Point Cloud Data Management for Analytics in a Lakehouse

Balthasar Teuscher [1] and Martin Werner [1]

[1]TUM School of Engineering and Design, Technical University of Munich, Germany

Correspondence: Balthasar Teuscher (balthasar.teuscher@tum.de)

Abstract. Over time, the peculiarities of point clouds brought forth ample dedicated and specialized solutions for analyzing and managing point cloud data. However, providing analytical capabilities and visualization at scale remains challenging. We present a next-generation point cloud data management approach inspired by the Lakehouse pattern. It is exemplified by combining point clouds stored in raw files with a query engine, which instantly gives us an analysis-ready database management system with an SQL and DataFrame interface. We further demonstrate how to simplify and optimize this system through conversion to a columnar file format and a novel versatile repartitioning approach. Compared to existing solutions, the evaluation exhibits compelling performance, extraordinary flexibility, and exceptional simplicity.

Submission Type. analysis, infrastructure, algorithm, software.

BoK Concepts. [DM1] Foundations for Data Modelling Storage and Exploitation, [DA1] System design.

Keywords. point cloud, data management system, lakehouse architecture, query processing.

1 Introduction

There are ample specialized solutions to store, manage, visualize, and analyze point cloud data, each addressing shortcomings of traditional, generic data management systems (Isenburg, 2013; Schütz, 2016; Butler et al., 2021). However, scalability remains challenging, and the interoperability and integration with the broader big data analytics ecosystem require expensive transformation and conversion steps, leading to time-consuming data ingestion, duplication, and data amplification (Lokugam Hewage et al., 2022; Melnik et al., 2010; Yang et al., 2024).

A proposition to mitigate the high amount of processing time spent on record shredding, (un)marshaling, and (de)compression occurring throughout the whole point cloud data life cycle is the NoDB philosophy, which introduced a paradigm that proposes to access raw data files directly through a query engine (Alagiannis et al., 2012). Various dedicated point cloud data formats have been evaluated for ad-hoc querying with promising results (Bormann et al., 2024). Advancements in file-based solutions for big data analytics culminated in the Lakehouse architectural pattern targeted at cloud environments (Armbrust et al., 2021). It promotes decoupling computing and storage, embraces the NoDB philosophy of direct access to raw files, and relies on formats with chunked columnar layouts, including statistics and variable encoding schemes aiding storage efficiency and retrieval performance.

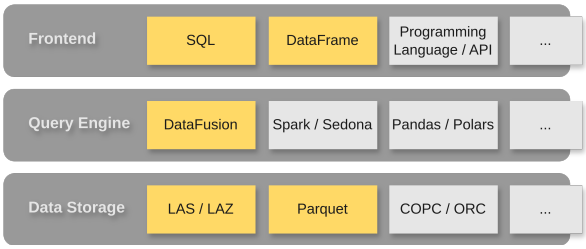

Figure 1. Lakehouse architecture for point cloud data analytics.

The main contribution of this paper is to establish a flexible data representation scheme for point clouds based on recent developments in the big data community and an evaluation of the implied performance for a selection of queries executed on real-world datasets. The evaluation shows that by adopting these widely accepted big data technologies, point cloud data can be scalably processed and visualized in the context of big data applications.

2 Related Work

Traditionally, large point clouds are provisioned as a collection of files and processed with tools such as PDAL (Butler et al., 2021) or LAStools (rapidlasso, 2007). Visualization commonly relies on dedicated formats such as Potree (Schütz, 2016) with levels of detail (LoD) based on layered space partitioning schemes (Gobbetti and Marton, 2004). However, these solutions do not accommodate

general-purpose and user-defined analytics and visualization workloads simultaneously, as their internal data structures and domain-specific language interfaces require dedicated tooling for creation and data access.

Database management systems (DBMS) gained popularity for point cloud data management integrating declarative interfaces, indexing capabilities, and scalability. Still, the relational model, combined with flat tables, fails to cope well with excessive amounts of point records, which provoked custom extensions to organize point clouds in patches or exploit space-filling curves to map multidimensional point clouds to a single dimension (pgPointcloud, 2013; van Oosterom et al., 2015; Psomadaki et al., 2016; Cura et al., 2017; Meijers, 2022).

