IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received May 2, 2022, accepted May 18, 2022, date of publication May 23, 2022, date of current version June 2, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3176961

Modified Bilateral Filter for Feature Enhancement in Mesh Denoising

HYEON-DEOK HAN[1] AND JONG-KI HAN[2]
[1]Department of Information and Communication Engineering, Sejong University, Gwangjin-gu, Seoul 05006, South Korea
[2]Department of Electrical Engineering, Sejong University, Gwangjin-gu, Seoul 05006, South Korea

Corresponding author: Jong-Ki Han (hjk@sejong.edu)

This work was supported by the Institute for Information and Communications Technology Promotion (IITP) by the Korean Government through the Ministry of Science and ICT (MSIT) under Grant 2017-0-00486.

ABSTRACT Mesh models resulting from scanners are inevitably noisy; hence, removing the noise in scanned meshes becomes an essential task in the services using three-dimensional mesh models. Filtering-based methods are simple but have some constraints in eliminating noise because they degrade the features in the mesh models while removing the noise. In this study, we design a feature enhancement filter that is combined with a conventional denoising filter to remove the noise while enhancing the features. The designed enhancement filter is applied only to the feature areas in the mesh model. Results from experiments on synthetic and natural scanned models validate that the proposed method can restore false features by integrating conventional filtering-based methods, and outperforms other state-of-the-art methods.

INDEX TERMS Bilateral filter, feature enhancement, mesh denoising, mesh normal filtering.

I. INTRODUCTION

As the demand for services based on three-dimensional (3D) mesh models increases, high-fidelity scanners have recently been developed. The mesh models resulting from scanners are inevitably noisy, even when high-fidelity scanners are used [1]. Therefore, removing the noise from the scanned meshes becomes an essential task in the services using 3D mesh models. The most challenging task in mesh denoising is to preserve features while noise is removed. In the literature, many algorithms have been introduced to remove undesired noise while preserving mesh features, such as sharp edges and corners. The algorithms are classified into isotropic and anisotropic methods, filtering-based and optimization-based algorithms, and data-driven methods, according to their core techniques [2], [8], [17], [24], [28].

In isotropic methods, surface geometry is not considered when noisy data is removed. Vollmer *et al.* [2] used Laplacian smoothing to remove noise without considering the preservation of sharp features. Taubin [3] proposed a non-shrinking and two-step smoothing method. Desbrun *et al.* [4] used mean curvature flow for surface fairing based on a simplified mass matrix. Liu *et al.* [5] introduced a smoothing approach that preserves the volume on triangular meshes. Nealen *et al.* [6] and Su *et al.* [7] proposed global systems of equations for the isotropic smoothing method.

The isotropic approaches do not consider features in 3D mesh models; whereas, recent researches have focused on anisotropic techniques, which remove the noise while preserving the feature data. Perona and Malik [8] presented an anisotropic diffusion algorithm, which is a partial differential equation (PDE)-based denoising method. Similarly, Clarenz *et al.* [9] proposed a diffusion tensor for the anisotropic diffusion method. Ohtake *et al.* [10], and Ohtake and E. Belyaev [11] introduced the anisotropic diffusion method using the denoised face normal, which is derived from a weighted sum of neighboring face normals. The level set surface model, based on the anisotropic diffusion of surface normal was proposed by Tasdizen *et al.* [12] to derive denoised mesh models. Hildebrandt *et al.* [13] presented the prescribed mean curvature-based surface evolution method to avoid volume shrinkage and preserve features. Anisotropic diffusion-based methods preserve features in 3D mesh models more efficiently than isotropic methods. However, when the 3D mesh models are contaminated with heavy noise, there is a limit to the noise the anisotropic methods can eliminate.

The filtering-based method to remove noise is one of the most studied techniques because it can eliminate noise effectively with low computational complexity. Because early methods directly filter the positions of vertices to remove

The associate editor coordinating the review of this manuscript and approving it for publication was Ikramullah Lali.

noise [14]–[16], these methods do not preserve features when the noise intensity is heavy. To overcome this problem, recent methods include steps for normal filtering and vertex updating. Zheng *et al.* [17] used bilateral filters to update normal filtering. Zhang *et al.* [18] proposed a joint bilateral filter, which is based on a well-designed guidance normal field. Later, Zhang *et al.* [19] introduced a scale-aware normal filter, which uses static and dynamic guidance. Yadav *et al.* [20] presented a normal filter in a statistics framework with the isotropic regularization factor. Arvanitis *et al.* [21] developed a coarse-to-fine framework based on graph spectral processing, and subsequently refined the initial output by iteratively smoothening face normals and vertices. Zhao *et al.* [22] proposed a normal filter with guidance normals computed by graph-cut method. Wang *et al.* [23] introduced an implicit global optimization function with a feature-aware trilateral filter for guidance.

Optimization-based methods introduce a hypothesis for an ideal denoised mesh, and formulate mesh optimization as a global energy function whose minimization satisfies this hypothesis. The optimization-based techniques assume features are sparse on a clean mesh. He and Schaefer [24] introduced L_0 optimization-based algorithm, which involves the design of an edge operator for arbitrary triangular meshes, and the algorithm maximizes flat regions piecewise to remove noise. The algorithm shows robust performance even in the presence of heavy noise. However, the highly non-convex nature of L_0 optimization makes the calculation more complex; moreover, a piece-wise flat effect is seen to occur in this method. Wang *et al.* [25] constructed a weight L_1 optimization method to solve these problems. Zhang *et al.* [26] proposed filtered face normals, which use the total variation of face normals to measure sharp feature sparsity. These optimization-based methods are suitable only for simple models because they are based on the hypothesis that the mesh consists of plain regions.

Recently, various studies based on data-driven methods have also been conducted. A data-driven method was proposed by Diebel *et al.* [27]. They formulated a pairwise normal potential function for learning, which considers application-specific geometry shapes. More recently, Wang *et al.* [28] used local geometry descriptors as inputs for training neural networks. The network consists of single-hidden layer feed forward networks in a cascaded way. Wang *et al.* [29] proposed a similar design that uses local geometry descriptors, where a second normal estimation is utilized to recover features lost during the first step. Li *et al.* [30] constructed a convolution neural network based on a non-local similarity approach. Zha *et al.* [31] also proposed a CNN-based method, including a cascaded structure. The performance of the data-driven methods depends on the quality of the training dataset. Furthermore, the computation cost of the training process for these methods is high.

In addition to the aforementioned classified methods, there are other methods that use a normal voting tensor [32], [34], classify feature areas [32], [33], and use non-local similarity [35], [36]. Fan *et al.* [32] used the eigenanalysis of a normal voting tensor (NVT) to separate meshes into feature and non-feature regions, where denoising algorithms are applied to each region differentially. Bian and Tong [33] separated meshes using the k-means clustering algorithm. Yadav *et al.* [34] used element-based NVT (ENVT), which is a method similar to [32] and [33], but does not include a process of dividing the mesh area into smaller regions. In this algorithm, mesh denoising is performed using binary optimization on the eigenvalues of the ENVT. In the non-local similarity method, Wei *et al.* [35] and Li *et al.* [36] proposed low-rank matrix recovery by co-filtering similar patches. Rudin *et al.* [37] first introduced total variation (TV) regularization for 2D image denoising. Owing to its good edge-preserving property, it has many extensions for 2D image restoration and denoising [38], [39] as well as for mesh denoising [40]. Zhang *et al.* [40] combined TV and anisotropic Laplacian regularization. Duchamp and Stuetzle [41] proposed an extension of spline smoothing using a finite element method where sharp features are not preserved. By contrast, the surface reconstruction technique [42] based on splines shows impressive performance.

Among the aforementioned conventional algorithms, filtering-based method is one of the simplest algorithms to eliminate noise. However, the method may fail to preserve features during noise removal, because it applies the denoising filter several times to mesh data. To overcome this problem, we design an efficient enhancement filter, which can be jointly used with filtering-based methods. The combined method removes the noise corruption in mesh data, while enhancing the features in the mesh structure.

This paper is organized as follows. We review the related works in Section II. In Section III, we explain the motivation for our algorithm. In Section IV, we propose an efficient algorithm to remove the noise corruption in mesh data while enhancing the mesh features. The performance of the proposed algorithm is evaluated and compared with those of various conventional methods in Section V. Finally, we present our conclusions in Section VI.

II. RELATED WORKS

This section describes filtering-based methods, which are simple although they may degrade feature information during removal of noise in the mesh structure.

Most recent filtering-based methods, [17], [18], [20], and [43], consist of two-step processes, which modify face normals in the first step and then update vertices according to the modified face normals. Sun *et al.* [43] classified two-step filtering-based methods into one-stage and two-stage techniques. In one-stage techniques, the two steps are coupled as a pair to give a single overall iteration procedure, whereas two separate iteration phases are performed in two-stage techniques.

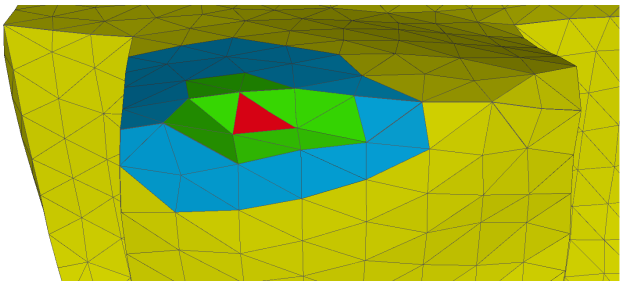

FIGURE 1. Example of f_i, Φ_i, and Ω_i.

A. NOTATION

In this section, we define some notations that are used in filtering-based methods. $\mathbb{F} = \{f_1, f_2, \ldots, f_N\}$ is a set of faces, where f_i is the i^{th} face of the mesh. $\mathbb{V} = \{\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_M\}$ is the set of coordinates of vertices. $\mathbb{N} = \{\boldsymbol{n}_1, \boldsymbol{n}_2, \ldots, \boldsymbol{n}_N\}$ is the set of normal vectors, where $\boldsymbol{n}_i$ is the normal vector of f_i. The center coordinate and area of f_i are denoted by $\boldsymbol{c}(f_i)$ and A_i, respectively. ∂f_i denotes the set of edges that constitute the boundary of f_i. We denote $N_{\boldsymbol{v}}(i)$ as the 1-ring vertex neighborhood of a vertex $\boldsymbol{v}_i$. Similary, the set of faces that share a vertex $\boldsymbol{v}_i$ is denoted by $F_{\boldsymbol{v}}(i)$.

For a f_i, the set of 1-ring neighbor faces is defined as follows:

$$\Phi_i = \{f_k \mid 1 - ring\ neighbor\ with f_i\}. \tag{1}$$

The set of neighbor faces for f_i is defined as follows:

$$\Omega_i = \left\{f_j \mid \left\|\boldsymbol{c}(f_i) - \boldsymbol{c}(f_j)\right\| < r\right\}. \tag{2}$$

where r is the radius determined by a user or is a fixed value. Fig. 1 shows an example of f_i, Φ_i, and Ω_i, where red face is f_i, green faces are included in Φ_i. Ω_i consists of blue and green faces.

B. BILATERAL NORMAL FILTER

The bilateral normal filter (BNF) method is a two-stage technique, which updates normals *Niter* times and then updates vertices *Viter* times. The process of the denoising algorithm using the BNF can be represented as $(update\ normals)^{Niter} + (update\ vertices)^{Viter}$. The BNF updates the normal vector using the weighted average of noisy neighbor normal vectors as follows:

$$\tilde{\boldsymbol{n}}_i = \frac{1}{K_i} \times \sum\nolimits_{f_j \in \Phi_i} A_j W_C\left(\left\|\boldsymbol{c}(f_i) - \boldsymbol{c}(f_j)\right\|, \sigma_c\right) W_S\left(\left\|\boldsymbol{n}_i - \boldsymbol{n}_j\right\|, \sigma_s\right) \boldsymbol{n}_j \tag{3}$$

where K_i is the normalization factor. σ_c and σ_s control the kernel widths of Gaussian functions W_C and W_S, respectively, as follows.

$$W_C\left(\left\|\boldsymbol{c}(f_i) - \boldsymbol{c}(f_j)\right\|, \sigma_c\right) = \exp\left(-\frac{\left\|\boldsymbol{c}(f_i) - \boldsymbol{c}(f_j)\right\|^2}{2\sigma_c^2}\right), \tag{4}$$

$$W_S\left(\left\|\boldsymbol{n}_i - \boldsymbol{n}_j\right\|, \sigma_s\right) = \exp\left(-\frac{\left\|\boldsymbol{n}_i - \boldsymbol{n}_j\right\|^2}{2\sigma_s^2}\right). \tag{5}$$

Function W_C gives small weight when the distance between the centroids (i.e. the geometric centers) of f_i and neighboring faces is large. Function W_S gives more weight as the similarity between $\boldsymbol{n}_i$ and neighboring normals increases.

C. GUIDED NORMAL FILTER

The guided normal filter (GNF) method is a modified one-stage technique. The process by which the noise removal algorithm uses the GNF method can be represented as $\left(update\ normals + (update\ vertices)^{Viter}\right)^{iter}$. The updated normal vector is derived as follows:

$$\tilde{\boldsymbol{n}}_i = \frac{1}{K_i} \times \sum\nolimits_{f_j \in \Omega_i} A_j W_C\left(\left\|\boldsymbol{c}(f_i) - \boldsymbol{c}(f_j)\right\|, \sigma_c\right) W_S\left(\left\|\boldsymbol{g}_i - \boldsymbol{g}_j\right\|, \sigma_s\right) \boldsymbol{n}_j \tag{6}$$

where $\boldsymbol{g}_i$ is a guided normal vector of f_i and is designed to be robust to noise. For a given face f_i, a patch is selected among a set of candidate patches that contain the face. The selected patch has the most consistent normal directions. The average normal of the chosen patch is used as the guidance normal $\boldsymbol{g}_i$ for the face f_i. The GNF preserves features better than BNF because $\boldsymbol{g}_i$ is more robust to noise than $\boldsymbol{n}_i$ [18]. Notably, Ω_i is used in GNF whereas faces in Φ_i are considered in BNF.

