Open Source First Person View 3D Point Cloud Visualizer for Large Data Sets

Arnaud Palha, Arnadi Murtiyoso, Jean-Christophe Michelin, Emmanuel Alby and Pierre Grussenmeyer

Abstract The use of laser scanning techniques has become a common way to measure the real world. Millions of points could be generated by this system, which brings forth the problem of visualizing and eventually performing analyses on them. In the open source domain, various software packages exist to visualize 3D point clouds. However, most of these software packages do not allow the real-time rendering of large point clouds. Few of them also allow visualization in an immersive manner. The research aims to create an open source viewer for large 3D point clouds, which enables a dynamic and immersive visualization. In order to do so, rendering and point cloud management strategies must be implemented to avoid overloading the computer's memory. The rendering is done using OpenGL engine by utilizing the graphic card's memory in order to perform faster visualization. The program consists of a pre-processing stage in which the point cloud files are divided into quadtrees and then subsampled using the random tree sampling method. The visualization itself will calculate the distance between the point of view and the center of nodes generated by the pre-processing stage. The amount of points rendered within each node will depend on this distance; the farther away the node is from the point of view, the fewer points are rendered. A loading and unloading function enables the point cloud to be rendered dynamically. With the point cloud

A. Palha · A. Murtiyoso · E. Alby · P. Grussenmeyer (✉)
Photogrammetry and Geomatics Group, ICube Laboratory
UMR 7357, INSA, Strasbourg, France
e-mail: pierre.grussenmeyer@insa-strasbourg.fr

A. Palha
e-mail: arnaud.palha@insa-strasbourg.fr

A. Murtiyoso
e-mail: arnadi.murtiyoso@insa-strasbourg.fr

E. Alby
e-mail: emmanuel.alby@insa-strasbourg.fr

J.-C. Michelin
SNCF Réseau, 6 Avenue François Mitterrand, 93574
Saint-Denis, France
e-mail: jean-christophe.michelin@reseau.sncf.fr

I. Ivan et al. (eds.), *The Rise of Big Spatial Data*, Lecture Notes
in Geoinformation and Cartography, DOI 10.1007/978-3-319-45123-7_3

management algorithm implemented, the resulting program is able to load large point clouds generated by a mobile laser scanner using an ordinary computer. The resulting program as well as the source code will be available for the public due to its open source nature.

Keywords Visualization · Large point cloud · Immersive · Quadtree · Random tree sampling

1 Introduction

Massive 3D point clouds such as those generated by LiDAR are usually difficult to visualize using computers with low configurations. Several open source point cloud processing software such as CloudCompare (http://www.danielgm.net/cc/) and MeshLab (http://meshlab.sourceforge.net/) have improved communication and exchange within the 3D community (laser scanning, photogrammetry, and computer vision). Unfortunately these software packages do not allow the real-time rendering of point clouds and are limited by the size of the point cloud.

This project was initiated from the needs of the French National Railway Company (SNCF) to visualize large data sets of point clouds measured using mobile laser scanners. The objective of this project is therefore to create an open source viewer that loads 3D pre-divided node files. This loading will be done on-the-fly and concerns only the visible points. In order not to overload the RAM, real-time sub-sampling algorithms were implemented which allows the computer to load the point cloud in a much reduced size while still maintaining its topographic information for measurement purposes.

The design of the viewer will largely take inspiration from video games, mainly the first person genre as to give a more immersive environment. The open source nature of the work is essential since today only commercial solutions of the same type are available on the market. An example of such commercial solution is the Australian software EuclideonGeoverse (http://www.euclideon.com/products/geoverse/), which offers similar features.

Immersive visualization is a type of visualization where the camera shows a head-track and stereoscopic view of the surrounding (Kreylos et al. 2008). It has several advantages, such as a direct interaction between the data and the user, increased quality control capability and accuracy due to the ease of use as well as a potential for a better analysis on the 3D data.

