Graphical Models 141 (2025) 101296

Contents lists available at ScienceDirect

Graphical Models

journal homepage: www.elsevier.com/locate/gmod

Review Article

Feature line extraction based on winding number

Shuxian Cai [a], Juan Cao [b], Bailin Deng [c], Zhonggui Chen [a,*]

[a] School of Informatics, Xiamen University, Xiamen, 361005, China
[b] School of Mathematical Sciences, Xiamen University, Xiamen, 361005, China
[c] School of Computer Science and Informatics, Cardiff University, Cardiff, CF24 4AG, United Kingdom

ARTICLE INFO

Keywords:
Feature detection
Feature extraction
Feature lines
CAD models
Winding number

ABSTRACT

Sharp feature lines provide critical structural information in 3D models and are essential for geometric processing. However, the performance of existing algorithms for extracting feature lines from point clouds remains sensitive to the quality of the input data. This paper introduces an algorithm specifically designed to extract feature lines from 3D point clouds. The algorithm calculates the winding number for each point and uses variations in this number within edge regions to identify feature points. These feature points are then mapped onto a cuboid structure to obtain key feature points and capture neighboring relationships. Finally, feature lines are fitted based on the connectivity of key feature points. Extensive experiments demonstrate that this algorithm not only accurately detects feature points on potential sharp edges, but also outperforms existing methods in extracting subtle feature lines and handling complex point clouds.

Contents

1. Introduction

Advances in 3D scanning technology have greatly facilitated the digitization and reconstruction of the real world, impacting fields such as 3D modeling, aerospace, and industrial design. Point clouds, which serve as the raw output from 3D scanners, are a primary data type used in graphics and visual tasks. However, issues like instrument accuracy,

* Corresponding author.
E-mail addresses: caishuxian0202@163.com (S. Cai), juancao@xmu.edu.cn (J. Cao), DengB3@cardiff.ac.uk (B. Deng), chenzhonggui@xmu.edu.cn (Z. Chen).

https://doi.org/10.1016/j.gmod.2025.101296
Received 26 June 2025; Received in revised form 26 July 2025; Accepted 29 July 2025
Available online 13 August 2025

lighting variations, occlusions, and human error can result in noisy and uneven point cloud data. Consequently, extracting sharp features from imperfect point clouds remains a challenging task in geometry processing.

Sharp edges are key geometric features in 3D models that can abstractly represent complex shapes and aid in tasks such as surface reconstruction, shape classification, and normal vector estimation. Current methods for extracting these edges typically involve two main steps: detecting feature points and extracting feature lines. The process begins by identifying feature points located at sharp edges or corners. These points are then connected using a predefined topology, and the connections are subsequently converted into parameterized curves via spline fitting to effectively capture the model's sharp geometric details.

Recent methodologies use deep neural networks to detect edge points, followed by grouping, inferring line endpoints, and curve fitting [1–3]. However, these approaches require high-quality, noise-free point cloud data and may still produce unsatisfactory results even after denoising. In contrast, traditional methods often show greater robustness and adaptability to varying point cloud qualities. Nevertheless, accurate feature curve fitting depends on precise feature detection, and many previous methods have struggled with detecting distinct feature points, leading to redundant points that affect the accuracy of the final fitting.

We observe that in previous detection methods [4–6], an improper neighborhood search radius is a primary reason for inaccurate feature point detection. Due to the varying shape complexities of point clouds, employing a uniform search radius often overlooks sharp features. To address this, we propose filtering feature points based on winding numbers. Winding numbers play a crucial role in tasks such as normal estimation [7], point cloud denoising [8], and edge detection [9]. Points on sharp edges have distinct winding numbers compared to those on planar surfaces, enabling consistent detection of accurate feature points in complex models.

Building on this foundation, this paper introduces FlexLine, a novel framework for extracting feature lines from 3D point clouds. It consists of two main components: feature point detection and feature line extraction. Initially, the winding number of the input point cloud is computed. By applying a threshold, points exhibiting significant differences in winding number are identified as detected feature points. These feature points are then partitioned into cuboids to infer adjacency information. Next, key feature points are selected from the feature points within each cuboid. Finally, based on the adjacency information, the key feature points are interconnected to extract piecewise linear curves, which are then fitted with B-splines to generate feature lines.

In summary, our contributions are as follows:

- We propose a method for detecting feature points using winding numbers, which enhances the accuracy of feature detection in complex models and supports subsequent steps.
- We design a cuboid structure to accurately capture neighboring point information and introduce an adaptive metric for point cloud quality, enabling precise topological connections and feature line extraction.
- We introduce a multi-step framework for extracting feature lines from 3D point clouds, which remains robust to varying point cloud qualities and accurately extracts sharp geometric features.

The remainder of this paper is structured as follows: Section 2 reviews relevant research on feature point detection and feature line extraction. Section 3 introduces our framework in detail. Section 4 presents experimental configurations and results. Finally, Section 5 concludes with a summary, limitations, and future prospects.

2. Related work

Feature point detection. Feature points are defined as points located on potential sharp edges within point clouds, and they are essential for tasks such as point cloud processing, feature enhancement, and preservation-based reconstruction. Traditional approaches typically begin by computing the neighborhood of each point in the point cloud and then identify sharp feature points based on residuals from polynomial fitting within these neighborhoods [10,11]. Weber et al. used Gaussian graph clustering within local neighborhoods to filter out points that are unlikely to be sharp features [12]. Mérigot et al. and Bazazian et al. defined covariance matrices based on Voronoi cells [13] and k-nearest neighbors [14] of the point cloud, respectively, to capture sharp feature information. Hackel et al. employed a binary classifier to predict the contour score of each point, selecting the most significant ones [15]. Xia et al. detected candidate edges by analyzing the ratio of eigenvalues derived from the gradient of the point cloud [16].

Network-based methods often partition the point cloud into multiple patches and use edge-aware joint loss functions to facilitate network training [17,18]. Raina et al. utilized sharpness fields defined through Moving Least-Squares to identify sharp edges [19]. Wang et al. ranked elements in an over-complete set of edges and corner points to determine feature points [1]. PCEDNet encoded shape differential information around each point in a Scale Space Matrix (SSM), enabling the neural network to learn edge characteristics [20]. Hurtado et al. applied feature candidate selection and neighborhood clustering to assist in network-based feature prediction [21]. MSL-Net developed a deep learning approach using intrinsic neighbor shape descriptors to detect sharp features from 3D point clouds [22]. In addition, graph convolutional networks [23], multilayer perceptrons [24,25], capsule network architectures [26], and multitask neural networks [27] have also been employed to distinguish edge points from non-edge points in point clouds.

Despite these advancements, the feature points identified by above methods often lack continuity and may include wrong points. These approaches are either constrained by global parameters, which can lead to the oversight of local features, or require extensive manual parameter tuning, which is inconvenient. Our method leverages winding numbers for feature point detection, which better accounts for local details and produces clearer and more comprehensive feature points.

Feature line extraction. Feature lines are crucial for summarizing the geometric shapes of 3D models and represent a key data type in reverse engineering. Traditional methods often utilize the raw structural information from point clouds to aid in feature line extraction. For instance, Gumhold et al. analyzed point relationships to fit wedges to crease lines and angles at connections to reconstruct feature lines [28]. Pauly et al. developed a multi-scale classification operator combined with an adaptive active contour model [29]. Demarsin et al. employed first-order segmentation to extract candidate feature points, which were then processed as a graph to recover sharp feature lines [30]. Daniels et al. applied robust Moving Least-Squares to locally fit potential surface features [31]. Nie introduced a Smooth Shrink Index (SSI) and integrated it with principal component analysis (PCA) [6]. Xia et al. proposed a graph-based edge connection algorithm to link detected feature points [16]. Additionally, generating initial feature line segments and subsequently fitting complete curves based on these segments can also produce reliable results [32,33].

In contrast, deep learning-based methods handle diverse input data, such as point clouds, images, or meshes. PIE-NET employs an end-to-end learnable approach to identify feature edges in 3D point cloud data and derive a set of parametric curves [1]. DEF calculates a distance scalar field from a point to the feature line and extends these distance features to large-scale point clouds [34]. NEF uses rendering-based differentiable optimization and iterative methods to extract final parametric 3D curves [2]. NerVE first converts point clouds into voxelized models and then extracts piecewise linear curves containing

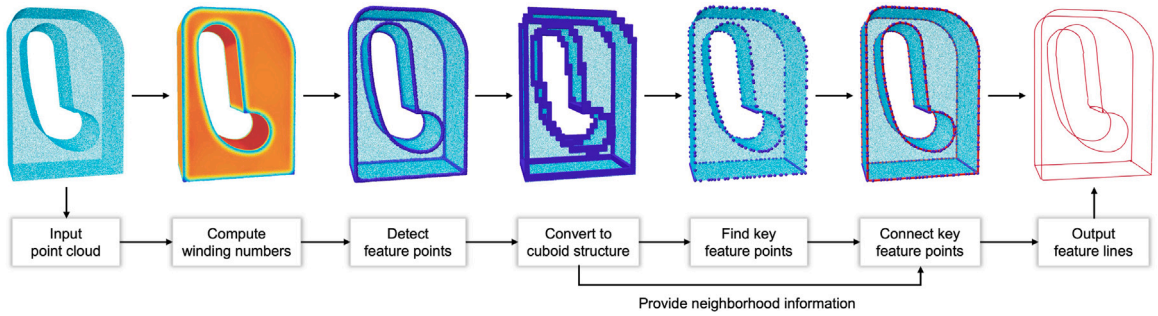

Fig. 1. Pipeline of the proposed method. Given an input point cloud, first compute the winding number for each point. Feature points are then detected by examining the differences in winding numbers on potential planes and edges, with these points marked in purple. To accurately determine the adjacency between points, the feature points are converted into a structure composed of individual cuboids, and key feature points are identified within each cuboid. By analyzing the connectivity among these cuboids, the relationships between key feature points are inferred, allowing the connection of key feature points to form curves. Finally, these curves are fitted to generate the final feature lines.

feature points [3]. SepicNet introduces an adaptive point cloud sampling technique based on curve fitting to more effectively capture sharp features [35]. Even with these improvements, many methods still face challenges. They may either overlook some feature regions, produce discontinuous or non-smooth curves, or rely heavily on high-quality input point clouds. Our framework, which is based on precise feature detection, is more robust to low-quality point clouds, ensuring accurate results and producing continuous and smooth feature lines.

