Advance Access publication on 20 December 2022 https://doi.org/10.1093/comjnl/bxac179

Interactive Visualization of Large Point Clouds Using an Autotuning Multiresolution Out-Of-Core Strategy

Diego Teijeiro*, Margarita Amor, Ramón Doallo and David Deibe

Computer Arquitecture Group, CITIC, Universidad de A Coruña, Campus de Elviña s/n, 15071 A Coruña, España

**Corresponding author: diego.teijeiro@udc.es*

Due to the increasingly large amount of data acquired into point clouds, from LiDAR (Light Detection and Ranging) sensors and 2D/3D sensors, massive point clouds processing has become a topic with high interest for several fields. Current client-server applications usually use multiresolution out-of-core proposals; nevertheless, the construction of the data structures required is very time-consuming. Furthermore, these multiresolution approaches present problems regarding point density changes between different levels of detail and artifacts due to the rendering of elements entering and leaving the field of view. We present an autotuning multiresolution out-of-core strategy to avoid these problems. Other objectives are reducing loading times while maintaining low memory requirements, high visualization quality and achieving interactive visualization of massive point clouds. This strategy identifies certain parameters, called performance parameters, and defines a set of premises to obtain the goals mentioned above. The optimal parameter values depend on the number of points per cell in the multiresolution structure. We test our proposal in our web-based visualization software designed to work with the structures and storage format used and display massive point clouds achieving interactive visualization of point clouds with more than 27 billion points.

Keywords: LiDAR; web-visualization; efficient data structures; multiresolution; out-of-core strategy

Received 29 October 2022; Revised 5 October 2022; Editorial Decision 28 October 2022
Handling editor: Suchendra Bhandarkar

1. INTRODUCTION

Light Detection and Ranging (LiDAR) technology provides extremely useful high-resolution data in the form of point clouds that can be applied in a wide range of fields, including geology, agriculture, forestry, urban planning and infrastructure maintenance. The massive amount of spatial information that may be acquired by LiDAR technology entails an enormous challenge when developing applications focused on handling such amounts of data, such as DTM generation [1], land coverage classification [2] or semantic-based classification [3]. Real-time interaction, memory requirements and loading times involved in handling massive LiDAR point clouds are some of the susceptible problems to be addressed.

Multiresolution techniques are widely used in these situations. They are based on the use of different levels of detail (LODs) or resolutions to represent and handle the data at different scales, maintaining good performance on every LOD. Hierarchical data structures that allow processing and visualization of massive point clouds are usually used, such as k-tree, quadtrees and octrees. These structures allow the reduction of the data that need to be managed employing multiresolution techniques; moreover, they allow out-of-core or external memories strategies to load and unload data if they exceed available memory [4, 5].

Multiresolution techniques are used for large volumetric data handling, from storage to visualization and, in some cases, processing or modification. These volumetric data are usually stored using voxels, which is the basic building block of the multiresolution techniques [6, 7], with special focus on efficient ray-casting on the volumes [8, 9]. A review of several techniques and structures used for volume visualization can be read in [10], focused on the use of GPUs. One of the objectives is to allow the visualization and the processing of point clouds in several platforms [11–13]. In addition, when the data size

exceeds available resources, out-of-core or external memory algorithms are used [13–16]. With respect to massive point clouds, in some cases, the point clouds are voxelized [17], allowing the application of the techniques explained above, or simply as an intermediate step for building other structures [18].

In contrast with other volumetric data, massive point clouds are irregular in nature and the previous techniques are not always adequate or efficient. Multiresolution approaches tailored for point clouds are still widely used, from earlier approaches such as *Surfels* [19] and *QSplat* [20] to more recent ones. A multiresolution technique that groups the points in chunks, where each chunk contains points that are added to the contents of its parent, was proposed in [21]. Chunk-based techniques became the standard for large-scale point clouds rendering due to their high efficiency in GPU-based rendering. Further works improved upon them or introduced alternatives to parts of these techniques. A nested *octree* system is introduced in [22]. This was refined in [23] with the introduction of a Poisson-disk distribution to perform the same sampling process.

Some works take advantage of the spatial structure to store LODs efficiently [24], propose a multiresolution out-of-core system with no data redundancy supported in a layered cell hierarchy [15], use a multiresolution kd-tree [13] and use an octree. Reference [25] exploit the order of the points in storage to obtain LODs, in such a way that loading more points increases the detail, filling the gaps among previously loaded points. Reference [14] use geo-morphing and interpolation to create the data for each resolution, while reference [26] generate the LODs information in real-time from the octree structure that stores the points, but this information is not stored in disk explicitly [23].

The biggest drawback of these proposals is that they group together and perform the rendering of points in hierarchically organized *chunks*. The different extension and point density of the *chunks* produces sudden changes between two adjacent levels that are visualized together. Additionally, they can cause artifacts when the rendering of incoming and outgoing *chunks* is performed. In order to avoid these artifacts, some proposals [25, 27] exploit the order of the points, either in storage or in memory after loading, to create a point density gradient in such a way that points later in the order are located among previously loaded points, progressively increasing the density. These approaches cannot facilitate direct access to a specific Level-of-Detail on demand as it is not clear what point density is achieved at a certain amount of points without loading them.

We propose a new out-of-core multiresolution autotuning strategy aimed at reducing the structure building time as well as achieving a reduction of loading times while maintaining low-memory requirements. The tuning strategy, named TVPC (Tuning Visualization of Point Clouds), is based on the automatic computation of two performance parameters, in accordance with a set of premises, which have a direct influence on the performance: the number of points per cell and the number of LODs for the multiresolution structure. Our strategy takes into account the characteristics of the point cloud, such as point density and area covered as well as the size of the point budget, which takes into consideration the characteristics of the visualization system.

To evaluate our proposal, we use a web-based visualization software, Pebbles, which will be presented in Section 2. The client-server arquitecture introduces new constraints that have been overcome through many different approaches in several web-based visualization platforms such as: Dielmo3D [28], Potree [29], Cesium [30], GVLiDAR [31], ViLMA [24] or more broad tools such as PCL [32, 33]. Each one of the platforms uses their own file formats, data structures and multiresolution methods. Pebbles shows high performance, achieving real-time interaction, handling around 27 billion points with a high level of interaction.

The rest of the paper is organized as follows: Section 2 presents our efficient strategy for the processing of large point cloud based on a tuned multiresolution structure; in Section 3, we evaluate our proposals using *Pebbles*; finally, Section 4 presents the main conclusions and future work.

2. STRATEGY OVERVIEW FOR TUNING VISUALIZATION OF POINT CLOUDS

This section explains the strategy for efficiently tuning visualization of large point clouds. The interactive visualization of massive point clouds, exceeding available memory capacity, demands the use of multiresolution out-of-core techniques. Our proposal is focused on improving the interactivity and requirements of memory. We consider a high interactivity performance based on two measurements: low loading time, time when the whole dataset is obtained to the desired LOD, and high frames per second, to measure the fluidness of the interaction.

An overview of the main stages of the TVPC (Tuning Visualization of Point Clouds) strategy is represented in Fig. 1. The *tuning* stage processes the source point cloud and computes the best parameters tailored to the dataset. Those parameters and the point data are fed to the *structure building* stage, creates the multiresolution data structure tuned for that dataset, storing the point data compressed in disk ready for the *application* stage. In the first stage (*Tuning stage* in the figure), we need to determine the main parameters which influence performance for the visualization of large point clouds and establish a set of premises. Based on these premises, a number of tuneable parameters is obtained and, for each case, the optimal values are chosen. In the second phase, the tuned multiresolution structure is built with the suitable tuneable parameters obtained in the previous stage. Furthermore, the resulting LiDAR data are also compressed in order to reduce the amount of information to be stored and potentially transferred through the network. The

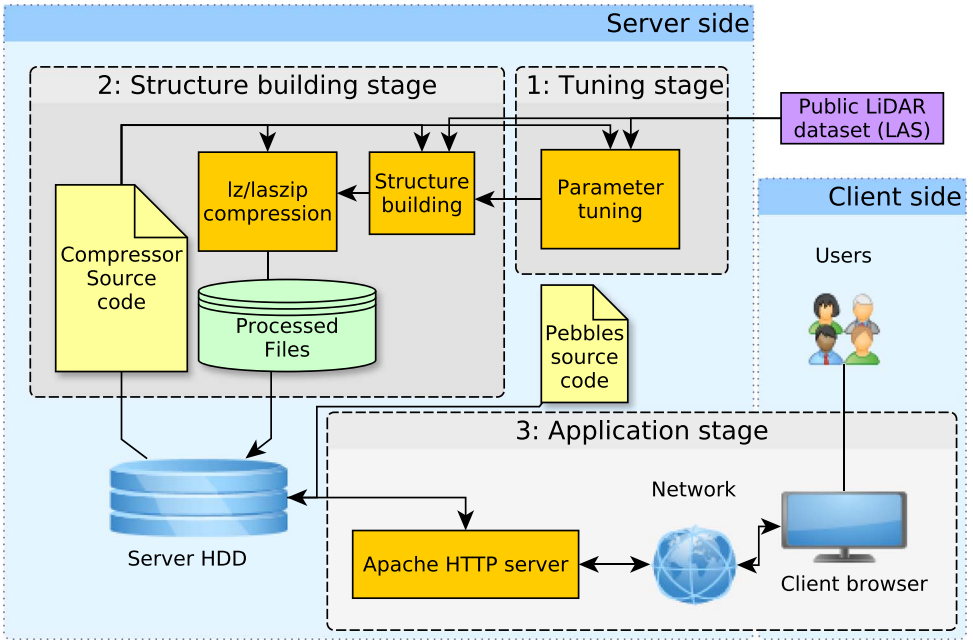


FIGURE 1. General overview of the architecture of our TVPC strategy.

resulting structure is stored in disk ready to be used by other applications, referenced as *application stage*. For this paper, we use our own web-based visualization tool.