The viewer uses the principle of the free-fly camera, commonly used in first person video games. The principle of this camera is to emulate the user's eyes' field of view, hence its name. The user controls the movement of the camera through the keyboard and mouse. The first person perspective is often used in shooter video games, as it gives the player an experience of directly being in the scene. This is also exactly the reason why the research draws inspiration from the first person video game genre. The visualization of the point cloud in this research is meant to

be immersive. The first person system is therefore very well suited to give the user a closer and more interactive view of the point cloud as well as to render the viewer more immersive.

As regards to point cloud management, the approach to be taken in this research will be based on a distance-based sub-sampling of the raw data. The idea is to divide the massive point cloud into smaller quadtree nodes (rectangles with a part of the point cloud inside them). The program will then measure the distance between the user (the camera) and the nodes' centers. The closest nodes will be visualized in full resolution, while farther ones will be sub-sampled. This way, the time to visualize the point cloud will be greatly reduced, as the computer will no longer need to load the whole point cloud at the same time.

2 State of the Art

Various algorithms have been used in several published works. The majority of these publications concern the Level of Detail (LoD) data structure (division of the large point cloud into smaller entities) and the memory management problem with respect to the rendering issue. Both Richter and Döllner (2014) and van Oosterom et al. (2015) noted that in today's existing visualization systems, the most referred data structures used are the quadtree and the octree.

Kovac and Zalik (2010) and Zhang and Ma (2012) prefer to use the quadtree division to visualize their terrestrial data sets. Goswami et al. (2013) and de la Calle et al. (2012) used the kd-tree approach, while Kreylos et al. (2008) and more recently Elseberg et al. (2013) proposed the use of the octree subdivision method which is a derivative of the kd-tree.

Another approach proposed by Meng and Zha (2004) and Klein et al. (2004) involves the random tree sampling method (Fig. 1). In this method, the point structure within the point cloud file is randomized, and then divided into files of equal number of points. The appropriate reduced files can be called whenever necessary, for example only one reduced file is called for rendering during camera

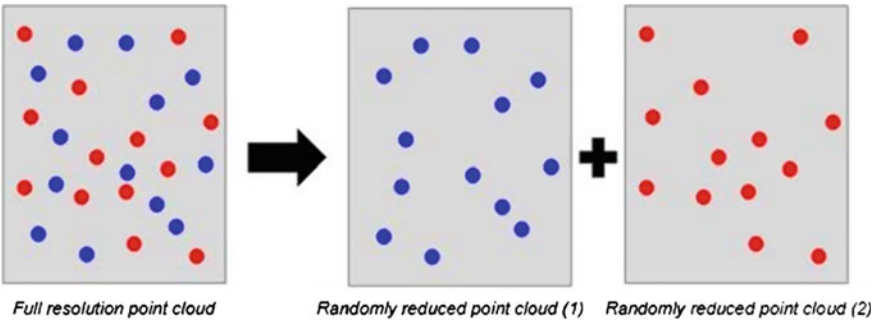

Fig. 1 Illustration of the random sampling subdivision of a full resolution point cloud

movement. A rendering of all the reduced files at the same time gives a full resolution of the point cloud.

Richter and Döllner (2014) concluded that the octree approach is more suitable for 3D point clouds with a varying resolution, distribution and density. Quadtrees, however, are more commonly used for datasets with a 2.5 dimensional characteristic. Therefore in this research the octree approach will not be too suitable since the dataset is mainly in the form of elevation models. The quadtree will be used to further divide the data in the first place into smaller nodes, and then followed by an implementation of the random tree sampling on the quadtree nodes to address the multi LoD facet of the problem.

The principal rendering engine used in off-line visualization of 3D data is OpenGL. Johansson et al. (2015) used this engine to render an interactive BIM (Building Information Model) in real time. The authors also noted the requirement to migrate from older versions of OpenGL to newer ones in order to use the VBO (Vertex Buffer Objects) to render the objects faster by using the graphic card's memory. The older OpenGL enables the loading of data directly from the CPU's memory (RAM). This is a very straight-forward method which is easy to implement, however the loading takes much more time. The VBO approach is therefore more recommended (Wand et al. 2008; Kovac and Zalik 2010; Johansson et al. 2015). Despite this, de la Calle et al. (2012) warn about the limitations of OpenGL. In their tests, it is concluded that depending on the hardware used, OpenGL can handle only up to a few million points at the same time.