Distributed systems and NoSQL-based solutions like MongoDB, Casandra, AtlasHDF, or Hadoop were explored in response to issues with the relational model (Deibe et al., 2018; Vo et al., 2019; Béjar-Martos et al., 2022; Werner and Li, 2022; Yang et al., 2024). Even though they can accommodate the vast size of point clouds, general problems prevail, including slow data ingestions, suboptimal indexing, and intransparent data storage obscured by domain-specific languages and extensions for access and retrieval.

Systems designed for online analytical processing (OLAP) workloads, especially columnar architectures, have shown promising results in query performance (Martinez-Rubi et al., 2015; Goncalves et al., 2016; Pavlovic et al., 2017). In this regard, the Lakehouse pattern emerged, combining the strength of Data Lake and Data Warehouse systems to overcome their limitations (Armbrust et al., 2021; Schneider et al., 2024). Initial integration of spatial semantics was investigated, though by adaptation of the underlying format and neglecting visualization even though emphasized as an integral part, and without mentioning point clouds at all (Ait Errami et al., 2023; Zhang et al., 2023).

3 Background

The *Lakehouse* pattern comprises open direct-access data formats, first-class support for machine learning and data science workloads, and state-of-the-art performance (Armbrust et al., 2021; Schneider et al., 2024). It targets online analytics in the cloud, where the data resides in an object store, and computing is disaggregated and on-demand, which offers the benefit of adapting resources and tools to the analytical workload without migrating the data (Ait Errami et al., 2023). As illustrated in Figure 1, the main components of a lakehouse architecture for point cloud data analytics are a user-facing frontend like SQL or DataFrame API, a query engine or compute engine like DataFusion or Spark, and a data store for objects like LAS/LAZ or Parquet files.

3.1 File Formats

Many public point cloud datasets from large-scale laser scanning or photogrammetric reconstruction are distributed as LAS files or LAZ files, which additionally integrates LASzip, a non-progressive block compression scheme (Isenburg, 2013). While delivering exceptional compression ratio, drawbacks are the computationally expensive compression scheme, which lacks random access, the row-based data layout, and the absence of fain-grained statistics.

File formats such as Apache Parquet and Apache Arrow are explicitly designed for online analytics, incorporating features tailored to NoDB or Lakehouse patterns. Apache Arrow is a language-agnostic columnar data layout specification for flat and hierarchical data. It supports the key feature of zero-copy data exchange across language, program, and system boundaries, which puts it in the same performance class as HDF5, Zarr, NumPy, and Tensorflow (Ahmad, 2022). Through native support in typical analytical toolings like GDAL, PDAL, Pandas, Tensorflow, and PyTorch, exchanging data between these tools is possible at zero cost.

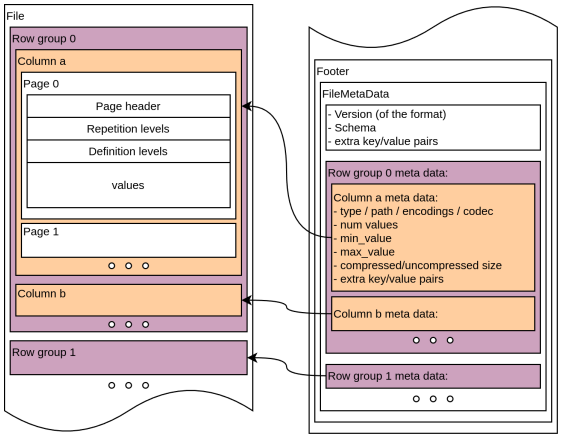

Figure 2. The Apache Parquet file format layout.

Apache Parquet is a column-oriented data file format for efficient data storage and retrieval, that is heavily inspired by Dremel, a design for interactive online analytics (Melnik et al., 2010, 2020). The format uses a hybrid storage model, coined Partition Attributes Across (PAX), where attributes are partitioned vertically (Ailamaki et al., 2002). These partitions are called *row groups* and contain a batch of records where the attributes are stored in *column chunks*, further split into *pages* (Figure 2). Metadata is located in the footer of the file, including optional min-max statistics, enabling one to skip parts of the file by evaluating predicates against it. As such, this serves as a sparse index for ad-hoc queries, and more importantly, it can faithfully represent an axis-aligned bounding box of a set of points.

