

POINT CLOUDS IN POSTGRESQL: STORE AND PUBLISH

Speaker notes

This talk discusses point clouds, the Pointcloud extension for storing point clouds in PostgreSQL, and the LOPoCS lightweight server for streaming point clouds on the web.

ÉRIC LEMOINE

Developer @ Oslandia
FOSS4G developer and enthusiast since 2007

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

Speaker notes

My name is Éric Lemoine. I work at Oslandia. And I've been working in the FOSS4G field since 2007.

OSLANDIA



Oslandia provides service on open-source software

- GIS
- 3D
- DATA

Speaker notes

Oslandia is an open-source company working on GIS, 3D and Data Science. QGIS, PostGIS and the iTowns 3D WebGL framework are examples of software components we are working on.

POINT CLOUDS!

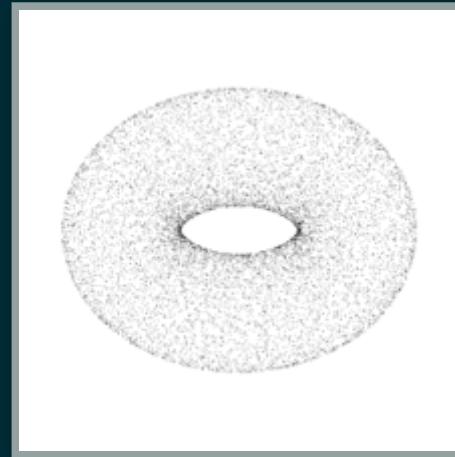


Speaker notes

Let's talk about point clouds in general first!

POINT CLOUDS

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



... ...

Speaker notes

A point cloud is just a set of data points in space. Nothing more. Point clouds provide a way to represent objects of our environment. A church and streets around it in the previous slide, and a donut here.

POINT CLOUDS

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

Speaker notes

What can produce point clouds? Point clouds are generally produced by 3D scanner. This is the LiDAR (Light Detection And Ranging) technology. Point clouds can also be produced using photogrammetry techniques (through homolog points).

LIDAR

Terrestrial, Airborne, Mobile, Unmanned



Speaker notes

There are several types of LiDAR acquisitions: Terrestrial (fixed tripods), Airbone (planes or helicopters), Mobile (Google Car like), and Unmanned (drones).

MANY APPLICATIONS!

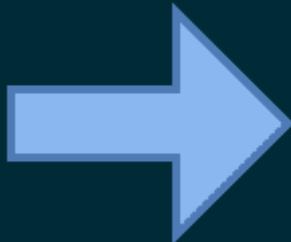
- Create Digital Elevation Models (DEMs)
- Create 3D models
- Detect objects and obstacles
- etc.



Speaker notes

Point clouds have a wide range of applications. Examples include creating Digital Elevation Models, Digital Surface Models, 3D models, and detecting objects and obstacles. Autonomous cars use LiDAR! For the creation of 3D models, 3D surfaces are derived from point clouds.

