This is an Accepted Manuscript of an article published by Taylor & Francis in INTERNATIONAL JOURNAL OF GEOGRAPHICAL INFORMATION SCIENCE on 28 Nov 2018, available at: https://doi.org/10.1080/13658816.2018.1549734

RESEARCH ARTICLE

Supporting multi-resolution out-of-core rendering of massive LiDAR point clouds through non-redundant data structures

David Deibe[a], Margarita Amor[a] and Ramón Doallo[a]

[a]Faculty of Computer Science, University of A Coruña (UDC), A Coruña, Spain

ARTICLE HISTORY
Compiled November 12, 2018

ABSTRACT
In recent years, the evolution and improvement of LiDAR (*Light Detection and Ranging*) hardware has increased the quality and quantity of the gathered data, making the storage, processing and management thereof particularly challenging. In this work we present a novel, multi-resolution, out-of-core technique, used for web-based visualization and implemented through a non-redundant, data point organization method, which we call *Hierarchically Layered Tiles* (HLT), and a tree-like structure called *Tile Grid Partitioning Tree* (TGPT). The design of these elements is mainly focused on attaining very low levels of memory consumption, disk storage usage and network traffic on both, client and server-side, while delivering high performance interactive visualization of massive LiDAR point clouds (up to 28 billion points) on multiplatform environments (mobile devices or desktop computers). HLT and TGPT were incorporated and tested in ViLMA (**Vi**sualization for **L**iDAR data using a **M**ulti-resolution **A**pproach), our own web-based visualization software specially designed to work with massive LiDAR point clouds.

KEYWORDS
LiDAR; web-visualization; efficient data structures; multi-resolution; out-of-core

1. Introduction

LiDAR (*Light Detection and Ranging*) technology provides extremely useful high-resolution data in the form of point clouds that can be applied in a wide range of fields, such as agriculture, archaeology, biology, geology or forestry. Earth-science applications of LiDAR include coastal-change studies (Sallenger *et al.* 1999), extraction of geomorphologic features (Passalacqua *et al.* 2010), creation of detailed large-scale city models (Lafarge and Mallet 2012), analysis of land sliding process (Ventura *et al.* 2011), volcanoes (Kereszturi *et al.* 2012) and active tectonics (Arrowsmith and Zielke 2009, Brunori *et al.* 2013), among others. Useful reviews of the use of LiDAR data in earth surface processes are provided in Roering *et al.* (2013), Tarolli (2014), Yan *et al.* (2015).

The use of LiDAR technology has been experiencing notable growth in many scientific fields, particularly in recent years, with the rise of unmanned aerial vehicles (UAV). Nowadays, the huge amount of spatial information that may be acquired by modern LiDAR hardware entails an enormous challenge when developing applications

CONTACT David Deibe. Email: david.deibe@udc.es

focused on handling and managing such amounts of data. On the server-side of GIS centres, or any other large company that constantly generates those volumes of information, the simple storage demands a significant cost in terms of economic and technical resources. On the client-side, the popularization of tablets, laptop hybrids and smartphones, in addition to the release of HTML5, has favoured the appearance of applications with high portability, flexibility and availability. This is a major advance in comparison with classic desktop applications which, in many cases, are only designed for one specific operating system or device. Nevertheless, the advantages of web applications come with an important handicap, as there are strong restrictions on main memory and disk storage capacities and performance in terms of execution time. The need of processing and visualization algorithms for LiDAR datasets becomes a key point in different fields, such as in González-Ferreiro *et al.* (2013), where the examination of point clouds is required in order to provide forest inventory and biomass estimation. Real-time interaction with the 3D point clouds or the storage capabilities for holding LiDAR datasets are some of the particularly sensitive problems to be addressed, as can be seen in other fields, such as neuroscience (Al-Awami *et al.* 2014).

In this article, we present a novel, multi-resolution, out-of-core technique, implemented through a point data organization method which we call *Hierarchically Layered Tiles* (HLT), used offline during a preprocessing stage, together with a tree-like structure called *Tile Grid Partitioning Tree* (TGPT), created only in runtime to efficiently manage the point clouds. Both elements have a non-redundant data design, and lossless compression methods are applied over all preprocessed data, greatly reducing the amounts of data to be handled. The storage requirements are lower than for other multi-resolution strategies using conventional static precomputed models with high data redundancy. The concept of layered points with a view to avoiding redundant data was explored in Gobbetti and Marton (2004) but with totally different goals, approaches and context. These techniques were tested in *ViLMA* (**Vi**sualization for **L**iDAR data using a **M**ulti-resolution **A**pproach), our web application specially designed to work with massive LiDAR point clouds with up to 28 billion points even in devices with relatively few resources, achieving very low memory consumption, client and server-side disk storage usage and network traffic, while maintaining real-time interaction and allowing data queries based on spatial restrictions to be made.

The rest of the article is organized as follows. In Section 2, related work is reviewed. In Section 3, the general system structure and design of *ViLMA* is described. Section 4 presents the HLT and TGPT data structures. In Section 5, several design decisions regarding performance are discussed, while in Section 6, we evaluate our proposals using *ViLMA*. Finally, Section 7 presents the main conclusions and future work.

2. Related Work

The visualization of point clouds is a widely discussed topic and there are currently a number of software solutions capable of handling all kinds of point clouds.

Multi-resolution strategies provide great performance benefits in a wide variety of applications involving large point clouds (Comino *et al.* 2017, Yuan *et al.* 2017) being particularly relevant in graphics. In Kovač and Žalik (2010), the visualization of large LiDAR datasets is introduced using real-time point-based rendering techniques. A similar approach is used in Kuder and Žalik (2011) for a web-based environment. An approach that addresses the 2.5D nature of aerial LiDAR points is proposed in Gao *et al.* (2014), wherein an interactive and visually-complete rendering of aerial LiDAR

point clouds of cities is proposed. Rodríguez *et al.* (2013) propose a web system for mobile devices based on a compressed, multi-resolution model and an efficiency-based mesh representation. Interactive visualization of massive 3D point clouds, which exceed available memory resources and rendering capabilities, is achieved by out-of-core or external memory algorithms (Goswami *et al.* 2013, Richter *et al.* 2015). Spatial data structures and level of detail (LOD (Tomas Akenine-Möller 2008)) techniques are used in these proposals. In Goswami *et al.* (2013), the authors achieve the rendering of massive points using geo-morphing and smooth point interpolation. In Richter *et al.* (2015), an interactive, out-of-core rendering is presented based on a layered, multi-resolution kd-tree. In Discher *et al.* (2017), an out-of-core, real-time rendering system for massive 3D point clouds is combined with interactive and view-dependent see-through lenses to enhance recognition of objects, semantics, and temporal changes within 3D point cloud depictions.

Currently available platforms focused on web visualization and handling of LiDAR data include: *Dielmo3D*[1], *Potree*[2], *MegaTree*[3] or *GVLiDAR* (Deibe *et al.* 2017). All of these platforms use their own file formats, data structures and multi-resolution methods. In order to evaluate our proposals (HLT and TGPT), we have used our own web-based visualization software, *ViLMA*, that will be presented in following Section 3. *ViLMA* has been designed with the goal of providing specific measurement tools to help Agricultural Engineers in specific tasks of their field of knowledge, as this type of tools were not offered by any other platform. For example, the creation of complex volumetric objects that could help on wood mass estimations(González-Ferreiro *et al.* 2013). *GVLiDAR* offers this type of tools, but it lacks of multi-resolution out-of-core capabilities, while *Potree* is very focused on visualization, using very efficient multi-resolution out-of-core techniques, but lacking of specific tools. On the other hand, with *ViLMA*, we have tried to offer both elements, in addition to data queries based on spatial restrictions. We should note here that, the concepts presented in this work regarding non-redundant data structures, could be applied to any other visualization software besides *ViLMA*. In Section 6.6, a comparison between *Potree* and *ViLMA* is presented, as, among the currently available web applications, *Potree* is the most similar to *ViLMA*.

3. Structure of ViLMA

ViLMA is a web-based application designed for the interactive 3D visualization and exploration of large LiDAR point clouds achieving real-time interaction; i.e., over 24 frames per second (FPS). *ViLMA* was developed for being used by Agricultural Engineers and was intended to meet the requirements of these type of professionals; hence it was mainly focused on providing accurate geospatial measurement tools to be used directly over the 3D point clouds and allowing data queries based on spatial restrictions. None of the applications cited at the end of Section 2 were designed following such requirements, with the exception of *GVLiDAR* which lacks of multi-resolution out-of-core capabilities.

Figure 1 shows the general system structure of *ViLMA*. It follows the conventional client-server structure used in most web applications. The front-end, written in JavaScript and HTML5, can be executed in any WebGL-compatible web browser.

