Generation of 3D Reconstructions for AI Algorithm Training

Gerardo Hernandez Widman
Data Engineering
Universidad Politécnica de Yucatán
Km. 4.5. Carretera Mérida — Tetiz
Tablaje Catastral 4448. CP 97357
Ucú, Yucatán. México
Email: 2209090@upy.edu.mx

Jorge Luis Perez Gonzale¡
IIMAS Yucatan, México
Email: JORGE.PEREZ@IIMAS.UNAM.MX

Alan Alejandro García Pastrana
Universidad Politécnica de Yucatán
Km. 4.5. Carretera Mérida — Tetiz
Tablaje Catastral 4448. CP 97357
Ucú, Yucatán. México
Email: alan.garcia@upy.edu.mx

Abstract

This technical report addresses the creation of a new 3D database for training machine learning algorithms and neural networks. Leveraging the RevoPoint Mini and RealSense systems, existing infrastructure and implemented software are utilized to capture diverse and high-quality data. The process involves reconstructing three-dimensional representations, ensuring a comprehensive dataset for training purposes. The combination of RevoPoint Mini and RealSense offers precision and detail in capturing 3D information from various perspectives, facilitated by pre-developed algorithms and specialized software. The resulting database is expected to be versatile, applicable to a wide range of AI applications such as object recognition and 3D image processing. This approach aims to equip machine learning models with enhanced accuracy and adaptability to real-world challenges. The report provides an overview of the strategy employed, emphasizing the significance of RevoPoint Mini and RealSense in generating 3D data for AI algorithm training, contributing to continuous improvement in machine learning models and effective AI application across diverse fields.

Index Terms

Point clouds, Mesh, Normals, DBSCAN, epsilon, euclidean distance, outlier, euclidean distance

Generation of 3D Reconstructions for AI Algorithm Training

I. Introduction

The convergence of three-dimensional data acquisition and artificial intelligence has emerged as a crucial area for advancement in various disciplines. Focusing on the creation and management of a database of three-dimensional models, specifically utilizing PLY files (Polygon File Format), this initiative aims to provide a robust set of three-dimensional data. The primary purpose is to serve as essential input for fueling artificial intelligence algorithms in future developments and applications.

Nowadays, there are several databases of 3D models, which are collections of digital files representing three-dimensional objects. These models can be used for various purposes, including:

- **Design and Engineering**: 3D models are valuable for creating virtual prototypes of products, assisting designers and engineers in testing and refining their designs before actual construction.
- **Entertainment**: 3D models find application in creating video games, movies, and other forms of entertainment.
- **Education**: 3D models serve as educational tools to teach students about various objects and concepts.
- **Research**: 3D models play a role in researching the structure and behavior of objects.

II. Objectives

1) Development of Two Point Cloud Filtering Algorithms in Python: The primary objective is to develop two robust algorithms in Python for filtering point clouds obtained from the Revopoint Mini Scan 3D device. These algorithms will focus on removing noise and redundant data to ensure the accuracy and quality of the resulting 3D models.
2) Creation of a 50 .PLY Database: Another objective is to compile a database consisting of 50 cleaned and processed .PLY files. These files will represent a diverse range of objects and scenes, providing a comprehensive dataset for training and testing AI models.
3) Creation of a report: A report will be created to document the quality of the generated models. This documentation will include the results of comparing the original objects with the scanned models using the Hausdorff distance metric. The README file will serve as a reference for assessing the fidelity of the generated models.

A. Problem Statement

The problem addressed in this technical report revolves around establishing a database of 3D models to fuel artificial intelligence (AI) systems. The process involves utilizing the Revopoint Mini Scan 3D device to capture three-dimensional data and subsequently developing a point cloud filtering code to clean and process this data. Once the 3D models are cleaned and processed, it is necessary to store them in a suitable format for integration into the database.

This problem presents several technical challenges. Firstly, capturing three-dimensional data with devices like the Mini Scan 3D can generate noise and redundant data that need to be filtered to obtain accurate and useful models. Additionally, the point cloud filtering process is crucial for removing or filling any imperfections or errors in the captured data.

Another challenge lies in developing efficient and effective code to perform this filtering automatically, ensuring the consistency and quality of the processed models. Furthermore, the format in which the clean models are stored is important to ensure their accessibility and usefulness in the 3D model database.

B. Justification

The justification for this technical project lies in the growing importance of 3D models across various fields, from manufacturing to medicine and entertainment. With advancing technology, especially in the field of artificial intelligence (AI), the demand for accurate and high-quality three-dimensional data is on the rise.

By establishing a database of clean and well-structured 3D models, AI systems can be fed with reliable data for a variety of applications, such as object recognition, computer vision, simulation, and more. This database can serve as a valuable resource for both academic research and industry, enabling the development and continuous improvement of AI algorithms.

Furthermore, developing an automated process for capturing, cleaning, and storing 3D models can save time and resources, increasing efficiency in creating and managing databases of this kind. This could have a significant impact in sectors where accuracy and speed are critical, such as manufacturing and product design.

III. State of the Art

In the dynamic field of 3D modeling, the availability of extensive repositories plays a pivotal role in facilitating various applications across industries. As technology advances and demands grow, the creation of new 3D model databases becomes increasingly significant, offering fresh opportunities to enhance the accessibility and diversity of digital assets.

ShapeNet stands as an expansive repository, boasting an impressive collection of over 5 million 3D models. Spanning diverse categories such as animals, buildings, furniture, and vehicles, it serves as a comprehensive resource for various applications. One of its significant advantages lies in its accessibility - the database is freely available to all users. However, while ShapeNet offers an extensive array of models, it is essential to note that the quality may vary considerably across the collection, and some models might lack completeness in associated metadata.

The Smithsonian 3D Collection represents a curated selection of high-quality 3D models, meticulously documented and focusing predominantly on objects of American history and culture. Although smaller in scale compared to ShapeNet, this collection provides valuable resources for educational and research endeavors. With free access for all users, it serves as an excellent tool for exploring historical artifacts and cultural heritage. However, users should be aware of its limited size and potential delays in model downloads.

The Stanford 3D Scanning Repository offers a unique collection of 3D models derived from real-world objects, ensuring exceptional accuracy in representation. Despite its smaller size relative to ShapeNet, this repository covers a wide range of categories, including natural objects, artificial items, and human body parts. Its free of charge accessibility to all users makes it a valuable resource for various applications. Nonetheless, users should consider potential limitations such as file format compatibility issues and the accuracy of associated metadata.

