

Comparative Analysis of 3D Point Cloud Libraries with Focus on Efficiency and Practical Use

João P. S. Alves, Matheus F. Silva, and Felipe A. P. de Figueiredo

Abstract—Point cloud processing is essential for modern robotics and three-dimensional mapping systems. This study investigates the computational efficiency of the Open3D, PyntCloud, and Trimesh libraries when handling point cloud data in PCD, PLY, and LAS formats. The approach evaluates usability, visualization, and algorithmic support under different computational demands. The results indicate that Open3D consistently outperforms the other libraries in terms of memory efficiency, execution time, and scalability across datasets of increasing size, while maintaining a simple and consistent Python interface. These characteristics make Open3D particularly effective for large-scale point cloud processing workflows.

Keywords—Point Clouds, Open3D, Geometric algorithms, PyntCloud, Trimesh, Efficiency

I. INTRODUCTION

Representing three-dimensional data as point clouds requires efficient storage and processing strategies due to their unstructured nature and large volumes. Despite the availability of several processing libraries, there is a lack of unified studies jointly analyzing their data structures, performance, and file format support within scientific workflows. The Point Cloud Library (PCL) established the foundation for point cloud processing using high-performance C++ structures [1]. More recently, libraries such as Open3D [2], PyntCloud [3], and Trimesh [4] have emerged, combining computational efficiency with Python flexibility. Their growing adoption highlights the need for a clearer understanding of differences in data representation, processing behavior, and compatibility with formats such as Point Cloud Data (PCD) [5], Polygon File Format (PLY) [6], and LASer File Format (LAS) [7]. This work presents a comparative experimental analysis evaluating how these frameworks structure point cloud data, their performance characteristics, and their support for common file formats, aiming to provide practical insights for scientific computing applications.

II. METHODOLOGY

This work presents a comparative analysis of 3D point cloud processing libraries, focusing on the handling, processing, and visualization of 3D point data.

Experiments were conducted on a machine equipped with an Intel Core i7-7700 CPU, 16 GB of 2400 MHz RAM, and a 1 TB SSD, without GPU acceleration. Although absolute performance depends on the hardware platform, the relative behavior of the evaluated libraries remains representative under identical experimental conditions.

The software environment consisted of Python 3.12.10 in a Conda [16] virtual environment with Jupyter Notebook. The evaluated libraries were Open3D (v0.19.0), PyntCloud

(v0.3.1), and Trimesh (v4.11.5). The benchmark implementation is publicly available on GitHub [17].

The datasets were the original Stanford Bunny, Thai Statue, and Lucy point clouds from the Stanford 3D Scanning Repository [18]. Reported sizes correspond to the original datasets, while voxel downsampling and normal estimation were evaluated only as processing operations.

Each operation was executed 15 times after three warm-up iterations. File loading benchmarks were performed under warm-cache conditions, as the operating system page cache was not explicitly cleared. Reported results correspond to the average of the 15 executions.

CPU utilization was computed from the process CPU time (user and system), normalized by the elapsed wall-clock time and the number of logical CPU cores. Results are reported on a per-core basis (0–100%), where 100% corresponds to the full utilization of one logical CPU core. Memory usage was measured as the variation in Resident Set Size (Δ RSS) using `psutil`, with `gc.collect()` invoked between iterations to reduce memory accumulation.

The methodology evaluates the libraries according to the four aspects described next.

A. File Compatibility

The libraries are tested for their ability to read and process common point cloud file formats, including PLY, PCD, and LAS. This assessment ensures that libraries can handle a variety of real-world datasets.

B. Ease of Use

Ease of use is evaluated based on the installation process, integration within Python environments, and implementation of basic algorithms. Python-based libraries enable rapid prototyping and interactive experimentation, facilitating the development and evaluation of point cloud processing workflows.

C. Geometric Algorithms

An analysis of the geometric algorithms available in each library is presented, focusing on their functionalities rather than their direct application to datasets. The comparison considers the presence of key structures and methods, such as k-dimensional tree (KD-Tree) [8] and Octree for efficient spatial organization and neighbor search, as well as algorithms such as Random Sample Consensus (RANSAC) [9] for robust model fitting in the presence of noise and outliers, and Iterative Closest Point (ICP) [10] for point cloud registration and alignment. These tools are essential for enabling common point cloud processing tasks, including segmentation, filtering, and geometric reconstruction.