[1]Dielmo3D website: http://www.dielmo.com/en/
[2]Potree website: http://potree.org/
[3]MegaTree website: http://wiki.ros.org/megatree

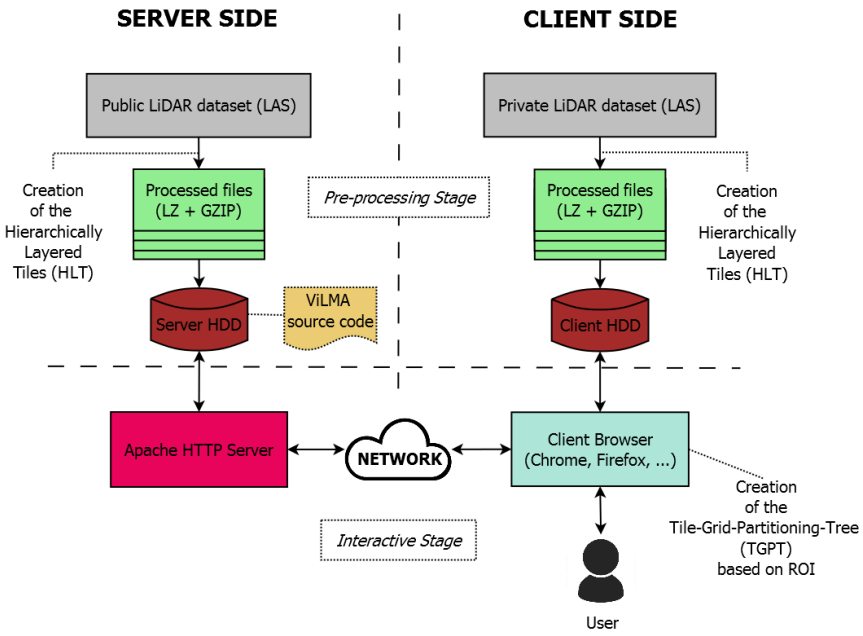


Figure 1. General structure of *ViLMA*.

For the back-end, an Apache HTTP Server[4] was deployed for serving the application code and the point data requests. Although *ViLMA* is a web application that mainly provides data on-demand from the Apache server, it is also designed to work with local data. In addition to the public datasets available online, users may select a local folder to which their own preprocessed LiDAR datasets can be loaded.

As can be observed in Figure 1, the structure of *ViLMA* is divided into two different stages: the *Preprocessing Stage* and the *Interactive Stage*. The *Preprocessing Stage* takes place off-line on the server-side and/or on the client-side. Datasets preprocessed on the server-side are intended to be public and accessed online through the Apache server; meanwhile, datasets preprocessed on the client-side are loaded directly by *ViLMA* from the user's local disk. During the *Preprocessing Stage*, points from the original LiDAR datasets are rearranged and stored avoiding data redundancy in order to support efficient, multi-resolution and out-of-core techniques together with data queries based on spatial restrictions. This is achieved through HLT and TGPT data structures, in addition to lossless compression methods applied over the preprocessed data, which will be discussed further in the following sections.

The *Interactive Stage* takes place online on the client-side, where users are able to visualize, interact and analyse LiDAR point clouds through their web browser. Regions of interest (ROI) can be requested from the entire point cloud using geographic coordinates. The use of ROIs has several performance implications that are further discussed in Section 6.4.

ViLMA includes several options for the visualization and filtering of the point clouds based on LiDAR properties such as classification, intensity, return number or RGB. It also incorporates measuring tools, such as distance between points, areas on an orthographic projection, fully 3D surface areas and complex volume measurements comprising a polygonal contour, irregular bottom surface and orthographic projected top surface.

[4]Apache server website: https://httpd.apache.org/

Although the objective of *ViLMA* is not to provide an ultra-realistic representation of the point clouds, three image enhancement techniques have been included in order to improve the quality of the images in terms of object recognition, object definition and depth perception. Circular points, dynamic point size and Eye-Dome Lighting (Boucheny 2009), have been implemented through the programmable components (shaders) of the graphic processing unit (GPU). More information about this kind of image enhancements can be found in Gross and Pfister (2007).

4. Multi-Resolution Out-Of-Core Data Structures

The interactive visualization of massive LiDAR point clouds, exceeding available memory resources, demands the use of multi-resolution out-of-core techniques. Our proposal is focused on the following factors: minimizing the consumption of both, system memory (RAM) and GPU memory (VRAM), leveraging of communications between the client browser and the data server, and reducing disk storage usage on both client and server-sides. In this section, we present the two main data structures used for reaching those goals, HLT (Section 4.1) and TGPT (Section 4.2). Section 4.3 is dedicated to explain the fundamentals of the rending process using the cited data structures.

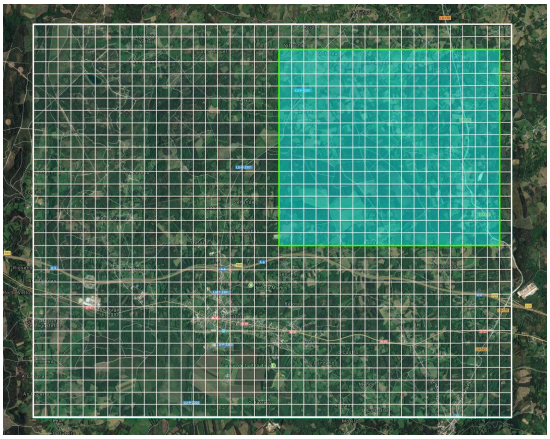
4.1. *Hierarchically-Layered-Tiles (HLT)*

Traditionally, in computer graphics, multi-resolution approaches involve the creation of several different detailed versions of the same 3D model, which implies data redundancy among all model versions (further information about multi-resolution models can be found in Tomas Akenine-Möller (2008) and Gross and Pfister (2007)). Our proposal avoids data redundancy in order to achieve the above factors. There are no static, precomputed, multi-resolution models of the point clouds, but a specific rearrangement and storage of the points, aimed to act as separate pieces with which to create the different multi-resolution models at runtime joining those pieces as necessary.

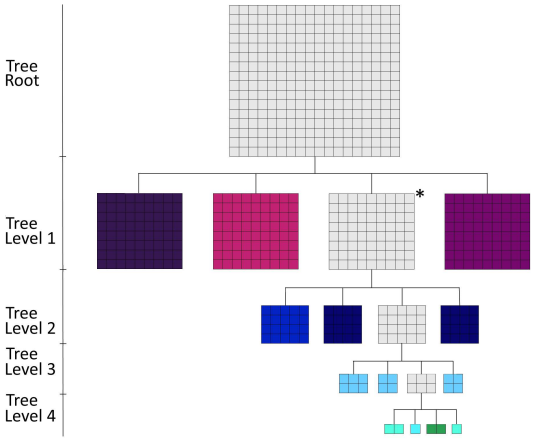
To achieve this, the bounding box of the point clouds is divided into T equally-sized tiles forming a regular grid (see Figure 2(a) as an example of gird of tiles). Points are distributed in the tiles using their geographic position. For each tile in the grid, points are scattered into L layers, creating a heap of layers of different point densities. An input parameter, called downsampling factor (df), defines the percentage of points that are scattered in each layer of a tile. The points in each layer are uniformly distributed over the surface. Given a tile t containing a total amount of N_t points, the number of points in the layer l is defined by:

$$N_{t,l} = \begin{cases} N_t \times (1-df)^{L-l} \times df & l > 1 \\ \\ N_t \times (1-df)^{L-1} & l = 1 \end{cases} \tag{1}$$

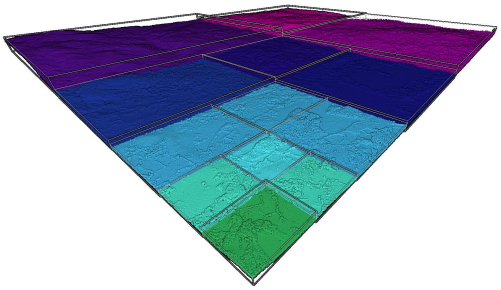
No point is repeated in more than one layer, so this, the superposition of the points of all layers from a given tile, is an identical representation of the original tile. Thus, LOD representation of a tile with a level LOD_l consists of the overlapping of the points of its layers, from layer 1 to layer l:



(a)



(b)



(c)

Figure 2. Construction of a TGPT from an arbitrary ROI and its posterior usage for computing the different levels of detail of the image. (a) Illustration of an ROI defined by user (inner shaded rectangle, overlapping 16×18 tiles) over a dataset grid (outer rectangle, 32×39 tiles). (b) TGPT structure generated during the multi-resolution process fitting the ROI shown in (a). (c) Point cloud rendered by *ViLMA* obtained from the TGPT shown in (b).

