Point Cloud Classification of Roofs and Facades Using Object-Oriented Programming (OOP)

Jarence David D. Casisirano

GmE 205 - Object Oriented Programming for Geomatics Applications

Final Project Paper

Abstract

Acquiring point cloud data and processing such information to generate insights have become very prominent today with the advent of precise and accurate measuring techniques and technologies. Because the stages in processing point cloud data can be repetitive and/or iterative, there is an opportunity to explore automation for this particular use case. This project explores the application of Object-Oriented Programming (OOP) principles and related photogrammetric concepts in the classification of roofs and facades from point cloud data. Leveraging modular and reusable code architecture, the study demonstrates a comprehensive pipeline for point cloud processing, segmentation, and classification, with an emphasis on scalability and maintainability.

1. Introduction

Point cloud data has become a critical asset in 3D modeling, urban planning, and geospatial analysis. However, processing and extracting meaningful information from such data can be challenging due to its unstructured nature. This project focuses on classifying roofs and facades from point clouds using an object-oriented programming paradigm. The structured approach ensures a reusable and scalable framework while maintaining clarity and modularity in implementation.

1.1 Rationale Behind the Project

Generating 3D models have become a great advantage in many regards, particularly in urban development and planning. Accurate 3D models are being used not only for visualization purposes but also for simulations and scenario analysis. One common method of generating accurate 3D models is by using point cloud data—a collection of data points in a three-dimensional (3D) space that represents the external surface of an object or an environment. Each point in a point cloud has coordinates (e.g., X, Y, and Z) representing its position in 3D space.

The rapid advancement of LiDAR technology (i.e., light detection and ranging) and photogrammetry has enabled the acquisition of high-resolution point cloud data. While these datasets are highly detailed, they lack explicit structure, making it difficult to extract specific features like roofs and facades efficiently. Developing an automated, modular pipeline for classifying roofs and facades addresses a key need in urban modeling and related applications (e.g., solar potential estimation, wind simulation, etc.). This project leverages object-oriented programming (OOP) using Python to create a scalable, reusable framework that simplifies and modularizes the processing, segmentation, and classification of point cloud data.

1.2 Significance of the Project

This project contributes to advancing research in 3D modeling and its related applications in the following ways:

- **Urban Planning**: Efficient extraction of roofs and facades enables detailed city modeling, which supports urban planning and infrastructure development.
- **Building Information Modeling (BIM)**: Automating the classification of building features like roofs and facades can streamline BIM workflows for architects and engineers.
- **Disaster Management**: Accurate 3D models can assist in identifying structural damage to buildings after natural disasters.
- **Solar Potential Estimation**: Extracting and analyzing roof planes aids in determining suitable areas for solar panel installations.
- **Heritage Preservation**: Detailed classification of building exteriors facilitates 3D reconstruction for cultural heritage preservation.

1.3 Objectives of the Project

The primary objectives of this project are:

1. To develop a modular pipeline for processing, segmenting, and classifying point cloud data.
2. To implement object-oriented programming principles for enhanced modularity, reusability, and scalability.
3. To accurately classify roofs and facades from point cloud data using geometric properties such as plane orientation and height thresholds.
4. To provide effective visualization of classified structures for further analysis.
5. To demonstrate the applicability of the developed pipeline for real-world 3D modeling and geospatial applications.

1.4 Related Concepts

This project draws on a range of foundational concepts and techniques. These key concepts include:

RANSAC algorithm: The Random Sample Consensus (RANSAC) algorithm is critical for detecting the planar surfaces within the point cloud data, especially for noisy datasets. It identifies planes by iteratively fitting models to subsets of points while excluding

outliers making it ideal for roof and facade classification (Hamid-Lakzaeian, 2019). In simpler terms, it works by repeatedly trying to fit a plane to random subsets of points and finding the best fit for the majority of the points in the data.

Geometric features of planes: After identifying the planes, the normal vectors to these planes are calculated and then used to classify the planes as either roofs or facades based on their orientation (Kushwaha et al., 2019). Planar surfaces in 3D space are characterized by normal vectors, which indicate their orientation. This project classifies roofs and facades by analyzing the angles of these vectors and applying height thresholds to distinguish between a roof and a facade. Planes with normal vectors predominantly oriented upwards (i.e., a large Z-component) were classified as roofs, whereas those with normal vectors oriented more horizontally (i.e., a smaller Z-component) were classified as facades.

