

# Point clouds in PostgreSQL: store and publish

# Éric Lemoine

Developer @ Oslandia  
FOSS4G developer and enthusiast since 2007

✉ eric.lemoine@oslandia.com  
⌚ @elemoine  
🐦 @erilem

# Oslandia



Oslandia provides service on open-source software

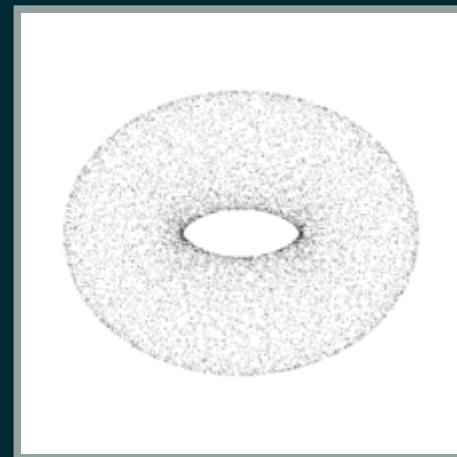
- GIS
- 3D
- DATA

# Point clouds!



# Point clouds

« A point cloud is a set of data points in space. »



source: [wikipedia](#)

# Point clouds

- Generally produced by 3D scanners (LiDAR)
- Can also be created using Photogrammetry

# LiDAR

Terrestrial, Airborne, Mobile, Unmanned



Phoenix Aerial AL3-16

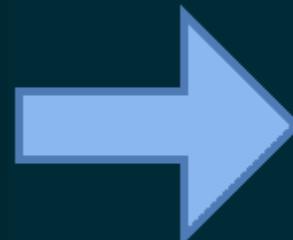
LIDAR Mapping System

# Many applications!

- Create Digital Elevation Models (DEMs)
- Create 3D models
- Detect objects and obstacles
- etc.



# Point clouds in PostgreSQL



# Pointcloud

"PostgreSQL extension for storing point cloud data"

<https://github.com/pgpointcloud/pointcloud>

The screenshot shows the GitHub repository page for `pgpointcloud/pointcloud`. The page includes the repository name, a star count of 156, a fork count of 69, and links for Unwatch, Star, Fork, and Settings. Below these are links for Code, Issues (36), Pull requests (1), Projects (0), Wiki, Insights, and Settings. A brief description states: "A PostgreSQL extension for storing point cloud (LIDAR) data." There are buttons for Add topics, Edit, 499 commits, 6 branches, 3 releases, and 16 contributors. At the bottom, there are buttons for Branch: master, New pull request, Create new file, Upload files, Find file, and Clone or download. A recent commit by elemoine is shown: "Merge pull request #203 from elemoine/news".

# Pointcloud

- Initially developed by Paul Ramsey (funded by Natural Resources Canada)
- Now developed and maintained by Oslandia and IGN

# Goals

- Storing LiDAR data in PostgreSQL
- Leveraging that data for analysis in PostGIS

# Why not use PostGIS?

Column	Type
id	integer
geom	geometry(PointZ)
intensity	double precision
returnnumber	double precision
numberofreturns	double precision
classification	double precision
scananglerank	double precision
red	double precision
green	double precision
blue	double precision

# Why not use PostGIS?

- One point per row means billions of rows
- Does not work!

# Patches of points

- Organize the points into patches
- → Millions of rows instead of billions

Column	Type
id	integer
pa	pcpatch(1)

# Two types

- `PcPoint(pcid)`
- `PcPatch(pcid)`

# Use Pointcloud

```
CREATE EXTENSION pointcloud;
CREATE EXTENSION postgis;          -- optional
CREATE EXTENSION pointcloud_postgis; -- optional
```

# Use Pointcloud

Schema	Name	Type
public	geography_columns	view
public	geometry_columns	view
public	pointcloud_columns	view
public	pointcloud_formats	table
public	raster_columns	view
public	raster_overviews	view
public	spatial_ref_sys	table

