ResGEM: Multi-scale Graph Embedding Network for Residual Mesh Denoising

Ziqi Zhou, Mengke Yuan†, Mingyang Zhao†, Jianwei Guo, *Member, IEEE,* and Dong-Ming Yan, *Member, IEEE*

Abstract—Mesh denoising is a crucial technology that aims to recover a high-fidelity 3D mesh from a noise-corrupted one. Deep learning methods, particularly *graph convolutional networks* (GCNs) based mesh denoisers, have demonstrated their effectiveness in removing various complex real-world noises while preserving authentic geometry. However, it is still a quite challenging work to faithfully regress uncontaminated normals and vertices on meshes with irregular topology. In this paper, we propose a novel pipeline that incorporates two parallel normal-aware and vertex-aware branches to achieve a balance between smoothness and geometric details while maintaining the flexibility of surface topology. We introduce ResGEM, a new GCN, with multi-scale embedding modules and residual decoding structures to facilitate normal regression and vertex modification for mesh denoising. To effectively extract multi-scale surface features while avoiding the loss of topological information caused by graph pooling or coarsening operations, we encode the noisy normal and vertex graphs using four *edge-conditioned embedding modules* (EEMs) at different scales. This allows us to obtain favorable feature representations with multiple receptive field sizes. Formulating the denoising problem into a *residual learning problem*, the decoder incorporates residual blocks to accurately predict true normals and vertex offsets from the embedded feature space. Moreover, we propose novel regularization terms in the loss function that enhance the smoothing and generalization ability of our network by imposing constraints on normal consistency. Comprehensive experiments have been conducted to demonstrate the superiority of our method over the state-of-the-art on both synthetic and real-scanned datasets.

Index Terms—Mesh Denoising, Multi-scale Embedding, Graph Convolution, Residual Structures, Joint Loss

1 Introduction

With the popularization of depth cameras and the rapid development of scanning techniques, obtaining editable 3D meshes has become more convenient. However, these scanned meshes are often contaminated with noise, making subsequent geometry analysis and geometry processing difficult. To reduce the noise interference and restore the underlying surface information, mesh denoising becomes highly demanding in computer graphics. Notably, it is hard to distinguish fine-scale features from the noise, which poses a high risk of losing accurate surface information while denoising. Consideration should also be given to the gap between the synthetic noise and the real-world noise introduced by 3D scanners or surface reconstruction. Moreover, existing methods usually suffer from generalization to different intensities and distributions of noise.

Numerous methods for denoising meshes have been proposed and can be categorized into *traditional* and *data-driven* approaches. Traditional methods [1], [2], [3] typically involve a leading filtering process for the face normals, followed by updating the vertex positions to obtain a final denoised mesh. However, focusing solely on normal filtering is prone to *over-smoothing*. In recent years, data-driven methods [4], [5], [6], [7], [8], [9] have become popular, while they mostly follow the same two-step pipeline as the traditional ones. On the other hand, vertex positions contain more fine-level feature information, and recovering them directly is apparently a more efficient approach. However, this is less practical because vertex positions are less reliable than face normals in indicating the underlying surface geometry. To circumvent the limitations of using face normals or vertex positions alone, recent works [10], [11] have proposed dual-domain approaches that learn surface representations from both domains simultaneously, achieving promising results. Nevertheless, this concurrent training scheme can lead to excessively regular topology, limiting the ability to represent intricate surface features (see Fig. 18).

To address the above mentioned problems, we propose a novel two-branch pipeline for mesh denoising that takes full advantages of both face normals and vertex positions by operating on the normal-aware and vertex-aware branch sequentially. Our pipeline comprises a preliminary normal prediction process and a further refined normal prediction process, with a vertex modification step parallel to the latter. This approach is advantageous because modifying vertex positions from a smoothed surface is more effective in retaining surface structural patterns, and it allows for independent training of the vertex modification process, resulting in better flexibility maintenance of surface topology. Additionally, for the network training in each branch, regularization terms are integrated into loss functions by constraining on local geometric information such as normal consistency and fidelity, which facilitates the convergence

• *Z. Zhou, M. Yuan, J. Guo and D.-M. Yan are with the State Key Laboratory of Multimodal Artificial Intelligence Systems (MAIS), Institute of Automation, Chinese Academy of Sciences, and the School of Artificial Intelligence, University of Chinese Academy of Sciences, Beijing, China; M. Yuan is also with PIESAT Information Technology Co Ltd, Beijing, China; M. Zhao is with CAIR, Hong Kong Institute of Science & Innovation, Hong Kong, China. E-mails: zhouziqi2022@ia.ac.cn, {mannix.yuan, migyangz, gjianwei.000, yandongming}@gmail.com*

†*Corresponding authors: Mengke Yuan and Mingyang Zhao*

of the denoising network and improves its generalization ability on various surface topology.

Our study primarily focuses on deep learning methods, as traditional methods and simple networks struggle to simulate the highly non-linear function of noise patterns. Deep learning methods use either *patch-to-one* or *patch-to-patch* strategies for training and inference. The input of both strategies are *a patch of* features, but the output is *one* target value or *a patch of* target values respectively. Specifically, for an input mesh patch, the *patch-to-one* approach takes in feature vectors of all patch elements and predicts the target value solely for the central element, which might be the central facet normal or the central vertex position. In contrast, the *patch-to-patch* method simultaneously estimates the target value for every element within the given input patch. Existing *patch-to-one* methods [4], [5], [6], [7], [9] require extracting a neighboring patch for each mesh element, resulting in problems such as normal discontinuities, inaccurate vertex restoration, and computationally intensive inference. Although current *patch-to-patch* methods [8], [10] effectively compute results for all patch elements at once, they involve graph pooling and up-sampling operations to achieve multi-scale receptive field for the network, which sacrifices the topology consistency throughout the intermediary layers, resulting in the omission of crucial surface information, such as fine-scale details represented by very small triangles. Moreover, the computational effort required for generating the coarsened graphs cannot be overlooked.

To realize efficient and topology-consistent (having continuous accessibility to the original surface topological structure across all network layers) *patch-to-patch* mesh denoising, our method investigates a novel multi-scale graph embedding network that enables node-wise feature extraction from neighborhoods of multiple sizes without altering the surface topology and obtains a more fluent denoised surface than *patch-to-one* methods. Additionally, our network architecture strategically balances learning capacity and computational efficiency by employing hybrid convolutional structures in the encoder and decoder. Simplified convolutional units in the encoder efficiently aggregate information from extensive neighborhoods, while the decoder utilizes more complex modules to reconstruct topology and geometry information with enhanced representational abilities. Furthermore, regarding noise removal as a *residual learning problem*, we formalize the decoding part of our network with residual structures to learn the noise patterns from the embedded feature space more precisely and effectively. Therefore, our network is termed ResGEM, which stands for ***Res****idual*, ***G****raph* ***E****mbedding*, and ***M****ulti-scale*. Incorporated into the two-branch pipeline mentioned above, the network shows its edge in both normal-aware and vertex-aware manners with the well-designed joint loss functions.

To summarize, the main technical contributions of this work are as follows:

- We present a novel mesh denoising network named ResGEM, which effectively enables topology-consistent and *patch-to-patch* denoising by combining hybrid graph convolutions.
- We develop a two-branch pipeline to balance surface smoothing and feature recovery while maintaining the topological flexibility of underlying surfaces.
- We design joint loss functions comprising fidelity terms and regularization terms to ensure optimal network performance and superior generalization ability.

Extensive visual and numerical comparisons with state-of-the-art methods demonstrate that the proposed approach achieves advanced denoising results and optimal generalization ability. Additionally, comprehensive ablation studies are provided to validate the rationality and superiority of all design choices.

2 Related Work

2.1 Mesh Denoising

As mentioned in the introduction section, mesh denoising methods can be mainly separated into two categories, traditional methods and learning-based methods.

2.1.1 Traditional Methods

The traditional filtering-based methods [1], [2], [3] try to regress the original normal of each facet by making use of the normal information in the local neighborhood and have achieved reliable results under certain circumstances, but with troublesome parameters tuning. Some methods explore priors such as the non-local similarity [12], [13], [14] or sparsity regularization [15], [16], [17]. In addition, according to Zhu et al. [18], Wei et al. [19] and Yadav et al. [20], with the notion of voting on the surface tensors, a mesh denoising process can be guided to achieve better feature preservation. Although traditional methods have a good performance on various noisy meshes, the tedious parameter adjustment for each model is not simple.