3.2 Visualization

Data visualization is a fundamental aspect of point cloud data analytics. When visualizing large point cloud datasets, one needs to respect the constraint of the graphics processor unit (GPU) being only capable of displaying several million points. Given such a point budget, retrieving a representative sample in the field of view is the main retrieval task required to facilitate interactive visualization.

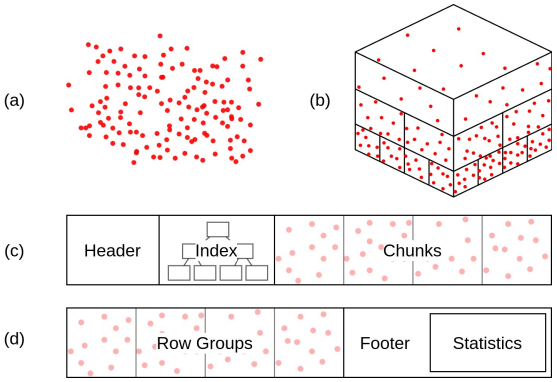

Figure 3. Data organization for point cloud visualization.

Current state-of-the-art file-based solutions for interactive point cloud visualization predominantly rely on a layered octree to organize the data (Gobbetti and Marton, 2004). The fundamental idea behind this is to sample points and assign them to a certain level of the hierarchical space partitioning tree corresponding to the Level of Detail (LoD) for efficient retrieval. Figure 3 illustrates the basic steps of partitioning a point cloud (a) into a layered space partitioning tree (b) and the corresponding layout of standard formats (c) compared to Parquet (d).

4 Methods

This chapter outlines the method employed to integrate point clouds into a Lakehouse architecture.

4.1 Point Cloud Representation

In its basic form, a point cloud can be formalized as a set of points P in space, most commonly a three-dimensional Euclidean space. Each point $p_i \in P$ is a set of attribute values of which coordinates represent the spatial location. Additionally, other attributes like time or importance can be considered as dimensions, leading to the following definition of a point cloud:

$$P \subseteq \mathbb{R}^n \times \prod_{i=1}^{m} D_i \tag{1}$$

where $\mathbb{R}^n$ is the metric space and D_i a specific attribute domain. From this definition, it follows naturally that point clouds can be represented as a table where rows are point records and columns are attributes with a subset thereof representing a geometric point in $\mathbb{R}^n$. Correspondingly, the LAS Point Data Record Formats can be represented lossless in Apache Arrow.

4.2 Data Partitioning

The fundamental aspect of partitioning is to support performant data retrieval by facilitating lowering the amount of data required to be processed to evaluate search predicates (Ait Errami et al., 2023). For spatial data, space or data partitioning schemes like grid, trees, and space-filling curves are commonly employed to split records into bounding volumes with low extent and overlap (Beckmann et al., 1990; Samet, 1984). In this work, we propose a novel windowed bounding volume partitioning algorithm to efficiently organize point clouds into arbitrary space partitioning schemes (Algorithm 1).

Algorithm 1 Windowed bounding volumes partitioning.

```
procedure PARTITION(P)                          ▷ set of points P
    E ← extent of P
    BV ← split E into a set of bounding volumes
    W ← split E into a set of reading windows
    C ← ∅                                       ▷ cash (in-memory)
    for all w ∈ W do                            ▷ iterate over windows
        P_w ← points contained by w
        BV_w ← bv ∈ BV intersecting w
        for all bv ∈ BV_w do                    ▷ map points to volumes
            P_bv ← {p | p ∈ P_w} intersecting bv
            C[bv] ← C[bv] ∪ P_bv
        end for
        W ← W \ w                               ▷ remove current window
        for all bv ∈ BV_w do
            if bv disjoint W then
                evict C[bv]                     ▷ write to disk
            end if
        end for
    end for
end procedure
```

The algorithm splits the extent of the point cloud into a set of reading windows W and target partitions BV. The reading windows are processed iteratively and define the subset of data to read, which is then mapped to the overlapping bounding volumes of the target partitions in memory. They are written to disk once they do not overlap with any remaining reading windows.

