Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

Visualization of point cloud quality

Bachelor Thesis

2025
Lukáš Heldák

Comenius University in Bratislava
Faculty of Mathematics, Physics and Informatics

Visualization of point cloud quality

Bachelor Thesis

Study Programme:	Computer Science
Field of Study:	Computer Science
Department:	Department of Algebra and Geometry
Supervisor:	Mgr. Marcel Makovník, PhD

Bratislava, 2025
Lukáš Heldák

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Lukáš Heldák
Študijný program: informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
Študijný odbor: informatika
Typ záverečnej práce: bakalárska
Jazyk záverečnej práce: anglický
Sekundárny jazyk: slovenský

Názov: Visualization of point cloud quality
Vizualizácia kvality mračien bodov

Anotácia: Práca sa zaoberá rôznymi charaktersitikami mračien bodov, ktoré sú používané pri vyhodnocovaní ich kvality v rôznych úlohách (napr. nadvzorkovanie, redukcia šumu). Špeciálne sa sústredíme na ohodnotenie podobnosti dvoch mračien bodov a ich vizualizáciu.

Cieľ: Naštudovať a zosumarizovať bežne používané miery kvality mračien bodov a podobnosti mračien bodov (napr. vzdialenosť skosenia a jej modifikácie). Vytvoriť softvérové dielo, ktoré používateľsky prívetivým spôsobom zobrazuje kvalitu mračien bodov s možnosťou výberu miery kvality a exportu pre ďalšie použitie. Navrhnúť vlastné modifikácie existujúcich mier kvality a následná demonštrácia na konkrétnych experimentoch.

Literatúra: WU, Tong, et al. Density-aware chamfer distance as a comprehensive metric for point cloud completion. In: Proceedings of the 35th International Conference on Neural Information Processing Systems. 2021. p. 29088-29100.

JAVAHERI, Alireza, et al. Improving PSNR-based quality metrics performance for point cloud geometry. In: 2020 IEEE International Conference on Image Processing (ICIP). IEEE, 2020. p. 3438-3442.

Kľúčové slová: mračno bodov, kvalita, podobnosť

Vedúci: Mgr. Marcel Makovník, PhD.
Katedra: FMFI.KAG - Katedra algebry a geometrie
Vedúci katedry: doc. Mgr. Tibor Macko, PhD.

Spôsob sprístupnenia elektronickej verzie práce:
bez obmedzenia

Dátum zadania: 30.10.2024

Dátum schválenia: 30.10.2024

doc. RNDr. Dana Pardubská, CSc.
garant študijného programu

.....
študent

.....
vedúci práce

Comenius University Bratislava
Faculty of Mathematics, Physics and Informatics

THESIS ASSIGNMENT

Name and Surname:	Lukáš Heldák
Study programme:	Computer Science (Single degree study, bachelor I. deg., full time form)
Field of Study:	Computer Science
Type of Thesis:	Bachelor´s thesis
Language of Thesis:	English
Secondary language:	Slovak

Title: Visualization of point cloud quality

Annotation: This work discusses various characteristics of point clouds, which are used in the evaluation of their quality in different tasks (e.g. upsampling, noise reduction). We specifically focus on the evaluation of the similarity of two point clouds and its visualization.

Aim: Study and summarize commonly used measures of point cloud quality and point cloud similarity (e.g. chamfer distance and its modifications). Create a software that displays the quality of point clouds in a user-friendly way, with the possibility to select the quality measure and export it for further use. Propose our own modifications to existing quality measures and subsequent demonstration on practical experiments.

Literature: WU, Tong, et al. Density-aware chamfer distance as a comprehensive metric for point cloud completion. In: Proceedings of the 35th International Conference on Neural Information Processing Systems. 2021. p. 29088-29100.

JAVAHERI, Alireza, et al. Improving PSNR-based quality metrics performance for point cloud geometry. In: 2020 IEEE International Conference on Image Processing (ICIP). IEEE, 2020. p. 3438-3442.

Keywords: point cloud, quality, similarity

Supervisor:	Mgr. Marcel Makovník, PhD.
Department:	FMFI.KAG - Department of Algebra and Geometry
Head of department:	doc. Mgr. Tibor Macko, PhD.
Assigned:	30.10.2024
Approved:	30.10.2024

doc. RNDr. Dana Pardubská, CSc.
Guarantor of Study Programme

.....
Student

.....
Supervisor

Declaration of Authorship

I hereby declare that I have completed the entire bachelor thesis on the topic "Visualization of point cloud quality", including all its appendices and illustrations, independently, using the literature listed in the attached bibliography and artificial intelligence tools. I declare that I have used artificial intelligence tools in accordance with applicable legal regulations, academic rights and freedoms, ethical and moral principles, while upholding academic integrity, and that their use is appropriately indicated within the thesis.

Acknowledgments: I would like to sincerely thank my supervisor, Marcel Makovník, for his expert guidance and valuable advice throughout the process of working on this project. His support and insights were instrumental in overcoming the challenges I encountered.

Abstrakt

Táto práca sa zameriava na vývoj softvérového nástroja na porovnávanie mračien bodov, ktoré sa bežne využívajú v oblastiach ako kontrola kvality v priemysle, robotika či 3D modelovanie. Nástroj umožňuje používateľom merať a vizualizovať rozdiely medzi dvoma množinami trojrozmerných dát. Okrem bežne používaných metód podporuje aj nové, na mieru navrhnuté porovnávacie techniky, ktoré boli vyvinuté v rámci tejto práce. Tie používateľom pomáhajú odhaliť chyby, odchýlky alebo zmeny medzi modelmi a sú užitočné v rôznych praktických prípadoch. Aplikácia ponúka prehľadné a používateľsky prívetivé rozhranie a umožňuje exportovať výsledky porovnania na ďalšie spracovanie alebo integráciu do iných nástrojov.

Kľúčové slová: mračno bodov, kvalita, podobnosť

Abstract

This thesis introduces a software tool for comparing 3D point clouds, which are widely used in fields like industrial inspection, robotics, and 3D modeling. The tool allows users to measure and visualize differences between two sets of spatial data. It supports commonly used methods as well as new custom-designed comparison techniques developed as part of this work. These help users identify errors, defects, or changes between models and support various practical use cases. The application offers a user-friendly interface and includes the ability to export comparison results for further processing or integration into other tools.

Keywords: point cloud, quality, similarity

Contents

Introduction

In recent years, 3D point clouds have become commonly used representation of surfaces. These data sets are crucial in diverse fields including industrial inspection, cultural heritage preservation, autonomous navigation, and medical imaging. However, due to noise, resolution limits, occlusions, and imperfect reconstructions, it is essential to evaluate and compare the quality of point clouds, especially when assessing the performance of different data acquisition methods or reconstruction algorithms.

The motivation behind this work stems from the lack of robust, flexible, and insightful tools for comparing point clouds. Currently, the open-source software *MeshLab* [3] allows users to calculate the Hausdorff distance and visualize differences by coloring one cloud based on distance to another. However, its capabilities are limited. It only supports one type of distance measure, lacks support for intuitive visualizations, and fails to provide the level of detail and customization that advanced users and researchers require. By creating a tool specifically designed for this purpose, this thesis contributes not only a practical utility but also a foundation for future research and development.

In addition to implementing standard measures such as the Hausdorff and chamfer distance, the software also introduces several custom-designed quality measures developed specifically within this thesis. These new measures aim to capture other aspects of point cloud similarity that are often overlooked by traditional methods, offering users a richer and more nuanced analysis of differences.

The tool developed in this thesis has numerous practical applications. For example:

- **Algorithm Benchmarking:** When developing new algorithms for denoising, downsampling, or registering point clouds, researchers need a reliable way to compare the algorithm's output against a ground truth. The software can compute precise error distributions and help visualize which areas are affected most by algorithmic changes.

- **Industrial Inspection:** In automated manufacturing, 3D scanners can be used on assembly lines to verify that products conform to design specifications. By comparing a scan of a manufactured item to a reference model, the tool can highlight defects, deformations, or missing parts in real time.

- **Change Detection:** In construction or civil engineering, periodic 3D scans can be compared to detect structural shifts or unauthorized modifications over time. The software can assist in visualizing such changes clearly and quantitatively.
- **Education and Demonstration:** The tool can be used in academic settings to teach students about point cloud quality measures, error analysis, and 3D data processing, providing immediate visual feedback that enhances understanding.

The software and methodologies introduced in this work are designed to be modular and reusable. As such, they can serve as a base for future bachelor's and master's theses, scientific publications, or industry tools. The impact of this project thus goes beyond a single use case and aims to foster broader advancement in 3D data processing and analysis. The tool is designed to be user-friendly, providing an intuitive interface for selecting quality measures, adjusting parameters and visualizing results. The tool allows to export results for further analysis.

The following chapters provide an in-depth look at the software's architecture, the mathematical foundations of the implemented measures, the design of the visualization components, and real-world testing scenarios that demonstrate the tool's capabilities. Chapter 1 introduces the concept of point clouds and discusses the data structures used for efficient storage and processing. Chapter 2 focuses on the theoretical background and properties of quality measures used to evaluate the similarity and structure of point clouds. Chapter 3 describes the implementation of the developed tool, including the graphical user interface, interaction logic, and visualization techniques. Finally, Chapter 4 presents experimental results and proposed modifications to the existing measures, demonstrating the effectiveness and versatility of the application in practice.

Chapter 1

Point Cloud and Data Structures

This chapter provides a comprehensive overview of point clouds and the data structures employed for their efficient storage and processing. It introduces the fundamental properties of point clouds, their role in representing spatial data, and the challenges associated with their unstructured nature. To address these challenges, various data structures are explored, with a focus on their ability to optimize search operations, spatial partitioning, and data retrieval. Understanding these structures is imperative for efficient processing, analysis, and visualization of large-scale three-dimensional datasets.

1.1 Point Cloud

A point cloud is a collection of discrete data points in three-dimensional space, representing the external surface of an object or scene (see Figure 1.1). Each point is defined by its coordinates $(x, y, z) \in \mathbb{R}^3$, and often supplemented with additional attributes such as color or normal vectors.

Definition: A point cloud can be formally defined as a finite set of points in $\mathbb{R}^3$, given by:

$$A = \{a_1, a_2, \ldots, a_n;\ a_i = (x_i, y_i, z_i) \in \mathbb{R}^3\} \tag{1.1}$$

Point clouds are widely used in fields such as computer graphics, geospatial mapping, and 3D scanning. They offer a raw, unstructured way of capturing spatial data directly.