Another possibility is to use the WebGL environment which is a JavaScript written engine based on the OpenGL ES 2.0. It also has the capabilities to directly access the graphic card's memory (Callieri et al. 2015). One notable example of the implementation of WebGL is the online visualizer Potree (Liu and Boehm 2015).

The web-based approach is an interesting choice which enables the data to be diffused for a large community. However, in this research the main data to be tested are point clouds measured for the internal purposes of the SNCF. The off-line approach by using OpenGL will therefore be primarily used in order to create an executable application which may be opened by computers without the aid of the internet. This however, does not change the open-source nature of the resulting program and its codes.

3 Development of the Viewer

The principal tool used in this research is the OpenGL engine. OpenGL is the premier environment for developing portable, interactive 2D and 3D graphics applications. It is an environment which is suited for the rendering and visualization of 3D data. The programming language to be used in interaction with this engine is the C++. The choice of programming language is based on its open source nature, but also on its robustness, maneuverability and existing libraries and functions.

Other dependencies used include the SDL and liblas libraries for C++. SDL is a set of C functions which provides simple interaction between the user and the computer via the keyboard and mouse. It is widely used in the development of video games, but SDL is limited to 2D environments. By coupling SDL's capabilities for human-machine interaction and OpenGL's 3D rendering, we can create an interactive and immersive platform. This model has been used in many implementations of OpenGL in video games.

The library liblas is a set of C/C++ functions which translates the common LAS format of laser scanning products into the program. This will then enable us to analyze and perform manipulations on the data such as sub-sampling and quadtree division as well as visualization via OpenGL.

3.1 Rendering Methods

In rendering the scene of a 3D object in OpenGL, two approaches are available. Older versions of OpenGL use a traditional approach of loading the data into the computer's RAM, and then reading the data from this internal memory. It is one of the simplest ways to render in OpenGL. However, this method has a disadvantage in that the RAM can become easily overloaded with large amounts of data. This is because this approach actually takes the longer path to render an object in the computer screen. The data will have to be loaded first into the computer's memory and then passed onto the graphics card to be rendered.

Starting from OpenGL 3.0, a new protocol for scene rendering was implemented. The use of VBO is recommended. The VBO bypasses the CPU memory by loading the object directly into the graphic card's memory (Video RAM or VRAM). The VRAM is admittedly lower than the normal RAM; however it is very efficient since the loading time can be cut by dismissing the loading of files in the RAM first before rendering them in the screen (Fig. 2).

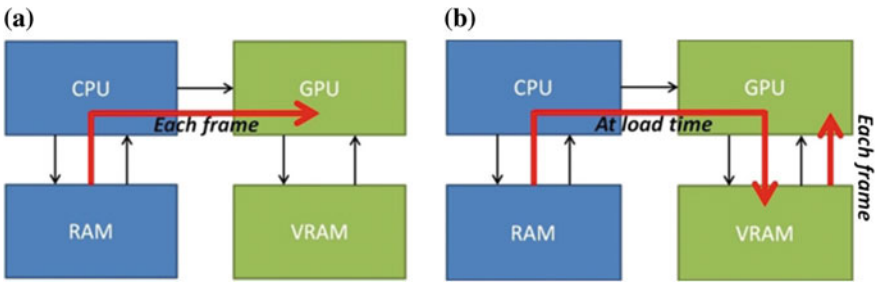

Fig. 2 The two different approaches in rendering a scene in OpenGL (modified from Yaldex 2015): **a** the regular and older RAM-based method loads the data from the RAM and then renders each frame in the same manner, **b** the VBO approach loads the data from the RAM only once and passed it to the VRAM

3.2 Data Pre-processing

The first part of the program is the pre-processing of the raw data. In this part, the point cloud will be subdivided into quadtree nodes. The resulting nodes will then be subsampled randomly to create several point clouds with limited number of points.

The initial input parameters are written by the user in a parametrisable text file. This file includes the path for the results of the pre-processing (quadtree nodes with their randomly sub-sampled files), the limit of points requested for each sub-sampled file, and the path for the raw LAS point cloud (Fig. 3).

