PhD Thesis

Geospatial Processing and Visualization of Point Clouds: from GPUs to Big Data Technologies

David Deibe Seoane

2019

UNIVERSIDADE DA CORUÑA

Geospatial Processing and Visualization of Point Clouds: from GPUs to Big Data Technologies

---

David Deibe Seoane

PhD Thesis

October 2019

PhD Advisors:

Margarita Amor López

Ramón Doallo Biempica

PhD Program in Information Technology Research

UNIVERSIDADE DA CORUÑA

Dra. Margarita Amor López
Profesora Titular de Universidade
Dpto. de Enxeñaría de Computadores
Universidade da Coruña

Dr. Ramón Doallo Biempica
Catedrático de Universidade
Dpto. de Enxeñaría de Computadores
Universidade da Coruña

CERTIFICAN

Que a memoria titulada "*Geospatial Processing and Visualization of Point Clouds: from GPUs to Big Data Technologies*" foi realizada por D. David Deibe Seoane baixo a nosa dirección no Departamento de Enxeñaría de Computadores da Universidade da Coruña, e conclúe a Tese de Doutoramento que presenta para a obtención do título de Doutor en Enxeñaría Informática pola Universidade da Coruña coa Mención de Doutor Internacional.

En A Coruña, a de de 2019.

Asdo.: Margarita Amor López
Directora da Tese de Doutoramento

Asdo.: Ramón Doallo Biempica
Director da Tese de Doutoramento

Asdo.: David Deibe Seoane
Autor da Tese de Doutoramento

Aos meus pais,
por darme a oportunidade de chegar ata aquí.

Agradecementos

En primeiro lugar, gustaríame agradecerlles a Marga e Ramón, directores desta tese, a oportunidade que me concederon para participar nun proxecto tan interesante e con tantas posibilidades coma este e pola axuda e dedicación prestadas ao longo destes anos.

Gustaríame tamén dar as grazas a todos os membros do corpo docente cos que tiven a sorte de traballar, colaborar e aprender e que sempre se mostraron tan dispostos a participar e axudar no proceso de creación desta tese. Tanto a aqueles da Facultade de Informática da Coruña como aos da Universidade de Santiago de Compostela, grazas. Con especial mención ao Grupo de Arquitectura de Computadores no que desenvolvín a miña tarefa investigadora. Tampouco podería esquecer aos membros do grupo LaboraTe, especialmente a Sandra, por toda a axuda e tempo prestados.

Of course, I'd also want to thank Dr. Jürgen Döllner for hosting and advising me during my visit to the University of Potsdam, as well as to Beate and all my colleagues at the Hasso-Plattner-Institut, especially Daniel, Willy and Vlad for their amazing welcome and the support they gave me.

A título persoal, gustaríame dedicar tamén unhas palabras especiais a todas aquelas persoas que supuxeron un apoio moral, que estiveron ao meu carón sempre que o precisei, por todos eses momentos de alegría e troula e, en definitiva, por ser unha parte tan importante na miña vida en xeral e nesta etapa en particular. Aos meus pais e a Laura, grazas por todo, pois fostes sen dúbida as principais persoas coas que puiden chegar ata aquí.

Por último, gustaríame estender o meu agradecemento a tódalas entidades que

fixeron posible o financiamento deste proxecto: a rede de investigación HiPEAC, á Xunta de Galicia (Ref. ED431C 2017/04, Ref. GRC 2013/055, Ref. R2016/037, Ref. R2014/049, Ref. ED431G/01), ao Goberno de España (TIN2016-75845-P, TIN2013-42148-P) e á empresa Inditex polo seu financiamento da miña estadía.

David Deibe Seoane

Quote me as saying I was mis-quoted.

(Citadme diciendo que me han citado mal)

Groucho Marx

Resumo

A tecnoloxía LiDAR (*Light Detection And Ranging*) é actualmente unha das máis valiosas fontes de información xeográfica xa que permite, mediante dispositivos de dixitalización láser, a obtención de modelos tridimensionais de alta resolución de grandes áreas de terra. Os datos LiDAR, normalmente almacenados como nubes de puntos, utilízanse nun gran número de campos científicos e profesionais como un elemento fundamental do traballo. Debido á enorme cantidade de información que pode ser xerada mediante esta tecnoloxía, os conxuntos de datos LiDAR sempre foron considerados coma un gran desafío á hora de desenvolver aplicacións software capaces de manexar tales volumes de información de xeito rápido e eficiente.

Toda a investigación realizada durante esta Tese centrouse no desenvolvemento de novas técnicas, algoritmos e sistemas que mellorasen o rendemento, a eficiencia e a calidade dos múltiples e diversos elementos críticos dos contornos LiDAR. Así, desenvolvéronse sistemas web de tipo cliente-servidor para visualizar e procesar en tempo real grandes nubes de puntos de resolución completa, permitindo o acceso desde calquera tipo de dispositivo, dende tabletas a equipos de sobremesa, adaptando as súas funcionalidades e características aos requisitos e necesidades de campos específicos do coñecemento científico. As elevadas esixencias de almacenamento típicamente asociadas aos datos LiDAR, así como o intenso tráfico de rede que pode xerarse en aplicacións de tipo web, levou ao desenvolvemento de métodos de compresión de datos *sen perdas* xunto con novas estruturas de datos baseadas na non redundancia de información. Estes novos elementos foron utilizados para proporcionar un soporte altamente eficiente para técnicas multi-resolución *out-of-core* para a visualización en tempo real de nubes de puntos masivas, reducindo de xeito significativo os requisitos de almacenamento, o consumo de memoria principal e de vídeo, así como a conxestión no tráfico de rede. Por último, estableceuse como obxectivo da fase final da Tese, superar as limitacións derivadas da execución de soft-

ware en computadores compostos por unha única máquina para o almacenamento e a computación sobre grandes conxuntos de datos LiDAR masivos. A partir dun estudo preliminar para analizar a idoneidade de diferentes solucións *big data* para almacenar, distribuír e dar soporte ao envío de datos a varios clientes LiDAR, desenvolveuse un sistema altamente escalable para a computación distribuída sobre os volumes de datos mencionados. Como punto de partida, implementáronse diversas propostas utilizando como caso de estudo a creación de modelos dixitais do terreo (MDT), servindo como base tecnolóxica para un futuro servizo coa capacidade de ofrecer unha biblioteca de múltiples procesos xeoespaciais.

Resumen

La tecnología LiDAR (*Light Detection And Ranging*) es actualmente una de las fuentes de información geográfica más valiosas ya que permite obtener, mediante el uso de dispositivos de escaneo láser, modelos tridimensionales de alta resolución de amplias extensiones de terreno. Los datos LiDAR, almacenados típicamente como nubes de puntos, son utilizados en un gran número de campos científicos y profesionales como elemento fundamental de trabajo. Debido a las enormes cantidades de información que pueden generarse, los conjuntos de datos LiDAR siempre han supuesto un enorme desafío a la hora de desarrollar software capaz de manejar dichos volúmenes de información de manera rápida y eficiente.

Todas las investigaciones realizadas durante esta Tesis se centraron en desarrollar nuevas técnicas, algoritmos y sistemas que permitiesen mejorar el rendimiento, la eficiencia y calidad de múltiples y diversos elementos críticos de los entornos LiDAR. Así, se desarrollaron sistemas web de tipo cliente-servidor para la visualización en tiempo real y el procesamiento de grandes nubes de puntos de resolución completa desde cualquier tipo de dispositivo, desde PCs hasta tabletas, adecuando sus funcionalidades y prestaciones a los requerimientos y necesidades de campos de conocimiento científico específicos. Las altas demandas de almacenamiento asociadas a los datos LiDAR, así como también el intenso tráfico de red que puede darse en aplicaciones de tipo web, llevaron al desarrollo de métodos de compresión de datos *sin pérdida* junto con nuevas estructuras de datos basadas en la no-redundancia de información. Estos nuevos elementos se emplearon para dar soporte altamente eficiente a técnicas de multiresolución *out-of-core* para la visualización en tiempo real de nubes de puntos masivas, logrando reducir de manera notable los requisitos de almacenamiento, el consumo de memoria principal y de vídeo, así como también la congestión de red. Por último, se fijó como objetivo de la etapa final de la tesis, el de superar las limitaciones derivadas de la ejecución de software en equipos compuestos

por una sola máquina para el almacenamiento y computación de grandes conjuntos de datos masivos LiDAR. Partiendo de un estudio preliminar para analizar la idoneidad de distintas soluciones *big data* a la hora de almacenar, distribuir y dar soporte al envío de datos hacia múltiples clientes LiDAR, se desarrolló un sistema altamente escalable para la computación distribuida sobre los citados volúmenes de datos. Como punto de partida, se implementó un conjunto de propuestas utilizando como caso de estudio la creación de modelos digitales de terreno (MDT), sirviendo este como base tecnológica para un futuro servicio con la capacidad de ofrecer una librería de diversos procesos geoespaciales.

Abstract

Currently, LiDAR (*Light Detection And Ranging*) technology is one of the most valuable sources of geographic information, allowing us to obtain, through the use of laser scanning devices, high-resolution three-dimensional models of large tracts of land. LiDAR data, typically stored as point clouds, have been used in a wide range of scientific and professional fields as a fundamental element of work. Due to the extremely large volumes of data that can be generated, LiDAR datasets have always been a challenge for developing software capable of handling such volumes of information in a fast and efficient way.

All the research carried out during this Thesis was focused on developing new techniques, algorithms and systems to improve the performance, efficiency and quality of several critical elements in LiDAR environments. Thereby, client-server web systems were developed for real-time visualization and processing of large point clouds of full-resolution grating their access from any type of device, from desktop PCs to tablets, adapting their functionalities and features to the requirements and needs of specific fields of scientific knowledge. The high storage demands typically associated with LiDAR, as well as the intense network traffic that can be produced in web environments, led to the development of lossless data compression methods along with novel data structures based on non data redundancy. These new elements were used to provide highly efficient support for out-of-core multi-resolution techniques for real-time visualization of massive point clouds, achieving a notable reduction in storage requirements, main memory and video memory consumptions, as well as a reduction in network congestion. The objective in the final stage of the Thesis was to overcome all storage and computational constraints related to the use of a single machine with large collection of massive LiDAR datasets. Starting from a preliminary study to analyse the suitability of different big data solutions for storing, distributing and supporting the concurrent access to the data from several

LiDAR clients, a highly scalable system was developed offering distributed computing capability for processing said volumes of data. As a starting point, a series of proposals was implemented using as a case study the obtention of digital terrain models (DTM), serving as a technological basis for a future service with the ability to offer a much wider library of several geospatial processes.

Preface

Introduction and Motivation

LiDAR (*Light Detection And Ranging*) technology stands up as one of the most important and valuable sources of geospatial information among the different remote-sensing technologies available nowadays. By providing data in the form of highly detailed point clouds, this technology has brought great benefits to a large variety of scientific and professional fields, such as agroforestry, archaeology, robotics, or autonomous vehicles, among many others.

Some of the most detailed and highest quality point clouds may contain several billion points with great deal of per-point information, such as (x, y, z) coordinates, RGB colour or GPS time. Large collections of these massive datasets usually surpass the terabyte, or even the petabyte, in size, which makes LiDAR technology a challenge when it comes to developing efficient applications to handle such volumes of geographical information. In professional and scientific environments working under high data collection rates, the use of highly scalable systems for storing, distributing and processing all new data becomes a critical requirement. Furthermore, highly specialised users also demand optimal performance and efficiency on their client machines, not only on high-end desktop solutions but tablets, laptop hybrids or even smartphones. From the point of view of hardware capacity, almost any kind of hardware resource; such as GPU and CPU computing power, network bandwidth or main memory, fall short to handle large volumes of LiDAR data, demanding the use of very efficient software algorithms and systems.

Objectives and Work Methodology

The aim of this Thesis is to propose a series of novel approaches, algorithms and systems to improve, or to present new solutions, to a wide range of issues related to the use of aerial LiDAR data. As LiDAR technology stands out as a field involving a large number of dimensions, ranging from storage, distribution, concurrent access to data, visualization or processing, we have firstly established a hierarchy based on the size of the problems to solve, which in the LiDAR field is directly related to the size and number of the point clouds. Then, a series of global objectives were established on each level of the hierarchy. Parts, or the whole proposals and contributions of one level, were exploited in the next ones, helping to build, step by step, a complete and multifunctional system for aerial LiDAR datasets. Said hierarchy and its objectives have been set as the following:

- **Large point clouds (hundreds of million points)**: Research focused on full-resolution visualization and client-side processing. Objectives: flexibility, adequate workflows, field-specific measurement tools and optimizations for performance maximization (real-time rendering and fast data loading).
- **Massive point clouds (billion points)**: Research focused on multi-resolution and out-of-core visualization. Objectives: optimization of computational resources for real-time visualization systems based on client-server patterns.
- **Large collections of massive point clouds**: Research focused on overcoming single-machine constraints through big data approaches offering solutions throughout two different stages:
  - Distributed storage: Maximization of storage capacity, latency and throughput of back-end systems supporting LiDAR applications.
  - Distributed computing: Maximization of computing capacity, latency and throughput of geospatial processes of high computational complexity involving extremely large volumes of LiDAR datasets.

Some measurements and geospatial processes employed in the LiDAR field are intended to operate on specific sub-regions within the point clouds. Additionally,

the use of certain point densities became unnecessary, since once a certain number of points per square meter was reached no further improvements or quality increase could be achieved beyond said density for such measurements and geospatial processes. An example of this can be found in [51]. In contexts like this, the number of points handled, while very high (hundreds of millions of points), is far from the billions handled in other contexts or use cases. Due to this difference, we decided to follow two different approaches for each of the contexts. Hence, with the aim of accomplishing the objectives of the hierarchy presented above, the research of this Thesis has evolved across four different stages following an incremental methodology. During an initial stage, improvements and contributions for the narrowed contexts (framed outside the definition of big data) were developed. After said initial stage, the Thesis was aimed at facing more complex problems in more demanding contexts until reaching a purely big data environment during its final stages.

First Stage

In the first stage of this Thesis (Chapter 2), all research was focused on the full-resolution visualization and client-side processing of large point clouds, establishing four main goals: flexibility, an adequate workflow, field-specific measurement tools and optimizations for performance maximization (real-time rendering and fast data loading). All research and contributions were included and tested in a visualization application specially developed to do so. The goals of this stage were accomplished through the following strategies and approaches:

- **Flexibility**: WebGL [78, 106] was the graphics application programming interface (API) chosen for the development of the framework since it allows the production of powerful web-based visualization solutions. A web-based approach makes it possible to not be tied to any operating system (OS) or device in particular, granting instant access to any user from any location with just running a WebGL-compatible web browser. Our proposal was conceived as a service-oriented approach; hence application and data would be stored on a remote server and retrieved by clients as needed, further increasing the mobility and flexibility of the framework.
- **Adequate workflow**: In certain work contexts, although very large point

clouds may be collected, the way they are meant to be explored or processed only requires operating on specific sub-regions, usually involving a waste of valuable hardware resources handling unnecessary data located outside said sub-regions. Considering this, data queries based on spatial restrictions were implemented through spatial hashing techniques and the definition of regions of interest (ROI) over the point clouds.

- **Field-specific measurement tools**: Client-side tools to make geospatial measurements directly over the point clouds rendered on screen were implemented through JavaScript (JS) [38] and WebGL in order to provide more useful tools in comparison to what was offered by other visualization solutions. Specific tools such as facade surface measurement or other much more complex tools like volumes of irregular base, polygonal contour and projected top, were included in the framework to expand its functionality.

- **Optimizations for performance maximization**: As previously commented, the research in this first stage of the Thesis was focused on full-resolution visualization in order to maximize the accuracy of the measurement tools and the image fidelity for visual inspections in real-time. By following this approach, performance becomes a critical element during the development process. Thereby, several optimization strategies were implemented trying to maximize the performance of the rendering process and the acquisition and loading of remote data.

The research of this stage is based on a bachelor's thesis[1] and it was published in [34] and [35] and registered in [33].

Second Stage

In the second stage of the Thesis (Chapter 3), new directions in the use of multi-resolution and out-of-core techniques were explored. As a result, a novel visualization strategy was developed supported by a non-redundant data point organization method called *Hierarchically Layered Tiles* (HLT), and a tree-like structure called

[1]Bachelor's thesis permanent link: http://kmelot.biblioteca.udc.es/record=b1514673 S1*gag

Tile Grid Partitioning Tree (TGPT). These efficient data structures were designed with the aim of avoiding the data redundancy always associated to the creation and management of LODs for point clouds. The main ideas behind the structures were that the points of each point cloud were rearranged and stored in layered tiles in such a way that no point was repeated across layers. These layered tiles serve as puzzle pieces to compute and create different LODs at runtime, as needed, by merging two or more of those pieces.

For testing and analytic purposes, the new structures were included in a new iteration of the visualization application presented in the first stage of the Thesis, reorienting its design towards the fast visualization of entire massive point clouds through a multi-resolution and out-of-core approach. The first visualization framework was capable of efficiently rendering full-resolution clouds and sub-regions of clouds containing between 50 and 100 million points. This new approach was not intended to replace the previous one, but to expand its functionalities by including the capability to efficiently exploring massive datasets containing billions of points. A wide range of optimizations were achieved on both server side and client side, thanks to the aforementioned contributions:

- **Server-side storage requirements**: As a result of not storing fully precomputed LODs and with the avoidance of data redundancy, storage requirements were notably reduced.
- **Network traffic**: On a client-server application system, high levels of traffic may cause network congestion. Avoiding data redundancy also has an impact on the total amount of bytes moved through the network, as lower amounts of points are sent to the clients.
- **Client-side storage requirements**: Same principle as for server-side storage requirements applies here, since avoiding data redundancy leverages storage requirements also on the client side, since part of the performance of web-based visualization tools relies on the use of the client-side browser cache. Data retrieved from server are stored on client's disk for accelerating subsequent use of the same data. Handling large volumes of data may cause same storage issues as on the server side, or even worse, due to the lower storage capacities of most client machines, especially laptops or handheld devices.

- **RAM consumption**: Similar layered and non-redundant data organization has been followed to copy the points in main memory during the execution of the point rendering process. LODs are computed and discarded in real-time as required, avoiding the need to hold unnecessary data on RAM. LODs are recalculated every time the 3D scene camera detects a considerable change on the point of view (POV). This approach brings great benefits for mid and low-end systems, tablets or laptops mounting moderate amounts of RAM.

- **VRAM consumption**: New LODs computed in real-time are streamed to a fixed-size GPU buffer based on a user-defined visual point budget (PB). Unlike other rendering techniques, this proposal keeps VRAM usage constant all along the point rendering process.

All the research of this Chapter was published in [32] and registered in [28].

Third Stage

In the third stage of this work (Chapter 4), it was analysed how applications making an intense use of large collections of massive LiDAR datasets, in particular web-based applications focused on real-time rendering, could benefit from the adoption of big data storage technologies. Additionally, a study is presented on the advantages and disadvantages that could determine the choice of the most suitable option among the currently available on the market depending on the requirements and characteristics of each use case.

Several analyses and comparisons were carried out using four of the most adopted and mature big data storage technologies. It was demonstrated how big data technologies could be employed as the back-end of said LiDAR applications with no drawback or penalty in performance or user experience, while gaining all of the usual advantages associated with big data solutions, such as reliability, availability and scalability. With a view towards the future, any system adopting such type of storage technology would be already prepared to incorporate other technologies for distributed computing, such as Spark, Flink or Storm.

All the research of this Chapter was published in [29].

Fourth Stage

In the fourth and final stage of the Thesis (Chapter 5), a big data approach on geospatial processing for massive aerial LiDAR point clouds was fully developed. The system was intended to support the execution of any kind of geospatial process; nonetheless, as an initial case of study, the research focused only on fast DTM obtention from massive point clouds.

Following the analysis and conclusions presented in the previous chapter, data distribution was done using Cassandra [95], while the computing distribution was accomplished with Spark [100], due to its versatility, source code compatibility and batch-oriented design. Thanks to this approach, it was possible to greatly reduce the time required for processing very large extents of aerial point clouds compared to other single machine approaches. Another important contribution presented in this stage was an automated classification error correction strategy that improved the quality of the DTMs obtained while minimizing user intervention.

The research presented in this chapter was submitted for publication to [31].

Main Contributions of the Thesis

The main contributions of this Thesis can be summarized as:

- Two web-based visualization and client-side processing applications for aerial LiDAR data.
- An efficient method to retrieve remote data through queries based on spatial restrictions.
- Optimization strategies for fast data loading of remote LiDAR point clouds.
- Optimization strategies for rendering full-resolution point clouds in real-time.
- A series of field-specific geospatial measurement tools to work directly on rendered images.

- A novel approach on point cloud rearrangement and storage based on the non-redundancy of information to support multi-resolution and out-of-core algorithms.
- An up-to-date analysis of big data storage technologies and their benefits for LiDAR applications.
- A highly scalable system for the distributed computing of geospatial processes that can be applied over large collections of massive point clouds. Specifically, as an initial use case, the system offers the possibility to obtain rasters of ground-only points for DTM generation.
- A strategy for correcting classification errors on the boundaries of adjacent zones of a point cloud that have been independently processed.
- A lossless compression algorithm for point cloud files.
- As a whole, all previous contributions make a complete and multifunctional client-server system for aerial LiDAR data.

Structure of the Thesis

This Thesis is organized as follows:

- Chapter 1 introduces the LiDAR technology, main concepts about aerial data acquisition, file formats and software tools related to this technology. Additionally, it describes some basic concepts about GPUs and 3D rendering, cloud computing and big data solutions applied to the storage and processing of geospatial information. Finally, an analysis of the challenges related to handling and processing LiDAR datasets is presented in detail.
- Chapter 2 presents a series of geospatial measurement tools specially designed to work in specific LiDAR fields, such as agroforestry, an efficient method to perform data queries based on spatial restrictions and several optimization strategies for high performance rendering and remote data loading for full-resolution point cloud visualization. All of these proposals are analysed

and compared to some of the most used and well-known LiDAR visualization applications.

- Chapter 3 addresses a novel approach on multi-resolution and out-of-core techniques based on non-redundant data structures aiming to optimise several computational resources required on visualization systems following a client-server pattern. All improvements presented were analysed and compared to one of the most known and best valued visualization frameworks.

- Chapter 4 analyses four of the most adopted and mature big data storage solutions with the aim of overcoming all single-machine constraints related to the use of traditional server solutions in environments related to the use of large collections of massive point clouds, placing special attention on web-based visualization.

- Chapter 5 extends the research presented in the previous chapter with the aim of overcoming single-machine constraints related to the execution of geospatial processes of very high computational complexity through the use of a big data approach. A highly scalable system for fast DTM generation is presented together with a method to correct common errors related to processes of this type.

- Chapter 6 extracts the conclusions of the Thesis and points out a number of lines for future work.

Funding and Technical Means

The following means and funding have been used to carry out the Thesis:

- Working material, as well as human and financial support provided by the Computer Architecture Ground (GAC) of the University of A Coruña.

- Access to bibliographical material through the library of University of A Coruña.

- Access to clusters, supercomputers and other computing platforms: Pluton cluster (Computer Architecture Ground of the University of A Coruña).
- Funding through the following research projects:
  - Xunta de Galicia: Consolidation Programme of Competitive Reference Groups, co-founded by ERDF funds from the EU [Ref. ED431C 2017/04] and [Ref. GRC 2013/055].
  - Xunta de Galicia: Consolidation Programme of Competitive Research Units, co-founded by ERDF funds from the EU [Ref. R2016/037] and [Ref. R2014/049].
  - 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].
  - Ministry of Economy and Competitiveness of Spain and ERDF funds from the EU [TIN2016-75845-P] and [TIN2013-42148-P].
- Three-month research visit to the Hasso-Plattner-Institut, University of Potsdam, Germany, funded by the Inditex-UDC 2019 collaboration grant.

Third party datasets

All LiDAR point clouds used in this work belong to:

- *LiDAR-PNOA* data repository [55]: Provided by Instituto Geográfico Nacional de España (IGN) [54].
- *Guitiriz*: Provided by Laboratorio do Territorio (LaboraTe) [108].
- *PG&E Diablo Canyon Power Plant (DCPP): San Simeon*, CA Central Coast [74], *PG&E Diablo Canyon Power Plant (DCPP): Los Osos*, CA Central Coast [75] and *Sunset Crater Volcano National Monument*, AZ [76]: 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.

- *ISPRS Filter Test dataset*: Samples from the International Society for Photogrammetry and Remote Sensing, provided by the University of Twente [109].

Registered Software

- David Deibe, Margarita Amor, Ramón Doallo, Rafael Crecente, David Miranda and Miguel Cordero: *VGLiDAR 1.0 Visualizador Gallego de Datos LiDAR.*
  Software registration: Universidade da Coruña y Universidade de Santiago de Compostela, Spain. C-423-2015 11/23/2015.

- David Deibe, Margarita Amor, Ramón Doallo: *ViLMA (Visualization for LiDAR data using a multi-resolution approach).*
  Software registration: Universidade da Coruña y Universidade de Santiago de Compostela, Spain. C-388-2018 11/19/2018.

Publications from the Thesis

Journal Papers

- David Deibe, Margarita Amor, Ramón Doallo, David Miranda and Miguel Cordero: GVLiDAR: an interactive web-based visualization framework to support geospatial measures on lidar data. In *International Journal of Remote Sensing*, volume 38, issue 3, pages 827-849, published online January 2017.
  JCR Impact Factor 1.782, Q2 in IMAGING SCIENCE & PHOTOGRAPHIC TECHNOLOGY.
  DOI: 10.1080/01431161.2016.1271476

- David Deibe, Margarita Amor and Ramón Doallo: Supporting multi-resolution out-of-core rendering of massive LiDAR point clouds through non-redundant data structures. In *International Journal of Geographical Information Science*, volume 33, issue 3, pages 593-617, published online November 2018.
  JCR Impact Factor 3.545, Q1 in COMPUTER SCIENCE, INFORMATION

SYSTEMS.
DOI: 10.1080/13658816.2018.1549734

- David Deibe, Margarita Amor and Ramón Doallo: Big data geospatial processing for aerial LiDAR datasets. Submitted for publication.

Conferences

- David Deibe, Margarita Amor, Ramón Doallo, David Miranda and Miguel Cordero: VGLiDAR: Una herramienta de procesamiento de datos LiDAR en la GPU usando WebGL. In *XXVI Jornadas de Paralelismo (JP2015)*, pages 146-151. September 2015.

- David Deibe, Margarita Amor and Ramón Doallo: Big data storage technologies: a case study for web-based LiDAR visualization. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 3831-3840, December 2018. DOI: 10.1109/BigData.2018.8622589

- David Deibe, Margarita Amor and Ramón Doallo: BETi: Sistema para la gestión y procesamiento de datos masivos LiDAR. In *XXX Jornadas de Paralelismo (JP2019)*, pages 516-523. September 2019.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In the past few decades, the usage of different remote-sensing techniques has undergone a notable increase, driven by the need to obtain a wide range of earth surface information, while achieving satisfactory cost-efficiency ratios, high data precision and short data acquisition times. In particular, LiDAR (Light Detection And Ranging) surveying has stood out as one of the most valuable sources of geographic information, providing very useful high-resolution datasets in the form of point clouds that can be applied in a wide variety of scientific, professional and governmental fields, such as public policy planning, agriculture, archaeology, biology or forestry.

Some of the most detailed and highest quality point clouds may contain several billion points, reaching or even surpassing a terabyte of disk space usage. For years, due to the extraordinary large amounts of data that could be collected, LiDAR point clouds have been considered as an extraordinary challenge when it comes to their storage, processing or visualization, making the use of efficient and highly scalable systems a fundamental requirement.

On GIS centres, governmental institutions, or any other group constantly collecting new point clouds of those characteristics, only the storage of such volumes of raw data entails a significant cost in terms of economic and technical resources.

The popularization of tablets, laptop hybrids and smartphones, in addition to the release of HTML5, has favoured the appearance of web applications offering

high portability, flexibility and availability, these features being a major advance in comparison with classic desktop applications. Despite their advantages, web applications come with an important handicap, since in such software and hardware environments, there are strong restrictions and limitations on the main and video memory available, the disk storage capacity and the performance in terms of execution times.

Applications for LiDAR data can be found in coastal-change studies [89], monitoring of landslides [111], analysis of volcanoes, from lava flows to volcanic deformation [49, 57], urban and land cover classification [14, 113], forest inventory and biomass estimation [11, 45], the study of active tectonics [5, 13], automatic extraction of geo-morphologic features [79] or in the creation of detailed large-scale city models [61]. Moreover, the combination of LiDAR with other types of remote sensing data can provide additional benefits in several contexts [117]. An extensive list of further applications can be found in [93], while a useful review on the achievements and advantages of the usage of LiDAR is provided in [88].

Other popular digital terrain representations, such as two-dimensional (2D) grids, three dimensional (3D) grids or Triangulated Irregular Networks (TIN), which in some cases are derived from LiDAR data, have also been widely used for different types of qualitative and quantitative analysis. Numerous studies have been carried out regarding the individual or combined usage of these elements, especially in the field of data visualization [9, 58, 77, 114]. Many of these approaches provide high resolution information, which may improve the accuracy for recognizing certain topographic features, but fall short in recognizing some structures, such as drainage ditches or levees, which may be misinterpreted during topographic analysis if high-resolution data are not used [94]. On the other hand, LiDAR models are capable of clearly showing buildings, trees or sea ice surface roughness [62], while in most other digital models, those kinds of structures may be barely recognizable or non-existent.

In the following sections of this chapter, LiDAR data technology is presented in detail, as it is the core element of the whole Thesis. Section 1.1 introduces the main concepts of LiDAR technology, in special, the main characteristics of airborne laser scanning (ALS) for land surveying, since every software component presented in the Thesis was built upon this source of geospatial information. The following sections cover the main topics of this work, thereby, a brief summary on GPUs and 3D

rendering is presented in Section 1.2 while big data technologies and its paradigms are presented in Section 1.3.

1.1. LiDAR technology

LiDAR is a surveying method that employs laser pulses to measure distances between a sensor and a given target. Differences in pulse return times provide 3D information about the surfaces being reached by the pulses. All information gathered is presented in the form of 3D point clouds.

Besides the spatial coordinates (x, y, z), additional information may be collected during the surveying process, or even included later during post-processing stages. Therefore, points can provide further useful information, such as pulse intensity, RGB colour or classification (building, vegetation, water, etc.), among others.

For the purpose of scanning large tracts of land, LiDAR sensors can be mounted on almost any type of vehicle, from unmanned aerial vehicles (UAV) to airplanes, cars or boats; however, sensors may also be mounted on static structures to capture indoor 3D scenes or outdoor areas of reduced dimensions.

The precision achieved by LiDAR allows the creation of very accurate 3D representations of any type of target surface, including elements of great complexity or elements that are hard to identify or capture through other types of surveying methods; e.g., surfaces below dense tree canopies or thin power lines.

Among all the major remote-sensing systems, LiDAR stands out as one of the most important and useful available nowadays.

1.1.1. Aerial LiDAR data

Figure 1.1 shows a schematic representation of land surveying using airborne laser scanning (ALS). In the figure, an aircraft equipped with a LiDAR sensor flies over a target area while an on-board sensor emits laser pulses towards the surface of the land. Pulses are usually emitted following a striped pattern, although other emission patterns, such as zigzag or elliptic, can be also used. After hitting the

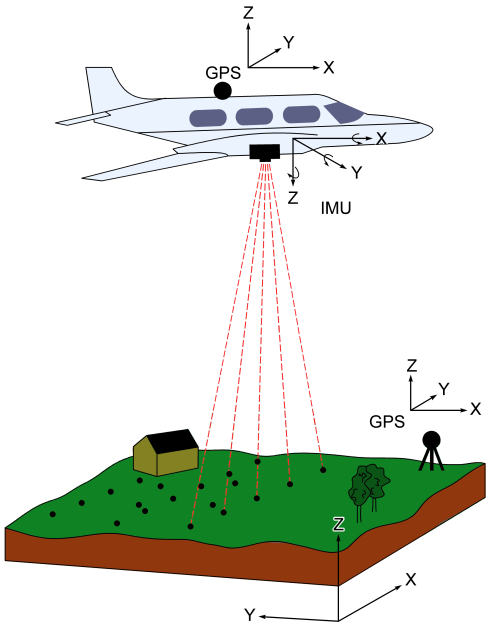


Figure 1.1: Land surveying using ALS. Source: [67].

surface, pulses return to the sensor and the elapsed time between the emission thereof and the reception of their echoes are measured to calculate points on the surface. The surveying system is also equipped with GPS and inertial measurement unit (IMU) subsystems. The GPS subsystem continuously measures the position of the aircraft in terms of longitude, latitude and height. On the other hand, the IMU subsystem continuously measures deviations (roll, yaw and pitch) suffered by the aircraft due to manoeuvres and turbulence. The precise 3D position of each point is obtained through combining the information provided by these two subsystems and the information provided by the laser sensor. 3D coordinates are presented using some global coordinate system, such as WGS84, or other regional systems; e.g., EPSG:23029 or EPSG:23030, for Spain coordinates [52].

Depending on the surveying system capabilities and the type of surface reflecting the laser pulses, one or more points can be obtained from a single laser pulse. After emitting a pulse, and as it travels towards the ground, the pulse can be split into as many returns as reflective surfaces it encounters. Figure 1.2 may help to understand this phenomenon. The first return is produced by the first found surface while the last one is produced by the last reflective surface; for example, the first one can be a

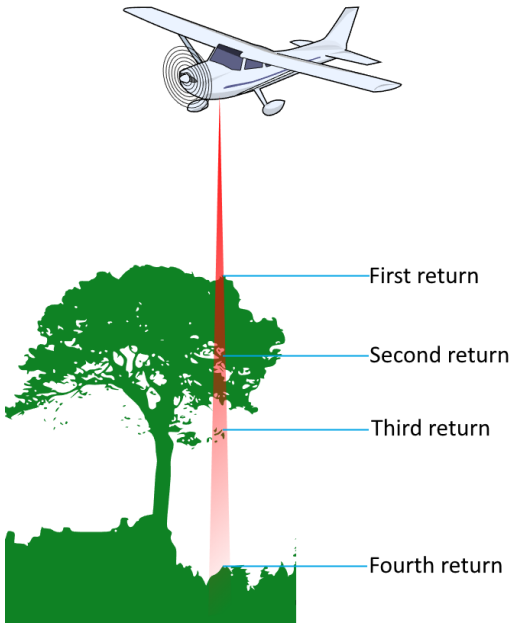


Figure 1.2: Four different returns produced by a laser pulse as it travels towards the ground. Source: [6].

reflection from the top of a tree and the last one can be reflected by the ground below a forest canopy. Return information is highly useful as it is used by filters in charge of classifying the points of a cloud into categories such as building, vegetation, water or ground.

1.1.2. File format

Point clouds can be stored in a wide variety of file formats; nevertheless, LAS is the standard format used in the LiDAR field. The LAS specification was created by the American Society for Photogrammetry and Remote Sensing (ASPRS) [4] with the aim of providing an open format for different LIDAR hardware and software tools.