Voxel Downsampling: To handle the high density of point cloud data, voxel grid downsampling was employed to reduce the number of points while retaining the overall structure (Lyu et al., 2024). Voxel downsampling is a preprocessing technique used to reduce the density of a point cloud by dividing the 3D space into small cubes (voxels) of a specified size. Within each voxel, the points are replaced by their centroid, preserving the overall structure of the point cloud while reducing the number of points. This simplifies processing and computation while allowing control over the level of detail by adjusting the voxel size. This preprocessing step balances computational efficiency with data accuracy.

Statistical Outlier Removal: To improve the reliability of segmentation and classification as well as manage the computational requirements of the project, noise is reduced by removing points that deviate significantly from their neighbors using the Statistical Outlier Removal filter. This ensures that detected planes and structures are based on accurate data.

Object-oriented programming (OOP): This project leverages OOP principles to design a modular and scalable codebase. By encapsulating specific tasks (e.g., loading, preprocessing, segmentation) into independent classes, the pipeline ensures reusability and maintainability, which are essential for large-scale geospatial projects.

2. Methodology

2.1 Workflow Overview

The methodology followed a structured workflow, as shown in Figure 1. The project began with loading the raw point cloud data, followed by preprocessing to remove noise and downsample the dataset. The cleaned data was then segmented to isolate regions of interest, such as building structures. Planar surfaces were detected from the

segmented data and classified into roofs and facades based on their geometric properties. Finally, the results were visualized for visual analysis and evaluation.

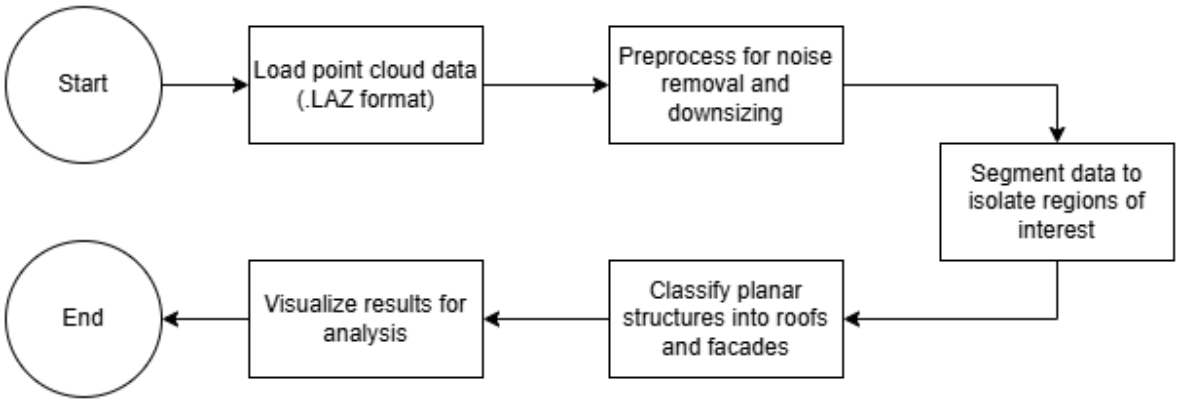

Figure 1 Point cloud classification workflow.

2.2 System Architecture

The system architecture, depicted in Figure 2, consists of modular components implemented as Python classes. These classes were designed following object-oriented programming principles to ensure separation of concerns and reusability.