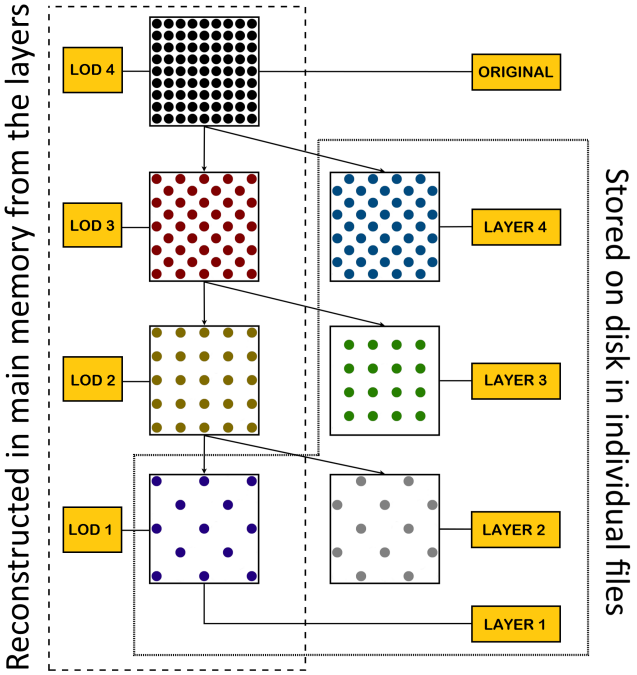


Figure 3. Layer generation of a single tile carried out during the *Preprocessing Stage*. Starting with the original point set (upper square) four layers are generated (labelled squares).

$$LOD_l = \bigcup_{i=1}^{l} L_i \tag{2}$$

where L_i is the layer i of a given tile.

Figure 3 shows an example of layer generation. In the example given, a $df = 0.5$ was used for simplification purposes; hence, all subsets contain half of the points of their parent set. The top square (labelled as *Original*) represents a tile from the grid containing all the points. The points are split into two subsets: the subset on the right is stored in an individual file labelled as *Layer 4*; while, the subset on the left is split again into a further two subsets, repeating the same process. The subset on the right is then stored in a separate file labelled as *Layer 3*, while the subset on the left is split again. Finally, the last two subsets are stored in files with the labels *Layer 1* and *Layer 2*, respectively. All files generated during this stage are lossless compressed, adhering to a method that will be discussed in Section 5.3. Point subsets labelled as: *LOD 1*, *LOD 2*, *LOD 3* and *LOD 4* are reconstructed in main memory during run-time from the points contained in the different layers and they work as the actual 3D models used during the rendering process.

The layering approach of the HLT avoids any kind of data redundancy, which implies a considerable reduction in memory and disk storage usage on both server and client sides, as well as a reduction in network bandwidth requirements.

Following the example above and considering $N_t = 1000$, in a traditional approach to multi-resolution, each LOD would have associated a precomputed 3D model stored on server-side, in this case, 4 files containing 1000, 500, 250 and 125 points. The amount of points stored after creating the different LODs is 1.875 times the original amount

of points, while with our approach the amount would always be the same. This level of redundancy may vary depending on the number of LODs used, the type of tree structure used (quadtree, kdtree, octree, ...) or how many points are selected for each LOD. In this small example data increase almost by 2, for very large and dense point clouds, managed by deep and complex tree structures, the level of redundancy may be notable.

4.2. *Tile Grid Partitioning Tree (TGPT)*

The management and handling of the tiles into which a point cloud is divided is done though the tree-like structure, TGPT. Unlike other classic data structures, such as quadtrees or octrees, which may have been entirely precomputed and stored in disk (on server or client-side), the TGPT is not a static, precomputed structure but a structure generated on the client-side, at runtime as needed and always fitting a given ROI. The TGPT is stored in the client RAM. Once an ROI is defined, the TGPT is initially built, creating the root node which represents all tiles overlapping the ROI. The layer 1 of each of those overlapped tiles is retrieved from server. This initial set of points is the lowest resolution reconstruction of the point cloud within the ROI. The information contained in the layer 1 includes the number of points in the remaining layers and the minimum and maximum values of the coordinates X, Y and Z in each tile. Using these coordinates, a bounding box (the minimum volume that wraps the entire set of points) is created for the root node. This is the initial and most basic state of the TGPT. On the basis thereof, the tree grows as needed, depending on the decisions of the multi-resolution techniques. We should stress here that the nodes of the TGPT do not store any points at all, they only stand as a set or subset of tiles from the ROI storing the indices that indicate the range of tiles they contain, the bounding box that encloses those tiles, and other minor variables.

All nodes of the TGPT are created as needed during the rendering process using a criterion based on the screen projections of their bounding boxes (this will be further detailed in Section 4.3). Both, the number of new children created for a given parent node and the subsets of tiles assigned to each child, depend on the number of tiles in the parent. The width and length (in tile units) of the parent node are divided by two in order to delimit subsets as proportionally as possible in terms of their number of tiles. For example, taking the node marked with an asterisk in Figure 2(b) as a reference, this node contains a set of 8×9 tiles and it is split into four children, each one containing a unique subset of tiles: two subsets of 4×5 tiles and another two subsets of 4×4. Occasionally, this could lead to split a parent node into only two children. At the tree level 4 of Figure 2(b), the nodes containing a set of 1×2 tiles would only be split into 2 children with one tile each.

4.3. *Multi-Resolution Out-Of-Core Rendering Techniques*

The HLT, along with the TGPT, are the core elements of the multi-resolution, out-of-core technique used by *ViLMA* and it has two main steps. The first is the creation of an LOD-distribution-list. Traditionally, multi-resolution approaches use some kind of point limit or point budget (PB) to avoid consuming all available memory or surpassing computational capabilities. Following the same approach, the second step, is the calculation of an LOD for each node of the list, trying to use as many points as possible without surpassing a defined PB. Higher budgets produce better image

quality, to the detriment of performance and vice versa; hence, the choice of a value for the PB is a subjective task focused on finding a balance between performance and quality. This type of balance is discussed broadly in the literature (Debattista *et al.* 2017).

In the first step, view frustum culling (the process of removing objects that lie completely outside the camera of the scene) is used in order to determine visible or partially visible nodes. If the node is a leaf node, it is put in the LOD-distribution-list for the subsequent computation of its LOD. If it is not a leaf, its bounding box is used to compute the number of pixels projected on screen. If the projected area is larger than a system-defined percentage of the screen size, the node is considered to be too close to the camera's perspective, and the process continues through its child nodes. The TGPT is constructed as needed, so if the current node has children but they are not currently existent in the tree, they are created immediately. If the node is not too close, it is put in the LOD-distribution-list. As a result, at the end of the process the LOD-distribution-list contains all visible tiles, grouped in nodes.

In the second step, computing each LOD individually for each tile is not a viable option in terms of performance and scalability, due to the large number of tiles into which some datasets may have been divided. Instead of using individual tiles, LOD is computed over groups of tiles; thus, the nodes collected in the LOD-distribution-list are used for that task. When an LOD l is assigned to a node, this means that all tiles contained therein must be displayed with the given level l. All the point layers required to build the LOD are retrieved from the server, unless they are already in the memory or in the browser cache. The data retrieval process is further detailed in Appendix A.

The LOD of each node in the LOD-distribution-list is determined by the projection on screen of its bounding box and the number of points contained in the different layers of its tiles. The objective is to assign to each node the highest possible LOD, as long as the number of points displayed in the node is equal or inferior to the number of pixels projected on screen by its bounding box. This method attempts to avoid situations where too many points are drawn in the same area of the screen, thus causing the loss of image quality due to an excessive overlapping effect. An LOD is assigned to each node with a view to providing more detail in nearby nodes and less detail in those further away, while not exceeding the PB.

Figure 2(a) shows an arbitrary ROI (inner shaded rectangle, 16×18 tiles) overlapping a dataset grid (outer rectangle, 32×39 tiles). Figure 2(b) shows a TGPT structure built during the multi-resolution process fitting the specific ROI where each tree node represents a subset of tiles contained in the ROI. The final 3D representation of the multi-resolution process can be observed in Figure 2(c). For explanation purposes, white bounding boxes are displayed over the point cloud rendered. These boxes are the nodes selected for the LOD-distribution-list. Additionally, colours were used as reference to represent each LOD in the 3D scene and in the TGPT, to make it easier to identify the nodes in the TGPT and their corresponding bounding boxes in the scene.

5. Performance considerations

While traversing a point cloud very quickly or when the camera makes very abrupt movements, the detail of areas not loaded in memory could pop up with a slight delay, showing gaps or no points for a short period of time. Thanks to the use of fixed-size

GPU buffers, VRAM consumption can be kept very low and constant; however, this implies that the buffers must be updated regularly to adapt to the camera movements. The update process can be particularly demanding, especially for a high PB and a low GPU memory bandwidth, so the buffers are not updated in every single frame but once every 0.25 seconds, which can lead to the aforementioned gaps. We should remember here that only a WebGL-compatible web browser is required for running the application. All GPU-related techniques are carried out through this graphics API.