LAS files contain binary data consisting of an initial header block, any number of optional variable length records (VLRs), a block with all the point data records and finally any number of optional extended variable length records (EVLRs). The header block contains generic information about the file, such as the number of point

Table 1.1: Data contained into one data record of LAS (format 0).

Item	Format	Size	Required
x	long	4 bytes	*
y	long	4 bytes	*
z	long	4 bytes	*
Intensity	unsigned short	2 bytes	
Return Number	3 bits (bits 0 - 2)	3 bits	*
Number of Returns	3 bits (bits 3 - 5)	3 bits	*
Scan Direction Flag	1 bit (bit 6)	1 bit	*
Edge of Flight Line	1 bit (bit 7)	1 bit	*
Classification	unsigned char	1 byte	*
Scan Angle Rank	char	1 byte	*
User Data	unsigned char	1 byte	
Point Source ID	unsigned short	2 bytes	*

data records stored, the byte position where they begin, or the date when the data were obtained. Both the VLRs and EVLRs are optional blocks that can be used by the different LiDAR software in order to store information which they may require.

The LAS specification defines different *Point Data Record* formats with different properties each. Table 1.1 shows the properties of a single point stored in an LAS file following the format number 0.

Although LAS is the standard file format, in many fields where large volumes of data must be handled, LAS files are compressed using the LASzip format (LAZ). The technical details regarding this format can be found in [56].

1.1.3. LiDAR software

In recent years, the use of LiDAR technologies has increased along with the number of software solutions designed to handle data of this type. Some of the most known and commonly used applications are listed below, since they will be used in many of the chapters of this Thesis for comparative reasons.

1. *LAStools* (3D visor) [41]: This is probably the most commonly used desk-

top application for LiDAR data. It is free, easy to use and includes several processing tools. Its 3D visor performs many different visualization options, such as intensity, classification, height, number of return, RGB or scan direction. Users can filter the point cloud by point properties, or create a full 3D model by triangulating the point set. Nevertheless, the 3D visor offers low performance, rendering over 4,700,000 points under 10 frames per second (FPS). Furthermore, it only works with local files loaded by users, and it lacks geospatial measuring tools over the 3D point cloud.

2. *FugroViewer* [39]: Simple and easy-to-use desktop visor, it provides different visualization options, such as intensity, height, classification, return number or RGB. A complete 3D model can be obtained through a triangulation of the point cloud. There are two visualization modes: 2D and 3D. The 2D mode applies level of detail (LOD [2]) techniques over the points set; therefore, when the camera is near, all points are rendered, but from a distant perspective only one subset of the points are shown. The 3D mode achieves high performance, although it only renders the visible points that are shown in a 2D window, so that, when zooming over the 3D model, no more points are loaded and detail is lost. The LiDAR data has to be provided by users. Finally, a tool for measuring distances is available, but only over the 2D image.

3. *Global Mapper* [7]: Another desktop tool, very similar to the FugroViewer but with wider features and computing capabilities. It can process and handle many types of geospatial information and file formats. It can mix and display together many different data sources with ease, including online data sources. The basic 3D rendering of the point clouds offers very good performance, although it has some limitations in the way the point clouds can be rendered, lacking advance rendering features such as shadows, contour highlighting or gap filling. It offers a considerable amount of useful measurement tools for use on 3D images, probably being the most complete set of tools among all the visualization application of this list. Despite all of its useful features, it can take over a minute to load 10 or more million points, and may even cause the application to crash.

4. *IDECanarias* [44]: Web application well-known in Spain which provides LiDAR data from the Canary Islands on-demand. It has low retrieval times,

around 5 seconds for 350,000 points (the maximum amount of points retrieved), and is easy to use. Nevertheless, in many systems it does not obtain a fluid interaction due to its Flash programming, achieving about 36 FPS with 250,000 points. Areas of a few square meters and large areas of several square kilometres are rendered with almost the same number of points. While the detail obtained in the former case may be good, in the latter case it may not be enough for recognizing certain land features or structures. The application has limited visualization options (intensity, height and mixed) and it lacks geospatial measuring tools over the 3D point clouds.

5. *LiDAR Online* (3D visor) [64]: Based on *Dielmo 3D* technology [36], this web visor allows LiDAR data from various locations around the world to be obtained on-demand. The 3D visor has good visualization options (classification, intensity, height and RGB) and offers highly satisfactory performance results, up to 60 FPS displaying around 1 million points. Nevertheless, transferring more than 1 million points is not permitted because, like *IDECanarias*, the number of points shown barely change along with the size of the selected area: small and large areas are retrieved with almost the same number of points, so the latter are displayed with reduced detail. Furthermore, the times are quite long; more than a minute to retrieve 1 million points, and sometimes the process may get blocked. It lacks geospatial measuring tools over the 3D point clouds.

6. *Plas.io* [107]: This web application is based on WebGL. It focuses on the visualization of local files (in LAS and LAZ formats) achieving 45 FPS, handling around 16 million points. It offers different rendering modes, such as RGB, intensity or height, but lacks geospatial measuring tools over the 3D point clouds.

7. *Lidarview* [3]: Another web application based on WebGL that offers good performance with around 50 FPS handling 10 million points. It implements some visualization options such as intensity, classification, height and RGB. It is easy to use but only works with local files loaded by users (LAS or xyz file formats). The visor is limited to 10 million point clouds and lacks geospatial measuring tools over the 3D point cloud.

8. *Potree* [90]: This web application, also based on WebGL, is an interactive out-of-core rendering solution for massive datasets (over 1 billion points) using a multi-resolution octree structure. It has the most similar features to our proposal; however, it is conceived as a general-purpose point cloud visualization tool and does not focus solely on LiDAR data. *Potree* allows direct measurement on the 3D images. Nevertheless, its measurement tools are basic and not focused on LiDAR data, it does not facilitate information such as the distance of a point to a plane, the area of complex surfaces or the square meters of facades (frequently required by professionals in the fields of engineering and architecture). Furthermore, it provides no geographic information on the images, such as the reference system used or the geographic coordinates of each point. Therefore, we can consider *Potree* as a tool mainly focused on the fast visualization of large areas of land or highly detailed objects, prioritizing realistic rendering against the functionality of the measurement tools.

9. *MegaTree* [86]. The designs of *Potree* and *Megatree*, follow a very similar client-server structure and their performance relies on the use of multi-resolution, out-of-core techniques supported by an octree structure. Nevertheless, this software solution merely offers visualization capabilities with no measurement tools or any other additional feature.

As a summary, Table 1.2 shows a comparison between the frameworks listed above. The second column in the table indicates whether the application offers remote data retrievals through queries based on spatial restrictions (on-demand data). The following columns show the maximum number of rendered points allowed, provision of geospatial measuring tools over the 3D point clouds (client-side) and real-time interaction capabilities (considering the ability to display at least 10 million point cloud achieving above 20 FPS).

1.2. GPUs and 3D rendering

For the last years, iterative graphic systems have become a hot topic, widely developed by the community. GPU (*Graphics Processing Unit*) research has been supported by an increasing real-time and interactive rendering demand for complex

Table 1.2: Comparative of LiDAR applications based on their performance and available features. * Indicates the use of a multi-resolution approach.

LiDAR framework	**On-demand data**	**Max. points (millions)**	**Measuring Tools**	**Real-time Interaction**
IDECanarias	√	∼0.35		
LiDAR Online	√	∼1		
Lidarview		∼10		√
LAStools		No limit		
FugroViewer		No limit		√
Global Mapper	√	No limit	√	√
Plas.io		16		√
Potree		No limit *	√	√
MegaTree		No limit *		√

and realistic models across many engineering and scientific areas and the video games industry.

In graphic environments, the computational power of the GPUs is accessed through an application programming interface (API). APIs such as DirectX or OpenGL have been widely used to create all manner of applications demanding very efficient 3D rendering. Each API has its own unique pipeline consisting of several linked stages that can be processed sequentially. These stages determine how the GPU should create the 3D images on screen. Only certain stages are meant to be programmed in order to change the behaviour and output of the pipeline. Two of the most common programmable stages are the vertex shader and the pixel shader. The vertex shader is traditionally employed for vertex transformation and per-vertex computation, while the pixel shader is usually employed for the computation of each fragment's colour.

Figure 1.3 shows an example of a basic graphics pipeline. In this figure, a set of vertex from a 3D object (the teapot) are used as input to the rendering process, obtaining as output a series of fragments with a colour related to them that define the 2D representation of the 3D object.

Modern pipelines, like the one shown in Figure 1.4, may contain multiple stages,

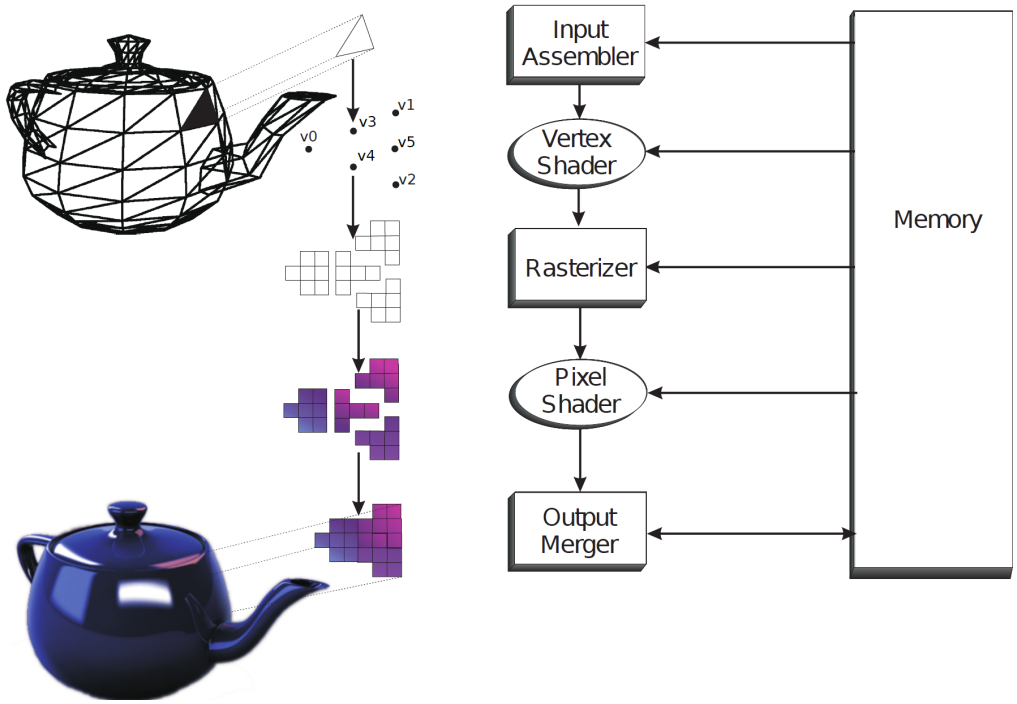


Figure 1.3: DX9 Pipeline [19].

some of them being highly programmable. By programming these stages, developers can define, for example, how the 3D models are built or modified, depending on the point of view of the camera in the scene.

One of the most recent APIs that have been developed is WebGL [78, 106]. This API was introduced with the aim of vastly improving the performance of GPU-intensive applications running in a web browser. This novel graphics API, is based on OpenGL ES 2.0 (OpenGL for Embedded Systems [71]), another API specially developed for handheld devices such as smartphones and tablets. Like the pipeline depicted in Figure 1.3, the pipeline of WebGL only contains two programmable stages: the vertex shader and the pixel shader. The appearance of WebGL has led to the creation of a lot of new 3D applications with a level of quality close to classic desktop versions.

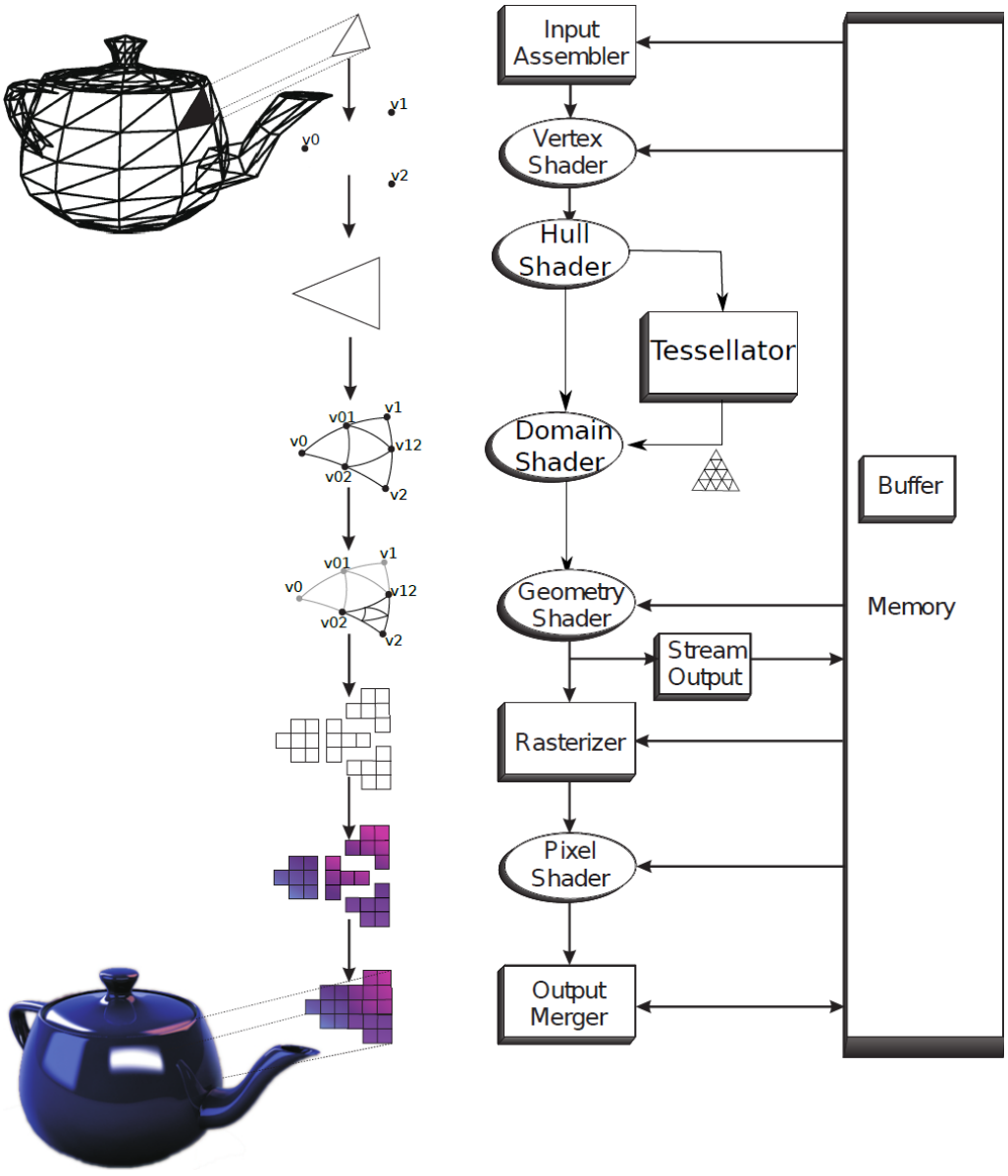


Figure 1.4: DX11 Pipeline [19].

1.2.1. Point cloud visualization

In general, GPUs are designed to operate mainly with triangles [19]; nevertheless, other primitives, such as lines or points, can also be processed by GPUs. Research and literature about point-based rendering is quite extensive, as many data structures, compression methods or even techniques for computing image shadows, have been specially designed to adapt to the unique properties of such primitives.

The interactive visual exploration of 3D scenes whose features exceed the hardware capabilities of client machines generally requires the use of some kind of multi-resolution and out-of-core approach, which always involves a trade-off between performance and visual quality [2]. With regard to large point clouds, these approaches may provide enormous performance benefits in a wide variety of applications [18, 116]. They have been employed in many visualization tools achieving significant performance results [46, 84] and they have also been used together with point-based rendering techniques to enhance image fidelity or object recognition [37, 59].

Multi-resolution and out-of-core approaches are generally supported by generic and well known data structures, such as quadtrees, octrees or kdtrees [47]. A number of academic publications have already discussed the advantages and disadvantages of those structures in web environments [40, 60] and their usefulness when used on mobile devices [87]. Nevertheless, the implementation of the recommended structures was always almost a straight adaptation of their generic desktop counterparts, with little optimization concerning the special characteristics of the streaming nature of the whole system. Conventional multi-resolution and out-of-core approaches for web-based LiDAR visualization are characterized by the use of static pre-processed LODs and data structures, such as graphs or tree-like structures. On systems following such approaches, given a point cloud with P points, the total amount of points that must be stored and handled after the pre-processing stage is always greater than P since, depending on the tree structure used and the number of resolution levels computed, several points must be replicated on different LODs.

In some scientific and professional environments, the accuracy of the measurement tools and the fidelity of the 3D point clouds displayed on screen while carrying out visual analyses may be considered as critical requirements of said tasks. In situa-

tions like this, a full-resolution visualization and processing approach acquires some advantages against multi-resolution approaches, since in the latter, the presence of artificial points, included by interpolation or other methods alike, in addition to the strong data simplification applied to create all different LODs, can be counterproductive for the aforementioned requirements.

1.3. Cloud computing and big data

In the last decade, the volume and complexity of all sorts of information generated every day around the world has been growing exponentially, to such an extent that, in some cases, traditional software solutions are simply unable to handle it or they do so in a very inefficient way. In this context arises the term *big data*, and with it a great number of new technologies and paradigms exclusively designed to store, process, visualize and analyse such volumes of information in a fast and efficient way [50].

Currently, there is a wide variety of big data solutions [16, 48] that can be applied in a wide variety of fields, such as healthcare, economics, earth science, education or even sports. Big data solutions bring enormous benefits, such as strong reductions in computational costs and economic resources, or the improvement in the quality of the services provided, which has a direct impact on the customers satisfaction [53].

Based on their main purpose, big data technologies can be divided into two categories: technologies for distributed storage and technologies for distributed computing.

From the point of view of distributed storage, we can find NoSQL databases. These are more scalable and provide much better performance than traditional relational databases. Unlike the latter, NoSQL data models were designed to address issues related to working with massive volumes of new and rapidly changing data types, typically offering dynamic data schemas, data manipulation through an object-oriented API and very straightforward mechanisms of horizontal scalability. In general, the main advantages of these technologies are: low latency and high throughput, horizontal scalability, high availability through data replication, data distribution across several machines or integration with big data computing

frameworks.

NoSQL databases can be divided into the four main categories, listed below:

- **Document stores**: In this type of store, each key is paired with a more or less complex data structure known as a document. Data is mostly processed through notations such as JSON (JavaScript Object Notation); examples: MongoDB [69] and Couchbase [21].
- **Key-value stores**: This type of store is based on a simple schema where items are stored using just an attribute name or key, together with its corresponding value; examples: Redis [83] and Memcached [23].
- **Wide-column stores**: The design of these stores is focused on providing efficient queries over large datasets with dynamic columns; examples: Cassandra [95] and HBase [97].
- **Graph stores**: Stores of this type are employed to handle relations between elements and networks of information; examples: Neo4J [72] and Giraph [101].

From the point of view of distributed computing, two main paradigms can be found: batch and stream processing:

- **Batch processing**: Batch processing involves operating over extremely large collections of static datasets, returning results at a later point in time, only after the whole computational process have finished. Typically, batch processing operates over finite and well delimited elements of data backed by some type of NoSQL store, distributed file system or any other type of permanent store; examples: Hadoop [103], Spark [100] and Flink [102]. Although Flink was designed with stream-oriented workloads in mind, there are modules available to operate as a batch processing framework.
- **Stream processing**: Stream processing frameworks operate over data as they reach the system. Data operations and tasks are only applied over chunks or individual items of data instead of whole datasets. Usually, the total amount of data that is going to be processed is unknown with a event-based processing

mechanism. As new data arrive, new results are generated or previously results are updated, not finishing until the processing is explicitly stopped; examples: Storm [104], Flink [102] and Spark [100]. Spark is in the opposite situation to Flink, as it was designed with batch-oriented workloads in mind, but specific modules have been included to operate over streams of data.

The advantages of following a big data approach in geospatial information contexts have been studied in [63, 66, 112, 115], some of which discuss the benefits of using big data in the specific field of LiDAR [8, 12]. In [8], the storage and querying capabilities of NoSQL technologies are explored, and in [12], it is shown how to take advantage of the usage of the Spark framework on point data visualization.

These LiDAR-oriented publications are part of a much larger literature comparing and analysing big data solutions from similar points of view, such as in [17], where an analysis on how big data may provide added value in the context of remote sensing applications, or in [27] where the performance and reliability of MongoDB, Hadoop and HDFS is analysed from the point of view of scientific data analysis. More general benchmarks and performance comparisons are presented in [1, 20, 81, 110], considering several different big data technologies.

1.3.1. Distributed storage for LiDAR data

During the research carried out in the Chapter 4 of this Thesis, four big data storage technologies were employed: HDFS [96], MongoDB [69], Cassandra [95] and Redis [83]. These four technologies were chosen with a view to including different types of storage design among the most adopted and mature technologies currently available. With them, the following designs have been covered: distributed file systems, document stores, wide column stores and key-value stores. For more information about the usage and popularity of these technologies, a complete list of database management systems can be checked in [92].

We should point out here that, although HDFS is the only technology that is not, strictly speaking, a database but a distributed file system, it is widely used across many big data environments for storing and retrieving files. We chose HDFS over other popular distributed file systems, such as LizardFS or Lustre, because of

their limited integration within the big data ecosystem.

Additionally, graph databases were excluded since they are best-suited for analysing interconnections or networks of information, which does not apply in our use case. Finally, GridFS, a special implementation featured in MongoDB designed for storing and retrieving binary files, was discarded in our analysis due to it only being recommended for storing files exceeding 16 MB. In out-of-core multi-resolution applications, especially in web applications where taking advantage of browser's cache is a key point in performance, the size of the files is much lower than 16 MB.

HDFS

HDFS [96] is a distributed file system used by big data computing technologies such as Hadoop, Spark or Flink. It provides fault tolerance and data replication. Commonly, it operates over files of more than 1 gigabyte or 1 terabyte in size. These files are automatically distributed throughout the nodes of the cluster. Before being distributed, data are divided into chunks of a fixed size. Although the files stored in HDFS are commonly accessed by Hadoop, Spark or Flink, for computational purposes it implements a REST API (WebHDFS) allowing any kind of application to retrieve data directly from the system. The emphasis of HDFS is on high throughput data access rather than low latency data access.

In an HDFS cluster there are 2 main components: the name-nodes and the data-nodes. The data-nodes manage the data storage while the name-nodes act as master servers, managing the file system namespace, regulating the client's/application's access to files among other tasks. HDFS was not designed to work with a large number of small files but with a moderate number of files of a very large size. For performance reasons, each time a name-node starts it loads into main memory all metadata about the files, folders and file chunks contained in the cluster. Each metadata entry leaves a fixed-size memory footprint in the name-node. Either for storing an excessive large number of files/folders, or for using an inappropriate (too small) block size causing large files to be split into a very large number of file chunks, the excessive amounts of entries produced could end up consuming all available memory in the name-node.

MongoDB

MongoDB [69] is a document store where data are handled in the form of JSON-like documents. Aside from implementing the same common features presented in all technologies (data distribution and replication, fault tolerance and high scalability), MongoDB stands out for its automatic data balancing mechanism. The balancing mechanism ensures an even distribution of the data across all nodes of a cluster regardless of its initial distribution.

The main components of a MongoDB cluster are: the shards, the configuration servers and the query routers. A shard is a single MongoDB instance that stores some percentage of the total amount of data contained in the cluster. This role is assumed in the cluster by *mongod* (MongoDB Daemon) processes. No third-party applications can communicate directly to shards, it being necessary to do so through query routers. Query routers are elements in charge of receiving and serving data queries from any kind of application. They are very lightweight and low CPU consuming, so they can be deployed in the same node along with a shard or a configuration server. This role is assumed in the cluster by *mongos* (MongoDB Shard) processes. Configuration servers store the metadata for the cluster. Metadata holds the state and organization of all stored data and the different components of the cluster. Query routers need the information provided by the configuration servers in order to read and write data. This role is also assumed by *mongod* processes and by design, it is mandatory to have at least one node exclusively deployed with this role. Nodes cannot be shards and configuration servers at the same time; accordingly, in a cluster with N nodes only $N - 1$ nodes can be deployed to store dataset.

Cassandra

Cassandra [95] is a wide column store, a class of data base where data are stored in records with the capability of holding very large numbers of dynamic columns. It provides features such as fault tolerance, data distribution and data replication. It is highly scalable, being used by some of the most important companies around the world, such as Apple or Netflix, with deployments of over 75000 nodes in the former case and 2500 in the latter. Its performance has been benchmarked against other solutions in [81], showing a clear advantage of Cassandra over the rest of

alternatives (HBase, Redis, Voldemort, MySQL and VoltDB). Nevertheless, it is important to highlight the years that have passed since the study and that some of the technologies, like Redis, were in an early stage of development at that time.

Redis

Redis [83] is an in-memory, key-value structure store with all the common advantages of big data technologies, data distribution and replication, fault tolerance and high scalability. Redis achieves especially good performance during data readings, thanks to its in-memory design, which allows it to serve all data queries directly from main memory, unlike other solutions, where data are served from disk and only most recently used data are served from main memory.

In a Redis cluster, two types of nodes can be found: masters and slaves. All data stored in Redis are divided throughout the master nodes presented in the cluster. All master nodes have zero or more slaves associated to them, and each slave is in charge of storing replicas of the data from its corresponding master node. In Redis, it is mandatory to deploy at least three master nodes, while the number of slaves depends on the replication factor.

1.3.2. Distributed computing for LiDAR data

GIS elevation models, such as DSMs (Digital Surface Models) or DTMs (Digital Terrain Models), are one of the most important and valuable products derived from LiDAR point clouds, as these raster-type three-dimensional (3D) models are the core element in many geospatial processes, e.g., biomass estimation [80] or linear feature extraction [119]. Additionally, DTMs and DSMs can be used together with their source data for carrying out many different visual analyses or simply to compare the quality of different procedures and techniques employed for their creation [35].

The quality of these elevation models is strongly related to the accurate classification of LiDAR points under the categories of ground or non-ground, usually being this a heavy time-consuming process. Computational times and storage requirements involved in this type of classification may become a critical issue on environments under high data collection rates, such as the already mentioned GIS

centres. Considering the massive volumes of data that must be handled in such environments, during the execution of complex computational tasks a single computer and the use of conventional software solutions may suffer some important problems, such as lack of scalability and availability, low throughput and high latency levels.

During the research carried out in the Chapter 5 of this Thesis, Spark was employed to provide computational capabilities on top of a Cassandra storage system in order to develop a highly scalable solution to execute filtering algorithms for DTM generation (or any other very complex geospatial process) on large collections of massive point clouds.

As briefly commented in Section 1.3, Spark is a fast processing engine compatible with HDFS, HBase, Cassandra, Hive and any Hadoop InputFormat. It was designed to perform batch processing in addition to other types of modern workloads such as streaming, interactive queries or machine learning. Spark has been employed by many major brands, companies and organizations, many of them with clusters of thousands of nodes deployed in their facilities and labs. In terms of data size, Spark is able to work properly up to petabytes.

Chapter 2

Interactive full-resolution visualization and processing of large aerial LiDAR point clouds

The first stage of this Thesis focused on full-resolution point cloud rendering since it was considered the most suitable approach for accurate measuring and visual analysis under high image fidelity requirements. Nevertheless, this type of approach demands a considerable amount of hardware resources in order to offer interactive levels on visualization and processing.

In this chapter, several optimizations for performance maximization on full-resolution rendering and processing are presented, some of which based on a bachelor's thesis[1]. Said strategies have been grouped into two categories: strategies for fast data loading and strategies for high performance rendering. These strategies were incorporated and tested into a visualization framework specially developed to do so and called *GVLiDAR*. The use of an own visualization framework ensured entire freedom during all development stages, avoiding all possible limitations and constraints that cloud appear when using as development foundation a pre-existent API or software package. Additionally, some key features developed and included in *GVLiDAR* are also presented here as they were included to grant flexibility and adequate workflow and to provide field-specific geospatial measurement tools, features

[1]Bachelor's thesis permanent link: http://kmelot.biblioteca.udc.es/record=b1514673 S1*gag

that are particularly relevant in some professional areas like agroforestry.

The rest of the chapter is organized as follows. Section 2.1 describes the internal structure and main components of *GVLiDAR*. The data querying method included in the framework is described in Section 2.2. In Section 2.3, all measurement capabilities of the framework are explained. Strategies for fast data loading are described in Section 2.4, while strategies for full-resolution rendering are described in Section 2.5. In Section 2.6, several performance tests are shown. The work presented in this chapter was originally introduced in [34] and [35].

2.1. System structure

GVLiDAR is a web-based LiDAR visualization framework specially designed to carry out detailed visual analysis and to make complex measurements over fully detailed 3D point clouds. It allows users to retrieve remote data on-demand through queries based on spatial restrictions. By defining regions of interest (ROI), it is possible to visualize and process sub-areas of a much larger point cloud without transferring unnecessary points. Multiple source files acquired during different scanning flights can be pre-processed and stored together on the server allowing, for instance, the visualization of a road stored over multiple files or datasets.

In some technical disciplines, not only is a realistic representation of the data necessary, but also the capacity to take fast and accurate measurements over the 3D objects and terrains under study. Considering this, our proposal does not follow a multi-resolution and out-of-core approach, but an approach to obtain great performance rendering point clouds at full-resolution. Furthermore, all measurement tools included in *GVLiDAR* were developed by consulting several experts in the field of agronomy engineering trying to ensure the maximum usefulness of the tools.

Figure 2.1 shows the general system environment. This framework follows a common client-server architecture, having two clearly separated parts. The first part is an HTTP web server, in our case Apache HTTP Server [98], which stores the application source code and the LiDAR files. The second part includes any client-side WebGL compatible web browser. This type of system allows multiple users to concurrently access the same data through their browsers, rendering the final image

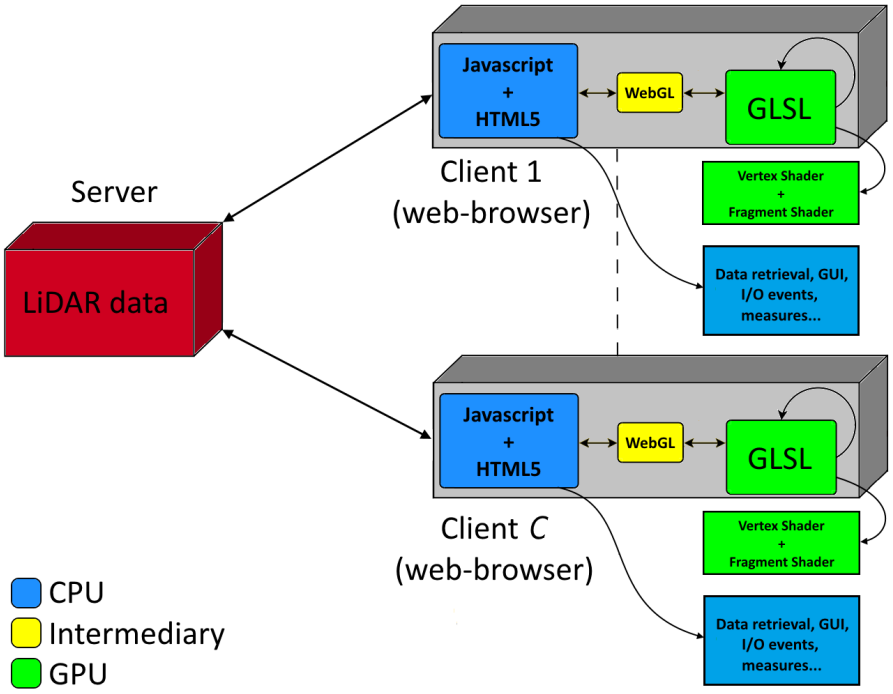


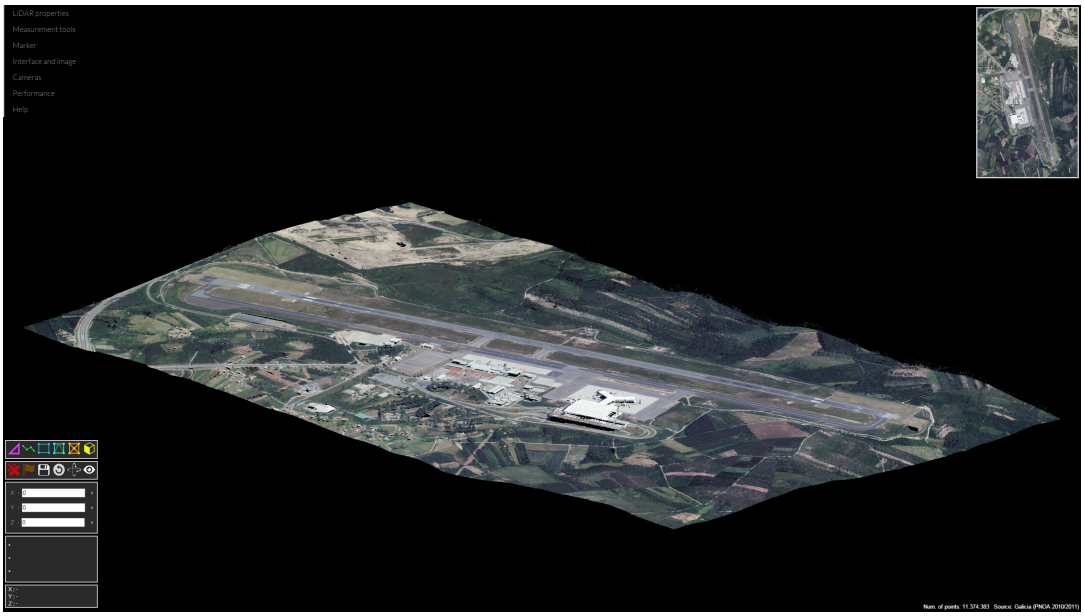
Figure 2.1: General structure of *GVLiDAR*.

on their own platform.

GVLiDAR uses three different programming languages and the WebGL graphics API. HTML5 and JavaScript, both executed by the CPU, are responsible for tasks such as handling I/O events, generating the user interface, requesting and loading LiDAR data from remote servers and computing the geospatial measurements. Although WebGL is also JavaScript code, it is a very distinctive part of the framework. It is the intermediary between the CPU and GPU, allowing communication and information delivery to the GPU. Finally, OpenGL Shading Language (GLSL) [71,105] is used to program the GPU shaders, *Vertex Shader* and *Fragment Shader*. *Shaders* are programs that provide a significant flexibility in order to implement rendering and computational algorithms on GPUs. In *GVLiDAR*, shaders are used for tasks such as point colour generation, point erasing or mouse picking, which allows the geospatial measurements tools to be implemented efficiently.

