A POSTGRES SERVER FOR POINT CLOUDS STORAGE AND PROCESSING

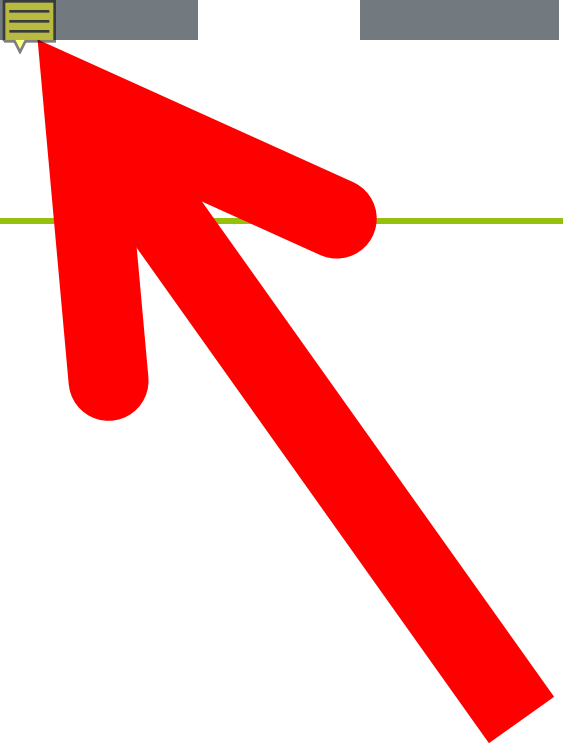
PRESENTED AT POSTGRESL PARIS SESSION 6.

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

RÉMI CURA Remi dot Cura … gmail

JULIEN PERRET – NICOLAS PAPARODITIS – GILDAS LE MEUR

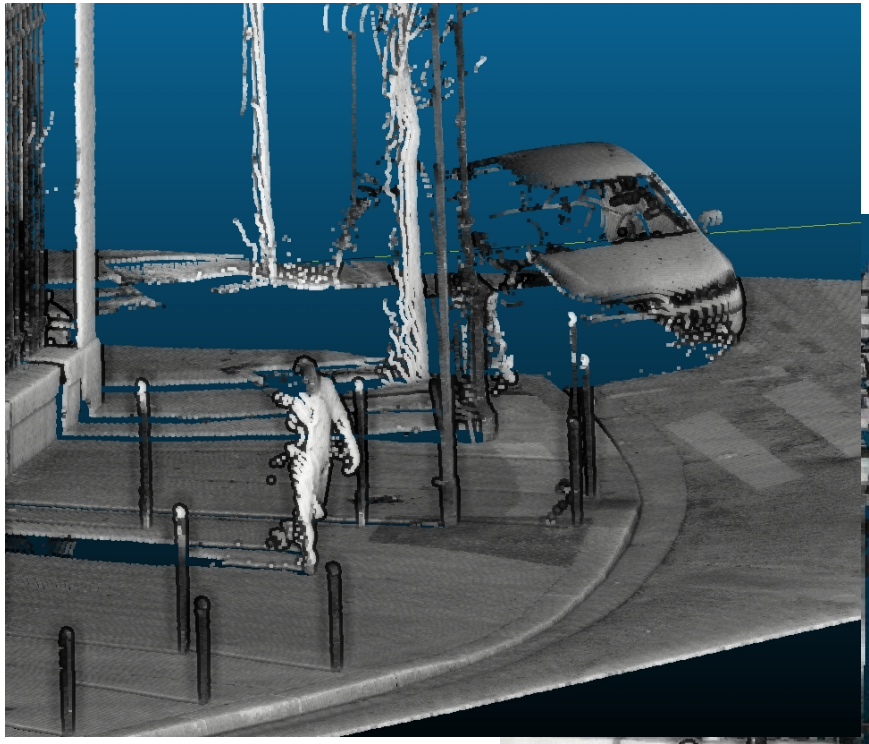
NOTE

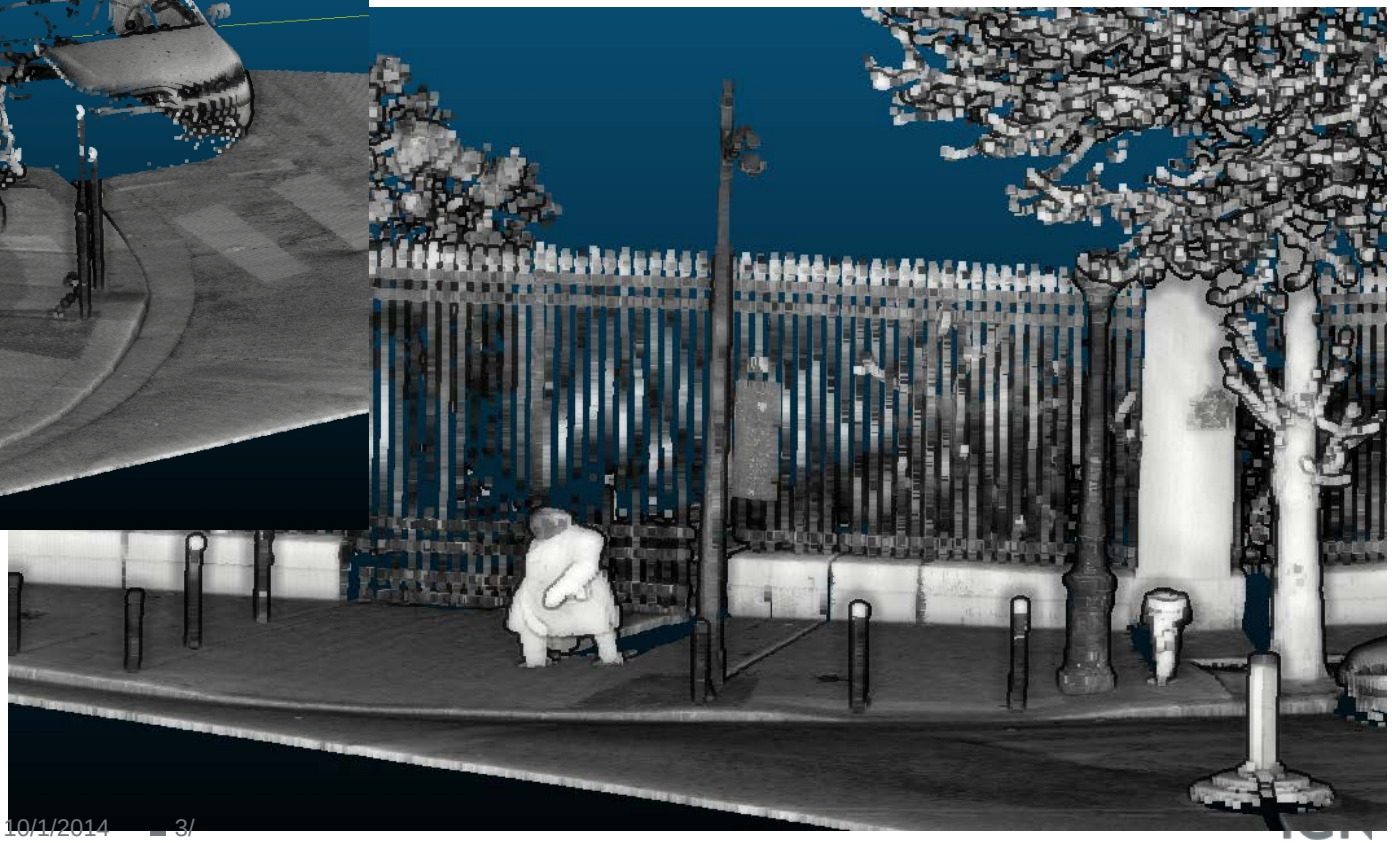


- **COMMENTS HERE ON EVERY PAGE**

INTRODUCTION TO POINT CLOUDS

- **NEW KIND OF HUMAN: POINT CLOUD HUMAN!**





CONTENTS

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

INTRODUCTION TO POINT CLOUDS

- WHAT ARE POINT CLOUDS
- ORDER OF MAGNITUDE

INTRODUCTION TO POINT CLOUDS

Riegl reflectance

Riegl reflectance

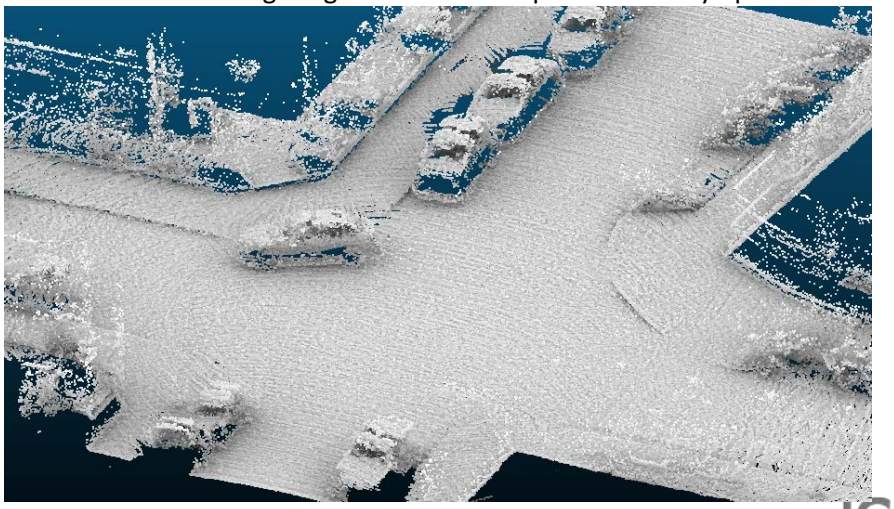
INTRODUCTION TO POINT CLOUDS

Riegl reflectance

Riegl reflectance

Velodyn Laser, viewing reflectance

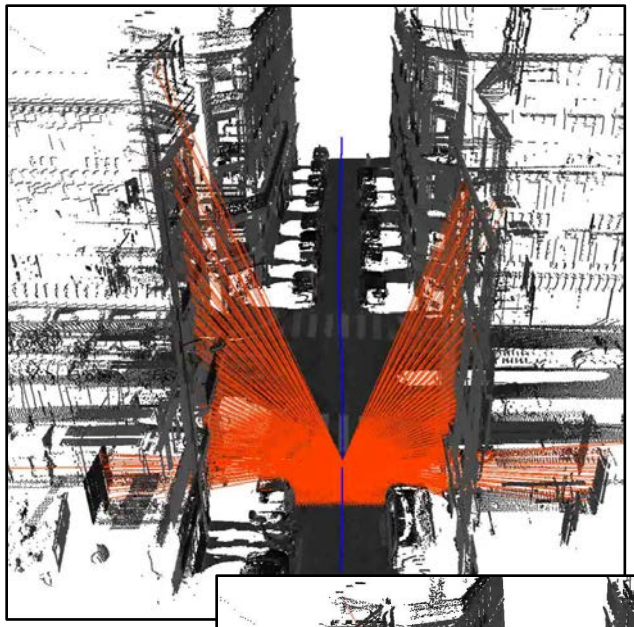
Ambient occlusion lighting with Cloud Compare on Velodyn points



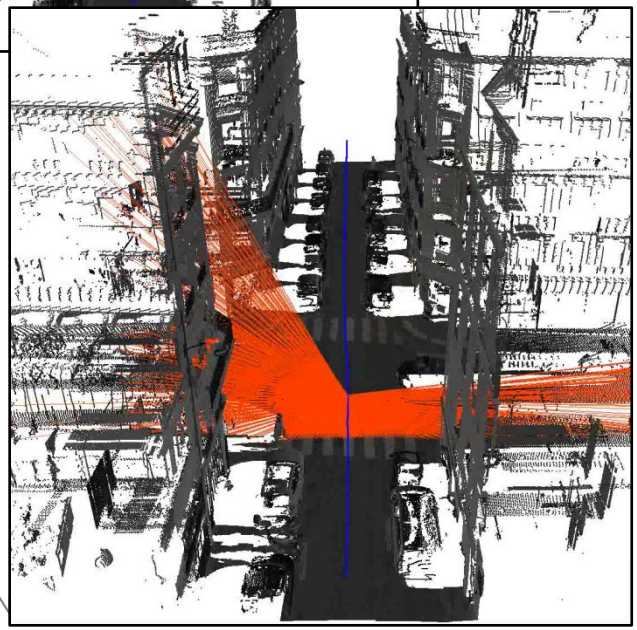
INTRODUCTION TO POINT CLOUDS

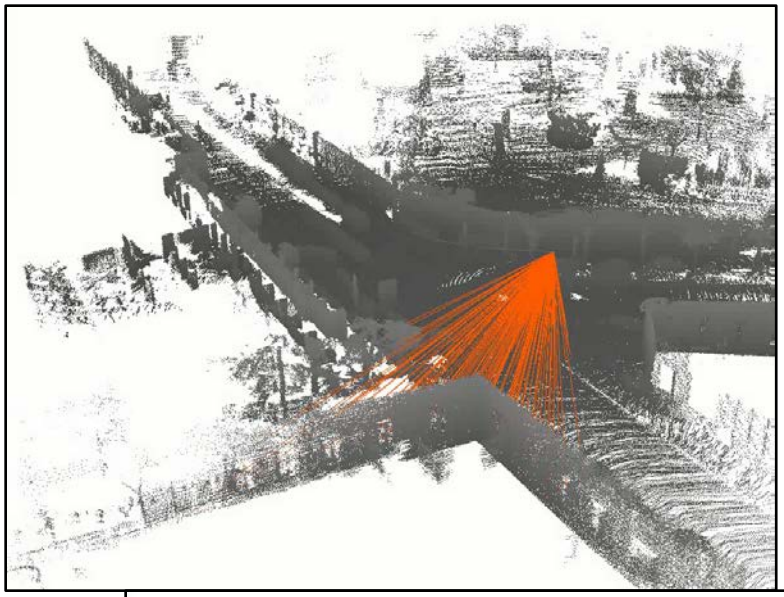
- **What are Point Clouds?**
- **A set of unordered 3D points with attributes resulting from a sensing operation.**
  - Unordered: don’t know who the neighbors are .
  - Attributes: ex: intensity of returning light, class id, angle …
  - Sensing: physical sensing of reality, not like a vector point representing 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)