2.1.2 Learning-based Methods

Thanks to the great success learning-based methods have achieved in image denoising, the exploration of them on mesh denoising has been gaining increasing attention. However, different from images, the convolutional operation cannot be directly applied to 3D meshes due to their irregular data structures. Various solutions to this problem have been proposed. At the beginning, hand-crafted feature descriptors [4] or voxel representation [5] were proposed, but they led to inferior performance in terms of accuracy or speed. Later, Li et al [7] proposed a deep learning method regressing directly on the face normals of noisy meshes with the corresponding ground truth meshes, and proved the effectiveness of convolutional neural networks in mesh denoising, yet it ignored the connectivity in local topology. More recently, considering the topological structure on the surface of a mesh model can be naturally regarded as a graph, several works [8], [9], [10], [11] designed diverse types of graph convolutional networks (GCNs) for mesh denoising. Making full use of the local topology information, these methods extraordinarily improved their performance on mesh denoising.

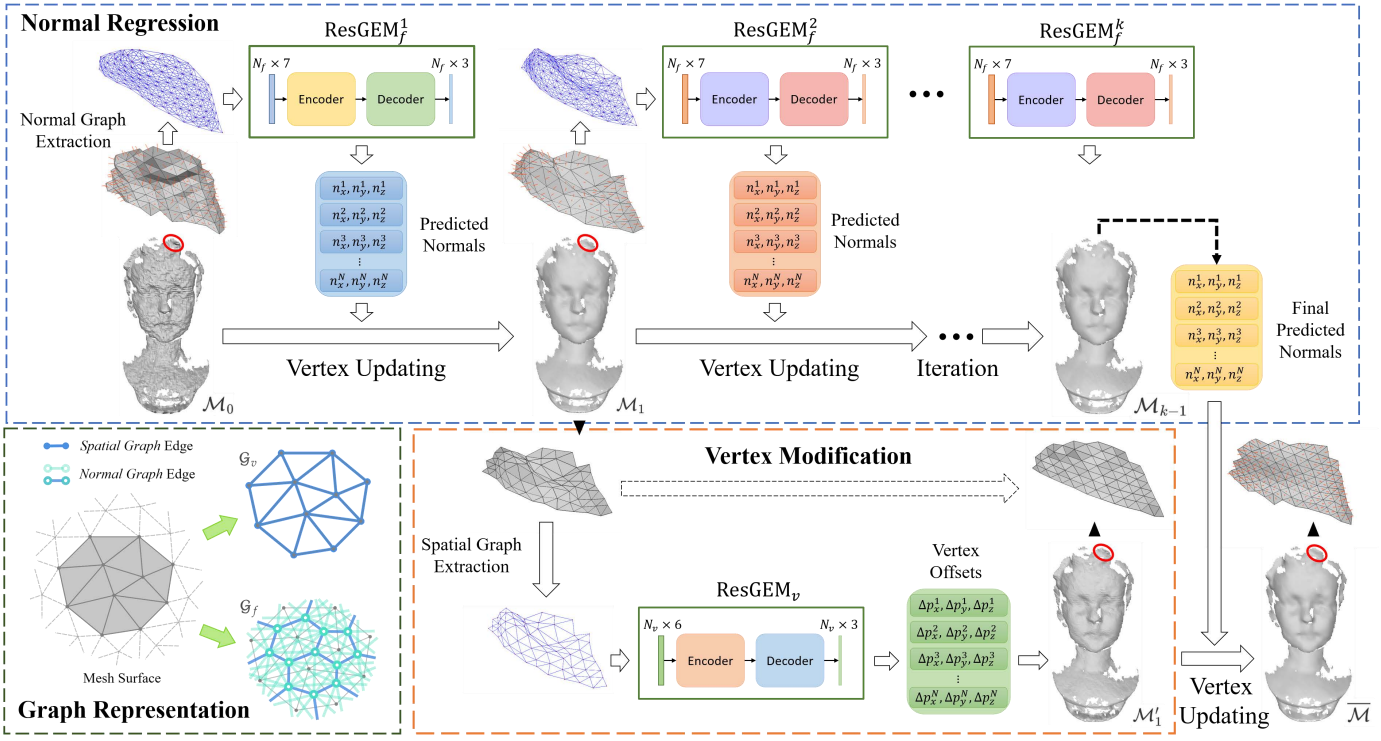


Fig. 1: The two-branch mesh denoising framework based on the proposed network. Each input mesh is converted into multiple graphs and fed into multiple $ResGEM_f$s to compute the denoised normals (normal-aware branch). A vertex updating scheme is performed on the predicted normals from each $ResGEM_f$ to update the mesh surface. Then the updated mesh is fed into the next $ResGEM_f$ similarly. Note that 3 $ResGEM_f$s are employed for normal regression in our implementation and the last vertex updating process is performed on the vertex modification result from $ResGEM_v$, which learns the vertex coordinate offsets of the roughly denoised mesh $\mathcal{M}_1$ to recover the vertex distribution from pure normal filtering (vertex-aware branch). The left bottom area demonstrates a detailed version of the two kinds of graph representations. Note that the edges in the *normal graph* are denoted in two colors to distinguish between edges that connect adjacent faces based on a shared vertex and edges that connect adjacent faces based on a shared edge.

2.2 Graph Convolutional Networks

Graph convolutional networks are one of the common tools for processing irregular data structures such as point clouds [21], [22] and triangular meshes, with a large number of methods developed. Apart from some early works that operated on static graph structures [23], [24], a series of approaches with dynamic graph construction have emerged recently. For instance, Valsesia et al. [25] and Wang et al. [26] (DGCNN) used the k-nearest-neighbors algorithm to dynamically build neighboring nodes at each layer. Zhang et al. [27] combined static and dynamic filtering to mesh geometry for the first time, which was later introduced to the field of mesh denoising by Shen et al. [9] (GCN-Denoiser). Unfortunately, this method adopted a *patch-to-one* strategy which suffers from a surface discontinuity issue and costs much redundant computation during inference. There are also convolutional operators specifically developed for 3D meshes. Hanocka et al. [28] (MeshCNN) performed convolution and pooling on mesh edges while Schult et al. [29] (DualConvMesh-Net) executed that on mesh vertices. Verma et al. [30] (FeaStNet) proposed a novel convolution on the spatial graph structure of mesh surfaces, which was first introduced to mesh denoising by Armando et al. [8] in the normal domain and later extended to simultaneous training on bi-domain (normal and spatial domains) by Zhang et al. [10] (GeoBi-GNN). However, to increase the receptive field for the network trained in a *patch-to-patch* manner, they both have a time-consuming graph pooling operation that causes topology change and may lose important surface information.

3 Method

Overview Fig. 1 shows the overall pipeline of our method. We exploit the graph representations from both normal and spatial (vertex) domains for our network. Given an input noisy mesh model $\mathcal{M}_0$, we first convert it into graph representation (Section 3.1) with each facet as a node and then align its average length to the unit length and its centroid at the origin. Next, we feed the normal graph into our ResGEM network (Section 3.2) to compute the denoised normals of each face. Finally, with predicted face normals, we obtain the denoised mesh through vertex updating (Section 3.4) on the noisy input mesh. Owing to the complexity of noise distributions, we iteratively predict the noise-free surface normals in a cascaded manner to achieve better results, which we denote as the normal-aware branch on the top of Fig. 1. Let $ResGEM_f^i$ denote the network applied in the i-th iteration, $\mathcal{M}_i$ denote the denoised mesh after the i-th iteration. To alleviate the vertex distortion and preserve fine-scale features, we perform vertex modification as a feature

recovery process on $\mathcal{M}_1$ through another ResGEM network represented by $ResGEM_v$, which predicts a position offset vector for each vertex to obtain a modified mesh $\mathcal{M}_1'$, see the parallelizing vertex-aware branch at the bottom. A final combination of the results of the two branches is achieved by vertex updating on mesh $\mathcal{M}_1'$ with the normal prediction result of the last $ResGEM_f$, which is $ResGEM_f^3$ in our implementation. Then a noise-free mesh $\overline{\mathcal{M}}$ with fine-tuned surface topology and rich recovered features is obtained. Note that we magnify the highlighted area (inside the red circle) of the example mesh to exhibit the surface graph representation and thus the change of its normal and topology condition throughout each stage of the pipeline are more distinguishable.

3.1 Data Preparation

3.1.1 Graph Representation

We adopt two kinds of graph representations on a mesh surface for ResGEM to learn both normal regression and vertex modification. A visualized demonstration can be found in the left bottom area of Fig.1.