# Schema

pcid	srid	schema
1	4326	<?xml version="1.0" encoding="UTF-8"?> <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> <pc:dimension> <pc:position>2</pc:position> <pc:size>4</pc:size> <pc:description>Y 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>Y</pc:name> <pc:interpretation>int32_t</pc:interpretation> <pc:scale>0.01</pc:scale> </pc:dimension> <pc:dimension> <pc:position>3</pc:position> <pc:size>4</pc:size> <pc:description>Z coordinate as a long integer. You must use the scale and offset information of the header to determine the double value.</pc:description> </pc:dimension> </pc:PointCloudSchema>

```
SELECT pa FROM patches LIMIT 1;
```

```
-----  
0101000000020000000900000002100000000400000060CEFFFFBC9A78560000  
(1 row)
```

```
SELECT PC_AsText(pa) FROM patches LIMIT 1;
-----
{"pcid":1,"pts":[[[-126.99,45.01,1,0],[-126.98,45.02,2,0],[-126.97,45.03,3,0]]]}  
(1 row)
```

# Working with real data



# Load data using PDAL

```
{  
    "pipeline": [  
        {  
            "type": "readers.las",  
            "filename": "inrap.las"  
        },  
        {  
            "type": "filters.chipper",  
            "capacity": "400"  
        },  
        {  
            "type": "writers.pgpointcloud",  
            "connection": "dbname=lopocs host=localhost user=lopocs",  
            "schema": "public",  
            "table": "inrap",  
            "compression": "none",  
            "srid": "3946",  
            "overwrite": "true",  
            "column": "points",  
            "scale_x": "0.01",  
            "scale_y": "0.01",  
            "scale_z": "0.01",  
            "offset_x": "831587.0631",  
            "offset_y": "6287650.923",  
            "offset_z": "30.921565055000002"  
        }  
    ]  
}
```

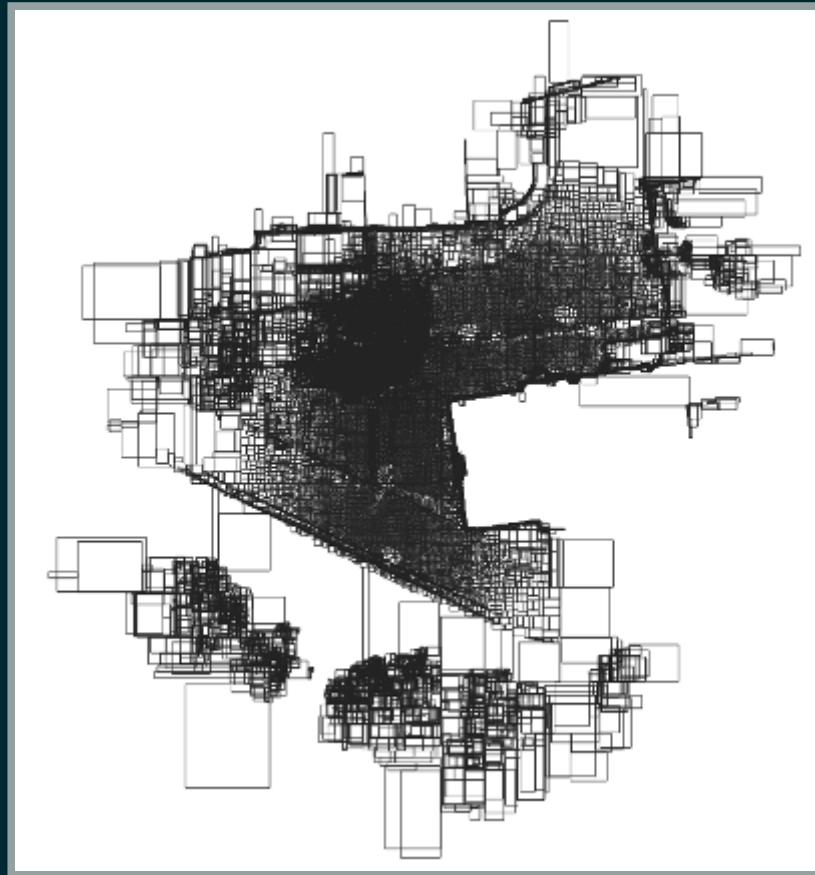
```
SELECT count(*) num_patches,  
       sum(PC_NumPoints(points)) num_points  
FROM inrap;
```

num_patches		num_points
45952		18380597

(1 row)