INTRODUCTION TO POINT CLOUDS

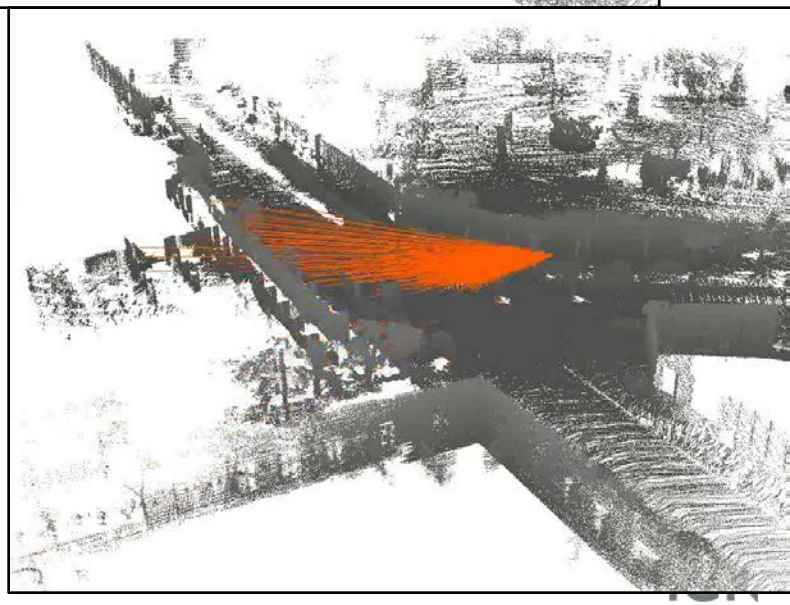


Riegl laser





Velodyn
Laser



Illustrations : F. Monier, IGN

INTRODUCTION TO POINT CLOUDS

EXAMPLE : AERIAL AND TERRESTRIAL POINT CLOUDS

Non constant very high density

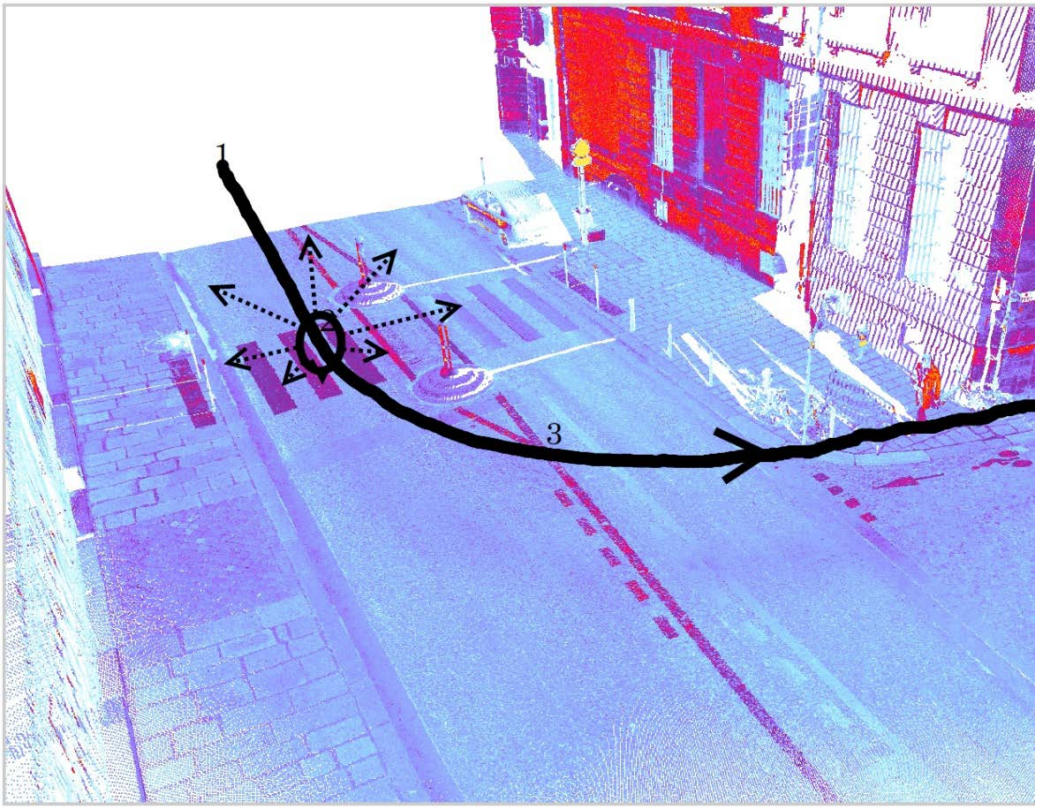


Image from J. Demantké Thesis, 2013

Almost constant low density

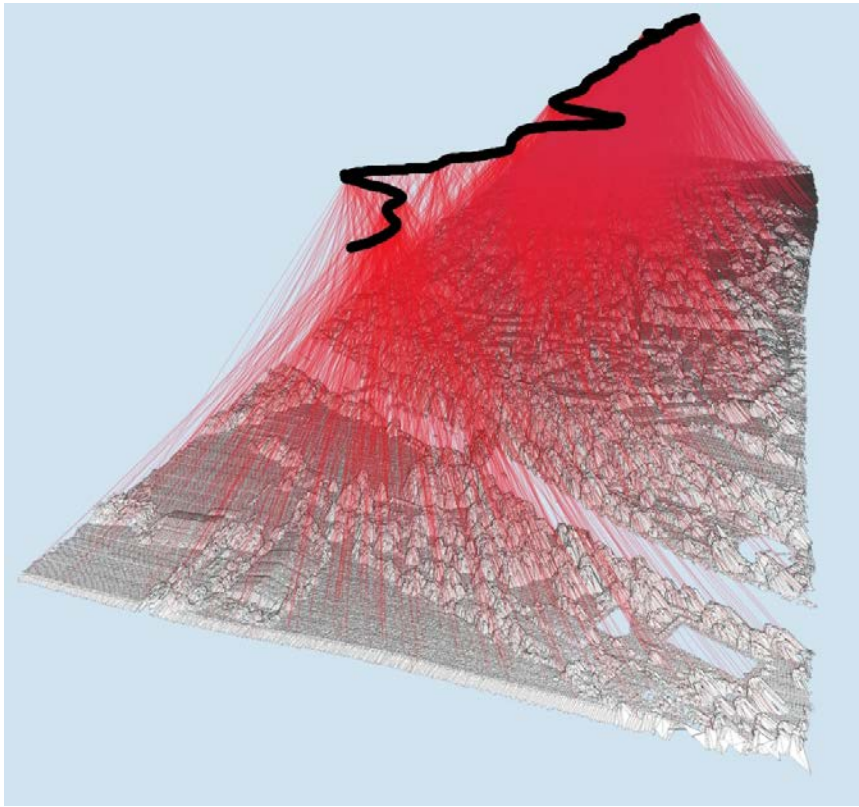


Image from J. Demantké Thesis, 2013

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.

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.

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

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).**

Coverage	Folder	Folder number
	/s1	4^0
	/s1/s1..4	4^1
	/s1/s1..4/s1..4	4^2

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!**

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

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?

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.

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

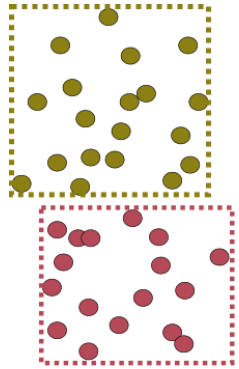
- STORING POINT CLOUDS IN DBMS : BLUE OR RED PILL?
- WHAT IS POINTCLOUD?
- EFFICIENT STORAGE
- FAST QUERYING
- FAST LOADING
- ORDER OF MAGNITUDE

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
Billions of row	->	**big** indexes
Double/float storage	->	**wasted** storage
1 row = 1 point	->	**no** compression
Different laser	->	**no** compatibility
Different laser	->	**custom** code everywhere

Millions of row	->	1 table per dataset
Millions of row	->	small indexes
Custom bit width	->	efficient storage
1 row = N point	->	compress by redundancy
Different laser	->	1 family type
Different laser	->	postgres extension

