposition.FastICA
  - (Project result on Z)
  - Output measure
- Idea : descriptor at patch level of verticality : is this patch façade/road/ tree/ mixed? => allow filtering at patch level



THALES - IGN / COGIT - MATIS ■ 01/10/2014 ■ 37/

In the previous slides we presented 3 clustering methods using PL/R or PL/Python.

Such processing is done in-base but could be done outside of the db.

For this application the ability to use it in-base is essential.

The idea is for instance to add a column to the patch table called descriptor. So for every patch of points, we have a descriptor.

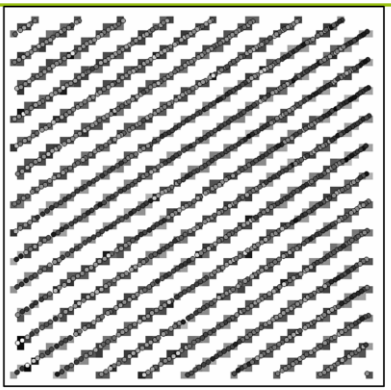
Such descriptor is very useful to filter the patch, or try to guess what object the points in patch represent.

The capacity to compute the descriptor in base allows for **automatic update of the descriptor** whenever the points in a patch changes or when a new patch is added (via triggers).

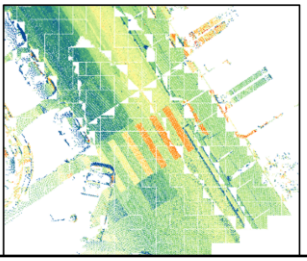
This guarantee that the descriptor is up to date with the patch.

IN BASE PROCESSING

- PL/PGSQL PATCH->RASTER (PROOF OF CONCEPT)
- For a patch
  - Get Bbox
  - Round X Y to pixel size
  - Group by X,Y rounded
  - For each attributes
    - Add a band
    - File the band with Mean/max/min of attribute for computed pixels
- Very slow (0.1 to 1 sec/patch , 13 bands)



Concept : raster+points. Color = reflectance. Qgis



A street part viewed in QGIS trough GDAL PostGIS raster driver

IGN

In this example we demonstrate use of another script language : pl/pgsql. This language is interesting because it is very tightly integrated into postgres and interacts very well with SQL.

This proof of concept function convert a point cloud into a raster (image viewed from above).

It is formally a projection of points along the Z axis, then a quantization to form pixel of the given size.

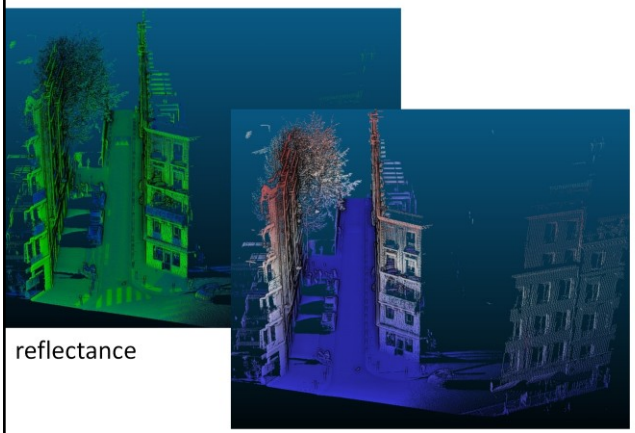
It is essential to be able to convert point clouds to image because image processing algorithm can be greatly accelerated using GPU.

Moreover, in the image dimension we can leverage the neighborhood information (we know close pixels) that is missing in point cloud.

Next slides demonstrate use of image processing in base.

IN BASE PROCESSING

- PL/PYTHON IMAGE PROCESSING **(PROOF OF CONCEPT)**
  - Convert set of patches to raster (projection on Z of the points, rasterisation)



reflectance

Relative height

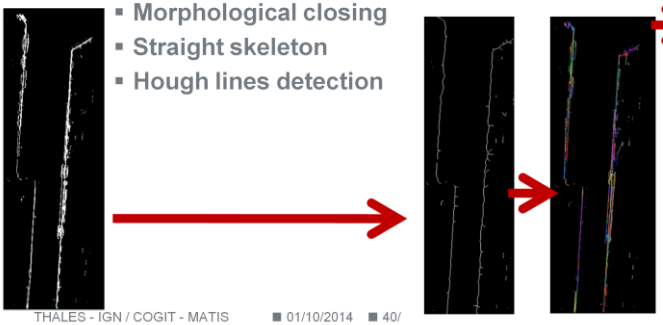
Accum. raster. Lot's of "NoData"

IGN

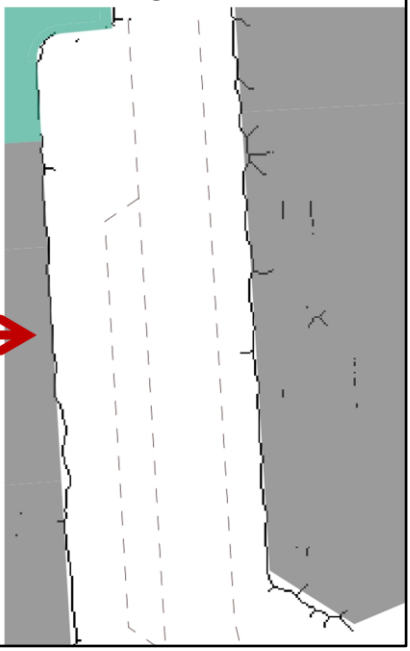
We now use the conversion from patches to raster to perform python image processing on a point cloud.