IV. Methods and Tools

To establish a good production of reconstructions, there has to be a combination of RevoScan 5 for data gathering and Open3D python library for data filtration.

In this exploration of methods and tools, we delve into the intricacies of 3D data processing, examining techniques for noise reduction, feature extraction, and meshing. From advanced filtering algorithms to sophisticated visualization methods, we uncover the arsenal of tools at the disposal of researchers and practitioners in the field.

Furthermore, we explore the role of information visualization techniques in conveying insights derived from 3D data, shedding light on strategies for intuitive data representation and analysis. Through a comprehensive examination of methods and tools, we aim to illuminate the multifaceted process of transforming raw scans into actionable insights and immersive visualizations.

A. Revoscan5

Revoscan5 is a 3D scanning software developed by Revopoint 3D. It is a versatile tool that enables capturing real-world objects with high precision and turning them into digital 3D models. Revo Scan 5 is compatible with the next equipment:

Miniscan 3D: compact and lightweight desktop 3D scanner that can be used with Revo Scan 5 to capture small objects with high precision. features:

- Accuracy: Up to 0.05 mm
- Resolution: Up to 0.1 mm
- Scanning Volume: 100 x 100 x 100 mm
- Easy to use: Plug-and-play, ideal for beginners

The Dual Axis Turntable: motorized turntable that can be used with Revo Scan 5 to automate the data capture of 3D objects. It allows for:

- Automatic rotation: Rotates the object in two axes for a complete 360-degree capture.

- Precise control: Adjust the rotation speed and direction to optimize the scanning process.
- Increased efficiency: Saves time and effort by capturing data automatically.

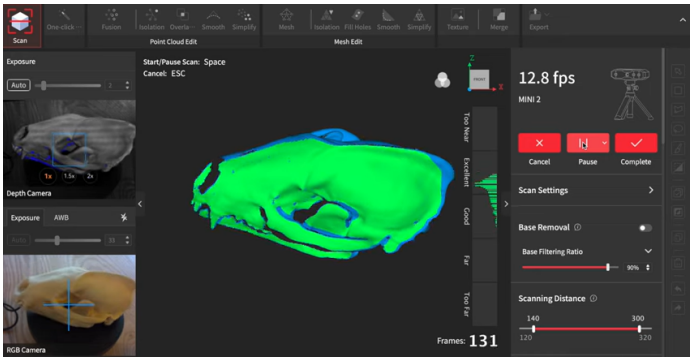
Capturing cloud points relies heavily on the intricate process of laser triangulation. This advanced technique forms the backbone of our data acquisition method.

The scanner emits a precisely calibrated laser pattern onto the object under scrutiny. Simultaneously, two high-resolution cameras meticulously capture images of this projected pattern, accounting for any distortions caused by the object's surface. The software then meticulously analyzes these images, meticulously calculates the precise distances between the scanner and various points on the object's surface.

To ensure comprehensive coverage, our scanning setup incorporates a dual-axis turntable. This turntable facilitates seamless movement in every direction, allowing us to capture the object from multiple angles and perspectives. This dynamic movement capability ensures that we obtain a comprehensive set of data points, capturing the intricate details of the object's surface from all possible vantage points.

Together, these sophisticated components and processes form a robust system for capturing cloud points with unparalleled precision and accuracy. By leveraging laser triangulation and advanced motion control technology, we can confidently generate detailed and reliable digital representations of objects, enabling a wide range of applications in fields such as manufacturing, design, and research.

In the Revoscan5 software, precise measurement of RGB and light exposures during scanning is paramount. While initially, appropriate values may be selected for scanning, it is crucial to be prepared to adjust them independently as per the specific needs of each scenario. In special situations, these adjustments may necessitate constant changes during the scanning process.



The rationality behind these continuous adjustments lies in the need to maintain an optimal balance between light exposures and color levels. When dealing with extremely dark or light colors, the disproportion between them can lead to inconsistencies in scanning. Altering the light intensity and color values affects the color perception in the scan. Continuing scanning may cause the software to interpret these changes as new colors, resulting in unwanted combinations.

Another challenge arises in manipulating objects during scanning. The Mini Scan3D device has limitations regarding its detection range, meaning it will lose details if the object is too far away. This limitation becomes even more apparent when attempting to scan tall or detailed objects. Often, we have to approach strategies such as placing the object horizontally or laterally to ensure all necessary details are captured.

However, despite investing considerable time in the scanning process, it is important to recognize that inevitable failures may arise. These imperfections can be rectified through adjustments in the code, but they can still affect the overall quality of the scan. An additional challenge arises as the scanning progresses: the accumulation of points can become confusing, making it difficult to interpret the data. Merging

these points is a critical step in correcting this issue, but unfortunately, this process cannot be fully automated through code. The program requires manual completion before allowing the download of scanned data. This means that only after scanning and point fusing are completed can errors be identified and addressed, although often, at this point, it may be too late for significant corrections.

B. PLY

After the scan and fusion with the 3D mini scanner, you receive a file with the extension .PLY. This stands for Polygon File Format and is a versatile way to store information about 3D objects. One of its most important uses is for storing point clouds. These are collections of points that represent the surface of an object in 3D space.

A .PLY file is made up of two main parts, first we have the header, this contains information about the structure of the file, such as the number of points, the data type used for coordinates, and any additional properties stored for each point, and then we have the data part, this section stores the actual coordinates of each point in the cloud, along with any additional properties mentioned in the header. Storing Point Clouds in PLY files as a list of X, Y, and Z values

Example:

-0.12345	0.67890	1.23456
-0.45678	0.98765	2.34567

Beyond just the coordinates, PLY files can also store additional information for each point, such as:

- **Color**: This can be specified as RGB values or a grayscale value.
- **Intensity**: Represents the strength of the signal reflected at that point, often used in laser scanning data.
- **Normal**: Vector that Indicates the direction perpendicular to the surface at the point.
- **Classification**: Categorizes each point into different classes, like ground, vegetation, or buildings.

There are two main types of .PLY files that could be employed during any of the processes:

- **PLY ASCII**: This format is human-readable but takes up more storage space.
- **PLY Binary**: This format is more compact but requires specific software to interpret.