D. Performance Evaluation

Performance is evaluated based on RAM usage, processing time, and CPU utilization during the execution of point cloud processing tasks. These metrics are used to assess the efficiency and scalability of each library when handling large datasets.

III. RESULTS

A. File Compatibility

The compatibility of each library with common 3D data formats, including point clouds, meshes, and images, is evaluated. Open3D supports common point cloud formats such as PLY, PCD, and XYZ, including color information (RGB), as well as mesh formats such as OBJ and STL, and RGB-D data [11], while LAS/LAZ require external libraries (e.g., *laspy*). PyntCloud handles PLY and PCD natively and supports LAS/LAZ via *laspy*, in addition to mesh formats like OBJ and tabular data such as CSV and TXT, relying on Pandas DataFrames for data manipulation [12]. Trimesh, in turn, is oriented toward mesh processing and supports formats such as PLY, STL, OBJ, OFF, and GLTF/GLB.

B. Ease of use

Ease of use is evaluated considering both the installation process and the implementation of basic algorithms. The analyzed libraries can be easily installed via the Python package manager *pip*, enabling quick setup and straightforward integration into development environments. As Python-based tools, they also provide advantages in accessibility and rapid development, with support for recent Python versions (3.5+), facilitating adoption and maintenance. Additionally, they integrate well with interactive computing environments, such as Jupyter Notebook, allowing efficient experimentation and visualization through step-by-step code execution and immediate feedback.

C. Visualization

The visualization capabilities of each library are analyzed, focusing on the availability of native viewers and their level of interactivity.

Open3D provides an intuitive OpenGL-based viewer with interactive controls (e.g., zoom, rotation, and translation), requiring minimal code for rendering and offering a stable environment for exploring large point clouds (Figure 1). In contrast, PyntCloud does not include a native visualization engine. It relies on external libraries and is primarily used for data processing tasks before rendering. Trimesh, focused on mesh processing, follows a headless approach and depends on third-party tools (e.g., Pyglet [14], matplotlib [15]) for visualization.

D. Geometric algorithms

Open3D provides a comprehensive set of geometric algorithms, including KD-Tree structures for efficient nearest-neighbor searches and Octrees for hierarchical spatial partitioning. These data structures are used to accelerate operations such as normal estimation, which relies on local neighborhood queries, and voxel grid-based downsampling, which leverages spatial partitioning to reduce point cloud density. Additionally, they support advanced registration techniques such as ICP

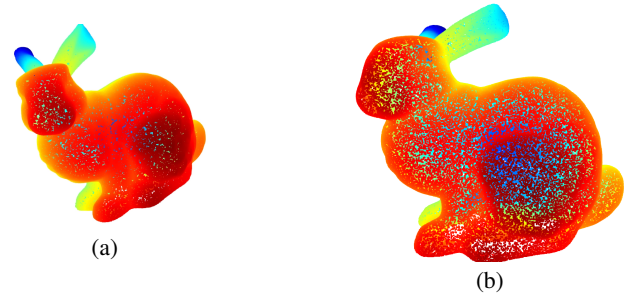


Fig. 1. Interactive visualization of a point cloud in Open3D using an OpenGL-based viewer. (a) front view and (b) rear view. Since the point cloud does not contain RGB color information, Open3D automatically applies a color map based on the normalized z -coordinate.

and RANSAC, enabling accurate alignment and robust model fitting in the presence of noise and outliers.

PyntCloud supports KD-Tree operations and voxel grid representations for tasks such as downsampling, but these structures require explicit construction and rely on higher-level data representations. While normal estimation is available via neighborhood analysis, the absence of native registration methods such as ICP and RANSAC limits its use in alignment and model-fitting tasks.

Trimesh focuses on mesh processing (vertices/faces) over raw point clouds. While it offers basic proximity queries, it lacks optimized spatial indexing for large scales and native support for advanced registration (ICP, RANSAC), limiting its use for precise alignment and robust geometric estimation.

To summarize the availability of geometric algorithms across the evaluated libraries, Table I presents a comparative overview.

