MASTER THESIS

Improved Normal-Guided Pointcloud Denoising through Feature Detection and Update Strategies

Exploring point classification and normal-guided update strategies for improved pointcloud reconstruction.

Supervisors:
Dr. Klaus Hildebrandt
Dr. Ruben Wiersma

Author:
Ruben Band
https://github.com/Ruubje/Normal-Guided-Pointcloud-Denoiser

August 5, 2025

Acknowledgements

I would like to express my gratitude to my supervisor, Klaus Hildebrandt, for his continuous guidance and theoretical insights throughout the three years of this thesis project. His expertise and patience have been very valuable. I am also deeply thankful to Ruben Wiersma for his practical support and thoughtful feedback. Even after leaving TU Delft, his willingness to stay involved and support me has meant a great deal. Special thanks go to Marc Meijnema for being my study partner at the start of this journey. Working together on our own thesis helped me develop an effective workflow. I would like to thank everyone I've rowed with at my rowing association. EJL24 and the members of Opzet kept me physically and mentally strong during the times when my thesis felt endless. Your discipline and dedication kept me track. To my girlfriend, Marije Meijndert, from the start until the finish, thank you for your unwavering support and all the fun moments that gave me energy to push forward. Finally, I want to thank my friends and family for their encouragement and support throughout these three years. I couldn't have done it without you.

Abstract

This thesis presents an improved normal-guided pointcloud denoising pipeline that enhances the quality and efficiency of 3D pointcloud reconstruction. Building on the Constraint-based Point Set Denoising (CPSD) method by (Yadav et al., 2018), several modifications and extensions are proposed to improve the denoising process. The key contributions include a revised point classification approach, dedicated point update formulas for different point classes and a pipeline optimization and evaluation. Experiments were performed on synthetic and real-world scanned datasets, using Chamfer Distance (CD) and single-sided Chamfer Distance (sCD) as evaluation metrics. Results demonstrate that the proposed method achieves lower error scores than existing pipelines while requiring fewer iterations. Additionally, the modular nature of the pipeline enables future integration of neural networks or curvature-aware point update functions, opening pathways for further improvements in denoising pipelines.

Contents

2

Chapter 1

Introduction

pointclouds are a fundamental data representation in various fields such as autonomous driving (Li et al., 2020), robotics, virtual reality (Stets et al., 2017) and cultural heritage preservation (Garagnani and Manferdini, 2013). pointclouds capture spatial information in the form of a set of points with d-dimensional coordinates. In our case, we look at pointclouds in 3-dimensional space that represent an underlying curved surface. This information can be captured by sensors like LiDAR (Dong and Chen, 2017), structured light (Rocchini et al., 2001) or stereo cameras (Leberl et al., 2010). These methods introduce noise in the form of environmental factors (Charron et al., 2018), sensor limitations (Gokturk et al., 2004) or motion distortion (Yang et al., 2021; Chen et al., 2021). Noise in the pointcloud reduces the fidelity of the underlying surface representation and negatively impacts the performance of downstream processing tasks. In applications such as autonomous driving, for instance, noisy pointclouds can result in inaccurate obstacle detection or a misinterpretation of the surrounding environment (Luo and Hu, 2021b).

pointcloud denoising is a fundamental preprocessing step that refines the raw data so that the distance between the points and the underlying surface is smaller. Effective denoising methods should smooth flat or curved parts of the surface and retain sharp features such as edges and corners. Recent research uses normal guided denoising methods and replaces components of existing methods with neural networks to improve the results (Wei et al., 2021; Li et al., 2022). Nevertheless, this work focuses on improving the classical pipeline itself. During early experiments, it became evident that existing normal-guided methods can struggle even under ideal conditions, revealing opportunities to enhance their robustness and performance. This thesis explores those opportunities and proposes a refined pipeline that improves denoising quality.

A representative example of sharp feature preserving normal-guided denoising methods is proposed by (Yadav et al., 2018). They propose a method where the noise in the point normals is filtered out by binary eigenvalue optimization, the points are classified as flat, edge, or corner points, and different point updating schemes are applied for the different point classes. This method is among the earliest normal-guided pointcloud denoising approaches to effectively address sharp feature preservation without relying on multiple normals per point. Its simple and

interpretable design, along with a modular pipeline structure, made it an ideal foundation for systematic experimentation of individual components and the exploration of alternative techniques per component to improve overall performance.

This thesis will look at the normal guided denoising method from (Yadav et al., 2018) and proposes an updated version that needs fewer iterations and improves the performance. The contributions of this work to the existing pipeline are the following:

- **Revised feature-based point classification.** Improved feature extraction and classification approach for a more accurate labeling points as flat, edge, or corner, enabling tailored denoising strategies for each class.
- **Revised point update strategy.** Developed a position updating framework with adaptive diffusion speeds and class-dependent update formulas, resulting in better preservation of geometric features while reducing required iterations.
- **Pipeline optimization and evaluation.** Designed a more computationally efficient denoising pipeline and benchmarked it against existing methods, demonstrating improved reconstruction accuracy and robustness across different noisy pointclouds.

By introducing this updated method, this thesis tries to push the boundaries of pointcloud processing by improving on fundamental components of the pointcloud denoising pipeline.

Chapter 2

Related Work

Numerous methods have been proposed to address the challenge of denoising pointclouds. In this section, we review deterministic approaches and learning-based techniques, highlighting their principles, advantages, and limitations. Understanding these existing approaches provides a foundation for motivating the contributions of our proposed method.

Deterministic methods

There are multiple deterministic methods for pointcloud denoising. A simple method applies the bilateral filter. The bilateral filter was originally developed for image processing (Tomasi and Manduchi, 1998). To smooth a pixel, it compares the intensity values of neighboring pixels, assigning weights based on both spatial proximity and similarity. A weighted average of these neighbors is then computed to smooth the pixel. The bilateral filter for pointclouds operates in a similar manner. Each point is smoothed by computing a weighted average of its neighbors, where the weights are based on spatial proximity and similarity. Spatial proximity is measured using Euclidean distance, while similarity is determined in different ways, often using normal vectors. One of the first ways is by comparing the normal vectors with an inner product (Miropolsky and Fischer, 2004). Another way, as discussed by (Digne and De Franchis, 2017), is defining the similarity as the neighbor's distance to the tangent plane at the current point. Another well-known deterministic method is Moving Least Squares (MLS) (Levin, 1998), which was used as inspiration for pointcloud denoising by (Alexa et al., 2001), where a local neighborhood is selected for each point, and a local reference frame is computed. Within this frame, a polynomial surface is fitted to approximate the underlying geometry. The original point is then projected onto this surface. While MLS is effective at smoothing curved surfaces, it often blurs sharp features. Another example is Locally Optimal Projection (LOP) (Lipman et al., 2007), which updates point positions based on a balance of attraction and repulsion forces from neighboring points. Weights based on spatial proximity are used to define the influence of each neighbor. LOP simplifies the process by avoiding the need for normal estimation, plane fitting or polynomial fitting, but like MLS, it struggles to preserve sharp geometric features due to its smoothing properties. To

address this limitation, (Sun et al., 2015) proposed a method based on L_0 minimization (Zhang et al., 2013), which aims to better preserve sharp features. Their approach formulates pointcloud denoising as an optimization problem that minimizes the L_0 norm of point deviations from local approximated tangent planes. This encourages piecewise smoothness while allowing for discontinuities, making it well-suited for preserving edges and corners in the geometry. However, due to the non-convex nature of the L_0 norm, the optimization is computationally expensive.

Learning-based methods

In addition to deterministic approaches, numerous learning-based methods have been proposed for pointcloud denoising. Even though this thesis will not focus on them, they deserve to be mentioned, since their performance on the denoising task are outstanding. These methods often aim to enhance or replace specific components of traditional pipelines to improve performance or robustness. Both supervised and unsupervised learning techniques have been explored for pointcloud denoising. Supervised models, such as PointNet (Qi et al., 2017), directly predict clean point positions from noisy inputs or estimate more accurate point normals (Boulch and Marlet, 2016). Unsupervised methods, by contrast, look at patterns within the data itself. Autoencoders, for example, learn latent representations that capture the underlying structure of pointclouds without requiring labeled data (Hermosilla et al., 2019). A more recent class of unsupervised models is diffusion models, which learn to reverse a gradual noising process to generate or recover point sets from corrupted inputs. Notable examples include Diffusion Probabilistic Models for 3D pointcloud Generation (Luo and Hu, 2021a) and LION (Vahdat et al., 2022), both of which show strong potential in learning robust and flexible denoising strategies. Overall, learning-based approaches offer powerful alternatives, especially in scenarios where hand-made rules fall short.

Normal guided methods

The method proposed in this thesis follows the normal-guided denoising paradigm, which draws inspiration from the concept of guided filtering introduced in image processing by (He et al., 2012). In guided filtering, a separate guidance signal is used to preserve important structures while removing noise assuming a linear transformation between the guidance image and the denoised result. Unlike the bilateral filter, which directly uses the guidance signal (or intensity) to produce a denoised output, the guided filter constructs a local linear model guided by the signal and computes the output by minimizing a cost function based on this local linear model. This provides more freedom in designing guidance signals and cost functions, enabling the development of a well-structured pipeline tailored to specific goals, such as preserving edges and corners in the denoising process. This concept is adapted to pointclouds by first using the pointcloud itself as the guidance signal (Han et al., 2018a). Shortly thereafter, using surface normals was proposed as a more informative guidance cue (Han et al., 2018b; Yadav et al., 2018), where normals were iteratively refined and used to guide the point updating process. However, it

was soon recognized that a single normal vector per point was insufficient for capturing complex geometric configurations near sharp features like edges and corners. To address the problem of smoothing at sharp features, multi-normal guided methods were introduced (Zheng et al., 2017; Liu et al., 2020), allowing each feature point to be associated with multiple normals corresponding to intersecting surface regions. While this approach improves feature preservation, it introduces computational overhead. Since (Yadav et al., 2018) achieves feature preservation in a similar fashion, our methodology will focus on their pipeline to improve upon.

Chapter 3

Background

Since this thesis aims to explore and build upon existing normal-guided denoising methods, it is essential to first develop a detailed understanding of these approaches. This section begins by clarifying neighborhood construction notation. Following that, we provide an in-depth explanation of the full denoising pipeline proposed by (Yadav et al., 2018) and finally, we examine the point updating scheme introduced in (Wei et al., 2021), which serves as an additional updating scheme for comparison.

3.1 Neighborhood Construction

We will introduce notation to distinguish two different neighborhood construction methods to create clarity when referring to neighborhoods. A K-nearest neighbors (KNN) neighborhood is defined as the set of K points that are closest to a given point v_i in terms of Euclidean distance. In contrast, a range-based neighborhood of point v_i includes all points whose Euclidean distance to v_i is less than or equal to a predefined radius r. We denote the KNN neighborhood of point i by Ω_i^k and the range-based neighborhood of point i by Ω_i^r.

3.2 Normal Guided Denoising Pipeline

In general, normal guided pointcloud denoising methods generate normal vectors for each point and use those as a guiding signal for denoising the pointcloud. Then, the methods do a predefined number of denoising iterations. An iteration consists of first denoising the normal vectors and then the points themselves. This thesis will modify the normal guided denoising pipeline from (Yadav et al., 2018) and therefore it is important to understand the pipeline. The pipeline, illustrated in Figure 3.1, consists of four steps: Normal Estimation, where initial normals are estimated per point; Normal Smoothing, where the normals are denoised; Feature Detection, where each point is classified; and Position Updating, where each point is updated with a formula chosen based on the classification. The algorithm iterates over the last three steps. After a pre-defined number

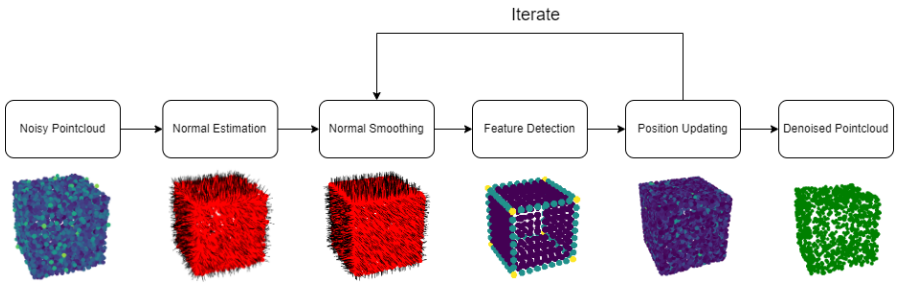

Figure 3.1: Flowchart of the pipeline of pointcloud denoising. (Yadav et al., 2018)

of iterations, the pointcloud is considered to be denoised. In the next sections, the steps will be explained in more detail.

3.2.1 Normal Estimation

To estimate these normals, the method from (Hoppe et al., 1992) is used, in which a Point Voting Tensor (PVT)

$$\mathbf{T}_i^P = \sum_{j \in \Omega_i^k} (\mathbf{v}_j - \bar{\mathbf{v}}) (\mathbf{v}_j - \bar{\mathbf{v}})^T \tag{3.1}$$

is created, where $\bar{\mathbf{v}} = \frac{1}{|\Omega_i^k|} \sum_{j \in \Omega_i^k} \mathbf{v}_j$ is the center of mass of the neighborhood. After eigendecomposition of the voting tensor, the eigenvector with the smallest eigenvalue is chosen to be the normal vector. To orient the normal vectors in the same direction, an algorithm is applied that creates and traverses a minimal spanning tree. While traversing, it compares and flips the normal vectors to point to similar directions by minimizing the angle between the normal vectors. Normal estimation is only done once to create an initial normal field to smooth over iterations.

3.2.2 Normal Smoothing

In this step, the normal field as guidance signal is smoothed to better represent the normal direction of the underlying surface. Normal smoothing is done as the first step of each iteration to remove noise components from the point normals. Binary Eigenvalues Optimization (BEO) creates a normal voting tensor (NVT)

$$\mathbf{T}_i^N = \sum_{j \in \Omega_i^r} w_{ij} \mathbf{n}_j \mathbf{n}_j^T, \tag{3.2}$$

from which a smoothed normal vector can be calculated. The weights are defined as binary values that determine whether or not to include neighboring votes, based on the angle between the normal vectors. The weights are defined as

$$w_{ij} = \begin{cases} 1 & \text{if } \angle(\mathbf{n}_i, \mathbf{n}_j) \leq \rho \\ 0 & \text{if } \angle(\mathbf{n}_i, \mathbf{n}_j) > \rho \end{cases}, \tag{3.3}$$

where $\angle(\cdot,\cdot)$ denotes an angle between two vectors and $\rho \in (0,\pi)$ is the angular threshold. Since $\mathbf{T}_i^N$ is a symmetric and positive semidefinite matrix, eigendecomposition can be applied to obtain

$$\mathbf{T}_i^N = \sum_{l=1}^{3} \lambda_{i,l} \mathbf{x}_{i,l} \mathbf{x}_{i,l}^T, \tag{3.4}$$

where $\lambda_{i,l}$ is the eigenvalue with its corresponding eigenvector $\mathbf{x}_{i,l}$, such that $\lambda_{i,1} \leq \lambda_{i,2} \leq \lambda_{i,3}$. BEO modifies the summation above such that the eigenvalues are binary. The modified eigenvalues are defined as

$$\tilde{\lambda}_{i,l} = \begin{cases} 1, & \text{if } \lambda_{i,l} > \tau \\ 0, & \text{if } \lambda_{i,l} \leq \tau \end{cases}, \tag{3.5}$$

where τ is similar to the noise intensity. With the binary eigenvalues, the binary normal voting tensor is calculated as

$$\tilde{\mathbf{T}}_i^N = \sum_{l=1}^{3} \tilde{\lambda}_{i,l} \mathbf{x}_{i,l} \mathbf{x}_{i,l}^T. \tag{3.6}$$

The new denoised normal is defined as

$$\tilde{\mathbf{n}}_i = d\mathbf{n}_i + \tilde{\mathbf{T}}_i^N \mathbf{n}_i, \tag{3.7}$$

where $d = 3$ is the dampening factor and the final vector is normalized. The last step is equivalent to an interpolation between the denoised and the original normal. The normal $\tilde{\mathbf{n}}_i$ is the new filtered or denoised normal of the point i and is used in the next section.

3.2.3 Feature Detection

There are various point position updating schemes in normal-guided denoising, each with distinct effects. Some promote smoothing, while others cause points to snap toward feature locations. Applying these indiscriminately can be counterproductive, because smoothing at feature points can blur important details, while forcing points to jump to features when they are not nearby any features may introduce artifacts. Therefore, it is crucial to select the appropriate updating strategy for each point. To do so, we need a classification that tells us what points need which update formula. The classification must identify geometric features such as edges and corners. For detecting features, a filtered PVT is created similar to 3.1 with

$$\tilde{\mathbf{T}}_i^P = \frac{1}{\sum_{j \in \Omega_i^r} \tilde{w}_{ij}} \sum_{j \in \Omega_i^r} \tilde{w}_{ij} (\mathbf{v}_j - \bar{\mathbf{v}}) (\mathbf{v}_j - \bar{\mathbf{v}})^T, \tag{3.8}$$

where

$$\tilde{w}_{ij} = \begin{cases} 1 & \text{if } \angle(\tilde{\mathbf{n}}_i, \tilde{\mathbf{n}}_j) \leq \rho \\ 0 & \text{if } \angle(\tilde{\mathbf{n}}_i, \tilde{\mathbf{n}}_j) > \rho \end{cases} \tag{3.9}$$

is the same binary weight as 3.3, but with the denoised normal $\tilde{\mathbf{n}}_i$ as input instead of the original normal $\mathbf{n}_i$ and $\bar{\mathbf{v}} = \frac{1}{\sum_{j \in \Omega_i} \tilde{w}_{ij}} \sum_{j \in \Omega_i^r} \tilde{w}_{ij} \mathbf{v}_j$ is the center of mass of the points included in the calculations

by the binary weights. Because the voting tensor $\tilde{\mathbf{T}}_i^P$ is a symmetric positive semidefinite matrix, eigendecomposition can be applied to retrieve the eigenvectors $\mathbf{y}_{i,l}$ with corresponding eigenvalues $\mu_{i,l}$ where $\mu_{i,1} \leq \mu_{i,2} \leq \mu_{i,3}$ such that

$$\tilde{\mathbf{T}}_i^P = \sum_{l=1}^{3} \mu_{il} \mathbf{y}_{il} \mathbf{y}_{il}^T. \tag{3.10}$$

To classify the points, the number of eigenvalues above a certain threshold τ is counted. This will result in the set of flat points

$$V^f = \{v_i \in V | \mu_{i,1}, \mu_{i,2} \geq \tau \wedge \mu_{i,3} < \tau\}, \tag{3.11}$$

the set of edge points

$$V^e = \{v_i \in V | \mu_{i,1} \geq \tau \wedge \mu_{i,2}, \mu_{i,3} < \tau\} \tag{3.12}$$

and the set of corner points

$$V^c = \{v_i \in V | \mu_{i,1}, \mu_{i,2}, \mu_{i,3} \geq \tau \vee \mu_{i,1}, \mu_{i,2}, \mu_{i,3} < \tau\}. \tag{3.13}$$

This division of points into classes is used for updating the position of the point, which is explained in the next section.

3.2.4 Position Updating

After point classification, the position of the points are updated according to their classification. (Yadav et al., 2018) proposes an update method for corner points, edge points, and flat points. For each type of point, a restricted quadratic error metric is formulated and the goal is to minimize the error by calculating the optimal position for the point. In the following sections, the used metric will be explained and how the point should be updated to achieve a minimal error.