In the case of RevoScan 5, we would be working with binary PLY files because RevoScan 5 utilizes this specific format to store the data it captures from your 3D scans. This format offers several advantages that make it well-suited for the RevoScan workflow.

C. Open 3D

Once the scan has been finished and gets our hands in ply, the next critical step is to cleanse the data. While Revonscan5 boasts a comprehensive process capable of handling various functionalities, it falls short when encountering objects in less-than-ideal environments. To address such situations, the database incorporates two distinct cleansing codes, Specifically for this purpose. We are utilizing Open3D, which is a modern library for processing 3D data. It offers a comprehensive set of tools and functionalities tailored for working with various types of 3D geometric data, such as point clouds and meshes.

Open3D is designed to be efficient, user-friendly, and suitable for a wide range of applications involving 3D data processing, analysis, and visualization. The next codes function as a fail safe, ensuring that regardless of the condition of the scanned objects, the data undergoes through purification, readying it for further analysis or application. This dual approach enhances the versatility and reliability of the scanning system, providing robust performance across a spectrum of scenarios and environmental conditions.

To make use of open3D we have to use a virtual environment in python in any kind of coding framework as visual studio code and then use the command “pip install open3d”. For further improvements and versatility here is a list of libraries needed:

- **Matplotlib**: It can be used to create various types of plots and visualizations, which can be helpful for analyzing data processed with Open3D.
- **NumPy**: provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays. Open3D often uses NumPy for handling data arrays.
- **Scikit-learn**: A machine learning library for Python that provides simple and efficient tools for data mining and data analysis. You can use scikit-learn in conjunction with Open3D for tasks such as feature extraction, dimensionality reduction, and classification on 3D data.

From these libraries there will only be the necessity of derivative libraries and functions, mostly for specific tasks or visualization that have no effect on the data, but still useful when checking the state at which data goes through.

D. Downsampling

The first step when handling point clouds will always be downsampling, expecting data to be massive it is an understatement, most data in the database will have over 7 million vertices, in other words, over 7 million rows with 3 columns per object, and that only if we look to preserve form but not RGB of the vertex and Normals that that will be used for further processing, so trying to analyze point by point will end up being eternal, that is why in Open3D exists downsampling, which refers to techniques that reduce the number of points in a point cloud. This is useful for processing large point clouds which can be computationally expensive. There are 3 different approaches:

- **Random Downsampling**: This approach randomly selects a subset of points from the original point cloud to keep. It is a quick and easy method, but it can introduce a bias in the representation of the data, especially for features with low density.
- **Uniform Downsampling**: This approach keeps a fixed number of points uniformly distributed throughout the point cloud. This ensures a more balanced representation but may not be suitable for all scenarios, particularly for non-uniformly distributed data.
- **Voxel Downsampling**: This approach divides the space containing the point cloud into small cubes called voxels. Then, it keeps only one point from each voxel that contains at least one point from the original data. This method is useful for preserving the overall shape of the object while reducing the number of points significantly.

To find a compelling choice for building efficient and effective point cloud reconstruction databases, particularly when dealing with large-scale datasets. There is a need to look for advantages stemming from its ability to strike a balance between preserving essential structural information while significantly reducing the computational burden.

Preserving Shape and Topology: capturing the overall shape and topology of the object. This ensures that the reconstructed model retains the basic structure and form of the object, even when the number of points is substantially reduced.

Noise Reduction: by reducing the points we look to make noise as light as possible so it can be analyzed easier

Voxel downsampling comes as the preferred choice for building large-scale point cloud reconstruction databases due to its effectiveness in preserving structural information, reducing computational complexity, mitigating noise, enhancing memory efficiency, and promoting scalability. It strikes a favorable balance between data fidelity and computational efficiency, making it an invaluable tool for managing and analyzing extensive point cloud collections.

E. Filtering methods

Even with the downsampling, data will be too heavy to simply analyze, if we take a one path solution, we will end up with empiric codes that will take over 40 minutes to accomplish and may fail due to the amount of unstable data each object provides, Another problem is the information residing inside each point cloud, which are borderline unusable, as any ply stands, there is only one cloud, one set of data, and with every scan, all data changes, space, density, noise, form, colors, center, normals, and size, there is not a predictable factor that can be acquired from it, this is important to remark, because one will always search to do formulas with some sort of average, standard deviation, or threshold distance but will utterly fail unless there is a combination of techniques and analysis

The next list provides some of the different ways that were used to analyze this specific line of point clouds, and even if they were not useful here, they could be for different scenarios, at the end of the day when one searches for point clouds on the internet, we are welcomed with images that focus on forms rather than conserving as many details possible form of real-life objects

POINT REMOVAL

- **Radius Outlier Removal**: requires two parameters: nb_points, which is the minimum number of points within the specified radius for a point not to be considered an outlier, and radius, which defines the radius within which neighboring points are considered for outlier removal. The algorithm iterates through each point in the point cloud and counts how many neighboring points are within the specified radius. If the count is below the threshold (nb_points), the point is considered an outlier and removed from the cloud

```
pc.remove_radius_outlier
(nb_points=20, radius=0.1)
```

- **Statistical Outlier Removal**: requires two parameters: nb_neighbors, which specifies the number of neighbors to consider for calculating the mean distance and standard deviation, and std_ratio, which is the standard deviation ratio. Points with distances beyond this ratio multiplied by the standard deviation are considered outliers. The algorithm calculates the mean distance to neighbors for each point based on the specified number of neighbors (nb_neighbors) and computes the standard deviation of these distances. If the distance between a point and its neighbors exceeds a certain multiple of the standard

deviation (std_ratio), the point is considered an outlier and removed

```
pc.remove_statistical_outlier
(nb_neighbors=20, std_ratio=2.0)
```

CROPPING

- **AABB Cropping**: selects a region defined by a box aligned with the coordinate axes (X, Y, Z). You specify the minimum and maximum coordinates of the box to define the cropping region.
- **Bounding Sphere Cropping**: selects a region defined by a sphere with a specified center point and radius.
- **Bounding Cylinder Cropping**: selects a region defined by a cylinder with a specified axis, center point, radius, and height.
- **Half Space Cropping**: selects a region defined by a plane that divides space into two half-spaces. You specify the coefficients of the plane equation to define the cropping region.

