IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received July 18, 2018, accepted October 3, 2018, date of publication October 23, 2018, date of current version November 19, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2877319

A Developer-Friendly "Open Lidar Visualizer and Analyser" for Point Clouds With 3D Stereoscopic View

JORGE MARTÍNEZ, OSCAR G. LORENZO, DAVID L. VILARIÑO, TOMÁS F. PENA, (Senior Member, IEEE), JOSÉ C. CABALEIRO, AND FRANCISCO F. RIVERA

Centro Singular de Investigación en Tecnoloxías da Información, Universidade de Santiago de Compostela, 15705 Santiago, Spain

Corresponding author: Jorge Martínez (jorge.martinez@usc.es)

This work was supported in part by the Ministry of Education, Culture and Sport, Government of Spain under Grant TIN2016-76373-P, in part by the Xunta de Galicia under Grant GRC R2014/008, Grant R2016/045, and Grant R2016/037, in part by the Consellería de Cultura, Educación e Ordenación Universitaria (accreditation 2016-2019), under Grant ED431G/08, and in part by the European Union (European Regional Development Fund - ERDF).

ABSTRACT Light detection and ranging is being a hot topic in the remote sensing field, and the development of robust point cloud processing methods is essential for the adoption of this technology. In order to understand, evaluate, and show these methods, it is a key to visualize their outputs. Several visualization tools exist, although it is usually difficult to find the suited one for a specific application. On the one hand, proprietary (closed source) projects are not flexible enough because they cannot be modified to adapt them to particular applications. On the other hand, current open source projects lack an effortless way to create custom visualizations. For these reasons, we present Olivia, a developer-friendly open source visualization tool for point clouds. Olivia provides the backbone for any type of point cloud visualization, and it can be easily extended and tailored to meet the requirements of a specific application. It supports stereoscopic 3-D view, aiding both the evaluation and presentation of processing methods. In this paper, several cases of study are presented to demonstrate the usefulness of Olivia along with its computational performance.

INDEX TERMS Data visualization, LiDAR point clouds, open source software, stereoscopic 3D.

I. INTRODUCTION

Light Detection and Ranging (LiDAR) is a remote sensing technique analogous to radar, but it uses light instead of radio waves. The scanner measures the time for a laser pulse to travel from the sensor to the target, which is used to derive the distance. After combining these measurements with inertial measurement unit (IMU) and GPS data, a set of three-dimensional points in a georeferenced coordinate system is obtained. LiDAR has some advantages compared to traditional techniques such as photogrammetry: short data acquisition and processing times; relatively high accuracy; reduced cost; ability to penetrate through the canopy; high flexibility to atmospheric conditions.

Practically, a point cloud can be seen as an accurate digital record of space. This is a valuable information for a wide range of applications, being the most notorious the generation of Digital Elevation Models (DEMs). The reason for this is that first, it has been proven to enable the production of highly accurate and high resolution DEMs [1], and second, nearly all applications need to perform a ground filtering. Practical applications include land-cover classification, urban modeling, hydrologic modeling, coastal monitoring, forest inventory and archaeological retrieval, among many others.

In recent years, a gradual cost reduction and accuracy increase of the scanners have been experimented. The surveying methods have also evolved to lower cost approaches, from the use of Airborne Laser Scanning (ALS) to Unmanned Laser Scanning (ULS) or Mobile Laser Scanning (MLS). These factors have lead to a rapid adoption of LiDAR, which has drew the attention of many researchers.

Point cloud processing methods often consist in multi-step frameworks with complex algorithms, and the correctness of each step needs to be evaluated to ensure the overall robustness. To aid the understanding, evaluation and presentation of these steps, visualization is highly demanded. Several tools exist to visualize point clouds, but it can be difficult to choose

the appropriate one for each particular purpose. When developing processing methods, displaying very specific drawings are frequently needed. Closed source projects do not offer flexibility because the source code cannot be adapted to develop specific drawings. Alternatively, in open source projects, attempting to create such custom visualizations can be difficult and time consuming because the broad scope of the projects or the lack of guidance.

To deal with these issues, we introduce a new open source stereoscopic 3D visualization tool for LiDAR point clouds. This tool, named Olivia, can be easily extended and tailored to the needs of any particular point cloud processing application. It could also be adapted to other point cloud sensor sources if needed with some additional effort. The implementation, architecture and functionalities of the tool are described in the paper. By performing a visual inspection of the results, valuable feedback can be gathered to refine the algorithms used in these use cases.

The rest of the paper is organized as follows: in Section II several open source visualization tools are described and compared with our proposal. Section III introduces the Olivia architecture, while Section IV describes its main functionalities. In Section V, four use cases that take advantage from our tool are described. In Section VI the performance in terms of rendering speed and memory usage is analyzed. Finally, concluding remarks are presented in Section VII.

II. RELATED WORK

The visualization of point clouds can be easily done with several existing tools, but displaying application-specific content may not be straight forward. Many of the tools are not open source thus they cannot be modified to add additional functionalities, because of this only open source projects are considered in this brief summary. We selected three of the most popular tools, an overview of them and the proposed tool is shown in Table 1.

TABLE 1. Popular open source visualization tools and olivia.

Name	*Release*	*Language*	*License*
ParaView	2002	C++	BSD-3
CloudCompare	2009	C++	GPL-2
Point Cloud Library	2011	C++	BSD-3
Olivia	2018	Java	GPL-3

ParaView [2] is an open-source, multi-platform data analysis and visualization application. The project started in 2000 as a collaborative effort between Kitware Inc. and Los Alamos National Laboratory. Its main strength is the ability to analyze extremely large datasets using distributed memory computing resources. It allows to load, display, filter, query and animate data. Its built-in features include subsampling, cropping, time-varying data, streaming and parallel processing, etc. To grant more processing capabilities, the ParaView-PCL Plugin can be installed to provide access to the algorithms and VTK filters of the Point Cloud Library (PCL). Other features include Python integration, batching processing and virtual reality visualization. A JavaScript library is also provided to support interactive visualization in a web browser. The use of this tool is described in detail in the 251-page ParaView Guide (Community Edition), as the number of functionalities in this tools is impressive. It is also possible to write custom applications, although it can be overwhelming to deal with such a big project.