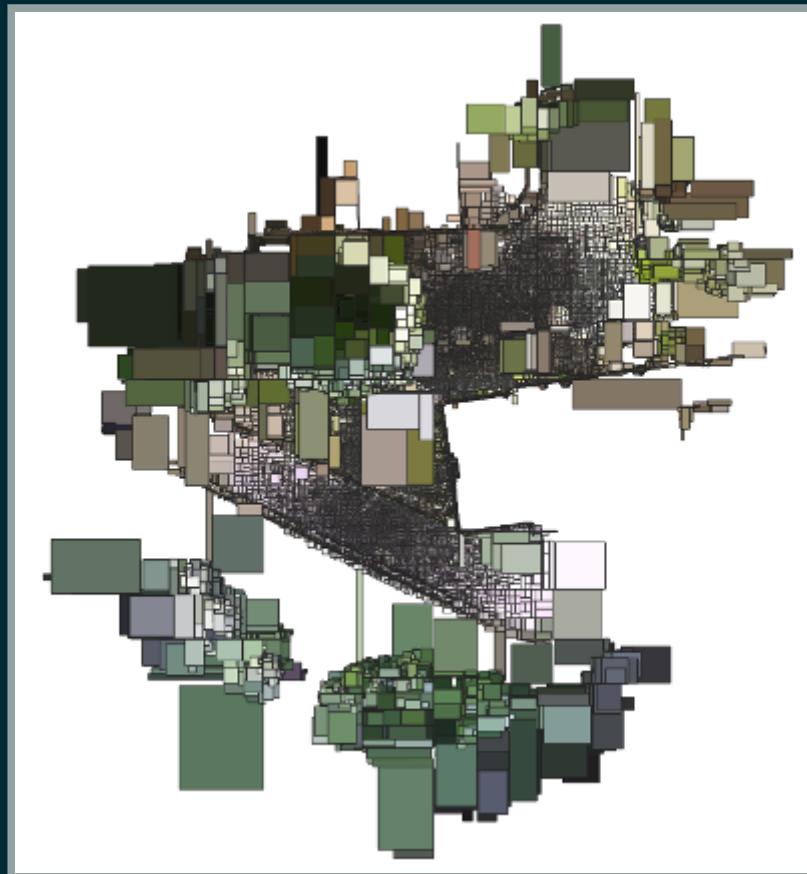
# Visualize in QGIS

```
SELECT id, points FROM inrap
```



# Visualize in QGIS

```
SELECT id, points, PC_PatchAvg(points, 'red') || ',' ||  
      PC_PatchAvg(points, 'green') || ',' ||  
      PC_PatchAvg(points, 'blue') || ',', 255' color FROM inrap;
```



# v1.1.0

Released the 2018-04-31

New functions include:

- PC\_Patch{Avg,Max,Min}(p pcpatch, dimname text)
- PC\_Range(p pcpatch, start int4, n int4)
- PC\_SetPCId(p pcpatch, pcid int4, def float8 default 0.0)
- PC\_Transform(p pcpatch, pcid int4, def float8 default 0.0)
- PC\_BoundingDiagonalAsBinary(p pcpatch)



# "Light OpenSource PointCloud Server"

<https://github.com/Oslandia/lopocs>

The screenshot shows the GitHub repository page for `Oslandia/lopocs`. The page includes the repository name, a star and fork count, and links to code, issues, pull requests, projects, wiki, insights, and settings. Below this is a description of the project as a "Light OpenSource PointCloud Server" with a link to its website. The repository has 233 commits, 5 branches, 0 releases, 6 contributors, and is licensed under LGPL-2.1. A pull request from `elemoine` is visible, along with file upload and download buttons. The commit history shows a merge from `OslandiaPy` and a recent update to `conf`.