SEGMENTATION

- **Region Growing Segmentation**: groups neighboring points with similar properties into regions. The `cluster_dbscan` function can be used for region growing segmentation. It requires parameters such as eps, which defines the maximum distance between points to be considered in the same cluster, and min_points, which specifies the minimum number of points required to form a cluster. This algorithm iterates through each point in the point cloud and forms clusters by expanding regions based on the similarity of neighboring points. The resulting clusters represent segmented regions within the point cloud.

  labels = $\texttt{source}_{p}c.cluster_{d}bscan(eps = 0.02, min_{p}oints = 10)$

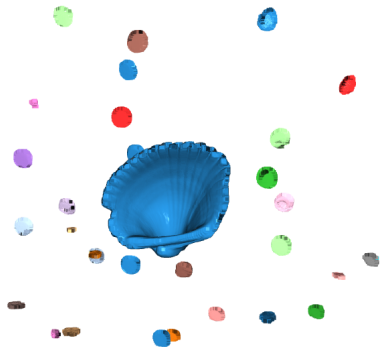
- **Plane Segmentation**: The `segment_plane` function is used for this purpose. It requires parameters such as distance_threshold, ransac_n, and num_iterations. The algorithm iteratively fits planes to subsets of the point cloud data using the RANSAC algorithm. The parameters control the maximum distance from points to the plane (distance_threshold), the number of points required to fit a plane (ransac_n), and the number of RANSAC iterations (num_iterations). After fitting, it returns the equation of the plane and the indices of the points classified as inliers. These inliers represent the points that belong to the identified planes, which can then be separated from the rest of the point cloud.

inliers = $\texttt{source}_{p}c.segment_{p}lane(distance_{t}hreshold = 0.01, ransac_{n} = 3, num_{i}terations = 1000)$

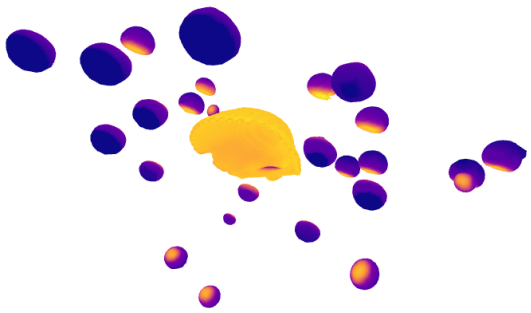
ANALYSIS AND MANIPULATION

This part rather than extending filtering methods is a continuation to segmentation or any other way of clustering, filtration, or point removal, as once the point clouds are divided or edited, there is certain data that can be analyzed and seen to avoid errors or to understand how to search the correct focus on the problem.

- **Color Clustering**: We then apply DBSCAN clustering (model.fit) to cluster the points based on their proximity in the feature space. After clustering, we map the cluster labels to colors using a colormap (plt.get_cmap) and assign these colors to the points in the projected point cloud. Finally, we visualize the clustered point cloud using o3d.visualization.draw_geometries.

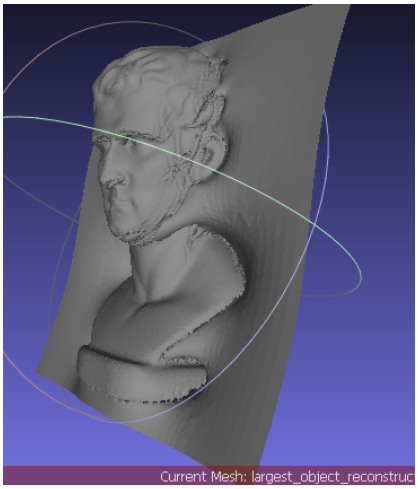


- **Density Visualization**: using Poisson surface reconstruction. We create a triangle mesh from the point cloud using These colors are assigned to the vertices of the mesh, and the density mesh is visualized using o3d.visualization.draw_geometries.



- **Normal Visualization**: we visualize the normals of the points in the point cloud. We simply use o3d.visualization.draw_geometries with the point_show_normal=True parameter to visualize the point cloud with normals displayed.

This is very important to check at any step, not realizing the normals have changed will result in wrong meshing, an important process that will be discussed later, but the results with altered normals would look like this:



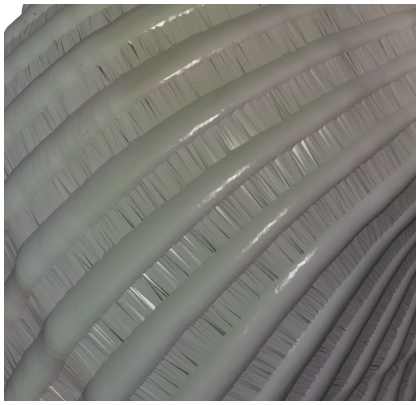
And finally, data information, that can be acquired through data lecture:

- **density** = `len(pcd.points) / pcd.get_volume()`
- **std_dev_distance** = `np.std(np.linalg.norm(np.asarray (pcd.points) - centroid, axis=1))`
- **surface_area** = `mesh.get_surface_area()`
- **volume** = `mesh.get_volume()`
- **curvatures** = `mesh.compute_vertex_curvature()`

F. Meshing

The last step will always be to transform the point cloud with meshing, essentially taking a collection of connected vertices and form triangles or other polygonal shapes to create a surface, this is principally used not only to allow the point clouds to be solidified but reconstruct surface parts that cannot be scanned in any possible way, there are only 3 different ways and each will be tested with the use of a shell scan:

Alpha Mesh: This approach iteratively creates a series of progressively finer triangle meshes based on an alpha value that controls the detail level.



This was the first meshing method ever made, it is not prepared for extremely dense point clouds, as it cannot imitate the textures of a shell, which is usually smooth, instead, it does multiple strips in attempt of a successful result

Ball Pivoting Mesh: This method builds the mesh by considering the local neighborhoods around each point defined by a set of radii. It iteratively connects points within these neighborhoods to form a surface.

When using this method, we can see an effective meshing and generation of triangulation; however, it is not able to reconstruct other than minuscule holes, leaving human error unhideable, with the multiple situations that could

happen, only perfection could make this method fit our necessities.

Poisson Mesh: This technique utilizes the Poisson equation to reconstruct a smooth surface that approximates the underlying point cloud. The depth parameter controls the smoothness of the resulting mesh.