CloudCompare [3] is a 3D point cloud and triangular mesh editing and processing software. The project started in 2003 as a collaboration between Telecom ParisTech and the R&D division of EDF and it has been released in the public domain around 2009. It was originally designed to perform direct comparison between dense 3D point clouds, making it suitable for change detection and analysis. Its built-in features include many processing algorithms (registration, resampling, color/normal vectors/scalar fields management, statistics computation, etc.) as well as display enhancement tools (color ramps, color and normal vectors, calibrated pictures, OpenGL shaders...). This tool is oriented to processing and lacks an easy way of creating custom visualizations.

Point Cloud Library (PCL) [4] is a large scale, open project for 2D/3D image and point cloud processing. The project started in 2010 at the robotics research lab Willow Garage being first officially released in May 2011. It contains numerous state-of-the art algorithms including filtering, feature estimation, surface reconstruction, registration, model fitting and segmentation. The library also offers a visualization module providing methods for rendering and setting visual properties, drawing basic 3D shapes, histograms, range images, and geometry and color handlers, which is handy for analyzing the algorithms within the library. This tool is also oriented to processing, but future developments of this visualization module should be taken into account. Also the C++ language presents a higher barrier to entry than Java.

Summarizing, these tools have been developed for several years, and they provide robust and broad functionalities, although we observe two disadvantages. First, most of them are oriented to point cloud processing instead of visualization. Second, the learning curve to use, modify or improve these tools is high. For these reasons, we have developed an open source software focusing on the following features:

- Designed for visualization (no processing) purposes.
- Ease to quickly develop customized visualizations.
- High portability across systems.

Olivia does not compete with the tools described above, in fact, initially, our tool offers less functionalities in comparison. The goal is to build an open source tool which can be easily adopted by the user with minimum friction in order to create custom visualizations. All the basic functionalities expected from a visualization tools such as point cloud rendering, geometry drawing and point/cluster selection are already provided. Some additional functionalities are implemented as well, such as stereoscopic 3D or neighbors display. Users can build in a short time frame custom visualizations and take advantage of these embedded functionalities.

Also, thanks to its highly portability it can be used in almost any computer. Our aim is to help researchers and users alike to understand, evaluate, and show point cloud processing algorithms in a quick and easy way requiring only basic Java knowledge.

III. OLIVIA ARCHITECTURE

A. IMPLEMENTATION

Olivia is written in the Java programming language for two main reasons. First, Java is one of the most -if not the most- popular programming language. It has a higher popularity rank than many other languages (see TIOBE [5], RedMonk [6] and IEE Spectrum [7] ratings) and clearly higher rank than C++, the language used in the aforementioned tools. Second, Java is well known for its great portability, the source code is compiled to byte code that can be executed in any platform with the use of the Java Virtual Machine (JVM).

This software uses the Open Graphics Library (OpenGL), the industry's most widely used, supported and best documented API for rendering 2D and 3D graphics. To be able to use OpenGL from Java, the wrapper library Java OpenGL (JOGL) [8] is used, which implements Java bindings for OpenGL providing full access to the API, as well as almost all vendor extensions (OpenCL, OpenMAX and OpenAL) through the use of the Java Native Interface (JNI). JOGL integrates with the Java own windowing system, the Abstract Window Toolkit (AWT), Swing and SWT widget sets, and with custom windowing toolkits using the NativeWindow API, as well as providing its own Native Windowing Toolkit (NEWT). The points are stored into Vertex Buffer Objects (VBOs), which allows vertex array data to be stored in video device memory for faster rendering. An interleaved vertex format is used to improve memory locality for each vertex.

Furthermore, the wrapper library JavaCV [9] is also used to access to some functionalities provided by the OpenCV [10] and FFmpeg [11] libraries. Finally, Substance [12] is used for the look and feel of the graphical user interface.

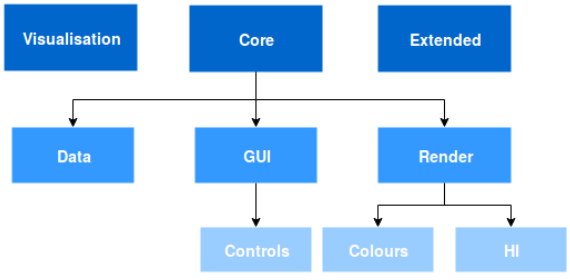

FIGURE 1. Package structure of Olivia.

B. PACKAGE STRUCTURE

Olivia is split into three main packages: core, extended and visualization as shown in Figure 1. The core package implements all the basic functionalities, the extended package implements functionalities that are shared across visualizations, and the visualization package implements all the visualization-specific functionalities. In order to create a custom visualization, it is necessary to implement an own visualization package.

- **Core**: This package is the backbone of the application and implements the main functionalities that are common to any type of visualization. It acts as a layer of abstraction providing data structures and methods, from which users can benefit making little to none modifications. The package stores the main class of the application, which parses the arguments and manages the creation of visualizations. This package is split into the following subpackages.
  - Data: Specifies the characteristics of the data. There are three structures available: point, point array and cluster. A point is understood as a 3D point. These points are encapsulated in a point array, which is a structure that provides access to the rendering methods. This point array can be encapsulated into a cluster, a structure with an identifier and some additional information such as the sum of the coordinates of the points.
  - GUI: Defines the graphical user interface, this is the graphical components and its controls. One of these components is reserved for the visualizations, which can be modified by the visualization package.
  - Render: Provides all the methods related to the rendering. This package also manages the plotting of OpenGL primitives (e.g. lines, triangles, quads, etc.), the movement of the camera and the screen capture (image or video). Particularly two features are controlled by this package:
    - Colors: Manages the colors of the points.
    - HI: Manages the human interaction through the mouse and keyboard. The selection of points is handled using a Ray-Casting technique.
- **Visualization**: This package contains the visualization-specific logic. A visualization needs to implement three main tasks: reading the data, displaying the data and creating the visualization panel. These methods are easy to implement by extending the methods inherited from the core package. Three visualization types are incorporated to serve as guidance for developing tailored visualizations: basic, segmenter and classifier.
- **Extended**: This package holds structures that are common to some visualizations, but are not common to all, thus they are not included in the core package. In this manner the duplication of code is minimized.