From a user's perspective, the framework consists of two separate web pages. The *Selection* page (see Figure 2.2a) shows a map through GoogleMaps API 3.0.

(a)



(b)

Figure 2.2: (a) *Selection* page of *GVLiDAR* over Santiago de Compostela (Spain) International Airport. In this page we use the GoogleMaps API in order to help users to locate their areas of interest. The green rectangle represents the area of the map that contains point data while the blue rectangle represents the specific data selection performed by the user. (b) *Visualization* page of *GVLiDAR* rendering the point cloud contained in the ROI defined by the user in the *Selection* page.

Over this map users are able to obtain their geographic location or define an ROI to work with. Once users have defined the ROI, the *Visualization* page (see Figure 2.2b) pops up showing the rendered model of the point cloud contained in the ROI. In this page, different LiDAR properties such as height, intensity, classification or return number can be chosen in order to change the way the scene is displayed. There are also different camera options, marker tools and rendering options, such as point size, line width or projection mode.

2.2. Remote data querying

During a normal workflow, it may be necessary to select a small sub-region of a much larger area to work with, which may involve downloading a large file containing many more points than necessary, even for multi-resolution approaches. Remote data queries based on spatial restrictions were implemented through the use of spatial hashing techniques. Spatial hashing allows fast and efficiently remote data to be obtained from the server without requiring any additional back-end software, like a data base. The coordinates delimiting the user-defined ROI can be transformed into the positions of tiles belonging to a regular grid, each tile being related to a specific file on the server.

Figure 2.3 shows a very simple example of the process. The yellow circles are user-defined coordinates which delimit the ROI. This is done in the Selection Page described on Section 2.1. Dark tiles represent LiDAR files created during the pre-processing stage, while the inner blue area is the user-defined ROI. The list of files that must be retrieved from the server are obtained from the geographical coordinates of the two points (yellow circles) delimiting the ROI using spatial hashing techniques. For example, $(43.3357, -8.4542)$ is converted into [4_1], which is the ID of the file under the point; the rest of the files are derived based on the shape of the ROI. In the example depicted in the figure, the retrieval of the fourth column of tiles is not required, reducing load times, saving memory and CPU resources. The points outside the ROI contained in the rest of tiles are discarded rapidly on the client side during the load process.

Figure 2.3: Simple schema representing a regular tile grid distribution with a user-defined ROI over it. The list of files that must be retrieved from the server are obtained from the coordinates of the two points delimiting the ROI using spatial hashing techniques.

2.3. Geospatial measurement tools

Some common measurement tools were developed to be included in *GVLiDAR*, for example, the projected flat area shown in Figure 2.4. Nevertheless, one of the main differences between *GVLiDAR* and most of the frameworks listed in Section 1.1.3 is the inclusion of field-specific geospatial measurement tools that can be performed directly over the 3D point clouds. Many of aforementioned frameworks offer very basic measurement tools, or they do not fulfil specific requirements of professionals on the LiDAR field, as is the case for the agroforestry experts that were consulted for this research.

A good example of this lack of specific features can be found while measuring the height of some structures or terrain shapes. The measurement mechanics of other frameworks require picking up or selecting rendered points on the image, so height measurements like the one shown in Figure 2.5 cannot be directly obtained or even

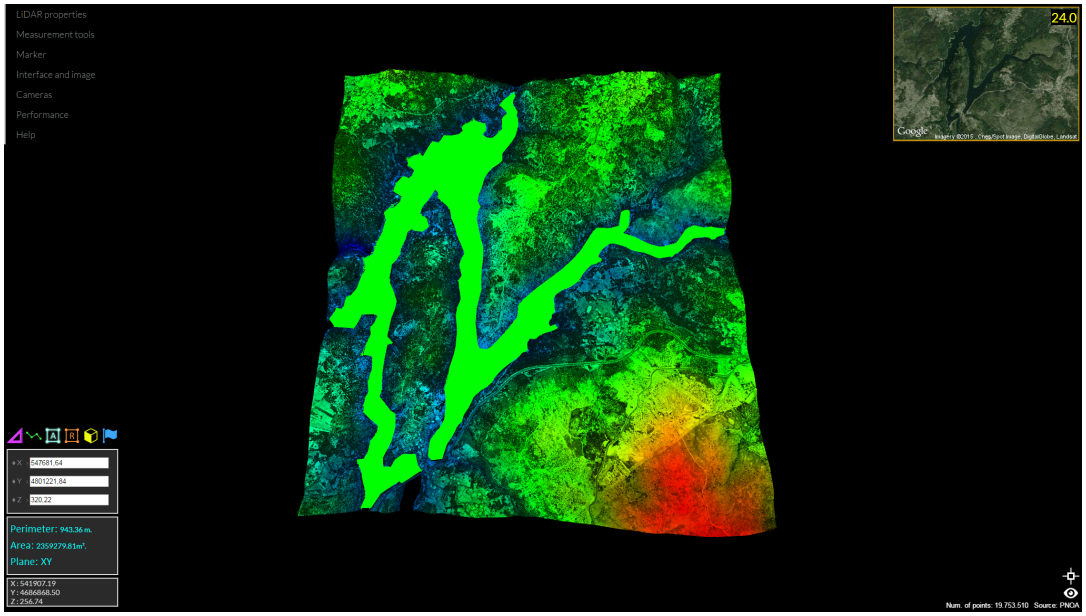


Figure 2.4: Projected flat area measurement (bright green area) covering the entire surface of a river.

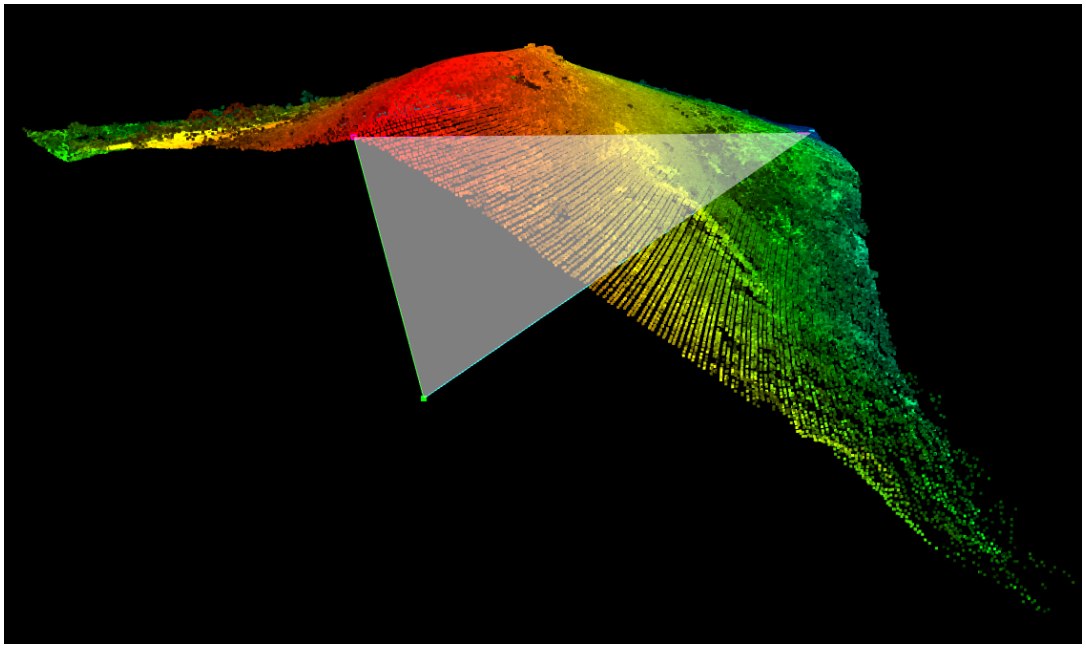


Figure 2.5: Vertical height measurement of a mountain.

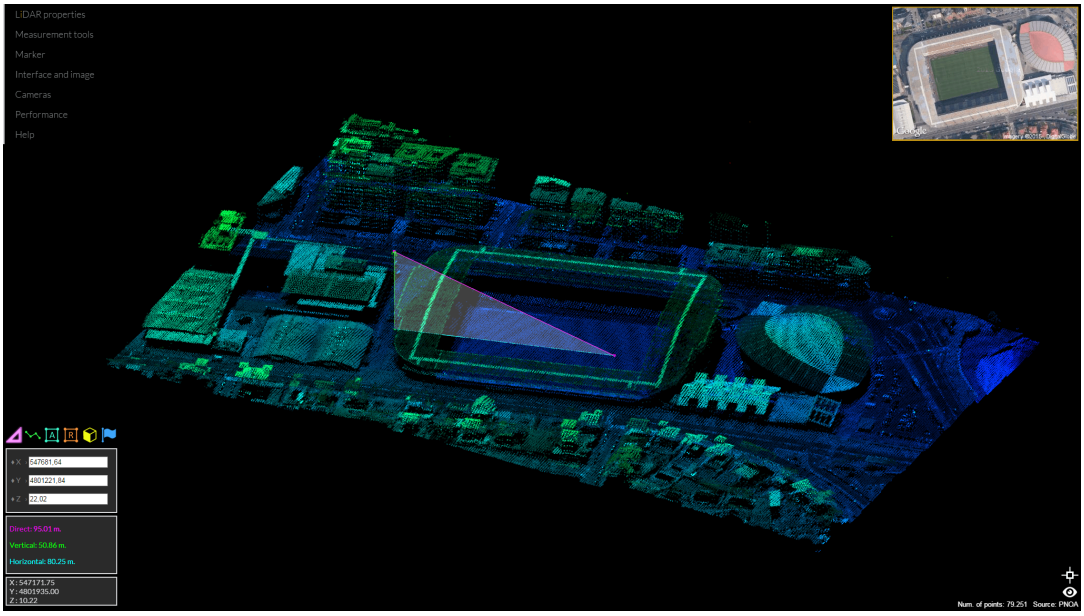


Figure 2.6: Vertical height measurement in Riazor stadium (A Coruña, Spain).

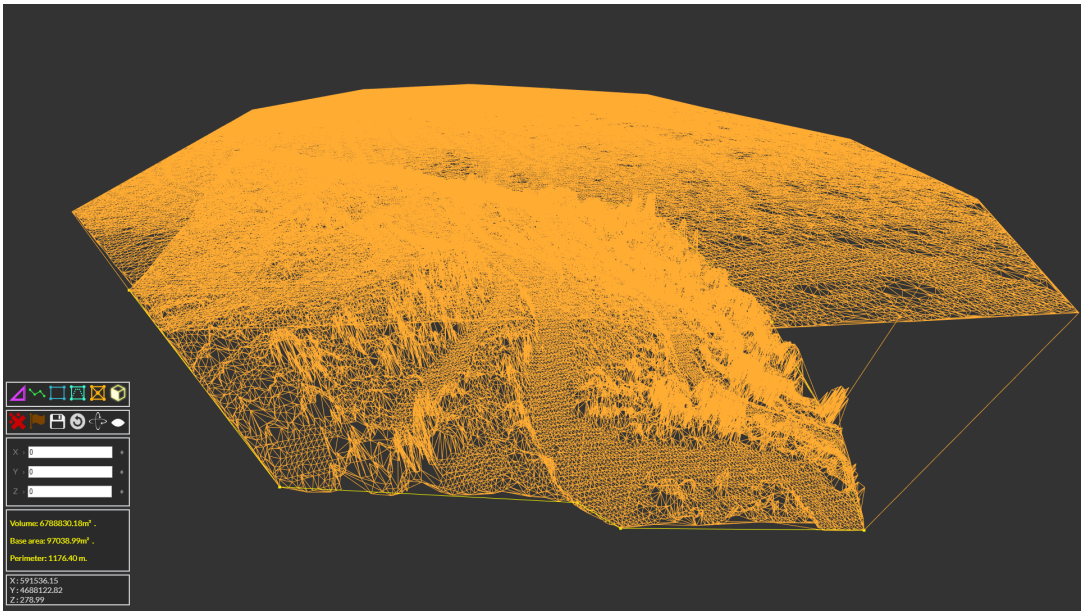


Figure 2.7: Complex volumetric object created using a triangulated base surface, a projected top and an irregular contour.

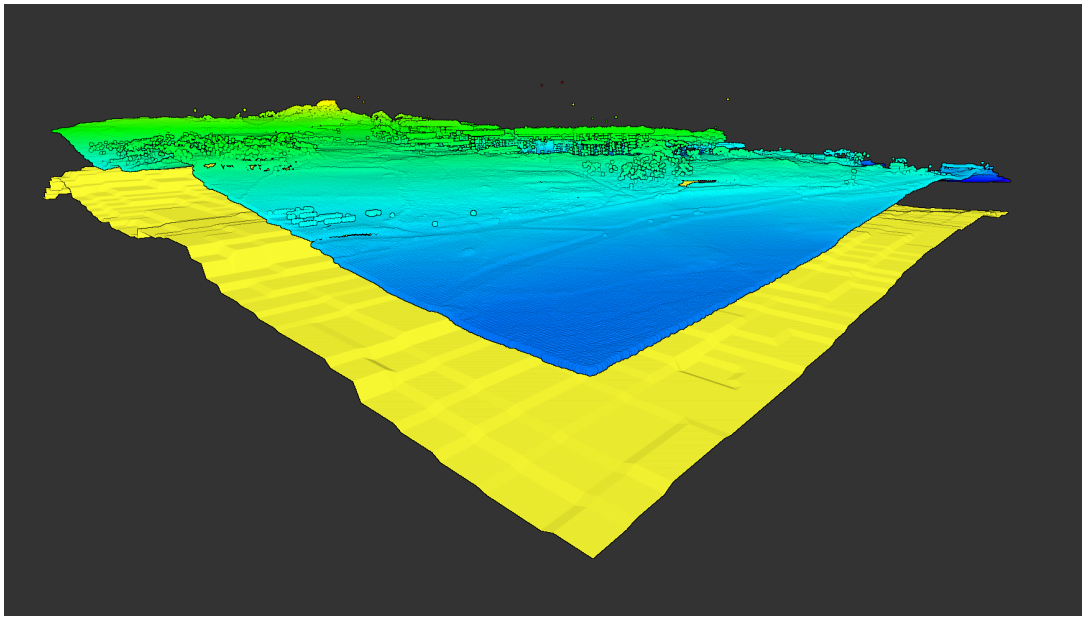


Figure 2.8: DTM overlapping a LiDAR point cloud. This method helps to compare the quality of the DTM in comparison to the source material (the point cloud).

obtained at all. Said figure shows a vertical height measurement (white triangle) of a mountain from its top down to a flat plane located at the level of a nearby river. The green vertical line determines the distance from the top to its virtual bottom. The green dot (lower) is generated by *GVLiDAR* from the pink (upper) and the blue (right) points, which were selected by the user. In all frameworks listed in Section 1.1.3, dots of this type (like the lower one) cannot be created or defined, only allowing the height of structures to be obtained directly in scenarios such as the one shown in Figure 2.6, where selectable points are available at the bottom of the structure. Other unique measurement tools not presented include: the capability of defining flat areas projected in arbitrary planes allowing to measure façades or slopes and tools to accurately measure highly complex volumetric objects (see Figure 2.7).

Another feature presented in *GVLiDAR* is the capacity of overlapping the point cloud over a DTM. This, for instance, allows researchers who develop algorithms for DTM obtention to compare their algorithm results (the DTMs) with their source data (point clouds), allowing them to judge the quality of any DTM overlapping it with a high dense and precise point cloud. These type of visual analysis demands to render as many points as possible in order to obtain good results. Inaccurate

point clouds derived from hard under sampling or point interpolations may not have enough quality for these type of analysis or other accurate measurements. Figure 2.8 shows a DTM overlapping a LiDAR point cloud obtained from different sources, this test allows us to compare the quality of the first one.

As a summary for this section, it can be asserted that most of the measurement tools available on other frameworks are far from being appropriate for many professionals in fields using LiDAR data, such as agronomy, urban planning or civil engineering.

2.4. Strategies for fast data loading

In this section, a series of strategies for fast data loading are presented. They were implemented in order to minimize data loading times by optimizing the size of the LiDAR data and the reading functions and by making efficient remote data transmissions.

All strategies described in following subsections are carried out during a preprocessing offline stage and the results permanently stored on the server-side to be retrieved through Apache HTTP Server by any client connected to the system.

2.4.1. Data cleaning and transformation

In practice, some parts of the information stored in LAS files are often not useful from the point of view of visualization or the geospatial measurements, which unnecessarily increases the size of these files, demanding longer download times and storage capacity. Owing to this, it is essential to apply some kind of compression or file optimization to the original LiDAR file format in order to reduce the amount of data to be stored, transferred and handled. For example, the entire LiDAR dataset of Galicia (a region within Spain) gathered by the National Program of Aerial Ortophography (Plan Nacional de Ortofotografía Aérea, PNOA [54]) has around 900 GB of information; nonetheless, after removing all the unnecessary data, its size is reduced to 450 GB, a reduction of 50%. The superfluous information in these files includes: the entire header block, all VLRs, all EVLRs and point properties, such

as *Scan Direction Flag*, *Edge of Flight Line*, *Scan Angle Rank*, *User Data* and *Point Source ID*. Furthermore, some LAS files may contain fields with all their values set to *null* or zero, so they become also useless data. PNOA files have a *RGB* property in each point record but it is always set to zero, which represents 12 bytes of worthless information per point, so that this property is also deleted.

Additionally, the group of agroforestry experts consulted was not interested in the numeric values of the return properties; instead, they demanded information in the form of return tags; that is, a classification of each point based on its return information mixing the return number of the point and the number of returns of its pulse. Thus, 5 tags or categories were created, allowing this information to be stored in 3 bits instead of the previously 5 bits used in the ASPRS specification (see Section 1.1.2). These 5 categories allow the point clouds to be filtered in many useful ways, while helping to further reduce the size of the files. The return tags are defined as follows:

1. First return (return number 1 out of R, being $R > 1$). This tag corresponds to objects like forest canopy.
2. Middle return (return number r out of R, being $R > 2$, $r \neq 1$ and $r \neq R$). This tag represents points placed between the canopy and the ground, such as branches or leaves.
3. Last return (return number R out of R, being $R > 1$). A last return represents points placed in the ground.
4. No return (return number 0 out of 0). This tag denotes an artificial point. These are points added to the point cloud after being collected by the laser scanner. Normally used to outline rivers or lakes.
5. Unique return (return number 1 out of 1 pulse). This tag correspond to points obtained from a solid surface such as buildings, roads or stones.

2.4.2. Bulk data operations

The two most common ways of storing vertex, or point attributes, in the GPU memory are the array of structures (AOS) and the structure of arrays (SOA). Figures

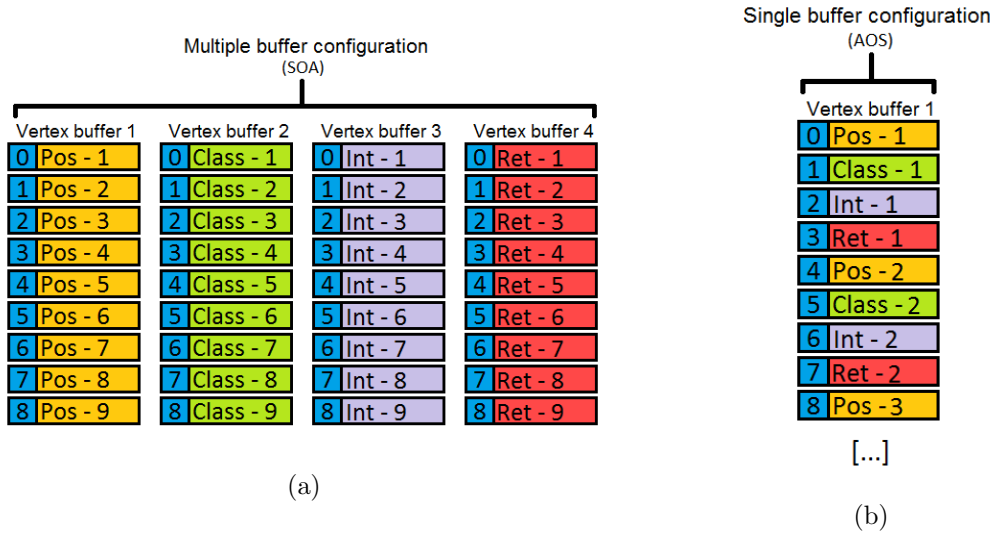


Figure 2.9: The two most common ways of storing vertex attributes: 2.9a Structure of arrays (SOA). 2.9b Array of structures (AOS).

2.9 shows the main differences between the two types. The AOS (Figure 2.9b) is implemented by using a single buffer configuration, storing all point attributes together. On the other hand, the SOA (Figure 2.9a) is implemented by using a multiple buffer configuration; each buffer stores one type of attribute.

In original LAS files, all properties are stored *per point*, using an AOS approach, while for the design of *GVLiDAR* the SOA was chosen to program the GPU shaders in order to obtain better performance. The fact that original LAS files and inputs of the *Vertex Shaders* use different kinds of structures demands a structure transformation from AOS to SOA. Performing this transformation in the client would be inefficient; hence, it is done in the server during the pre-processing stage.

Another benefit of the SOA approach is that entire sequential block of the same type of information can be read out of the retrieved files (bulk data reading) and sent directly to the GPU buffers (bulk data writing), since the values of all points are packed together based on their type. For example, the entire block of coordinates (x, y, z) can be read using a single JavaScript function knowing the offset and length of the coordinates inside the data pack obtained from the server, and the same principle can be applied to the rest of point properties, such as classification, return or intensity. On the other hand, in an AOS approach, the value of each property

Table 2.1: Basic format structure of a pre-processed LAS file containing P points.

Item	**Size (Bytes)**
x, y, z	$3 \times 4 \times P$
Intensity	$4 \times P$
Ret. + Class.	P

from each point must be individually read and rearranged in order to be stored in the corresponding data structures, heavily penalizing data load times.

The (x, y, z) values stored in the original LAS files (and sometimes in other file formats) are not the actual geographical coordinates of the points; each coordinate has to be adjusted according to a scale and offset. Thereby, as an additional improvement to accelerate data reading, during the pre-processing stage the coordinate values in the files are analysed. If they can be stored as 32 bits float numbers without losing precision, they are stored this way. This prevents carrying out extra operations on the client side each time the point clouds are loaded. If the use of floating point numbers implies losing precision, offsets and scales are used. Each dataset owns a metadata file (in JSON format) containing all information required to retrieve, load and render the dataset properly. These metadata files store information such as the name of the dataset, its coordinate system and the offset and scale (if used) of the point clouds.

Table 2.1 shows an example of the final internal data distribution and total file size of the pre-processed files after applying the changes described in this and previous subsections. Here it should be noted that, in case additional properties would be required, such as GPS Time or RGB, they could be easily included in auxiliary files if they were required in the future. These files would be stored apart from the current data and only retrieved from server if required. This provides flexibility and scalability, allowing new properties to be included as needed and even including custom properties not included in the LAS format. If new information were included, the auxiliary files would not be sent to the majority of the clients that simply wish to visualize the point clouds using basic properties, they would be retrieved from server only when required by a specific client.

2.4.3. Data caching

The large sizes of the original LiDAR files make hard to implement efficient spatial hashing techniques since some ROIs may need to retrieve large tiles containing an excessive amount of unnecessary information located outside the ROI. To solve this problem, all LiDAR files were subdivided into smaller files of variable size ensuring that all stay below 5 MB. The limit of 5 MB allows the optimal utilization of web cache of the most used browsers.

A web cache system stores web documents on local disk so subsequent requests of the same data may be retrieved from the cache instead of from the remote server. The small size of the files along with a data distribution in the form of a regular tile grid allows *GVLiDAR* to perform efficient data queries based on spatial restrictions while taking advantage of the browser's cache system.

Following the example depicted in Figure 2.3, 12 tiles, from [1_3] to [4_1], are retrieved from the server and stored in local cache. These tiles will be load immediately on later situations when defining an ROI over the same already cached tiles. The points contained in the tiles partially covered by the ROI are rapidly discarded client-side during the load process to avoid being rendered on screen. These discarded points serve as pre-cached data that will also be loaded immediately when defining an adjacent ROI.

2.5. Strategies for high performance full-resolution rendering

A full-resolution approach, like the one presented here, may become a challenge when it comes to obtaining good performance rendering some point clouds with a large number of points. WebGL is the foundation for accomplishing such a task, thanks to its capability to exploit the power of the client's GPU from a web browser. Nonetheless, additional rendering optimizations are required to reach good frame rates during the visualization process. In *GVLiDAR* two main strategies were implemented: a common view-frustum-culling technique and a technique very similar to an occlusion-culling.

- **View Frustum Culling**: The idea behind this technique is to avoid the rendering of objects that lie completely outside the viewing frustum of the camera in the 3D scene, saving GPU processing power and increasing the frame rate. For this, LiDAR files contained in the ROI are managed by a tree-like structure, so it can be efficiently traversed to determine which sections of the point cloud are visible and which are not.

- **Occlusion-culling**: This technique follows a very similar concept, avoiding the rendering of objects that are not visible because of being occluded behind other objects. *GVLiDAR* implements an approximation of this idea by changing the value of the parameter *stride* in the WebGL function *vertexAttribPointer*. Some sections of the point cloud may be located at a certain distance from the camera, so their points are so close on screen that they overlap and some are drawn on top of the others. The function *vertexAttribPointer* helps to mitigate this issue while increasing the frame rate, by allowing the GPU to render only ranges of points instead of rendering them all.

More information about this type of rendering optimizations can be found in [2].

2.6. Results and comparison

In this section, the results of the evaluation of our proposals are presented and analysed encompassing three different key points: the analysis of the functionality, the performance in terms of FPS and the data retrieval times observed. Our test platform is an Intel Core i7 4790, 32 GB of RAM DDR3 with a Nvidia GeForce GTX Titan (Maxwell). The screen resolution was 2048 × 1152 under a set of different browsers.

Two datasets, collected through airborne sensors, were used for the analyses: *PNOA* and *CA13*. The *PNOA* dataset is available in the Spanish GIS database (IDEE) [54]. Specifically, we have used the region of Galicia, containing over 27.6 billion points (see Figure 2.10) with a point density of 0.5 $point/m^2$. *CA13* [74] (see Figure 2.11) is located in San Luis Obispo County (California), it contains 17.7 billion points with a point density of 22.06 $points/m^2$.

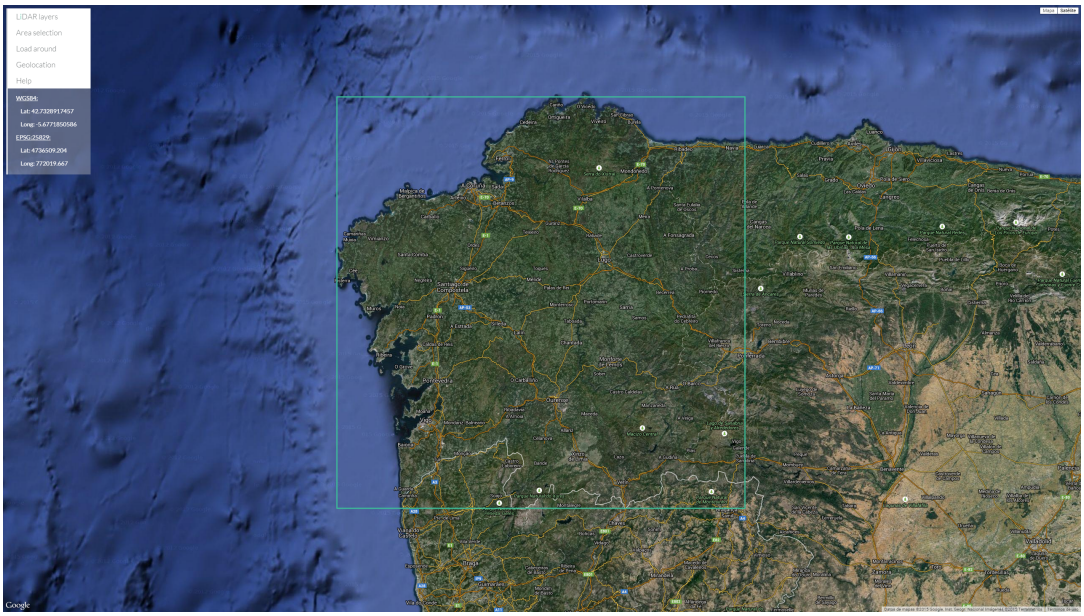


Figure 2.10: *Selection* page of *GVLiDAR*. GoogleMaps API was employed in order to show users the datasets available and their extension. The green square delimits the area with point data, in this case, the *PNOA* (Galicia, Spain) dataset used in this paper.

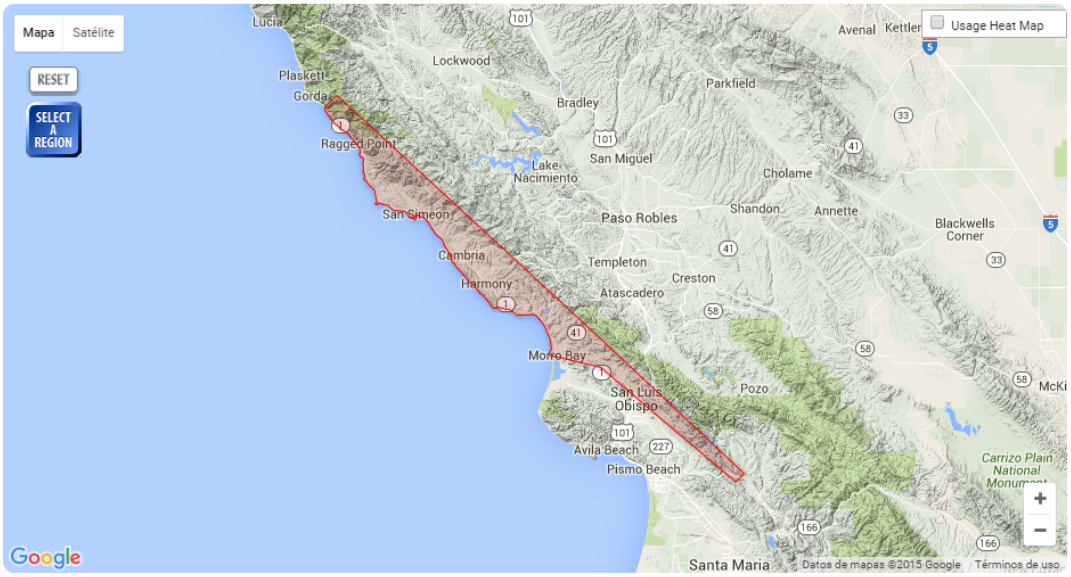


Figure 2.11: OpenTopography website, specifically its data selection tool. The red zone indicates *CA13* dataset, located in San Luis Obispo County, California.

2.6.1. Functionality and workflow

The functionality and workflow test presented in this subsection was carried out using the web browser Windows Chrome 42.0.2311.135 (64 bits) and the *CA13* dataset. *GVLiDAR* is compared to *Potree*, since among all frameworks described in Section 1.1.3, it is the most similar to *GVLiDAR*.

For the test, we have recreated a use case where a user must take accurate measurements of the length of roads, which is a common use case for some institutions responsible for calculating the cost of road maintenance. In general, the measurement process requires working from a point of view close to the road in order to carry out a fine-grained point picking and thus obtain very accurate measurements. Figure 2.12 shows the zone of road that we want to measure, starting at position A and finishing at position B.

Potree visualizes up to 1 million points using a multi-resolution technique in order to maintain good frame rates and low memory consumption. This limit can be raised up to 4 million; nevertheless, the default configuration of the framework was used due to 1 million points being sufficient to show LiDAR data with high detail in order to measure the road. *Potree* uses around 1.3 GB of RAM at the beginning of the procedure of measurement on position A; however, it needs up to 4GB of RAM when the user reaches position B. All points rendered during the measurement procedure are kept in browser memory which could reach the limitations of software or hardware due to the requirements of memory used. Chrome has a limit of 4GB of RAM per tab; therefore, *Potree* provokes the closing of its tab during the test losing all the measurements. This memory limitation can be found in all 32 bit web browsers and the 64 bit versions of Chrome for Windows.

GVLiDAR was used for taking the same measurement but using its data querying capabilities. Figure 2.13 shows the section of the road loaded from position A to position B with a total memory usage of 500 MB of RAM and 800 of VRAM. *GVLiDAR* shows a considerable margin regarding Chrome's 4 GB limit allowing the completion of the whole process without problems and using the maximum resolution along all road sections.

In summary, with high levels of resolution and a prolonged and intense use of the application, *Potree* may fail due to the high memory requirements related to

Figure 2.12: Distance measurement of one of the roads contained in the $CA13$ LiDAR dataset, from A to B, using GoogleMaps only.

Figure 2.13: *GVLiDAR* point cloud render of the road from $CA13$ with the same distance measurement previously taken in GoogleMaps.

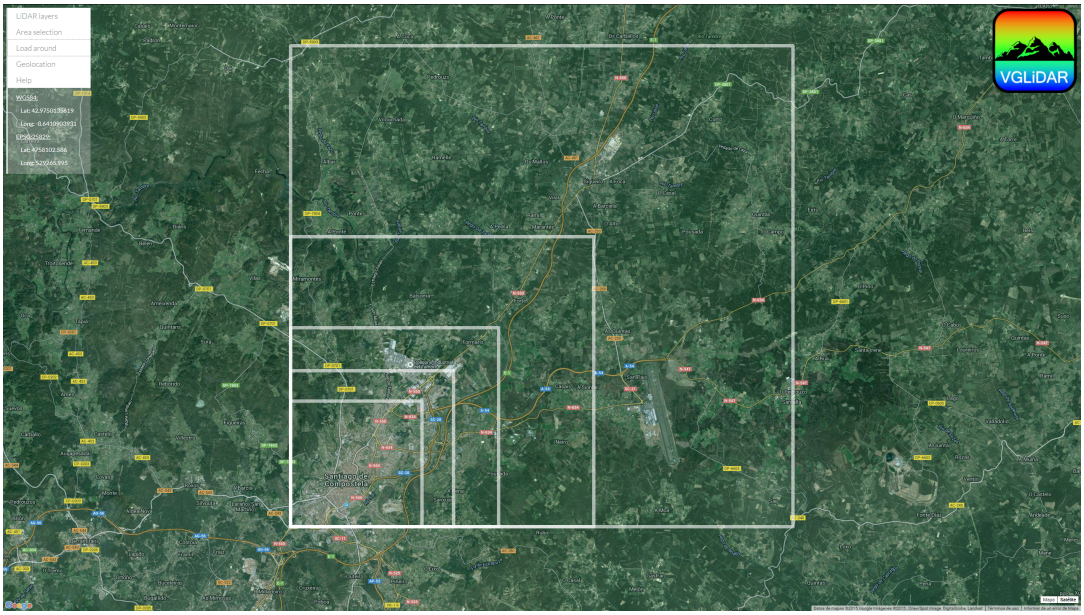
the use and handling of the multiple LODs. On the other hand, *GVLiDAR* exploits the benefits of the use of an ROI, minimizing memory requirements while displaying fully detailed point clouds. We should stress here that this analysis was carried out with the aim of highlighting the importance and usefulness of the use of the ROI, showing how, for some working contexts, even a single-resolution approach could get better performance and image quality compared to a multi-resolution approach.

