The Visual Computer (2023) 39:5823–5848
https://doi.org/10.1007/s00371-022-02698-6

ORIGINAL ARTICLE

3D point cloud denoising using anisotropic neighborhoods and a novel sharp feature detection algorithm

Jan Hurtado[1] · **Marcelo Gattass**[1] · **Alberto Raposo**[1]

Accepted: 4 October 2022 / Published online: 22 October 2022

Abstract
3D point cloud denoising is a fundamental task in a geometry-processing pipeline, where feature preservation is essential for various applications. The literature presents several methods to overcome the denoising problem; however, most of them focus on denoising smooth surfaces and not on handling sharp features correctly. This paper proposes a new sharp feature-preserving method for point cloud denoising that incorporates solutions for normal estimation and feature detection. The denoising method consists of four major steps. First, we compute the per-point anisotropic neighborhoods by solving local quadratic optimization problems that penalize normal variation. Second, we estimate a piecewise smooth normal field that enhances sharp feature regions using these anisotropic neighborhoods. This step includes bilateral filtering and a novel corrector procedure to obtain more reliable normals for the subsequent steps. Third, we employ a novel sharp feature detection algorithm to select the feature points precisely. Finally, we update the point positions to fit them to the computed normals while retaining the sharp features that were detected. These steps are repeated until the noise is minimized. We evaluate our method using qualitative and quantitative comparisons with state-of-the-art denoising, normal estimation, and feature detection procedures. Our experiments show that our approach is competitive and, in most test cases, outperforms all other methods.

Keywords Point cloud denoising · Normal estimation · Sharp feature detection · Anisotropic neighborhoods

1 Introduction

Constructing and analyzing 3D digital models used to represent real environments are critical in the age of Industry 4.0 for various tasks, such as planning, maintenance, training, and education. 3D acquisition techniques can be used on a target scene or object to obtain accurate surface data, typically represented by an unordered set of 3D points, that is, a 3D point cloud. Although these raw data can be useful for constructing and analyzing these models, they present undesired noise due to physical limitations or inconsistencies during acquisition. Noise can disturb further processing, such as segmentation [52], compression [11], registration [5], recognition [69], mesh generation [7], or rendering [40]. Consequently, point cloud denoising is a fundamental task in common geometry-processing pipelines.

Because point clouds lack connectivity information and surface details are difficult to differentiate from noise, point cloud denoising is challenging. Early attempts only aimed to smooth the points [1]. However, this is not sufficient to obtain good-quality results for applications such as mesh generation or rendering, where feature preservation is essential. Several methods have been proposed to address this problem [30], ranging from classical moving least square-based methods to deep learning-based methods. Most of these rely on a two-step-based scheme, with the first step consisting of computing denoised normals and the second step consisting of updating point positions to fit these normals.

We propose a sharp feature-preserving point cloud denoising method consisting of four main steps. In the first step, we used local quadratic optimization problems to compute anisotropic neighborhoods. In the second step, we used these anisotropic neighborhoods to filter the point cloud normal

✉ Jan Hurtado
hurtado@tecgraf.puc-rio.br

Marcelo Gattass
mgattass@tecgraf.puc-rio.br

Alberto Raposo
abraposo@tecgraf.puc-rio.br

[1] Department of Informatics, Pontifical Catholic University of Rio de Janeiro, RJ 22541-041 Rio de Janeiro, Brazil

field and reduce noise. This step includes a normal corrector operation that minimizes the problems in the subsequent steps. In the third step, using the filtered normals, we detect sharp feature points and treat them differently. We classified all the points into non-feature, edge, and corner points. Finally, we updated the point positions using this classification and the filtered normal field. These steps were repeated until the noise was removed. The resulting point cloud had piecewise smooth regions and enhanced sharp feature regions, preserving surface details.

Herein, we compare our method to several state-of-the-art denoising methods using point and normal distance error, obtaining competitive results and outperforming them in most test cases. We include several visual comparisons using challenging test cases to demonstrate how our denoising method better preserves the surface details. In addition, because our method incorporates solutions for normal estimation and feature detection problems, we evaluate them numerically. The main contributions of this study are summarized as follows:

- We propose a point cloud denoising algorithm that focuses on processing surfaces that present sharp features. The algorithm can minimize noise while preserving the surface details.
- We present an extension to point clouds of the mesh-based anisotropic neighborhood computation proposed in [38]. This method allows us to compute piecewise smooth normals while preserving the hard transitions at the sharp feature regions.
- We present a normal correction algorithm for refining the normals of points near sharp feature regions. This algorithm provides more reliable normals for use in the denoising algorithm.
- We introduce a sharp feature detection algorithm that tends to detect thin sharp feature regions. This algorithm is useful in the denoising algorithm to avoid the generation of gaps and the excessive accumulation of points near the sharp feature regions.

The remainder of this paper is structured as follows. Section 2 presents related studies on the denoising problem. Section 3 introduces our proposed denoising method. Section 4 explains the computation of connectivity information and geometric measurements that are useful for the steps of the method. Sections 5, 6, 7, and 8 describe the four primary steps of the proposed method. Section 9 presents the experiments and results. Finally, in Sect. 10, we present our conclusions and discuss future research.

2 Previous work

Point cloud denoising has been addressed in various ways. Some approaches are based on the moving least squares (MLS) method, which aims to approximate an underlying surface and project noisy points onto it. This scheme was adapted to preserve details and achieve greater robustness against noise [23,25,26,50,53,67].

A different approach to address the denoising problem is to use sparse representations adopted by several geometry-processing algorithms. These methods assume that most noisy points can be approximated using piecewise smooth surfaces with sparse transitions. The denoising task is formulated as the sparse reconstruction of point normals and/or point positions using regularization based on the metric ℓ_1 [2,41], the metric ℓ_0 [61], or low-rank matrix approximations [13,49].

The non-local self-similarity methods proposed for images [10,16] were extended to denoising of point clouds. Different patch representations are used, such as polynomial surfaces [17], variations in height fields [18], local displacements [28,59], local probing fields [19], point normals [46], and sampled collaborative points [56], which exploit the similarity between non-local surfaces.

The relationships between the points can be represented using graphs. To address the denoising problem, some methods use this representation to exploit graph properties and Laplacian regularizers in a local [20,21,60] and non-local [73] fashion. Furthermore, the authors proposed a feature graph learning framework applied to the denoising problem in [34], using point coordinates and normals as relevant features.

Surface denoising is commonly performed in two steps, where the first step denoises the normal field and the second step fits the noisy point positions to the denoised normals. These steps are complemented by neighborhood clustering [33,65] and/or feature detection [43,71,77,78]. In a different way, Béarzi et al. introduced geometric structures named Wavejets to represent local surfaces [4]. These structures are then used to reduce noise and enhance detail by updating the point positions and normals.

The locally optimal projection (LOP) method involves sampling and projecting a set of uniformly distributed particles onto the underlying surface that the noisy point cloud represents [42]. This method was extended to be more robust against irregular distributions [35] and deal with feature preservation [36,45,54,70].

Recently, geometric deep learning methods have become popular owing to their impressive results in computer graphics applications. The point cloud denoising problem was tackled by this type of method [6,9,22,27,31,47,48,55,57,74]; however, most of them did not deal with feature preservation correctly. Yu et al. introduced an LOP-based network

for feature-aware point cloud consolidation, which can process noisy inputs [72]. Lu et al. proposed a network capable of predicting feature points and noise-free normals [44]. These predictions are used to iteratively update the point positions until the noise is removed. Similarly, Wei et al. proposed a normal refinement network that processes adaptive geometric descriptors, that is, the local height and normal fields [68]. The refined normals are then used to update the point positions.

Although the presented methods attempt to deal with detail preservation, most of them focus on the preservation of smooth features without correctly dealing with sharp features. Sparse- and two-step-based methods have shown excellent performance in the preservation of this type of feature, particularly those that include feature detection procedures in their denoising pipeline. The proposed method, which belongs to a family of two-step methods, also focuses on the preservation of sharp features. More specifically, we presented a novel method for robustly estimating a clean normal field based on anisotropic neighborhoods and a novel normal correction operation. For anisotropic neighborhood computation, we extend the idea proposed in our previous work [38], introducing a new functional and its discretization to deal with point clouds. Then, using a clean normal field, we introduce a novel sharp feature detection algorithm that is more selective than the previous methods. This behavior helps to avoid point agglomeration near feature regions and the formation of gaps. Finally, the point positions are updated using clean normals and feature detection information, similar to the method proposed by Yadav et al. [71].

3 Overview

The input of our denoising algorithm is a 3D point cloud consisting of a set of unorganized points $\mathcal{P} = \{p_i\}_{i=1}^{n}$ sampled from a piecewise 2-manifold $\mathcal{X}$ embedded in $\mathbb{R}^3$, where n is the number of points. The set of points should be equipped with a consistently oriented normal field. Let us denote the point coordinates and the corresponding normals as sets of 3-dimensional vectors $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ and $N = \{\mathbf{n}_1, \ldots, \mathbf{n}_n\}$, where $\mathbf{x}_i \in \mathbb{R}^3$ and $\mathbf{n}_i \in \mathbb{R}^3$ represent the coordinates and the normal at point p_i, respectively. The algorithm generates a noise-free point cloud with a clean normal field. This point cloud is sampled on a 2-manifold $\mathcal{X}'$, which is a good approximation of $\mathcal{X}$.

We assume that a common point cloud should be preprocessed using a normal estimation method that consistently preserves the normal orientation because our algorithm requires an initial set of normals. This method does not need to be accurate in the computation of normals; it only needs to maintain a consistent orientation. In addition to the normal initialization, we normalized the point positions to avoid high variations when tuning the algorithm parameters. We computed a simple average distance by considering each point's distance to their corresponding 10 nearest neighbors. The inverse of the average distance was then used to define the point cloud's normalization scale factor.

The algorithm operates in an iterative manner, where each iteration consists of four main steps, as described below. First, as a preparation procedure useful for representing the point cloud topology and geometry, we compute point neighborhoods, normals, areas, and distances.

In the first step, we compute anisotropic neighborhoods that define normal-based piecewise smooth regions assigned to each point. These neighborhoods are obtained by solving local quadratic optimization problems that minimize the normal variation and distances to the evaluated point.

In the second step, we compute a new set of normals based on anisotropic neighborhoods. These normals enhance the feature regions because they are fitted to piecewise smooth regions, preserving the hard transitions between them. They may contain noise because they are computed using noisy point positions. As a result, we used bilateral filtering to smooth them out while retaining the enhanced regions. We also introduce a normal correction operation because anisotropic neighborhood-based normals are prone to incorrect orientation owing to noisy input normals and the parameters used for the optimization problem. This corrector operation evaluates the neighboring piecewise smooth regions for each point to determine which region best fits it. If the current normal differs considerably from the average normal of the selected region, we assign this average normal to the new normal for the corresponding point. We begin by selecting candidate feature points that may require this evaluation to define neighboring piecewise smooth regions. We focus on potential feature points because the points that belong to the smooth regions do not require normal correction. We then use a clustering operation that aims to segment the points around the evaluated point based on their normals. Each cluster is considered a neighboring piecewise smooth region whose average normal can be used for normal correction. This normal filtering step allows us to define reliable normals and minimize artifacts in the subsequent steps.

In the third step, we apply the same feature candidate selection and neighborhood clustering operations, though we use the corrected normals in this case. Then, we detect sharp feature points using this information, classifying all points into non-features, edges, or corners. This classification is based on measuring the proximity of each point to the intersection of neighboring planes approximated to the corresponding piecewise smooth regions.

Finally, in the fourth step, we updated the point positions to fit them to the filtered normals. The point-updating scheme depends on the point class defined in the previous step.

Algorithm 1 Point cloud denoising

```
1: procedure DENOISE(P)
2:   for it_ext ← 1 to n_ext do
3:     preparingProcedure(P,ε_d, k, r_s, r_r, r_b)
4:     anisotropicNeighborhoods(P,α, β, γ, a_0)
5:     for it_int ← 1 to n_int do
6:       anisotropicNeighborhoodNormals(P,τ_u)
7:       for it_ns ← 1 to n_ns do
8:         normalSmoothing(P,σ_ns, σ_nn)
9:       end for
10:      for it_nc ← 1 to n_nc do
11:        roughFeatureClassification(P, τ_n, τ_ta)
12:        neighborhoodClustering(P)
13:        normalCorrection(P, ε_mn)
14:      end for
15:      roughFeatureClassification(P,τ_n,τ_ta)
16:      neighborhoodClustering(P)
17:      pointConvexityAnalysis(P,ε_cc)
18:      sharpFeatureDetection(P,θ,δ_cc)
19:      for it_fp ← 1 to n_fp do
20:        non-featurePointUpdate(P,τ_o,σ_ps,σ_pn,υ_f)
21:      end for
22:      featurePointUpdate(P,τ_o,υ_e,υ_c)
23:    end for
24:  end for
25: end procedure
```

The number of iterations for the full set of steps is defined by parameter n_{ext} (external). However, because computing anisotropic neighborhoods is expensive, we also consider an iterative scheme that applies the following steps within the full iteration: normal filtering, feature classification, and point updating. The number of internal iterations is defined by parameter n_{int}. Algorithm 1 summarizes the proposed pipeline, in which the color maps the main steps.

4 Preparing procedure

This section describes how we compute the connectivity information and geometric measurements required for the algorithm's main steps. Let us first denote the standard k nearest neighborhood of a point p_i as $\mathcal{N}_k(p_i)$, where p_i is also included. The *rough normal* of a point p_i is computed using principal component analysis (PCA) on the points included in $\mathcal{N}_k(p_i)$, where the eigenvector with the smallest eigenvalue defines the corresponding direction. These normals are corrected (switching sign) by checking the orientation of the input normals because they have no consistent orientation.

Then, in an attempt to define an analogous representation to the first ring neighborhood as a mesh-based representation for each point p_i, we computed a local 2D Delaunay triangulation using the $\mathcal{N}_k(p_i)$ points projected on the plane defined by p_i and its corresponding *rough normal*. Because the point cloud can present multiple points that are very close to each other, when we compute the triangulation, we ignore the points that are at a distance less than $\epsilon_d l_{ro}$ in the 3D space, where ϵ_d is a tolerance factor and l_{ro} is a *rough average distance* computed considering the distances from each point to the closest 6 points. The latter allows us to obtain a more reliable triangulation for the subsequent operations. For each p_i, the points that correspond to the first ring in the triangulation, including p_i, comprise neighborhood $\mathcal{N}_1(p_i)$ on the point cloud.

Similar to the average edge length computed on mesh-based representations, we computed the *average distance* l_μ between points considering the distances from each p_i to $p_j \in \mathcal{N}_1(p_i)$, s.t. $j \neq i$. In addition, using the computed local triangulations, the area where a point p_i represents is obtained by computing the corresponding barycentric cell area in the 3D space, that is, one-third of the total area of all triangles that p_i shares. We denote all areas as the vector $\mathbf{a} = (a_1, \ldots, a_n)^T$, where a_i represents the area of point p_i.

Then, we define three types of neighborhoods: small, regular, and big, denoted as $\mathcal{N}_s$, $\mathcal{N}_r$ and $\mathcal{N}_b$, respectively, which are defined as follows:

$$\begin{aligned}
\mathcal{N}_s(p_i) &= \left\{ p_j \in \mathcal{P} \,\middle|\, \|\mathbf{x}_j - \mathbf{x}_i\| < r_s l_i \right. \\
&\quad \left. \wedge \langle \mathbf{n}_j, \mathbf{n}_i \rangle > 0 \right\} \cap \mathcal{N}_k(p_i), \\
\mathcal{N}_r(p_i) &= \left\{ p_j \in \mathcal{P} \,\middle|\, \|\mathbf{x}_j - \mathbf{x}_i\| < r_r l_\mu \right. \\
&\quad \left. \wedge \langle \mathbf{n}_j, \mathbf{n}_i \rangle > 0 \right\} \cap \mathcal{N}_k(p_i), \\
\mathcal{N}_b(p_i) &= \left\{ p_j \in \mathcal{P} \,\middle|\, \|\mathbf{x}_j - \mathbf{x}_i\| < r_b l_\mu \right. \\
&\quad \left. \wedge \langle \mathbf{n}_j, \mathbf{n}_i \rangle > 0 \right\} \cap \mathcal{N}_k(p_i),
\end{aligned} \tag{1}$$

where l_i is the maximum distance from point p_i to any $p_j \in \mathcal{N}_1(p_i)$, and r_s, r_r and r_b are parameters used to define the size of the neighborhoods, s.t. $(1 \leq r_s) \wedge (r_r < r_b)$. $\mathcal{N}_s$ maps the immediate interaction between points, $\mathcal{N}_r$ represents a local surface for each point, and $\mathcal{N}_b$ represents a larger local surface whose convexity or concavity is more evident. These neighborhoods were used in different steps of our denoising algorithm.

Finally, we compute the set of *regular normals* using the PCA-based method on the *regular neighborhood* points $\mathcal{N}_r(p_i)$, maintaining orientation consistency. The anisotropic neighborhoods were then computed using these normals.

5 Anisotropic neighborhoods computation

The computation of anisotropic neighborhoods is a common technique used for denoising [24,29,38,64,76,79], where the notion is to compute local shape adaptive structures that can then be used to filter out noise while preserving the features. We present an extension to the point clouds of mesh-based anisotropic neighborhood computation introduced in [37,38]. These anisotropic neighborhoods aim to compute pointwise descriptors that define the membership

of neighboring points to a piecewise smooth surface region represented by the evaluated points. We assume that a smooth region presents low normal variation such that the difference between the two normals of any two points within this region is minimal. In addition, the shape of the region should be as regular as possible, centered at the evaluated point, represent a considerable area on the underlying surface, and present similar point normals to the normal of the evaluated point.