Vertex representation The original topology structure of the mesh naturally forms a graph, defined as $\mathcal{G}_v = (\mathcal{Q}_v, \mathcal{E}_v, \Phi_v)$, where a graph node $q_v^i \in \mathcal{Q}_v$ is created for each vertex v_i. If v_i and v_j are adjacent, an edge $e = (q_v^i, q_v^j) \in \mathcal{E}_v$ between the corresponding graph nodes q_v^i and q_v^j is built. Φ_v is the set of node features. For each $\phi_v^i \in \Phi_v$ corresponding to vertex v_i, $\phi_v^i = (\mathbf{p}_i^\mathbf{T}, \mathbf{n}_i^\mathbf{T})$, where $\mathbf{p}_i$ and $\mathbf{n}_i$ denote the vertex coordinate and the normal vector of vertex v_i respectively, thus ϕ_v^i is a 6D vector. We use this spatial graph representation for the vertex-aware branch, which is described as the *spatial graph* below.

Normal representation In the normal domain for normal prediction, we build graph structures with faces as nodes, and each facet is connected to its neighbors, which are the faces that have shared vertices with this facet. Let $\mathcal{G}_f = (\mathcal{Q}_f, \mathcal{E}_f, \Phi_f)$ denote the normal graph representation, where a graph node $q_f^i \in \mathcal{Q}_f$ is created for each facet and an edge $e = (q_f^i, q_f^j) \in \mathcal{E}_f$ between node q_f^i and q_f^j would be built if f_i and f_j are adjacent. Φ_f is the set of node features. For each facet f_i, the corresponding feature vector $\phi_f^i = (\mathbf{n}_i^\mathbf{T}, \mathbf{c}_i^\mathbf{T}, a_i)$, where $\mathbf{n}_i$, $\mathbf{c}_i$ and a_i denote the normal vector, centroid position, and area of facet f_i respectively, so ϕ_f^i is a 7D vector. This kind of graph representation is described as the *normal graph* below.

3.1.2 Graph Extraction and Alignment

We extract a series of patch graphs with a fixed number of nodes from the surface of the training meshes to form the training set, driving the network to learn normal prediction or vertex modification. The graph extraction process is realized through iterative neighborhood expansion from located seed points. Given a mesh $\mathcal{M}$, we first locate a set of vertices on it by key points extraction as seeds to generate graphs. Let $\mathcal{Q}_v^i$ and $\mathcal{Q}_f^i$ denote the node set of the *spatial graph* and the *normal graph* grown from vertex v_i respectively. Each pair of $\mathcal{Q}_v^i$ and $\mathcal{Q}_f^i$ will consist of a batch of neighboring vertices and faces and are initially empty. For each seed vertex v_i, we first add itself into $\mathcal{Q}_v^i$ and its adjacent faces into $\mathcal{Q}_f^i$. Whenever there are new nodes in $\mathcal{Q}_v^i$ or $\mathcal{Q}_f^i$, their adjacent nodes should also be inserted. New nodes are kept on put into $\mathcal{Q}_v^i$ and $\mathcal{Q}_f^i$ until the number of elements in them reaches a certain number N. To avoid the influence of scale differences, for all generated graphs from the same mesh, we set their centroids to the origin, and scale each of them so that the average edge length of the mesh is unit length.

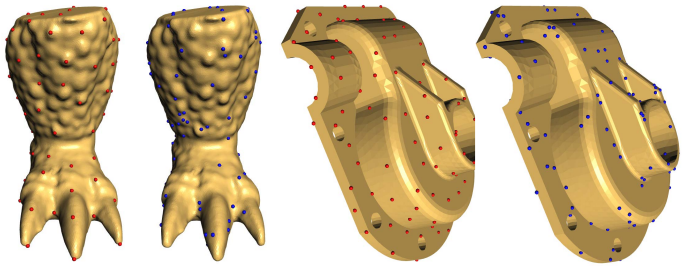


Fig. 2: The results of key points detection (depicted as red points) is compared to the results of random farthest point sampling (FPS) (depicted as blue points) on the *Leg* model (left) and the *Casting* model (right). The Harris 3D algorithm effectively covers areas with both sharp and smooth features. In contrast, FPS samples the surface unevenly, resulting in some missed sharp features such as the claw tips.

3.1.3 Key Points Based Training Set Generation

To make the training data cover as many surface patterns as possible so that the network can better learn surface features, for each mesh model in the training set, we first identify a series of key points on the noise-free surface, and then use each of them as a seed point for two kinds of graph extractions on both the noisy model and ground-truth model. Plenty of graph pairs are generated to form the training set and the corresponding ground-truth set.

Harris 3D key points extraction The key point locations are determined by leveraging a 3D interest points detector called *Harris 3D* [31], which is a 3D generalization of the Harris interest points detector for images [32]. The calculation steps are as follows:

1) Pick the *k*-ring neighborhood of each vertex v_i in mesh $\mathcal{M}$, and let $V_k(v_i)$ denote the point set composed of vertices within k rings around v_i;
2) For each vertex v_i, set the centroid of $V_k(v_i)$ to the origin, and align the normal of its best fitting plane (the eigenvector with the lowest associated eigenvalue after *Principal Component Analysis*) to *z-axis*;
3) Fit a quadratic surface to the set of transformed vertices $\overline{V}_k(v_i)$ through a paraboloid function with six terms:
$$z = f(x,y) = \frac{p_1}{2}x^2 + p_2xy + \frac{p_3}{2}y^2 + p_4x + p_5y + p_6, \quad (1)$$
4) Calculate the Harris operator value of vertex v_i by:
$$h(v_i) = det(E) - k(tr(E))^2, \quad (2)$$
where $E = \begin{pmatrix} A & C \\ C & B \end{pmatrix}$, $A = p_4^2 + 2p_1^2 + 2p_2^2$, $B = p_5^2 + 2p_2^2 + 2p_3^2$ and $C = p_4p_5 + 2p_1p_2 + 2p_2p_3$;

Algorithm 1 Interest Points Clustering

Input: Set P of pre-selected interest points in decreasing order of Harris operator value
Output: Final set of interest points

Let Q be a set of points
function KEYPOINTSCLUSTER(Q)
  $Q \leftarrow \emptyset$
  for $i = 1 \rightarrow |P|$ **do**
    if $\min_{j \in [1,|Q|]} \|P_i - Q_j\|_2 > \rho$ **then**
      $Q \leftarrow Q \cup \{P_i\}$
    end if
  end for
  return Q
end function

5) Select the interest points by the Harris operator values:
(a) Preserving the vertices with local maximum Harris response. Select the vertices which hold the condition that $h(v) > h(w), \forall w \in V_k(v)$ as candidate key points;
(b) Obtaining good distribution of interest points on the mesh surface through Interest Points Clustering. Sort the pre-selected points by their Harris operator values in a descending order, then cluster the sorted points by Alg. 1 to get the final set of key points, where ρ can be considered as the scale of clustering which is usually a fraction of the diagonal of the object bounding box and can impact the number of interest points returned.

According to the above steps, the interest points detection result of the *Leg* model and the *Casting* model is shown via red points in Fig. 2 ($\rho = 0.08 l_d$, $k = 2$, where l_d is the length of the diagonal of the model's bounding box). Comparing with the random farthest point sampling (FPS) algorithm [33], it can be seen that the interest points calculated through the Harris 3D algorithm cover not only the areas with sharp features, but also smoother ones, guaranteeing the diversity of the generated graph set for training, therefore making our network more faithful to all kinds of surface features. In contrast, the random FPS approach can result in uneven sampling, inadequately covering all surface types. It may also generate neighboring sample points, leading to overlap in the extracted surface patches, which renders the training set redundant and less effective for feature learning.

3.2 Network Architecture

We use the same network architecture ResGEM to learn both normal regression and vertex modification. As shown in Fig. 3, the ResGEM network consists of two parts: An encoder and a decoder. Taking into account the trade-off between the learning capacity and computational complexity associated with different convolutional structures, we use distinct convolution blocks in these two parts. The encoding part combines a static edge-conditioned embedding module (EEM) with multi-scale dynamic EEMs to extract geometric features from different spatial scales without graph pooling or coarsening operations that might drop important surface

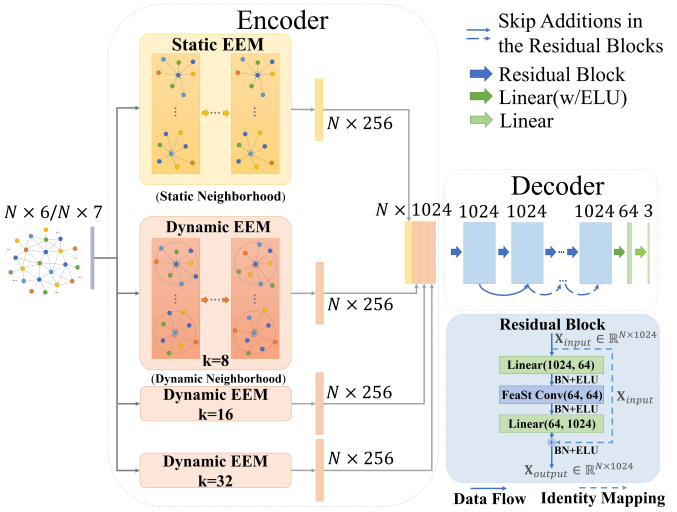


Fig. 3: Our ResGEM architecture. The encoder consists of four parallel edge-conditioned embedding modules (EEMs) to capture multi-scale local information while the decoder employs residual graph convolutional blocks to learn noise patterns.

information. Meanwhile, for the decoding part, we consider noise pattern learning as an iterative residual learning process and design stacked residual structures over a graph convolution operator [30] to exploit the representation power of our network. Detailed explanations can be found in the *Supplementary Material* about the design of the network.