2.6.2. Performance in terms of FPS

For this analysis, a performance test in terms of FPS was carried out on the same hardware described at the beginning of this section, but with the browser Firefox 41 (64 bits) instead of Chrome, in order to avoid the 4 GB limit. The five point clouds used for this test are shown in Figure 2.14 highlighted with white squares (see Figure 2.14(a)). Figure 2.14(b) shows a close view over the largest ROI with a maximum amount of points of 281,152,780.

Figure 2.15 shows the results of the performance tests in terms of frame rate (FPS) for each ROI. For each ROI, the maximum number of points rendered is shown in the horizontal axis. Two different points of view, $VP1$ and $VP2$, were used to place the camera, specifically, for $VP1$ the camera is located at 1/4 of the full 2D convex hull while for $VP2$ it is located at 3/4. These two points of view were used to analyse the benefits of the rendering optimization techniques described in Section 2.5.

Results show that, when 20,179,576 points are rendered, up to 60 FPS are achieved. For larger regions (up to 51 million points), *GVLiDAR* achieves real-time interaction (above 20 FPS) for both VP1 and VP2. Even rendering 103 million points, it is capable of reaching 42 FPS for VP2. It should be noted that *GVLiDAR* is able to render up to 281 million points, which is not achieved in other applications, with the exception of *Megatree* and *Potree*, which manage to achieve good frame rates by using multi-resolution approaches, although they really never show more that 4 or 5 million points on screen. Frameworks such as *IDECanarias* or *Lidar-Online* show a similar performance for a small number of points rendered (350,000 or 1,000,000), while although *FugroViewer* and *Global Mapper*, also employ optimization techniques very similar to a multi-resolution approach, they only show



(a)

(b)

Figure 2.14: Test point clouds: (a) Each white square highlights a different ROI; (b) Point cloud rendered on the *Visualization* page using a close view over the largest ROI.

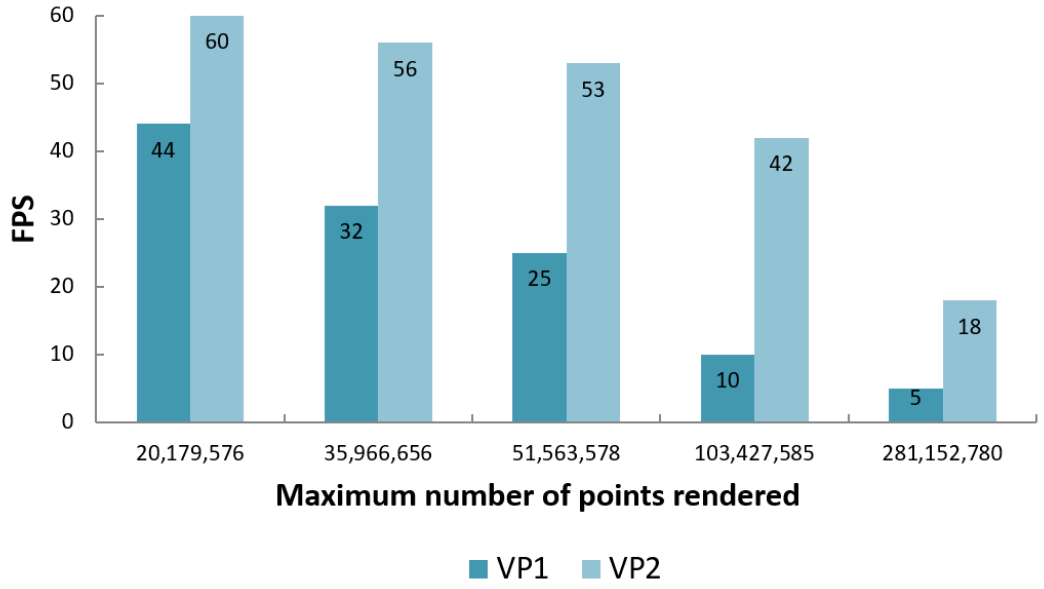


Figure 2.15: *GVLiDAR* Performance in terms of FPS using different amount of points to render in the *Visualization* page.

stable behaviour under a limited amount of points, getting stuck and failing when surpassing 10 million points.

These performance tests have led us to establish two different theoretical point limits in *GVLiDAR*, considering the current GPU market. The maximum amount of points that can be stored in the 12 GB of VRAM available on the Titan X (Maxwell) is 281 million, meanwhile, 103 million points is the performance limit of the GPU, that is, the maximum amount of points that is able to handle with relative good performance (between 10 and 42 FPS).

2.6.3. Data retrieval and data load times

For this test, retrieval times and load times were taken to confirm the practical usability of *GVLiDAR*. Retrieval times are considered as the time required for moving the LiDAR data from the server side to the client machine, while load times are considered as the time it takes to read, set and send the data already retrieved from server to the GPU buffers. The sum of both times (retrieval + load) is called *wait time* and it represents the elapsed time between a user making a data query

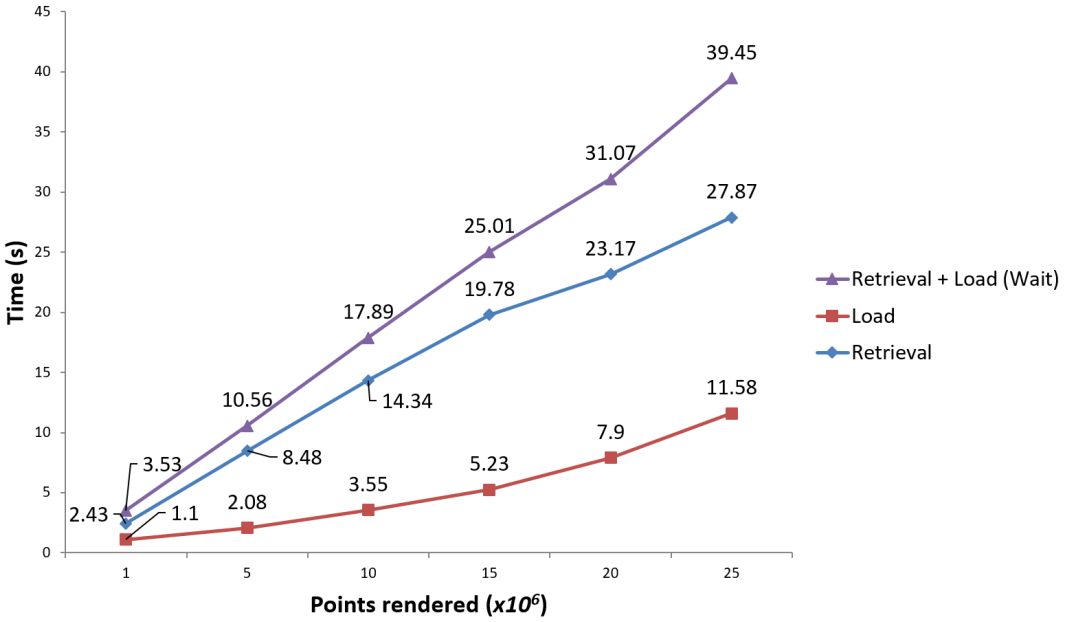


Figure 2.16: Retrieval times from remote server.

and the point cloud being rendered on the screen. This test was carried out using an Apache 2.215 server, running CentOS 6 on an Intel Xeon ES-2603v3, 64 GB of RAM and a 4TB SATA3 hard drive. The network transfer speed is 90 Mbps and the browser used was Firefox 41 (64 bits).

Figure 2.16 shows how the load, the retrieval and the wait time vary with the number of points requested. The graph reflects the worst possible scenario, where all data selected by the user must be entirely retrieved from server. It is quite clear from the graph that times increase almost linearly along with the number of points rendered due to the size of each point always being the same, in this case 17 bytes.

Considering the best possible scenario, where all data contained in the ROI are already cached in the client disk (as explained in Section 2.4.3), the retrieval time would be zero, making the wait time equal to the load time.

Although the time spent retrieving non-cached data may become high when starting to retrieve more than 25 million points, these results were obtained without using any type of compression algorithms. As already mentioned at the beginning of this Thesis, all the research focused mainly on the performance and functionality of the client side, with the strategies outlined in Section 2.4 being only an initial,

but efficient, approach.

Chapter 3

A multi-resolution, out-of-core approach for rendering massive aerial LiDAR point clouds

In this chapter, novel multi-resolution and out-of-core techniques for web-based visualization are presented in detail, the key performance elements thereof being a non-redundant data design and a runtime-only LOD computation system. Our approach is based on a special point data organization called *Hierarchically Layered Tiles* (HLT) together with a tree-like structure called *Tile Grid Partitioning Tree* (TGPT). The non-redundant data nature of the new approach, together with a lossless compression method specially developed to be applied on the LiDAR point clouds, allowed us to greatly reduce the amounts of data handled on both the client and server sides. Reduction in storage requirements is particularly relevant, being notably lower compared to traditional multi-resolution approaches using conventional static precomputed models with high levels of data redundancy. The concept of layered points with a view to avoiding redundant data was explored in [43], but with totally different goals and a different execution context.

The new approach was tested in *ViLMA* (**Vi**sualization for **L**iDAR data using a **M**ulti-resolution **A**pproach), a web-based visualization framework that was built using many of the main features of the software developed during the first stage of the Thesis and presented in the previous chapter. Unlike *GVLiDAR*, *ViLMA* was

designed to visually explore massive LiDAR point clouds, with visual fidelity being secondary in relation to performance (see Section 1.2.1 for more information about performance/visual-fidelity approaches).

In Section 3.4.6, several performance optimizations were validated though a performance comparison between *Potree* and *ViLMA*. For the same reasons explained in the previous chapter, *Potree* was selected due to its being, among the currently available web applications (see Section 1.1.3), one of the most well known and highly valued web-based visualization options and the most similar to *ViLMA*. 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*, as it is only a means to test our proposals.

The rest of the chapter is organized as follows. In Section 3.1, the general system structure and design of *ViLMA* is described. Section 3.2 presents the HLT and TGPT data structures. In Section 3.3, several design decisions regarding performance are discussed, while in Section 3.4, proposals are analysed using *ViLMA*. The work presented in this chapter was originally introduced in [32].

3.1. Structure of ViLMA

ViLMA is a web-based application designed for the interactive 3D visualization and exploration of massive LiDAR point clouds. It shares with *GVLiDAR* some of its main features, such as the field-specific geospatial measurement tools and the data queries based on spatial restrictions, but pursuing different goals for visualization and measurement accuracy, as was already mentioned in previous paragraphs.

Figure 3.1 shows the general system structure of *ViLMA*. The front-end, written in JavaScript and HTML5, can be executed in any WebGL-compatible web browser. For the back-end, an Apache HTTP Server [98] was deployed for serving the application code and the point data requests. *ViLMA* was also designed to work with local datasets in addition to the public datasets available online; hence users may select a local directory from which their own pre-processed LiDAR point clouds can be loaded. As shown in Figure 3.1, the structure of *ViLMA* is divided into two different stages: the *Pre-processing Stage* and the *Interactive Stage*.

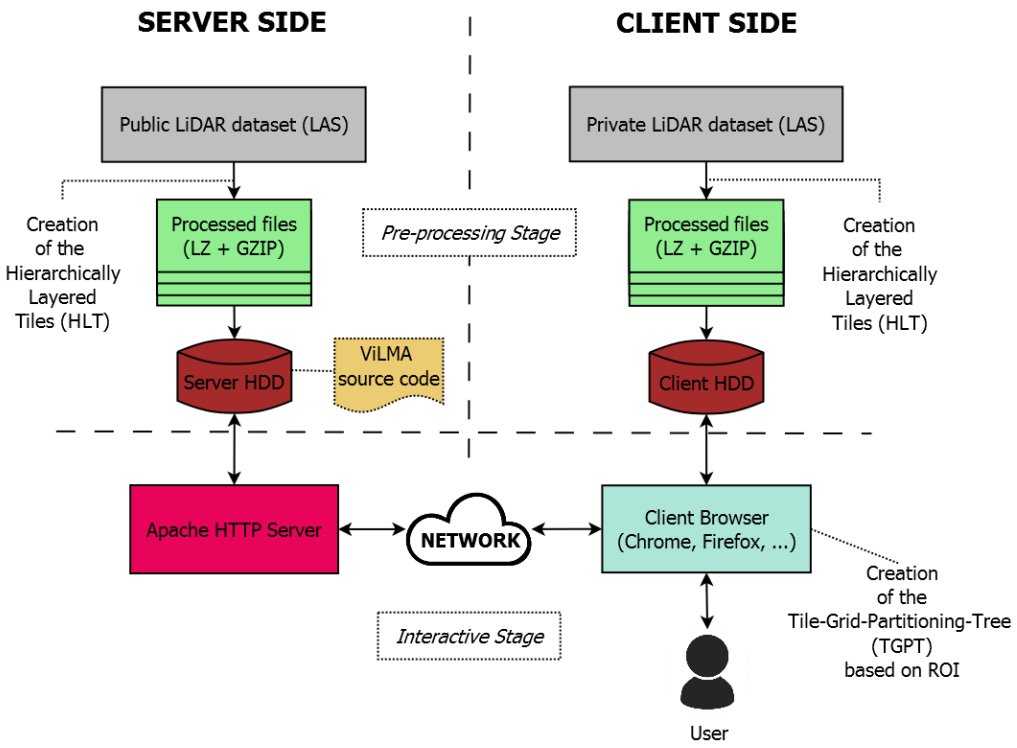


Figure 3.1: General system structure of *ViLMA*.

The *Pre-processing Stage* takes place off-line on the server side and/or on the client side. Datasets pre-processed on the server side are intended to be public and accessed online through the Apache server; meanwhile, datasets pre-processed on the client side are loaded directly by *ViLMA* from the user's local disk. During this 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 a lossless compression method applied over the pre-processed data (see Section 3.3.3).

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 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 3.4.4.

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.

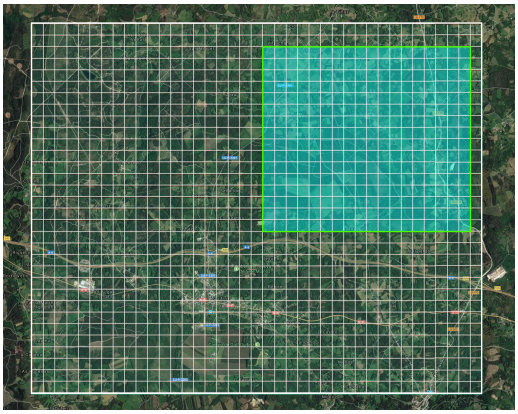
Although the objective of *ViLMA* is not to provide an ultra-realistic representation of the point clouds, three image enhancement techniques have been included in order to improve the quality of the images in terms of object recognition, object definition and depth perception. Circular points, dynamic point size and Eye-Dome Lighting [10], 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 [47].

3.2. Multi-resolution, out-of-core data structures

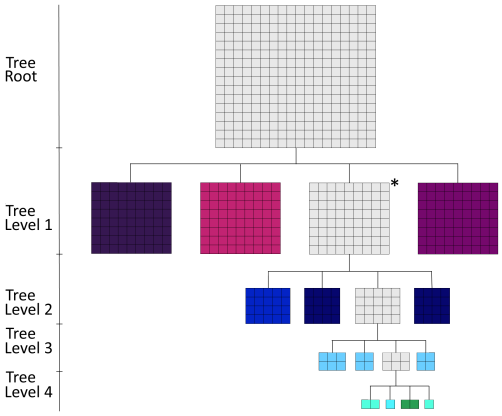
The interactive visualization of massive LiDAR point clouds, exceeding available memory resources, demands the use of multi-resolution, out-of-core techniques. The proposal presented here is focused on the following factors: minimizing the consumption of both system memory (RAM) and GPU memory (VRAM), 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 3.2.1) and TGPT (Section 3.2.2). Section 3.2.3 is dedicated to explain the fundamentals of the rending process using the cited data structures.

3.2.1. Hierarchically-Layered-Tiles (HLT)

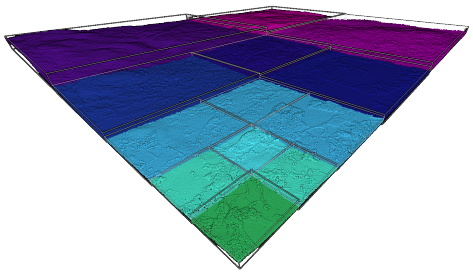
Traditionally, in computer graphics, multi-resolution approaches involve the creation of several different detailed versions of the same 3D model, which implies data redundancy among all model versions (further information about multi-resolution models can be found in [2] and [47]). The proposal presented in this chapter of the Thesis avoids data redundancy in order to achieve the above factors. There are



(a)



(b)



(c)

Figure 3.2: Construction of a TGPT from an arbitrary ROI and its posterior usage for computing the different LODs of the image. (a) Illustration of an ROI defined by a 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).

no static, precomputed, multi-resolution models of the point clouds, but a specific rearrangement and storage of the points, intended 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 (the minimum volume that wraps the entire set of points) of the point clouds is divided into T equally-sized tiles forming a regular grid (see Figure 3.2a as an example of grid 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 the tiles. The points in each layer are uniformly distributed over the surface. Given a tile t containing a total amount of P_t points, the number of points in the layer l is defined by:

$$P_{t,l} = \begin{cases} P_t \times (1-df)^{L-l} \times df & l > 1 \\ \\ P_t \times (1-df)^{L-1} & l = 1 \end{cases} \tag{3.1}$$

No point is repeated in more than one layer, hence, 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:

$$LOD_l = \bigcup_{i=1}^{l} L_i \tag{3.2}$$

where L_i is the layer i of a given tile.

Figure 3.3 shows an example of layer generation. In the example given, a $df = 0.5$ was used for simplification purposes; hence, all subsets contain half 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 *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 *Layer 3*, while the subset on

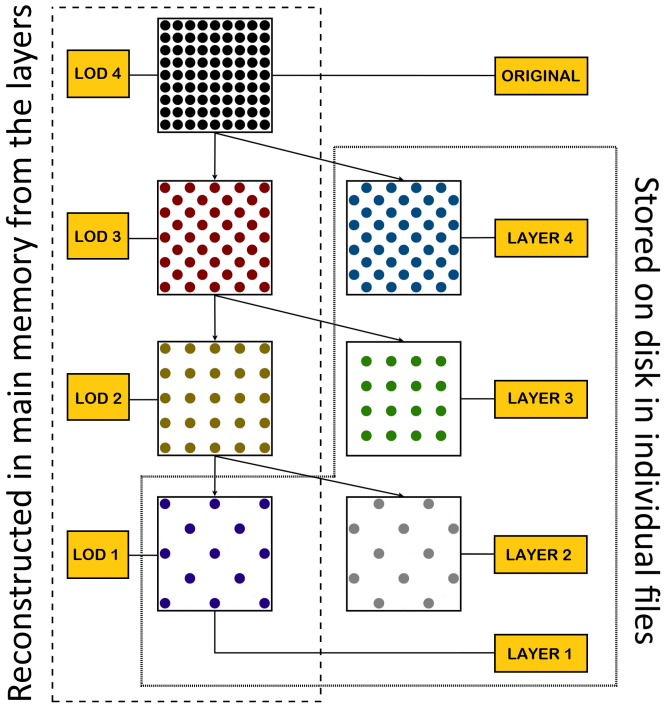


Figure 3.3: Layer generation of a single tile carried out during the *Pre-processing Stage*. Starting with the original point set (upper square) four layers are generated (labelled squares).

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 3.3.3. Point subsets labelled as *LOD 1*, *LOD 2*, *LOD 3* and *LOD 4* are reconstructed in main memory during runtime 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 $P_t = 1000$, in a traditional approach to multi-resolution, each LOD would have associated a precomputed 3D model stored on the 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, etc.) or how many points are selected for each LOD.

3.2.2. Tile Grid Partitioning Tree (TGPT)

The management and handling of the tiles into which a point cloud is divided is performed though a tree-like structure, the 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. 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 layer 1 includes the number of points in the remaining layers and the minimum and maximum values of the coordinates (x, y, z) in each tile. Using these coordinates, a bounding box 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 system. It should be stressed 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 explained further in Section 3.2.3). 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 3.2b 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 3.2b, the nodes containing a set of 1×2 tiles would only be split into 2 children with one tile each.

3.2.3. 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, attempting 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 has been discussed broadly in the literature [25].

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 into 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 into the LOD-distribution-list. As a result, at the end of the process the LOD-distribution-list contains all visible tiles, grouped into 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. After an LOD l has been assigned to a node, 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 at the end of this section.

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 3.2a shows an arbitrary ROI (inner shaded rectangle, 16×18 tiles) overlapping a dataset grid (outer rectangle, 32×39 tiles). Figure 3.2b 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 3.2c. 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.

Back-end retrievals

During the *Pre-processing 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. On 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 the approach presented, 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.

3.3. 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.

A number of important decisions must be made regarding how the datasets are pre-processed, as they have a direct impact on the performance; namely, the size of the tiles and the number of layers per tile (LPT) (Sections 3.3.1 and 3.3.2). These are subjective decisions that must aim to strike a balance between performance and quality [25]. In this section, we also address an additional non-subjective performance issue regarding the compression of the data (Section 3.3.3).

3.3.1. Tile size

The size of the tiles affects the accuracy of the view frustum culling techniques [2], the efficiency of data queries based on spatial restrictions, and the proper use of the browser cache. These three concepts were already introduced in previous chapters, so only their main characteristics are highlighted below:

- *View Frustum Culling.* In a 3D scene, view frustum culling techniques are

implemented to detect partially or fully visible objects from the 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.* As was briefly commented during the previous chapter, web browsers have a special memory space reserved on 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 the 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 pre-processing stages, so the requirements of the cache are more likely to be met.

Although the choice of a small tile size (TS) 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 creating 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 TS 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 TSs. As a base measurement, we obtain a global RAM

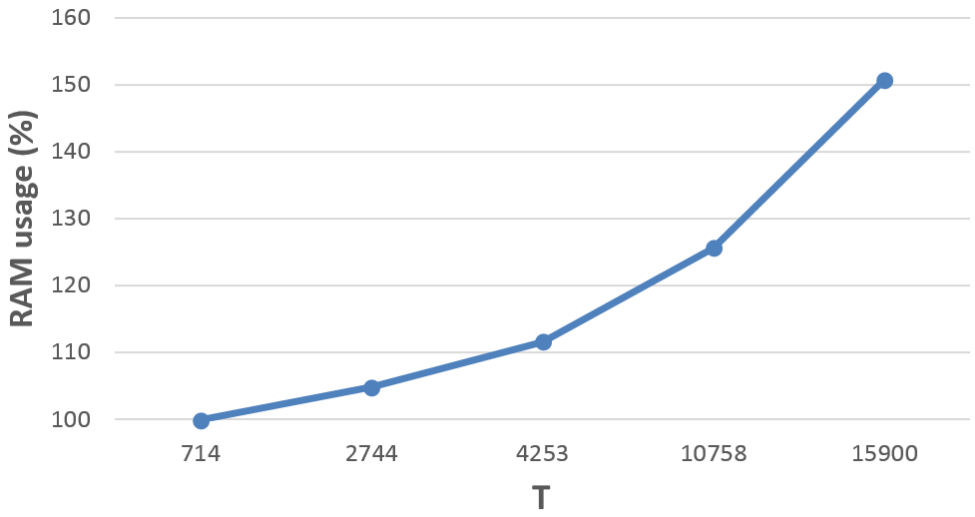


Figure 3.4: Memory consumption variation with respect to the increase in the number of tiles.

consumption of 250 MB with 714 tiles. Figure 3.4 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 TS, since the final number of tiles depends on the extent of the dataset and the balance of quality and performance desired by the users.

3.3.2. Number of LPT

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 3.3, 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 LODs. On the other hand, lower *df* values produce more LPT. 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 3.1, 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 3.4, we have measured the wait time when zooming in close to the ground using 8 LPT and 18 LPT. 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 LPT), the wait time was 1.6 times the time spent in the first case (8 LPT). 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 TS, the choice of *df* must aim to strike a balance between LOD quality and retrieval times.

3.3.3. Compressing the point layers

Based upon the optimization strategies presented in Section 2.4, a LiDAR compression format (called *LZ* for LiDAR-Zipped) was developed for providing suitable support for the HLT structure. With *LZ*, LiDAR point clouds are lossless compressed minimizing both client and server disk usage while reducing remote data retrieval times and network congestion.

Currently, the best lossless compression methods for LiDAR data are LASzip [56] and LAS Compression software, which implements the method presented in [70]. 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

ViLMA are discarded. Second, delta encoding, where LiDAR properties are stored in the form of differences (deltas). And third, GZIP [42] compression, where all generated data are compressed using this software tool. As a result of these three steps, LAS files are reduced, on average, by around 88%. Additional details on the compression method used are provided below, while a comparison between LASzip and our compression method is included in Section 3.4.

Data compression method

Three main tasks are carried out in order to generate *LZ* files: data cleaning, delta encoding and GZIP compression:

- **Data cleaning**: A similar process to what was described in Section 2.4.1 is followed here. Unused or unneeded 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.

- **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.

- **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 [42], 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 excellent 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.

3.4. Experimental results

In this section, we evaluate the performance of the data structures and techniques presented in previous sections using *ViLMA*. Performance is presented and analysed in terms of memory consumption, wait times, 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* [68, 90], once again as it is the most similar tool to *ViLMA* found in the current literature. The main specifications of the platforms and the software used during the tests are described in Table 4.1 and Table 4.2. *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 3.3 lists the three datasets used to evaluate *ViLMA* along with the number of points of each dataset (P), 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 pre-process each one of them. The *PNOA*

Table 3.1: Hardware specifications.

Platform	**O.S.**	**CPU**	**GPU**	**RAM***	**VRAM***	**Display**	**Bw****
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 Wifi
Server	CentOS 6.7	Intel Xeon E5-2603 v3	-	64	-	-	-

Values measured in: $*GB$, $**Mbps$ (Bandwidth).

Table 3.2: 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

(National Plan of Aerial Orthophotography, Spain) dataset is available in the Spanish GIS database (IDEE) [55]. 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 [74]. Finally, the *Volcano* dataset contains 0.55 billion points, with a point density of 13.71 $points/m^2$ being available at OpenTopography [76].

During the tests, the point budget (PB) employed was changed, 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 visual representations of the original point clouds.

Table 3.3: LiDAR datasets used and their information regarding the *Pre-Processing Stage*. P: Number of points. FS: Total file size of the dataset (original LAS files). FS_{LZ}: Total file size of the dataset (pre-processed files). *Ratio*: Compression ratio of the pre-processed files. TS: Tile size. LPT: Number of layers per tile.

Dataset	P*	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, ***$Meters$.

3.4.1. Memory consumption

Figure 3.5 shows the memory consumption observed for rendering each dataset using three different PBs. Both, the RAM and VRAM values were 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 their corresponding section.

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.

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

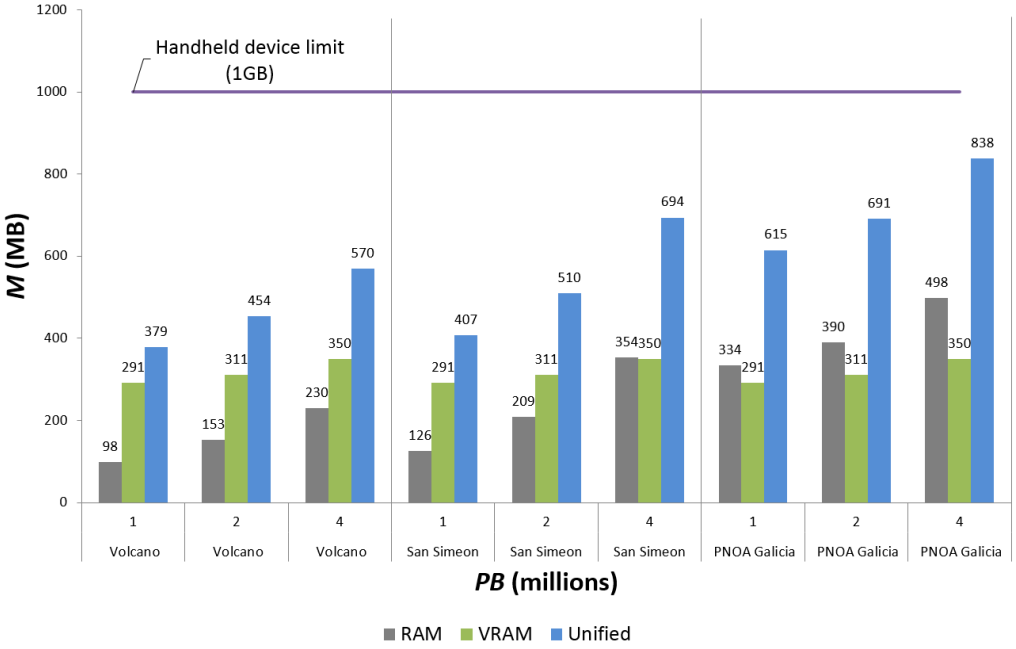


Figure 3.5: RAM, VRAM and Unified memory (RAM + VRAM) consumption during the performance tests for different point budgets.

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 3.5, where the VRAM 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.

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 [2]), 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 3.5 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. It was 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 currently running. In Figure 3.5 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.

3.4.2. Wait times

Figure 3.6 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 only a static top-view camera. For testing purposes, no ROIs have been used in order to analyse the most resource demanding scenario for each dataset. Times

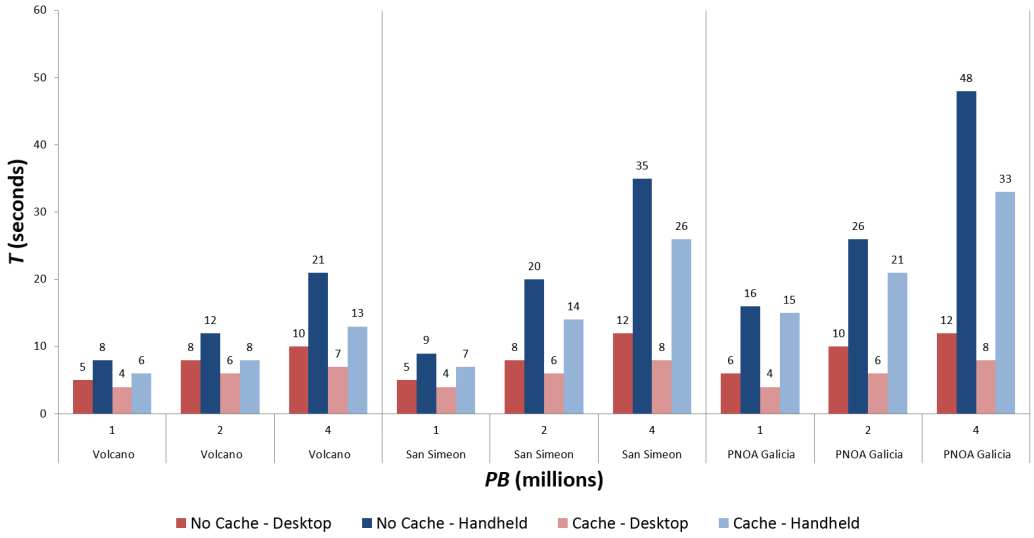


Figure 3.6: Wait times (retrieval time + load time) obtained among three different datasets with and without browser cache.

were taken using a PB of 1, 2 and 4 million points, with and without data caching.

The bandwidth values shown in Table 4.1 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.

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 decreases greatly, which increases 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.

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.

3.4.3. Interactive visualization

Figure 3.7 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.

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.

(a) (b)

(c) (d)

Figure 3.7: 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.

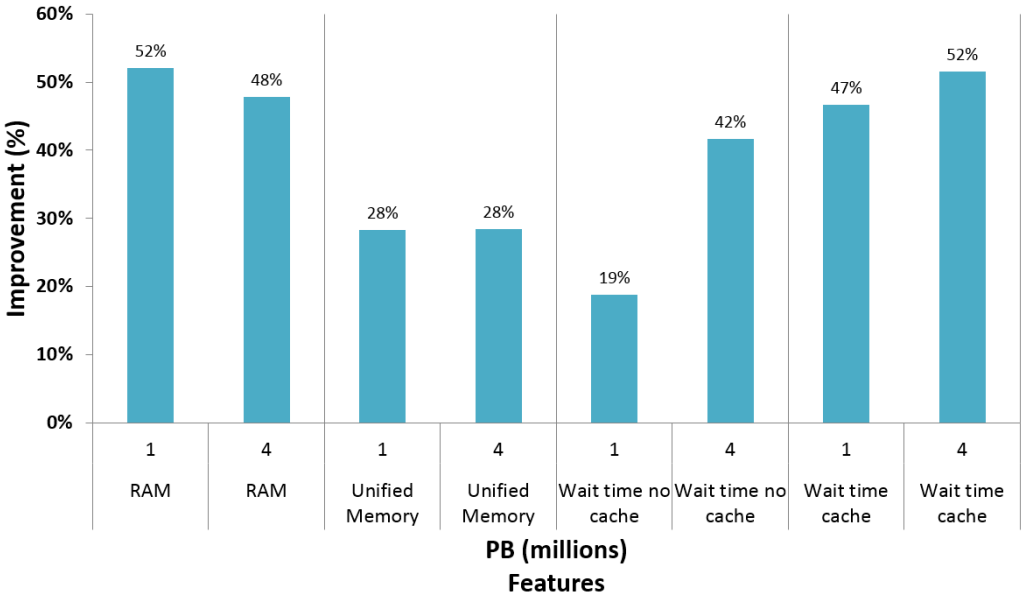


Figure 3.8: Performance comparison between loading the full dataset of *PNOA* and loading only an ROI from it.

3.4.4. 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 may 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 3.8 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 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.

3.4.5. Compression ratio

Results from a comparison between our *LZ + GZIP* compression method and LASzip can be observed in Figure 3.9. 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*.

3.4.6. Comparison against *Potree*

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

Figure 3.9: Compression formats comparison.

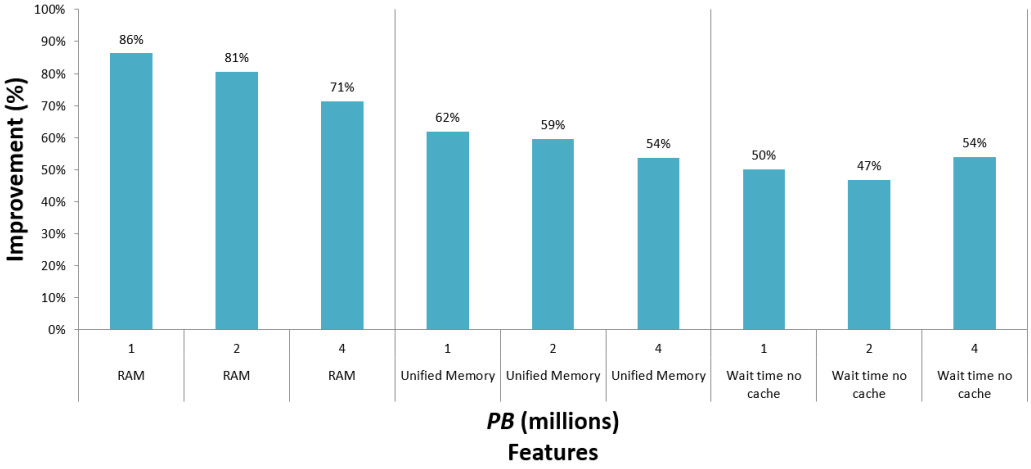


Figure 3.10: Comparison between *ViLMA* and *Potree*.