This module will retrieve information on the quadtree node (level n) file's limits (X_{min}, X_{max}, Y_{min}, Y_{max}) using the liblas library from the files' header. Using this information, Algorithm (a) calculates new limits for the nodes by simply dividing the intervals of X and Y respectively by two. In this case of quadtree sampling, the Z axis is not taken into account, due to the data's characteristics which lean more towards a 2.5D data (i.e. a plane representation of the terrain with lack of 3D details) rather than a full 3D model. Taking into account the Z axis would have changed the quadtree into an octree. Each new quadtree node generated will be identified by an alphabet identifier.

Afterwards, another function is called to shuffle the order of points inside the quadtree files. From this shuffled file, smaller files are created by simply truncating the quadtree file into the desired number of points. In Algorithm (b), NumFilePoints is the number of points in the node, while NumTreePoints is the threshold number of points for each random-sampling. The subsampled files will be identified by their parent quadtree node and an Arabic number indicating their order of creation.

Metadata are stocked in an indexing ASCII file as to give quick access to these data. This indexing file contains a list of the quadtree nodes and their supporting information such as the 2D center, the number of files for each node as well as the path to the required file. The radius between the user's point of view and the nodes are computed with the data in this index file.

(a)

```
Data: Quadtree node (level n), X_min, X_max, Y_min, Y_max
Result: Quadtree nodes (level n + 1) (A.las, B.las, C.las,D.las)
initialization;
load Quadtree node (level n) (.las);
while ReadNextPoint() do
    if X < (( X_max-X_min)/2)+X_min then
        if Y < (( Y_max-Y_min)/2)+Y_min then
            write point into A.las;
        else
            write point into B.las;
        end
    else if X > (( X_max-X_min)/2)+X_min then
        if Y < (( Y_max-Y_min)/2)+Y_min then
            write point into C.las;
        else
            write point into D.las;
        end
end
```

(b)

```
Data: Quadtree node (A.las), NumFilePoints, NumTreePoints
Result: Random-sampled files (A_R_0.las, A_R_1.las, etc.)
initialization;
load A.las;
random_shuffle;
for int i=0;i<=NumFilePoints/NumTreePoints;i++ do
    int j=0;
    while j<NumTreePoints and ReadNextPoint() do
        write point into A_R_[i].las;
        j++;
    end
end
```

Fig. 3 General algorithms for **a** quadtree subdivision and **b** random tree sampling

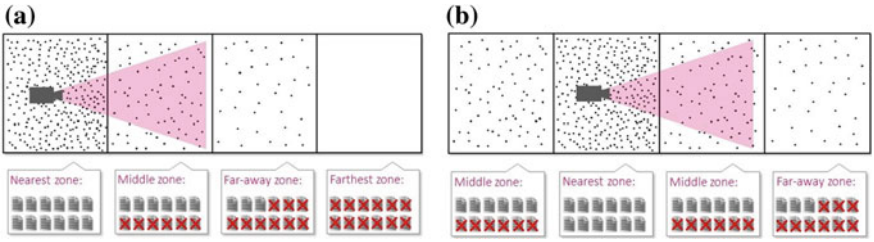

Fig. 4 The point cloud visualization zones based on the number of random-sampled files loaded. In **a** the camera is located at the *left*-most node, and the number of files loaded depends on other nodes' distance to the camera. In **b** as the camera moves to the *right*, the zoning changes and the distances between nodes are recalculated accordingly

3.3 Visualization Strategies

In order to be able to load more points, a point cloud visualization strategy has to be implemented. Loading all points at the same time would not be effective, even by using VRAM. Indeed, it must be noted that even though it is more effective than ordinary RAM, the VRAM usually has lower memory. Such strategy is often referred to as the out-of-core rendering method (Meng and Zha 2004; Kovac and Zalik 2010; Richter and Döllner 2014). A conceptual drawing of the distance-based visualization strategy is shown in Fig. 4.