How the reading windows W are defined is crucial for the algorithm's performance. It is a tradeoff between the coverage of bounding volumes and reading window size that can result in either inefficient reading or high memory usage. However, the benefit is that one can adapt to the existing data characteristics and desired partitioning scheme to optimize for the available resources, and since the bounding volumes are mutually independent, parallelization is straightforward.

4.3 Importance Augmentation

A semantic notion of importance is required to partition point cloud data for visualization. Considering a layered tree, points are assigned to the nodes so that each node contains an equal amount of points, constituting a representative sample within its bounds. Another approach is assigning a continuous importance level (van Oosterom et al., 2022), which is embodied in this work through augmenting each point with a random uniform importance value $i \in [0,1[$.

Defining a quadtree or octree hierarchy incorporating the above importance semantics while somewhat respecting a certain amount of points per cell m requires finding out the depth d of the tree. This can be naively done by the following formula:

$$d = \lceil \log_p(\frac{n}{m}) \rceil \tag{2}$$

where n is the number of points normalized by the area of the minimum bounding rectangle, and p is the number of child nodes. From this, the importance range for a certain level can be derived and added as a dimension to the target bounding volumes, which then can be used to sample points to the respective nodes. This method has the added benefit that arbitrary sampling queries can be formulated and executed as range queries (Teuscher and Werner, 2024).

4.4 Implementation

The system used to exemplify this approach is built around DataFusion, an extensible query engine for building data-centric systems using Apache Arrow as the in-memory format (Lamb et al., 2024). DataFusion offers a *TableProvider* interface that facilitates the integration of custom data sources. Integrating LAZ files entails transforming the chunks into a stream of record batches which enables the ad-hoc use of SQL and the DataFrame API on LAZ files in DataFusion. Optionally, statistics for LAZ chunks are extracted through a complete scan to improve retrieval performance by pruning with spatial range predicate push-down. For Apache Parquet, such statistics are readily available from the metadata in a much more refined way. While these statistics are a key feature enabling the format's viability for a NoDB, respective Lakehouse approach, internal partitioning needs careful consideration to facilitate effective and efficient query processing.

4.5 Data and Software Availability

Research code and computational workflows supporting this publication are available on GitHub (https://github.com/tum-bgd/2025-AGILE-Lakehouse).

5 Evaluation and Experiments

In this section, we present the evaluation of our approach using different experiments and metrics. The dataset used is the Actueel Hoogtebestand Nederland (AHN3) in the LAZ file format, respectively two subsets thereof: one with about 200 million and another with about 2 billion points, to make the results comparable to benchmarks from the literature (van Oosterom et al., 2015). All experiments hereafter are conducted on a mobile computer with eight cores (16 threads), 32 GB memory, and 1 TB SSD.

5.1 Storage Footprint

The storage footprint of representing point clouds in Apache Parquet is shown in Table 1. It features different coordinate encodings and compression methods compared to LAS and LAZ as a relative baseline.

Table 1. Storage footprint of various coordinate encodings compared relative to LAZ.

Format	Coords	cLoI	Compression	Size (rel.)
LAS	i32[a]	-	-	4.25
LAZ	i32[a]	-	LASzip	1.00
Parquet	i32[b]	-	-	2.76
Parquet	f64		-	3.71
Parquet	f64	f32	-	4.26
Parquet	i32[b]	-	zstd(level3)	1.56
Parquet	f64	-	zstd(level3)	1.59
Parquet	f64	f32	zstd(level3)	2.10
Potree	(missing)	-	-	4.12

[a] Scale and offset in header.
[b] Scale and offset per point.

The results show that the storage footprint outperforms LAS without compression and is only about 1.5 times larger than LAZ with compression for grid-rounded coordinate encoding. With resolved coordinates and added importance, the uncompressed version is still in the range of LAS and Potree, while compressed about twice that of LAZ. The evaluation shows that storing point clouds in Apache Parquet is acceptable in terms of storage footprint. While compression for archival and publication still makes sense, abundant and cheap storage strongly suggests not applying block compression for analytical fidelity, and if so, prefer faster codecs with lower compression performance (Zeng et al., 2023).