Finally, Poisson mesh unites both strengths as it can reconstruct using the normal of the vertices in an outstanding, mostly predicting forms and colors and giving a good amount of triangles, giving the exact texture of the scanned mesh.

It is no surprise why Poisson mesh is a need for the filtration and treatment of point clouds, as it is the most recent and was made using both of its predecessors as the base to achieve its effectiveness.
To use this function we need to establish the boundaries, the value of depth in the next code function cannot be lower than 2 because the Poisson surface reconstruction algorithm utilizes a data structure known as an octree to represent the three-dimensional geometry of the input point cloud.

```
mesh, densities =
open3d.geometry.TriangleMesh.create_
from_point_cloud_poisson(point_cloud,
depth=9)
```

This octree divides the three-dimensional space into smaller cells, allowing for an efficient and scalable representation of the geometry. The depth of the octree specifies how many recursive subdivisions are made in each dimension of the three-dimensional space.
When the depth of the octree is less than 2, it means that there is only one subdivision in each dimension, resulting in a very basic and detailed representation of the geometry. This is not sufficient for performing an accurate surface reconstruction, leading to an error.

And even if we try to give an input of 2, the results would be insufficient at best, as the next example shows it:
The value of 9 is set as default from the Open3D documentation website and seems to be enough to capture most of the details and form like in the previous photos, but if we were to increase the value, the processing time has gargantuan increases of processing time and will show to minimal to no changes in the final results. However, it is still plausible to increase the number if wanted; parameters with the value of 10 to 14 do not have a big enough spike to consume all the computer processing power.

Fig. 1. poisson with depth=2

V. Development

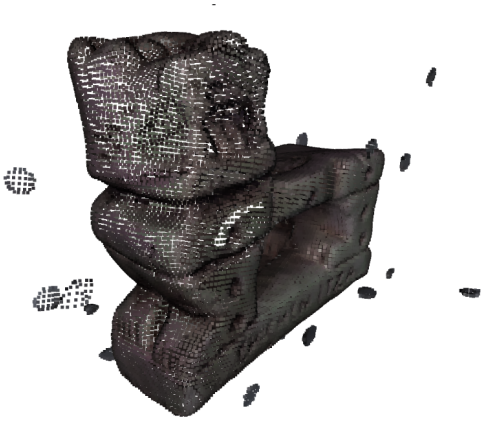
Processing this data represents a relatively novel challenge within the realm of documentation. Historically, there has been a lack of established methodologies for filtering extremely dense point clouds, leaving practitioners with limited options. Consequently, the development of intricate solutions involving multiple steps has emerged as the primary approach to address this gap in processing techniques. This necessity for complex workflows underscores the evolving nature of 3D data processing and highlights the ongoing quest for innovative solutions to effectively handle dense point cloud datasets.

A. Filter 1

Setting Up: The code begins by importing the two essential libraries: open3d for 3D data processing and visualization, and numpy for numerical operations on arrays.

```
import open3d as o3d
import numpy as np
```

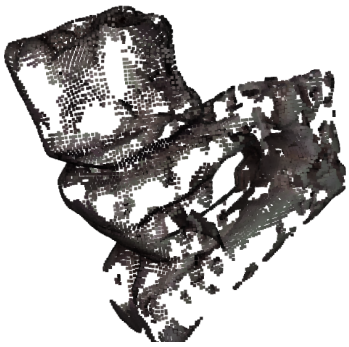
Downsampling: To reduce the density of the point cloud as much as possible, we use voxel downsampling to its max potential. The downsampled point cloud is stored in downsampled_pcd.



Outlier Removal: First, there has to be Radial outliers removed from the downsampled point cloud using the remove_radius_outlier method. This method takes two parameters: nb_points specifies the number of neighboring points, and radius specifies the maximum distance for points to be considered inliers.

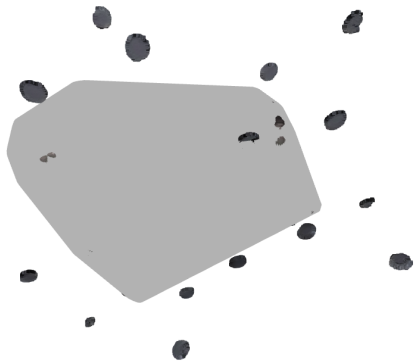
Next, statistical outliers are removed from the point cloud with radial outliers removed. This is achieved using the remove_statistical_outlier method, which requires two parameters: nb_neighbors, the number of neighbors to analyze for each point, and std_ratio, which controls the threshold for considering a point as an outlier.

Once this process has passed the point cloud is massacred, as any detail other than the basic form is left, and it may seem as bad, but it is all we need.



Convex Hull Computation: We talked about bounding boxes, a cropping technique that is useless due to the inclination factor, as mentioned before, each time a ply is gathered everything changes, including the inclination, we can try adjusting the angle but will end up in everlasting codes, that is why we need a convex hull, a 3D geometric shape that tightly encloses a set of points.

The convex hull is an object by itself, as it does not need the filtered point cloud anymore, that is why it can be used to visualize the volume it creates in a space, but even when computed, it has to be expanded, because it generates under the limit point, finally, we can visualize both the original pc and the ant filtered convex to see what would be extracted when applied.



B. Parameters 1

voxel_size: Adjust voxel_size based on the level of detail you want to capture in the final mesh. A smaller size preserves finer details but increases processing time.

```
(voxel size=0.9)
```

We need the number as high as possible, which is 0.9, to make further calculations as easy as possible.

Outlier removal parameters: The next numbers were selected as an overestimation. At the end, all we need are the edges to survive from the removals, as we want the heavy noise to be cut out. The remaining point cloud after this will have no value.

```
 nb_points=100, radius=5,
nb_neighbors=50, std ratio=0.01.
```

Expand factor: This scaling operation can be useful in various applications, such as creating a buffer around the original point cloud data or ensuring that certain objects fit within the convex hull with some margin. Adjusting the expand_factor allows control over how much to enlarge the convex hull.

```
(expand_factor = 1.03)
```

It means that the convex hull will be scaled up by a factor of 1.03. So, each coordinate of the convex hull's vertices will be multiplied by 1.03, effectively expanding the hull's size by 3 percent, enough to cover all points the hull was formed with.