POINT CLOUDS IN POSTGRESQL



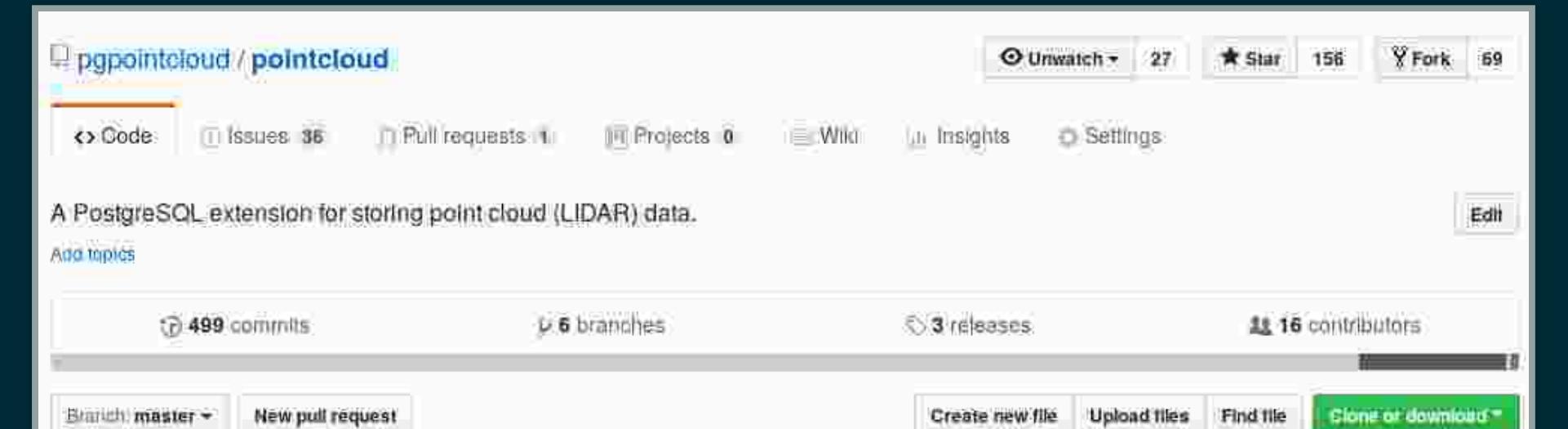
Speaker notes

Now let's discuss the Pointcloud extension for PostgreSQL.

POINTCLOUD

"PostgreSQL extension for storing point cloud data"

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



Speaker notes

The Pointcloud extension allows storing point cloud data in PostgreSQL databases. Pointcloud is open-source and available on GitHub. It's easy to build and install, and it's well documented.

POINTCLOUD

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

Speaker notes

The initial development of Pointcloud was funded by Natural Resources Canada, and done by Paul Ramsey, one of the main PostGIS developers. It is currently developed and maintained by Oslandia and IGN (mostly).

GOALS

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

Speaker notes

Storing LiDAR data in PostgreSQL enables all sort of analysis, by using PostGIS and Pointcloud together. For example determining all the points that are within a polygon is both a very easy and very fast operation.

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

Speaker notes

By the way, why not using PostGIS instead of creating a specific extension? PostGIS has a PointZ geometry type that could be used, hasn't it?

WHY NOT USE POSTGIS?

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

Speaker notes

Because point clouds may have billions of points, which would mean billions of database rows, which wouldn't work.

PATCHES OF POINTS

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

Column	Type
id	integer
pa	pcpatch(1)

Speaker notes

For that reason Pointcloud organizes points into patches. A patch typically includes several hundreds or several thousands points, which translates into millions of rows rather than billions of rows. This is still big, but manageable.

TWO TYPES

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

Speaker notes

Pointcloud actually defines two new types: `PcPoint` and `PcPatch`. `PcPatches` are collections of `PcPoints`. `PcPoints` are packings of point dimensions (X, Y, Z, ...). Dimensions are packed in byte arrays.

USE POINTCLOUD

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

Speaker notes

Enabling Pointcloud in a database is done the way as enabling PostGIS.

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

Speaker notes

After enabling Pointcloud in a database the pointcloud_columns view and the pointcloud_formats table are added to the database. The pointcloud_columns view includes information about all the PcPoint and PcPatch columns that exist in the database. The pointcloud_formats table includes XML documents that define how dimensions are encoded in PcPoints.

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>

Speaker notes

This is an example of an PointCloudSchema XML document.

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

```
-----  
010100000002000000090000000210000000400000060CEFFFBC9A78560000  
(1 row)
```

Speaker notes

An SQL query that selects a patch (PcPatch) returns a sort of WKB (Well Known Binary) string representing the patch.

```
SELECT PC_AsText(pa) FROM patches LIMIT 1;
-----
{"pcid":1,"pts":[[[-126.99,45.01,1,0],[-126.98,45.02,2,0],[-126.9
(1 row)
```

Speaker notes

The Pointcloud extension provides functions for manipulating points and patches. For example the PC_AsText function returns a JSON representation of patches.

WORKING WITH REAL DATA



Speaker notes

Let's look at how real point cloud data can be inserted into a PostgreSQL database?