In a continuous setting, let us consider the 2-manifold $\mathcal{X}$, evaluated point $x' \in \mathcal{X}$, and continuous membership function $u^* : \mathcal{X} \rightarrow [0, 1]$ describing the anisotropic neighborhood, where 0 denotes no membership and 1 denotes full membership. We aim to determine an optimal membership function u^* by solving the following optimization problem:

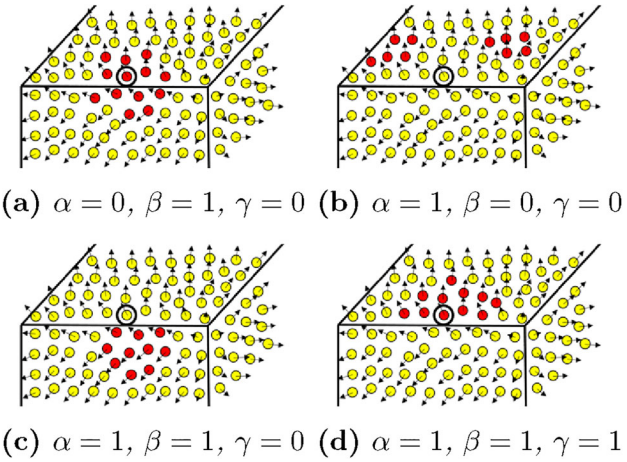
$$\begin{aligned} u^* = \underset{u}{\operatorname{argmin}}\, & \alpha \int_{x_i \in \mathcal{X}} \int_{x_j \in \mathcal{X}} \|n_i - n_j\| u_i u_j da da \\ & + \beta \int_{x_i \in \mathcal{X}} \|x' - x_i\| u_i da + \gamma \int_{x_i \in \mathcal{X}} \|n' - n_i\| u_i da \\ & s.t. \quad u \in [0, 1] \ \wedge \ \int_{x_i \in \mathcal{X}} u da = a_0, \end{aligned} \tag{2}$$

where n' is the normal of x', the first term penalizes the normal variation, the second term penalizes the distance to x', the third term penalizes the normal difference regarding n', the upper and lower bound constraints maintain the values of u^* between 0 and 1, the linear constraint helps to avoid 0 area solutions by defining an area a_0 to be covered by u^*, and the parameters α, β, and γ control the behavior of the solution. We do not include a term to control the regularity of u^*, such as gradient norm penalization, unlike [38], because point cloud connectivity is not as well defined as mesh connectivity. We indirectly control the regularity of u^* by tuning β.

In a discrete setting, considering a point cloud $\mathcal{P}$ as a sample of $\mathcal{X}$, we can represent the coordinates of the evaluated point x' as $\mathbf{x}'$, n' as $\mathbf{n}'$, the membership function u^* as the vector $\mathbf{u}^* = (u_1^*, \ldots, u_n^*)^T$, the distances between all the points and $\mathbf{x}'$ as the vector $\mathbf{d} = (d_1, \ldots, d_n)^T$, where $d_i = \|\mathbf{x}_i - \mathbf{x}'\|$, the distances between all the point normals with $\mathbf{n}'$ as the vector $\mathbf{f} = (f_1, \ldots, f_n)^T$, where $f_i = \|\mathbf{n}_i - \mathbf{n}'\|$, and the area for each point as the vector $\mathbf{a}$, described previously. The optimization problem can then be described as follows:

$$\begin{aligned} \mathbf{u}^* = \underset{\mathbf{u}}{\operatorname{argmin}}\, & \alpha \mathbf{u}^T \mathbf{A}^T \mathbf{Q} \mathbf{A} \mathbf{u} + \beta \mathbf{d}^T a' \mathbf{A} \mathbf{u} \\ & + \gamma \mathbf{f}^T a' \mathbf{A} \mathbf{u} \quad s.t. \quad \mathbf{0} \leq \mathbf{u} \leq \mathbf{1} \wedge \mathbf{a}^T \mathbf{u} = a_0, \end{aligned} \tag{3}$$

where $\mathbf{Q}$ is a square matrix whose entries are defined by $q_{ij} = \|\mathbf{n}_i - \mathbf{n}_j\|$, $\mathbf{A}$ is a diagonal matrix containing the point areas as diagonal elements, that is, $a_{ii} = a_i$, $\mathbf{0}$ is a vector of zeros, $\mathbf{1}$ is a vector of ones, and a' is the area of the evaluated point that is used to alleviate the difference between the linear and quadratic terms. We compute an optimal solution $\mathbf{u}^*$ for each point that allows us to describe anisotropic neighborhoods. For practical purposes, we constrained the domain of u^* to the *regular neighborhood* $\mathcal{N}_r$ of the evaluated point; that is, $\mathbf{u}^* = (u_1^*, \ldots, u_{|\mathcal{N}_r|}^*)^T$, where $|\mathcal{N}_r|$ denotes the number of elements in $\mathcal{N}_r$. In addition, we define a_0 as the proportion of the total area represented by the corresponding neighborhood.



(a) $\alpha = 0,\ \beta = 1,\ \gamma = 0$ **(b)** $\alpha = 1,\ \beta = 0,\ \gamma = 0$

(c) $\alpha = 1,\ \beta = 1,\ \gamma = 0$ **(d)** $\alpha = 1,\ \beta = 1,\ \gamma = 1$

Fig. 1 Anisotropic neighborhood computation behavior. For all cases, the evaluated point is marked by a black circle, the black arrows represent the point normals, and the point color defines whether it corresponds to the neighborhood (red) or not (yellow)

Figure 1 illustrates the behavior and importance of each term in the proposed optimization problem. If we consider only the distance to the evaluated point, we obtain a solution similar to a regular geometric neighborhood, as shown in Fig. 1a. We can obtain solutions that are not sufficiently close to the evaluated point or irregular and sparse solutions by simply considering the normal difference within the neighborhood, as shown in Fig. 1b. If we consider the distance to the evaluated point, we can control the regularity and closeness of the evaluated points, as shown in Fig. 1c and 1d. Figure 1c presents a solution that lies on the incorrect face of the shape. The penalization of the normal difference with respect to the evaluated point normal can help compute the desired neighborhood, as shown in Fig. 1d.

Although all terms are important for obtaining the desired solution, we assign more importance to the normal variation penalization within the neighborhood in our experiments. Figure 2 shows an example of the computation of anisotropic neighborhoods on a noisy point cloud of a cube. The selected points present challenging situations. The first case shows a flat point close to a corner, where the computed neighborhood represents the correct face of the cube. The second case depicts an edge point where the computed neighborhood is defined on only one of the shared faces of the cube. The third case depicts a corner point, where the computed neighborhood is defined on only one of the three possible

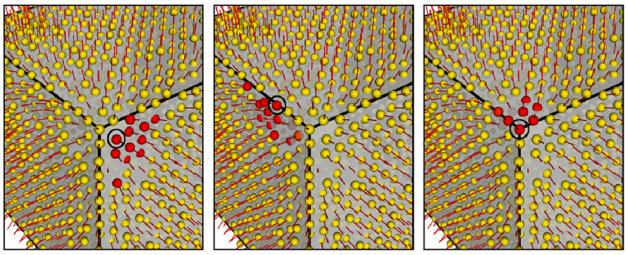


Fig. 2 Examples of anisotropic neighborhoods computed using $\alpha = 1$, $\beta = 0.1$, and $\gamma = 0.5$ on the Cube point cloud. The point color maps the membership value, which ranges from 0 to 1 (yellow to red). A black circle denotes the evaluated point. The red arrows represent the point regular normals

faces, as in the previous case. In both cases, the choice of one of the involved faces was irrelevant for the remainder of our denoising pipeline.

6 Normal filtering

Our normal filtering step consists of four main sequential operations. The operations are explained as follows.

6.1 Normal estimation using anisotropic neighborhoods

We use the anisotropic neighborhoods to estimate the feature-preserving point normals after we compute them. The anisotropic neighborhood for a point p_i is defined by a membership function $\mathbf{u}^*$ with values between 0 and 1. We apply a simple threshold operation to select the member points and construct the neighborhood $\mathcal{N}_a(p_i) = \left\{p_j \in \mathcal{N}_r(p_i) \middle| u_j > \tau_u\right\}$, where τ_u defines an acceptable membership function value. We compute the PCA-based normal using all included points, whose orientation consistency is controlled by the *regular normal*. These anisotropic neighborhood normals enhance feature regions, as shown in Fig. 3.

The selection of τ_u is not critical because the values of $\mathbf{u}^*$ tend to be close to 0 or 1 when using the appropriate values for α, β, and γ. In contrast to the function proposed in [38], we do not force smooth $\mathbf{u}^*$ transitions between neighboring points, so the optimization process is guided by the normal difference and spatial distance only. Because the input is a noisy point cloud, including more points with low membership function values (e.g., 0.2), it increases the normal variability and the distance to the evaluated point when compared to a solution with only a few points with values of $\mathbf{u}^*$ equal to 1.

6.2 Normal smoothing using bilateral filter

Because anisotropic neighborhood points are noisy, the resulting PCA-based normals can also be noisy. For this reason, we smoothed them using a bilateral filtering scheme,

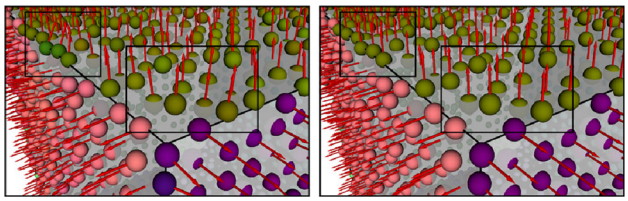


Fig. 3 Normal smoothing using bilateral filtering on the Cube point cloud. The red arrows represent the normals. The point color maps the normal direction. Left: anisotropic neighborhood normals. Right: smoothed normals

aiming to preserve high normal variations in feature regions. The bilateral filter is applied iteratively to the normal field by considering spatial and normal distances. The new normal $\tilde{\mathbf{n}}_i$ for each iteration is computed as follows:

$$\tilde{\mathbf{n}}_i = \frac{1}{W_n(p_i)} \sum_{p_j \in \mathcal{N}_r(p_i)} w_{ij} \mathbf{n}_j, \tag{4}$$

where $w_{ij} = K_{ns}\left(\|\mathbf{x}_i - \mathbf{x}_j\|\right) K_{nn}\left(\|\mathbf{n}_i - \mathbf{n}_j\|\right)$, K_{ns} and K_{nn} are Gaussian kernel functions used to smooth spatial and normal distances, $W_n(p_i)$ is a normalization factor, i.e., $W_n(p_i) = \sum_{p_j \in \mathcal{N}_r(p_i)} w_{ij}$, and the behavior of the kernels K_{ns} and K_{nn} is defined by the standard deviations σ_{ns} and σ_{nn}, respectively. The number of smoothing iterations is defined by the parameter n_{ns}. Figure 3 shows an example in which noisy anisotropic neighborhood normals are processed to obtain smoother normals and preserve the high variation in feature regions. These normals better represent the underlying surface.

This operation is used to refine anisotropic neighborhood normals. Thus, we apply only a few iterations using a low value for σ_{nn} and fix σ_{ns} as a proportion of the average edge length l_μ. The first row in Fig. 4 shows the normal estimation error for the four test cases using different numbers of iterations and σ_{nn} values. To measure the normal estimation error, we use the *root mean square measure with a threshold for multiple normals* ($RMSM_\tau$) introduced in [75]. The bottom-left corner location of each subfigure represents the error of the anisotropic neighborhood normals. It is worth noting that in all cases, we can reduce the normal error by performing a few smoothing iterations with an appropriate σ_{nn} value. For the first three test cases, we should avoid values of σ_{nn} that are greater than 0.5. The fourth test case represents a surface with smooth features; therefore, $\sigma_{\text{nn}} = 0.6$ seems to show a greater improvement. For more details about the test cases and error measurement, see Sect. 9.

6.3 Rough feature classification and neighborhood clustering

This operation is required for normal correction and sharp feature detection. The goal is to estimate the feature candidate

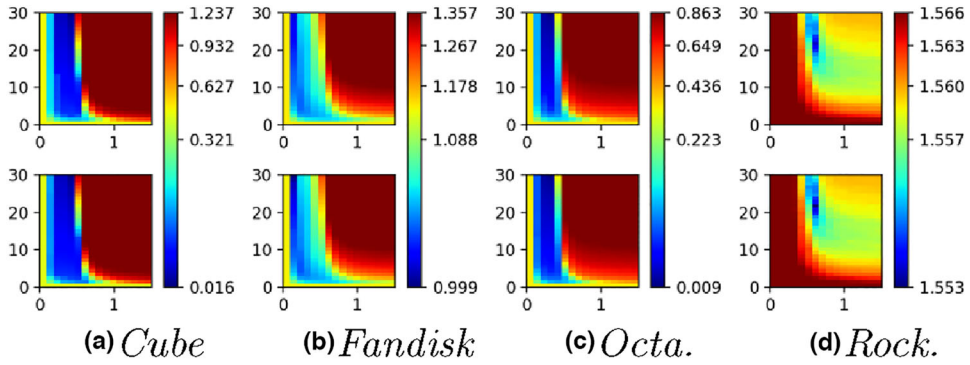


Fig. 4 Normal estimation error using $RMSM_\tau$ on the synthetic point clouds. For each subfigure, the horizontal axis represents different values for σ_{nn}, the vertical axis represents different values for n_{ns}, and the color represents the $RMSM_\tau$ values. The color map follows an exponential behavior. The first row corresponds to the results after normal smoothing, and the second row corresponds to the results after normal correction

points and cluster their *regular neighborhoods* $\mathcal{N}_r$ so that each cluster represents a piecewise smooth region regarding the processed normals.

First, we select an initial set of candidate feature points $\mathcal{P}^m$ based on the maximum normal variation within their *small neighborhoods* $\mathcal{N}_s$:

$$\mathcal{P}^m = \left\{ p_i \in \mathcal{P} \middle| \max_{p_j, p_k \in \mathcal{N}_s(p_i)} \|\mathbf{n}_j - \mathbf{n}_k\| > \tau_n \right\}, \tag{5}$$

where τ_n is a threshold value used to define whether the difference between normals is sufficient to consider the corresponding point as a possible feature.

As in [71], we define three types of points: non-feature, edge, and corner. We assume that any points not included in $\mathcal{P}^m$ are non-features. Then, considering only the candidate feature points, we clustered their corresponding *regular neighborhoods* $\mathcal{N}_r$ to obtain normal-based piecewise smooth regions. The clustering process, which uses the processed normals as inputs, is explained below.

The processed normals can be represented as points on the Gaussian sphere; therefore, we select the 3 farthest points and connect them, generating a triangle within the sphere. In the case of non-feature points, we expect that this triangle will present an area close to zero because the Gauss sphere points will be close to each other. We anticipated two dominant locations in the case of edge points, resulting in an irregular triangle with an area close to zero. We anticipated distant points and a triangle with a larger area in the case of corner points.

Based on these assumptions, we define a threshold triangle area τ_{ta} to determine whether the evaluated point can be considered as a corner point candidate. Otherwise, we check if it is an edge point candidate by measuring the largest distance between the 3 sampled points. If this distance is greater than τ_n, we consider the corresponding point as an edge point candidate. Otherwise, the point is considered a non-feature point candidate. Although we filtered non-feature points in the feature candidate selection operation, we filtered them again because we used $\mathcal{N}_r$ instead of $\mathcal{N}_s$. Let us define $\mathcal{P}^{\text{nfc}}$, $\mathcal{P}^{\text{ec}}$, and $\mathcal{P}^{\text{cc}}$ as the sets containing non-feature, edge, and corner candidate points, respectively. $\mathcal{P}^{\text{nfc}}$ also contains the points that are not included in $\mathcal{P}^m$. We clustered the neighborhood points based on this classification. If the evaluated point is a non-feature candidate, we assign a single cluster that contains all *regular neighborhood* points. If the point is an edge candidate, we define two clusters whose seeds are the most distant point normals. The points are then assigned to the cluster with the closest seed with respect to the normal difference. If the point is a corner candidate, we define 3 clusters whose seeds are three 3 sampled normals. However, if the fourth farthest normal is at a distance greater than 1 from these seeds, it is included as an additional seed. The points were assigned as in the previous case. Figure 5 shows some examples of *regular neighborhood* clustering at different points in a cone point cloud. It is worth noting that the previous operations can produce smooth transitions between normals in areas where the surface is smooth. Furthermore, the normal at the cone's corner point represents one of the surrounding surfaces assigned during anisotropic neighborhood computation, as shown in Fig. 2.

As previously stated, these operations are required in two stages. We did not include the evaluated points in their corresponding clusters in the case of normal correction because the idea of normal correction is to select the neighboring cluster that best fits the evaluated point. In the case of the sharp feature detection step, we retain the evaluated points in their corresponding clusters because they are based on corrected normals, resulting in more reliable clusters used to define sharp feature regions.

We adopted these simple operations because we expected clean normals as inputs. Thus, τ_n and τ_{ta} can be tuned by ignoring noise. τ_n works as a feature threshold, and τ_{ta} defines the possible corner points. Although the number of clusters is limited to the maximum number of seeds, i.e., 4, we can approximate various feature types using them.

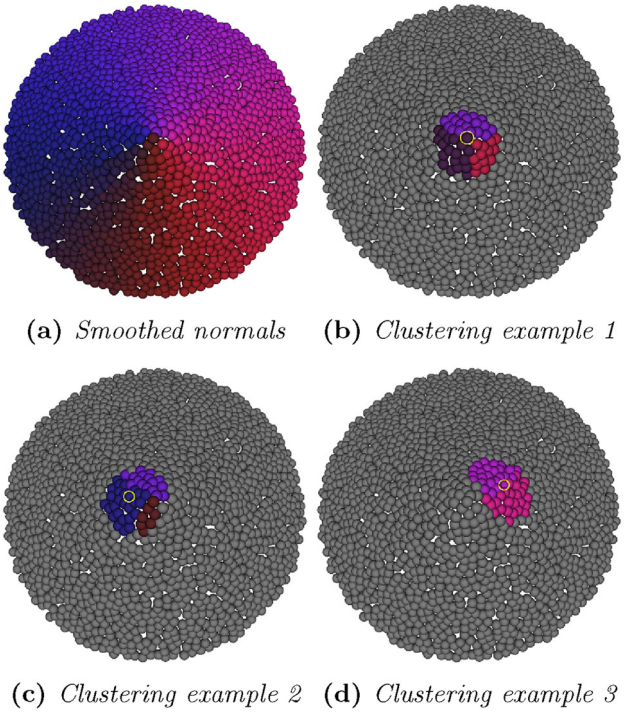


Fig. 5 Clustering examples on a cone point cloud viewed from the top. The first subfigure shows the normals obtained in the previous operations, where the point color maps the corresponding direction. The other subfigures show different clustering examples for different points (highlighted in yellow), where the point color maps the corresponding cluster average normal

6.4 Normal correction

Some estimated normals can point in the wrong direction due to the initial noisy PCA-based normals and the use of non-ideal parameters for the anisotropic neighborhood computation step, especially for points close to sharp feature regions. In this operation, we correct these undesired normals to represent the underlying geometry better and avoid problems in the following steps.