The application used, *Pebbles*, follows a conventional client-server structure, such as most web applications. The application is capable of rendering massive point clouds employing the screen area used, as well as a configurable point budget to determine the level of detail to use for different areas of the point cloud currently being rendered. In addition to point cloud visualization, it includes the same functionalities as *GVLiDAR* [31]. Specifically, Pebbles is written in HTML5, JavaScript and WebGL as graphics API so it can be executed in any modern browser. The application files and dataset data are served through a standard Apache HTTP Server. The clients interact with *Pebbles*, which requests the required files from the data structure stored in the server.

2.1. Tuning stage: automatic parameter estimation

Our proposal allows the design of a multiresolution data structure with little effort while obtaining competitive performance with respect to hand-tuned approaches. This strategy is based on two phases: resource analysis and parameter tuning. In the first phase, a set of performance parameters are obtained together with some premises that allow high interactivity and low-memory requirements to be achieved. The second phase assigns the suitable values for the performance parameters obtained in the first phase. This helps to reduce the time needed to process a point cloud, as otherwise some time would be wasted finding the correct values for the parameters. Commonly used techniques, such as trial-and-error, can take a long time with large point clouds since the multiresolution structure construction process can take hours to complete.

Our strategy uses a precomputed multiresolution structure to store the data for the *application* stage. Our proposal can be adapted to several hierarchical data structures and in our case, due to the 2.5D nature of the point clouds from aerial LiDAR, a quadtree structure fits adequately. All of the points in the raw point cloud are stored in the leaf nodes of the tree, according to their location. The internal nodes of the tree contain copies of those points, sampled to have a suitable point density for the level of detail determined by the depth of the node in the tree. This redundancy is introduced to help reduce loading time, as explained in more detail in Results section. Each node (both leaf and internal) stores the points in disk in an independent file, called a *cell*. Our proposal is focused on the reduction of the sudden appearance of sets of points in a client-server tool for visualization. Moving the camera does not cause the sudden appearance of points on a previously empty area, but new data transferred will fill in the gaps between currently rendered points. The visualization application starts loading the tree at the root, and as long as it maintains the higher nodes in cache, there are always points ready to be rendered in every part of the dataset.

In our quadtree structure, the number of LODs (*NL*) determines the area covered by each leaf cell; meanwhile, the sampling rate (*SR*) sets up the number of points in the internal cells. Each internal cell contains a subset of points from its children cells. Our quadtree structure is focused on a client-server system with the objective of reducing loading time,

which is determined by the cell size in bytes (*CS*). Therefore, our proposal calculates SR and NL, directly dependent on the desired size of cells according to the following expressions:

$$\log_4 NL = \frac{\delta \times A}{CS/PS} \tag{1}$$

$$SR = \sqrt{CS/PS}, \tag{2}$$

where A is the covered area of the whole dataset, δ is the point density (in points per area unit) and PS is the size in bytes of each point.

Our TVPC strategy also takes into account the performance characteristics of the client system. Even the most modern powerful computers would be unable to load and handle massive point clouds in their entirety. Multiresolution data structures provide the means to access a simpler version of the data (lower detail) that can be loaded by the computers, but a way of guiding which LOD is the correct one is needed. One of the simplest ways of controlling the LOD to use is setting a limit on the amount of points being rendered. The application will load points increasing the detail until the limit is reached. This limit is called a point budget (PB) and should be set according to the performance of the client computer to maximize the detail used without exceeding the capabilities of the hardware. In this paper, we focus on high-performance systems and the limits of usable point budgets are tested.

We focus on finding the best *CS* value, as it has a big impact on the used metrics: loading time and memory requirements. Based on the effects that the value of *CS* has on the performance, some premises to guide the decision of the *CS* value to use can be defined:

- Premise 1: *CS* shoud be adjusted for the PB. High values for *CS* paired with low point budgets result in very few cells being able to fit into the point budget, limiting the cells that can be managed by the application at the same time and negatively impacting the image quality due to under-utilizing the available point budget. Replacing one cell with its children would exceed the point budget so a relevant part of it is not used, with the consequent lower detail being displayed. In our TVPC proposal, the size of *CS* must be small enough to allows several cells to fit inside the PB

$$CS << PB. \tag{3}$$

- Premise 2: Use high *CS* that minimizes *NL*. High *CS* values make it possible to reduce *NL*, which reduces the amount of cells consequently the cost in storage space and the amount of cells necessary to visualize. Higher values for *CS*, without causing a reduction in *NL*, create internal cells with more points that can fill the point budget using fewer cells, improving the loading times at the cost of storage space. Comparing two different values for *CS*, CS_i and CS_j, the higher value CS_i is better if the following relation is met:

$$Ncell_i \times CS_i < Ncell_j \times CS_j$$
$$being \quad CS_i > CS_j \; and \; NL_i < NL_j, \tag{4}$$

where $Ncell_i$ and NL_i are the number of cells and LODs created by using CS_i and $Ncell_j$ and NL_j the number of cells created by using CS_j.

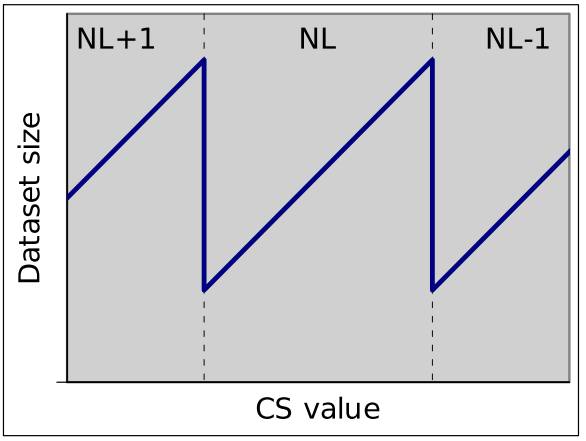


FIGURE 2. Dataset total size in relation to *CS* value and required LODs (*NL*).

In summary, higher *CS* values are recommended, as long as the point budget in use is high enough to avoid the aforementioned quality problems.

Figure 2 shows a representation of the change in total dataset size against the value of *CS*, as well as the changes in the LODs of the structure (*NL*). If the amount of LODs required does not change between *CS* values, the total dataset will increase, as each cell contains more points but the number of cells is the same. However, when an increase in the value of *CS* allows us to use fewer LODs, the total size is drastically reduced, since now there are fewer cells, even though each one is bigger. The ideal values for *CS* are those which cause that reduction in NL, since they provide the benefit of reducing the total size of the tree while keeping the inner cells as small as possible for that *NL*, reducing the redundancy overhead. In Section 3, we will check the validity of these premises.