D. ROBUST AND HIGH-FIDELITY FILTER

The robust and high-fidelity filter (RHF) method is a one-stage technique that has one iteration parameter. The process of the RHF method can be represented as $(update\ normals + update\ vertices)^{iter}$. The RHF updates normal vectors in a similar manner as BNF:

$$\tilde{\boldsymbol{n}}_i = \frac{1}{K_i} \times \sum\nolimits_{f_j \in \Omega_i} A_j W_C\left(\left\|\boldsymbol{c}(f_i) - \boldsymbol{c}(f_j)\right\|, \sigma_c\right) W_T\left(\left\|\boldsymbol{n}_i - \boldsymbol{n}_j\right\|, \sigma_s\right) \boldsymbol{n}_j \tag{7}$$

where Tukey's bi-weight function is used instead of Gaussian function W_S as follows.

$$W_T\left(\left\|\boldsymbol{n}_i - \boldsymbol{n}_j\right\|, \sigma_s\right) = \begin{cases} \left(1 - \left(\dfrac{\left\|\boldsymbol{n}_i - \boldsymbol{n}_j\right\|}{\sigma_s}\right)^2\right)^2 & if\ \left\|\boldsymbol{n}_i - \boldsymbol{n}_j\right\| \le \sigma_s \\ 0 & otherwise \end{cases} \tag{8}$$

Tukey's bi-weight function preserves the features better than Gaussian function because it does not allow diffusion across sharp features. The RHF method removes noise components along sharp features using Tukey's bi-weight function.

E. VERTEX POSITION UPDATING

The two-step techniques, including BNF and GNF, update the coordinates of the vertices using the method proposed by Sun *et al.* [43]. They constructed an error function using the fact that ∂f_i must be perpendicular to the updated

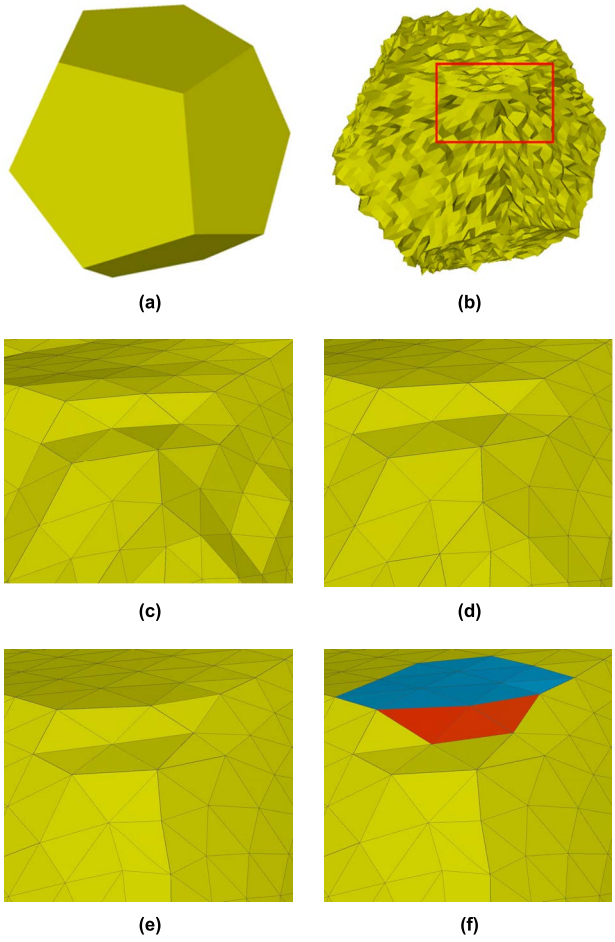

FIGURE 2. Example of areas where features have not been preserved. (a) is the ground-truth mesh, (b) is noisy mesh. (c)-(e) are results in 5, 10, and 20 iterations of RHF. (f) is colored version of (e).

normal $\tilde{\boldsymbol{n}}_i$. Therefore, to minimize the error function, the gradient descent method is used as follows.

$$\tilde{\boldsymbol{v}}_i = \boldsymbol{v}_i + \frac{1}{3\left|F_{\boldsymbol{v}}\left(i\right)\right|} \times \sum_{j\in N_{\boldsymbol{v}}(i)} \sum_{(i,j)\in\partial f_k} \tilde{\boldsymbol{n}}_k \left(\tilde{\boldsymbol{n}}_k \cdot \left(\boldsymbol{v}_j - \boldsymbol{v}_i\right)\right). \quad (9)$$

RHF also updates the coordinate of vertices using this concept; however, this involves an additional term, $\lambda \boldsymbol{R}_i$. The term $\boldsymbol{R}_i$ is a combination of the differential coordinate and tangential component. It makes the algorithm robust against high-intensity noise. Because term $\boldsymbol{R}_i$ has a similar effect to isotropic smoothing, using the same λ value in all iterations causes a shrinkage problem. To overcome this problem, the value of λ decreases as the iterations increase.

III. MOTIVATION

Fig. 2 shows the results of using the RHF method on the Twelve model over several iterations. Figs. 2a and 2b represent the original and the noisy models, respectively. Figs. 2c, 2d, and 2e are enlarged views of the red border region of Fig. 2b. They show results after 5, 10, and 20 iterations respectively. From Figs. 2c, 2d, and 2e, we observe that features are not preserved during all iterations. This is because some false features are regarded as features. Fig. 2f is the colored version of Fig. 2e, where the faces, including the false features and correct normals, are painted red and blue, respectively.

BNF, GNF, and RHF methods give high weights to neighboring normal, which are similar to the current normal in (3), (6), and (7), respectively. Thus, when algorithms are applied to one of the normals in the red area of Fig. 2f, high weights are assigned to only the neighboring normals in the red area. As a result, even if filtering is performed several times, the noisy normals cannot be corrected. This problem is due to using the distance between $\boldsymbol{n}_i$ and $\boldsymbol{n}_j$ in the weighting functions of (3), (6), and (7).

In this paper, we propose an efficient filtering process for normal vectors to solve the aforementioned problem. The basic idea of the proposed algorithm is to recover features using the normals in the blue areas when filtering the normals in the red areas in Fig. 2f. The proposed algorithm consists of denoising and feature enhancement modules, where the denoising part is implemented by a conventional algorithm, such as BNF, GNF, and RHF, based on (3), (6), and (7), respectively. The enhancement module is a form of the filter-based technique that can be used to fix noisy features. When the proposed enhancement filter is applied to the normal vector $\boldsymbol{n}_i$ of the current face f_i, we assign the high-weighted values to the normal vectors of the neighbor faces, which have the feature property. Details of the proposed algorithm and the enhancement module are described in Section IV.

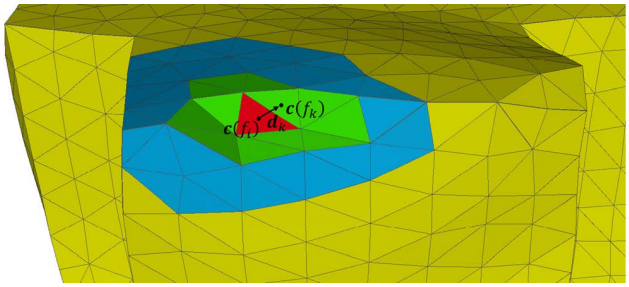

FIGURE 3. Example of d_k.

IV. PROPOSED ALGORITNM

A. DEFINITION OF SET Λ

We define a direction vector from current face f_i to one of the neighboring faces f_k as follows:

$$\boldsymbol{d}_k = \boldsymbol{c}\left(f_k\right) - \boldsymbol{c}\left(f_i\right). \quad (10)$$

As shown in Fig. 3, $\boldsymbol{d}_k$ is a vector from the center of face f_i to that of face f_k. According to the colors used in Fig. 1, f_i is marked in red, faces in the set Φ_i are green, and faces of blue and green regions are in the set Ω_i. Based on these notations,

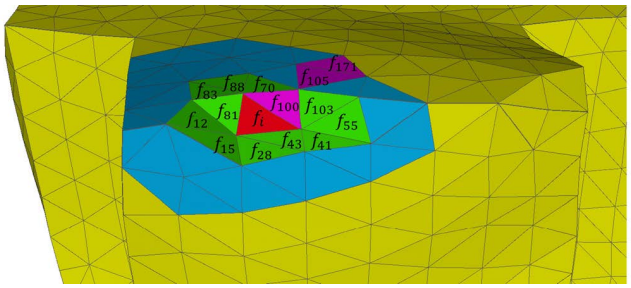

FIGURE 4. Example of a set Λ_{100} marked with pink.

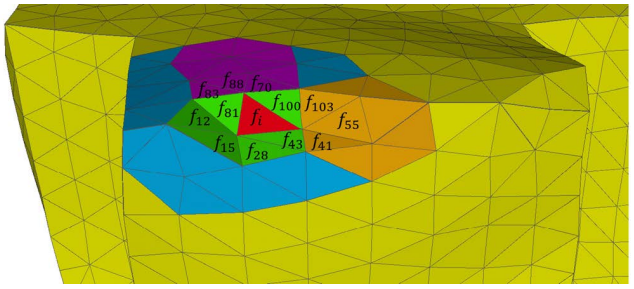

FIGURE 5. Examples of ψ_{88} and ψ_{55} marked in pink and orange, respectively.

we define the set Λ_k for each f_k in Φ_i as follows:

$$\Lambda_k = \left\{ f_l \mid \left(\boldsymbol{d}_l \cdot \frac{\boldsymbol{d}_k}{\|\boldsymbol{d}_k\|} \right) \geq \left(\boldsymbol{d}_l \cdot \frac{\boldsymbol{d}_m}{\|\boldsymbol{d}_m\|} \right), f_l \in \Omega_i, \forall f_m \in \Phi_i \right\}. \tag{11}$$

Fig. 4 shows an example of set Λ_{100} which includes f_{100},f_{105}, and f_{171} where directions of $\boldsymbol{d}_{105}$ and $\boldsymbol{d}_{171}$ are closer to the direction $\boldsymbol{d}_{100}$ than other $\boldsymbol{d}_m(\forall f_m \in \Phi_i, m \neq 100)$. We construct the sets Λ_ks for all one-ring neighbor faces f_ks. For example, we can construct Λ_{12}, Λ_{15}, Λ_{28}, Λ_{41}, Λ_{43}, Λ_{55}, Λ_{103}, Λ_{100}, Λ_{70}, Λ_{88}, Λ_{83}, and Λ_{81} in Fig. 4.

B. DEFINITION OF SET Ψ

For each face f_k in the one-ring neighbor Φ_i, we construct the set Γ_k,which consists of indices of neighboring faces satisfying the following condition.

$$\Gamma_k = \left\{ n \mid \cos^{-1} \left(\frac{\boldsymbol{d}_k \cdot \boldsymbol{d}_n}{\|\boldsymbol{d}_k\| \, \|\boldsymbol{d}_n\|} \right) < \theta, \forall f_n \in \Phi_i \right\}. \tag{12}$$

Γ_k is a set of indices of faces f_ns, which are in one-ring neighborhood and whose $\boldsymbol{d}_n$s are within an angle θ from $\boldsymbol{d}_k$. If θ is set to a small value, the subsequent processes are noise-sensitive. By contrast, if θ is set to a large value, the computational complexity of the algorithm increases substantially. It means that the selection of θ has a trade-off relationship between noise sensitivity and computational complexity. θ is set to 60° in our algorithm after experimental tests. We construct the sets Γ_ks for all one-ring neighbor faces f_ks. Based on the sets Γ_ks, we construct the sets Ψ_ks for all one-ring neighbor faces f_ks as follows.

$$\Psi_k = \bigcup\nolimits_{n \in \Gamma_k} \Lambda_n. \tag{13}$$

Fig. 5 shows examples of Γ_k and Ψ_k, where Γ_{55} is $\{41, 55, 103\}$ and Γ_{88} is $\{70, 83, 88\}$. The elements of Ψ_{88} and Ψ_{55} are marked with pink and orange, respectively.

C. FEATURE AND NON-FEATURE REGIONS

To determine whether the current face f_i is in a feature region or not, *MinMaxRatio* is calculated as follows.

$$MinMaxRatio = \frac{\min\limits_k \frac{1}{|\Psi_k|} \sum\limits_{f_l \in \Psi_k} (\boldsymbol{n}_i \cdot \boldsymbol{n}_l)}{\max\limits_k \frac{1}{|\Psi_k|} \sum\limits_{f_l \in \Psi_k} (\boldsymbol{n}_i \cdot \boldsymbol{n}_l) + \varepsilon}, \tag{14}$$

where $|\Psi_k|$ is the total number of faces in Ψ_k. In the feature region, the averaged dot product value of the current normal vector $\boldsymbol{n}_i$ and the normal vectors $\boldsymbol{n}_l$s belonging to Ψ_k depends on the geometric characteristic of Ψ_k. If set Ψ_k contains normal vectors that are similar to the normal vector of the current face such as Ψ_{88} in Fig. 5, the averaged dot product value is approximated to 1. By contrast, if Ψ_k contains normal vectors whose directions are different from the normal vector of the current face such as Ψ_{55} or Ψ_{43} in Fig. 5, the averaged dot product value decreases to -1. As the averaged angle between $\boldsymbol{n}_i$ and $\boldsymbol{n}_l$s increases, the averaged dot product decreases to -1 through 0 from 1.

In the feature region, as the normal vectors included in Ψ_k have a larger angle with the normal vector of the current face, the averaged dot product value decreases to -1 through 0 from 1. By contrast, in a non-feature region, regardless of the location of Ψ_k, the normal vector of the current face and the normal vectors of the neighboring faces have similar directions, so the averaged dot product values for all Ψ_ks are similar and close to 1.

In the numerator of (14), we calculate the averaged dot product value between the normal vector $\boldsymbol{n}_i$ of the current face f_i and normal vectors $\boldsymbol{n}_l$ of faces f_l in set Ψ_k. Among them, we select the minimum value that occurred when the averaged angle between $\boldsymbol{n}_i$ and $\boldsymbol{n}_l$s is largest. If f_i is in a feature region, the minimum of the numerator in (14) is a small value (down to -1), because there is at least one Ψ_k whose normal vectors are considerably misaligned to $\boldsymbol{n}_i$ of the current face f_i. By contrast, in the non-feature region, because the normal vectors $\boldsymbol{n}_i$ and $\boldsymbol{n}_l$s are mostly aligned, the minimum value is approximately 1.