IN BASE PROCESSING

- **PL/PYTHON FAÇADE DETECTION (PROOF OF CONCEPT)**
  - For a raster (accum) from a projected point cloud
    - Keep pixels high enough (higher than acquisition vehicle roof)
    - Remove pixels close to no data
    - Threshold on number of accumulated point per pixel.
    - Morphological closing
    - Straight skeleton
    - Hough lines detection



Result with ground truth



The first application is to **detect façades of buildings**.

We convert the point cloud to raster, then use various filters to detect façade in it.

The right illustration shows the result, that are very close to the ground truth (Open Data Paris Building data).

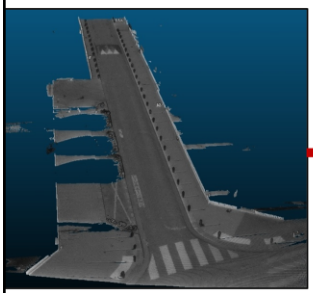
The small variation may be explained by the un-precise way we define building border : are we taking into account balcony, flap, etc.?

This detection example is based on geometrical information.

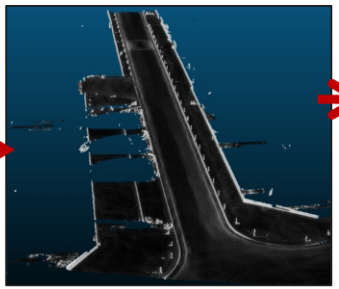
IN BASE PROCESSING

- PL/PYTHON SIDEWALK DETECTION (PROOF OF CONCEPT)
  - For a raster (height) from a projected point cloud
    - Keep pixels low enough (at vehicle wheels height)
    - Remove pixels close to no data
    - Compute gradient of height (height variation)
    - Filter gradient to keep sidewalk (0.1 to 15 cm)

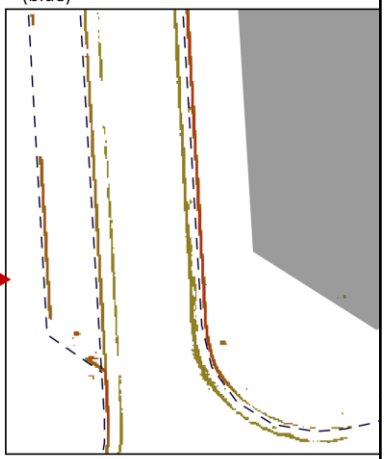
Reflectance at wheel level



Gradient at wheel level



Result (orange to red) with ground truth (blue)



THALES - IGN / COGIT - MATIS ■ 01/10/2014 ■ 41/

IGN

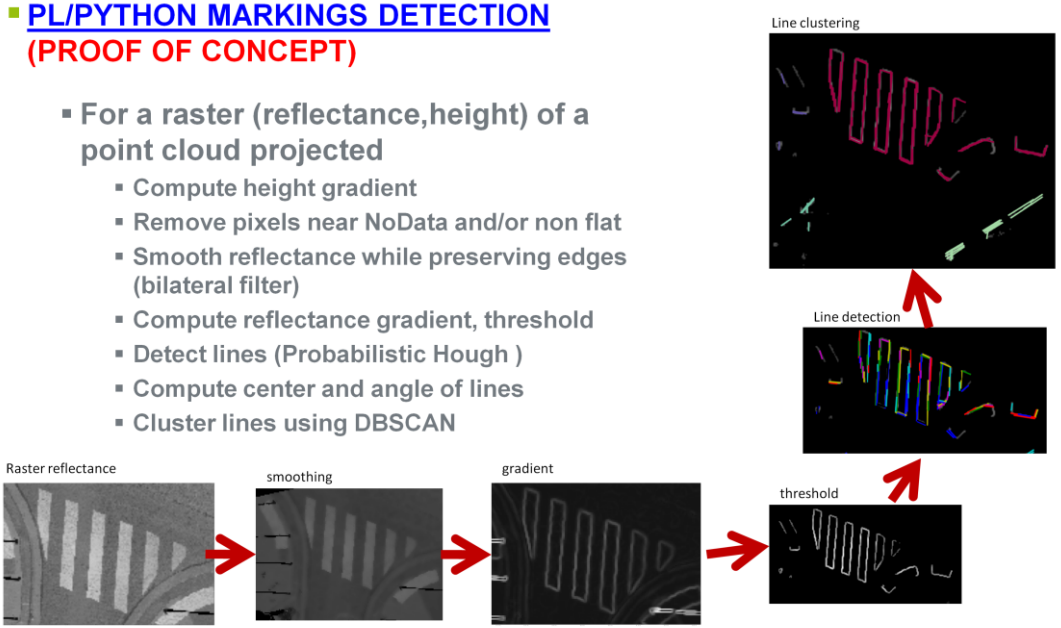
Another image processing example with pl/python .
Here we detect sidewalk (and gutters).

Such information is very important because it strongly structures street space.

The detection is based on geometrical information.