Optimal Position For Corner Points

For corner points $\mathbf{v}_i^c \in V^c$, the idea is to create tangent planes for all neighboring points based on their smoothed normal $\tilde{\mathbf{n}}_j$. The error is defined as the distance to the tangent planes, which results in the loss function

$$E^c(\mathbf{t}_i^c) = \sum_{j \in \Omega_i^r} ||\tilde{\mathbf{n}}_j \cdot (\mathbf{t}_i^c - \mathbf{v}_j)||^2. \tag{3.14}$$

To find the new position $\mathbf{t}_i^c$ for which $E^c(\mathbf{t}_i^c)$ is minimal, the linear system $\nabla E^c(\mathbf{t}_i^c) = 0$ is solved and the optimal position with the minimal error can be calculated as

$$\mathbf{t}_i^c = \left(\sum_{j \in \Omega_i^r} \tilde{\mathbf{n}}_j \otimes \tilde{\mathbf{n}}_j \right)^{-1} \sum_{j \in \Omega_i^r} (\tilde{\mathbf{n}}_j \otimes \tilde{\mathbf{n}}_j \mathbf{v}_j). \tag{3.15}$$

Optimal Position For Edge Points

For edge points $\mathbf{v}_i^e \in V^e$, the edge direction $\mathbf{y}_{i,1}$ defines a plane through $\mathbf{v}_i^e$ on which all neighboring points and normals are projected. This results in the projected neighboring points $\mathbf{v}_j^\pi = \mathbf{v}_j - ((\mathbf{v}_j - \mathbf{v}_i^e) \cdot \mathbf{y}_{i,1})\mathbf{y}_{i,1}$ and the projected neighboring smoothed normals $\tilde{\mathbf{n}}_j^\pi = \tilde{\mathbf{n}}_j - (\tilde{\mathbf{n}}_j \cdot \mathbf{y}_{i,1})\mathbf{y}_{i,1}$. After projection, the loss function is defined as

$$E^e(\mathbf{t}_i^e) = \sum_{j \in \Omega_i^r} \left(||\tilde{\mathbf{n}}_j^\pi \cdot (\mathbf{t}_i^e - \tilde{\mathbf{v}}_j^\pi)||^2 + \frac{1}{|\Omega_i^r|} ||\mathbf{y}_{i,1} \cdot (\mathbf{t}_i^e - \mathbf{v}_j^\pi)||^2 \right), \tag{3.16}$$

which minimizes the distance towards the plane created by the edge vector $\mathbf{y}_{i,1}$ and minimizes the distance towards the planes of the projected normals. Solving the linear equation $\nabla E^e(\mathbf{t}_i^e) = 0$ gives the optimal position that can be calculated with

$$\mathbf{t}_i^e = \left(\sum_{j \in \Omega_i^r} \tilde{\mathbf{n}}_j^\pi \otimes \tilde{\mathbf{n}}_j^\pi + \mathbf{y}_{i,1} \otimes \mathbf{y}_{i,1} \right)^{-1} \sum_{j \in \Omega_i^r} \left(\tilde{\mathbf{n}}_j^\pi \otimes \tilde{\mathbf{n}}_j^\pi \mathbf{v}_j + \mathbf{y}_{i,1} \otimes \mathbf{y}_{i,1} \mathbf{v}_i^e \right). \tag{3.17}$$

Optimal Position For Flat Points

For flat points $\mathbf{v}_i^f \in V^f$, the optimal position is retrieved in an iterative manner. An energy function similar to Equation 3.15 is minimized by gradient descent with the equation

$$\mathbf{t}_i^f = \mathbf{v}_i^f + \frac{\alpha}{\sum_{j \in \Omega_i^r} W_{ij}} \sum_{j \in \Omega_i^r} W_{ij}(\tilde{\mathbf{n}}_j, \mathbf{v}_j - \mathbf{v}_i^f)\tilde{\mathbf{n}}_i, \tag{3.18}$$

where the weight $W_{ij} = \exp\left(-\frac{16|\tilde{\mathbf{n}}_i - \tilde{\mathbf{n}}_j|^2}{\delta^2}\right) \cdot \exp\left(-\frac{4|\mathbf{v}_j - \mathbf{v}_i^f|^2}{\delta^2}\right)$ is a bilateral filter using relative position distance and normal distance as input. δ is set to be the range of the neighborhood Ω_i^r and α is the point diffusion speed that limits the amount of smoothing.

Position Update Restriction

Before updating a point, a restriction is set where a point can move at most a distance of ε. For every point $\mathbf{v}_i \in V$ it holds, that the updated position

$$\tilde{\mathbf{v}}_i = \begin{cases} \mathbf{t}_i \text{ if } \mathbf{d}_i \leq \varepsilon \\ \mathbf{v}_i \text{ if } \mathbf{d}_i > \varepsilon \end{cases}, \tag{3.19}$$

will be the temporary optimal position if the relative distance $\mathbf{d}_i = \mathbf{t}_i - \mathbf{v}_i$ is small enough. With this last step, one of the iterations of the pipeline is done.

3.3 Constrained Two-Direction Quadric Error Metrics

To critically assess the point position updating schemes discussed in the previous section, a comparative evaluation is necessary to validate their effectiveness. In this context, we also

examine the update scheme proposed by (Wei et al., 2021), which applies a uniform strategy to all points, regardless of their geometric context. The method is based on minimizing three energy functions. The first energy function is the same as Equation 3.15. They improved the energy function by discouraging lateral movement and shape distortion with the second and third energy functions. The second energy function to discourage lateral movement is

$$E^l(\tilde{\mathbf{v}}_i) = (\tilde{\mathbf{v}}_i - \mathbf{v}_i) \cdot (\tilde{\mathbf{v}}_i - \mathbf{v}_i) - (\text{proj}_{\mathbf{n}_i}(\tilde{\mathbf{v}}_i - \mathbf{v}_i)) \cdot (\text{proj}_{\mathbf{n}_i}(\tilde{\mathbf{v}}_i - \mathbf{v}_i)), \tag{3.20}$$

where $\tilde{\mathbf{v}}_i$ is the new location of the point, $\mathbf{v}_i$ the old location, and $\text{proj}_n(v) = (\mathbf{v} \cdot \mathbf{n})\mathbf{n}$ projects the vector $\mathbf{v}$ onto $\mathbf{n}$ assuming $\mathbf{n}$ has unit length. The third energy function penalizes shape distortion by comparing the distance of the neighboring points to the tangent plane of the new location. It is defined as

$$E^d(\tilde{\mathbf{v}}_i) = \sum_{j \in \Omega_i^k} ||\tilde{\mathbf{n}}_i \cdot (\tilde{\mathbf{v}}_i - \mathbf{v}_j)||^2. \tag{3.21}$$

When adding all energy functions together you get final energy function

$$E^t(\tilde{\mathbf{v}}_i) = E^c(\tilde{\mathbf{v}}_i) + E^l(\tilde{\mathbf{v}}_i) + E^d(\tilde{\mathbf{v}}_i) \tag{3.22}$$

that needs to be minimized. Since all linear equations can be solved, the exact solution to minimizing E^t with $\nabla E^t(\tilde{\mathbf{v}}_i) = 0$ is

$$\tilde{\mathbf{v}}_i = \left(\mathbf{I} + \sum_{j \in \Omega_i^k} (\tilde{\mathbf{n}}_j \otimes \tilde{\mathbf{n}}_j) + \left(1 + \|\Omega_i^k\|\right) \tilde{\mathbf{n}}_i \otimes \tilde{\mathbf{n}}_i \right)^{-1} \left((\mathbf{I} + \tilde{\mathbf{n}}_i \otimes \tilde{\mathbf{n}}_i)\, \mathbf{v}_i + \sum_{j \in \Omega_i^k} (\tilde{\mathbf{n}}_j \otimes \tilde{\mathbf{n}}_j \mathbf{v}_j) + \tilde{\mathbf{n}}_i \otimes \tilde{\mathbf{n}}_i \sum_{j \in \Omega_i^k} \mathbf{v}_j \right), \tag{3.23}$$

where $\mathbf{I}$ denotes the 3-by-3 identity matrix.

Chapter 4

Method

In this section, a proposal is done on how to examine and refine the pipeline proposed by (Yadav et al., 2018). Now that the theoretical foundations have been established, we explore how each step of the pipeline can be modified and improved. Through a series of experiments, the effect of different design choices is assessed and we demonstrate how an adjusted version of the pipeline can outperform the original.

The first two steps, normal estimation and normal smoothing, remain unchanged. The absolute accuracy of estimated normal vectors is less critical, because estimated normals serve primarily as a guidance signal rather than an exact representation of the ground truth. What matters more is that the normals are structured in a way that effectively supports the subsequent steps in the pipeline. Therefore, our method focuses on analyzing the impact of feature detection and position updating strategies, as these are more directly linked to improvements in denoising performance. The part where we are going to focus on is marked with a red square in Figure 4.1.

4.1 Feature Detection

To thoroughly evaluate the feature detection process, we divide the testing into two parts. First, we conduct visual inspections to gain intuitive insight into how different parameters influence the results. Second, we perform quantitative evaluations by measuring the percentage of correctly classified points. Together, these tests help fine-tune the feature detection parameters for optimal point classification. To support the visual inspections, we propose a feature space visualization that helps interpret the role of each parameter. In Figure 4.1, the feature detection pipeline is illustrated. We walk through each step, discussing the available choices and outlining the experiments that will be conducted. This will be done in reverse for convenience, starting with the Feature Extraction and Classification step. These investigations aim to refine the pipeline to achieve more effective and reliable feature detection. In Table 4.1 an overview is given of the parameters that will be examined.

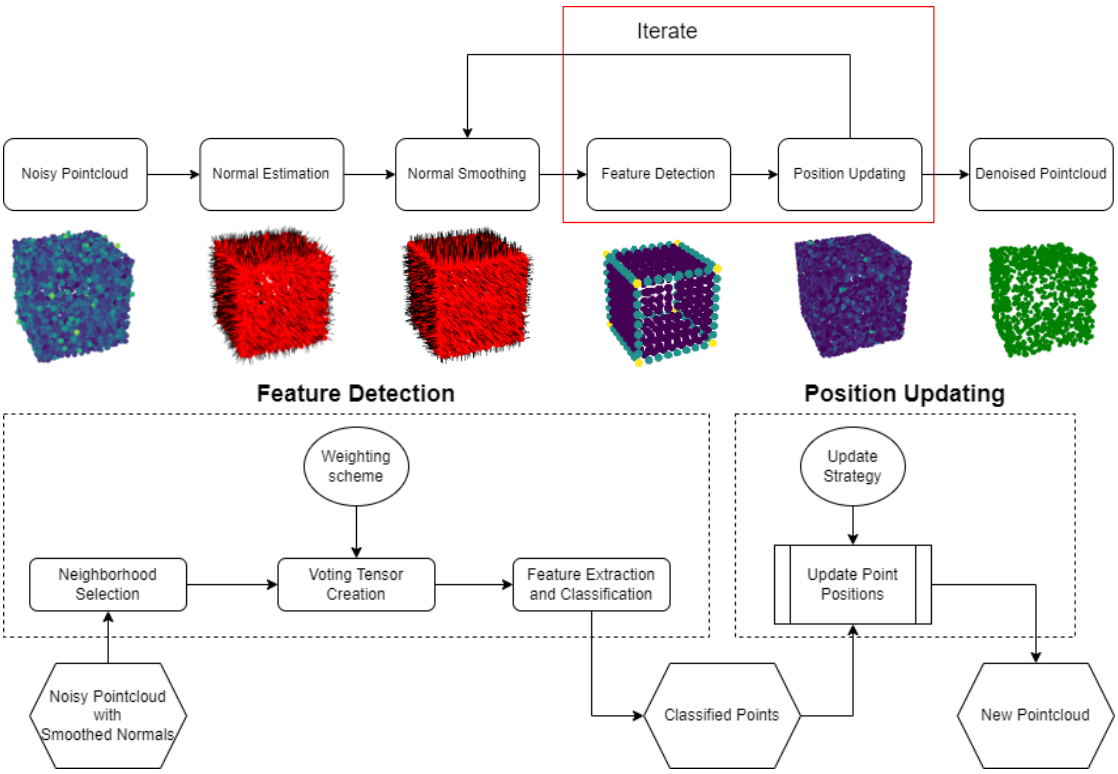

Figure 4.1: Proposed normal-guided pointcloud denoising pipeline. The lower part of the figure zooms in on the feature detection and position updating steps, detailing the process to get from a noisy pointcloud with smoothed normals to classified points and, ultimately, a denoised pointcloud. These are the components of the pipeline that are the primary focus of the proposed method.

Pipeline Step	Original Parameter	New Parameters
Neighborhood Selection	Ω_r	Ω_k
Voting Tensor Creation	$\mathbf{T}_i^P$	$\tilde{\mathbf{T}}_i^N$
Weighting Scheme	w_{ij}	$\hat{w}_{ij}$
Feature Extraction and Classification	τ	ω and $\mathbf{f}_i$

Table 4.1: Table of parameters that will be examined and the new proposed parameters that will be compared against the originals.

4.1.1 Feature Extraction and Classification

We begin with the feature extraction and classification step of the pipeline. Starting here is beneficial, because it allows us to visualize the feature space, which in turn provides intuition for how parameter adjustments influence feature detection. As described in Section 3.2.3, the original pipeline classifies points by directly thresholding the eigenvalues obtained from eigendecomposition with τ. However, a limitation of this approach is that eigenvalues are sensitive to the scale and size of the neighborhood, making it less reliable when working with neighborhoods of varying sizes. To address this issue, (Demantké et al., 2011) propose computing geometric features from the eigenvalues. This method offers a more robust and interpretable basis for point classification. Therefore, we integrate this method into the pipeline from Yadav et al. (2018) and it works as follows. After performing eigendecomposition on a symmetric semi-definite voting tensor (constructed from either relative positions or normal vectors), a set of eigenvectors with corresponding eigenvalues μ_{i1}, μ_{i2} and μ_{i3} is obtained. These eigenvalues describe the geometric distribution of points within the local neighborhood. As proposed by (Demantké et al., 2011), the geometric features linearity, planarity, and scattering (from now on referred to as sphericity) can be derived from these eigenvalues with

$$\mathbf{f}_i = \begin{bmatrix} \text{Planarity} \\ \text{Linearity} \\ \text{Sphericity} \end{bmatrix} = \begin{bmatrix} \frac{\mu_{i1}-\mu_{i2}}{\mu_{i1}} \\ \frac{\mu_{i2}-\mu_{i3}}{\mu_{i1}} \\ \frac{\mu_{i3}}{\mu_{i1}} \end{bmatrix} . \tag{4.1}$$

As a side note, the term sphericity should not be mistaken for how closely the points fit a spherical surface. Rather, it refers to how well the points are distributed throughout the volume of a sphere, as opposed to being confined to a plane or a line. In other words, the points are spread across all three dimensions, rather than just two or one. As seen in the equation, the three features are combined in a feature vector $\mathbf{f}_i$, such that it can be visualized. Because these features sum to one, they can be represented as coordinates within a 3D triangle defined by the corners (1, 0, 0), (0, 1, 0), and (0, 0, 1). In this representation, each axis corresponds to one of the features, allowing for intuitive visualization. An example of an empty plot is shown in 4.2.

These visualizations will be used in later experiments to analyze the influence of various parameters. The final step is to classify the points based on their feature vector. The classification is performed by scaling $\mathbf{f}_i$ with the weight vector ω and finding the largest feature value, such that

$$\text{class}_i = \text{argmax}\left(\omega \mathbf{f}_i\right). \tag{4.2}$$

This will set boundaries, where certain areas in the feature space are labeled as certain classes.

4.1.2 Weighting Scheme

The original weighting scheme (Equation 3.9) uses a binary filter, assigning a weight of either 0 or 1 to each neighbor. A neighbor is included only if the angle between its normal and the normal

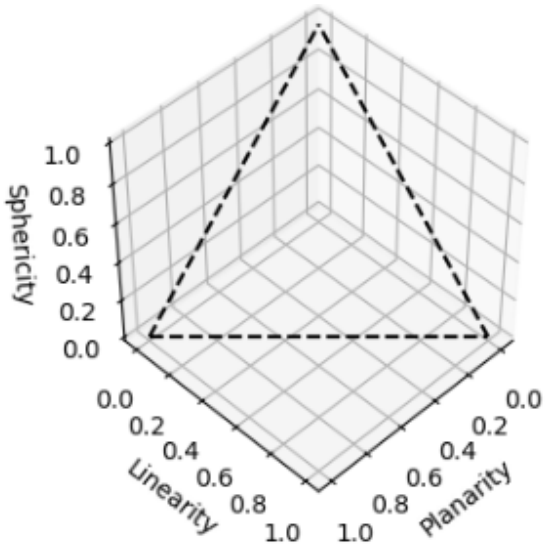

Figure 4.2: Example of an empty feature space scatter plot, where each axis represents a calculated feature. The left corner represents Planarity, the right corner represents linearity and the top corner represents Sphericity.

of the query point is smaller than a threshold ρ. This approach effectively removes neighbors from opposing sides of sharp features, which is beneficial for points on smooth or curved surfaces, as it helps maintain surface consistency by excluding unrelated subsurface regions.

However, this strategy becomes problematic for feature points, where normal vectors often differ significantly from those of neighboring points. As a result, these neighbors are likely to be filtered out, even though they are essential for estimating the location of the feature.

To address this, we propose an alternative weighting scheme that filters neighbors based on local curvature or distance to the neighboring tangent plane rather than normal vector similarity. To the best of our knowledge, this specific angular thresholding scheme, which filters neighbors based on distance to tangent plane defined by the neighboring point normal, has not been explored in existing normal-guided denoising methods. The scheme is formulated as

$$\hat{w}_{ij} = \begin{cases} 1 & \text{if } |\theta_{ij} - \frac{\pi}{2}| \leq \rho \\ 0 & \text{if } |\theta_{ij} - \frac{\pi}{2}| > \rho \end{cases} \text{for } \rho \in [0, \frac{\pi}{2}], \tag{4.3}$$

where $\theta_{ij} = \arccos\left(\frac{\mathbf{d}_{ij} \cdot \tilde{\mathbf{n}}_j}{\|\mathbf{d}_{ij}\|}\right)$ and $\mathbf{d}_{ij} = \mathbf{v}_j - \mathbf{v}_i$. The weight is one when the angle between the relative position and the neighboring normal is close to perpendicular and zero otherwise. To illustrate the behavior of the new weight $\hat{w}_{ij}$, Figure 4.3 presents several diagrams demonstrating how the angle θ_{ij} is computed from the relative position vector $\mathbf{d}_{ij}$ and the neighboring normal vector $\tilde{\mathbf{n}}_j$ across surfaces with varying curvature. The diagrams show that as the curvature of the surface increases, the angle θ_{ij} deviates more significantly from $\frac{\pi}{2}$. What it means to be close to perpendicular is defined by the threshold ρ. By excluding neighbors with high curvature, this scheme allows feature points to keep more relevant neighbors, increasing the likelihood of

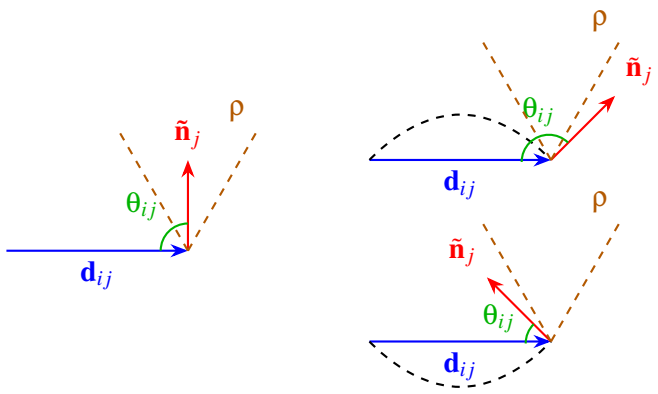

Figure 4.3: Diagrams of different situations describing how the angle θ_{ij} affects the weight $\hat{w_{ij}}$. The blue vectors $\mathbf{d}_{ij}$ denote the relative position vectors from point $\mathbf{v}_i$ to point $\mathbf{v}_j$. The dashed black line denotes the underlying surface. The red vectors $\tilde{\mathbf{n}}_j$ denote the smoothed point normal at point $\mathbf{v}_j$. The dashed orange lines represent the threshold ρ, which depicts the boundaries in which the normal vector $\tilde{\mathbf{n}}_j$ must fall in order for the weight $\hat{w}_{ij}$ to be one and thus include the point into the neighborhood for further calculations. In the diagrams ρ is set to be $\frac{\pi}{3}$. The green arcs show the angle θ_{ij} between the relative position vector and normal vector. The left diagram shows an example of a flat surface where $\theta_{ij} = \frac{\pi}{2}$. The right diagrams show examples of surfaces with high curvature where the angles are $\theta_{ij} = \frac{3}{4}\pi$ for the top diagram and $\theta_{ij} = \frac{\pi}{4}$ for the bottom diagram. The diagrams show that surfaces with higher curvature have angles θ_{ij} further away from perpendicular.

accurately detecting the feature locations. At the same time, flat and curved surface points still benefit from the filtering of irrelevant neighbors or neighbors far along the surface.

4.1.3 Voting Tensor Creation

In this step of the pipeline, a voting tensor is created. The key question to address is whether to use relative position vectors, as in the original method calculated by Equation 3.8, or normal vectors to form the tensor. The normal voting tensor can be calculated with

$$\tilde{\mathbf{T}}_i^N = \sum_{j \in \Omega_i^r} w_{ij} \tilde{\mathbf{n}}_j \tilde{\mathbf{n}}_j^T, \tag{4.4}$$

with a weight w_{ij} per neighbor. Note that equation 3.2 is exactly the same, but here we use smoothed normals instead of estimated normals. By applying the same neighborhood selection and weighting scheme to both approaches, we generate voting tensors and extract their corresponding feature spaces. A visual analysis of these spaces should provide sufficient insight into which method is more effective for classifying the points.