C. GRAPHICAL USER INTERFACE

An overview of the interface is illustrated in Figure 2. The layout of the tool is composed by a menu bar holding the main options and three panels. The control panel, labeled 1 in the figure, holds some quick options (camera speed, point size,

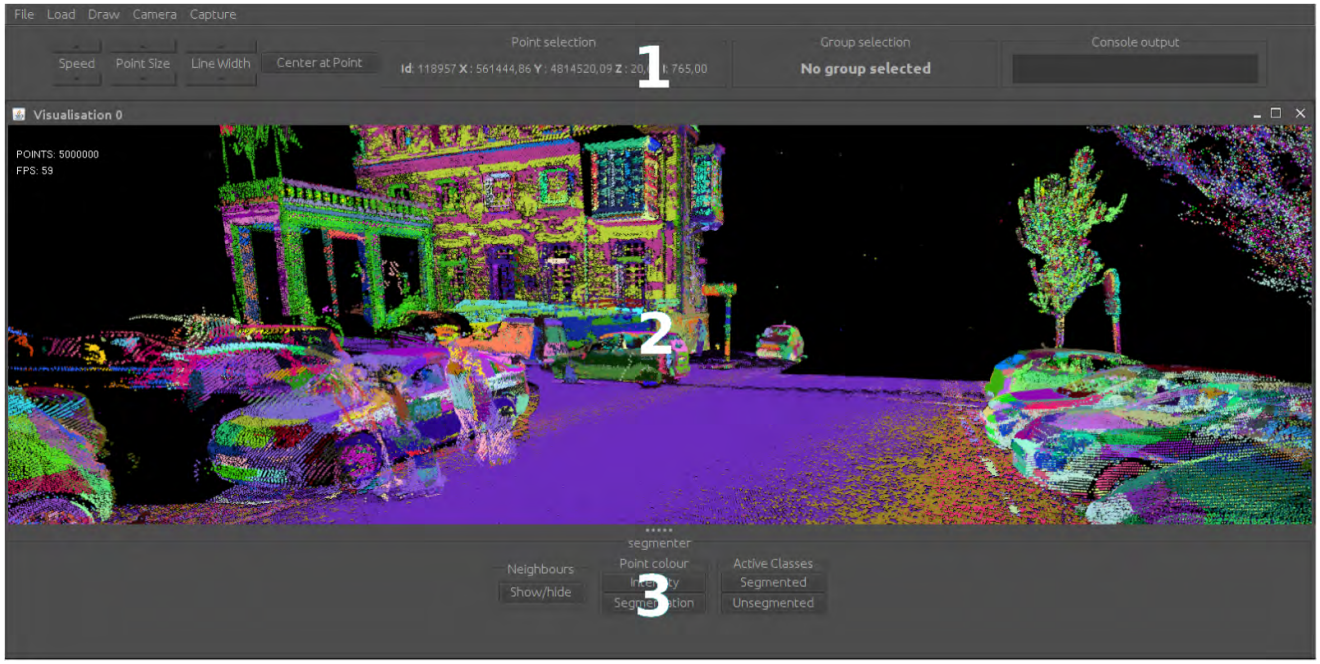

FIGURE 2. Screen capture of a segmentation visualization showing the panel composition: control panel (1), render panel (2) and visualization panel (3).

line width, center at point), the information about the selected point and cluster, and the console output. This console gives feedback about the execution of the tool, for displaying errors and aiding troubleshooting. The render panel, labeled 2 in the figure, is where the rendering is shown. Several visualizations can be displayed in the render panel, which are resized automatically. The visualization panel, labeled 3 in the figure, is set free for visualization-specific controls.

IV. USAGE

A. FUNCTIONALITIES

Currently, Olivia main functionalities are the following.

- **Point selection**: Points can be selected using the left mouse click showing information about the point in the control panel. Additional information can be displayed in the visualization panel.
- **Cluster selection**: Clusters can be selected using the mouse wheel click showing information about the cluster in the control panel. When the cluster is selected only that cluster will be shown to give a clear view of it. This is particularly useful for analyzing segmented point clouds, when isolating a specific cluster makes the task easier. Note that this functionality is related to clustering methods.
- **Neighbors visualization**: The neighbors of each point can be displayed on any visualization if a neighbors file is available. Currently Olivia supports the format used in the Fast Library for Approximate Nearest Neighbors (FLANN) [13]. Almost every point cloud processing method needs to compute the neighborhood of the points, so its evaluation is a key issue. If this option is activated, the neighborhood of the selected point will be shown, displaying a connection from that point to its neighboring points as well as the distance or weight in each connection.
- **Drawing of OpenGL primitives**: Different representation of the point streams can be plotted. This allows to draw primitives such as lines, triangles and quads on top of the point cloud. This is useful for displaying the output of different algorithms. For example, the hull of a set of points or the normal vector of each point can be rendered overlaying the point cloud.
- **Camera state**: It is possible to save and load the camera state, namely, its position and rotations. This facilitates the comparison of point clouds across different executions of the algorithm under analysis.
- **Camera mirroring**: The camera of the visualizations can be synchronized, meaning that all visualization will move simultaneously. This is helpful for comparing the same point cloud in different visualizations. For example, the user can run two different segmentation algorithms for the same point cloud, load each segmentation in one visualization panel, and then activate the camera mirroring. This way the position of the point clouds will be the same regardless the point cloud which is being moved, making the visual comparison much easier.
- **Stereoscopic 3D view**: It is possible to render the scene twice, producing one image for the left eye and other image for the right eye of the viewer. These images are combined when using 3D glasses to give the perception of depth. This feature is extremely useful for both in-depth analysis of the point cloud and for