IN BASE PROCESSING

- **PL/PYTHON MARKINGS DETECTION (PROOF OF CONCEPT)**
  - For a raster (reflectance,height) of a point cloud projected
    - Compute height gradient
    - Remove pixels near NoData and/or non flat
    - Smooth reflectance while preserving edges (bilateral filter)
    - Compute reflectance gradient, threshold
    - Detect lines (Probabilistic Hough)
    - Compute center and angle of lines
    - Cluster lines using DBSCAN



IGN

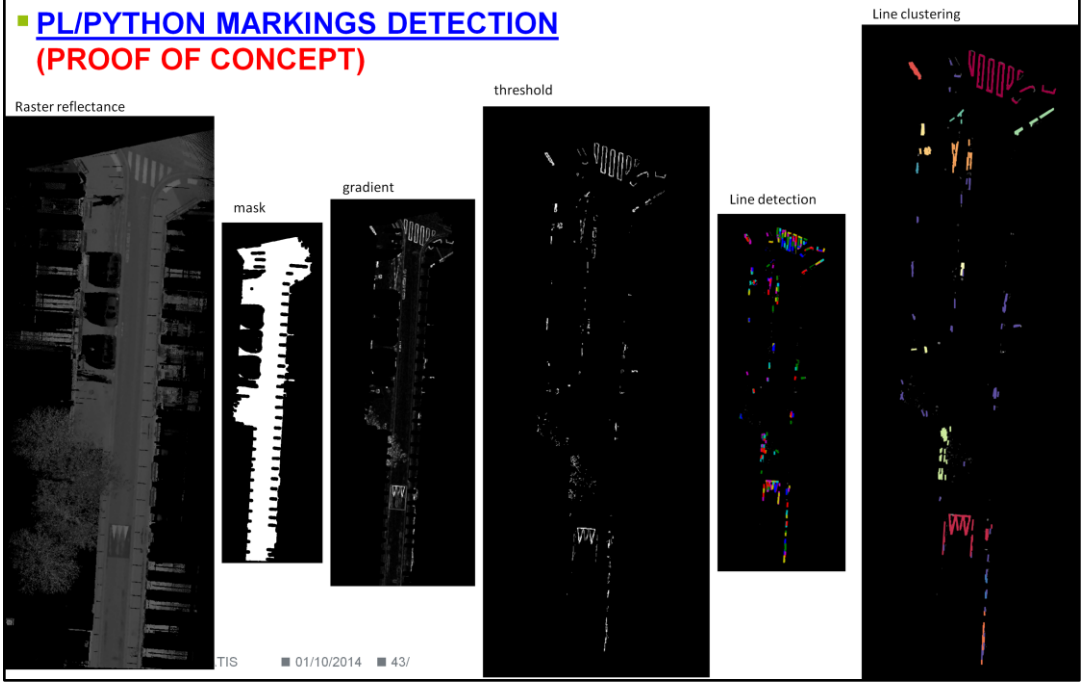
We demonstrate a more complex image processing pipeline **to detect street markings.**

The complexity comes from the mix between image processing (image world) and clustering (feature world).
This example also use both geometrical information and reflectance information (intensity of returned laser impulse).

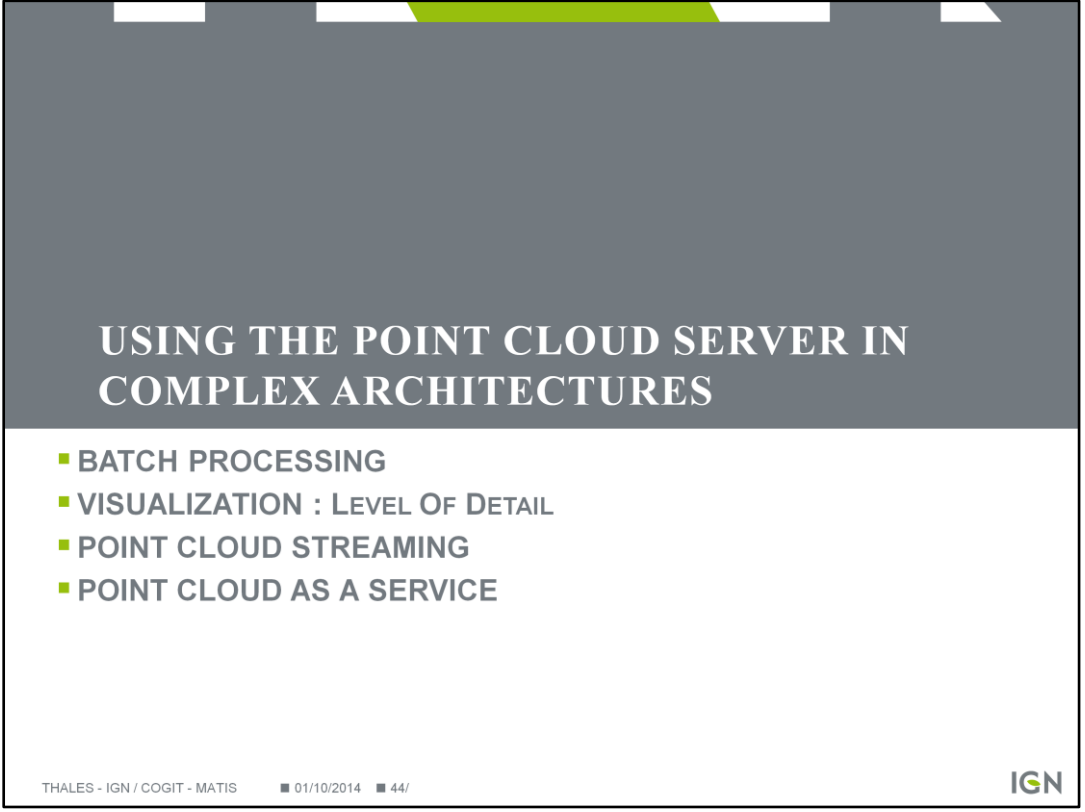
This kind of prototyping is very fast (~1 day, python beginner).