5.2 Data Loading

Data loading is often evaluated as the time it takes to ingest the data from its source into a system until it is ready for querying. In our case, this entails converting from one file format to another. In Table 2, we measured the data loading time and throughput of four LAZ source files containing about 2 billion points.

Table 2. Data loading performance of ∼2B points.

Format	Partitioning	Statistics	Time	Throughput
LAZ	-	file	0.0s	-
LAZ	-	chunk	90.6s	22.0MP/s
Parquet	-	page	209.3s	9.5MP/s
Parquet	grid(xy)	page	375.6s	5.3MP/s
Parquet	grid(xyi)	page	358.6s	5.6MP/s
Parquet	quadtree	page	398.2s	5.0MP/s
Potree	octree	node	424.5s	4.7MP/s

Loading LAZ files only takes a split second since no data is processed except extracting the spatial extent from the headers to create statistics for pruning files. When additionally extracting the statistics of the LAZ chunks and cache in memory, a throughput of about 22 million points per second is reached with one io lane and decoding in parallel on all threads.

Partitioning with the windowed bounding volumes partitioning algorithm takes about the same time regardless of the applied scheme and levels the state-of-the-art performance of the PotreeConverter 2.0 (Schütz et al., 2020). Using Parquet files as the source for partitioning reduces the time taken by half compared to partitioning from LAZ with chunk statistics.

5.3 Query Performance

The query performance is evaluated with various range queries: a small and medium-sized rectangle in the plane (S_RECT, M_RECT), a neighborhood query with 1000 nearest neighbors (NN_1000), and a sampling range query with a fixed point budget of 700'000 points (I_700k). To assess the scalability, each query is evaluated on the 200 million and 2 billion points (Table 3). The results are based on uncompressed Parquet files, whereas with zstd compression, the runtime of the queries increased on average by about a factor of two. Additionally, the queries are run directly on the compressed LAZ files with and without chunk statistics for comparison.

Overall, the results match the highest-performing solutions among the state-of-the-art PostgreSQL with pgPointcloud, PDAL, MonetDB, and even Oracle Exadata (van Oosterom et al., 2015). Across all queries and datasets, Parquet with hierarchical partitioning is the only approach with < 0.5s query time.

5.4 Interactive Visualization

To evaluate the potential of our approach for interactive visualization, we simulated typical visualization queries at various levels of detail over the dataset with 2 billion points partitioned with a hierarchical scheme. Once the queries are retrieved with DataFusion and alternatively, an R*-tree from the statistics is created to identify intersecting row groups, which are then read directly from the Parquet files and refined by filtering.

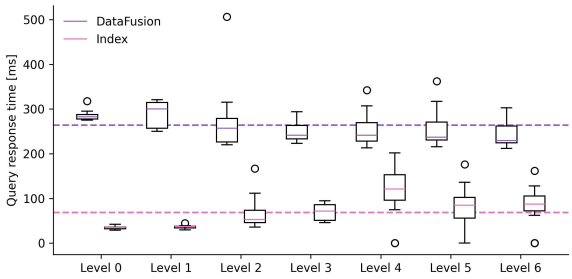

Figure 4. Vizualization query performance on different levels.

The results in Figure 4 show that the runtime of the visualization queries is, on average, below 300 ms with DataFusion and even below 100 ms when using an R*-tree index. The result set contains, on average, about 350'000 points, and reducing the result set to about 85'000 points displayed only slightly lower response times. This is fast enough for interactive visualization and leaves room for network overhead to power web-based viewers, outperforming similar approaches (Meijers, 2022).

6 Conclusion

In this paper, we investigate the potential of emerging approaches and well-established technologies from big data for point cloud data management. Founded on the Lakehouse pattern, we presented a novel approach that offers scalable and performant analytics capabilities while simultaneously supporting visualization through an efficient and versatile space partitioning procedure. Without any active components, such as a database producing continuous resource usage, it is ideally suited to cloud and elastic computing. The presented approach further shows that relying on generic and composable systems can bring certain advantages compared to niche domain-specific solutions. Especially for point clouds, it is possible to make geo non-special for many use cases and thereby access the potential of a feature-rich state-of-the-art ecosystem.