PDAL

<https://www.pdal.io/>

Speaker notes

PDAL can be used for that!

LOAD DATA USING PDAL

```
{  
  "pipeline": [  
    {  
      "type": "readers.las",  
      "filename": "inrap.las"  
    },  
    {  
      "type": "filters.chipper",  
      "capacity": "400"  
    },  
    {  
      "type": "writers.pgpointcloud",  
      "connection": "dbname=lopcodes host=localhost user=lopcodes",  
      "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"  
    }  
  ]  
}
```

Speaker notes

This creates a PDAL pipeline whose source is a LAS file and sink is a Pointcloud database table. The filter in between the source and the sink is a so-called "chipper" filter. The "chipper" filter is responsible for creating patches of points – 400-point patches here.

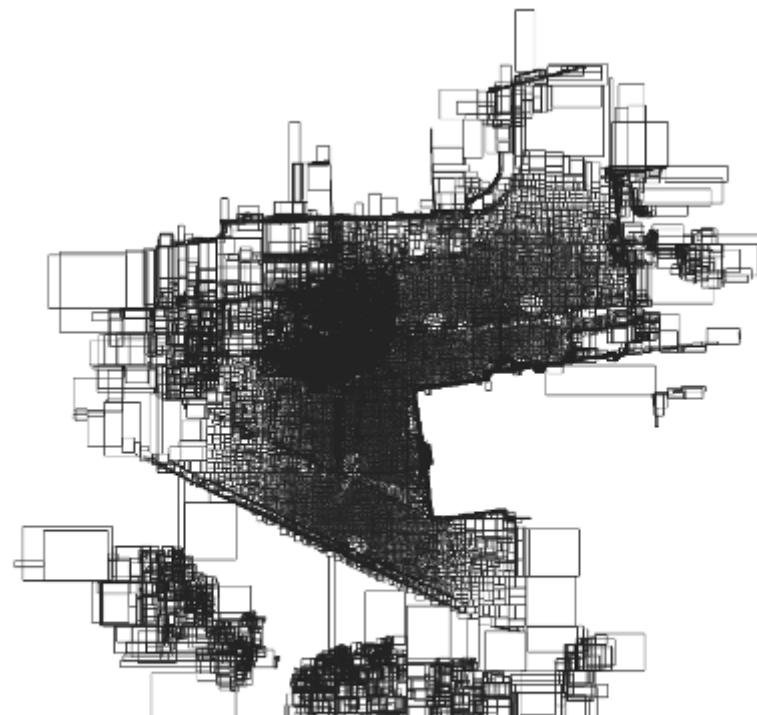
```
SELECT count(*) num_patches,  
       sum(PC_NumPoints(points)) num_points  
FROM inrap;  
  
num_patches | num_points  
-----+-----  
 45952 | 18380597  
(1 row)
```

Speaker notes

The point cloud has been loaded into PostgreSQL. We can start throwing some SQL at it! The above SQL query just counts the total number of patches and points.