IN BASE PROCESSING

- **PL/PYTHON MARKINGS DETECTION (PROOF OF CONCEPT)**



Illustrations of the same method, de-zoom.



We presented some processing methods that take place in a DBMS.

Yet some very intense processing may better be done outside of the server, on a different computer.

The same is true for legacy programs necessitating files as input.

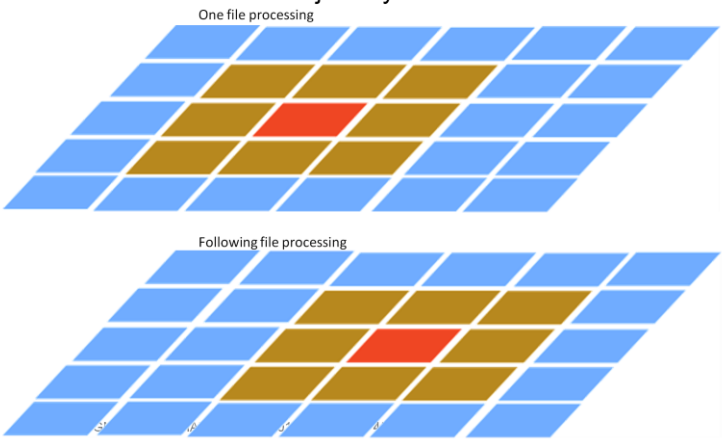
For this purpose the point cloud server can also be used in 2 more ways :

- To generate **on-the-fly point cloud files** for traditional programs
- To **stream points** for adapted programs.

The 2nd usage allows the construction of complex architecture.

USING THE POINT CLOUD SERVER IN COMPLEX ARCHITECTURES

- BATCH PROCESSING:
- How to launch (parallel) processing on overlapping areas?
  - Using file system : manual buffer. Need to load lot's of files **(x9)**
  - Crude grid pattern. Custom for every use case.
  - Difficult to use trajectory



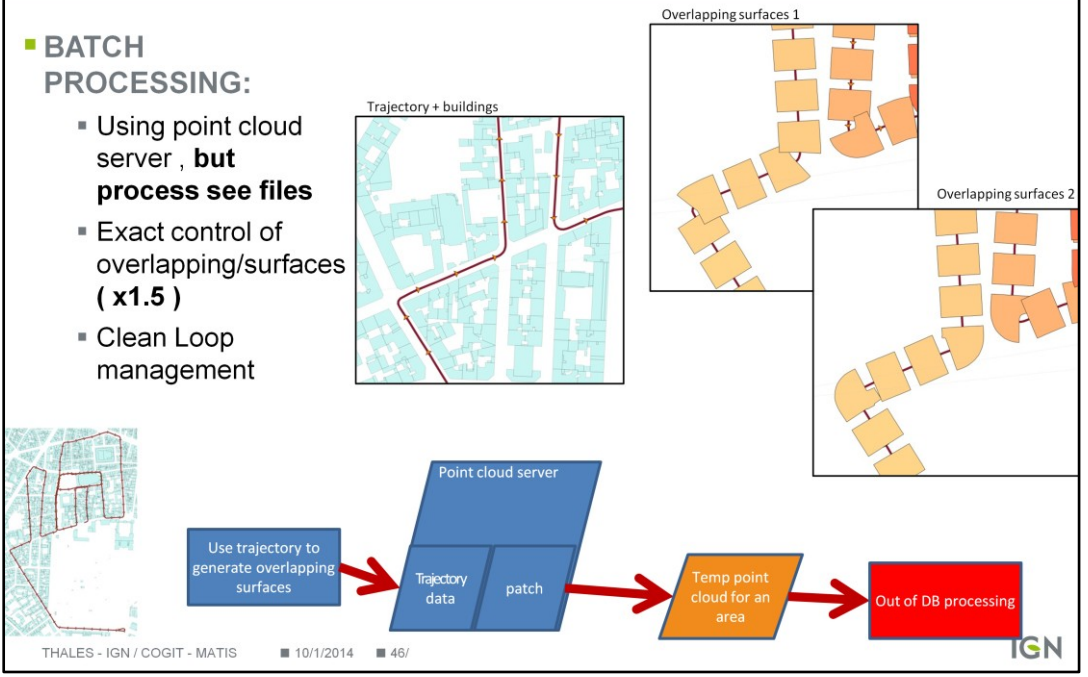
IGN

When processing big point clouds, a common problem is to **process small pieces** of the point cloud at a time.

The difficulty is that it may be necessary to have overlapping processing area to guarantee that points on the **border** are correctly handled.

If the point cloud is stored in a file system, this has to be done manually, and is very limited and inefficient.

The illustrations shows an example of processing a single file. All nearby files must also be loaded. So to process one file we must load 9 files !