Once we define the feature candidate points, that is, $\mathcal{P}^{ec} \cup \mathcal{P}^{cc}$, we evaluate their regular neighborhood clusters to define the ones that better fit the corresponding points. Recall that the clustering applied for this operation did not include the evaluated points. We used two types of point-to-cluster distances to measure how well a cluster fits a feature candidate point.

First, assuming that each cluster represents a plane, we compute the average point-to-plane distance considering all planes formed by the cluster points and their corresponding normals. The average point-to-plane distance d_{plane} for an evaluated point p_i and cluster $\mathcal{C}$ is defined as follows:

$$d_{plane}(p_i, \mathcal{C}) = \frac{1}{|\mathcal{C}|} \sum_{p_j \in \mathcal{C}} \left\| \langle \mathbf{n}_j, \mathbf{x}_i - \mathbf{x}_j \rangle \right\|, \tag{6}$$

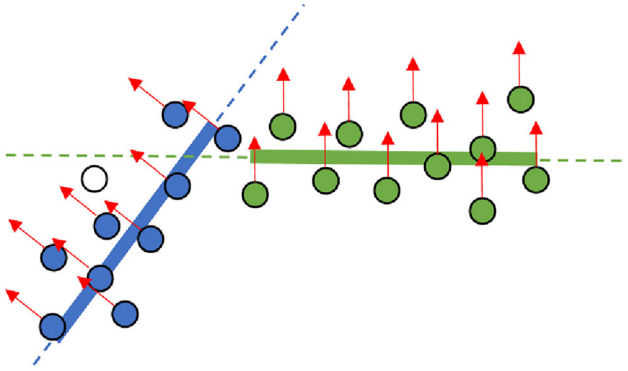


Fig. 6 Importance of the d_{ch} distance. The white point represents the evaluated point whose normal should be corrected. The green and blue points represent two different clusters. The red arrows represent the normals for each point. The dashed colored lines represent the corresponding approximated planes for each cluster, i.e., $(\mathbf{x}_\mathcal{C}, \mathbf{n}_\mathcal{C})$. The colored rectangles represent the CH regions on the corresponding planes. The evaluated point is closer to the approximated planes of the green cluster, but it is distant from the corresponding CH region. Considering both distances, d_{plane} and d_{ch}, the evaluated point is closer to the blue cluster

where $|\mathcal{C}|$ denotes the number of elements in $\mathcal{C}$,

Second, we assumed that each cluster represents a bounded plane region. To define this region, we compute the average normal $\mathbf{n}_\mathcal{C}$ and average position $\mathbf{x}_\mathcal{C}$ for cluster $\mathcal{C}$, representing the plane of the cluster. We project all the cluster points onto this plane and use them to compute a 2D convex hull (CH) shape that represents the boundaries of the cluster. We denote the 2D points of the CH as the ordered set $\mathcal{C}^\pi_{ch}$, which also represents the CH polygon. Distance d_{ch} in this region is defined as follows:

$$d_{ch}(p_i, \mathcal{C}) = \begin{cases} d_{ch\_out}(p_i, \mathcal{C}) & \text{if } \mathbf{x}_i^\pi \text{ lies out of CH shape} \\ 0 & \text{otherwise} \end{cases}, \tag{7}$$

where $\mathbf{x}_i^\pi$ is the projection of $\mathbf{x}_i$ onto the plane defined by $\mathbf{n}_\mathcal{C}$ and $\mathbf{x}_\mathcal{C}$. Term $d_{ch\_out}(p_i, \mathcal{C})$ denotes the conventional point-to-polygon distance between $\mathbf{x}_i^\pi$ and the corresponding CH polygon $\mathcal{C}^\pi_{ch}$.

The final point-to-cluster distance is defined as $d_{cluster}(p_i, \mathcal{C}) = 0.5 d_{plane}(p_i, \mathcal{C}) + 0.5 d_{ch}(p_i, \mathcal{C})$. Figure 6 illustrates the importance of combining both distances, with the ideal cluster for the evaluated point being blue. However, if we consider the d_{plane} distance only, the green cluster will be assigned because its corresponding approximated planes are closer than those approximated to the blue cluster. Considering a bounded region in this measurement (i.e., the CH) allows us to choose the correct cluster.

Then, for each $p_i \in \mathcal{P}^{ec} \cup \mathcal{P}^{cc}$, we define the average normal of the closest cluster as $\mathbf{n}^*_\mathcal{C}$ with respect to distance

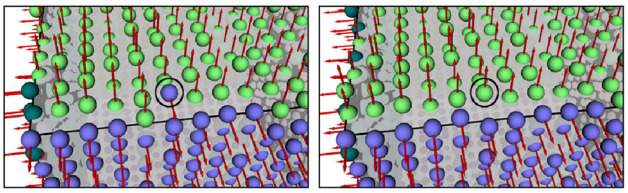


Fig. 7 Normal correction on the Cube point cloud. The red arrows represent the normals. The point color maps the normal direction. The circled point presents an incorrect normal direction which is modified after normal correction. Left: normals generated after the smoothing operation. Right: corrected normals

$d_{cluster}$. If $\|\mathbf{n}_i - \mathbf{n}^*_{\mathcal{C}}\| > \tau_n$ we assign $\mathbf{n}^*_{\mathcal{C}}$ as the corrected normal for p_i. Note that the normal is updated only if the new normal is significantly different. For example, in the cone point cloud presented in Fig. 5, the normals will not undergo modifications because the corresponding cluster average normals are similar to the current normals. This corrector operation is iterative, where the number of iterations is defined by the parameter n_{nc}, and rough feature classification and neighborhood clustering procedures are applied for each iteration. However, the greater the number of iterations, the greater the chance of introducing artifacts. In our preliminary experiments, we found that 2 or 3 iterations were sufficient to improve the smoothed normals. The second row in Fig. 4 shows the $RMSM_\tau$ results when using this corrector operation, where we can see a slight improvement. Figure 7 shows an example of how some normals are corrected to generate a more reliable normal field. This operation is one of the main novelties of our proposal, and it is required in the following steps to define accurate edge lines.

The distance to non-selected clusters can be very similar to the distance to the closest cluster in the case of true edges and corner points. This allows us to adopt a multi-normal scheme that is useful for the point-updating step, particularly for the updating of non-feature points (more details in the following sections). We denote the closest cluster as $\mathcal{C}^*$. We consider the average normal $\mathbf{n}_{\mathcal{C}}$ of another cluster $\mathcal{C}$ to be an additional normal if $d_{plane}(p_i, \mathcal{C}) < d_{plane}(p_i, \mathcal{C}^*) + \epsilon_{mn}$, where ϵ_{mn} is the tolerance value used to define the proximity criteria. We do not consider the distance d_{ch} for this estimation because the CH of the closest cluster can include consecutive edge points, generating a considerable d_{ch} distance from the other clusters. Figure 8 illustrates this case, where the evaluated point presents d_{ch} distance equal to 0 for the green cluster and a considerable d_{ch} distance for the blue cluster. Thus, the blue cluster will never be considered for multi-normal assignment if the tolerance value is too restrictive.

7 Sharp feature detection

We aim to detect feature points precisely using the corrected normals by estimating the edges of the underlying surface and selecting only the closest points to them as features, avoiding the inclusion of nearby points that can be considered part of the surrounding smooth regions.

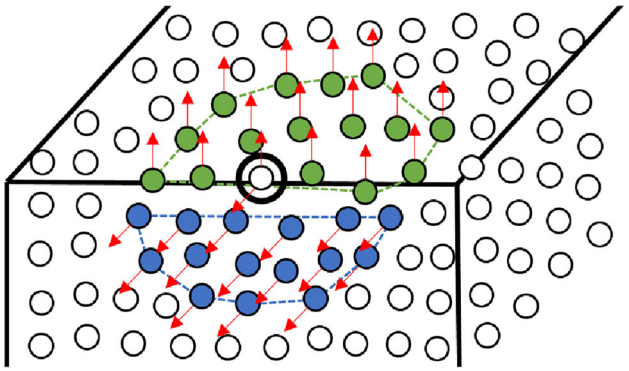


Fig. 8 Excluding d_{ch} for multi-normal analysis. The white point represents the evaluated point, for which we expect a multi-normal assignment since it is located on the edge of the underlying surface. The green and blue points represent two different clusters. The red arrows represent the normals for each point. The colored dashed lines represent the corresponding CH regions. The evaluated point is considerably distant from the blue cluster, regarding d_{ch}

First, we apply the same process used for normal filtering to cluster point *regular neighborhoods* of feature candidate points; however, in this case, we include the evaluated point p_i in the corresponding closest cluster $\mathcal{C}_1$. We define the other possible clusters as $\mathcal{C}_m$, where $m \in \{2, 3\}$ or $m \in \{2, 3, 4\}$ for the corner candidate points and $m = 2$ for the edge candidate points.

Then, to define if p_i is a feature point, p_i should be the closest point to the edge between $\mathcal{C}_1$ and $\mathcal{C}_m$ within a narrow neighborhood whose principal direction crosses the edge tangent line perpendicularly at the closest point to p_i. Figure 9 illustrates this concept, where we show that multiple points p_i are selected as feature points because they are the closest points to the edge of the underlying surface within their narrow neighborhoods. Note that in all cases, the green and blue clusters represent curved and flat surfaces, respectively. Thus, the edge of the underlying surface is a curve that is difficult to approximate owing to the point distribution and presence of noise. Therefore, we use multiple planes to approximate the local surface of p_i and the local surfaces for each $p_j \in \mathcal{C}_m$, generating intersection lines between the plane of p_i and the planes of each p_j. These lines approximate the edge of the underlying surface, and we use them to define whether point p_i is a feature point.

Consider $\mathscr{P}_i$ and $\mathscr{P}_j$ as the planes formed by the coordinates and normals of p_i and point $p_j \in \mathcal{C}_m$. The intersection between $\mathscr{P}_i$ and $\mathscr{P}_j$ is considered the edge line for this pair of points, the direction of which is denoted by $\mathbf{e}_{ij}$. Subsequently, we compute a vector $\mathbf{e}_i^\perp$ on $\mathscr{P}_i$ pointing in a direction orthogonal to $\mathbf{e}_{ij}$, that is, $\mathbf{e}_i^\perp = \mathbf{n}_i \times \mathbf{e}_{ij}$. Following the same idea, we compute a vector $\mathbf{e}_j^\perp$ on $\mathscr{P}_j$ orthogonal to $\mathbf{e}_{ij}$, that is,

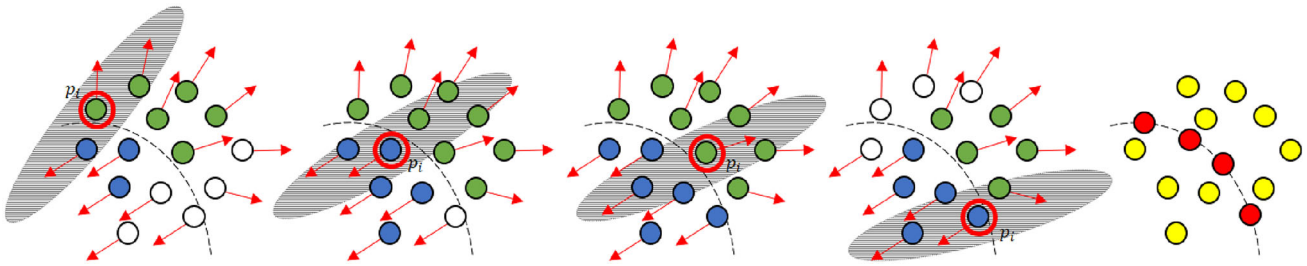


Fig. 9 Sharp feature detection intuition. In the first four subfigures, we show the sharp feature detection analysis for four different points p_i (circled in red) that are considered sharp feature points. The red arrows represent the corrected normals, the point color of the corresponding cluster, and the dashed line at the edge of the underlying surface. The shaded region defines the region we should analyze (narrow neighborhood) for each point to define if it is the closest one to the edge of the underlying surface. The last subfigure shows how the detected sharp feature points should be updated to approximate the edge of the underlying surface, where the yellow points are non-feature points and the red points are sharp feature points

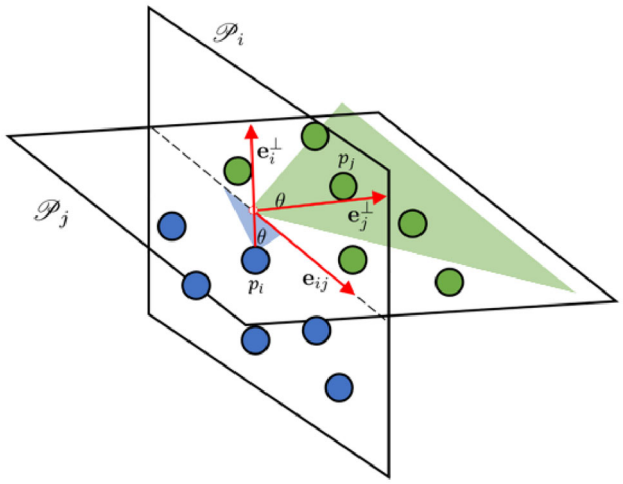


Fig. 10 Sharp feature detection narrow neighborhood. The blue points correspond to $\mathcal{C}_1$. The green points correspond to $\mathcal{C}_m$. The shaded region defines the field of view on $\mathscr{P}_i$ and $\mathscr{P}_j$. The points that lie in this region are considered part of the narrow neighborhood

$\mathbf{e}_j^{\perp} = \mathbf{n}_j \times \mathbf{e}_{ij}$. Using $\mathbf{e}_i^{\perp}$ and $\mathbf{e}_j^{\perp}$, we define the narrow neighborhood for both clusters as follows. For $\mathcal{C}_1$, We projected all cluster points on the plane $\mathscr{P}_i$. Subsequently, we trace a line centered at $\mathbf{x}_i$ in the direction $\mathbf{e}_i^{\perp}$. This line, combined with a tolerance angle θ, defines the field of $\mathscr{P}_i$. Thus, if we trace a line from $\mathbf{x}_i$ to another projected point included in $\mathcal{C}_1$ and the angle between both lines is lower than $\theta/2$, we consider the corresponding point as part of the narrow neighborhood. For $\mathcal{C}_m$, we first define the projection of $\mathbf{x}_i$ onto $\mathscr{P}_j$ as $\mathbf{x}_i^{\pi_j}$. Then, we project all the points included in $\mathcal{C}_m$ onto the plane $\mathscr{P}_j$. Similar to the previous case, we trace a line centered at $\mathbf{x}_i^{\pi_j}$ along the direction $\mathbf{e}_j^{\perp}$, which, combined with the same tolerance angle θ, defines a field of view of $\mathscr{P}_j$. As in the previous case, we consider the points that lie in the field of view as part of a narrow neighborhood. Figure 10 illustrates the definition of this narrow neighborhood.

Then, for each point p_j, we check if p_i is the closest point to the intersection line between the planes $\mathscr{P}_i$ and $\mathscr{P}_j$ by considering the points included in the corresponding narrow neighborhoods defined on $\mathcal{C}_1$ and $\mathcal{C}_m$. If p_i is the closest point for all $p_j \in \mathcal{C}_m$, we consider p_i to be a feature point regarding $\mathcal{C}_m$. Using this information, we classify each point p_i into non-features, edges, or corners as follows: If p_i is not the closest point considering all available $\mathcal{C}_m$ clusters, we assign it to the set $\mathcal{P}^{nf}$ of non-feature points. $\mathcal{P}^{nf}$ also includes non-feature candidate points $\mathcal{P}^{nfc}$. If p_i is the closest point to only one cluster $\mathcal{C}'_m$, we assign it to the set $\mathcal{P}^e$ of the edge points. For this set of points, we also included a global edge direction $\mathbf{e}_i$ by selecting the direction $\mathbf{e}_{ij}$ of the closest edge line regarding the cluster $\mathcal{C}'_m$. If p_i is the closest point of the two clusters, we measure whether it is a local maximum or a minimum regarding an approximated tangent plane. We define this tangent plane using the *regular normal* computed in the preparation procedure and point position $\mathbf{x}_i$. If all the points in $\mathcal{N}_b(p_i)$ are below this plane or all of them are over it, we assign p_i to the set of corner points $\mathcal{P}^c$; otherwise, we assign it to $\mathcal{P}^e$. This analysis does not consider all possible corner types; it focuses on salient corners, that is, peaks and valleys.

Owing to the noise in the point cloud, the edge lines can pass through the feature candidate points, allowing non-ideal points to be considered the closest. Figure 11 shows an example in which the points are projected onto a plane defined by $\mathbf{e}_{ij}$. The edge line between p_i and p_j is represented by a single point (red), which is the intersection of the corresponding planes $\mathscr{P}_i$ and $\mathscr{P}_j$, represented by black lines. We expect that p_i should be considered as the edge point in this case because the local neighborhood represents a convex surface. However, p_i is not the closest point to the edge line because of the noisy point positions and estimated normals. To avoid this problem, we estimate a more external edge line for convex regions and a more internal edge line for concave regions; that is, we apply a displacement of δ_{cc} to planes $\mathscr{P}_i$ and $\mathscr{P}_j$, following their normal directions for convex points and following their opposite normal directions for concave

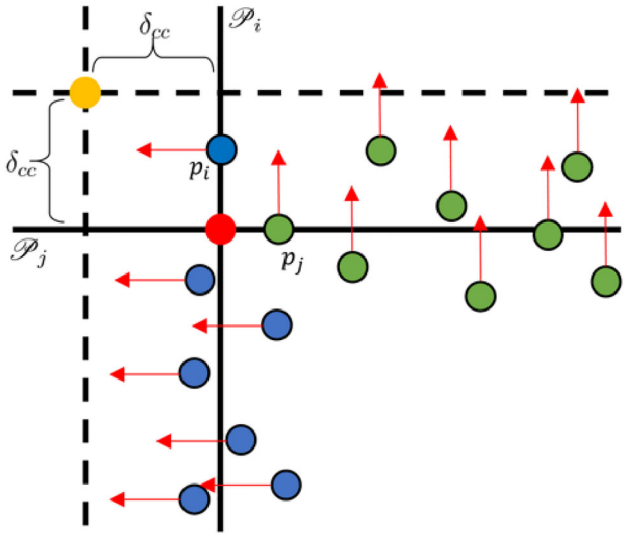


Fig. 11 Convexity-based displacement of $\mathscr{P}_i$ and $\mathscr{P}_j$. The blue points correspond to $\mathcal{C}_1$. The green points correspond to $\mathcal{C}_m$. The red arrows represent the point normals. The red point represents the edge line between $\mathscr{P}_i$ and $\mathscr{P}_j$. The yellow point represents the edge line after applying the displacement δ_{cc}. Note that p_i is not the closest point to the red point, but it is the closest to the yellow one

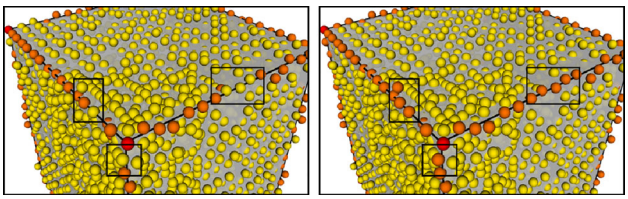


Fig. 12 Sharp feature detection on the Cube point cloud. Non-feature points are colored in yellow; edge points are colored in orange and corner points are colored in red. Left: result using $\delta_{cc} = 0$. Right: result using $\delta_{cc} = 0.5l_\mu$

points. The dashed lines in Fig. 11 represent the new plane positions and the yellow point represents the new edge line. p_i is the point closest to the edge line. Figure 12 shows an example of how this displacement affects the selection of the sharp feature points. The selection of an appropriate δ_{cc} value depends on the noise level. The greater the noise, the higher the value of δ_{cc}.