Declaration of Generative AI in writing

The authors declare that they have not used Generative AI tools in the preparation of this manuscript.

Author contributions. **Balthasar Teuscher**: Conceptualization, Methodology, Software, Investigation, Writing, Visualization. **Martin Werner**: Writing - Review & Editing, Supervision.

Competing interests. There are no competing interests to declare.

Table 3. Query performance of selected queries on ∼200M and ∼2B points; small rectangle (S_RECT), medium rectangle (M_RECT), nearest neighbours (NN_1000) and importance (I_700k).

Dataset	AHN3 extract with ∼200M				AHN3 extract with ∼2B			
Query	S_RECT	M_RECT	NN_1000	I_700k	S_RECT	M_RECT	NN_1000	I_700k
Points returned	74k	726k	1k	700k	74k	726k	1k	700k
Selectivity	0.5‰	4.0‰	0.025‰	3.50‰	0.037‰	0.363‰	0.002‰	0.350‰
LAZ	12.976s	13.003s	13.599s	13.787s	23.304s	23.295s	23.312s	90.415s
LAZ + statistics	0.427s	0.848s	0.378s	13.359s	**0.194s**	0.514s	**0.104s**	114.891s
Parquet (convert)	0.212s	0.282s	0.219s	0.819s	0.311s	0.364s	0.537s	11.130s
Parquet (grid xy)	**0.126s**	**0.173s**	**0.123s**	0.773s	0.288s	**0.321s**	0.313s	12.185s
Parquet (grid xyi)	0.192s	0.264s	0.151s	0.460s	0.328s	0.362s	0.446s	1.305s
Parquet (quadtree)	0.149s	0.221s	0.130s	**0.167s**	0.311s	0.376s	0.333s	**0.488s**

References

Ahmad, T.: Benchmarking Apache Arrow Flight - A wire-speed protocol for data transfer, querying and microservices, in: Benchmarking in the Data Center: Expanding to the Cloud, pp. 1–10, ACM, Seoul Republic of Korea, https://doi.org/10.1145/3527199.3527264, 2022.

Ailamaki, A., DeWitt, D. J., and Hill, M. D.: Data page layouts for relational databases on deep memory hierarchies, The VLDB Journal The International Journal on Very Large Data Bases, 11, 198–215, https://doi.org/10.1007/s00778-002-0074-9, 2002.

Ait Errami, S., Hajji, H., Ait El Kadi, K., and Badir, H.: Spatial big data architecture: From Data Warehouses and Data Lakes to the LakeHouse, Journal of Parallel and Distributed Computing, 176, 70–79, https://doi.org/10.1016/j.jpdc.2023.02.007, 2023.

Alagiannis, I., Borovica, R., Branco, M., Idreos, S., and Ailamaki, A.: NoDB: efficient query execution on raw data files, in: Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data, pp. 241–252, ACM, Scottsdale Arizona USA, https://doi.org/10.1145/2213836.2213864, 2012.

Armbrust, M., Ghodsi, A., Xin, R., and Zaharia, M.: Lakehouse: A New Generation of Open Platforms that Unify Data Warehousing and Advanced Analytics, 11th Annual Conference on Innovative Data Systems Research (CIDR '21), 2021.

Beckmann, N., Kriegel, H.-P., Schneider, R., and Seeger, B.: The R*-tree: an efficient and robust access method for points and rectangles, in: Proceedings of the 1990 ACM SIGMOD international conference on Management of data, pp. 322–331, ACM, Atlantic City New Jersey USA, https://doi.org/10.1145/93597.98741, 1990.

Bormann, P., Krämer, M., Würz, H. M., and Göhringer, P.: Executing Ad-Hoc Queries on Large Geospatial Data Sets Without Acceleration Structures, SN Computer Science, 5, 647, https://doi.org/10.1007/s42979-024-02986-z, 2024.

Butler, H., Chambers, B., Hartzell, P., and Glennie, C.: PDAL: An open source library for the processing and analysis of point clouds, Computers & Geosciences, 148, 104 680, https://doi.org/10.1016/j.cageo.2020.104680, 2021.