When the mesh model consists of sphere-shaped parts, although the faces are in the non-feature region where edges are not included, $\boldsymbol{n}_i$ and $\boldsymbol{n}_l$s are not aligned, and the minimum value of the numerator of (14) is not approximated to 1. This tendency differs from that of the non-feature regions that do not include sphere-shaped parts. Therefore, in order to classify a sphere-shaped part as a non-feature region, the averaged dot product is normalized by the denominator of (14).

In the denominator of (14), the maximum value is selected among the averaged dot products. In the homogeneous region (i.e., non-feature region) that does not include the sphere-shaped part, most normal vectors $\boldsymbol{n}_l$ of faces in the set Ψ_k

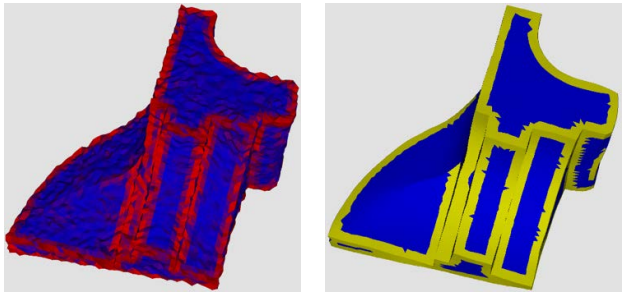

FIGURE 6. Example of feature area determined by the proposed algorithm. (a) Visualizes the feature regions by using MinMaxRatio in (14), (b) shows the feature regions with yellow color after thresholding.

are aligned with $\boldsymbol{n}_i$. Thus, the maximum value is approximately 1. By contrast, in the feature region, although many faces are misaligned with the current face, there is at least one normal vector of faces in set Ψ_k that is approximately aligned to the normal vector of the current face, because the extremely sharp mesh-like needle-shape rarely occurs. Therefore, in the feature region, the maximum value of the denominator is approximately 1. Note that the denominator of (14) is approximately 1 for the feature as well as non-feature regions (without including the sphere-shaped part). When the non-feature region consists of the sphere-shaped parts, the value of the denominator of (14) is approximately equal to that of the numerator. Therefore, the *MinMaxRatio* of (14) is approximately 1 for the non-feature region (regardless of whether sphere-shaped parts are included), whereas it has a much smaller value than 1 for the feature region.

However, we can consider a special case of the feature region where all the neighboring faces of the current face are perpendicular to the current face, although this scenario rarely occurs. In this case, the numerator and the denominator are both 0. In order to avoid the denominator being 0, a very small value of ε is added to the denominator in (14). In this special case, the *MinMaxRatio* in (14) is 0. Therefore, by comparing the *MinMaxRatio* with a threshold value between 0 and 1, the considered regions can be classified into feature or non-feature regions. In our algorithm, ε and the threshold τ_{MMR} are set to 10^{-6} and values in the range [0.7, 0.9], respectively.

Fig. 6a visualizes the feature regions by using *MinMaxRatio* in (14), where the red and blue colors are set to the faces as much as (1−*MinMaxRatio*) and *MinMaxRatio* of the faces, respectively. In Fig. 6a, to represent it clearly, the assigned color values are clipped in the range of [0, 255]. In this study, when *MinMaxRatio* of f_i is less than a threshold τ_{MMR}, we consider that f_i is in a feature region. Using the thresholding, Fig. 6b shows the feature regions with yellow color.

D. TARGET NORMAL VECTOR

In this section, we define the target normal vector $\hat{\boldsymbol{n}}_i$, which is derived from the neighbor faces whose normal vectors are aligned to $\boldsymbol{n}_i$ of the current face f_i. In order to select those neighbor faces, among all Ψ_ks, we choose one whose normal vectors are most aligned to $\boldsymbol{n}_i$ of the current face f_i according to the following equation.

$$k^* = \arg\max_k \left(\frac{1}{|\Psi_k|} \sum_{f_l \in \Psi_k} (\boldsymbol{n}_i \cdot \boldsymbol{n}_l) \right). \tag{15}$$

We can explain the meaning of (15) with an example of Fig. 5, where because faces in the orange area are over a feature, the value of (15) corresponding to the orange area is small. On other hand, the pink area has a large value of (15) because all normals of the pink region are similar to $\boldsymbol{n}_i$. In the example of Fig. 5, the value of k^* is set to 88. From k^*, the target normal vector $\hat{\boldsymbol{n}}_i$ of a f_i in a feature region is derived as follows:

$$\hat{\boldsymbol{n}}_i = G\left(\sum\nolimits_{f_l \in \Lambda_{k^*}} W_C\left(\left\| \boldsymbol{c}\left(f_i\right) - \boldsymbol{c}\left(f_j\right) \right\|, \sigma_c \right) \boldsymbol{n}_l \right), \tag{16}$$

where $G(x)$ is a normalization function of x. W_C is a Gaussian function and σ_c controls the kernel widths of Gaussian function W_C.

E. FEATURE ENHANCEMENT FOR NORMAL VECTOR

In our algorithm, the feature-enhanced normal $\tilde{\boldsymbol{n}}_i$ of the current face f_i is calculated using the following filter:

$$\tilde{\boldsymbol{n}}_i = G\left(\begin{array}{l} \sum_{f_k \in \Phi_i} \sum_{f_l \in \Lambda_k} \left\{ W_C\left(\left\| \dfrac{\boldsymbol{d}_{k^*}}{\|\boldsymbol{d}_{k^*}\|} - \dfrac{\boldsymbol{d}_k}{\|\boldsymbol{d}_k\|} \right\|, \sigma_c' \right) \right. \\ \left. \times W_T\left(\left\| \hat{\boldsymbol{n}}_i - \boldsymbol{n}_l \right\|, \sigma_s' \right) \boldsymbol{n}_l \right\} \end{array} \right) \tag{17}$$

where W_C is a Gaussian function and W_T is a Tukey's bi-weight function. $\sigma_c{}'$ and $\sigma_s{}'$ control the kernel widths of Gaussian functions W_C and W_S, respectively. The function W_C gives high-weighted values to normals that are near the set Λ_{k^*}. For example, if k^* is 88 in Fig. 5, high-weighted values are assigned to the normals of faces belonging to Λ_{70} and Λ_{83} because they are close to Λ_{k^*}. By contrast, low weights are given to the normals of the sets far from Λ_{k^*}, such as Λ_{43}, Λ_{28}, and Λ_{81}. In (17), the center position of the Tukey's bi-weight function is shifted from the current normal vector to the target normal vector, where the target normal vector is computed using (16). The function W_T enhances features by giving high weights to neighboring normals, which are similar to the target normal vector $\hat{\boldsymbol{n}}_i$ of the current face, f_i. Tukey's bi-weight function produces more robust results than the least square and Gaussian-based estimators because it completely cuts off the diffusion at sharp features [20]. As (17) is a filter to enhance the feature, it is more desirable to avoid assigning weights to outliers whose direction is very different from the target normal vector $\hat{\boldsymbol{n}}_i$, so we used Tukey's bi-weight function. Notably, (17) enhances the features; it does not remove the noise. Consequently, the enhancement module based on (17) is combined with a conventional denoising algorithm, such as BNF, GNF, and RHF, to remove the noise while enhancing the features.

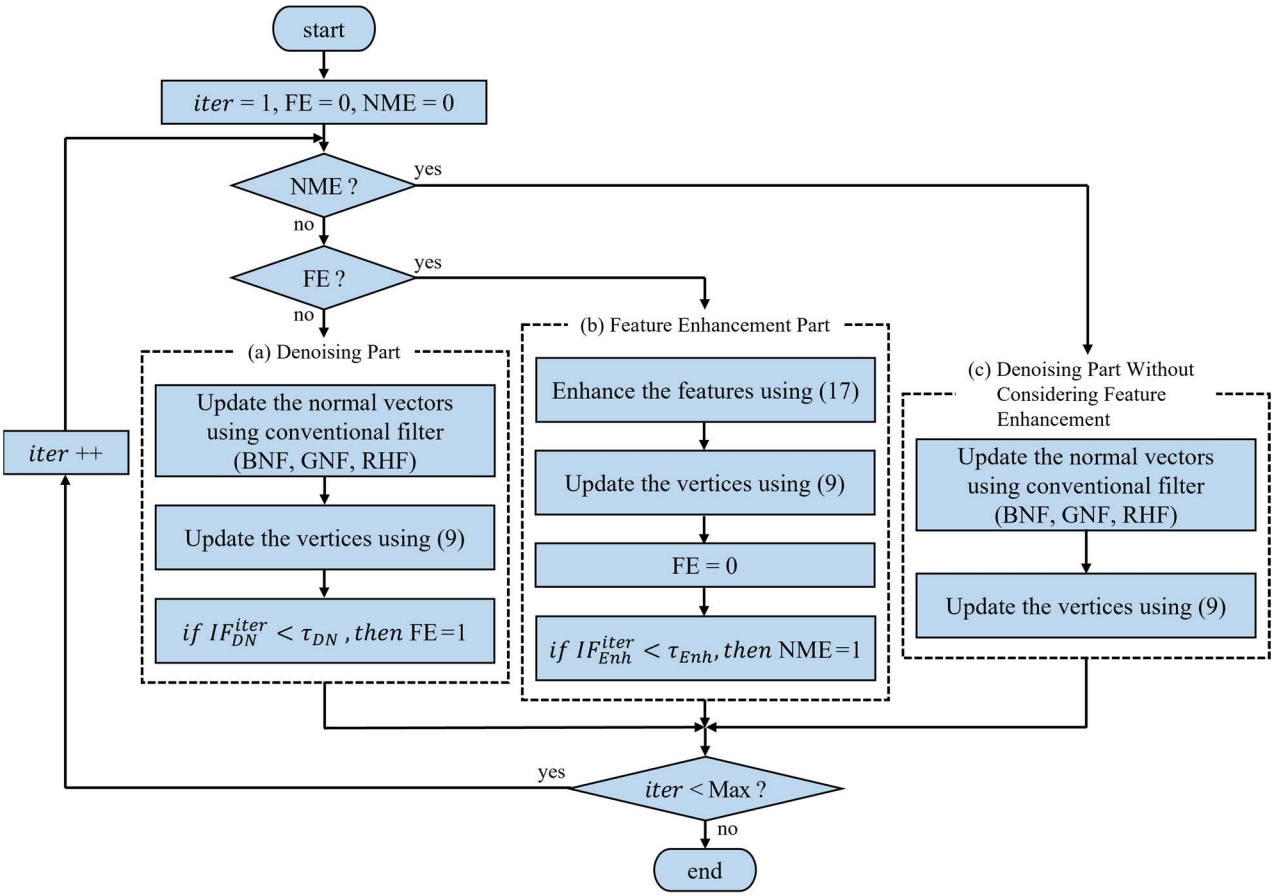

FIGURE 7. Flowchart of the proposed algorithm to remove noise and enhance the features of a mesh model.

F. CRITERION FOR IMPROVEMENT

The proposed technique applies filters to the noisy data iteratively, as many conventional algorithms do, to remove the noise. However, the proposed algorithm performs the filtering for denoising and that for feature enhancement alternatively. When the denoising filter is applied, one of the conventional denoising methods, such as BNF, GNF, and RHF, is used to remove the noise iteratively. In the denoising step, if the improvement effect of the denoising filter is saturated, we apply the feature-enhancing filter of (17). After the features have been enhanced, the denoising process is applied alternatively.

In the end of each iteration, we calculate the improvement factor (IF) to check whether the improvement resulting from the applied filters has been saturated. To calculate the IF in the $iter^{th}$ iteration, alteration B of the normal vectors is evaluated as follows.

$$B^{iter} = \frac{1}{N_T} \sum\nolimits_{\forall k} |\angle (\boldsymbol{n}_k, \bar{\boldsymbol{n}}_k)|, \tag{18}$$

where $\bar{\boldsymbol{n}}_k$s are normal vectors obtained after the vertices have been updated by (9) according to the filtered normal vectors $\tilde{\boldsymbol{n}}_i$ in (3), (6), (7), and (17). N_T is the total number of total normal vectors. (18) implies the averaged absolute difference between angles of $\boldsymbol{n}_k$s and $\bar{\boldsymbol{n}}_k$s. To check the IF of the $iter^{th}$ iteration effectively, we calculate the moving average of the first derivative of B^{iter} as follows.

$$IF^{iter} = \frac{1}{4} \sum\nolimits_{k=0}^{3} \alpha^{iter-k}, \tag{19}$$

where

$$\alpha^{iter} = \left| B^{iter} - B^{iter-1} \right|. \tag{20}$$

Fig. 7 illustrates the flowchart of the proposed algorithm, which consists of a (a) denoising process, (b) feature enhancement, and (c) denoising process without considering feature enhancement. In the denoising part, one of the conventional denoising filters in (3), (6), and (7) is used to remove the noise in the mesh model. Then, the vertices are updated by (9). At the end of each iteration of the denoising part, we evaluate the IF of (19) to check whether the noise is removed sufficiently. If IF_{DN}^{iter} is less than a threshold τ_{DN}, the flag of FE is set to 1 and the process is switched to the feature enhancement part (b) in the next iteration. IF_{DN}^{iter} is the value of IF^{iter} calculated with data resulting from part (a). In part (b), we apply the feature enhancement filter of (17) to enhance the degraded features. The feature enhancement is not applied consecutively, because consecutive feature

enhancements may amplify the noise. Thus, FE is set to 0 after each application of the feature enhancement. At the end of each iteration of the (b) feature enhancement part, if IF_{Enh}^{iter} is less than a threshold τ_{Enh}, we decide that the feature enhancement is saturated sufficiently and no more enhancement is needed by setting NME = 1. Notably, IF_{Enh}^{iter} is the value of IF^{iter} calculated with the data resulting from part (b). After NME is set to 1, only the (c) part is performed until the end of the iteration.