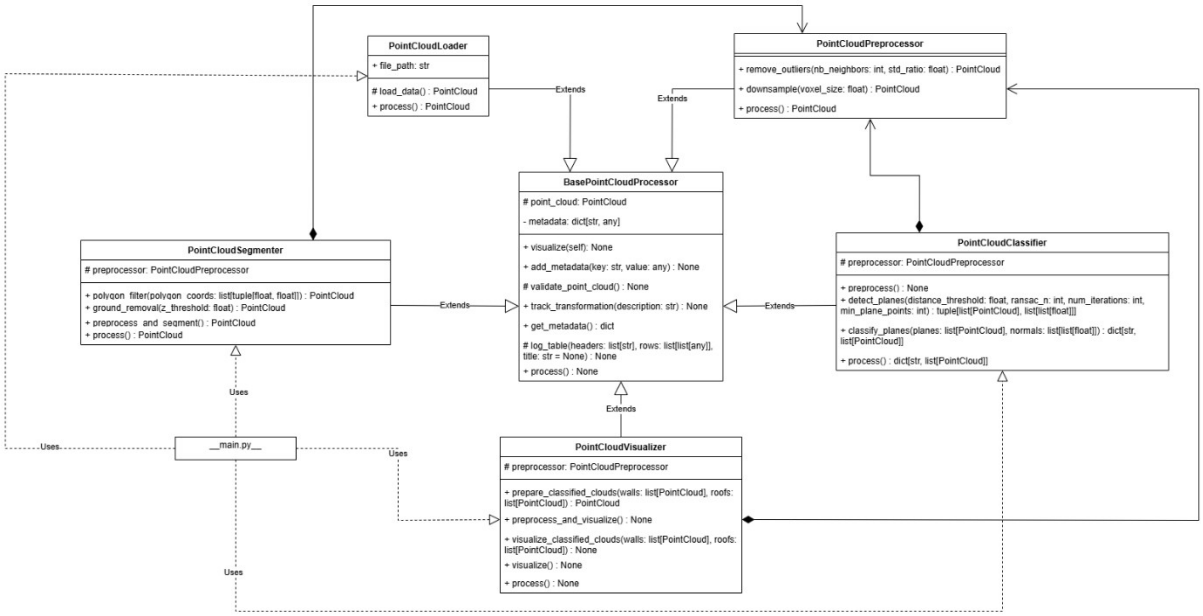

Figure 2 UML diagram of the system architecture.

The pipeline was broken down into the following main components:

1. **Data Loading**: The *PointCloudLoader* class handled the reading of .laz point cloud files. Using the laspy library, the raw data was extracted and converted into an Open3D-compatible point cloud object. This ensured compatibility with the Open3D library for further processing.
    a. **Attributes**:
        i. file_path (str): Path to the input .laz file.
    b. **Methods**:
        i. load_data(): Reads and processes the file into an Open3D point cloud.
        ii. process(): Executes the data loading pipeline.
2. **Preprocessing:** The *PointCloudPreprocessor* class was responsible for cleaning the raw point cloud data. This was done through statistical outlier removal to filter noise and voxel grid downsampling to reduce the point cloud density.
    a. **Methods**
        i. remove_outliers(nb_neighbors, std_ratio): Filters noisy points using statistical analysis.
        ii. downsample(voxel_size): Reduces data density using voxel grid filtering.
        iii. process(): Runs the preprocessing steps sequentially.
3. **Segmentation**: The *PointCloudSegmenter* class isolated building structures from the preprocessed data. This involved polygon-based filtering to retain points within a specific spatial extent and ground removal to eliminate points below a height threshold.
    a. **Methods**:
        i. polygon_filter(polygon_coords): Applies spatial filtering using a user-defined polygon boundary.
        ii. ground_removal(z_threshold): Removes ground-level points based on Z-coordinate values.
        iii. preprocess_and_segment(): Combines preprocessing and segmentation tasks.
4. **Classification**: The *PointCloudClassifier* detected and classified planar surfaces into roofs and facades. Planes were identified using the RANSAC algorithm, which extracted flat surfaces from the segmented point cloud. The orientation of each plane's normal vector and its height were analyzed to distinguish between walls and roofs.
    a. **Methods**:
        i. detect_planes(): Identifies planar surfaces using the RANSAC algorithm.
        ii. classify_planes(planes, normals, min_roof_height): Classifies planes into walls (vertical) and roofs (horizontal).
        iii. process(): Executes the entire classification pipeline.

5. **Visualization:** The *PointCloudVisualizer* rendered the results for analysis. Classified walls were colored blue, and roofs were colored red to provide clear visual distinction.
    a. **Methods**:
        i. prepare_classified_clouds(walls, roofs): Combines walls and roofs into a single colored point cloud.
        ii. visualize(): Displays the current point cloud.
        iii. visualize_classified_clouds(walls, roofs): Visualizes walls and roofs with assigned colors.