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

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>
```

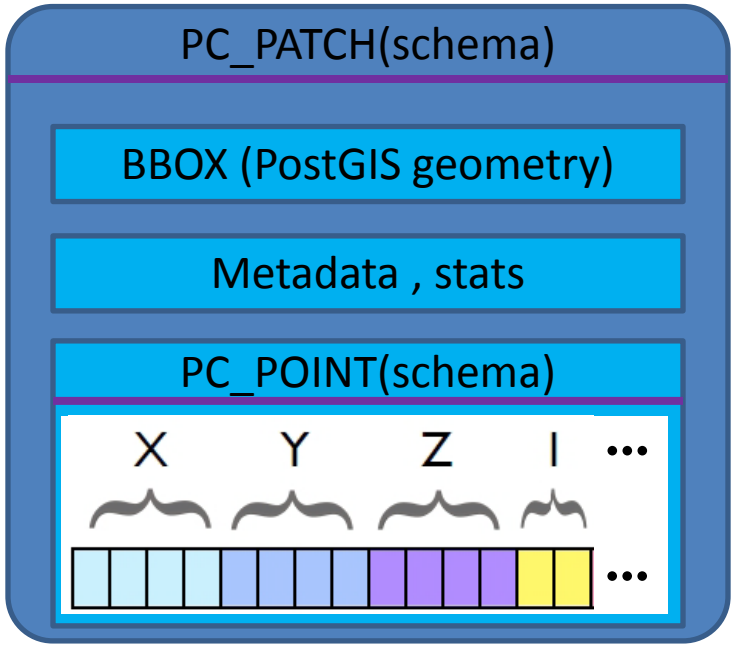


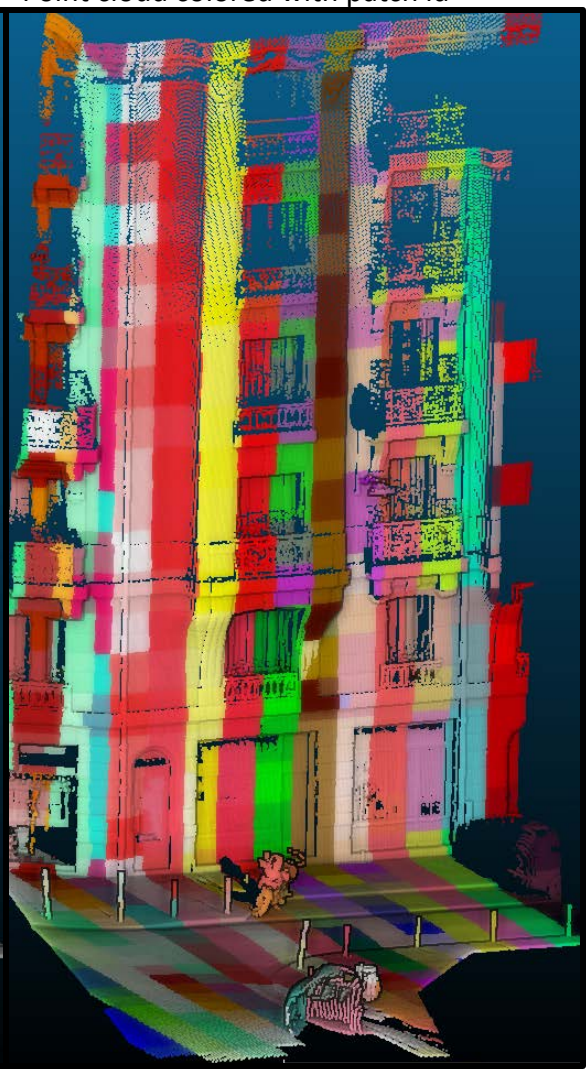
Illustration
from [Ramsey,2013]

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



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)

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**

EFFICIENT STORING/QUERYING/LOADING IN POSTGRES

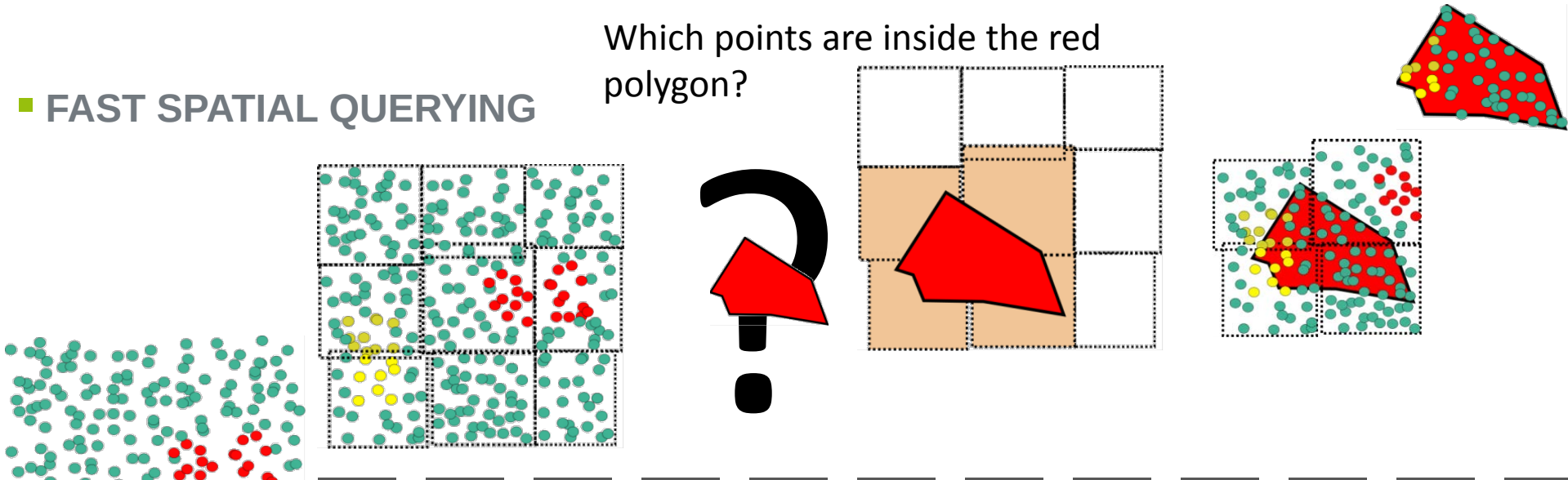
- **FAST SPATIAL QUERYING**
  - We index bounding box of patches.
    CREATE INDEX ON patch USING GIST(patch::geometry)
  - Index on several S.R.S possible
    CREATE INDEX ON patch USING GIST(ST_Transform(patch::geometry,4326))
- **FAST ATTRIBUTE QUERYING**

  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')

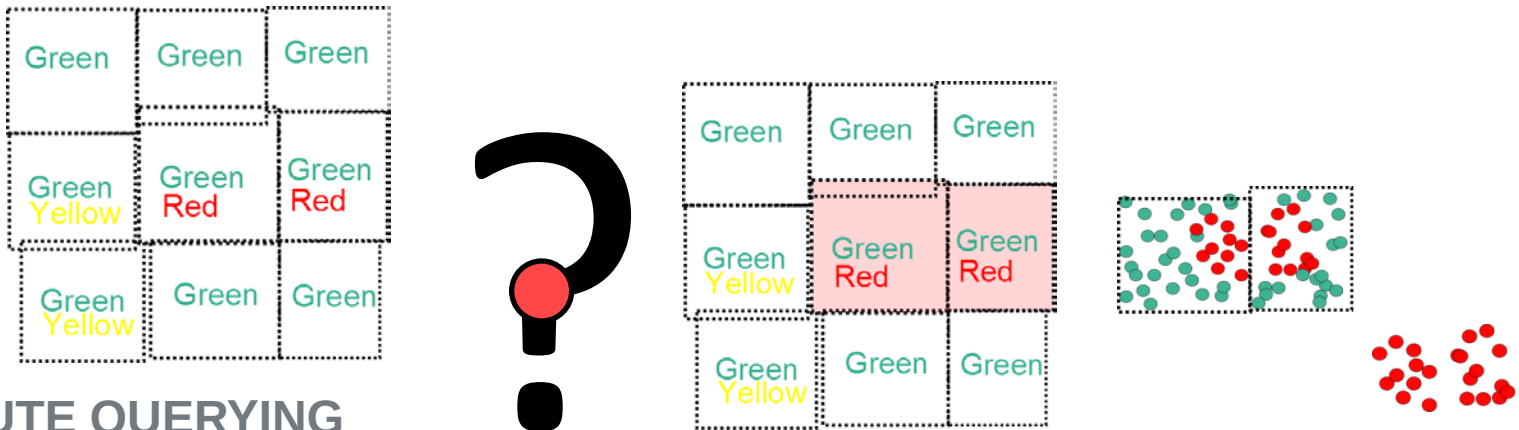
EFFICIENT STORING/QUERYING/LOADING IN POSTGRES

- **FAST SPATIAL QUERYING**

Which points are inside the red polygon?



- **FAST ATTRIBUTE QUERYING**

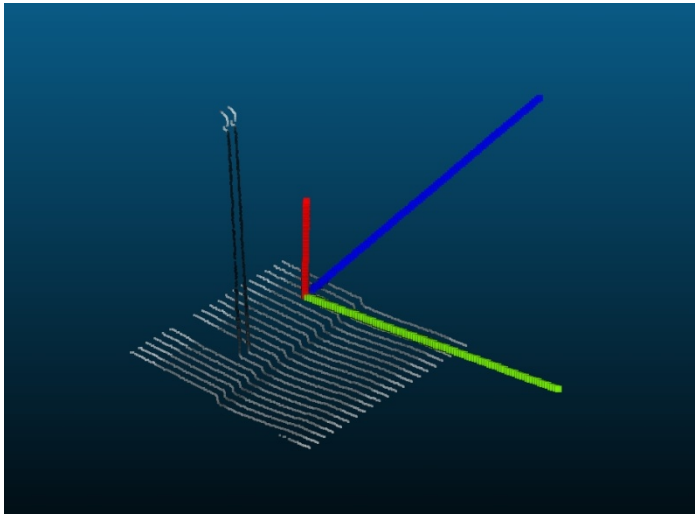


Which points have color red?

EFFICIENT STORING/QUERYING/LOADING IN POSTGRES

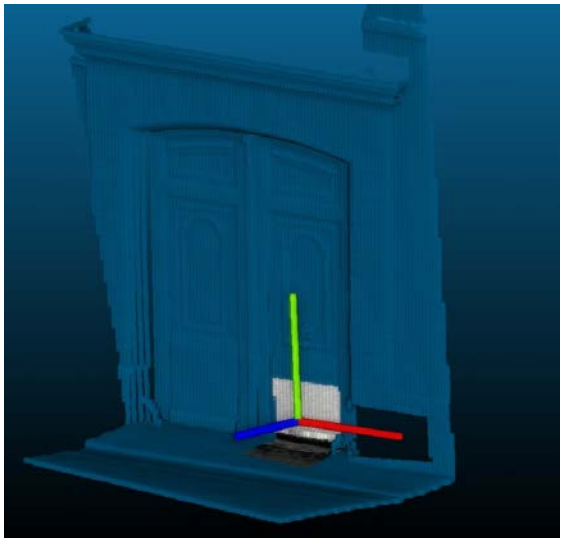
- **BONUS : POSTGRES SUPER-COOL FEATURE :**
- **FAST FUNCTION QUERY : INDEX ON ND DESCRIPTORS**
  - Example :
  - Define a function computing ND descriptors of a patch
    - example : **How vertical is the patch?** (based on I.C.A.)
  - Create an index on it

Verticality descriptor for a patch containing a pole



(**0.1** ,0.002,0.008)

