DISTRIBUTED DIMENSONALITY-BASED RENDERING OF LIDAR POINT CLOUDS

Mathieu Brédif, Bruno Vallet, Benjamin Ferrand

Université Paris-Est, IGN/SR, MATIS,
73 avenue de Paris, 94160 Saint Mandé, France
firstname.lastname@ign.fr

Commission III, WG III/5

ABSTRACT:

Mobile Mapping Systems (MMS) are now commonly acquiring lidar scans of urban environments for an increasing number of applications such as 3D reconstruction and mapping, urban planning, urban furniture monitoring, practicability assessment for persons with reduced mobility (PRM)... MMS acquisitions are usually huge enough to incur a usability bottleneck for the increasing number of non-expert user that are not trained to process and visualize these huge datasets through specific softwares. A vast majority of their current need is for a simple 2D visualization that is both legible on screen and printable on a static 2D medium, while still conveying the understanding of the 3D scene and minimizing the disturbance of the lidar acquisition geometry (such as lidar shadows). The users that motivated this research are, by law, bound to precisely georeference underground networks for which they currently have schematics with no or poor absolute georeferencing. A solution that may fit their needs is thus a 2D visualization of the MMS dataset that they could easily interpret and on which they could accurately match features with their user datasets they would like to georeference. Our main contribution is two-fold. First, we propose a 3D point cloud stylization for 2D static visualization that leverages a Principal Component Analysis (PCA)-like local geometry analysis. By skipping the usual and error-prone estimation of a ground elevation, this rendering is thus robust to non-flat areas and has no hard-to-tune parameters such as height thresholds. Second, we implemented the corresponding rendering pipeline so that it can scale up to arbitrary large datasets by leveraging the Spark framework and its Resilient Distributed Dataset (RDD) and Dataframe abstractions.

1 INTRODUCTION

1.1 Context

In the last decade new sensor technologies and new applications have caused a drastic upsurge in remotely-sensed data collection such as image and lidar, which may be acquired from the pedestrian, vehicle, UAV, aerial and satellite perspectives. In urban environments, one of the most well-known and multi-faceted application is the reconstruction of a textured 3D city model that may be used for mapping, for flood or telecommunication simulation, as a proxy for virtual surveying or for communication and tourism. Among other applications, underground network managers are required by law in many countries, such as by the DT-DICT regulation in France, to provide a detailed mapping of their network (energy, telecommunications, water...) within a prescribed accuracy, in order to prevent network degradations (electricity black-out, gas leak...) caused by constructions. This often requires the registration of legacy local maps from relative or imprecise coordinates to a given spatial reference system. To assist this registration and provide such a reference map of the public space, we propose to derive a centimetric terrestrial orthographic view of the ground of the public space. This orthophotography-like dataset is proposed to be derived from a mobile mapping system for its precision and its surveying productivity which is similar to regular traffic speeds in urban environments, without the masking caused by trees and overhang structures experienced from higher sensor acquisitions (UAV, aerial...). Lidar was also preferred to images as it provides a direct certified reading of depth measurements, while meeting the sampling density requirements of the targeted application. Apart from this application, this end-product which we denote the (street-level) **ortholidar** has proved to be a valuable intermediate result that may be visualized very early in the production pipeline while being easy to interpret. As such its production is of prime interest for data quality inspection (manual validation of the dataset georeferencing) and standard image-based interaction, such as selection of ground control points or tie points for georeferencing. Thus, depending on the usage, the ortholidar may either be produced as (i) an image pyramid during a preprocess for optimal throughput or (ii) on-demand for low latency visualization of a given restricted extent only. While the high throughput qualities of (i) are easy to motivate in terms of a production pipeline, the low latency of (ii) would enable a user to peek interactively at intermediate results within the production pipeline without incurring the time costs of exporting a whole image pyramid.

1.2 Geospatial bigdata

Eldawy and Mokbel (2014) note the explosion of geospatial datasets and the need for a distributed computing, analysis and visualization. The acquisition of lidar datasets in particular is increasing drastically both in terms of the number of surveys and in terms of the number of points per survey. It is thus natural to consider distributed computing frameworks in order to be able to cope with the volume of data and be able to guarantee a given velocity, either in terms of latency (for navigation and exploration purposes) or in terms of throughput (for postprocessing data quicker than they are acquired). van Oosterom et al. (2015) recently benchmarked a number of possible management systems for point clouds in standard relational databases, distributed NoSQL databases and a file-based alternative. They emphasize the respective strength and limitations of these systems.