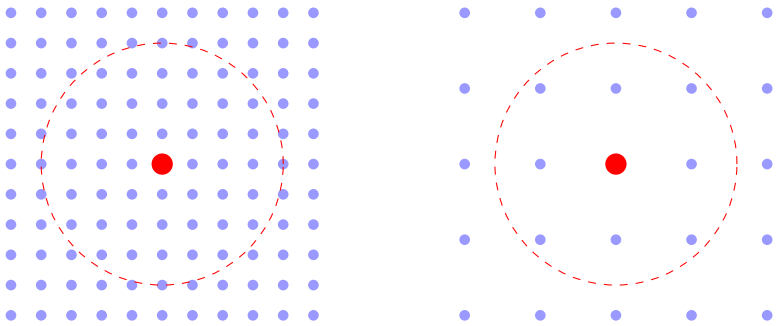

Figure 4.4: An illustration of the limitation of range-based neighborhood selection. Blue dots represent points in the pointcloud, and the red circle indicates the fixed search radius used to define a neighborhood. The left diagram depicts a high-density region where many neighbors fall within the radius, while the right diagram shows a low-density region where significantly fewer neighbors are captured.

4.1.4 Neighborhood Selection

In the original method, a range-based neighborhood Ω_r is used, where all points within a global radius parameter r are included in the neighborhood. However, in regions of the pointcloud with low density, this can result in neighborhoods with very few or in extreme cases no neighbors. An illustration depicting this effect can be seen in Figure 4.4. This choice originates from an implementation tailored for mesh denoising, where KNN neighborhoods were not considered viable. Given that KNN neighborhoods have been used in various pointcloud processing tasks, we opt to return to using them in our approach. Therefore, for the neighborhood selection step, we will use a KNN neighborhood and examine how the choice of neighborhood size k, the number of neighbors considered per point, impacts the resulting feature space. This experiment explores the effects of using larger versus smaller neighborhoods, which can be used during the selection of an appropriate value for k.

4.1.5 Parameter Tweaking

With the previous experiments completed, we now move on to the final step: fine-tuning the parameters k and ρ. To evaluate the effect of different settings, we will manually classify the pointclouds to create ground truth labels and compare them against the classifications produced by the pipeline under various parameter configurations for k and ρ. The evaluation will be conducted on pointclouds with varying densities to assess how point density influences the results. Specifically, we will use pointclouds sampled from a cube and a tetrahedron, allowing us to study the impact of feature sharpness on classification accuracy. In the end, we will compare the optimal parameter settings for each model and density level to identify the most robust configuration overall.

4.2 Position Updating

Following the classification step, we shift our focus to the position update process. First, we identify that diffusion speeds are missing from some of the update formulas and reintroduce them. Finally, we define the concept of an update strategy and describe the strategy used in our final pipeline implementation.

4.2.1 Missing Diffusion Speeds

After examining the position update formulas (Equation 3.15, Equation 3.17, Equation 3.18 and Equation 3.23) for the points, we noticed that only the formula for flat points includes a diffusion speed α. Since this was inconsistent, we decided to introduce a diffusion speed to all formulas. This change results in that the temporary position becomes the proposed new position, with the diffusion speed determining how much each point shifts towards it. Each diffusion speed is matched to its corresponding update formula; α_c is used when updating points with Equation 3.15, α_e is used when updating points with Equation 3.17, α_f is used when updating points with Equation 3.18 and α_q is used when updating points with Equation 3.23. The new diffusion speeds can be incorporated in Equation 3.19 and the denoised positions are now defined by

$$\tilde{\mathbf{v}}_i = \mathbf{v}_i + \alpha * \begin{cases} \mathbf{d}_i \text{ if } \mathbf{d}_i \leq \varepsilon \\ 0 \text{ if } \mathbf{d}_i > \varepsilon \end{cases}, \tag{4.5}$$

where the chosen diffusion speed $\alpha \in \{\alpha_c, \alpha_e, \alpha_f, \alpha_q\}$ corresponds to the selected update formula.

4.2.2 Position Update Strategy

For each classified point, there are two update formulas considered as viable options: the formulas from Yadav et al. (2018) (Equation 3.15, Equation 3.17 or Equation 3.18) or those from Wei et al. (2021) (Equation 3.23). The choice of formula for each point class defines the update strategy. To determine the most effective strategy, we conduct a series of experiments where only the points from a single class are updated, and we analyze how different strategies and parameter settings affect reconstruction accuracy. Each experiment involved multiple denoising iterations until a minimal error is reached. The minimal error and the number of iterations required to reach it are reported in the tables. These results are then analyzed to identify the optimal strategy. In our final pipeline implementation, we use Equation 3.18 for flat points and Equation 3.23 for edge and corner points, performing two iterations in total.

Chapter 5

Experiments and Results

We begin this section by describing the process used to obtain the pointclouds used for evaluation. Next, we aim to design an improved denoising pipeline through a series of experiments. We begin this series of experiments by refining the feature detection process through various modifications. Next, we explore improvements in the point position updating step. Finally, we integrate all improvements into a revised pipeline and demonstrate that it outperforms both the original method and a method that uses the CTD-QEM updating scheme.

5.1 Data Acquisition

The pointclouds used in the following sections are generated by sampling from known mesh models. These samples are taken directly from the mesh surfaces using gaussian sampling to simulate the typical scenario, where only unstructured pointclouds are available, as is common in real-world 3D scans. To simulate measurement noise, Gaussian noise is added to the sampled points along the direction of their surface normals. The standard deviation of this Gaussian distribution is determined by multiplying a set noise level with the average distance between points in a KNN neighborhood, where k is set to 6. The average distance represents the local density of the pointcloud. Unless stated otherwise, this sampling and noise addition setup is used consistently across all experiments with a noise level of 0.3.

5.2 Feature Detection

In this section, we present all experiments conducted for the feature detection stage. We begin by comparing the use of relative position vectors versus normal vectors for constructing voting tensors and evaluating their effectiveness in separating their feature space. Next, we investigate how varying the neighborhood size affects the resulting feature space. We then compare different weighting schemes, providing reasoning for why one outperforms the other. After that, we briefly examine the feature space threshold and demonstrate that the selected threshold yields reliable

results. Finally, we perform a comprehensive classification experiment to identify the optimal combination of angle threshold and neighborhood size for accurately classifying points in the pointcloud.

5.2.1 Relative Position vs Normal Vector

First, we will visualize the feature spaces of the PVT (Equation 3.1) and NVT (Equation 3.2). To determine which would be a better choice for the feature detection step, we visualize different models for each voting tensor with their feature space in Figure 5.1. While reviewing the figures, we aim to identify a method that ensures each type of point occupies a distinct region in the feature space. This distinction can be observed if each point type is assigned a unique coloring that is separable from that of another type. The figure illustrates that the PVT is more susceptible to noise in the bottom region of the feature space than the NVT, as flat points are assigned either a red or green color in a random fashion. Additionally, the distinction between corner and edge points appears less clear in the PVT. Both can be attributed to the number of dimensions spanned by the vectors used in each method for different types of points. On flat surfaces, relative position vectors span two dimensions, while normal vectors span only one. As a result, the PVT typically yields two large eigenvalues for flat regions, whereas the NVT yields only one. This causes flat points to spread along the bottom edge of the feature space triangle in the PVT, whereas in the NVT they cluster tightly near the Planarity corner. This effect is particularly evident in the figures of the sphere, as it consists solely of flat (or slightly curved) points. Consequently, flat points in the NVT feature space appear consistently red, while in the PVT they show more variation, appearing either red or green in a noisy fashion. For edge and corner points, relative position vectors span all three dimensions, producing three large eigenvalues in the PVT, thus making it harder to distinguish between edges and corners. In contrast, the NVT captures a more structured variation: normals at edges span two dimensions, and at corners they span all three dimensions. This dimensional difference enables clearer separation between edges and corners in the NVT feature space. Therefore, from these findings we conclude that it is better to use the NVT in our implementation to detect features.

5.2.2 Neighborhood Size

In the following experiment, we try to analyze what effect the neighborhood size has on the feature space. We will use a variation of different values for k, compute the point-based voting tensor from equation 3.2, and analyze the feature space. The results can be found in Figure 5.2.

At the figures with low number of neighbors, the coloring is more influenced by noise and blue dots can be seen on the flat surface. In the figures with high numbers of neighbors on the right, we can see that the points close to features are more influenced by the features than needed. In the next section, we will show how the neighborhood can be filtered, such that the features can be detected more accurately.

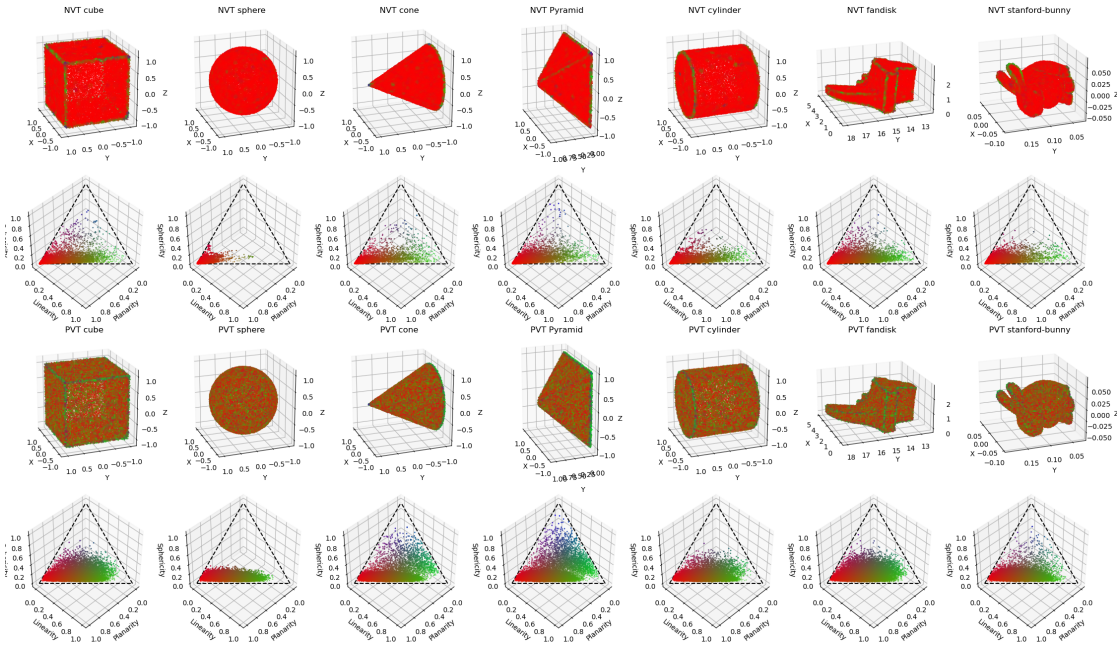

Figure 5.1: Different pointclouds with noise colored according to their planarity, linearity and sphericity in the feature space. The top figures are calculated with the NVT and the bottom figures are calculated with the PVT.

5.2.3 Filtering Points with Weighting Schemes

Currently the neighborhoods take into account all neighbors equally. However, in some cases there are points that fall in the neighborhood, because they are close in euclidean space, but should not be part of the neighborhood, because they are far away along the surface. To exclude these points in the computation, we add a filter. (Yadav et al., 2018) proposes a filter based on the difference in normals, but we propose a filter based on the angle between the neighboring normal vector and the relative position vector with the neighbor. The difference in filter weights is shown in Figure 5.3. The figure shows the average weights activation per neighborhood and a difference in weight activation on feature points. Our filter will not activate on features, which means it will include almost all neighbors for further calculations. The weight based on normal difference activates the most on feature points, which means the most neighbors are filtered out on feature points, which increases the chance of filtering out neighbors with important and relevant information. To show the effect of our filter, Figure 5.4 shows the feature space for different values for threshold ρ and neighborhood size k. The effect of the filter becomes evident as fewer points near features are classified as feature points when the threshold approaches $\frac{\pi}{2}$. This causes a more precise labeling of the points. Setting ρ too close to $\frac{\pi}{2}$ is also risky, since normals are noisy and points can be misjudged.

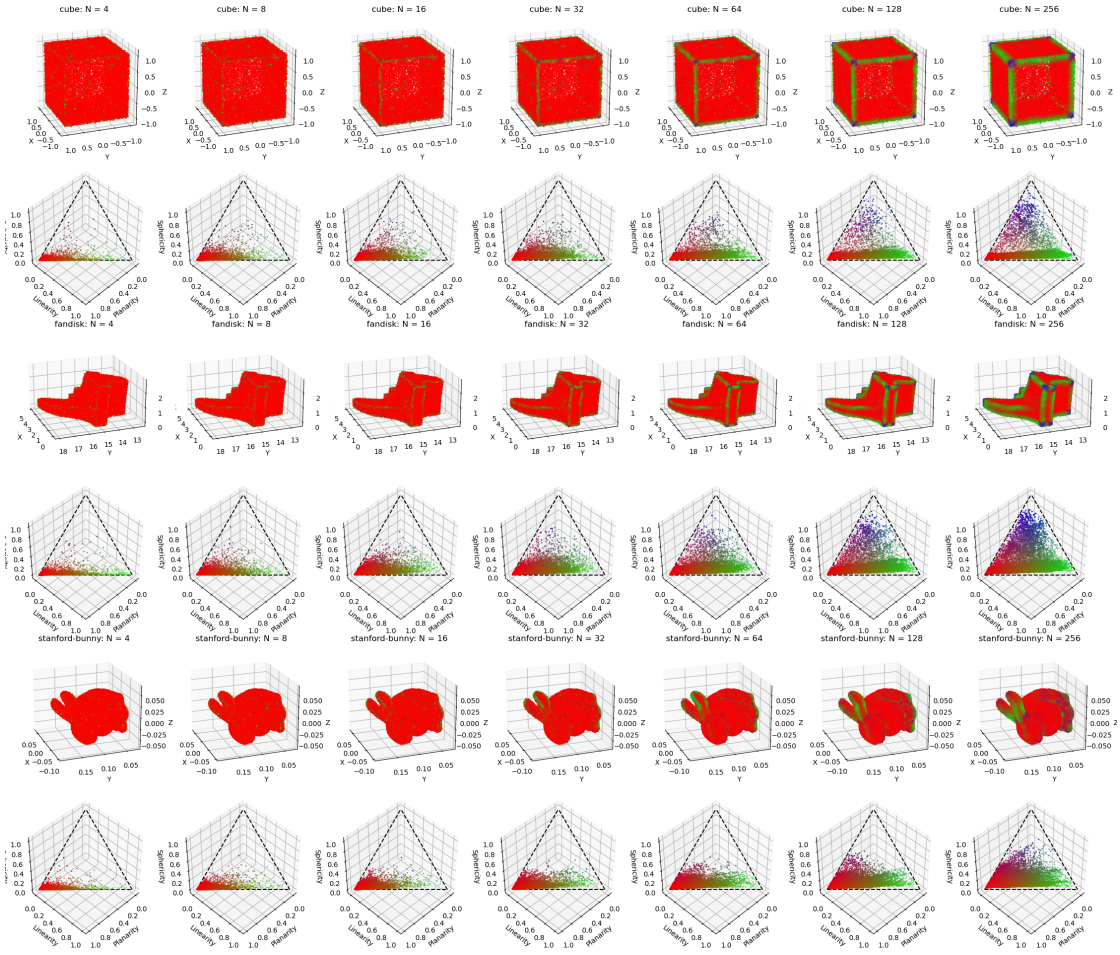

Figure 5.2: Pointclouds of three models with noise. Per visualization, the pointclouds are colored by their features. Horizontally, the knn neighborhood is increased where the number of neighbors is defined above the plot.

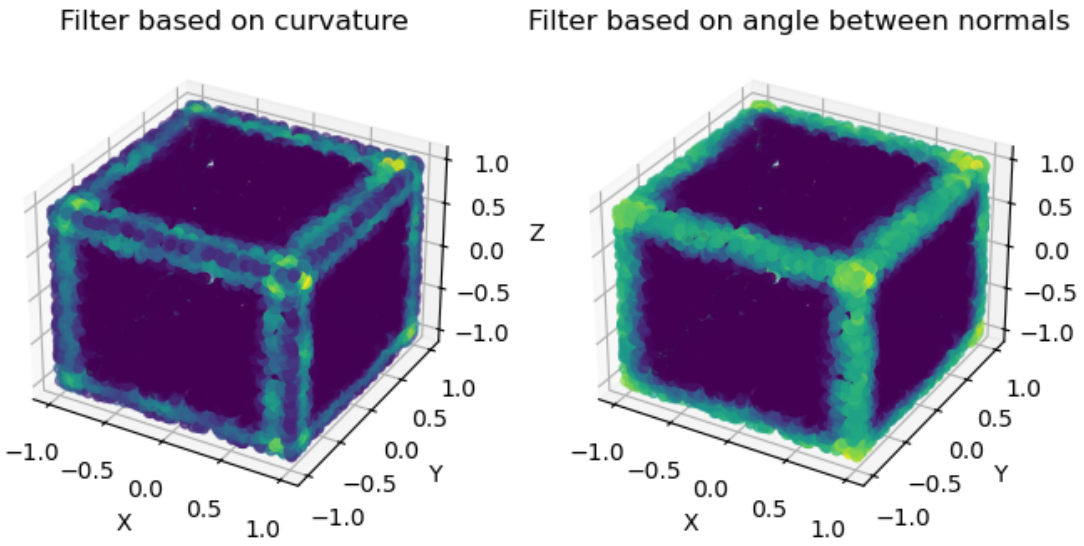

Figure 5.3: Visualization of the average filterweights per neighborhood. Yellow means high average and purple means a low average.

5.2.4 Feature Space Scaling

The final step in the pipeline involves classifying each point based on its position in the feature space. Using the features planarity, linearity, and sphericity, each point is assigned a class corresponding to the highest feature value. However, before doing this, a scaling is applied to better reflect the significance of each feature. In our implementation, we use the weighting vector $\omega = [0.2, 1, 1]$, reducing the influence of planarity, as it tends to dominate the feature space. Figure 5.5 illustrates the classification results on several models: a cube, torus, fandisk, and the Stanford bunny, alongside their respective feature spaces. Visually, the classifications appear to be accurate. Notably, the torus contains only non-feature points, and these are correctly identified. In the following section, we will determine the optimal values for the neighborhood size k and angle threshold ρ based on this fixed feature scaling.

5.2.5 Point classifications

After understanding the effect of the parameters, we want to find which parameters will result in the most accurate classification of our points. In these tests, we apply gaussian noise with noise level 0.2 to different models. Then we do one iteration of normal estimantion, normal smoothing and classification and compare the classification to the manual classification of the model. If the classification is correct, we mark the point with a purple color and otherwise with a yellow color. The percentage of correctly classified points is stated above the figure together with the parameters used for the experiment. The results of all experiments on different models with different parameter settings can be found in the Appendix A.

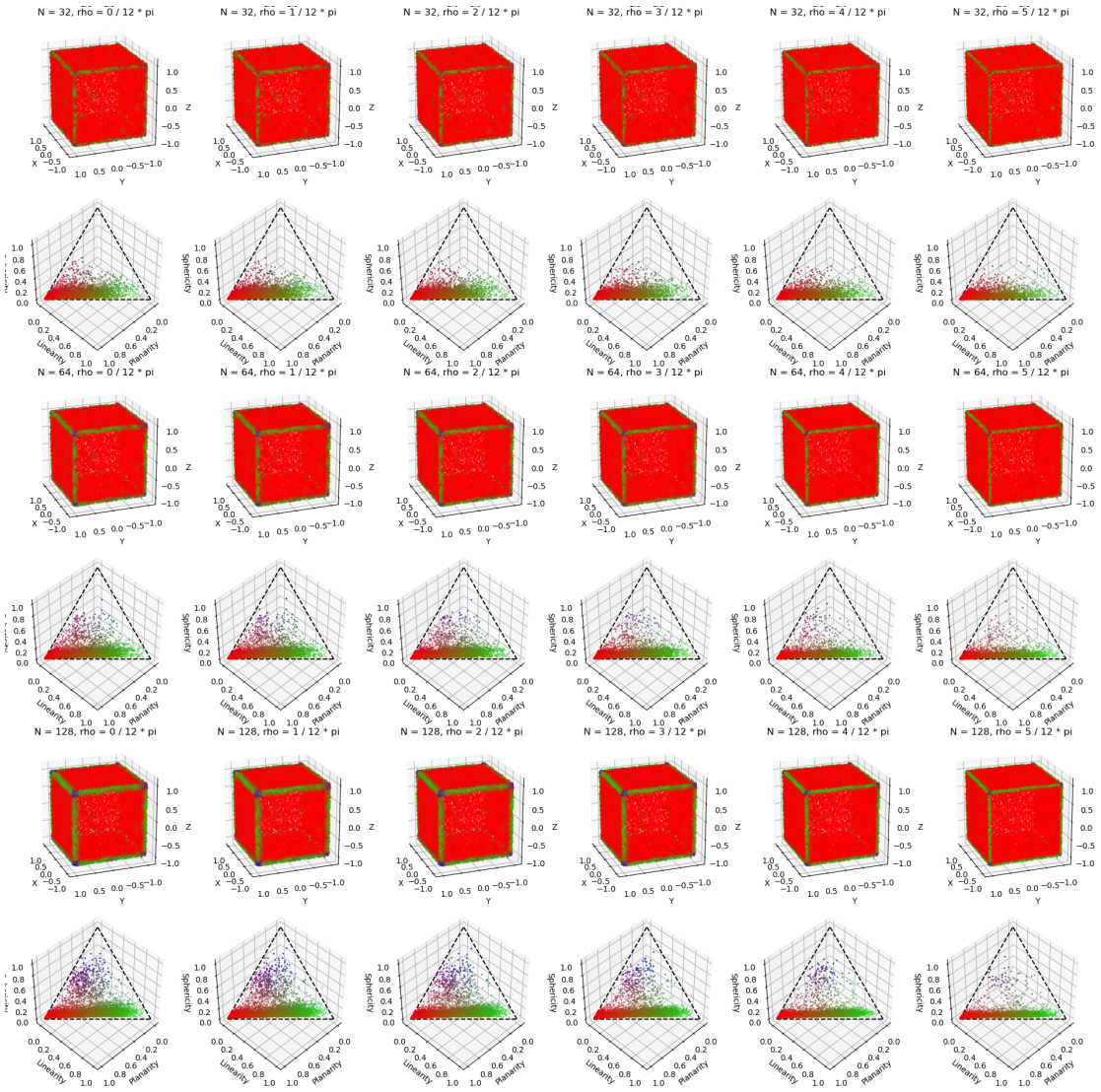

Figure 5.4: Visualization of the feature space with different number of neighbors and angles for filtering. From top to bottom the number of neighbors is increased and from left to right the angle rho is increased.