Point clouds are typically generated by 3D scanning technologies, such as:

- **3D Scanning:** Devices such as structured light scanners or depth cameras emit patterns or capture depth data directly to reconstruct the geometry of an object. It is more suitable for small, detailed objects.

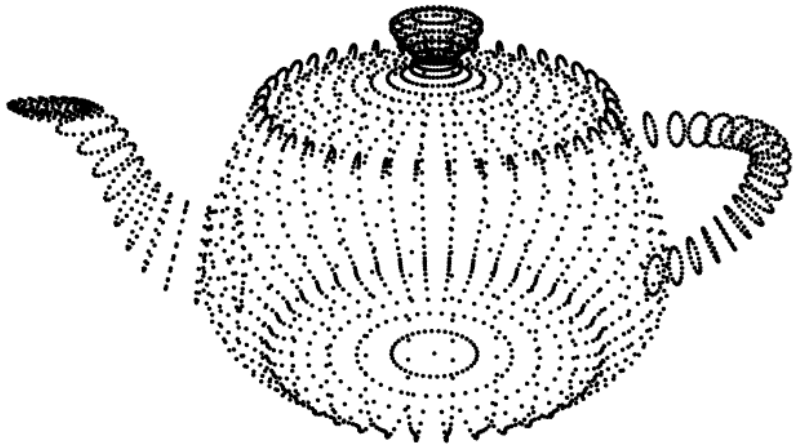

Figure 1.1: Visualization of a point cloud representing a teapot.

- **Photogrammetry:** Combines images taken from multiple perspectives to create 3D reconstructions.
- **LIDAR (Light Detection and Ranging):** Uses lasers to measure distances and generate dense, accurate point clouds. It excels in capturing large-scale outdoor environments

Point clouds are a versatile data format with applications in various domains:

- **3D Reconstruction:** Used to create digital models of real-world objects or scenes for gaming, virtual reality, or industrial design.
- **Robotics and Autonomous Systems:** Enable obstacle detection, path planning, and object recognition for robots and self-driving cars.
- **Geospatial Analysis:** Support terrain modeling, urban planning, and environmental monitoring.
- **Cultural Heritage Preservation:** Facilitate the digital preservation of historical artifacts and sites.

Point clouds differ from structured 3D representations like meshes or voxel grids in that they lack explicit connectivity between points. This unstructured nature makes point clouds flexible but also presents several challenges:

- **High Density:** Point clouds can contain millions or billions of points, requiring efficient processing techniques.
- **Noise and Outliers:** Points may include measurement errors or irrelevant data.

- **Irregular Distribution:** Points are not evenly spaced, leading to potential gaps or overlaps in the representation.

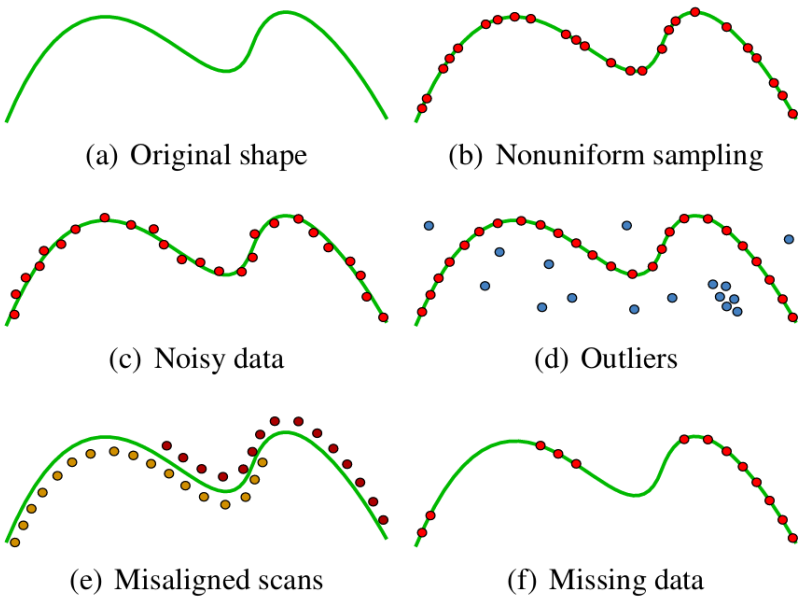

Figure 1.2: Different forms of point cloud artifacts, shown here in the case of a curve in 2D. Source: [2]

To address these challenges, algorithms such as filtering, down-sampling, and surface reconstruction are commonly applied. Below are common types of artifacts, which can be seen in Figure 1.2:

(a) Original Shape: The ground truth representation of the object or scene, providing a reference for comparison.

(b) Nonuniform Sampling: The density of points varies across the shape, leading to areas with high or low point concentration. This can affect interpolation and reconstruction accuracy.

(c) Noisy Data: Small perturbations are introduced to the points, often due to sensor inaccuracies or environmental factors, causing rough or inaccurate surface estimations.

(d) Outliers: Some points are located far from the actual surface, usually resulting from measurement errors or reflections. These outliers can negatively impact surface fitting and modeling.

(e) Misaligned Scans: Multiple scans of the same object are not perfectly aligned, leading to duplicate or offset point distributions. This misalignment often occurs in multi-scan 3D reconstruction.

(f) Missing Data: Some parts of the object are not captured due to occlusions or sensor limitations, leading to gaps in the point cloud representation.

Processing and visualizing point clouds effectively requires advanced algorithms and tools to manage their large size, noise, and unstructured nature.

1.2 k-D Tree

To efficiently represent and work with point clouds, it is appropriate to use a k-D tree. A k-D tree (k-dimensional tree) is a binary tree that recursively partitions a k-dimensional space along its coordinates. Each node in the tree represents a point in the dataset and defines a hyperplane that splits the space into two half-spaces. At each level of the tree, the data is split along one dimension, cycling through all k dimensions. Points with values smaller than the current node's value in the selected dimension are placed in the left subtree, while larger points go to the right subtree. This recursive binary partitioning is illustrated in Figure 1.3, and it is particularly effective for organizing and querying multi-dimensional data. Therefore, k-D trees are well suited for operations such as nearest neighbor search, range search, and clustering.

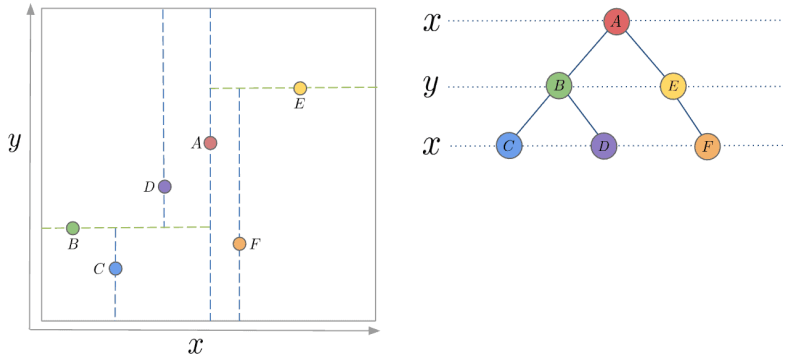

Figure 1.3: An example of k-D tree partitioning in 2D space. Source: [4].

In some cases, an octree [8] is used to organize three-dimensional space by recursively subdividing it into eight octants at each node. This adaptive subdivision continues until a specified depth is reached or when each leaf node contains fewer than a set number of points, making octrees particularly effective for managing sparse datasets and volumetric data. Octrees are commonly applied in areas like volumetric rendering, collision detection, and level-of-detail modeling due to their ability to efficiently handle dynamic data insertion and deletion. Despite these advantages, k-D trees are generally preferred for point cloud processing because they offer more balanced partitioning in low-dimensional spaces, leading to faster search operations, efficient nearest neighbor search through axis-aligned splits, and better memory efficiency when handling dense, uniformly distributed data.

1.2.1 Construction and Insertion

The k-D tree is constructed recursively using the following process:

1. **Select Splitting Axis:** At depth d, the splitting axis is determined as d mod k, where k is the number of dimensions.
2. **Choose Median Point:** Sort the points along the selected axis and select the median to ensure balanced partitions.
3. **Recursive Partitioning:** Split the points into left and right subsets around the median and recursively build the subtrees.

The C++ implementation of the k-D tree construction is shown in Algorithm 1.1.

```
struct Node
{
    vector<double> point;
    Node* left = nullptr;
    Node* right = nullptr;

    Node(vector<double> pt) : point(pt) {}
};

Node* buildKDTree(vector<vector<double>>& points, int depth, int k)
{
    if (points.empty()) return nullptr;

    // Determine splitting axis
    int axis = depth % k;

    // Sort points along the current axis
    sort(points.begin(), points.end(),
         [axis](const vector<double>& a, const vector<double>& b) {
             return a[axis] < b[axis];
         });

    // Select median point
    int medianIdx = points.size() / 2;
    Node* node = new Node(points[medianIdx]);

    // Build left and right subtrees
    vector<vector<double>> leftPoints(points.begin(), points.begin() +
    medianIdx);
    vector<vector<double>> rightPoints(points.begin() + medianIdx + 1,
    points.end());
```

```
30      node->left = buildKDTree(leftPoints, depth + 1, k);
31      node->right = buildKDTree(rightPoints, depth + 1, k);
32
33      return node;
34  }
```

Algorithm 1.1: k-D tree Construction in C++

Inserting a new point into a k-D tree involves traversing the tree and comparing values along the splitting axes:

1. **Traverse the Tree:** Start at the root node and compare the new point with the current node's value along the splitting axis.

2. **Move to a Child:** Depending on the comparison, move to the left or right child node.

3. **Insert as a Leaf Node:** When an empty position is reached, insert the new point as a leaf node.

Algorithm 1.2 presents the C++ code for inserting a point into a k-D tree.

```
1  Node* insertKDTree(Node* node, vector<double> point, int depth, int k)
2  {
3      if (node == nullptr) return new Node(point);
4
5      // Determine splitting axis
6      int axis = depth % k;
7
8      // Traverse to the appropriate child
9      if (point[axis] < node->point[axis]) {
10         node->left = insertKDTree(node->left, point, depth + 1, k);
11     } else {
12         node->right = insertKDTree(node->right, point, depth + 1, k);
13     }
14
15     return node;
16 }
```

Algorithm 1.2: k-D tree Insertion in C++

However, frequent insertions and deletions may cause the tree to become unbalanced, potentially degrading performance. Therefore, periodic rebalancing or reconstruction is necessary for dynamic datasets.