C. Filter 2

Setting Up: The code starts by importing several libraries: numpy for numerical operations, open3d for working with 3D data, DBSCAN for data clustering from scikit-learn, and StandardScaler for data normalization from scikit-learn.

Clustering: The apply_dbscan method uses a clustering algorithm called DBSCAN to identify groups (clusters) of points that are close together in space. It takes two parameters: eps (maximum distance between points in a cluster) and min_samples (minimum number of points required to form a cluster). The method assigns a label to each point, indicating which cluster it belongs to (stored in self.labels).

Finding the Largest Cluster: After clustering, the find_largest_cluster method identifies the cluster with the most points. It analyzes the labels assigned by DBSCAN and determines which cluster has the highest number of points. It then extracts the points and their corresponding indices belonging to the largest cluster for further processing.

Generating the Mesh: The generate_mesh method takes the points from the largest cluster and uses a technique called Poisson surface reconstruction to create a 3D mesh. This mesh represents a surface that approximates the shape of the point cloud data. The detail level of the mesh can be controlled by a parameter called depth.

Visualization: The visualize_mesh method allows you to see the final, smoothed mesh using Open3D's visualization tools. This helps you inspect the outcome of the processing pipeline.

Saving the Mesh: Finally, the save_mesh method saves the smoothed mesh as a new file. It creates a filename based on the original point cloud file name but replaces "_pc" with "_mesh" to indicate the mesh format.

D. Parameters 2

voxel_size: Adjust voxel_size based on the level of detail you want to capture in the final mesh. A smaller size preserves finer details but increases processing time.

```
(voxel size=0.001)
```

This parameter perhaps is the one with the most dire consequences in altering the filter method, starting for the processing time, the number can be adjusted to a bigger one if the user does not search for a great quality form, but it here it is left as small as possible to preserve as much.

eps and min_samples: Experiment with eps and min_samples to achieve the desired granularity of groups. A smaller eps and a larger min_samples can create tighter groups, while a larger eps and a smaller min_samples can lead to fewer, larger groups.

```
(eps=0.02) (min samples=2)
```

The min samples remain with the value of 2 to catch every single vertex in the point cloud, higher number would mean larger clusters in general but there will be lost of info, also it cannot be lower than 2, otherwise, you would get a cluster for each vertex detected.

depth: The optimal value of depth for mesh generation depends on the complexity of the point cloud and the desired mesh detail. A higher depth creates a more detailed mesh but takes longer to compute.

```
(depth=9)
```

It is the default value and has been proven to be more than enough to keep most details; this has been proven through Hausdorff distance between ply archives with different depth levels.

Iterations: The number of smoothing iterations depends on the desired smoothness level for the final mesh. More iterations result in a smoother mesh but can also remove some sharp features.

```
(iterations=3)
```

And to finish, it is rather one's judgment whether to increase it or decrease it, as it barely affects the processing time and can lead to nice and smooth models. This process is rather useful when working with meshes like chess pieces where you want to copy its texture, but when someone works with porous surfaces and wants to keep them that way we use a low level of iterations, like the one in the code.

E. Validation

Models for measuring the proximity between two sets of points are fundamental tools in point cloud processing. These models allow the evaluation of the quality of a 3D model, the identification of differences between two surfaces, and the determination of the similarity between two datasets. Of course, we have to acknowledge the fact that the 2 datasets we will have to compare are the real-life one and our scans.

To measure how close our models are to the actual figures, we use the following methods:

Hausdorff Distance (HD): It measures the maximum distance between two sets of points. It is a simple, intuitive, and easy-to-compute method but sensitive to outliers and does not consider direction.

Modified Hausdorff Distance (MHD): Combines HD with the mean distance to obtain a more robust measure and less sensitivity to outliers.

Chamfer Distance: Calculates the average distance between two sets of points, considering the direction. It is more accurate than HD in some cases but also computationally more complex.

Point-to-Point Error (L2): Measures the Euclidean distance between each point in the model and its closest point on the reference surface. It is a simple and accurate method but does not provide information about the distribution of differences.

Curvature-based metrics: Compare the local curvature of the model surface with the curvature of the reference surface. They are useful for evaluating the accuracy of curved and complex shapes.

Point density-based metrics: Compare the point density between two sets. They are useful for evaluating the uniformity of the point distribution in the model.

F. Hausdorff Distance

The main reason we use Hausdorff Distance for measuring proximity in point clouds is its simplicity, efficiency, and interpretability. This makes it easy to understand, implement, and interpret the results. Unlike some more complex methods, this model has a low computational cost, making it ideal for real-time applications or those with limited resources.

Now, as any model that exists, it has flaws, like it can be sensitive to outliers, meaning a single misplaced point can significantly distort the result. Additionally, HD doesn't consider the direction in which points deviate from the reference surface. Changes in the model structure, like holes or edges, can also affect the accuracy.

This is where the code's strength lies. By isolating the clusters or statistically removing the outliers and performing Poisson surface reconstruction, we can analyze the object's geometry in a more comprehensive way, overcoming the limitations.

G. Calculation of Hausdorff

To calculate the approximate Hausdorff distance between two meshes, this algorithm works as follows:

1. Nearest Neighbor Search: For each vertex v_i in the first mesh, MeshLab finds the closest vertex v_j in the second mesh using a k-nearest neighbors (KNN) search algorithm. The KNN algorithm searches for the k most similar points in a dataset to a given query point. In this case, the query point is v_i and the dataset is the set of all vertices in the second mesh.

2. Distance Calculation: Once the nearest neighbor v_j for each vertex v_i has been found, MeshLab calculates the distance between v_i and v_j. This distance is typically calculated using the Euclidean distance formula.

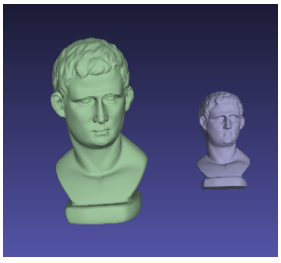
$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

3. Hausdorff Distance Estimation: The Hausdorff distance between the two meshes is then estimated as the maximum distance between any pair of corresponding vertices. In other words, the Hausdorff distance is the largest value of $d(v_i, v_j)$ for all pairs of vertices v_i and v_j in the two meshes.