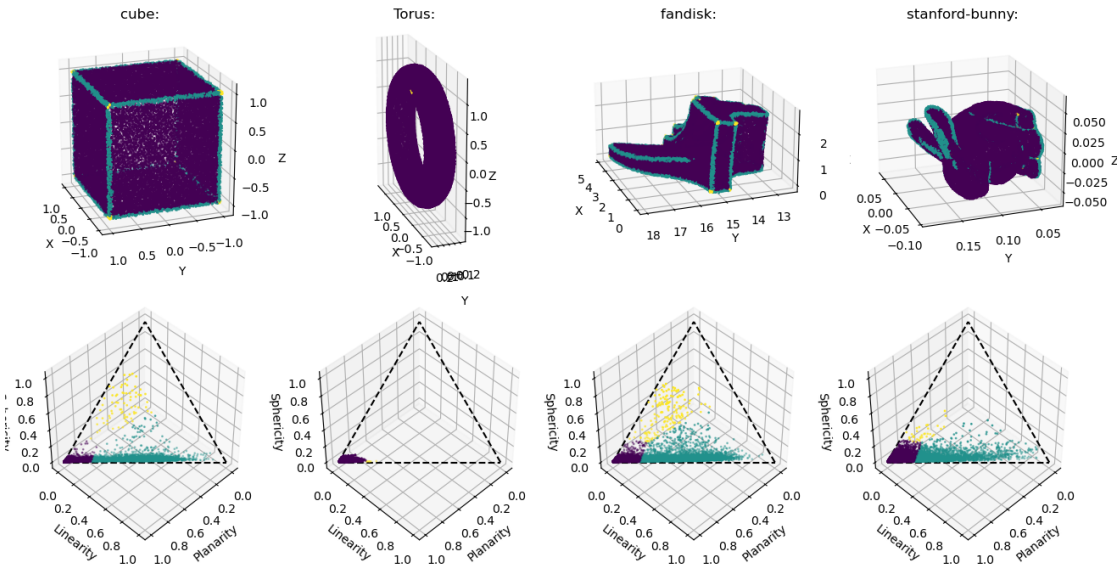

Figure 5.5: Classification of points in pointclouds of different models. Models contain 2^{15} points with a noise level of 0.3. Classification colors are the following: Corner points are yellow, edge points are green and flat points are purple.

The models used in the experiments are different density versions of the Cube and Tetrahedron. The Cube contains right-angled features, while the Tetrahedron has sharper features. This difference allows us to investigate the effect of the feature angle on the classification performance. Additionally, varying the pointcloud density enables us to study how density influences the classification results.

From the experiments, we learn that the smaller the neighborhood and the closer the angle threshold ρ is to $\frac{\pi}{2}$, the more misclassifications are made on the feature points. In reverse, when ρ is further away from $\frac{\pi}{2}$, more flat points close to the features are misclassified as features. It can be seen that there is an optimal ratio between these parameters to get the optimal classification.

Furthermore, the cube and tetrahedron differ in the sharpness of their features, as the angles at edges and corners are sharper for the tetrahedron than for the cube. In the experiments, it can be seen that the optimal ratio between the angle threshold and the neighborhood size is different for the cube and the tetrahedron. As the angle of the feature gets sharper, the angle of the threshold should also get sharper. This means that there is no setting, such that points for all shapes are correctly classified using this method.

The angle of a feature can be anything between 0 and π. Since a cube only has features of $\frac{\pi}{2}$, which is the mean of all possible angles, we choose a setting for k and ρ that works the best for the cube. Therefore, in our pipeline, $k = 16$ and $\rho = \frac{5}{12}\pi$. The resulting classification experiments for these settings on the different models are summarized in Figure 5.6

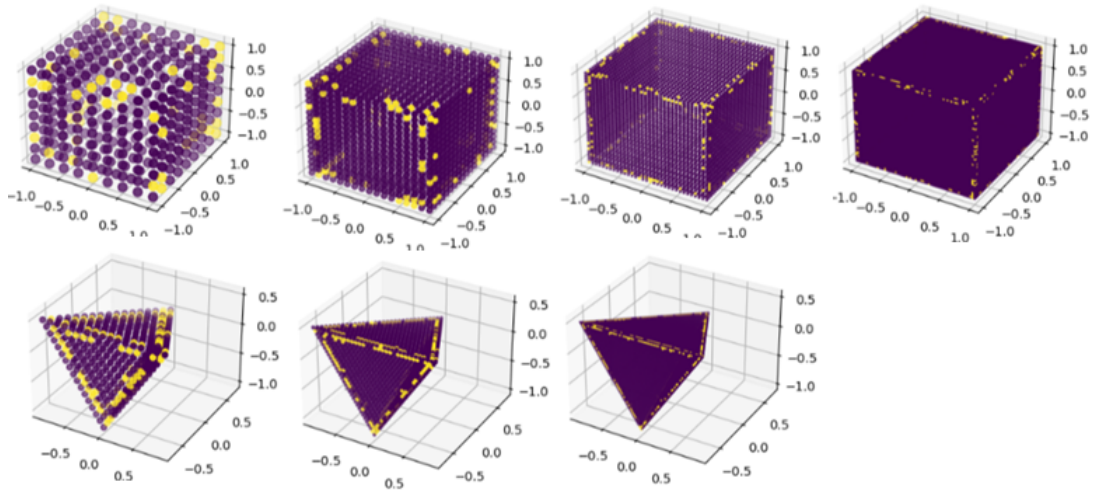

Figure 5.6: Summary of the classification experiments for the chosen parameter settings, where $\rho = \frac{5}{12}\pi$ and $k = 16$. Yellow points are misclassified while purple points are correctly classified. The cube has been sampled with 4 different densities and the tetrahedron with 3 different densities.

5.3 Position updating

For the position updating step, we aim to determine which update formula performs best for each point class. To evaluate the effectiveness of the denoising, we first introduce two error metrics that help us assess the denoising process. We then describe the experimental setup designed to identify the optimal strategy and parameter settings for the pipeline.

5.3.1 Error Metrics

To compare the denoised pointcloud with the ground truth pointcloud, the Chamfer Distance (CD) is used as an error metric. The formula is defined in 5.1 for pointsets S_1 and S_2, where L_r is the range of the minimal sphere in which the pointset fits.

$$CD(S_1,S_2) = \frac{1}{L_r|S_1|}\sum_{\mathbf{x}\in S_1} \min_{\mathbf{y}\in S_2} ||\mathbf{x}-\mathbf{y}||_2^2 + \frac{1}{L_r|S_2|}\sum_{\mathbf{y}\in S_2} \min_{\mathbf{x}\in S_1} ||\mathbf{x}-\mathbf{y}||_2^2 \tag{5.1}$$

The single-sided Chamfer Distance (sCD) is defined in 5.2. The sCD can validate if the denoised pointcloud is close to the ground truth. However, it cannot see if point clustering occurs on the surface. This is why we still use CD to see if all ground truth points are covered by all denoised points.

$$sCD(S_1,S_2) = \frac{1}{L_r|S_1|}\sum_{\mathbf{x}\in S_1} \min_{\mathbf{y}\in S_2} ||\mathbf{x}-\mathbf{y}||_2^2 \tag{5.2}$$

5.3.2 Parameter Exploration for Position Updating

In the position updating step, we want to compare what update formulas result in a better denoising result. The parameters that have to be experimented with are the strategy, the diffusion speed α and the number of iterations the algorithm needs to minimize the error. A strategy determines which updating formula is used based on the classification of a point. In the experimental setup, we use the feature detection setup from the previous section to classify the points. In the experiments, we compare $\mathbf{v}_i^{\text{flat}}$ and $\mathbf{v}_i^{\text{feature}}$ for flat points, $\mathbf{v}_i^{\text{feature}}$ and $\mathbf{v}_i^{e}$ for edge points, and $\mathbf{v}_i^{c}$ and $\mathbf{v}_i^{\text{feature}}$ for corner points. Given a value for α and the error metrics sCD or CD, we denoise the points with the chosen classification until a local minimum in the error is reached. The number of iterations needed and the final local minimal error is noted. The results for different models are provided in Table 5.1, 5.2, 5.3 and 5.4.

A simple correlation observed in the tables is that the number of iterations required to reach a local minimum in the error metric increases as the diffusion speed decreases. This is expected, as a smaller diffusion speed results in smaller steps towards the optimal solution, thereby requiring more iterations to converge. While fewer iterations are desirable for algorithm efficiency, higher diffusion speeds can cause the algorithm to overshoot the optimal solution. Therefore, a higher diffusion speed does not always yield the lowest possible error.

diffusion speeds α	Error Metric $\times 10^{-5}$	Fandisk											
		Flat Points				Edge Points				Corner Points			
		$\mathbf{v}_i^{\text{flat}}$		$\mathbf{v}_i^{\text{feature}}$		$\mathbf{v}_i^{e}$		$\mathbf{v}_i^{\text{feature}}$		$\mathbf{v}_i^{c}$		$\mathbf{v}_i^{\text{feature}}$	
		Error	Iterations	Error	Iterations	Error	Iterations	Error	Iterations	Error	Iterations	Error	Iterations
1	sCD	0.9415	1	0.8398	13	1.309	3	1.308	2	1.476	1	**1.472**	**2**
	CD	2.068	1	2.122	1	2.429	0	2.427	1	2.429	0	2.429	0
0.9	sCD	0.9397	2	0.8407	14	1.306	5	1.308	3	1.478	0	1.473	2
	CD	2.067	2	2.123	1	2.429	0	2.416	1	2.429	0	2.429	0
0.8	sCD	0.9388	2	0.8398	17	1.305	5	1.309	3	1.475	3	1.473	2
	CD	**2.066**	**2**	2.131	1	2.429	0	2.407	1	2.429	0	2.429	0
0.7	sCD	0.9433	2	0.8397	24	1.303	6	1.309	3	1.475	2	1.473	3
	CD	2.069	2	2.147	1	2.429	0	2.400	1	2.429	0	2.429	0
0.6	sCD	0.9448	3	0.8427	24	1.302	7	1.310	4	1.474	2	1.473	3
	CD	2.070	3	2.152	2	2.429	0	2.395	1	2.429	0	2.429	0
0.5	sCD	0.9468	4	0.8434	25	1.302	7	1.311	5	1.474	2	1.473	4
	CD	2.071	4	2.154	2	2.426	1	2.393	1	2.429	0	2.429	0
0.4	sCD	0.9516	7	0.8610	14	1.303	6	1.312	7	1.474	2	1.473	5
	CD	2.075	7	2.166	4	2.414	1	2.394	1	2.429	0	2.429	0
0.3	sCD	0.9516	7	8.610	14	1.304	12	1.313	9	1.474	3	1.473	7
	CD	2.075	7	2.166	4	2.408	1	2.398	1	2.429	0	2.429	1
0.2	sCD	0.9549	10	0.8435	70	1.305	13	1.313	13	1.474	4	1.473	11
	CD	2.077	10	2.172	6	2.407	1	**2.397**	**2**	2.429	0	2.429	1
0.1	sCD	0.9572	22	0.8642	46	1.306	27	1.315	27	1.474	8	1.474	9
	CD	2.079	22	2.177	11	2.406	3	2.399	5	2.429	0	2.429	2

Table 5.1: Table of results for experimentation on the Fandisk. For a selected point class, update formula, diffusion speed α, and error metric, points belonging to the class are denoised over multiple iterations until the error metric reaches a minimum. The minimum error value and the corresponding number of iterations are recorded in the table. For the Fandisk, using $\mathbf{v}_i^{\text{flat}}$ for flat points and $\mathbf{v}_i^{\text{feature}}$ for edge and corner points with diffusion speeds $\alpha = [0.8, 0.2, 1]$ for 2 iterations is optimal. Results of experiments using these settings are marked in bolt.

diffusion speeds α	Error Metric $\times 10^{-5}$	Cube											
		Flat Points				Edge Points				Corner Points			
		$\mathbf{v}_i^{\text{flat}}$		$\mathbf{v}_i^{\text{feature}}$		$\mathbf{v}_i^{e}$		$\mathbf{v}_i^{\text{feature}}$		$\mathbf{v}_i^{c}$		$\mathbf{v}_i^{\text{feature}}$	
		Error	Iterations	Error	Iterations	Error	Iterations	Error	Iterations	Error	Iterations	Error	Iterations
1	sCD	1.628	2	1.446	10	2.647	5	2.648	4	2.812	1	2.844	1
	CD	**3.806**	**2**	3.932	1	4.652	0	4.652	0	4.591	1	4.652	0
0.9	sCD	1.630	2	1.439	11	2.641	4	2.647	4	2.817	1	2.848	0
	CD	3.807	2	3.936	1	4.652	0	4.643	1	4.602	1	4.652	0
0.8	sCD	1.634	2	1.443	12	2.639	6	2.648	5	2.821	2	2.848	0
	CD	3.812	2	3.957	1	4.652	0	4.633	1	**4.609**	**2**	4.652	0
0.7	sCD	1.636	3	1.444	14	2.640	8	2.650	6	2.821	2	2.848	0
	CD	3.813	3	3.995	1	4.652	0	4.625	1	4.606	4	4.652	0
0.6	sCD	1.640	4	1.446	16	2.641	8	2.650	7	2.823	5	2.848	0
	CD	3.816	4	3.997	2	4.652	0	4.619	1	4.609	5	4.652	0
0.5	sCD	1.643	5	1.447	19	2.639	9	2.651	8	2.822	6	2.848	0
	CD	3.818	5	4.002	2	4.651	1	4.617	1	4.609	8	4.652	0
0.4	sCD	1.645	6	1.449	23	2.641	9	2.653	10	2.820	7	2.848	0
	CD	3.819	6	4.018	3	4.633	1	4.618	1	4.605	14	4.652	0
0.3	sCD	1.648	8	1.451	32	2.642	13	2.653	14	2.821	9	2.848	0
	CD	3.822	8	4.029	4	4.623	1	4.621	1	4.606	25	4.652	0
0.2	sCD	1.652	13	1.451	48	2.643	21	2.655	21	2.821	15	2.848	0
	CD	3.825	13	4.039	6	4.624	1	**4.621**	**2**	4.605	43	4.652	0
0.1	sCD	1.655	26	1.452	96	2.644	42	2.656	41	2.845	1	2.848	0
	CD	3.827	26	4.050	12	4.625	3	4.623	4	4.648	1	4.652	0

Table 5.2: Table of results for experimentation on the Cube. For a selected point class, update formula, diffusion speed α, and error metric, points belonging to the class are denoised over multiple iterations until the error metric reaches a minimum. The minimum error value and the corresponding number of iterations are recorded in the table. For the Cube, using $\mathbf{v}_i^{\text{flat}}$ for flat points and $\mathbf{v}_i^{\text{feature}}$ for edge points and $\mathbf{v}_i^{c}$ for corner points with diffusion speeds $\alpha = [1, 0.2, 0.8]$ for 2 iterations is optimal. Results of experiments using these settings are marked in bolt.

diffusion speeds α	Error Metric $\times 10^{-5}$	Stanford Bunny											
		Flat Points				Edge Points				Corner Points			
		$\mathbf{v}_i^{\text{flat}}$		$\mathbf{v}_i^{\text{feature}}$		$\mathbf{v}_i^{e}$		$\mathbf{v}_i^{\text{feature}}$		$\mathbf{v}_i^{c}$		$\mathbf{v}_i^{\text{feature}}$	
		Error	Iterations	Error	Iterations	Error	Iterations	Error	Iterations	Error	Iterations	Error	Iterations
1	sCD	1.308	45	0.8078	8	1.411	1	1.372	2	1.468	1	1.461	3
	CD	2.334	38	2.157	1	2.428	0	2.428	0	2.428	0	**2.422**	**1**
0.9	sCD	1.307	49	0.8073	9	1.405	2	1.373	2	1.468	2	1.461	3
	CD	2.334	38	**2.150**	**1**	2.428	0	2.428	0	2.428	0	2.422	1
0.8	sCD	1.308	54	0.8055	10	1.400	2	1.374	3	1.467	2	1.461	4
	CD	2.334	50	2.152	1	2.428	0	2.428	0	2.428	0	2.422	1
0.7	sCD	1.309	62	0.9072	11	1.397	2	1.374	3	1.467	2	1.461	5
	CD	2.334	56	2.162	1	2.428	0	2.428	0	2.428	0	2.422	1
0.6	sCD	1.309	74	0.8080	14	1.395	2	1.375	4	1.467	2	1.461	3
	CD	2.335	65	2.181	1	2.428	0	2.428	1	2.428	0	2.422	1
0.5	sCD	1.311	91	0.8079	16	1.394	2	1.376	5	1.466	2	1.461	8
	CD	2.337	69	2.184	2	2.428	0	2.424	1	2.428	0	2.422	1
0.4	sCD	1.311	121	0.8149	11	1.395	3	1.377	6	1.466	2	1.462	5
	CD	2.336	102	2.187	2	2.428	0	**2.421**	**1**	2.428	0	2.422	1
0.3	sCD	1.313	159	0.8120	25	1.394	4	1.378	7	1.466	2	1.463	8
	CD	2.337	124	2.194	3	2.428	0	2.421	1	2.428	0	2.422	2
0.2	sCD	1.314	239	0.8156	28	1.395	5	1.380	9	1.466	4	1.462	12
	CD	2.338	181	2.201	5	2.423	1	2.422	1	2.423	1	2.422	3
0.1	sCD	1.316	475	0.8172	57	1.396	12	1.381	19	1.466	6	1.463	33
	CD	2.340	358	2.207	9	2.423	1	2.422	3	2.423	1	2.422	6

Table 5.3: Table of results for experimentation on the Stanford Bunny. For a selected point class, update formula, diffusion speed α, and error metric, points belonging to the class are denoised over multiple iterations until the error metric reaches a minimum. The minimum error value and the corresponding number of iterations are recorded in the table. For the Stanford Bunny, using $\mathbf{v}_i^{\text{feature}}$ for all points with diffusion speeds $\alpha = [0.9, 0.4, 1]$ for a single iteration is optimal. Results of experiments using these settings are marked in bolt.

diffusion speeds α	Error Metric $\times 10^{-5}$	Sphere				Torus			
		Flat Points							
		$\mathbf{v}_i^{\text{flat}}$		$\mathbf{v}_i^{\text{feature}}$		$\mathbf{v}_i^{\text{flat}}$		$\mathbf{v}_i^{\text{feature}}$	
		Error	Iterations	Error	Iterations	Error	Iterations	Error	Iterations
1	sCD	2.167	2	1.907	9	1.121	2	0.9893	8
	CD	**5.534**	**2**	5.902	1	**2.897**	**2**	3.078	1
0.9	sCD	2.170	3	1.957	3	1.123	2	0.9900	9
	CD	5.539	3	5.917	1	2.898	2	3.069	1
0.8	sCD	2.173	3	1.908	11	1.128	3	0.9933	8
	CD	5.542	3	5.931	1	2.903	2	3.075	1
0.7	sCD	2.178	4	1.881	11	1.129	3	0.9879	12
	CD	5.549	3	5.989	1	2.904	3	3.096	1
0.6	sCD	2.181	5	1.909	14	1.133	4	0.9970	7
	CD	5.550	4	6.075	1	2.909	3	3.132	1
0.5	sCD	2.185	6	1.908	18	1.136	5	0.9881	16
	CD	5.555	5	6.117	2	2.910	4	3.129	2
0.4	sCD	2.188	7	1.905	22	1.138	6	0.9984	11
	CD	5.558	7	6.149	2	2.913	6	3.141	2
0.3	sCD	2.190	10	1.913	29	1.141	8	0.9998	14
	CD	5.560	9	6.152	4	2.915	8	3.151	3
0.2	sCD	2.194	15	1.965	16	1.144	13	0.9999	22
	CD	5.564	14	6.098	5	2.918	12	3.162	5
0.1	sCD	2.198	32	1.906	90	1.146	26	1.001	43
	CD	5.567	29	6.193	9	2.920	24	3.172	10

Table 5.4: Table of results for experimentation on the models without features: The sphere and the torus. Denoising only flat points, since the models contain no edge or corner points. For the flat points, an update formula, a diffusion speed α, and an error metric; points belonging to the class are denoised over multiple iterations until the error metric reaches a minimum. The minimum error value and the corresponding number of iterations are recorded in the table. For both the sphere and the torus, using $\mathbf{v}_i^{\text{flat}}$ for all points with diffusion speeds $\alpha_0 = 1$ for 2 iterations is optimal. Results of experiments using these settings are marked in bolt.