3. Method

Feature lines capture the key geometric features of a 3D model, and their accuracy depends heavily on the quality of feature detection. Classical methods [5,6,13,15,36] typically detect feature points based on normals or curvature. These methods rely on local differential properties of the point cloud, identifying regions with sharp variations in normals or extreme values of principal curvature as feature points. Due to their strong focus on local geometric information, such methods are often sensitive to noise and variations in point cloud density, which can degrade accuracy. In contrast, the winding number approach emphasizes global geometric variation and is more aligned with topological characteristics. It evaluates the extent to which a point is enclosed by its potential surface neighborhood, capturing the "enclosure relationship". At sharp edges or in regions of high concavity or convexity, the degree of enclosure changes dramatically, leading to abrupt shifts in the winding number, which are used to identify feature points. Therefore, compared to normal- or curvature-based methods, the winding number method generally performs better in the presence of noise, sparsity, or non-uniformly distributed data.

For the subsequent extraction of feature lines, accurately determining the connectivity between feature points is crucial. Common approaches involve constructing a k-nearest neighbors (KNN) tree [4] or a minimum spanning tree [6] to traverse the feature point set and identify the nearest neighbors for connection. However, these methods typically requires setting thresholds for distance or quantity and eliminating redundant nearest points in the same direction. Unlike the aforementioned methods, our approach transforms the point cloud into a cubical structure to infer point connectivity. This structure does not require the cube edges to align with feature points; rather, it partitions points into multiple groups based on their spatial positions, where each group corresponds to a cube in the cubical structure. Key feature points are selected from each group, and the adjacency relationships among cubes are leveraged to assist in establishing the neighborhood relationships between the key feature points. Fig. 1 illustrates the overall framework, and the following sections provide detailed explanations of each step.

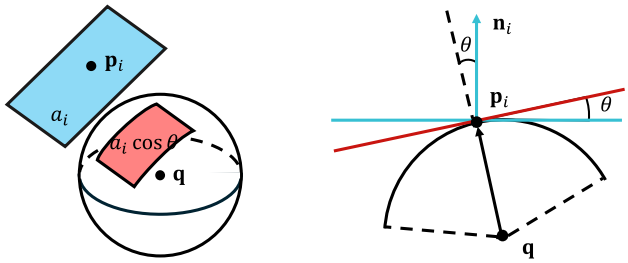

Fig. 2. Illustration of the winding number computation. (a) Projection of the control region of a point in three-dimensional space onto a sphere centered at the query point. (b) relationship between θ, the potential plane, and the normal from a two-dimensional perspective. The potential surfaces of the point cloud are highlighted in blue, while the sphere's surface centered at the query point and its cross-section are shown in red.

3.1. Feature point detection

In mathematics, the winding number of a closed curve around a point on a plane is an integer representing the total number of times the curve wraps around that point. For certain open curves, this number can be a non-integer. The winding number depends on the curve's direction, with counterclockwise wrapping yielding a positive number. It can be extended to polygonal meshes [9], triangular soups, and point clouds [37].

Given a set of points $\mathcal{P} = \{\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3, \dots, \mathbf{p}_N\}$ on the potential surface of the model, where each point $\mathbf{p}_i$ has a normal $\mathbf{n}_i$, the winding number w at a query point $\mathbf{q}$ can be computed as the signed surface area sum of the projections of $\mathcal{P}$ onto a sphere centered at $\mathbf{q}$ (see Fig. 2(a)). This effectively calculates the total signed surface area of the point cloud wrapped around $\mathbf{q}$:

$$\mathrm{w}(\mathbf{q}) = \sum_{i=1}^{N} \frac{a_i \cos\theta_i}{4\pi \|(\mathbf{p}_i - \mathbf{q})\|^2} \tag{1}$$

where a_i is the dominating area of point $\mathbf{p}_i$, and θ_i is the angle between the potential surface at $\mathbf{p}_i$ and the tangent plane of the sphere centered at the query point $\mathbf{q}$ (see Fig. 2(b)).

For computing a_i, Barill et al. [37] and Xu et al. [7] used KNN trees and Voronoi diagrams, respectively. However, these methods either struggle with non-uniform point distributions or are computationally intensive due to the calculation of Voronoi cell cross-sectional areas. We observe that a_i essentially represents the weight of point $\mathbf{p}_i$ in the point cloud. Therefore, a_i can be approximated by the area of a circle centered at $\mathbf{p}_i$, with a radius of half the distance from $\mathbf{p}_i$ to the nearest point. This modification not only makes the method applicable

to non-uniform point clouds but also significantly reduces computation time.

Since θ_i cannot be directly measured, it can be computed using the vector $\mathbf{qp}_i$ and the unit normal vector $\mathbf{n}_i$. θ_i is the angle between these two vectors (see Fig. 2(b)). Deriving this, we obtain the generalized formula for computing the winding number in point clouds:

$$\begin{aligned} \mathrm{w}(\mathbf{q}) &= \sum_{i=1}^{N} \frac{a_i}{4\pi\|\mathbf{p}_i-\mathbf{q}\|^2} \cdot \frac{(\mathbf{p}_i-\mathbf{q})\cdot\mathbf{n}_i}{\|\mathbf{p}_i-\mathbf{q}\|\cdot\|\mathbf{n}_i\|} \\ &= \sum_{i=1}^{N} a_i \frac{(\mathbf{p}_i-\mathbf{q})\cdot\mathbf{n}_i}{4\pi\|\mathbf{p}_i-\mathbf{q}\|^3} \end{aligned} \tag{2}$$

Notably, as $\mathbf{q}$ approaches $\mathbf{p}_i$, $\|\mathbf{p}_i-\mathbf{q}\|$ approaches 0, causing $\mathrm{w}(\mathbf{q})$ to become singular. To address this issue, we adopt the approach outlined in Parametric Gauss Reconstruction (PGR) [38] and modify $\mathrm{w}(\mathbf{q})$ as:

$$\tilde{\mathrm{w}}(\mathbf{q}) = \begin{cases} \mathrm{w}(\mathbf{q}), & \|\mathbf{p}_i-\mathbf{q}\| \geq \mathrm{d}(\mathbf{q}). \\ \sum_{i=1}^{N} a_i \frac{(\mathbf{p}_i-\mathbf{q})\cdot\mathbf{n}_i}{4\pi\mathrm{d}(\mathbf{q})^3}, & \|\mathbf{p}_i-\mathbf{q}\| < \mathrm{d}(\mathbf{q}). \end{cases} \tag{3}$$

where $\mathrm{d}(\mathbf{q})$ is a positive function that specifies the modification radius, with its detailed computation provided in PGR. Thus, the winding number can be used to infer the position of a point in the point cloud. If point $\mathbf{q}$ is inside the point cloud, the winding number approaches 1; if outside, it approaches 0.

For a query point on the potential surface of the point cloud, the winding number tends to approach 0.5 [37,38]. Further computational analysis reveals that the winding number varies according to different local structures within the point cloud. This pattern can be leveraged for feature point detection. Fig. 3 illustrates the variation of winding numbers across different regions in a two-dimensional setting. Since the winding number quantifies the contribution area of the model projected onto a circle centered at the query point, its value increases as the model's contribution to the circle grows. When the point lies on a convex corner, the model's contribution is minimal, leading the winding number to approach 0. Conversely, when the point is located on a concave corner, the model's contribution is more significant, causing the winding number to approach 1.

Fig. 4 extends this analysis from two-dimensional models to three-dimensional point clouds, illustrating the distribution pattern of winding numbers in 3D space. Given that a two-dimensional plane can be considered a cross-section of a three-dimensional model, the winding number values in 3D can be broadly classified into three categories. The first type occurs on convex edges, where the point cloud contributes only a small portion to the sphere centered at the points in this region, resulting in a winding number less than 0.5 and trending toward 0. The second type is found in planar regions, where the winding number approaches 0.5. The third type is located on concave edges, where the point cloud contributes significantly to the sphere centered at points in this region, leading to a winding number greater than 0.5 and trending toward 1. Based on this pattern, by applying thresholds a and b (typically set to 0.4 and 0.8, respectively), points with winding numbers below a or above b are filtered out and classified as detected feature points.