A number of important decisions must be made regarding how the datasets are preprocessed, as they directly impact on the performance, namely: the size of the tiles and the number of layers per tile (Sections 5.1 and 5.2). These are subjective decisions that must aim to strike a balance between performance and quality (Debattista *et al.* 2017). In this section, we also address an additional non-subjective performance issue regarding the compression of the data (Section 5.3).

5.1. *Tile size*

The size of the tiles affects the accuracy of the view frustum culling techniques (Tomas Akenine-Möller 2008), the efficiency of data queries based on spatial restrictions, and the proper use of the browser cache.

- *View Frustum Culling.* In a 3D scene, view frustum culling techniques are implemented to detect partially or fully visible objects from camera's perspective in order to send all detected objects to the rendering process, thus increasing the GPU performance by discarding the non-visible ones. Tiles with a small size form fine-grained grids; hence, these types of techniques can discard larger non-visible areas by detecting more tiles in a fine-grained grid than in a coarse-grained one.
- *Data queries based on spatial restrictions.* As commented in previous sections, the use of an ROI allows the computing resources to be focused in a limited area. All tiles located outside the ROI can be completely discarded. In a similar way to what happens with the view frustum culling, tiles with a small size form fine-grained grids so that tiles can be discarded more accurately.
- *Browser cache.* Web browsers have a special memory space reserved in the client disk called the Browser cache. This is used to store downloaded files so they can be reused later instead of being retrieved again from server, thus speeding up the web page loading. Each web browser has a maximum file size allowed when storing files in cache. Depending on the version of the browser, the maximum size may vary from 5 MB, on some mobile browsers, to 25 MB, on desktop versions. Files not stored in cache must be retrieved from the server each time they are requested. The use of small tiles reduces the size of files generated during the preprocessing stages, so the requirements of the cache are more likely to be met.

Although the choice of a small tile size has great performance benefits, it has a counterpart in memory consumption. During the rendering process, each tile must be handled and managed separately, which implies to create one object in memory for each tile. These objects have a very small footprint in memory but if the number of tiles is too large, the memory consumption may not be suitable for the requirements of certain users. Thus, the choice of tile size must be balanced between memory consumption and the benefits described above.

We have measured the increase in RAM consumption when loading an arbitrary dataset using different tile sizes. As a base measurement, we obtain a global RAM

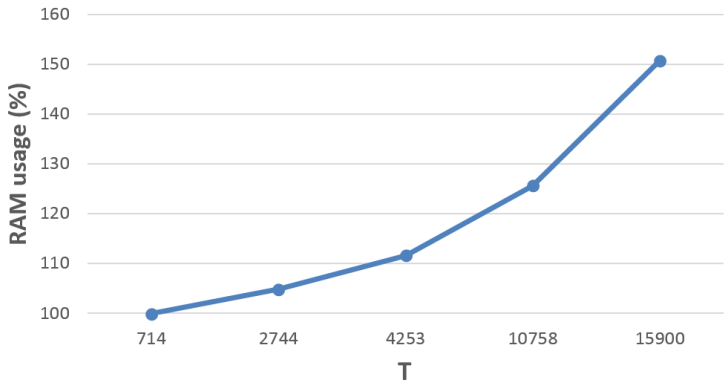


Figure 4. Memory consumption variation with respect to the increase in the number of tiles.

consumption of 250 MB with 714 tiles. Figure 4 shows how the RAM consumption increases along with the number of tiles created (T).

The number of tiles generated for a given dataset is not only determined by the size of the tiles, but also by the extent of the dataset itself; therefore, there is no ideal tile size, since the final number of tiles depends on the extent of the dataset and the balance of quality and performance desired by the users.

5.2. *Number of layers per tile*

One key point for all multi-resolution applications is the suitable creation of the different resolution models that are going to be used during the execution of the software. Resolution transitions between consecutive levels (either to increase or to decrease detail) should be carried out smoothly, avoiding abrupt changes and popping effects.

In the example shown in Figure 3, a $df = 0.5$ was selected solely for simplification purposes, but in a production environment, this value may be too high. The change from a given LOD l to the next one $l+1$ entails doubling the number of points, which may be visually too abrupt. df values around 0.25 produce better results, obtaining softer transitions between consecutive levels of detail. On the other hand, lower df values produce more layers per tile. For instance, given a tile with 1000 points, if we need to generate a layer 1 with around 25% of the points (250 points) and a $df = 0.5$, using Equation 1, the result is that at least 3 LOD must be used: $1000 * (1-0.5)^{3-1} = 250$. And using $df = 0.25$, then 6 LOD would be required: $1000 * (1-0.25)^{6-1} \approx 237$. The increase in the number of layers also increases the number of data retrievals, so two equal tiles divided into a different number of layers would take different times to render, even when displayed with the same detail.

Using the same dataset as in Figure 4, we have measured the wait time when zooming in close to the ground using 8 layers per tile and 18 layers per tile. We define *wait time* as the sum of retrieval time and load time. The retrieval time is defined as the time required to download all necessary data from the server, while the load time is defined as the time spent reading and decoding downloaded data and creating any other elements required to handle them. In the second case (18 layers per tile), the wait time was 1.6 times the time spent in the first case (8 layers per tile). This difference becomes almost negligible when retrieving the data from cache, as the retrieval times are zero in both cases, and load times are almost equal. Once again, as described for choosing the tile size, the choice of df must aim to strike a balance between LOD

Table 1. Hardware specifications.

Platform	O.S.	CPU	GPU	RAM*	VRAM*	Display	Bandwidth**
Client PC	Windows 7	Intel Core i7 4790	GeForce Titan X	32	12	2560×1440 @144Hz	90 (Wired)
Client Mobile	Android 7.0	Tegra K1	Tegra K1	2 (Unified)	-	1920×1200 @60Hz	65 (Wi-Fi)
Server	CentOS 6.7	Intel Xeon E5-2603 v3	-	64	-	-	-

Values measured in: $*GB$, $**Mbps$.

quality and retrieval times.

5.3. *Compressing the point layers*

LiDAR datasets may reach huge file sizes, demanding large storage capacity and high bandwidth in a web application environment when needing to send parts of or the entire point cloud through the network, or when they must be stored in local disk storage.

LiDAR data are usually stored in LAS[5] format, a standard in the field of LiDAR solutions, although there are other formats, such as PNT, CSV or XYZ. The LAS format provides properties such as (X, Y, Z) coordinates, intensity, pulse return information, scan direction flag and point classification, among others. Nevertheless, some of these properties are often not useful from the point of view of visualization or geospatial measurements.

We have developed our own LiDAR compression format (called *LZ* for LiDAR-Zipped) focused on providing suitable support for the HLT structure. LiDAR data are lossless compressed minimizing both client and server disk usage and reducing data retrieval times.

Currently, the best lossless compression methods for LiDAR data are LASzip (Isenburg 2015) and LAS Compression software, which implements the method presented in Mongus and Žalik (2011). LASzip (LAZ file format) is considered as the standard in LAS compression and it outperforms all other general-purpose techniques. With our lossless compression format, the objective is not to propose an alternative to LAZ but to efficiently support HLT structure. To achieve this, three main tasks are carried out in order to generate each *LZ* file. First, data cleaning, where LiDAR properties not used by our framework are discarded. Second, delta encoding, where LiDAR properties are stored in the form of differences (deltas). And third, GZIP[6] compression, where all generated data are compressed using this software. As a result of these three steps, LAS files are reduced by around 88%. The three tasks are further detailed in Appendix B and a comparison between LASzip and our compression method is included in Section 6.

6. Experimental Results

In this section, we evaluate the performance of the data structures and techniques presented in previous sections using our own web-based visualization software *ViLMA*. Performance is presented and analysed in terms of memory consumption, wait times,

[5]LAS file format standard definition: https://www.asprs.org/
[6]GZIP website : https://www.gnu.org/software/gzip/

Table 2. Software specifications.

Type	Name	Version
BackEnd	Apache HTTP Server	2.4.28
PC Browser	Google Chrome (64 bits)	58.0.3029.110
Mobile Browser	Google Chrome	58.0.3029.83

Table 3. LiDAR datasets used and their information regarding the *Pre-Processing Stage*. N: Number of points. FS: Total file size of the dataset (original LAS files). FS_{LZ}: Total file size of the dataset (preprocessed files). *Ratio*: Compression ratio of the preprocessed files. TS: Tile size. LPT: Number of layer per tile.