3.2.1 Encoder

Our encoder consists of multiple edge-conditioned embedding modules (Fig. 4), which adopt the Edge-Conditioned Convolution (ECC) strategy [24], [34] to encode graph features. A detailed introduction about the corresponding convolution unit can be found in the *Supplementary Material*.

In analyzing mesh surfaces, one must be cognizant of the complex relationship between geometric topology and surface connectivity. Due to the lack of a one-to-one correspondence between these two concepts, relying solely on the original graph structure of the mesh may result in some loss of information during convolution, hindering comprehensive feature learning on graph nodes. To address this issue, one solution often employed is using graph pooling layers to generate coarsened graphs [8], [10], thereby establishing links between graph nodes that were not previously directly connected. This method can facilitate a more nuanced understanding of the graph's interconnectedness, but it may still compromise fine-scale information from the original graph topology. To efficiently denoise surfaces while preserving their structure, we propose a novel approach that encourages the network to learn multi-scale information for each node in the input patch graph simultaneously. This enables the use of a *patch-to-patch* strategy without altering the inherent surface topology. Two types of feature embedding modules are designed with their structures demonstrated in Fig. 4. The static edge-conditioned embedding module (static EEM) implements convolutions on the original static

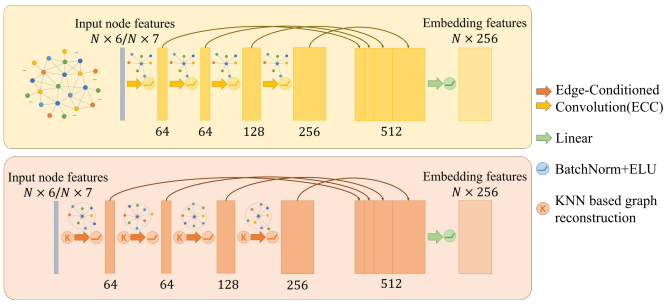


Fig. 4: The structures of (top) static and (bottom) dynamic edge-conditioned embedding modules (EEMs).

graph structure, while the dynamic EEM uses a dynamic graph construction scheme to allow different numbers of not adjacent but nearby graph nodes to be connected. The neighborhood of each node varies at each layer according to the proximity in feature space, which is calculated through the k-nearest-neighbors (KNN) algorithm. Three dynamic EEMs are adopted and the values of ks are set to $(8, 16, 32)$ to enrich the receptive fields of graph nodes and better extract local geometric information under multiple scales. Inside each EEM, the outputs from all hidden layers are concatenated and fed into a fully connected layer to further increase the embedding scale diversity. As different numbers of neighbors are connected, both fine-scale and coarse-scale local information are taken into account, with no need of discarding any graph node or topology structure. As shown in Fig. 1, the outputs of the four EEMs are concatenated before flowing to the downstream network. The final output is a feature matrix $\mathbf{X} \in \mathbb{R}^{N\times F}$, where N denotes the number of nodes and F denotes the dimension of features which equals to 1024 in our implementation.

3.2.2 Decoder

After embedding the noisy surface into a multi-level feature space, we transform the denoising problem into an iterative residual learning task for our decoder, which is piled up with a series of identical residual blocks followed by MLPs, see Fig. 3. The k-th block is in the form of the following equation, performed by a shortcut connection and an element-wise addition:

$$\mathbf{X}^k_{output} = \sigma(\mathcal{F}^k_\Theta(\mathbf{X}^k_{input}) + \mathbf{X}^k_{input}), \tag{3}$$

where $\mathcal{F}^k_\Theta$ is the residual function, σ includes the operation of batch normalization and activation function, and $\mathbf{X}^k_{input}$ and $\mathbf{X}^k_{output}$ represent the input and output of the k-th residual block respectively, which are of the same dimensions as the output feature map $\mathbf{X} \in \mathbb{R}^{N\times F}$ from the encoder. The components of the residual function $\mathcal{F}$ are flexible, and it has already been proven to be highly efficient with regular convolutional layers [35] in image-related tasks. We believe that it is also applicable to graph convolution operations for mesh denoising, as it is easier for the solver to find the noise pattern with an identity mapping than barely learning one. The effectiveness of our residual graph convolution structures is verified by extensive experiments demonstrated in the following experiments. The detailed formulation of each residual function $\mathcal{F}^k_\Theta$ is also exhibited in Fig. 3, including two linear layers for dimension transformations and a graph convolutional layer in between. Note that the graph convolution layers (*FeaSt Conv*) we adopt as the core units of our residual blocks is an extension of the regular convolution on 2D images, which is introduced to mesh surfaces by [30]. A more concrete introduction to this kind of graph convolution is presented in the *Supplementary Material*. In our implementation, there are four residual blocks in the decoder.

3.3 Loss Function

To optimize the performance of the network, we integrate regularization terms into the loss functions for both normal and vertex predictions. These regularization terms act as constraints on local geometric information, ensuring that the predicted normals and vertices remain consistent with the underlying surface geometry.

3.3.1 Loss Function for Normal Regression

We design a normal joint loss function with a regularization term in the normal-aware branch. 1) *normal recovery loss* and 2) *smooth regularization loss*.

Normal recovery loss For an input normal graph, we use the L_2 loss on the normal vectors to encourage the predicted facet normals to be consistent with the ground-truth normals:

$$L_n = \frac{1}{N_f}\sum_{i=1}^{N_f} \|\bar{\mathbf{n}}_i - \mathbf{n}_i^{gt}\|^2, \tag{4}$$

where N_f is the number of face nodes in the input *normal graph*, $\bar{\mathbf{n}}_i$ and $\mathbf{n}_i^{gt}$ are the predicted normal vector and the ground-truth normal vector of facet f_i respectively.

Smooth regularization loss Inspired from the total variation loss (TV Loss) [36] in image denoising, we design a smooth regularization term L_{smooth} to minimize the average square distance between normal vectors of all adjacent face nodes in the input graph, so that the smoothing ability of our network can be further enhanced:

$$L_{smooth} = \frac{1}{N_e}\sum_{(q_i,q_j)\in\mathcal{E}} \|\bar{\mathbf{n}}_i - \bar{\mathbf{n}}_j\|^2, \tag{5}$$

where N_e is the edge number of the input *normal graph* (the number of adjacent facet pairs). q_i, q_j are the corresponding graph nodes of facet f_i and f_j, then $(q_i, q_j) \in \mathcal{E}$ means facet f_i and f_j are adjacent.

Normal joint loss Overall, we formulate the joint loss function for normal regression as a combination of the above two functions:

$$L = L_n + \lambda L_{smooth}, \tag{6}$$

where the parameter is empirically set as: $\lambda = 0.01$.

3.3.2 Loss Function for Vertex Modification

As we formulate the vertex-aware branch a vertex position modification process for feature recovery and topology reconstruction, we propose to learn the coordinate offsets for the vertices rather than directly regressing the denoised positions, with experiments proving that the network converges faster and performs better on the former (Section 4.4). For training we adopt a vertex joint loss with two terms as

well, including a vertex fidelity term and a normal fidelity term as regularization, namely 1) *vertex recovery loss* and 2) *normal regularization loss*.

Vertex recovery loss We apply the L_2 norm of the distance from the modified vertex coordinate to the ground-truth coordinates to encourage the predicted vertex offsets to be as precise as possible:

$$L_v = \frac{1}{N_v}\sum_{i=1}^{N_v}\|(\mathbf{p}_i + \Delta\mathbf{p}_i) - \mathbf{p}_i^{gt}\|^2, \tag{7}$$

where N_v is the number of vertices in the input *spatial graph*, $\Delta\mathbf{p}_i$ is the offset of vertex v_i predicted by the network, and $\mathbf{p}_i$ and $\mathbf{p}_i^{gt}$ are the original and ground-truth coordinates of vertex v_i respectively.