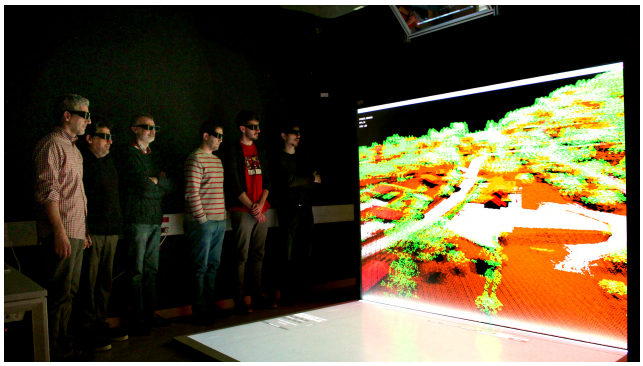

FIGURE 3. Demonstrating classification results using the 3D stereoscopic mode in the Airborne LiDAR Solutions stand. II Edition of Inside the Lab, a Technological Demonstration Day at CiTIUS.

demonstration purposes as shown in Figure 3. Note that because the scene is rendered twice, the computational cost doubles, which can be a performance issue for less powerful computers.

- **Screen capture**: Both image and video can be captured. Only the render screen is captured, hiding the other panels and taking into account the current resolution. If the stereoscopic 3D is enabled, two videos (left and right) are recorded in order to build the 3D video later.

B. GETTING THE TOOL

Both the source code and the executable are available in the official repository: https://github.com/citiususc/olivia. The application can be automatically built using the project management and build automation tool Apache Maven [14]. Installation and use instructions are provided in the repository. Because of the open nature of the project, developers can contribute to its development. We hope to bring the community together to improve Olivia.

C. CREATING CUSTOM VISUALIZATIONS

The main advantage of Olivia is its simplicity to add new tailored visualizations. Developers only need to be familiarized with the Java programming language, although some OpenGL knowledge may be required for more complex visualizations. Adding a new visualization is performed by adding a new package extending the default visualization package. Then, three simple elements must be implemented: 1) read the data, 2) display the data and 3) create the visualization panel. These elements are easy to implement by using inherited functionalities provided by the tool. Displaying the point cloud can be performed by simply calling the drawing method of the point array structure holding the points. Source code for three different visualizations are included to provide examples on how to add custom visualizations, and make the development easier. If there is the need of drawing functionalities not supported by Olivia, these should be implemented in Java classes within the custom visualization package. Only if these functionalities benefit all types of visualization will be added in the core package.

V. EVALUATION AND RESULTS

In this section the usefulness of the tool in four use cases is shown, ranging from common point cloud processing scenarios to specific applications.

A. VISUALIZING NEIGHBORS

The most common step for point cloud processing is the calculation of neighbors. Several neighborhoods can be chosen for this task: kth Nearest Neighbors (kNN), spatial neighbors (voxel, sphere, cylinder...), feature neighbors, etc. The output of these methods is used as input to nearly every subsequent processing step, thus the neighbors calculation is a critical task of any framework and it must be evaluated carefully. In this example the calculation using a voxelized neighborhood is inspected. For this, first an octree is created to speed up the search. The vertices of the octree nodes are saved in a file and then loaded as quads primitives, as shown in the left image of Figure 4. With a quick glance we can have some awareness about the depth of the octree and how it is partitioned. This can also be done for only the nodes containing points as shown in the right image of the figure. In this example, the stopping criteria is related to the number of points: a node will continue splitting until the number of points inside is lower than a certain threshold. This overview of the leaf nodes can help to select the appropriate threshold.

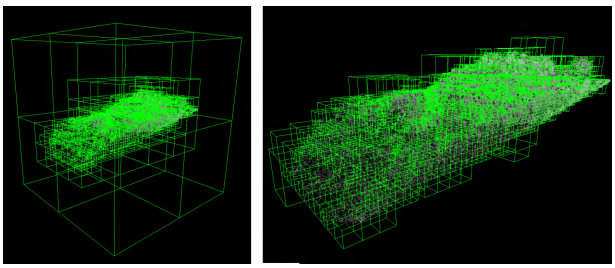

FIGURE 4. Octree nodes showed in green: All (left), only leafs (right).

Afterwards the result of the computed neighborhood is examined. For this, we provide the capability of displaying the neighbors of each point. In Figure 5 the neighbors of a conflictive building point (in blue) are shown, using three different neighborhoods. This point at the boundary of the building shares neighbors with both the building itself (right) and a nearby tree (left). This situation can cause problems for further processing, e. g. estimating the normal vector of the point. Visualizing the neighbors can help to verify that the selected neighborhood model is appropriate. The feature of displaying the neighbors of a selected point is inherited by any type of visualization whenever a file describing the neighbors is available.

B. VISUALIZING SEGMENTATIONS

Other common step in point cloud processing is the segmentation. In this section our procedure to evaluate the segmentation proposed in [15] is described. In summary, this method performs two region-growing iterations: the first one using

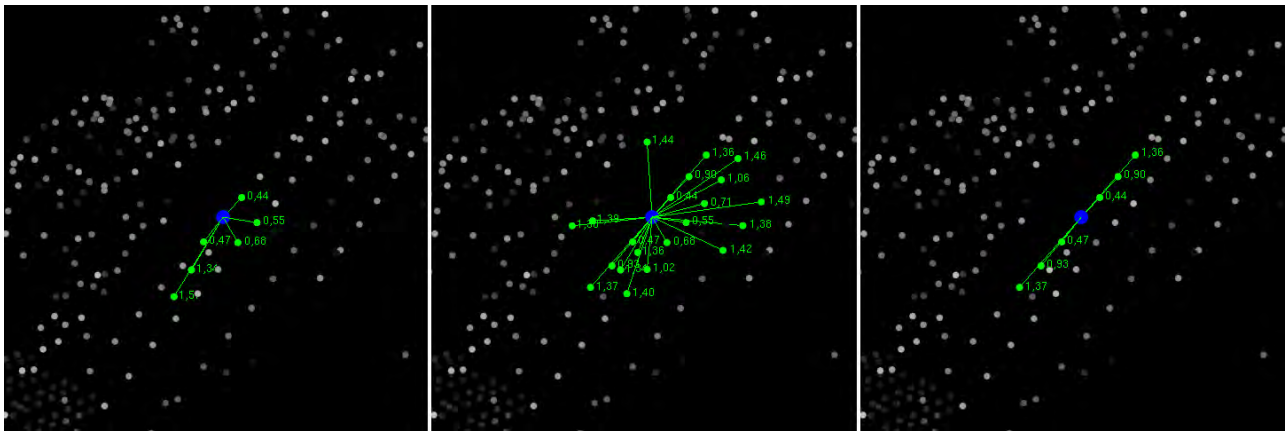

FIGURE 5. Roof point (in blue) and its neighbors (in green) for different methods: Triangular Irregular Network (left), sphere (center) and orientated cylinder (right).