Another observation from the tables is that one of the error metrics often increases while the other decreases. For example, in Table 5.1, when examining flat points with a diffusion speed of $\alpha = 1$, we see that the sCD is higher for $\mathbf{v}_i^{flat}$ points, whereas the CD is higher for $\mathbf{v}_i^{feature}$ points. This effect occurs because the $\mathbf{v}_i^{feature}$ points are generally closer to the ground truth points than the $\mathbf{v}_i^{flat}$ points. However, it is important to note that the CD metric also measures how well the ground truth points are covered by the denoised points or how well the denoised points cover the surface of the model. Because $\mathbf{v}_i^{feature}$ points are allowed to move freely in three dimensions, rather than being constrained to movement along the normal direction, the denoised result can leave holes in the pointcloud, resulting in a higher CD. This effect is commonly observed when trying to denoise flat points using the $\mathbf{v}_i^{feature}$ update formula.

The tables show that five models were used in the experiments. The rationale for selecting these models is as follows. The Cube and Fandisk contain sharp features with angles around $\frac{\pi}{2}$. These right-angle features can be accurately detected by PCA-based algorithms, making these models suitable for testing the baseline performance on feature points. The Sphere and Torus contain no sharp features, making them ideal for evaluating denoising performance on flat or slightly curved surfaces. Finally, the Stanford Bunny includes curved surfaces adjacent to curved edges, a characteristic not present in the other models, making it a valuable addition to the dataset for testing under more complex geometric conditions.

For the final pipeline, we need to select the update strategy, diffusion speed, and number of iterations in advance. To determine the optimal configuration, we choose settings where the number of iterations required to reach a minimal error is the same across all point classes. We focus on minimizing the CD error, as the sCD metric does not sufficiently penalize uncovered ground truth points. Among the configurations with equal iteration counts per class, the one with the lowest combined minimal error is selected. The optimal settings for each model are presented in the tables.

Given the results, in our pipeline, the strategy is to use $\mathbf{v}_i^{\text{flat}}$ for flat points and $\mathbf{v}_i^{\text{feature}}$ for other points with a diffusion speed of $\alpha = [1, 0.2, 1]$ for 2 iterations.

5.4 Method Comparison

In this section, we will compare our final setup with the method CPSD (Constraint-based Point Set Denoising) from (Yadav et al., 2018) and CTD-QEM (Constrained Two-Direction Quadric Error Metrics) from (Wei et al., 2021). CPSD uses range based neighborhoods Ω_i^r, where

$$r = \frac{1}{|V||\Omega_i^k|} \sum_{i=0}^{|V|} \sum_{j \in \Omega_i^k} \mathbf{v}_j - \mathbf{v}_i \tag{5.3}$$

is the average distance between points in the pointcloud and $k = 6$. Since the CTD-QEM algorithm itself is a position updating algorithm, it does not do normal smoothing. Therefore, in this comparison, it used our normal smoothing algorithm to smooth the point normals. For

Figure 5.7: Test set consisting of models from Shen et al. (2022), which were 3D printed and subsequently scanned to capture real-world noise characteristics. The zoom-in highlights printing artifacts, such as the vertical lines visible on the Suit Man model.

this comparison, CPSD does 150 iterations, because Yadav et al. (2018) proposes this number of iterations for CAD models like the cube and the fandisk, CTD-QEM does 15 iterations as suggested by Wei et al. (2021) and our method does 2 iterations as discussed in the previous section. We compare on different models by adding gaussian noise with noise level 0.2. We test with the two error metrics sCD and CD and show the results in Table 5.5 with visualizations and zoom-ins in Appendix B. Furthermore, we evaluate our method on a test dataset from Shen et al. (2022), consisting of models that were 3D printed and subsequently scanned to capture real-world noise introduced by the printing and scanning processes (see Figure 5.7). The results are presented in Table 5.6, with visualizations provided in Appendix C.

Table 5.5 shows that all denoising methods achieve lower sCD values, indicating that the denoised pointclouds are closer to the ground truth points on average. At first glance, the CTD-QEM method appears to perform best in terms of sCD. However, it does not outperform the other methods on the CD metric. This discrepancy can be attributed to the clustering behavior of

Model	Error Metric $\times 10^{-5}$	Noisy	CPSD	CTD-QEM	Ours
Cube	sCD	2.880	1.589	**1.260**	1.493
	CD	4.735	4.570	5.607	**3.810**
Torus	sCD	2.204	1.920	**1.061**	1.162
	CD	3.586	4.820	4.891	**2.931**
Fandisk	sCD	1.462	0.978	**0.683**	0.833
	CD	2.455	2.583	3.202	**2.070**
Stanford Bunny	sCD	1.499	1.075	**0.709**	1.347
	CD	2.437	2.545	3.405	**2.365**
Pyramid	sCD	1.717	1.430	**0.853**	0.929
	CD	2.870	3.801	3.967	**2.354**
Sphere	sCD	4.308	3.334	**2.087**	2.245
	CD	6.940	8.205	8.949	**5.621**
Cylinder	sCD	3.241	2.241	**1.571**	1.711
	CD	5.389	6.132	7.183	**4.397**
Cone	sCD	1.544	1.191	**0.737**	0.815
	CD	2.560	3.105	3.313	**2.072**
Tetrahedron	sCD	1.646	1.146	**0.823**	0.892
	CD	2.706	3.043	3.562	**2.221**
Armadillo	sCD	1.297	1.797	**0.681**	0.854
	CD	2.186	3.663	3.323	**1.975**
Cow	sCD	1.052	1.024	**0.519**	0.637
	CD	1.718	2.201	2.531	**1.505**
Teapot	sCD	1.445	1.274	**0.738**	0.852
	CD	2.429	3.097	3.424	**2.081**

Table 5.5: Two error metrics of noisy or denoised pointclouds compared to ground truth pointclouds for 12 different models. Visualizations with zoom-ins can be found in Appendix B.

Model	Error Metric $\times 10^{-5}$	Noisy	CPSD	CTD-QEM	Ours
Ankylosaurus	sCD	0.379	0.350	**0.191**	0.221
	CD	0.440	0.521	0.471	**0.347**
Armor Cat	sCD	0.751	0.715	**0.454**	0.482
	CD	1.182	1.416	1.406	**1.090**
Cat Fangs	sCD	0.727	0.673	**0.451**	0.495
	CD	0.802	0.949	0.895	**0.706**
Deer	sCD	1.150	1.211	**0.958**	1.015
	CD	**1.092**	1.337	1.308	1.133
Direwolf	sCD	1.121	1.067	**0.838**	0.912
	CD	**2.347**	2.430	2.890	2.429
Emissary Wolf	sCD	0.394	0.388	**0.274**	0.279
	CD	**1.252**	1.493	1.587	1.313
Girl	sCD	0.890	0.797	**0.481**	0.527
	CD	1.312	1.487	1.484	**1.127**
Goblin	sCD	0.634	0.659	0.462	**0.432**
	CD	**11.572**	12.142	13.435	12.525
Goku	sCD	0.509	0.477	**0.394**	0.425
	CD	**0.621**	0.714	0.776	0.661
Messi	sCD	1.008	0.858	**0.587**	0.650
	CD	1.433	1.599	1.589	**1.298**
Minion Ghost	sCD	1.178	0.971	**0.653**	0.748
	CD	1.373	1.482	1.410	**1.157**
Nut	sCD	10.685	12.818	**7.455**	10.685
	CD	**11.500**	17.874	17.617	12.304
Putin	sCD	1.196	0.951	**0.667**	0.748
	CD	1.376	1.444	1.453	**1.131**
Snoopy Flying Face	sCD	1.439	1.367	**0.750**	0.876
	CD	1.824	2.232	2.047	**1.521**
Snoopy	sCD	1.781	1.443	**1.295**	1.305
	CD	2.038	2.103	2.501	**1.929**
Spaghetti Detective	sCD	0.683	0.641	**0.435**	0.456
	CD	2.055	2.058	2.380	**2.005**
Stitch Guitar	sCD	0.758	0.841	**0.487**	0.546
	CD	2.009	2.253	2.394	**1.988**
Stitch Stand	sCD	0.593	0.545	**0.423**	0.454
	CD	0.904	0.949	1.074	**0.884**
Suit Man	sCD	2.075	2.101	**1.558**	1.668
	CD	2.334	2.893	3.053	**2.286**
Tp	sCD	1.167	1.102	**0.925**	0.950
	CD	**3.994**	4.068	4.970	4.366

Table 5.6: Two error metrics of noisy or denoised pointclouds compared to ground truth pointclouds for different models.

CTD-QEM: it relocates multiple points to the same optimal positions, resulting in sparse regions or holes in the pointcloud, which can also be seen in the visualizations in Appendix B. As a result, when calculating CD, where each ground truth point must find the closest point in the denoised pointcloud, many points remain unconvered, increasing the overall error. This highlights the need for a balance between minimizing point-wise distances and preserving spatial coverage. Our method, while scoring slightly worse on the sCD metric, significantly outperforms on the CD metric, indicating better surface coverage and fewer holes in the reconstruction.

Table 5.6 shows that our method does not outperform the other methods in every test case. Our method fails to achieve lower CD errors for the models Deer, Direwolf, Emissary Wolf, Goblin, Goku, Nut, Suit Man, and Tp. Upon closer inspection, this is likely caused by discrepancies between the ground truth models and their corresponding noisy versions. Many ground truth models include internal surfaces that do not exist in the printed and scanned noisy versions. As these surfaces disappear in the scanning process, the CD error increases significantly due to uncovered ground truth points. Additionally, deformations introduced during the printing process contribute to unreliable error measurements. For example, the feet of the Deer model are merged to increase stability during printing, resulting in a structural deviation from the ground truth that causes mismatches with the scanned pointcloud. Similar artifacts and discrepancies are illustrated in Figure 5.8. The Nut model (top left) has a missing piece at the edge of the hat and its edges are slightly curved upwards, leading to higher errors in those regions. The Goku model (top right) includes a hidden surface on the bottom of the foot that does not appear in the scan. The Goblin model (bottom left) contains an internal surface separating the torso and hip, which is lost during scanning. The Suit Man model (bottom middle) shows minor deformations in the feet, while the Emissary Wolf model (bottom right) contains a ring-shaped surface representing a necklace that is not captured in the noisy scan. These types of hidden or altered surfaces introduce large Chamfer Distance errors that cannot be mitigated by any denoising strategy, as the missing geometry fundamentally prevents accurate reconstruction. Despite these challenges, the sCD error shows that the absolute distance from the noisy points to the ground truth decreases after denoising. This indicates that the denoising process is effective, even if the CD metric does not fully reflect this improvement due to structural differences in the models.

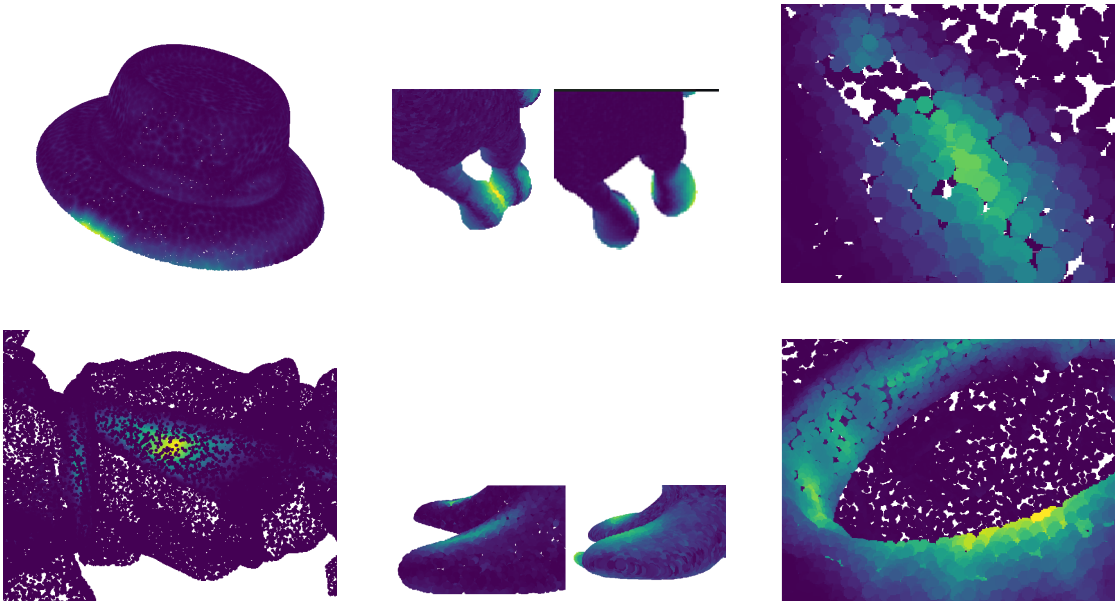

Figure 5.8: Six models from the test set exhibiting artifacts. The color shows the Chamfer Distance, where yellow points indicate regions where no corresponding noisy points are found nearby in Euclidean space. The models shown from top left to bottom right are Nut, Deer, Goku, Goblin, Suit Man, and Emissary Wolf. The artifacts include surface deformations and surfaces within the volume of the model.

Chapter 6

Conclusion

In this thesis, the CPSD (Constraint-based Point Set Denoising) method proposed by Yadav et al. (2018) was explored, analysed and improved upon. The proposed improvements were compared to both the original CPSD method and the CTD-QEM (Constrained Two-Direction Quadric Error Metrics) method from Wei et al. (2021). In the feature detection stage, various voting tensors, neighborhood sizes, neighborhood filters and feature space thresholds were evaluated to optimize the classification of points into flat, edge and corner categories. For the optimized classification, experiments were conducted to minimize the reconstruction error and determine the optimal hyperparameters for the pipeline.

The experimental results demonstrate that the proposed method achieves better performance on both the CD and sCD error metrics compared to the original CPSD implementation, while also reducing computational time due to the fewer number of iterations required for convergence. This highlights the potential of the improved pipeline for efficient and accurate pointcloud denoising tasks.

However, some limitations remain. The current update formulas assume that underlying surfaces are locally planar, neglecting curvature, which can reduce reconstruction quality in areas with complex geometry. Additionally, the evaluation used real-world noisy scans with synthetic ground truth models that contained structural discrepancies. This, combined with the CD metric, leads to unfair or inaccurate evaluations, as internal surfaces present in synthetic models often do not exist in the scanned data. Developing a test set without such internal surfaces or structural deformations introduced for printing stability could result in more reliable evaluations.

For future work, it is recommended to explore alternative update formulas that account for surface curvature to further improve reconstruction accuracy. Introducing a curvature-based guidance signal alongside normal vectors could increase update accuracy, particularly on curved surfaces such as those in the Sphere, Torus, and Stanford Bunny models. This would likely yield more precise optimal point positions and reduce inflation effects, where models expand over multiple iterations. Furthermore, exploring alternative error metrics that not only measure point-wise closeness but also penalize point clustering while ignoring irrelevant internal surfaces

in the ground truth models could provide more meaningful evaluations of denoising performance. Finally, since the pipeline is modular, any component can be replaced with a neural network to further enhance reconstruction accuracy. For instance, existing neural networks that predict normal vectors or classify points could be integrated into this pipeline to improve overall performance.

In conclusion, this thesis presents a substantial improvement to the CPSD denoising pipeline, both in terms of reconstruction accuracy and computational efficiency. The results demonstrate that careful design of classification and updating strategies can lead to significant gains, providing a solid foundation for further research in pointcloud denoising and processing.

Bibliography

M. Alexa, J. Behr, D. Cohen-Or, S. Fleishman, D. Levin, and C. T. Silva. Point set surfaces. In *Proceedings Visualization, 2001. VIS'01.*, pages 21–29. IEEE, 2001.

A. Boulch and R. Marlet. Deep learning for robust normal estimation in unstructured point clouds. In *Computer Graphics Forum*, volume 35, pages 281–290. Wiley Online Library, 2016.

N. Charron, S. Phillips, and S. L. Waslander. De-noising of lidar point clouds corrupted by snowfall. In *2018 15th Conference on Computer and Robot Vision (CRV)*, pages 254–261. IEEE, 2018.

X. Chen, S. Li, B. Mersch, L. Wiesmann, J. Gall, J. Behley, and C. Stachniss. Moving object segmentation in 3d lidar data: A learning-based approach exploiting sequential data. *IEEE Robotics and Automation Letters*, 6(4):6529–6536, 2021.

J. Demantké, C. Mallet, N. David, and B. Vallet. Dimensionality based scale selection in 3d lidar point clouds. In *Laserscanning*, 2011.

J. Digne and C. De Franchis. The bilateral filter for point clouds. *Image Processing On Line*, 7: 278–287, 2017.

P. Dong and Q. Chen. *LiDAR remote sensing and applications*. CRC Press, 2017.

S. Garagnani and A. M. Manferdini. Parametric accuracy: building information modeling process applied to the cultural heritage preservation. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 40:87–92, 2013.

S. B. Gokturk, H. Yalcin, and C. Bamji. A time-of-flight depth sensor-system description, issues and solutions. In *2004 conference on computer vision and pattern recognition workshop*, pages 35–35. IEEE, 2004.

X.-F. Han, J. S. Jin, M.-J. Wang, and W. Jiang. Guided 3d point cloud filtering. *Multimedia Tools and Applications*, 77:17397–17411, 2018a.

X.-F. Han, J. S. Jin, M.-J. Wang, and W. Jiang. Iterative guidance normal filter for point cloud. *Multimedia Tools and Applications*, 77:16887–16902, 2018b.

K. He, J. Sun, and X. Tang. Guided image filtering. *IEEE transactions on pattern analysis and machine intelligence*, 35(6):1397–1409, 2012.

P. Hermosilla, T. Ritschel, and T. Ropinski. Total denoising: Unsupervised learning of 3d point cloud cleaning. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 52–60, 2019.

H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Surface reconstruction from unorganized points. In *Proceedings of the 19th annual conference on computer graphics and interactive techniques*, pages 71–78, 1992.

F. Leberl, A. Irschara, T. Pock, P. Meixner, M. Gruber, S. Scholz, and A. Wiechert. Point clouds. *Photogrammetric Engineering & Remote Sensing*, 76(10):1123–1134, 2010.

D. Levin. The approximation power of moving least-squares. *Mathematics of computation*, 67 (224):1517–1531, 1998.

Q. Li, Y.-S. Liu, J.-S. Cheng, C. Wang, Y. Fang, and Z. Han. Hsurf-net: Normal estimation for 3d point clouds by learning hyper surfaces. *Advances in Neural Information Processing Systems*, 35:4218–4230, 2022.

Y. Li, L. Ma, Z. Zhong, F. Liu, M. A. Chapman, D. Cao, and J. Li. Deep learning for lidar point clouds in autonomous driving: A review. *IEEE Transactions on Neural Networks and Learning Systems*, 32(8):3412–3432, 2020.

Y. Lipman, D. Cohen-Or, D. Levin, and H. Tal-Ezer. Parameterization-free projection for geometry reconstruction. *ACM Transactions on Graphics (ToG)*, 26(3):22–es, 2007.

Z. Liu, X. Xiao, S. Zhong, W. Wang, Y. Li, L. Zhang, and Z. Xie. A feature-preserving framework for point cloud denoising. *Computer-Aided Design*, 127:102857, 2020.

S. Luo and W. Hu. Diffusion probabilistic models for 3d point cloud generation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 2837–2845, 2021a.

S. Luo and W. Hu. Score-based point cloud denoising. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4583–4592, 2021b.

A. Miropolsky and A. Fischer. Reconstruction with 3d geometric bilateral filter. In *Proceedings of the ninth ACM symposium on Solid modeling and applications*, pages 225–229, 2004.

C. R. Qi, H. Su, K. Mo, and L. J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660, 2017.

C. Rocchini, P. Cignoni, C. Montani, P. Pingi, and R. Scopigno. A low cost 3d scanner based on structured light. In *computer graphics forum*, volume 20, pages 299–308. Wiley Online Library, 2001.