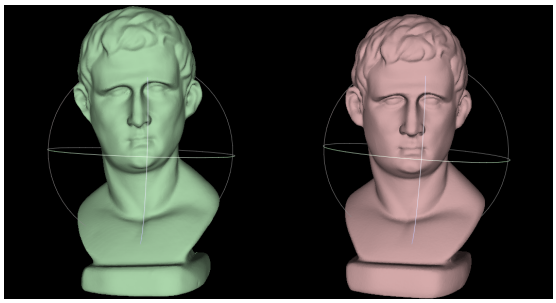
H. MeshLab

MeshLab is an open-source software designed for 3D mesh processing and visualization, offering an extensive array of tools tailored for working with point clouds. Among its functionalities are mesh generation, editing capabilities, and analytical tools. But now we will only delve into employing MeshLab to quantify the similarity between two point clouds using the Hausdorff Distance. Before any calculations, it is important to prepare the point clouds to ensure the accuracy of the results. This involves two main steps:

Scaling: Normalizing the coordinates of the point clouds to be on the same scale. This ensures that distance measurements are consistent and comparable.



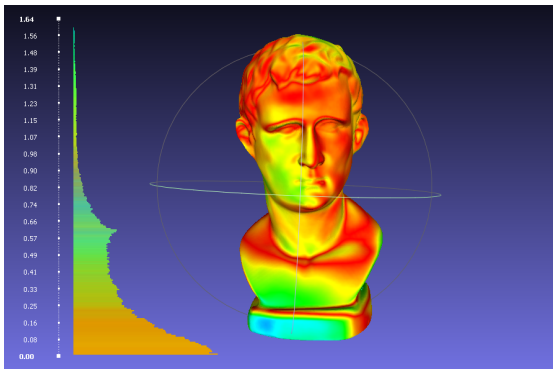
Alignment: Aligning the point clouds to be in the same coordinate system. This facilitates direct comparison between the point clouds.



Once the point clouds are processed, the Hausdorff Distance can measure the similarity between the point clouds in two different ways:

Numerical Representation: The calculated Hausdorff Distance can be displayed as a numerical value. This provides a basic understanding of the similarity between the point clouds and can be shown as a histogram beside the chosen 3D model.

Color Mapping: By visualizing the Hausdorff Distance as a color map on the point clouds, patterns of similarity and difference can be identified. Typically, a color range is used where shades of blue indicate proximity (shorter distances) and shades of red indicate distance (longer distances).



VI. Results

A. Unfiltered Models

The state in which objects emerge from scanning, despite concerted efforts, is often less than perfect due to inherent challenges associated with the RevoPoint Mini scanner. The presence of noise and complications arising from the diverse gamut of colors in objects frequently results in blended tones within the scanned output. Despite exhaustive attempts to capture darker areas, certain points may still elude capture. These limitations are an unavoidable aspect of scanning with the RevoPoint Mini, necessitating meticulous attention and extensive post-processing to attain satisfactory results.

And as I mentioned before, the software can not completely fix all the imperfections inherent to the RevoPoint Mini scanning process.

Noise Reduction: RevoScan 5 has tools for removing noise, but overuse can affect the model's surface quality. In some cases, noise may be difficult to eliminate entirely without impacting the model's geometry.

Recovering Details in Dark Areas: RevoScan 5 has tools to enhance visibility in dark areas, but they can't always fully recover details lost during scanning. In some cases, dark areas may remain incomplete.

Scan Alignment: The software offers tools to align and combine multiple scans, but this process can be complex and error-prone, especially with large or intricate objects, and trying to fix this issue often brings more errors or the total loss of the scan.

B. Filtered Models

It is thanks to our codes that we can achieve the next points so our models reduce to:

Noise Reduction: A careful noise reduction approach can be implemented to avoid compromising the model's surface quality. This may involve moderate use of the noise removal tools available in RevoScan 5, bearing in mind that excessive use could negatively affect the model's geometry.

Recovering Details in Dark Areas: While RevoScan 5 has tools to improve visibility in dark areas, it's important to recognize that it may not always be possible to fully recover details lost during scanning in these areas. Additional approaches, such as alternative lighting techniques during scanning, may be needed to enhance detail capture in dark areas.

Scan Alignment: The process of aligning and merging multiple scans can be complex and error-prone. It's important to acknowledge that addressing this issue may require a more careful and manual approach, especially when dealing with large or intricate objects. In some cases, additional post-processing techniques or even manual alignment may be needed to avoid introducing additional errors.

And even if models have most of these issues, results are mostly homogeneous, uniform, and when scans are good, impeccable like the following examples:

C. Hausdorff Results

One has to understand that both codes work equally effectively so this means that through the filtering, both codes maintain the same amount of vertex of the point clouds, so there is no reason to make more than one Hausdorff distance due to the fact that both are basically the same.

As mentioned before, it is really hard to keep a stable number when scaling and aligning so there is no stable answer, but through many attempts we got this result with both of the filtered models compared to the real-life digitization:

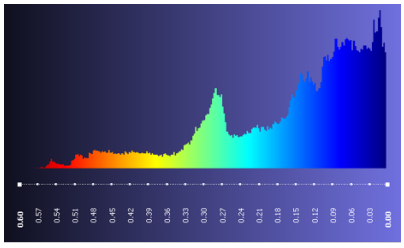
min : 0.000002
max : 0.599495
mean : 0.155529
RMS : 0.201905

Values w.r.t. BBox Diag (86.938652)

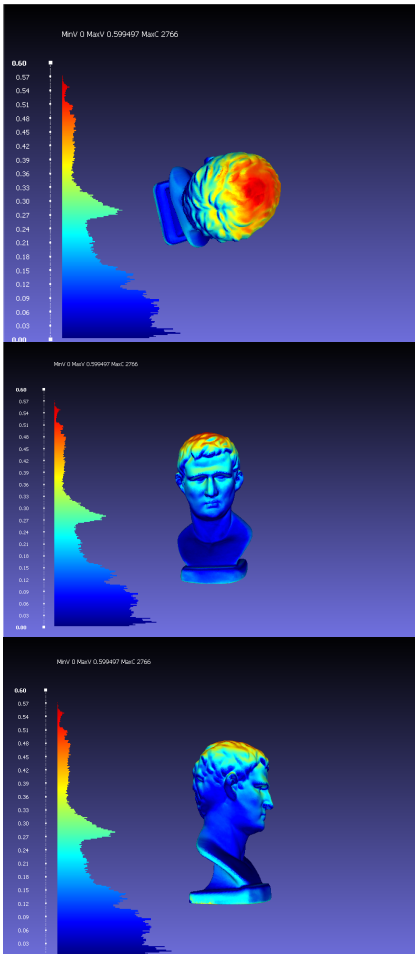
min : 0.000000
max : 0.006896
mean : 0.001789
RMS : 0.002322

The provided Hausdorff distance results indicate the proximity between the filtered models and the real-life digitalization of the objects. The minimum Hausdorff distance of 0.000002 and maximum of 0.599495 suggest that even the farthest points between the filtered models and the real-life digitalization are relatively close. The mean Hausdorff distance of 0.155529 and RMS of 0.201905 further reinforce the notion of close alignment between the models and the real objects.