2.2. Structure building stage: generation of a suitable multiresolution structure

In this section, the description of the quadtree estructure focused towards a server-client system is explained in detail. Additionally, the cell compression format used to optimize the loading times and storage requirements is described. The

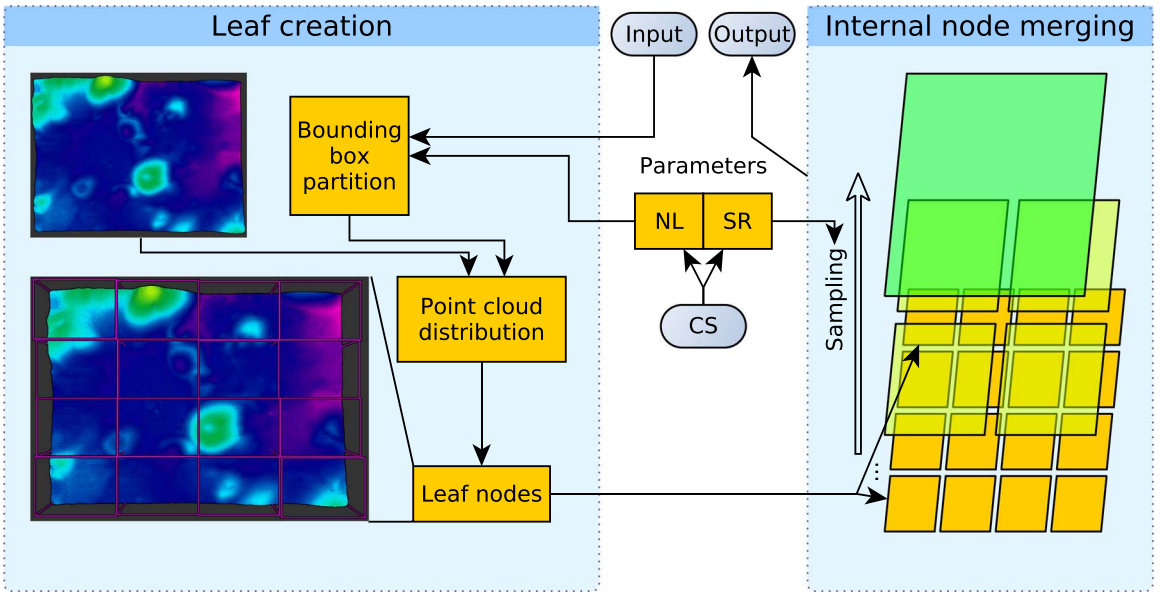


FIGURE 3. Diagram of the building process to create the completed data structure.

values for *CS* indicated throughout this paper are given without taking compression into account, since the actual compression ratio achieved varies from cell to cell, depending on the values of the points being compressed.

First, the bounding box of the point cloud is divided following a quadtree pattern: with *NL* and bounding box known, the leaf cells of uniform size are created with the correct dimensions in the deepest LOD. The size of the cells in the horizontal plane are divided by two in each level. Since the datasets do not always fill the bounding box, there may be empty leaf cells and cells with fewer points than expected by the point density. Then, the points are assigned to their corresponding leaf cell on the tree, and the structure is completed by merging the cells using a sampling process to populate the inner cells with points.

The sampling process is performed by dividing the parent cell into the same amount of divisions in each axis, creating a regular grid of tiles and selecting one point per tile, taking the points from the children cells. If some tiles have no points, other tiles can select more than one point, making use of the available space caused by the empty tiles. Figure 3 shows a representation of the building process, starting with the division of the point cloud to create the leaf cells and finishing with the sampling process to create the internal cells of the tree.

The original LiDAR files are usually compressed in order to reduce the amount of data to be stored, transferred and processed. To the best of our knowledge, LASzip [34] and LZ [31] are the best lossless compression methods for LiDAR data. Both formats use delta compression to store the differences in the properties of contiguous points using smaller data types when possible. TVPC uses a small variation of LZ, described in detail in [31]; the main difference is that it is fully lossless.

The storage format is not limited to TVPC or Pebbles; therefore, it is not known which data fields of the LAS format will be used; accordingly, all of the LAS data fields from record formats 0 to 5 are added to the compressed format.

Although the new data format can store every field, we added the possibility of saving space by not storing RGB data or GPS time data. These options can be toggled independently, in the event that the user knows that data will not be needed, the storage requirements can be reduced by disabling the fields not needed. We limit the options to GPS time and RGB data as they use an important amount of space and are more likely to not be used. For its use in Pebbles, RGB is stored but GPS time is removed.

The method for compressing the data is still delta compression, which is based on storing the differences between adjacent points in the file, using smaller data types when possible. There is one difference with respect to the version in [24]: since the GPS time is stored as a double precision floating point in the LAS format, we need to add support for 8-byte wide data. For simplicity, we process the GPS time as if it were a long integer; thus, we can use the already existing data structures for storing the differences, and we only need to add two more data structures: one for storing differences larger than a short integer using a 32 bit integer and the original values as a 64 bit integer, when the difference exceeds what a 32 bit integer can store.

Finally, after all the previous steps, we apply GZIP compression to the resulting data to maximize space savings as much as possible. GZIP compression is a very common compression

TABLE 1. Hardware specifications.

Platform	O.S.	CPU	GPU	RAM	Display	Bandwidth
Client desktop PC	Windows 10	AMD Ryzen 9 3900X	NVIDIA RTX 2080 Ti	32 GB	1920x1080 @60Hz	Gigabit Ethernet
Server	CentOS 6.7	Intel Xeon E5-2603 v3	–	64 GB	–	–

TABLE 2. Point clouds statistics.

Point cloud	N	SCP
PNOA	$28x10^9$	802 GB
Volcano	$550x10^6$	15.7 GB

format that has a very low cost on modern hardware, easily supported on most programming languages and applications. Specifically to our application, most web browsers support automatic decompression of incoming files if they are compressed with GZIP, eliminating that step for the decompression of the data in *Pebbles*.

3. EXPERIMENTAL RESULTS

An implementation of TVPC strategy, described in this paper, has been integrated into our visualization application, *Pebbles*, to evaluate the results produced with several point clouds.

Table 1 describes the test platform used in our experiments, using an NVIDIA RTX 2080 Ti GPU. On the software side, the visualization application is served from a server through an instance of Apache HTTP Server version 2.4.28. The application is accessed in the client using the open source web browser *Chrome*, version 92.0.4515.107, and only results for this browser are presented for the sake of clarity, since it offers good performance overall and provides adequate development tools. Nonetheless, other browsers have been tested and provide similar results.

Table 2 contains information on the datasets used for testing our strategy: the number of points, N, and original size of the full point cloud, SCP. The *PNOA* [35] (National Plan of Aerial Orthophotography, Spain) point cloud covers the region of Galicia, Spain, totaling to 28 billion points. This point cloud comes from an airborne LiDAR survey at a point density of 0.5 $point/m^2$ and is available in the Spanish GIS database (IDEE) (Infraestructura de Datos Espaciales de España IDEE). The *Volcano* [36] point cloud contains 0.55 billion points, with a point density of 13.71 $point/m^2$. This point cloud is available at OpenTopography.

In this section, each test is performed 10 times and the average is used. When the browser cache is involved, the process is repeated after enabling the browser cache (and performing an initial load not measured to allow any data to be brought to cache).

TABLE 3. Compression ratios for each dataset and *CS* value. *LR* stands for *Leafs Ratio* and *TR* stands for *Total Ratio.*

Dataset	CS	LR (%)	TR (%)	NL
PNOA LASzip	-	12.44	-	-
PNOA TVPC	2.5	12.77	38.90	12
	5	12.43	33.41	11
	10	12.43	38.02	11
	25	12.30	34.78	10
	50	12.30	39.27	10
Volcano LASzip	–	12.10	–	–
Volcano TVPC	2.5	13.37	19.18	7
	5	13.37	16.56	6
	10	13.50	19.29	6
	25	13.56	17.19	5
	50	13.66	15.56	4

3.1. Storage analysis

The first point of view of the analysis is with regards to the requirements of storage of multiresolution structure. Table 3 shows the compressed ratio, calculated as $\frac{Size_{\text{compressed}}}{Size_{\text{original}}} \times 100$, of the leaf cells alone as *leafs ratio* (*LR*) and the compression ratio of the entire multiresolution structure as *total ratio* (*TR*), due to the addition of data redundancy to support efficient multiresolution. Both ratios are expressed in percentage of the original size. *NL* is the number of LODs required for storing the dataset. This table allows an analysis regarding the advantage indicated in Premise 2 and the benefit of the compression format used.

Figure 4 shows the relative size with respect to the original size of the dataset. It shows the size of the leaf cells and the size of the inner part of the tree. The leaf cells contain the original points and the inner cells contain the redundant data to support the multiresolution technique.

The part of the graph that corresponds with PNOA datasets clearly follows the pattern shown in Fig. 2, with the drops in size ocurring for *CS* values between 2.5 and 5 MB as well as between 10 and 25 MB. These are the points where the *NL* changes. For the Volcano dataset, there are 4 reductions in NL, between 2.5 and 5 MB, between 10 and 25 MB and between 25 and 50 MB. For these two datasets, the best values for *CS* would be 5 or 25 MB for PNOA; 2, 25 or 50 MB in the case of Volcano. The selection between those depends on Premise 1.