The algorithm divides the space into four categories. The 2D distance between the camera and the center of each node is computed for all the nodes created by the pre-processing. The first distance category is called the "nearest" zone in which the highest density of points is loaded. This means that all the random-sampled files produced from the pre-processing stage for nodes within the set radius will be loaded. Farther away is called the "middle" zone. Already in this zone a reduced number of files are loaded. The "far-away" zone loads the least amount of files. The last zone is called the "farthest" zone in which no points are loaded. The radius as well as the number of files to be loaded for each zone can be modified by the users according to their needs. This algorithm is computed dynamically for each frame. This means that when the camera moves forward, the distance between the camera and each node is recalculated and the zones are re-attributed accordingly (see Fig. 4).

4 Results and Discussions

The first analyzable results are those from the pre-processing stage, which involves two simultaneous methods. These results are then used as input for the visualizing program with the help of a text file (called the project file), which describes the available point clouds to load. This descriptive file is generated automatically at the end of the pre-processing phase.

Tests were performed on a first data set of a sample of 16 million points (around 500 MB) and another second much larger main data set with over 300 million points (around 10 GB). Both data were obtained from a mobile laser scanning system. The computer used for the tests is an ordinary laptop with 2.7 GHz Inter Core i7 processor, 4 GB of RAM with 1333 MHz frequency, 384 MB integrated VRAM Intel HD Graphics 3000 (650 MHz), and a hard drive with 5400 RPM.

4.1 Pre-processing Results

The pre-processing generates quadtree nodes for all input files and then subsamples these nodes into several more sparse files. This is the longest stage of processing, since the visualization will only use the results of this pre-processing. It should be noted that the pre-processing takes less than a minute for the first data set (about 500 MB) with the computer specifications used, while the second data set (about 10 GB) takes around 40 min to finish.

Figure 5 illustrates the division of a point cloud file into 4 quadtree nodes. The process is straightforward and the results are then used for a further sampling using the random-tree method. Figure 6 shows the results of this sampling process. The quadtree node is divided into several smaller files having a limited number of points (the limit is determined by the user). The idea is to load only a certain number of files depending on the distance between the camera and the node concerned. In Fig. 6, the point cloud in (a) is the result of the loading of a majority of the smaller files at the same time, which is equal to nearly full-resolution. Point cloud (b) loads only half of the amount of files generated by the pre-processing, while (c) loads only one subsampled file. For these tests, each data was divided into quadtree nodes of 50 $\times$ 50 meters and then random sampled into files of 300,000

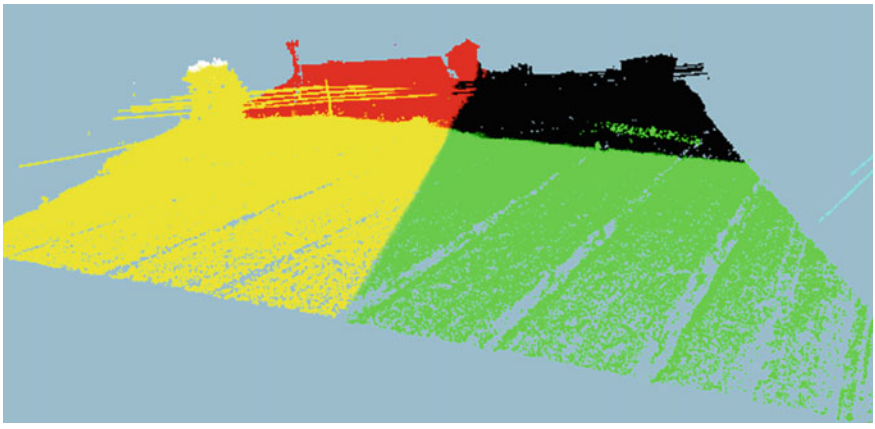

Fig. 5 Results of the quadtree sampling with each node in different *colors* to showcase the division

Fig. 6 Results of the random tree sampling. The point cloud in **a** with the highest density shall be loaded for the nearest node, while **b** is less dense and will be loaded for the middle-distanced node. The least dense **c** will be loaded for the far-away nodes. Farthest nodes will not show any points

points. This configuration was chosen because it offers a balance between point density and performance for this particular test computer. These values were empirically determined.