2.3 Pipeline Execution

The workflow proceeded as follows (note that the steps are done in order):

1. The *PointCloudLoader* loaded the raw .laz point cloud file.
2. The *PointCloudPreprocessor* cleaned the data by removing outliers and downsampling.
3. The *PointCloudSegmenter* filtered the data to isolate building structures and removed ground points.
4. The *PointCloudClassifier* detected planar surfaces and classified them into roofs and facades.
5. The *PointCloudVisualizer* rendered the final results for analysis.

Each stage in the pipeline was implemented as an independent class. This ensures the modularity and reusability of each component while also keeping the scalability of the overall project. The detailed execution of these stages, combined with the structured object-oriented design, allowed for an efficient and maintainable point cloud classification process.

The codebase is executed by running the command *'python main.py'* in the command line. The *main.py* then executes each of the step above one-by-one as shown in Figure 3. It is important to first activate the virtual environment containing the required libraries and dependencies of the scripts. The required dependencies are configured in an accompanying *environment.yml* file.

main.py

Scripts > main.py > ...

```python
from point_cloud_loader import PointCloudLoader
from point_cloud_visualizer import PointCloudVisualizer
from point_cloud_segmenter import PointCloudSegmenter
from point_cloud_classifier import PointCloudClassifier


Codeium: Refactor | Explain | Generate Docstring |
def main():
    file_path = "../data/plaza_roma.laz"

    print("\n===== Step 1: Loading the Point Cloud =====")
    loader = PointCloudLoader(file_path)
    raw_cloud = loader.process()

    print("\n===== Step 2: Visualizing the Raw Point Cloud =====")
    visualizer = PointCloudVisualizer(raw_cloud)
    visualizer.visualize()

    print("\n===== Step 3: Preprocessing and Segmenting the Point Cloud =====")
    segmenter = PointCloudSegmenter(raw_cloud)
    preprocessed_cloud = segmenter.preprocess_and_segment()
    visualizer.point_cloud = preprocessed_cloud
    visualizer.visualize()

    print("\n===== Step 4: Classifying Planes (Walls and Roofs) =====")
    classifier = PointCloudClassifier(preprocessed_cloud)
    classified_planes = classifier.process()

    print("\n===== Step 5: Visualizing Classified Planes =====")
    visualizer.visualize_classified_clouds(
        classified_planes["walls"], classified_planes["roofs"]
    )

    print("\n===== Process Complete =====")
    print("Pipeline execution completed successfully.")


if __name__ == "__main__":
    main()
```

Figure 3 Contents of Main.py.

2.4 Important Libraries

Laspy

Laspy was chosen for this project because it is a lightweight and straightforward library specifically designed for reading and writing LAS/LAZ files. It directly supports the LAS specification, making it ideal for working with point cloud data in its native format. Additionally, Laspy has built-in support for compressed LAZ files which are widely used for large datasets eliminating the need for external decompression tools. Its seamless integration with NumPy and Open3D made it an efficient choice for preprocessing and

analysis in this project.

Other libraries were considered such as PyLAS, PDAL, and Open3D. PyLAS, while lightweight and fast, does not natively support compressed LAZ files and requires additional tools for decompression. PDAL is a robust library with advanced functionality, but its complexity and steep learning curve made it less suitable for this project's simpler workflow. Open3D provides point cloud functionality but has limited support for LAS/LAZ formats, requiring additional conversion steps. Laspy was ultimately selected for its balance of simplicity, compatibility, and efficiency.

3. Results & Discussion

The pipeline was tested on the plaza roma point cloud, a subset of Intramuros drone imagery data provided by Datum PH to OpenStreetMap Philippines under the 2021 Drone Imagery Collection Grant by Datum & OSMph.