G. CONTRIBUTIONS

Conventional bilateral filter-based methods [17], [18], [20] used Gaussian functions to assign higher weights to the normal vectors whose corresponding faces' centroids are near that of the current face. In addition, a Gaussian function or Tukey's bi-weight function was used to assign higher weights for the neighbor normal vectors, which are similar to the normal vector of the current face. While the conventional methods consider distance as one of the most important parameters, the proposed method considers the direction from the location of the current face to those of the neighboring faces. The proposed method gives higher weights to normal vectors because the locations of neighboring faces are nearer on the line of direction set in (15). This is done so as to give low weights to the neighboring normal vectors across the edge, even though the distance from the current face is low. We proposed the set Λ_k of (11) to define the direction from the location of the current face to those of the neighboring faces. We also defined sets Γ_k, Ψ_k of Eqns. (12), (13) to determine the direction in which to assign high weights.

Some conventional mesh denoising methods [25], [34] used covariation matrices and principal component analysis (PCA) to classify meshes into feature and non-feature regions. These classification processes have high computational complexities. By contrast, we proposed a simple method to distinguish between feature regions and non-feature regions, where the parameter *MinMaxRatio* of (14) is used. According to the value of (14), the feature enhancement filter can be selectively applied to feature regions.

We also modify Tukey's bi-weight function to assign higher weights for the neighbor normal vectors that are similar to the target normal vector. In (17), the center position of the Tukey's bi-weight function is shifted from the current normal vector to the target normal vector, where the target normal vector is computed using (16). As the target normal vector in the feature region is more robust to noise than the current normal vector, the proposed filter better preserves the feature.

V. EXPERIMENTAL RESULTS

In this section, we evaluate the performance of the proposed algorithm and compare it with those of various conventional methods. As we can see in Fig. 7, the proposed algorithm imports one of the conventional denoising methods, such as the bilateral normal filter (BNF) [17], the guided normal filter (GNF) [18], and the robust and high-fidelity filter (RHF) [20] to remove noise from mesh models. The performance of our proposed algorithm is compared with those of other methods such as l_0 minimization (L0M) [24], the non-local low-rank filter (NLLR) [36], and the cascaded normal regression (CNR) filter [28] as well as BNF, GNF, and RHF. The source codes of GNF, RHF, and NLLR have been kindly provided by their authors, whereas BNF and L0M are implemented by a third party. The trained neural networks of CNR have been kindly provided by their authors [28].

When the conventional methods are integrated with the proposed algorithm, the parameters of the methods are set to harmonize with those of the proposed algorithm. When BNF is used as in [17], it requires two iteration parameters to be set because it is a two-stage method. However, when BNF is integrated with the proposed method, one iteration parameter is required. As RHF is a one-stage method like the proposed method, it can be used in the proposed algorithm without any alteration.

A. PARAMETERS SETTING

The proposed algorithm has some parameters, such as radius r in (2), a threshold τ_{MMR} to be compared with *MinMaxRatio* of (14), σ_c' and σ_s' in (17), a threshold τ_{DN} to check the effect of the denoising, a threshold τ_{Enh} for the enhancement.

The radius r parameter represents the range of neighborhood of the current face. BNF constitutes neighboring faces with 1-ring neighbor. GNF calculates $r = c_a \times \mu$ for a radius to constitute neighboring faces, where c_a is the average distance among the centroids of all faces, and μ is a parameter set by the user. In RHF, r is set to $c_a \times 2$. When BNF and RHF are respectively integrated with the proposed algorithm, neighboring faces are constructed in their ways, and when GNF is used, μ is set to 2. In (11)-(17), r is set to $c_a \times 3.5$.

τ_{MMR} depends on the noise level and is set to values in the range [0.7, 0.9]. From empirical test data, we found that the performance of the proposed algorithm is not sensitive to τ_{MMR}.

The parameters σ_c and σ_s in (3), (6), and (7) are set to values used in BNF, GNF, and RHF. The σ_c' in (17) is set to the average distance c_a among the centroids of faces as was done by Yadav *et al.* [20]. σ_s' is set by the user as in [18], [20].

From empirical test data, we found that τ_{DN} and τ_{Enh} are independent of noise levels and characteristics of the mesh models. In the simulations of this section, we set τ_{DN} and τ_{Enh} as 1.0 and 0.15, respectively.

B. TEST MODELS

To evaluate the performances of conventional and proposed methods, we used various mesh models, such as Fandisk, Block, Sharpsphere, Twelve, Nicolo, Julius, Bunny, Cone, and Pyramid, which are commonly used in the literature. Fandisk, Block, Sharpsphere, Twelve, Nicolo, Julius, and Bunny are CAD-generated models and they contain differing noise levels. Cone and Pyramid are scanned by Microsoft Kinect v1 (Microsoft's motion sensor add-on) via the Kinect-Fusion technique [28]. Fandisk, Block, Sharpsphere and

TABLE 1. Performance comparison with BNF [17].

Models	Methods	MSAE	$E_v(\times 10^{-3})$	Parameters
Fandisk Gaussian noise 0.3*le*	BNF	3.349	8.998	(optimized)
	OURS + BNF	**2.412**	**6.881**	(0.4, 0.7, 80)
Block Gaussian noise 0.4*le*	BNF	5.306	70.21	(optimized)
	OURS +BNF	**3.861**	**61.40**	(0.4, 0.7, 80)
Sharpsphere Gaussian noise 0.3*le*	BNF	6.705	47.50	(optimized)
	OURS + BNF	**5.848**	**45.49**	(0.45, 0.7, 60)
Twelve Impulsive noise 0.5*le*	BNF	7.368	12.62	(optimized)
	OURS + BNF	**3.681**	**7.554**	(0.35, 0.9, 80)
Nicolo Gaussian noise 0.2*le*	BNF	5.798	**264.0**	(optimized)
	OURS + BNF	**5.785**	264.7	(0.75, 0.7, 13)
Julius Gaussian noise 0.2*le*	BNF	6.212	0.557	(optimized)
	OURS + BNF	**5.918**	**0.539**	(0.45, 0.7, 22)
Bunny Gaussian noise 0.2*le*	BNF	5.675	0.968	(optimized)
	OURS + BNF	**5.582**	**0.887**	(0.5, 0.7, 30)
Cone Acquired by Kinect	BNF	8.102	**1250.5**	(0.55, 27, 260)
	OURS + BNF	**7.748**	1267.4	(0.55, 0.9, 400)
Pyramid Acquired by Kinect	BNF	10.38	1153.5	(0.60, 15, 40)
	OURS + BNF	**8.976**	**1029.9**	(0.5, 0.9, 100)

TABLE 2. Performance comparison with GNF [18].

Models	Methods	MSAE	$E_v(\times 10^{-3})$	Parameters
Fandisk Gaussian noise 0.3*le*	GNF	2.624	**6.750**	(optimized)
	OURS + GNF	**2.512**	7.037	(0.3, 0.7, 70, 2)
Block Gaussian noise 0.4*le*	GNF	3.572	56.19	(optimized)
	OURS + GNF	**3.342**	**53.30**	(0.3, 0.9, 60, 5)
Sharpsphere Gaussian noise 0.3*le*	GNF	10.17	60.82	(optimized)
	OURS + GNF	**7.185**	**49.85**	(0.45, 0.7, 60, 1)
Twelve Impulsive noise 0.5*le*	GNF	2.754	6.347	(optimized)
	OURS + GNF	**2.374**	**5.978**	(0.25, 0.9, 100, 5)
Nicolo Gaussian noise 0.2*le*	GNF	6.501	**272.0**	(optimized)
	OURS + GNF	**6.174**	275.0	(0.65, 0.7, 17, 1)
Julius Gaussian noise 0.2*le*	GNF	6.132	**0.567**	(optimized)
	OURS + GNF	**6.088**	0.579	(0.6, 0.7, 17, 1)
Bunny Gaussian noise 0.2*le*	GNF	5.354	**0.792**	(optimized)
	OURS + GNF	**5.286**	0.804	(0.5, 0.7, 12, 1)
Cone Acquired by Kinect	GNF	7.408	**1237.1**	(2, 0.45, 150, 50)
	OURS + GNF	**7.281**	1240.2	(0.35, 0.9, 150, 30)
Pyramid Acquired by Kinect	GNF	7.446	1014.8	(2, 0.75, 90, 10)
	OURS + GNF	**7.145**	**1009.3**	(0.45, 0.9, 100, 10)

Twelve contain distinct feature areas, whereas Nicolo, Julius and Bunny consist of smooth surfaces. For CAD-generated models, we add impulse noise to Twelve model, and we corrupted other CAD-generated models with Gaussian noise.

C. OBJECTIVE EVALUATIONS

The two most widely used error metrics, mean square angular error (MSAE) and L_2 vertex-based surface-to-surface error [44], were used for objective evaluations of the models used in our study. MSAE represents the mean angular difference between the matching normal vectors of the ground-truth and denoised models as follows,

$$MSAE = \frac{1}{|\mathbb{N}|}\sum_{i=1}^{|\mathbb{N}|} \angle\left(\boldsymbol{n}_i, \boldsymbol{n}_i'\right), \tag{21}$$

where $\boldsymbol{n}_i$ is the normal vector of ground-truth, $\boldsymbol{n}_i'$ is the normal vector of denoised mesh, and $|\mathbb{N}|$ is the number of normal vectors. The L_2 vertex-based error is a metric that calculates how close the positions of vertices in the denoised model are to those in the ground-truth model. The L_2 vertex-based error is defined as follows,

$$E_v = \sqrt{\frac{1}{3\sum_{j\in\mathbb{F}} A_j}\sum_{i\in\mathbb{V}}\sum_{j\in F_{\boldsymbol{v}}(i)} A_j dist\left(\boldsymbol{v}_i', T\right)^2}, \tag{22}$$

where T is the nearest face in the ground-truth mesh to the vertex $\boldsymbol{v}_i'$ of the denoised mesh. The L_2 vertex-based error is calculated based on the weighted average of the distance between vertices, where the weighting factors are areas of the surrounding faces of the corresponding vertex. As discussed in [23] and [45], MSAE metric is more correlated to visual quality than L_2 vertex-based metric.

In Table 1, the performance of the proposed algorithm incorporating BNF is compared with that of BNF. In simulations with the CAD models, the parameters of BNF were set to the optimal values provided in [18]. The parameters (σ_s, *Niter*, *Viter*) of BNF for Cone and Pyramid models and the parameters (σ_s', τ_{MMR}, *iter*) of the proposed algorithm for all models were set to the optimal values, which provide the best performances in the empirical tests.

From Table 1, we can observe that the proposed method outperforms BNF for all models in respect to MSAE; the proposed algorithm reduces MSAE significantly in the models Fandisk, Block, Sharpsphere, Twelve, and Pyramid, which contain many distinct features. As for Nicolo, Julius, and Bunny models, which have few features and consists of smoothed edges, the proposed algorithm slightly improve the performance in respect to MSAE compared to BNF.

In the E_v column, the proposed algorithm outperforms BNF in all models except for Nicolo and Cone models. The performance of the proposed algorithm is approximately equal to that of BNF for Nicolo and Cone models.

TABLE 3. Performance comparison with RHF [20].

Models	Methods	MSAE	$E_v(\times 10^{-3})$	Parameters
Fandisk Gaussian noise 0.3*le*	RHF	3.319	9.052	(optimized)
	OURS + RHF	**2.666**	**7.347**	(0.55, 0.1, 0.7, 100)
Block Gaussian noise 0.4*le*	RHF	4.268	74.42	(0.55, 0.2, 100)
	OURS + RHF	**3.836**	**72.79**	(0.55, 0.2, 0.9, 100)
Sharpsphere Gaussian noise 0.3*le*	RHF	**7.113**	57.54	(0.85, 0.1, 80)
	OURS + RHF	7.345	**56.68**	(0.9, 0.05, 0.7, 80)
Twelve Impulsive noise 0.5*le*	RHF	2.842	7.416	(0.55, 0.2, 100)
	OURS + RHF	**2.613**	**7.273**	(0.55, 0.2, 0.8, 100)
Nicolo Gaussian noise 0.2*le*	RHF	6.310	302.8	(optimized)
	OURS + RHF	**6.268**	**260.7**	(0.65, 0.1, 0.7, 15)
Julius Gaussian noise 0.2*le*	RHF	6.490	0.554	(0.55, 0.1, 30)
	OURS + RHF	**6.350**	**0.549**	(0.55, 0.1, 0.8, 20)
Bunny Gaussian noise 0.2*le*	RHF	5.386	0.841	(0.6, 0.1, 20)
	OURS + RHF	**5.353**	**0.833**	(0.7, 0.1, 0.7, 18)
Cone Acquired by Kinect	RHF	7.698	1256.9	(0.95, 0.3, 400)
	OURS + RHF	**7.697**	1256.9	(0.9, 0.3, 0.9, 400)
Pyramid Acquired by Kinect	RHF	7.456	**850.4**	(0.45, 0.2, 300)
	OURS + RHF	**7.314**	852.0	(0.65, 0.2, 0.9, 300)

In the models Fandisk, Block, Sharpsphere, Twelve, and Pyramid, which contain many distinct features, E_v is significantly improved.

As we observe in Table 1, the proposed algorithm provides large improvement for models having distinct features, because the main idea of the proposed algorithm is to enhance the features while the defects in the plain regions are removed.

Table 2 represents the performances of GNF and the proposed algorithm incorporating GNF for various models. When GNF is implemented, the parameters (μ, σ_s, *Niter*, *Viter*) for the CAD models are set to the optimal values provided by [18]. The parameters of GNF for Cone and Pyramid models and the parameters (σ_s', τ_{MMR}, *Niter*, *Viter*) of the proposed algorithm for all models are set to the optimal values, which provide the best performances in the empirical tests.

From MSAEs of BNF and GNF in Table 1 and 2, we observe that MSAEs of GNF are much lower than those of BNF for the distinct feature models Fandisk, Block, and Twelve. It means that GNF preserves features better than BNF.

In Table 2, in the MSAE column, the proposed algorithm outperforms the GNF method for all models. We observe that the tendencies of MSAE improvement of the proposed algorithm are similar in Tables 1 and 2. From the results for

TABLE 4. Performance comparison with other methods.