the normal vector to extract only planar patches and a second one without it in the oversegmented clusters. There are some stages in this method which must be evaluated. First, the hull of the point cloud is calculated in order to estimate the density of the point cloud. To visualize this, the points of the hull have to be stored in a file. Then, the resultant hull could be plotted to visually verify that the calculation is correct. Once the point density is estimated, the size of the neighborhood needed to compute normal vectors is chosen accordingly. We can also perform a visual inspection of the normal vectors. As mentioned before, the value of the normal vectors needs to be stored in a file in order to visualize them afterwards. In Figure 6, the visualization of the normal vectors is shown. For the end user, it is important to see that sometimes the computation produces tricky situations, such as in boundaries among objects and areas with irregular point density.

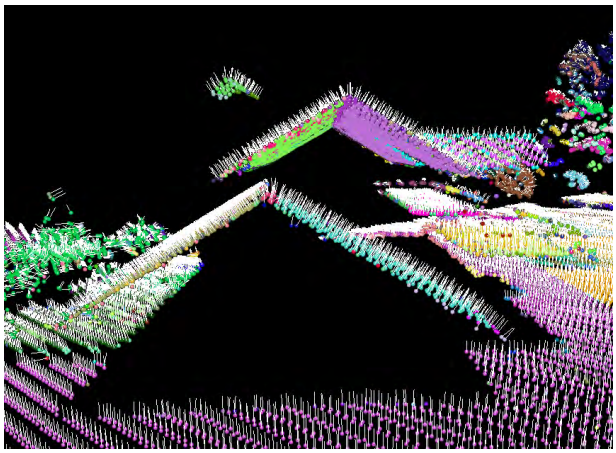

FIGURE 6. Normal vectors (white lines). Points colored by cluster.

After verifying that the previous steps are correct, the segmentation result itself is analyzed. Both points and clusters can be selected with the mouse to provide information such as identifier, coordinates, intensity, etc. Furthermore, if a cluster is selected only points of that specific cluster are displayed. This provides a clean overview of the cluster, allowing to quickly recognize its shape and boundaries. Some road segments can easily be identified in this particular segmentation, thus giving the idea that the algorithm is well suited for road classification. If the road clusters are selected, it can be quickly identified that they also include some asphalt of entries to houses nearby. This can be a problem depending on the application. Hints like this can be easily highlighted using the cluster selection feature.

Finally, different segmentation results can be compared. Although extensive research has been done in the quantitative evaluation of segmentation methods, it is still difficult to assess whether one result is more accurate than other [16], [17], thus visual inspection is still important to accomplish this task. The comparison can be performed among different segmentation methods or within the same method among different parameters. In this example we address the later. First, two segmentations are loaded, each one in one window, and the camera mirroring option is activated. This will facilitate the task by synchronizing the camera movement among the segmentations. Furthermore, the selection can also be synchronized, greatly simplifying the comparison of single clusters between segmentations. For example, in Figure 7 the effects of modifying the intensity threshold parameter can be observed. The default value is set to 7.5% of the intensity range (right). If the threshold is changed to 15% (left), we see that the cluster is able to include also points from the top road segment, but also includes more points near the boundaries of the road which belong to the ground. Olivia helps the user to identify and evaluate this kind of trade-offs.

C. VISUALIZING CLASSIFICATIONS AND BEYOND

As other visualization tools, the results of classification can be displayed. Each class has its own color, and it is possible to enable or disable the drawing or each class. For example, all classes could be disabled except the ground class, to have

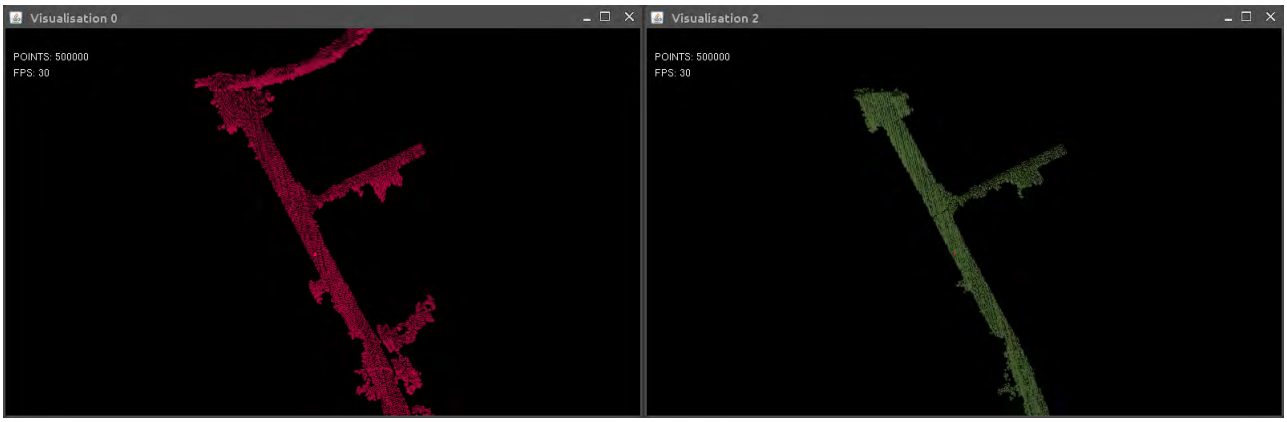

FIGURE 7. Usage of both selection and camera mirroring. Comparing the same cluster changing the intensity threshold in the segmentation process.

a clear view of the terrain. The classification labels must be the same as the defined in the LAS Specification [18].