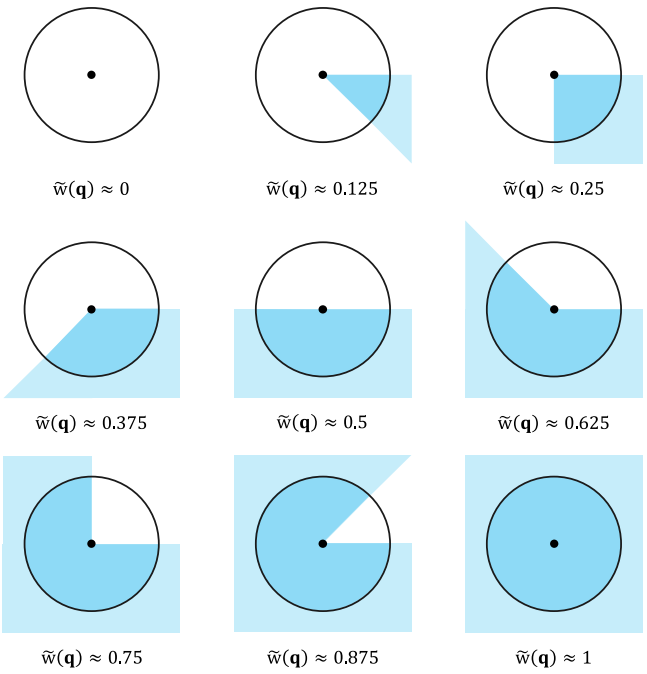

Fig. 3. Illustration of winding number value types in the two-dimensional case. The winding number value depends on the model's contribution to the circle centered at the query point. As the model's contribution increases, the winding number value also grows accordingly.

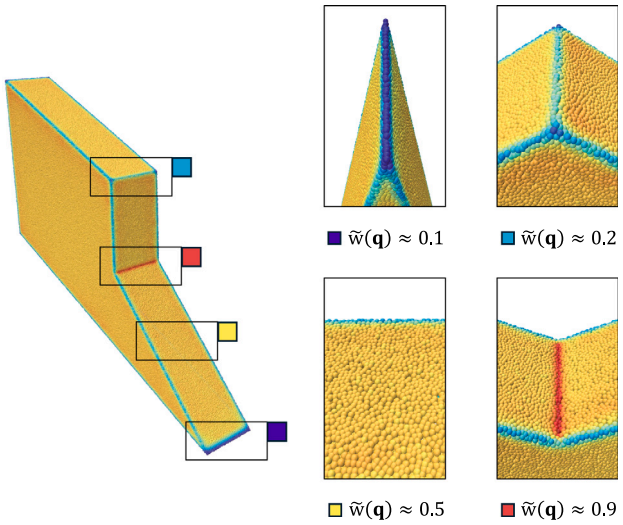

Fig. 4. Illustration of winding number value types for a point cloud. The colors correspond to different feature regions, with each region having a distinct color. Specifically, blue points are located on convex edges, yellow points are on potential planar surfaces, and red points are on concave edges. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

3.2. Feature line extraction

To extract and fit feature lines from the detected feature points, we employ a cuboidal structure to define the adjacency relationships among them, as illustrated in Fig. 5. First, we identify the maximum and minimum values of the feature point set along the three coordinate axes (x, y, and z) to construct the largest encompassing cuboid. Next, based on a predefined resolution, this cuboid is subdivided into multiple smaller cuboids of equal size. Finally, each feature point is assigned to its corresponding cuboid according to its spatial location, and any cuboids that do not contain any feature points are removed.

After constructing the cuboids, we can determine their adjacency relationships, which in turn allows us to infer the connectivity between feature points. Fig. 6 illustrates different adjacency scenarios among cuboids. If two cuboids share a face, they are considered face-adjacent; if they share an edge, they are classified as edge-adjacent. Both cases indicate that the two cuboids are adjacent. Since each cuboid contains at least one feature point, the adjacency of two cuboids implies that the feature points they contain are also adjacent.

If feature points are connected directly based on adjacency relationships, ambiguity may arise because a single cuboid can contain multiple feature points. To address this, key feature points are selected from each cuboid to serve as the connected points. However, since the input point

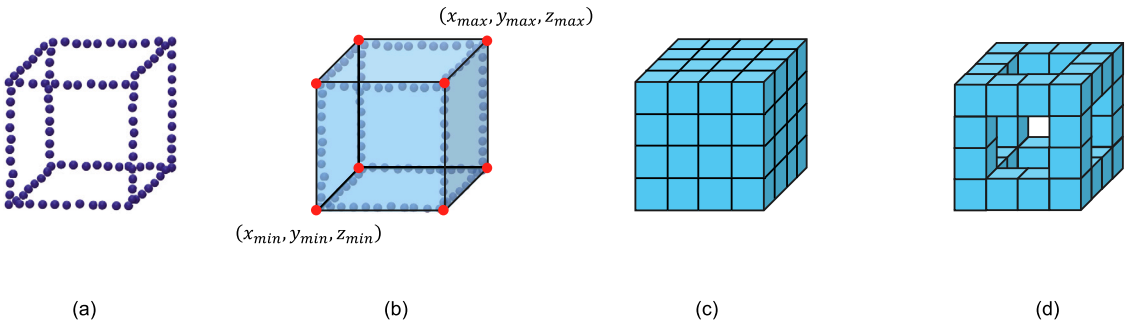

Fig. 5. Process of constructing cuboids: (a) detecting the initial feature points; (b) constructing a large cuboid based on the distribution of the initial feature points and obtaining eight vertices; (c) subdividing the large cuboid into multiple smaller cuboids by partitioning the distances between the eight vertices according to the specified resolution; (d) retaining the cuboids that contain feature points to form a set of cuboids.

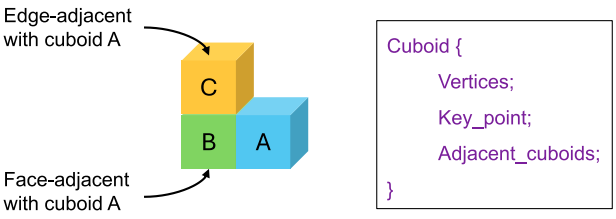

Fig. 6. Different types of adjacent cuboids and the structure of the cuboids.

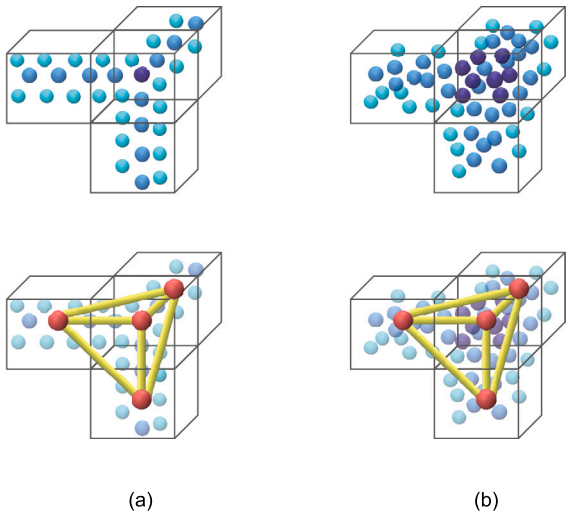

Fig. 7. Rules of point connections. Blue points represent the input (a) noise-free point cloud and (b) noisy point cloud, where darker colors indicate smaller winding numbers. Red points denote key feature points, while yellow line segments represent the connections between points. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

cloud may be either noise-free or noisy, a single selection rule is not suitable for all cases. To ensure robustness, different key feature point selection strategies are employed for noise-free and noisy point clouds, as shown in Fig. 7.

But manual assessment of whether a point cloud contains noise can be inconvenient and subjective to some extent. Based on this, we propose to classify point clouds using a local plane fitting residual metric. For each point $\mathbf{p}_i$ in the point cloud, we consider its k-nearest neighborhood $\mathcal{N}_i$ and fit a local plane using Principal Component Analysis (PCA) [39]. The fitting residual is computed as:

$$r_i = |\mathbf{n}_i \cdot (\mathbf{p}_i - \mathbf{p}_{\mathcal{N}_i})|, \tag{4}$$

where $\mathbf{p}_{\mathcal{N}_i}$ denotes the centroid of the neighborhood $\mathcal{N}_i$, and $\mathbf{n}_i$ is the direction corresponding to the smallest eigenvalue from PCA. The residual r_i represents the perpendicular distance from point $\mathbf{p}_i$ to the fitted plane. After computing the residuals for all points, the mean residual $\hat{r}$ of the entire point cloud is obtained. If $\hat{r}$ exceeds 0.002, the point cloud is classified as noisy. Therefore, the local plane fitting residual metric can be effectively used to determine whether a point cloud contains noise.

In the case of a noise-free point cloud, we observe that within cuboids containing potential corner positions, the feature point closest to the corner location tends to have a winding number that is an extremum among all feature points in the cuboid. Specifically, if the corner is concave and all winding numbers in the cuboid exceed 0.5, the extremum is the maximum value. Conversely, if the corner is convex and all winding numbers are below 0.5, the extremum is the minimum value. Based on this observation, the feature point with the extremum winding number within each cuboid is directly selected as the key feature point. This selection method is not limited to potential corner regions but can also be applied to potential feature edge regions. Since the points do not contain any extra points outside the potential plane, the variation in winding numbers along an edge region depends on their proximity to the feature edge. As a result, points closer to the feature edge exhibit relatively small differences in winding numbers, making the extremum-based selection method effective in identifying key feature points for edge regions as well.

When dealing with noisy point clouds, the irregular distribution of points introduces significant challenges for feature curve extraction. Noisy points often appear near potential corners or feature edges, as presented in Fig. 7(b), further complicating the selection process. To mitigate the impact of noise and ensure that the chosen key feature points better reflect the denoised structure, we employ an averaging approach. This involves computing the average coordinates of all points within a cuboid and using this averaged position as the key feature point. Since this calculated point represents the geometric center of all feature points in the cuboid, it is more likely to be located near or within the feature region, reducing the influence of noise.

Once the key feature points have been selected, we utilize the previously established cuboid adjacency relationships to systematically connect the corresponding key feature points in each cuboid. This process ultimately results in the construction of a piecewise linear (PWL) curve that effectively represents the underlying structure of the point cloud.