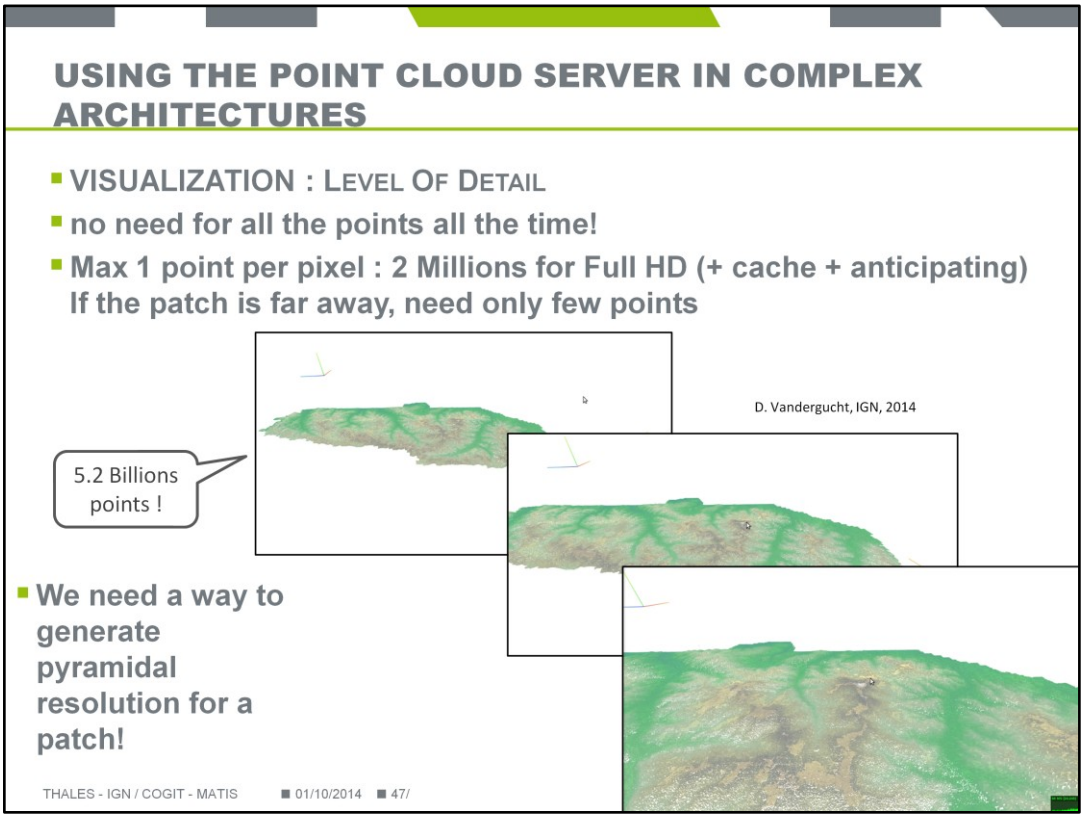


When doing batch processing with a point cloud server, it is easy to leverage other information already stored in the DBMS, like the trajectory of the acquisition vehicle.

This way, we can precisely control how much overlap we need, and **create on the fly temporary point cloud files on the disk** for the out of db processing.

Illustrations shows how a trajectory has been automatically cut into overlapping surfaces. Then the processing is done one surface at a time,

each time temporary writing on disk all the points in a surface.



A big challenge with point cloud data is visualization.

Visualization has been proven essential to understand data.

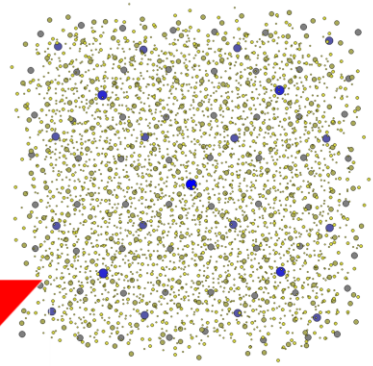
Yet printing 5 billion points on screen is not an option.

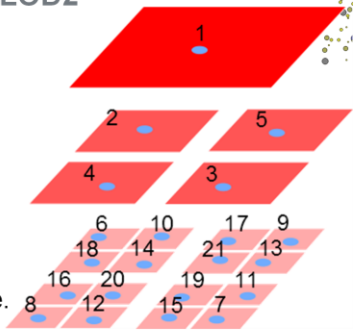
Instead we need to adapt the number of point we print depending of the distance of the patch.
If the patch is very **big** on screen, it needs to be **detailed**. If the patch takes only **one pixel** on screen, it needs only **1 point**.

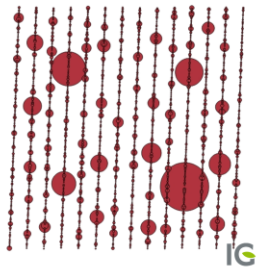
Thus we need visually acceptable method to **make the number of points per patch vary**.

USING THE POINT CLOUD SERVER IN COMPLEX ARCHITECTURES

- VISUALIZATION : LEVEL OF DETAIL FOR PATCH (PROOF OF CONCEPT)
  Full pl/pgsql
- Idea : no duplication :
- Simply write point in a given order
- The more we read, the more we get LODs
- LOD0 then LOD1 then LOD2 … then the rest
- What order?
  - For each level N:
  - 2^N points closer to center of level N of a Quad tree
  - Order in each level : random or inversed Z order curve.