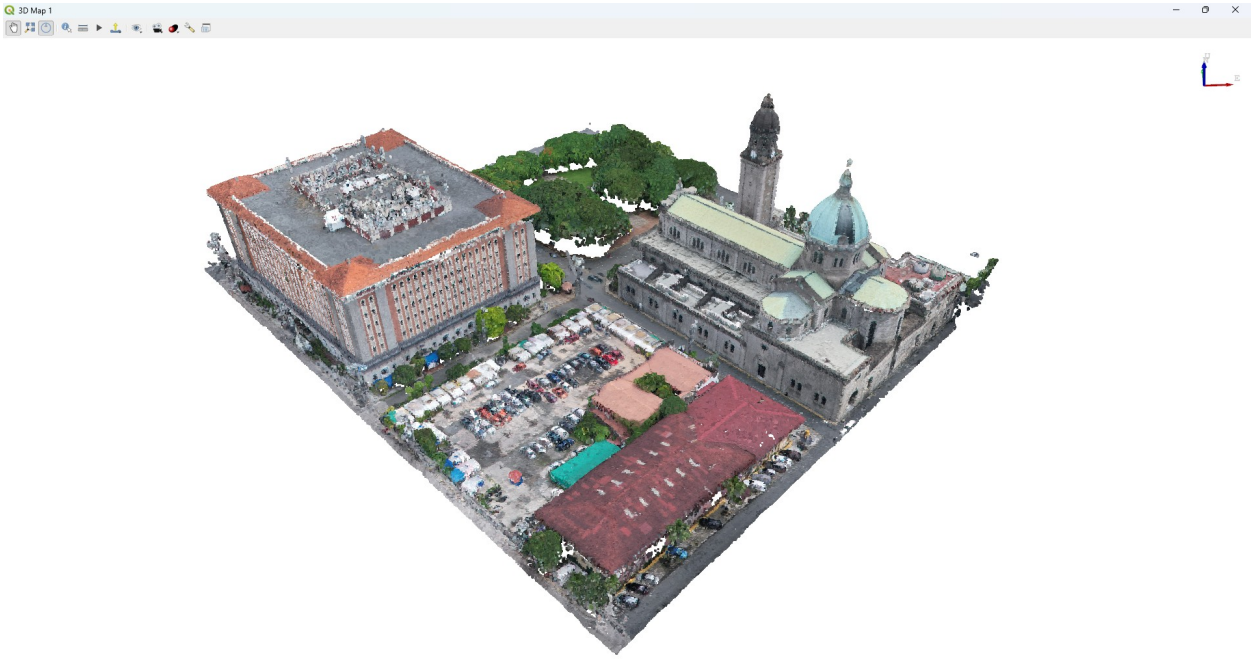

Figure 4 Plaza Roma point cloud viewed in QGIS 3D map viewer.

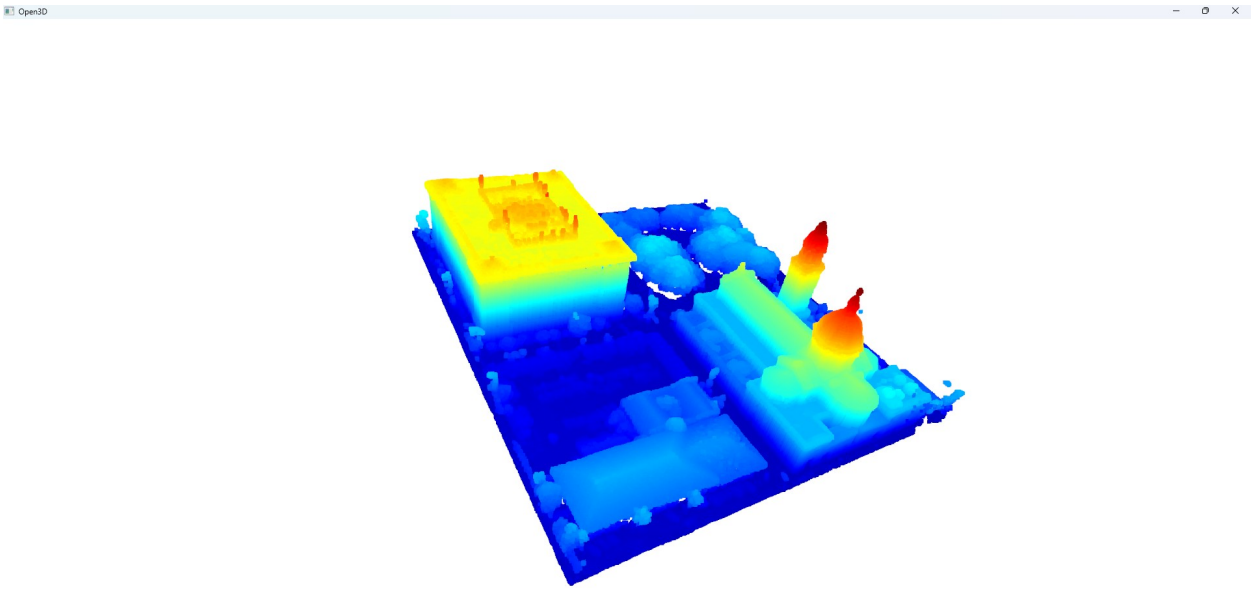

Figure 5 Plaza Roma point cloud viewed in Open3D.