The classification of the point cloud is the first stage for a broad number of applications. One of them, is the estimation of building footprints. Different algorithms can be used to compute the hull of the buildings. In a previous comparative of these methods [19] we used Olivia in order to visualize how they behave in different scenarios. To this end, both the calculated hulls and areas for each algorithm can be plotted, as shown in Figure 8. This allows us to quickly identify the shortcomings of the algorithms for different building shapes.

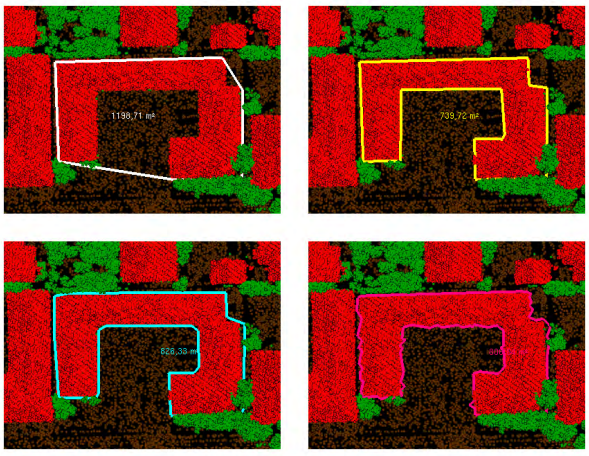

FIGURE 8. Computed hulls and areas using different algorithms; from left-to-right to top-to-bottom: convex, concave, exterior, voronoi.

D. VISUALIZING SPECIFIC APPLICATIONS: ADAPTATION TO A LANDING SITE DETECTION

More specific applications will also benefit from the functionalities provided by Olivia. In this section we present how the tool is used to analyze a Landing Site Detection Algorithm (LSDA) previously developed in [20]. In short, the algorithm classifies the suitability of the points for an helicopter to land. This classification takes into account the features for both the terrain (planarity, slope, etc.) and the helicopter (skid size, rotor size…). There are several classification labels, which are color coded for visualization, from worse to better: unsuitable (red and light red), risky (blue), suitable (yellow, light green, green, and white).

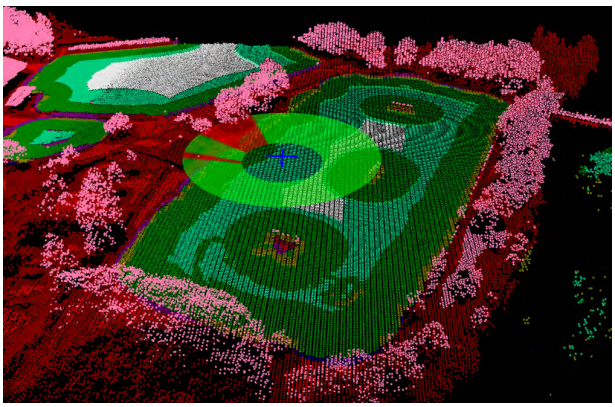

FIGURE 9. Result of the LSDA, the suitability for landing is represented in different colors. The approximation cone is shown for the selected point.

The result of executing the algorithm in a particular case is shown Figure 9. While in the white areas, the helicopter can perform an approximation from any angle, in the rest of suitable areas, obstacles may invalidate some approximation angles. Thus it is necessary to plot from where the helicopter can come in those cases. For this, the approximation cone calculated by the algorithm is stored in a file, and then, OpenGL methods are used to plot our customized approximation cone (see Figure 10). The cone shows in green the angles from where the helicopter can approximate, and in red the angles discarded for the presence of obstacles. Its drawing is triggered by the selection of a point. Olivia allows us to short the time and effort needed to evaluate the results.

VI. PERFORMANCE ANALYSIS

Rendering speed and memory footprint are analyzed in order to measure the performance of Olivia and give an idea of the hardware requirements for a particular case of study.

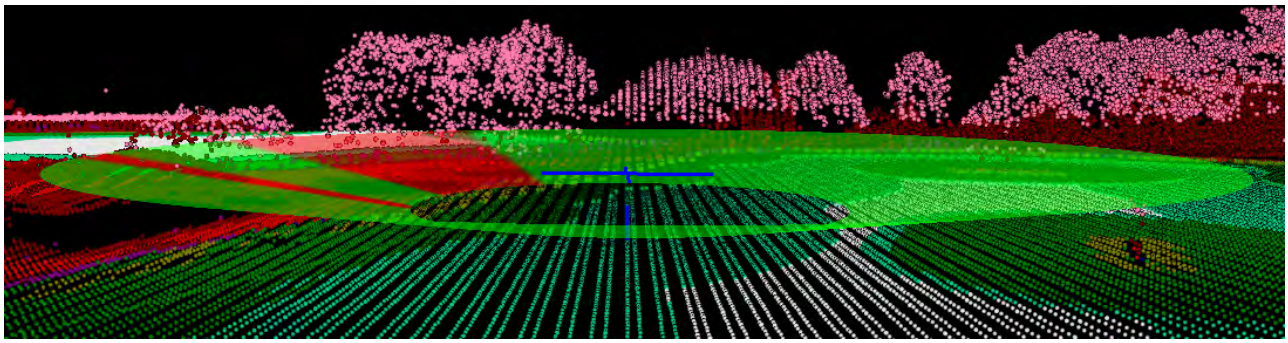

FIGURE 10. A detailed view of the approximation cone: landing position (blue bars), valid approximation angles (in green) and invalid approximation angles (in red)

These measurements should be taken as lower limits, as further developments could improve performance considerably. A MLS point cloud of a street from Narón (Spain) is selected for the benchmark given its high density. The point cloud is split into five samples of different sizes ranging from 1 to 75 million points. Benchmarking was carried out in a Linux (Ubuntu 14.04.5 LTS) desktop computer with an Intel Core i7-4790 processor and 16 GB of RAM memory. Each sample is inspected using the basic visualization mode.