Béjar-Martos, J. A., Rueda-Ruiz, A. J., Ogayar-Anguita, C. J., Segura-Sánchez, R. J., and López-Ruiz, A.: Strategies for the Storage of Large LiDAR Datasets—A Performance Comparison, Remote Sensing, 14, 2623, https://doi.org/10.3390/rs14112623, 2022.

Cura, R., Perret, J., and Paparoditis, N.: A scalable and multi-purpose point cloud server (PCS) for easier and faster point cloud data management and processing, ISPRS Journal of Photogrammetry and Remote Sensing, 127, 39–56, https://doi.org/10.1016/j.isprsjprs.2016.06.012, 2017.

Deibe, D., Amor, M., and Doallo, R.: Big data storage technologies: a case study for web-based LiDAR visualization, in: 2018 IEEE International Conference on Big Data (Big Data), pp. 3831–3840, IEEE, Seattle, WA, USA, https://doi.org/10.1109/BigData.2018.8622589, 2018.

Gobbetti, E. and Marton, F.: Layered point clouds: a simple and efficient multiresolution structure for distributing and rendering gigantic point-sampled models, Computers & Graphics, 28, 815–826, https://doi.org/10.1016/j.cag.2004.08.010, 2004.

Goncalves, R., Zlatanova, S., Kyzirakos, K., Nourian, P., Alvanaki, F., and van Hage, W.: A columnar architecture for modern risk management systems, in: 2016 IEEE 12th International Conference on e-Science (e-Science), pp. 424–429, IEEE, Baltimore, MD, USA, https://doi.org/10.1109/eScience.2016.7870929, 2016.

Isenburg, M.: LASzip: lossless compression of LiDAR data, Photogrammetric Engineering and Remote Sensing, 2013.

Lamb, A., Shen, Y., Heres, D., Chakraborty, J., Kabak, M. O., Hsieh, L.-C., and Sun, C.: Apache Arrow DataFusion: A Fast, Embeddable, Modular Analytic Query Engine, in: Companion of the 2024 International Conference on Management of Data, pp. 5–17, ACM, Santiago AA Chile, https://doi.org/10.1145/3626246.3653368, 2024.

Lokugam Hewage, C. N., Laefer, D. F., Vo, A.-V., Le-Khac, N.-A., and Bertolotto, M.: Scalability and Performance of LiDAR Point Cloud Data Management Systems: A State-of-the-Art Review, Remote Sensing, 14, 5277, https://doi.org/10.3390/rs14205277, 2022.

Martinez-Rubi, O., Van Oosterom, P., Gonçalves, R., Tijssen, T., Ivanova, M., Kersten, M. L., and Alvanaki, F.: Benchmarking and improving point cloud data management in MonetDB, SIGSPATIAL Special, 6, 11–18, https://doi.org/10.1145/2744700.2744702, 2015.

Meijers, M.: PCSERVE – ND-POINTCLOUDS RETRIEVAL OVER THE WEB, ISPRS Annals of the Photogrammetry,

Remote Sensing and Spatial Information Sciences, X-4/W2-2022, 193–200, https://doi.org/10.5194/isprs-annals-X-4-W2-2022-193-2022, 2022.

Melnik, S., Gubarev, A., Long, J. J., Romer, G., Shivakumar, S., Tolton, M., and Vassilakis, T.: Dremel: Interactive Analysis of Web-Scale Datasets, Proceedings of the VLDB Endowment, 3, 330–339, https://doi.org/10.14778/1920841.1920886, 2010.

Melnik, S., Gubarev, A., Long, J. J., Romer, G., Shivakumar, S., Tolton, M., Vassilakis, T., Ahmadi, H., Delorey, D., Min, S., Pasumansky, M., and Shute, J.: Dremel: A Decade of Interactive SQL Analysis at Web Scale, Proceedings of the VLDB Endowment, 13, 3461–3472, https://doi.org/10.14778/3415478.3415568, 2020.

Pavlovic, M., Bastian, K.-N., Gildhoff, H., and Ailamaki, A.: Dictionary Compression in Point Cloud Data Management, in: Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, pp. 1–10, ACM, Redondo Beach CA USA, https://doi.org/10.1145/3139958.3139969, 2017.

pgPointcloud: pgPointcloud - A PostgreSQL extension for storing point cloud (LIDAR) data., https://pgpointcloud.github.io/pointcloud/, 2013.