Normal regularization loss Taking the advantage of the graph structure we build with the surface topology, for each input *spatial graph* patch, in addition to necessary vertex information, the ground truth normal of each facet and the indices of its three correlated vertices are also recorded. Thus we can calculate the normal fidelity of each facet in the patch accordingly as a normal regularization term to strengthen the surface geometric constraint for the network:

$$L_{normal} = \frac{1}{N_f}\sum_{i=1}^{N_f}\|normalize(\tilde{\mathbf{n}}_i) - \mathbf{n}_i^{gt}\|^2, \tag{8}$$

where N_f is the number of faces in the input *spatial graph*, $\mathbf{n}_i^{gt}$ is the recorded ground truth normal of the *i*-th facet and $\tilde{\mathbf{n}}_i$ is obtained by solving the following function:

$$(\mathbf{e}_{i1}, \mathbf{e}_{i2})^{\mathbf{T}}\tilde{\mathbf{n}}_i = \mathbf{0}, \tag{9}$$

where $\mathbf{e}_{i1} = \tilde{\mathbf{p}}_{i_0} - \tilde{\mathbf{p}}_{i_1}$ and $\mathbf{e}_{i2} = \tilde{\mathbf{p}}_{i_0} - \tilde{\mathbf{p}}_{i_2}$ are the two edge vectors of the *i-th* triangular facet with $i_k(k = 0, 1, 2)$ being the vertex index of the *k*-th vertex of it, $\tilde{\mathbf{p}}_{i_k} = \mathbf{p}_{i_k} + \Delta\mathbf{p}_{i_k}$ is the corresponding predicted vertex position. Specifically, $\tilde{\mathbf{n}}_i$ can be calculated by $\tilde{\mathbf{n}}_i = (y_{i1}z_{i2} - y_{i2}z_{i1}, z_{i1}x_{i2} - z_{i2}x_{i1}, x_{i1}y_{i2} - x_{i2}y_{i1})$ with $\mathbf{e}_{i1} = (x_{i1}, y_{i1}, z_{i1})$ and $\mathbf{e}_{i2} = (x_{i2}, y_{i2}, z_{i2})$.

Vertex joint loss Overall, we formulate the joint loss function for vertex modification as a combination of the above two functions:

$$L = L_v + \lambda L_{normal}, \tag{10}$$

where the parameter is empirically set as: $\lambda = 0.01$.

3.4 Inference

This section presents the implementation details of the proposed method. The *patch-to-patch* strategy is adopted to enable inference on each facet of the input mesh or patch simultaneously. Given a test noisy mesh, the trained network $ResGEM_f^1$ is used to predict the noise-free normals of its faces. A roughly denoised mesh $\mathcal{M}_1$ is then obtained after vertex updating, as explained in the following paragraph, on the original noisy surface. Next, the vertex-aware branch of the proposed method applies a $ResGEM_v$ to perform vertex modification, while the normal-aware branch employs two $ResGEM_f$s to predict more accurate normal vectors. Finally, the final denoised mesh $\overline{\mathcal{M}}$ is generated by updating the vertices on the vertex-modified mesh $\mathcal{M}_1'$ from the vertex-aware branch with the facet normals predicted by $ResGEM_f^3$. To address the limited computing power of a single GPU, meshes with more than 20k faces or vertices are segmented into several parts, each containing 20k faces or vertices. Normal prediction or vertex modification is performed separately for each part, and the predicted results are merged back into the original mesh. The results on the overlapped areas are averaged to improve performance on the edge of each split patch. The impact of the input patch size on the network's performance is found to be negligible, as verified in the ablation study (Section 4.4).

Vertex updating To obtain a denoised mesh surface with consistent face normals and vertex positions, an iterative process is applied to the noisy mesh and its inferred noise-free normals for each face. The process involves updating the positions of the face vertices iteratively until the final denoised mesh surface is obtained. This vertex updating scheme is based on a method described in [37], and is described as follows:

$$\tilde{\mathbf{x}}_i = \mathbf{x}_i + \frac{1}{|F_v(i)|}\sum_{k\in F_v(i)}\bar{\mathbf{n}}_k(\bar{\mathbf{n}}_k \cdot (\mathbf{c}_k - \mathbf{x}_i)), \tag{11}$$

where $\mathbf{x}_i$ and $\tilde{\mathbf{x}}_i$ are the positions of vertex v_i before and after updating respectively, $F_v(i)$ is the set of indices of faces in the 1-ring neighborhood of vertex v_i, $\bar{\mathbf{n}}_k$ is the predicted normal vector of the *k*-th neighbor facet and $\mathbf{c}_k$ is the centroid of it. In our experiment, the iteration number for each test mesh is set to 30.

4 Experimental Results

In this section, we introduce the datasets used for training and testing and the evaluation details of our method. Then the proposed network is validated and evaluated from both quantitative and qualitative analysis, as well as the comparison with state-of-the-art mesh denoising methods and ablation studies.

4.1 Datasets

To obtain more reliable and practical denoise results, our experimental datasets contain both synthetic and real-scanned data. We build our training datasets based on the models provided by [4], including 1) *Synthetic dataset*; 2) *Kinect v1 dataset*; 3) *Kinect v2 dataset* and 4) *Kinect Fusion dataset*. Detailed descriptions of these four datasets are as follows.

4.1.1 Synthetic Dataset

There are 21 synthetic models for training in total, consisting of CAD models (CAD), smooth models (smooth) and models with rich fine-scale features (feature). For each model, we synthesize three levels of Gaussian noise (the deviations are set to 0.1, 0.2 and 0.3 of the average edge length $\bar{l}_e$ of each mesh) for training. After interest points detection (Section 3.1, $\rho = 0.01$, $k = 2$, the same below), 800 to 2000 points are detected for each model, and then about $100k$ graphs in both spatial and normal domain are generated as the training set, with $N = 1000$ nodes in each graph.

4.1.2 Real-scanned Dataset

Real-scanned models are all obtained from meshes scanned by Microsoft Kinect. Three types of scanning results are included:

- *Kinect v1 dataset.* There are 71 frames from four models scanned by Kinect v1, generating about $80k$ graphs after interest points detection and graph extraction, with $N = 1000$ faces or vertices in each graph.
- *Kinect v2 dataset.* There are 72 frames from four models scanned by Kinect v2, generating about $60k$ graphs after interest points detection and graph extraction, with $N = 1000$ faces or vertices in each graph.
- *Kinect Fusion dataset.* There are 3 meshes scanned by Microsoft Kinect V1 via Kinect-Fusion technique [38], generating $6k$ patches after interest points detection and patch extraction, with $N = 1000$ faces in each patch.

4.1.3 Test Data

Corresponding to the above four datasets, we also adopt the test models provided in [4] to form our testing sets, which are all invisible for the network. For the *Synthetic dataset*, three levels of Gaussian noise are added to 29 test meshes, including 14 CAD models, 7 smooth models and 8 models with rich features. For real-scanned data, there are four meshes in the *Kinect Fusion dataset*: *Boy01*, *Boy02*, *David* and *Pyramid*; and there are 73 and 72 frames of scanned meshes in the *Kinect v1* and the *Kinect v2 dataset* respectively, which come from four statues: *Boy* (24 frames), *Cone* (12 frames), *Girl* (25 frames in the *Kinect v1 dataset* and 24 frames in the *Kinect v2 dataset*) and *Pyramid* (12 frames). Several real-scanned models without corresponding ground-truths and models with extremely intensive or impulsive noise are also taken into account to test the generalization ablility of our method (Section 4.3.3).

4.2 Evaluation Strategy

4.2.1 Comparison Methods

We compare our method both qualitatively and quantitatively with other mesh denoising methods, including traditional methods and state-of-the-art learning-based methods. Traditional methods contain guided normal filtering (GNF) [3] and non-local low-rank normal filtering (NLLR) [12], and the learning-based methods consist of cascaded normal regression (CNR) [4], facet graph convolutions (FGC) [8], NormalF-Net (NFN) [6], GCN-Denoiser (GCN) [9] and GeoBi-GNN (BGNN) [10].

The source code or pre-trained models of most of the above methods are provided by their authors or implemented by a third party. For NFN [6] and BGNN [10] that have no publicly available or runnable source code, their results are kindly provided by the authors.