Verticality descriptor for a patch half façade half ground

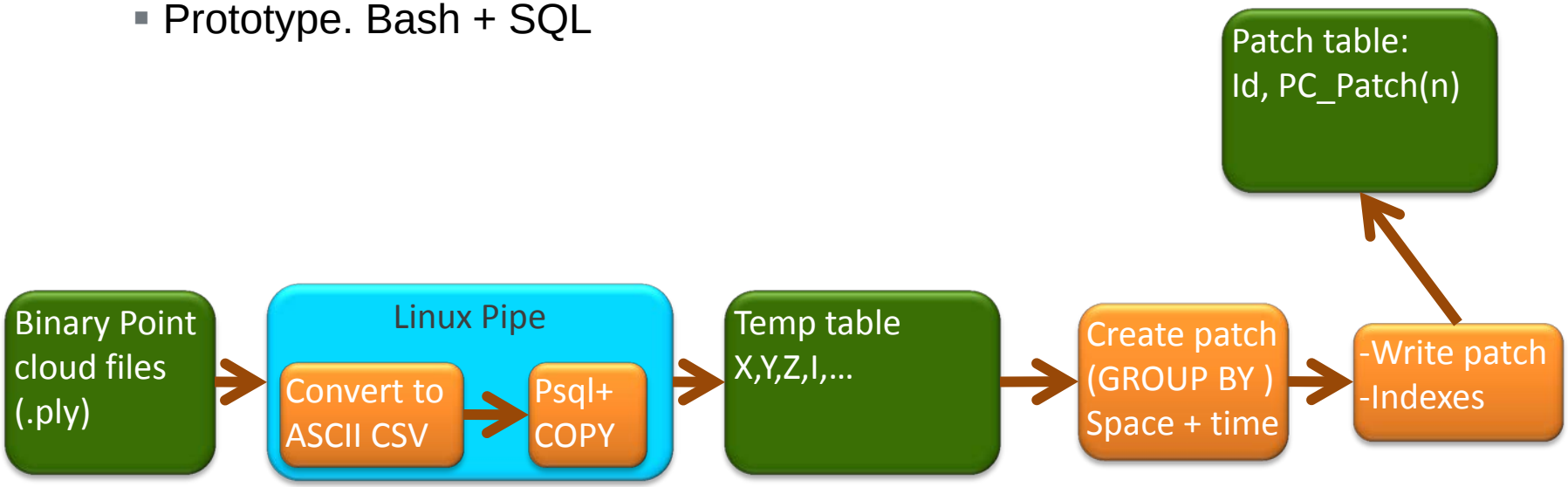


(**0.7** ,0.001,0.005)

EFFICIENT STORING/QUERYING/LOADING IN POSTGRES

- **FAST LOADING IN POINTCLOUD**
- **For Simple test : use PDAL (GDAL for point clouds wannabe)**

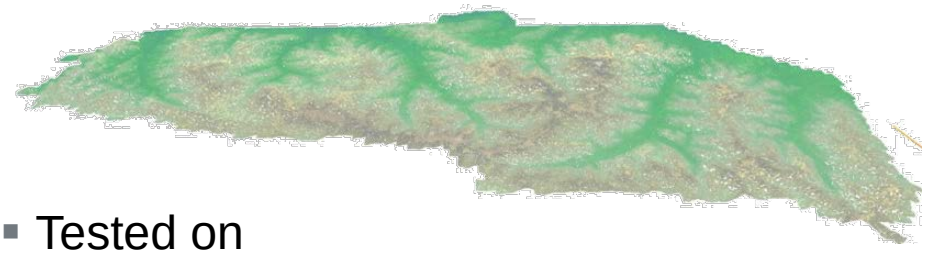
- **An example of fast loading: project PointCloud_in_db (PROTOTYPE)**
  - Parallel loading into a server : we can **load as fast as we acquire** data !
  - Prototype. Bash + SQL



EFFICIENT STORING/QUERYING/LOADING IN POSTGRES

ORDER OF MAGNITUDE

Vosges aerial acquisition, 3D rendering



- 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.

Paris terrestrial acquisition 2013
Above view

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

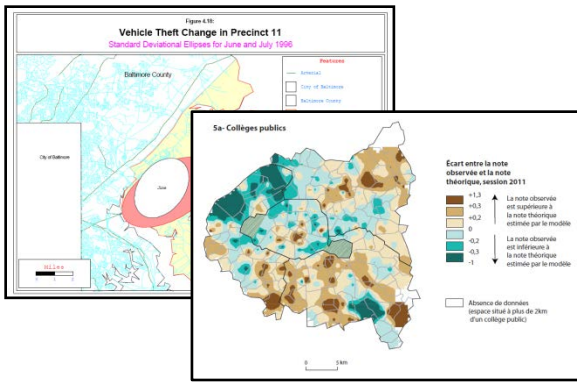
IN BASE PROCESSING

- IN BASE PROCESSING : WHAT IS SO COOL ABOUT POSTGRES?
- PL/R : CLUSTERING
- PL/PYTHON : CLUSTERING
- PL/PYTHON : PLAN DETECTION
- PL/PGSQL : PATCH->RASTER
- PL/PYTHON : IMAGE PROCESSING : DETECTION

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

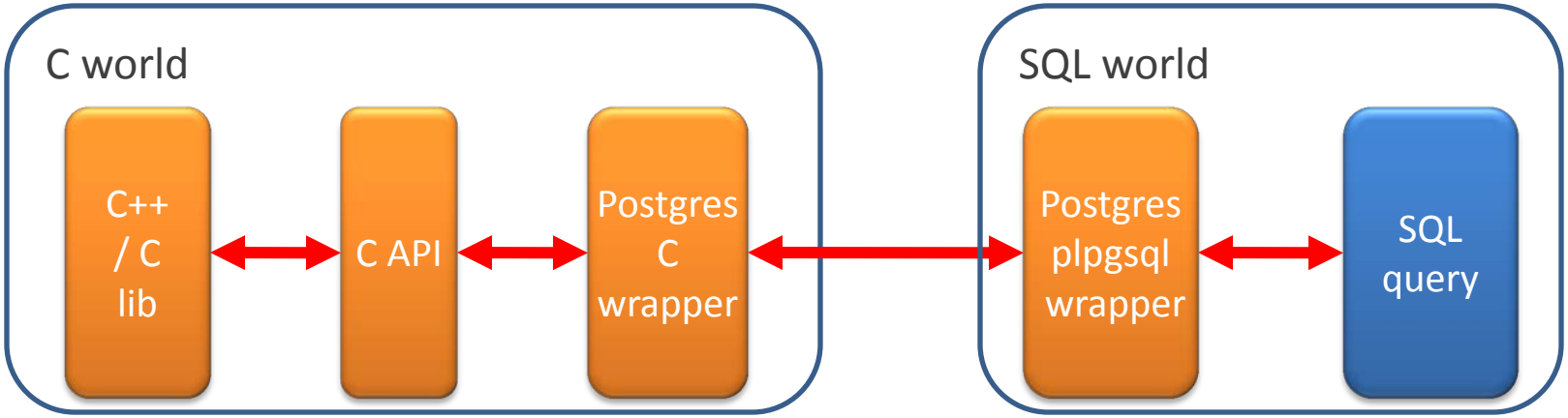
Illustrations from [Cura,2014]



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

IN BASE PROCESSING

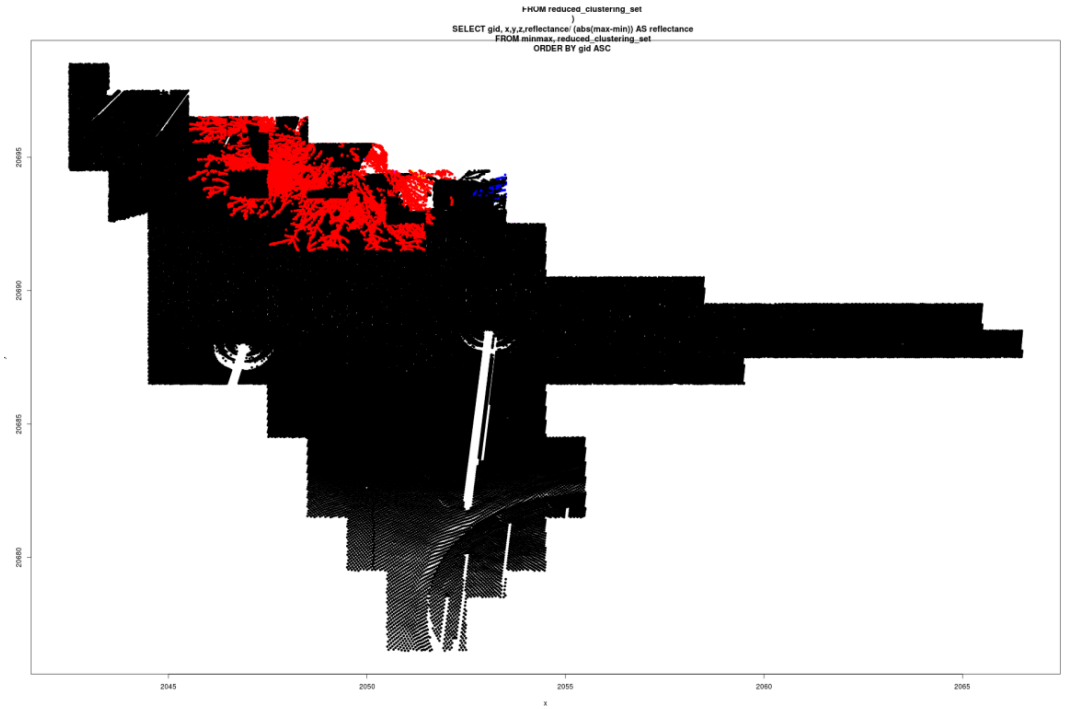
- **IN BASE PROCESSING : WHAT IS SO COOL ABOUT POSTGRES?**
- **Easy to create a postgres extension**
  - ~~High level languages~~
  - C , C++
    - Very efficient (memory/CPU)
    - Integrating into Postgres forces common meta API/data
    - This allows re-use / complex processing pipeline.