VISUALIZE IN QGIS

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

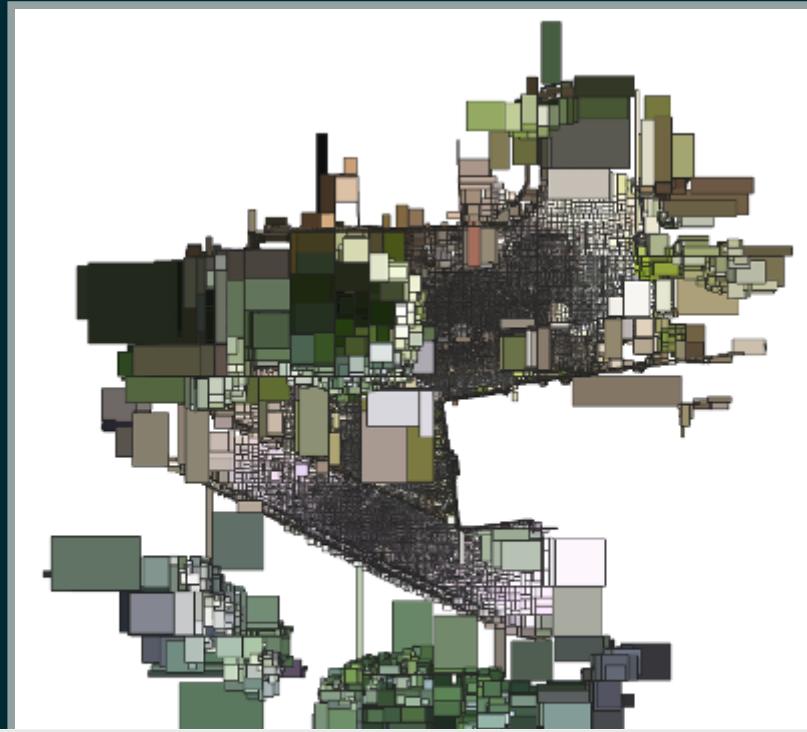


Speaker notes

Point clouds in PostgreSQL can be displayed in QGIS! As a 2D viewer QGIS actually displays the 2D bounds (X/Y bounds) of patches. This is actually very useful for testing and debugging.

VISUALIZE IN QGIS

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



Speaker notes

We can even add some colors!

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)

Speaker notes

The Pointcloud extension includes many PcPoint and PcPatch manipulation functions. The 1.1.0 version, which was released the 2018-04-31, brings new functions that are useful both for analysis and visualization.



"Light OpenSource PointCloud Server"

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

A screenshot of a GitHub repository page. The header shows the repository name "Oslandia/lopcos" and various navigation links like "Code", "Issues 4", "Pull requests", "Projects", "Wiki", "Insights", and "Settings". The main content area displays the project description "Light OpenSource PointCloud Server" and a link to its website "https://oslandia.github.io/lopcos/". Below the description are several topic tags: "pointcloud", "3d", "gl", "lidar", "opensource", "lawnmower", "cesiumjs", "python", and "postgis". Key statistics are shown: 233 commits, 5 branches, 0 releases, 6 contributors, and the license is LGPL-2.1. At the bottom, there are buttons for "Create new file", "Upload files", "Find file", and a prominent green "Clone or download" button.

Speaker notes

Interestingly people often want to visualize their data! Oslandia created LOPoCS for that. LOPoCS is not a viewer, it's a light open-source pointcloud streaming server.

LOPoCS

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

Speaker notes

LOPoCS is able to stream point cloud data stored in PostgreSQL/Pointcloud. LOPoCS implements existing protocols. The Greyhound and 3D Tiles protocols are currently supported. This makes LOPoCS work with various point cloud web viewers, including Potree, Cesium and iTowns.