Dataset	N*	FS**	FS_{LZ}**	*Ratio*	TS***	LPT
PNOA	28	802	118	14.71%	500×500	16
San Simeon	17.7	561	132	23.52%	400×400	36
Volcano	0.55	14.6	2.78	19.04%	200 ×200	24

Values measured in: $*Billion$, $**GB$, $***Meters$.

frames per second and multi-resolution image quality. Additionally, we have included a brief analysis of our compression method, *LZ*, and finally a performance comparison with *Potree*[7], which is the most similar tool to *ViLMA* found in the literature. The main specifications of the platforms and the software used during the tests are described in Table 1 and Table 2.

ViLMA was tested in several browsers; nevertheless, for the sake of clarity and simplicity, only the results obtained with Google Chrome are shown, as this was the browser with the best overall performance. Note that the objective of this evaluation is not to compare web browsers but to present the performance of *ViLMA*.

Table 3 lists the three datasets used to evaluate *ViLMA* along with the number of points of each dataset (N), their original and compressed file size using our proposal (FS and FS_{LZ}, respectively), the compression ratios obtained (*Ratio*) defined as ($Size_{compressed}$ / $Size_{uncompressed}$) and expressed as a percentage, the tile size (TS) and the layers per tile (LPT) used to preprocess each one of them. The *PNOA* (National Plan of Aerial Orthophotography, Spain) dataset is available in the Spanish GIS database (IDEE) (Infraestructura de Datos Espaciales de España IDEE). Specifically, we have selected the region of Galicia, which contains around 28 billion points. The airborne LiDAR survey of the selected area was taken with a point density of 0.5 $point/m^2$. The *San Simeon* dataset was taken from the region of San Simeon, California - Central Coast, and it contains 17.7 billion points, with a point density of 22.06 $points/m^2$ and it is available at OpenTopography[8]. Finally, the *Volcano* dataset contains 0.55 billion points, with a point density of 13.71 $points/m^2$ being available at OpenTopography[8].

During the tests, the point budget (PB) was varied taking values of 1, 2 and 4 million points. These quantities were chosen as they can be easily handled by most systems, regardless of whether they are low-end or high-end, allowing good results to be achieved in terms of performance, while obtaining fairly good representations of the original point clouds.

We should stress here that, in Section 6.1 and Section 6.2, the full datasets are used, with the extreme case of *PNOA* reaching up to 28 billion points.

[7]Potree website: http://potree.org/

[8]OpenTopography website: http://www.opentopography.org/

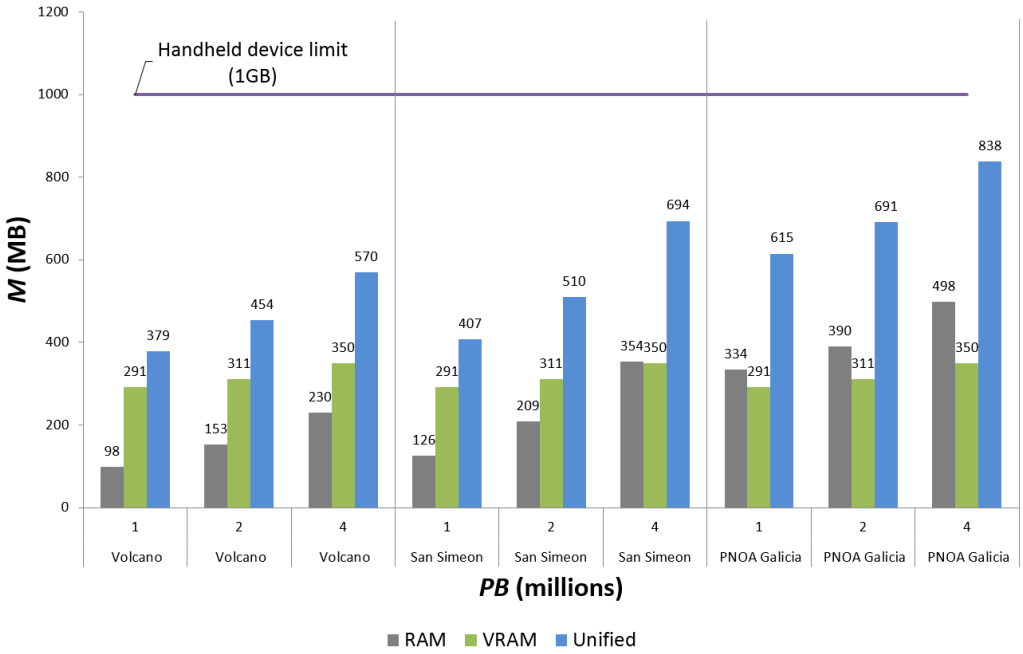


Figure 5. RAM, VRAM and Unified memory (RAM + VRAM) consumption during the performance tests for different point budgets.

6.1. *Memory consumption*

Figure 5 shows the memory consumption observed for rendering each dataset using three different *PBs*. Both, the RAM and VRAM values where taken from the Task Manager provided by the desktop version of Google Chrome. Unified memory values represent the memory consumption on the tablet and they are further explained in Section 6.1.3.

6.1.1. RAM

The RAM consumption rises along with the *PB*; this behaviour is expected, as the application must store more points and manage more point layers. During the tests, consumption ranges between 98 MB and 498 MB. Taking into account that most current desktop PC configurations are equipped with 8 or more GB of RAM, we can consider the RAM consumption of *ViLMA* in desktop devices as notably low.

6.1.2. VRAM

Each point property, such as RGB colour or intensity, has its own buffer in the client GPU but, as long as a property is not necessary for rendering purposes, it will not be sent to the GPU, which helps to leverage the VRAM consumption. By default, point clouds are rendered by *ViLMA* as height maps based on the *Z* coordinate of their points, so properties such RGB or intensity are kept in RAM but not in VRAM.

Throughout the entire execution of *ViLMA*, GPU buffers have a fixed size that always matches the current *PB*; therefore, while the *PB* does not change, the use of GPU memory remains constant. The use of fixed-size buffers also implies an equal consumption of VRAM across different datasets as long as the same *PB* is used. This can be clearly seen in the results of the three datasets in Figure 5, where the VRAM

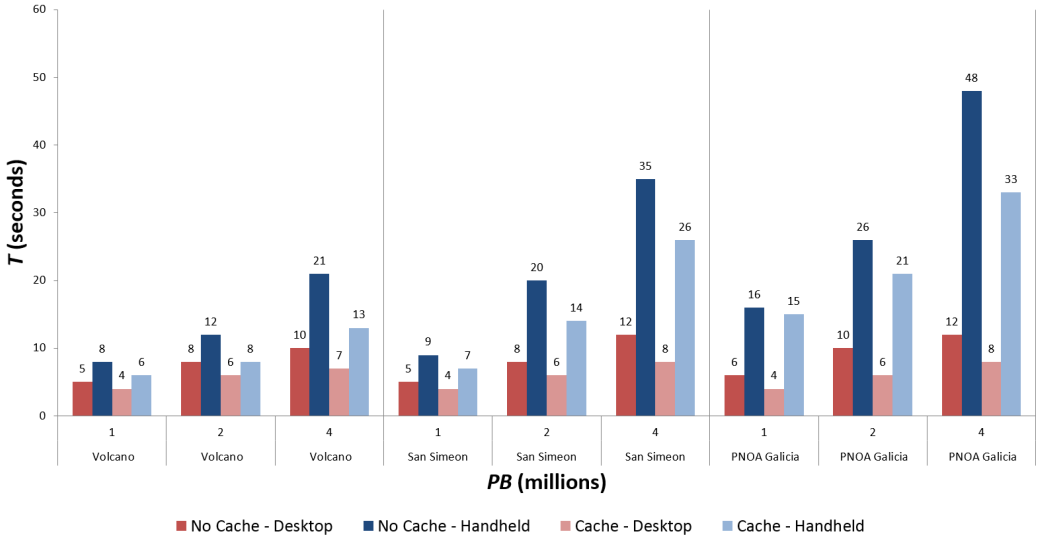


Figure 6. Wait times (retrieval time + load time) obtained among three different datasets with and without browser cache.

consumption of each PB is equal across all of them.

We should stress here that the VRAM usage is always constant throughout the entire use of the application. VRAM may vary only if measurement tools are used, since new elements derived from those measurements, such as the triangulation of a surface, are stored in VRAM once they are created. This is a critical optimization element, given that other multi-resolution approaches increase the VRAM consumption as new resolution levels are loaded in the GPU.

As even current low-end GPUs are equipped with 2 or more GB of VRAM, the VRAM consumption of *ViLMA* in desktop devices can be considered moderately low as during the tests it ranges between 291 MB and 350 MB.