To define whether a point p_i is convex or concave, we first computed a smooth version of the point cloud using regular Laplacian smoothing on the Riemannian graph defined by $\mathscr{N}_s$. For this process, we used 10 smoothing iterations and 0.2 step size. This smooth representation enables the assessment of a cleaner local surface approximation for p_i. For each p_i, we compute the average *regular normal* $\overline{\mathbf{n}}_i$ of the neighboring points regarding $\mathscr{N}_s$ and the centroid $\mathbf{c}_i$ of the neighboring points regarding $\mathscr{N}_b$ using the point positions of the smooth representation. We then define the plane defined by p_i on the smooth surface and $\overline{\mathbf{n}}_i$ as $\mathscr{P}$. If $\mathbf{c}_i$ is at a distance greater than ϵ_{cc} from $\mathscr{P}$, is below it, we consider p_i, and is below it. If $\mathbf{c}_i$ is at a distance greater than ϵ_{cc} from $\mathscr{P}$ and is over it, we

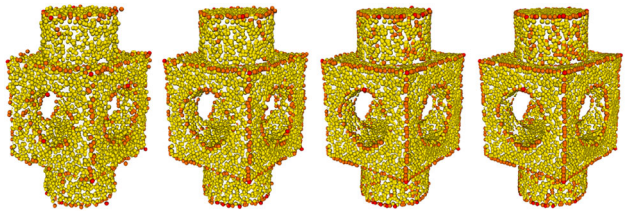


Fig. 13 Sharp feature detection on the Block point cloud through different denoising iterations. Non-feature points are colored in yellow, edge points are colored in orange, and corner points are colored in red. From left to right: first iteration, second iteration, third iteration, and fourth iteration

consider p_i to be concave. Otherwise, we consider p_i to be undefined, and no displacement is applied to $\mathscr{P}_i$ and $\mathscr{P}_j$. We introduced *big neighborhood* $\mathscr{N}_b$ for the computation of $\mathbf{c}_i$ because $\mathscr{N}_r$ can be too small for convexity analysis. If we increase the size of $\mathscr{N}_r$, we can disturb other procedures.

Figure 13 shows an example of the behavior of the sharp feature detection method for different iterations of the denoising algorithm. We can see that the feature points are precisely selected, even in the presence of high noise levels, and that they quickly begin converging to the edge lines of the underlying surface. In contrast to other methods, we do not require an explicit threshold parameter defining if a point is a feature because we select the closest points to the edge lines instead. In Sect. 9, we present examples of how the denoising algorithm benefits from this approach. The sharp feature detection procedure is summarized in Algorithms 2 and 3, where we assume that neighborhood clustering and convexity analysis are performed. In these algorithms, the function *intersection* returns the intersection line between two planes, function *angle* returns the minimum angle between two vectors, function *distance* returns the distance between a point and a line, function *projection* returns the projection of a point on a plane, and function *cornerCheck* represents the last filtering operation applied to the possible corner points.

8 Point updating

Once we have a set of filtered normals and the point classification, we update the point positions using an adaptation of the method proposed in [71], which applies a different point-updating scheme depending on the point class. For all updating operations, we consider the neighborhood $\mathscr{N}_p$, which denotes a small neighborhood without including p_i, that is, $\mathscr{N}_p(p_i) = \mathscr{N}_s(p_i) - \{p_i\}$. We first iteratively update non-feature points to update the edge and corner points simultaneously. As in [71], we constrained the possible displacement of each point to a Euclidean sphere with radius τ_o centered at the original noisy point position. In the following

Algorithm 2 Sharp feature detection

procedure SHARP FEATURE DETECTION($\mathcal{P}$,θ,δ_{cc})
  $\mathcal{P}^{nf} \leftarrow \mathcal{P}^{nfc}$, $\mathcal{P}^{e} \leftarrow \{\}$, $\mathcal{P}^{c} \leftarrow \{\}$
  for each $p_i \in \mathcal{P}^{ec} \cup \mathcal{P}^{cc}$ **do**
    $\mathcal{L} \leftarrow$ getNeighborhoodClusters(p_i)
    $C_1 \leftarrow \mathcal{L}[1]$
    if p_i is convex **then**
      $\delta \leftarrow \delta_{cc}$
    else if p_i is concave **then**
      $\delta \leftarrow -\delta_{cc}$
    else
      $\delta \leftarrow 0$
    end if
    $\mathscr{P}_i \leftarrow (\mathbf{x}_i + \delta \mathbf{n}_i, \mathbf{n}_i)$
    $feature\_count \leftarrow 0$
    for each $C_m \in \mathcal{L} - \{C_1\}$ **do**
      $is\_feature\_cluster \leftarrow True$
      for each $p_j \in \mathcal{C}_m$ **do**
        $\mathscr{P}_j \leftarrow (\mathbf{x}_j + \delta \mathbf{n}_j, \mathbf{n}_j)$
        $(\mathbf{e}_{ij}, \mathbf{x}_{ij}) \leftarrow$ intersection($\mathscr{P}_i$,$\mathscr{P}_j$)
        $\mathbf{e}_i^{\perp} \leftarrow \mathbf{n}_i \times \mathbf{e}_{ij}$, $\mathbf{e}_j^{\perp} \leftarrow \mathbf{n}_j \times \mathbf{e}_{ij}$
        $is\_feature\_cluster\_point \leftarrow True$
        $\mathcal{N}^{i}_{\text{narrow}} \leftarrow$ narrowNeighs(C_1,$\mathscr{P}_i$,$\mathbf{x}_i$,θ)
        $\mathcal{N}^{j}_{\text{narrow}} \leftarrow$ narrowNeighs(C_m,$\mathscr{P}_j$,$\mathbf{x}_i$,θ)
        $\mathcal{N}_{\text{narrow}} \leftarrow \mathcal{N}^{i}_{\text{narrow}} \cup \mathcal{N}^{j}_{\text{narrow}}$
        $d \leftarrow$ distance($\mathbf{x}_i$,($\mathbf{e}_{ij}$, $\mathbf{x}_{ij}$))
        for each $p_k \in \mathcal{N}_{\text{narrow}}$ **do**
          if distance($\mathbf{x}_k$,($\mathbf{e}_{ij}$, $\mathbf{x}_{ij}$))$< d$ **then**
            $is\_feature\_cluster\_point \leftarrow False$
            break
          end if
        end for
        if $is\_feature\_cluster\_point$ is $False$ **then**
          $is\_feature\_cluster \leftarrow False$
          break
        end if
      end for
      if $is\_feature\_cluster$ is $True$ **then**
        $feature\_count \leftarrow feature\_count + 1$
      end if
    end for
    if $feature\_count > 1$ **then**
      if cornerCheck(p_i) is $True$ **then**
        $\mathcal{P}^{c} \leftarrow \mathcal{P}^{c} \cup \{p_i\}$
      else
        $\mathcal{P}^{e} \leftarrow \mathcal{P}^{e} \cup \{p_i\}$
      end if
    else if $feature\_count == 1$ **then**
      $\mathcal{P}^{e} \leftarrow \mathcal{P}^{e} \cup \{p_i\}$
    else
      $\mathcal{P}^{nf} \leftarrow \mathcal{P}^{nf} \cup \{p_i\}$
    end if
  end for
  return ($\mathcal{P}^{nf}$, $\mathcal{P}^{e}$, $\mathcal{P}^{c}$)
end procedure

Algorithm 3 Narrow neighborhood

procedure NARROW NEIGHS($\mathcal{C}$,$\mathscr{P}$,$\mathbf{x}$,θ)
  $\mathcal{N}_{\text{narrow}} \leftarrow \{\}$
  $\mathbf{x}^{\pi} \leftarrow$ projection($\mathbf{x}$,$\mathscr{P}$)
  for each $p_k \in \mathcal{C}$ **do**
    $\mathbf{x}_k^{\pi} \leftarrow$ projection($\mathbf{x}_k$,$\mathscr{P}$)
    if angle($\mathbf{x}_k^{\pi} - \mathbf{x}^{\pi}$, $\mathbf{e}_j^{\perp}$) $\leq \theta/2$ **then**
      $\mathcal{N}_{\text{narrow}} \leftarrow \mathcal{N}_{\text{narrow}} \cup \{p_k\}$
    end if
  end for
  return $\mathcal{N}_{\text{narrow}}$
end procedure

section, we describe the updating scheme for each point class. Please refer to [71] for further details on updating operations,

Non-feature point updating for each point $p_i \in \mathcal{P}^{nf}$, the new position $\tilde{\mathbf{x}}_i$ is computed as follows:

$$\tilde{\mathbf{x}}_i = \mathbf{x}_i + \upsilon_{nf} \left(\frac{1}{W_p(p_i)} \sum_{p_j \in \mathcal{N}_p(p_i)} w_{ij} \langle \mathbf{n}_j^*, \mathbf{x}_j - \mathbf{x}_i \rangle \mathbf{n}_i \right), \tag{8}$$

where $w_{ij} = K_{ps}\left(\|\mathbf{x}_i - \mathbf{x}_j\|\right) K_{pn}\left(\|\mathbf{n}_i - \mathbf{n}_j^*\|\right)$, K_{ps} and K_{pn} are Gaussian kernel functions used to smooth spatial and normal distances, $W_p(p_i)$ is a normalization factor, i.e., the sum of all the used weights w_{ij}, the behavior of the kernels K_{ps} and K_{pn} is defined by the standard deviations σ_{ps} and σ_{pn}, respectively, $\mathbf{n}_j^* = \text{argmax}_{\mathbf{n}_k \in \mathcal{M}_j} K_{pn}\left(\|\mathbf{n}_i - \mathbf{n}_k\|\right)$, $\mathcal{M}_j$ denotes the set of multi-normals for the point p_j, computed during normal correction, and υ_{nf} controls the amount of displacement for each update. The multi-normals allow the usage of close edge or corner points whose main normal points in a different direction from $\mathbf{n}_i$. This updating operation is applied iteratively, where n_{nfp} denotes the number of iterations.

Edge point updating for each point $p_i \in \mathcal{P}^{e}$, the new position $\tilde{\mathbf{x}}_i$ is computed as follows:

$$\begin{aligned} \tilde{\mathbf{x}}_i = & \mathbf{x}_i + \upsilon_e \left(\left(\sum_{p_j \in \mathcal{N}_p(p_i)} \mathbf{n}_j^{\pi} \left(\mathbf{n}_j^{\pi}\right)^T + \mathbf{e}_i \mathbf{e}_i^T \right)^{-1} \right. \\ & \left. \left(\sum_{p_j \in \mathcal{N}_p(p_i)} \left(\mathbf{n}_j^{\pi} \left(\mathbf{n}_j^{\pi}\right)^T \mathbf{x}_j + \mathbf{e}_i \mathbf{e}_i^T \mathbf{x}_i \right) \right) - \mathbf{x}_i \right), \end{aligned} \tag{9}$$

where $\mathbf{n}_j^{\pi} = \mathbf{n}_j - \langle \mathbf{n}_j, \mathbf{e}_i \rangle \mathbf{e}_i$ and υ_e controls the amount of displacement.

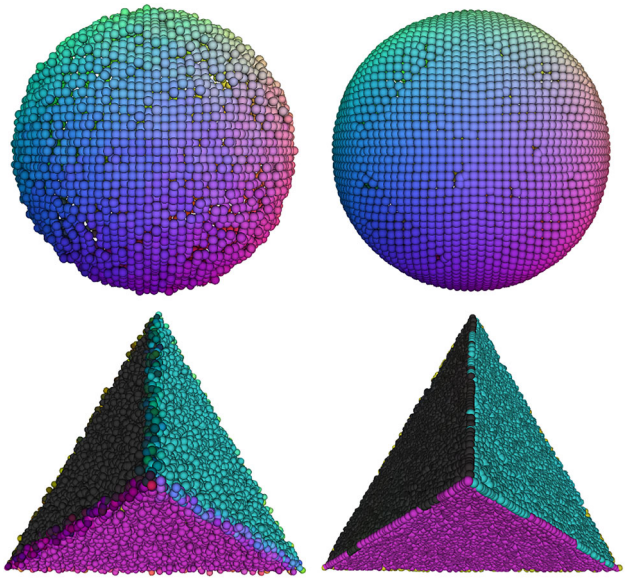


Fig. 14 Denoising examples on the sphere and tetrahedron point clouds. Left: noisy point clouds. Right: denoised point clouds using the proposed method. The point color maps the corresponding normal direction

Corner point updating for each point $p_i \in \mathcal{P}^c$, the new position $\tilde{\mathbf{x}}_i$ is computed as follows:

$$\tilde{\mathbf{x}}_i = \mathbf{x}_i + \upsilon_c \left(\left(\sum_{p_j \in \mathcal{N}_p(p_i)} \mathbf{n}_j \mathbf{n}_j^T \right)^{-1} \left(\sum_{p_j \in \mathcal{N}_p(p_i)} \mathbf{n}_j \mathbf{n}_j^T \mathbf{x}_j \right) - \mathbf{x}_i \right), \tag{10}$$

where υ_c controls displacement.

Figure 14 shows some simple examples using the proposed denoising method, and we can see that it is capable of dealing with smooth and sharp surfaces.

9 Results

We compared our method to point cloud denoising, normal estimation, and feature detection methods. We used seven non-uniform mesh models for the numerical evaluation, with the vertices acting as input point clouds and corrupted with synthetic noise (see Fig. 15). These synthetic point clouds are named Armadillo, Casting, Cube, Dragon, Fandisk, Octahedron, and RockerArm, and are corrupted with Gaussian noise in random directions with $\sigma = 0.5l_{\text{ro}}$, $\sigma = 0.5l_{\text{ro}}$, $\sigma = 0.3l_{\text{ro}}$, $\sigma = 0.5l_{\text{ro}}$, $\sigma = 0.28l_{\text{ro}}$, $\sigma = 0.3l_{\text{ro}}$, and $\sigma = 0.3l_{\text{ro}}$, respectively. These models are used in [66] and [71]. We used point clouds generated from raw scans of objects with sharp features for visual evaluation. We refer to them as shutter, iron, and tool, respectively. These data were used by [36]. We also consider the Gargoyle point cloud, which is a partial scan included in [71], the Building point cloud, which is a sampled point cloud from [58], the Twelve and Cube2 point clouds, which are included in [45], the Block model used in [38], the Mug model included in [63], and the BoxUnion point clouds used in [55].

9.1 Parameter setting

Although we introduce several parameters in our denoising pipeline, we define the following default values: $\epsilon_d = 0.01$, $k = 50$, $r_s = 1.5$, $r_r = 3$, $r_b = 2.5r_r$, $\alpha = 1$, $\beta = 0.1$, $\gamma = 0.5$, $a_0 = 0.35 \sum_{a_i \in \mathbf{a}} a_i$, $\tau_u = 0.3$, $\sigma_{\text{ns}} = 1.5l_\mu$, $\sigma_{\text{nn}} = 0.3$, $n_{ns} = 7$, $\tau_n = 0.2$, $\tau_{ta} = 0.2$, $n_{nc} = 2$, $\epsilon_{\text{mn}} = 0.1l_\mu$, $\theta = 110°$, $\delta_{cc} = 0.5l_\mu$, $\epsilon_{\text{cc}} = 0.2l_\mu$, $\tau_o = 2l_\mu$, $\sigma_{\text{ps}} = 2l_\mu$, $\sigma_{\text{pn}} = 0.5$, $\upsilon_{\text{nf}} = 0.3$, $n_{\text{nfp}} = 3$, and $\upsilon_e = \upsilon_c = 0.5$. These values were defined empirically by independently evaluating the corresponding operations on a set of test cases. For practical purposes, we define a subset of these parameters, including n_{int} and n_{ext}, which have a significant impact on the denoising task. These parameters and their recommendations are presented below.

The feature size of the *regular neighborhoods* are used in several operations of the pipeline, and their size is based on k and r_r. Depending on the point cloud feature sizes, r_r can be tuned to better preserve them. r_r defines the proportion of the average distance l_μ that is used to define the radius of the *regular neighborhood*. The smaller the feature size, the smaller the regular neighborhood. If we note that the small features are represented simply by immediate points, $r_r = 1.5$ can be the appropriate value. If a feature region requires several points to approximate its shape, we should consider a higher value for r_r. The Gargoyle point cloud, for example, displays very small features represented by a few points, whereas the Iron point cloud displays larger features represented by several points. Parameter k is used to define the maximum number of points to be included in the *regular neighborhoods*. Thus, the higher the value of r_r, the higher the value of k. In addition, consider k as a critical parameter regarding the execution time because it defines the complexity of the quadratic optimization problems used for anisotropic neighborhood computation.