Models	Methods	MSAE	$E_v(\times 10^{-3})$	Parameters
Fandisk Gaussian noise 0.3*le*	OURS + BNF	**2.412**	**6.881**	(0.4, 0.7, 80)
	OURS + GNF	**2.512**	7.037	(0.3, 0.7, 70, 2)
	OURS + RHF	2.666	7.347	(0.55, 0.1, 0.7, 100)
	L0M	5.529	18.84	(optimized)
	NLLR	8.150	10.57	(0.3, 10, 10)
	CNR	2.696	**6.980**	(parameter-free)
Block Gaussian noise 0.4*le*	OURS + BNF	3.861	**61.40**	(0.4, 0.7, 80)
	OURS + GNF	**3.342**	**53.30**	(0.3, 0.9, 60, 5)
	OURS + RHF	**3.836**	72.79	(0.55, 0.2, 0.9, 100)
	L0M	4.973	117.9	(optimized)
	NLLR	9.961	94.53	(0.4, 14, 9)
	CNR	4.664	74.08	(parameter-free)
Sharpsphere Gaussian noise 0.3*le*	OURS + BNF	**5.848**	**45.49**	(0.4, 0.7, 60)
	OURS + GNF	**7.185**	**49.85**	(0.45, 0.7, 60, 1)
	OURS + RHF	7.345	56.68	(0.9, 0.05, 0.7, 80)
	L0M	12.96	124.0	(optimized)
	NLLR	10.95	74.21	(0.3, 16, 9)
	CNR	8.376	59.77	(parameter-free)
Twelve Impulsive noise 0.5*le*	OURS + BNF	3.681	7.554	(0.35, 0.9, 80)
	OURS + GNF	**2.374**	**5.978**	(0.2, 0.8, 100, 3)
	OURS + RHF	**2.613**	**7.273**	(0.55, 0.2, 0.8, 100)
	L0M	8.463	20.09	(optimized)
	NLLR	6.505	10.53	(0.5, 14, 17)
	CNR	3.541	7.705	(parameter-free)
Nicolo Gaussian noise 0.2*le*	OURS + BNF	**5.785**	**264.7**	(0.75, 0.7, 13)
	OURS + GNF	6.174	275.0	(0.65, 0.7, 17, 1)
	OURS + RHF	6.268	**260.7**	(0.65, 0.1, 0.7, 15)
	L0M	7.583	358.4	(optimized)
	NLLR	6.618	287.7	(optimized)
	CNR	**5.981**	276.9	(parameter-free)
Julius Gaussian noise 0.2*le*	OURS + BNF	**5.918**	**0.539**	(0.45, 0.7, 22)
	OURS + GNF	**6.088**	0.579	(0.6, 0.7, 17, 1)
	OURS + RHF	6.350	**0.549**	(0.55, 0.1, 0.8, 20)
	L0M	7.977	0.868	(optimized)
	NLLR	6.485	0.625	(0.2, 13, 10)
	CNR	6.465	0.653	(parameter-free)
Bunny Gaussian noise 0.2*le*	OURS + BNF	5.582	0.887	(0.5, 0.7, 30)
	OURS + GNF	**5.286**	**0.804**	(0.5, 0.7, 12, 1)
	OURS + RHF	**5.353**	0.833	(0.7, 0.1, 0.7, 18)
	L0M	7.214	1.109	(optimized)
	NLLR	5.638	0.828	(0.2, 15, 7)
	CNR	5.360	**0.823**	(parameter-free)
Cone Acquired by Kinect	OURS + BNF	7.748	1267	(0.55, 0.9, 400)
	OURS + GNF	**7.281**	**1240**	(0.35, 0.9, 150, 30)
	OURS + RHF	7.697	1257	(0.9, 0.3, 0.9, 400)
	L0M	**7.205**	**1245**	(0.001257, 10)
	NLLR	8.023	1258	(0.5, 30, 150)
	CNR	7.922	1259	(parameter-free)
Pyramid Acquired by Kinect	OURS + BNF	8.976	1030	(0.5, 0.9, 100)
	OURS + GNF	**7.145**	1009	(0.45, 0.9, 100, 10)
	OURS + RHF	7.314	**852.0**	(0.65, 0.2, 0.9, 300)
	L0M	**6.588**	966.0	(0.0008, 5)
	NLLR	8.540	973.2	(0.5, 35, 45)
	CNR	7.493	**942.6**	(parameter-free)

Sharpsphere including the curved sharp edges in Table 2, we observe that MSAE and E_v of the proposed algorithm

are much smaller than those of GNF. This implies that the proposed method preserves the curved sharp edges better than the GNF method. In the column of E_v in Table 2, the proposed algorithm is slightly inferior to GNF in some models, such as Fandisk, Nicolo, Julius, Bunny, and Cone.

The comparison between performances of RHF and the proposed algorithm incorporating RHF is summarized in Table 3. The parameters (σ_s, λ, *iter*) of the RHF method for Fandisk and Nicolo models are set to the optimal values provided by [20]. The parameters of RHF for other models and the parameters (σ_s', λ, τ_{MMR}, *iter*) of the proposed algorithm for all models are set to the optimal values, which provide the best performances in the empirical tests. The tendency of the results shown in Table 3 is very similar to that of Table 2, except for results of the Sharpsphere model.

In Table 4, the performances of the proposed algorithms respectively integrated with BNF, GNF, and RHF methods are compared with those of L0M, NLLR, and CNR, where the best and second-best performances are shown in red and blue colors, respectively. As we observe from Table 4, one of the proposed algorithms shows the best performances of MSAE and E_v for all CAD models, such as Fandisk, Block, Sharpsphere, Twelve, Nicolo, Julius, and Bunny. As for Cone and Pyramid models, the proposed algorithm has the second best MSAE and the best E_v. Notably, Cone and Pyramid models consist of overall flat regions, except for few feature regions over small areas. This property of those models is matched to the hypothesis of the L0M method. It is the reason why L0M has the best MSAE values for the Cone and Pyramid models.

In Table 5, the time required to complete the processes of various algorithms is compared. BNF is the fastest method because it only considers the distance between the current face and neighboring faces and the difference between the current normal vector and neighboring normal vectors. However, it results in degraded meshes when the models include distinct features, as shown in Figs. 8b and 9b. In the case of OURS+BNF, although the time consumed increases slightly, enhanced results are obtained, as shown in Figs. 8c and 9c. The GNF method needs the longest computation times because it requires many iterations for updating vertices, where the total number of iterations of updating vertices is *Viter* $\times$ *Niter*. In OURS+GNF, the number of iterations of updating vertices is substantially decreased because the proposed filter effectively removes noise with a small *Viter* while preserving features satisfactorily. For example, in the Block model, the GNF method has a *Niter* value of 40 and a *Viter* value of 30, resulting in 1200 iterations of updating vertices. By contrast, because the OURS+GNF method has a *Niter* value of 70 and a *Viter* value of 2, only 140 iterations are performed. Thus, the consumed times of OURS+GNF are smaller than those of GNF. When RHF and OURS+RHF are compared, OURS+RHF requires slightly higher computational times than RHF, because the sets Λ_k, Γ_k, Ψ_k of (11), (12), (13) and the target normal vector should be calculated. However, OURS+RHF produces better results than RHF, as shown in Figs. 8g and 9g, in a reasonable time, because the feature enhancement filter of the proposed algorithm is no longer applied when the feature enhancement is saturated according to (19).

TABLE 5. Consumed times (sec) of the techniques.

Models	Fandisk	Block	Sharpsphere	Twelve
BNF	2.094	4.155	1.627	2.134
OURS+BNF	7.618	10.40	11.09	5.386
GNF	14.58	47.33	30.20	32.42
OURS+GNF	11.38	19.16	16.57	15.57
RHF	5.801	7.765	8.605	4.154
OURS+RHF	8.739	14.12	14.00	7.012

D. SUBJECTIVE EVALUATIONS

In this section, the subjective performances of various algorithms are evaluated for a variety of mesh models. In Figs. 8, 9, 10, and 11, we can visually compare the meshes denoised by BNF, OURS+BNF, GNF, OURS+GNF, RHF, and OURS+RHF.

Figs. 8 and 9 show the results for Fandisk, Block, Sharpsphere, and Twelve models, which have distinct feature areas. In the Fandisk model, challenging regions, including corners, are indicated by two red windows. As shown in the windows, the proposed method ensures that the features are well reconstructed. The region in the red window of the Block model consists of triangular faces whose density is very high. Whereas most conventional methods do not recover features in this region, the proposed method efficiently reconstructs the features in the challenging areas. In the enlarged window of the Sharpsphere model, we observe that the BNF method fails to recover the features. Conversely, the GNF method reconstructs its features to some extent, but does not recover its curved edge. The proposed algorithm, integrated with the BNF or GNF method can perfectly reconstruct the features and effectively enhance the curved edges. In columns (f) and (g) of Fig. 9, both the RHF method and the proposed algorithm incorporating RHF show excellent reconstruction and enhancement of features of the Sharpsphere model. In the Twelve model, one of the edges is enlarged in the red window. Among the results (b), (d), and (f) obtained from the BNF, GNF, and RHF methods for the Twelve model, respectively, the GNF method results in the best visual quality, but the left end of the edge is distorted in the result. Conversely, all denoised results by the proposed algorithm show perfectly reconstructed edges. Note that the BNF method is not effective for impulse noise, but the proposed algorithm preserves the features corrupted from impulse noise.

Fig. 10 shows the denoised meshes of the Nicolo, Julius, and Bunny models, which have sparse features. In the Nicolo model, there is no significant difference between visual qualities of meshes resulting from BNF and the proposed algorithm incorporating BNF. When GNF method is applied on the noisy Nicolo model, some wrong features are generated

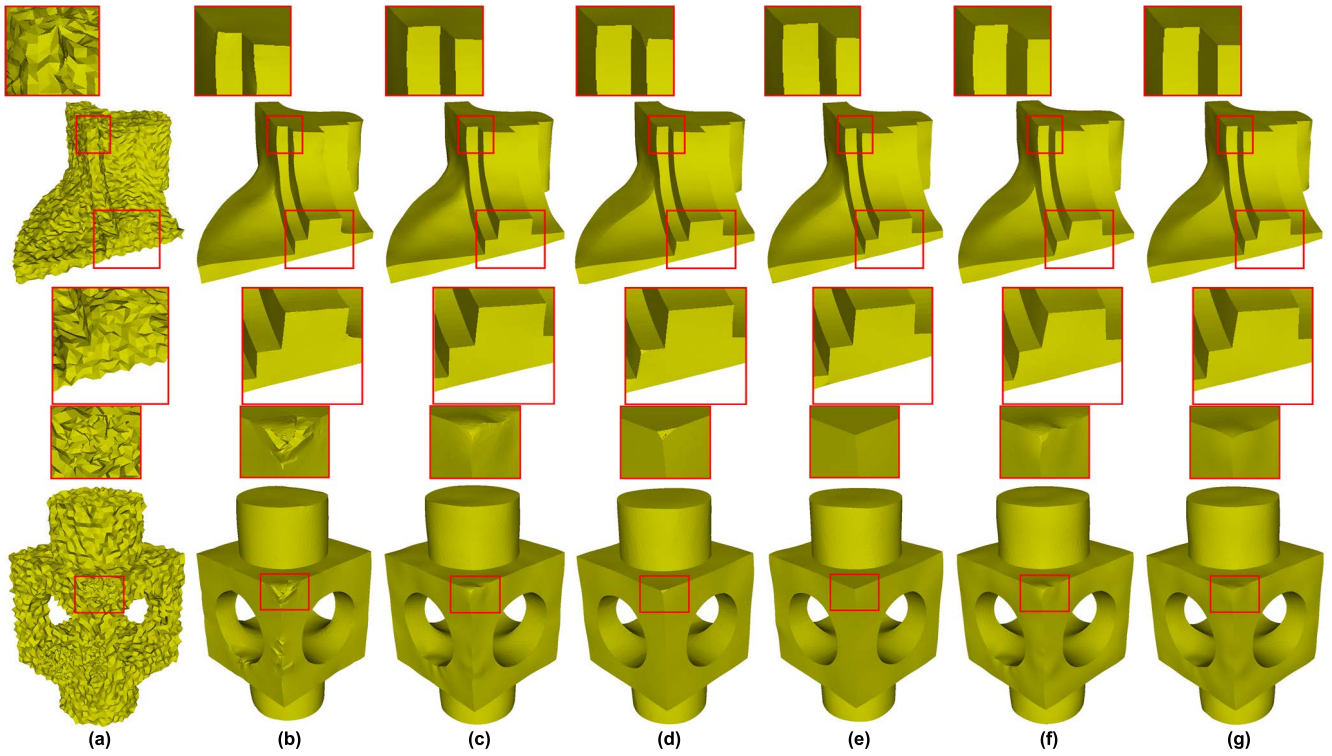

FIGURE 8. Illustration of the denoised results on Fandisk and Block. (a) noisy meshes, (b) - (g) the meshes denoised by BNF, OURS+BNF, GNF, OURS+GNF, RHF, and OURS+RHF.