Moreover, when considering the values with respect to the bounding box diagonal (86.938652), the normalized Hausdorff distances remain exceptionally low. The minimum normalized distance of 0 and maximum of 0.006896 indicate that even when accounting for the size of the image, the distance between points remains negligible. The mean normalized distance of 0.001789 and RMS of 0.002322 reaffirm the high level of precision in the alignment between the models and the real objects.



Overall, these Hausdorff distance results are favorable as they indicate a high level of accuracy and proximity between the scanned models and the real-world objects. This is essential because this will ensure the following scan data will assuredly be in shape to be used in Artificial intelligence.



D. Data Exploration

These objects were scanned to assess the quality and accuracy of the scanner in capturing a wide range of shapes, sizes, textures, and materials. Each object presents unique challenges for the scanner, allowing for evaluation of its performance under different conditions. Additionally, scanning a variety of objects enables assessment of the scanner's ability to capture fine details and complex shapes, as well as its capability to handle different surfaces and textures. This comprehensive scanning approach helps ensure that the scanner can deliver accurate and high-quality results across a variety of applications and environments.

- **Different types of shells.**
- **A complete chess set with one piece of each type for both white pieces.**
- **Statues**
- **Mariachi Skull**
- **Action figures**
- **Stone figures**
- **Ceramic masks**

And more, the totality of the list will be seen in the database that will be provided at the end of this project, we hope not only to maintain it but increase it in the near future with more specifically separated groups rather than random objects chosen to test the strengths and quality both of the filters and the miniscan 3d from RevoPoint.

VII. Conclusion

The significance of 3D models within the realm of artificial intelligence lies in their ability to provide a more faithful and comprehensive representation of the three-dimensional environment. Unlike two-dimensional or textual data, 3D models empower AI systems to comprehend and interact with the real world in a more natural and precise manner. This is crucial across a wide array of applications, ranging from robotics to augmented reality, where understanding the environment in three dimensions is pivotal for automated decision-making and effective user interaction.

The Revopoint Mini has emerged as a promising tool for capturing 3D models, particularly concerning the preservation of fine details in scanned objects. Its capability to maintain high resolution is essential for applications requiring a high degree of precision, such as engineering part manufacturing or cultural heritage preservation. However, when conducting multiple scans with the Revopoint Mini, certain limitations arise. The necessity for accurately aligning different individual scans can become a labor-intensive and error-prone task, especially for objects with complex shapes or significant overlap areas. Additionally, information loss in overlapping regions can impact the overall quality of the final 3D model, limiting its utility in applications demanding precise reproduction of the scanned object.

Conversely, Open3D offers a versatile and robust solution for manipulating and visualizing point clouds generated by devices like the Revopoint Mini. With a broad range of functionalities, from scan registration and fusion to surface reconstruction and interactive visualization, Open3D stands out as an indispensable tool in the toolkit of any developer or researcher working with three-dimensional data. Its efficiency and scalability ensure swift processing even for large and complex datasets, while its modular design and user-friendly interface make it accessible to users of all experience levels.

In conclusion, while the Revopoint Mini shows promising results in terms of detail preservation in 3D models, its practical use may be constrained by challenges such as precise alignment of multiple scans and information loss in overlapping areas. However, tools like Open3D provide robust and flexible solutions to overcome these limitations, enabling

users to harness the full potential of three-dimensional data in artificial intelligence applications and beyond.

VIII. Acknowledgment

I would like to thank Jorge Luis Perez Gonzales for the money invested in the project, it is thanks to him and his interest we have such a variety of objects to show the abilities of the code and the 3D scanner.

References

[1] ShapeNet. ShapeNet. Accessed on April 15, 2024. [Online]. Available: https://shapenet.org/
[2] The Stanford 3D Scanning Repository. Computer Graphics at Stanford University. Accessed on April 15, 2024. [Online]. Available: https://graphics.stanford.edu/data/3Dscanrep/
[3] 3D Digitization —. 3D Digitization —. Accessed on April 16, 2024. [Online]. Available: https://3d.si.edu/
[4] revoscan5. Revopoint 3D US. Accessed on April 16, 2024. [Online]. Available: https://www.revopoint3d.com/pages/revoscan5
[5] REVOPOINT MINI 3D Scanner. ELEGOO Official. Accessed on April 16, 2024. [Online]. Available: https://www.elegoo.com/products/revopoint-mini-3d-scanner
[6] PLY - Polygon 3D File Format. File Format Docs. Accessed on April 16, 2024. [Online]. Available: https://docs.fileformat.com/3d/ply/
[7] Q.-Y. Zhou, J. Park, and V. Koltun, "Open3D – A Modern Library for 3D Data Processing," Open3D – A Modern Library for 3D Data Processing, Accessed: Apr. 15, 2024. [Online]. Available: https://www.open3d.org/
[8] Surface Reconstruction — Open3D latest (664eff5) documentation. Open3D – A Modern Library for 3D Data Processing. Accessed on April 16, 2024. [Online]. Available: https://www.open3d.org/docs/latest/tutorial/Advanced/surface_reconstruction.html
[9] What is Hausdorff distance? Posted on October 2, 2019. Statistical Odds and Ends. Available: https://statisticaloddsandends.wordpress.com/2019/10/02/what-is-hausdorff-distance/
[10] ALOOPINGICON. Measuring the difference between two meshes. MeshLab Stuff. Posted on January 10, 2010. Accessed on April 16, 2024. [Online]. Available: https://meshlabstuff.blogspot.com/2010/01/measuring-difference-between-two-meshes.html