Since adjacent cuboids may only share an edge, the intersection points of multiple edges can create triangular loops, as shown in Fig. 8. To address this, we have implemented a post-processing step to remove these loops. We first construct an adjacency list that stores information about the endpoints of all edges involving each point. Then, using depth-first search, we traverse edges to identify loops with a length of 3 steps. Finally, we calculate the length of each edge in the loop and remove the longest edge to obtain a cleaner PWL set. Fig. 8 illustrates the main workflow of this post-processing step, which improves result accuracy. After removing loops, we apply Zhu et al.'s method [3] to extract final parametric feature lines from the PWL set.

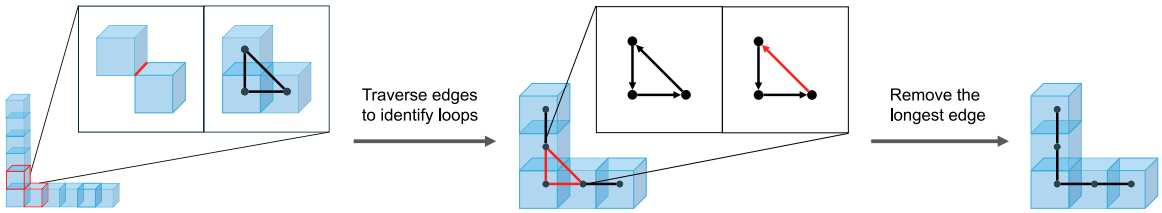

Fig. 8. Procedure of post-processing. Due to the presence of three adjacent cuboids that are each neighboring one another, triangular loops can occur, leading to incorrect point connections. To resolve this issue, all edges are traversed to identify loops, and the longest edge within each loop is removed to ensure cleaner connections.

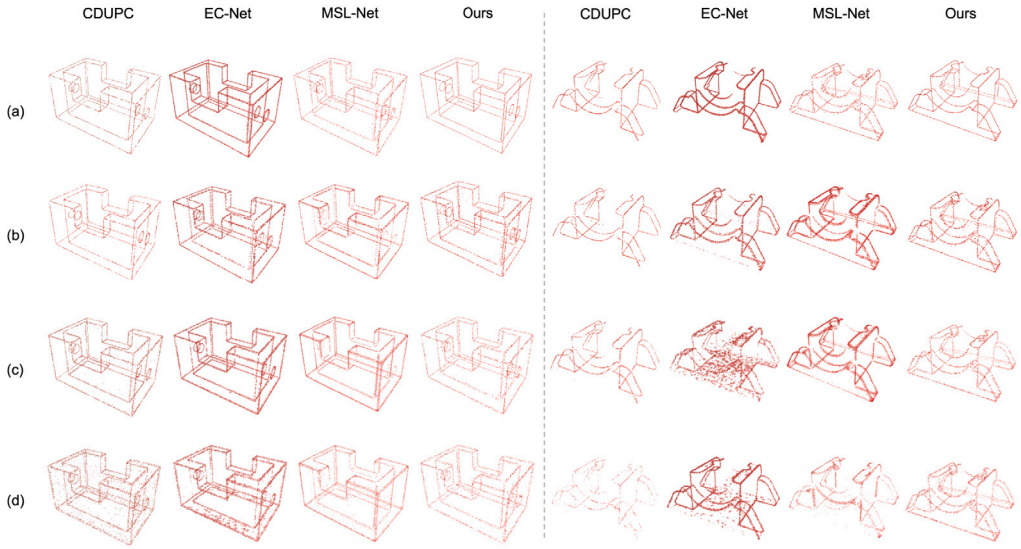

Fig. 9. Comparison of results from different feature detection methods with various input point clouds. (a) Uniform, noise $\sigma = 0.0\%$; (b) non-uniform, noise $\sigma = 0.0\%$; (c) uniform, noise $\sigma = 0.6\%$; (d) uniform, noise $\sigma = 1.2\%$.

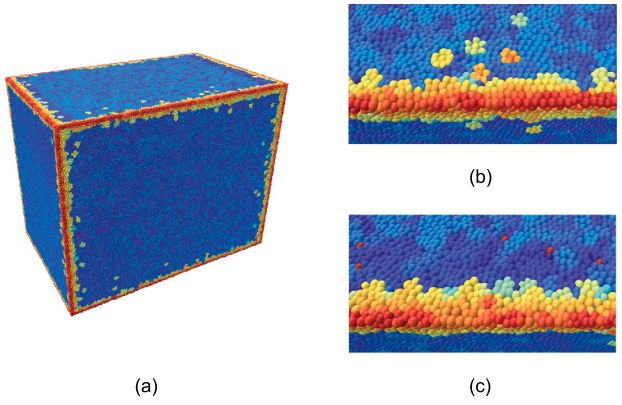

Fig. 10. Result of curvature computation. (a) Visualization of input point cloud. (b) Local details of the detected non-feature points (in orange). (c) Local details of the erroneously computed non-numerical points (in red). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

4. Experiments

4.1. Experiment setup

We implemented our framework in C++ on a computer equipped with an Intel Core i7-12700F CPU at 2.1 GHz and 16 GB of RAM. Deep learning experiments were performed using an NVIDIA RTX 3090 Ti GPU. All experimental data were obtained from the uniform, non-uniform, and noisy point clouds generated by Huang et al. [40]. All data were normalized to the range [0,1]. These point clouds were sampled from the ABC dataset [41], which consists of various CAD models.

To assess the accuracy of the detected feature points and the extracted feature lines, we randomly selected 100 models from the ABC dataset. We then evaluated the proximity of our results to the ground truth using Chamfer Distance (CD) [42] and Hausdorff Distance (HD) [43].

4.2. Comparisons

Our framework consists of two main parts: feature detection and feature extraction. The former identifies the initial feature points within the point cloud, while the latter connects these initial feature points into lines and performs fitting.

4.2.1. Feature point detection

The criteria for effective feature detection involve identifying feature points that clearly and continuously represent the sharp features of a point cloud while preserving local details. We assessed our method's effectiveness by comparing it with CDUPC [15], EC-Net [17] and MSL-Net [22] across three types of input data: uniform, noisy, and non-uniform.

Fig. 9 presents a comparative analysis of feature detection results from different input point clouds. As the quality of the point cloud deteriorates, CDUPC tends to detect excessive redundant points. EC-Net employs an edge-aware technique for feature point detection and performs well on both uniform and non-uniform point clouds. However, in the presence of noise, it often detects many irrelevant points. Meanwhile, MSL-Net, due to its use of non-adaptive parameters, successfully

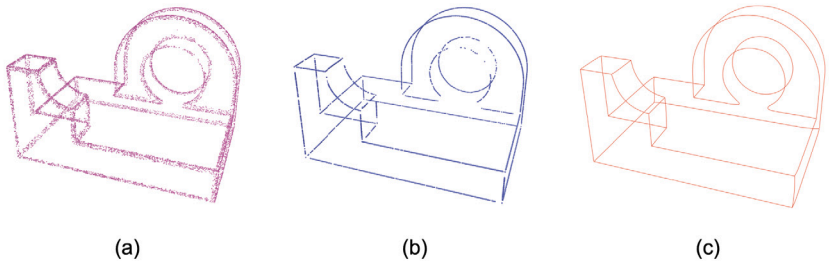

Fig. 11. Feature enhancement and extraction results of RFEPS [4] and our method, respectively. (a) Feature regions detected by RFEPS; (b) feature points enhanced by RFEPS; (c) feature lines extracted using our method.

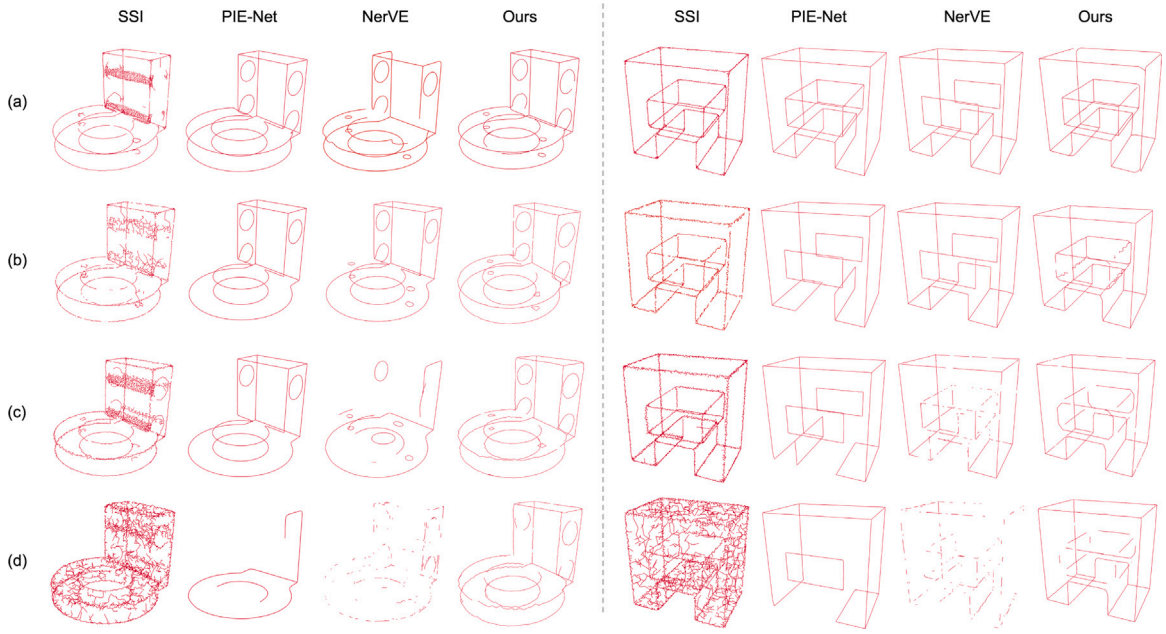

Fig. 12. Comparison of results from different feature extraction methods with various input point clouds. (a) Uniform, noise $\sigma = 0.0\%$; (b) non-uniform, noise $\sigma = 0.0\%$; (c) uniform, noise $\sigma = 0.6\%$; (d) uniform, noise $\sigma = 1.2\%$.

identifies major features but struggles to capture local sharp features. In contrast, our method leverages the winding number, effectively overcoming point cloud imperfections while accurately detecting complete sharp features and minimizing the presence of erroneous points.