A comparatively new framework for distributed computing is Spark[1], which extends the standard map-reduce Hadoop paradigm for in-memory processing. Spark introduced recently a Dataframe API into its Spark SQL library, which enables a columnar SQL handling of Spark's resilient distributed dataset (RDD) and performs Just-in-time SQL execution plan optimizations such as predicate pushdown (Armbrust et al., 2015). Point cloud visualization, also known as point-based rendering, is an entire field of

[1] Apache Spark project, http://Spark.apache.org

This contribution has been peer-reviewed.
Editors: M. Brédif, G. Patanè, and T. Dokken
doi:10.5194/isprsarchives-XL-3-W3-559-2015

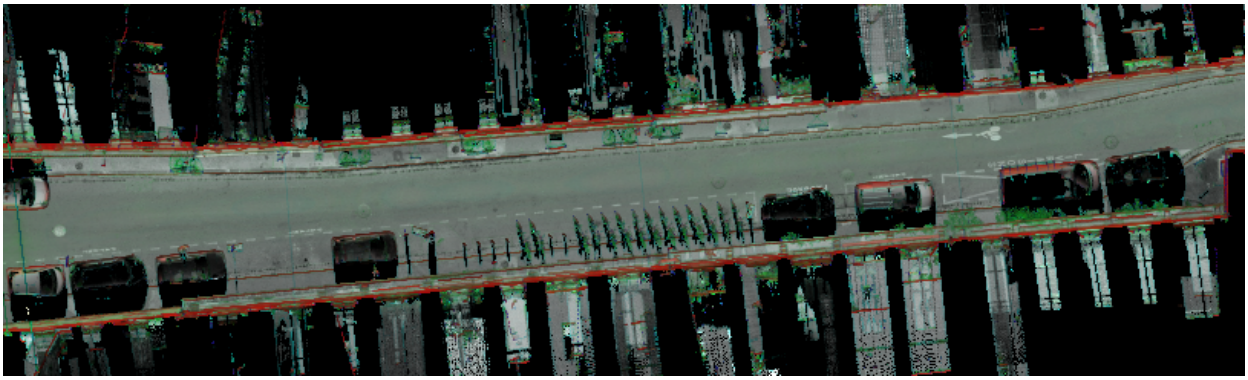

Figure 1: Proposed orthographic visualization of MMS lidar point cloud (about 12 million points).

research in itself. For instance, Richter et al. (2015) recently proposed to the geospatial community a lidar point cloud visualization approach that scales thanks to an out-of-core approach.

1.3 Contributions

This paper claims two contributions:

1. First, we propose a rendering of point clouds which leverages both local geometry and reflectance attributes. This rendering has been guided by user requirements.
2. Second, this rendering pipeline of point clouds is a first example of a new library that builds on the Spark framework to distribute the processing of a lidar survey on a cluster of machines.

The remaining of the paper is organized as follows. Section 2 analyses the visualization requirements and propose a styling for ortholidar visualization of (x, y, z, i) point clouds (claim 1). Section 3 develops the proposed approach and implementation of the distributed computing of the ortholidars (claim 2). Section 4 presents some results while section 5 draws conclusions and discusses future work.

2 ORTHOLIDAR STYLING

2.1 Ortholidar styling requirements

After discussing with end users and operators, we have collected the following set of requirements for the orthographic visualization of (x, y, z, r) lidar point clouds where (x, y, z) is the position and r stands generically for a radiometric attribute such as intensity, reflectance or the non depth-corrected amplitude attribute, radiometric correction being out of the scope of this paper:

- The rendering should degenerate to a simple gray-image of the radiometric attribute r when the scene is flat and horizontal, so as to imitate the optical panchromatic orthophotography users and operators are used to interpret.
- Vertical structures such as façades, walls, road signs, and tree trunks should be highly distinguishable and have a precise localization, as they are used to provide some 3D context of the scene and some understanding of its vertical dimension which was lost during the orthographic projection. Furthermore, these features are highly recognizable and may be used by operators to input points on these features for vectorization or registration purposes.
- Trees should be rendered in green. They are important cues for contextualization and scene understanding, but their depiction should not interfere with the understanding of the scene which lies below them. Thus they should not influence the texture.
- The ortholidar image should be highly compressible for easy management and handling and its no-data background would rather be white as hard-copy printing is still the dominant use case today.