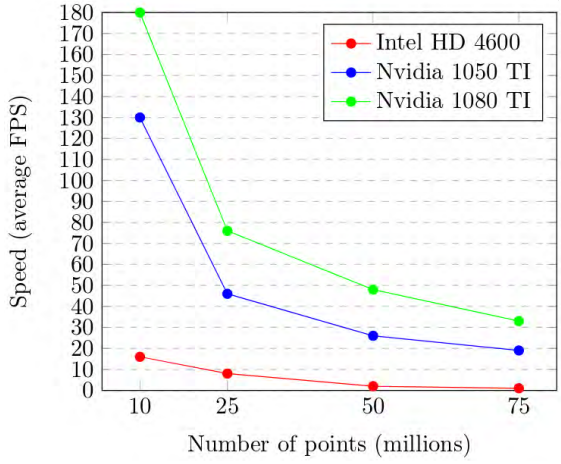

FIGURE 11. Rendering speed of Olivia for different graphic devices.

The average frames per second (FPS) is used as metric to measure rendering speed. This value is provided by a built-in method within the JOGL library. Three graphic devices were used to represent different ranges: an integrated Intel HD4600 for low-range, a Nvidia GTX 1050 Ti for mid-range and a Nvidia GTX 1080 Ti for high-range. Results are shown in Figure 11. While the integrated graphic struggles to achieve a smooth rendering (30 FPS) with 10 million points, both dedicated graphics achieve fluid renderings until 50 million points. For higher amounts of points only the high-range graphic card still offers a smooth rendering.

The memory footprint is gathered for host (RAM) and device (VRAM) memory. Furthermore, for the host memory, both allocated and actually used memory are measured. The host stores the full point data and Java-related objects, while the device only stores the coordinates and colors of the points. We use the NetBeans [21] Java profiling tool and the Nvidia X Server to measure host and device memory, respectively. Results are shown in Figure 12. The memory consumption for both host and device scales linearly. Note that for the host memory, Java allocates an excess of memory if plenty of memory still available, which can mislead the real memory usage.

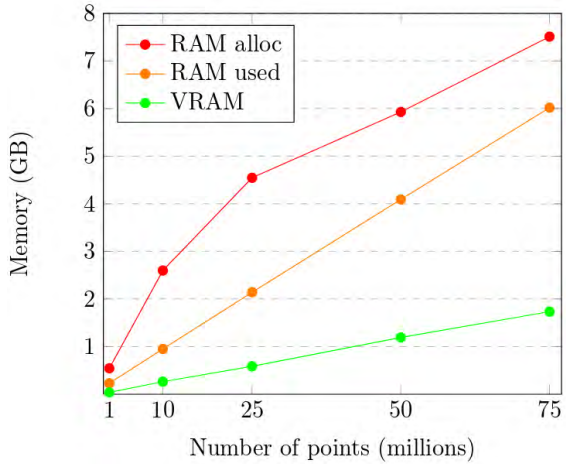

FIGURE 12. Memory usage of Olivia for host and device memory.

As expected, the hardware requirements directly depend on the number of points to be displayed, hence they will vary from one user to another. Nevertheless, because is common to work with points clouds of a few tens of millions of points, we recommend a system with at least 8 GB of RAM memory and a dedicated graphic device with 2 GB of VRAM memory.

VII. CONCLUSION

LiDAR point cloud processing is a hot topic, and visualization is important to understand, evaluate and present this related research results. Given the broad number of possible applications, it is difficult to find a visualization tool that meets the requirements of a specific application. Only an open source tool can be flexible enough. Most of the current available

tools are not open source, and those which are open source may require a noticeable learning curve giving its wide scope. Because of this, we present a tool that provides the backbone for any type of visualization, that can be easily extended and tailored to allow the creation of custom visualizations, quickly and with minimum effort. Researchers and users alike will benefit for such tool.

The proposed tool, named Olivia, is written in Java, one of the most common languages, which allows high portability and a clear package design. To perform the rendering it uses OpenGL, the most well-known API for 3D graphics. The tool offers out of the box several functionalities such as point cloud rendering, point/cluster selection, OpenGL primitives drawing or 3D stereoscopic view. Building custom visualizations becomes an easy task thanks to the packaging structure and the inherited functionalities of Olivia. Users can extend and modify the tool to fit their needs, giving high flexibility. We want to highlight the utility of the 3D stereoscopic view, not only it is a very engaging way to demonstrate results to the public, but also is very well suited for error identification.

Through this paper we show how to use the tool in different processing scenarios: neighborhood calculation, segmentation, classification, and landing site detection. Using Olivia, the results can be inspected and valuable information can be obtained to validate and refine them. A brief benchmark shows that Olivia does not need high hardware requirements and a regular desktop computer is suitable to visualize tens of millions of points.

In the near future, we pretend to improve the control over the OpenGL primitives that can be drawn overlaying the point cloud, by adding the functionality to perform translations, rotations and animations. We also want to add the capability of using heat maps and colors gradients. Other future work may be to add more visualization modules, adapt Olivia to virtual reality environments or improve the performance by implementing advanced rendering techniques.

REFERENCES

[1] Z. Chen, B. Gao, and B. Devereux, "State-of-the-art: DTM generation using airborne LIDAR data," *Sensors*, vol. 17, no. 1, p. 150, 2017.

[2] J. Ahrens, B. Geveci, C. Law, C. Hansen, and C. Johnson, "ParaView: An end-user tool for large data visualization," in *Visualization Handbook*, vol. 717. Burlington, NJ, USA: Butterworth-Heinemann, 2005.

[3] D. Girardeau-Montaut. *CloudCompare: 3D Point Cloud and Mesh Processing Software*. Accessed: May 25, 2018. [Online]. Available: https://www.danielgm.net/cc/

[4] R. B. Rusu and S. Cousins, "3D is here: Point cloud library (PCL)," in *Proc. IEEE Int. Conf. Robot. Automat. (ICRA)*, Shanghai, China, May 2011, pp. 1–4.