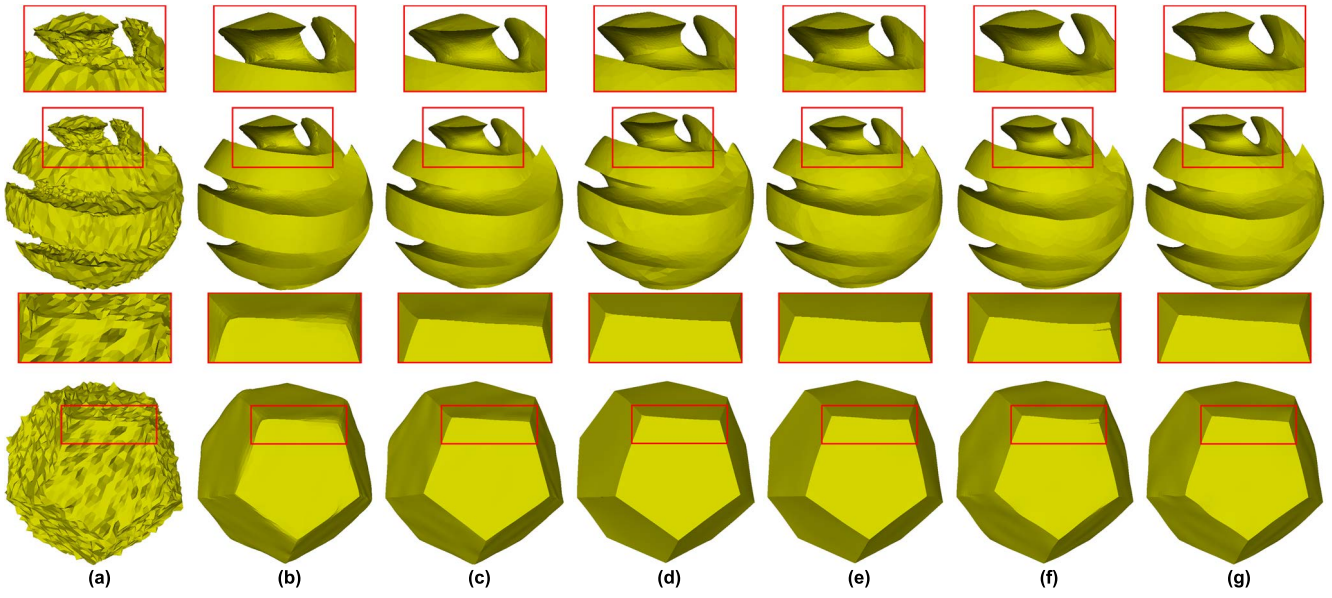

FIGURE 9. Illustration of the denoised results on Sharpsphere and Twelve. (a) noisy meshes, (b) - (g) the meshes denoised by BNF, OURS+BNF, GNF, OURS+GNF, RHF, and OURS+RHF.

in the nose area of the resulting mesh. Conversely, the proposed algorithm using GNF provides the enhanced features in the nose area. In the test of RHF for Nicolo model, it blurs the model excessively, whereas the proposed algorithm incorporating RHF reconstructs the features efficiently. When the Julius model is used for various algorithms, the results obtained from GNF, OURS+GNF, RHF, and OURS+RHF shows good visual quality. Conversely, BNF method degrades the features in the areas of the hair and mouth of the Julius model, whereas the OURS+BNF enhances those features

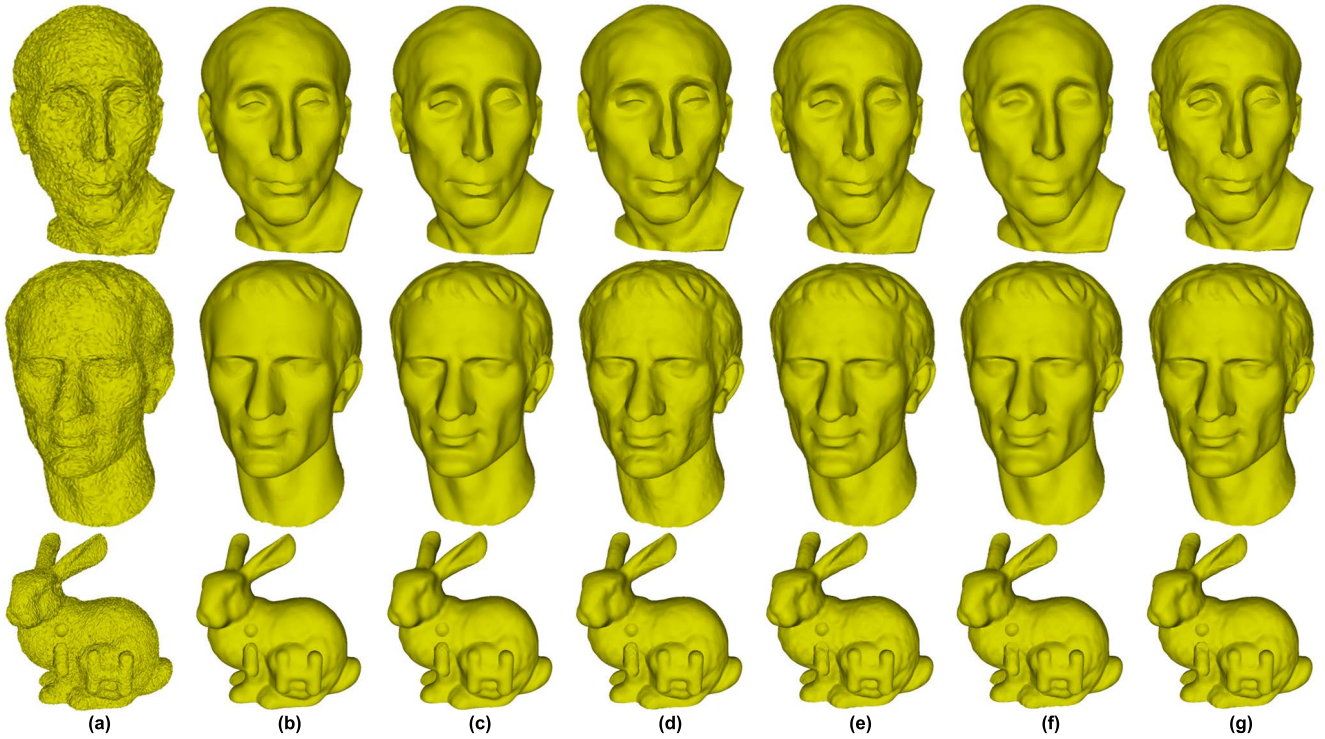

FIGURE 10. Illustration of the denoised results on Nicolo, Julius, and Bunny. (a) noisy meshes, (b) - (g) the meshes denoised by BNF, OURS+BNF, GNF, OURS+GNF, RHF, and OURS+RHF.

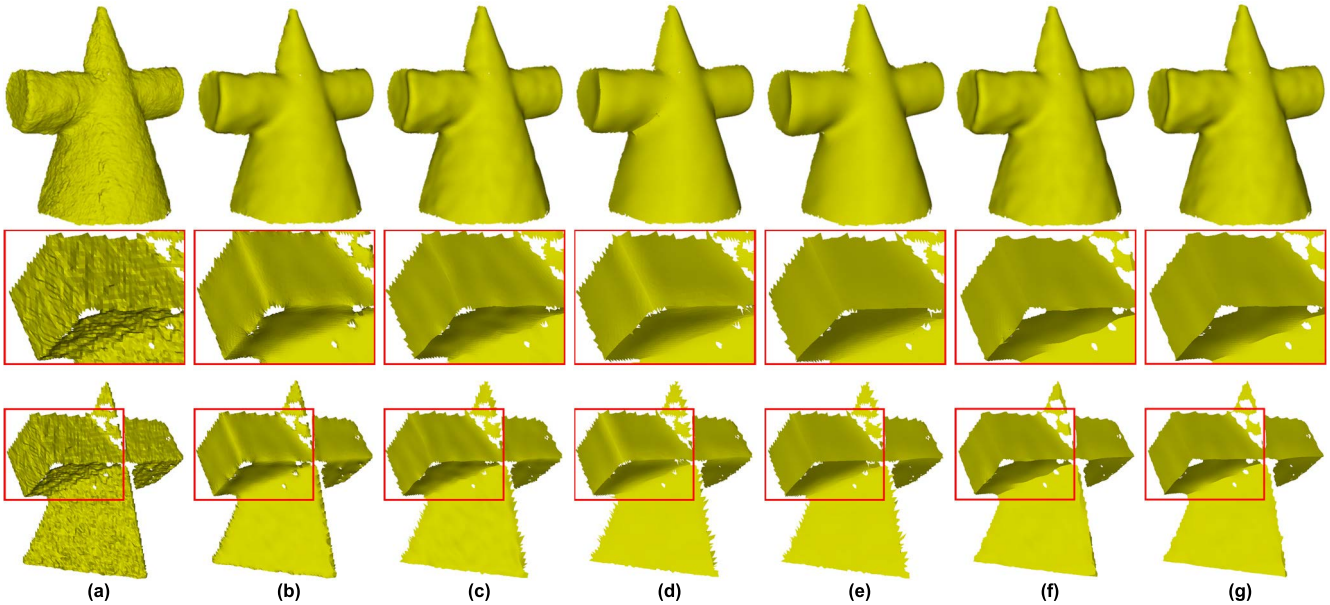

FIGURE 11. Illustration of the denoised results on Cone and Pyramid. (a) noisy mesh, (b) - (g) the meshes denoised by BNF, OURS+BNF, GNF, OURS+GNF, RHF, and OURS+RHF.

efficiently. As for the Bunny model, because it has no feature to be enhanced, similar results are generated from all algorithms.

Fig. 11 shows the simulation results for the Cone and Pyramid models. In the tests with the Cone model, GNF generated false features in the lower part of the left arm, whereas OURS+GNF reconstructed the features correctly. The Cone models denoised by BNF, OURS+BNF, RHF, and OURS+RHF methods have blurred features. Because the triangular faces of the Cone model are very small and the area of a set of neighboring faces is also very small, the proposed algorithms (OURS+BNF and OURS+RHF) do not

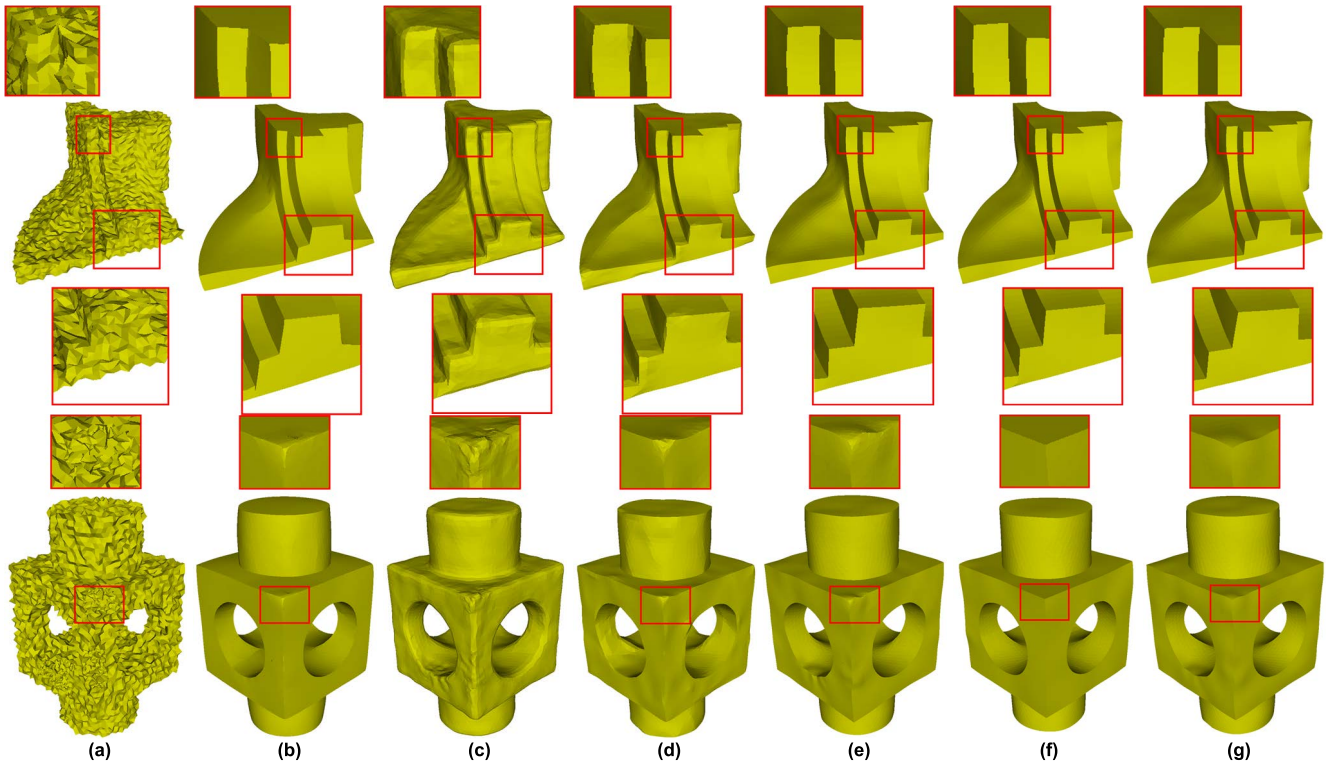

FIGURE 12. Illustration of the denoised results on Fandisk and Block. (a) noisy mesh, (b) - (g) the meshes denoised by L0M, NLLR, CNR, OURS+BNF, OURS+GNF, and OURS+RHF.

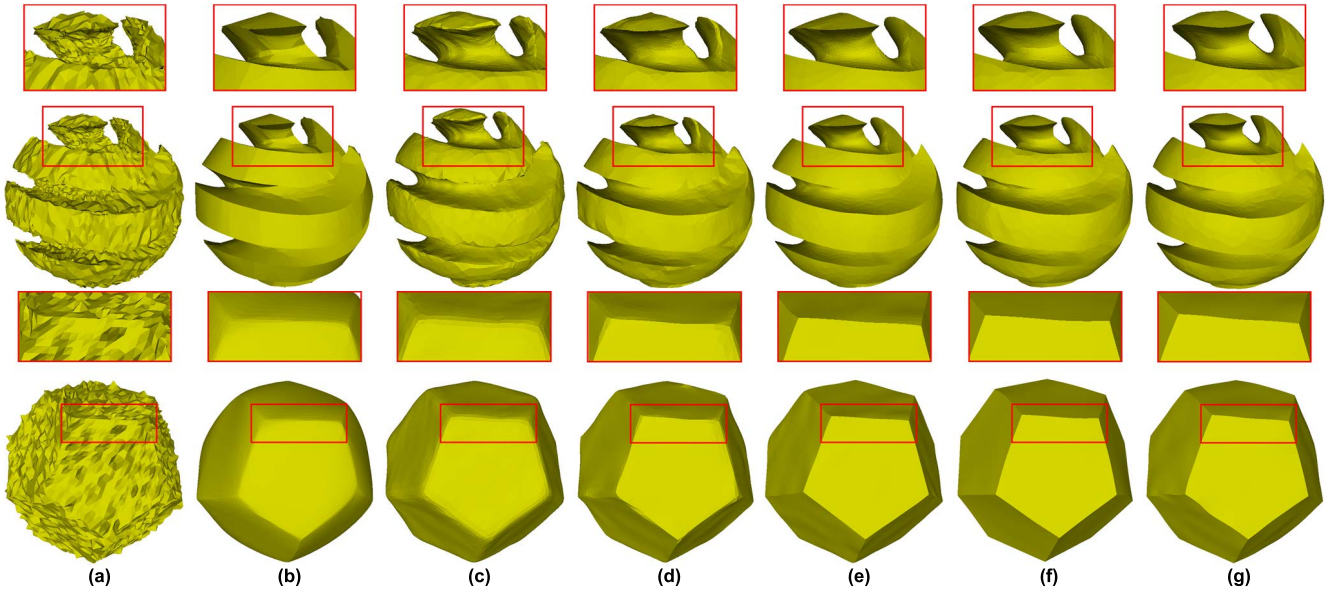

FIGURE 13. Illustration of the denoised results on Sharpsphere and Twelve. (a) noisy mesh, (b) - (g) the meshes denoised by L0M, NLLR, CNR, OURS+BNF, OURS+GNF, and OURS+RHF.

recover the noisy features, which are located actually in the near neighbor but out of the set Ω_i. When the Pyramid model is used as a test data, we observe from results of the enlarged red windows that the conventional methods of BNF, GNF, and RHF blur out corners and edges, whereas the proposed algorithm, integrated with those conventional methods, reconstructs the features efficiently.

