SPSLiDAR: towards a multi-purpose repository for large scale LiDAR datasets

Article *in* International Journal of Geographical Information Science · March 2022

DOI: 10.1080/13658816.2022.2030479

5 authors, including:

Antonio J. Rueda
Universidad de Jaén
50 PUBLICATIONS **464** CITATIONS
SEE PROFILE

Rafael Jesús Segura
Universidad de Jaén
68 PUBLICATIONS **768** CITATIONS
SEE PROFILE

Jorge Delgado
Universidad de Jaén
104 PUBLICATIONS **1,383** CITATIONS
SEE PROFILE

SPSLiDAR: Towards a multi-purpose repository for large scale LiDAR datasets

Antonio J. Rueda[a], Carlos J. Ogáyar-Anguita[a], Rafael J. Segura-Sánchez[a], Juan A. Béjar-Martos[a], Jorge Delgado-Garcia[a]

[a]CEATIC, University of Jaén, 23071, Spain.

ABSTRACT

The widespread use of LiDAR technology in a multitude of domains has produced a growing availability of massive high-resolution point datasets that demand new approaches for efficient organization and storage, filtering using different spatio-temporal criteria, selective/progressive visualization, processing and analysis, and collaborative editing. Ideally, LiDAR data coming from multiple sources and organized in different datasets should be accessible in a simple, uniform and ubiquitous way to comply with the FAIR principle proposed by the Open Geospatial Consortium: Findable, Accesible, Interoperable and Reusable. With this goal in mind we present SPSLiDAR, a conceptual model with a simple interface for repositories of LiDAR data that can be adapted to the needs of different applications. SPSLiDAR includes aspects such as the arrangement of related datasets into workspaces on a world scale, support for overlapping datasets with different resolutions or acquired at different times, and hierarchical organization of point data, enabling levels of detail and selective download. We also describe in detail an implementation of this model aimed at visualization and downloading of large datasets using the MongoDB database. Finally we show some experimental results of this implementation using real data, such as its space requirements, upload latency, access latency and throughput.

KEYWORDS

LiDAR; Big Data; point cloud; spatial data structure; NoSQL

1. Introduction

LiDAR scanning has become an indisputable tool in fields such as civil engineering, surveying, archaeology, forestry or environmental engineering. The widespread use of terrestrial and airborne LiDAR, powered by the fast evolution of the scanning technology, is generating an unprecedented amount of data. For instance, scanning speeds of one million points per second with an accuracy in the range 3-5 mm have become common today in terrestrial scanning. Handling such a massive amount of data poses multiple challenges related to its storage, transmission, organization, visualization, edition and analysis. These challenges increase in magnitude when multiple datasets, spread throughout the territory and acquired at different times, have to be managed. Comparing several datasets of the same site, taken at different moments, is of great

This is an Accepted Manuscript of an article published by Taylor & Francis in International Journal of Geographical Information Science on March 2022, available at: http://dx.doi.org/10.1080/13658816.2022.2030479

CONTACT Antonio J. Rueda-Ruiz. Email: ajrueda@ujaen.es

interest during an archaeological excavation for assessing its progress and planning future campaigns. In architecture, engineering and construction, it enables quality assurance, cross-checking with the plan and early detection of errors. For the geologist, it is a way to monitor changes in the terrain that can help to prevent natural disasters or evaluate the effects of global warming in glaciers or polar ice. Combined access to multiple datasets located in a given area is also useful for comparing different models, detecting common features or performing global analyses of data.

In general, most users of LiDAR do not have a way to organize their datasets in an efficient way, and over time they end up with terabytes of information scattered across diverse primary and secondary storage media in hundreds of files (sometimes in multiple formats). This makes it difficult to perform spatial and temporal analyses such as those mentioned above. Even if a strict file naming convention and directory structure is established, combining multiple datasets according to several spatial-temporal criteria can be a complex task that has to be done manually. Ultimately, the problems arising from the access and processing of multiple LiDAR models are common to most big data applications, and therefore, similar approaches and techniques are applied (Evans *et al.* 2014b).

With the aim of making progress in the integration of LiDAR datasets and to overcome these shortcomings, this work presents SPSLiDAR, a general specification for point cloud data repositories, the main characteristics of which are summarized below:

- Support for an arbitrary number of datasets stored in a single server or distributed in a cluster.
- Straightforward uniform access through a RESTful API that allows querying datasets by spatial or temporal criteria, and retrieval of their associated metadata or data.
- Two-level spatial indexing for organizing datasets at the territory level and point data inside each dataset, enabling efficient queries and retrieval of information.
- Progressive transmission of point data through the network with support of different levels of detail (LODs).
- Uniform and efficient storage and transmission of scanned data through the use of industry-standards such as the LAS/LAZ format (Isenburg 2013).

The Open Geospatial Consortium (OGC) Testbed-14, in the Point Cloud Data Handling Engineering Report (Boehler *et al.* 2018), identifies similar requirements for web services of point cloud data: services should be flexible, with configurable data sources, spatial partitioning methods for accelerated queries, and attribute filtering. Services should also support alternative representations for point cloud data, such as rasterizations or levels of detail, and include a compelling compression storage supported across multiple computing platforms.

The SPSLiDAR API provides a convenient method for uploading or browsing LiDAR data from/to desktop, web or mobile clients. It also enables straightforward and uniform access for data analysis and processing, or training and running machine learning algorithms for segmentation or feature extraction. In this regard, deep learning methods have recently shown remarkable results with point clouds (Wang *et al.* 2019, Zhao *et al.* 2019, Boulch 2020, Xie *et al.* 2020), and they are expected to play a greater role in the coming years (Poux and Billen 2019a). Many government agencies and institutions provide free access to nationwide airborne LiDAR data through specific web portals, such as the USGS 3DEP (US Geological Survey 2021), the NOAA Digital Coast (National Oceanic and Atmospheric Administration 2021) or PNOA-LiDAR in

Spain (Instituto Geográfico Nacional 2021), to cite a few. However, these portals basically operate as download centers, and therefore do not provide advanced features such as interactive visualization of the datasets, partial/progressive download or an API to enable applications to query, access, analyze or process data.The target users for a SPSLiDAR repository are public institutions aiming at making nationwide LiDAR data publicly available, open organizations supporting public collaborative repositories, or private companies or institutions with the need for a centralized repository in a private cloud for their LIDAR data acquired in different projects or campaigns.

The remainder of this paper is organized as follows. Section 2 reviews the most relevant work related to point cloud models: applications, processing, data structures for its organization and out-of-core storage in files and databases. Section 3 describes the conceptual model of SPSLiDAR and our proposal for a general RESTful API for accessing LiDAR data. Section 4 describes in detail an implementation of the SPSLIDAR specification focusing on visualization of large LiDAR datasets, and shows some experimental results obtained after testing this implementation with real data. Finally, Section 5 presents the conclusions and outlines future work.