Y. Shen, H. Fu, Z. Du, X. Chen, E. Burnaev, D. Zorin, K. Zhou, and Y. Zheng. Gcn-denoiser: mesh denoising with graph convolutional networks. *ACM Transactions on Graphics (TOG)*, 41 (1):1–14, 2022.

J. D. Stets, Y. Sun, W. Corning, and S. W. Greenwald. Visualization and labeling of point clouds in virtual reality. In *SIGGRAPH Asia 2017 Posters*, pages 1–2. 2017.

Y. Sun, S. Schaefer, and W. Wang. Denoising point sets via l0 minimization. *Computer Aided Geometric Design*, 35:2–15, 2015.

C. Tomasi and R. Manduchi. Bilateral filtering for gray and color images. In *Sixth international conference on computer vision (IEEE Cat. No. 98CH36271)*, pages 839–846. IEEE, 1998.

A. Vahdat, F. Williams, Z. Gojcic, O. Litany, S. Fidler, K. Kreis, et al. Lion: Latent point diffusion models for 3d shape generation. *Advances in Neural Information Processing Systems*, 35: 10021–10039, 2022.

M. Wei, H. Chen, Y. Zhang, H. Xie, Y. Guo, and J. Wang. Geodualcnn: Geometry-supporting dual convolutional neural network for noisy point clouds. *IEEE Transactions on Visualization and Computer Graphics*, 29(2):1357–1370, 2021.

S. K. Yadav, U. Reitebuch, M. Skrodzki, E. Zimmermann, and K. Polthier. Constraint-based point set denoising using normal voting tensor and restricted quadratic error metrics. *Computers & Graphics*, 74:234–243, 2018.

W. Yang, Z. Gong, B. Huang, and X. Hong. Lidar with velocity: Motion distortion correction of point clouds from oscillating scanning lidars. *arXiv preprint arXiv:2111.09497*, 2021.

Y. Zhang, B. Dong, and Z. Lu. ℓ_0 minimization for wavelet frame based image restoration. *Mathematics of Computation*, 82(282):995–1015, 2013.

Y. Zheng, G. Li, S. Wu, Y. Liu, and Y. Gao. Guided point cloud denoising via sharp feature skeletons. *The Visual Computer*, 33:857–867, 2017.

Appendix A

Point Classification Experiments

This appendix contains the results of point classification experiments. Many settings are tried, where points are marked purple if they are correctly classified and yellow otherwise. The visualizations contain the ground truth models, because it is easier to derive conclusions from them. This is because it's easier to identify the location of the features and whether feature points or surrounding flat points are misclassified.

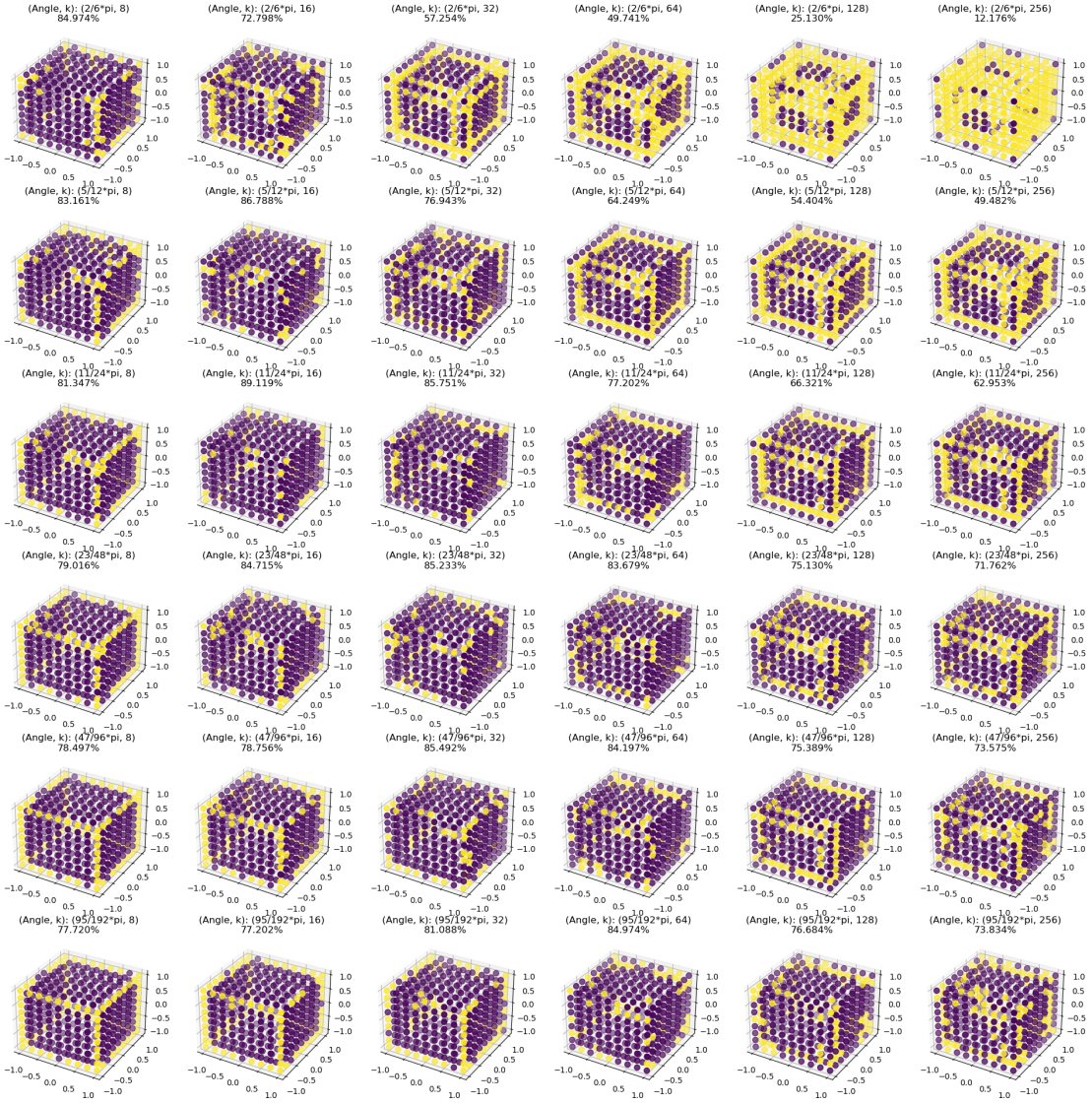

Figure A.1: A cube with 386 points is classified with different setting for the threshold angle ρ and different neighborhood sizes k. Purple points are correctly classified and yellow points are wrongly classified. The percentage is the percentage of correctly classified points.

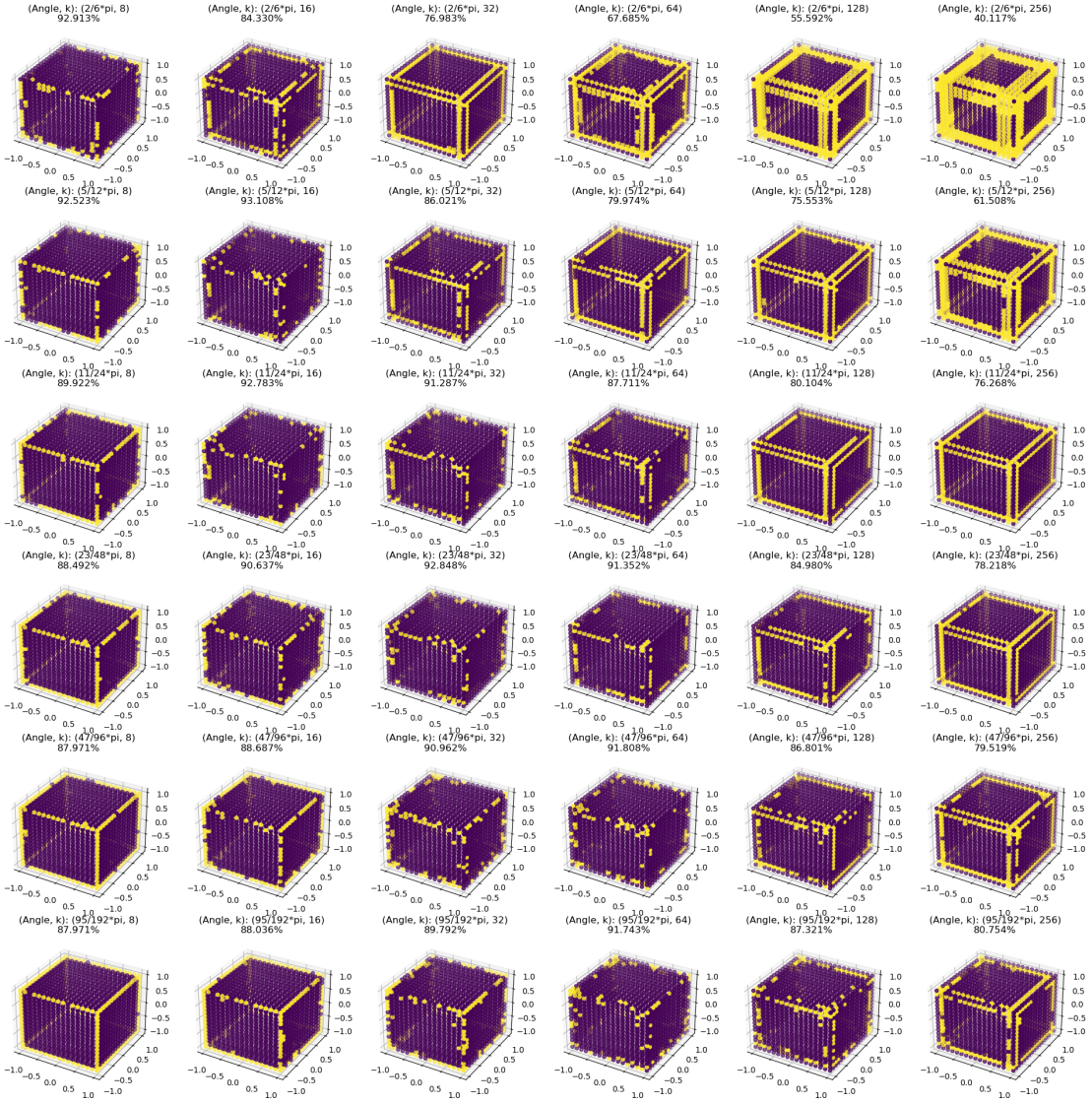

Figure A.2: A cube with 1538 points is classified with different setting for the threshold angle ρ and different neighborhood sizes k. Purple points are correctly classified and yellow points are wrongly classified. The percentage is the percentage of correctly classified points.

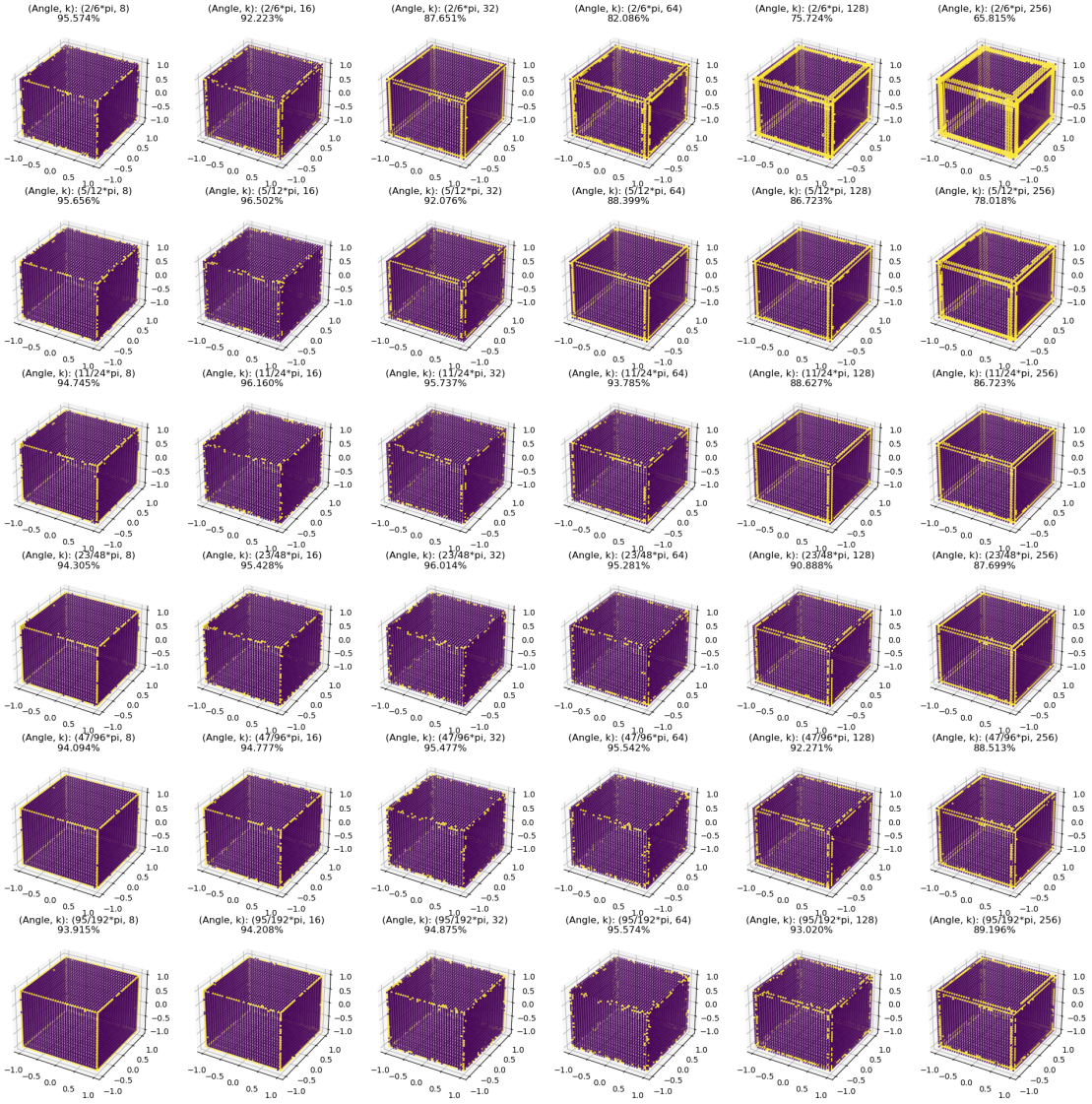

Figure A.3: A cube with 6146 points is classified with different setting for the threshold angle ρ and different neighborhood sizes k. Purple points are correctly classified and yellow points are wrongly classified. The percentage is the percentage of correctly classified points.

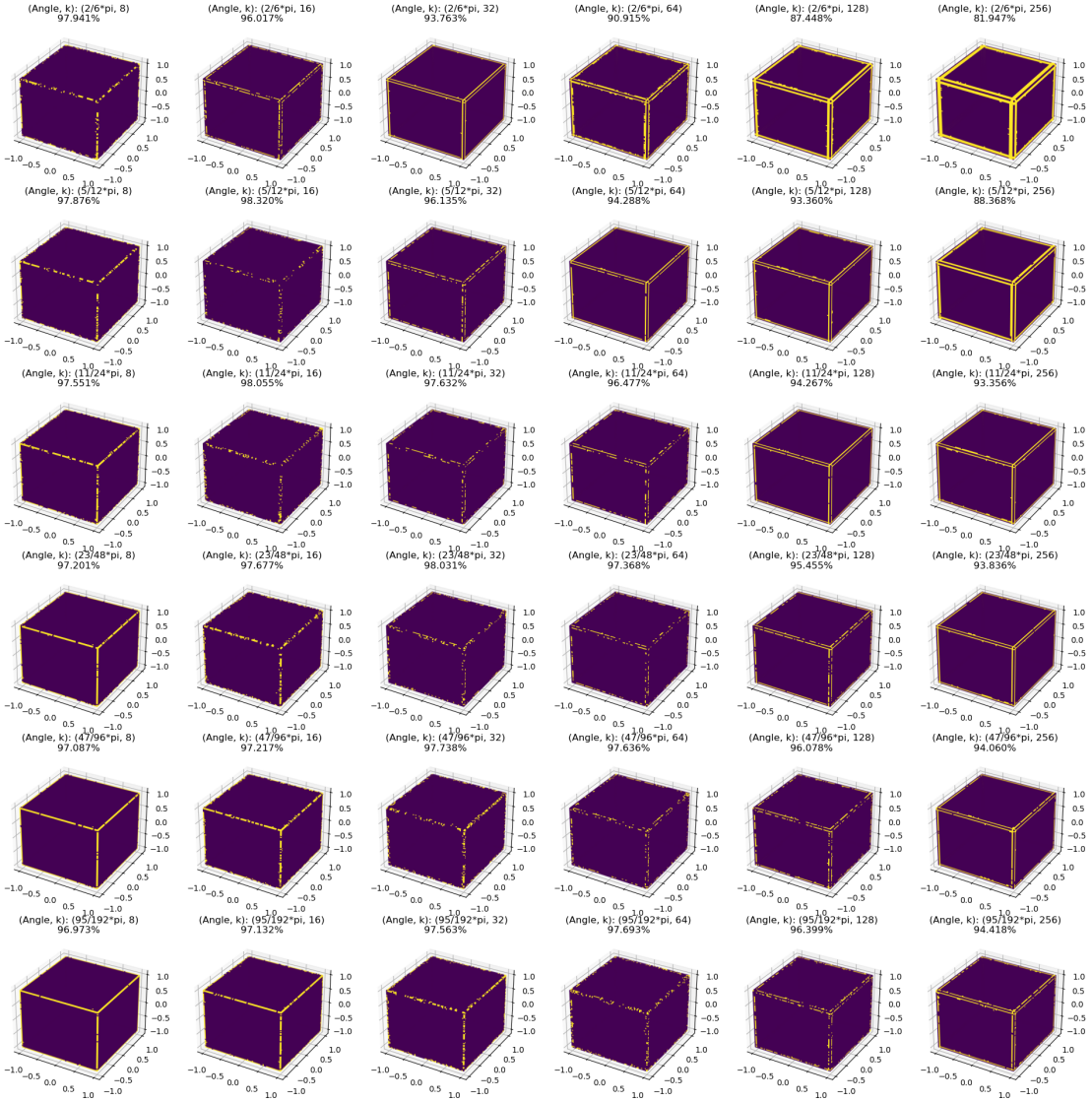

Figure A.4: A cube with 24578 points is classified with different setting for the threshold angle ρ and different neighborhood sizes k. Purple points are correctly classified and yellow points are wrongly classified. The percentage is the percentage of correctly classified points.

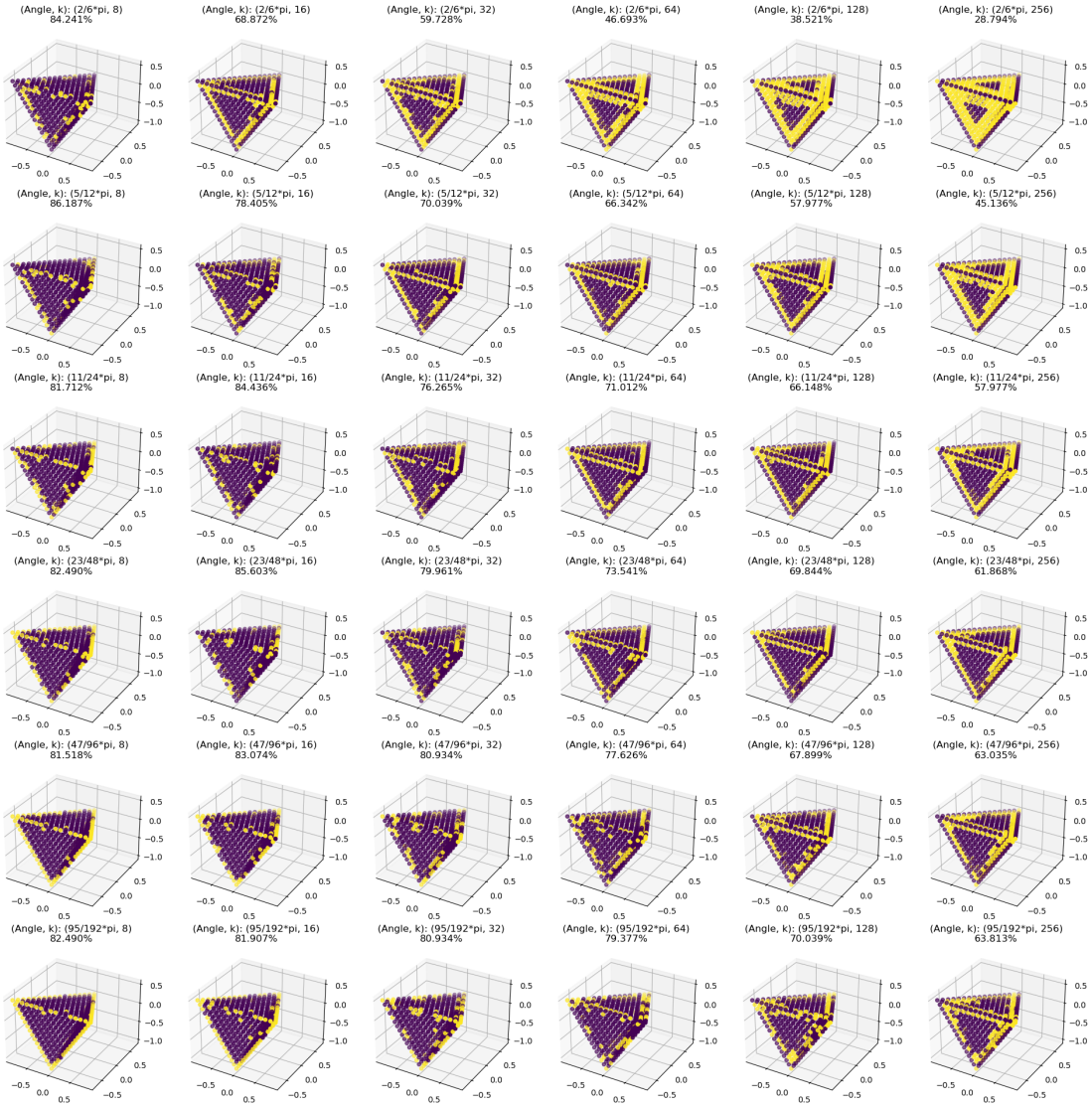

Figure A.5: A tetrahedron with 514 points is classified with different setting for the threshold angle ρ and different neighborhood sizes k. Purple points are correctly classified and yellow points are wrongly classified. The percentage is the percentage of correctly classified points.