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 3.5, as has already been explained in Section 3.4.1. Figure 3.10 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, *ViLMA* shows better results. Between 71% and 86% lower in RAM consumption and between 54% and 62% in 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 relevant 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 use other 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.

Chapter 4

Big data storage solutions for large collections of massive LiDAR point clouds

In this chapter, we analyse how web-based applications for LiDAR data visualization can benefit from the adoption of big data storage technologies, as well as the advantages and disadvantages that may determine the choice of one of them when considering different scenarios and use cases. Specifically, four of the most adopted and mature big data storage solutions were selected for testing: HDFS [96], MongoDB [69], Cassandra [95] and Redis [83].

The rest of the chapter is organized as follows. In Section 4.1, we present a brief system overview of both big-data-oriented and non-big-data-oriented web-based visualization applications. In Section 4.2, the storage technologies used in this chapter and their deployment are detailed. Finally, experimental results are shown in Section 4.3. The work presented in this chapter was originally introduced in [29].

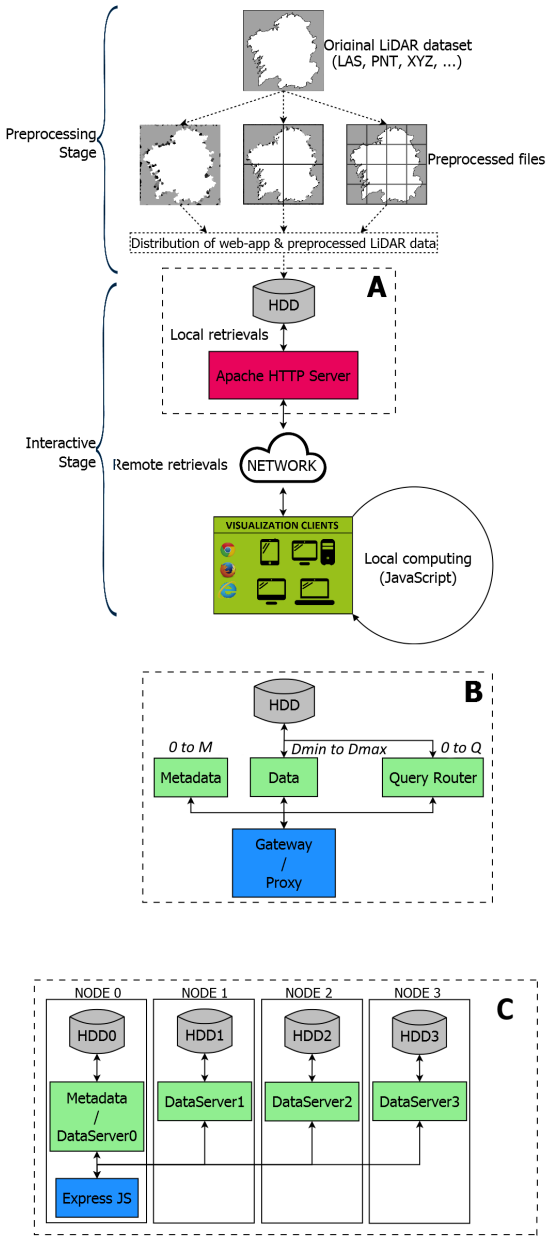


Figure 4.1: General overview of a conventional and non-big-data-oriented deployment of a web application for LiDAR data visualization. Box **A** encloses the components that must be replaced by the components of box **B** in order to transform the system into a big-data-oriented deployment. Box **C** is a more specific deployment of the general components shown in box **B**, and it is the deployment that will be used during the analysis described in Section 4.3.

4.1. Web-based LiDAR visualization: Migrating to a big data deployment

Figure 4.1 (without considering boxes **B** and **C**) shows the general structure of a simple and traditional web application for LiDAR data visualization. This type of application follows the common client-server architecture deployed over a standard Apache HTTP Server using a standard hard disk device (HDD); any compatible web browser can connect to the server in order to retrieve both, the application code and the point data.

When attempting to render massive LiDAR point clouds, many visualization applications could end up being not able to store all point data in main memory, and if so, the computational power of the GPU of the running machine may not be enough for handling such amount of points. In order to overcome these hardware limitations, out-of-core multi-resolution techniques [46] are used by many LiDAR applications. These techniques make it possible to load and render simplified models from the original point cloud offering a visual quality very similar to the original model. Point clouds are shown on screen using a user-defined or algorithm-defined maximum amount of points, established so as not to exceed the hardware capabilities of the client's machine. This point limit is usually called point budget (PB) [26].

Higher LODs are loaded as the point cloud is traversed, while off-camera points are discarded. This process demands the rearrangement of the points in multiple pre-processed files and the creation of auxiliary complex data structures in order to efficiently handle and load the points as they are required [47, 60]. Applications for LiDAR visualization retrieve the information of each different LOD of the point cloud, directly from local disk or from remote servers. In the case of applications retrieving data from a remote server, the software running in the back-end becomes a key point in the global performance, as it has to serve the data fast enough for obtain a real-time interaction.

For GIS centres, or any other kind of big company or governmental institution, that may use a very large number of LiDAR datasets, conventional server software solutions may suffer from several problems, such as scalability, availability and throughput. Considering the massive volumes of LiDAR data that could have to

be stored, a stand-alone server might not be enough for storing all LiDAR datasets and, if so, the throughput of the system may not be enough for handling very high levels of network traffic, being an important issue when attempting to offer real-time interaction for all connected clients.

Additionally, Apache stands out as a single point of failure, meaning that, any malfunction on the server could cause a temporary shutdown of the service or the permanently loss of data. In order to overcome those problems, it is necessary to deploy big data technologies to the detriment of the standard Apache HTTP Server or any other traditional server software.

In order to migrate from a non-big-data-oriented approach to a big-data-oriented one, components enclosed in the box **A** (Figure 4.1) must be replaced by the components of box **B**. In general, a big data storage deployment consists of the following components:

- Gateways or proxies. Web applications should not access or communicate with big data storage technologies directly for security, access control and workload balance reasons. Instead, applications use some kind of gateway or proxy that work as intermediaries in order to grant access to storage functionalities.

- Data servers. The common factor of all big data storage technologies are the data servers. These components are in charge of handling all read/write operations of the data. The number of these components may vary from a minimum D_min to a maximum D_max servers. In most technologies, the minimum is 1, but in technologies like Redis it is 3, while the theoretical maximum is unlimited.

- Metadata servers. Some technologies, such as MongoDB or HDFS, have dedicated components in charge of handling and managing all kind of metadata, from information about folders and files to information about the cluster topology and the distribution of the data. Not all technologies have metadata servers, so clusters may have from 0 to M elements of this type.

- Query routers. Some technologies, such as MongoDB, may have dedicated components in charge of receiving and serving data queries. No software can

access data; they need to be accessed through components of this type. Clusters may have from 0 to Q elements of this type.

We should note here that this general description of the components of the storage technologies may vary when describing each technology in detail. For example, Cassandra has no exclusive components dedicated to handling metadata or data queries, instead, all nodes in a Cassandra cluster are in charge of handling and managing queries from users, data and metadata.

4.2. Big data storage technologies: deployment analysis

Four big data storage technologies are analysed in order to determine which could be the best candidate to replace the Apache HTTP Server for the management of massive LiDAR datasets: HDFS [96], MongoDB [69], Cassandra [95] and Redis [83]. The motivation behind this selection was already explained in Section 1.3.1, where detailed descriptions about the four technologies were also presented. During the following subsections, we will go on explaining certain relevant issues about the deployment of the testing cluster and each one of the four technologies.

4.2.1. Testing cluster

Our testing cluster consists of four nodes. At this stage of the Thesis, we were unable to use more nodes; nevertheless, although it could not reflect the ideal real-world setup of a large scale deployment, it is enough to notice the main differences between technologies, as will be shown during Section 4.3.

Box **C** in Figure 4.1 shows the topology followed for deploying all storage technologies that will be used during the analysis presented in Section 4.3. Only one node can be accessed from outside the cluster, so is in this machine where we have deployed an Express server [73]. Express works as an access-point; it receives all data queries from the visualization clients and connects with the different technologies in order to send back the queried data from them. Node 0 is the one where

the Express server is deployed, connecting the cluster with the rest of the network. Node 0 is also where the metadata warehouses of the four technologies (if needed) are going to be deployed in order to obtain direct communication between them and the Express server. All of the nodes, from 0 to 3, are intended to store LiDAR data.

As was previously mentioned, in a production environment, it is strongly recommended to provide high availability and fault tolerance through data replication, deploying several nodes in the cluster in order to store copies of the original data. All technologies offer data replication and fault tolerance, so no technology is worse than the others at this point. Owing to this, and due to limitations in the number of nodes of our cluster, data replication has been avoided during the comparison between the storage technologies.

4.2.2. HDFS

Since it is in a web-based visualization context, the system employed here operates on a large number of small files, which is not an ideal use case, as described in Section 1.3.1. In order to solve this problem, we have used the Hadoop Archives (HAR) solution provided by Hadoop. Hadoop is able to pack multiple small files into a single larger file which can be stored more efficiently in HDFS. In addition to the HAR file, Hadoop creates a lookup file to access the individual files contained in the pack. HAR files are a copy of the originals, so, although after the creation of the HAR files the originals can be deleted, at certain point in the process, a given dataset uses twice the space it needs, which can cause disk space issues. The use of HAR is a solution to the small-files problem, but it is still a time-consuming workaround. The insertion of new datasets in the system is less straightforward than in other technologies, with the additional storage problems cause by the data duplication of the HAR creation process. Furthermore, all data queries have a small-time penalty due to the additional search needed to access the individual file store into the HAR.

For the deployment of HDFS, since, for performance reasons, it is strongly recommended not to place a data-node and a name-node in the same machine, in our testing cluster (see label **C** of Figure 4.1), Node 0 holds the name-node, so the Express server can retrieve the data directly from the same machine. The rest of the nodes (from Node 1 to Node 3) are used for hosting the data-nodes.

4.2.3. MongoDB

Unlike common SQL technologies, the internal data structures of MongoDB are not so rigid or immutable; therefore, new attributes can be added to each file in the system as needed. It provides native geospatial queries, which may be very useful when working with geo-referenced data, and native built-in Map-Reduce computational capabilities. MongoDB is compatible with some big data computing solutions, such as Spark, which is an added value to MongoDB for future applications beyond the storage service.

All technologies distribute data throughout the nodes of a cluster attempting to store almost the same amount of data in every node. This balancing is aimed at achieving a whole parallel access to the data which helps to speed up read/write operations. Nevertheless, obtaining this speed up is only possible if the access pattern to the data is similar to the distribution pattern, which is not always the case. In [65], the authors have developed a monitoring software for MongoDB capable of redistributing data based on the workload and the access pattern to it.

Taking into account the characteristics of MongoDB, in our deployment, Node 0 hosts the query router so that the Express server can retrieve the data directly from the same machine. Along with them, the configuration server is deployed leaving the rest of the nodes for the shards (Node 1 to Node 3).

4.2.4. Cassandra

Unlike HDFS or MongoDB, Cassandra can take advantage of all storage capacity of the nodes in a cluster, this may be especially relevant for clusters with a small number of nodes. Big data computing solutions like Spark or Flink are also compatible with Cassandra, adding value to it for future applications.

In order to deploy Cassandra, every node in the cluster is identical, being able to store data and resolve client/application queries. In a cluster with N nodes, data can be divided among the N nodes, so for our analysis, a Cassandra instance was deployed in each of the four nodes. We should also remark here, as an important feature of Cassandra, its fast and easy deployment process.

4.2.5. Redis

One of Redis's strengths is to be designed as an in-memory database, allowing it to always serve data from main memory, which is much faster than serving data from disk. Nevertheless, all technologies use some kind of data caching, which keeps the most recently used (MRU) elements in main memory speeding up later retrievals of the same elements. The four analysed technologies, and also Apache HTTP Server, can be configured to behave like Redis, using a large amount of main memory for data caching purposes, while simple scripts can be used to send all datasets from disk to main memory at start-up time. In fact, as Redis stores all datasets in memory without being aware of the data usage, it may be end up wasting space storing data rarely or never used. In a multi-resolution out-of-core visualization context, highly detailed point information about a sub-area without interest for users inside a larger point cloud, has a good chance of never been retrieved, since users should move the camera specifically very close to that area. Therefore, an MRU strategy is more suitable for the nature of the cited context.

In the deployment for our analysis, nodes from 0 to 3 host master nodes for storing data with no slaves, as data replication has not been included. Like Cassandra, it is important to highlight the fast and easy deployment process of Redis.

4.3. Experimental results

In this section, we analyse the performance of the four previously described storage technologies. The analysis was carried out considering: latency, throughput and storage capacity. In addition to the big data technologies, we have also measured the performance of Apache HTTP Server using a classic deployment (Figure 4.1, box **A**) as reference for comparing results against a non-big-data-oriented approach.

Table 4.1 and Table 4.2, show the hardware and software specifications of the equipment used during the analysis and useful information about the datasets of the analysis is shown in Table 4.3. The information listed in Table 4.3 is: the number of points of each dataset (P), the total disk space required to store their original LAS files (OFS), the total disk space required to store the pre-processed files (PFS),

Table 4.1: Hardware specifications.

Platform	**O.S.**	**CPU**	**RAM** (GB)	**Storage** (TB)
Client Latency	Windows 7	Intel Core i7 4790	32	0.25 SATA3 SSD
Client Throughput	CentOS 6.9	2 x Intel Xeon E5-2650 v2	64	1 SATA3 7.2k
Server (×4)	CentOS 6.7	Intel Xeon E5-2603 v3	64	4 SATA3 7.2k

Table 4.2: Software specifications.

Type	**Name**	**Version**	**Node Driver**
Browser	Google Chrome	63 (64-bit)	NA
Server	Apache HTTP Server	2.4.28	NA
	ExpressJS	4.16.2	NA
Storage Technology	HDFS	2.9.0	NA[a]
	MongoDB	3.4.9	mongodb 3.0.3
	Cassandra	3.9.0	cassandra-driver 3.4.1
	Redis	3.2.0	ioredis 3.2.2

[a]Does not require any specific driver as it is accessed through WebHDFS using conventional HTTP requests.

that is, the files retrieved by our visualization software for rendering the different resolution models, and finally, the total number of pre-processed files (NPF). The network connection used for the analysis was Gigabit Ethernet.

To store the LiDAR data, we have followed the same schema in all analysed technologies. An indexed *id* field of string type for storing file names and a *data* field of a BLOB type for storing the raw binary data. With regard to distributing data after their storage, although each technology distributes the data using their own procedures, all nodes in charge of storing data ended up with an even distribution. All technologies except HDFS distribute data using the keys (*id*) of each entry. By default, Redis and Cassandra apply hash functions over each key to obtain the node where values must be stored. This ensures an almost perfect distribution throughout the nodes. On the other hand, by default MongoDB uses the raw keys, without any

Table 4.3: LiDAR datasets used during the analysis.

Dataset	**P** (billion)	**OFS** (GB)	**PFS** (GB)	**NF**
PNOA Galicia	28	802	118	436,960
PNOA Asturias	13.4	425	NA	NA
San Simeon	17.7	561	132	436,824
Los Osos	6.1	161	NA	NA

modification, so the final distribution depends on the randomness of the keys chosen by the user. In our case, the file names were not enough for an even distribution, so we had to create a hashed index (a functionality inside MongoDB) to achieve the same behaviour as for Cassandra and Redis. HDFS, by default, randomly selects a data-node to store each block of data.

In all the tests, data are retrieved from the storage systems using the standard HTTP method, GET. In the case of Apache, requests are sent directly to it and data are fetched and sent back by Apache from local storage. For the rest of technologies, GET requests are sent to the Express server and from there, Express connects to the storage technologies using the corresponding driver, except for HDFS, which is accessed by using again GET requests through WebHDFS.

4.3.1. Performance in terms of latency

We have used our own web-based visualization software for LiDAR data as the client for the latency analysis. We have measured the retrieval time obtained while loading the PNOA Galicia dataset. Retrieval time can be defined as the time spent retrieving data from the server since the application starts loading a given dataset until it ends and the 3D representation of the point cloud appears on screen for the first time. This measurement can be considered as the latency of the system, as it represents the time it takes for a single user to obtain the minimum data necessary to show the point cloud in screen. Figure 4.2 shows the point cloud of PNOA Galicia rendered by our visualization software just after being loaded.

Retrieval time is measured using the Google Chrome's DevTools network monitor. When measuring latency through a network, there are some uncontrollable

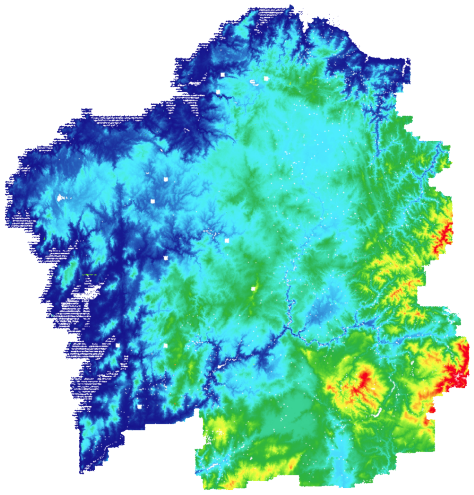


Figure 4.2: Point cloud of Galicia (Spain) from the *PNOA* dataset.

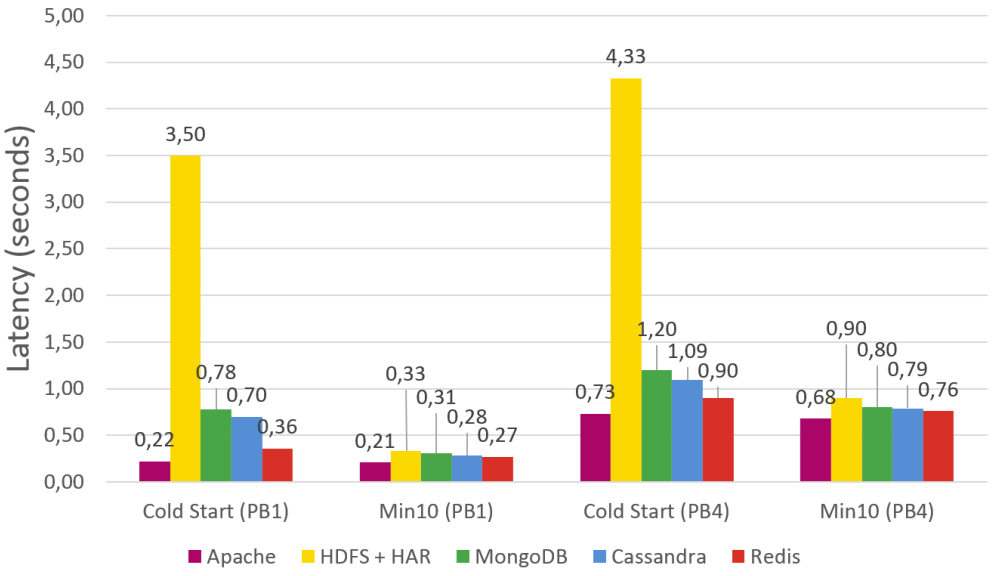


Figure 4.3: Latency obtained for 1 million points (PB1) and 4 million points (PB4). Results were obtained for cold start and for the minimum latency obtained in 10 tries (Min10)

variables that may slightly alter the final results; thus, we have taken the minimum latency obtained after 10 tries (Min10). We have also analysed the latency for a cold start in order to show the differences between cached and non-cached data on the server side.

As we are working with a multi-resolution out-of-core application, only one subset, dependent on the PB, is retrieved from the whole dataset (118 GB in 436,960 files). For this analysis, we have measured the latency with a PB=1 and PB=4 million points. With PB=1 million points, our framework sends to the server 32 concurrent requests, retrieving a total of 13 MB. For PB=4 million points, it sends 114 concurrent requests retrieving a total of 46 MB.

Figure 4.3, show the results obtained in the analysis. As can be observed, Apache achieves the lowest times in all cases. This is an expected result, as Apache represents a thin layer of software in comparison to the other technologies. Unlike Apache, in the rest of technologies, data have to pass through a larger number and more complex layers of software; moreover, data have to be moved throughout nodes, adding additional delay. Considering only the four storage technologies, Redis is the fastest one in all cases, followed by Cassandra, MongoDB and finally HDFS which obtains the highest results.

We should note here that Redis and Cassandra use 4 nodes to store the LiDAR data, so that, at least one of their nodes in charge of storing data is placed in the same machine as the Express server. This gives them an advantage when the files been retrieved are stored in the node of the Express server, as those files do not have to be sent from other nodes.

In cold start, Redis shows the lowest times of the four technologies due to in-memory design. After 10 retrievals, Redis keeps showing the lowest result, but they are almost identical to those shown by MongoDB and Cassandra thanks to the different levels of caching provided by the operating system and the technologies themselves.

The high results obtained in cold start by HDFS+HAR are partly due to Express, which, upon receiving the first data request, must build the access indexes to the files in main memory, adding around 2.8 seconds to the final latency. This procedure follows the same philosophy as the name-nodes, storing all data in memory to speed

up subsequent data requests.

Except for a cold start in HDFS+HAR, the highest difference observed between the times of the storage technologies and Apache is only 0.56 seconds in favour of Apache so, even though differences in latency between technologies do exist, from the point of view of the user experience, they are almost negligible, which means that, replacing a traditional server software, like Apache HTTP Server, with any of the big-data-oriented solutions, has a very limited impact on the latency of a web-based multi-resolution out-of-core application.

4.3.2. Performance in terms of throughput

The throughput analysis has been carried out using Apache JMeter [99]. Table 4.1 shows the specifications of the client machine used for running the client software. Three throughput tests were performed using from 8 to 256 concurrent users, each of them using 6 simultaneous connections to simulate the behaviour of most web browsers.

The way a point cloud is pre-processed largely determines the size of the files handled by a multi-resolution system. The files handled, the format of the files, the number of LODs chosen, and the resolution they display are some of the most important features that determine the final size of the files. All technologies show performance differences based on file size; hence, three throughput tests were carried out attempting to simulate three different workloads based on different file sizes. In the first test, each user requests a total of 10,000 files of 10 KB. In the second one, 1,000 files of 100 KB and in the third one, 100 files of 1,000 KB. Each test was performed 10 times taking the throughput as the mean of all of them.

As can be observed in Figure 4.4, for a size of 10 KB, Redis, Cassandra and MongoDB, obtain a much higher throughput than Apache, except for HDFS, which is slightly below. For files of 100 KB (see Figure 4.5), Apache has an irregular performance, being considerably inferior to the rest of solutions as the number of concurrent users increases, but being almost equal or slightly higher when users are 16 or less, once again, with the only exception of HDFS, which obtains lower throughputs except for 64 users. Regarding the test for 1,000 KB (see Figure 4.6),

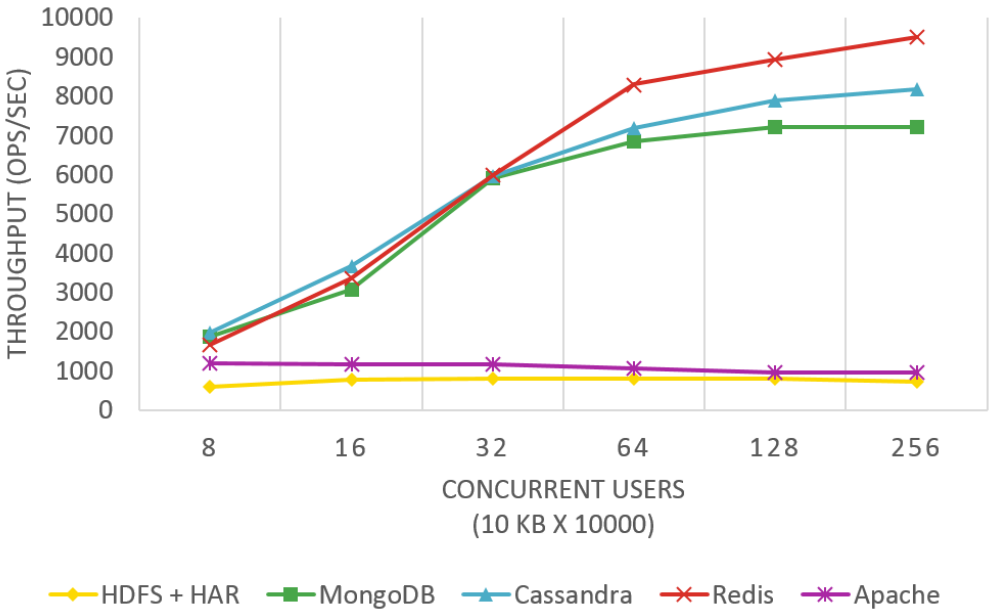


Figure 4.4: Throughput obtained for different concurrency levels. Each user makes 10000 requests of 10 KB.

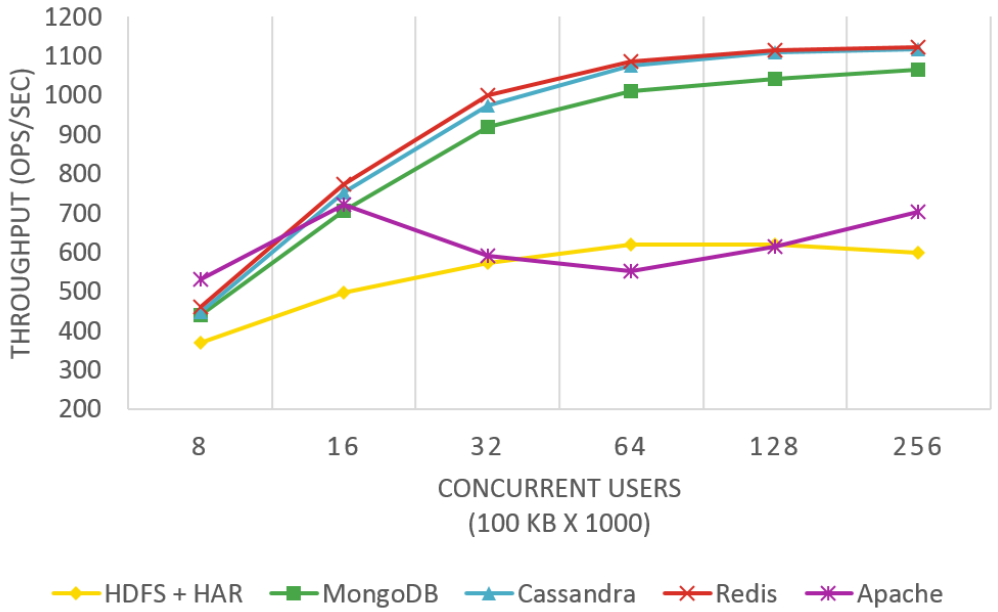


Figure 4.5: Throughput obtained for different concurrency levels. Each user makes 1000 requests of 100 KB.

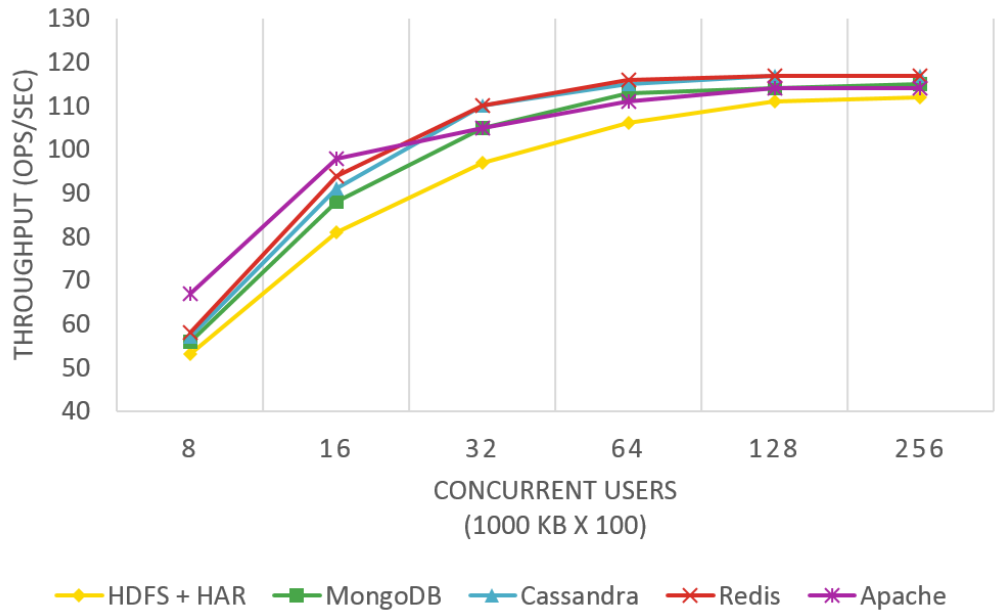


Figure 4.6: Throughput obtained for different concurrency levels. Each user makes 100 requests of 1000 KB.

Apache obtains slightly higher results for 16 users or less. For 16 users or more, the performance of all the technologies is almost the same, since all of them reach the maximum of 12.5 MB/s of the Gigabit Ethernet network.

Visualization applications powered by multi-resolution techniques may handle a large variability of files sizes, from a few KB up to several MB. This variability is determined by how the point cloud has been pre-processed. The file formats and the compression techniques used (Gzip, Delta-Encoding, etc), the type of data structures selected to handle and manage the point clouds (kd-tree, quad-tree, octree, etc), the number of LODs, as well as the point density and resolution of each LOD, have a direct impact in the number and size of the files that are going to be stored in the storage systems.

Visualization applications that use fine-grained data structures and efficient compression techniques, where points end up stored in small files, will take much more advantage of the performance provided by the big data solutions.

Another point to consider would be the type of network used. A big data system that is accessed through a Gigabit Ethernet connection could not offer any improve-

ment in terms of throughput if files larger than 1 Mb are only retrieved from the system.

4.3.3. Performance in terms of storage capacity

Figure 4.7 shows a comparison between the storage capacity of the technologies, expressed as a percentage with respect to the capacity of Apache HTTP Server. Each node of the analysis has a 4 TB disk, but only 2 TB are available for data storage. This is the space available for Apache, so it is taken as reference for comparison. Cassandra, MongoDB and HDFS, clearly overcome the storage capacity of Apache and Redis, with Cassandra being even superior to HDFS and MongoDB, thanks to its extra node for storing data. The capacity of Redis is very limited, due to its being an in-memory database. This means that, Redis has only 256 GB available for storing data.

Although, in theory, MongoDB and HDFS have the same storage capacity when using the same nodes, every time a dataset of B bytes must be inserted in HDFS as an HAR, it would require at least $2 \times B$ bytes of free space in the cluster, which could be an important issue, especially when trying to insert massive datasets.

Table 4.4 shows a comparison between the storage capacity of the technologies taking the size of real datasets as reference. The PNOA Galicia dataset was managed by all technologies without problems during the latency analysis. When trying to include the pre-processed files of another dataset (San Simeon (*Pre*)), Redis was the only one unable to store it. When trying to store the LAS files of the cited datasets, Apache reached its maximum capacity, while MongoDB, Cassandra and HDFS can store those datasets and another two very large datasets (the last two columns on the right of Table 4.4) without even consuming half of their storage capacity.

In conclusion, just two pre-processed datasets and their original data are enough to overcome the storage capacities of Apache and Redis, which highlights how the use of highly scalable storage technologies, such as HDFS, MongoDB or Cassandra, is critical nowadays, especially in the field of LiDAR data. Despite its results on the storage comparison, Redis achieved the lowest latency and the highest throughput in the previous tests. Being an in-memory database has performance advantages

Table 4.4: Datasets with original LAS files (O) and pre-processed files (Pre). The star marks show the datasets that can be stored in each technology.

	PNOA Galicia (Pre)	San Simeon (Pre)	PNOA Galicia (O)	San Simeon (O)	PNOA Asturias (O)	Los Osos (O)
Apache	⋆	⋆	⋆	⋆		
HDFS + HAR	⋆	⋆	⋆	⋆	⋆	⋆
MongoDB	⋆	⋆	⋆	⋆	⋆	⋆
Cassandra	⋆	⋆	⋆	⋆	⋆	⋆
Redis	⋆					

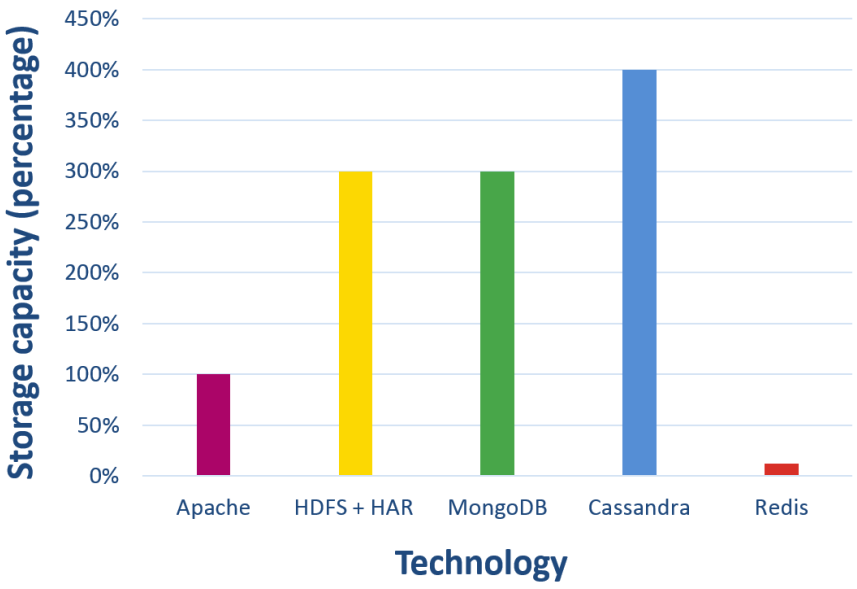


Figure 4.7: Storage capacity of the technologies.

but makes the vertical scalability much more economically expensive, depending on the total size of the point clouds to be stored.

Chapter 5

Big data geospatial processing for large collections of massive LiDAR point clouds

In this chapter, a big data approach on geospatial processing for massive aerial LiDAR point clouds is presented. The system is intended to support the execution of any kind of geospatial process; nonetheless, as an initial case of study, we have focused on fast ground-only rasters obtention to generate DTMs from massive point clouds. Thanks to this approach, it was possible to greatly reduce the time required for processing large extents of aerial point clouds in comparison with single machine approaches, while also obtaining all the common advantages associated with big data technologies (reliability, availability, scalability). Following the analysis and conclusions presented in the previous chapter, data distribution was performed using Cassandra [95], while the computing distribution was accomplished with Spark [100], due to its versatility, source code compatibility and batch oriented-design.