2. Previous work

Point cloud models are a widely used resource in all types of decision-making processes. The rapid evolution of 3D data capture hardware, including low-cost solutions, poses a challenge for the management of the huge volumes of information generated (Poux, Florent 2019). Current sensors allow, using various technologies, to obtain dense information at different scales, from small objects to large areas of the territory (Bräunl 2020). This enables the acquisition of massive digital information from the real environment, which is integrated into the so-called Geospatial Big Data (Evans *et al.* 2014b, Lee and Kang 2015, Pääkkönen and Pakkala 2015, Deng *et al.* 2019), where the five main Vs of big data are met (Evans *et al.* 2014b). The data to be processed are of a massive scale (Volume); sources of different nature are used, mainly depending on the capture technology used (Variety); today, data can be captured and stored almost in real time (Velocity), although there are later some difficulties in processing this data with the same agility; the information obtained is usually authentic (Veracity), although its traceability can be complicated; finally, this information is used for all kinds of analysis and decision support (Value).

Point cloud processing involves a series of steps from its capture to the extraction of the desired information, including registering, filtering, segmentation, classification and conversion to other 2D/3D representation schemes (Józsa *et al.* 2013, Ullrich and Pfennigbauer 2019). Each of these processes has its own research topic, and notable advances have been made in all of them in the last decade. The objective of this work is the definition of a framework for the storage of point clouds within Big Data that serves as the basis of a comprehensive management process, which would include the tasks mentioned above. This is because one of the main bottlenecks in the processing of point cloud datasets is access to information in a delocalized and efficient way (Poux, Florent 2019) and at the required level of detail. This information should be organized taking into account time information (4D), and should refer to different types of information, associated with attributes of each point (Mallet and Bretar 2009). This should allow the use of spatial data from different types of applications and with different user profiles. The processing of data by different users is often a problem, both due to the lack of automation of most basic operations, and the synchronization and collaboration

within workgroups. This is critical when the volume of data is as high as it is today.

It is clear that the data structures used, as well as the information storage and access mechanisms determine the design of the algorithms and the interaction methods that are built on top. The data structures used are strongly coupled to their specific use, especially in rendering-oriented systems (Goswami *et al.* 2013, Richter *et al.* 2015, Schütz 2016, Ströter *et al.* 2020). In recent years, with faster GPUs, more memory and better communication bandwidth, the main challenge is to have massive information available for analysis from various applications (Poux and Billen 2019a). In this sense, there are several papers that focus on mass storage in secondary memory, in the network or in the cloud (Scheiblauer and Wimmer 2011, Richter *et al.* 2015, Deibe *et al.* 2019, Schütz *et al.* 2020). All these approaches can always be used for efficient rendering by optimizing the transfer of remote data to the GPU (Kulawiak and Kulawiak 2017, Discher *et al.* 2019, Kang *et al.* 2019, Schutz and Wimmer 2019).

The most widely used data structure for point cloud storage is the octree, especially with out-of-core approaches (Scheiblauer and Wimmer 2011, Elseberg *et al.* 2013, Schütz *et al.* 2020, Huang *et al.* 2020). Alternative approaches are based on data structures such as the k-d tree (Goswami *et al.* 2013), variants of the R-tree (Gong *et al.* 2012, Wang *et al.* 2020), a sparse voxel structure (Kämpe *et al.* 2013, Baert *et al.* 2014, Poux and Billen 2019b) or a simple grid (Hongchao and Wang 2011). Some of these proposals include the use of several nested or hybrid spatial structures, mainly oriented to rendering with LODs (Yang and Huang 2014, Schütz 2016, Deibe *et al.* 2019) and to the optimization of search operations (Scheiblauer and Wimmer 2011, Lu *et al.* 2019). In most of those papers, one of the priorities is to maintain a spatial data structure as simple as possible that allows working with LODs (Schütz 2016, Deibe *et al.* 2019). This has the disadvantage that spatial indexing does not adapt well to all possible point spatial distributions. One of the most prominent cases is the poor adaptation of the octree to the spatial distribution of points along a large surface, as occurs with scans of large areas of territory. In these cases quadtree and grid planar structures are usually more appropriate (Hongchao and Wang 2011, Boehm *et al.* 2016, Deibe *et al.* 2019). However, in many tasks oriented both to storage and processing within Geospatial Big Data, octree continues to be a common option (Pajić *et al.* 2018).

Regardless of the type of spatial data structure used to index point clouds, there are several strategies for remote and out-of-core storage of data (Deng *et al.* 2019). The simplest approach is direct storage in secondary memory using local files (Scheiblauer and Wimmer 2011, Schütz *et al.* 2020). Another more evolved variant consists of using storage in distributed file systems (Krämer and Senner 2015, Deibe *et al.* 2018), typically indexed through a spatial structure hosted in the form of a master index or in a database. Point cloud files (especially LiDAR) are the most common form of data storage and exchange. The most common options are non-standard ASCII formats, the LAS format of the American Society for Photogrammetry and Remote Sensing (ASPRS), its compressed variant LAZ (Isenburg 2013), SPD (Bunting *et al.* 2013), PCD, HDF5 and other general 3D data file formats such as OBJ or PLY. In addition, there are also other proposals for specific formats closely linked to specific applications, which usually include additional attributes for points and custom compression algorithms. The most desirable formats are those that allow the use of data compression (Isenburg 2013, Sugimoto *et al.* 2017, Cao *et al.* 2019). In the present paper the LAZ format is used, since it allows a very efficient compression of LiDAR data, and it can be considered as a widespread standard (Boehm 2014, Ladra *et al.* 2019).

In addition to file-oriented storage, the usual option is the use of databases. The

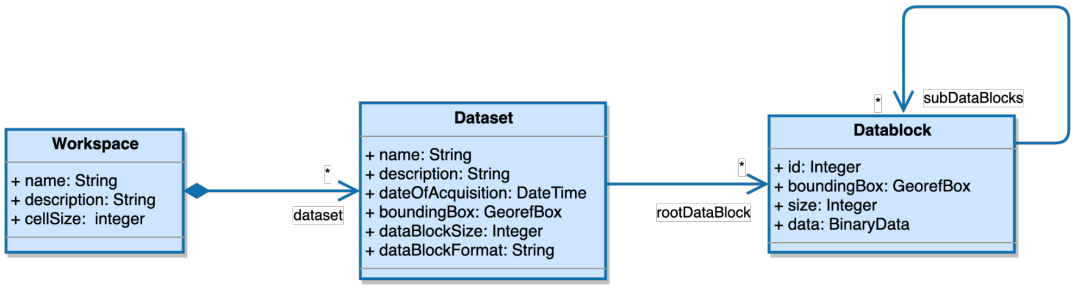

Figure 1. UML representation of the conceptual model of SPSLiDAR.

simplest way to store point clouds is to use a row for each point in a relational database, including all its attributes. However, this scheme is not viable for a large amount of data (Boehm 2014), therefore an alternative solution is to store the point clouds in groups in each row, as systems such as Oracle Spatial or PostGIS do. Despite the above, there are also proposals and experiments based on relational databases, with and without extensions for working with point clouds and spatial information (Schön *et al.* 2013, Lee and Kang 2015).

NoSQL databases solve some of the limitations of relational ones in big data applications. Among these, document-oriented databases (MongoDB, Cassandra, Couchbase, etc.) are the most common, and can handle large volumes of data in an efficient and scalable way through the use of sharding. Not only NoSQL, but also other types of distributed databases are the preferred option for storing point clouds with a Big Data approach (Hongchao and Wang 2011, Boehm 2014, Deibe *et al.* 2018). In the present work, LAZ files are stored in a NoSQL database, which allows working with large volumes of compressed data using a standard format and in a scalable way. This approach is more versatile than using distributed storage like HDFS.