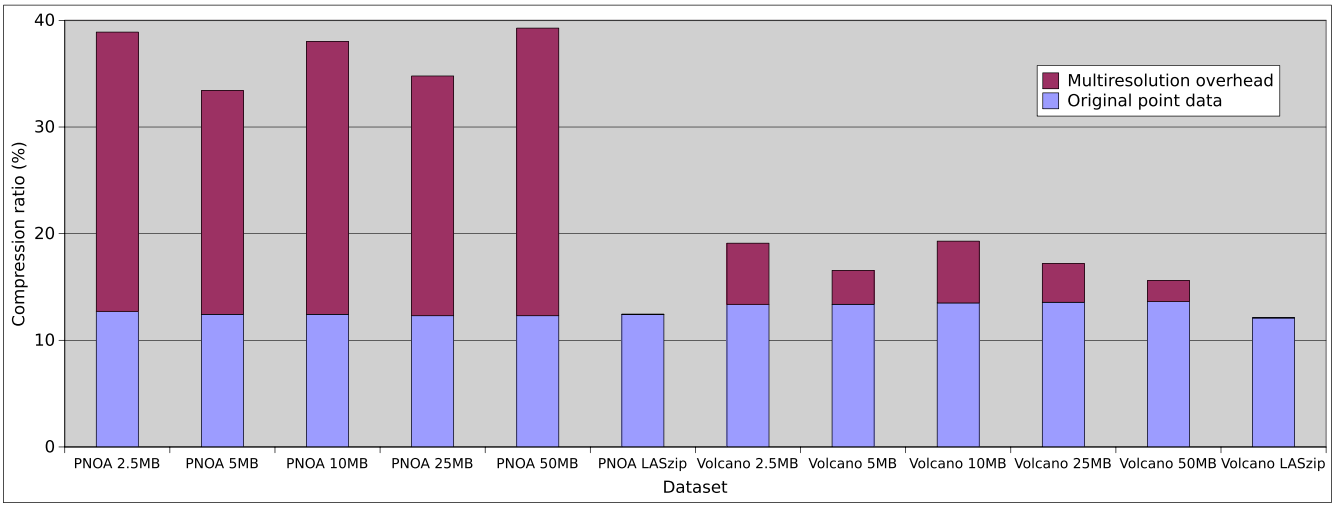


FIGURE 4. Compression ratios for each dataset and *CS* value.

We can compare our compression format with LASzip. The multiresolution structure stores the original points only in the leaf cells of the tree, and only original points in those cells. The compression format used is competitive with LASzip, especially in the PNOA dataset, with lower compression ratio in the most of the cases. A small difference with LASzip is expected due to reductions in delta encoding efficiency caused by the high amount of independent files required. In the PNOA case, 100 GB spread into files of a maximum size of *CS*, so ranging from 41 thousand files using 2.5 MB as *CS* to 2 thousand files in the 50 MB case.

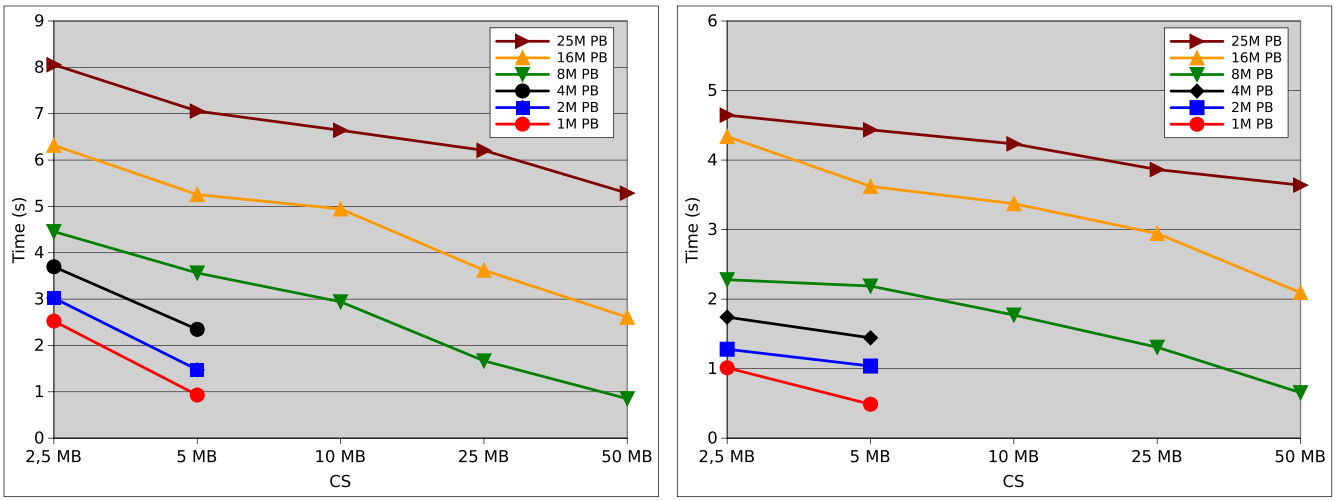
3.2. Loading times

As stated previously, this work is aimed towards high-performance systems, so we will test the limits with high point budgets to see the effects of different values for *CS*. All points budgets are tested with the two datasets processed with 5 different values for *CS*: 2.5, 5, 10, 25 and 50 MB, to study the changes in loading times. Values of 1, 2, 4, 8, 16 and 25 million points for the point budget will be tested since most systems can handle the first three, irrespective of whether they are high-end or low-end systems, and they allow us to compare with other visualization tools. The other three point budgets allow us to push more powerful systems. These values allow good results both in terms of performance and accurate representation of the original point clouds; 16 and 25 million points (and 8 M to a lesser extent) provide diminishing returns with current day technology. While 4k(UHD) screens are becoming more common, they would mainly take advantage of 8 M point budget, while higher resolutions such as 6k or 8k are needed to have more pixels than rendered points on higher point budgets. We measure loading time without browser cache as that would be the worst case.

Figure 5(a) shows the loading times for the PNOA dataset. Figure 5(b) shows the loading times for the Volcano dataset. The full datasets are loaded, allowing the multiresolution algorithm to decide the LOD to use according to the point budget and data loaded, using a downward-looking camera position showing the entire dataset.

It is clear that there is a downward trend in loading time when the value for *CS* is increased, due to the reduction in the total data downloaded which results from the increased *CS* values. Considering the most demanding point budgets, increasing the value of *CS* can provide speedups of up to 2.4x for the PNOA dataset and 2.1x for the volcano dataset. While one would be tempted to always use the highest *CS* value, in low point budgets, it results in lower image quality, as it will be verified in Section 3.3. There are no data for 10, 25 and 50 MB as *CS* with the first three point budgets because they are too low, they violate the first premise in Section 2.1, and in some cases, it is not possible to load even 1 cell with the given point budget. For example using 50 MB as *CS*, the root cell in the Volcano dataset (the lowest detail possible) contains 3.09 million points, one cell alone has more points than allowed in the lowest 2 point budgets and cannot be displayed.

Expanding on this behavior, in the Volcano dataset, using 50 MB as *CS* as before and 8 million points as point budget, only the 3.09 million points from the root are rendered. The visualization tool cannot replace the root cell with its four children as the point count of the four children (aproximately 12 million points) exceeds the point budget, and it has to replace one cell with the four children since the four children are in the field of view, otherwise parts of the dataset would not be rendered. If a *CS* value of 10 MB is used instead, 25 cells are rendered at the same time, replacing one with its children has a lesser impact on the total point count, and less available points are required to allow each LOD change.



(a) PNOA. (b) Volcano.

FIGURE 5. Loading times for the test datasets with different point bugdet and *CS* values.

In higher point budgets, this is not a problem because there are more cells being rendered, even on high *CS* values. There is more flexibility to choose which cell gets replaced with its children; therefore, there is a greater chance of filling a larger part of the point budget. This is the source of the first rule in Section 2.1.

With all of the data presented, we can make an informed decision on which value for CS is recommended. Due to the inability of lower budgets to display datasets that use higher values for *CS*, we will make two different recomendations: one for lower points budgets intended for low end computers or mobile devices and another one for higher point budgets intended for high-performance computers. The best values are shown in Table 4. For lower PBs, speedups in loading time of using 5MB as *CS* with respect to 2.5 MB are between 1.58x and 2.72x in the PNOA dataset and between 1.21x and 1.58x in the volcano dataset. For higher PBs, comparing against the 2.5MB for *CS*, the speedups in loading time range from 1.30x and 2.67x for the PNOA dataset using 25MB as *CS*, and between 1.28x and 3.48x in the Volcano dataset using 50MB as *CS*. We recomend 25MB for *CS* in the PNOA dataset given that with 50 MB, there is no reduction in *NL*, so the total size is disk increases notably with that *CS*, while in Volcano, there is a reduction in *NL* going from 25 to 50 MB for *CS* and the total size in disk is reduced.

To finish this discussion, we analyze whether or not the different values for *CS* affect the loading time; in other ways, in Fig. 6, we show the relation between the total size of the downloaded data and the loading time, for each case in the two previous graphs. Each data point corresponds with a combination of point budget and *CS* value, so 2 data points for lower budgets and 5 data points for 8, 16 and 25 million points, for each dataset. For reference, the curves for transfer speeds of 200, 300 and 400 Mbps are also displayed.