We present a way to solve the problem of multi-resolution patch without duplication of data.

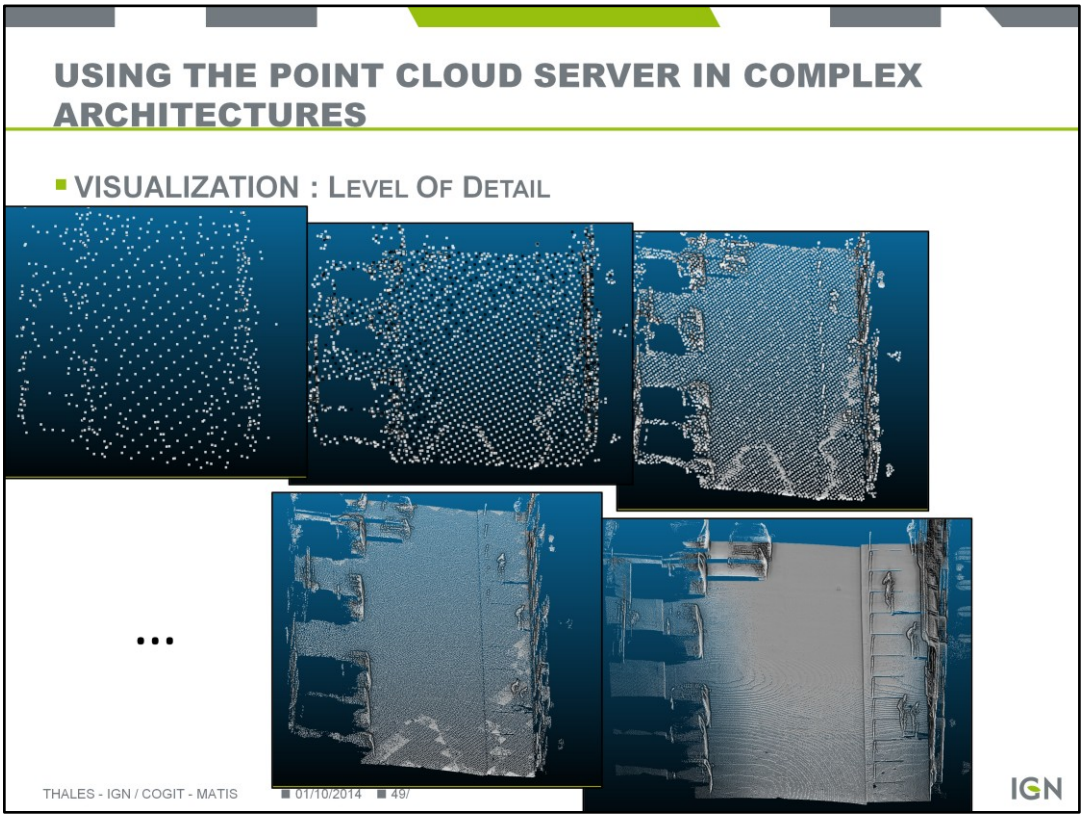
The idea is simply to store the points of a patch in a determined order.

Then, **reading more or less points from the beginning of the patch will make a patch more or less resolute.**

Right illustrations show a point cloud where the size of the point is proportional to its ordering.

We can see that if we take the point starting from the bigger and going toward the smaller, we always have a good approximation of the total point cloud.

The more we go toward the smaller points, the more detailed is the patch.



Here is an example on real data (some levels are skipped).

There is no duplication of data, only reordering.

In the first illustration, we read only one point per patch, in the second we read 1+4 points per patch, in the third 1+4+16 points per patch.

Last illustration shows all the points.

USING THE POINT CLOUD SERVER IN COMPLEX ARCHITECTURES

- ASYNCHRONOUS POINT CLOUD STREAMING TO BROWSER (PROTOTYPE)

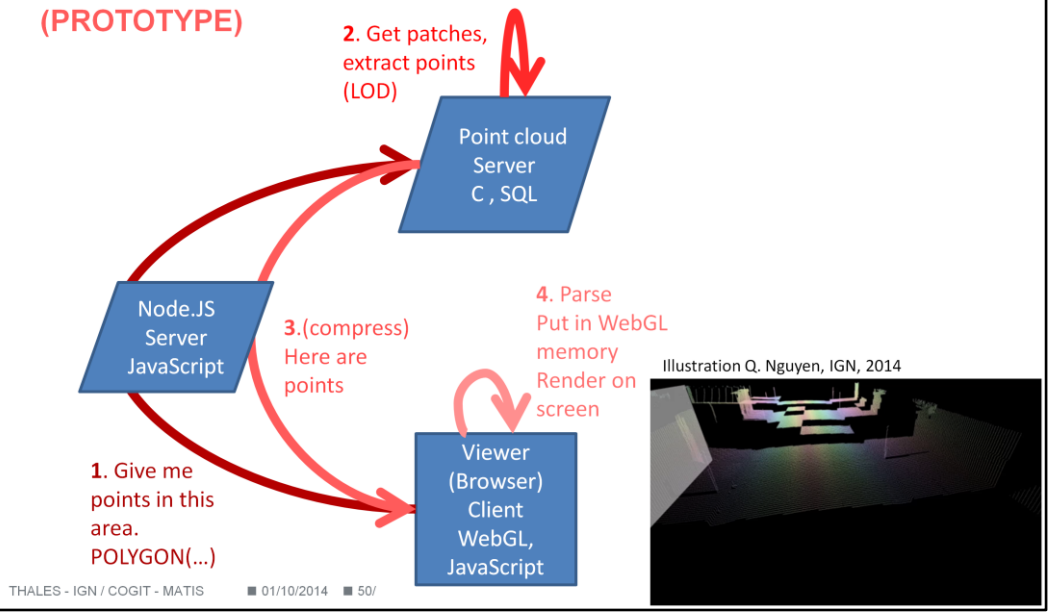


Illustration Q. Nguyen, IGN, 2014

We can use the point cloud server to asynchronously stream 3D points to a browser.