6.1.3. Unified memory

Mobile devices have a unified memory architecture, meaning that there is only a single main memory storage unit shared between the CPU and GPU. Any device running *ViLMA* will use the same JavaScript code with the same data structures and data formats. This implies that loading the same point cloud with the same PB will consume the same amount of memory (RAM and VRAM), irrespective of the device used, with the only exception of a small percentage of VRAM that depends on the device's screen resolution. In WebGL, graphic elements, such as textures and framebuffers (for further information, see Tomas Akenine-Möller (2008)), are used as part of the rendering process. These elements have a footprint in VRAM which is directly proportional to the screen size of the device used. In our case, our tablet uses 10 MB of VRAM less than the desktop PC, as its screen resolution is lower. The *unified* columns of Figure 5 represent the tablet's memory consumption, and they are simply the addition of the RAM and VRAM values minus the aforementioned 10 MB difference.

Despite the tablet being equipped with 2 GB of unified memory, this is not completely available for user applications. We observed that, on average, only 1 GB of memory is available. The free memory may vary depending on the previous usage of the device, the background tasks of the operating system or other applications cur-

rently running. In Figure 5 the memory limit is marked with a horizontal line. Point clouds and PBs with consumption values close to the limit may be feasible but they depend on the current state of the memory. As can be observed, we were able to load the three datasets without any problems, even when using a PB of 4 million points.

6.2. *Wait times*

Figure 6 shows the wait times (data retrieval time + data load time) observed from the moment when a full dataset is selected until it is displayed on screen using just a top-view camera. For testing purposes, no type of ROI has been used in order to analyse the most resource demanding scenario for each dataset. Times were taken using a PB of 1, 2 and 4 million points, with and without data caching.

The bandwidth values shown in Table 1 are not theoretical speeds, but the maximum values obtained after performing several network speed tests on both client platforms. We observed that the Wi-Fi performance is 28% lower than the wired connection and; therefore, this difference should be taken into account in the results of this section.

6.2.1. Data not in cache

Considering the wait times for first-time retrievals (the data are not in the browser cache) we obtained between 5 and 10 seconds for the *Volcano* dataset on PC platform and between 8 and 21 in the tablet. Between 5 and 12 seconds for the *San Simeon* dataset on the PC and between 9 and 35 on the tablet. For *PNOA* datasets, times between 6 and 12 seconds were obtained on PC and between 16 and 48 on the tablet.

All times obtained for the desktop PC were considerably lower, being above 10 seconds in just a couple of cases. Despite the differences in computing power and network speeds between the two systems, times obtained on the tablet are higher than on PC but also acceptably low, with the only exception of *PNOA* and *San Simeon* with a PB of 4 million points. As the memory consumption starts to reach the memory limit, the general performance of the tablet highly decreases which rises the time needed to read and prepare the retrieved data. We should stress here that in the extreme case of use (28 billion points), and despite of the increment in times, we were able to load said dataset on the tablet.

6.2.2. Data in cache

The size of all the data retrieved from the server by *ViLMA* is, whenever possible, small enough to be cached by all browsers, both desktop and mobile. When the data are cached, they are retrieved from local storage, so the retrieval times are zero, significantly reducing the wait times in all cases. The positive effect of data caching can be clearly observed in the results obtained for the two devices. The three datasets are loaded, between 4 and 8 seconds with any PB on the PC platform and between 6 and 33 on the tablet.

6.3. *Interactive visualization*

Figure 7 shows the satellite image of a small area of the *San Simeon* dataset and three renderings of its point cloud using different PBs. The images are zoomed close to the ground to better appreciate the quality of the multi-resolution techniques and the difference between the three selected PBs.

(a) (b)

(c) (d)

Figure 7. Small part (∼1.5 km^2) of the *San Simeon* dataset (803 km^2) rendered by *ViLMA* using different point budgets: (a) Satellite image of the zoomed area. (b)-(d) Rendered images using 1, 2 and 4 million points respectively.

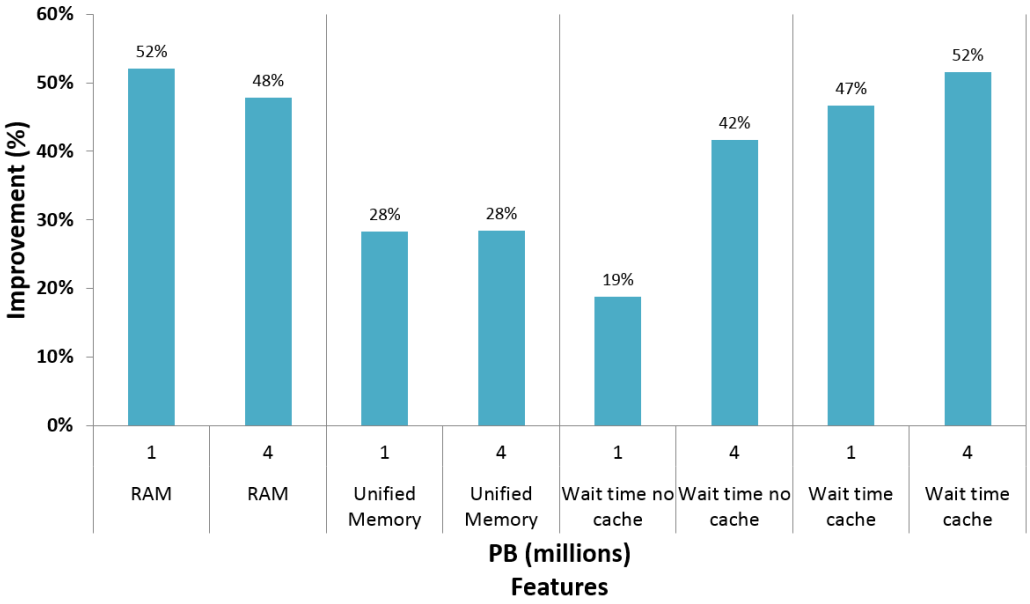


Figure 8. Performance comparison between loading the full dataset of *PNOA* and loading only an ROI from it.

On the desktop PC, FPS benchmark results were constant at 144 FPS for all datasets and PBs. The refresh rate of the screen used in the tests was 144 Hz, which explains why the FPS were locked at 144. On the tablet, for all datasets, we attained a stable rate of 60 FPS using 1 million points, up to 55 FPS with 2 million points, and up to 38 FPS for 4 million points.

6.4. *Performance improvements when using an ROI*

In *ViLMA*, the use of an ROI can be considered a before-load spatial restriction which decreases the memory consumption and the number of data retrievals. Other approaches achieve similar results by allowing users to manually crop the point cloud after-load or by cropping the point cloud beforehand in a pre-processing stage. In the first case, if the cropping is done after having loaded the point cloud, many unnecessary data could be retrieved or loaded, which could be a problem in contexts with small amounts of memory. In the second case, users are limited to the use of previously cropped point clouds, which could not fully meet their requirements.

We have analysed the differences when using an ROI on a massive dataset like *PNOA*. The chosen ROI was the city centre of Santiago de Compostela (Spain), with a total amount of 19 million points. Figure 8 shows the improvements when using PBs of 1 and 4 million points on the tablet. Given the adaptation of the TGPT to the size of the ROI, there is a notable reduction in the RAM consumption, which is 52% with $PB = 1$ and 48% with $PB = 4$. The VRAM consumption is the same in both cases, so the unified memory is also reduced, but not at the same degree as the RAM. Unified memory is reduced by around 28%. With regard to wait times, when using $PB = 1$ million, these were reduced by 19% without cache and by 47% with cache. For $PB = 4$ million, times were reduced by 42% without cache and by 52% with cache.

These improvements benefit both platforms, PC and tablet, but they have a special

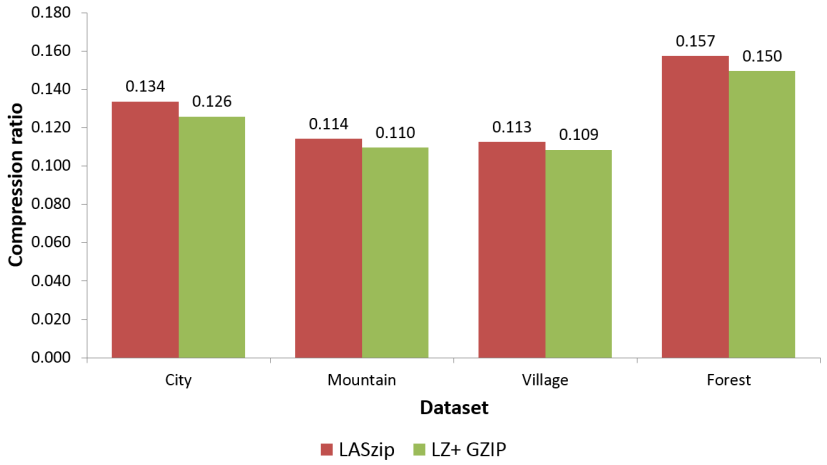


Figure 9. Compression formats comparison.

relevance in the latter due to its hardware limitations. The point cloud inside the ROI can be displayed on the tablet with shorter wait times, using significantly less memory and showing much more detail, as the points are distributed in a smaller and highly delimited area.