The Plaza Roma dataset originallyy contains 12,394,522 points. After pre-processing and segmenting the Plaza Roma point cloud data, the number of points is significantly reduced to a more manageable level. The statistical outlier removal filter brings the count to just 12,281,477 and the voxel grid downsampling further reduces the number of points to just 2,346,082. It should be noted that this was done to decrease the computational demand of the data albeit at the cost of data integrity and accuracy. However, the scripts can be easily altered to keep more points, if necessary. After these pre-processing techniques, the point cloud was then segmented to only include one building (i.e., the rectangular building on the left in Figure 4) and subjected to a ground removal filter. This then brings down the total number of points to only 600,978.

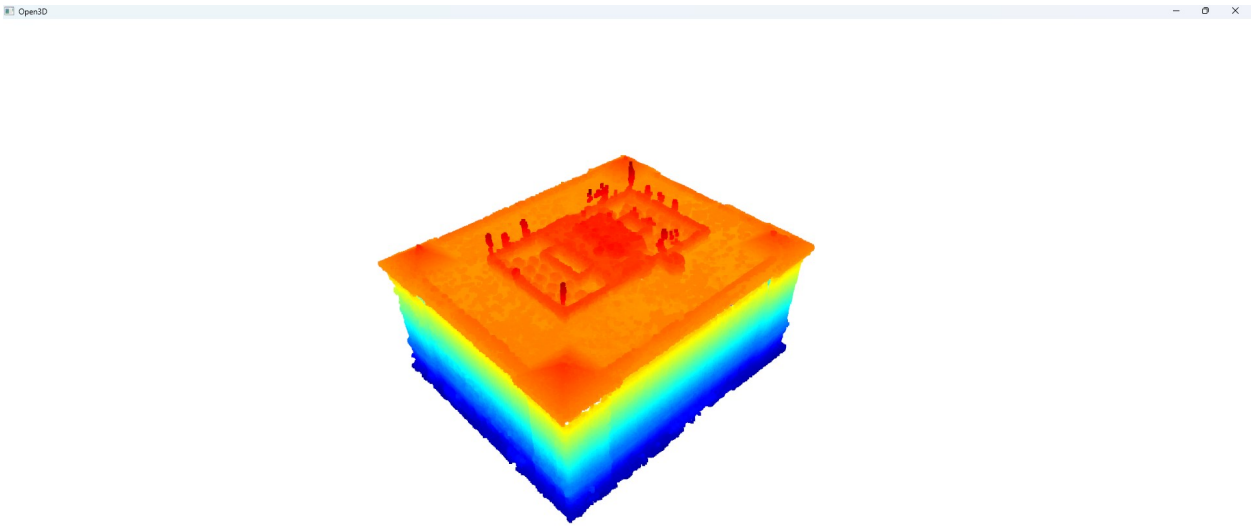

Figure 6 The pre-processed and segmented building point cloud viewed in Open3D.

After subjecting the remaining point cloud data to the classifier using RANSAC algorithm, the roofs and walls are distinguished from each other as shown in Figure 7. The red points were the points classified as roof while the blue ones were classified as walls.