Regarding the transmission of data over the network, and related to the above, we must mention proposals aimed at streaming data mainly for progressive rendering (Martinez-Rubi *et al.* 2015, Schütz 2016, Deibe *et al.* 2018, Discher *et al.* 2019). Some of the most popular technologies for web-based rendering can be found in Evans *et al.* (2014a). In addition to this, there are several interesting proposals that use web interfaces and protocols to develop web clients for data access, including RESTful web services such as the one proposed in this paper, although they are applied to polygonal 3D objects instead of point clouds (Doboš and Steed 2012, Doboš *et al.* 2013, Houston *et al.* 2013, Desprat *et al.* 2015).

3. SPSLiDAR specification

This section describes a specification of a repository for collections (or workspaces) of LiDAR datasets supporting the basic operations of upload, query and retrieval. Our aim has been to define a simple model yet powerful enough to handle an arbitrary number of related datasets using a flexible organization for point data.

3.1. *Conceptual model*

The conceptual model of SPSLiDAR, depicted in Figure 1, comprises three entities: *workspace*, *dataset* and *datablock*. A workspace represents a set of related datasets,

such as, for instance, those generated by the LiDAR coverage of several counties or provinces, or different scanning campaigns in related archaeological sites. It is identified by a unique name and also contains a detailed description. The attribute *cellSize* is a hint for the spatial data structure (grid, quadtree, etc.) used to index the datasets at the territory level. A workspace comprising datasets at a local level can use small cell sizes (1-10 km) whereas another with datasets distributed in a wide area and/or with a large scale (e.g., acquired from airborne LiDAR) are best suited for larger cell sizes (10-100 km). Although convenient, the use of a spatial index for the datasets is optional in our model, therefore this parameter can be omitted.

A dataset comprises one or more point clouds acquired at a particular site. It is identified by a unique name and contains a detailed description, a date of acquisition and a bounding box defined by two georeferenced coordinates. The points of a dataset are organized into a structure of datablocks starting at one or more root datablocks. The size and internal format of each datablock can be specified (ASCII or binary raw, LAS, LAZ, etc.). The structure of datablocks can be organized in many ways and has a double purpose: to implement a progressive network-friendly transmission of data from the server to the client applications, and eventually implement LODs of the dataset, that can be useful for visualization. Organizing LiDAR data using tiles or a hierarchical data structure is a common solution for dealing with large point dataset both in main memory and offline (Baert *et al.* 2014, Deibe *et al.* 2019, Ladra *et al.* 2019, Schütz *et al.* 2020).

A datablock contains a unique identifier and the actual size of the datablock (which may vary from the preferred size given in the dataset). A datablock may also contain the identifiers of several child datablocks (or subdatablocks). A datablock with subdatablocks represents a level of detail of the model, and therefore contains a subset of representative points in the portion of space defined by its bounding box. Notice that this general model does not impose a particular number of children per datablock or the number of levels in the internal hierarchical structure. Therefore, a regular grid, k-d tree, quadtree, octree, or any nested combination of them can be used to organize data in a dataset. For instance, a regular grid can be implemented by including in each root datablock a single level of subdatablocks defining a spatial partition of its bounding box. In the case of an octree, each root datablock contains 8 subdatablocks representing their corresponding octree regions. These subdatablocks are recursively organized in the same manner until no more spatial subdivisions are necessary. When a hierarchical structure such as the latter is used, both redundant or non-redundant strategies can be used. A redundant strategy allows a subset of points in a datablock to be replicated in its ancestors, which can streamline certain operations.

Table 1. Summary of the SPSLiDAR RESTful API (part I). For the sake of simplicity services related to the update and removal of resources have been omitted.

#	Method	URL	Description

1	GET	**/workspaces**	Lists the registered workspaces. Sample response:
			`[`
			`  "/workspaces/CHJaen"`
			`]`

2	POST	**/workspaces**	Registers a new workspace. Sample request body:
			`{`
			`  "name": "CHJaen",`
			`  "description": "Cultural heritage sites in the province of Jaén (Spain)",`
			`  "cellSize": 10000`
			`}`

3	GET	**/workspaces**/{workspace_name}	Returns the given workspace metadata. Sample response similar to the request body shown above.

4	GET	**/workspaces**/{workspace_name}/**datasets**?
		southWest={sw_coord}&
		northEast={ne_coord}&
		fromDate={from_date}&
		toDate={to_date}
			Returns the datasets of the workspace in a given window. An optional temporal interval can be provided. Sample response:
			`[`
			`  "/workspaces/CHJaen/datasets/CtSCat"`
			`]`

5	POST	**/workspaces**/{workspace_name}/**datasets**	Registers a new dataset. Data must be uploaded next through service 9. Sample request body:
			`{`
			`  "name": "CtSCat",`
			`  "description": "Castillo de Santa Catalina (Jaén)",`
			`  "dateOfAcquisition": "2019-05-03T12: 10: 00Z",`
			`  "boundingBox": {`
			`  "southWestBottom": {`
			`  "easting": 429522, "northing": 4180270,`
			`  "height": 790, "zone": "30S"`
			`  },`
			`  "northEastTop": {`
			`  "easting": 429638, "northing": 4180363,`
			`  "height": 820, "zone": "30S"`
			`  }`
			`  },`
			`  "dataBlockSize": 10000,`
			`  "dataBlockFormat": "LAZ"`
			`}`

Table 2. Summary of the SPSLiDAR RESTful API (part II).

#	Method	URL	Description
6	GET	**/workspaces**/{workspace_name}/ **datasets**/{dataset_name}	Returns the dataset metadata. Sample response similar to the request body shown above, but including the list of root datablocks (e.g., "rootDataBlocks": [0]).

7	GET	**/workspaces**/{workspace_name}**/datasets**/ {dataset_name}**/datablocks**/{datablock_id}	Returns the datablock metadata. Sample response:
			{
			"id": 0,
			"bbox": {
			"southWestBottom": {
			"easting": 429522, "northing": 4180270,
			"height": 790, "zone": "30S"
			},
			"northEastTop": {
			"easting": 429638, "northing": 4180363,
			"height": 820, "zone": "30S"}
			},
			"size": 10000,
			"subDataBlocks": [1, 2, 3, 4, 5, 6, 7, 8]
			}

8	GET	**/workspaces**/{workspace_name}/**datasets**/ {dataset_name}/**datablocks**/ {datablock_id}/**data**	Returns the binary data associated with the datablock.
9	PUT	**/workspaces**/{workspace_name}/**datasets**/ {dataset_name}/**data**	Uploads the point data of a dataset. This data is organized into a hierarchical structure of datablocks.
10	GET	**/workspaces**/{workspace_name}/**datasets** /{dataset_name}/**data**	Returns the complete dataset.

3.2. *RESTful API*

Based upon the previous conceptual model, we defined a simple RESTful API to enable access to the datasets from any client application (desktop, web or mobile). Our aim was to define a set of services as readable and intention-revealing as possible. The reasons for choosing a RESTful API are its relative simplicity, its popularity today (which makes it familiar to any developer), the ease of access from virtually every platform using any programming language through a simple HTTP client, and the good adaptation to a CRUD (Create-Retrieve-Update-Delete) scheme, that fits the needs of SPSLiDAR.