TABLE 4. *CS* value recomendations.

	Point budget	
Dataset	1M-4M	8M-25M
PNOA	5MB	25MB
Volcano	5MB	50MB

Once past the first tens of MB of transferred data, the transfer speed remains fairly constant, indicating that the main bottleneck for loading time is the transfer speed. With lower point budgets and low amounts of data transferred, the time required for loading the page and executing the visualization code becomes the dominant part, adding a fixed amount to the loading time.

The reductions in loading time are achieved by reducing the amount of data transferred, not by sending the data faster, as that would show up in the graph. Therefore, higher values for *CS* have a beneficial impact on the amount of data that needs to be moved through the network, improving the efficiency of the process of filling the point budget. Higher values for *CS* allow the point budget to be filled with fewer cells; even though each cell is bigger, the total size is lower. This is due to the need for fewer intermediate cells, cells that are not rendered at the end of the load as there is enough space in the point budget to render their children. These intermediate cells could be considered a *waste* of data; they are rendered only until the four children are ready. Downloading and rendering these intermediate cell is beneficial as it allows us to use the visualization tool immediately after the root cell is downloaded and rendered, and the user can interact with the dataset, while the rest of the cells are still downloading. This allows us to

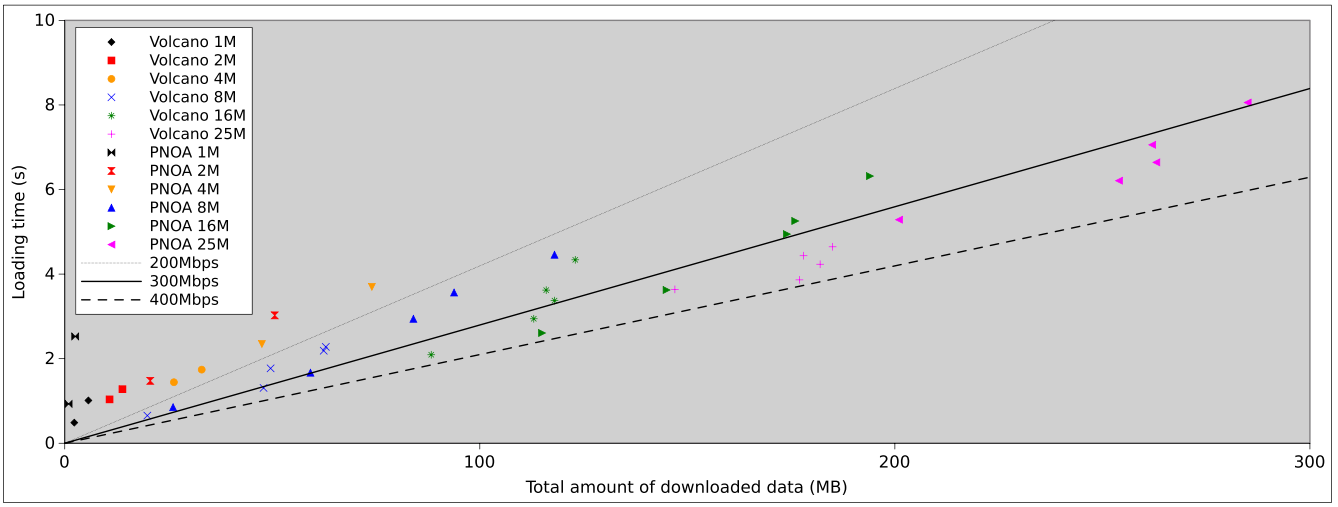


FIGURE 6. Loading time versus download size.

improve the interactivity with respect to other proposals that need to wait until the final LOD is downloaded. We measure the time to load a full dataset as it is the moment when the dataset is at the highest detail possible, as well as being a more fair comparison with other visualization tools that do not have this *while-loading* interaction capability.

3.3. Interactive visualization

With regards to the interactivity in terms of frames per second, all of the tests performed maintained 60 frames per second on the testing computer, even in the more demanding cases, such as using 25 million points as point budget.

On the other hand, the image quality can be impacted by the relation between *CS* value and point budget, according to the first premise in Section 2.1. Lower values of *CS* provide good results in all point budgets, with the higher loading times seen previously, but higher values require higher point budgets to maintain good visual quality in all cases. In extreme cases, for example 1 or 2 million points as PB and 50 MB as *CS*, they cannot be used at the same time since not even one cell can fit into those PB.

Figure 7 shows the same position on the PNOA dataset, using the same PB but two different values for *CS*, 5 MB in Fig. 7(a) and 50 MB in Fig. 7(b). Figure 8 does the same in the Volcano dataset, this time using the height as a color gradient. As can be seen, Figs 7(a) and 8(b) have a higher point density, which shows notably more detail in the urban area in Fig. 7(a), allowing the differentiation of streets and buildings, or the trees in Figs 8(a), 7(b) and 7(b) have lower point density. The area rendered is split in two or more cells, and in the 50 MB, those cells do not fit into the PB at the same time. The visualization tool has to load the first common parent of those cells that covers the entire area rendered. This is the main reason to include the first premise in Section 2.1, to provide a guide to avoid this situations.

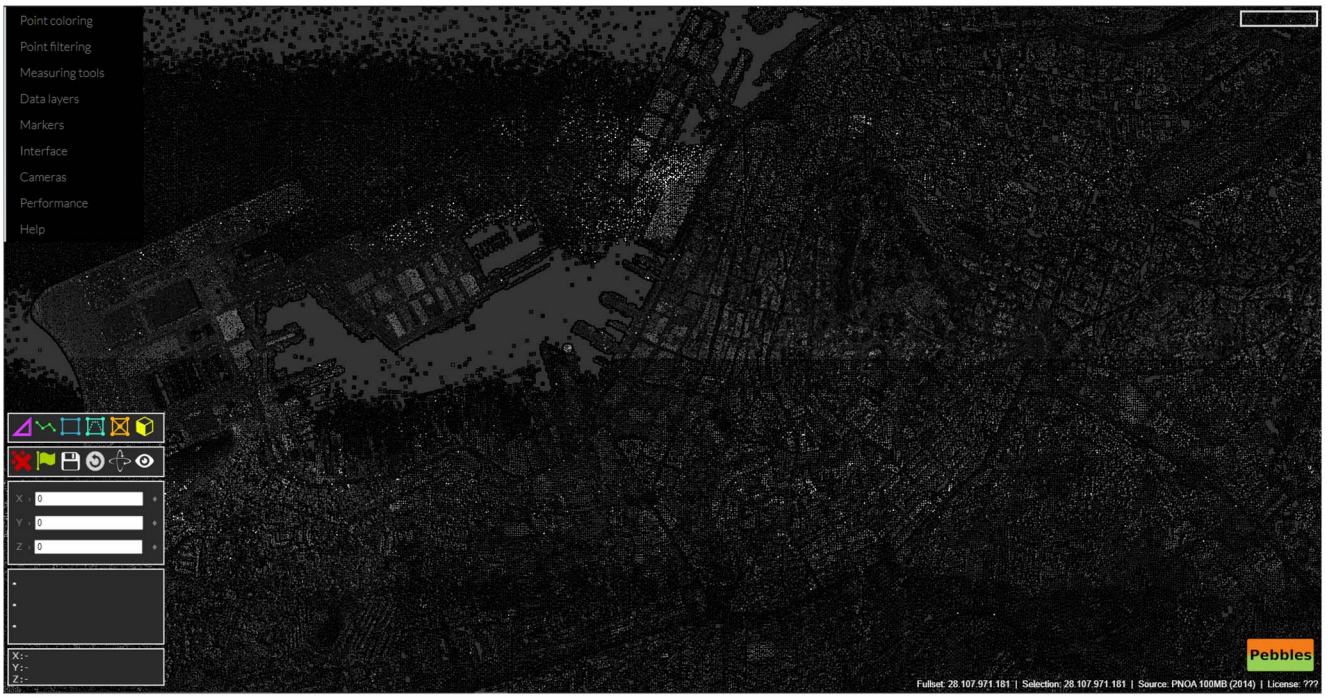
To close this section and show the visual quality of the images obtained with our proposal achieving 60 FPS, Fig. 9 shows the same localization in the PNOA dataset displaying color data taken from satellite images. The *CS* and PB are the same as Fig. 7(a), 5MB and 8 million points, respectively.

3.4. Comparison against Vilma

The comparison is focused on memory consumption and loading time. We compare with a previous framework, *Vilma* [24], designed to work on multiple platforms in a client-server system, from mobile devices to desktops and, to the best of our knowledge, the best proposal in terms of loading times or memory requirements. In [24], the performance of *Vilma* is compared with *Potree* (version 1.5), another well-known alternative for web-based visualization. The results show an improvement of 54–86% in memory requirements and 47–54% in loading times over *Potree* using a classic octree approach.