Moreover, Chen [36] proposed utilizing the relationship between total curvature and Dirichlet energy for feature perception. As shown in Fig. 10, while this method can detect sharp features, some points near the feature regions may be mistakenly identified as feature points (e.g., orange points in Fig. 10(b)). Additionally, numerical anomalies in the detection results may affect the filtering of feature points (e.g., red points in Fig. 10(c)).

Another traditional method, RFEPS [4], demonstrates strong performance in feature enhancement. However, for feature detection, it identifies feature regions rather than specific feature points located on potential feature lines, as illustrated in Fig. 11(a). The method then shifts the points within these regions to obtain enhanced feature points, as shown in Fig. 11(b). Nevertheless, due to the use of spherical neighborhoods, points near sharp corners tend to be moved toward the corner itself. As a result, although RFEPS accurately detects feature regions, the final set of feature points remains discontinuous. Even after applying curve fitting to these points, it will still fail to produce continuous feature lines as effectively as our method.

Table 1 provides a quantitative accuracy comparison. Our method not only performs better visually but also demonstrates a significant numerical advantage over other methods.

4.2.2. Feature line extraction

Feature lines are closely related to detected feature points, and many feature line extraction methods include a feature detection step. We compared the complete feature curve extraction frameworks, SSI [6], PIE-Net [1], NerVE [3], and our algorithm using various input point clouds.

Fig. 12 shows some experimental results of comparisons. In the SSI method, using a uniform radius to compute the smooth shrink index can lead to the misclassification of two closely spaced surfaces as a feature region. Moreover, its strong emphasis on local details makes it less robust to noise. For PIE-Net, the use of curve proposal generation yields continuous feature curves, but the results remain sensitive to point cloud quality. Evidently, NerVE fails to extract certain sharp features and is notably less robust to defective point clouds compared to our method. More results of our method can be found in Fig. 13. Table 2 compares the results with the ground truth, indicating that our method's extracted feature lines deviate less from the ground truth than those from SSI, PIE-Net and NerVE.

Furthermore, testing NerVE on additional noisy data shows that it fails to produce results in approximately 44% of cases due to quality issues of the input point cloud, leading to feature extraction failure. The underlying reason is that when processing noisy point clouds, NerVE tends to generate numerous extremely short and discontinuous line segments. These segments do not meet the degree requirements for connectable segments and are consequently discarded during the

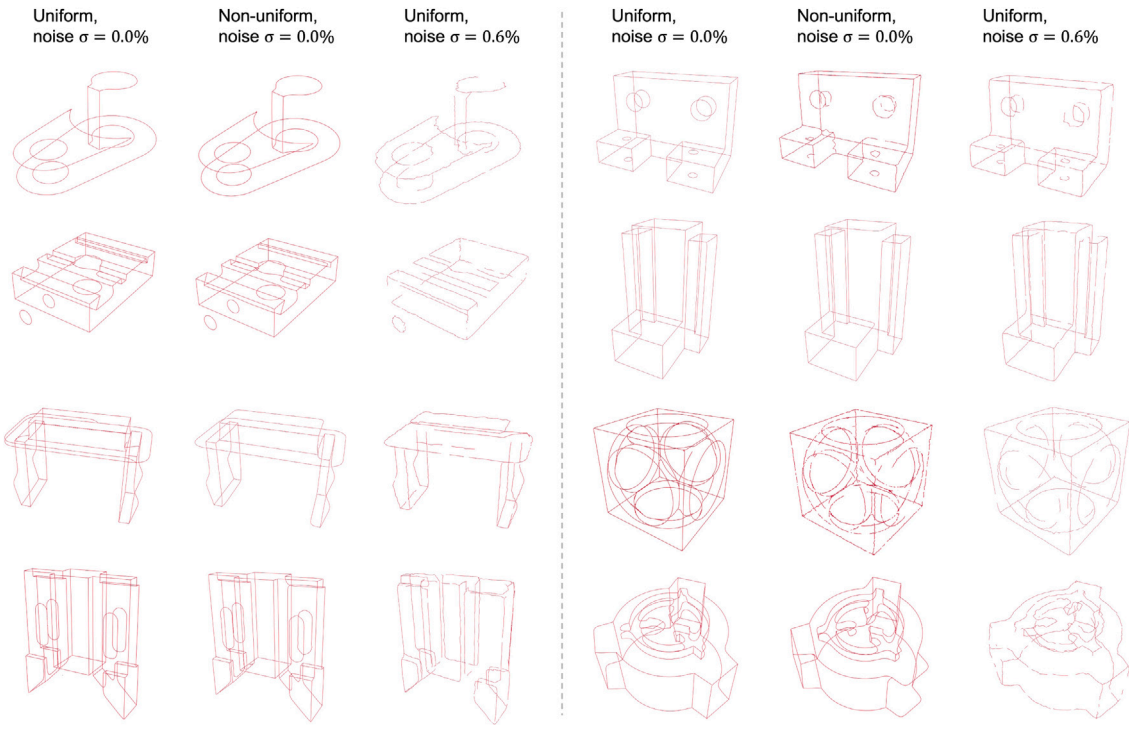

Fig. 13. More results on feature extraction using our method.

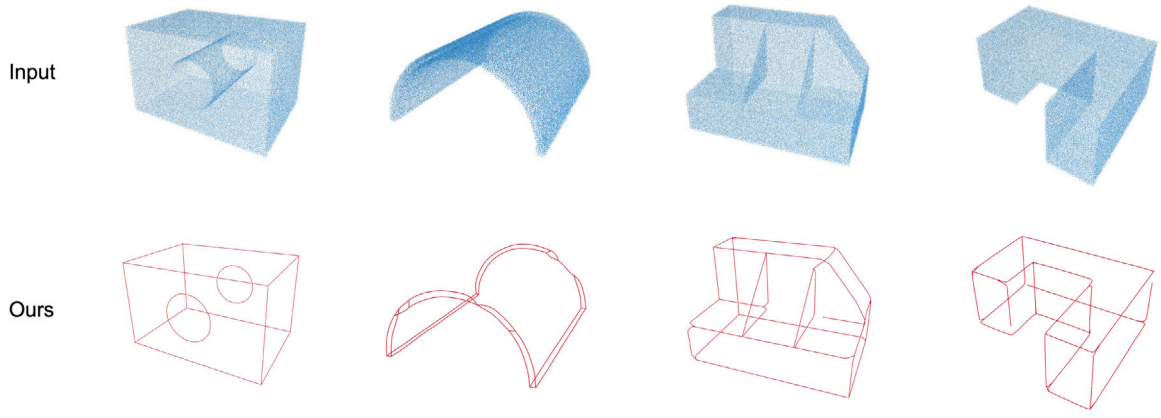

Fig. 14. Feature extraction results where NerVE fails, but our method successfully generates outcomes.

Table 1
Accuracy assessment of various feature detection methods with different types of input point clouds.

Types	Uniform, noise $\sigma = 0.0\%$				Non-uniform, noise $\sigma = 0.0\%$				Uniform, noise $\sigma = 0.6\%$				Uniform, noise $\sigma = 1.2\%$			
Methods	CDUPC	EC-Net	MSL-Net	Ours	CDUPC	EC-Net	MSL-Net	Ours	CDUPC	EC-Net	MSL-Net	Ours	CDUPC	EC-Net	MSL-Net	Ours
CD↓	0.0126	0.0102	0.0124	**0.0034**	0.1323	0.0927	0.1315	**0.0582**	0.1460	0.1486	0.1354	**0.0961**	0.3710	0.3807	0.3486	**0.1973**
HD↓	0.2519	0.1053	0.2647	**0.0398**	0.3375	0.2194	0.3269	**0.1492**	0.3560	0.3685	0.3431	**0.1830**	0.5821	0.5947	0.5370	**0.3224**

Table 2
Accuracy assessment of various feature extraction methods with different types of input point clouds.

Types	Uniform, noise $\sigma = 0.0\%$				Non-uniform, noise $\sigma = 0.0\%$				Uniform, noise $\sigma = 0.6\%$				Uniform, noise $\sigma = 1.2\%$			
Methods	SSI	PIE-Net	NerVE	Ours	SSI	PIE-Net	NerVE	Ours	SSI	PIE-Net	NerVE	Ours	SSI	PIE-Net	NerVE	Ours
CD↓	0.1632	0.0159	0.0240	**0.0038**	0.2568	0.2105	0.1896	**0.0865**	0.3304	0.2331	0.2372	**0.0989**	2.3528	3.2458	1.8923	**0.9620**
HD↓	0.5152	0.1290	0.2858	**0.0838**	0.8495	0.7026	0.6318	**0.3950**	0.9815	0.7241	0.7123	**0.4107**	3.0966	3.2594	2.6346	**1.0384**

subsequent connection and fitting process, resulting in a failure to output feature extraction results. This highlights the instability of NerVE and its low tolerance to point cloud quality. In contrast, our method successfully extracts features even from low-quality point clouds, as demonstrated by examples in Fig. 14.