Not surprisingly, the API, summarized in Tables 1 and 2, is built upon three main resources: workspaces, datasets and datablocks. Services 1 to 3 allow creating and querying workspaces. Service 4 allows querying a workspace for datasets in a window defined by two georeferenced coordinates. A temporal interval can also be given to narrow the search. Service 5 is also important since it allows registering new dataset entries, although the data upload requires an extra request through service 9. This data, usually one or several LAS/LAZ files, is structured by the server into datablocks that can be retrieved using services 7 and 8. These services can also be used for dataset in-depth browsing, as they can also retrieve subdatablocks identifiers. Finally, service 10 enables a bulk download of the dataset for client applications that require it instead of a progressive access through datablocks.

Notice that requests to services 8-10 can take a long time to complete. Therefore it is advisable to implement them in an asynchronous manner to avoid blocking the server and allow processing data in the client side without waiting for the request to be completed.

4. Example of SPSLiDAR implementation for data visualization

The previous model and RESTful API can be implemented in many ways, depending on the characteristics of the LiDAR data and the application requirements. In this section we describe a reference implementation focusing mainly on visualization and data downloading.

We chose the Java programming language since it is a mature language suitable for developing large and complex systems, and the most widely used language for the server side. Among the many existing Java frameworks for implementing a back-end, Spring (The Spring Team and VMware 2021) is the most popular. Spring has many useful features such as support for dependency injection, configuration files, database access through object/relational mapping, flexible URL-to-service mapping, and implementation of security policies. Spring Boot allows a simple setup of a Spring application including an embedded Apache Tomcat web server for exposing the SPSLiDAR RESTful API. A recent interesting addition to Spring is the WebFlux API, that we used for implementing a reactive non-blocking RESTful API. Compared to a traditional blocking implementation, it has a better performance, particularly with many concurrent requests. On the client side it has the advantage of allowing processing or visualization of the points in a datablock without having to wait for its complete reception.

Storage is a critical part of the system, because of the huge volume of data that has to be managed. We chose MongoDB (MongoDB, Inc 2021) for three main reasons:

- Its support for horizontal scaling through sharding. This allows distributing data

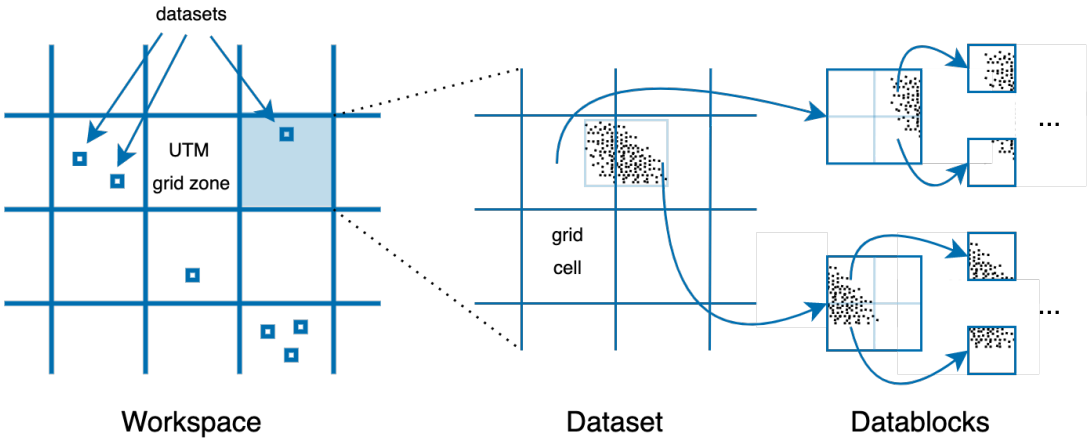

Figure 2. Spatial data structures at the workspace and dataset levels. For the sake of simplicity a dataset is represented using quadtrees instead of octrees.

across several servers, providing faster access and increased storage capability up to petabyte scale.

- Support for storage of large files in the database itself through the GridFS specification. Storing files in a system-level filesystem is the preferred option for most systems working with massive LiDAR datasets Deibe *et al.* (2018), due mainly to performance issues. However, we preferred storing files in the database, since it has a simpler implementation (particularly in a distributed system), avoids inconsistencies between metadata and files, and supports transactions during updates.
- Its superior performance compared to relational databases in key-value accesses or geospatial queries (Agarwal 2017, Pandey 2020). Among NoSQL databases, MongoDB only performs slightly worse than Cassandra Deibe *et al.* (2018) although the latter has currently a worse integration with Spring WebFlux and does not directly support storing large files in the database (a custom user-defined schema similar to GridFS is required for this purpose).

4.1. *Data spatial organization*

Currently, our implementation only supports UTM coordinates for georeferencing datasets and point data. The spatial data structures used in our implementation are depicted in Figure 2. Datasets in a workspace are stored using a two-level regular grid. Each cell of the first level corresponds to a UTM grid region, whereas each cell of the second level within a UTM grid region has the predefined cell size of the workspace (Section 3.1). This grid enables efficient window and neighbor dataset queries in the workspace. Notice, however, that the benefits of this spatial index would only be evident with a significant number of datasets.

As explained in the Section 3.1, our conceptual model can be implemented using different spatial data structures, depending on the type of LiDAR data and the needs of the application. For instance, large aerial LiDAR datasets, of a 2.5D nature, fit well in a regular grid or quadtree Gao *et al.* (2014). In our implementation we chose an octree since it is a very general data structure, suitable for both ground-based and aerial LiDAR data, and particularly appropriate for data visualization or selective

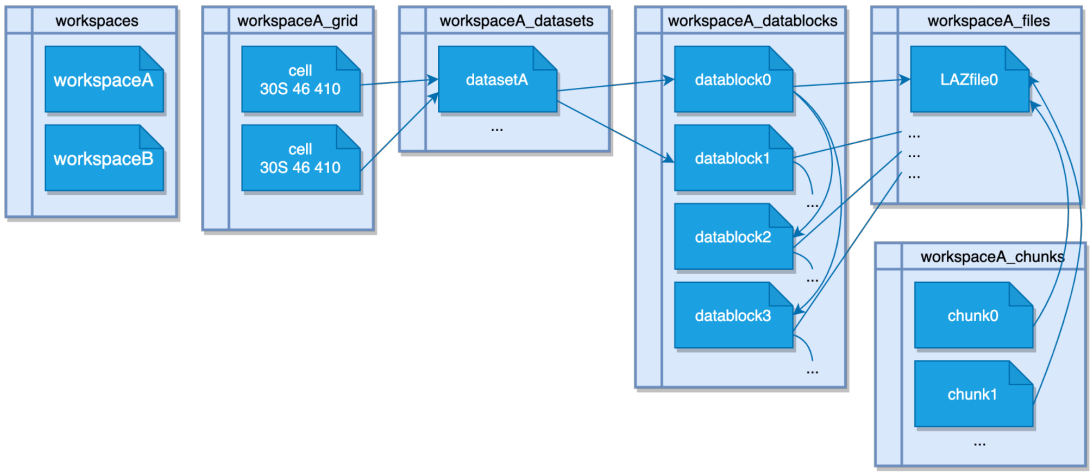

Figure 3. Structure of the MongoDB database in SPSLiDAR. For the sake of simplicity datablocks are shown connected to two subdatablocks instead of eight.

download.