The noise intensity n_{ext} and n_{int} control the number of iterations of the proposed algorithm. Ideally, we should fix $n_{\text{int}} = 1$ and tune n_{ext} only to execute all the algorithm steps for each iteration. However, applying the anisotropic neighborhood computation several times results in high execution times when the input point cloud contains a high number of points. Thus, n_{int} functions as a practical parameter to avoid multiple full-step executions. The tuning of these parameters is proportional to the amount of noise in the input; the more noise, the more iterations are required. In addition to

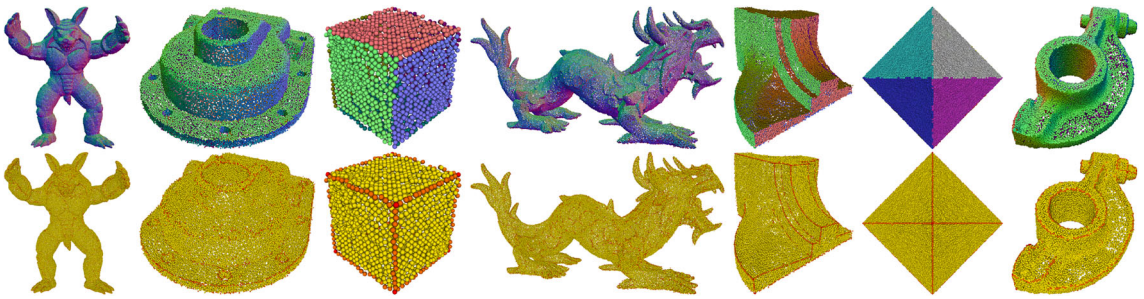


Fig. 15 Normal estimation (first row) and feature detection (second row) results obtained from the first iteration of our denoising pipeline. For normal estimation, the point color maps the normal direction. For feature detection, non-feature points are yellow, edge points are orange, and corner points are red. From left to right: Armadillo, Casting, Cube, Dragon, Fandisk, Octahedron, and RockerArm

n_{int} and n_{ext}, when the noise level is high, we should tune δ_{cc} and τ_o. As explained in the previous sections, when the noise level is high, the reference edge lines are inappropriate for sharp feature detection (see Fig. 11). Therefore, the higher the noise, the higher the value of δ_{cc}. When the noise level is high in the case of τ_o, some points are too far from their ideal positions. Thus, for these cases, we should increase the distance constraint to the original point positions introduced in the point-updating scheme, that is, τ_o. We recommend that δ_{cc} and τ_o depend on l_μ as many of the default values.

The sharpness n_{ns} and σ_{nn} allow us to control the smoothness of the normal field. n_{ns} denotes the number of smoothing iterations. For sharp feature preservation, we do not require many iterations. In case we are handling smooth features, we can increase the number of iterations to obtain a normal field that fits the smooth surfaces better, as shown in Fig. 4. Furthermore, we can increase the value of σ_{nn}, which defines the normal weight difference in the bilateral filter. As in several bilateral normal filtering schemes, the recommendation is to use σ_{nn} values close to 0.3 for sharp surfaces and σ_{nn} values close to 0.8 for smooth surfaces. It is important to note that if we oversmooth the normal field, sharp features will be lost. In addition to these parameters, τ_n is used as a feature threshold in the different steps of our algorithm. It defines an acceptable normal difference value to consider a region as a sharp feature candidate. Similar to the parameters used in the bilateral filter, for smooth surfaces, we can increase the value of τ_n to detect fewer features.

We did not see a significant improvement in the denoising task in our preliminary experiments despite the fact that the other parameters could be tweaked. Furthermore, many of them are independent of the evaluated point cloud, or their default values are based on relative measurements that make them robust.

The tuned parameters for each point cloud are described as follows: Armadillo: ($n_{\text{ext}} = 2$, $n_{\text{int}} = 1$, $r_r = 2.5$, $n_{\text{ns}} = 2$). Casting: ($n_{\text{ext}} = 2$, $n_{\text{int}} = 2$, $r_r = 2.5$, $n_{\text{ns}} = 2$). Cube: ($n_{\text{ext}} = 9$, $n_{\text{int}} = 1$). Fandisk: ($n_{\text{ext}} = 1$, $n_{\text{int}} = 5$). Dragon: ($n_{\text{ext}} = 2$, $n_{\text{int}} = 1$, $r_r = 2.5$, $n_{\text{ns}} = 2$). Octahedron: ($n_{\text{ext}} = 1$, $n_{\text{int}} = 10$). RockerArm ($n_{\text{ext}} = 1$, $n_{\text{int}} = 3$). Shutter: ($n_{\text{ext}} = 2$, $n_{\text{int}} = 5$). Iron: ($n_{\text{ext}} = 3$, $n_{\text{int}} = 5$, $k = 75$, $r_r = 4$). Tool: ($n_{\text{ext}} = 1$, $n_{int} = 5$). Gargoyle: ($n_{\text{ext}} = 1$, $n_{\text{int}} = 3$, $r_r = \{2, 2.5, 3\}$, $n_{\text{ns}} = 1$). For the block point clouds, we fixed $n_{\text{int}} = 1$, and n_{ext} was modified depending on the noise level. Default values were used when the parameter was not specified. The input normals for all cases were computed using the PCA-based normal estimation method (20 neighbors) implemented in the MeshLab software [14].

9.2 Normal estimation evaluation

Normal estimations can be compared in different ways. In this experiment, we adopted the feature-preserving evaluation method described in [75]. We can compute the per-point ground-truth multi-normals by assigning the normals of the clean faces shared by the corresponding vertex because synthetic noisy point clouds correspond to the perturbation of the vertices of a clean mesh. Then, given an estimated normal in the point cloud data, we can compare it with the most similar normal from the available ground-truth multi-normals. $RMSM_\tau$ allows us to measure the normal estimation error.

For our method, we selected the normal filtering results of the first iteration in the denoising pipeline. We compared these with the results of [12,32,44,51,75], which we refer to as PCA, JET, VCM, PCV, and DFP, respectively. For the PCA, JET, and VCM methods, we used the implementation provided in the CGAL library [62]. In the case of PCA and JET, we used 18 neighbors because they generated the best results for $RMSM_\tau$. Similarly, for the VCM method, we use $1.5l_\mu$ for both the offset and convolutional radii. For the PCV method, we used the implementation provided by the authors, considering $S^* = 100$ for all the point clouds, as in their experiments. In the case of the DFP method, we used the implementation of the authors and selected the predicted normals of the first iteration in their denoising pipeline. This implementation presents some limitations when processing

Table 1 Normal estimation results using $RMSM_\tau$

Method	Arma.	Cast.	Cube	Drag.	Fand.	Octa.	Rock.
PCA	0.46238	0.70172	0.89779	0.64127	1.32275	1.33386	**1.56447**
JET	0.47469	0.72091	0.91207	0.64531	1.32273	1.33400	1.56509
VCM	**0.44712**	0.62568	0.91020	**0.62283**	1.31394	1.33260	1.56515
PCV	0.54849	**0.51202**	0.06235	0.70066	1.26109	**1.30732**	1.56655
DFP	–	–	0.51596	–	1.29163	1.31418	1.56811
Ours	0.69567	0.62780	**0.03169**	0.81419	**1.24833**	1.30740	1.56730

The best results are highlighted in bold

Table 2 Feature detection accuracy results

Method	Arma.	Cast.	Cube	Drag.	Fand.	Octa.	Rock.
VCM	0.747	0.823	0.749	0.714	0.835	0.905	**0.857**
FREEUPC	**0.758**	0.756	0.844	**0.753**	0.789	0.915	0.839
DFP	–	–	0.724	–	0.784	0.879	0.713
Ours	0.757	**0.879**	**0.997**	0.687	**0.932**	**0.953**	0.811

The best results are highlighted in bold

Table 3 Denoising results using Gaussian noise

Method	Metric	Arma.	Cast.	Cube	Drag.	Fand.	Octa.	Rock.
APSS	D_p	0.17347	0.00033	0.01386	0.20402	0.00619	0.00117	0.07060
	$RMSM_\tau$	0.40497	0.66547	0.79453	0.63235	0.65859	0.44468	**1.56428**
RIMLS	D_p	**0.16603**	0.00030	0.01828	**0.18762**	0.00602	0.00101	**0.06017**
	$RMSM_\tau$	**0.36583**	0.59619	0.87196	**0.56784**	0.58685	0.44420	1.56535
MRPCA	D_p	0.17764	0.00034	0.01841	0.19699	0.00577	0.00084	0.07746
	$RMSM_\tau$	0.52573	0.82874	0.48890	0.62868	0.49439	0.31465	1.56547
DFP	D_p	–	–	0.01842	–	0.00711	0.00106	0.08189
	$RMSM_\tau$	–	–	0.50299	–	0.53786	0.36327	1.56814
CNVT	D_p	0.19440	0.00054	0.00645	0.27554	0.01083	0.00043	0.16511
	$RMSM_\tau$	0.93385	0.97103	0.48283	0.98734	0.38188	0.01685	1.56616
PCNET	D_p	0.19966	0.00037	–	0.23177	0.01072	0.00245	0.11312
	$RMSM_\tau$	–	–	–	–	–	–	–
PFILTER	D_p	0.26721	0.00036	–	0.35317	0.00568	0.00125	0.07893
	$RMSM_\tau$	–	–	–	–	–	–	–
SCORE	D_p	0.19138	0.00036	0.02004	0.21706	0.01125	0.00299	0.14737
	$RMSM_\tau$	–	–	–	–	–	–	–
Ours	D_p	0.16840	**0.00030**	**0.00385**	0.18958	**0.00308**	**0.00041**	0.07770
	$RMSM_\tau$	0.57709	**0.48298**	**0.00829**	0.72447	**0.23504**	**0.00997**	1.56678

The best results are highlighted in bold

point clouds with a high number of points (e.g., armadillo, dragon, and casting).

Table 1 lists the $RMSM_\tau$ results for the seven main synthetic point clouds. For the cube and Fandisk cases, our method outperformed the others. For the Octahedron, we obtained the best results with the PCV method. In the case of RockerArm, all methods achieved results with similar $RMSM_\tau$. VCM obtains the best results for Armadillo and Dragon and PCV for casting. It should be noted that there is no dominant method that achieves good performance in all test cases, and we believe that our algorithm parameters were tuned for the iterative denoising problem. The first row in Fig. 15 shows normal results. We observe that our normals are piecewise smooth, and the transitions are enhanced. In addition, these normals can maintain a certain smoothness at curved surfaces, as shown in the Fandisk model.

9.3 Feature detection evaluation

The feature detection problem can be modeled as a classification problem, and it can be evaluated using an accuracy metric. To generate ground-truth feature points on the syn-

thetic noisy point clouds, we select the corresponding vertex in the clean mesh for each point and check whether its shared face normals form an angle greater than 18°. If the latter occurs, the evaluated point is labeled as a feature point. This methodology for ground-truth feature point estimation was also used by [44].

We selected $\mathcal{P}^e \cup \mathcal{P}^c$ from the first iteration in our denoising pipeline as the detected features. We compared our results with those of [3,44,51], called VCM, FREEUPC, and DFP. We used the CGAL implementation of VCM with $3l_\mu$ as the offset radius and $1.5l_\mu$ as the convolutional radius. For the FREEUPC method, we used the implementation provided by the authors and the following parameters for each point cloud. Armadillo: ($k = 25$, $\sigma = 0.05$). Casting: ($k = 25$, $\sigma = 0.05$). Cube: ($k = 18$, $\sigma = 0.07$). Dragon: ($k = 25$, $\sigma = 0.05$). Fandisk: ($k = 25$, $\sigma = 0.05$). Octahedron: ($k = 25$, $\sigma = 0.05$). RockerArm: ($k = 25$, $\sigma = 0.05$). We tuned the parameters for both methods, VCM and FREEUPC, to achieve the highest accuracy. We chose the first iteration's feature detection results for the DFP method.

Table 2 presents the accuracy results. For the casting, cube, Fandisk, and Octahedron, our method outperforms the others, with accuracies higher than 0.87. In the case of Armadillo, our method presents an accuracy similar to that of the FREEUPC results. The VCM method achieved the highest accuracy for the RockerArm. Our method performs better when the features are sharp, which is not the case with this model. The second row of Fig. 15 shows the results. We can see that the detected feature regions tend to be thin, which is an important condition for our denoising pipeline to avoid the generation of gaps and excessive agglomeration of points at the underlying feature regions. Furthermore, as illustrated in Fig. 13, the estimated feature points were refined via denoising iterations. Other methods estimate thicker feature regions, which is undesirable for denoising algorithms.

9.4 Denoising evaluation

We used two different metrics to perform a numerical evaluation of denoising results on synthetic point clouds. First, we measure the double-sided average Euclidean distance D_p between the denoised point positions and ground-truth positions, that is, the positions of the original point cloud. Second, we use $RMSM_\tau$ to measure the normal error between the denoised normals of the last denoising iteration and the ground-truth multi-normals defined on the original point cloud. The closest points between the point sets are chosen to define their correspondence.

For synthetic point clouds, we compared our method with [25,44,48,49,53,55,71,74], namely APSS, RIMLS, MRPCA, DFP, CNVT, PCNET, PFILTER, and SCORE, respectively. For APSS and RIMLS we used the MeshLab implementation, whereas, for the others, we used the implementations provided by the corresponding authors. For the APSS, RIMLS, and CNVT, the parameters used on the cube, Fandisk, Octahedron, and RockerArm were the same as those described in [71]. In the case of the Armadillo, casting, and dragon, we use ($h = 4$, $n_{\text{its}} = 15$, $\alpha = 0$) for the APSS and ($\sigma_r = 4$, $\sigma_n = 0.75$) for the RIMLS. For the CNVT, we used ($\tau = 0.2$, $\rho = 0.9$, $p = 15$) for the Armadillo and Dragon, and ($\tau = 0.3$, $\rho = 0.9$, $p = 150$) for casting. The parameters for the MRPCA method were tuned to achieve the best results for D_p. These parameters are defined as follows: Armadillo: ($k = 30$, $\sigma = 15$, $r = 3$). Casting: ($k = 30$, $\sigma = 15$, $r = 3$). Cube: ($k = 30$, $\sigma = 15$, $r = 5$). Dragon: ($k = 30$, $\sigma = 15$, $r = 3$). Fandisk: ($k = 30$, $\sigma = 15$, $r = 3$). Octahedron: ($k = 30$, $\sigma = 15$, $r = 4$). RockerArm: ($k = 30$, $\sigma = 15$, $r = 3$). In the case of data-driven methods, that is, DFP, PCNET, PFILTER, and SCORE, we use the corresponding pre-trained models and the following number of iterations: 2, 1, 2, and 1. PCNET, PFILTER, and SCORE did not compute the final normal field. Therefore, we cannot compare them using $RMSM_\tau$.

Table 3 shows the D_p and $RMSM_\tau$ results for point clouds corrupted with Gaussian noise. Our method outperforms the others by considering both metrics for the cube and Octahedron. In Fig. 16, we can observe that the APSS, RIMLS, and MRPCA methods generate rounded regions at the corners and edges, but MRPCA produces sharper normal transitions at the edge regions. For some points, the DFP attempts to preserve the sharp features; however, the overall results appear unstable and noisy. In comparison to previous methods, the CNVT result is noise-free in flat regions and better preserves edges and corners. Nonetheless, it has gaps near the edges, and multiple points converge to the same position. This phenomenon occurred because the coarse feature points detected by CNVT were displaced to the edge lines. On the other hand, our method presents a clean surface, enhances sharp feature regions, and maintains a more uniform distribution, avoiding the generation of gaps and agglomeration of points at the underlying sharp feature regions.

In contrast to the cube and Octahedron, the casting and Fandisk present curved regions in addition to the flat and sharp regions. Our method also outperformed the others in these cases for both metrics, D_p and $RMSM_\tau$. Figure 17 shows the visual results for Fandisk, where we can see the same phenomena as in the previous cases for the APSS, RIMLS, MRPCA, and DFP methods. Additionally, the APSS and RIMLS generate false bumpy features. Although CNVT produces clean flat and curved regions, the edges are noisy. Our method is capable of preserving and enhancing sharp feature regions while keeping curved regions smooth.

In the case of Armadillo and Dragon, we obtained the best D_p results with RIMLS. However, RIMLS obtained considerably better $RMSM_\tau$ results. The latter occurs because these point clouds present several smooth features, which are not

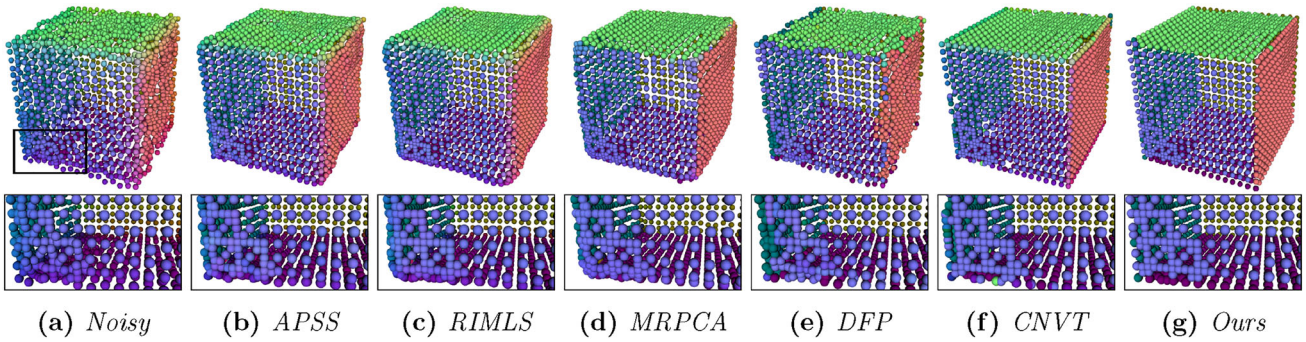


Fig. 16 Results obtained on the Cube point cloud. The point color maps the direction of the normals

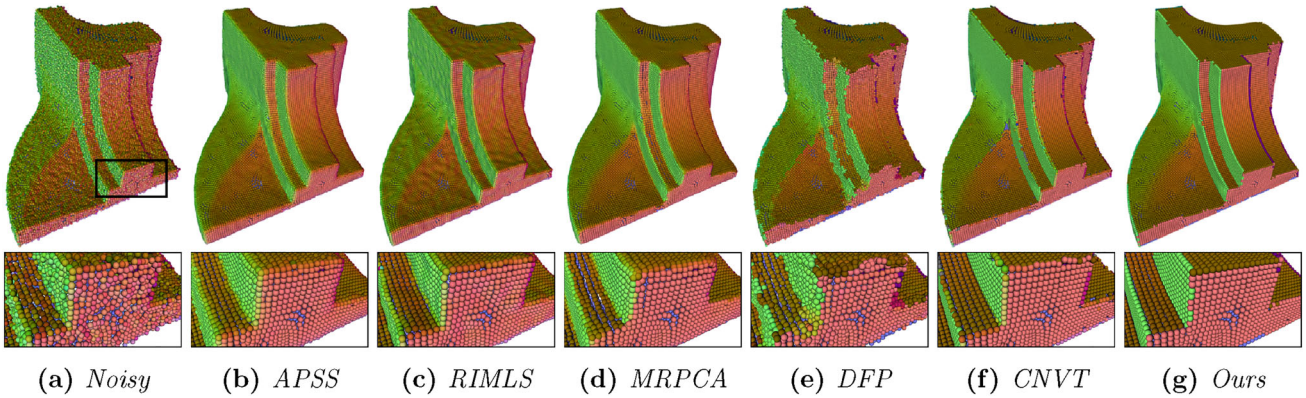


Fig. 17 Results obtained on the Fandisk point cloud. The point color maps the direction of the normals