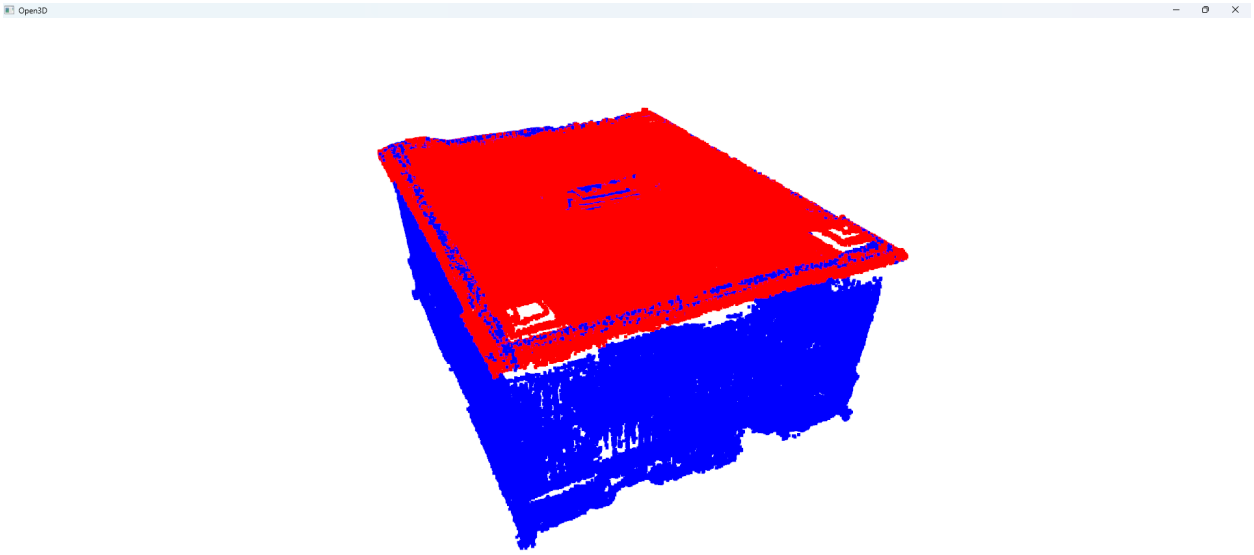

Figure 7 The classified building point cloud viewed in Open3D (red=roof, blue=wall).

The results demonstrate classification of roofs and facades, with effective visualization of the outcomes. In total, 153,442 points were classified walls while 173,960 points were classified as roofs. In terms of the number of planes, key results include:

- Number of detected planes: 20
- Number of classified roofs: 5
- Number of classified walls: 15

The complete detail of the results (which is too long to include without cluttering this document) are printed on the terminal upon executing the codebase. The information included are the number of planes, the number of points per plane, the calculated normal vector/orientation, and the height range of the plane.

The results achieved by this project show some of the advantages of an OOP approach to point cloud classification:

- **Modularity:** Independent classes facilitate easier debugging, testing, and maintenance.
- **Reusability:** Classes can be adapted for similar projects with minimal modifications.
- **Scalability:** New functionalities can be added without affecting existing code.

However, there are still challenges and limitations to this approach. The first challenge is about performance. Processing large point cloud datasets can be computationally intensive and may be unavoidable. Second is about accuracy because the quality of classification depends on point cloud density and noise levels. As seen in the method employed above, the point cloud density was reduced significantly to improve performance but in turn affects the accuracy of the classification. A balance between these two should be explored.

For future work, advanced techniques should be explore for more accurate classification. It may also be beneficial to explore how processing large-scale datasets can be optimized. Finally, and more importantly, future work should look into how the pipeline employed in this project can be extended to include generating CityGML LOD2-compliant 3D models.

5. Conclusion

This project demonstrates a robust, modular pipeline for point cloud classification of roofs and facades using object-oriented programming and python. By providing an efficient and reusable framework, the study contributes to advancing 3D modeling, urban planning, and related geospatial applications. Future work may explore optimizing

computational performance and integrating machine learning techniques for improved classification accuracy.

6. Github Repository

Please visit this github repository to see the working scripts and other details of the project.

https://github.com/jarencecasisirano/pointcloud-oop

References

- Open3D Documentation. Retrieved from https://www.open3d.org/
- laspy Documentation. Retrieved from https://laspy.readthedocs.io/
- Hamid-Lakzaeian, F. (2019). Structural-based point cloud segmentation of highly ornate building façades for computational modelling. *Automation in Construction, 108,* 102892. https://doi.org/10.1016/j.autcon.2019.102892
- Kushwaha, S. K. P., Dayal, K. R., Singh, A., & Jain, K. (2019). Building facade and rooftop segmentation by normal estimation from UAV derived RGB point cloud. *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, XLII-2/W17,* 173–177. https://doi.org/10.5194/isprs-archives-XLII-2-W17-173-2019
- Lyu, W., Ke, W., Sheng, H., Ma, X., & Zhang, H. (2024). Dynamic Downsampling Algorithm for 3D Point Cloud Map Based on Voxel Filtering. Applied Sciences, 14(8), 3160. https://doi.org/10.3390/app14083160