An octree is usually constructed by starting with a subdivision into octants of an axis-aligned bounding box of the points in the dataset, but in SPSLiDAR we chose to start from the grid cell containing the dataset. This has the advantage of ensuring the same octants for the octrees of all the datasets in the same grid cell, simplifying the visualization and comparison of multiple overlapping datasets on the client side. However, a dataset can span two or more grid cells, which implies that it may need more than one octree to organize its point data. As a result, a dataset can have more than one root datablock. As described in Section 3.1, each datablock stores a sampling of the dataset at the corresponding octant with the predefined size set for the dataset. A datablock has no subdatablocks (i.e., corresponds to a leaf node of the octree) if the sampling contains all the remaining points of the dataset at the corresponding octant without exceeding the predefined size. In order to support an optimized LOD-based rendering at the client side we could use a redundant model, replicating all the points in a non-leaf node in its descendants. Another advantage of this approach is that it simplifies the total or partial retrieval of the dataset, since the complete point data can be found at the leaf nodes. However, we have chosen a non-redundant model to prioritize the space requirements of the database.

4.2. *Database structure*

Each entity depicted in Figure 1 is represented in MongoDB as a document with the specified attributes, and stored in a collection (similar to a table in the relational model). The collection named workspaces stores each workspace defined in the system. For each new workspace *ws* defined, five collections are created: *ws_grid*, *ws_datasets*, *ws_datablocks*, *ws_files* and *ws_chunks*, as shown in Figure 3.

The collection *ws_grid* contains all the documents corresponding to non-empty cells of the grid, that is, those with at least one dataset. Each cell document is identified by their UTM zone, row and column. An index is created for each of these three fields in the collection to accelerate spatial queries. They are also used to generate a hash that is used as the *oID* (object identifier) of the document. This enables fast access to a particular cell without extra cost. A cell document also includes a list of the *oIDs*

of the datasets totally or partially within the cell. These datasets are stored in the collection *ws_datasets*. Each dataset document contains a list of the *oIDs* of one or more root datablocks in the ws_datablocks collection. Each datablock comprises an *oID* to the point data, stored as a file in LAZ format in the *ws_files* and *ws_chunks* collections, that follows the GridFS specification of MongoDB for the storage of files of arbitrary size. We use LAZ files because it is the most efficient format for LiDAR point data storage and transmission over the network. Finally, each datablock is completed with a list of the oIDs of its subdatablocks (i.e. its children in the implicit octree). Figure 3 illustrates the content of each of the collections of the database.

4.3. *Data upload and bulk download*

The most expensive operations of the SPSLiDAR API are the dataset data upload and bulk download (operations 9 and 10 in Table 2). Data upload is only executed once per dataset, and bulk downloads are uncommon, since the preferred access is through datablocks (operations 7 and 8). Data upload comprises several steps: transfer of one or more point data files in LAS, LAZ or ASCII formats to the server, conversion to UTM if necessary, construction of the octree, and insertion of a datablock per octree node in the database. During the construction of the octree, the creation of a node implies the extraction of the subset of points of the dataset in a given rectangular region of space and its sampling to meet a maximum number of points per datablock. In order to facilitate this, we implemented a simple Java wrapper that automates the subdivision of a node stored in a LAZ file into eight LAZ files representing each of its children by chaining the execution of several of the tools in the *LAStools* suite (Hug *et al.* 2004).

Bulk download of the entire dataset is carried out by traversing the datablocks of the dataset and merging the corresponding LAZ files into a single one through the LAStools, that is finally served to the client.

4.4. *Experimental assessment*

We designed several experiments to evaluate the performance of the implementation described in the previous section. In particular, we wanted to study the impact of the datablock size in several of the operations of the repository. It is important to note that the results presented here are only indicative of the performance of a production-grade repository based on the SPSLiDAR conceptual model. Our implementation was primarily intended to demonstrate the flexibility and possibilities of this model, and therefore is not fully optimized. For the same reason we deployed the repository in a single server instead of a cluster of MongoDB instances. This server has an Intel i7-10700 Octa Core at 2.9Ghz, 16GB RAM, 1TB SSD drive and a 8TB SATA3 HDD, running Windows 10 Enterprise 64. It was equipped with OpenJDK 15 and MongoDB 4.2.10. The full size of the database used for the experiment is about 1TB, including multiple datasets downloaded from the repositories of PNOA-LiDAR (Spain) and SITNA (Regional Government of Navarre, Spain). The client computer was a Macbook Pro laptop with an Intel Core i7-4770HQ Quad Core at 2.2GHz with 16GB RAM and a 256TB SSD drive, running macOS Mojave. The client was connected to the server through a 1Gb/s Ethernet cable network.

Table 3. Datasets description.

	Dataset 1	**Dataset 2**	**Dataset 3**	**Dataset 4**
Name	Pamplona2011	Pamplona2017b (reduced version of dataset 4)	Pamplona2017c (reduced version of dataset 4)	Pamplona2017
Number of points	30 401 539	244 894 563	534 073 172	1 068 146 345
Density (points/m^2)	0.5	2.5	5	10
Grid size (meters)	10 000	10 000	10 000	10 000
Size (MBs)	138.29	1 850.21	4 085.94	8 054.36
Point data record length	34	38	38	38
LAS point data record format	3	8	8	8

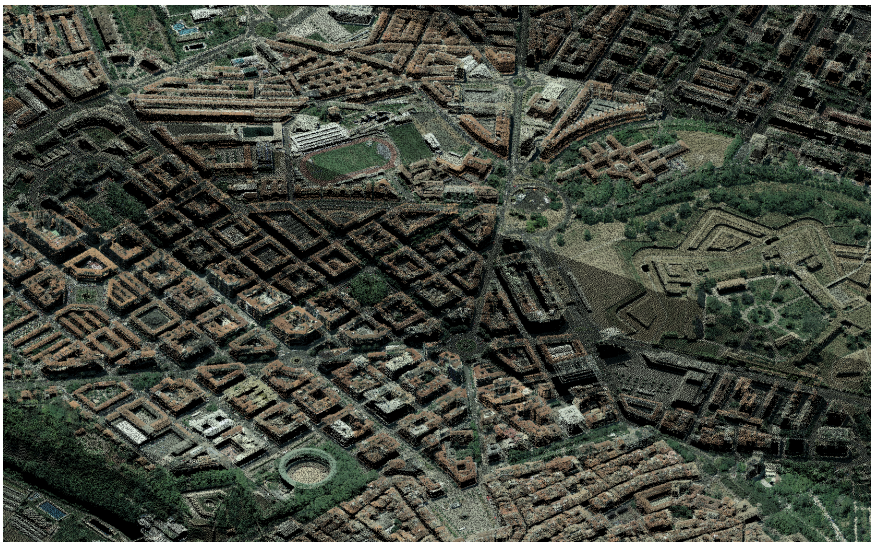

Figure 4. Partial view of Dataset 1 (Pamplona2011).

In the experiments we used four public datasets of the city of Pamplona (Spain) (Figure 4), acquired at different campaigns (2011 and 2017) and with increasing resolution and size, from 30M points to 1068M points (Table 3). The point data record format and number of attributes are also different for dataset 1 and datasets 2-4. These datasets are publicly available from SITNA at the following URL: `https://filescartografia.navarra.es/`.