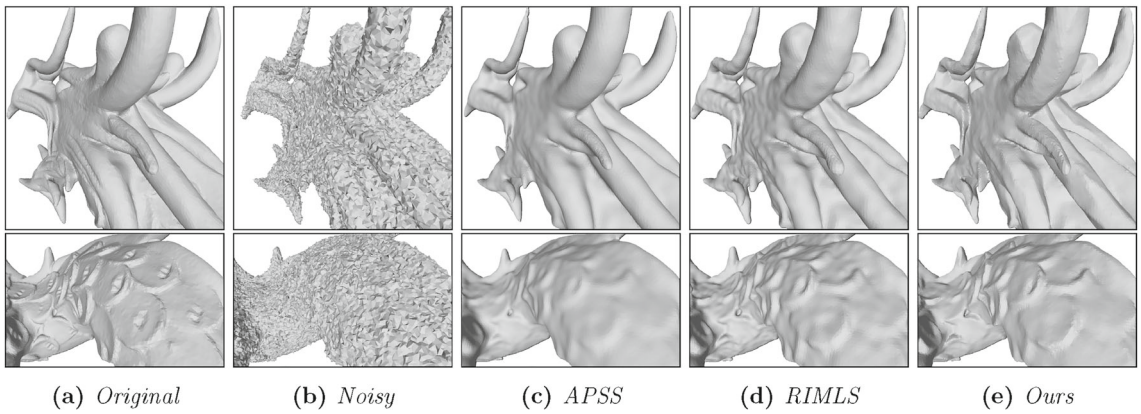


Fig. 18 Results obtained on the Dragon point cloud. For visualization purposes, we reconstruct mesh models using [39]. The models are rendered using flat shading. The APSS oversmooths the surface and the RIMLS generates bumpy regions

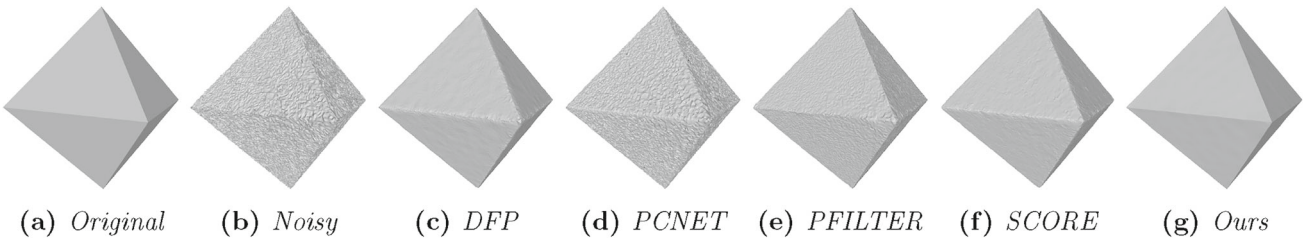
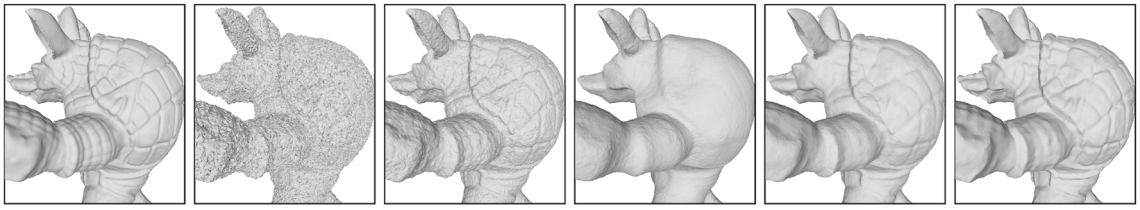


Fig. 19 Results obtained on the Octahedron point cloud. Comparison with data-driven methods. For visualization purposes, we reconstruct mesh models using [8]. The models are rendered using flat shading



(a) *Original* (b) *Noisy* (c) *PCNET* (d) *PFILTER* (e) *SCORE* (f) *Ours*

Fig. 20 Results obtained on the Armadillo. Comparison with data-driven methods. For visualization purposes, we reconstruct mesh models using [39]. The models are rendered using flat shading

the main focus of our method, and whose error contribution is dominant. Figure 18 shows the mesh reconstruction of the Dragon point cloud results using [39], where we can see that the RIMLS reconstruction presents the same bumpy artifacts as those shown in Fig. 17. In addition, we can see how APSS oversmooths the sharp feature regions. Our method preserves and enhances sharp features and does not introduce bumpy artifacts.

As mentioned previously, the features of RockerArm are not sharp. The RIMLS method achieved the best result for this case, and the CNVT was the worst regarding D_p. Our algorithm enhances the smooth features, generating sharp regions that are not present on the underlying surface. This effect may be expected, depending on the application. For example, a shape-compression algorithm may prefer well-defined transitions between piecewise smooth regions for surface splitting. Although we can tune other parameters to obtain better results regarding D_p and a smoother surface, we use default parameters to demonstrate their robustness against different examples.

Except for DFP, the rest of the data-driven methods do not use specific processing for sharp features. As a result, when compared to feature-preserving methods, they did not achieve good numerical results in our test cases. Figure 19 shows a visual comparison of data-driven methods on the Octahedron. We can see that our method outperforms these other methods in preserving sharp features. In addition, note that the other methods do not completely remove the noise. Similarly, Fig. 20 shows the results for Armadillo, where we can see how our method better preserves the details and removes noise efficiently. Data-driven methods strongly depend on the generality of the training dataset. Despite their efforts to learn how to reconstruct surface details, conventional geometry-processing methods work better for restoring sharp features.

To evaluate our method against impulsive noise, we corrupted synthetic point clouds using it. For each synthetic point cloud, 50% of the points are corrupted using $\sigma = 0.5l_{ro}$. For all methods, we used the same parameters as those used for the Gaussian noise test cases. Table 4 lists the corresponding D_p and $RMSM_\tau$ results. We can see similar behavior in Table 3, showing that our method is also robust to this type of noise. Figure 21 shows the casting results, where we can see how our method also preserves the sharp features and curved regions in the presence of impulsive noise. The APSS, RIMLS, and MRPCA oversmooth the sharp features, whereas the CNVT made feature points converge to edges that were not present on the underlying surface. Our method successfully removes both types of noise-Gaussian and impulsive-without the need for separate parameter tuning for each.

In addition to the synthetic examples, we visually evaluated our method on real scans of objects with sharp features. In this case, we compared our method to MRPCA and CNVT because they handle sharp features better. We also included the results of [43] (FPF) for the iron point cloud provided by the authors. In the case of MRPCA, we use the same parameters described in the paper for the Shutter and the Iron, and ($k = 30$, $\sigma = 15$, $r = 12$), ($k = 30$, $\sigma = 15$, $r = 3$), and ($k = 30$, $\sigma = 15$, $r = 3$), for the tool, Gargoyle, and Building, respectively. We used the same parameters described in the paper for Gargoyle and the following parameters for the other point clouds in the case of the CNVT. Shutter: ($\tau = 0.25$, $\rho = 0.9$, $p = 20$). Iron: ($\tau = 0.25$, $\rho = 0.9$, $p = 50$). Tool: ($\tau = 0.25$, $\rho = 0.9$, $p = 10$). Building: ($\tau = 0.25$, $\rho = 0.9$, $p = 20$).

Figure 22 shows the denoising results of the shutter point cloud. We can see that MRPCA generates a clean point cloud; however, it shrinks the surface and generates noisy and blurred edges. CNVT preserves edges better, but it accumulates several points close to them, resulting in gaps. Our method produced well-defined edges while leaving no gaps. Furthermore, our method can recover the flat surface in front of the object's base, which other methods ignore. It is worth noting that both the CNVT and our method retain a high degree of fidelity to the surfaces represented by the input point cloud.

Figure 23 shows the results for the iron point cloud, where, as in the other figures, the point color of the noisy point cloud maps the direction of the input normals. The noisy point cloud presents some outlier normals owing to the limitations of the method used to estimate the input normals. In this case, the

Table 4 Denoising results using impulsive noise

Method	Metric	Arma.	Cast.	Cube	Drag.	Fand.	Octa.	Rock.
APSS	D_p	0.08462	0.00017	0.01952	0.10261	0.00790	0.00210	**0.07670**
	$RMSM_\tau$	**0.41846**	0.68993	1.21323	**0.61602**	0.70473	1.16284	1.56538
RIMLS	D_p	**0.07333**	0.00014	0.02009	**0.09219**	0.00746	0.00188	0.09052
	$RMSM_\tau$	0.47149	0.77999	0.88884	0.63939	0.75351	0.71856	1.56600
MRPCA	D_p	0.09189	0.00022	0.01991	0.10950	0.00608	0.00111	0.08151
	$RMSM_\tau$	0.52812	0.87811	0.47087	0.62150	0.44516	0.30937	1.56590
DFP	D_p	–	–	0.02881	–	0.00898	0.00155	0.10911
	$RMSM_\tau$	–	–	0.68116	–	0.62392	0.37165	**1.34044**
CNVT	D_p	0.11571	0.00045	0.01258	0.20728	0.01371	**0.00067**	0.33049
	$RMSM_\tau$	0.94103	0.97949	0.53649	0.99562	0.92752	**0.01773**	1.56674
PCNET	D_p	0.13294	0.00027	–	0.17473	0.00936	0.00304	0.11642
	$RMSM_\tau$	–	–	–	–	–	–	–
PFILTER	D_p	0.22340	0.00025	–	0.32101	0.00643	0.00129	0.09045
	$RMSM_\tau$	–	–	–	–	–	–	–
SCORE	D_p	0.14118	0.00029	0.02740	0.17324	0.01219	0.00299	0.16904
	$RMSM_\tau$	–	–	–	–	–	–	–
Ours	D_p	0.08321	**0.00012**	**0.01065**	0.10241	**0.00486**	0.00073	0.10471
	$RMSM_\tau$	0.59775	**0.48930**	**0.01755**	0.73721	**0.24345**	0.02377	1.56888

The best results are highlighted in bold

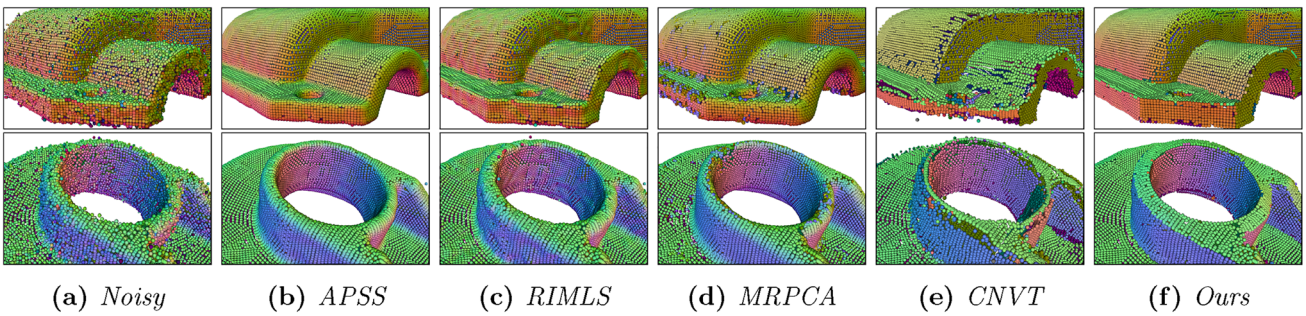


Fig. 21 Results obtained on the Casting point cloud, which is corrupted with impulsive noise. The point color maps the direction of the normals

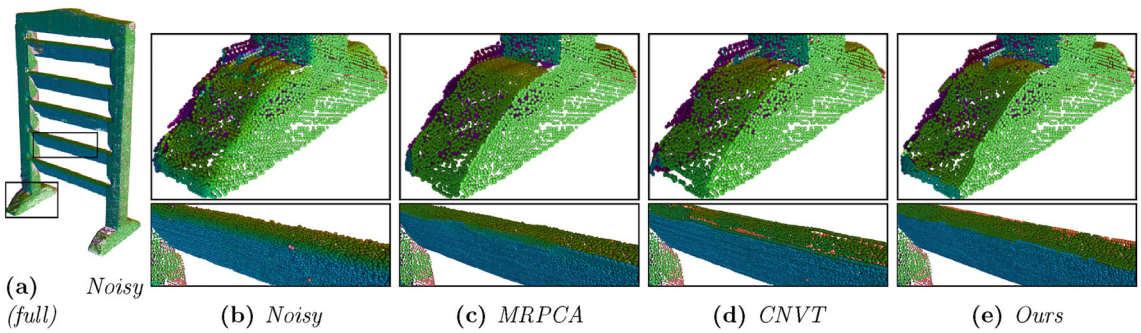


Fig. 22 Results obtained on the Shutter point cloud. The point color maps the direction of the normals

FPF does not properly remove the noise and does not enhance the edges. CNVT preserves some edges, whereas the others are blurred. In addition, the points that presented outlier normals were not treated. MRPCA generates piecewise smooth regions, even for curved regions, as shown in the first zoomed view. Although these regions enhance the features, the edges and corners are noisy and blurry. Our method works with flat and curved regions while maintaining sharp edges and corners. However, as shown in the second zoomed-in view, our method is also sensitive to normal outliers.

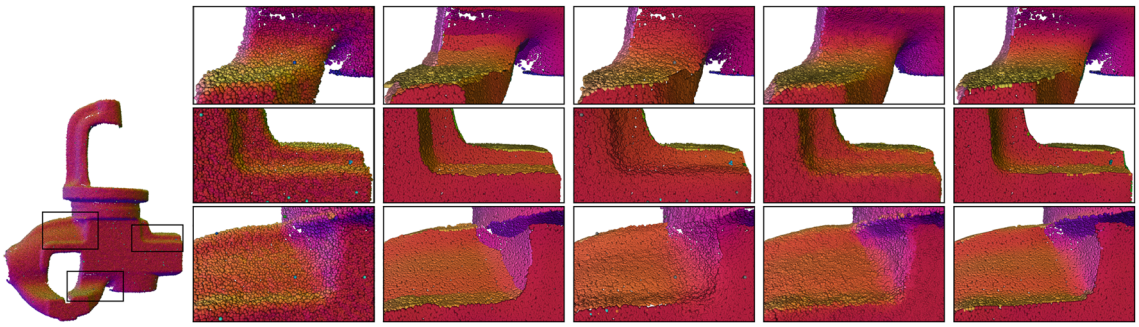


Fig. 23 Results obtained on the Iron point cloud. The point color maps the direction of the normals

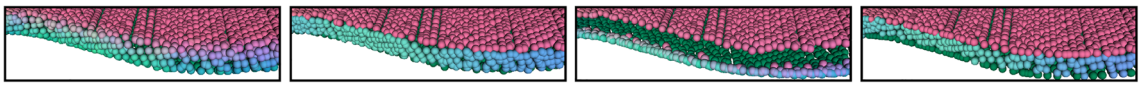


Fig. 24 Results obtained on the Tool point cloud. The point color maps the direction of the normals

The results for the tool point cloud are shown in Fig. 24, where we can emphasize the importance of precise sharp feature detection introduced in our denoising algorithm. The figure focuses on a point cloud region that presents two parallel sharp feature curves that are successfully recognized and reconstructed by the proposed method and MRPCA. However, in the CNVT result, both curves converge to a single curve generating undesired gaps and an incorrect shape. This phenomenon occurs because all points of the lateral face are classified as edge points by the CNVT. In contrast, the proposed sharp feature detection method can recognize both sharp feature curves and the flat region between them.

The Gargoyle point cloud shown in Fig. 25 presents features at different scales. We apply our method using different values of r_r, which determine the size of the regular neighborhoods. We can see that for $r_r = 2.0$, the method better preserves small features such as those located close to the neck. We blurred them when using $r_r = 3.0$ because more areas than needed were used to compute the anisotropic neighborhoods. Thus, r_r allows us to control the level of detail that needs to be recovered. For all the included r_r values, we can see that our method preserves the sharp features better than MRPCA and CNVT. In addition, the surface presents several holes because it is a partial scan. Our method tends to preserve the hole boundaries because the point displacements are based on local surface projections without including attractive or repulsive behavior.

Figure 26 shows the results for the building point cloud obtained using a LiDAR sensor. The three methods are sensitive to the noise generated by the distance bias error. This noise produces false salient regions, which are taken into account by the three methods. As a result, rough surfaces were generated for the building's lateral wall. Although this noise cannot be completely removed, it is effectively minimized. In the case of CNVT, we use a few filtering iterations to avoid the introduction of artifacts like gap generation and excessive point agglomeration. Therefore, the CNVT results present some noise. MRPCA generates rounded edge regions, whereas our method tends to generate sharp edge regions. Furthermore, our results present fewer bumpy artifacts than MRPCA on the front wall of the building. In this example, we can also see that our result presents a certain fidelity to the input point cloud, thereby preserving the hole boundaries.