[5] *TIOBE Index for May 2018*. Accessed: May 25, 2018. [Online]. Available: https://www.tiobe.com/tiobe-index/

[6] *The RedMonk Programming Language Rankings: January 2018*. Accessed: May 25, 2018. [Online]. Available: http://redmonk.com/sogrady/2018/03/07/language-rankings-1-18/

[7] *Interactive: The top Programming Languages 2017*. Accessed: May 25, 2018. [Online]. Available: https://spectrum.ieee.org/ns/IEEE_TPL_2017/index/2017/1/1/1/1/1/50/1/50/1/50/1/30/1/30/1/30/1/20/1/20/1/5/1/5/1/20/1/100/

[8] S. Gothel, R. Santina, and P. Koutsolampros. *JOGL: Java Binding for the OpenGL API*. Accessed: May 25, 2018. [Online]. Available: http://jogamp.org/jogl/www/

[9] S. Audet. *JavaCV: Java Interface to OpenCV, FFmpeg, and More*. Accessed: May 25, 2018. [Online]. Available: https://github.com/bytedeco/javacv

[10] G. Bradski, "The open source computer vision library (OpenCV)," *Dr. Dobb's J. Softw. Tools*, 2000. [Online]. Available: https://github.com/opencv/opencv/wiki/CiteOpenCV

[11] F. Bellard *et al. FFmpeg. A Complete, Cross-Platform Solution to Record, Convert and Stream Audio and Video*. Accessed: May 25, 2018. [Online]. Available: https://www.ffmpeg.org/

[12] K. Grouchnikov *et al. Substance Java Look and Feel*. Accessed: May 25, 2018. [Online]. Available: http://insubstantial.github.io/insubstantial/substance/

[13] M. Muja and D. G. Lowe, "FLANN, fast library for approximate nearest neighbors," in *Proc. Int. Conf. Comput. Vis. Theory Appl. (VISAPP)*, vol. 3. INSTICC Press, 2009, pp. 331–340.

[14] F. P. Miller, A. F. Vandome, and J. McBrewster, *Apache Maven*. Orlando, FL, USA: Alpha Press, 2010.

[15] J. Martínez, F. F. Rivera, J. C. Cabaleiro, D. L. Vilariño, T. F. Pena, and D. Miranda, "A rule-based classification from a region-growing segmentation of airborne lidar," in *Proc. SPIE*, vol. 10004, p. 100040F, Oct. 2016.

[16] H. Zhang, J. E. Fritts, and S. A. Goldman, "Image segmentation evaluation: A survey of unsupervised methods," *Comput. Vis. Image Understand.*, vol. 110, no. 2, pp. 260–280, 2008.

[17] T. H. Duong and L. L. Hoberock, "On selecting the best unsupervised evaluation techniques for image segmentation," in *Proc. Int. Conf. Image Process., Comput. Vis., Pattern Recognit. (IPCV)*, 2016, p. 193.

[18] *LAS Specification Version 1.4*, ASPRS, Las Vegas, NV, USA, Nov. 2011.

[19] E. Rozas, F. F. Rivera, J. C. Cabaleiro, T. F. Pena, and D. L. Vilariño, "Comparative study of building footprint estimation methods from LiDAR point clouds," *Proc. SPIE*, vol. 10427, p. 104270R, Oct. 2017.

[20] O. G. Lorenzo, J. Martínez, D. L. Vilariño, T. F. Pena, J. C. Cabaleiro, and F. F. Rivera, "Landing sites detection using LiDAR data on manycore systems," *J. Supercomput.*, vol. 73, no. 1, pp. 557–575, Jan. 2017.

[21] *NetBeans Integrated Development Environment*. Accessed: May 25, 2018. [Online]. Available: https://netbeans.org/

JORGE MARTÍNEZ received the B.Sc. degree in computer science engineering from the University of Santiago de Compostela in 2014 and the M.Sc. degree in high-performance computing from the University of A Coruña in 2016. He is currently pursuing the Ph.D. degree with the University of Santiago de Compostela. His research interests include image and point cloud processing, and parallel computing.

OSCAR G. LORENZO received the B.Sc. degree in computer science engineering, the M.Sc. degree in high-performance computing, and the Ph.D. degree from the University of Santiago de Compostela in 2010, 2012, and 2016, respectively. He is currently a Post-Doctoral Researcher with the Centro Singular de Investigación en Tecnoloxías da Información.

DAVID L. VILARIÑO received the Ph.D. degree from the University of Santiago de Compostela in 2001. He is currently an Associate Professor with the Department of Electronics and Computer Science, University of Santiago de Compostela, and a Senior Researcher with the Centro Singular de Investigación en Tecnoloxías da Información. His research interest is on the design and implementation of image and LiDAR data processing algorithms with fast and efficient computation as main concerns.

TOMÁS F. PENA is currently an Associate Professor and a Senior Researcher with the Centro Singular de Investigación en Tecnoloxías da Información, University of Santiago de Compostela, Spain. His main research lines include high-performance computing, architecture of parallel systems, development of parallel algorithms for clusters and supercomputers, prediction and improvement of the performance of parallel applications, development of applications and middleware for Grid and Cloud, and the use of big data technologies for scientific and NLP applications.

JOSÉ C. CABALEIRO is currently an Associate Professor and a Senior Researcher with the Centro Singular de Investigación en Tecnoloxías da Información, University of Santiago de Compostela, Spain. His research interests include the architecture of parallel systems, the development of parallel algorithms for irregular problems and particularly for processing LiDAR data, and the prediction and improvement of the performance of parallel applications.

FRANCISCO F. RIVERA is currently a Full Professor at the University of Santiago de Compostela, Spain. Throughout his career, he has supervised researches and published extensively in the areas of computer-based applications, parallel processing, and computer architecture. His current research interests include the compilation of irregular codes for parallel and distributed systems, the analysis and prediction of performance on parallel systems and the design of parallel algorithms, memory hierarchy optimizations, GIS, and so on.