1.2.2 Nearest Neighbor Search

Nearest neighbor search is one of the most fundamental operations performed on a k-D tree, allowing efficient retrieval of the closest point to a given query point in a k-dimensional space. The key advantage of k-D trees in nearest neighbor search lies in faster pruning of the search space compared to brute force methods, as illustrated in Figure 1.4.

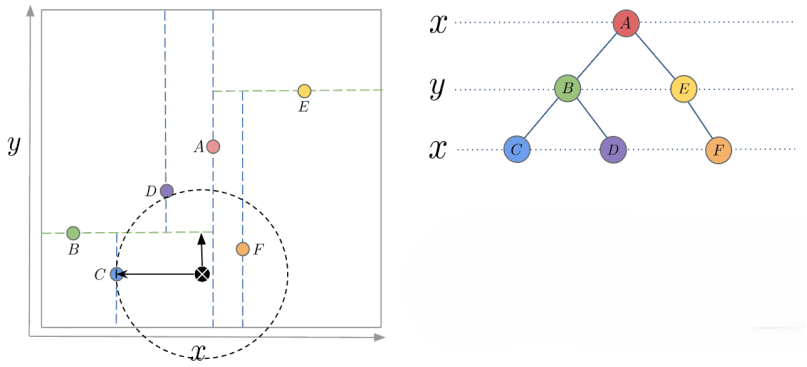

Figure 1.4: Nearest neighbor problem visualized on a k-D tree. Source: [4].

The nearest neighbor search algorithm proceeds as follows:

1. **Traverse the Tree:** Compare the query point at each node and move to the left or right child.
2. **Reach a Leaf:** Compute the distance to the query point.
3. **Backtrack if Needed:** Check if the other subtree could contain closer points by comparing distances to the splitting hyperplane.
4. **Return the Nearest Neighbor:** Return the closest point.

The search process takes advantage of the tree structure to minimize comparisons by eliminating large sections of the dataset at each step. By narrowing the search to areas that are more likely to contain closer points, the method greatly enhances search efficiency, especially in high-dimensional spaces.

```
1 double distance(const vector<double>& p1, const vector<double>& p2) {
2     double sum = 0;
3     for (int i = 0; i < p1.size(); i++) {
4         sum += (p2[i] - p1[i]) * (p2[i] - p1[i]);
5     }
6     return sum;
7 }
```

```
8  void nearestNeighbor(Node* node, const vector<double>& queryPoint, int
       depth, int k, Node*& bestNode, double& bestDist) {
9      if (node == nullptr) return;
10
11     // Calculate the squared distance
12     double dist = distance(queryPoint, node->point);
13
14     // If the current node is closer, update the best match
15     if (dist < bestDist) {
16         bestDist = dist;
17         bestNode = node;
18     }
19
20     // Determine splitting axis
21     int axis = depth % k;
22
23     // Traverse to the appropriate child
24     if (queryPoint[axis] < node->point[axis]) {
25         nearestNeighbor(node->left, queryPoint, depth + 1, k, bestNode
       , bestDist);
26         // Check the other side if a closer point might be found
27         if (abs(queryPoint[axis] - node->point[axis]) < bestDist) {
28             nearestNeighbor(node->right, queryPoint, depth + 1, k,
       bestNode, bestDist);
29         }
30     } else {
31         nearestNeighbor(node->right, queryPoint, depth + 1, k,
       bestNode, bestDist);
32         // Check the other side if a closer point might be found
33         if (abs(queryPoint[axis] - node->point[axis]) < bestDist) {
34             nearestNeighbor(node->left, queryPoint, depth + 1, k,
       bestNode, bestDist);
35         }
36     }
37 }
```

Algorithm 1.3: k-D tree Nearest Neighbor Search in C++

In many cases, when performing nearest neighbor searches, we are only interested in comparing distances relative to each other rather than calculating the exact distance. In this context, we can use the squared Euclidean distance to avoid the computational overhead of square roots. The computation of this distance is implemented in the *distance*() function within Algorithm 1.3, following the mathematical formulation:

$$d^2(a, b) = (x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2 \tag{1.2}$$

where $a = (x_1, y_1, z_1)$ and $b = (x_2, y_2, z_2)$.

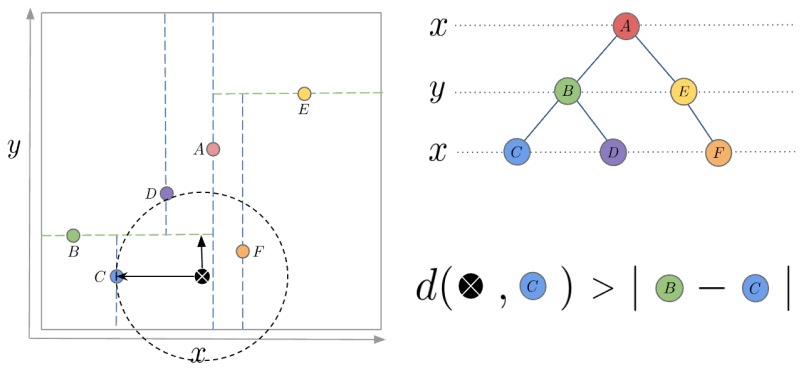

Figure 1.5: Here the Euclidean distance is bigger, indicating that the hypersphere intersects the cutting plane of B, prompting a switch to traversing the E branch. Source: [4].

1.2.3 Advantages for Point Clouds

k-D trees are highly efficient for point cloud processing, particularly in nearest neighbor searches, which are crucial for point cloud similarity calculations. They offer both time and space efficiency:

- **Time Complexity:** The time complexity of constructing a balanced k-D tree is $O(n \log n)$, where n is the number of points, due to the need to sort the points along each axis during the recursive construction. Nearest neighbor search in a balanced k-D tree has an average time complexity of $O(\log n)$, as the search space is halved at each level, making it much faster than brute-force methods with $O(n)$ time complexity.
- **Space Complexity:** The space complexity is $O(n)$, where n is the number of points, as each node stores a point and two child pointers.

Their performance is best with uniformly distributed datasets but may degrade with sparse or skewed data distributions. Nonetheless, they remain a robust choice for point cloud similarity in dense datasets.

Chapter 2

Quality Measures

Quality measures provide quantitative metrics to assess the similarity between point clouds, evaluate reconstruction accuracy, and identify discrepancies or errors. These measures are essential for applications such as point cloud registration, surface reconstruction, and shape comparison. They are used to measure the similarity or dissimilarity between two sets of points in $\mathbb{R}^3$. In this work, we focus on some of the most frequently used quality measures: Hausdorff distance, chamfer distance and earth-movers distance.

The aim of this chapter is to provide a detailed overview of these measures, their mathematical formulations, applications, and advantages. This chapter also establishes the foundation for evaluating point cloud quality in the subsequent implementation and experimentation phases of this thesis. For the purposes of this work, all definitions and formulations in this chapter are tailored to the context of point clouds, which are represented as finite non-empty sets according to Definition 1.1.

2.1 Hausdorff Distance

The Hausdorff distance is widely utilized in comparing geometric shapes, point clouds, and surfaces, particularly in fields such as computer vision, 3D modeling, and medical imaging.

Definition: Let $A = \{a_1, a_2, \ldots, a_n\}$ and $B = \{b_1, b_2, \ldots, b_m\}$ be two non-empty finite subsets of $\mathbb{R}^3$ with a distance function $d(a, b)$. The directed Hausdorff distance from A to B is defined as:

$$d_H(A, B) = \max_{a \in A} \min_{b \in B} d(a, b), \tag{2.1}$$

where $\min_{b \in B} d(a, b)$ is the minimum distance from a point $a \in A$ to any point $b \in B$, and $\max_{a \in A}$ is the greatest of these minimum distances over all points in A. [5]

The bidirectional (or symmetric) Hausdorff distance is then defined as:

$$d_H(A, B) = \max\left(\max_{a\in A}\min_{b\in B} d(a, b), \max_{b\in B}\min_{a\in A} d(b, a)\right). \tag{2.2}$$

This ensures that the distance considers both sets equally, accounting for the worst-case discrepancy between them.

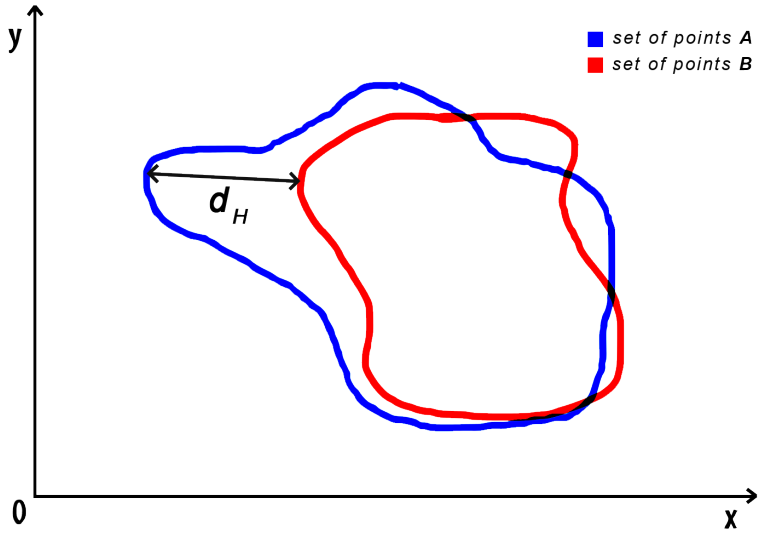

Figure 2.1: Visualization of the Hausdorff distance between two sets of points. The Hausdorff distance is determined by the longest shortest distance, shown as the black line connecting a point in A to its closest point in B.