In this prototype, the **browser** is set at a **location**. It will then **asks** the point cloud server all the **points** that are around this location

(level of detail and complex spatial query are possible).

The point cloud server will **find** extremely fast which **patch** is close to the location, and **send** the corresponding **points**.

These points are then **rendered** on the browser screen using WebGL.

Using a node server allows the a-synchronicity : the **browser is not blocked when loading points**, it is a background task.

Schema has been greatly simplified.

USING THE POINT CLOUD SERVER IN COMPLEX ARCHITECTURES

- ASYNCHRONOUS POINT CLOUD STREAMING TO BROWSER (PROTOTYPE)

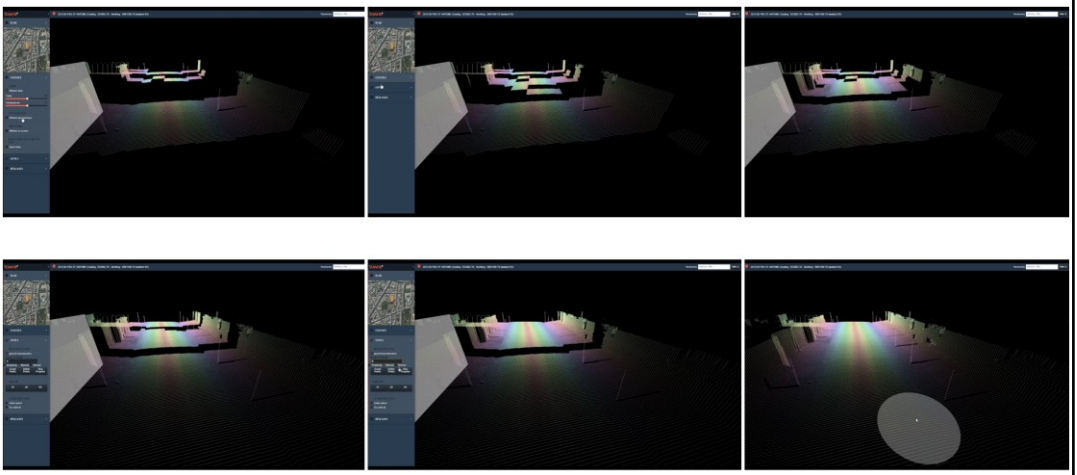


Illustration Q. Nguyen, IGN, 2014

IGN

Illustrations (reading left to right top to bottom) of the points being streamed to the browser.

Points are streamed patch by patch (1 m^3)

The camera point of view change because the user can explore the data, and does not have to wait for all the data to be loaded.

USING THE POINT CLOUD SERVER IN COMPLEX ARCHITECTURES

- INTERACTIVE 3D SIDEWALK RECONSTRUCTION (PROTOTYPE)

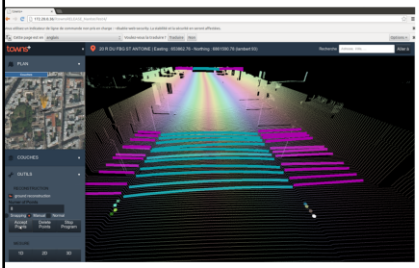
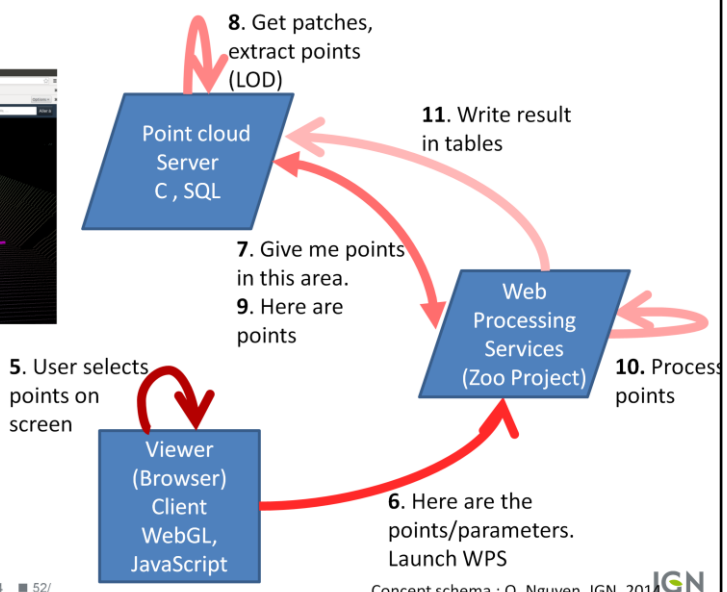


Illustration A.Hervieu, IGN, 2014



Concept schema : Q. Nguyen, IGN, 2014

We present an even more complex architecture.
Here the point cloud server is used to stream points to browser for visualization,

And also to stream points to a web processing service.