In order to meet these requirements, we propose the following general guidelines:

- **Hue**: local geometry, shape, conveying the lost z dimension.
- **Saturation**: confidence in the shape determination for points that are not locally mostly horizontal surfaces.
- **Luminance**: radiometric texture of horizontal surfaces.

2.2 Dimensionality attributes for local shape visualization

The dimensionality analysis of a point cloud over a fixed or adaptive (Demantke et al., 2011) neighborhood yields for each point new attributes. Depending on the exact method the point weighting and neighborhood shape and size vary but, in the end, a Principal Component Analysis (PCA) is usually performed to produce for each lidar point a symmetric 3x3 covariance matrix which encodes the scattering of points in its vicinity. From the singular value decomposition (SVD) of that covariance matrix, we can define a local frame with its eigenvectors (e_1, e_2, e_3) which come with positive eigenvalues $\lambda_1 > \lambda_2 > \lambda_3$ expressing the local point scattering in their respective directions. Using standard deviations $\sigma_i = \sqrt{\lambda_i}$, Demantke et al. (2011) define dimensionality features (d_1, d_2, d_3) as:

$$d_1 = \frac{\sigma_1 - \sigma_2}{\sigma_1} \qquad d_2 = \frac{\sigma_2 - \sigma_3}{\sigma_1} \qquad d_3 = \frac{\sigma_3}{\sigma_1} \tag{1}$$

(d_1, d_2, d_3) have the nice properties of defining a partition of unity $\left(\sum_i d_i = 1, d_i \geq 0\right)$ and expressing the local shape of a point to be locally linear ($d_1 \simeq 1$), surfacic ($d_2 \simeq 1$) or volumetrically scattered ($d_3 \simeq 1$). We further refine the linear d_1 and surfacic d_2 classes by distinguishing horizontal (d_{1h}, d_{2h}) and vertical (d_{1v}, d_{2v}) features.

$$d_{1v} = d_1.|e_{1z}| \qquad d_{1h} = d_1 - d_{1v} \tag{2}$$

$$d_{2v} = d_2.|e_{3z}| \qquad d_{2h} = d_2 - d_{2v} \tag{3}$$

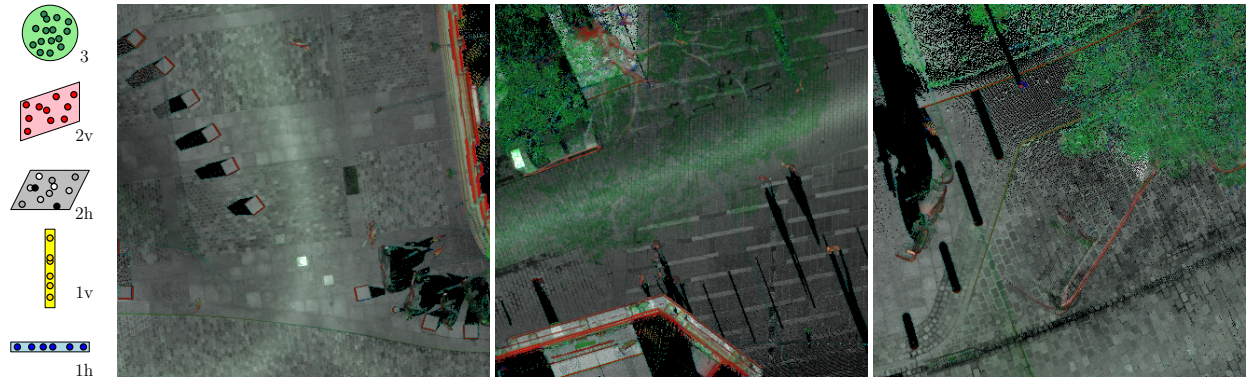

Figure 2: From left to right: (a) the schematic legend depicting the 5 geometric cases (b) the white track is a hot-spot like effect due to the specularity of the pavement that is not taken into account. Note that the cubes and façades are highly discernible and may be positioned accurately due to their red outlines. (c) The tree structure and horizontal extent is easy to interpret but has minimal impact on the reading of the objects below the vegetation (pavement, cars...). (d) This section has been acquired twice, the "ghost" car was not parked here during both acquisitions.