Example: Consider two point sets in a 2D Euclidean space:

$$A = \{(1, 1), (2, 2), (3, 3)\}, \quad B = \{(2, 2), (3, 4)\}.$$

For the directed distance from A to B:

$$\min_{b\in B} d((1, 1), b) = \sqrt{2}, \quad \min_{b\in B} d((2, 2), b) = 0, \quad \min_{b\in B} d((3, 3), b) = 1.$$

The maximum of these minimum distances is:

$$\max_{a\in A}\min_{b\in B} d(a, b) = \sqrt{2}.$$

Repeating for B to A, we find:

$$d_H(B, A) = 1.$$

The bidirectional Hausdorff distance is thus:

$$d_H(A, B) = \max(\sqrt{2}, 1) = \sqrt{2}.$$

A key property of the Hausdorff distance is its sensitivity to outliers. This makes it particularly useful in detecting major deviations and errors. Another of its main advantages is its applicability across any metric space, making it versatile for tasks ranging

from 2D shape analysis to higher-dimensional comparisons. Additionally, its geometric interpretation as the longest shortest distance is easy to visualize and understand, as shown in Figure 2.1.

For example, in 3D modeling, it can measure how well a scanned object matches a reference model, identifying maximum deviations and ensuring adherence to tolerance levels. Its directed form is also useful when only one direction of comparison is needed, such as validating predictions against ground truth data.

2.2 Chamfer Distance

The chamfer distance is another widely used metric for comparing sets of points. Unlike the Hausdorff distance, which focuses on the maximum deviation between two sets, the chamfer distance considers the average closeness between all points in the sets. Figure 2.2 illustrates this interpretation, where the chamfer distance is the average of the shortest distances between corresponding points in two sets. It is particularly valuable in applications where a smooth, more forgiving comparison between point clouds or shapes is desired. We proceed to the definition of chamfer distance as introduced in [17].

Definition: Let $A = \{a_1, a_2, \ldots, a_n\}$ and $B = \{b_1, b_2, \ldots, b_m\}$ be two non-empty finite subsets of $\mathbb{R}^3$ with a distance function $d(a, b)$. The chamfer distance from A to B is defined as the sum of the squared distances from each point in A to its nearest neighbor in B, averaged over all points in A:

$$d_C(A, B) = \frac{1}{|A|} \sum_{a \in A} \min_{b \in B} d(a, b)^2. \tag{2.3}$$

Similarly, the chamfer distance from B to A is given by:

$$d_C(B, A) = \frac{1}{|B|} \sum_{b \in B} \min_{a \in A} d(b, a)^2. \tag{2.4}$$

The bidirectional chamfer distance is then defined as the average of the chamfer distances from A to B and from B to A:

$$d_C(A, B) = \frac{1}{2} \left(\frac{1}{|A|} \sum_{a \in A} \min_{b \in B} d(a, b)^2 + \frac{1}{|B|} \sum_{b \in B} \min_{a \in A} d(b, a)^2 \right). \tag{2.5}$$

Example: Consider the same two point sets in a 2D Euclidean space as in the Hausdorff distance example:

$$A = \{(1, 1), (2, 2), (3, 3)\}, \quad B = \{(2, 2), (3, 4)\}.$$

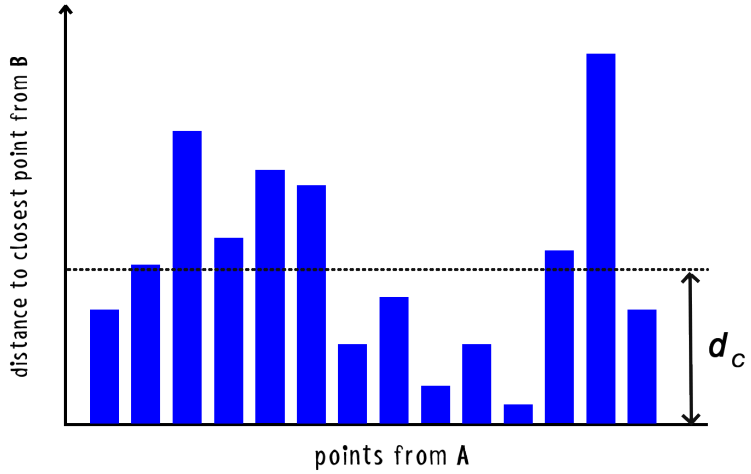

Figure 2.2: Visualization of the chamfer distance between two sets of points. The chamfer distance is the average squared distance from each point in one set to the closest point in the other set.

For the chamfer distance from A to B, we calculate:

$$\min_{b\in B} d((1,1),b) = \sqrt{2}, \quad \min_{b\in B} d((2,2),b) = 0, \quad \min_{b\in B} d((3,3),b) = 1.$$

The squared distances are:

$$\left(\sqrt{2}\right)^2 = 2, \quad 0^2 = 0, \quad 1^2 = 1.$$

Thus, the average squared distance from A to B is:

$$\frac{1}{3}(2+0+1) = 1.$$

Similarly for B to A we calculate:

$$\min_{a\in A} d((2,2),a) = 0, \quad \min_{a\in A} d((3,4),a) = 1.$$

The squared distances are:

$$0^2 = 0, \quad 1^2 = 1.$$

Thus, the average squared distance from B to A is:

$$\frac{1}{2}(0+1) = 0.5.$$

The bidirectional chamfer distance is:

$$d_C(A,B) = \frac{1}{2}(1+0.5) = 0.75.$$

As can be seen, despite the identical input, the chamfer and Hausdorff distances produce different results.

Chamfer distance is often employed in point cloud registration [6], where the objective is to align two or more point clouds. By minimizing the chamfer distance between the point sets, the optimal alignment between the clouds can be determined.

In the context of deep learning, chamfer distance is commonly used as a loss function in tasks where the goal is to generate a point cloud that resembles a target point cloud. The chamfer distance loss helps guide the generated point cloud to be as similar as possible to the target [13].

2.3 Earth Mover's Distance

The earth mover's distance (EMD) [9], also known as the Wasserstein distance, is a function used to measure the distance between two distributions, often represented as weighted point sets in $\mathbb{R}^3$. It is grounded in the theory of optimal transport [16], where the goal is to compute the minimum cost required to transform one distribution into another. Imagine each distribution as a pile of dirt on a plane. The goal is to move the dirt from one pile to another while minimizing the total effort. The earth mover's distance represents the minimum amount of "work" required to transform one pile into the other. Figures 2.3 and 2.4 provide a conceptual illustration of EMD.

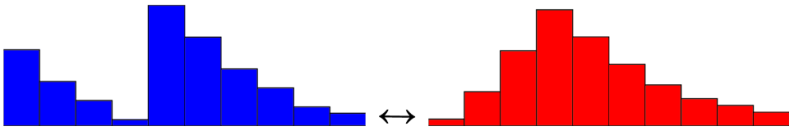

Figure 2.3: Example of two distributions P and Q.

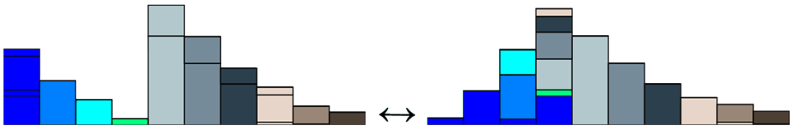

Figure 2.4: Optimal transportation between P and Q.

Definition: Let $P = \{p_1, p_2, \ldots, p_n\}$ and $Q = \{q_1, q_2, \ldots, q_m\}$ be two distributions of points, where each point has an associated weight $w(p_i)$ and $w(q_j)$ such that $\sum_{i=1}^{n} w(p_i) = \sum_{j=1}^{m} w(q_j)$. The earth mover's distance is defined as the minimum cost

of transporting the weights in P to match Q, given a ground distance $d(p, q)$ between any two points p and q. More precisely, it is expressed as:

$$\text{EMD}(P, Q) = \min_{F} \sum_{i=1}^{n} \sum_{j=1}^{m} f_{ij} d(p_i, q_j), \tag{2.6}$$

where $F = [f_{ij}]$ represents the optimal flow and f_{ij} is the flow from p_i to q_j.

In the context of point clouds, we represent each point cloud as a distribution of points in $\mathbb{R}^3$. For a given point cloud $P = \{p_1, p_2, \ldots, p_n\}$ and $Q = \{q_1, q_2, \ldots, q_m\}$, equal weights are asigned to all points (uniform distributions) since the total mass of both distributions is normalized to 1.

2.3.1 Hungarian algorithm

The computation of EMD can be cast as a linear optimization problem. For discrete distributions, this is equivalent to solving the assignment problem [1], where the goal is to find an optimal matching between two sets to minimize the total cost. This problem is analogous to the classical "jobs and workers" scenario, where each worker must be assigned a job such that the total cost is minimized. The Hungarian algorithm (or Hungarian method) [7] efficiently solves this problem in polynomial time.

This algorithm operates by maintaining a set of potential values for jobs and workers and iteratively updating them to minimize reduced costs. The algorithm proceeds as follows:

- **Matrix representation:** Let $C[i][j]$ represent the cost of assigning job j to worker i. The goal is to minimize the total assignment cost.
- **Step 1: Row reduction:** For each row in the matrix, subtract the smallest value in that row from every element in that row.
- **Step 2: Column reduction:** For each column in the matrix, subtract the smallest value in that column from every element in that column.
- **Step 3: Cover zeros:** Cover all the zeros in the matrix with the minimum number of horizontal and vertical lines.
- **Step 4: Adjust the matrix:** If the number of lines used to cover the zeros is equal to the size of the matrix, the algorithm stops. Otherwise:
  - Find the smallest uncovered value in the matrix.
  - Subtract this value from all uncovered elements.

  - Add this value to all elements that are covered by both a horizontal and a vertical line.

- **Step 5: Optimal assignment:** Once the matrix has been adjusted, select the zeros in such a way that no two zeros are in the same row or column. This gives the optimal assignment.

The EMD problem shares similarities with the maximum flow problem in graph theory, where flow is routed through a network to minimize or maximize some objective function [11]. By interpreting the points in P and Q as nodes in a flow network, and distances as edge weights, the EMD computation can also be framed in this context. However, the Hungarian algorithm provides a more direct and efficient approach for discrete distributions.

In the case of point clouds, the cost matrix C represents pairwise distances between points in P and Q. By plugging the distance matrix into the algorithm, we can compute the EMD efficiently. Each entry $C[i][j]$ in the matrix corresponds to the distance between the i-th point in P and the j-th point in Q, allowing the Hungarian algorithm to calculate the optimal transport cost.