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

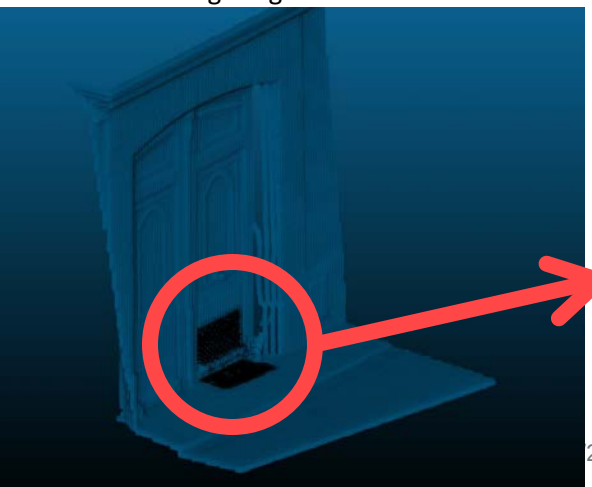


IN BASE PROCESSING

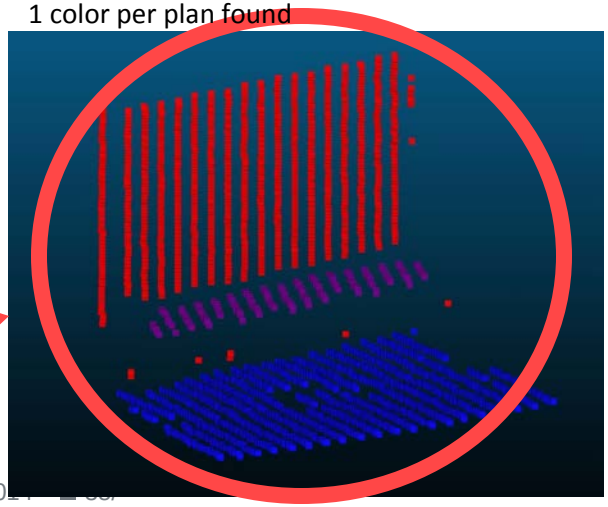
- **PL/PYTHON PLAN DETECTION (PROOF OF CONCEPT)**
- **For list of points (double[ ])**
  - Convert list to numpy array
  - Convert numpy array to python-pcl point cloud
  - Until no more plan is found
    - Find a plan using p-Ransac
    - Remove points of the plan from the cloud

Context : 3D + reflectance

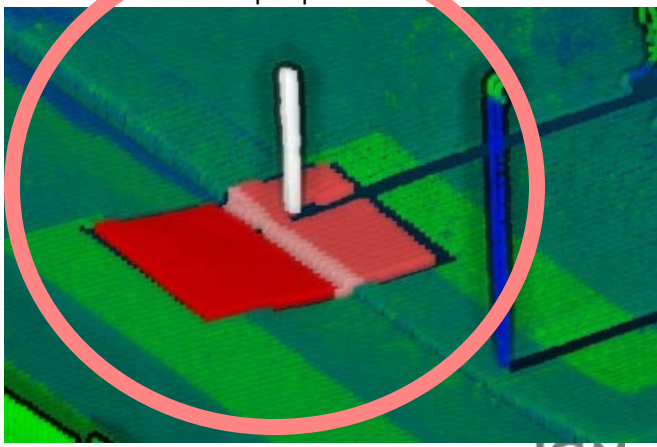
Context : 3D + lighting



1 color per plan found



1 color per plan found white to red

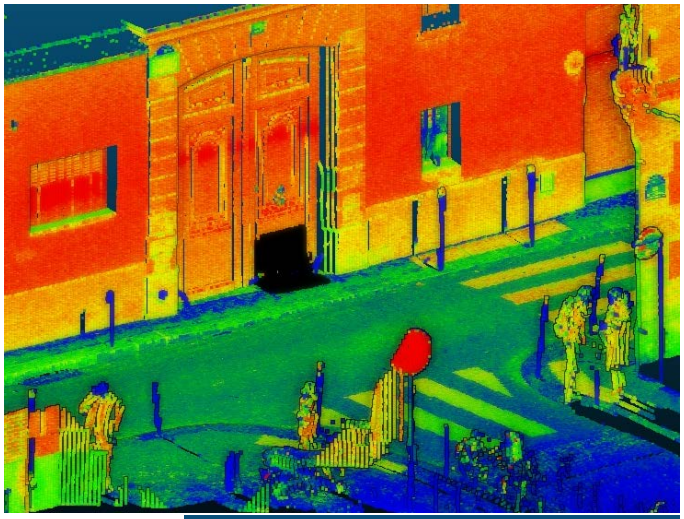


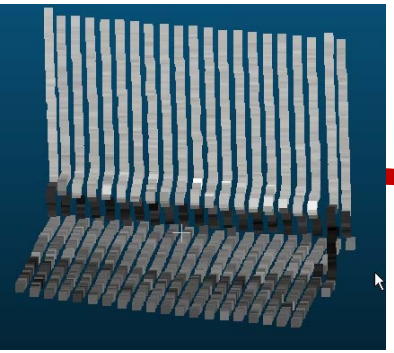
IN BASE PROCESSING

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

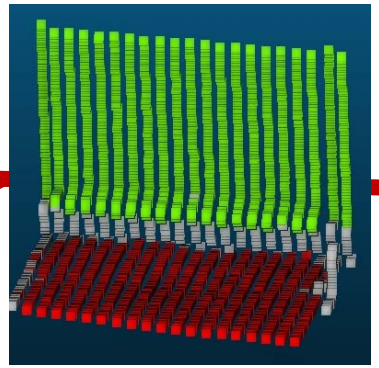
- **For list of points + reflectance (double[ ])**
  - Convert list to numpy array
  - (Compute normals using python-pcl)
  - Cluster using sklearn.cluster.DBSCAN
  - Join clustering label to point id

context

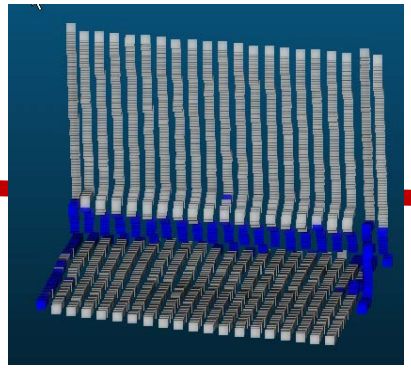




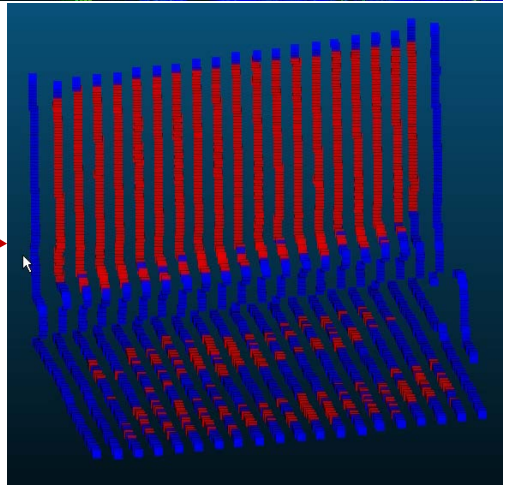
3D + reflectance



labels

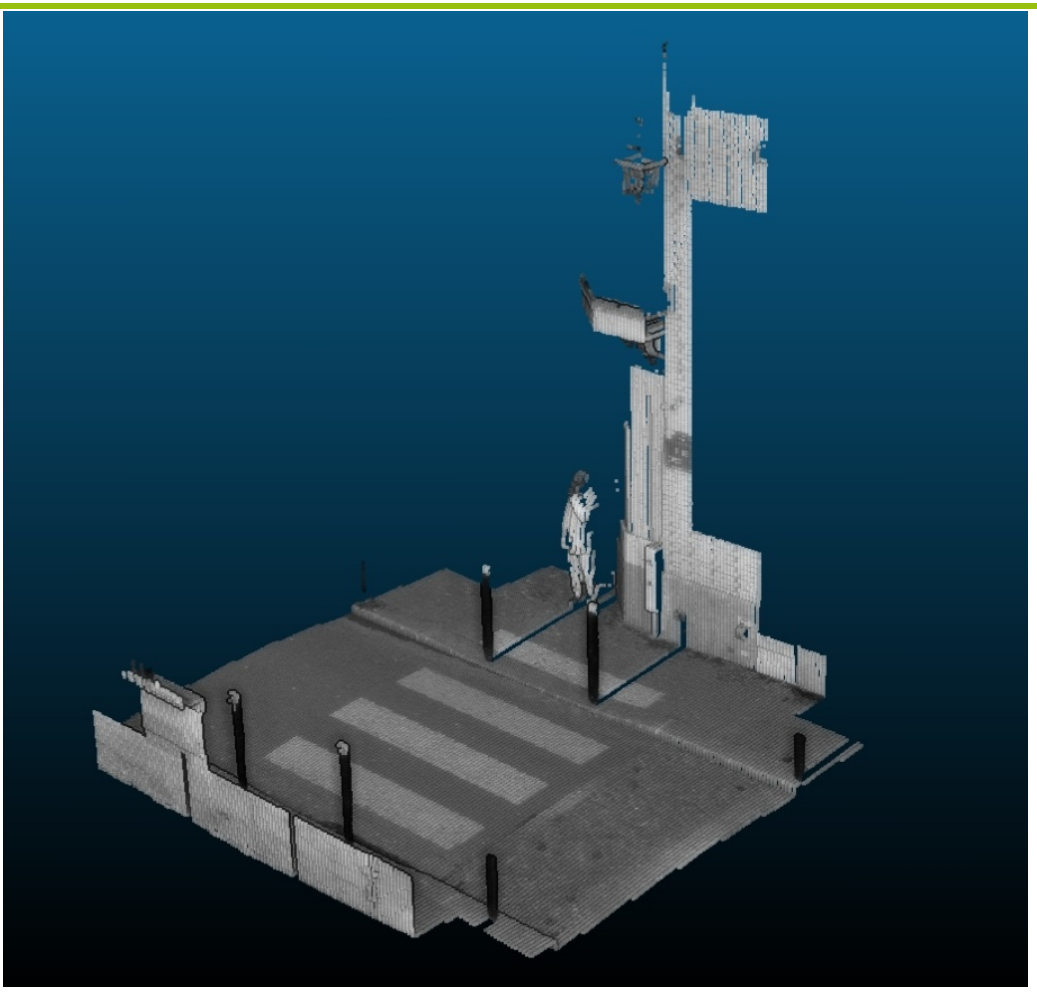


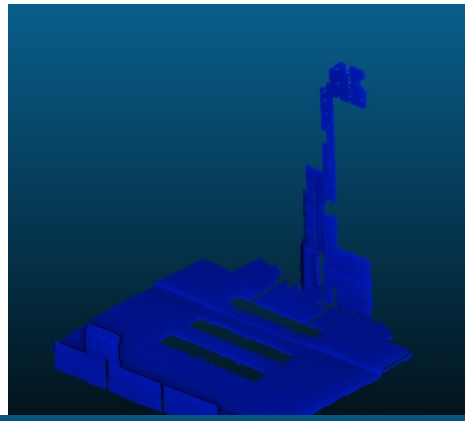
outliers

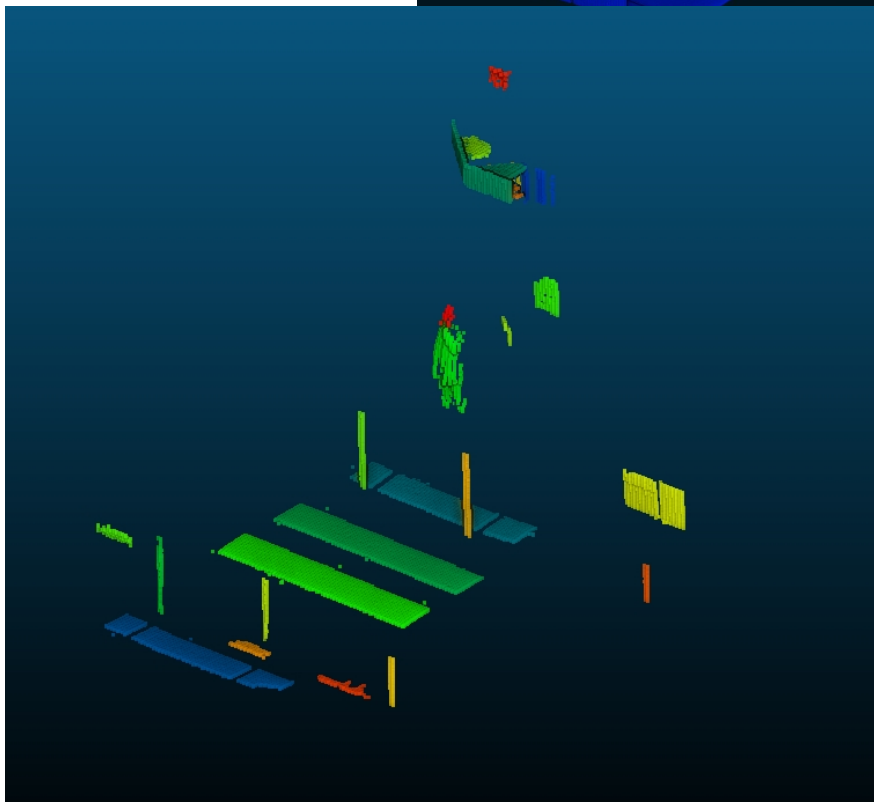


core

IN BASE PROCESSING





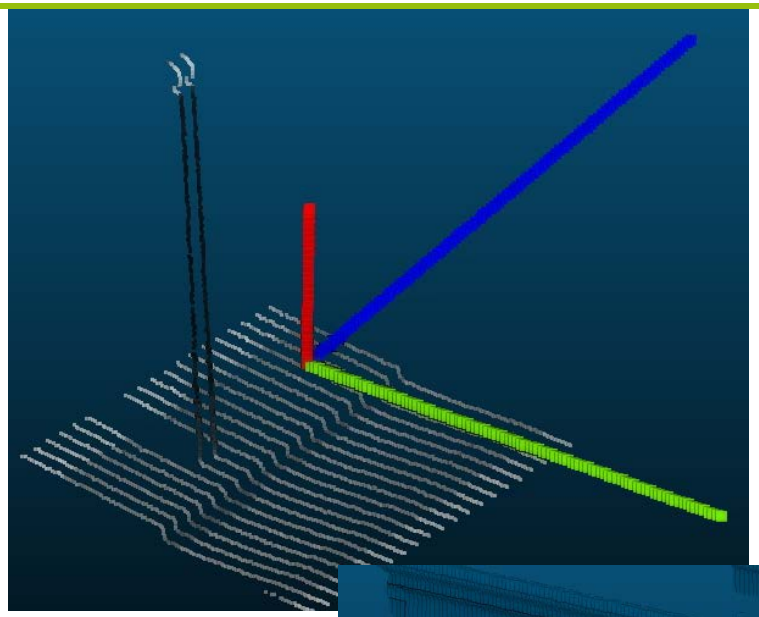


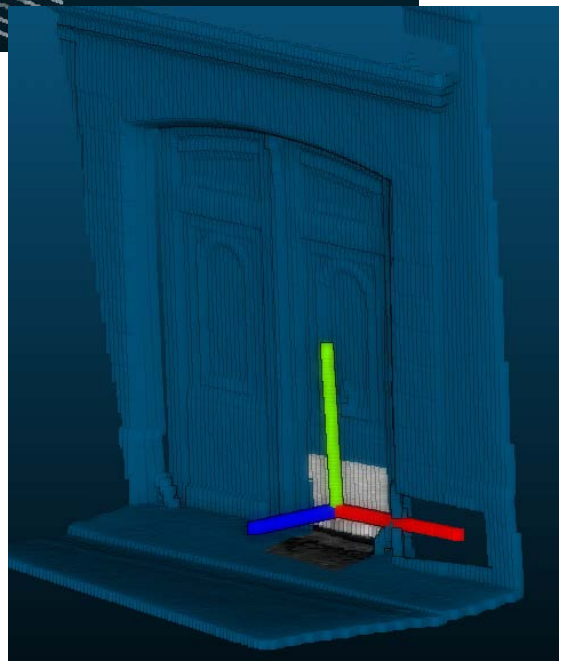
IN BASE PROCESSING

- **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**



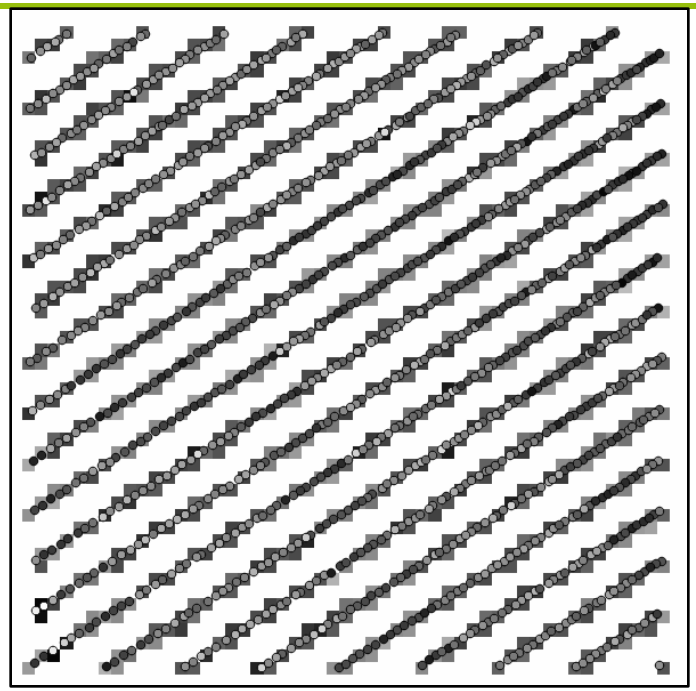


IN BASE PROCESSING