4.2 Visualization and Performance Analysis

The program manages to handle both data sets, albeit several issues are still present. Some screenshots of the loaded point cloud inside the program are shown by the Fig. 7. In this figure, two perspectives of the point cloud are shown. The first involves a bird's eye overall view, which is attained simply by moving the camera to the zenith of the point cloud. Note that this is not the default point of view of the viewer. The second one involves the first person view, in which the user can move around using the keyboard as well as rotate the camera using the mouse.

The visualizing module of the program was first used to test the first data set. No rendering problem was encountered in this case, and the algorithm manages to load and unload the defined subsampled files dynamically. A performance graph of this test can be seen in Fig. 8 where the abscissa denotes the observation of scenes or steps rendered and the ordinate the time required for the rendering. Despite some

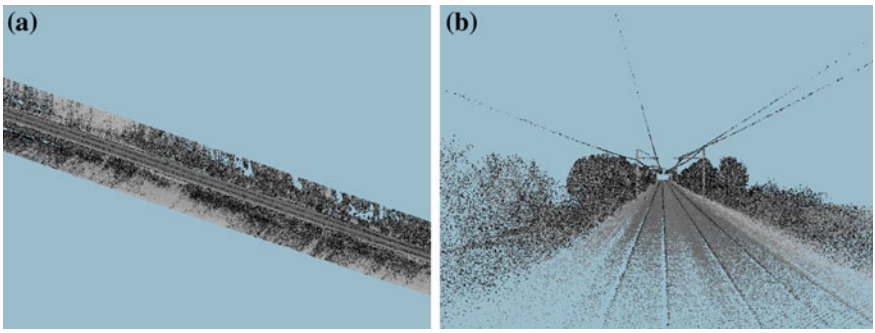

Fig. 7 Visualization results from the second data set in the program. In **a** a small part of the 300 million points is shown from a bird's eye view, while **b** shows a first person perspective of the data

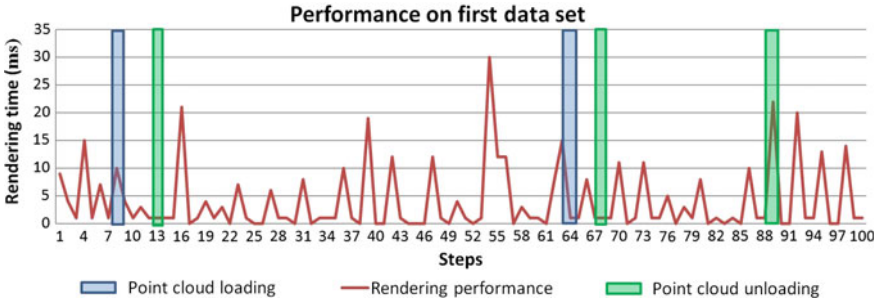

Fig. 8 Performance graph for the first data set with events marked by *colored bars*. Six subsampled files were loaded for the nearest, two for medium, and one for far-away zones

minor spikes in the graph, the rendering time is rather stable. No other systematic spikes can be observed except during loading and unloading of files, in which the rendering time takes in average about 10–20 ms longer than normal rendering. However, one larger spike amounting to 30 ms can be observed around the 54th step. At this state we are unable to detect the cause of this solitary spike, whether it is accidental or systematic.

The second (main) data set, tested with the same configuration as the previous one, shows large anomalies with the rendering time. This significantly slows down the performance of the program. The low specifications of the computer are presumed to be unable to handle such amount of data, considering that the VRAM is very limited and even the RAM is low. However, this condition enables us to perform some performance analysis based on the amount of points loaded. Another test is therefore conducted with a reduced number of points to determine the correlation between the implementation of our algorithms and the large spikes in rendering time.

In this analysis, rather than using the same number of files to be loaded for each distance category as in the previous case, a reduced number is used. Different scenarios with this loading configuration were tested, in which the rendering time is taken into account. In Fig. 9 two scenarios are observed in which the rendering time is analyzed as in the case of the first data set.

Similar with the previous case, Fig. 9a shows a spike of around 10–20 ms during loading and unloading of new data. However, a more significant spike of 40 ms can be observed after the second loading event, in which the camera moves around in this environment. The spike receded when the program unloads some files and reloads other files to render the scene following the movement of the camera. A preliminary hypothesis is made where the loading and unloading is not related to the unsystematic spikes.