Psomadaki, S., Van Oosterom, P. J. M., Tijssen, T. P. M., and Baart, F.: USING A SPACE FILLING CURVE APPROACH FOR THE MANAGEMENT OFDYNAMIC POINT CLOUDS, ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences, IV-2/W1, 107–118, https://doi.org/10.5194/isprs-annals-IV-2-W1-107-2016, 2016.

rapidlasso: LAStools (rapidlasso), https://rapidlasso.de/, 2007.

Samet, H.: The Quadtree and Related Hierarchical Data Structures, ACM Computing Surveys, 16, 187–260, https://doi.org/10.1145/356924.356930, 1984.

Schneider, J., Gröger, C., Lutsch, A., Schwarz, H., and Mitschang, B.: The Lakehouse: State of the Art on Concepts and Technologies, SN Computer Science, 5, 449, https://doi.org/10.1007/s42979-024-02737-0, 2024.

Schütz, M.: Potree: Rendering large point clouds in web browsers, Ph.D. thesis, Technische Universität Wien, Wien, https://publik.tuwien.ac.at/files/publik_252607.pdf, 2016.

Schütz, M., Ohrhallinger, S., and Wimmer, M.: Fast Out-of-Core Octree Generation for Massive Point Clouds, Computer Graphics Forum, 39, 155–167, https://doi.org/10.1111/cgf.14134, 2020.

Teuscher, B. and Werner, M.: Random Data Distribution for Efficient Parallel Point Cloud Processing, AGILE: GIScience Series, 5, 1–10, https://doi.org/10.5194/agile-giss-5-15-2024, 2024.

van Oosterom, P., Martinez-Rubi, O., Ivanova, M., Horhammer, M., Geringer, D., Ravada, S., Tijssen, T., Kodde, M., and Gonçalves, R.: Massive point cloud data management: Design, implementation and execution of a point cloud benchmark, Computers & Graphics, 49, 92–125, https://doi.org/10.1016/j.cag.2015.01.007, 2015.

van Oosterom, P., van Oosterom, S., Liu, H., Thompson, R., Meijers, M., and Verbree, E.: Organizing and visualizing point clouds with continuous levels of detail, ISPRS Journal of Photogrammetry and Remote Sensing, 194, 119–131, https://doi.org/10.1016/j.isprsjprs.2022.10.004, 2022.

Vo, A. V., Hewage, C. N. L., Russo, G., Chauhan, N., Laefer, D. F., Bertolotto, M., Le-Khac, N.-A., and Oftendinger, U.: Efficient LiDAR point cloud data encoding for scalable data management within the Hadoop eco-system, in: 2019 IEEE International Conference on Big Data (Big Data), pp. 5644–5653, https://doi.org/10.1109/BigData47090.2019.9006044, 2019.

Werner, M. and Li, H.: AtlasHDF: an efficient big data framework for GeoAI, in: Proceedings of the 10th ACM SIGSPATIAL International Workshop on Analytics for Big Geospatial Data, pp. 1–7, ACM, Seattle Washington, https://doi.org/10.1145/3557917.3567615, 2022.

Yang, Y., Zuo, X., Zhao, K., and Li, Y.: Integrating NoSQL, Hilbert Curve, and R*-Tree to Efficiently Manage Mobile LiDAR Point Cloud Data, ISPRS International Journal of Geo-Information, 13, 253, https://doi.org/10.3390/ijgi13070253, 2024.

Zeng, X., Hui, Y., Shen, J., Pavlo, A., McKinney, W., and Zhang, H.: An Empirical Evaluation of Columnar Storage Formats, Proceedings of the VLDB Endowment, 17, 148–161, https://doi.org/10.14778/3626292.3626298, 2023.

Zhang, Y., Peng, B., Du, Y., and Su, J.: GeoLake: Bringing Geospatial Support to Lakehouses, IEEE Access, 11, 143 037–143 049, https://doi.org/10.1109/ACCESS.2023.3343953, 2023.