Filtered rasters created from the isolated processing of adjacent zones (something that is usually done when dealing with very large point clouds) may exhibit errors located on the boundaries of the zones in the form of misclassified points. These issues must be corrected through manual or semi-automatic procedures. In this chapter, we also present an automated strategy for correcting errors of this type, improving the quality of the DTMs obtained while minimizing user intervention.

This chapter is structured as follows: The whole big data approach on geospatial processing is presented in Section 5.1. Our automated strategy for the correction of errors located on the boundaries between different processing units is described in Section 5.2. Analyses in terms of quality and performance are shown in Section 5.3, along with the applicability of the framework for generating DTMs close to real time. The work presented in this chapter was originally introduced in [30] and [31].

5.1. A scalable big data approach on geospatial processing

The main goal in this stage of the Thesis was to develop a highly scalable geospatial processing system through the distributed computing and storage capabilities of big data technologies. Using said technologies, LiDAR datasets are distributed throughout a cluster of N nodes, granting all common advantages associated to big data storage solutions (reliability, availability and scalability [15]) while making it possible to parallelize complex geospatial processes to reduce their execution times.

The system presented here provides users with the ability of easily launching any kind of computational task using as input one or more massive aerial LiDAR point clouds. The whole system design was conceived to offer great versatility by expanding its functionality with the inclusion of different types of geospatial processes. Nonetheless, as first case of study, we have focused solely on fast DTM obtention from massive aerial LiDAR point clouds, since, as was briefly explained in the introduction, DTMs are one of the most valuable data resources that can be obtained from LiDAR datasets.

The global structure of the system is composed of three main elements: a geospatial process library containing a progressive morphological filter for carrying out the main point classification tasks (Section 5.1.1); a wide column store for distributing and storing the LiDAR datasets (Section 5.1.2), and a batch processing technology for distributing all computational tasks (Section 5.1.3). Figure 5.1 shows the global system structure just described.

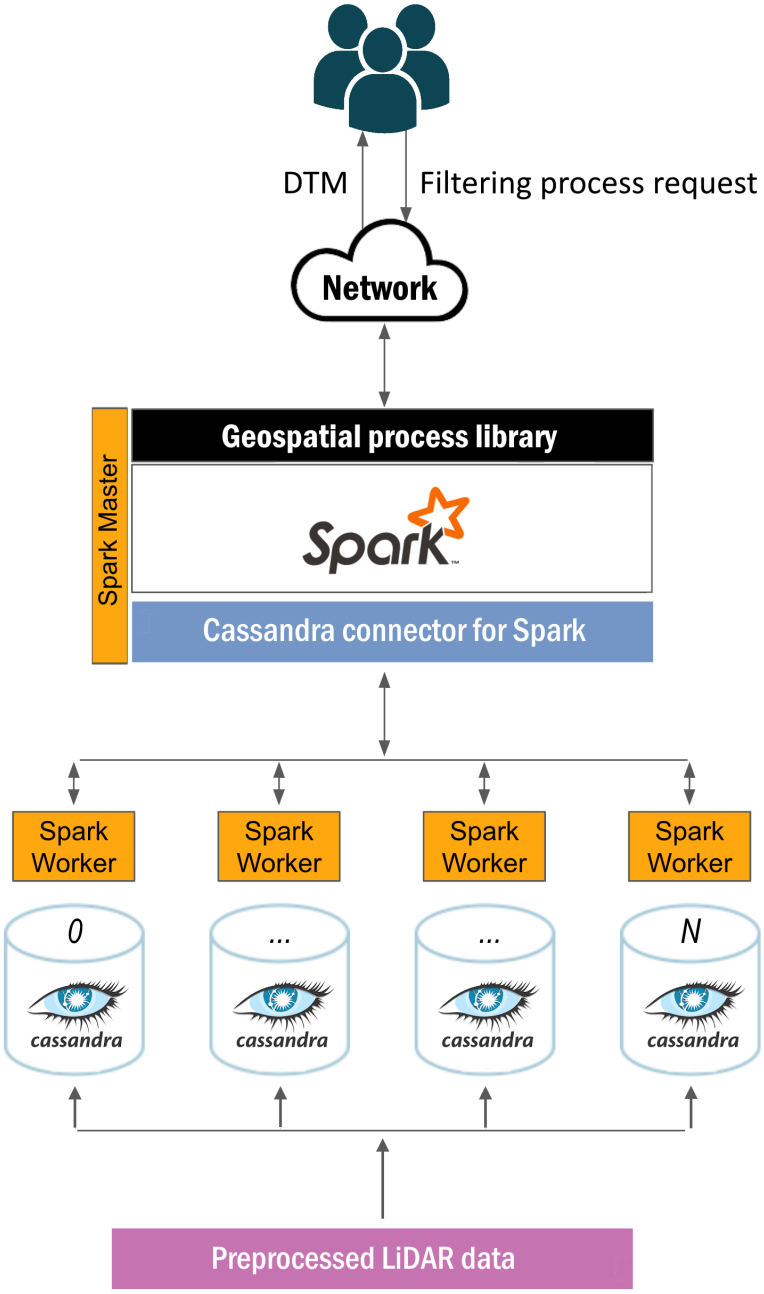


Figure 5.1: Global system structure.

5.1.1. Geospatial processing: fast DTM obtention

The general method we have employed to obtain DTMs begins with the creation of a raster from the input point cloud and then applying a progressive morphological filtering [118] in order to classify its points. As a result, a raster containing only ground points is obtained, which represents the final DTM. It should be stressed here that the output is not stored on disk in the form of an already triangulated file format, but like a regular point cloud, so each user can triangulate the results using the software and properties they consider most appropriate.

SC-091-12 [22] was the base algorithm selected to carried out the tasks described above. The reasons behind this selection were: the good results of *SC-091-12*, in terms of quality and performance, in comparison to Fusion [91] for the generation of DTMs [82]; and its potential integration with Spark and Cassandra, since the algorithm is coded in Java, and this is one of the programming languages supported by Spark and the Spark-Cassandra connectors.

The point classification accuracy of *SC-091-12* relies directly on a series of main input parameters: initial window size (IWS), maximum window size (MWS), initial elevation threshold (IET), maximum elevation difference (MED) and cell size (CS). Further technical information about these parameters and the functioning of the filter used by *SC-091-12* can be found in [118]. In [82], the authors found optimal values for all parameters except for CS, which had to be adjusted depending on the characteristics and type of the point cloud being processed; e.g., urban, rural, flat or mixed. Table 5.1 shows those input parameters along with their optimal values. These values will be used during the analyses of Section 5.3.

As has already been mentioned, the point classification is not applied directly to the whole input cloud but to a raster version of it. In our proposal, the raster is obtained by dividing the surface of the point cloud into a grid of regular cells, whose size is defined by the input parameter CS, and then selecting the point with the lowest Z component in each cell. This point selection method improves the final quality of the DTMs, as the points located at the lowest heights have a higher probability of being ground points. In the event of finding more than one point sharing the lowest value, the point closest to the centre of the cell is selected. This rule is applied to ensure that the same point is always selected

Table 5.1: Optimal values for the input parameters of *SC-091-12* algorithm.

Input parameter	**Optimal value**
IWS	3
MWS	30
IET (m.)	0.5
MED (m.)	5
CS (m.)	-

through different executions. Selecting, for example, just the first point that has been loaded with the lowest height may produce different output results in different executions, as during our error correction stage (see Section 5.2), points may be loaded in different order from one execution to another. Selecting the point closest to the centre may sometimes also improve the quality of the classification. This occurs when selecting the most central point avoids the selection of a point placed very close to the boundaries of the cell, as at such positions the point may be not very representative of the rest of the points in the cell.

In order to minimize execution times, the *SC-091-12* algorithm, in its original version, is able to benefit from the use of multiple CPU cores. During the filtering process of a single LiDAR zone, all cores available on the CPU are used to complete the different internal tasks of the filter, e.g., dilation or erosion operations (see [118] for more technical information about this and other internal operations in the filter). However, the CPU usage observed during the filtering process was very irregular over time, both in the number of threads used and in the workload of each of them, since occasionally one or more cores were inactive or they did not work at full capacity, wasting computing power that could have been used to complete additional tasks. Considering this, the design of the big data version of the filtering algorithm was oriented towards zone-level parallelism in such a way that several zones in a point cloud could be processed in parallel, using all the available cores in the CPU.

Aiming to achieve the aforementioned level of parallelism, in the current version of the filtering algorithm, each raw point cloud (an original and unprocessed version of the point cloud) is subdivided following a two-dimensional (2D) tile grid pattern. No other more complex data partitioning is required thanks to the 2.5 dimensional

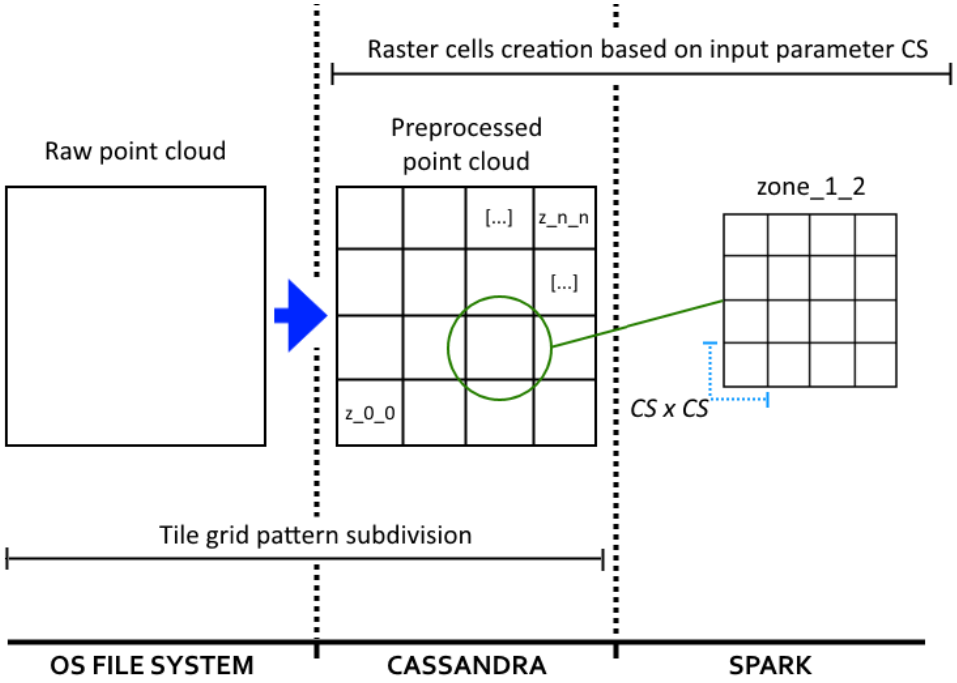


Figure 5.2: Data divisions carried out during the whole computational process. During an offline preprocessing stage, raw point clouds are divided following a tile grid pattern. Then, each tile (or zone) is inserted and stored permanently in Cassandra. During runtime, a raster is created for each zone by subdividing the zone into cells whose dimensions are defined by the input parameter CS.

nature of the aerial LiDAR point clouds. The points of each tile are packed together in a single file, then individually inserted into Cassandra to be distributed throughout the cluster. This subdivision is carried out during an offline preprocessing stage and it is up to the users to decide the number and size of the zones (see Figure 5.2). Such decisions will have a direct impact on the performance of the whole system in general, and the DTM obtention process in particular, as will be explained in Section 5.3. Inserted files are considered as independent zones and will be used later by Spark as parallel processing units.

5.1.2. Distributed storage: Cassandra

In the previous chapter, four of the most currently adopted and mature storage technologies were analysed in order to determine the best option for the management

of massive LiDAR datasets, as well as for supporting a web-based visualization framework. In the cited analysis, HDFS [96], MongoDB [69], Cassandra [95] and Redis [83] were studied, attempting to include with each one of them a different type of database design: distributed file systems, document stores, wide column stores and key-value stores [92].

Considering the conclusions of the aforementioned analysis, Cassandra was the storage technology selected for this chapter of the Thesis, since its querying capabilities are broad and very similar to those featured in other classic SQL databases, and its integration with big data computing solutions, such as Spark, is efficient, stable and robust. Additionally, all nodes in a Cassandra cluster play the same role, so, unlike other technologies, it is not mandatory to deploy nodes for exclusive tasks such as metadata storage. This helps to maximize the level of parallelism in the system, since all nodes available can be used for LiDAR data distribution. All these excellent qualities are further reinforced by the performance, in terms of latency and throughput, achieved on executing read operations, which are a key point in the proper functioning of the system.

LiDAR files are stored in Cassandra using a simple two-column schema. A string type column for the name of the files, which is used as the main key of the table, and a BLOB (binary large object) type column for the binary data of the files. This schema allows the files to be inserted directly into the storage system as binary data without any kind of additional modification or transformation. As described in the previous chapter, the size of the BLOBs directly affects the performance of the system. As the size of the BLOBs increase, the latency and throughput of the system worsen. This means that the file format of the files, as well as the extent of the point clouds they contain (i.e., the size of the zones), should be chosen carefully in order to avoid causing a negative impact on the system performance. For example, text files tend to be larger and more expensive in computational time than raw binary files when it comes to reading data out of them. In addition, files compressed with efficient methods may improve the system performance as long as the overhead of the decompression are small enough. The entire big data system tends to perform better with files containing small extents of land; nevertheless, it can produce a decrement in the quality of the output using certain algorithms. These issues will be further explained in Section 5.2 and taken into account for the

performance analysis of Section 5.3.1.

5.1.3. Distributed computing: Spark

The big data system presented here can be considered as a batch processing system, since geospatial processes operate without supervision on static LiDAR datasets. Map-reduce is one of the best-known paradigms of distributed computing for batch processing in big data contexts; nevertheless, this paradigm does not entirely fit the way the filtering algorithm works, in addition to being very rigid and limited, which could become a problem in the future when the inclusion of new geospatial processes would be required. Taking all this into account, map-reduce was discarded along with the use of Hadoop [103] and other similar computing solutions. Other technologies, such as Storm [104] or Flink [102], were also discarded as they are mainly focused on stream processing rather than batch processing. On the other hand, Spark stands out as the most suitable option for the proposal presented here thanks to its great integration with Cassandra through the Spark-Cassandra connector [24], as Java is one of its programming languages (same as the base source code of the filtering algorithm used), and owing to its design focused on batch processing and its operational versatility.

Since Spark implements a Master/Slave design pattern, our big data approach ensures maximum parallelism during the execution of the geospatial processes by deploying a Spark worker on each Cassandra cluster node. The global system depicted in Figure 5.1 shows a Spark master deployed on its own node separated from the rest of the components (Cassandra nodes and Spark workers) on the system. This would be the ideal structure on a production environment since, usually, only the Spark master would be directly accessible for the users, allowing them to send a computational request to the system without exposing the rest of components on it. Additionally, such a deployment allows the Spark master to use as much CPU power as needed without interfering with the execution of other system components. However, in the case of node exposure not being important and the workload of the Spark master always being very low, the master could be deployed together with one of the Cassandra nodes and a Spark worker, without causing any negative impact on their performance and with the additional benefit of not requiring an additional

node for the master alone. This last deployment configuration is the one used for the analysis of Section 5.3.

Moving logic or algorithms between nodes is faster than moving data, this is one of the main principles of big data. Following this principle, the Spark master distributes a copy of the geospatial process code (explained in Section 5.1.1) among the Spark workers so they can apply its computational tasks on subsets of LiDAR zones retrieved from Cassandra. The Spark-Cassandra connector ensures data locality, this means that each Spark worker only operates with the LiDAR zones stored on its own node, avoiding data movements between nodes. This is the general rule; nevertheless, it is not always followed or even desirable, as will be explained during the following sections. Pursuing the zone-level parallelism described in Section 5.1.1, each Spark worker operates only on its local subset of LiDAR zones, which are processed in parallel by being distributed among all physical cores available on each worker's CPU.

Cassandra uses table keys to distribute data evenly throughout the nodes. The optimal performance of our application is achieved when the data distribution pattern matches the workload distribution pattern. If every node on the cluster stores the same percentage of the total data, and a geospatial process is executed using the whole data as input, every Spark worker will be assigned almost the same amount of workload. This is always the scenario during the analysis of Section 5.3. Workload distribution optimization [65] is beyond the scope of the research presented here.

5.2. Automated boundary error correction

The classification of a given point under the categories of ground or non-ground largely depends on the features of its neighbouring points. Points located at the boundaries of adjacent zones filtered independently lack important information about their neighbours, so it is very common to find classification errors at such locations. Figure 5.3 shows a raster of ground points obtained after filtering a point cloud that was subdivided into four zones during the preprocessing stage. Each zone was filtered independently from the rest and the classification results displayed together. As can clearly be seen, the final classification shows a regular pattern of gaps, far

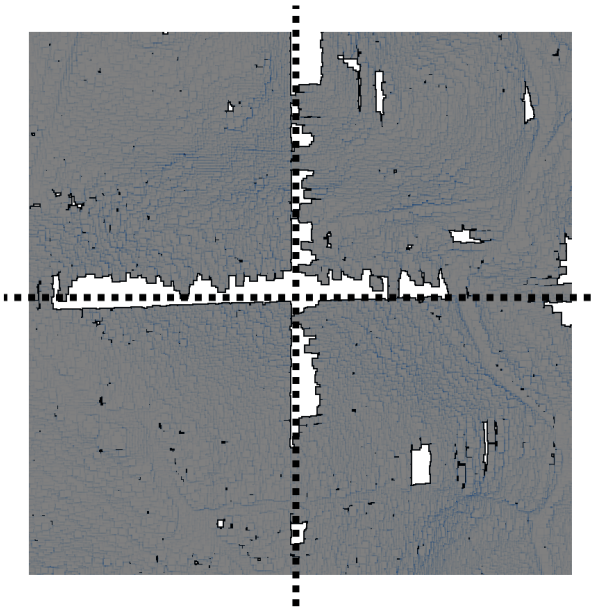


Figure 5.3: Raster displaying only ground points. The raster was obtained by independently filtering four adjacent zones and joining the results. Dotted lines highlight zone boundaries.

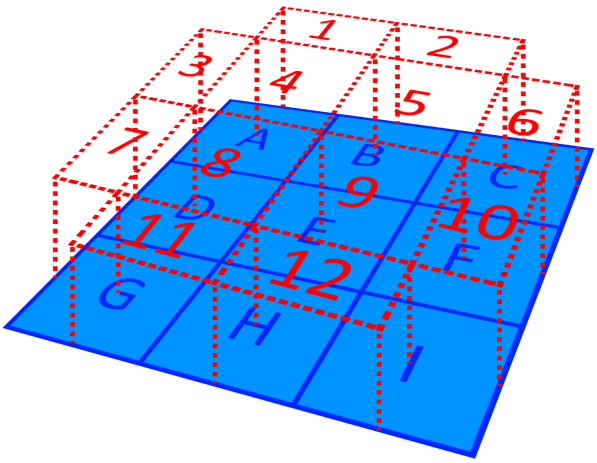


Figure 5.4: Schematic representation of the automated boundary error correction strategy. Squares labelled with letters and outlined with continuous lines correspond to LiDAR zones. Rectangles labelled with numbers and outlined with dots delimit overlapping sections between adjacent zones that will be used as *correction patches* to detect and remove classification errors located on the boundaries of the zones.

from being produced by natural geographic features or human-made structures, which match the boundaries of each zone. These gaps represent points that were misclassified as non-ground and thus not appear on the image.

Usually, this type of errors are avoided through manual or semi-automatic (scripted) procedures. Users must firstly define overlapping sections between adjacent zones, then filter the zones together with the points from the overlapped sections, and finally crop the results to fit the original extent of each zone. In any case, these type of solutions always imply some kind of user intervention.

In this chapter, we present an automated error correction strategy which, taking advantage of the computational power of the big data system, automatically creates new additional overlapping rasters encompassing boundary sections between adjacent zones. The points from these overlapping rasters are also classified, obtaining as a result a collection of ground points that will be used to detect and correct the errors that may appear during the classification of the points of the zone rasters. For convenience, we call these collections *correction patches*. For example, in Figure 5.4, the squares outlined with continuous lines represent LiDAR zones, while the rectangles outlined with dots delimit the areas of the new overlapping rasters. The whole process is explained in following subsections.

5.2.1. Creation of the correction patches

New overlapping rasters are created through a distributed process also carried out by the Spark + Cassandra system. Following the pattern depicted in Figure 5.4, a list with these new elements is made, taking into account all the LiDAR zones that are going to be filtered. Then, the elements of the list are evenly distributed throughout the Spark workers. Workers must retrieve from Cassandra all the necessary LiDAR zones to gather the points of each overlapping raster. For instance, in order to collect the points of the overlapping raster 5 (see Figure 5.4), the worker in charge must request zones B, C, E and F. In the same way as for the rasters of the zones, the input parameter CS is used to define the cell dimensions on the overlapping rasters (see Figure 5.2).

Once the points of each overlapping raster have been gathered, the final cor-

rection patches are created by passing all those rasters to the classification process as input and obtaining as a result a collection of ground-only points for each one of them. The correction patches are stored temporally in Cassandra and deleted once the entire process is finished, since they are only valid for the specific input parameters used in the current classification process.

5.2.2. Filtering of the LiDAR zones and error correction

All LiDAR zones are passed as input to the filtering algorithm. In order to ensure data locality, each Spark worker executes the algorithm over the zones stored in the same node it is deployed. A raster is created for each zone using CS (as depicted in Figure 5.2) and then its points classified under the ground or non-ground categories. This output is a provisional classification, very likely to display errors on the boundary zones between zones.

After filtering a given LiDAR zone, each Spark worker retrieves from Cassandra the corresponding correction patches overlapping its surface. For instance, in order to correct the errors on the raster obtained from zone H (see Figure 5.4), the worker in charge must request correction patches 8, 9, 11 and 12. The information from the raster of the zone and the correction patches is combined to correct the errors found in the first one. Given two already classified points located at overlapping raster cells, one point taken from the raster of the LiDAR zone (zone-raster-point or ZRP) and the other point taken from the correction patch (correction-patch-point or CPP), the rules to correct the errors are defined as follows:

- If both points are labelled with the same class, the ZRP is considered as correctly classified, as was confirmed by the results of the patch.
- If a ZRP is classified as ground, but the CPP is classified as non-ground, the ZRP is considered as correctly classified. The overlapping rasters have their own boundary errors; if this scenario were considered as a classification error, the final results would display gaps that would match with the boundaries of the correction patches, so instead of correcting errors, new ones would be introduced.

- If a ZRP is classified as non-ground, but the CPP is classified as ground, the ZRP is considered as misclassified and ZRP should be labelled as ground. Additionally, following the same logic described in Section 5.1.1 for the selection of raster points, if the Z coordinate of ZRP is higher than the Z from CPP, the ZRP is replaced entirely by the CPP.

Figure 5.5 shows an example of the third case listed above. Squares outlined with an outer bold continuous line correspond to LiDAR zones, the vertical rectangle outlined with dots represents a correction patch and the rest of smaller squares are the raster cells from their corresponding rasters. In this example, an error is detected, as ZRP was classified as non-ground meanwhile CPP was classified as ground. The error is corrected by not only changing the class label of ZRP but replacing it entirely by CPP, since CPP has a lower Z value.

5.3. Result analysis

This section covers the analysis, in terms of executions times, on the performance and scalability of our proposal (Section 5.3.1), along with it, a visual and quantitative analysis of the boundary error correction strategy (Section 5.3.2), and finally, further functional and computational considerations that should be highlighted regarding pre-processing decisions (Section 5.3.3), full point classification (Section 5.3.4) and point triangulation (Section 5.3.5).

Two massive raw point clouds, whose details are described in Table 5.2, were preprocessed in order to obtain the datasets employed for the analyses of this section. Four different datasets, whose details are described in Table 5.3, were obtained out of the two raw point clouds. Datasets D0, D1 and D2 were created from the same point cloud (*PNOA*, see Figure 5.6) varying the size of their processing units by using three different zone extents. On the other hand, D3 (created from the point cloud *Guitiriz*, see Figure 5.7) was included to analyse system performance under specially unfavourable conditions, forcing the system to handle large processing units (29,690 KB per zone on average). During the preprocessing stage, not only was the subdivision of the point clouds carried out, but also a compression of the resultant files. A compression method, very similar to the one described in [32], was employed

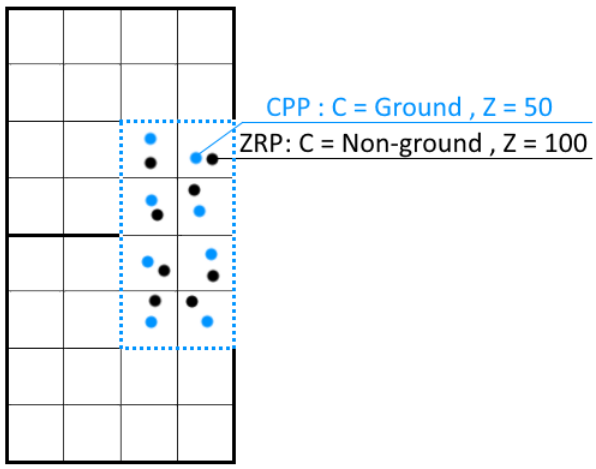


Figure 5.5: Schematic example of a classification error correction. The zone-raster-point (ZRP) has been misclassified as non-ground, and must be entirely replaced by the correction-patch-point (CPP) as the ZRP has a higher Z value.

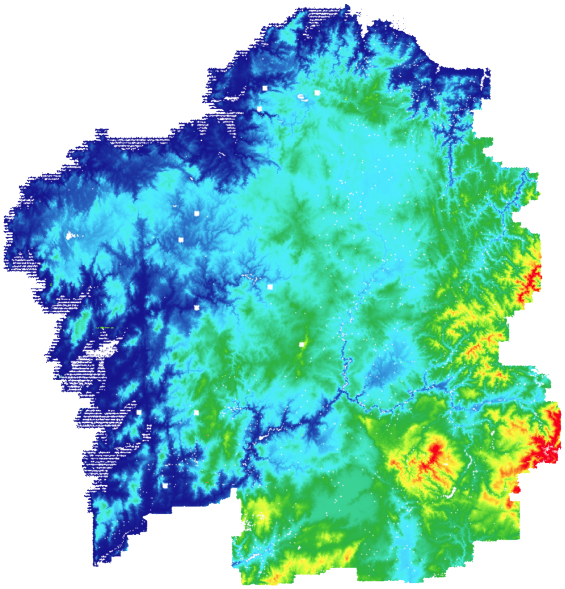


Figure 5.6: Rendering of the PNOA point cloud, specifically the region of Galicia (Spain).

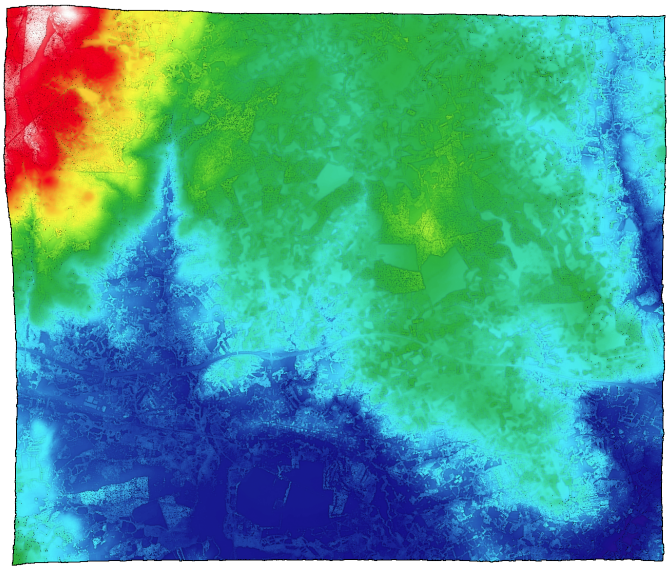


Figure 5.7: Rendering of the Guitiriz point cloud. Village and surroundings (Spain).

Table 5.2: Properties of the raw LiDAR point clouds selected for preprocessing. **NoP** = Number of points (billions). **NoF** = Number of files. **FE** = File extent (meters). **FS** = File size (average kilobytes per file). **TS** = Total point cloud size (GB).

Point cloud	**NoP**	**NoF**	**FE**	**FS**	**TS**
PNOA (Region)	28	8697	2000×2000	96731	802
Guitiriz (Village)	1	304	500×500	126194	37

to reduce the size of the data to be handled by the system.

The filtering algorithm was configured with the input parameters shown in Table 5.1, setting CS to 1.5 following recommendations from [82].

All machines used on the system deployment follow the specifications shown in Table 5.4. As was already explained in Section 5.1.3, the Spark master runs along with a Spark worker and a Cassandra node due to the negligible workload on the master and because the exposure of the nodes is irrelevant in this context.

Both, Spark and Cassandra, offer a large number of configurable settings. In order to obtain the best performance results, these settings must be configured

Table 5.3: Properties of the datasets used for the analysis. **NoZ** = Number of zones. **ZE** = Zone extent (meters). **ZS** = Zone size (average kilobytes per zone). **TS** = Total dataset size (GB).

Dataset	**Source**	**NoZ**	**ZE**	**ZS**	**TS**
D0	*PNOA*	11662	1600×1600	8925	99
D1	*PNOA*	181823	400×400	563	99
D2	*PNOA*	2875097	100×100	39	108
D3	*Guitiriz*	304	500×500	29696	9

Table 5.4: System specifications of the machines used for the analysis.

O.S.	**CPU**	**RAM**	**HDD**	**Network**
CentOS (6.10)	2×Intel Xeon E5-2660 (16 Cores / 32 Threads)	64 GB (DDR3)	SATA3 (7.2k)	InfiniBand

taking into account the nature of the algorithms executed as well as the amount and type of data involved. The most relevant settings configured for Spark 2.4.0 are described in Table 5.5, while the settings for Cassandra 3.11.3 are described in Table 5.6.

Several performance differences between Java 8, 9, 10 and 11 were found during the research of this stage of the Thesis, with Java 11 being the version that offers best performance results. Hence, the big data version of the filtering algorithm was compiled using JDK11 and both, the Spark Master and the Spark workers, are launched using JRE11 as well. However, the latest Java version supported by Cassandra is Java 8, so it is launched using JRE 8.

Spark's documentation defines *spark.locality.wait* as how long Spark must wait to launch a data-local task before giving up and launching it on a less-local node. The same wait is used to step through multiple locality levels (process-local, node-local, rack-local and then any). After a number of tests, it was determined that a value of 0 seconds (default is 3) offers the best results. This configuration implies that, as soon as a node gets idle, data are moved from another node and a task is assigned to the idle node. This improves performance as, in most cases, the time

Table 5.5: Relevant Spark configuration settings.

Setting	Value
Java Runtime Environment (JRE)	11
spark.locality.wait	0
spark.serializer	DEFAULT
spark.driver.memory	5 GB
spark.executor.instances	4, 8, 16
spark.executor.cores	16
spark.executor.memory	32 GB

penalty for moving data between nodes is lower than the time penalty for having a node idle.

Due to the volumes of data handled by the system, the JVM parameters *-Xms* and *-Xmx* configured in Cassandra must be raised as needed. For the analysis presented here, and considering the amount of memory presented on each node, *-Xms* and *-Xmx* were set to 24 GB. Memory configuration ends up with 32 GB for Spark, 24 GB for Cassandra and 8 GB for the OS. Additionally, some parameters such as *native_transport_max_frame_size_in_mb* and *commitlog_segment_size_in_mb* were also raised as needed when handling D0 and D3, the two datasets with the largest byte size per zone.

5.3.1. Performance in terms of execution times

In order to analyse the scalability and performance of the system, we have measured the time it took to filter each of the four datasets described in Table 5.3. We should remember here that the whole filtering process encompasses the creation of the rasters, the classification of the points from the rasters and, if selected, the error correction. Times were taken for two execution scenarios: one using the error correction strategy (EC) and the other one without using it (NO-EC). The analysis was carried out using 4, 8 and 16 computing nodes. As a base comparison, a local non-big-data version of the system was also tested. This local version was specially configured to run in one node, without using Cassandra or Spark, being capable

Table 5.6: Relevant Cassandra configuration settings.

Setting	**Value**
Java Runtime Environment (JRE)	8
Heap size	24 GB
Garbage collector	G1
Read/Write/Request timeouts	10–120 seconds
native_transport_max_frame_size_in_mb	256–1024
commitlog_segment_size_in_mb	32–128
disk_optimization_strategy	spinning
concurrent_reads	16
concurrent_writes	128
concurrent_counter_writes	16
concurrent_materialized_view_writes	16

of processing in parallel several zones by sharing the workload among the 16 local nodes.

Performance results for D0, D1 and D2 can be observed in Figures 5.8, 5.9 and 5.10, respectively. Figures show execution times (in hours) and speed-ups in comparison to the reference value (the local version of the algorithm).

Regarding speed-ups, results obtained for D0 were 1.57×, 3.62× and 7.92× with EC and 2.44×, 4.67× and 9.42× with NO-EC (using 4, 8 and 16 nodes, respectively). For D1, speed-ups observed were almost the same in both execution scenarios, being around 2.1×, 4.2× and 8.4×. Finally, for D2, speed-ups were 2.14×, 4.54× and 9.16× with EC and 1.78×, 3.77× and 7.61× with NO-EC. Considering these results, it can be asserted that the base speed-up obtained when moving from a local execution to the big data system using 4 nodes is, on average, 2×. Hence, the big data system shows lineal scalability for all node configurations and datasets, both with EC and NO-EC, by doubling the performance when doubling the nodes available.

Regarding execution times, the fastest configuration observed offering the best quality was the 16 nodes configuration with EC and zones of 400×400 (D1) achieving 3.41 hours. Presumably, execution times observed for D2 should be always better than D1, and times observed for D1 better than D0, since a reduction in the KB per

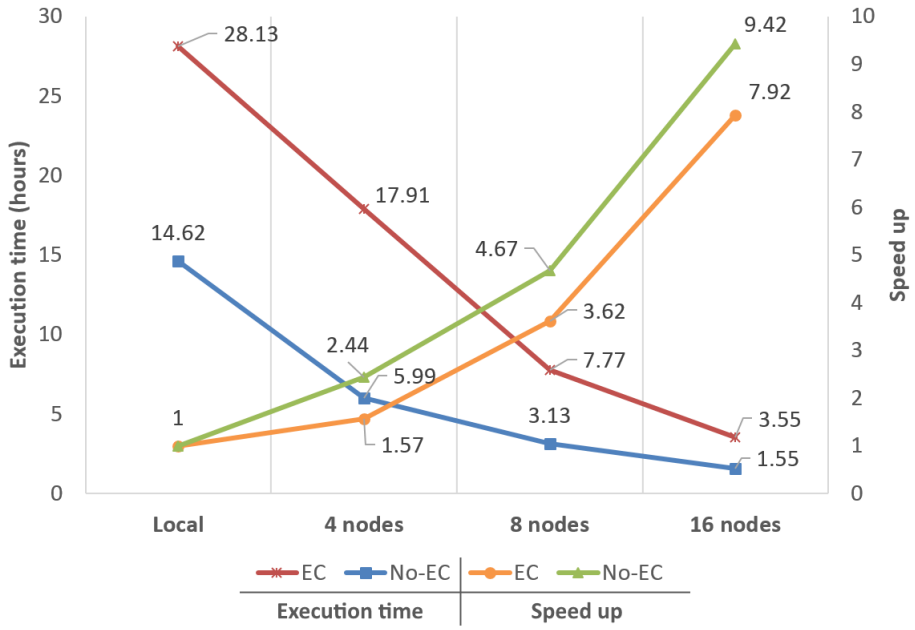


Figure 5.8: Performance analysis and scalability comparison using dataset D0 (1600×1600).