Additionally, DEF [34] is also included for comparison. Since the implementation code of DEF is not publicly available and its results are limited to uniform point cloud data, we compare it only on uniform inputs, using the experimental results provided in the official DEF repository. As shown in Fig. 15, our method significantly outperforms

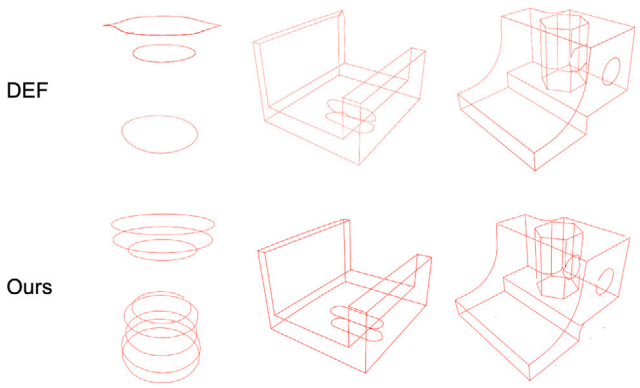

Fig. 15. Comparison of results from different feature extraction methods with uniform input point clouds.

DEF on uniform point clouds, particularly in regions with subtle sharp features or complex structures. The strength of our approach lies in its precise feature detection, which enables the identification of feature points along potential arc segments, resulting in more complete feature lines.

4.3. Influence of parameter settings

The values of key parameters significantly impact the quality of results. In feature detection, thresholds a and b are used to determine which points are marked as feature points. Fig. 16 illustrates detection results with various threshold combinations, where points with a winding number below a or above b are identified as feature points. It can be observed that when a is too small, feature points on outward protruding edges become discontinuous and sparse, whereas a larger a results in an excess of points. Similarly, if b is too small, inward concave edges yield many irrelevant points, whereas a larger b leads to sparse feature points. Inappropriate values for a or b may also introduce points from planar regions. Therefore, we set a and b to default values of 0.4 and 0.8, respectively, to ensure higher quality feature points.

In feature extraction, resolution determines the density of cubes, which in turn affects the connectivity between points. Fig. 17 provides the results of feature line extraction at different resolutions. When the resolution is too low, the number of cuboids is limited, causing many unrelated feature points to be connected. However, when the resolution is too high, the excessive number of cuboids may lead to the generation of numerous redundant fine structures. Therefore, we set the default resolution to 32 to ensure the extraction of more complete and accurate feature curves.

4.4. Influence of normal estimation methods

Our framework requires input point clouds to have oriented normals that are consistent and roughly accurate. While many public datasets include normals, some datasets or scanned data require users to estimate them. Different estimation methods may produce slightly different results. To validate the robustness of our method, we used both Principal Component Analysis (PCA) [39] and Quadric Surface Fitting [44] for normal estimation. Fig. 18 shows that these methods have minimal impact on feature detection performance. Therefore, users do not need a specific estimation method, but should ensure that the normals are approximately correct.

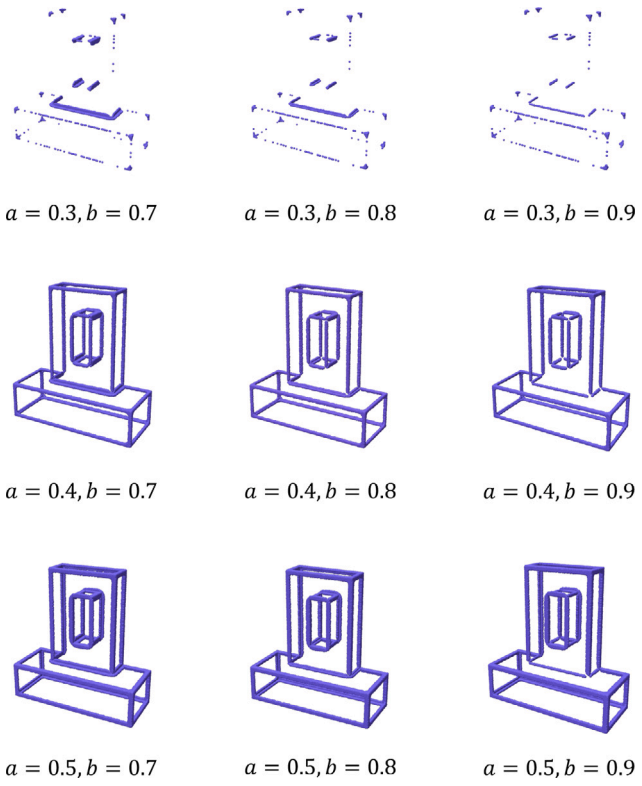

Fig. 16. Impact of different threshold settings. Points with winding numbers less than a or greater than b are classified as feature points. Inappropriate values for a or b can lead to incorrect feature points or missed features. In our method, a and b are set to 0.4 and 0.8, respectively, to achieve optimal feature detection results.

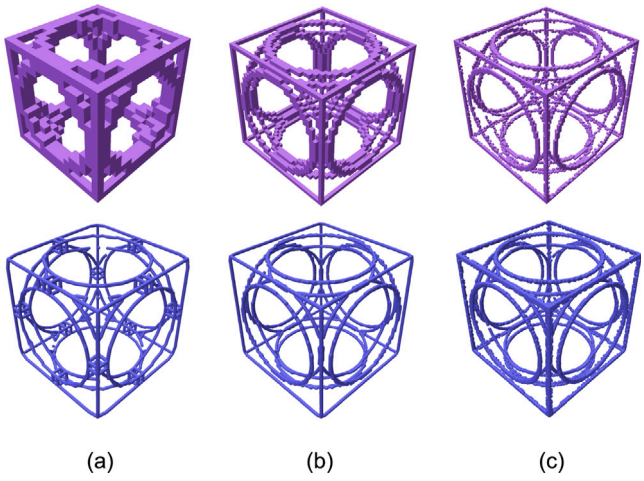

Fig. 17. Impact of different resolution settings at (a) 16, (b) 32 and (c) 64.

4.5. Influence of point density

To further demonstrate the advantages of the proposed method, we conduct a comparative experiment with the SSI method using sparse point clouds, as shown in Fig. 19. The proposed approach exhibits greater robustness to point cloud sparsity and successfully detects feature points even in sharp-angle regions. In comparison, the feature points detected by the SSI method are noticeably less complete.

4.6. Robustness to real-world objects

The primary objective of our method is to extract sharp feature lines from point clouds of real-world objects. To further validate our

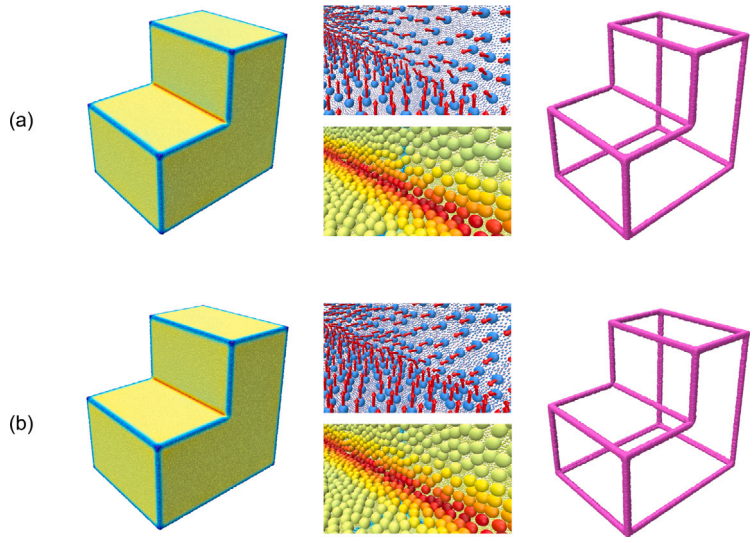

Fig. 18. Comparison of feature detection results using different normal estimation methods: (a) PCA and (b) Quadric Surface Fitting. From left to right: visualization of winding number calculations for the point cloud, local details of normal direction and the winding number (with blue indicating values near 0 and red indicating values near 1), and feature detection results. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

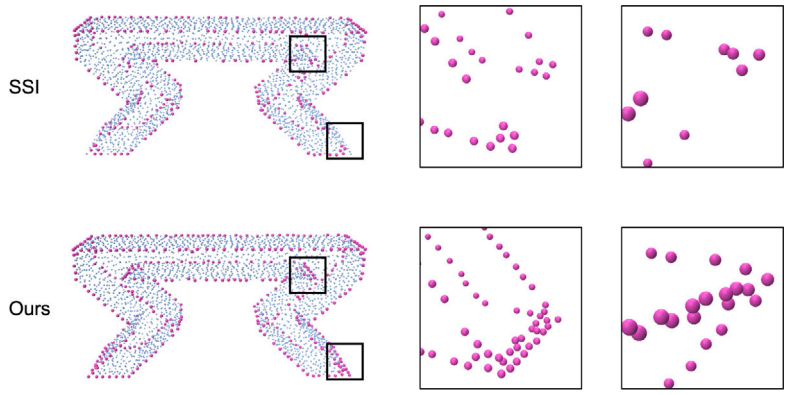

Fig. 19. Comparison of feature detection results on sparse point clouds.

method on real-world data, we performed experiments using point clouds generated by the Artec 3D scanner. As shown in Fig. 20, our method effectively captures most geometric features.

To more thoroughly evaluate the noise robustness of our method, we tested it on scanned point clouds with low noise (first row of Fig. 21) and low-quality data with severe noise, non-uniform density, and missing regions (second row of Fig. 21). Results show that our method effectively extracts feature lines from low noisy data and, despite some incompleteness, still captures the overall geometry in poor-quality point clouds.