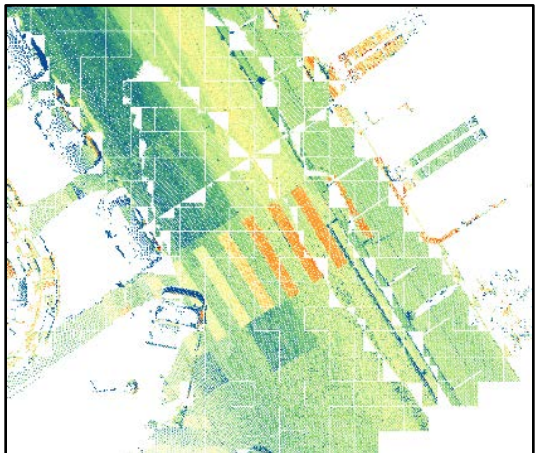
- **PL/PGSQL PATCH->RASTER (PROOF OF CONCEPT)**

- **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)**



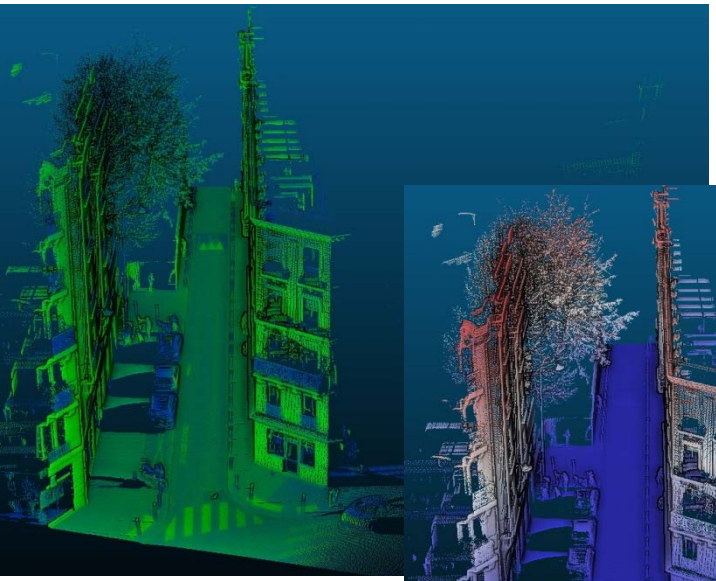
Concept : raster+points. Color = reflectance. Qgis



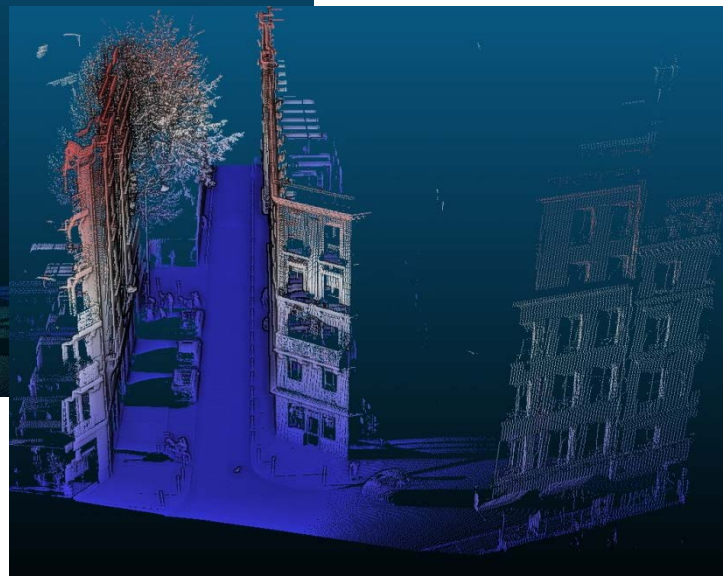
A street part viewed in QGIS trough GDAL PostGIS raster driver

IN BASE PROCESSING

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



reflectance

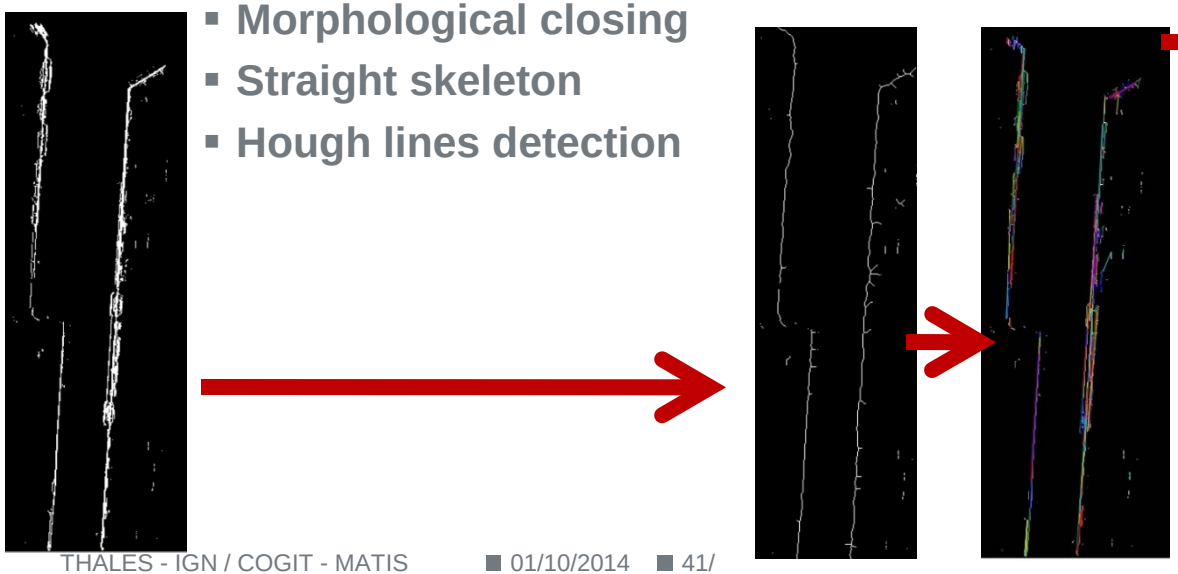


Relative height

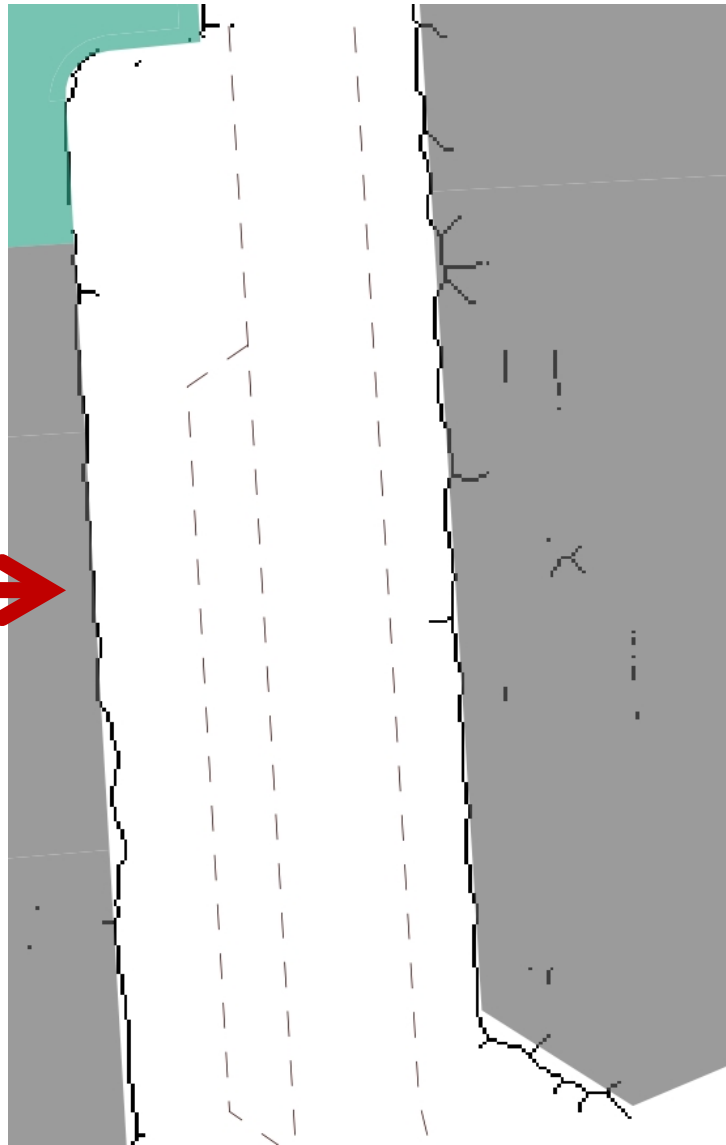
Accum. raster. Lot's of "NoData"

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



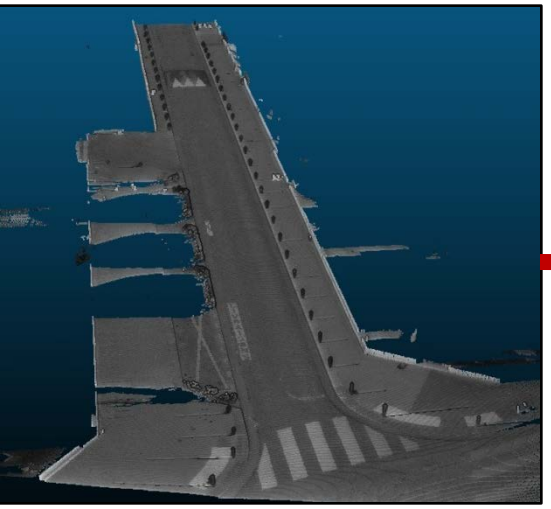
Result with ground truth



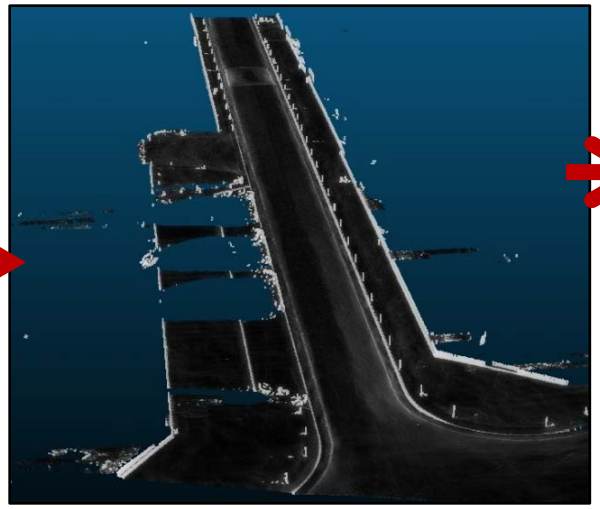
IN BASE PROCESSING

- **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)

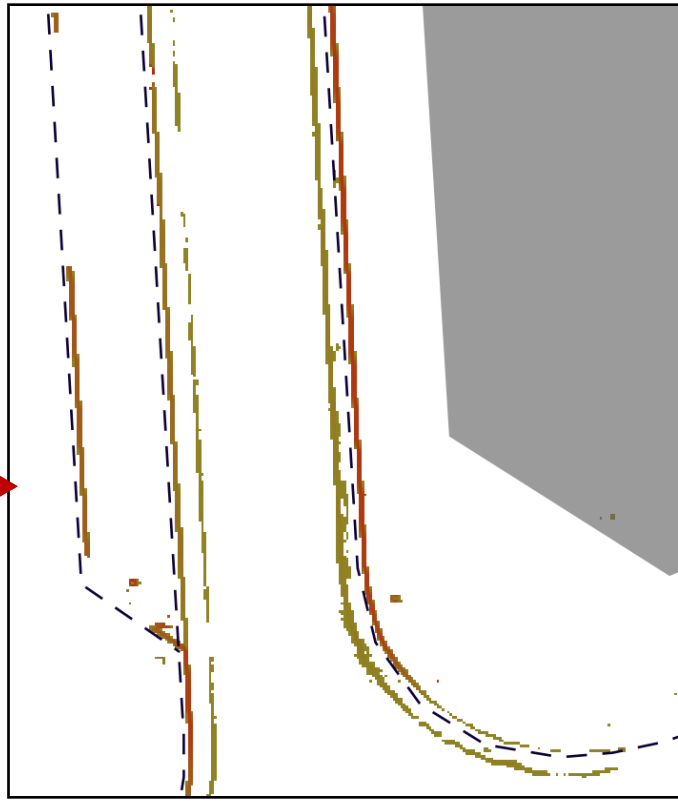
Reflectance at wheel level



Gradient at wheel level

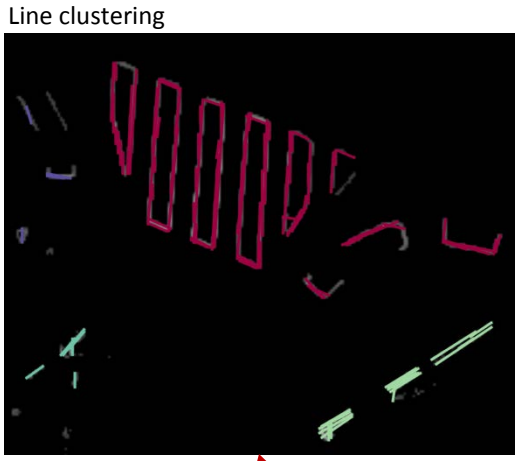


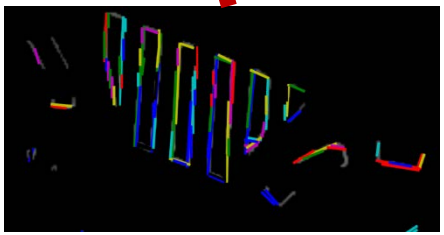
Result (orange to red) with ground truth (blue)

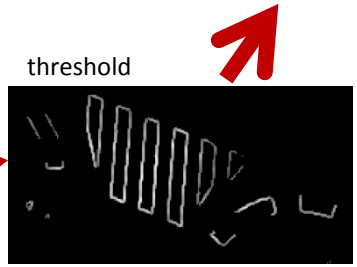


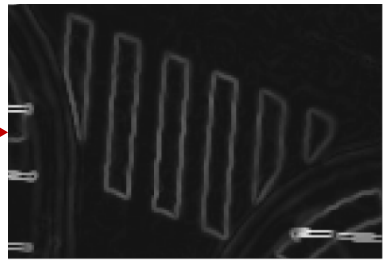
IN BASE PROCESSING

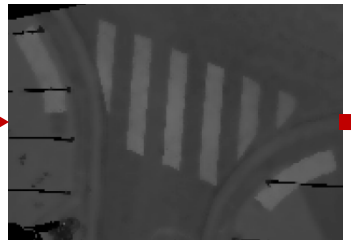
- **PL/PYTHON MARKINGS DETECTION (PROOF OF CONCEPT)**
  - For a raster (reflectance,height) of a point cloud projected
    - Compute height gradient
    - Remove pixels near NoData and/or non flat
    - Smooth reflectance while preserving edges (bilateral filter)
    - Compute reflectance gradient, threshold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