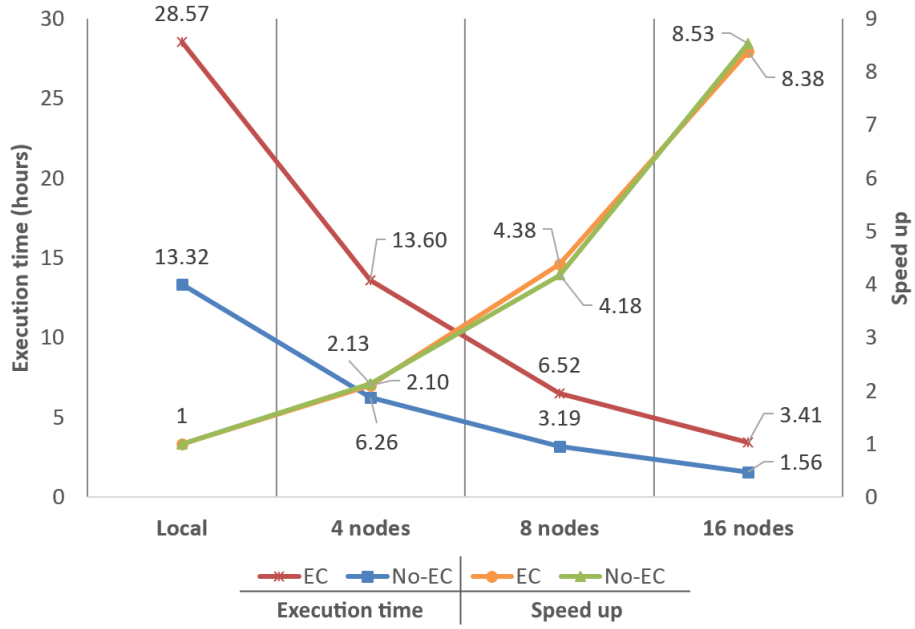


Figure 5.9: Performance analysis and scalability comparison using dataset D1 (400×400).

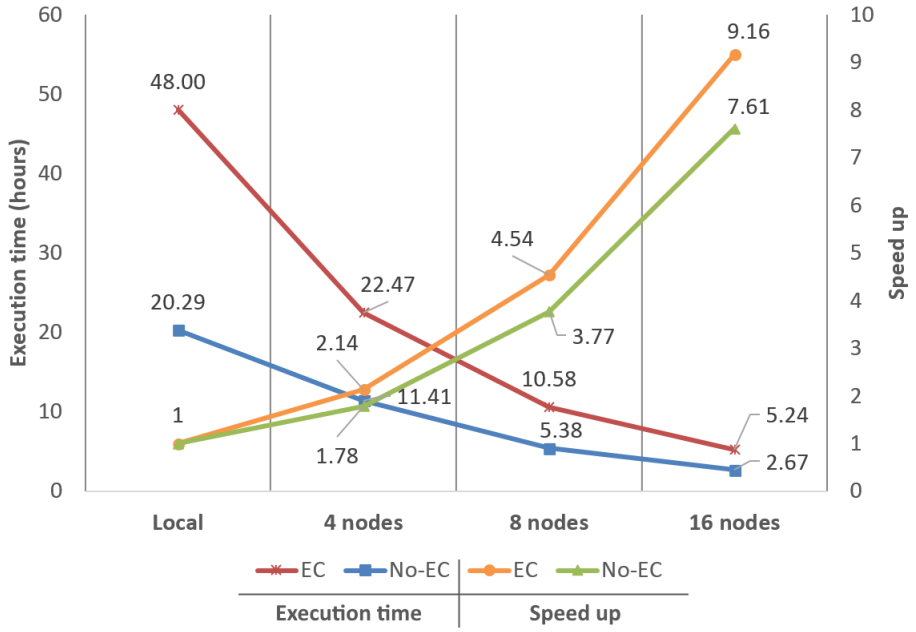


Figure 5.10: Performance analysis and scalability comparison using dataset D2 (100×100).

zone would improve the throughput and latency of Cassandra, as demonstrated in the previous chapter. However, in this comparison, the effects on the reduction in the extent of the zones showed very different results. These differences are explained by the variation on the amount of information that moves between nodes in the two different execution scenarios and the start-up/initialization time penalty of the filtering algorithm. Every time a given processing unit (zone) begins to be filtered, a certain number of data structures must be initialized and some initial computations must be made causing a small start-up/initialization time penalty; hence, the more zones that are filtered, the more start-up/initialization time penalties will occur.

In the first scenario (NO-EC), data movements between nodes are almost non-existent as the Spark-Cassandra connector ensures data locality. Spark workers only apply the filtering algorithm on the zones stored in their own nodes and, as a result, little to no reduction on execution times is obtained by reducing the KB per zone. On the other hand, the start-up penalty of the filtering algorithm increases along with the number of zones to be processed. As a result of the combination of these two performance issues, in this scenario a reduction in the extent of the zones is

very likely to always produce an increment in the execution times.

In the second scenario (EC), there are considerably more data movements between nodes; for example, a given correction patch should be moved between nodes for correcting the errors of adjacent zones that were stored in different nodes. Once the amount of data movements is significant enough, the potential performance gain related to the reduction of the KB per zone begins to show up. The time reduction related to the movement of the data between nodes may end up compensating the time increment related to the start-up penalties, producing an overall decrement in the execution times.

Figure 5.11 shows (using a logarithmic scale) how the execution times vary when changing the extent of the zones, both with NO-EC and EC. It can be clearly seen how there is always a reduction in the execution times when increasing the extent of the zones from 100×100 to 400×400. The reduction in the number of zones to be filtered causes an important improvement in the execution times observed, which helps to compensate the potential performance decrement produced by the increase in the number of bytes that must be moved throughout the system when the boundary errors are corrected. However, when going from 400×400 to 1600×1600, execution times almost stall. There is no a significant improvement with NO-EC, thereby, when the boundary errors are corrected, the amount of data becomes high enough to cause decrement in the performance of the system, leading to the obtention of worse times with EC.

Finally, speed-ups obtained for D3 (Figure 5.12) were 1.9×, 3.06× and 3.57× for NO-EC and 1.82×, 3.01× and 3.99× for EC. As can be observed, although the performance boost of the big data approach was notable, in comparison, results were not as good as those obtained with the other datasets, even starting to stall just at $\approx 4\times$ when running on 16 nodes. These results are explained by, not only the small amount of zones to process in comparison to the available cores (1.19 zones per core), but also by the very large size of the zones ($\approx$ 30 MBs) that are being processed (see the analysis of previous chapter for more information).

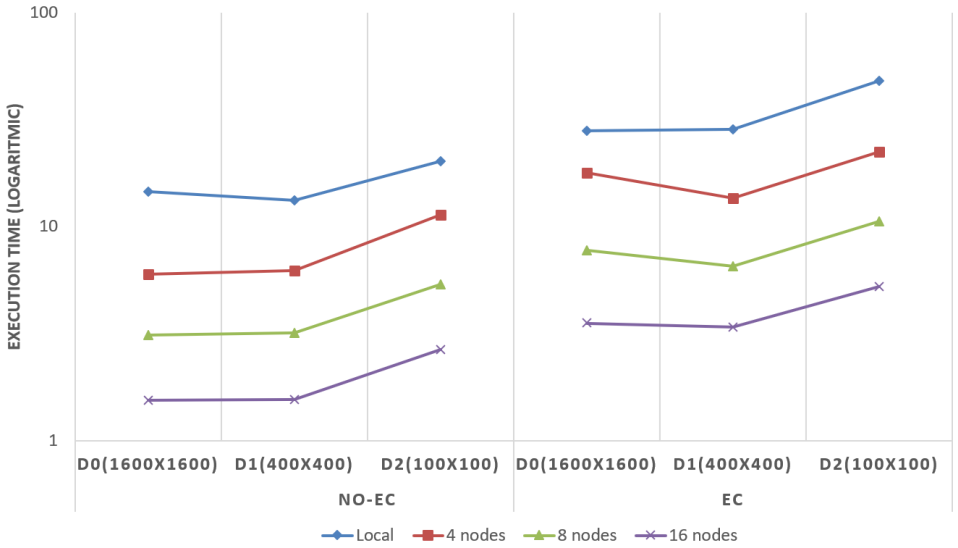


Figure 5.11: Execution times variation (in logarithmic scale) between the usage of different zone extents, both with EC and NO-EC.

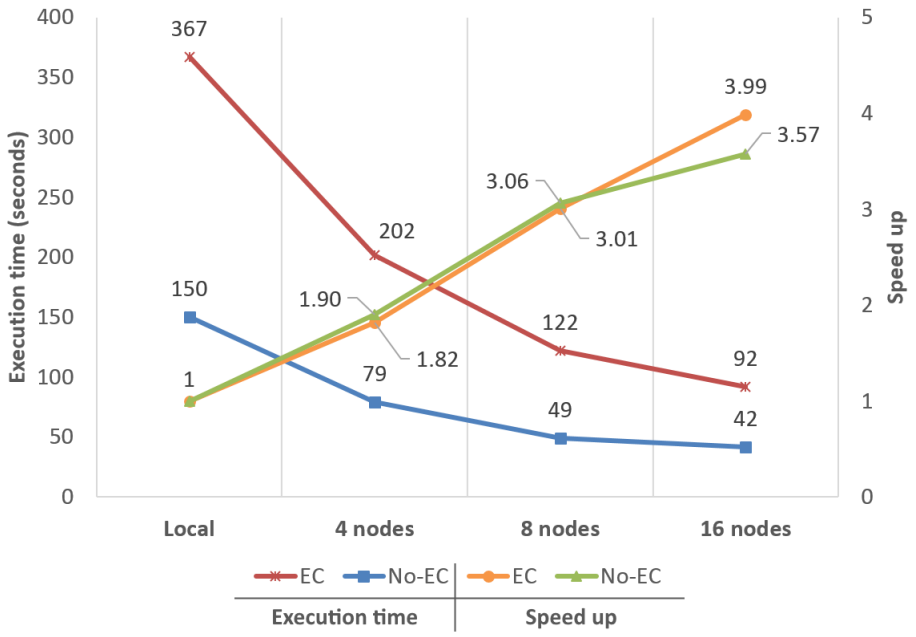


Figure 5.12: Performance analysis and scalability comparison between the local version of the system and the big data approach using 4, 8 and 16 nodes. This test was carried out using dataset D3 to analyse the system under specially unfavourable conditions.

5.3.2. Boundary error correction quality

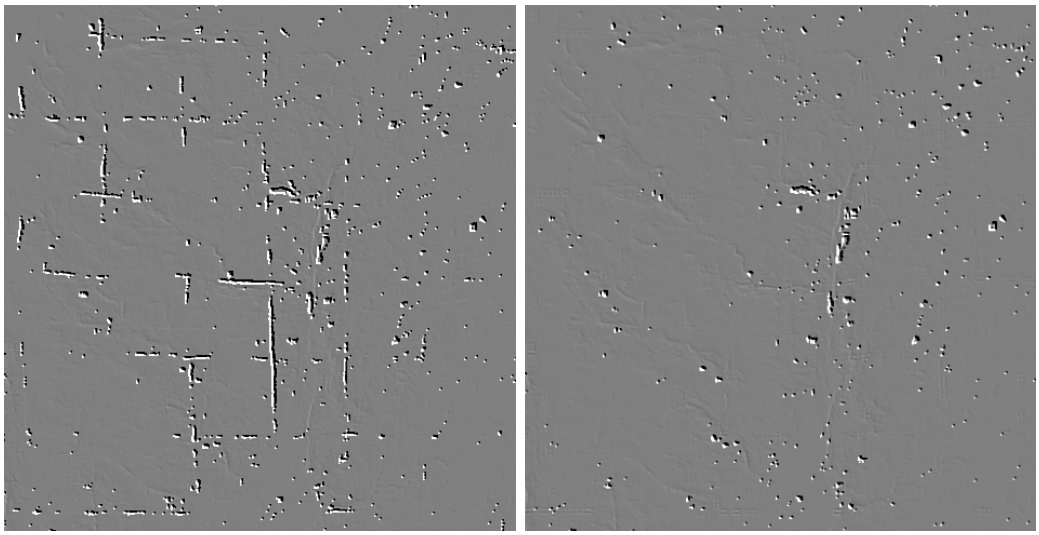
With the intention of analysing our boundary error correction strategy, we have employed two separate methods: a visual analysis using the massive dataset D3 and a quantitative analysis using several samples from the ISPRS Filter Test dataset [109].

Visual comparisons between EC and NO-EC can be observed in Figure 5.13 and Figure 5.14. Images from Figure 5.13 show a close view over two filtered rasters from D3 containing ground points only. The surface shown in Figure 5.13a presents noticeable errors with blank zones that are completely removed in Figure 5.13b. The only blank zones that can be seen in this image correspond to large zones of forest or small groups of houses. On the other hand, Figure 5.14 shows an even closer view over two fully triangulated rasters from the previous raster set. Ground points from the rasters were triangulated using the software tool Global Mapper [7]. As in the previous figure, clear errors can be observed in Figure 5.14a along the boundaries of the zones, while they are completely absent in Figure 5.14b.

A quantitative comparison between EC and NO-EC can be observed in Table 5.7. The comparison was carried out using samples from the ISPRS Filter Test dataset. Each sample was divided into 4 smaller and equally-sized zones to force the appearance of boundary errors. Type I error (rejection of ground points) was measured to compare the classification quality observed on the original samples and on their 4-split counterparts with EC and NO-EC. For these tests, CS was set to 1 due to the low point density of the samples. We should stress here that type I errors are calculated only with points from the rasters, so every point from a raw point cloud not included in the raster counts as an error, leading to the high percentages shown in the table.

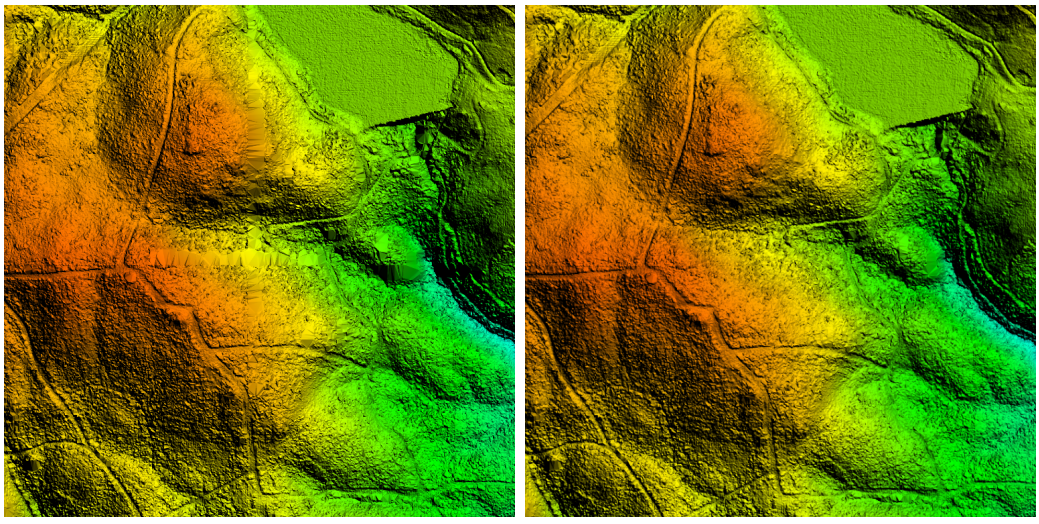
As it can be clearly seen, after applying the error correction strategy the percentage of errors is reduced beyond the level prior to the division of the samples. Errors with EC are slightly inferior to the originals, due to small amounts of extra points contained in the rasters of the divided samples.

Considering a raster created from a zone of 9×9 meters and using CS=1 meter, the resultant raster will contain 81 points distributed in a grid of 9×9 cells. After dividing the zone in 4 smaller areas of 4.5×4.5 meters, the 4 resultant rasters will contain 100 points distributed in 4 grids of 5×5 cells, since those half meters re-



(a) NO-EC (b) EC

Figure 5.13: Filtered rasters containing ground points only. Images represent a close view over an area with 16 zones (8×8 on 16 km^2) from the dataset D3.



(a) NO-EC (b) EC

Figure 5.14: Fully triangulated rasters containing ground points only. Images represent a very close view over an area with 4 zones (2×2 on 1 km^2) from the dataset D3.

Table 5.7: Type I error comparison (lower is better) between EC and NO-EC using several samples from the ISPRS Filter Test dataset.

Sample	**Type I Error (%)**		
	Original	Divided in 4 (NO-EC)	Divided in 4 (EC)
Sample 11	45.99	50.18	44.97
Sample 22	39.71	40.59	39.44
Sample 52	10.79	13.43	10.54
Sample 53	14.87	15.71	14.25
Sample 54	6.80	7.22	6.47
Sample 61	8.03	8.44	7.76

maining imply the presence of an additional cell on each grid dimension. This tiny extra information helps to improve the quality of the classification reducing, as a result, the type I errors.

5.3.3. The importance of an adequate point cloud preprocessing

All results presented in Section 5.3.1 and 5.3.2 reveal the great importance of the decisions taken during the preprocessing stages of the LiDAR datasets. Reducing the size of the zones by reducing their extent may improve the overall system performance; nevertheless, once a certain extent is reached, the performance improvement may get stuck or even get worse, depending on the nature of the geospatial processes running on the system. Additionally, it should also be taken into account whether the quality of the output is related to the size or extent of the zones or not. As shown in Table 5.7 (second and third column), if the size of the zones is reduced by reducing their extent, the amount of boundary surfaces will increase, raising the number of errors on the output.

By way of conclusion, it should be stressed how critical it is to find a suitable balance between the number of zones (NoZ), the size per zone (ZS) and the output quality of the geospatial process. Developing an automated method to determine the optimal number of zones, or their extent, is beyond the scope of this work, but

it would be considered as an interesting part of the future work.

5.3.4. Full point classification

It is intended to expand the type of output of the current filtering algorithm with the goal of offering full point classification. We have developed a naive approach to analyse the potential of this additional feature. To accomplish this task, the current filter output is used as input for a new final stage to classify all the points in the raw point cloud. Based on their X and Y coordinates, each point from the LiDAR zones is placed in a cell from the already filtered rasters. If the height difference between the point located in the raster cell and the point from the zone is less than an user-defined parameter, which we have called height threshold (HT), the new unclassified point is labelled using the same class as the raster point, otherwise the point will be labelled with the opposite class.

Table 5.8 shows a comparison between this naive approach and LAStools [41] using several samples from the ISPRS Filter Test dataset. The LAStools results were obtained from [85], as it is one of the latest research articles about ground classification from LiDAR point clouds. The analysis was carried out comparing total errors (percentage of misclassified points), type I errors (rejection of ground points) and type II errors (acceptance of non-ground points as ground points). The CS parameter was set to 1, again due to the low point density of the samples, while the HT parameter was set to 30 centimetres.

The results obtained show how our approach is better, on average, than LAStools when considering the total errors and the type I errors, but slightly worse when considering type II errors. This is an expected result, since the core design of the *SC-091-12* algorithm was focused on the obtention of rasters with ground points that could be easily triangulated for generating high quality DTMs.

The inclusion of the full point classification has been established as future work as well as the reduction in the percentage of errors.

Table 5.8: A comparison between the full point cloud classification errors (lower is better) obtained by the naive method proposed and LAStools.

Sample	**Total Error (%)**		**Type I Error (%)**		**Type II Error (%)**	
	Our approach	LAStools	Our approach	LAStools	Our approach	LAStools
Sample 11	17.05	17.67	27.06	26.94	3.61	5.18
Sample 12	5.28	6.97	8.25	12.87	2.16	0.77
Sample 21	1.79	6.66	1.42	7.98	3.09	1.87
Sample 53	14.33	14.37	14.55	14.84	8.92	3.24
Sample 61	7.29	17.24	7.38	17.85	4.89	0.40
Average	9.15	12.58	11.732	16.10	4.53	2.29

5.3.5. Point triangulation

As was already explained in Section 5.1.1, it is up to the users to triangulate the results obtained from the filtering process by using the software and properties they consider most appropriate. In order to obtain the fully triangulated representation of the four filtered zone rasters shown in Figure 5.14, the software tool Global Mapper was employed running on commodity hardware, taking no more than 2 seconds to finish the triangulation of the whole surface.

It is important to highlight that, although several sets of rasters could be triangulated in parallel, it would be necessary to blend them all together in order to obtain a continuous surface across many different sets, which would imply additional computation time. However, it is here where the importance and potential of a system like the one presented in this chapter clearly appears, bringing the opportunity of including additional computational stages for obtaining full triangulated DTMs, achieving very low processing times in comparison to traditional desktop solutions, and even reducing them close to real time.

Chapter 6

Conclusions and future work

Nowadays, LiDAR technology stands out as one of the most valuable sources of geospatial information and it is considered as a challenge when it comes to developing efficient software to handle the volumes of data this surveying method is able to collect. This particularity forces researchers to constantly look for new approaches and solutions in order to overcome all constraints related to the manipulation of such volumes of information. Throughout this Thesis, a wide range of contributions have been presented to drastically improve the performance and functionality of many critical elements in the field of aerial LiDAR data.

The visualization framework presented in Chapter 2 offers great flexibility and an adequate workflow for users thanks to its web-based approach, together with its efficient data querying capabilities and its fast data loading mechanics. Unlike most of the other proposals, it shows high stability and performance, achieving real time interaction, handling around 103 million points and up to 281 million with an acceptable level of interaction. Additionally, its geospatial measurement tools constitute one of the main advantages over the other proposals available.

It could be argued that a full-resolution approach, like the one presented in Chapter 2, may not be useful or it could show poor performance under a situation where an excessively large point cloud were fetched into the ROI (region of interest); nevertheless, as was briefly explained during the preface of this work, since this approach was meant to be included in an application with the purpose of providing accurate measurement tools over very delimited areas, said situation will be highly

unlikely. For example, it would be very unlikely to create a complex volumetric object over very large extents of land surfaces, such as an entire province or region, or even an entire city, with the definition of ROIs being limited to relatively small to medium sized areas in most situations.

In Chapter 3, we have demonstrated how multi-resolution, out-of-core techniques can be implemented though non-redundant data structures, to support web-based point cloud rendering. Through a special non-redundant rearrangement and storage of the points, we were able to avoid the creation of unnecessary static and precomputed elements usually required in other multi-resolution approaches. Thanks to our proposal, depending on the characteristics of the point clouds and the different LODs (levels of detail) created for them, the reduction in the storage requirements on the server side can be notable, as well as the reduction in the network traffic. These optimizations become especially meaningful achievements in contexts where large amounts of LiDAR datasets are constantly collected 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 and rendered in real time, even in mobile devices, where memory capacity is very limited, or in browsers with hard memory restrictions, such as Google Chrome. The proposal presented in Chapter 3 was tested against one of the most known and well rated LiDAR visualization frameworks, overcoming all its performance results.

During Chapter 4 of this Thesis, it has been proven how web-based real-time visualization of massive LiDAR datasets can be supported by big data storage technologies without any drawback or penalty in performance or user experience, while gaining all of the usual benefits of big data, such as scalability and fault tolerance. Additionally, with a view towards the future, systems using these kinds of technologies would be already prepared to incorporate large-scale data processing technologies such as Spark or Flink. In Chapter 4, some of the most adopted and mature storage technologies were chosen, not only to analyse their viability as substitutes of traditional storage technologies, but also to compare their advantages and disadvantages, forming a guide (or starting point) for their adoption or avoidance in other use cases involving LiDAR data.

In Chapter 5, we have proven how big data technologies, in this case Cassan-

dra and Spark, provide very powerful solutions for the large-scale parallelization of geospatial processes. The distributed computing system presented in this chapter allowed us to greatly improve the performance of a filtering algorithm [22] specially designed to obtain rasters of ground-only points for DTM generation. In addition to the outstanding computing capacity of these two technologies working together, their highly programmable design provides the opportunity to further improve and expand the functionality of the geospatial processes through the easy inclusion of new computational stages, such as the error correction stage also introduced in Chapter 5. Said strategy allowed us to correct the classification errors that can be found on the boundaries of adjacent zones independently processed by the filtering algorithm. Performance results obtained show how our proposal was capable of reducing the time it took to process 28 billion points in a single machine, from 28.57 hours down to 3.41 hours when correcting the boundary errors, and from 13.32 hours down to 1.56 with no error correction, obtaining a speed-up of 8.4× for deployments of 16 nodes (see Figure 5.9 of Chapter 5). The use of a big data approach does not only bring distributed computing to LiDAR data, but also all the common storage advantages associated to this type of technologies, such as high data availability and scalability or fault tolerance. GIS centres, governmental institutions or any other kind of group working with very large volumes of data will greatly benefit from the proposal presented in this chapter and, with the appropriate amount of computational resources, the entire process of obtaining fully triangulated DTMs could get near to real-time processing.

Finally, during the development of the Thesis, a lossless compression method was also presented to support many of the main features, algorithms and systems described. Reaching compression ratios of 0.1 (a file size reduction of 90%), our method outperforms LASzip, the preferred compression method used with LAS files, while allowing total control over the data due to not being restricted to the characteristics of a pre-existing format, and therefore, data files employed may be adapted to maximize the performance of the elements that make use of them.

Future work

Currently, we are heading towards finishing the full classification of point clouds using a more refined and complex algorithm than the one presented in Section 5.3.4, so that users can choose whether to obtain just a filtered raster from a given point cloud or the whole filtered point cloud. Additionally, we plan to include an optional final processing stage for obtaining fully triangulated DTMs, as explained in Section 5.3.5. The development of an automated method to determine the optimal number and extent of the processing units could be considered for minimizing the user intervention while maximizing the performance of the distributed computed system, as was already mentioned in Section 5.3.3.

The autonomous nature of all computing stages, along with the low processing times achieved, opens up the possibility of considering the system as a service-oriented solution for on-demand DTM/DSM generation, which would be a highly useful and unique service for many users in the LiDAR field, and one which could get near to real-time processing with the appropriate amount of computational resources. The distributed computed system presented in Chapter 5 serves as base system to a larger one offering several geospatial processes in the form of a library of geospatial processes, with the aim of applying those process over entire point clouds or only ROIs defined by the users. One of these additional options could be the massive point triangulation commented in Section 5.3.5.

All contributions presented in this Thesis will be integrated onto the same system, BETi (**B**ig data for the **E**xploration of **T**errain **I**nfomation) offering a complete and multi-functional solution for data distribution, replication, availability and client-side and server-side processing. The system will be capable of offering parallel access to a very large collection of massive point clouds, including a wide geospatial process library powered by a highly scalable distributed computing system under a queue system. Visualization of public and private datasets will be granted through the web-based clients together with all geospatial measurement tools already described. Finally, we should highlight here that part of the research presented during this work is now being used in another Thesis developed in the Computer Graphics Systems research group of the Hasso Plattner Institute and the Computer Architecture Group of the University of A Coruña. Many of the concepts presented in

Chapter 3 regarding efficient adaptive data structures generated at runtime are being used to improve the output quality and performance of semantic enrichment algorithms for indoor point clouds. Additionally, an improved revision of the multi-resolution and out-of-core approach presented in said chapter is being developed with the aim of correcting some of its weakness and further expanding the resource optimizations achieved.

Bibliography

[1] V. Abramova and J. Bernardino. Nosql databases: Mongodb vs cassandra. In *Proceedings of the International C* Conference on Computer Science and Software Engineering*, pages 14–22, 07 2013.

[2] T. Akenine-Moller, E. Haines, and N. Hoffman. *Real-Time Rendering*. A. K. Peters, Ltd., 3rd edition, 2008.

[3] Alexander Krivutsenko. Lidarview. `http://lidarview.com/`. Accessed on 02/20/2019.

[4] American Society for Photogrammetry and Remote Sensing. LAS specification. `https://www.asprs.org/divisions-committees/lidar-division/laser-las-file-format-exchange-activities`. Accessed on 02/20/2019.

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

[6] A. Beck. Airborne laser scanning discrete echo and full waveform signal comparison. `https://commons.wikimedia.org/wiki/File:Airborne_Laser_Scanning_Discrete_Echo_and_Full_Waveform_signal_comparison.svg`. New labels and text by David Deibe, https://creativecommons.org/licenses/by-sa/3.0/legalcode. Accessed on 02/19/2019.

[7] Blue Marble Geographics. Global mapper - all-in-one gis software. `https://www.bluemarblegeo.com/products/global-mapper.php`. Accessed on 05/10/2019.

[8] J. Boehm. File-centric organization of large lidar point clouds in a big data context. In *IQmulus First Workshop on Processing Large Geospatial Data*, pages 69–76, 2014.

[9] M. Bóo, M. Amor, and J. Döllner. Unified hybrid terrain representation based on local convexifications. *GeoInformatica*, 11(3):331–357, February 2007.

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

[11] J. Boudreau, R. F. Nelson, H. A. Margolis, A. Beaudoin, L. Guindon, and D. S. Kimes. Regional aboveground forest biomass using airborne and spaceborne lidar in Québec. *Remote Sensing of Environment*, 112(10):3876 – 3890, 2008.

[12] M. Brédif, B. Vallet, and B. Ferrand. Distributed dimensonality-based rendering of lidar point clouds. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XL-3/W3:559–564, 2015.

[13] C. A. Brunori, R. Civico, F. R. Cinti, and G.Ventura. 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, 2013.

[14] S. Buján, E. González-Ferreiro, L. Barreiro-Fernández, I. Santé, E. Corbelle, and D. Miranda. Classification of rural landscapes from low-density lidar data: is it theoretically possible? *International Journal of Remote Sensing*, 34(16):5666–5689, 2013.

[15] R. Buyya, A. Dastjerdi, and R. Calheiros. *Big Data Principles and Paradigms*. Elsevier, 2016.

[16] C. P. Chen and C.-Y. Zhang. Data-intensive applications, challenges, techniques and technologies: A survey on big data. *Information Sciences*, 275:314 – 347, 2014.

[17] M. Chi, A. Plaza, J. A. Benediktsson, Z. Sun, J. Shen, and Y. Zhu. Big data for remote sensing: Challenges and opportunities. *Proceedings of the IEEE*, 104(11):2207–2219, Nov 2016.

[18] M. Comino, C. Andújar, A. Chica, and P. Brunet. Error-aware construction and rendering of multi-scan panoramas from massive point clouds. *Computer Vision and Image Understanding*, 157:43 – 54, 2017.

[19] R. Concheiro. *Real Time Rendering of Parametric Surfaces on the GPU.* Departamento de Electrónica y Sistemas, Universidade da Coruña, 2013.

[20] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears. Benchmarking cloud serving systems with ycsb. In *Proceedings of the 1st ACM Symposium on Cloud Computing*, SoCC '10, pages 143–154. ACM, 2010.

[21] Couchbase, Inc. Couchbase nosql database. `https://www.couchbase.com/`. Accessed on 06/13/2019.

[22] R. Crecente, E. González, D. A. Arias, D. Miranda, and M. Suárez. LIDAR2MDTPlus generación de modelos digitales de terreno de pendiente variable a partir de datos lidar mediante filtro morfológico adaptativo y computación paralela sobre procesadores multinúcleo. Software registration: Universidade de Santiago, Spain. SC-091-12 03/28/2012.

[23] Danga Interactive. Memcached - a distributed memory object caching system. `http://www.memcached.org/`. Accessed on 06/13/2019.

[24] Datastax. Github - datastax/spark-cassandra-connector: Datastax spark cassandra connector. `https://github.com/datastax/spark-cassandra-connector`. Accessed on 01/22/2019.

[25] K. Debattista, K. Bugeja, S. Spina, T. Bashford-Rogers, and V. Hulusic. Frame rate vs resolution: A subjective evaluation of spatiotemporal perceived quality under varying computational budgets. *Computer Graphics Forum*, 00(0):1–12, 2017.

[26] K. Debattista, K. Bugeja, S. Spina, T. Bashford-Rogers, and V. Hulusic. Frame rate vs resolution: A subjective evaluation of spatiotemporal perceived

quality under varying computational budgets. *Computer Graphics Forum*, 37(1):363–374, 2018.

[27] E. Dede, M. Govindaraju, D. Gunter, R. S. Canon, and L. Ramakrishnan. Performance evaluation of a MongoDB and Hadoop platform for scientific data analysis. In *Proceedings of the 4th ACM Workshop on Scientific Cloud Computing*, Science Cloud '13, pages 13–20. ACM, 2013.

[28] D. Deibe, M. Amor, and R. Doallo. ViLMA (VIsualization for Lidar data using a Multi-resolution Approach). Software registration: Universidade da Coruña and Universidade de Santiago de Compostela, Spain. C-388-2018 11/19/2018.

[29] D. Deibe, M. Amor, and R. Doallo. Big data storage technologies: a case study for web-based LiDAR visualization. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 3831–3840, Dec 2018.

[30] D. Deibe, M. Amor, and R. Doallo. BETi: Sistema para la gestión y procesamiento de datos masivos LiDAR. In *XXX Jornadas de Paralelismo - JP2019*, pages 516–523, September 2019.

[31] D. Deibe, M. Amor, and R. Doallo. Big data geospatial processing for aerial LiDAR datasets. 2019. Submitted for publication.

[32] D. Deibe, M. Amor, and R. Doallo. Supporting multi-resolution out-of-core rendering of massive LiDAR point clouds through non-redundant data structures. *International Journal of Geographical Information Science*, 33(3):593–617, 2019.

[33] D. Deibe, M. Amor, R. Doallo, R. Crecente, D. Miranda, and M. Cordero. VGLiDAR 1.0 visualizador gallego de datos lidar. Software registration: Universidade da Coruña and Universidade de Santiago de Compostela, Spain. C-423-2015 11/23/2015.

[34] D. Deibe, M. Amor, R. Doallo, R. Crecente, D. Miranda, and M. Cordero. VGLiDAR: Una herramienta de procesamiento de datos LiDAR en la GPU usando WebGL. In *XXVI Jornadas de Paralelismo - JP2015*, pages 146–151, September 2015.