# LOPoCS

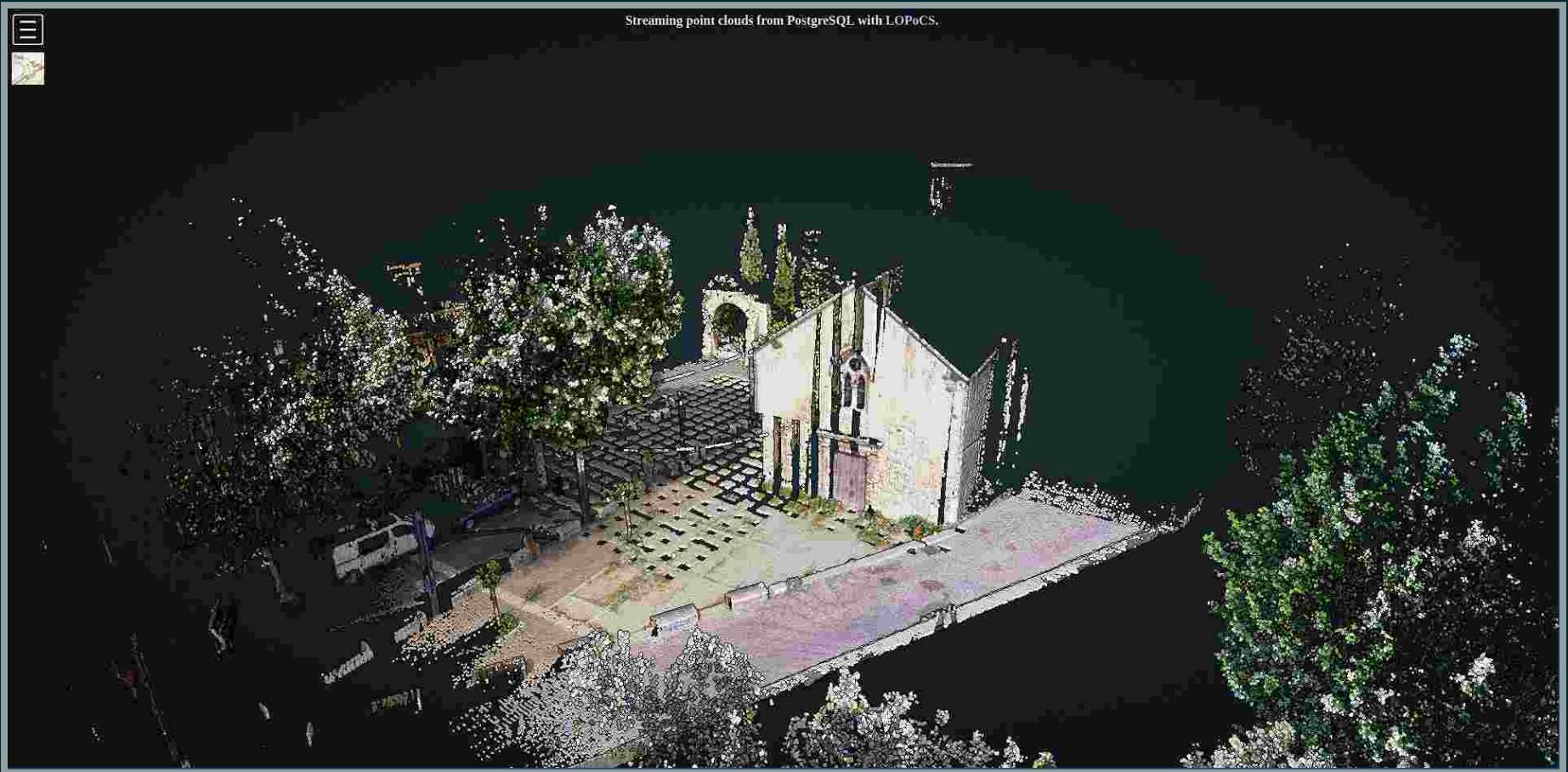
- Streams point cloud data stored in PostgreSQL
- Supports multiple streaming protocols  
(Greyhound and 3D Tiles currently supported)
- → works with Potree, Cesium, iTowns

# Greyhound in a nutshell

*Greyhound is a dynamic point cloud server architecture that performs progressive level-of-detail streaming of indexed resources on-demand*

<https://greyhound.io/>

# Potree/Greyhound



Streaming point clouds from PostgreSQL with LOPoCS.