This formulation makes EMD particularly suitable for comparing point clouds because it captures geometric and structural differences. For example, in evaluating the reconstruction quality of a 3D scan, EMD can quantify how much the reconstructed point cloud deviates from the original in terms of both spatial arrangement and distribution. Unlike simpler metrics such as the chamfer distance, EMD explicitly models the "flow" of mass, making it robust to outliers and localized differences.

Chapter 3

Implementation

This chapter describes the architecture of the application, detailing the key components and their interactions. The implementation of this application is written in C++ and is designed primarily for comparing point clouds. The main goal is to provide an easy-to-use graphical tool that enables users to calculate distances between point clouds efficiently. The application is built using the Point Cloud Library (PCL) for point cloud processing and Qt framework for graphical user interface development.

Additionally, the software features various colorization techniques to enhance visualization, making it easier to interpret differences between point clouds. The user interface is designed to be intuitive, providing options for loading, processing, and visualizing point clouds with minimal effort. Finally, the chapter provides insights into the design choices that contribute to the overall usability and efficiency of the tool.

The full source code is publicly available at `github.com`. The repository also includes automatically generated documentation created using Doxygen [15], which provides detailed descriptions of the project's structure, classes, and functions.

3.1 Point Cloud Library

The Point Cloud Library (PCL) [10] is an open-source, C++, large-scale library that provides tools for processing 2D/3D image and point cloud data. It is widely used in fields such as robotics, computer vision, and 3D data processing due to its efficiency, versatility, and extensive support for various algorithms and data structures. PCL offers numerous functionalities including filtering, feature extraction, surface reconstruction, registration, segmentation, and visualization, making it a powerful tool for working with 3D point cloud data.

The library provides many different structures to represent 3D points. For this thesis, `pcl::PointXYZRGBA` was selected. It is a structure that stores 3D coordinates (`X`, `Y`, `Z`) along with color information (`R`, `G`, `B`, `A`). The `pcl::PointCloud<PointT>`

object, which is defined as `PointCloudT`, is then used to store a collection of these points, forming a point cloud. This choice of structures enables easy visualization of distances in the form of color later on.

PCL also provides a set of functions that simplify the process of loading 3D point cloud data from various file formats. Functions such as `pcl::io::loadPCDFile()`, `pcl::io::loadOBJFile()` and `pcl::io::loadPLYFile()`, automatically handle complexities and allow for easy parsing of point cloud data from `PCD`, `OBJ` and `PLY` formats.

The `pcl::visualization::PCLVisualizer` is a versatile and powerful tool for visualizing point clouds. This class can create a visualization window, which is referred to as the "viewer" in this thesis. Not only does it allow for the visualization of 3D data, but it also enables user interaction, such as rotating, zooming, and panning. With its ability to set custom rendering properties, it simplifies the process of rendering and interacting with point clouds, providing an intuitive and flexible interface for visualization.

The most important aspect of this thesis is the use of the `pcl::KdTree` and its implementation of nearest neighbor search. `pcl::KdTree` simplifies the process by automatically constructing a k-D tree from the point cloud, eliminating the need for a custom implementation. Once the tree is built, it provides an efficient nearest neighbor search function, which returns distances in squared form.

3.2 Qt Framework

Qt [14] is a cross-platform application framework widely used for developing graphical user interfaces (GUIs). It is written in C++ and offers a robust set of libraries that simplify complex application development, making it a preferred choice for many software projects in fields such as desktop applications, embedded systems, and scientific computing.

For this thesis, Qt was chosen primarily for its seamless integration with the Point Cloud Library (PCL). Qt's signal-slot mechanism allows efficient event-driven programming, making it well-suited for handling user interactions such as button clicks, dropdown selections, and other UI controls. Additionally, its support for OpenGL enables smooth visualization of 3D data, complementing PCL's rendering capabilities.

Qt provides a modular architecture with various components that aid in application development. The most relevant modules for this thesis include:

- **QtWidgets** – Provides standard interface elements such as buttons, labels, sliders, and menus.
- **QtCore** – Includes core functionalities such as event handling and data structures.

- **QtGui** – Supports rendering, window management, and advanced graphical capabilities, essential for handling point cloud visualization.

A particularly important feature used in this thesis is the QVTK widget, which enables seamless integration of PCL's VTK-based rendering within the Qt interface. The Visualization Toolkit (VTK) [12] is a powerful open-source library for 3D computer graphics, image processing, and visualization. This was essential for displaying and interacting with 3D point cloud data in the developed application. Furthermore, Qt's popularity has led to a large community providing custom-made widgets, one of which, the RangeSlider, is utilized in this project to improve user interaction and parameter selection.

3.3 Implementation Details

The application is a Qt-based GUI tool for visualizing and analyzing point cloud data using the Point Cloud Library (PCL) and VTK for rendering. It manages two point clouds corresponding to two separate PCLVisualizer instances: one for the target cloud (left viewer) and one for the reference cloud (right viewer). All operations are applied with respect to these two clouds. See the Figure 3.2 for preview.

Probably the most fundamental function in the project is `getDistances()`, which computes the closest distance for each point in the first cloud to any point in the second cloud. It creates the PCL k-D tree from the cloud and then it gets closest distances thanks to the the PCL nearest search algorithm. The result is subsequently used for computing values of quality measures and colorization of target cloud. The exact implementation is provided in Algorithm 3.1.

```
1  std::vector<float> getDistances(PointCloudT::Ptr &cloud_a, PointCloudT
       ::Ptr &cloud_b) {
2
3      pcl::search::KdTree<PointT> tree_b;
4      tree_b.setInputCloud(cloud_b);
5
6      std::vector<float> distances(cloud_a->points.size());
7
8      for (size_t i = 0; i < cloud_a->points.size(); ++i) {
9          auto &point = cloud_a->points[i];
10         pcl::Indices indices(1); // To store index of the nearest
       point
11         std::vector<float> sqr_distances(1); // To store squared
       distance of the nearest point
12
```

```
13          tree_b.nearestKSearch(point, 1, indices, sqr_distances);
14
15          distances[i] = std::sqrt(sqr_distances[0]);
16      }
17      return distances;
18  }
19
```

Algorithm 3.1: Implementation of function `getDistances()` in C++.

When the user clicks the `Calculate` button (Figure 3.2 - G), the `onCalculate()` function handles the computation of quality measures. Depending on the selected measure, it calls the appropriate function from the following list:

- `computeHausdorffDistance(cloud1, cloud2)`
- `computeChamferDistance(cloud1, cloud2)`
- `computeEarthMoversDistance(cloud1, cloud2)`

These functions are implemented according to their definitions in Chapter 2. The computation of earth mover's distance uses the implementation from the OpenCV library.

Additionally, the application offers several key functionalities to enhance user experience and flexibility in handling point clouds:

Colorization: (Figure 3.2 – E) The application supports multiple colorization schemes, including *RGB*, $Yellow - Red$, *Grayscale*, $CMYK$, *Heatmap*, *Pastel*, and *Rainbow*. See Figure 3.1 for examples. The function `colorizeHandler()` automatically applies the selected color scheme to the target point cloud. Users can also utilize a range slider to adjust the colorization, allowing for better visualization of specific values within the distribution.

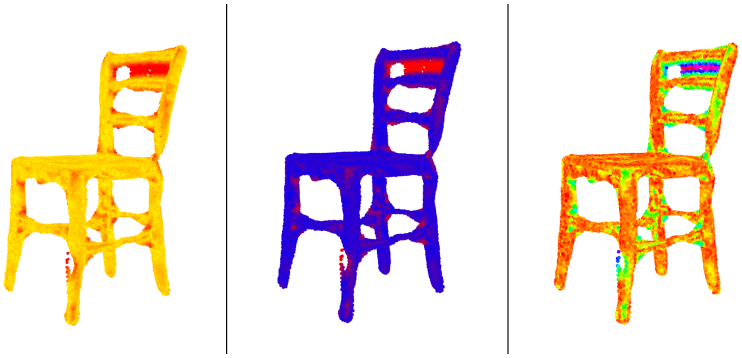

Figure 3.1: Example of different colorizations. $Yellow - Red$ on the left, *Heatmap* in the middle, *Rainbow* on the right.

Histogram: (Figure 3.2 – D) A histogram is provided to display the distribution of closest distances between loaded point clouds, offering a visual representation of their distribution.

Point size adjustment: (Figure 3.2 – A) A *QSlider* and *QDoubleSpinBox* are linked bidirectionally, allowing for smooth and precise adjustment of the point size for each viewer separately. The point size is updated dynamically in the respective viewer when changed.

Loading files: (Figure 3.2 – B) The application supports loading point cloud files in `.pcd`, `.obj`, `.ply`, and `.xyz` formats, with the first three handled by the PCL library and `.xyz` loaded via a custom implementation.

Export image: (Figure 3.2 – I) Users can export a PNG image of the clouds as displayed in their *viewer*. This feature captures the current view and saves it as an image file for further use or sharing.

Log: (Figure 3.2 – H) A log field stores information about the computed results, allowing users to track performed operations and their outcomes.

Camera views: (Figure 3.2 – C) The application allows users to select from predefined camera views or save their custom views for later use. This flexibility enables better control over how point clouds are visualized from different angles and perspectives.

Distance Exponent: (Figure 3.2 – F) A text edit field allows users to adjust the exponent applied to the Euclidean distance between points. Further information can be found in Section 4.4.