This defines for each point a 5-vector $D = (d_{1h}, d_{1v}, d_{2h}, d_{2v}, d_3)$ which is positive (since e_1 and e_3 are normalized, their z component is in $[-1, 1]$) and sums up to 1. D can thus be interpreted as the discrete probability vector accounting for the classification of a point into one of the 5 dimensional cases. The proposed point visualization style tries to address the easily interpretable communication of both the local geometry and the lidar texture measured by the radiometric attribute r. To convey the shape, we directly use the dimensionality vector D as barycentric coordinates to interpolate between 5 user-selectable colors $C = (c_{1h}, c_{1v}, c_{2h}, c_{2v}, c_3)$. For easy human interpretation the c_3 color corresponding to volumetric points should be green such that the trees which correspond to the majority of points with a high d_3 probability show up green in the final rendering.

2.3 Proposed ortholidar styling

The radiometric texture is highly informative and easily interpretable on horizontal surfaces as it resembles a single channel orthophotography. The radiometric values of vertical features will project vertically in few pixels, such that their vertically aggregated value is less informative and distracts the user from a straightforward interpretation. Lastly we have found that the radiometric values of linear and volumetric points ($d_2 \simeq 0$) have a high level of noise, due to partial footprints caused by silhouettes and occlusions. Thus their projected radiometric texture presents a high frequency content which is not as easily interpretable as the a horizontal surfacic texture. Thus, we propose to use the radiometric value only for horizontal features, which is luckily aligned with the user requirements. In order to prevent artifacts, the average radiometric value is weighted by d_{2h}, which leads to the following equation for the color of the ortholidar pixel $c(x, y)$:

$$c(x,y) = f\left(\frac{\sum_{p\in P_{xy}} d_{2h}(p)\, r(p)}{\epsilon + \sum_{p\in P_{xy}} d_{2h}(p)}\right) \frac{\sum_{p\in P_{xy}} D(p)\cdot C}{\sum_{p\in P_{xy}} 1} \quad (4)$$

where ϵ is an infinitesimal term preventing a division by 0, P_{xy} is the set of lidar points projecting to pixel (x, y), $D \cdot C$ denotes the scalar product in $\mathbb{R}^5$ and $f(\cdot)$ is a tone-mapping function that maps the range of radiometric values r to $[0, 1]$, such as a clamped linear mapping $r \mapsto \min\left(1, \max\left(0, \frac{r - r_{\min}}{r_{\max} - r_{\min}}\right)\right)$.

By choosing c_{2h} to be white, and other colors to be saturated, we address all but the last user requirements and approximately follow our guidelines regarding hue, intensity and saturation, with a cost-effective computation. In the figures, we used $c_{1h} = (0, 1, 1)$: yellow, $c_{1v} = (0, 0, 1)$: blue, $c_{2h} = (1, 1, 1)$: white, $c_{2v} = (1, 0, 0)$: red, $c_{2v} = (0, 1, 0)$: green.

3 DISTRIBUTED PROCESSING OF POINT CLOUDS

Lidar surveys are typically encoded in a number of files and may be seen as an ordered collection of records (the lidar point measurements) with both implicit attributes (Point ordering, filename ...) and explicit attributes (position in sensor coordinates, position in georeferenced coordinates using the sensor trajectory and calibration, amplitude, distance corrected reflectance, time...). They are thus a perfect fit for database view where a table corresponds to a survey (possibly spanning multiple files), columns correspond to attributes and rows to lidar point samples. A naive RDBMS approach is likely not to be tractable for these huge datasets as RDBMSs tend to not scale well for tables consisting of billions of rows. A first approach (Ramsey, 2014) is to pack coherent patches of points into a single row to limit the number of rows and scale to larger datasets. NoSQL databases have also been considered to manage huge lidar surveys (van Oosterom et al., 2015).

3.1 Spark Lidar IO

In this work, we are exploring an alternative approach of using the Spark framework to scale out and distribute the processing of huge lidar datasets in a cloud computing context. Spark is a rising technology with a large momentum which is built on the Resilient Distributed Dataset (RDD) abstraction which distributes datasets in the distributed memory of a cluster, possibly spilling out to disk. RDDs are resilient to lost tasks or hosts due to its functional programming approach and their representation of a job as a Directed Acyclic Graph (DAG) of tasks. It provides 4 higher level distributed libraries: Streaming for micro batch processing, GraphX for graph computation, MLlib for machine learning which is used by Liu and Boehm (2015) for lidar point cloud classification, and SQL. Spark SQL enables SQL operations on RDDs and external data sources by introducing a Dataframe API which is a columnar database-like view of a RDD which construction is analyzed and optimized at run-time. To enable the reading of a set of lidar files as a Spark Dataframe, we developed a scala library which provides:

- Header readers for the commonly used PLY and LAS file formats. This enables the loading of the file metadata, the point record schema (name, types and offset of each attribute), and the binary section definition (given by a byte offset, a record count, and record length and stride in bytes). Note that extending this to other uncompressed binary formats is trivial and only requires the implementation of the specific header reader.
- A PLY writer to write a partition (a portion of a RDD resident in the memory of a single node) to a (distributed) file system.
- A splittable Hadoop InputFormat that may be used to read such a binary section in a file-format agnostic manner given its point record schema.
- A Spark Relation provider that enables the construction of Spark Dataframes, reading only attributes that are used in the subsequent computations. Further work will use the Spark ability to project filters to prune entire file sections at loading time if a prior indexing scheme is available.

3.2 Point and tile aggregation for pixel styling

Since points contributing to a given output pixel may come from multiple file splits / partitions, the resulting pixel is the result of aggregating both within partitions and across partitions. Equation 4 may be computed by accumulating a vector $A \in \mathbb{R}^7$:

$$c(x,y) \quad = \quad f\left(\frac{A_r}{\epsilon + A_h}\right)\frac{A_D \cdot C}{\sum A_D} \tag{5}$$

with $A = (A_r, A_h, A_D)$:

$$A_r = \sum_{p \in P_{xy}} \quad d_{2h}(p)\, r(p) \quad \in \mathbb{R} \tag{6}$$

$$A_h = \sum_{p \in P_{xy}} \quad d_{2h}(p) \quad \in \mathbb{R} \tag{7}$$

$$A_D = \sum_{p \in P_{xy}} \quad D(p) \quad \in \mathbb{R}^5 \tag{8}$$

given that $\sum_{p \in P_{xy}} 1 = \sum A_D$ since $\sum D(p) = 1$. This enables a constant-memory per pixel accumulation to compute its final color in the ortholidar. Note that we could have accumulated only 6 quantities by accumulating the color $A_D \cdot C \in \mathbb{R}^3$ and the cardinality $\sum A_D \in \mathbb{R}$ instead, but we would then be unable to dynamically change the color palette C after the fact. In order to improve locality and to prevent the need for a final shuffle to group pixels by tiles for writing images to disk or over the network to the client, we group pixels by tiles in the distributed memory using a `RDD[Tile]`. We experimented both with dense and sparse tile encoding and found that sparse encoding of the accumulation image yielded best result due to the low relative overhead of storing the within-tile pixel offset which is a small integer compared to the 7 accumulated floating point values, and to the presence of scattered points measured at a long range that make many tiles contain few measurements only. Implementation-wise, `Tile(w,h)` initializes a sparse empty accumulative tile of dimension `(w,h)`, while its `_:+_` and `_++_` operators accumulate a tile with respectively a single point or another tile. This enables the pre-shuffle aggregation (combiner) using `_:+_` within a file split, the shuffling of these partially aggregated sparse tiles and the post-shuffle reduction using `_++_` across file splits. A call `tile.setRGB(canvas,f,C)` implements equation 5 to write to a `canvas` its pixel colors given the tone-mapping function f and the palette C.

3.3 Single tile computation

Given an input axis-aligned extent $[x_0, x_1] \times [y_0, y_1]$ and a target resolution (w, h), it is easy to leverage the lidar point cloud Spark loader to compute independently for each point whether it falls into the computed tile and if so in which pixel it falls, then to accumulate tiles and render the final ortholidar image. Note that `D3` is not selected as it is redundant and equal to `1-D1-D2`.

```
// loading of a HDFS directory PLY files
val location = "HDFS://ip:port/path/*.ply"
val tile = sqlContext.plyFiles(location)
// selection and renaming of relevant attributes
  .select('x, 'y, 'reflectance as 'r, 'D1, 'D2,
      abs('eigenVector1z) as 'z1,
      abs('eigenVector3z) as 'z3)
// box filtering
  .where(('x >= x0) && ('x < x1))
  .where(('y >= y0) && ('y < y1))
// transformation in tile coordinates
  .map {case Row(x: Float, y: Float, r: Float,
      d1: Float, d2: Float, z1: Float, z3:
      Float) =>
       Point(w*(x-x0)/(x1-x0), h*(y-y0)/(y1-y0),
           r, z1, z3, d1, d2) }
// local and global aggregation in a sparse tile
  .aggregate(Tile(w,h))(_:+ _, _ ++ _)
// write tile to a canvas using eq. 5
tile.setRGB(canvas,f,C)
```