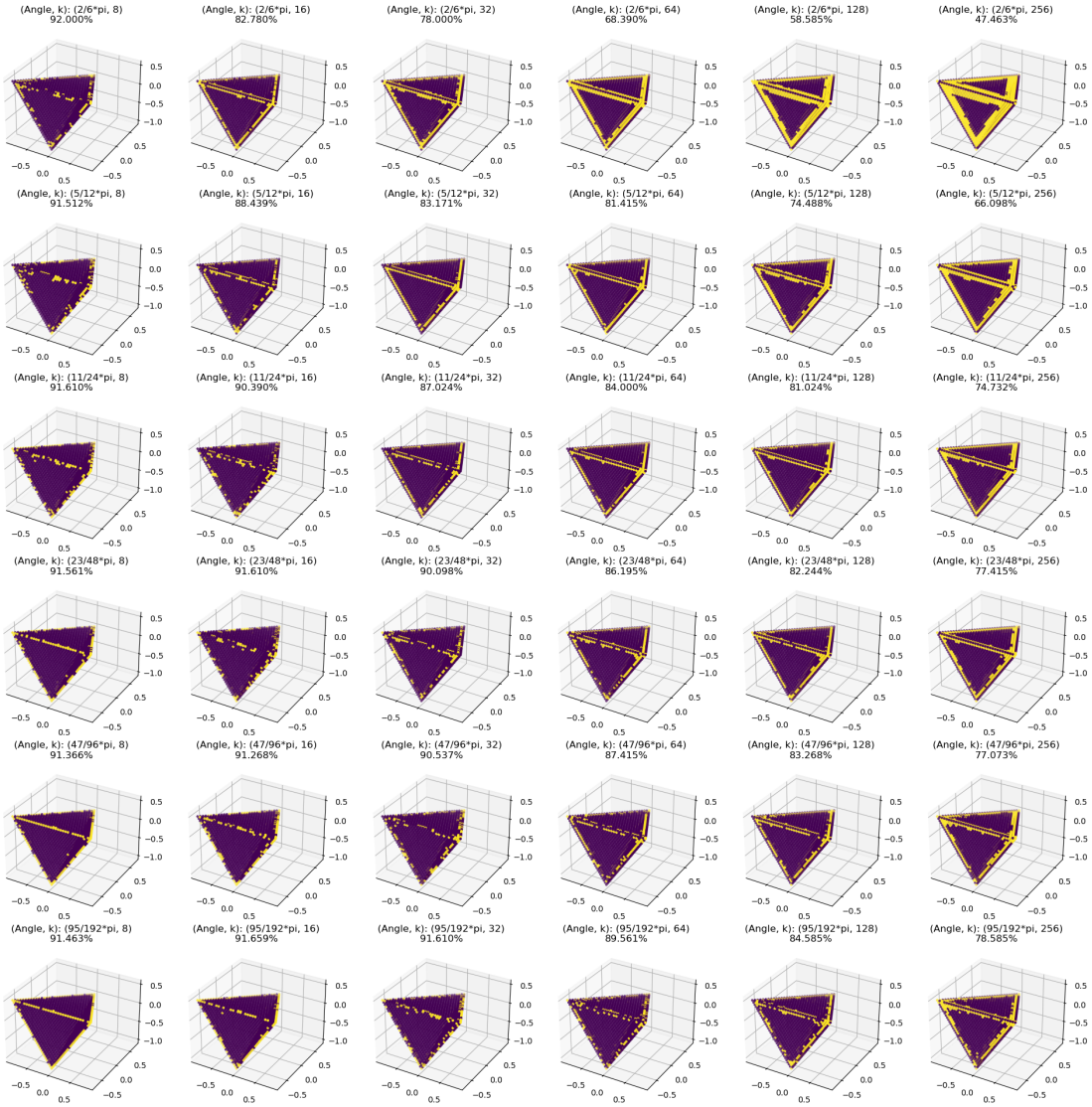

Figure A.6: A tetrahedron with 2050 points is classified with different setting for the threshold angle ρ and different neighborhood sizes k. Purple points are correctly classified and yellow points are wrongly classified. The percentage is the percentage of correctly classified points.

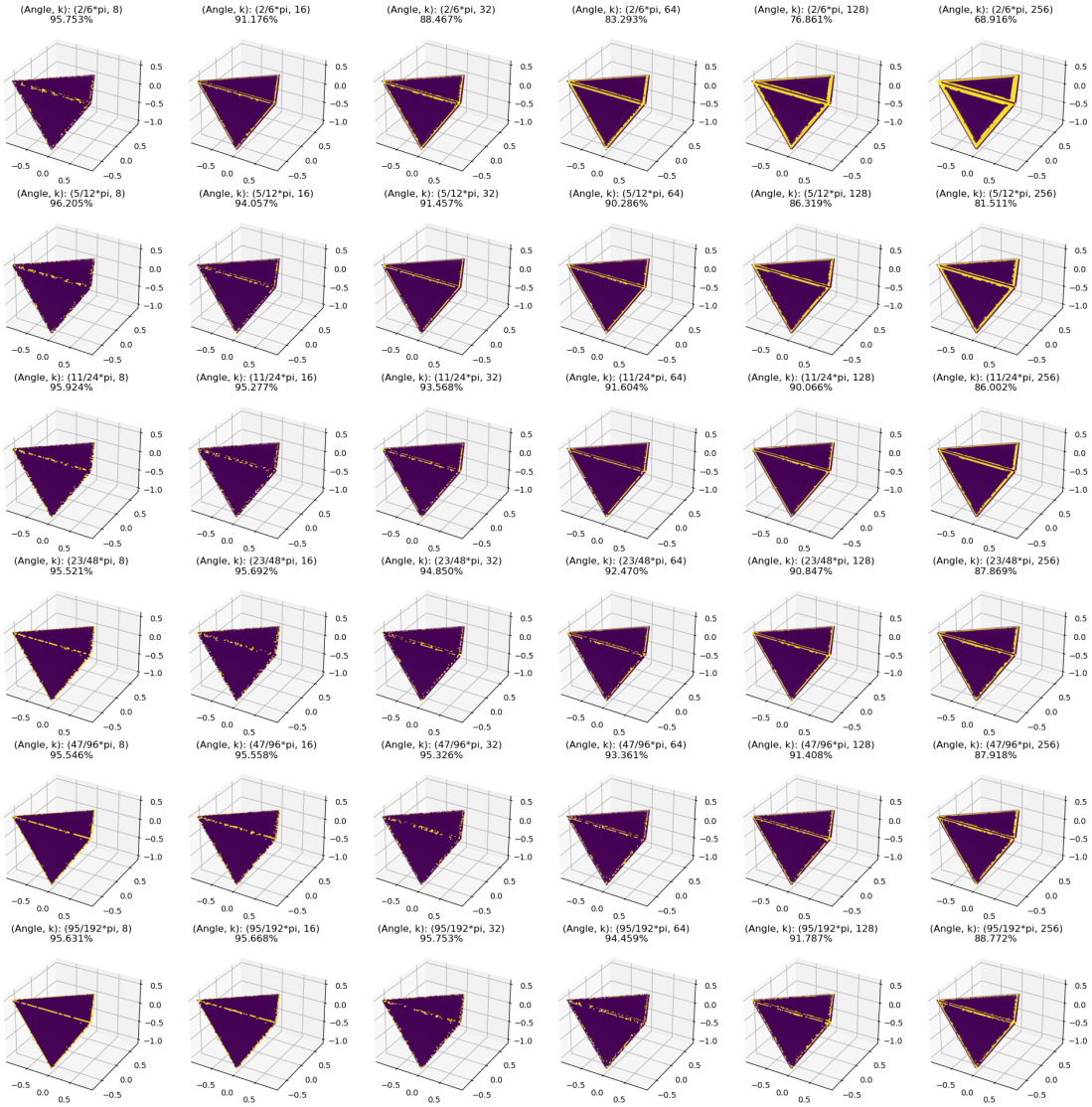

Figure A.7: A tetrahedron with 8194 points is classified with different setting for the threshold angle ρ and different neighborhood sizes k. Purple points are correctly classified and yellow points are wrongly classified. The percentage is the percentage of correctly classified points.

Appendix B

Visual Comparison of Denoising Methods on Synthetic Models

In the figures below, different denoising methods are compared on different models with noise level 0.2. 2^{14} points have been sampled on the models. The methods CPSD, CTD-QEM and our method are compared with the noisy input and eachother. The models Cube, Torus, Fandisk, Stanford Bunny, Pyramid, Sphere, Cylinder, Cone, Tetrahedron, Armadillo, Cow and Teapot have been chosen, because they contain a combination of flat and curved surfaces and sharp and complex features.

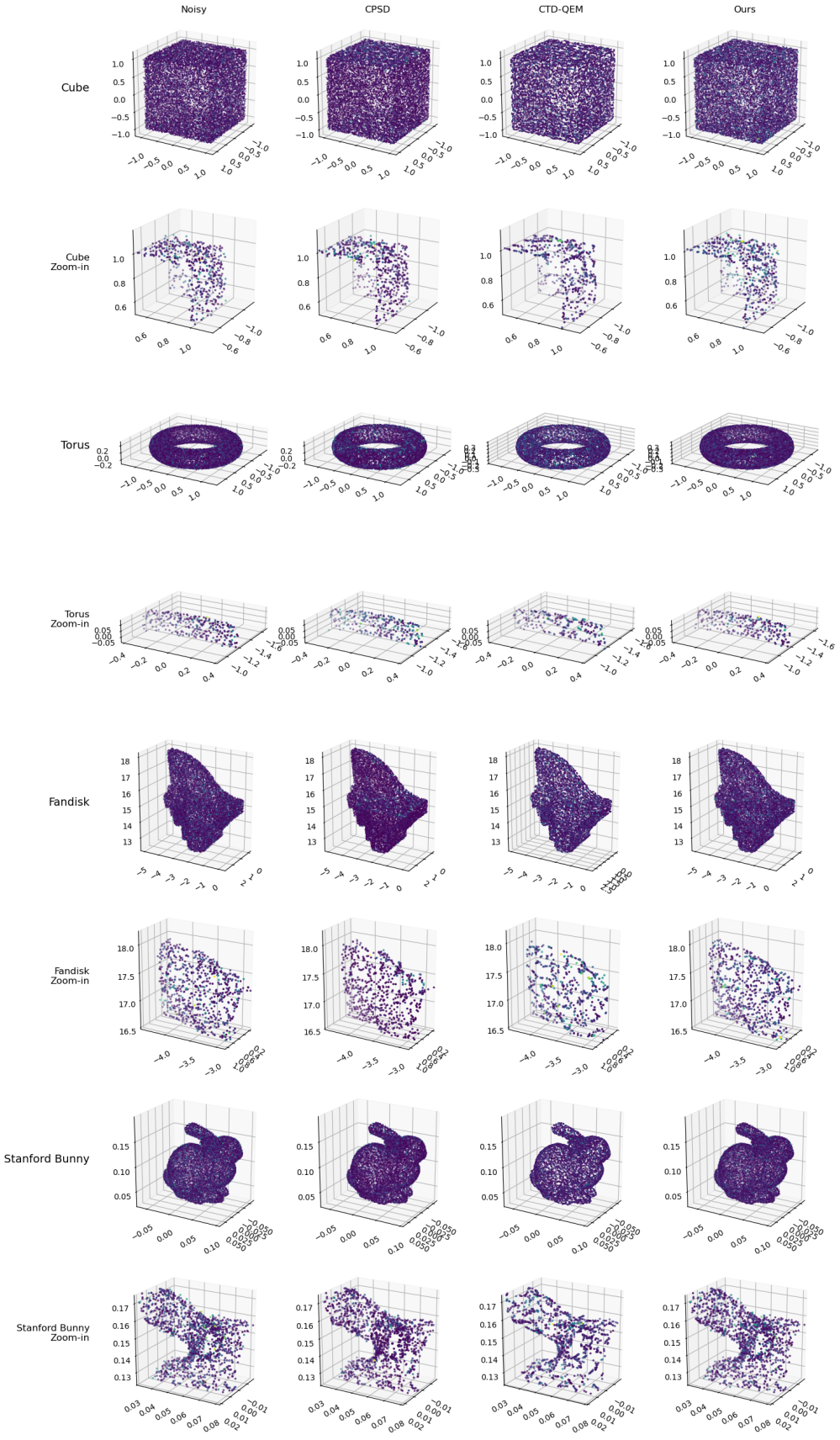

Figure B.1: From left to right the noisy model and denoised results are shown. Vertically, results for the models Cube, Torus, Fandisk and Stanford Bunny are shown.

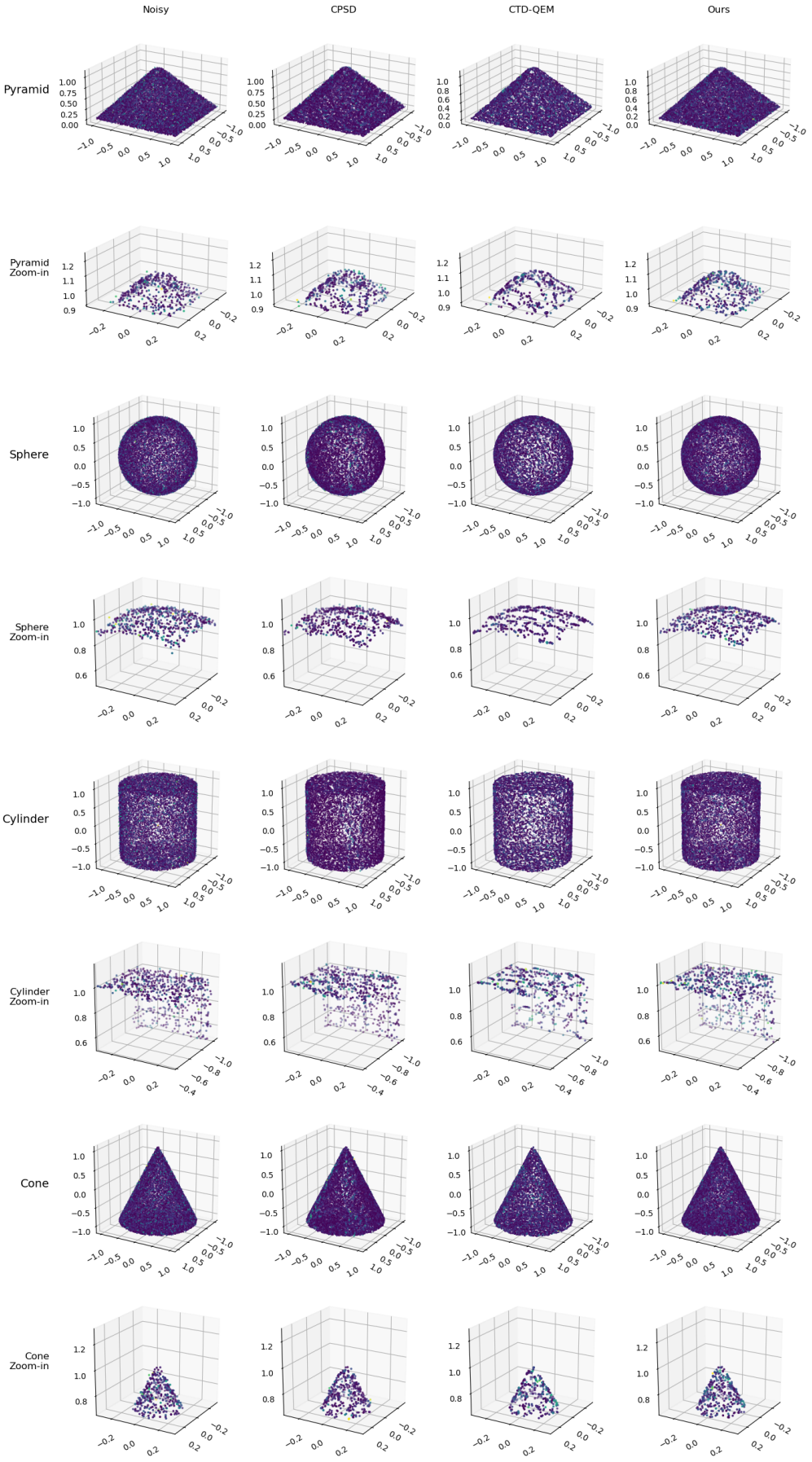

Figure B.2: From left to right the noisy model and denoised results are shown. Vertically, results for the models Pyramid, Sphere, Cylinder and Cone are shown.

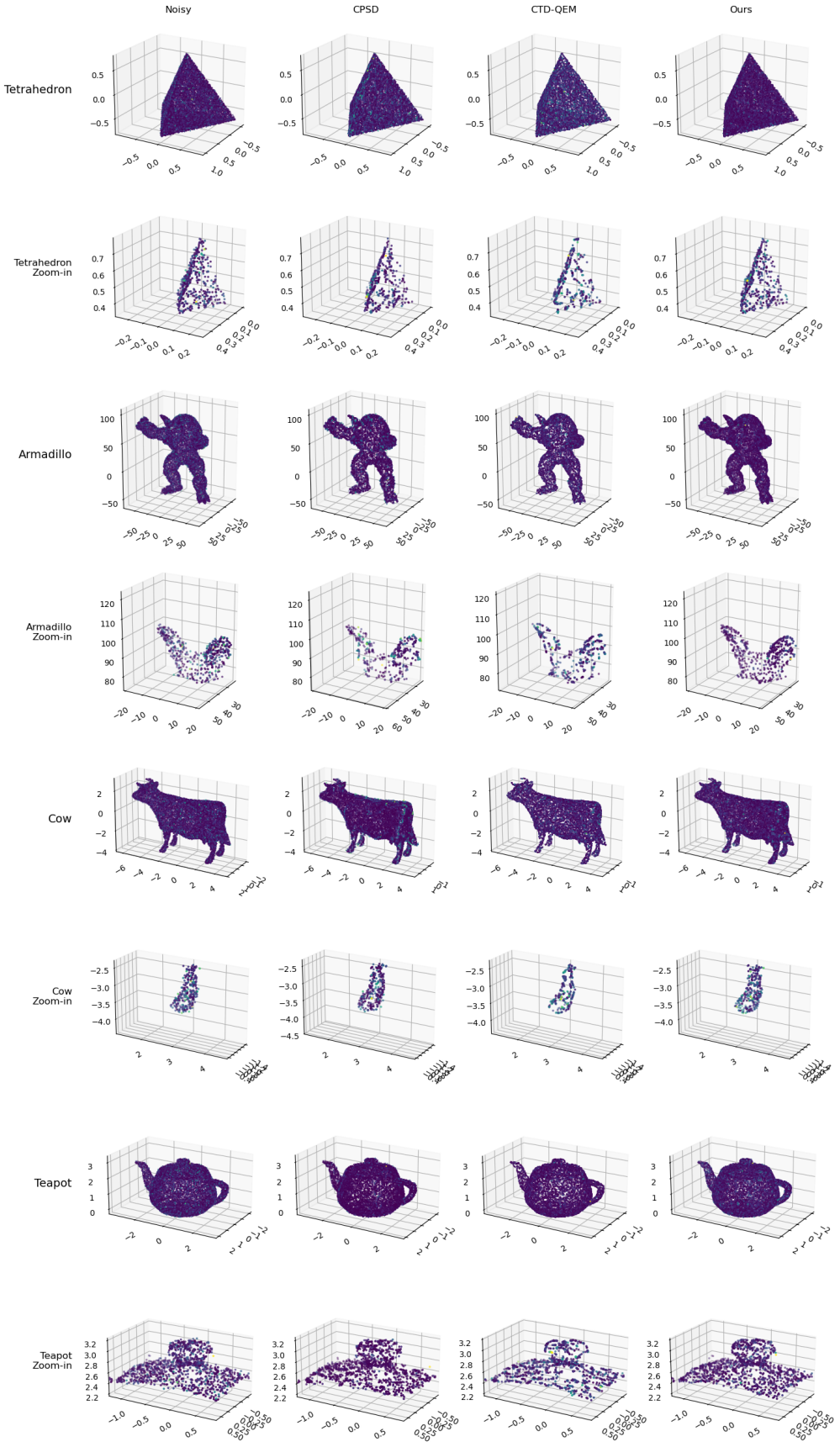

Figure B.3: From left to right the noisy model and denoised results are shown. Vertically, results for the models Tetrahedron, Armadillo, Cow and Teapot are shown.