4.2.2 Experimental Settings

With respect to the high distinction between noise patterns, the training for different datasets are conducted separately. For real-scanned models, the corresponding ground-truth models are also provided by [4]. In addition, we apply some random rotations to each graph before training to realize data augmentation and encourage our network to learn the permutation invariance. Notably, with the extracted graph representations, a node number of 1000 is sufficient for our network to learn the local noise pattern and inference a mesh or patch of any size with equal performance (Fig. 13 (a)).

We implement our method with the deep learning framework *Pytorch* and *Pytorch Geometric* [39], and adopt the Adam optimizer with a learning rate scheduler starting from 0.001. All experiments are conducted on a server with eight GeForce RTX 3080Ti GPUs. It takes about 20 hours for training the synthetic models and around 10 hours for the real scanned models on both normal regression and vertex modification. The geometric repository *Easy3D* [40] is employed for topology search and input data generation.

4.2.3 Error Metrics

We adopt two common metrics, namely 1) *mean angular error (MAE)* and 2) *mean vertex error (MVE)*, to assess our method and quantitatively compare it with state-of-the-art approaches. Detailed definitions can be found as follows.

- *Mean angular error (MAE).*

$$E_a = \frac{1}{N_f} \sum_{f_i \in \mathcal{M}} \arccos(\bar{\mathbf{n}}_i \cdot \mathbf{n}_i^{gt}), \tag{12}$$

  where E_a is the mean angular error (in degree) between corresponding face normals of a denoised mesh and the ground-truth mesh (the smaller, the better), $\bar{\mathbf{n}}_i$ and $\mathbf{n}_i^{gt}$ are the denoised normal and original noise-free normal of facet f_i respectively.
- *Mean vertex error (MVE).*

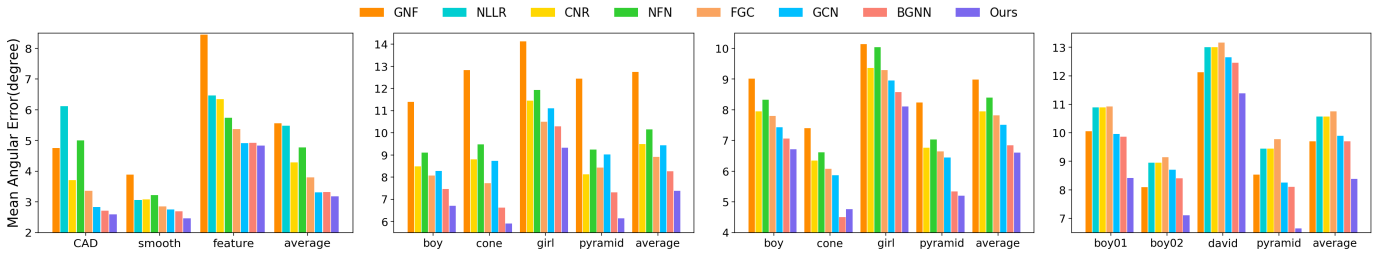
$$E_v = \frac{1}{N_v L_d} \sum_{v_i \in \mathcal{M}} |\bar{\mathbf{p}}_i - \mathbf{p}_i^{gt}|, \tag{13}$$

  where E_v is the mean vertex position error between corresponding vertices of a denoised mesh and the ground-truth mesh (the smaller, the better), L_d is the length of the diagonal of the bounding box of the mesh, $\bar{\mathbf{p}}_i$ and $\mathbf{p}_i^{gt}$ are the denoised coordinate and original noise-free coordinate of vertex v_i respectively.

4.3 Comparison with State-of-the-art Approaches

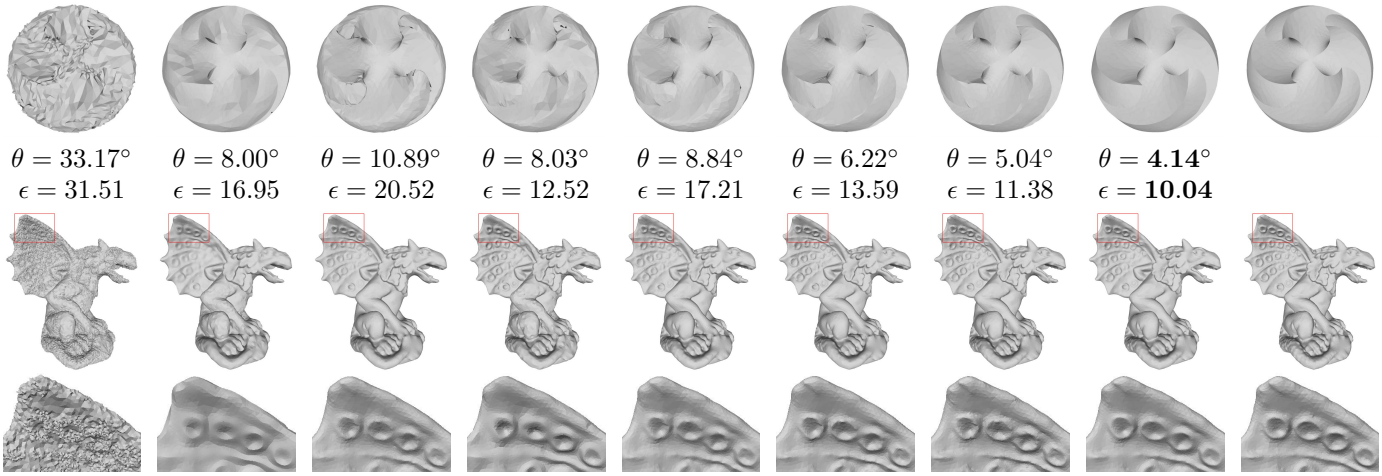
4.3.1 Results on Synthetic Models

The objective comparison of the average angular error on the *Synthetic dataset* is demonstrated in Fig. 5 (a), where *Ours* denotes the denoised results with our proposed pipeline. As illustrated, our method achieves the best performance in all three categories. Fig. 6 visualizes the qualitative comparison of different methods on several representative synthetic models with their MAE and MVE values listed underneath. The results of NLLR [12] and NFN [6] are similar because they are both based on non-local similarity. CNR [4] has disadvantages in feature preserving owing to its relatively simple network architecture. As for GCN-based methods including FGC [8], GCN [9], BGNN [10] and our method,



(a) Synthetic (b) Kinect v1 (c) Kinect v2 (d) Kinect Fusion

Fig. 5: The comparison result of mean angular error on synthetic and real-scanned datasets. Note that the source code provided by the authors of NLLR [12] can only handle watertight meshes, so it is not included in *Kinect v1* and *Kinect v2* comparison. The authors of NFN [6] did not provide the results on the *Kinect Fusion dataset*.



(a) Noisy (b) GNF (c) NLLR (d) CNR (e) NFN (f) FGC (g) BGNN (h) Ours (i) GT

Fig. 6: Visual comparison of different methods [3], [4], [6], [8], [10], [12] on the *Synthetic dataset*, including model (top) *SharpSphere* and (bottom) *Gargoyle* with the Gaussian noise level $0.3l_e$. Note that the θ and ϵ values underneath represent the MAE and 10^4 times of MVE values of the models respectively. Due to limited space, FGC [8] and BGNN [10] are selected as representatives for GCN-based methods.

the methods that adopt graph convolutional learning on the spatial graph representation of mesh surfaces (BGNN [10] and our method) perform better on fine-scale feature recovering than the others. With residual structures in the decoder, our network learns the noise pattern easier and more efficiently with the support of an identity mapping. Moreover, the well-designed encoder architecture allows our network to extract multi-scale geometric features dynamically from the mesh surfaces to better distinguish the underlying details and the noise, which proves to be an effective alternative for the graph pooling operations applied in FGC [8] and BGNN [10].