3.4 Tile pyramid computation

A tile pyramid is a standard approach to preprocess raster images to provide precomputed versions at discrete zoom levels. In order to generate such a tile pyramid, relatively few modifications have to be performed to the previous algorithm. By starting at the base level, composed as an array of tiles, a pixel is then referred to by a tile id and a pixel id within that tile. Line 11 should be modified to output the tile id as the key of the output record for it to be routed to the given tile while the x, y coordinates still point to the in-tile pixel id. Line 13 is then modified to an `aggregateByKey` call to aggregate independently the different base tiles. Finally, line 15 is replaced by an iterative loop that hierarchically writes the tiles to disk, aggregate down tiles into quarter tiles and shuffle quarter tiles to merge all its 4 children to a parent tile. Iterations terminate at the first level where a single tile is written to disk. As a side effect, metadata informations may be written to enable easy usage of the generated tile pyramid such as geolocation metadata files for single files (.pgw, .prj...) or per pyramid level (.vrt...) or a simple HTML page presenting a webmapping interface (e.g. leaflet).

4 RESULTS AND DISCUSSION

4.1 MMS Dataset

The proposed approach has been deployed on a 10km long MMS dataset acquired by the IGN's Stereopolis vehicle (Paparoditis et al., 2012). This represents 339 tiles of 3 million points, thus 1017 million points in total. The tiles have already been processed by the dimensionality analysis, yielding uncompressed binary PLY files with many extra-attributes. In total, this dataset occupies 81.439 GB on disk, 28.476 GB of which are actually relevant for our processing (7 attributes $(x, y, r, e_{1z}, e_{3z}, d_1, d_2)$ in single precision float format gives 28 bytes for each of the 1017 million