We have tested the lastest available version of *Potree*, 1.8, but we cannot obtain adequate measurements with this version following the same testing procedure. The testing procedure is loading the entire dataset without user interaction with the camera, loading in the default position and orientation, to avoid introducing user reaction times and consistency of movements into the variables. The default camera position in *Potree* seems to locate the camera and points far enough to not trigger the load of an increased LOD, regardless of the point budget used. If the user zooms in and moves the camera, more data are loaded and then the point budget comes into effect. For these reasons, we cannot test *Potree* to the same extent, only measuring with 1 M for PB in the Volcano dataset, obtaining 1.97 and 1.75 s

(a) 8 million points and 5MB as *CS*.

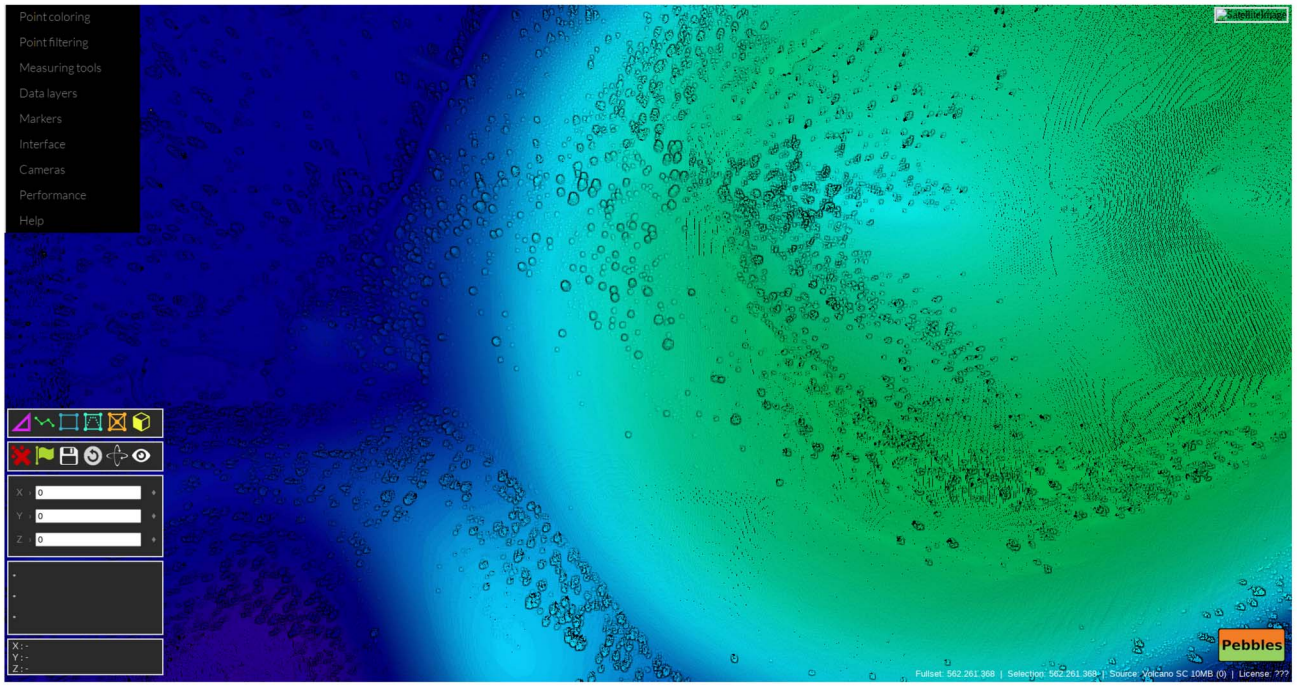


(b) 8 million points and 50MB as *CS*.

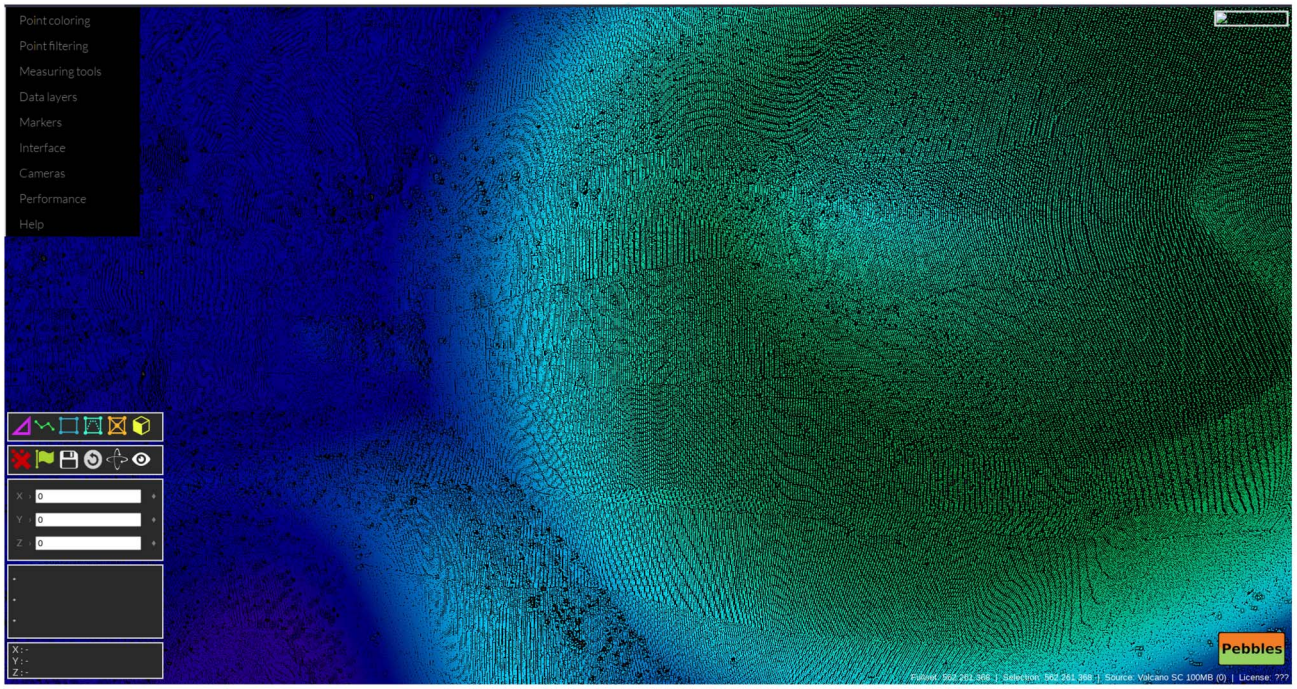
FIGURE 7. *CS* and PB relation comparison in PNOA dataset.

for loading times without and with cache, respectively, using 281 MB of RAM in the process. Pebbles is 4x faster loading the dataset without cache, 4.9x faster with cache and uses 40% less memory.

The memory consumption measured during the load of the full point cloud is shown in Fig. 10. Only results with the lower values of *CS* of 2.5 and 5 MB per cell are provided. Higher values would cause problems in these low point



(a) 8 million points and 5MB as *CS*.



(b) 8 million points and 50MB as *CS*.

FIGURE 8. *CS* and PB relation comparison in PNOA dataset.

budgets, following the premises shown in Section 2.1. There are no data for *Vilma* for 8 and 16 million points since it does not have those options owing to the hardware limitations of the target devices. We measure RAM and VRAM separately, obtaining the values using the Task Manager provided by the desktop version of the Chrome Web Browser. For these tests, we perform the loading of the entire area of the point cloud after clearing browser's cache, annotating

FIGURE 9. Part of PNOA dataset using 5 MB as *CS* being rendered with 8 million points using color data from satellite.

the highest value observed during the loading of all the data needed.

As expected, RAM usage increases with the point budget, due to the need for the application to store and manage more points and cells of the multiresolution structure. For point budgets of 1–4 million points, memory usage is under 1 GB in all cases. Even low-end laptops and desktop PCs have 4 or 8 GB of RAM available, therefore those lower end systems can still use *Pebbles* without reaching memory constraints. On the other hand, 8 and 16 million points budgets can push the memory usage to 1.5 GB, and they are only recomended for more powerful systems.

The VRAM variation is much more limited, with all cases under 800 MB. It also has less variation between datasets. This is due to the more limited data moved to the GPU, which is the main difference for the amount of points in each point budget. The space required to store a certain amount of point does not change from dataset to dataset. The amount of points displayed is the main contributor to the difference in VRAM use. As modern dedicated GPUs have at least 1 GB of VRAM, usually more, we consider this consumption adequate even for low end systems.

Since the use of browser cache can have a significant impact, we provide the results with and without cache. Figure 11(a) contains the results enabling and making use of the browser cache, while in Fig. 11(b), the results with cache disabled are shown. The time measurements are obtained using the developer tools of the Chrome Web Browser.