TABLE I
GEOMETRIC ALGORITHM SUPPORT ACROSS LIBRARIES.

Library	ICP	RANSAC	KD-Tree	Octree
Open3D	Yes	Yes	Yes	Yes
PyntCloud	No	Limited ¹	Yes	Yes
Trimesh	No	No	Limited	No

E. Performance

In the performance analysis, three datasets are considered: small (100 thousand points), medium (5 million), and large (15 million). The evaluation analyzes how each library behaves with increasing data volume, considering memory usage, execution time, and CPU utilization.

The reported performance metrics correspond to the experimental platform described in the methodology. Although absolute performance depends on the hardware and software configuration, the relative differences among the evaluated libraries remain representative, since all experiments were conducted under identical conditions.

Loading in Open3D is predominantly I/O-bound, as indicated by low CPU usage (10.32% for 15M points), which means that execution time is mainly driven by disk-to-memory transfer (Table II). For the largest dataset, Open3D requires 2.9 s and 321.08 MB, compared to 20.4 s and 1471.7 MB in Trimesh. This reduced memory footprint results from optimized data structures that minimize intermediate copies, making Open3D suitable for large-scale and memory-constrained scenarios.

¹"Limited" indicates partial or indirect support, typically requiring external libraries or providing only basic functionality.

PyntCloud exhibits mixed behavior, combining I/O-bound and CPU-bound stages, as indicated by high CPU usage (81.43%) and relatively low execution time (1.2873 s) for the 15M dataset. This is due to parsing operations and DataFrame-based conversions, which introduce overhead associated with Pandas memory management. Trimesh shows high CPU utilization (68.81%), reflecting significant overhead during mesh parsing and structure construction, which, combined with I/O costs, leads to poor scalability for dense datasets (Table II).

These differences highlight distinct execution profiles, ranging from I/O-bound (Open3D) to mixed I/O- and CPU-bound behavior (PyntCloud and Trimesh).

When memory is limited, Open3D enables efficient processing through voxel-based downsampling and spatial data structures such as octrees and KD-trees. PyntCloud can partially mitigate memory usage through its integration with Pandas DataFrames, allowing subset-based processing, while Trimesh lacks equivalent mechanisms, limiting its performance on very large datasets.

Downsampling also differs significantly. Open3D achieves low execution time (0.1961 s for 5M points) and minimal memory usage (0.06 MB), due to voxel-based aggregation over contiguous memory structures. PyntCloud shows higher CPU usage (up to 96%) and memory consumption (up to 802.38 MB for 15M), indicating limited scalability. Trimesh does not provide an equivalent optimized downsampling operation in this context. Table II shows the summarized results for downsampling.

Normal estimation, used to compute surface normals for point clouds, was evaluated exclusively in Open3D, as it is the only library among those analyzed that provides a native and efficient implementation with minimal overhead.

Overall, Open3D presents the most balanced performance, combining low memory usage and stable execution times (e.g., 0.176 s and 70.11% CPU for 15M points) as summarized in Table II. PyntCloud exhibits higher memory consumption under scaling, while Trimesh shows the highest cost in both memory and time, limiting its applicability in large-scale workflows, as shown in Table II.

TABLE II
PERFORMANCE COMPARISON ACROSS DATASETS

Dataset	Library	Operation	Memory (MB)	Time (s)	CPU (%)
100k	Open3D	Load	2.290	0.0135	9.53
		Downsample	0.002	0.0046	7.62
		Normals	0.018	0.0026	10.43
	PyntCloud	Load	4.57	0.0044	13.09
		Downsample	2.42	0.0171	83.72
	Trimesh	Load	4.58	0.0024	59.72
5M	Open3D	Load	114.440	1.0975	10.38
		Downsample	0.060	0.1961	11.97
		Normals	0.005	0.0156	18.14
	PyntCloud	Load	228.88	0.4512	58.63
		Downsample	158.39	0.5628	96.00
	Trimesh	Load	525.62	6.3697	76.86
15M	Open3D	Load	321.08	2.9378	10.32
		Downsample	3.43	0.6627	9.46
		Normals	4.24	0.1767	70.11
	PyntCloud	Load	642.15	1.2873	81.43
		Downsample	802.38	2.6800	74.86
	Trimesh	Load	1471.70	20.4504	68.81