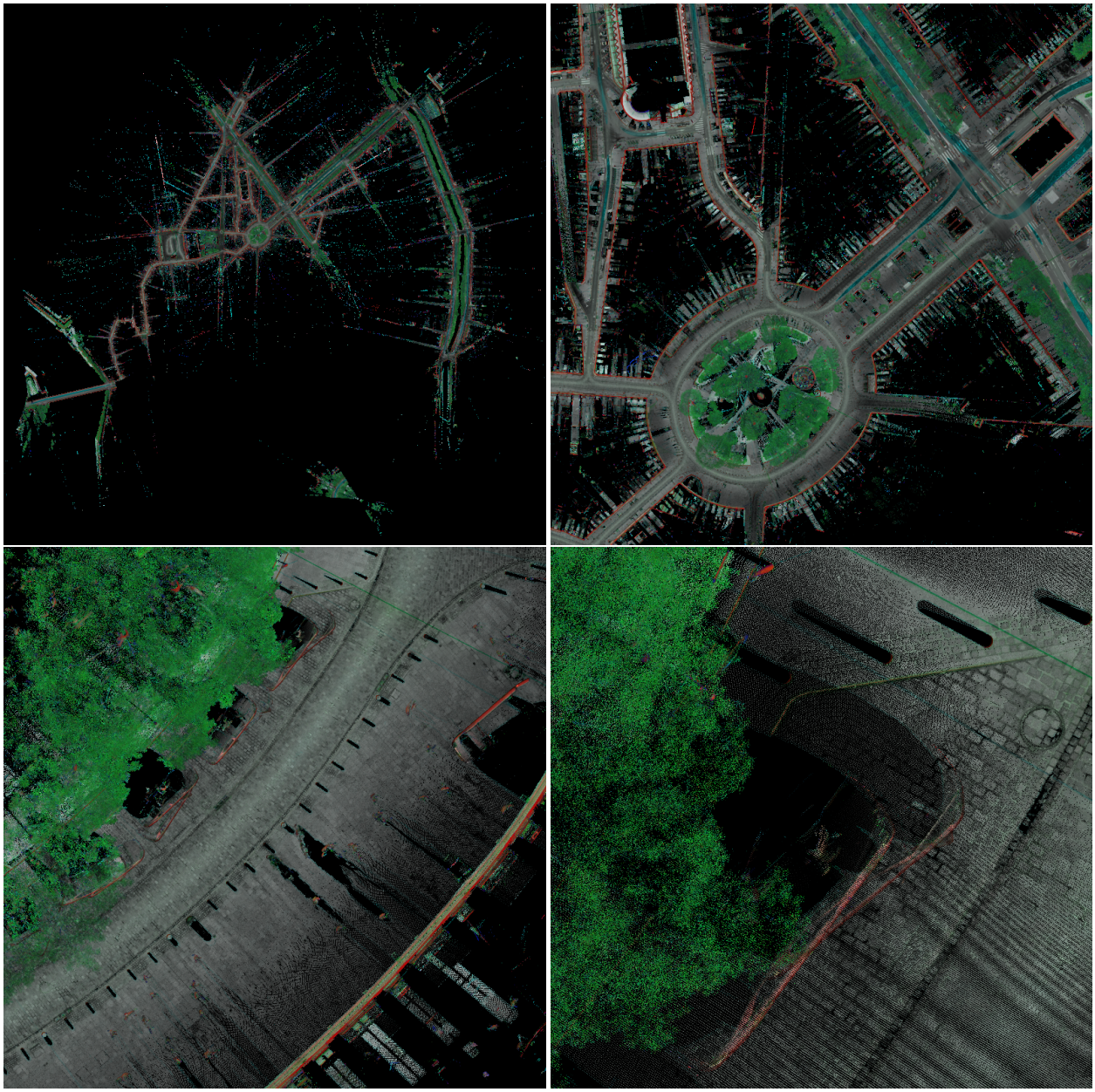

Figure 3: 800x800 screenshots with respective pixel scales of 256, 32, 4 and 1 cm. The hue denotes the local geometry analysis as being vertical (blue) / horizontal (cyan) linear structures, horizontal (gray) / vertical (red) surfaces or scattered points typically originating from vegetation (green). Saturated pixels are due to a majority of points having a non horizontal-surfacic neighborhood of homogeneous local geometry (dimensionality and horizontality). The brightness is given by the weighted reflectances on horizontal surfaces.

points). This dimensionality analysis has been performed using a standalone unoptimized C++ executable that processes independently each tile in less than 3 minutes. This overall preprocessing took 2 hours when dispatched over 8 nodes. Considering that this analysis is usually a basic preprocessing that is performed for subsequent processing steps, using it for visualization does not increase the overall computing costs. It however introduces a latency for visualization during its computation. Reducing or hiding the computing costs of this preprocessing is subject to future work, possibly by only performing this analysis on point clouds that have already been filtered by the viewing extent and undergo a pre-aggregation of point clusters.

This dataset was uploaded to a HDFS distributed file system accessible by a Spark cluster with 8 slaves. We devoted 6 GB of memory to the driver and the executors to prevent out-of-memory issues. The overall computation of the tile pyramid took 41 min at a resolution of 1cm and 17min at a resolution of 2cm. Sublinearity is due to the data-dependent nature of the algorithm, where tiles without points are not even considered. The cloud infrastructure provided by the IQmulus European project proposes an authenticated HTTP access to the HDFS directories. Thus the resulting PNG tiles may be served using a standard javascript webmapping html page for easy navigation in a web browser.

We experimented the embedding of this Spark application in a scala http server using spray[2]. Contrary to the Web-Map-Tile-Service (WMTS) style approach of the tile pyramid, the goal was to build an on-demand single image query service (similar to a Web-Map-Service - WMS). All the files were read at applica-

[2]Spray: http://spray.io/

tion start-up to compute the per-partition bounding boxes as a primary form of indexing in order to entirely cull partitions that are not visible in the query extent. Once cached the Spark SQL Dataframe effectively prunes out unneeded partitions which yield query completion times between 30s and 5 minutes depending on the query extent. Further work is needed here to achieve interactive response times through a tighter Spark SQL integration.

4.2 User feedback

Users gave a very positive feedback when presented the results. They found the provided ortholidar easy to interpret and to work with in their application, providing enough context (trees, buildings, cars...) while showing details on the ground (pavement, road markings...). Turning back to their initial requirements, we had to discuss that a white background exhibits unwanted artifacts: long range lidar returns tend to have low reflectance and thus appear as sparse dark dots which is disturbing on a white background.