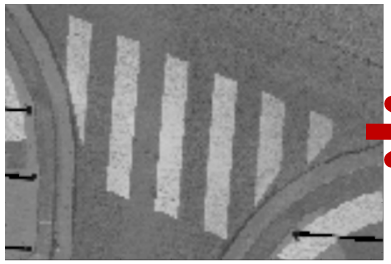












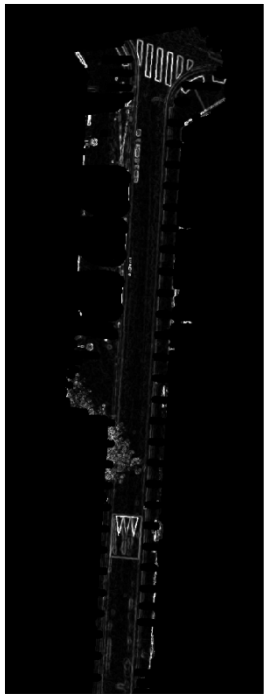
IN BASE PROCESSING

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

Raster reflectance

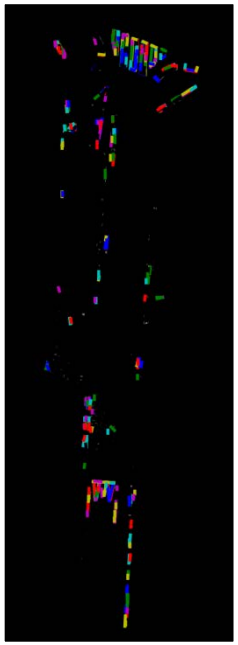
mask

gradient

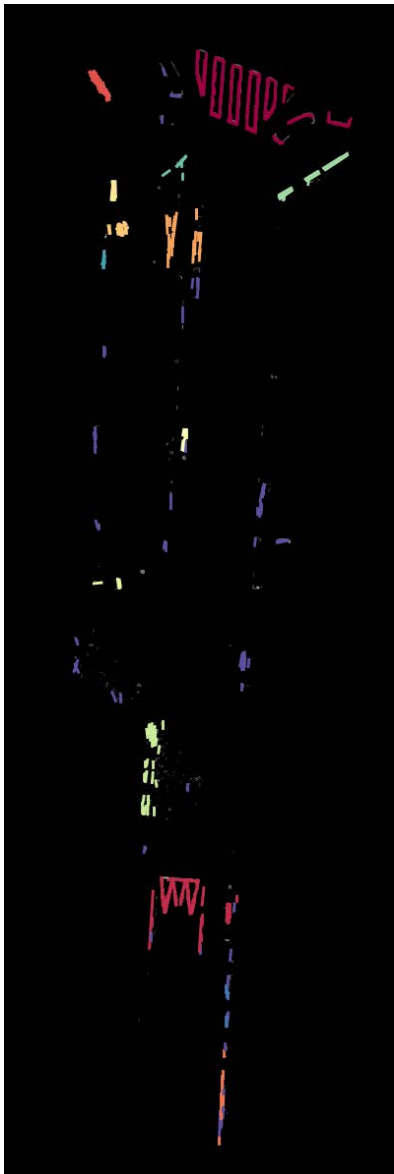


threshold

Line detection



Line clustering



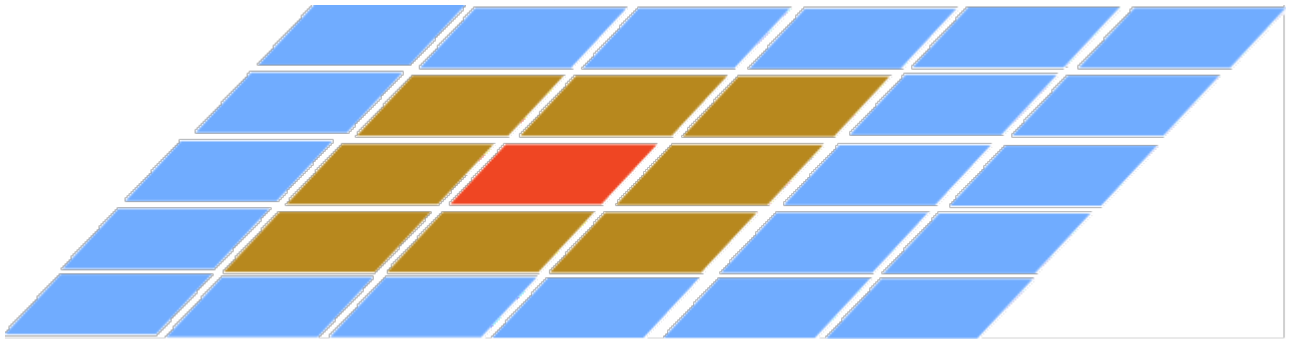
USING THE POINT CLOUD SERVER IN COMPLEX ARCHITECTURES

- BATCH PROCESSING
- VISUALIZATION : LEVEL OF DETAIL
- POINT CLOUD STREAMING
- POINT CLOUD AS A SERVICE

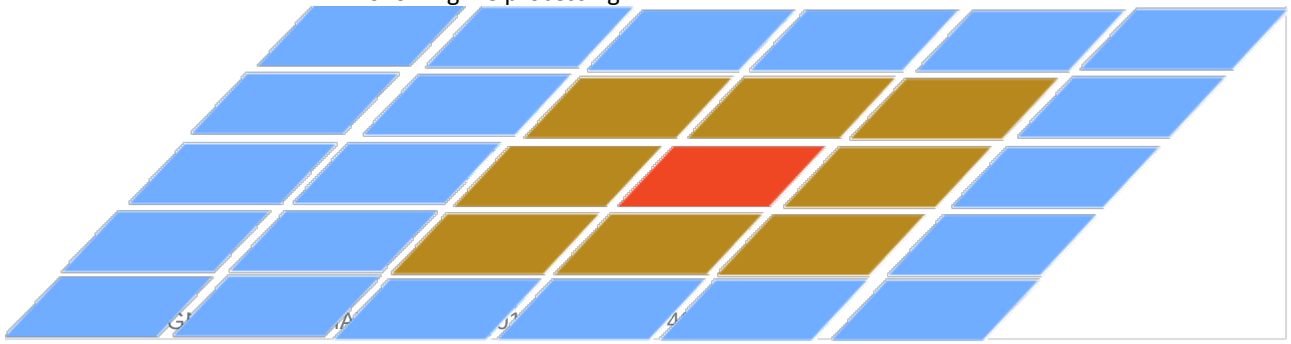
USING THE POINT CLOUD SERVER IN COMPLEX ARCHITECTURES

- **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

One file processing



Following file processing

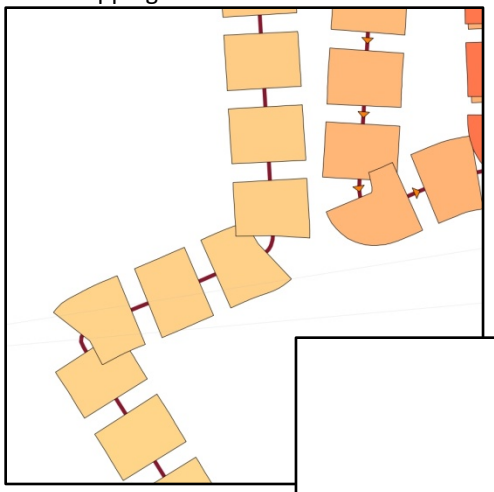


USING THE POINT CLOUD SERVER IN COMPLEX ARCHITECTURES

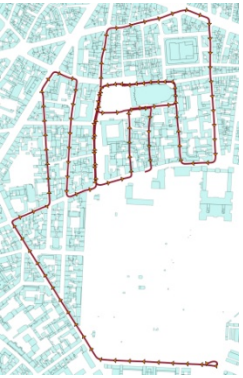
- **BATCH PROCESSING:**
  - Using point cloud server , **but process see files**
  - Exact control of overlapping/surfaces **(x1.5)**
  - Clean Loop management

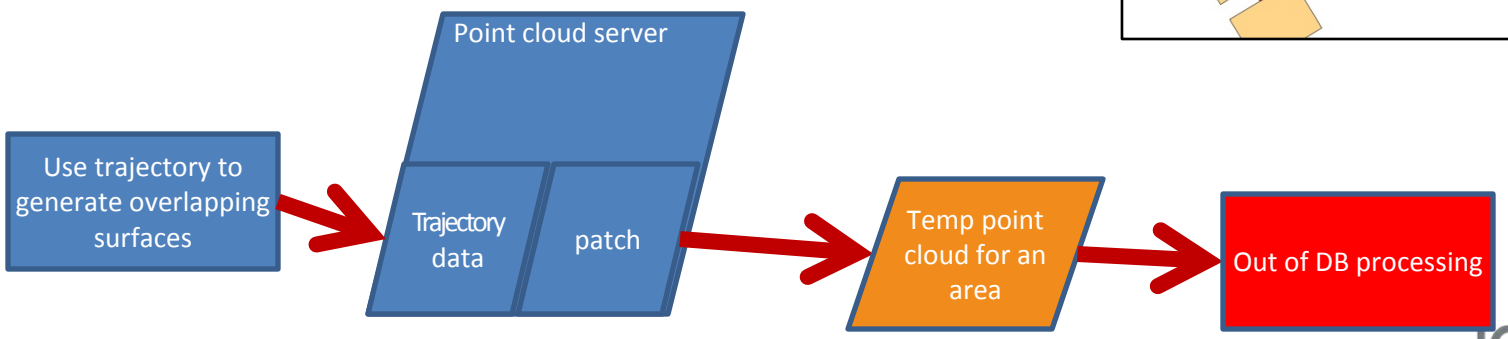
Trajectory + buildings

Overlapping surfaces 1



Overlapping surfaces 2

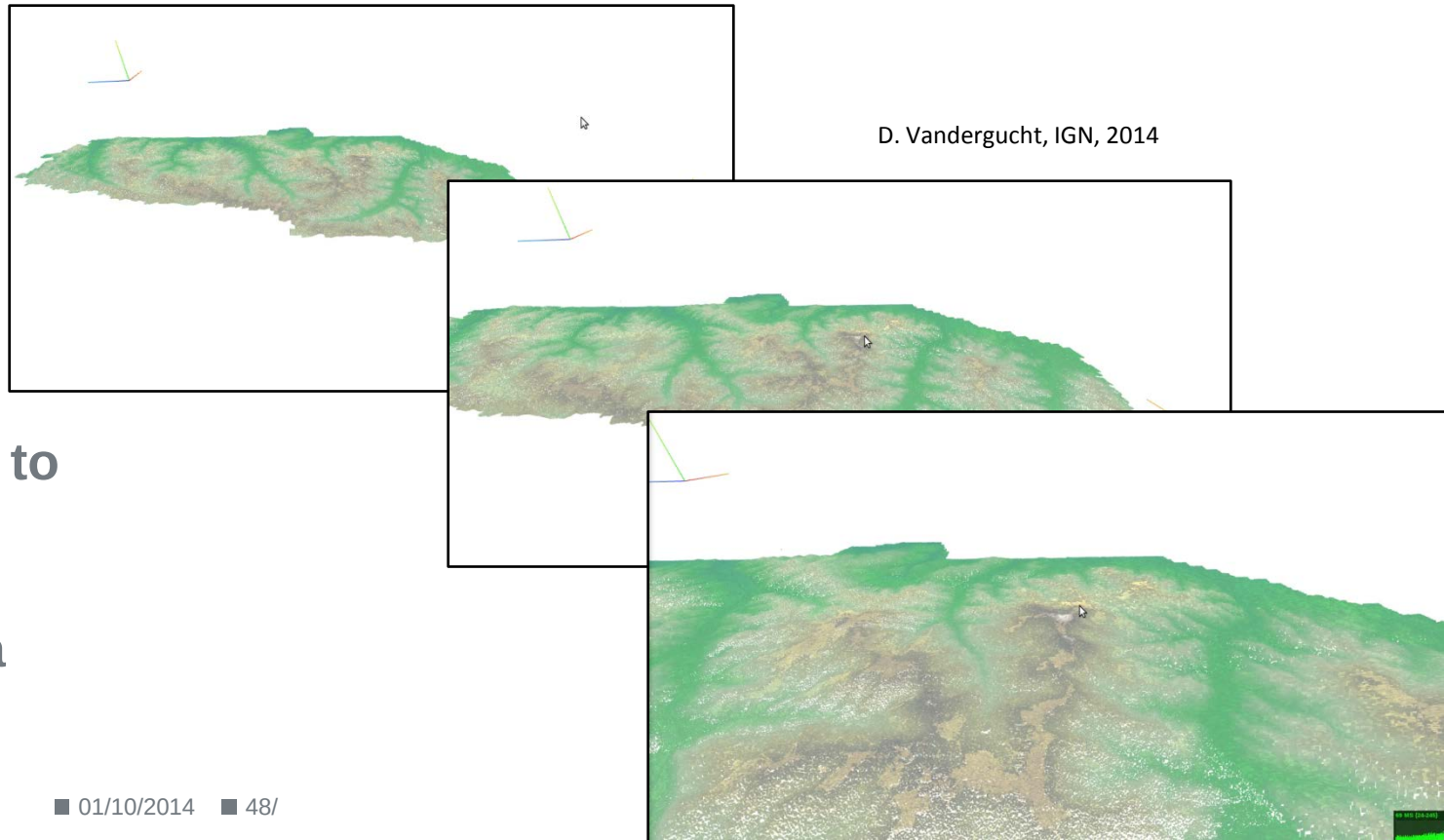




USING THE POINT CLOUD SERVER IN COMPLEX ARCHITECTURES

- **VISUALIZATION : LEVEL OF DETAIL**
- **no need for all the points all the time!**
- **Max 1 point per pixel : 2 Millions for Full HD (+ cache + anticipating)**
  If the patch is far away, need only few points

5.2 Billions points !