The web processing service is a method to semi-automatically reconstruct road surface and sidewalk surface.

The method starts by selecting a few points on the border of sidewalk.

Then the method automatically moves forward and try to find sidewalk again.

In fact the interaction is more complex, and the user can stop/start/redirect the method interactively.

We omitted the previous part of the schema for more clarity, and also simplified this part.

USING THE POINT CLOUD SERVER IN COMPLEX ARCHITECTURES

INTERACTIVE 3D SIDEWALK RECONSTRUCTION (PROTOTYPE)

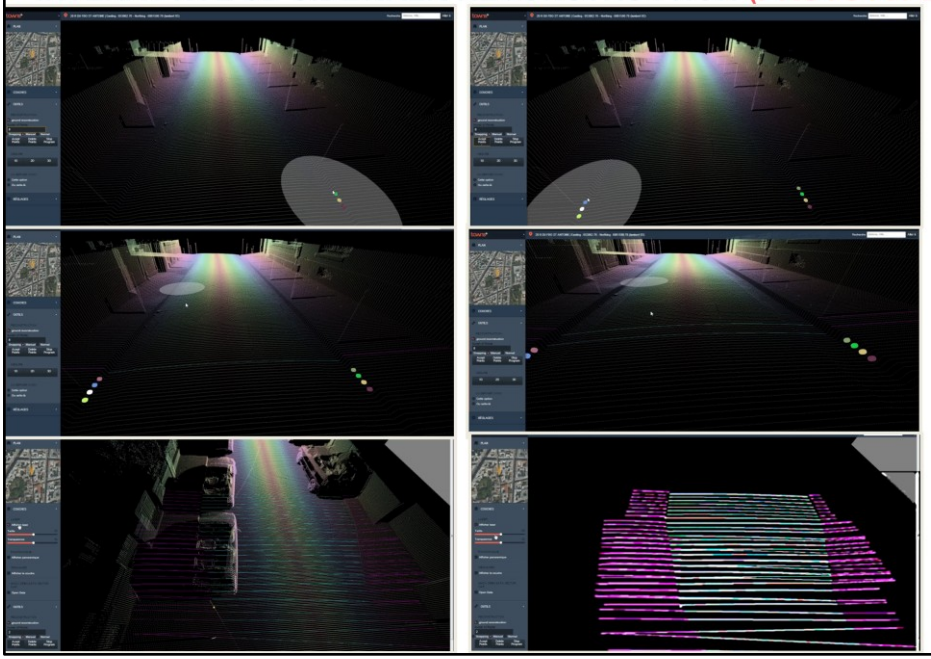


Illustration A.Hervieu, IGN, 2014

IGN

Illustration reading left to right top to bottom.

Illustrations 1 and 2: the user selects some starting points on the curbstone to initialize the algorithm.

Illustrations 3 and 4: the method models road and sidewalk surfaces, and moves forward.

Illustrations 5 and 6: results : road and sidewalk modeled.

CONCLUSION

CONCLUSION

- POINT CLOUD SERVER WITH POSTGRES/POSTGIS/POINTCLOUD
  - VERY POWERFUL.
  - NO NEED TO BE LINUS TORVALDS TO GET RESULTS.
  - IT'S ONLY THE BEGINNING.
  - MANY THINGS TO IMPROVE

CONCLUSION

- SOME EXAMPLE OF THINGS TO IMPROVE IN POINTCLOUD
  - I/O PERF
  - CONVERSION TO/FROM RASTER
  - CONVERSION TO/FROM MESHES
  - EFFICIENTLY GET ONLY SOME ATTRIBUTES OF POINTS IN A PATCH
  - EFFICIENTLY GET ONLY SOME POINTS IN A PATCH
  - USE BLOB TO STORE PATCHES (ALLOWS EFFICIENT STREAMING, AVOID TOAST)
  - INTEGRATE PCL
  - 3D BBOX, ORIENTED 3D BBOX.
  - …

RESSOURCES

- DOCUMENTATION
- TOOLS

DOC

- **[Demantké, 2014] J. Demantké, 2014: Thesis**
- **[Ramsey,2013] P. Ramsey presentation of PointCloud : boundlessgeo.com/wp-content/uploads/2013/10/pgpointcloud-foss4-2013.pdf**
- **[Oosterom, 2014] : Point cloud data management**
  P. van Oosterom, S. Ravada, M. Horhammer, O. Marinez Rubi, M. Ivanova, M. Kodde and T. Tijssen
  IQmulus Workshop on Processing Large Geospatial Data, 8 July 2014, Cardiff, Wales, UK
- **[Cura,2014] R: a free software for statistical analysis**
  Introduction to the analysis of geographical data with R; 2014 ; R. Cura, H. Mathian
- All terrestrial point clouds are from Stereopolis, IGN.

Thanks for the help of everybody.

TOOLS

- POSTGRES / POSTGIS
- POINTCLOUD
- CLOUDCOMPARE
- RPLY

- PDAL
- GDAL
- QGIS

- PL/PYTHON : PYTHON-PCL, SCIKIT-LEARN, PYTHON GDAL, NUMPY, SCIPY
- PL/R : NNCLUST