GREYHOUND IN A NUTSHELL

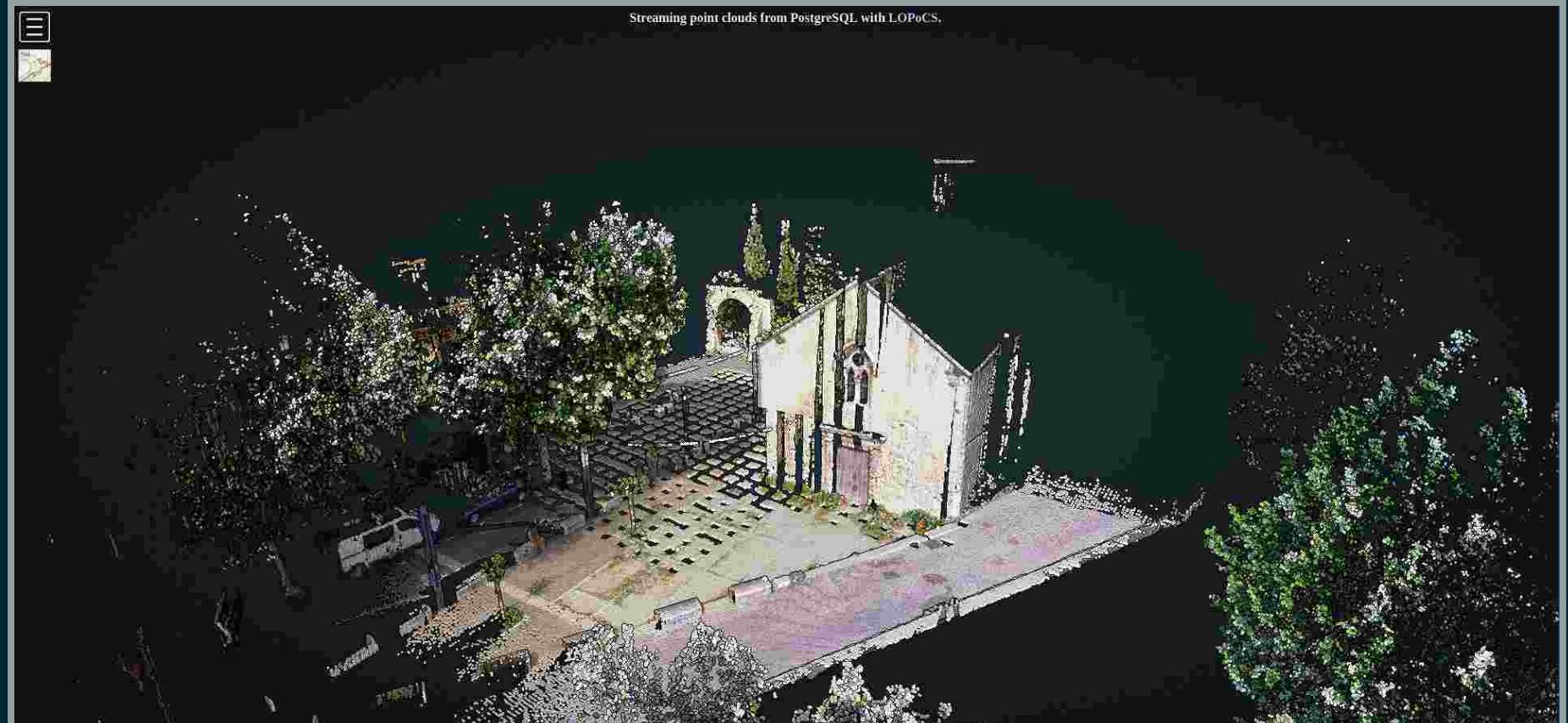
Greyhound is a dynamic point cloud server architecture that performs progressive level-of-detail streaming of indexed resources on-demand

<https://greyhound.io/>

Speaker notes

Greyhound is a point cloud data streaming protocol. It was created by Howard Butler and Connor Maning from Hubo, Inc. It is the protocol used by the Greyhound server.

POTREE/GREYHOUND



Speaker notes

This is a Potree application displaying a point cloud streamed by LOPoCS.

3D TILES IN A NUTSHELL

Specification for streaming massive heterogeneous 3D geospatial datasets

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

Speaker notes

3D Tiles a specification for streaming 3D content. Is is not specific to point cloud data. It can be used to stream building, trees, point clouds and vector data. It was created by the Cesium team, and has now entered the OGC Community Standard process.