Table 4. Upload time and space required by the test datasets in SPSLiDAR.

		1st Dataset	
Datablock size (n. of points)	**10 000**	**100 000**	**1 000 000**
Upload time (s)	1 368.55	213.72	78.19
Storage size (MBs)	235.65	201.76	199.04
Number of datablocks generated	9 183	941	94
Maximum octree depth	7	5	3
		2nd Dataset	
Upload time (s)	13 193.64	2 183.32	923.87
Storage size (MBs)	2 148.66	1 894.24	1 878.46
Number of datablocks generated	86 840	8 070	899
Maximum octree depth	8	6	5
		3rd Dataset	
Upload time (s)	22 544.29	3 685.26	1 817.54
Storage size (MBs)	4 959.6	4 312.33	4 238.78
Number of datablocks generated	171 184	17 489	1872
Maximum octree depth	8	6	4
		4th Dataset	
Upload time (s)	41 271.37	7 814.88	3 993.42
Storage size (MBs)	9 615.16	8 391.81	8 273.69
Number of datablocks generated	360 506	35 081	3 478
Maximum octree depth	8	7	5

Dataset upload The first experiment evaluates the time and space required to upload and store a dataset in the server, including its decomposition into datablocks, organization in the spatial data structure and storage in MongoDB. The upload times, shown in Table 4, although significant, are comparable to those of existing proposals for building and rendering huge point datasets (Martinez-Rubi *et al.* 2015, Schütz 2016, Schütz *et al.* 2020). These times decrease as the datablock size increases, since this implies a smaller number of subdivisions of the original LAZ files and database insertions. Anyway, the relevance of these times is relative, since the insertion operation is only performed once per dataset.

Table 4 also shows the space required for the storage of the four datasets in MongoDB. Again, this space decreases as the datablock size increases for two main reasons: first, many LAZ files are less space-efficient than a single one with the same point data because of the associated headers and metadata, and second, the required hierarchical data structure is also simpler, i.e., with a smaller number of levels. For small datasets (dataset 1) the additional space required is substantial, but for medium to large datasets (datasets 2, 3 and 4) the extra storage space is reasonable: between 2% and 5% of the original dataset size using 100K points datablocks.

Table 5. Time required (in seconds) to respond to a request of 1M points of each dataset using different datablock sizes.

Datablock size (n. of points)	**10 000**	**100 000**	**1 000 000**
Dataset 1	1.19	0.20	0.07
Dataset 2	1.44	0.19	0.09
Dataset 3	1.32	0.21	0.09
Dataset 4	1.65	0.20	0.08

Data download: single requests The second experiment aims at evaluating the performance of the system in transferring raw data to a client. For this purpose, we measured the time required to respond to a request of 1M points. The client script was implemented in Python 3.9.0 using the *Requests* HTTP library (Reitz 2021). Table 5 summarizes the results for the four test datasets using different datablock sizes. Times naturally decrease with an increasing datablock size, since the server has to gather a smaller number of them and the associated GridFS files. Overall, a datablock size of 100K points represents a good compromise between an effective spatial subdivision and a good performance when transferring large amounts of data.

Table 6. Workload test of the SPSLiDAR implementation against multiple concurrent requests.

Datablock size (n. of points)	**10 000**	**100 000**	**1 000 000**
	10 concurrent clients x 1M points each		
Number of datablock requests	1 000 (100 per client)	100 (10 per client)	10 (1 per client)
Average datablock request time (s)	0.027	0.206	1.603
Maximum datablock request time (s)	0.058	0.378	1.628
Throughput (Millions of points per second)	3.59	4.9	6.17
Total time (s)	2.78	2.06	1.62
	100 concurrent clients x 1M points each		
Number of datablock requests	10 000 (100 per client)	1 000 (10 per client)	100 (1 per client)
Average datablock request time (s)	0.251	2.204	15.77
Maximum datablock request time (s)	0.489	3.807	15.96
Throughput (Millions of points per second)	3.96	4.2	6.25
Total time	25.27	23.89	15.99

Data download: multiple concurrent requests Finally, the third experiment evaluates the response of the system to several simultaneous requests of 1M points from different clients. It was implemented using the *Loadtest* tool (Fernández 2021) in the client side that allows to test a RESTful API by sending a high number of concurrent requests. The results in Table 6 show how in this test a smaller datablock, although it leads to a slower total download time, facilitates a faster response to highly concurrent workloads, enabling visualization or processing of data in the clients before the requests are completed. Again, a datablock size of 100K points represents a good tradeoff between datablock request latency and total time to complete all the clients requests, delivering 4.2M-4.9M points/s. It is worth noting that the performance of the system does not degrade substantially with multiple concurrent requests: the throughput for a single request of 1M points with the same datablock size is 5M points/s (Table 5). In the case of datablocks of 1M points, the throughput is about 6M points/s both for single or multiple concurrent requests.

5. Conclusions and future work

In this work we have proposed a conceptual model, web service API and reference implementation for a repository that can virtually store, organize and provide efficient access to large amounts of LiDAR data at world scale. The proposed model, organized into workspaces, datasets and datablocks, is simple, powerful and extensible, in the sense that it can be adapted to any application and be implemented using multiple combinations of spatial data structures. In our experiments the proposed implementation, using a single MongoDB database instance, showed a reliable performance, serving 4.2M-5M points/s to single or multiple concurrent requests. The extra cost of storing data in a MongoDB database instead of the original LAS/LAZ files is less than 5% for medium or large datasets.

The SPSLiDAR conceptual model can be improved in many ways. For instance the current model assumes that each dataset is a single LAS/LAZ file, or that the original files can be merged into a single one. In the latter case, when certain information varies in each file, such as the date of acquisition, it is lost. A possible solution would be to include this information in each datablock, or create a different group of datablocks for each file in the dataset, although this would penalize performance. In the same way, SPSLiDAR assumes the use of a common datum such as WGS84 for all the datasets. However, in applications that require a great precision, the use of local datums may be mandatory. The solution is to use WGS84 for the coordinates of the bounding box of the dataset (*boundingBox* attribute) to allow spatial queries on the world scale, and include a new attribute in the dataset encoding the local datum of the points in the datablocks. Finally, a rich set of spatial query operators, besides the simple window query, would be desirable.

In terms of functionality, our next goal is to implement collaborative editing in SPSLiDAR. In many applications raw LiDAR datasets have to be cleaned up, refined, repaired or segmented before they can be used. This can be a labor-intensive task that may require the parallel work of several operators. The access to SPSLiDAR through a centralized API and the organization of the dataset into datablocks enable collaborative editing within certain limits. Aspects that need to be studied include the locking of datablocks containing data being modified by a user, and the broadcasting of the changes to all the active clients. Another interesting feature that could be implemented in SPSLiDAR to complement collaborative editing is version control.

This would preserve the original datasets and allow recovery from unwanted changes. Finally, although the described architecture assumes that processing is done at the client side, some processing capabilities at the server side would simplify the development of many applications and improve their overall performance. Significant future work is needed to define and implement a set of useful generic operations (filtering, segmentation, visualization, etc.), or an engine for flexible server-side processing.