6.5. *Compression ratio*

Results from a comparison between our *LZ + GZIP* compression method and LASzip can be observed in Figure 9. The efficiency of both compression methods varies depending on the topology and characteristics of the processed point clouds, so four sample files (city, mountain, village and forest) were taken for the comparison with a view to selecting different point distribution patterns. The samples were taken from the *PNOA* dataset.

As can be observed, compression ratios ($Size_{compressed}$ / $Size_{uncompressed}$) obtained with *LZ + GZIP* are slightly better than LASzip. The objective of *LZ + GZIP* is not to serve as an alternative for LASzip but to support our multi-resolution, out-of-core techniques. Considering the rest of the performance results shown in this Section, the support of *LZ + GZIP* is entirely suitable for *ViLMA*.

6.6. *Comparison with* Potree

In Section 2, several web-based visualization applications were cited. The technology presented by *Dielmo3D* is a proprietary technology and was tested through *LiDAR Online*[9]. This web application allows data to be obtained on demand through data queries based on spatial restrictions; nevertheless, all rendered point clouds are always displayed with around 1 million points, irrespective of the size of the area defined by the data queries, and the points rendered do not change as the camera moves around the scene. The design of *GVLiDAR* was focused on providing high-performance rendering of full-detailed point clouds so a multi-resolution, out-of-core approach was discarded. The designs of *Potree* and *Megatree*, follow a very similar client-server structure and

[9]LiDAR Online website: http://lidar-online.com/tools/maps

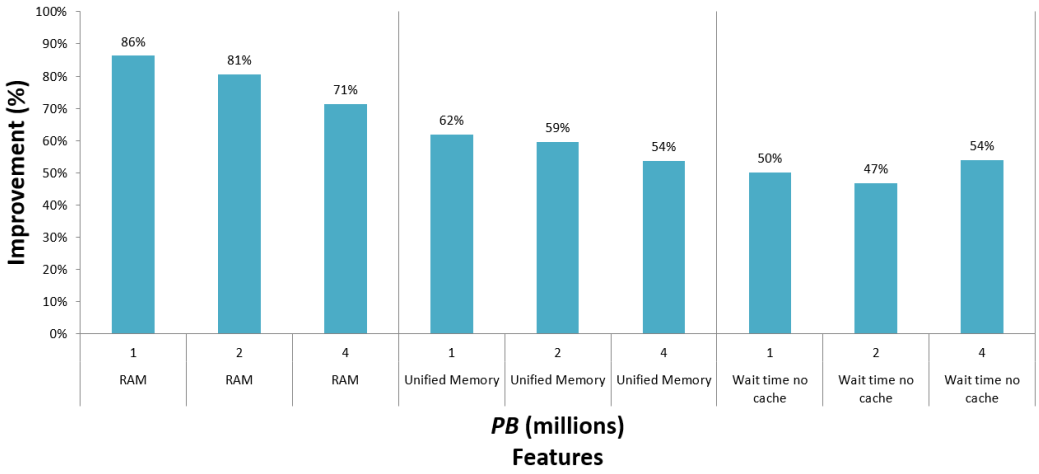


Figure 10. Comparison between *ViLMA* and *Potree*.

their performance relies on the use of multi-resolution out-of-core techniques supported by an octree structure. Given the aforementioned factors, in this section, a comparison between *ViLMA* and *Potree* has been carried out, as among the current freely available, web-based visualization applications, this is the most similar to ours and it is one of the best known.

The comparison is focused on memory consumption and wait times, two measures strongly related to the different data structures used by the applications: HLT and TGPT in *ViLMA*, and octree in *Potree*. The dataset used was *San Simeon* (17.7 billion points). Although on the *Potree* website it is indicated that the last stable version is 1.3, the release candidate 1.5 has been tested, as better results in memory consumption were reported. For a fair comparison, both *ViLMA* and *Potree* use the RGB values of the points for rendering the scene, which implies a slight increase in VRAM for *ViLMA* compared to what was shown in Figure 5, as already explained in Section 6.1.2. Figure 10 shows the percentage of improvement of *ViLMA* over *Potree*, analysing RAM and unified memory (RAM + VRAM) consumptions, and wait times.

In all cases regarding memory consumption, results are better using *ViLMA*. Between 71% and 86% lower on RAM consumption and between 54% and 62% on unified memory consumption. The multi-resolution approach of *Potree* consists in progressively loading several subsections of the point cloud with different resolution levels. This increases the amount of both RAM and VRAM used, as the user moves the camera across the point cloud. In *ViLMA*, the GPU buffers are immutable and constantly reused; hence, over time, *Potree* ends up consuming more VRAM than *ViLMA*, where consumption remains constant. In addition, *ViLMA* uses a non-redundant, multi-resolution approach which leaves a smaller footprint on RAM. These differences lead *Potree* to ultimately reach memory limits, such as the 1 GB of unified memory on the tablet or 4 GB of RAM security limit of Google Chrome. During the tests on the tablet, even though *Potree* was able to load the *San Simeon* dataset using a PB of 1 million points with a very high frame rate (a stable rate of 60 FPS), the memory limit was reached quickly as soon as the point cloud was zoomed and the camera moved. *ViLMA* is not exempt from progressively increasing its use of RAM; nevertheless, with the non-redundant data nature of its approach, the increment in RAM consumption is much slower. The difference over the unified memory consumption is especially rel-

evant when considering the real values obtained for *Potree*: 1064, 1258 and 1499 MB using a PB of 1, 2 and 4 million points, respectively. This means that *Potree* could not be used on the tablet with 2 and 4 million points, as it exceeds the 1 GB limit. Even for 1 million points, the correct performance of *Potree* would depend on the memory available at the moment of use (in fact, only after rebooting the tablet, without any other use than the web browser, was it possible to load the dataset using 1 million points).

Finally, regarding results about wait times without data caching, *ViLMA* obtains much better results (between 47% and 54% lower), which greatly helps the improvement of user's experience.

7. Conclusions and future work

In this paper we have demonstrated how multi-resolution, out-of-core techniques can be implemented though non-redundant data structures, for web-based, point cloud rendering. With a rearrangement and a specific storage of the points, we are able to avoid the creation of additional and unnecessary static, precomputed elements, that are normally required in a multi-resolution context. This approach is powered by two novel elements: a non-redundant, point data structure, HLT, and a dynamic runtime-created tree-like structure, TGPT, for managing full point clouds or specific sub-regions thereof. Due to this avoidance of static-precomputed structures and the non-redundant data approach, depending on the characteristics of the point clouds and how their multi-resolution models were created, the reduction in the storage requirements on the server-side can be notable, which is especially meaningful in contexts where large amounts of LiDAR data are constantly generated and entail a significant cost of economic and technical resources. As it has been demonstrated, on the client-side, memory consumption is remarkably low, allowing massive point clouds up to 28 billion points to be loaded, even in mobile devices where memory capacity is very limited or in browsers with hard memory restrictions, such as Google Chrome. Network traffic and the use of clients' local cache also benefit from the adoption of our approach.

Acknowledgments

This research was supported by Xunta de Galicia under the Consolidation Programme of Competitive Reference Groups, co-founded by ERDF funds from the EU [Ref. ED431C 2017/04]; Consolidation Programme of Competitive Research Units, co-founded by ERDF funds from the EU [Ref. R2016/037]; Xunta de Galicia (Centro Singular de Investigación de Galicia accreditation 2016/2019) and the European Union (European Regional Development Fund, ERDF) under Grant [Ref. ED431G/01]; and the Ministry of Economy and Competitiveness of Spain and ERDF funds from the EU [TIN2016-75845-P].

The LiDAR datasets used in this article belong to:

- LiDAR-PNOA[10] data repository. Provided by ©Instituto Geográfico Nacional de España.
- PG&E Diablo Canyon Power Plant (DCPP): San Simeon, CA Central Coast[11]

[10]LiDAR PNOA web site: http://pnoa.ign.es/presentacion

[11]OpenTopography web page of the San Simeon dataset: DOI: http://dx.doi.org/10.5069/G9CN71V5

and Sunset Crater Volcano National Monument, AZ[12]. This material is based on LiDAR Point Cloud Data Distribution and Processing services provided by the OpenTopography Facility with support from the National Science Foundation under NSF Award Numbers 1226353 & 1225810.

References

Al-Awami, A.K., *et al.*, 2014. Neurolines - a subway map metaphor for visualizing nanoscale neuronal connectivity. *IEEE Trans. Visualization and Computer Graphics*, 20 (12), 2369–2378.