Appendix C

Visual Comparison of Denoising Methods on Printed and Scanned Models

This appendix presents visual comparisons of the denoising results on models from the Printed Dataset Shen et al. (2022), using the original CPSD pipeline, an implementation of CTD-QEM, and our updated pipeline. In these visualizations, purple points indicate low error values, while yellow points indicate high error values.

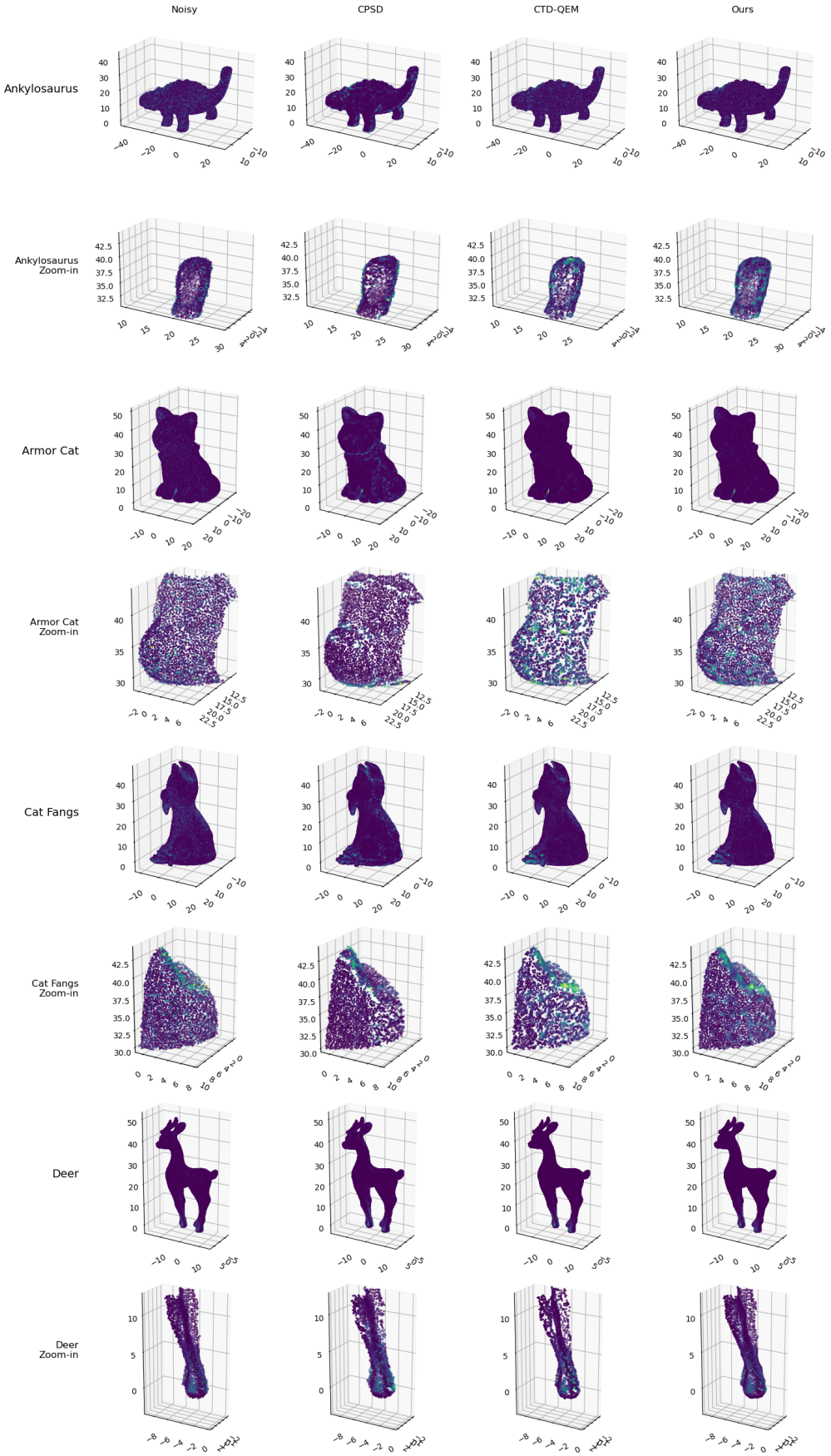

Figure C.1: From left to right the noisy model and denoised results are shown. Vertically, results for the models Ankylosaurus, Armor Cat Fangs and Deer are shown.

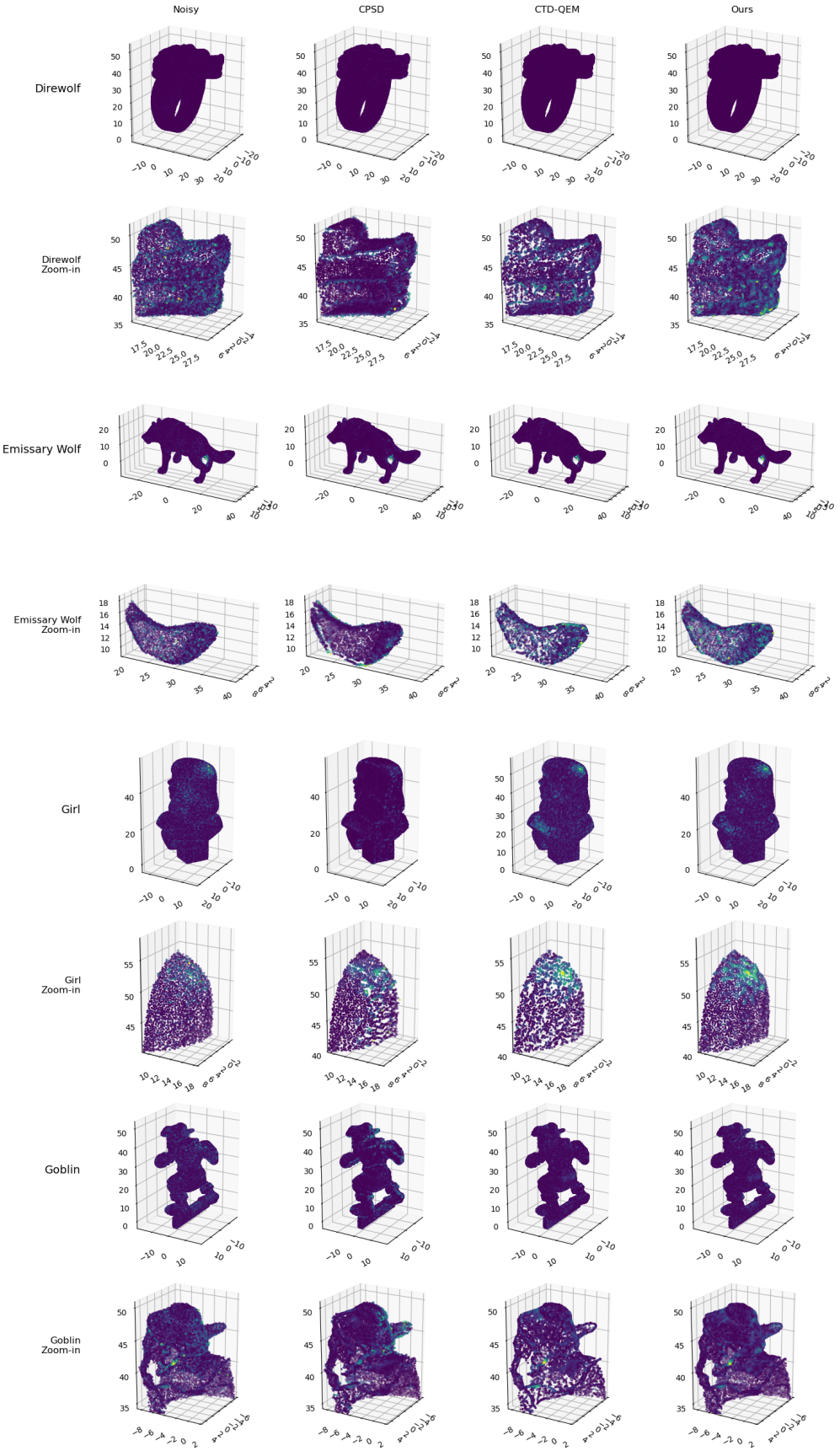

Figure C.2: From left to right the noisy model and denoised results are shown. Vertically, results for the models Direwolf, Emissary Wolf, Girl and Goblin are shown.

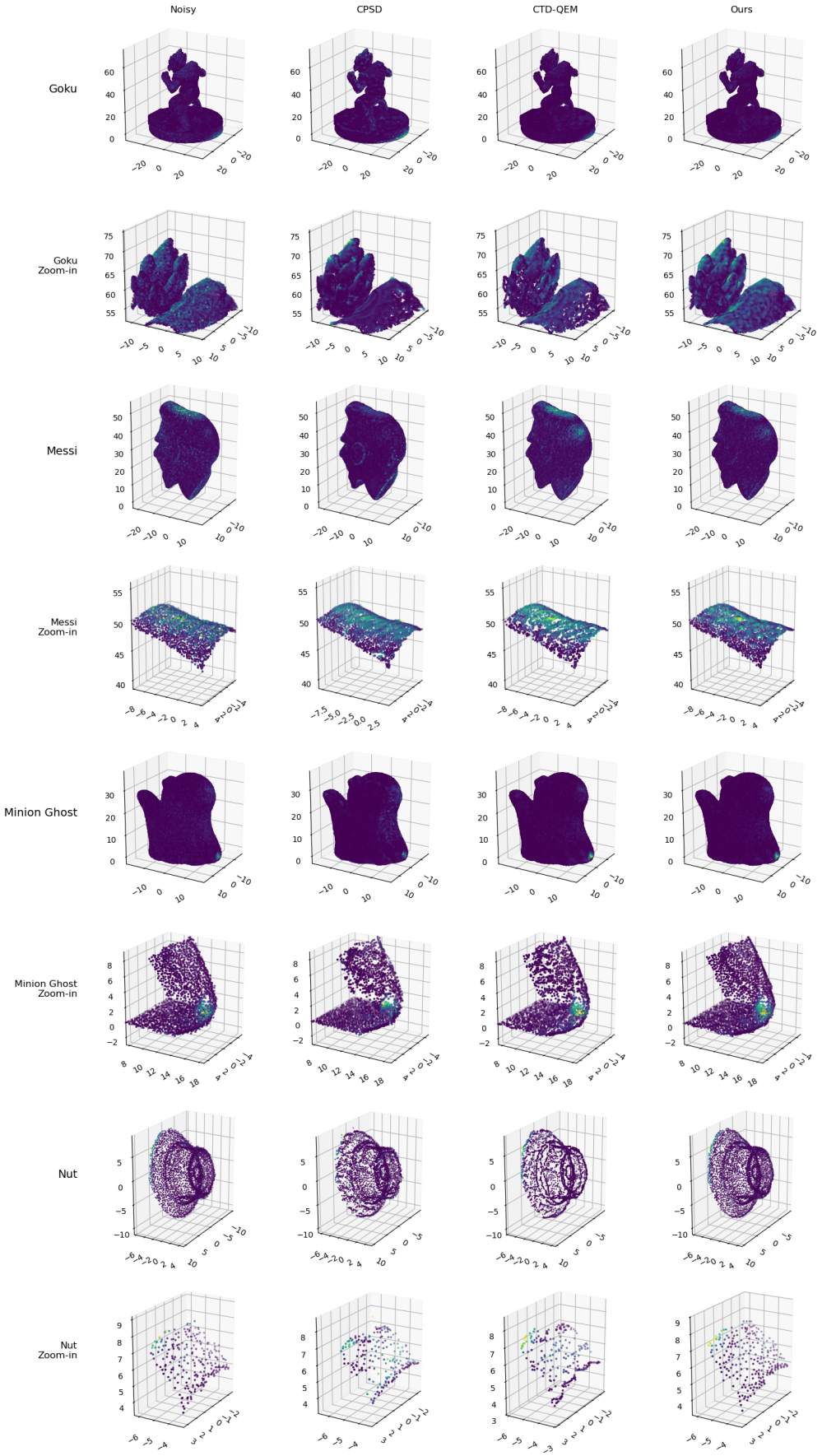

Figure C.3: From left to right the noisy model and denoised results are shown. Vertically, results for the models Goku, Messi, Minion Ghost and Nut are shown.

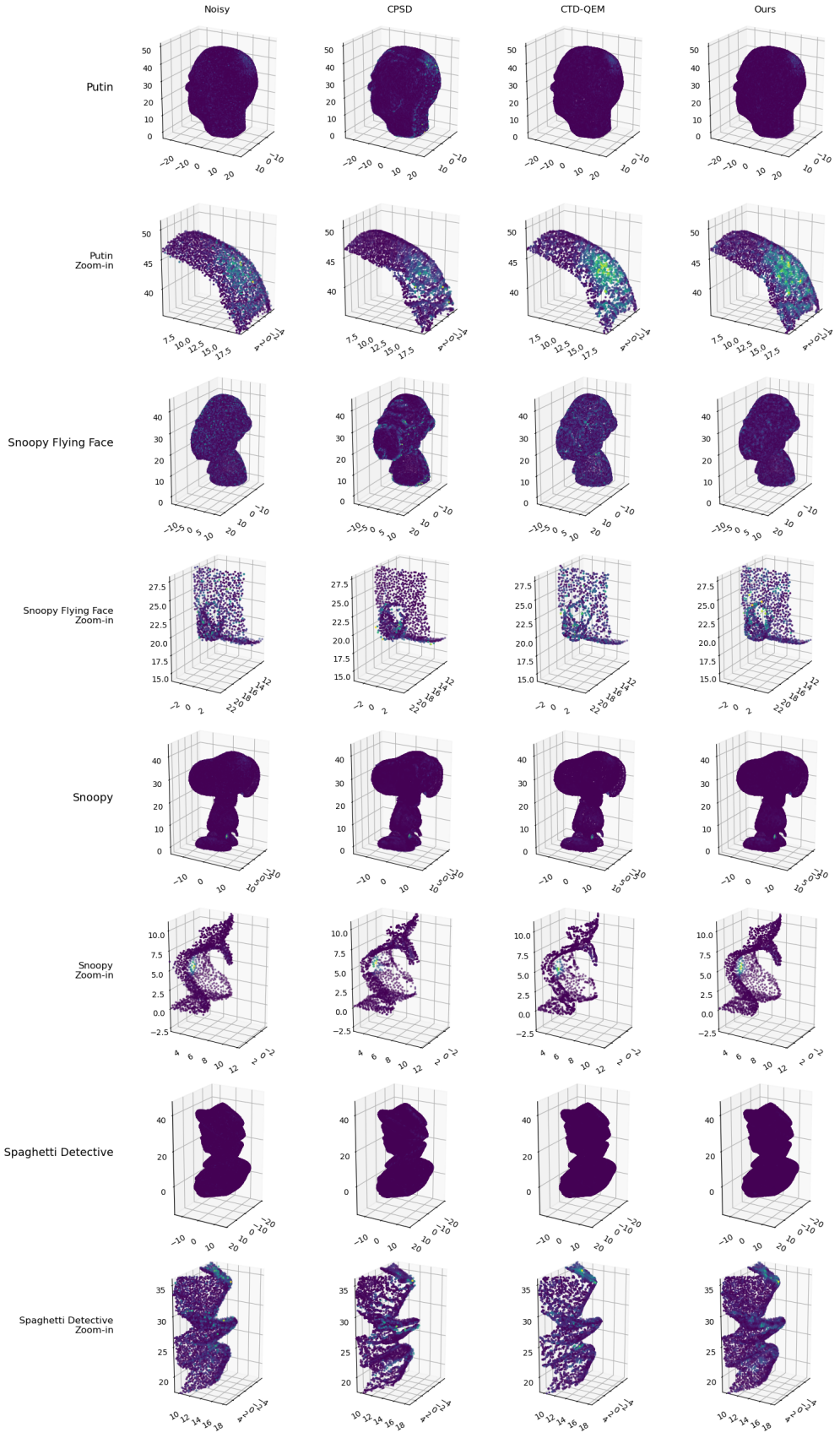

Figure C.4: From left to right the noisy model and denoised results are shown. Vertically, results for the models Putin, Snoopy Flying Face, Snoopy and Spaghetti Detective are shown.

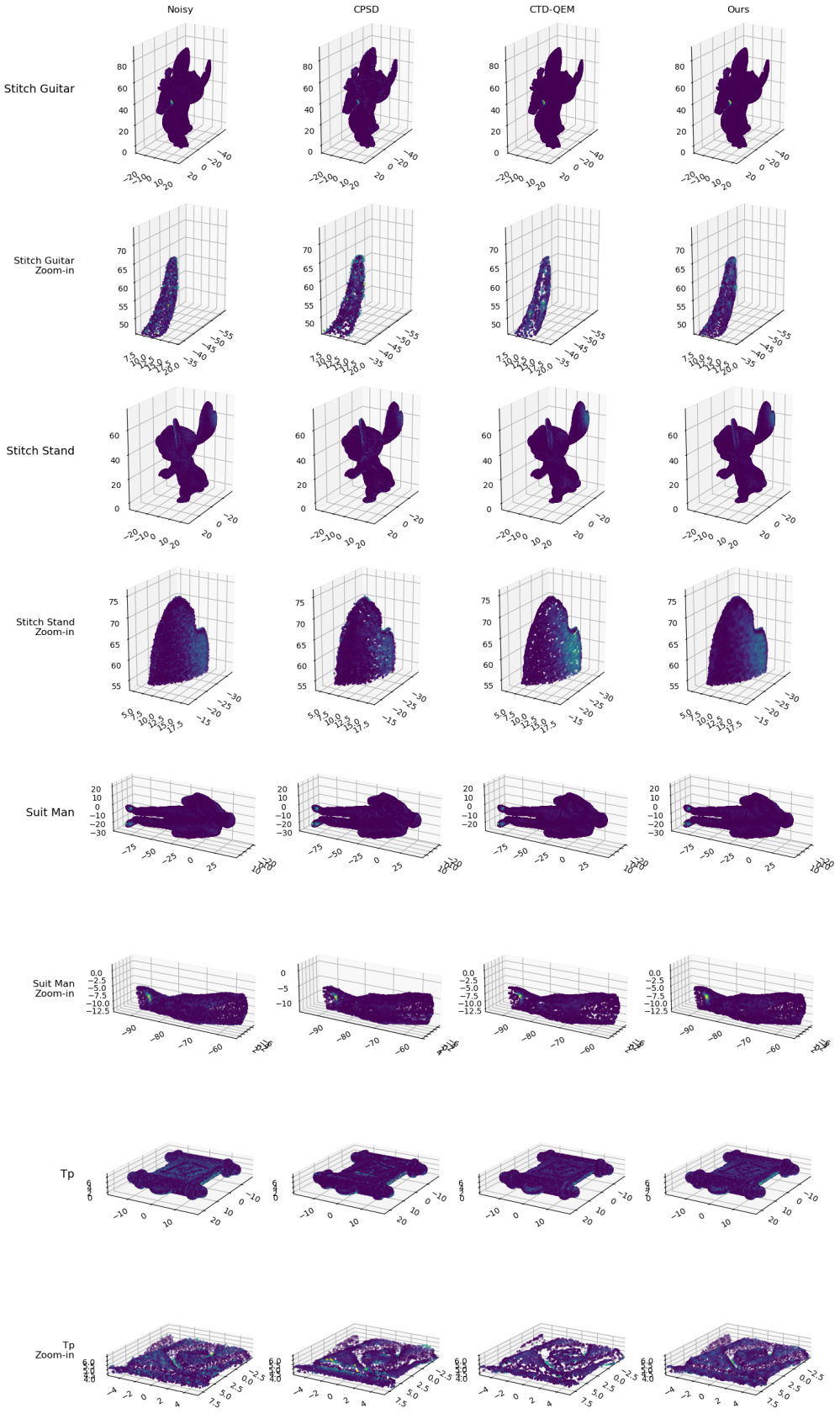

Figure C.5: From left to right the noisy model and denoised results are shown. Vertically, results for the models Stitch Guitar, Stitch Stand, Suit Man and Tp are shown.