In Fig. 9b, the amount of files is greatly increased even though it is still below the number of files loaded in the case with the first data set. With the test computer specifications, some large spikes are observed in the graph amounting up to 5 s. It is quite evident from this graph that the loading/unloading events do not necessarily correspond with the spikes. It should be noted that each loading/unloading event

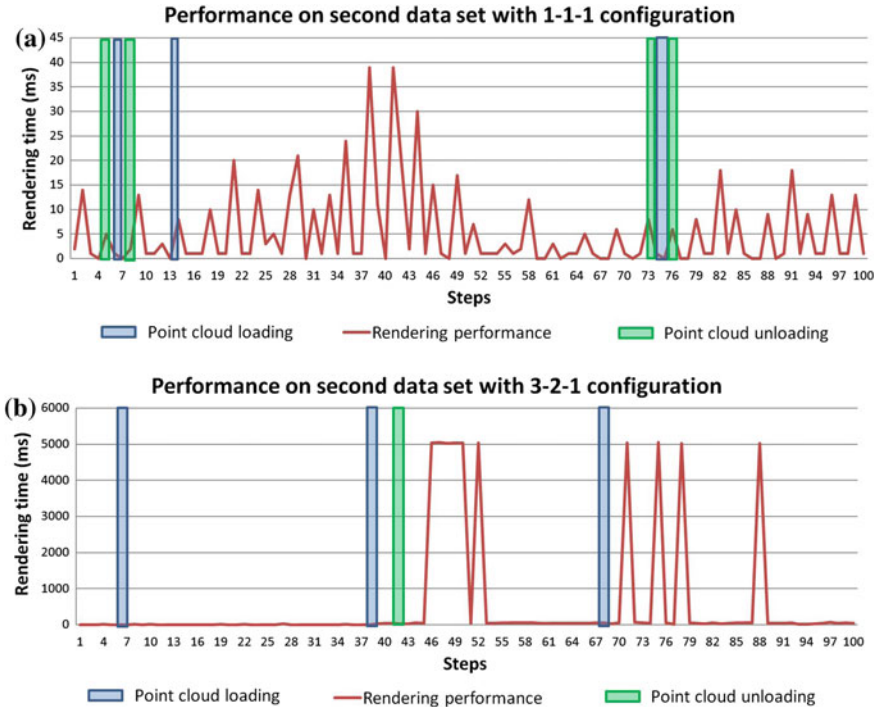

Fig. 9 Performance graph for the second data set. In **a** only one subsampled file is loaded for each zone, whereas in **b** three were loaded for the nearest, two for the middle, and one for the far-away zone

does not necessarily concern the same amount of points. There are no significant spikes observed after the first loading event, and the spikes occurring from steps 70 up to 90 do not reflect a systematic behavior in regards to the loading event which occurred just before them. The time used in the loading and unloading events themselves is still around 10–20 ms, even though the graph's scale cannot show this fact.

These anomalies are therefore not linked to the rendering time during the loading/unloading events themselves, which are more systematic in nature (of around 10–20 ms during each event). It is most probably due to the differing number of points rendered in each scene which depends on the camera's position. In this case the limitation of the test computer's VRAM is very much felt, since only 384 MB are allocated for the graphics card. It should also be noted that the VRAM is integrated with the RAM, which further reduces performance. Further benchmarking tests with other computer specifications should be performed to verify this analysis.

5 Conclusions and Future Works

The method developed to visualize large point clouds and large spatial data in this research has the advantage of being able to adapt the data to the computer's characteristics. With this program, a point cloud can be inspected using an ordinary computer. The density can be lowered if the device is not powerful enough to process massive 3D data. With this adaptive characteristic the viewer could improve 3D point cloud data sharing and collaborative work. The program has also turned out to be able to handle large point clouds, up to 300 million points by implementing the algorithm using an ordinary laptop.