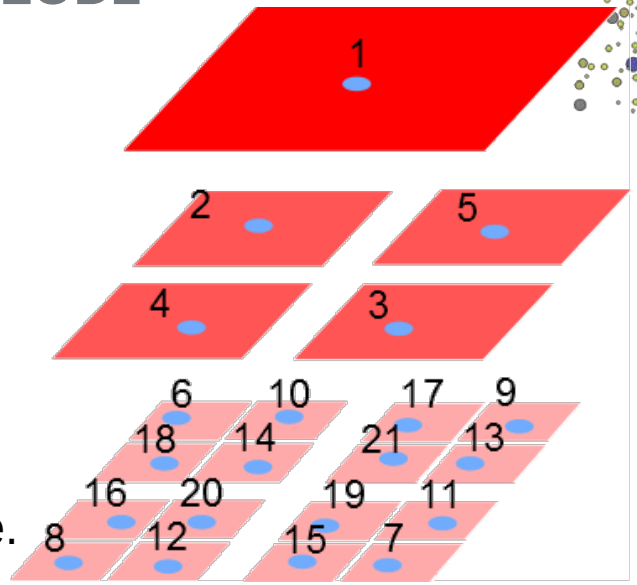


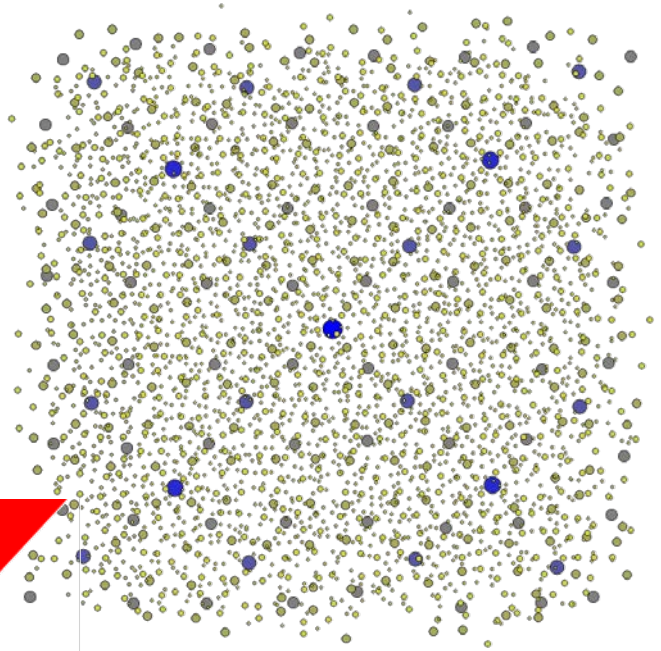
D. Vandergucht, IGN, 2014

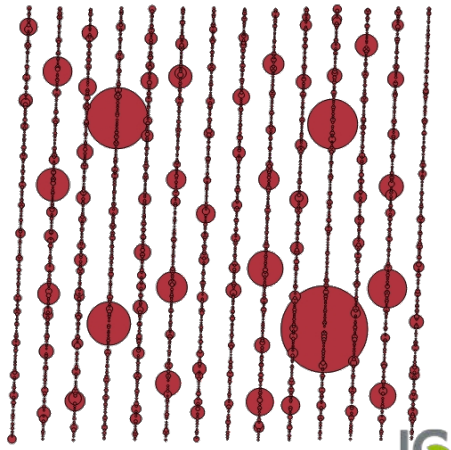
- **We need a way to generate pyramidal resolution for a patch!**

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.

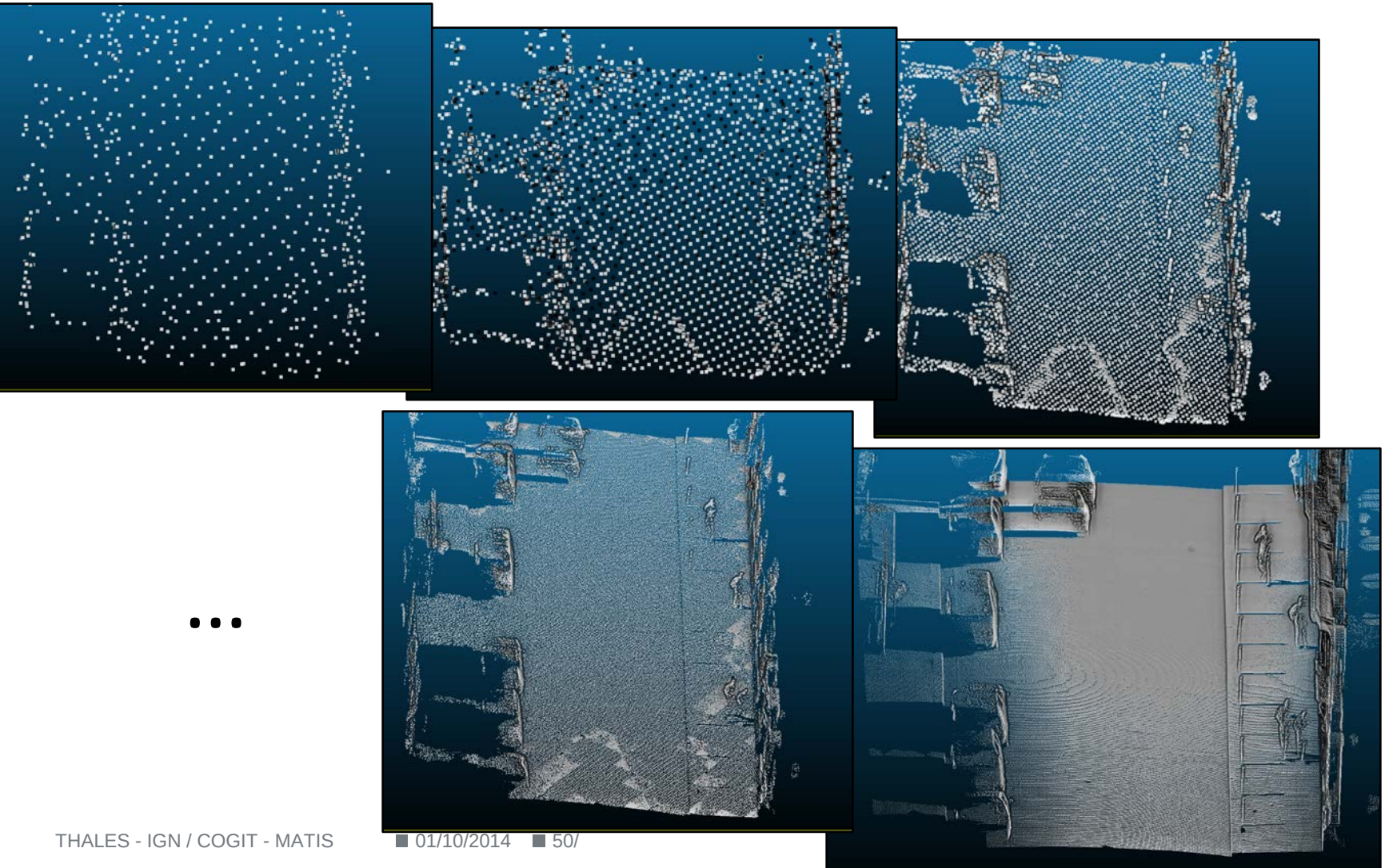






USING THE POINT CLOUD SERVER IN COMPLEX ARCHITECTURES

- **VISUALIZATION : LEVEL OF DETAIL**



...

USING THE POINT CLOUD SERVER IN COMPLEX ARCHITECTURES

- **ASYNCHRONOUS POINT CLOUD STREAMING TO BROWSER (PROTOTYPE)**

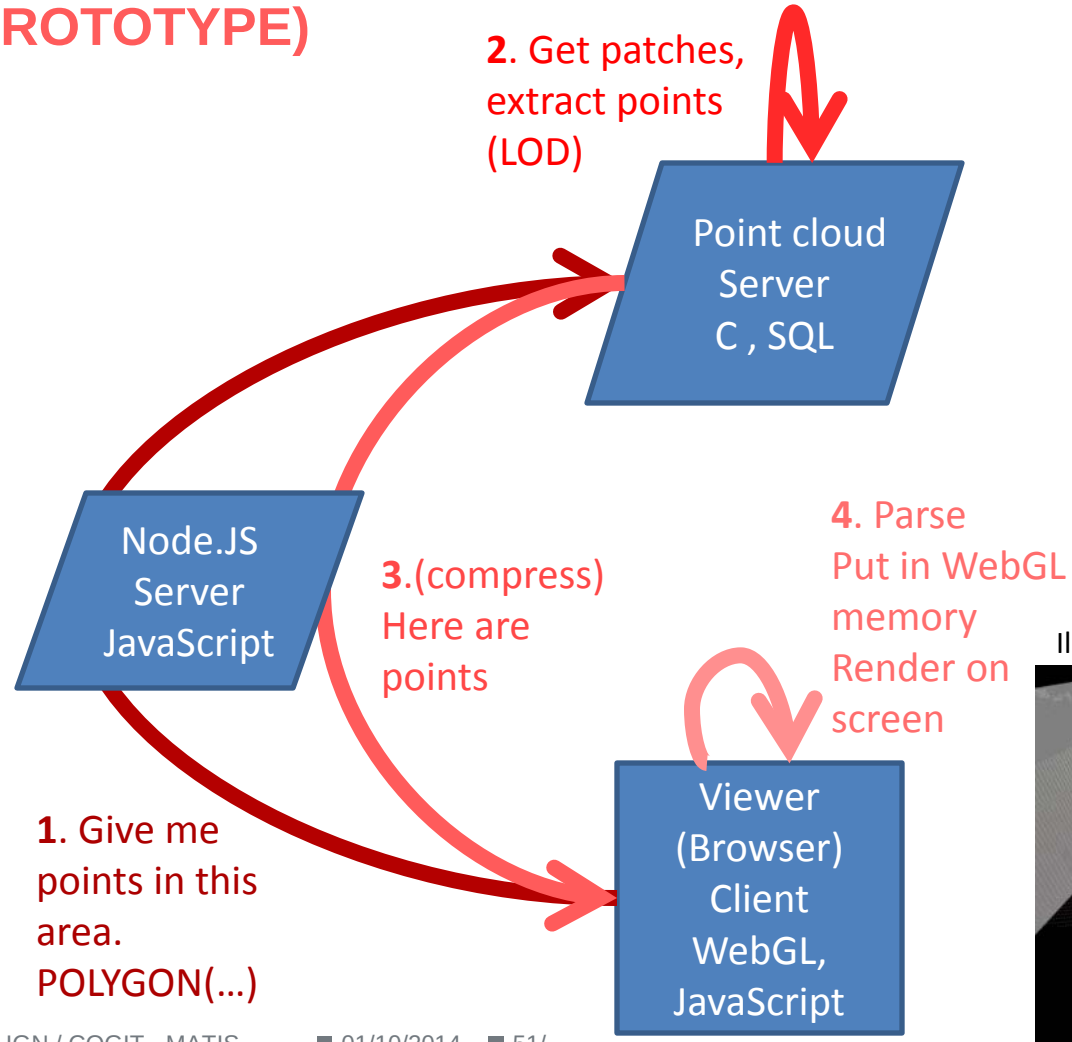
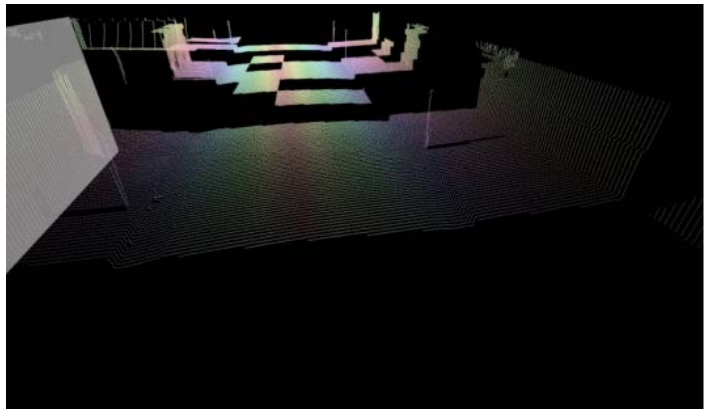


Illustration Q. Nguyen, IGN, 2014



USING THE POINT CLOUD SERVER IN COMPLEX ARCHITECTURES

- **ASYNCHRONOUS POINT CLOUD STREAMING TO BROWSER (PROTOTYPE)**

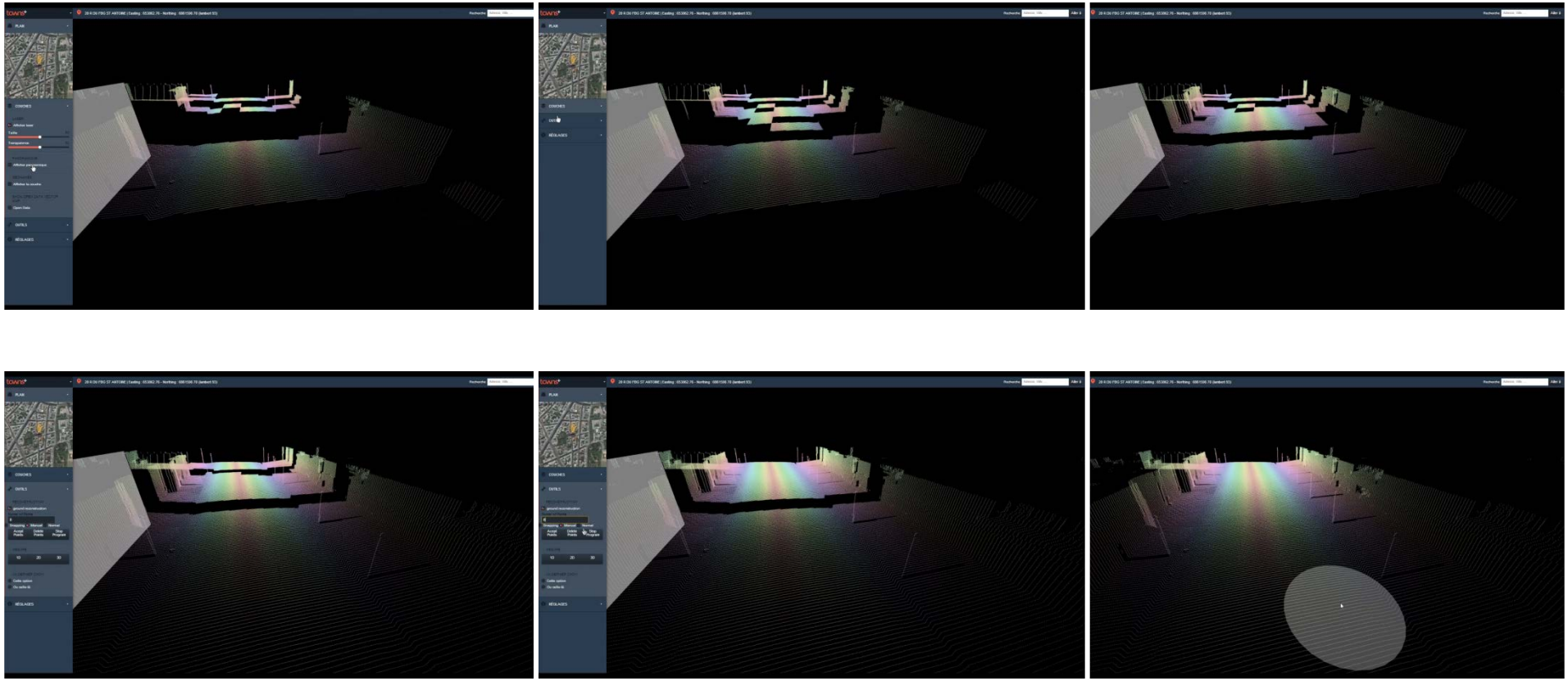


Illustration Q. Nguyen, IGN, 2014

USING THE POINT CLOUD SERVER IN COMPLEX ARCHITECTURES

- **INTERACTIVE 3D SIDEWALK RECONSTRUCTION (PROTOTYPE)**

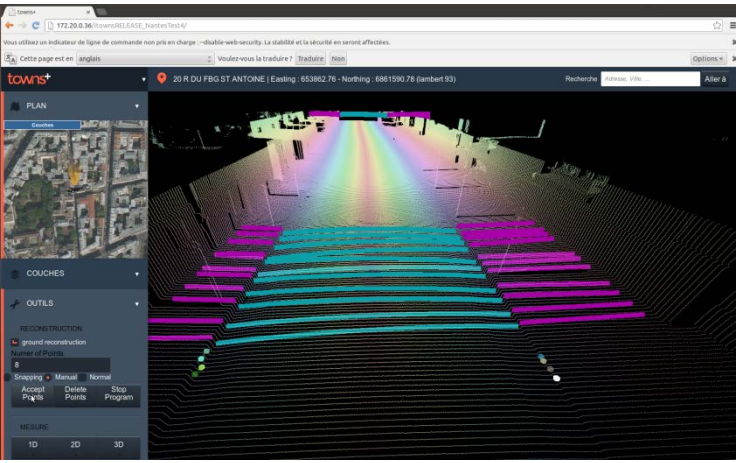
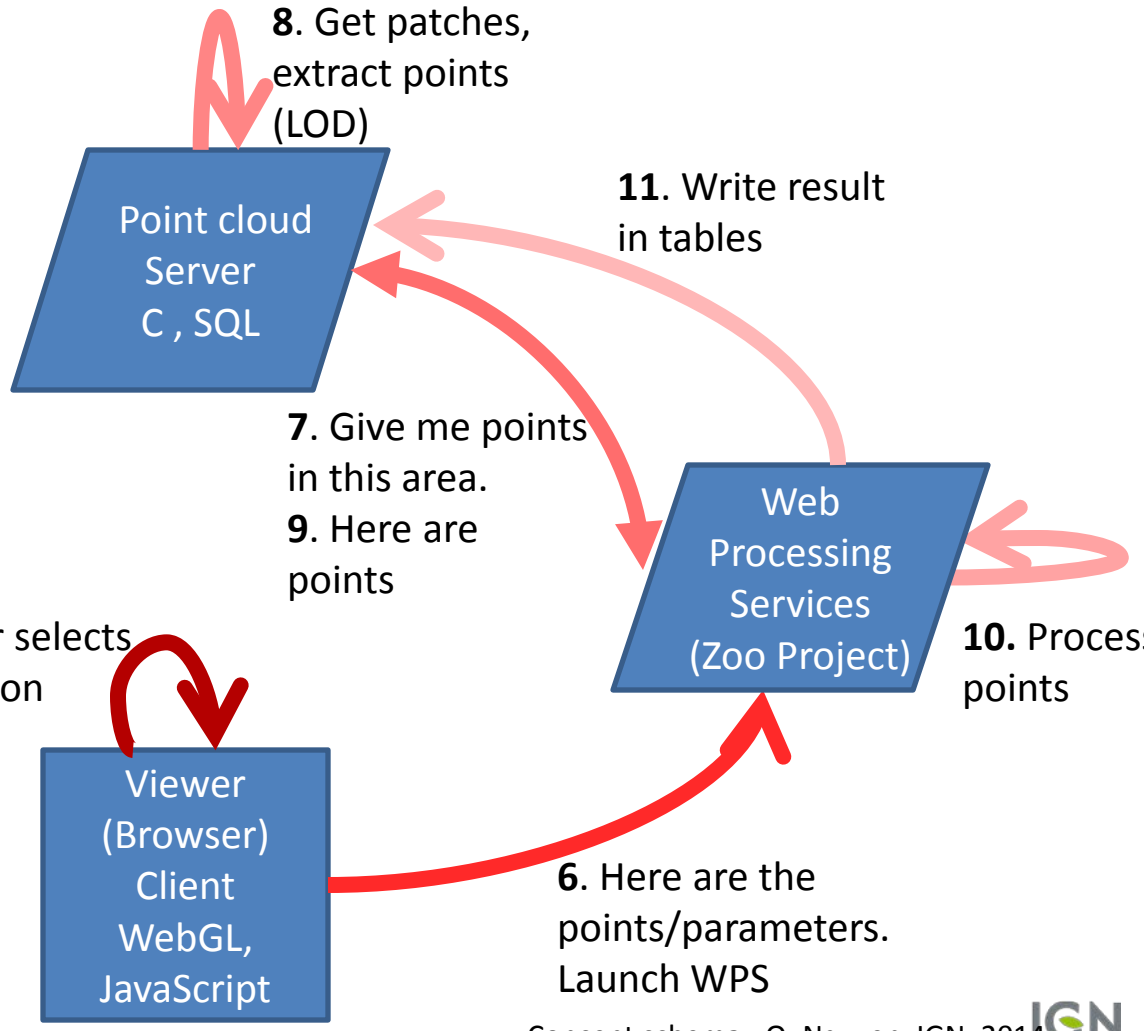


Illustration A.Hervieu, IGN, 2014



Concept schema : Q. Nguyen, IGN, 2014

USING THE POINT CLOUD SERVER IN COMPLEX ARCHITECTURES

INTERACTIVE 3D SIDEWALK RECONSTRUCTION (PROTOTYPE)

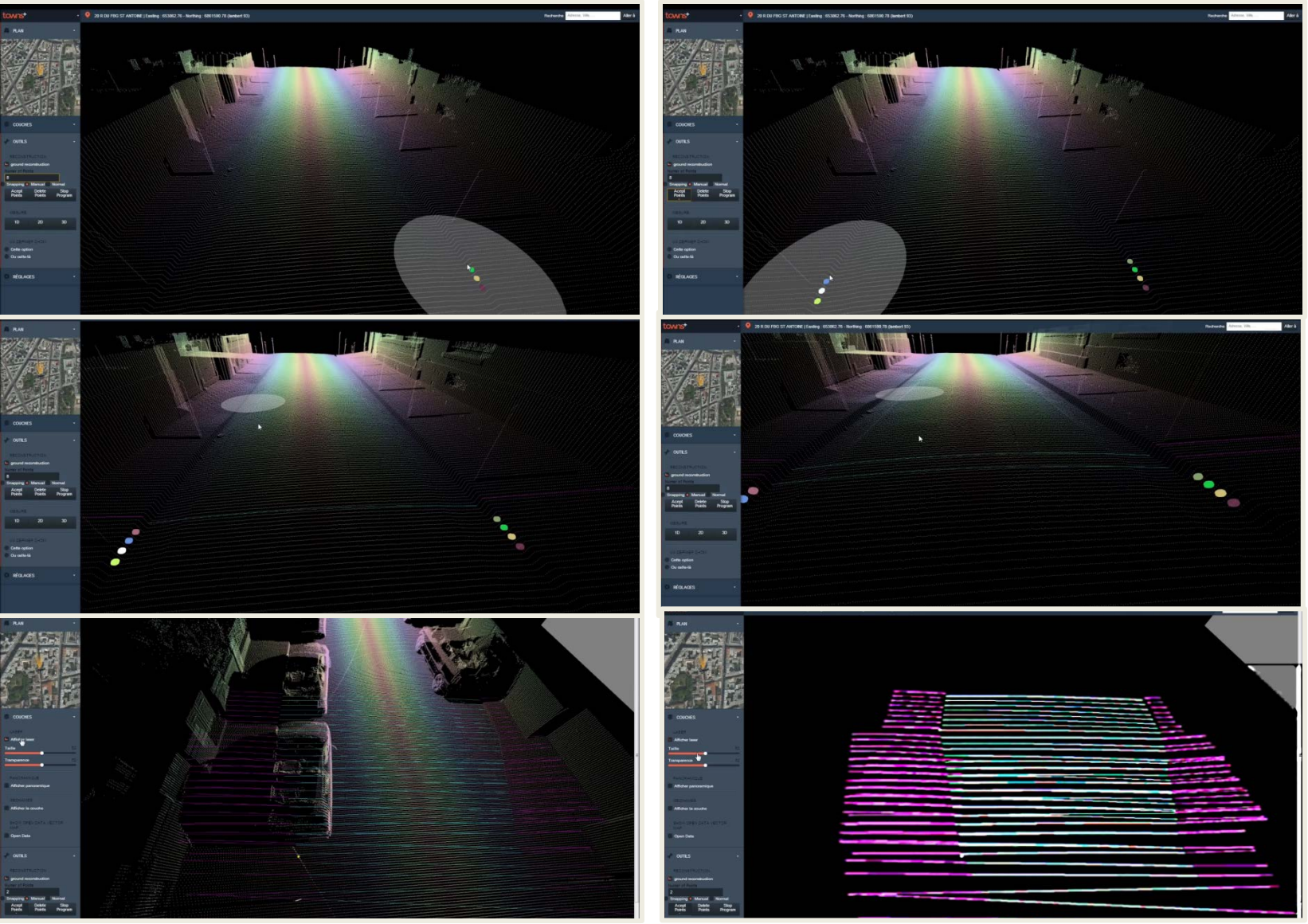


Illustration
A.Hervieu, IGN, 2014

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.

TOOLS

- **POSTGRES / POSTGIS**
- **POINTCLOUD**
- **CLOUDCOMPARE**
- **RPLY**

- **PDAL**
- **GDAL**
- **QGIS**

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