4.7. Computational efficiency

In addition to accuracy, computational efficiency is an important metric for evaluating the real-world performance of feature line extraction algorithms. Table 3 presents the runtime of our algorithm. The feature detection step has a time complexity of $\mathcal{O}(n^2)$, where n represents the number of points in the point cloud. As the number of points increases, the runtime increases significantly, making this step substantially longer than the subsequent steps. In comparison, the feature extraction step involves converting feature points into multiple cuboids and has a time complexity of $\mathcal{O}(m^2)$, where m is the number of cuboids. This step's runtime is closely related to the number of feature points and the complexity of the geometric features, so it is less influenced by the total point cloud size and rises more slowly.

Given that the most time-intensive part of the feature detection process is the calculation of winding numbers, we optimized the algorithm by parallelizing the loop computations across multiple threads and implementing it on the GPU to improve performance. We compared the execution time of the program on the CPU to that on the GPU with parallel threading. The results show a significant improvement in execution speed with GPU acceleration, greatly reducing processing time. Overall, while the execution time increases with the number of points in the point cloud, the computational overhead remains manageable.

5. Conclusion

This paper introduces a framework for extracting sharp feature lines from 3D point clouds. The method begins by calculating the winding number of the point cloud, then selects points that exceed a threshold as detected feature points. These points are converted into a cuboid structure to extract key feature points and infer the adjacency relationships between them. Finally, these key feature points are connected and fitted to produce feature lines. Experimental results show that our method has notable advantages, with detected feature points being more coherently arranged and predominantly located on potential edges. Furthermore, the final extracted feature lines effectively capture significant geometric features of the model.

Table 3
Running time (in seconds) of our method with respect to the number of points.

Point size	Detection (CPU)	Detection (GPU)	Extraction	Total (CPU)	Total (GPU)
20K	6.44	0.12	4.96	11.40	5.08
40K	21.27	0.19	6.90	28.17	7.09
60K	44.39	0.29	7.99	52.38	8.28
80K	92.58	0.37	9.27	101.85	9.66
100K	155.82	0.46	12.86	168.69	13.32

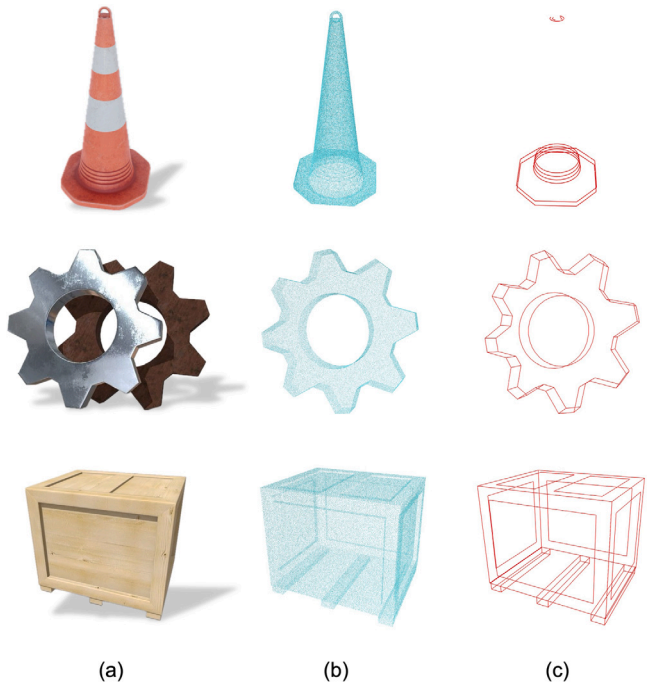

Fig. 20. Feature extraction results from uniform real-world scanned point clouds. (a) Real-world objects; (b) scanned point clouds; (c) extracted feature lines.

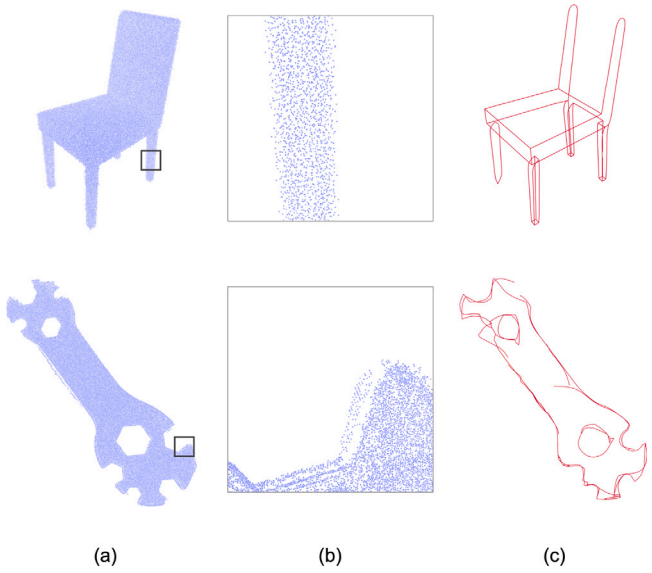

Fig. 21. Feature extraction results on low-quality real-world scanned point clouds with noise, non-uniform density, and missing regions. (a) Scanned point clouds; (b) local details of scanned point clouds; (c) extracted feature lines.

While our method proves to be more efficient than others, it does have certain limitations. For thin-shell models, although feature detection can identify accurate feature points, closely spaced edge points on opposite sides may be assigned to adjacent cubes during extraction, leading to incorrect intersecting feature lines. While increasing the resolution can alleviate this issue, it requires iterative manual parameter tuning based on experience, which is inconvenient. In the future, adaptive parameter setting based on point spacing could be developed to address this limitation more effectively.

On the other hand, the time complexity of the feature detection step is $\mathcal{O}(n^2)$. Although GPU acceleration can significantly reduce runtime, the computational cost still increases substantially when dealing with extremely large point clouds. In the winding number computation, points closer to the query point have a greater influence than those farther away. Therefore, future work may consider adopting efficient approximation strategies, such as the fast computation method proposed by Lin et al. [45], to reduce the number of calculations. Moreover, while the proposed method is robust to noisy point clouds, the accuracy of the winding number can still be affected when the normals are incorrect or noisy. To address this limitation, future improvements could involve designing a modified winding number formulation with perturbation terms to improve robustness to erroneous normals.

CRediT authorship contribution statement

Shuxian Cai: Writing – original draft, Visualization, Software, Methodology. **Juan Cao:** Writing – review & editing, Supervision, Project administration, Funding acquisition, Conceptualization. **Bailin Deng:** Writing – review & editing, Supervision, Methodology, Funding acquisition, Conceptualization. **Zhonggui Chen:** Supervision, Project administration, Methodology, Funding acquisition, Conceptualization, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

The work of Zhonggui Chen and Juan Cao was supported by the National Key R&D Program of China (No. 2022YFB3303400), National Natural Science Foundation of China (Nos. 62272402, 62372389), Natural Science Foundation of Fujian Province, China (No. 2024J01513 243), Special Fund for Key Program of Science and Technology of Fujian Province, China (No. 2022YZ040011), and Fundamental Research Funds for the Central Universities, China (No. 20720220037). The work of Bailin Deng was supported by the Xiamen Outward Mobility Fund from Cardiff University, United Kingdom.

Data availability

Data will be made available on request.

References

[1] X. Wang, Y. Xu, K. Xu, A. Tagliasacchi, B. Zhou, A. Mahdavi-Amiri, H. Zhang, PIE-Net: parametric inference of point cloud edges, in: Proceedings of the 34th International Conference on Neural Information Processing Systems, NIPS '20, Curran Associates Inc., Red Hook, NY, USA, 2020.

[2] Y. Ye, R. Yi, Z. Gao, C. Zhu, Z. Cai, K. Xu, NEF: Neural edge fields for 3D parametric curve reconstruction from Multi-View images, in: 2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR, IEEE Computer Society, Los Alamitos, CA, USA, 2023, pp. 8486–8495, http://dx.doi.org/10.1109/CVPR52729.2023.00820.

[3] X. Zhu, D. Du, W. Chen, Z. Zhao, Y. Nie, X. Han, NerVE: Neural volumetric edges for parametric curve extraction from point cloud, in: 2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR, 2023, pp. 13601–13610, http://dx.doi.org/10.1109/CVPR52729.2023.01307.

[4] R. Xu, Z. Wang, Z. Dou, C. Zong, S. Xin, M. Jiang, T. Ju, C. Tu, RFEPS: Reconstructing Feature-Line equipped polygonal surface, ACM Trans. Graph. 41 (6) (2022) http://dx.doi.org/10.1145/3550454.3555443.

[5] H. Huang, S. Wu, M. Gong, D. Cohen-Or, U. Ascher, H.R. Zhang, Edge-aware point set resampling, ACM Trans. Graph. 32 (1) (2013) http://dx.doi.org/10.1145/2421636.2421645.

[6] J. Nie, Extracting feature lines from point clouds based on smooth shrink and iterative thinning, Graph. Model. 84 (2016) 38–49, http://dx.doi.org/10.1016/j.gmod.2016.04.001.

[7] R. Xu, Z. Dou, N. Wang, S. Xin, S. Chen, M. Jiang, X. Guo, W. Wang, C. Tu, Globally consistent normal orientation for point clouds by regularizing the Winding-Number Field, ACM Trans. Graph. 42 (4) (2023) http://dx.doi.org/10.1145/3592129.

[8] D. Xiao, Y. Ma, Z. Shi, S. Xin, W. Wang, B. Deng, B. Wang, Winding clearness for differentiable point cloud optimization, 2024, arXiv:2401.13639 URL https://arxiv.org/abs/2401.13639.

[9] A. Jacobson, L. Kavan, O. Sorkine-Hornung, Robust inside-outside segmentation using generalized winding numbers, ACM Trans. Graph. 32 (4) (2013) http://dx.doi.org/10.1145/2461912.2461916.

[10] S. Fleishman, D. Cohen-Or, C.T. Silva, Robust moving least-squares fitting with sharp features, ACM Trans. Graph. 24 (3) (2005) 544–552, http://dx.doi.org/10.1145/1073204.1073227.