# 3D Tiles in a nutshell

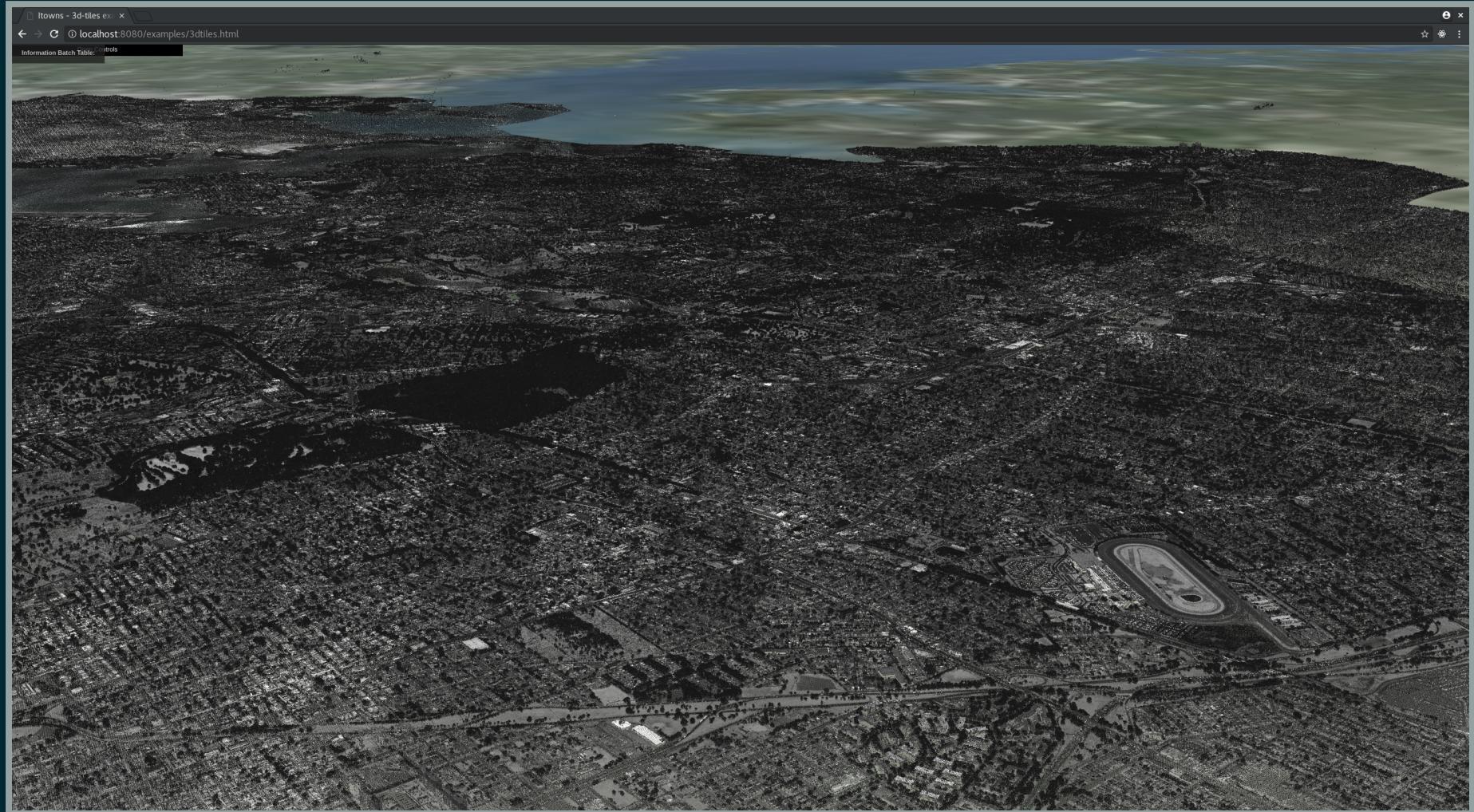
*Specification for streaming massive  
heterogeneous 3D geospatial datasets*

<https://github.com/AnalyticalGraphicsInc/3d-tiles>

# Cesium/3D Tiles



# iTowns/3D Tiles



# Motivation

- Stream point cloud data directly from Postgres
- No export/indexing step required
- Nice for pre-visualization and prototyping

# Technologies

- Language: Python 3
- Web framework: Flask
- Main libraries: `py3dtiles`, `lazperf`

# LOPoCS Future

- Improve the selection of points
- Make rendering as good and performant as possible



Thanks!