In Figs. 12, 13, 14, and 15, the subjective performance of proposed algorithm is compared with those of L0M, NLLR, and CNR methods. The test models used in

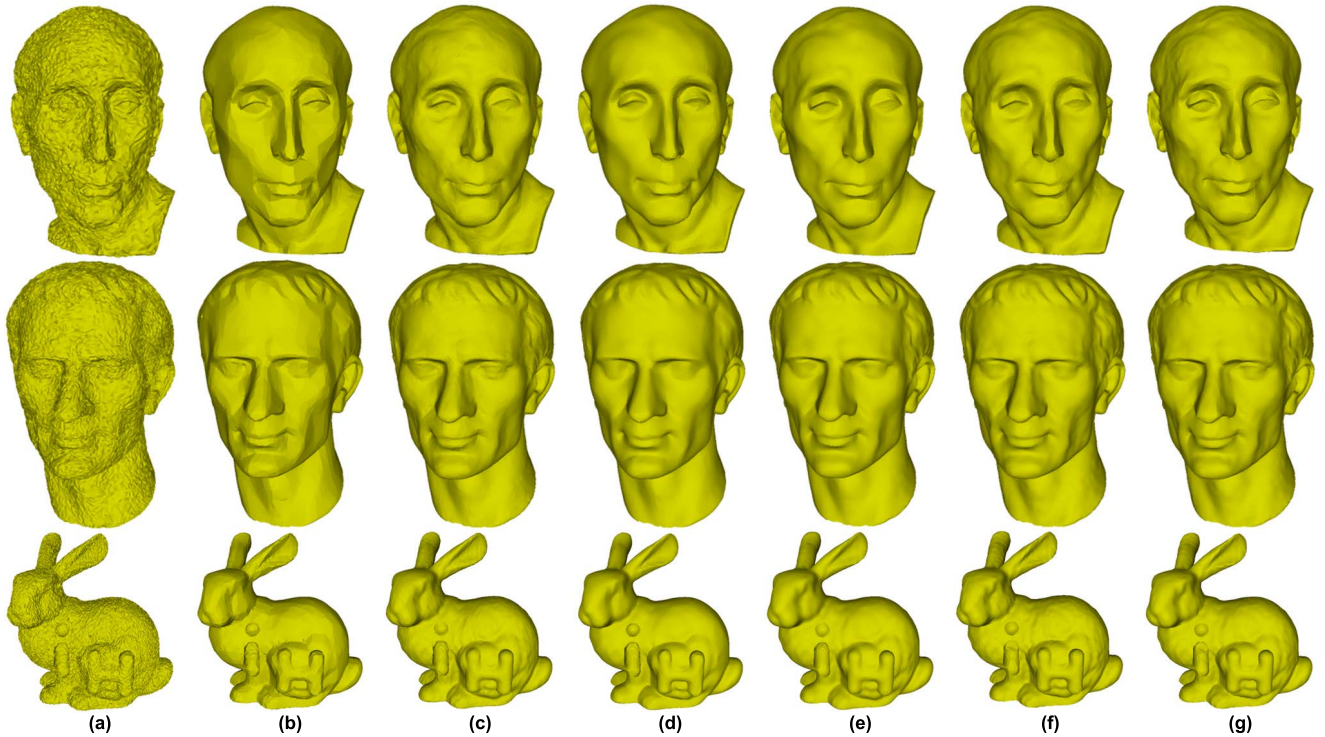

FIGURE 14. Illustration of the denoised results on Nicolo, Julius, and Bunny. (a) noisy mesh, (b) - (g) the meshes denoised by L0M, NLLR, CNR, OURS+BNF, OURS+GNF, and OURS+RHF.

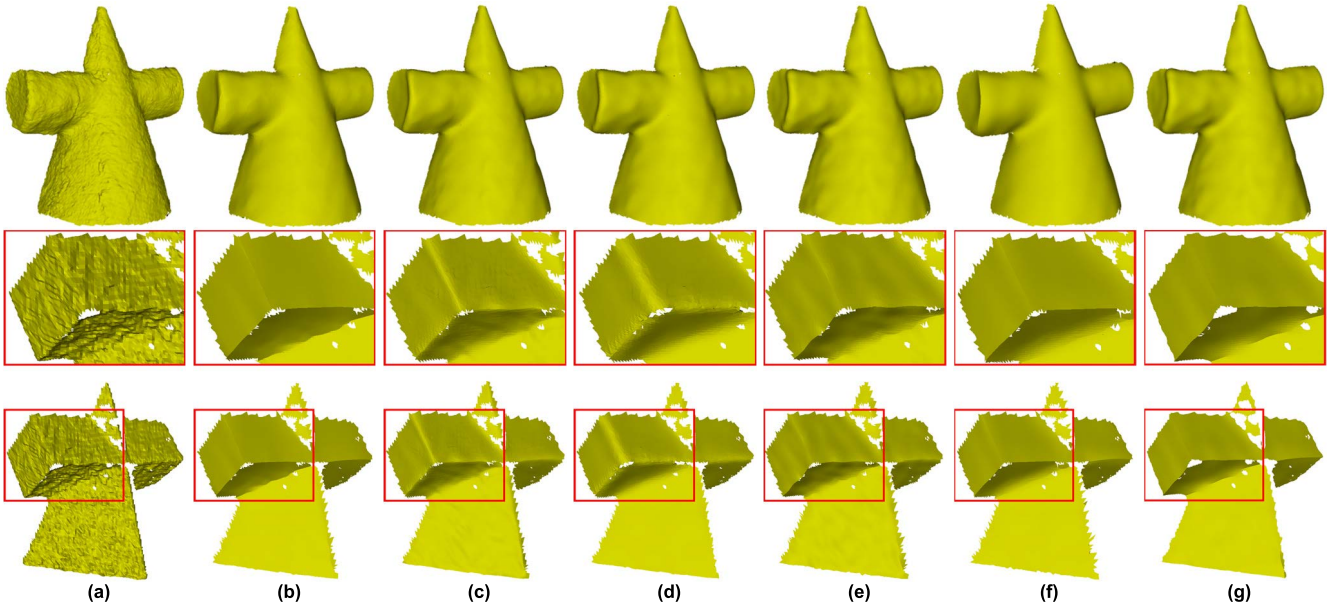

FIGURE 15. Illustration of the denoised results on Cone and Pyramid. (a) noisy mesh, (b) - (g) the meshes denoised by L0M, NLLR, CNR, OURS+BNF, OURS+GNF, and OURS+RHF.

Figs. 12, 13, 14, and 15 are same as those in Figs. 8, 9, 10, and 11, respectively.

In Fig. 12, the Fandisk models denoised by L0M, NLLR, and CNR methods have much more degraded features in the corner regions than those by the proposed algorithms. When the Block model is used, the L0M method reconstructs the features in the corner regions better than the NLLR and CNR methods. However, L0M does not reconstruct the curved area of the model. Conversely, the proposed method respectively incorporating the BNF and RHF methods, enhances the features in the corner areas efficiently. In the test on the Block model, OURS+GNF show the best performance in the corner

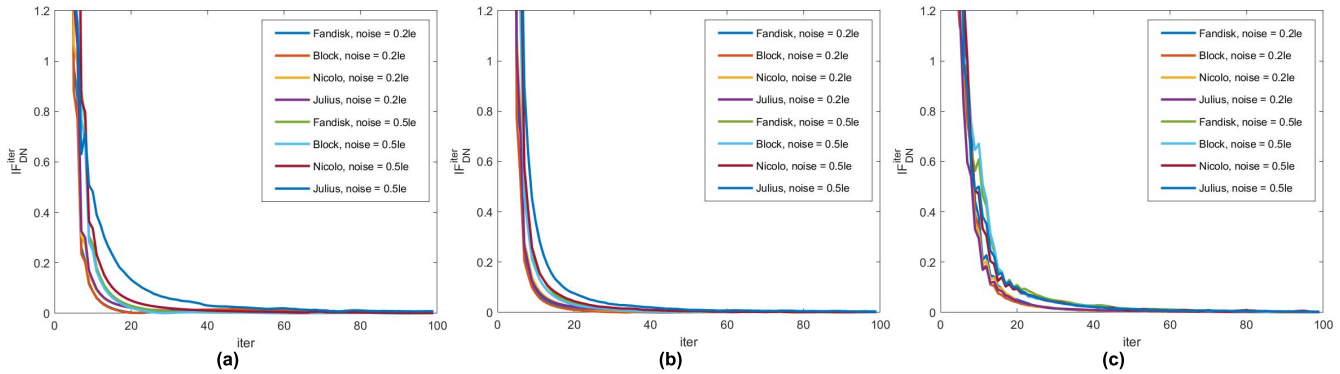

FIGURE 16. IF_{DN}^{iter} of (19) converges as the iteration number increases. (a) OURS+BNF, (b) OURS+GNF, (c) OURS+RHF.

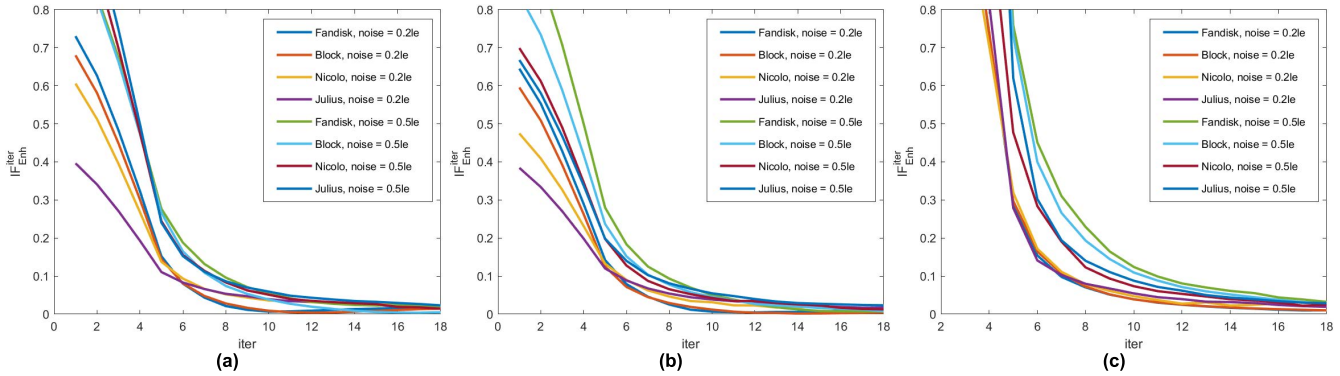

FIGURE 17. IF_{Enh}^{iter} of (19) converges as the iteration number increases. (a) OURS+BNF, (b) OURS+GNF, (c) OURS+RHF.

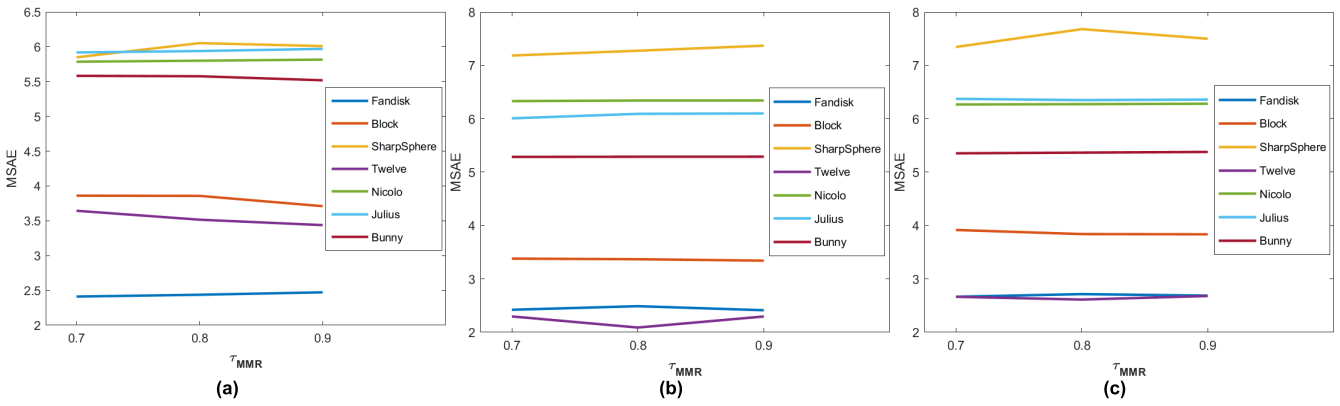

FIGURE 18. Variations of MSAE of meshes denoised by the proposed algorithm for various τ_{MMR}. (a) OURS+BNF, (b) OURS+GNF, (c) OURS+RHF.

areas. In the simulations with the Sharpsphere model, which consists of curved surfaces and edges, the proposed method outperforms the other methods. The results for the Twelve model have a similar trend as those of the Sharpsphere model.

In Figs. 12 and 13, we observe that the L0M method has difficulty in making the curved surface into a set of small flat areas, the NLLR method blurs the edge, and the CNR method does not recover the details of the features. Overall, the results using the proposed method have better performances than each of the L0M, NLLR, and CNR methods.