Figure 27 shows a visual comparison of the LOP-based EAR [36] and GPF [45] on the 12 and Cube2 point clouds. For both methods, we use the parameters provided in [45]. Our method was applied to the following pipeline: First, we denoise the input point cloud using ($n_{\text{ext}} = 2, n_{\text{int}} = 5$). Second, we applied EAR upsampling using the same parameters as in [45]. Third, to correct upsampling issues, we denoise the upsampled cloud point using ($n_{\text{ext}} = 1, n_{\text{int}} = 5, \delta_{\text{cc}} = 3l_\mu, \tau_o = 4l_\mu$). It is observed that EAR produces noisy edges and GPF rounded edges. Furthermore, GPF expands the surface represented by the point cloud, resulting in undesirable curved regions. Our method generates clean point clouds with sharp features in the first step of the pipeline. These point clouds better approximated the ideal surface than the results of the EAR and GPF. However, some applications require dense and uniform sampling, such as the mesh reconstruction algorithm introduced by [8]. Thus, by applying EAR upsampling to our denoising results, we obtained densely and uniformly distributed point clouds. Because the normals that our method estimates are not consistent on the edges,

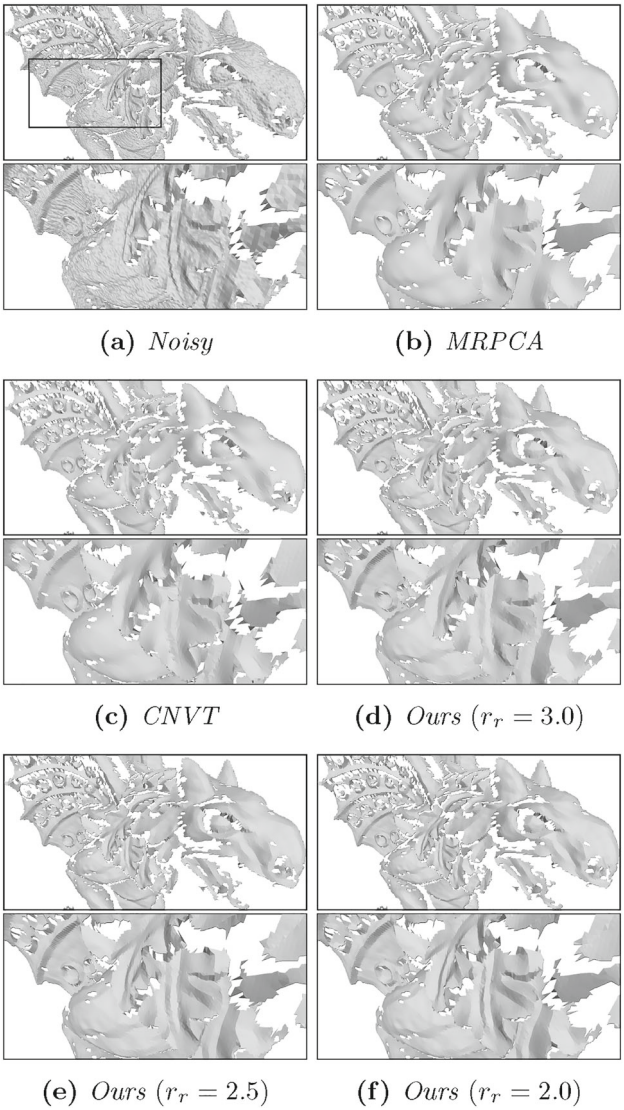


Fig. 25 Results obtained on the Gargoyle point cloud. For visualization purposes, we use the original triangulation of the noisy data as in [71]. The models are rendered using flat shading

that is, they can represent any of the shared faces, the EAR upsampling method can introduce artifacts. As a result, we used our method to remove them in a second filtering step, yielding high-quality point clouds suitable for a variety of applications that require dense and uniform sampling.

Figure 28 shows the results for the block point cloud corrupted with different noise intensities to demonstrate the robustness of our algorithm against different levels of noise. Our method yielded consistent results for different cases. Note that we only increased the number of external iterations n_{ext} while the noise level increased, and the rest of the parameters remained the same.

Our method can be used to remove geometric textures if we use sufficiently large regular neighborhoods and a certain normal field smoothness. Using the Mug mesh model, we sampled $140K$ points via Poisson disk sampling (MeshLab [14] implementation). We then used our method to remove the geometric texture from the sampled point cloud using the following parameters: ($n_{\text{int}} = 1, n_{\text{ext}} = 1, k = 75, r_r = 4, n_{\text{ns}} = 10, \sigma_{nn} = 0.8, \sigma_{\text{ns}} = 2l_\mu, n_{\text{fp}} = 50, \tau_o = 4l_\mu$). Figure 29 shows the result.

Figure 30 shows the limitations of the proposed method. In the presence of high noise levels, our method can create undesired artifacts such as those shown in Fig. 30a. Because of the noise, our method assumes small salient flat regions around the true surface. Furthermore, our method cannot denoise point clouds with multiple high-intensity outliers, as shown in Fig. 30b. Although the points closest to the target surface are denoised, our method uses outliers to create several floating islands around the object. When working with this type of noise, we recommend using an outlier removal algorithm as a pre-processing operation.

Our method can simultaneously deal with sharp and smooth features. However, in some cases, the introduced noise can generate regions that are similar to sharp edges. Our method can interpret this noise as a sharp feature and enhance it accordingly. Figure 30c, 30d, and 30e shows examples of this issue on the RockerArm point cloud. The original mesh model used to generate the synthetic point cloud, the noisy point cloud with the original triangulation, and the mesh model of our denoising result generated using [39] are shown in these figures. Note how nonexistent sharp regions are created in the resulting mesh. Our method is prone to over-sharpening the input point cloud in these cases.

Finally, Fig. 30f and 30g shows that our method is sensitive to topological issues. We show two surfaces that are close to each other. Our method interprets that edge lines are used to attract surrounding points at the intersection of these surfaces.

9.5 Execution time

Our denoising algorithm was implemented in C++ and uses CPLEX [15] for the numerical optimization. We measured the execution time on a 64-bit Intel(R) Core(TM) i7-8750H CPU 2.20 GHz with 32 GB RAM and a Windows 10 operating system. For the main denoising test cases, Table 5 shows the corresponding times in seconds. We included partial timings, percentages of the full execution time, and the number of executions for each step of the algorithm, including the preparation procedure.

The computation of anisotropic neighborhoods is the most computationally expensive step in our pipeline because several small quadratic optimization problems must be solved. However, because we constrained the maximum size of the optimization problems to the maximum size of regular neighborhoods, that is, k, this step presents a linear growth with

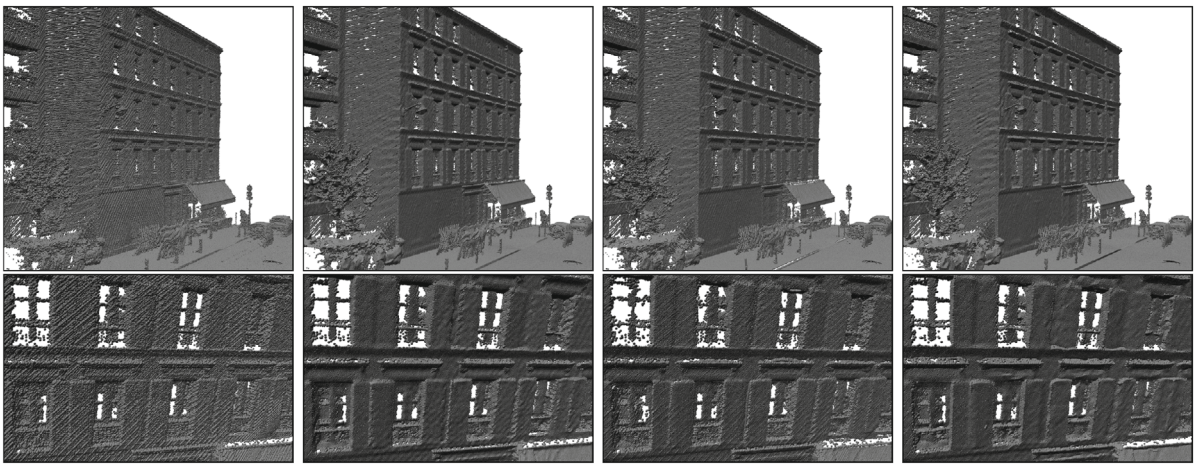


Fig. 26 Results obtained on the Building point cloud

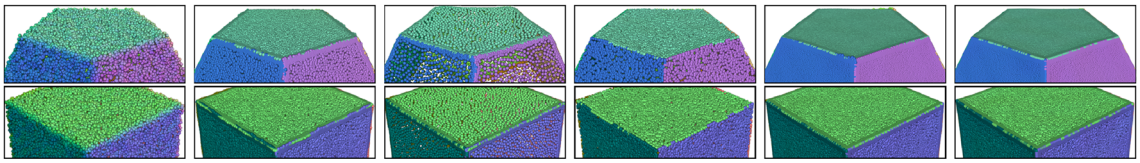


Fig. 27 Comparison with LOP-based methods. First row: Twelve point cloud. Second row: Cube2 point cloud. The point color maps the direction of the normals. The letter U represents the EAR upsampling and the letter C represents the correction using our denoising method

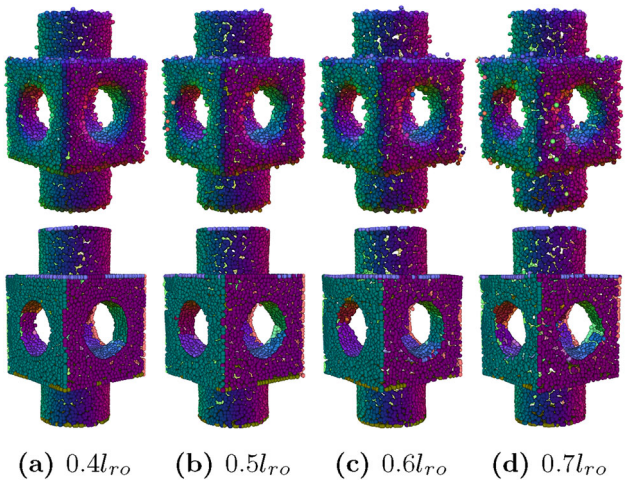


Fig. 28 The Block point cloud corrupted with different levels of noise. First row: noisy point clouds. Second row: our results. The point color maps the direction of the normals. The column captions denote the corresponding σ values

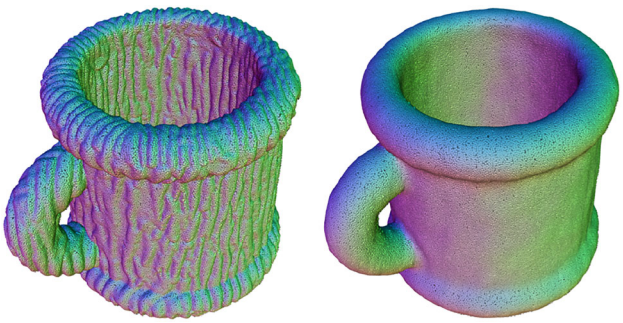


Fig. 29 Geometric texture removal applied on the Mug point cloud. Left: point cloud with geometric texture. Right: texture removal result using our method. The point color maps the direction of the normals

respect to the number of points. In the case of the iron point cloud, we increased k and r_r to obtain better visual results, resulting in a considerably higher computational time compared to $k = 50$ and $r_r = 3$.

To deduce the computational complexity of the denoising algorithm, we first analyzed the complexity of each step of the algorithm. The cost of the preparation procedure can be approximated by $\mathcal{O}(n \log n) + \mathcal{O}(nk \log k)$, where the dominant terms correspond to the spatial data structure indexation used to estimate the k nearest neighbors (i.e., a KD-Tree) and the local Delaunay triangulations used to define immediate neighbors. Notice that k should be lower than n or equal to n. The cost of the anisotropic neighborhoods computation can be approximated by $\mathcal{O}(nk^3)$. This term represents n quadratic optimizations of size k, assuming that the quadratic

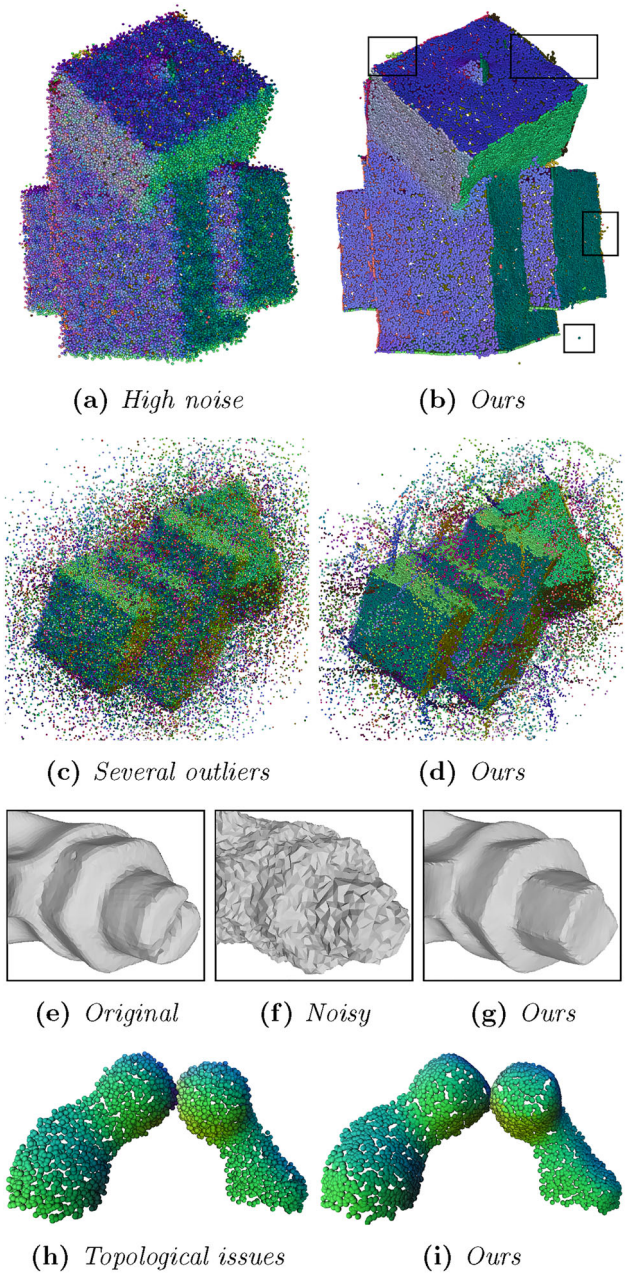


Fig. 30 Main limitations. First row: generation of artifacts when the noise intensity is high. Second row: unable to process wild high-intensity outliers. Third row: over-sharpening. Fourth row: sensitive to topological issues

programming algorithm cost can be approximated by $\mathcal{O}(k^3)$. However, in our implementation, this cost was determined by the proprietary algorithm used in the CPLEX library. The cost of the normal filtering step can be approximated by $n_{\text{ns}}\mathcal{O}(nk) + n_{\text{nc}}\mathcal{O}(nk^2)$, where the dominant terms represent normal smoothing and neighborhood clustering operations, respectively. The cost of the sharp feature detection step can be approximated by $\mathcal{O}(nk^2)$ because a paired analysis within the local neighborhood is required. The cost of the point-updating step can be approximated by $\mathcal{O}(nk)$.

The cost of the denoising algorithm can then be approximated by $n_{ext}(\mathcal{O}(n \log n) + \mathcal{O}(nk \log k) + \mathcal{O}(nk^3)) + n_{\text{ext}}n_{\text{int}}(n_{\text{ns}}\mathcal{O}(nk) + n_{\text{nc}}\mathcal{O}(nk^2))$. Thus, the computational complexity depends on the number of points in the point cloud, that is, n; the maximum number of points used to define the local neighborhoods, that is, k; and the number of iterations used in the algorithm, that is, n_{ext}, n_{int}, n_{ns}, and n_{nc}.

10 Conclusion and future work

We introduced a new method for point cloud denoising that includes solutions for normal estimation and feature detection problems. Numerous experiments demonstrated that our method outperforms state-of-the-art methods in processing point clouds with sharp features.

We presented an extension of the mesh-based anisotropic neighborhood computation introduced in [38] to the point clouds. This computation plays an imperative role in the denoising algorithm because it allows us to enhance the sharp feature regions by directly processing a set of noisy points and normals.

Combining anisotropic neighborhood normals with the proposed normal corrector operation allows us to avoid strong dependence on tuning α, β, and γ, identified as a critical task in [38]. Furthermore, as demonstrated in the experiments, most parameters can be adjusted without jeopardizing the generation of acceptable results.

We propose a novel sharp feature detection algorithm that helps the denoising algorithm reconstruct sharp feature regions without introducing undesired artifacts. In contrast to other algorithms, our algorithm works in a more selective manner, estimating thin sharp feature regions that work better on the point-updating scheme proposed in [71].

Our denoising method focuses on the preservation and enhancement of sharp features. Although we can tune the parameters to deal with smooth features, other methods such as RIMLS can perform better than ours.

A maximum of four clusters were used for regular neighborhood clustering. This limitation has no direct impact on the point-updating step, but it can introduce noise for normal correction and feature detection. In future work, we will introduce a more flexible clustering algorithm that allows us to represent a wider range of point types (e.g., nonmanifold corners), involving additional considerations for feature detection.

The computational cost of the proposed denoising algorithm is high, as demonstrated by the timing experiments. Future studies should aim to implement a faster version of

Table 5 Execution time in seconds of the proposed denoising algorithm on the main test cases

$\mathcal{P}$	n	Preparing procedure			Anisotropic Neigh.			Normal filtering			Feature detection			Point update			Full time
		Exec	Time	Perc (%)	Exec	Time	Perc (%)	exec	time	perc (%)	Exec	Time	Perc (%)	Exec	Time	Perc (%)	
Armadillo	172.9K	2	67.60	10.9	2	512.14	83.1	2	28.18	4.5	2	6.05	1.0	2	2.51	0.4	616.47
Casting	70.5K	2	28.98	10.5	2	218.63	79.5	4	20.51	7.4	4	4.95	1.8	4	2.05	0.7	275.13
Cube	1.9K	9	3.56	10.5	9	28.02	82.9	9	1.93	5.7	9	0.20	0.6	9	0.09	0.1	33.79
Dragon	151.5K	2	59.18	11.1	2	442.32	83.2	2	22.88	4.3	2	4.70	0.9	2	2.26	0.4	531.36
Fandisk	25.9K	1	4.99	7.8	1	47.68	74.5	5	9.29	14.5	5	1.35	2.1	5	0.71	1.1	64.03
Octahedron	40.2K	1	7.94	7.9	1	69.22	68.8	10	18.84	18.7	10	2.80	2.8	10	1.89	1.9	100.69
RockerArm	24.1K	1	4.59	7.1	1	47.96	74.1	3	10.17	15.7	3	1.48	2.3	3	0.53	0.8	64.73
Shutter	291.2K	2	129.55	6.9	2	1464.63	78.4	10	202.71	10.8	10	48.00	2.6	10	23.69	1.3	1868.58
Iron	161K	3	145.91	5.7	3	2055.44	80.7	15	265.00	10.4	15	61.59	2.4	15	19.84	0.8	2547.78
Tool	81.4K	1	16.60	7.8	1	162.46	76.6	5	25.99	12.2	5	4.81	2.3	5	2.34	1.1	212.19
Gargoyle	54.9K	1	11.41	8.4	1	108.04	79.4	3	13.57	10.0	3	1.94	1.4	3	1.04	0.8	136.00

this algorithm by replacing or improving the most expensive procedures, such as anisotropic neighborhood computation.