[11] J.I. Daniels, L.K. Ha, T. Ochotta, C.T. Silva, Robust smooth feature extraction from point clouds, in: IEEE International Conference on Shape Modeling and Applications 2007, SMI'07, 2007, pp. 123–136, http://dx.doi.org/10.1109/SMI.2007.32.

[12] C. Weber, S. Hahmann, H. Hagen, Sharp feature detection in point clouds, in: 2010 Shape Modeling International Conference, 2010, pp. 175–186, http://dx.doi.org/10.1109/SMI.2010.32.

[13] Q. Mérigot, M. Ovsjanikov, L.J. Guibas, Voronoi-Based curvature and feature estimation from point clouds, IEEE Trans. Vis. Comput. Graphics 17 (6) (2011) 743–756, http://dx.doi.org/10.1109/TVCG.2010.261.

[14] D. Bazazian, J.R. Casas, J. Ruiz-Hidalgo, Fast and robust edge extraction in unorganized point clouds, in: 2015 International Conference on Digital Image Computing: Techniques and Applications, DICTA, 2015, pp. 1–8, http://dx.doi.org/10.1109/DICTA.2015.7371262.

[15] T. Hackel, J.D. Wegner, K. Schindler, Contour detection in unstructured 3D point clouds, in: 2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR, 2016, pp. 1610–1618, http://dx.doi.org/10.1109/CVPR.2016.178.

[16] S. Xia, R. Wang, A fast edge extraction method for mobile lidar point clouds, IEEE Geosci. Remote. Sens. Lett. 14 (8) (2017) 1288–1292, http://dx.doi.org/10.1109/LGRS.2017.2707467.

[17] L. Yu, X. Li, C.-W. Fu, D. Cohen-Or, P.-A. Heng, EC-Net: An Edge-Aware point set consolidation network, in: V. Ferrari, M. Hebert, C. Sminchisescu, Y. Weiss (Eds.), Computer Vision – ECCV 2018, Springer International Publishing, Cham, 2018, pp. 398–414.

[18] L. Yu, X. Li, C.-W. Fu, D. Cohen-Or, P.-A. Heng, PU-Net: Point cloud upsampling network, in: 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2018, pp. 2790–2799, http://dx.doi.org/10.1109/CVPR.2018.00295.

[19] P. Raina, S. Mudur, T. Popa, Sharpness fields in point clouds using deep learning, Comput. Graph. (2019).

[20] C.-E. Himeur, T. Lejemble, T. Pellegrini, M. Paulin, L. Barthe, N. Mellado, PCEDNet: A lightweight neural network for fast and interactive edge detection in 3D point clouds, ACM Trans. Graph. 41 (1) (2021) http://dx.doi.org/10.1145/3481804.

[21] J. Hurtado, M. Gattass, A. Raposo, 3D point cloud denoising using anisotropic neighborhoods and a novel sharp feature detection algorithm, Vis. Comput. 39 (2023) 5823–5848, http://dx.doi.org/10.1007/s00371-022-02698-6.

[22] X. Jiao, C. Lv, R. Yi, J. Zhao, Z. Pan, Z. Wu, Y.-J. Liu, MSL-Net: Sharp feature detection network for 3D point clouds, IEEE Trans. Vis. Comput. Graphics 30 (9) (2024) 6433–6446, http://dx.doi.org/10.1109/TVCG.2023.3346907.

[23] M. Loizou, M. Averkiou, E. Kalogerakis, Learning part boundaries from 3D point clouds, Comput. Graph. Forum 39 (5) (2020) 183–195, http://dx.doi.org/10.1111/cgf.14078.

[24] D. Zhang, X. Lu, H. Qin, Y. He, Pointfilter: Point cloud filtering via Encoder-Decoder modeling, IEEE Trans. Vis. Comput. Graphics 27 (03) (2021) 2015–2027, http://dx.doi.org/10.1109/TVCG.2020.3027069.

[25] Y.-F. Feng, L.-Y. Shen, C.-M. Yuan, X. Li, Deep shape representation with sharp feature preservation, Comput.- Aided Des. 157 (2023) 103468, http://dx.doi.org/10.1016/j.cad.2022.103468.

[26] D. Bazazian, M.E. Parés, EDC-Net: Edge detection capsule network for 3D point clouds, Appl. Sci. 11 (4) (2021) 1833, http://dx.doi.org/10.3390/app11041833.

[27] T. Zhao, M. Yu, P. Alliez, F. Lafarge, Sharp feature consolidation from raw 3D point clouds via displacement learning, Comput. Aided Geom. Design 103 (2023) 102204, http://dx.doi.org/10.1016/j.cagd.2023.102204.

[28] S. Gumhold, X. Wang, R.S. MacLeod, Feature extraction from point clouds, in: Proceedings of the 10th International Meshing Roundtable, IMR, Sandia National Laboratories, 2001, pp. 11–14.

[29] M. Pauly, R. Keiser, M. Gross, Multi-scale feature extraction on Point-Sampled surfaces, Comput. Graph. Forum 22 (3) (2003) 281–289, http://dx.doi.org/10.1111/1467-8659.00675.

[30] K. Demarsin, D. Vanderstraeten, T. Volodine, D. Roose, Detection of closed sharp edges in point clouds using normal estimation and graph theory, Comput.- Aided Des. 39 (4) (2007) 276–283, http://dx.doi.org/10.1016/j.cad.2006.12.005.

[31] J. Daniels II, T. Ochotta, L. Ha, et al., Spline-based feature curves from Point-sampled Geometry, Vis. Comput. 24 (2008) 449–462, http://dx.doi.org/10.1007/s00371-008-0223-2.

[32] Y. Lin, C. Wang, J. Cheng, B. Chen, F. Jia, Z. Chen, J. Li, Line segment extraction for large scale unorganized point clouds, ISPRS J. Photogramm. Remote Sens. 102 (2015) 172–183, http://dx.doi.org/10.1016/j.isprsjprs.2014.12.027.

[33] Y. Cao, L. Nan, P. Wonka, Curve networks for surface reconstruction, 2016, arXiv:1603.08753 URL https://arxiv.org/abs/1603.08753.

[34] A. Matveev, R. Rakhimov, A. Artemov, G. Bobrovskikh, V. Egiazarian, E. Bogomolov, D. Panozzo, D. Zorin, E. Burnaev, DEF: deep estimation of sharp geometric features in 3D shapes, ACM Trans. Graph. 41 (4) (2022) http://dx.doi.org/10.1145/3528223.3530140.

[35] K. Cherenkova, E. Dupont, A. Kacem, I. Arzhannikov, G. Gusev, D. Aouada, Sepic-Net: Sharp edges recovery by parametric inference of curves in 3D shapes, in: 2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops, CVPRW, 2023, pp. 2727–2735, http://dx.doi.org/10.1109/CVPRW59228.2023.00273.

[36] C.H. Chen, Estimating discrete total curvature with per triangle normal variation, in: ACM SIGGRAPH 2023 Talks, SIGGRAPH '23, Association for Computing Machinery, New York, NY, USA, 2023, http://dx.doi.org/10.1145/3587421.3595439.

[37] G. Barill, N.G. Dickson, R. Schmidt, D.I.W. Levin, A. Jacobson, Fast winding numbers for soups and clouds, ACM Trans. Graph. 37 (4) (2018) http://dx.doi.org/10.1145/3197517.3201337.

[38] S. Lin, D. Xiao, Z. Shi, B. Wang, Surface reconstruction from point clouds without normals by parametrizing the Gauss Formula, ACM Trans. Graph. 42 (2) (2022) http://dx.doi.org/10.1145/3554730.

[39] I.T. Jolliffe, J. Cadima, Principal component analysis: a review and recent developments, Phil. Trans. R. Soc. A 374 (20150202) (2016) http://dx.doi.org/10.1098/rsta.2015.0202.

[40] Z. Huang, Y. Wen, Z. Wang, J. Ren, K. Jia, Surface reconstruction from point clouds: A survey and a benchmark, IEEE Trans. Pattern Anal. Mach. Intell. (2024) 1–20, http://dx.doi.org/10.1109/TPAMI.2024.3429209.

[41] S. Koch, A. Matveev, Z. Jiang, F. Williams, A. Artemov, E. Burnaev, M. Alexa, D. Zorin, D. Panozzo, ABC: A big CAD model dataset for geometric deep learning, in: The IEEE Conference on Computer Vision and Pattern Recognition, CVPR, 2019.

[42] P. Erler, P. Guerrero, S. Ohrhallinger, N.J. Mitra, M. Wimmer, Points2Surf learning implicit surfaces from point clouds, in: Computer Vision – ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part V, Springer-Verlag, Berlin, Heidelberg, 2020, pp. 108–124, http://dx.doi.org/10.1007/978-3-030-58558-7_7.

[43] R.T. Rockafellar, R.J.B. Wets, Variational Analysis, first ed., in: Grundlehren der Mathematischen Wissenschaften, Springer Berlin, Heidelberg, 1997, p. 736, http://dx.doi.org/10.1007/978-3-642-02431-3, Hardcover: 1997; Softcover: 2010; eBook: 2009.

[44] D.-M. Yan, W. Wang, Y. Liu, Z. Yang, Variational mesh segmentation via quadric surface fitting, Comput.- Aided Des. 44 (11) (2012) 1072–1082, http://dx.doi.org/10.1016/j.cad.2012.04.005.

[45] S. Lin, Z. Shi, Y. Liu, Fast and globally consistent normal orientation based on the winding number normal consistency, ACM Trans. Graph. 43 (6) (2024) http://dx.doi.org/10.1145/3687895.