IV. CONCLUSIONS

This study compared libraries for point cloud processing in terms of usability, geometric support, and performance. Open3D consistently demonstrated superior efficiency and scalability, combining low memory consumption, fast execution times, and an optimized Python interface with integrated visualization capabilities. These characteristics make Open3D the most suitable choice among the evaluated libraries for efficient point cloud processing workflows. Future work includes extending the evaluation to additional large-scale datasets and real-world applications, as well as incorporating GPU-based acceleration and out-of-core processing scenarios to analyze scalability under constrained computational resources further.

ACKNOWLEDGEMENTS

This work was partially funded by CNPq (302085/2025-4, 306199/2025-4), by FAPEMIG (PPE-00124-23, APQ-05305-23, RED-00194-23, APQ-03162-24), by the Brasil 6G project (01245.020548/2021-07), by the projects XGM-AFCCT-2024-9-1-1 and XGM-AFCCT-2024-5-1-2 supported by xGMobile-EMBRAPIL-Inatel, with financial resources from the PPI IoT/Manufatura 4.0 from MCTI grant number 052/2023, and by FINEP (nº 1060/2 contract 01.25.0883.00).

REFERENCES

- [1] R. B. Rusu and S. Cousins, "3D is here: Point Cloud Library (PCL)," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, 2011, pp. 1–4.
- [2] Q.-Y. Zhou *et al.*, "Open3D: A Modern Library for 3D Data Processing," *arXiv preprint arXiv:1801.09847*, 2018.
- [3] D. de la Iglesia Castro, "PyntCloud: A Python Library for Working with 3D Point Clouds," 2018. [Online]. Available: <https://github.com/daavoo/pyntcloud>. Accessed: Jul. 19, 2026.
- [4] M. Dawson-Haggerty *et al.*, "trimesh: Load and analyze triangular meshes," Computer software, 2019.
- [5] Point Cloud Library, "PCD (Point Cloud Data) File Format," [Online]. Available: https://pointclouds.org/documentation/tutorials/pcd_file_format.html. Accessed: Jul. 19, 2026.
- [6] G. Turk, "The PLY Polygon File Format," Technical Report, Stanford Computer Graphics Laboratory, Stanford University, 1994.
- [7] ASPRS, "LAS Specification Version 1.4," Technical report, 2019.
- [8] J. L. Bentley, "Multidimensional Binary Search Trees Used for Associative Searching," *Communications of the ACM*, vol. 18, no. 9, pp. 509–517, 1975.
- [9] M. A. Fischler and R. C. Bolles, "Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.
- [10] P. J. Besl and N. D. McKay, "A Method for Registration of 3-D Shapes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, no. 2, pp. 239–256, 1992.
- [11] A. Lopes, R. Souza, and H. Pedrini, "A Survey on RGB-D Datasets," *Computer Vision and Image Understanding*, 2022.
- [12] W. McKinney, "Data structures for statistical computing in Python," in *Proc. 9th Python in Science Conf.*, 2010, pp. 56–61.
- [13] W. Schroeder, K. Martin, and B. Lorensen, *The Visualization Toolkit: An Object-Oriented Approach to 3D Graphics*, 4th ed. Kitware, 2006.
- [14] A. Holkner, "Pyglet: A cross-platform windowing and multimedia library for Python," Computer software, 2006.
- [15] J. D. Hunter, "Matplotlib: A 2D Graphics Environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.
- [16] conda contributors, "conda: A system-level, binary package and environment manager," Computer software, 2026. [Online]. Available: <https://github.com/conda/conda>. Accessed: Jul. 19, 2026.
- [17] J. P. S. Alves, "Comparative Analysis of 3D Point Cloud Libraries," GitHub repository, 2026. [Online]. Available: <https://github.com/JoaoPedroSA2/Comparative-Analysis-of-3D-Point-Cloud-Libraries>. Accessed: Jul. 19, 2026.
- [18] Stanford Computer Graphics Laboratory, "The Stanford 3D Scanning Repository," Available: <https://graphics.stanford.edu/data/3Dscanrep/>. Accessed: Jul. 19, 2026.