Acknowledgements We would like to express our gratitude to the National Council for Scientific and Technological Development (CNPq) and the Tecgraf Institute (PUC-Rio) for their support.

Declarations

Conflict of interest The authors declare that they have no conflict of interests.

Code availability The code and binary files are included in the supplementary materials. The code is also uploaded to a public GitLab repository.

References

1. Alexa, M., Behr, J., Cohen-Or, D., et al.: Computing and rendering point set surfaces. IEEE Trans. Vis. Comput. Graph. **9**(1), 3–15 (2003)
2. Avron, H., Sharf, A., Greif, C., et al.: ℓ_1-sparse reconstruction of sharp point set surfaces. ACM Transactions Graph. (TOG) **29**(5), 1–12 (2010)
3. Bazazian, D., Casas, J.R., Ruiz-Hidalgo, J.: Fast and robust edge extraction in unorganized point clouds. In: 2015 international conference on digital image computing: techniques and applications (DICTA), IEEE, pp 1–8 (2015)
4. Béarzi, Y., Digne, J., Chaine, R.: Wavejets: A local frequency framework for shape details amplification. In: Computer Graphics Forum, Wiley Online Library, pp 13–24 (2018)
5. Bellekens, B., Spruyt, V., Berkvens, R., et al.: A benchmark survey of rigid 3d point cloud registration algorithms. Int. J. Adv. Intell. Syst. **8**, 118–127 (2015)
6. Ben-Shabat, Y., Lindenbaum, M., Fischer, A.: Nesti-net: Normal estimation for unstructured 3d point clouds using convolutional neural networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp 10,112–10,120 (2019)
7. Berger, M., Tagliasacchi, A., Seversky, L.M., et al.: A survey of surface reconstruction from point clouds. In: Computer Graphics Forum, Wiley Online Library, pp 301–329 (2017)
8. Bernardini, F., Mittleman, J., Rushmeier, H., et al.: The ball-pivoting algorithm for surface reconstruction. IEEE Trans. Vis. Comput. Graph. **5**(4), 349–359 (1999)
9. Boulch, A., Marlet, R.: Deep learning for robust normal estimation in unstructured point clouds. In: Computer Graphics Forum, Wiley Online Library, pp 281–290 (2016)
10. Buades, A., Coll, B., Morel, J.M.: A non-local algorithm for image denoising. In: 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05), IEEE, pp 60–65 (2005)
11. Cao, C., Preda, M., Zaharia, T.: 3d point cloud compression: a survey. In: The 24th International Conference on 3D Web Technology, pp 1–9 (2019)
12. Cazals, F., Pouget, M.: Estimating differential quantities using polynomial fitting of osculating jets. Comput. Aid. Geometric Des. **22**(2), 121–146 (2005)
13. Chen, H., Wei, M., Sun, Y., et al.: Multi-patch collaborative point cloud denoising via low-rank recovery with graph constraint. IEEE Trans. Vis. Comput. Graph. **26**(11), 3255–3270 (2019)
14. Cignoni, P., Callieri, M., Corsini, M., et al.: MeshLab: an open-source mesh processing tool. In: Scarano V, Chiara RD, Erra U (eds) Eurographics Italian Chapter Conference. The Eurographics Association, https://doi.org/10.2312/LocalChapterEvents/ItalChap/ItalianChapConf2008/129-136(2008)
15. CPLEX IBM ILOG (2020) ILOG CPLEX Optimization Studio 20.1: User's manual for CPLEX. https://www.ibm.com/products/ilog-cplex-optimization-studio
16. Dabov, K., Foi, A., Katkovnik, V., et al.: Image denoising by sparse 3-d transform-domain collaborative filtering. IEEE Trans. Image Process. **16**(8), 2080–2095 (2007)
17. Deschaud, J.E., Goulette, F.: Point cloud non local denoising using local surface descriptor similarity. IAPRS. **38**(3A), 109–114 (2010)
18. Digne, J.: Similarity based filtering of point clouds. In: 2012 IEEE computer society conference on computer vision and pattern recognition workshops, IEEE, pp 73–79 (2012)
19. Digne, J., Valette, S., Chaine, R.: Sparse geometric representation through local shape probing. IEEE Trans. Vis. Comput. Graph. **24**(7), 2238–2250 (2017)
20. Dinesh, C., Cheung, G., Bajić, I.V., et al.: Local 3d point cloud denoising via bipartite graph approximation & total variation. In:

2018 IEEE 20th International Workshop on Multimedia Signal Processing (MMSP), IEEE, pp 1–6 (2018)
21. Dinesh, C., Cheung, G., Bajić, I.V.: Point cloud denoising via feature graph Laplacian regularization. IEEE Trans. Image Process. **29**, 4143–4158 (2020)
22. Duan, C., Chen, S., Kovacevic, J.: 3d point cloud denoising via deep neural network based local surface estimation. In: ICASSP 2019–2019 IEEE International Conference on Acoustics, pp. 8553–8557. Speech and Signal Processing (ICASSP), IEEE (2019)
23. Fleishman, S., Cohen-Or, D., Silva, C.T.: Robust moving least-squares fitting with sharp features. ACM Transactions Graph. (TOG) **24**(3), 544–552 (2005)
24. Foi, A., Dabov, K., Katkovnik, V., et al.: Shape-adaptive DCT for denoising and image reconstruction. In: Nasrabadi, N.M., Rizvi S.A., Dougherty E.R., Astola J.T., Egiazarian K.O. (eds.) Image Processing: Algorithms and Systems, Neural Networks, and Machine Learning. International Society for Optics and Photonics, vol. 6064, pp. 203–214 SPIE, (2006). https://doi.org/10.1117/12.64283
25. Guennebaud, G., Gross, M.: Algebraic point set surfaces. In: ACM SIGGRAPH 2007 papers. p 23–es (2007)
26. Guennebaud, G., Germann, M., Gross, M.: Dynamic sampling and rendering of algebraic point set surfaces. In: Computer Graphics Forum, Wiley Online Library, pp 653–662 (2008)
27. Guerrero, P., Kleiman, Y., Ovsjanikov, M., et al.: Pcpnet learning local shape properties from raw point clouds. In: Computer Graphics Forum, Wiley Online Library, pp 75–85 (2018)
28. Guillemot, T., Almansa, A., Boubekeur, T.: Non local point set surfaces. In: 2012 Second International Conference on 3D Imaging, Modeling, Processing, pp. 324–331. Visualization & Transmission, IEEE (2012)
29. Guo, M., Song, Z., Han, C., et al.: Mesh denoising via adaptive consistent neighborhood. Sensors **21**(2), 412 (2021)
30. Han, X.F., Jin, J.S., Wang, M.J., et al.: A review of algorithms for filtering the 3d point cloud. Signal Process. Image Commun. **57**, 103–112 (2017)
31. Hermosilla, P., Ritschel, T., Ropinski, T.: Total denoising: Unsupervised learning of 3d point cloud cleaning. In: Proceedings of the IEEE International Conference on Computer Vision, pp 52–60 (2019)
32. Hoppe, H., DeRose, T., Duchamp, T., et al.: Surface reconstruction from unorganized points. In: Proceedings of the 19th annual conference on computer graphics and interactive techniques, pp 71–78 (1992)
33. Hu, G., Peng, Q., Forrest, A.R.: Mean shift denoising of point-sampled surfaces. Vis. Comput. **22**(3), 147–157 (2006)
34. Hu, W., Gao, X., Cheung, G., et al.: Feature graph learning for 3d point cloud denoising. IEEE Trans. Signal Process. **68**, 2841–2856 (2020)
35. Huang, H., Li, D., Zhang, H., et al.: Consolidation of unorganized point clouds for surface reconstruction. ACM Transactions Graph. (TOG) **28**(5), 1–7 (2009)
36. Huang, H., Wu, S., Gong, M., et al.: Edge-aware point set resampling. ACM Transactions Graph. (TOG) **32**(1), 1–12 (2013)
37. Hurtado, J.: Detail-preserving mesh denoising using adaptive patches. Master's thesis, Pontifícia Universidade Católica do Rio de Janeiro, Brazil (2018)
38. Hurtado, J., Gattass, M., Raposo, A., et al.: Adaptive patches for mesh denoising. In: 2018 31st SIBGRAPI Conference on Graphics, Patterns and Images (SIBGRAPI), pp 1–8 (2018)
39. Kazhdan, M., Hoppe, H.: Screened Poisson surface reconstruction. ACM Transactions Graph. (ToG) **32**(3), 1–13 (2013)
40. Kivi, P.E., Mäkitalo, M.J., Žádník, J., et al. Real-time rendering of point clouds with photorealistic effects: a survey. IEEE Access 10:13,151–13,173 (2022)
41. Leal, E., Sanchez-Torres, G., Branch, J.W.: Sparse regularization-based approach for point cloud denoising and sharp features enhancement. Sensors **20**(11), 3206 (2020)
42. Lipman, Y., Cohen-Or, D., Levin, D., et al.: Parameterization-free projection for geometry reconstruction. ACM Transactions Graph. (TOG) **26**(3), 22 (2007)
43. Liu, Z., Xiao, X., Zhong, S., Wang, W., Li, Y., Zhang, L., Xie, Z.: A feature-preserving framework 1504 for point cloud denoising. Comput. Aid. Des. **127**, 102857 (2020). https://doi.org/10.1016/j.cad.2020.102857. https://www.sciencedirect.com/science/article/pii/S0010448520300506
44. Lu, D., Lu, X., Sun, Y., Wang, J.: Deep feature-preserving normal estimation for point cloud filtering. Comput. Aid. Des. **125**, 102860 (2020). https://doi.org/10.1016/j.cad.2020.102860. https://www.sciencedirect.com/science/article/pii/S0010448520300531
45. Lu, X., Wu, S., Chen, H., et al.: Gpf: Gmm-inspired feature-preserving point set filtering. IEEE Trans. Vis. Comput. Graph. **24**(8), 2315–2326 (2017)
46. Lu, X., Schaefer, S., Luo, J., et al.: Low rank matrix approximation for 3d geometry filtering. IEEE Transactions Vis. Comput. Graph. (2020b)
47. Luo, S., Hu, W.: Differentiable manifold reconstruction for point cloud denoising. In: Proceedings of the 28th ACM International Conference on Multimedia, pp 1330–1338 (2020)
48. Luo, S., Hu, W.: Score-based point cloud denoising. In: Proceedings of the IEEE/CVF International Conference on Computer Vision, pp 4583–4592 (2021)
49. Mattei, E., Castrodad, A.: Point cloud denoising via moving rpca. In: Computer Graphics Forum, Wiley Online Library, pp 123–137 (2017)
50. Mederos, B., Velho, L., de Figueiredo, L.H.: Robust smoothing of noisy point clouds. In: Proc. SIAM Conference on Geometric Design and Computing, Citeseer, p 2 (2003)
51. Mérigot, Q., Ovsjanikov, M., Guibas, L.J.: Voronoi-based curvature and feature estimation from point clouds. IEEE Trans. Vis. Comput. Graph. **17**(6), 743–756 (2010)
52. Nguyen, A., Le, B.: 3d point cloud segmentation: a survey. In: 2013 6th IEEE conference on robotics, automation and mechatronics (RAM), IEEE, pp 225–230 (2013)
53. Öztireli, A.C., Guennebaud, G, Gross M.: Feature preserving point set surfaces based on non-linear kernel regression. In: Computer Graphics Forum, Wiley Online Library, pp 493–501 (2009)
54. Preiner, R., Mattausch, O., Arikan, M., et al.: Continuous projection for fast l1 reconstruction. ACM Trans. Graph. **33**(4), 1–47 (2014)
55. Rakotosaona, M.J., La Barbera, V, Guerrero P, et al (2020) Pointcleannet: Learning to denoise and remove outliers from dense point clouds. In: Computer Graphics Forum, Wiley Online Library, pp 185–203
56. Rosman, G., Dubrovina, A., Kimmel, R.: Patch-collaborative spectral point-cloud denoising. In: Computer Graphics Forum, Wiley Online Library, pp 1–12 (2013)
57. Roveri, R., Öztireli, A.C., Pandele, I., et al.: Pointpronets: Consolidation of point clouds with convolutional neural networks. In: Computer Graphics Forum, Wiley Online Library, pp 87–99 (2018)
58. Roynard, X., Deschaud, J.E., Goulette, F.: Paris-lille-3d: a large and high-quality ground-truth urban point cloud dataset for automatic segmentation and classification. Int. J. Robotics Res. **37**(6), 545–557 (2018)
59. Sarkar, K., Bernard, F., Varanasi, K., et al.: Structured low-rank matrix factorization for point-cloud denoising. In: 2018 International Conference on 3D Vision (3DV), IEEE, pp 444–453 (2018)
60. Schoenenberger, Y., Paratte, J., Vandergheynst, P.: Graph-based denoising for time-varying point clouds. In: 2015 3DTV-Conference: the true vision-capture, transmission and display of 3D video (3DTV-CON), IEEE, pp 1–4 (2015)

61. Sun, Y., Schaefer, S., Wang, W.: Denoising point sets via l0 minimization. Comput. Aid. Geometric Des. **35**, 2–15 (2015)
62. The CGAL Project (2021) CGAL User and Reference Manual, 5.2.1 edn. CGAL Editorial Board, https://doc.cgal.org/5.2.1/Manual/packages.html
63. Thompson, E.M., Biasotti, S., Giachetti, A., et al.: Shrec 2020: retrieval of digital surfaces with similar geometric reliefs. Comput. Graph. **91**, 199–218 (2020)
64. Wang, J., Zhang, X., Yu, Z.: A cascaded approach for feature-preserving surface mesh denoising. Comput. Aid. Des. **44**(7), 597–610 (2012)
65. Wang, J., Xu, K., Liu, L., et al.: Consolidation of low-quality point clouds from outdoor scenes. In: Computer Graphics Forum, Wiley Online Library, pp 207–216 (2013)
66. Wang, P.S., Liu, Y., Tong, X.: Mesh denoising via cascaded normal regression. ACM Trans. Graph. **35**(6), 1–232 (2016)
67. Weber, C., Hahmann, S., Hagen, H., et al.: Sharp feature preserving mls surface reconstruction based on local feature line approximations. Graph. Models **74**(6), 335–345 (2012)
68. Wei, M., Chen, H., Zhang, Y., Xie, H., Guo, Y., Wang, J.: GeoDual-CNN: Geometry supporting dual convolutional neural network for noisy point clouds. IEEE Transact. Vis. Comput. Graph. (2021). https://doi.org/10.1109/TVCG.2021.3113463
69. Williams, R.M., Ilieş, H.T.: Practical shape analysis and segmentation methods for point cloud models. Comput. Aid. Geometric Des. **67**, 97–120 (2018)
70. Wu, S., Huang, H., Gong, M., et al.: Deep points consolidation. ACM Transactions Graph. (ToG) **34**(6), 1–13 (2015)
71. Yadav, S.K., Reitebuch, U., Skrodzki, M., et al.: Constraint-based point set denoising using normal voting tensor and restricted quadratic error metrics. Comput. Graph. **74**, 234–243 (2018)
72. Yu, L., Li, X., Fu, C.W., et al.: Ec-net: an edge-aware point set consolidation network. In: Proceedings of the European Conference on Computer Vision (ECCV), pp 386–402 (2018)
73. Zeng, J., Cheung, G., Ng, M., et al.: 3d point cloud denoising using graph laplacian regularization of a low dimensional manifold model. IEEE Trans. Image Process. **29**, 3474–3489 (2019)
74. Zhang, D., Lu, X., Qin, H., et al.: Pointfilter: point cloud filtering via encoder-decoder modeling. IEEE Trans. Visual Comput. Graph. **27**(3), 2015–2027 (2020)
75. Zhang, J., Cao, J., Liu, X., et al.: Multi-normal estimation via pair consistency voting. IEEE Trans. Vis. Comput. Graph. **25**(4), 1693–1706 (2018)
76. Zhang, W., Deng, B., Zhang, J., et al.: Guided mesh normal filtering. In: Computer Graphics Forum, Wiley Online Library, pp 23–34 (2015)
77. Zheng, Y., Li, G., Wu, S., et al.: Guided point cloud denoising via sharp feature skeletons. Vis. Comput. **33**(6–8), 857–867 (2017)
78. Zheng, Y., Li, G., Xu, X., et al.: Rolling normal filtering for point clouds. Comput. Aid. Geometric Des. **62**, 16–28 (2018)
79. Zhu L, Wei M, Yu J, et al (2013) Coarse-to-fine normal filtering for feature-preserving mesh denoising based on isotropic subneighborhoods. In: Computer Graphics Forum, Wiley Online Library, pp 371–380

Jan Hurtado is a researcher and developer at the Tecgraf Institute (PUC-Rio). He holds a Ph.D. and MSc in computer science from the Pontifical Catholic University of Rio de Janeiro (PUC-Rio), Brazil. He graduated in Computer Science from the National University of Saint Augustine (Peru). His research interests include geometric processing and analysis, image processing and analysis, computer graphics, and deep learning. He participated in various research and development projects at Tecgraf Institute in collaboration with General Electric, Petrobras, and SIDIA/Samsung.

Marcelo Gattass is a professor at PUC-Rio since 1992 and director of the Tecgraf Institute. His research focuses on computer graphics, specifically in the following areas: visualization , numerical simulation, augmented reality, geometric modeling, and computational vision. In addition, he applies these areas to projects in his laboratory, helping professionals from various sectors, such as medicine and oil engineering, tackle their problems. He holds an undergraduate degree (1975) and a master's degree (1977) in Civil Engineering from PUC-Rio, and a Ph.D. degree (1982) in Civil Engineering from Cornell University, USA.

Alberto Raposo is an Associate Professor in the Department of Informatics at PUC-Rio. His research focuses on computer graphics, virtual reality (VR), and machine learning. He applied these concepts to areas such as serious games, training simulations, and medical image analysis. The development was performed at the Tecgraf Institute for Technical and Scientific Software Development at PUC-Rio. His research has been applied to projects in various sectors, from petroleum engineering to medicine, and in some cases, via partnerships with large companies such as Petrobras and General Electric.