The time needed for full loading without cache varies from 0.5 to 6 s for *Pebbles*. Overall, in 4 s, any dataset can be fully loaded with no cache (first loading, for example) on lower point budgets, more adequate for most computers, while more powerful ones can use higher point budgets which add a few seconds to the loading time. In this situation, *Pebbles* is up to 7.89 times faster using a *CS* value of 5 MB, and 3.8 times faster when using a *CS* value of 2.5MB.

When data are in cache, the loading time can be reduced by up to 1 s in the most demanding cases, while a minimum reduction of a few tenths of a second in the lower point budgets. As already seen, the higher value for *CS* provides better results, reinforcing the advantage of *Pebbles*. In this case, *Pebbles* is up to 10.84 times faster using a *CS* value of 5 MB, and 4.58 times faster when using a *CS* value of 2.5MB.

With these data, we can clearly show the strong points of the new viewer *Pebbles* versus our previous work *ViLMA*: *ViLMA* focused on lower memory footprint to allow use on mobile devices, which it achieved at the cost of loading times. On the other hand, *Pebbles* is faster in loading times and provides a better interactive experience, and the increase in memory requirements is not a problem in the high-performance computers that it is focused on, achieving the objective of reduced loading times.

We would like to point out that there are some differences in the systems involved with respect to our previous work, namely a large increase in network bandwidth and a more recent version of the chrome web browser, with the improvements

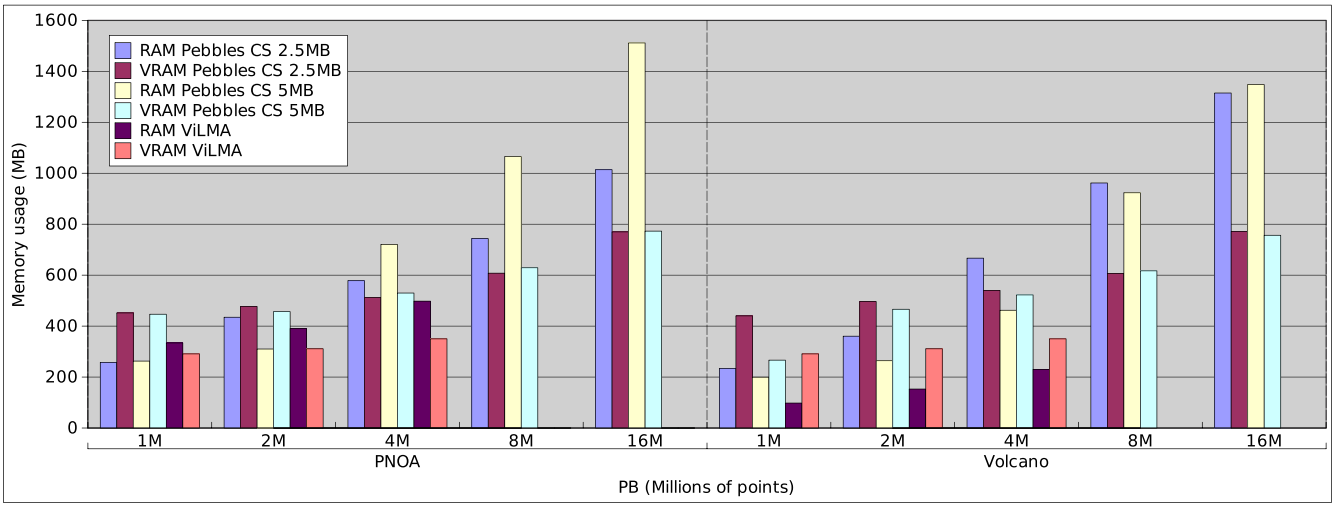
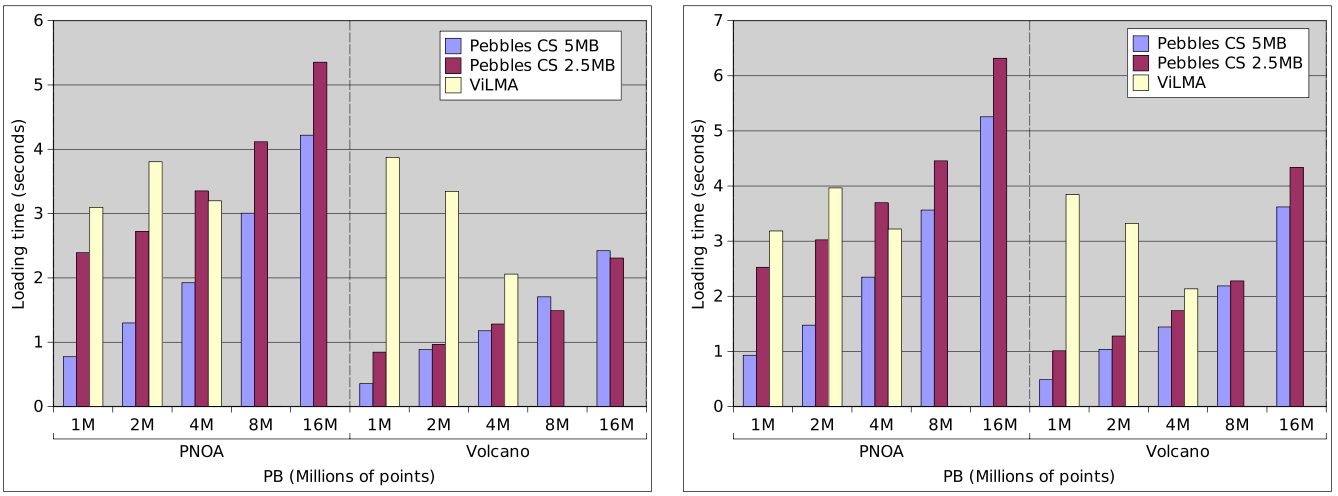


FIGURE 10. Comparison between *Vilma* and *Pebbles* in terms of memory consumption for each dataset and point budget, considering a *CS* of 2.5 and 5 MB for *Pebbles*.



(a) Cache enabled.

(b) Cache disabled.

FIGURE 11. Comparison between *Vilma* and *Pebbles* in terms of loading times for each dataset and point budget, using 2.5 and 5 MB as *CS* values for *Pebbles*.

that the newer version includes. This last part is relevant as it improved the parallel behaviour of multiple network requests, and *ViLMA* can leverage this two factors combined to greatly improve loading times. This is the source of the difference in *ViLMA*'s times compared with those published in [24].

4. CONCLUSIONS AND FUTURE WORK

This work presents an autotuning multiresolution out-of-core strategy with the objective of reducing loading times and keeping memory requirements low while achieving good quality interactive visualization of massive point clouds. A further objective is the minimizing of the time needed to build the multiresolution structure of a point cloud, as previous ones often use trial-and-error techniques, which can take hours or days with large point clouds.

Our proposal is tested in our own web-based visualization software, designed to work with the tuned structures, and shows good performance results. The loading times improve with respect to *ViLMA*, the best proposal to our knowledge. The interactive visualization is measured in frames per second and is kept over 60 fps, even in the most demanding cases with point budgets of 25 million points in display at the same time.

In terms of future work, there is the potential to extend this approach to other tools that use point clouds as input, not

just web-based visualization. This strategy may help bridge the transition of those tools to Big Data, especially for more complex geoprocessses with higher computational costs.

DATA AVAILABILITY STATEMENTS

Parts of the data that support the findings of this study are openly available in OpenTopography at https://doi.org/10.5069/G9K0726C, reference number "OTLAS.022015.26912.1". Parts of the data that support the findings of this study are openly available in Centro Nacional de Información Geográfica, CNIG at https://doi.org/10.7419/162.09.2020, product ID LiDAR-PNOA-cob1.

ACKNOWLEDGEMENTS

The point clouds and LiDAR datasets used in this work belong to

- PNOA: Data from the LiDAR-PNOA data repository, region of Galicia, were provided by Instituto Geográfico Nacional de España. Dataset license: LiDAR-PNOA 2009 CC-BY 4.0 scne.es
- Volcano: This work is based on Point Cloud Data Distribution and Processing services provided by the OpenTopography Facility with support from the National Science Foundation under NSF Award Numbers 1948997, 1948994 and 1948857.

FUNDING

Ministry of Science and Innovation of Spain (PID2019-104184RB-I00 / AEI / 10.13039/501100011033); Galician Government under the Consolidation Program of Competitive Research Units (Ref. ED431C 2021/30); Centro de Investigación de Galicia "CITIC"; Government of Galicia; European Union (European Regional Development Fund- Galicia 2014-2020 Program by grant ED431G 2019/01); Government of Galicia and the European Social Fund (ESF) of the European Union (predoctoral fellowship ref. ED481A-2019/231 to D.T.); Funding for open access charge: Universidade da Coruña/CISUG.