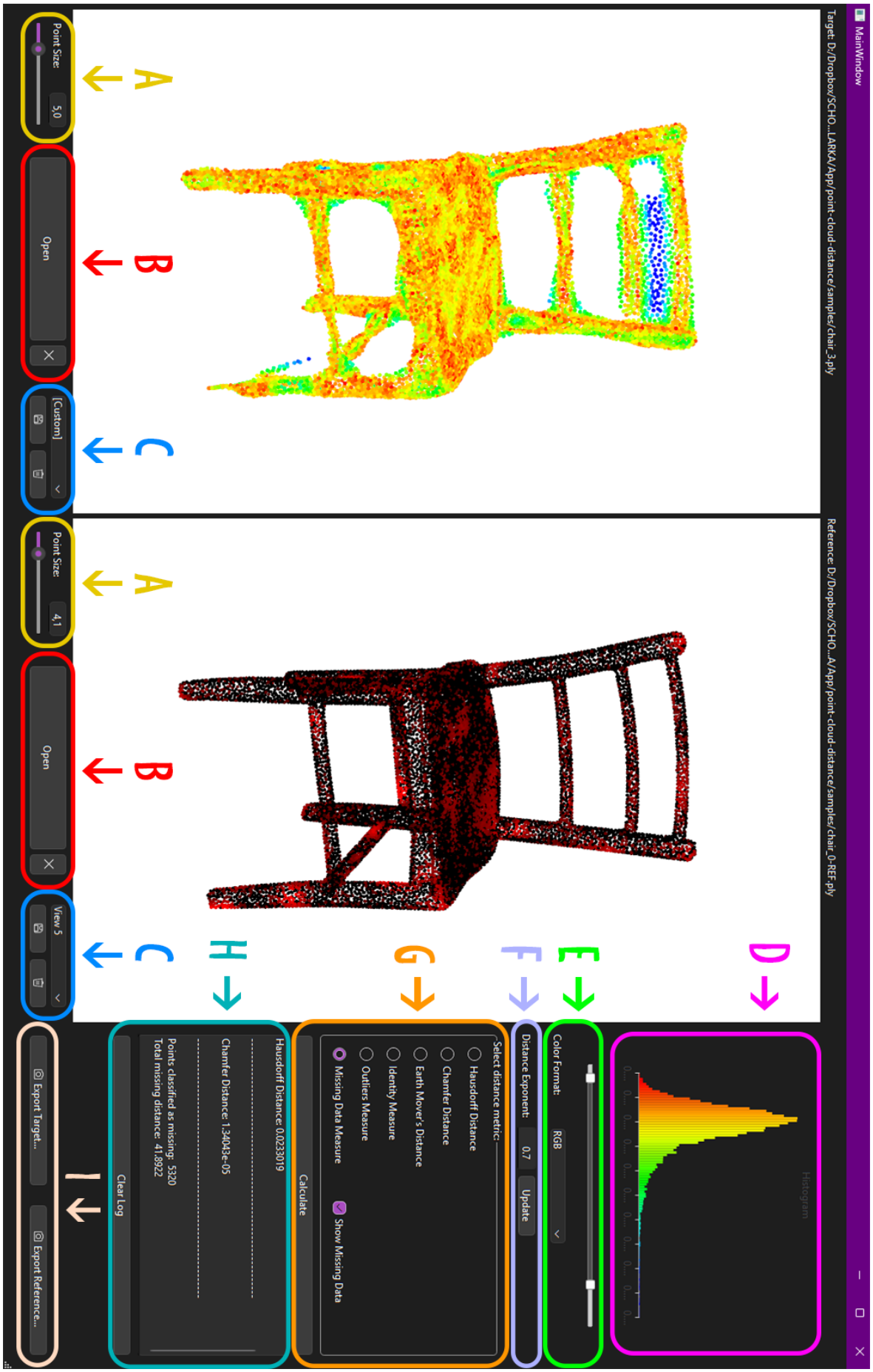

Figure 3.2: Screenshot of working application.

Chapter 4

Experiments and Modifications

Widely used methods like Hausdorff and chamfer distances are popular for comparing point clouds, but they work well only for specific cases. Hausdorff is sensitive to a single outlier, while chamfer distance mainly captures average distance. These methods do not tell much about other artifacts that occur in point clouds, which are shown in Figure 1.2.

This chapter introduces three new measures that provide some more targeted insights:

- **Identity Measure** shows how many points are exactly the same, which helps track how much a point cloud has changed.
- **Outliers Measure** captures how many points deviate strongly, not just if there is a single large error.
- **Missing Data Measure** estimates how much data is missing from the reference point cloud.

These measures are especially useful for identifying specific structural issues in point clouds and are designed to complement traditional tools.

4.1 Identity Measure

This measure evaluates how many points in one point cloud (`A`) are identical with the points in the reference cloud (`B`). A point (from cloud `A`) is considered identical if its nearest neighbor in cloud `B` lies at distance zero.

Definition: Let $D = \{d_1, d_2, \ldots, d_n\}$ be the set of distances from each point in `A` to its nearest neighbor in `B`. Then, the number of identical points is

$$\text{IdenticalCount} = \sum_{i=1}^{n} [d_i = 0]$$

$$\text{IdentityMeasure} = \frac{\text{IdenticalCount}}{n} \times 100\%$$

where $[d_i = 0]$ is 1 if $d_i = 0$, and 0 otherwise.

It can be helpful in detecting how many points remain unchanged or perfectly aligned between two versions of a point cloud.

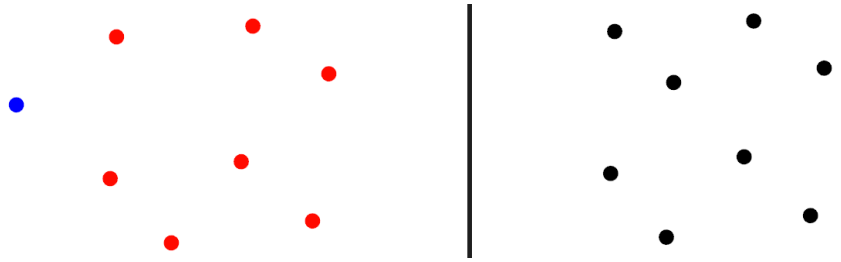

Figure 4.1: Identity measure example. Cloud `A` - left, Cloud `B` - right.

Figure 4.1 shows an example where:

- Total number of points in cloud `A`: $n = 8$
- Number of identical points (red dots): IdenticalCount $= 7$
- Therefore: IdentityMeasure $= 87.5\%$

4.2 Outliers Measure

As shown in Figure 4.2, the outliers measure highlights how much of the total discrepancy is caused by points that deviate significantly from the reference model, typically referred to as outliers. In this context, an outlier is defined as a point whose nearest neighbor distance exceeds twice the average nearest neighbor distance.

Definition: Let $D = \{d_1, d_2, \ldots, d_n\}$ be the set of distances from each point in cloud `A` to its nearest neighbor in cloud `B`, and let $\tilde{d}$ denote the median of these distances, i.e.

$$\tilde{d} = \begin{cases} d_{\frac{n+1}{2}}, & \text{if } n \text{ is odd} \\ \frac{1}{2}\left(d_{\frac{n}{2}} + d_{\frac{n}{2}+1}\right), & \text{if } n \text{ is even} \end{cases} \quad \text{(after sorting } D \text{ in ascending order)}$$

A point i is considered an outlier if:

$$d_i \geq 2\tilde{d}$$

The outliers measure is then defined as the sum of the distances of the outlier points:

$$\text{OutliersMeasure} = \sum_{i=1}^{n} \left[d_i \geq 2\tilde{d}\right] \cdot d_i$$

where $\left[d_i \geq 2\tilde{d}\right]$ is 1 if $d_i \geq 2\tilde{d}$, and 0 otherwise.

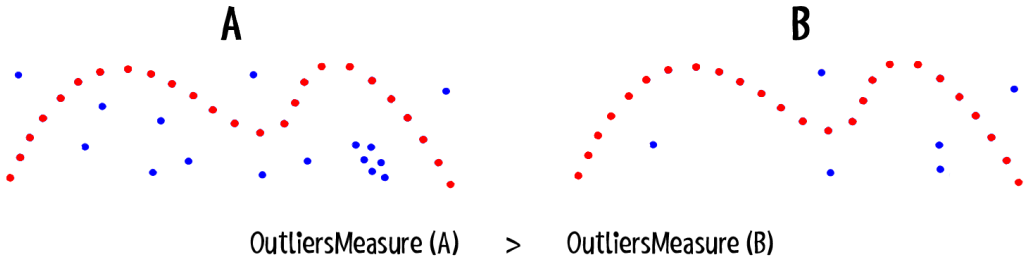

Figure 4.2: Outliers measure example. Blue points are outliers.

4.3 Missing Data Measure

The missing data measure tries to evaluate how well the reference cloud (B) is covered by the comparison cloud (A). It identifies areas where no suitable correspondence exists in A for points in B, indicating potentially missing or underrepresented regions in the reconstructed cloud.

Definition: Let $D = \{(i, j, d_i)\}$ be the set of all distances d_i from point $i \in$ B to nearest neighbor point $j \in$ A, and let D' be the sorted list of these distances in ascending order, where each d_i represents the nearest neighbor distance from point i in B to the nearest point j in A.

To compute the missing data measure:

1. Initialize all points in A as unassigned.
2. Iterate over sorted distances $d_i \in D'$:
   - If point $j \in$ A is unassigned, assign it to point $i \in$ B.
   - Otherwise, accumulate the distance d_i into the total missing penalty.

Let M be the set of distances where the assignment failed (i.e., the corresponding point in A was already assigned). The missing data measure is defined as:

$$\text{MissingDataMeasure} = \sum_{d \in M} d$$

It can be seen in Figure 4.3 that the measure is significantly larger in the top example because there are more missing parts in the point cloud. In contrast, the bottom example shows a more complete point cloud with fewer missing regions, resulting in a lower value of the measure.

To better understand where and how data might be missing, the application includes a feature that enables visualization of the missing data measure. In this visualization, points on the reference cloud (B) that could not find a unique match are colorized. The color intensity is based on the corresponding distance d_i — the larger the distance,

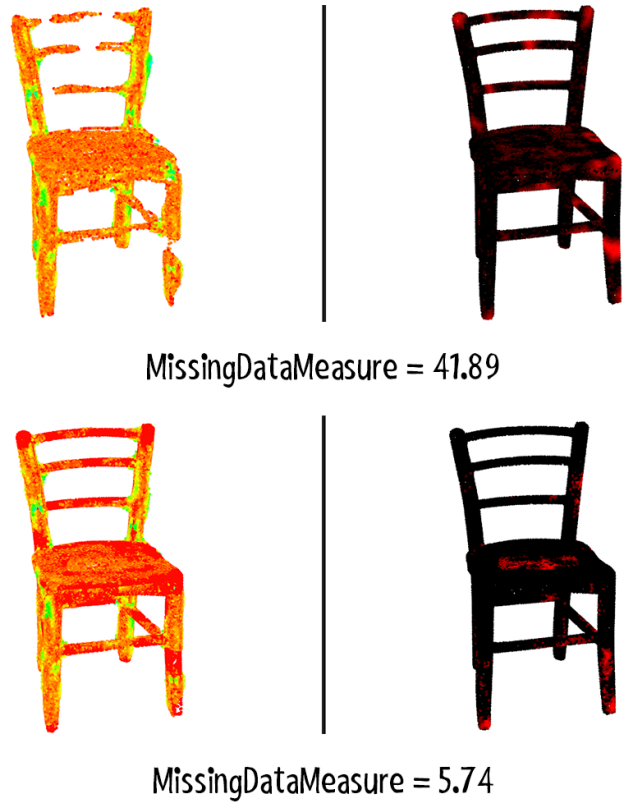

Figure 4.3: Missing data measure example. Cloud `B` (reference) - right, Cloud `A` - left.

the more intense the red color. Only unassigned points from the set M (i.e. those contributing to the missing measure) are colorized. This visual feedback helps to intuitively grasp the severity and spatial distribution of missing or damaged regions.