4.3 Limitations

The primary limitation expressed by users is the lack of interpolation between points when zoomed in. The proposed approach only uses a form of nearest neighbor filtering for the output ortholidar, which may produce aliasing artifacts and is not able to cope with situations where the lidar sampling is lower density that the ortholidar resolution. As a side effect, the output ortholidar has high frequency content in these regions and is thus poorly compressible using standard image compression techniques. To address this concern and produce a smoother more compressible ortholidar (up to a to-be-defined smoothing extent parameter), multiple solutions have to be explored. For high quality at the cost of computation, a Laplacian smoothing approach (Vallet and Papelard, 2015) may be employed, but getting artifact-free tile boundaries is not straightforward to achieve. More simply, a linear filtering or a radial basis function filtering with a limited support may be added, which will only incur more network overhead due to the necessity to shuffle boundary pixels to all the tiles that fall into their filter extent. Addressing these filtering issues is left for future work.

5 CONCLUSION

Through this experiment, we have proved the possibility of handling massive point clouds using the Spark bigdata framework. While computing times are not ground-breaking compared to optimized distributed C++ code, we show that the abstraction provided by the Spark framework allows to express simply in a few lines of code algorithms that run distributed on a cloud in a resilient manner (e.g. robust to partition and executor losses).

Future work will further investigate the coupling of the two proposed approaches to get the best of both: efficient direct access of huge precomputed areas using the tile-based approach and efficient and memory-efficient access to subsets using the partition-filtering approach. Furthermore, integrating this work with Geotrellis (Kini and Emanuele, 2014), a Spark-based geospatial raster library, would open new possibilities. Lastly, on-demand or Just-In-Time processing is another promising outlook so as to only compute and store intermediate results (such as the dimensionality features) when required and only temporarily cache the results.

ACKNOWLEDGEMENTS

This material is based upon work supported by the FP7 IQmulus Integrating Project, FP7-ICT-2011-318787, *A High-volume Fusion and Analysis Platform for Geospatial Point Clouds, Coverages and Volumetric Data Sets*, and the TerraMobilita Project of the Cap Digital Business Cluster. We also thank Electricité de France (EDF) for their discussions and feedbacks on the proposed ortholidar visualization.

References

Armbrust, M., Xin, R. S., Lian, C., Huai, Y., Liu, D., Bradley, J. K., Meng, X., Kaftan, T., Franklin, M. J., Ghodsi, A. and Zaharia, M., 2015. Spark sql: Relational data processing in spark. In: Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, SIGMOD '15, ACM, New York, NY, USA, pp. 1383–1394.

Demantke, J., Mallet, C., David, N. and Vallet, B., 2011. Dimensionality-based scale selection in 3D lidar point cloud. International Archives of Photogrammetry Remote Sensing and Spatial Information Sciences, Volume XXXVIII-5/W12 pp. 97–102.

Eldawy, A. and Mokbel, M. F., 2014. The Era of Big Spatial Data. In: Proceedings of the International Workshop of Cloud Data Management CloudDM 2015 co-located with ICDE 2015, Seoul, Korea.

Kini, A. and Emanuele, R., 2014. Geotrellis: Adding Geospatial Capabilities to Spark. http://spark-summit.org/2014/talk/geotrellis-adding-geospatial-capabilities-to-spark.

Liu, K. and Boehm, J., 2015. Tree Crown Classification Using Cloud Computing. International Archives of Photogrammetry Remote Sensing and Spatial Information Sciences, Volume XL-3/W5.

Paparoditis, N., Papelard, J.-P., Cannelle, B., Devaux, A., Soheilian, B., David, N. and Houzay, E., 2012. Stereopolis II: A multi-purpose and multi-sensor 3D mobile mapping system for street visualisation and 3D metrology. Revue Française de Photogrammétrie et de Télédtection 200, pp. 69–79.

Ramsey, P., 2014. A PostgreSQL extension for storing point cloud (LIDAR) data. https://github.com/pramsey/pointcloud.

Richter, R., Discher, S. and Döllner, J., 2015. Out-of-core visualization of classified 3d point clouds. In: 3D Geoinformation Science: The Selected Papers of the 3D GeoInfo 2014, Cham: Springer International Publishing, pp. 227–242.

Vallet, B. and Papelard, J.-P., 2015. Road orthophoto/DTM generation from mobile laser scanning. International Annals of Photogrammetry Remote Sensing and Spatial Information Sciences, Volume II-3/W5.

van Oosterom, P., Martinez-Rubi, O., Ivanova, M., Horhammer, M., Geringer, D., Ravada, S., Tijssen, T., Kodde, M. and Gonalves, R., 2015. Massive point cloud data management: Design, implementation and execution of a point cloud benchmark. Computers & Graphics 49, pp. 92 – 125.