Arrowsmith, J. and Zielke, O., 2009. Tectonic geomorphology of the San Andreas fault zone from high resolution topography: an example from the Cholame segment. *Geomorphology*, 113 (1–2), 70–81.

Boucheny, C., 2009. *Visualisation scientifique de grands volumes de données: Pour une approche perceptive*. Theses. Université Joseph-Fourier - Grenoble I.

Brunori, C.A., *et al.*, 2013. Characterization of active fault scarps from LiDAR data: a case study from Central Apennines (Italy). *International Journal of Geographical Information Science*, 27 (7), 1405–1416.

Comino, M., *et al.*, 2017. Error-aware construction and rendering of multi-scan panoramas from massive point clouds. *Computer Vision and Image Understanding*, 157, 43 – 54. Large-Scale 3D Modeling of Urban Indoor or Outdoor Scenes from Images and Range Scans, Available from: http://www.sciencedirect.com/science/article/pii/S1077314216301461.

Debattista, K., *et al.*, 2017. Frame rate vs resolution: A subjective evaluation of spatiotemporal perceived quality under varying computational budgets. *Computer Graphics Forum*, 00 (0), 1–12.

Deibe, D., *et al.*, 2017. GVLiDAR: An interactive web-based visualization framework to support geospatial measures on lidar data. *International Journal of Remote Sensing.*, 38 (3), 827–849.

Discher, S., Richter, R., and Döllner, J., 2017. *Interactive and view-dependent see-through lenses for massive 3d point clouds*. Springer International Publishing, 49–62.

Gao, Z., *et al.*, 2014. Visualizing aerial lidar cities with hierarchical hybrid point-polygon structures. *In*: *Proceedings of Graphics Interface 2014*, GI '14. Canadian Information Processing Society, 137–144.

Gobbetti, E. and Marton, F., 2004. Layered point clouds: a simple and efficient multiresolution structure for distributing and rendering gigantic point-sampled models. *Computers & Graphics*, 28 (6), 815 – 826.

González-Ferreiro, E., *et al.*, 2013. Modelling stand biomass fractions in galician eucalyptus globulus plantations by use of different lidar pulse densities. *Forest Systems*, 22 (3), 510–525.

Goswami, P., *et al.*, 2013. An efficient multi-resolution framework for high quality interactive rendering of massive point clouds using multi-way kd-trees. *The Visual Computer*, 29 (1), 69–83.

Gross, M. and Pfister, H., 2007. *Point-based graphics*. Morgan Kaufmann.

Infraestructura de Datos Espaciales de España (IDEE), 2002. Digital elevation models. http://www.idee.es.

Isenburg, M., 2015. LASzip: lossless compression of LiDAR data.

Kereszturi, G., *et al.*, 2012. LiDAR based quantification of lava flow susceptibility in the City of Auckland (New Zealand). *Remote Sensing of Environment*, 125, 198–213.

Kovač, B. and Žalik, B., 2010. Visualization of LiDAR datasets using point-based rendering technique. *Computers and Geosciences*, 36 (11), 1443–1450.

Kuder, M. and Žalik, B., 2011. Web-Based LiDAR Visualization with Point-Based Rendering.

[12]OpenTopography web page of the Volcano dataset: DOI: http://dx.doi.org/10.5069/G9K0726C

In: *2011 Seventh International Conference on Signal-Image Technology and Internet-Based Systems (SITIS)*. 38–45.

Lafarge, F. and Mallet, C., 2012. Creating large-scale city models from 3D point clouds: A robust approach with hybrid representation. *International Journal of Computer Vision*, 99 (1), 69–85.

Mongus, D. and Žalik, B., 2011. Efficient method for lossless LiDAR data compression. *International Journal of Remote Sensing*, 32 (9), 2507–2518.

Passalacqua, P., Tarolli, P., and Foufoula-Georgiou, E., 2010. Testing space-scale methodologies for automatic geomorphic feature extraction from LiDAR in a complex mountainous landscape. *Water Resources Research*, 46 (11), 1–17.

Richter, R., Discher, S., and Döllner, J., 2015. *3d geoinformation science: The selected papers of the 3d geoinfo 2014*. Springer International Publishing, Ch. Out-of-Core Visualization of Classified 3D Point Clouds, 227–242.

Rodríguez, M.B., *et al.*, 2013. Coarse-grained multiresolution structures for mobile exploration of gigantic surface models. *In*: *SIGGRAPH Asia 2013 Symposium on Mobile Graphics and Interactive Applications*, SA '13. ACM, 1–6.

Roering, J.J., *et al.*, 2013. You are HERE: Connecting the dots with airborne LiDAR for geomorphic fieldwork. *Geomorphology*, 200, 172–183.

Sallenger, A.H., *et al.*, 1999. Airborne laser study quantifies el niño-induced coastal change. *Eos, Transactions American Geophysical Union*, 80 (8), 89–92.

Tarolli, P., 2014. High-resolution topography for understanding earth surface processes: Opportunities and challenges. *Geomorphology*, 216, 295–312.

Tomas Akenine-Möller, Eric Haines, N.H., 2008. *Real-time rendering*. A.K. Peters Ltd.

Ventura, G., *et al.*, 2011. Tracking and evolution of complex active landslides by multi-temporal airborne LiDAR data: the Montaguto landslide (Southern Italy). *Remote Sensing of Environment*, 115 (12), 3237–3248.

Yan, W.Y., Shaker, A., and El-Ashmawy, N., 2015. Urban land cover classification using airborne lidar data: A review. *Remote Sensing of Environment*, 158 (Supplement C), 295 – 310.

Yuan, S., *et al.*, 2017. Feature preserving multiresolution subdivision and simplification of point clouds: A conformal geometric algebra approach. *Mathematical Methods in the Applied Sciences*, 0 (0).

Appendix A. Back-end retrievals

During *Preprocessing Stage*, layers in the same level are pre-packed together into a single file, allowing *ViLMA* to retrieve several layers at once in a single request to the server. Retrieving packs instead of individual layers helps to improve retrieval times if a very large number of layers had to be requested.

Packs of layers with small amounts of points contain many more layers than packs of layers with a large amount of points. In some occasions, especially when using an ROI, more layers than needed may be acquired when retrieving certain packs. This situation also arises when using conventional static, precomputed models; nonetheless, in our approach, as there is no data redundancy, the storage requirements are notably lower than for other multi-resolution models with high redundancy. In both cases (traditional approaches and HLT), information discarded outside the ROI can be considered as pre-cached data if requested in future uses.

Appendix B. Compressing the point layers

All LiDAR data handled by *ViLMA* is compressed in a format we called *LZ*. Three main tasks are carried out in order to generate *LZ* files: data cleaning, delta encoding and GZIP compression:

B.1. *Data cleaning*

Data *cleaning* consists in simplifying and rearranging the information stored for each point. Unused properties, such as Scan Direction Flag or Scan Angle Rank, among others, may not be included in the *LZ* files. Other properties, such as Intensity, Classification or the Return information, are adapted or modified in order to optimize their size, taking into account their function inside the application. Detailed information about this process can be found in Deibe *et al.* (2017).

B.2. *Delta encoding*

Delta encoding, also called delta compression, is a method for storing data in the form of differences or deltas (Δ) between sequential data. The properties of a given point are derived from the properties of its predecessor plus a series of differences.

Byte masks are used per point in order to specify whether properties have changed or not in comparison to the previously computed point; if they have changed, it also specifies the byte length of the delta that has to be used. By default, compressed data are generated using masks of 1 byte per point storing the geographic coordinates, the grey scale value of the intensity, the return tag and the classification of the point. If RGB values are found in the dataset or users require the inclusion of additional properties, a second byte is used for the mask. In the former case, between 1 and 9 bytes per point would be required, while in the latter, between 2 and 16 bytes would be required.

B.3. *GZIP compression*

Although general-purpose compression methods are not the best option for LiDAR data, when applied in conjunction with techniques such as delta encoding, the results obtained are highly lossless compressed files. GZIP compression[13], a general-purpose compression method based on the DEFLATE algorithm is currently used extensively in web applications and other web environments. Not only does it achieve great compression ratios, it is also supported, by default, by all the main web browsers. This means that all decompression tasks involving gzipped files are carried out automatically and efficiently by the browser. *ViLMA* only has to perform the delta decoding in order to obtain the raw point data. GZIP compression is commonly applied on-the-fly by the server (Apache HTTP Server) over files each time they are requested. The computational overhead of the compression process is greatly compensated by the improvement in the data retrieval times achieved thanks to the use of compressed files. Nevertheless, our files are pre-compressed with GZIP, so there is no additional overhead on the server.

[13]GZIP website : https://www.gnu.org/software/gzip/