Data and code availability

The source code of the SPSLiDAR implementation, together with the scripts and links to the datasets used in the experimentation can be found in the following GitHub repository: `https://github.com/spslidar/spslidar`

Funding

This result is part of the research project RTI2018-099638-B-I00 funded by MCIN/ AEI/10.13039/501100011033/ and ERDF funds "A way of doing Europe". Also, the work has been funded by the University of Jaén (via ERDF funds) through the research project 1265116/2020.

Notes on contributors

Antonio J. Rueda-Ruiz received the Ph.D. degree in computer science from the University of Málaga, in 2004. He is currently an Associate Professor of computer science with the University of Jaén, Spain. His main interests are related to Computer Graphics, focused on designing geometric algorithms, processing 3D laser scanned data, GPU computing and Computational Archaeology.

Carlos J. Ogayar-Anguita received the M.S. degree in computer science and the Ph.D. degree in computer science from the University of Granada, in 2001 and 2006, respectively. He is currently an Associate Professor of computer science with the University of Jaén, Spain. His researches focus are on GPU computing, geometric algorithms, virtual reality, and 3D scanned data processing.

Rafael J. Segura-Sánchez received the M.S. degree in computer science from the University of Granada and the Ph.D. degree in computer science from the University of Granada,in 2001. He is currently a Full Professor with the University of Jaén, Spain. He has been working on several topics related to computer graphics, including solid modeling, computational geometry, virtual and augmented reality, and GPGPU.

Juan A. Béjar-Martos received a Bachelor's Degree in Computer Science from the University of Jaén in 2019 and a Master's Degree in Computer Science from the University of Jaén in 2021. He contributed to the project by working on the development and testing of the server application. He is currently working as a software engineer in KX Systems. His fields of interest are software development and big data.

Jorge Delgado-García received his PhD in Geology (Geostatistics) from the University of Granada (Spain) in 1993. He is currently an Associate Professor in cartographic, geodetic and photogrammetric engineering with the University of Jaén (Spain). He is currently working in several international projects related with Land Administration funded by the World Bank. His main interests are photogrammetric (image and laser)

data acquisition, processing and 3D modelling.

Author Contributions. Concept, research and methodology: Antonio J. Rueda-Ruiz, Rafael J. Segura-Sánchez, Carlos J. Ogáyar Anguita; Implementation: Antonio J. Rueda-Ruiz, Juan A. Béjar-Martos; Experimentation: Juan A. Béjar-Martos, Antonio J. Rueda-Ruiz, Rafael J. Segura-Sánchez and Carlos J. Ogáyar Anguita; Validation: Antonio J. Rueda-Ruiz, Carlos J. Ogáyar-Anguita, Rafael J. Segura-Sánchez and Jorge Delgado-Garcia; Writing—original draft: Antonio J. Rueda-Ruiz, Carlos J. Ogáyar-Anguita, Rafael J. Segura-Sánchez; Writing—review and editing: Antonio J. Rueda-Ruiz, Carlos J. Ogáyar-Anguita, Rafael J. Segura-Sánchez.

References

Agarwal, Sarthak; Rajan, K., 2017. Analyzing the performance of NoSQL vs. SQL databases for Spatial and Aggregate queries. *In*: *Free and Open Source Software for Geospatial (FOSS4G) Conference Proceedings*.

Baert, J., Lagae, A., and Dutré, P., 2014. Out-of-Core Construction of Sparse Voxel Octrees: Out-of-Core Construction of Sparse Voxel Octrees. *Computer Graphics Forum*, 33 (6), 220–227.

Boehler, W., Bordas Vicent, M., and Marbs, A., 2018. *Ogc testbed-14: Point cloud data handling engineering report*. i3mainz, Institute for Spatial Information and Surveying Technology, FH Mainz, Holzstrasse 36, 55116 Mainz, Germany. Available from: "`http://www.opengis.net/doc/PER/t14-D013`".

Boehm, J., 2014. File-centric Organization of large LiDAR Point Clouds in a Big Data context. *In*: *Workshop on processing large geospatial data*, Cardiff, UK. 69–76.

Boehm, J., Liu, K., and Alis, C., 2016. Sideloading – ingestion of large point clouds into the apache spark big data engine. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XLI-B2, 343–348.

Boulch, A., 2020. ConvPoint: Continuous Convolutions for Point Cloud Processing. *arXiv:1904.02375 [cs]*.

Bräunl, T., 2020. Lidar Sensors. *In*: *Robot Adventures in Python and C*. Cham: Springer International Publishing, 47–51.

Bunting, P., *et al.*, 2013. Sorted pulse data (SPD) library. Part I: A generic file format for LiDAR data from pulsed laser systems in terrestrial environments. *Computers & Geosciences*, 56, 197–206.

Cao, C., Preda, M., and Zaharia, T., 2019. 3D Point Cloud Compression: A Survey. *In*: *The 24th International Conference on 3D Web Technology*, July, LA CA USA. ACM, 1–9.

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

Deibe, D., Amor, M., and Doallo, R., 2019. 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.

Deng, X., *et al.*, 2019. Geospatial Big Data: New Paradigm of Remote Sensing Applications. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 12 (10), 3841–3851.

Desprat, C., Luga, H., and Jessel, J.P., 2015. Hybrid client-server and P2P network for web-based collaborative 3D design. *In*: *23rd International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG 2015)*. 229–238.

Discher, S., Richter, R., and Döllner, J., 2019. Concepts and techniques for web-based visualization and processing of massive 3D point clouds with semantics. *Graphical Models*, 104, 101036.

Doboš, J., *et al.*, 2013. XML3DRepo: a REST API for version controlled 3D assets on the web.

In: *Proceedings of the 18th International Conference on 3D Web Technology - Web3D '13*, San Sebastian, Spain. ACM Press, 47.

Doboš, J. and Steed, A., 2012. 3D revision control framework. *In*: *Proceedings of the 17th International Conference on 3D Web Technology - Web3D '12*, Los Angeles, California. ACM Press, 121.

Elseberg, J., Borrmann, D., and Nüchter, A., 2013. One billion points in the cloud – an octree for efficient processing of 3D laser scans. *ISPRS Journal of Photogrammetry and Remote Sensing*, 76, 76–88.

Evans, A., *et al.*, 2014a. 3D graphics on the web: A survey. *Computers & Graphics*, 41, 43–61.

Evans, M.R., *et al.*, 2014b. Spatial Big Data. Case studies on volume, velocity, and variety. *In*: *Big Data: Techniques and Technologies in Geoinformatics*. CRC Press, 149–176.

Fernández, A., 2021. Loadtest 5.1.2. Available from: `https://pypi.org/project/loadtest/`.

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

Gong, J., *et al.*, 2012. An Efficient Point Cloud Management Method Based on a 3D R-Tree. *Photogrammetric Engineering and Remote Sensing*, 78, 373–381.

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

Hongchao, M. and Wang, Z., 2011. Distributed data organization and parallel data retrieval methods for huge laser scanner point clouds. *Computers & Geosciences*, 37 (2), 193–201.

Houston, B., *et al.*, 2013. Clara.io: full-featured 3D content creation for the web and cloud era. *In*: *ACM SIGGRAPH 2013 Studio Talks*, Anaheim, California. ACM Press, 1–1.

Huang, L., *et al.*, 2020. OctSqueeze: Octree-Structured Entropy Model for LiDAR Compression. *arXiv:2005.07178 [cs, eess]*.