The tests and the performance analysis show that the rendering time is largely independent from the loading and unloading algorithm. This is shown by the systematic rendering time of the loading/unloading events which is almost always around 10–20 ms. However, the anomalies are more likely due to the graphics card which is not designed to process massive 3D data. Other benchmarking tests using stronger computer specifications should be performed to assess this analysis. A beta testing of the program is also planned in order to obtain an objective perspective on the performance of this program.

The program currently only works with LAS format datasets, therefore the compatibility with other standard 3D formats should be added in the future to create a more accessible program for the 3D community. Also, the software was designed to process data which are mostly horizontal with few variations in elevation (2.5D structure). The pre-processing algorithm could be further improved by using octree rather than quadtree sampling to optimize the 3D visualization for point clouds with large variations of elevation.

Even though for now the program simply pre-processes the data and visualizes them, its open source nature makes it a good base to develop professional applications. Since the program keeps the original information from the raw data, tools to extract geometric information from the point cloud can be added eventually (distance measurements, coordinate inquiry, etc.).

Acknowledgments This research has been conducted during a Master research project at INSA Strasbourg, France (Graduate School of Science and Technology) in collaboration with SNCF Réseau (French National Railway Company). The LiDAR data used in the tests were furnished by SNCF Réseau.

References

Callieri M, Dellepiane M, Scopigno R (2015) Remote visualization and navigation of 3D models of archeological sites. ISPRS Int Arch Photogramm Remote Sens Spat Inf Sci XL-5/W4:147–154

de la Calle M, Gómez-Deck D, Koehler O, Pulido F (2012) Point cloud visualization in an open source 3D Globe. ISPRS Int Arch Photogramm Remote Sens Spat Inf Sci XXXVIII(5):135–140

Elseberg J, Borrmann D, Nüchter A (2013) One billion points in the cloud—an octree for efficient processing of 3D laser scans. ISPRS J Photogramm Remote Sens 76:76–88

Goswami P, Erol F, Mukhi R, Pajarola R, Gobbetti E (2013) An effcient multi-resolution framework for high quality interactive rendering of massive point clouds using multi-way kd-trees. User Model User Adapt Interact 29(1):69–83

Johansson M, Roupé M, Bosch-Sijtsema P (2015) Real-time visualization of building information models (BIM). Autom Constr 54:69–82

Klein J, Krokowski J, Wand M (2004) The randomized sample tree: a data structure for interactive walk-throughs in externally stored virtual environments. Presence Teleoper Virtual Environ 13 (6):617–637

Kovac B, Zalik B (2010) Visualization of LiDAR datasets using point-based rendering technique. Comput Geosci 36(11):1443–1450

Kreylos O, Bawden G, Kellogg L (2008) Immersive visualization and analysis of LiDAR data. Adv Vis Comput 846–55

Liu K, Boehm J (2015) Classification of big point cloud data using cloud computing. ISPRS Int Arch Photogramm Remote Sens Spat Inf Sci XL-3/W3:553–557

Meng F, Zha H (2004) An easy viewer for out-of-core visualization of huge point-sampled models. In: Proceedings 2nd international symposium on 3D data processing, visualization and transmission, Thessaloniki, 2004. IEEE

Richter R, Döllner J (2014) Concepts and techniques for integration, analysis and visualization of massive 3D point clouds. Comput Environ Urban Syst 45:114–124

van Oosterom P, Martinez-Rubi O, Ivanova M, Horhammer M, Geringer D, Ravada S, Tijssen T, Kodde M, Gonçalves R (2015) Massive point cloud data management: design, implementation and execution of a point cloud benchmark. Comput Graph 49:92–125

Wand M, Berner A, Bokeloh M, Jenke P, Fleck A, Hoffmann M, Maier B, Staneker D, Schilling A, Seidel HP (2008) Processing and interactive editing of huge point clouds from 3D scanners. Comput Graph 32:204–220

Yaldex (2015) Algorithms in game programming. http://www.yaldex.com/game-programming/0131020099_app02lev1sec10.html. Accessed 17 Dec 2015

Zhang H, Ma J (2012) The organization and visualization of point cloud data based on the octree. In: Zhang Y (ed) Future wireless networks and information systems. Springer, Berlin, pp 425–432