REFERENCES

[1] Gevaert, C., Persello, C., Nex, F. and Vosselman, G. (2018) A deep learning approach to DTM extraction from imagery using rule-based training labels. *ISPRS J. Photogramm. Remote Sens.*, 142, 106–123.

[2] Wen, C., Li, X., Yao, X., Peng, L. and Chi, T. (2021) Airborne LiDAR point cloud classification with global-local graph attention convolution neural network. *ISPRS J. Photogramm. Remote Sens.*, 173, 181–194.

[3] Wolf, J., Pietz, T., Richter, R., Discher, S. and Döllner, J. (2021) Image-Based Road Marking Classification and Vector Data Derivation from Mobile Mapping 3D Point Clouds. In Giovanni Maria Farinella, Petia Radeva, Jose Braz and Kadi Bouatouch (eds) *Proc. of the 16th Int. Joint Conf. on Computer Vision, Imaging and Computer Graphics Theory and Applications - Volume 4: VISAPP*, 8–10 February 2021, pp. 227–234. INSTICC SciTePress, Setúbal, Portugal.

[4] Comino, M., Andújar, C., Chica, A. and Brunet, P. (2017) Error-aware construction and rendering of multi-scan panoramas from massive point clouds. *Comput. Vis. Image Underst.*, 157, 43–54.

[5] Yuan, S., Zhu, S., Li, D.-S., Luo, W., Yu, Z.-Y. and Yuan, L.-W. (2018) Feature preserving multiresolution subdivision and simplification of point clouds: A conformal geometric algebra approach. *Math. Methods Appl. Sci.*, 41, 4074–4087.

[6] Baert, J., Lagae, A. and Dutré, P. (2013) Out-Of-Core Construction of Sparse Voxel Octrees. In *Proc. of the 5th High-Performance Graphics Conf.*, New York, NY, USA HPG '13, pp. 27–32. ACM, New York.

[7] Sarton, J., Courilleau, N., Remion, Y. and Lucas, L. (2019) Interactive visualization and on-demand processing of large volume data: a fully GPU-based out-of-core approach. *IEEE Trans. Vis. Comput. Graph.*, 26, 3008–3021.

[8] Gobbetti, E., Marton, F. and Iglesias Guitián, J. (2008) A single-pass GPU ray casting framework for interactive out-of-core rendering of massive volumetric datasets. *Vis. Comput.*, 24, 797–806.

[9] Hoetzlein, R.K. (2016) GVDB: Raytracing Sparse Voxel Database Structures on the GPU. In Ulf Assarsson and Warren Hunt (ed) *Eurographics/ ACM SIGGRAPH Symposium on High Performance Graphics*, June 20–22, pp. 109–117. The Eurographics Association, Geneve, Switzerland.

[10] Beyer, J., Hadwiger, M. and Pfister, H. (2015) State-of-the-art in GPU-based large-scale volume visualization. *Comput. Graph. Forum*, 34, 13–37.

[11] Gao, Z., Nocera, L., Wang, M. and Neumann, U. (2014) Visualizing Aerial LiDAR Cities with Hierarchical Hybrid Point-Polygon Structures. In Paul G. Kry and Andrea Bunt (ed) *Proc. of Graphics Interface 2014*, Montreal, Quebec, Canada, 7–9 May 2014, pp. 137–144. Canadian Information Processing Society, Mississauga, Canada.

[12] Rodríguez, M.B., Gobbetti, E., Marton, F. and Tinti, A. (2013) Coarse-Grained Multiresolution Structures for Mobile Exploration of Gigantic Surface Models. In *Proc. of the SIGGRAPH Asia 2013 Symposium on Mobile Graphics and Interactive Applications*, SA '13, Hong Kong, 19–22 November 2013, pp. 1–6. ACM, New York.

[13] Discher, S., Richter, R. and Döllner, J. (2019) Concepts and techniques for web-based visualization and processing of massive 3D point clouds with semantics. *Graph. Model.*, 104, 101036.

[14] Goswami, P., Erol, F., Mukhi, R., Pajarola, R. and Gobbetti, E. (2013) An efficient multi-resolution framework for high quality interactive rendering of massive point clouds using multi-way kd-trees. *Vis. Comput.*, 29, 69–83.

[15] 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, pp. 227–242. Springer Cham.

[16] Kuhn, A. and Mayer, H. (2015) Incremental Division of Very Large Point Clouds for Scalable 3D Surface Reconstruction. In *2015 IEEE Int. Conf. on Computer Vision Workshop (ICCVW)*, Santiago, Chile, 5–13 December 2015, pp. 157–165. Conference Publishing Services (CPS).

[17] Wang, L., Xu, Y. and Li, Y. (2017) Aerial LiDAR point cloud voxelization with its 3D ground filtering application. *Photogramm. Eng. Remote Sens.*, 83, 95–107.

[18] Yu, A. and Mei, W. (2019) Index model based on top-down greedy splitting r-tree and three-dimensional quadtree for massive point cloud management. *J. Appl. Remote. Sens.*, 13, 1.

[19] Pfister, H., Zwicker, M., Baar, J. and Gross, M. (2000) Surfels: Surface Elements as Rendering Primitives. In *Proc. of the ACM SIGGRAPH Conf. on Computer Graphics*, New Orleans, July 23–28, 2000, pp. 335–342. ACM, New York.

[20] Rusinkiewicz, S. and Levoy, M. (2000) QSplat: A Multiresolution Point Rendering System for Large Meshes. In *Proc. of the ACM SIGGRAPH Conf. on Computer Graphics*, New Orleans, July 23–28, 2000, pp. 343–352. ACM, New York.

[21] Gobbetti, E. and Marton, F. (2004) Layered point clouds: a simple and efficient multiresolution structure for distributing and rendering gigantic point-sampled models. *Comput. Graph.*, 28, 815–826.

[22] Wimmer, M. and Scheiblauer, C. (2006) Instant Points: Fast Rendering of Unprocessed Point Clouds. In *Symposium on Point-Based Graphics*, Massachusetts, 29–30 July, 2006, pp. 129–136. A K Peters/CRC Press, Natick, Massachusetts.

[23] Schütz, M. (2016) Potree: Rendering Large Point Clouds in Web Browsers. Master's Thesis. In *Institute of Computer Graphics and Algorithms, Vienna University of Technology Favoritenstrasse 9–11/E193–02*. A-1040 Vienna, Austria.

[24] Deibe, D., Amor, M. and Doallo, R. (2019) Supporting multiresolution out-of-core rendering of massive LiDAR point clouds through non-redundant data structures. *Int. J. Geogr. Inf. Sci.*, 33, 593–617.

[25] Cura, R., Perret, J. and Paparoditis, N. (2016) *Implicit LOD for processing, visualisation and classification in point cloud servers*. *ArXiv*, **abs/1602.06920**.

[26] Schütz, M., Krösl, K. and Wimmer, M. (2019) Real-Time Continuous Level of Detail Rendering of Point Clouds. In *2019 IEEE Conf. on Virtual Reality and 3D User Interfaces (VR)*, Osaka, 23–27 March 2019, pp. 103–110. IEEE.

[27] Schütz, M., Mandlburger, G., Otepka, J. and Wimmer, M. (2020) Progressive real-time rendering of one billion points without hierarchical acceleration structures. *Comput. Graph. Forum*, 39, 51–64.

[28] (2020). *Dielmo3D website*. http://www.dielmo.com/en. [Online; accessed 08-June-2022].

[29] Potree repository. https://github.com/potree/potree/. [Online; accessed 08-June-2022].

[30] Cesium website. https://cesium.com. [Online; accessed 08-June-2022].

[31] Deibe, D., Amor, M., Doallo, R., Miranda, D. and Cordero, M. (2017) GVLiDAR: an interactive web-based visualization framework to support geospatial measures on LiDAR data. *Int. J. Remote Sens.*, 38, 827–849.

[32] PCL website. https://pointclouds.org. [Online; accessed 08-June-2022].

[33] Butler, H., Chambers, B., Hartzell, P. and Glennie, C. (2021) PDAL: an open source library for the processing and analysis of point clouds. *Comput. Geosci.*, 148, 104680.

[34] Isenburg, M. (2013) LASzip: lossless compression of LiDAR data. *Photogramm. Eng. Remote Sens.*, 79, 209–217.

[35] PNOA dataset. https://pnoa.sign.es/. License: LiDAR-PNOA 2009 CC-BY 4.0 scne.es.

[36] (2015) *Sunset Crater Volcano National Monument, AZ airborne LiDAR*. https://doi.org/10.5069/G9K0726C. License: National Park Service (2015). Sunset Crater Volcano National Monument, AZ airborne LiDAR. Watershed Sciences, Inc. (WSI), distributed by OpenTopography.