4.3.2 Results on Real-scanned Models

Fig. 5 (b-d) illustrates the quantitative comparison of various methods on the real-scanned datasets. Since the source code provided by the authors of NLLR [12] can only handle watertight meshes, it is not included in the *Kinect v1* and *Kinect v2* comparison. The results of NFN [6] on the *Kinect Fusion dataset* are also not provided by the authors. As demonstrated, our method outperforms all other methods in all categories on the *Kinect Fusion* and *Kinect v1 dataset* and achieves optimal average result on the *Kinect v2 dataset*. The rendering results of four meshes are visualized in Fig. 7 and Fig. 8. The real-scanned meshes have much more complex noise distribution than the synthetic models, especially for the models in the *Kinect Fusion* and *Kinect v1 dataset*, which have stair-shaped noisy surfaces. However, quantitative results tell that our residual structured network performs extraordinarily well on them. The filtering-based traditional method GNF [3] is likely to over smooth the regions with fine-scale features while mistakenly treating the noise on the original plane surfaces as pseudo-features, having difficulty distinguishing the noise and underlying features. Most learning-based methods perform better than the traditional ones, but some surface details are still over-smoothed inevitably. As can be observed, our method denoises the large plane regions while preserving sharp features, such as edges in the *Pyramid02* model, the most faithfully to the ground truth. It also recovers the mouth and nose area on the *Boy02* model to the most, which is largely attributed to the vertex-aware branch in the denoising pipeline. More visualized

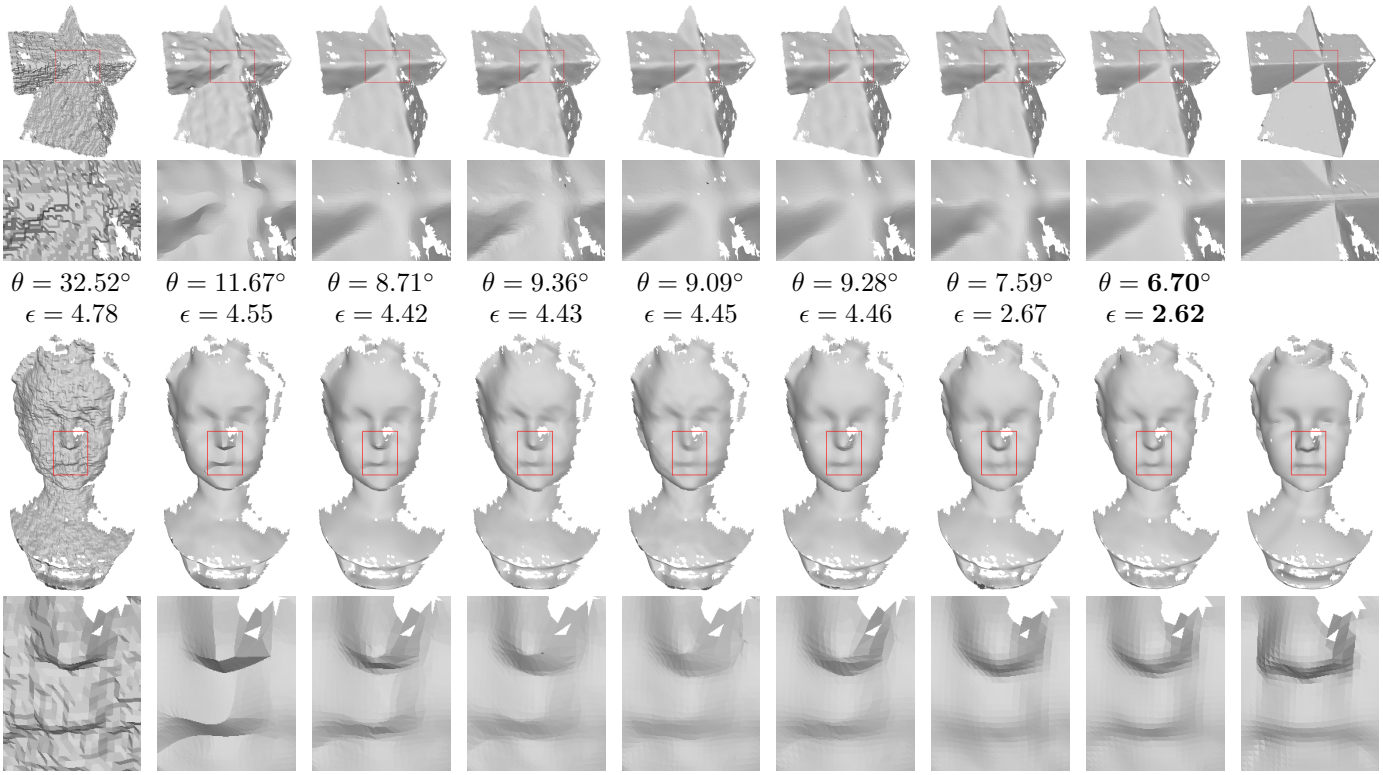


Fig. 7: Visual comparison of different methods [3], [4], [6], [8], [9], [10] on the real-scanned dataset, including the (top) *Pyramid02* model in the *Kinect v1 dataset* and the (bottom) *Boy02* model in the *Kinect v2 dataset*, where θ represents the value of the mean angular error and ϵ denotes the value of the mean vertex error magnified by 10^3.

TABLE 1: Running time comparisons (in seconds) on synthetic models with implementable methods.

Model	Faces Vertices	GNF [3]	NLLR [12]	CNR [4]	FGC [8]	GCN [9]	Ours
Gargoyle	171112 85558	581	36	6	160	146	93
Sharp_sphere	20882 10443	66	4	1	24	12	6
Eros100K	100000 50002	329	20	3	91	60	37
Fertility	27954 13971	85	7	1	33	13	8

comparsion can be found in the *Supplementary Material*.

4.3.3 Results on Unseen Noise Patterns

To verify the advanced generalization ability of our method, we further conduct experiments on models with unseen noise patterns such as extremely high-level of Gaussian or impulsive noise (level 0.6) and real-scanned noise for the network trained on the *Synthetic dataset* with only relatively low levels of Gaussian noise (level 0.1, 0.2 and 0.3). For the intense Gaussian and impulsive noise, the normal angular error heatmaps comparison of the denoised results of two models (only methods with runnable source code are included) are visualized in Fig. 16. As is demonstrated, our method outperforms the others both quantitatively and qualitatively. As for the *Fandisk* model corrupted by a high level of Gaussian noise, the sharp edges and the cylindrical surfaces are hardly recovered precisely by previous methods, however, our method achieves unprecedentedly high fidelity with elaborated network structure and pipeline. The fine-scale features of the *Nicolo* model under intense impulsive noise are preserved the best by our method as well. CNR [4] fails to identify some of the original surface features from the impulsive noise and FGC [8] suffers from the issue of large vertex displacements. Besides, it can be observed that the problem of surface discontinuity hinders the feature recovery effect of the method with *patch-to-one* strategy (GCN [9]). Several real-scanned models are also tested with the network model pre-trained on the *Synthetic dataset*, including toy models (Fig. 9) and scene-level outdoor meshes (Fig. 10). Note that these real-scanned models have no corresponding ground truths. It can be seen that our method finds the best balance between surface smoothing and feature recovery, and produces denoised results with the highest quality, presenting outstanding generalization capability over its competitors. Due to space limitation, more qualitative comparisons on unseen noise levels or distributions can be found in the *Supplementary Material*.

4.3.4 Running Time

Our method outperforms previous state-of-the-art GCN-based methods in terms of running efficiency, thanks to the adopted *patch-to-patch* strategy and the delicate network architecture. Tab. 1 demonstrates the denoising time compar-