In Fig. 14, all results denoised by the L0M method have a set of small flat areas, which degrade the visual quality significantly. Regarding removing the noise, NLLR shows the worst performance among the compared algorithms. OURS+BNF and CNR show the best performances. However, CNR does not recover the details of hair above the ear in the Julius model. OURS+GNF and OURS+RHF enhance the features efficiently.

For the Cone and Pyramid models of Fig. 15, L0M method shows good performance. In the results from NLLR,

the edges are slightly blurred and the noise in the flat area is not completely removed. The CNR method removes the noise in the flat area effectively, but the edge feature is considerably blurred. OURS+GNF and OURS+RHF efficiently removes the noise in the smooth surfaces while simultaneously enhancing the edge.

E. CONVERGENCE OF THE PROPOSED ALGORITHM

Figs. 16 and 17 show the convergence patterns of IF^{iter} of (19) for various mesh models, such as Fandisk, Block, Nicolo, and Julius, when the proposed algorithm, respectively incorporating the BNF, GNF, and RHF methods, is applied to remove the noise. In these simulations, we corrupted the test models with low- to high-levels of noise. The x-axis and y-axis in these figures represent the iteration number and IF^{iter}, respectively. The graphs in the figures show that IF^{iter} converges as the iteration number increases. It implies that the change of the normal vectors decreases and IF converges to zero regardless of the characteristics and noise level of the models, as the iteration number increases.

F. ROBUSTNESS OF THE PROPOSED ALGORITHM

This section explains the variations in the performance of the proposed algorithm with changes in the value of τ_{MMR}, which is set by the user.

Fig. 18 shows the MSAEs of the denoised models when the proposed algorithm is applied to the noisy Fandisk, Block, Sharpsphere, Twelve, Nicolo, Julius, and Bunny models with various values of τ_{MMR}. Fig. 18 shows that the performance of the proposed algorithm, respectively integrating BNF, GNF, and RHF method, is not affected by changes in the values of τ_{MMR}.

VI. CONCLUSION

This paper proposes a method to enhance the features in noisy mesh models. We found that the conventional filtering-based methods update the current normal by giving high weights to neighboring normals, which are most similar to the current normal. Due to this procedure, the normals of the neighboring faces having the wrong dominant features affect the update the normal of the current face. This constrains the performances of the filtering-based methods. Conversely, the proposed algorithm drives the target normal vectors and enhances the degraded features based on the target normal vector, while the noise is removed by integrating one of the conventional filtering-based methods. From various simulations, we showed that the proposed algorithm is efficient at both removing the noises and enhancing the features.

REFERENCES

[1] M. Levoy, J. Ginsberg, J. Shade, D. Fulk, K. Pulli, B. Curless, S. Rusinkiewicz, D. Koller, L. Pereira, M. Ginzton, S. Anderson, and J. Davis, "The digital Michelangelo project: 3D scanning of large statues," in *Proc. 27th Annu. Conf. Comput. Graph. Interact. Techn. (SIGGRAPH)*, Jul. 2000, pp. 131–144.

[2] J. Vollmer, R. Mencl, and H. Müller, "Improved Laplacian smoothing of noisy surface meshes," *Comput. Graph. Forum*, vol. 18, no. 3, pp. 131–138, Sep. 1999.

[3] G. Taubin, "A signal processing approach to fair surface design," in *Proc. 22nd Annu. Conf. Comput. Graph. Interact. Techn. (SIGGRAPH)*, 1995, pp. 351–358.

[4] M. Desbrun, M. Meyer, P. Schröder, and A. H. Barr, "Implicit fairing of irregular meshes using diffusion and curvature flow," in *Proc. 26th Annu. Conf. Comput. Graph. Interact. Techn. (SIGGRAPH)*, 1999, pp. 317–324.

[5] X. Liu, H. Bao, H.-Y. Shum, and Q. Peng, "A novel volume constrained smoothing method for meshes," *Graph. Models*, vol. 64, nos. 3–4, pp. 169–182, May 2002.

[6] A. Nealen, T. Igarashi, O. Sorkine, and M. Alexa, "Laplacian mesh optimization," in *Proc. 4th Int. Conf. Comput. Graph. Interact. Techn. Australasia Southeast Asia (GRAPHITE)*, 2006, pp. 381–389.

[7] Z.-X. Su, H. Wang, and J.-J. Cao, "Mesh denoising based on differential coordinates," in *Proc. IEEE Int. Conf. Shape Modeling Appl.*, Jun. 2009, pp. 1–6.

[8] P. Perona and J. Malik, "Scale-space and edge detection using anisotropic diffusion," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 12, no. 7, pp. 629–639, Jul. 1990.

[9] U. Clarenz, U. Diewald, and M. Rumpf, "Anisotropic geometric diffusion in surface processing," in *Proc. Vis. (VIS)*, 2000, pp. 397–405.

[10] Y. Ohtake, A. Belyaev, and I. Bogaevski, "Mesh regularization and adaptive smoothing," *Comput.-Aided Des.*, vol. 33, no. 11, pp. 789–800, Sep. 2001.

[11] Y. Ohtake and E. Belyaev, "Nonlinear diffusion of normals for stable detection of ridges and ravines on range images and polygonal models," in *Proc. Workshop Mach. Vis. Appl.*, 2000, pp. 497–500.

[12] T. Tasdizen, R. Whitaker, P. Burchard, and S. Osher, "Geometric surface smoothing via anisotropic diffusion of normals," in *Proc. IEEE Vis. (VIS)*, Nov. 2002, pp. 125–132.

[13] K. Hildebrandt and K. Polthier, "Anisotropic filtering of non-linear surface features," *Comput. Graph. Forum*, vol. 23, no. 3, pp. 391–400, Sep. 2004.

[14] S. Fleishman, I. Drori, and D. Cohen-Or, "Bilateral mesh denoising," *ACM Trans. Graph.*, vol. 22, no. 3, pp. 950–953, Jul. 2003.

[15] T. R. Jones, F. Durand, and M. Desbrun, "Non-iterative, feature-preserving mesh smoothing," in *Proc. ACM SIGGRAPH Papers*, 2003, pp. 943–949.

[16] C. C. L. Wang, "Bilateral recovering of sharp edges on feature-insensitive sampled meshes," *IEEE Trans. Vis. Comput. Graphics*, vol. 12, no. 4, pp. 629–639, Jul. 2006.

[17] Y. Zheng, H. Fu, O. K.-C. Au, and C.-L. Tai, "Bilateral normal filtering for mesh denoising," *IEEE Trans. Vis. Comput. Graphics*, vol. 17, no. 10, pp. 1521–1530, Oct. 2011.

[18] W. Zhang, B. Deng, J. Zhang, S. Bouaziz, and L. Liu, "Guided mesh normal filtering," *Comput. Graph. Forum*, vol. 34, no. 7, pp. 23–34, 2015.

[19] J. Zhang, B. Deng, Y. Hong, Y. Peng, W. Qin, and L. Liu, "Static/dynamic filtering for mesh geometry," *IEEE Trans. Vis. Comput. Graphics*, vol. 25, no. 4, pp. 1774–1787, Apr. 2019.

[20] S. K. Yadav, U. Reitebuch, and K. Polthier, "Robust and high fidelity mesh denoising," *IEEE Trans. Vis. Comput. Graphics*, vol. 25, no. 6, pp. 2304–2310, Jun. 2019.

[21] G. Arvanitis, A. S. Lalos, K. Moustakas, and N. Fakotakis, "Feature preserving mesh denoising based on graph spectral processing," *IEEE Trans. Vis. Comput. Graphics*, vol. 25, no. 3, pp. 1513–1527, Mar. 2018.

[22] W. Zhao, X. Liu, S. Wang, X. Fan, and D. Zhao, "Graph-based feature-preserving mesh normal filtering," *IEEE Trans. Vis. Comput. Graphics*, vol. 27, no. 3, pp. 1937–1952, Mar. 2021.

[23] Y. Wang, Y. Yang, and Q. Liu, "Feature-aware trilateral filter with energy minimization for 3D mesh denoising," *IEEE Access*, vol. 8, pp. 52232–52244, 2020.

[24] L. He and S. Schaefer, "Mesh denoising via L_0 minimization," *ACM Trans. Graph.*, vol. 32, pp. 64-1–64-8, Nov. 2013.

[25] R. Wang, Z. Yang, L. Liu, J. Deng, and F. Chen, "Decoupling noise and features via weighted ℓ_1-analysis compressed sensing," *ACM Trans. Graph.*, vol. 33, no. 2, p. 18, 2014.

[26] H. Zhang, C. Wu, J. Zhang, and J. Deng, "Variational mesh denoising using total variation and piecewise constant function space," *IEEE Trans. Vis. Comput. Graphics*, vol. 21, no. 7, pp. 873–886, Jul. 2015.

[27] J. R. Diebel, S. Thrun, and M. Brünig, "A Bayesian method for probable surface reconstruction and decimation," *ACM Trans. Graph.*, vol. 25, no. 1, pp. 39–59, Jan. 2006.

[28] P.-S. Wang, Y. Liu, and X. Tong, "Mesh denoising via cascaded normal regression," *ACM Trans. Graph.*, vol. 35, no. 6, pp. 1–12, Nov. 2016.

[29] J. Wang, J. Huang, F. L. Wang, M. Wei, H. Xie, and J. Qin, "Data-driven geometry-recovering mesh denoising," *Comput.-Aided Des.*, vol. 114, pp. 133–142, Sep. 2019.

[30] Z. Li, Y. Zhang, Y. Feng, X. Xie, Q. Wang, M. Wei, and P.-A. Heng, "NormalF-Net: Normal filtering neural network for feature-preserving mesh denoising," *Comput.-Aided Des.*, vol. 127, Oct. 2020, Art. no. 102861.
[31] W. Zhao, X. Liu, Y. Zhao, X. Fan, and D. Zhao, "NormalNet: Learning-based normal filtering for mesh denoising," Mar. 2019, *arXiv:1903.04015*.
[32] H. Fan, Y. Yu, and Q. Peng, "Robust feature-preserving mesh denoising based on consistent subneighborhoods," *IEEE Trans. Vis. Comput. Graphics*, vol. 16, no. 2, pp. 312–324, Mar. 2010.
[33] Z. Bian and R. Tong, "Feature-preserving mesh denoising based on vertices classification," *Comput. Aided Geometric Des.*, vol. 28, no. 1, pp. 50–64, Jan. 2011.
[34] S. Yadav, U. Reitebuch, and K. Polthier, "Mesh denoising based on normal voting tensor and binary optimization," *IEEE Trans. Vis. Comput. Graphics*, vol. 24, no. 8, pp. 2366–2379, Aug. 2017.
[35] M. Wei, H. Jin, X. Xie, L. Liu, and Q. Jing, "Mesh denoising guided by patch normal co-filtering via kernel low-rank recovery," *IEEE Trans. Vis. Comput. Graphics*, vol. 25, no. 10, pp. 2910–2926, Oct. 2019.
[36] X. Li, L. Zhu, C. W. Fu, and P. A. Heng, "Non-local low-rank normal filtering for mesh denoising," *Comput. Graph. Forum*, vol. 37, no. 7, pp. 155–166, Oct. 2018.
[37] L. I. Rudin, S. Osher, and E. Fatemi, "Nonlinear total variation based noise removal algorithms," *Phys. D, Nonlinear Phenomena*, vol. 60, nos. 1–4, pp. 259–268, Nov. 1992.
[38] D. N. H. Thanh, V. B. S. Prasath, L. M. Hieu, and S. Dvoenko, "An adaptive method for image restoration based on high-order total variation and inverse gradient," *Signal, Image Video Process.*, vol. 14, no. 6, pp. 1189–1197, Sep. 2020, doi: 10.1007/s11760-020-01657-9.
[39] D. N. H. Thanh and S. D. Dvoenko, "A mixed noise removal method based on total variation," *Informatica*, vol. 26, no. 2, pp. 159–167, 2016.
[40] S. Zhong, Z. Xie, W. Wang, Z. Liu, and L. Liu, "Mesh denoising via total variation and weighted Laplacian regularizations," *Comput. Animation Virtual Worlds*, vol. 29, nos. 3–4, May/Aug. 2018, Art. no. e1827.
[41] T. Duchamp and W. Stuetzle, "Spline smoothing on surfaces," *J. Comput. Graph. Statist.*, vol. 12, no. 2, pp. 354–381, Jun. 2003.
[42] S. P. Lim and H. Haron, "Surface reconstruction techniques: A review," in *Proc. Artif. Intell. Rev.*, Mar. 2012, pp. 1–20.
[43] X. Sun, P. Rosin, R. Martin, and F. Langbein, "Fast and effective feature-preserving mesh denoising," *IEEE Trans. Vis. Comput. Graphics*, vol. 13, no. 5, pp. 925–938, Sep. 2007.
[44] A. Belyaev and Y. Ohtake, "A comparison of mesh smoothing methods," in *Proc. Bi-Nat'l Conf. Geometric Modeling Comput. Graph.*, 2003, pp. 83–87.
[45] X. Lu, Z. Deng, and W. Chen, "A robust scheme for feature-preserving mesh denoising," *IEEE Trans. Vis. Comput. Graphics*, vol. 22, no. 3, pp. 1181–1194, Mar. 2016.

HYEON-DEOK HAN was born in Daejeon, South Korea, in 1996. He received the B.S. degree in electrical engineering from Sejong University, Seoul, South Korea, where he is currently pursuing the M.S. degree. His research interests include computer vision and 360 VR. In addition, he is proficient in the field of video coding based on high efficiency video coding (HEVC) and versatile video coding (VVC).

JONG-KI HAN was born in Seoul, South Korea, in 1968. He received the B.S., M.S., and Ph.D. degrees in electrical engineering from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, South Korea, in 1992, 1994, and 1999, respectively.

From 1999 to 2001, he was a Member of Technical Staff with the Corporate Research and Development Center, Samsung Electronics Company, Suwon, South Korea. He is currently a Professor with the Department of Electrical Engineering, Sejong University, Seoul. His research interests include image and video coding using high efficiency video coding (HEVC), versatile video coding (VVC), and image coding and processing for 360° virtual reality (VR). He has been participating in the standardization of HEVC and VVC, since 2009 and 2016, respectively.