[35] D. Deibe, M. Amor, R. Doallo, D. Miranda, and M. Cordero. GVLiDAR: An interactive web-based visualization framework to support geospatial measures on LiDAR data. *International Journal of Remote Sensing.*, 38(3):827–849, 2017.

[36] Dielmo 3D S.L. Dielmo. `http://www.dielmo.com/`. Accessed on 19/2/2019.

[37] S. Discher, R. Richter, and J. Döllner. *Interactive and View-Dependent See-Through Lenses for Massive 3D Point Clouds*, pages 49–62. Springer International Publishing, 2017.

[38] D. Flanagan. *JavaScript: The Definitive Guide.* O'Reilly Media, Inc., 6st edition, 2011.

[39] Fugro Geospatial Services. FrugoViewer. `http://www.fugroviewer.com/`. Accessed on 02/20/2019.

[40] Z. Gao, L. Nocera, M. Wang, and U. Neumann. Visualizing aerial lidar cities with hierarchical hybrid point-polygon structures. In *Proceedings of Graphics Interface 2014*, GI '14, pages 137–144. Canadian Information Processing Society, 2014.

[41] R. GmbH. Lastools. `https://rapidlasso.com/lastools/`. Accessed on 05/15/2019.

[42] GNU Project - Free Software Foundation. Gzip. `https://www.gnu.org/software/gzip/`. Accessed on 01/22/2019.

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

[44] Gobierno de Canarias. Visor IDECanarias. `http://visor.grafcan.es/visorweb/`. Accessed on 02/20/2019.

[45] E. González-Ferreiro, D. Miranda, L. Barreiro-Fernández, S. Buján, J. García-Gutiérrez, and U. Diéguez-Aranda. Modelling stand biomass fractions in galician eucalyptus globulus plantations by use of different lidar pulse densities. *Forest Systems*, 22(3):510–525, 2013.

[46] P. Goswami, F. Erol, R. Mukhi, R. Pajarola, and E. Gobbetti. 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, 2013.

[47] M. Gross and H. Pfister. *Point-Based Graphics*. Morgan Kaufmann Publishers Inc., 2007.

[48] J. Han, H. E, G. Le, and J. Du. Survey on nosql database. In *2011 6th International Conference on Pervasive Computing and Applications*, pages 363–366, Oct 2011.

[49] H. Hasegawa, H. P. Sato, and J. Iwahashi. Continuous caldera changes in miyakejima volcano after 2001. *Bulletin of Geospatial Information Authority of Japan*, 54:60–64, 2007.

[50] I. A. T. Hashem, I. Yaqoob, N. B. Anuar, S. Mokhtar, A. Gani, and S. U. Khan. The rise of big data on cloud computing: Review and open research issues. *Information Systems*, 47:98 – 115, 2015.

[51] B. Höfle and N. Pfeifer. Correction of laser scanning intensity data: Data and model-driven approaches. *ISPRS Journal of Photogrammetry and Remote Sensing*, 62(6):415 – 433, 2007.

[52] D. S. Howard Butler, Christopher Schmidt and J. Livni. Spatial reference. `http://spatialreference.org/`. Accessed on 02/27/2019.

[53] H. Hu, Y. Wen, T. S. Chua, and X. Li. Toward scalable systems for big data analytics: A technology tutorial. *IEEE Access*, 2:652–687, 2014.

[54] Instituto Geográfico Nacional. Plan Nacional de Ortofotografía Aérea (PNOA). `http://pnoa.ign.es/presentacion`. Accessed on 02/20/2019.

[55] Instituto Geográfico Nacional. PNOA, download center. `http://centrodedescargas.cnig.es/CentroDescargas/buscadorCatalogo.do?codFamilia=LIDAR`. Accessed on 02/19/2019.

[56] M. Isenburg. Laszip: lossless compression of LiDAR data. `http://lastools.org/download/laszip.pdf`. Accessed on 06/11/2019.

[57] G. Kereszturi, J. Procter, S. Cronin, K. Nemeth, M. Bebbington, and J. Lindsya. LiDAR based quantification of lava flow susceptibility in the City of Auckland (New Zealand). *Remote Sensing of Environment*, 125:198–213, 2012.

[58] C. Koca and U. Güdükbay. A hybrid representation for modeling, interactive editing, and real-time visualization of terrains with volumetric features. *International Journal of Geographical Information Science*, 28(9):1821–1847, 2014.

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

[60] M. Kuder and B. Žalik. Web-based LiDAR visualization with point-based rendering. In *2011 Seventh International Conference on Signal-Image Technology and Internet-Based Systems (SITIS)*, pages 38–45, 2011.

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

[62] J. C. Landy, A. S. Komarov, and D. G. Barber. Numerical and Experimental Evaluation of Terrestrial LiDAR for Parameterizing Centimeter-Scale Sea Ice Surface Roughness. *IEEE Transactions on Geoscience and Remote Sensing*, 53(9):4887–4898, 2015.

[63] S. Li, S. Dragicevic, F. A. Castro, M. Sester, S. Winter, A. Coltekin, C. Pettit, B. Jiang, J. Haworth, A. Stein, and T. Cheng. Geospatial big data handling theory and methods: A review and research challenges. *ISPRS Journal of Photogrammetry and Remote Sensing*, 115:119 – 133, 2016.

[64] LiDAR Online. LiDAR Online web tools. `http://www.lidar-online.com/`. Accessed on 02/20/2019.

[65] S. Lin, H. Chen, and F. Hu. A workload-driven approach to dynamic data balancing in MongoDB. In *2015 IEEE International Conference on Smart City/SocialCom/SustainCom (SmartCity)*, pages 786–791, Dec 2015.

[66] Y. Ma, H. Wu, L. Wang, B. Huang, R. Ranjan, A. Zomaya, and W. Jie. Remote sensing big data computing: Challenges and opportunities. *Future Generation Computer Systems*, 51:47 – 60, 2015.

[67] Marek9134. Lidar-i lend. `https://commons.wikimedia.org/wiki/File:LiDAR-i_lend.gif`. Colour, new labels and text by David Deibe, https://creativecommons.org/licenses/by-sa/3.0/legalcode. Accessed on 02/19/2019.

[68] O. Martinez Rubi, S. Verhoeven, M. van Meersbergen, M. Schütz, P. Oosterom, R. Goncalves, and T. Tijssen. Taming the beast: Free and open-source massive point cloud web visualization. In *Capturing Reality*, November 2015.

[69] MongoDB, Inc. Mongodb. `http://www.mongodb.com/`. Accessed on 01/22/2019.

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

[71] A. Munshi, D. Ginsburg, and D. Shreiner. *OpenGL ES 2.0 Programming Guide*. Addison-Wesley, 2008.

[72] Neo4j, Inc. Neo4j graph platform – the leader in graph databases. `https://neo4j.com/`. (Accessed on 06/13/2019).

[73] Node.js Foundation. Express - node.js web application framework. `http://expressjs.com/`. Accessed on 02/20/2019.

[74] OpenTopography. CA13, PG&E Diablo Canyon Power Plant (DCPP): San Simeon, CA Central Coast. `http://opentopo.sdsc.edu/datasetMetadata?otCollectionID=OT.032013.26910.2`. Accessed on 02/20/2019.

[75] OpenTopography. Pg&e diablo canyon power plant (dcpp): Los osos, ca central coast. `http://opentopo.sdsc.edu/lidarDataset?opentopoID=OTLAS.022013.26910.3`. Accessed on 02/19/2019.

[76] OpenTopography. Sunset crater volcano national monument, az. `http://opentopo.sdsc.edu/datasetMetadata?otCollectionID=OT.022015.26912.1`. Accessed on 02/19/2019.

[77] E. Paredes, M. Bóo, M. Amor, J. Bruguera, and J. Döllner. Extended hybrid meshing algorithm for multiresolution terrain models. *International Journal of Geographical Information Science*, 26(5):771–793, May 2011.

[78] T. Parisi. *WebGL: Up and Running*. O'Reilly Media, Inc., 1st edition, 2012.

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

[80] H. Persson, J. Wallerman, H. Olsson, and J. E. Fransson. Estimating forest biomass and height using optical stereo satellite data and a dtm from laser scanning data. *Canadian Journal of Remote Sensing*, 39(3):251–262, 2013.

[81] T. Rabl, S. Gómez-Villamor, M. Sadoghi, V. Muntés-Mulero, H.-A. Jacobsen, and S. Mankovskii. Solving big data challenges for enterprise application performance management. *Proc. VLDB Endow.*, 5(12):1724–1735, Aug. 2012.

[82] V. C. Recarey, D. M. Barrós, and D. A. A. Prado. Optimización de los parámteros del algoritmo de filtrado LiDAR2MDTPlus, para la obtención de Modelos Digitales del Terreno con LiDAR. Bachelor's Thesis. Higher Polytechnic Engineering School, University of Santiago de Compostela (USC), September 2014.

[83] Redis Labs. Redis. `http://redis.io/`. Accessed on 01/22/2019.

[84] R. Richter, S. Discher, and J. Döllner. *3D Geoinformation Science: The Selected Papers of the 3D GeoInfo 2014*, chapter Out-of-Core Visualization of Classified 3D Point Clouds, pages 227–242. Springer International Publishing, 2015.

[85] A. Rizaldi, C. Persello, C. Gevaert, and S. Oude Elberink. *Fully Convolutional Networks for Ground Classification from LiDAR Point Clouds*, pages 231–238. ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences. International Society for Photogrammetry and Remote Sensing (ISPRS), 6 2018.

[86] Robot Operating System. Megatree - ros wiki. `http://wiki.ros.org/megatree`. Accessed on 02/19/2019.

[87] M. B. Rodríguez, E. Gobbetti, F. Marton, and A. Tinti. Coarse-grained multiresolution structures for mobile exploration of gigantic surface models. In *SIGGRAPH Asia 2013 Symposium on Mobile Graphics and Interactive Applications*, SA '13, pages 1–6. ACM, 2013.

[88] J. J. Roering, B. H. Mackey, J. A. Marshall, K. E. Sweeney, N. I. Deligne, A. M. Booth, A. L. Handwerger, and C. Cerovski-Darriau. You are HERE: Connecting the dots with airborne LiDAR for geomorphic fieldwork. *Geomorphology*, 200:172–183, 2013.

[89] A. H. Sallenger Jr., W. Krabill, J. Brock, R. Swift, M. Jansen, S. Manizade, B. Richmond, M. Hampton, and D. Eslinger. Airborne laser study quantifies El Niño-induced coastal change. *Eos, Transactions, American Geophysical Union*, 80(8):89–92, 1999.

[90] M. Schuetz. Potree. `http://potree.org/`. Accessed on 02/20/2019.

[91] U. F. Service. Fusion/ldv lidar analysis and visualization software. `http://forsys.cfr.washington.edu/fusion/fusion_overview.html`. Accessed on 01/22/2019.

[92] solid IT. Db-engines ranking - popularity ranking of database management systems. `http://db-engines.com/en/ranking`. Accessed on 01/22/2019.

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

[94] P. Tarolli and D. G. Tarboton. A new method for determination of Most Likely Initiation Points and the evaluation of Digital Terrain Model scale in terrain stability mapping. *Hydrology and Earth System Sciences Discussions*, 3(2):395–425, Apr. 2006.

[95] The Apache Software Foundation. Apache cassandra. `http://cassandra.apache.org/`. Accessed on 01/22/2019.

[96] The Apache Software Foundation. Apache hadoop 3.0.0 - hdfs architecture. http://hadoop.apache.org\/docs/r3.0.0/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html. Accessed on 01/22/2019.

[97] The Apache Software Foundation. Apache hbase home. http://hbase.apache.org/. Accessed on 06/13/2019.

[98] The Apache Software Foundation. The apache http server project. http://httpd.apache.org/. Accessed on 02/20/2019.

[99] The Apache Software Foundation. Apache jmeter - apache jmeter™. https://jmeter.apache.org/. Accessed on 02/20/2019.

[100] The Apache Software Foundation. Apache spark™ - unified analytics engine for big data. http://spark.apache.org/. Accessed on 01/22/2019.

[101] The Apache Software Foundation. Welcome to apache giraph! http://giraph.apache.org/. Accessed on 06/13/2019.

[102] The Apache Software Foundation. Apache flink: Stateful computations over data streams. https://flink.apache.org/, 2019. Accessed on 03/11/2019.

[103] The Apache Software Foundation. Apache hadoop. http://hadoop.apache.org/, 2019. Accessed on 03/11/2019.

[104] The Apache Software Foundation. Apache storm. http://storm.apache.org/, 2019. Accessed on 03/11/2019.

[105] The Khronos Group Inc. OpenGL Shading Language. https://www.khronos.org/registry/OpenGL/index_gl.php#apispecs. Accessed on 02/20/2019.

[106] The Khronos Group Inc. WebGL. https://www.khronos.org/webgl. Accessed on 02/20/2019.

[107] Uday Verma. Plas.io. http://plas.io/. Accessed on 02/20/2019.

[108] Universidade de Santiago de Compostela (USC). Laboratorio do territorio (Laborate). http://laborate.usc.es/index.html. Accessed on 01/22/2019.

[109] University of Twente. Isprs test sites. `https://www.itc.nl/isprs/wgIII-3/filtertest/downloadsites`. Accessed on 04/27/2019.

[110] J. Veiga, R. R. Expósito, X. C. Pardo, G. L. Taboada, and J. Touriño. Performance evaluation of big data frameworks for large-scale data analytics. In *2016 IEEE International Conference on Big Data (Big Data)*, pages 424–431, Dec 2016.

[111] G. Ventura, G. Vilardo, C.Terranova, and E. B. Sessa. 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, 2011.

[112] L. Wang, Y. Ma, J. Yan, V. Chang, and A. Y. Zomaya. pipscloud: High performance cloud computing for remote sensing big data management and processing. *Future Generation Computer Systems*, 78:353 – 368, 2018.

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

[114] B. Yang, W. Shi, and Q. Li. An integrated TIN and grid method for constructing multi-resolution digital terrain models. *International Journal of Geographical Information Science*, 19(10):1019–1038, Nov. 2005.

[115] C. Yang, M. Yu, F. Hu, Y. Jiang, and Y. Li. Utilizing cloud computing to address big geospatial data challenges. *Computers, Environment and Urban Systems*, 61:120 – 128, 2017.

[116] S. Yuan, S. Zhu, D. Li, W. Luo, Z. Yu, L. Yuan, and D. Hildenbrand. Feature preserving multiresolution subdivision and simplification of point clouds: A conformal geometric algebra approach. *Mathematical Methods in the Applied Sciences*, 0(0), 2017.

[117] J. Zhang. Multi-source remote sensing data fusion: status and trends. *International Journal of Image and Data Fusion*, 1(1):5–24, 2010.

[118] K. Zhang, S.-C. Chen, D. Whitman, M.-L. Shyu, J. Yan, and C. Zhang. A progressive morphological filter for removing nonground measurements from

airborne lidar data. *IEEE Transactions on Geoscience and Remote Sensing*, 41(4):872–882, April 2003.

[119] X. Zhou, W. Li, and S. T. Arundel. A spatio-contextual probabilistic model for extracting linear features in hilly terrains from high-resolution dem data. *International Journal of Geographical Information Science*, 33(4):666–686, 2019.

Apéndice A

Resumo Estendido en Galego

Na actualidade, a tecnoloxía LiDAR (*Light Detection And Ranging*) constitúe unha das fontes de información xeográfica máis importantes e valiosas debido á súa capacidade para proporcionar datos xeográficos en forma de nubes de puntos cun alto nivel de detalle. Mediante este tipo de información, a tecnoloxía LiDAR trouxo grandes beneficios para unha extensa variedade de campos científicos e profesionais, como a enxeñaría agroforestal, a arqueoloxía, a robótica ou os vehículos autónomos, entre moitos outros.

Algunhas das nubes de maior detalle e mais calidade poden estar formadas por varios miles de millóns de puntos, tendo á súa vez cada punto asociadas diversas propiedades tales como as coordenadas (*x, y, z*), a cor RGB ou o tempo GPS no momento da súa obtención. As grandes coleccións que reúnen un gran número destas nubes de puntos masivas normalmente superan o terabyte, ou incluso o petabyte, de espazo en disco, o que provoca que a tecnoloxía LiDAR sexa considerada coma un reto formidable á hora de desenvolver aplicacións eficientes para xestionar tales volumes de información. En contextos profesionais ou científicos nos que existen altas taxas de recollida de datos, o uso de sistemas altamente escalables para almacenar, distribuír e procesar todolos datos de recente adquisición convértese nun requisito indispensable. Ademais, os usuarios altamente especializados tenden a demandar altos niveis de rendemento e eficiencia nas súas máquinas cliente, non só en solucións de escritorio de alta gama, senón tamén en tabletas, portátiles ou mesmo en teléfonos intelixentes. Dende o punto de vista da capacidade hardware, case calque-

ra tipo de recurso; ben sexa a potencia de computación do procesador (CPU), a potencia da tarxeta gráfica (GPU), a velocidade de rede ou a memoria principal, resulta insuficiente á hora de xestionar os grandes volumes de datos LiDAR, facendo indispensable o uso de algoritmos e sistemas de software altamente eficientes.

Así, o obxectivo desta tese consistiu en propoñer unha serie de novos enfoques, algoritmos e sistemas para mellorar ou aportar novidosas solucións nunha ampla gama de procesos relacionadas co uso de datos LiDAR de tipo aéreo (ALS). Como a tecnoloxía LiDAR destaca por ser un marco de coñecemento que abarca un gran número de dimensións, dende o almacenamento, a distribución, o acceso simultáneo a datos, a visualización ou procesamento, en primeiro lugar decidiuse establecer unha xerarquía baseada na magnitude dos problemas a resolver, que no campo LiDAR estarían directamente relacionada co tamaño e o número de puntos das nubes xestionadas. Logo da creación da comentada xerarquía, establecéronse unha serie de obxectivos globais en cada nivel da mesma. Tódalas propostas e contribucións de cada nivel foron explotadas nos seguintes, axudando a construír así, paso a paso, un sistema completo e multi-funcional para traballar e xestionar datos masivos LiDAR.

A xerarquía que foi establecida consta de tres niveis: grandes nubes de puntos (centos de millóns de puntos), nubes de puntos masivas (miles de millón de puntos) e grandes coleccións de nubes masivas. No primeiro destes niveis, a investigación da tese centrouse na visualización mediante resolución completa e no procesamento de datos no lado cliente, tendo como obxectivos principais a flexibilidade, os fluxos de traballo adecuados, a creación de ferramentas de medición adaptadas para campos profesionais específicos e optimizacións para o maximizado de rendemento (rendemento en termos de imaxes por segundo e carga rápida de datos), algunhas das cales foron presentadas nun traballo fin de carreira anterior[1]. No segundo nivel, a investigación centrouse na visualización multi-resolución e *out-of-core*, tratando de optimizar o uso de recursos computacionais para sistemas de visualización en tempo real baseados en patróns de tipo cliente-servidor. Finalmente, no terceiro nivel, a investigación centrouse en superar as restricións dunha única máquina a través de enfoques de tipo *big data*, ofrecendo solucións en dúas etapas ben diferenciadas; por unha parte, solucións para o almacenamento distribuído, tratando de maximizar a capacidade de almacenamento, latencia e rendemento de sistemas

[1]Link permanente ao TFC: http://kmelot.biblioteca.udc.es/record=b1514673 S1*gag

back-end para dar soporte a aplicacións LiDAR de tipo web. Na segunda etapa, desenvolvéronse solucións para a computación distribuída, maximizando a capacidade de computación, latencia e rendemento de diversos procesos xeo-espaciais de alta complexidade deseñados para facer uso de grandes volumes de datos LiDAR.

Algunhas medidas e procesos xeo-espaciais empregados no campo LiDAR están destinados a operar en sub-rexións específicas dentro de nubes de puntos de extensión moito mais ampla. Adicionalmente, desaparece a necesidade de empregar certas densidades de punto unha vez alcanzado un determinado número de puntos por metro cadrado, xa que non se conseguen melloras adicionais nin tampouco aumenta a calidade dos resultados dos xeo-procesos máis alá da devandita densidade. Un exemplo disto pódese atopar en [51]. En contextos coma este, o número de puntos manexados, aínda que pode chegar a ser moi elevado (centos de millóns de puntos), está lonxe dos miles de millóns manexados noutros contextos de traballo ou casos de uso. Debido a esta diferenza, tomouse a decisión de seguir dous enfoques diferentes para cada un dos contextos. Así, para lograr alcanzar os obxectivos da xerarquía presentada anteriormente, a investigación desta tese evolucionou durante catro etapas diferentes seguindo unha metodoloxía incremental. Durante unha etapa inicial, desenvolvéronse melloras e contribucións para os contextos reducidos (enmarcados fóra da definición de *big data*). Tras esta etapa inicial, afrontáronse problemas de cada vez maior complexidade en contextos cada vez máis esixentes ata chegar a marcos de traballo puramente *big data* durante as últimas etapas da tese.

Na primeira etapa desta tese (Capítulo 2), toda a investigación centrouse na visualización de nubes de puntos de gran tamaño mediante resolución completa e no procesamento no lado cliente. Establecéronse aquí catro obxectivos principais: flexibilidade, un fluxo de traballo adecuado, a creación de ferramentas de medición adaptadas a campos científicos e profesionais concretos, e optimizacións para maximizar o rendemento (rendemento en termos de imaxes por segundo e carga rápida de datos). Todas as investigacións e contribucións foron incluídas e probadas nunha aplicación de visualización especialmente deseñada para tal propósito.

WebGL [78, 106] foi a interface de programación de aplicacións gráficas (API) elixida para traballar, xa que esta permite a produción de potentes solucións de visualización de tipo web. Un enfoque baseado na web permite que as aplicacións non estean vinculadas a ningún sistema operativo (SO) ou dispositivo en particu-

lar, concedendo acceso instantáneo a calquera usuario, desde calquera lugar, con só executar un navegador web compatible con WebGL. A nosa proposta foi concibida coma un enfoque orientado ao servizo; polo tanto, a aplicación e os datos serían almacenados nun servidor remoto e recuperados polos clientes segundo fose necesario, aumentando aínda máis a mobilidade e flexibilidade da solución de visualización.

En determinados contextos de traballo, nubes de puntos de grandes dimensións están destinadas a ser exploradas ou procesadas só mediante sub-rexións específicas das mesmas. O uso da totalidade destas nubes implica, de maneira habitual, un mal aproveitamento de recursos hardware o ter que destinar parte dos mesmos ao procesamento de conxuntos de puntos situados fóra das devanditas sub-rexións. Tendo isto en conta, deseñáronse consultas de datos baseadas en restricións espaciais mediante técnicas de *hashing* espacial xunto coa definición de rexións de interese (ROI) sobre as nubes de puntos.

Mediante JavaScript (JS) [38] e WebGL, incorporáronse ferramentas de medición xeo-espacial para máquinas cliente que permiten realizar os cálculos directamente sobre as imaxes das nubes de puntos, proporcionando así funcionalidades de gran utilidade en comparación coas opcións dispoñibles noutras solucións de visualización, sendo incluídas ferramentas específicas como a medida da superficie de fachadas ou outras ferramentas moito máis complexas como os volumes de base irregular, contorno poligonal e superficie proxectada.

Como se comentou anteriormente, a investigación nesta primeira etapa da tese centrouse na visualización en resolución completa co obxectivo de maximizar a precisión das ferramentas de medida e a fidelidade das imaxes durante inspeccións visuais en tempo real. Nun enfoque coma este, o rendemento convértese nun elemento crítico e prioritario durante o proceso de creación de software. Debido a isto, deseñáronse varias estratexias de optimización intentando maximizar o rendemento do proceso de renderizado así como tamén o proceso de adquisición e carga de datos remotos.

A diferenza da maioría de propostas analizadas, o sistema de visualización web presentado neste capítulo mostra unha alta estabilidade e rendemento, logrando unha interacción en tempo real, manexando cerca de 103 millóns de puntos e ata 281 millóns cun nivel de interacción aceptable. Ademais, as súas ferramentas de medición

xeo-espacial constitúen unha das vantaxes principais respecto ás outras propostas. Toda a investigación desta etapa publicouse en [34] e en [35] e foi rexistrada en [33].

Na segunda etapa da tese (Capítulo 3), exploráronse novas direccións no uso de técnicas multi-resolución e *out-of-core*. Como resultado, deseñouse unha nova estratexia de visualización baseada en dous elementos principais: un método de organización de puntos sen redundancia de información denominado *Hierarchically Layered Tiles* (HLT), e unha estrutura de datos de tipo árbore denominada *Tile Grid Partition Tree* (TGPT). Estas eficientes estruturas de datos deseñáronse co obxectivo de evitar a redundancia de datos habitualmente asociada aos sistemas multi-resolución e á creación e xestión de múltiples niveis de detalle (LOD). Os fundamentos principais detrás destas estruturas foron a reorganización e almacenamento de puntos en celas capeadas, de tal xeito que ningún punto fose repetido entre capas. Cada unha destas capas actúa coma unha peza de crebacabezas, combinándose entre si segundo sexa necesario para calcular e crear os diferentes niveis de detalle en tempo de execución.

Por motivos de análise, as novas estruturas incluíronse nunha nova iteración da aplicación de visualización presentada na primeira etapa da tese, reorientando o seu deseño cara á visualización eficiente en tempo real de nubes de puntos masivas a través dun enfoque multi-resolución e *out-of-core*. O primeiro software de visualización foi capaz de representar de xeito eficiente nubes de resolución completa e sub-rexións das mesmas utilizando entre 50 e 100 millóns de puntos. A iteración presentada nesta etapa non ten como obxectivo substituír a anterior, senón expandir as súas funcionalidades, incluíndo a capacidade de explorar de xeito eficiente conxuntos de datos masivos que conteñen miles de millóns de puntos.

Como resultado de non ter que almacenar ningún nivel de resolución pre-computado nin de ter sobrecargas derivadas da redundancia de datos, os requisitos de almacenamento no lado servidor reducíronse de maneira notable. Nun sistema de tipo cliente-servidor, os altos niveis de tráfico e acceso concorrente poden causar grandes conxestións na rede. Evitar a redundancia de datos ten ademais un impacto beneficioso na cantidade total de bytes que teñen que ser movidos pola rede, xa que cantidades moito menores de puntos son enviadas dende os servidores ata os clientes.

Respecto dos requirimentos de almacenamento no lado cliente, o mesmo princi-

pio que no lado servidor aplícase aquí, xa que a ausencia de redundancia de datos minimiza os datos que son almacenados na caché do navegador. Os datos recuperados dende o servidor almacénanse no disco do cliente para acelerar o uso posterior dos mesmos. A manipulación de grandes volumes de datos pode causar problemas de almacenamento moito peores que os atopados no lado servidor debido ás menores capacidades de almacenamento da maioría das máquinas cliente, especialmente ordenadores portátiles ou dispositivos de man.

Para copiar os puntos na memoria principal (RAM) durante a execución do proceso de representación de puntos no lado cliente foi utilizada unha idéntica organización de datos en capas sen redundancia de información. Os diferentes niveis de detalle son calculados ou descartados en tempo real segundo sexa necesario, evitando a necesidade de almacenar datos innecesarios na memoria RAM. Estes niveis son recalculados cada vez que a cámara na escena 3D detecta un cambio considerable no punto de vista (POV). Este enfoque achega grandes beneficios para os sistemas de gama media e baixa, tabletas ou portátiles con cantidades moderadas de memoria RAM. Todos os niveis de detalle calculados son enviados ata un único búfer de tamaño fixo na GPU, utilizando como base para o seu cálculo un número límite de puntos definido polo usuario (PB). A diferenza doutras técnicas de representación, esta proposta mantén o uso de memoria de vídeo (VRAM) constante ao longo do proceso de representación de puntos.

Tralas investigacións e análises realizados durante esta etapa da tese, demostrouse como mediante técnicas multi-resolución *out-of-core* sen redundancia de información é posible dar soporte de alta eficiencia ao renderizado de nubes de puntos en aplicacións web. A través dunha reorganización e almacenamento de puntos moi específica, lograrouse evitar a creación de elementos estáticos e pre-computados innecesarios, elementos que normalmente son requiridos noutros enfoques multi-resolución. Grazas a esta proposta, en función das características das nubes de puntos e os distintos niveis de detalles creados, é posible chegar a conseguir unha notable redución dos requisitos de almacenamento no servidor e do tráfico da rede. Estas optimizacións poden considerarse logros especialmente significativos en contextos onde grandes cantidades de datos LiDAR están sendo constantemente recollidos, o que supón un importante custo de recursos económicos e técnicos. Como se demostrou, no lado cliente, o consumo de memoria é notablemente baixo, o que permite cargar e

renderizar en tempo real nubes masivas de puntos, de ata 28 mil millóns de puntos, incluso en dispositivos móbiles, onde a capacidade de memoria é moi limitada, así como tamén nos navegadores web con restricións de memoria moi exixentes, como Google Chrome. Tódalas propostas desta etapa da tese foron analizadas e comparadas utilizando unha das solucións de visualización LiDAR máis coñecidas e mellor valoradas, obtendo as nosas propostas resultados superiores en termos de rendemento en comparación coa citada solución. Todas as investigacións deste capítulo publicáronse en [32] e rexistráronse en [28].

Na terceira etapa deste traballo (Capítulo 4), analizouse como as aplicacións que utilizan de maneira intensiva grandes coleccións de datos masivos LiDAR, en particular as aplicacións baseadas na web e centradas na representación en tempo real, poderían beneficiarse da adopción de tecnoloxías de almacenamento *big data*. Adicionalmente, preséntase un estudo sobre as vantaxes e os inconvenientes que poderían determinar a elección da opción máis adecuada entre as dispoñibles actualmente en función dos requisitos e características de cada caso de uso.

Realizáronse varias análises e comparacións empregando catro das tecnoloxías de almacenamento *big data* máis utilizadas e maduras. Mediante estes análises, demostrouse como as tecnoloxías *big data* poden ser empregadas como *back-end* para dar soporte a aplicacións LiDAR de tipo web sen ningún inconveniente ou penalización no rendemento nin na experiencia do usuario, ao tempo que se obtiveron todas as vantaxes habituais asociadas a este tipo de solucións, como a fiabilidade, a dispoñibilidade e a escalabilidade. De cara ao futuro, calquera sistema que adopte este tipo de tecnoloxía de almacenamento xa estaría preparado para incorporar outras tecnoloxías de computación distribuída, como Spark, Flink ou Storm. Todas as investigacións deste capítulo publicáronse en [29].

Na cuarta e última fase da tese (Capítulo 5), mediante un enfoque de tipo *big data*, deseñouse un sistema completo para a computación distribuida de procesos xeo-espaciais de gran complexidade para extensas coleccións de datos LiDAR. O sistema está destinado a apoiar a execución de calquera tipo de proceso xeo-espacial; non obstante, como caso inicial de estudo, a investigación centrouse só na obtención rápida de modelos dixitais do terreo a partir de nubes masivas.

Tralos análises e conclusións presentados no capítulo anterior, a distribución de

datos realizouse mediante Cassandra [95], por outra parte, debido á súa versatilidade, compatibilidade de código fonte e deseño orientado a procesamento *batch*, a distribución de tarefas de cómputo foi realizada mediante Spark [100]. Grazas a este enfoque, foi posible reducir considerablemente o tempo necesario para procesar extensións moi grandes de nubes de puntos de tipo aéreo en comparación con outros enfoques dunha única máquina. Outra contribución importante presentada nesta etapa foi unha estratexia automatizada de corrección de erros de clasificación. Dita estratexia permitiunos corrixir os erros de clasificación comunmente atopados nos límites das zonas adxacentes procesadas de xeito independente polo algoritmo de filtrado, mellorando así a calidade dos modelos dixitais de terreo obtidos posteriormente ao tempo que se minimizou a intervención dos usuarios.

O sistema de computación distribuído presentado neste capítulo permitiunos mellorar enormemente o rendemento do algoritmo de filtrado [22], que foi especialmente deseñado para obter rasters de puntos terreo para a posterior xeración de modelos dixitais do terreo. Ademais da excelente capacidade de computación observada mediante o traballo conxunto de Cassandra e Spark, o seu deseño altamente programable ofrece a oportunidade de mellorar e ampliar aínda máis a funcionalidade dos procesos xeo-espaciais mediante a fácil inclusión de novas etapas computacionais, como a fase de corrección de erros anteriormente descrita. Os resultados de rendemento obtidos amosan como a nosa proposta foi capaz de reducir o tempo utilizado para procesar 28 mil millóns de puntos nunha única máquina, dende 28,57 horas ata soamente 3,41 horas corrixindo os erros das zonas límite, e dende 13,32 horas ata 1,56 sen utilizar a corrección de erros, o que supón unha aceleración de $8,4\times$ para sistemas con 16 nodos (ver Figura 5.9 do Capítulo 5). Os centros GIS, as institucións gobernamentais ou calquera outro grupo encargado de xestionar grandes volumes de datos LiDAR, podería obtener un gran beneficio mediante a proposta presentada neste capítulo e, coa cantidade adecuada de recursos computacionais, todo o proceso de obtención de modelos dixitais de terreo podería aproximarse a tempo real. A investigación presentada neste capítulo foi enviada para a súa publicación a [31].

Durante a realización da tese foi tamén deseñado un método de compresión sen perda de infamación para dar soporte a moitas das principais estruturas, algoritmos e sistemas descritos nela. Alcanzando uns índices de compresión de 0,1 (unha redución do tamaño do ficheiro do 90 %), o método presentado supera a LASzip, o método

de compresión estándar utilizado sobre os ficheiros LAS. O uso dun formato propio permitiunos asegurar o control total sobre os datos debido a que non foi necesario restrinxirse ás características e limitacións dun formato preexistente, e polo tanto, os ficheiros de datos empregados puidéronse adaptarse libremente para maximizar o rendemento dos elementos que os utilizan.

Todas as contribucións presentadas nesta tese serán integradas no mesmo sistema, BETi (**B**ig data for the **E**xploration of **T**errain **I**nfomation) ofrecendo unha completa solución multi-funcional para a distribución de datos, replicación, dispoñibilidade e procesamento tanto de tipo cliente como servidor. O sistema posuirá a capacidade de ofrecer acceso concorrente a unha colección moi ampla de nubes de puntos masivos, incluída unha extensa librería de procesos xeo-espaciais impulsada por un sistema de computación distribuído altamente escalable baixo un sistema de colas. A visualización de conxuntos de datos públicos e privados concederase a través de clientes baseados na web xunto con todas as ferramentas de medición xeo-espaciais anteriormente descritas. Por último, debemos resaltar aquí que parte da investigación presentada durante este traballo está a ser empregada noutra tese desenvolvida no grupo de investigación *Computer Graphics Systems* do *Hasso Plattner Institute* e no *Grupo de Arquitectura de Computadores* da *Universidade da Coruña*. Moitos dos conceptos presentados no Capítulo 3 relativos a estruturas de datos eficientes e adaptativas xeradas en tempo de execución están a empregarse para mellorar a calidade de resultados e o rendemento dos algoritmos de enriquecemento semántico para as nubes de puntos interiores.