Hug, C., Krzystek, P., and Fuchs, W., 2004. Advanced lidar data processing with LasTools. *In*: *ISPRS Archives – Volume XXXV Part B2*. 832–837.

Instituto Geográfico Nacional, 2021. PNOA-LiDAR.

Isenburg, M., 2013. LASzip. *Photogrammetric Engineering & Remote Sensing*, 79 (2), 209–217.

Józsa, O., Börcs, A., and Benedek, C., 2013. Towards 4d virtual city reconstruction from lidar point cloud sequences. *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*, II-3/W1, 15–20.

Kang, L., *et al.*, 2019. Efficient Randomized Hierarchy Construction for Interactive Visualization of Large Scale Point Clouds. *In*: *2019 IEEE Fourth International Conference on Data Science in Cyberspace (DSC)*, June, Hangzhou, China. IEEE, 593–597.

Krämer, M. and Senner, I., 2015. A modular software architecture for processing of big geospatial data in the cloud. *Computers & Graphics*, 49, 69–81.

Kulawiak, M. and Kulawiak, M., 2017. Application of Web-GIS for Dissemination and 3D Visualization of Large-Volume LiDAR Data. *In*: I. Ivan, A. Singleton, J. Horák and T. Inspektor, eds. *The Rise of Big Spatial Data*. Cham: Springer International Publishing, 1–12. Series Title: Lecture Notes in Geoinformation and Cartography.

Kämpe, V., Sintorn, E., and Assarsson, U., 2013. High resolution sparse voxel DAGs. *ACM Transactions on Graphics*, 32 (4), 1–13.

Ladra, S., *et al.*, 2019. Space- and Time-Efficient Storage of LiDAR Point Clouds. *arXiv:1912.11859 [cs]*, 11811, 513–527.

Lee, J.G. and Kang, M., 2015. Geospatial Big Data: Challenges and Opportunities. *Big Data Research*, 2 (2), 74–81.

Lu, B., Wang, Q., and Li, A., 2019. Massive Point Cloud Space Management Method Based on Octree-Like Encoding. *Arabian Journal for Science and Engineering*, 44 (11), 9397–9411.

Mallet, C. and Bretar, F., 2009. Full-waveform topographic lidar: State-of-the-art. *ISPRS Journal of Photogrammetry and Remote Sensing*, 64 (1), 1–16.

Martinez-Rubi, O., *et al.*, 2015. Taming the beast: Free and open-source massive point cloud web visualization. *Capturing Reality*.

MongoDB, Inc, 2021. MongoDB. Available from: `https://www.mongodb.com/`.

National Oceanic and Atmospheric Administration, 2021. NOOA Digital Coast. Available from: `https://coast.noaa.gov/digitalcoast`.

Pajić, V., Govedarica, M., and Amović, M., 2018. Model of Point Cloud Data Management System in Big Data Paradigm. *ISPRS International Journal of Geo-Information*, 7 (7), 265.

Pandey, R., 2020. Performance Benchmarking and Comparison of Cloud-Based Databases MongoDB (NoSQL) Vs MySQL (Relational) using YCSB. Available from: `https://www.researchgate.net/publication/344047197`.

Poux, F. and Billen, R., 2019a. A Smart Point Cloud Infrastructure for intelligent environments. *In*: *Laser Scanning: An Emerging Technology in Structural Engineering*. CRC Press, 127–149.

Poux, F. and Billen, R., 2019b. Voxel-based 3D Point Cloud Semantic Segmentation: Unsupervised Geometric and Relationship Featuring vs Deep Learning Methods. *ISPRS International Journal of Geo-Information*, 8 (5), 213.

Poux, Florent, 2019. *The Smart Point Cloud: Structuring 3D intelligent point data*. Thesis (PhD). Université de Liège, Liège, Belgique.

Pääkkönen, P. and Pakkala, D., 2015. Reference Architecture and Classification of Technologies, Products and Services for Big Data Systems. *Big Data Research*, 2 (4), 166–186.

Reitz, K., 2021. Requests 2.25.1. Available from: `https://pypi.org/project/requests/`.

Richter, R., Discher, S., and Döllner, J., 2015. Out-of-Core Visualization of Classified 3D Point Clouds. *In*: M. Breunig, M. Al-Doori, E. Butwilowski, P.V. Kuper, J. Benner and K.H. Haefele, eds. *3D Geoinformation Science*. Cham: Springer International Publishing, 227–242. Series Title: Lecture Notes in Geoinformation and Cartography.

Scheiblauer, C. and Wimmer, M., 2011. Out-of-core selection and editing of huge point clouds. *Computers & Graphics*, 35 (2), 342–351.

Schutz, M. and Wimmer, M., 2019. Rendering Point Clouds with Compute Shaders. *In*: *SIGGRAPH Asia 2019 Posters*, November, Brisbane QLD Australia. ACM, 1–2.

Schön, B., *et al.*, 2013. Octree-based indexing for 3D pointclouds within an Oracle Spatial DBMS. *Computers & Geosciences*, 51, 430–438.

Schütz, M., 2016. *Potree: Rendering Large Point Clouds in Web Browsers*. Thesis (PhD). Vienna University of Technology.

Schütz, M., Ohrhallinger, S., and Wimmer, M., 2020. Fast Out-of-Core Octree Generation for Massive Point Clouds. *Computer Graphics Forum*, 39 (7), 155–167.

Ströter, D., *et al.*, 2020. OLBVH: octree linear bounding volume hierarchy for volumetric meshes. *The Visual Computer*, 36 (10-12), 2327–2340.

Sugimoto, K., *et al.*, 2017. Trends in efficient representation of 3D point clouds. *In*: *2017 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*, December, Kuala Lumpur. IEEE, 364–369.

The Spring Team and VMware, 2021. Spring. Available from: `https://spring.io/`.

Ullrich, A. and Pfennigbauer, M., 2019. Advances in lidar point cloud processing. *In*: M.D. Turner and G.W. Kamerman, eds. *Laser Radar Technology and Applications XXIV*, May, Baltimore, United States. SPIE, 19.

US Geological Survey, 2021. USGS 3D Elevation Program (USGS 3DEP). Available from: `https://www.usgs.gov/core-science-systems/ngp/3dep`.

Wang, Y., Lv, H., and Ma, Y., 2020. Geological tetrahedral model-oriented hybrid spatial indexing structure based on Octree and 3D R*-tree. *Arabian Journal of Geosciences*, 13 (15), 728.

Wang, Y., *et al.*, 2019. Dynamic Graph CNN for Learning on Point Clouds. *ACM Transactions on Graphics*, 38 (5), 1–12.

Xie, Y., Tian, J., and Zhu, X.X., 2020. Linking Points With Labels in 3D: A Review of Point Cloud Semantic Segmentation. *IEEE Geoscience and Remote Sensing Magazine*, 8 (4), 38–59.

Yang, J. and Huang, X., 2014. A Hybrid Spatial Index for Massive Point Cloud Data Management and Visualization: Massive Point Cloud Management and Visualization. *Transactions in GIS*, 18, 97–108.

Zhao, Y., *et al.*, 2019. 3D Point Capsule Networks. *In*: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June, Long Beach, CA, USA. IEEE, 1009–1018.