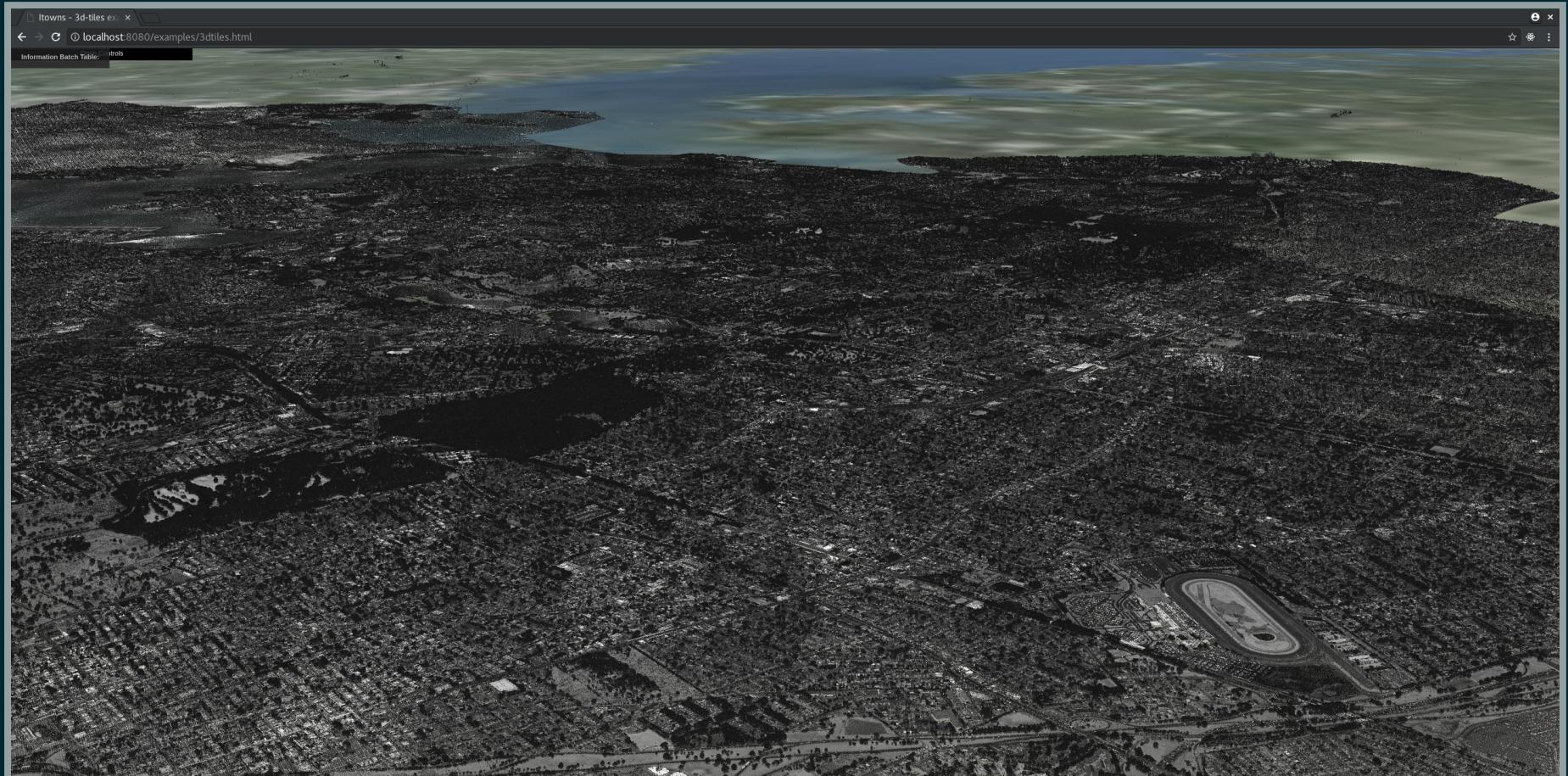
CESIUM/3D TILES



Speaker notes

This is a Cesium application displaying a point cloud streamed by LOPoCS.

iTOWNS/3D TILES



Speaker notes

This is an iTowns application displaying a point cloud streamed by LOPoCS.

MOTIVATION

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

Speaker notes

The motivation is to be able to stream point cloud data directly from PostgreSQL, without having to export and index the data outside the database. In particular this is useful for pre-visualization and prototyping. For serious visualizations exporting the data from indexed files (3D Tiles for example) will always lead to better performance.

TECHNOLOGIES

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

Speaker notes

LOPoCS is written in Python 3. It uses the Flask web framework. It is based on the `py3dtiles` library (for 3D Tiles) and the `laz-perf` library (for Greyhound).

LOPOCS FUTURE

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

Speaker notes

LOPoCS is still a young project. It still required work to improve the selection of points, and provide for a better rendering. We hope to make LOPoCS better and attract more users and developers in the future.



THANKS!