Such visualization can serve as a valuable qualitative diagnostic tool, particularly in preprocessing pipelines or registration algorithms where detecting incompleteness is critical. For example, in a factory line scanner system, where products such as boxes are checked for structural integrity, a reference point cloud of a perfect box could be compared against live scans. If a box on the assembly line has a hole or deformation, the missing data measure would highlight the problematic region with strong red markings, allowing immediate identification of defects. To better illustrate specific missing regions, Figure 4.4 shows a detailed view with highlighted areas where missing data is present.

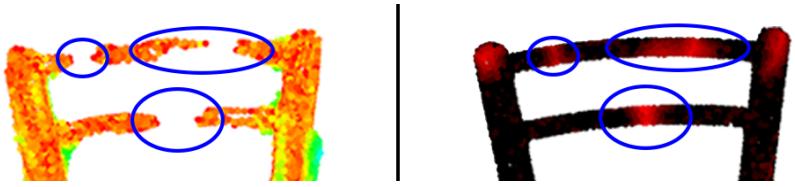

Figure 4.4: Detailed visualization of missing parts. Highlighted blue circles indicate areas with missing data.

4.4 Distance Exponent

The distance exponent is a user-adjustable parameter that modifies the computed distance by raising it to a specified power.

Definition: The distance is computed as:

$$d = \left(\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}\right)^{\text{exponent}}. \tag{4.1}$$

The default value of the exponent is 1, which results in the standard Euclidean distance. Modifying the exponent changes how distances are scaled, which affects the resulting values and their distribution in the histogram. For example, an exponent greater than 1 amplifies larger distances while diminishing the relative importance of smaller distances. Conversely, an exponent less than 1 compresses the range of distances, making larger and smaller distances appear closer together.

An illustrative example is shown in Figure 4.5, where results with exponents 0.2, 1, and 2 are compared. When using exponent = 2, larger gaps or deviations become much more pronounced, while exponent = 0.2 reduces the influence of larger distances, compressing the differences between small and large deviations.

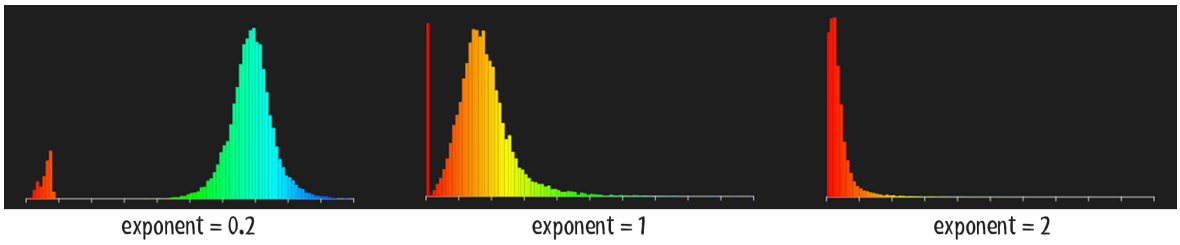

Figure 4.5: Effect of the distance exponent.

Adjusting the exponent is particularly useful in several scenarios:

- **Emphasizing severe deviations:** In applications where detecting large errors or missing regions is critical, increasing the exponent highlights such areas more strongly.
- **Smoothing noise:** Lowering the exponent (e.g., using values between 0 and 1) can make small random noise less significant in the resulting measures.
- **Adjusting histogram distributions:** By modifying the exponent, users can effectively shift and reshape the histogram of distances, making it easier to analyze the underlying data distribution and spot outliers or trends.
- **Custom scaling for specific applications:** Some domains may require nonlinear distance scaling for technical or perceptual reasons, and the exponent provides a simple mechanism for this adjustment.

Thus, the distance exponent adds flexibility to the distance computation, enabling better customization for different analysis needs.

4.5 Testing and Evaluation

To evaluate the behavior of the implemented similarity measures under different types of disturbances in point clouds, we conducted a series of qualitative tests. Each scenario highlights specific strengths or sensitivities of the individual metrics. Below are visualizations and commentary for several representative cases.

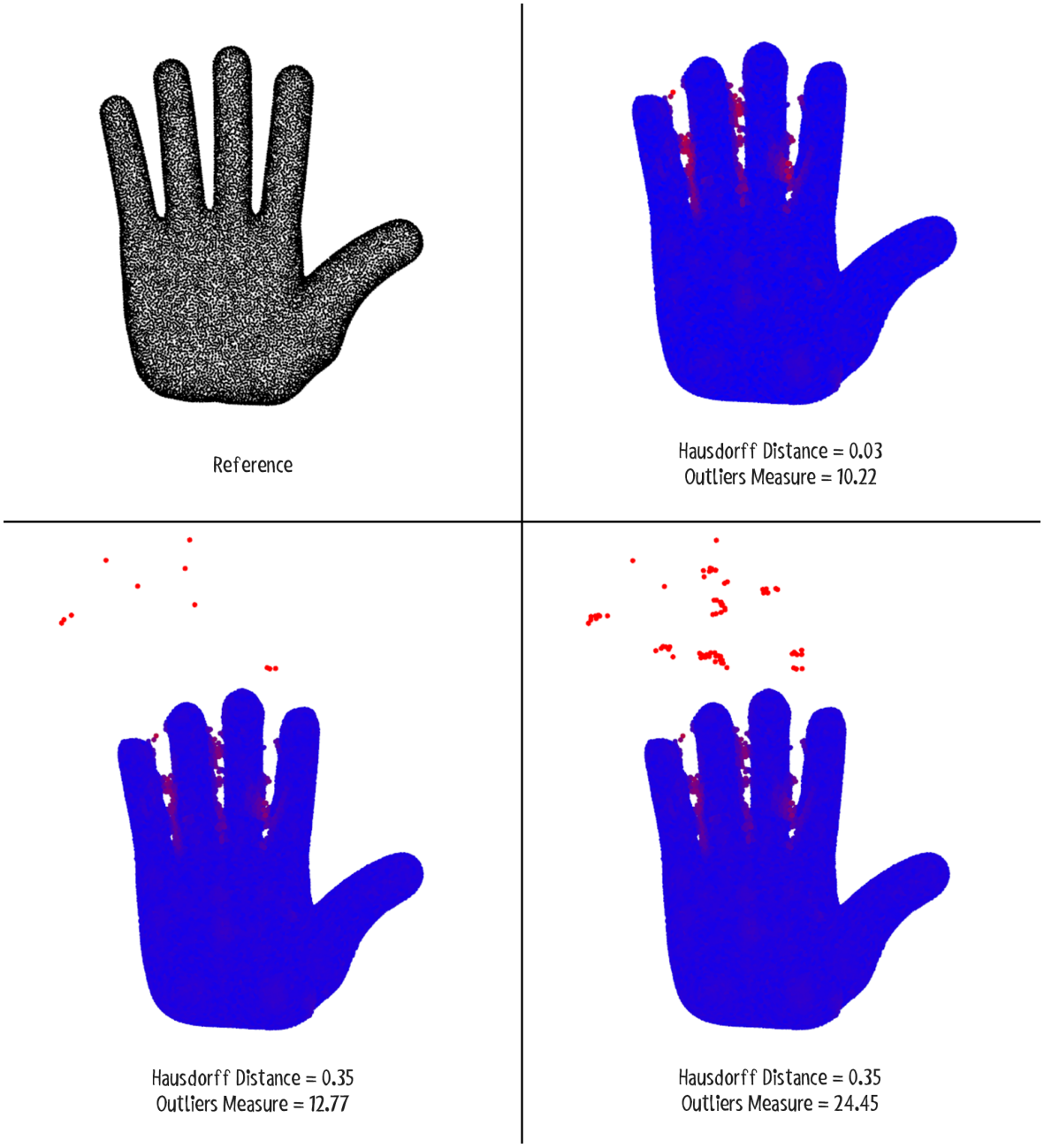

Figure 4.6: Visualization with added outliers.

In first scenario, a small number of points (outliers) were added to one of the point clouds. The Hausdorff distance is particularly sensitive to such outliers, as it reports the maximum distance between any point in one cloud to the nearest point in the other. As shown in Figure 4.6, a single far-away point can significantly inflate the Hausdorff distance value.

On the other hand, the custom Outliers measure focuses on the overall presence and distribution of distant points, producing higher values when more such outliers are present. Unlike the Hausdorff metric, it is not dictated by a single extreme value but reflects the cumulative deviation due to scattered anomalies.

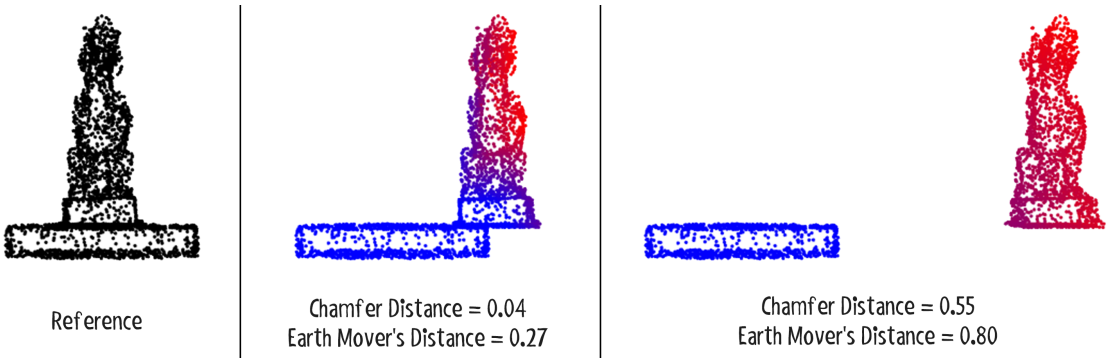

Figure 4.7: Visualization with translated cloud.

Figure 4.7 shows one cloud being shifted from the center. In such cases, the chamfer distance reflects an average minimal matching cost and increases proportionally with the shift, but without considering global structure. It simply matches each point to its closest counterpart.

In contrast, the earth mover's distance (EMD) captures this global translation more naturally. Since EMD models the minimal cost of transforming one distribution into another, it accumulates the necessary work to "move mass" between corresponding regions. As a result, EMD grows more consistently and meaningfully in such translation scenarios.

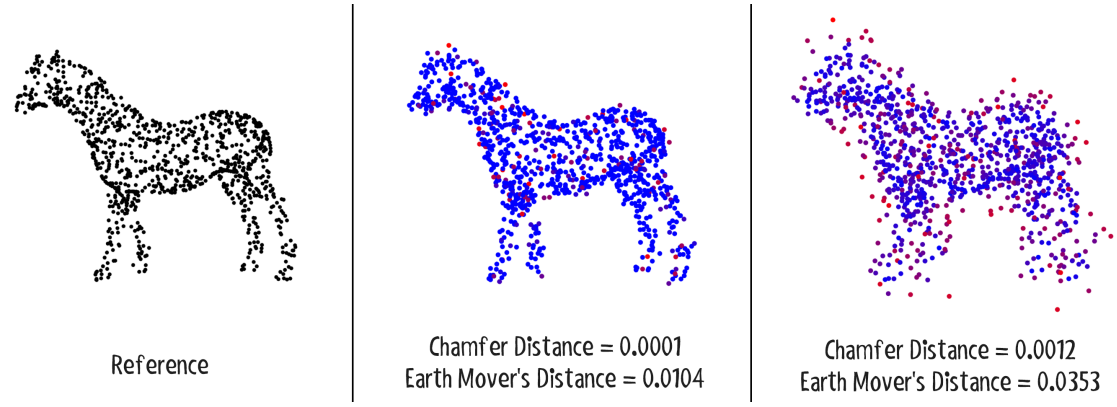

Figure 4.8: Visualization with added noise.

Figure 4.8 demonstrates the effect of introducing random noise into one of the point clouds. The leftmost image shows the original reference point cloud of a horse. In the middle image, we see a point cloud that closely matches the reference, with minimal noise. As expected, both the chamfer distance and the EMD are very low—indicating high similarity. However, in the rightmost image, random noise has been added to the point positions. This causes visible degradation in shape fidelity, which is reflected in the increased values of both distance measures.

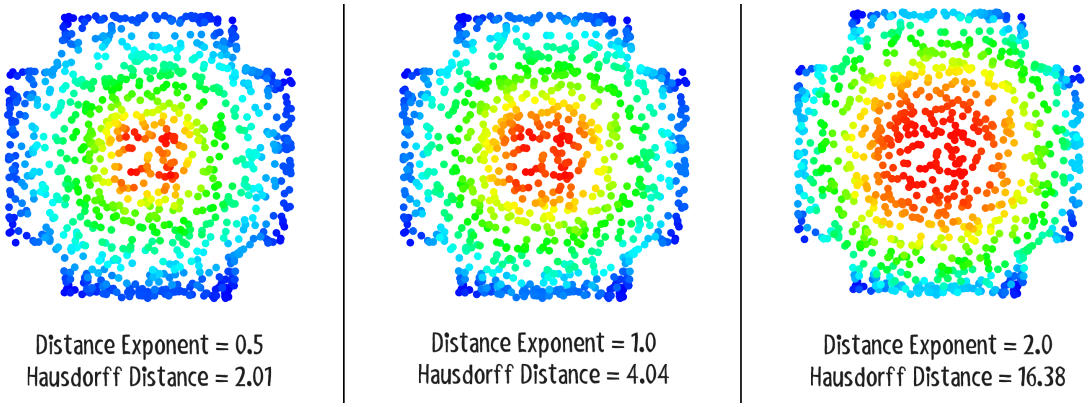

Figure 4.9: Effect of distance exponent.

Figure 4.9 illustrates how varying the distance exponent influences the computed Hausdorff distance. The reference object consists of four points located at the center. The distance exponent controls how distance contributions are aggregated, effectively amplifying or reducing the influence of larger distances. In this example, increasing the exponent from 0.5 to 1.0 to 2.0 causes the Hausdorff distance to rise from 2.01 to 4.04 to 16.38, demonstrating a strong sensitivity to outliers at higher powers.

This exponentiation mechanism is not limited to Hausdorff distance. As discussed in Section 4.4, the distance exponent can be applied to other similarity measures as well. By tuning the exponent, users can adjust the robustness or sharpness of the metric according to the nature of the data and the requirements of their application.

Conclusion

The primary objective of this thesis was to design and implement a flexible and user-friendly application for comparing 3D point clouds using a variety of quality measures and intuitive visualizations. As a result of this work, a desktop software tool was created that enables users to load two point clouds, compute distances between them using various established and custom-developed metrics, and visualize the spatial distribution of differences through both color mapping and histograms. The application was built with modularity and extensibility in mind, allowing it to serve as a foundation for future research or software projects in the domain of 3D data analysis.

One of the key contributions of this thesis is the implementation of custom distance measures tailored to specific comparison needs. These custom measures complement traditional methods like Hausdorff distance by offering alternative ways of quantifying similarity or deviation, which are often more relevant in practical applications. The software has a broad range of potential uses — it can assist researchers in evaluating the effectiveness of new reconstruction algorithms, support engineers in identifying defects in 3D-scanned products, or help developers assess the trade-offs introduced by compression and simplification techniques.

All of the intended goals have been fulfilled. The tool is not only functional and accurate, but it also provides valuable insights through a clear and interactive graphical interface. It allows users to export image results for further processing and offers enough flexibility to be adapted for diverse use cases, from scientific research to industrial inspection.

Nonetheless, there are some limitations that need to be acknowledged. In particular, the earth mover's distance, while offering a more perceptually meaningful comparison, is computationally demanding and becomes impractical for large point clouds. On such inputs, the computation may be excessively slow or even lead to application crashes due to high memory usage. This underscores the need for further optimization or approximation techniques when dealing with large-scale datasets.

Future development of the tool could take several promising directions. One such direction is the implementation of additional distance measures, such as those based on the Iterative Closest Point (ICP) algorithm, which could further enhance the robustness and utility of the tool. Another possible improvement is the integration of the

application into existing 3D processing software like MeshLab, which would broaden its accessibility and adoption. Furthermore, creating a web-based version of the tool would make it easier to use in collaborative or cloud-based environments. Finally, performance improvements—especially regarding memory management and algorithmic efficiency—would enable the application to handle much larger point clouds with minimal resource consumption.

In conclusion, this thesis presents a valuable software solution for point cloud quality assessment, offering a combination of precision, flexibility, and ease of use. It lays a strong foundation for both academic research and practical applications in the field of 3D data analysis, and opens the door for future enhancements that could further expand its capabilities and impact.

Bibliography

[1] Mustafa Akgül. The linear assignment problem. In *Combinatorial Optimization: New Frontiers in Theory and Practice*, pages 85–122. Springer, 1992.

[2] Matthew Berger, Andrea Tagliasacchi, Lee M Seversky, Pierre Alliez, Gael Guennebaud, Joshua A Levine, Andrei Sharf, and Claudio T Silva. A survey of surface reconstruction from point clouds. In *Computer graphics forum*, volume 36, pages 301–329. Wiley Online Library, 2017.

[3] Paolo Cignoni, Marco Callieri, Massimiliano Corsini, Matteo Dellepiane, Fabio Ganovelli, and Guido Ranzuglia. MeshLab: an Open-Source Mesh Processing Tool. In Vittorio Scarano, Rosario De Chiara, and Ugo Erra, editors, *Eurographics Italian Chapter Conference*. The Eurographics Association, 2008.

[4] Hristo Hristov. Introduction to k-D trees. Baeldung, March 2023. Url: https://www.baeldung.com/cs/k-d-trees. Accessed: 2025-01-19.

[5] Daniel P Huttenlocher, Gregory A. Klanderman, and William J Rucklidge. Comparing images using the Hausdorff distance. *IEEE Transactions on pattern analysis and machine intelligence*, 15(9):850–863, 1993.

[6] Marco Imperoli and Alberto Pretto. D co: Fast and robust registration of 3D textureless objects using the directional chamfer distance. In *International conference on computer vision systems*, pages 316–328. Springer, 2015.

[7] Harold W Kuhn. The Hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.

[8] Donald Meagher. Geometric modeling using octree encoding. *Computer graphics and image processing*, 19(2):129–147, 1982.

[9] Yossi Rubner, Carlo Tomasi, and Leonidas J Guibas. The earth mover's distance as a metric for image retrieval. *International journal of computer vision*, 40:99–121, 2000.

[10] Radu Bogdan Rusu and Steve Cousins. 3D is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 9-13 2011. IEEE.

[11] Alexander Schrijver. On the history of the transportation and maximum flow problems. *Mathematical programming*, 91:437–445, 2002.

[12] Will Schroeder, Ken Martin, and Bill Lorensen. *The Visualization Toolkit: An Object-Oriented Approach to 3D Graphics.* Kitware, Inc., 2006. Url: https://vtk.org. Accessed: 2025-03-16. Version 9.3.

[13] Simsangcheol. Chamfer distance. *Medium*, Feb 2023. Url: https://medium.com/@sim30217/chamfer-distance-4207955e8612. Accessed: 2025-01-21.

[14] The Qt Company. *Qt Framework*, 2025. Url: https://www.qt.io. Version 6.x.

[15] Dimitri van Heesch. Doxygen: Source code documentation generator, 1997. Url: https://www.doxygen.nl. Accessed: 2025-04-27.

[16] Cédric Villani et al. *Optimal transport: old and new*, volume 338. Springer, 2009.

[17] Tong Wu, Liang Pan, Junzhe Zhang, Tai Wang, Ziwei Liu, and Dahua Lin. Density-aware chamfer distance as a comprehensive metric for point cloud completion. In *Proceedings of the 35th International Conference on Neural Information Processing Systems*, pages 29088–29100, 2021.