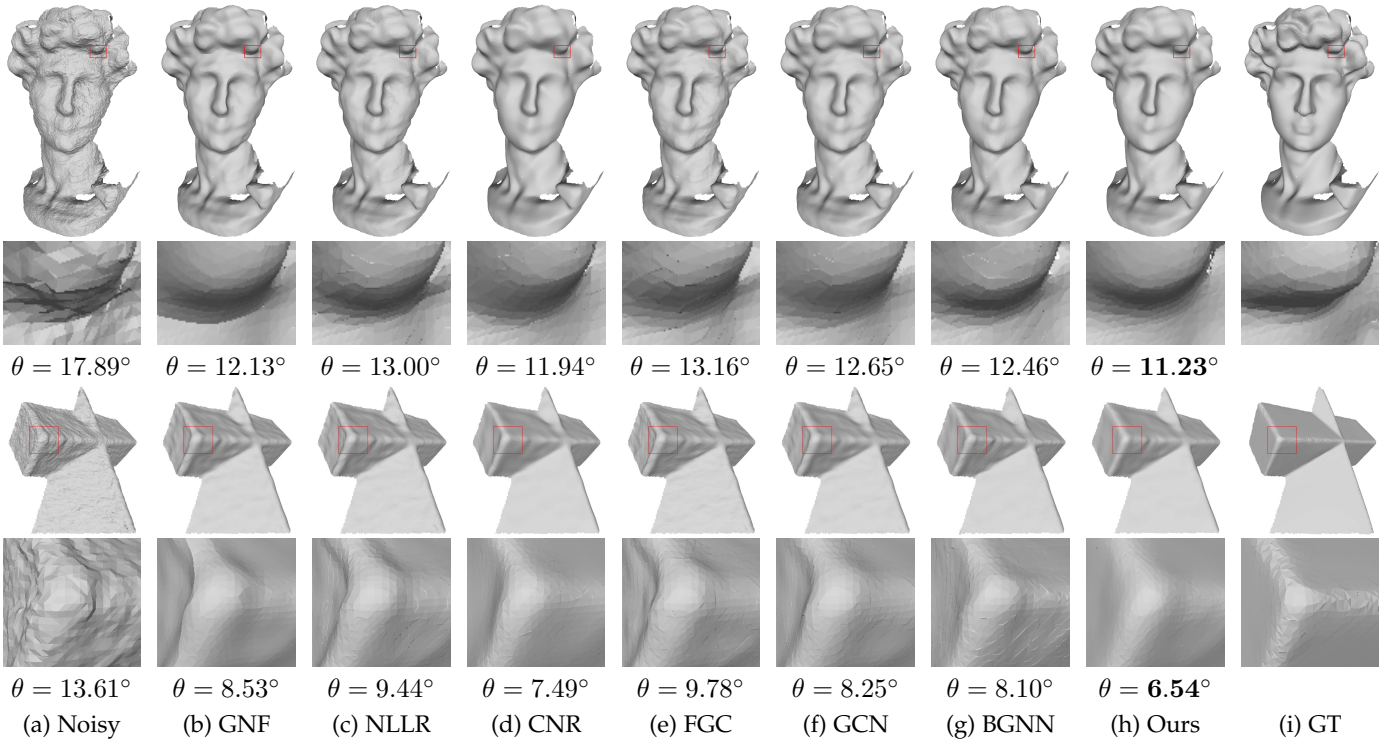


Fig. 8: Visual comparison of different methods [1], [3], [4], [10], [12] on the *David* model and the *Pyramid* model in the *Kinect Fusion dataset*, where θ represents the value of the mean angular error and the values of the mean vertex error are left out as they are close to each other.

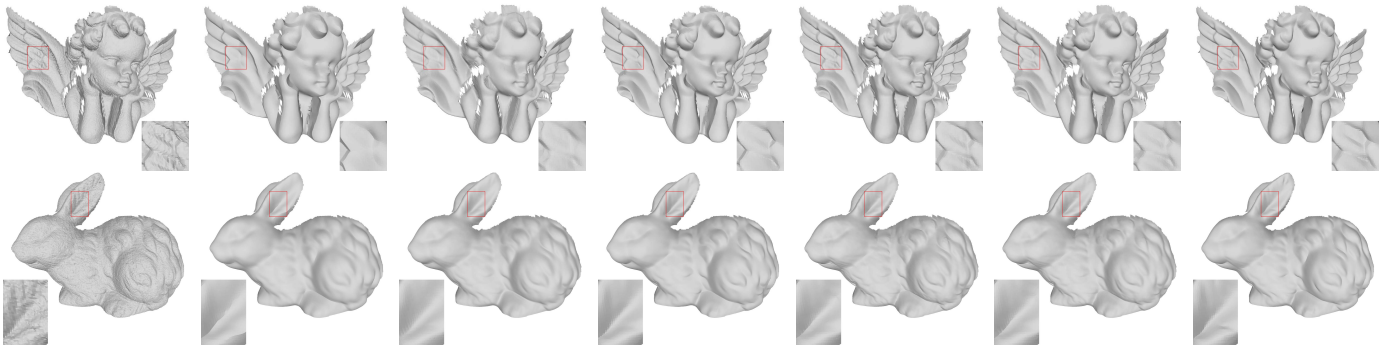


Fig. 9: Visual comparison of different methods [3], [4], [8], [9], [12] on two real-scanned models without ground truths, including the (top) *Angel* model and the (bottom) *Rabbit* model. Note that the learning based methods are all conducted with network parameters pre-trained only on the *Synthetic dataset*.

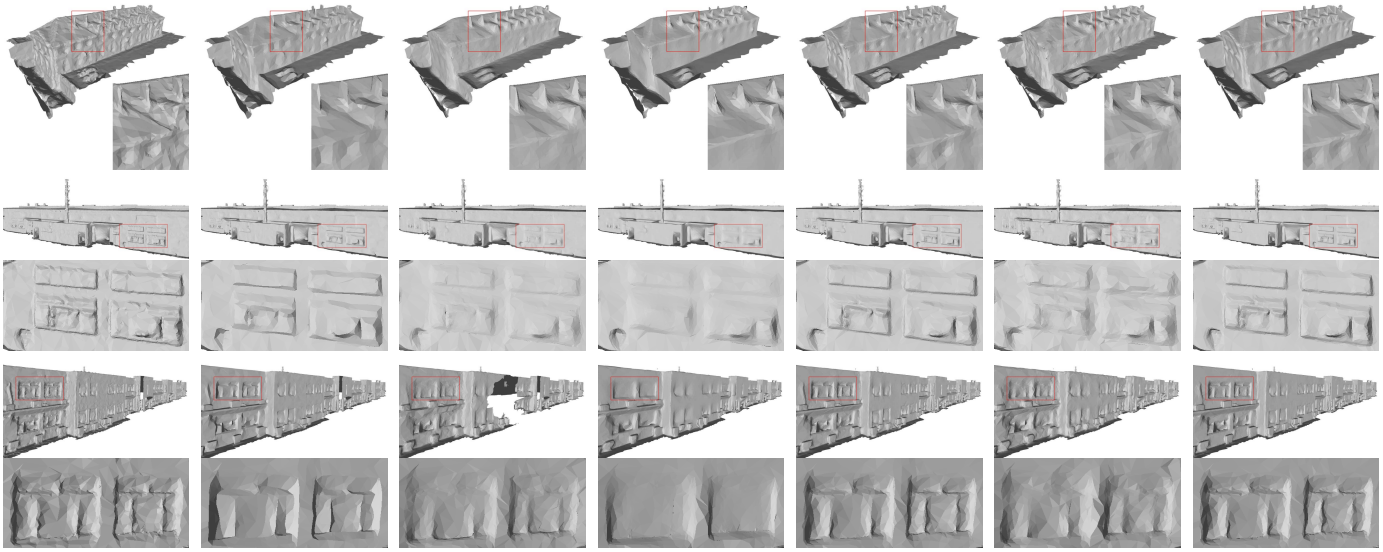
isons on several models. Among learning-based methods, our approach takes almost half the time of the *patch-to-one* based method GCN [9] and nearly a quarter of FGC [8] on some models, as it avoids the cumbersome graph coarsening and up-sampling pipeline. Although CNR [4] has the least running time with relatively simple network architecture, it leads to the least ability of feature preserving. Regarding traditional methods, GNF [3] adopts face-wise computation and takes even longer time as the face number of the input mesh increases. On the other hand, NLLR [12] transfers the denoising problem into a bunch of sparse matrix computations and saves plenty of running time. However, its unsatisfactory denoising results and tedious parameter-tuning process are not as appealing.

4.4 Ablation Study

We perform various ablation studies to explore the role of each part of the mesh denoising method we design.

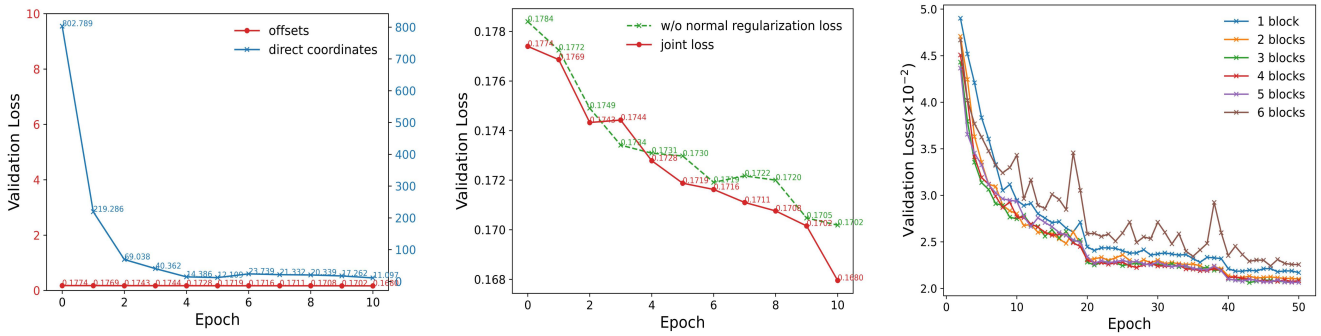
4.4.1 *Vertex Coordinate Learning*

In the vertex-aware branch of our pipeline, there are two ways for our network to learn the modified vertex coordinates: 1) regress the offsets and add them to the original coordinates; 2) regress the target coordinates directly. As modification means some adjustment of offsets, we adopt the first strategy. The validation losses of the first eleven epochs trained on the Kinect Fusion dataset with the above two strategies are demonstrated in Fig. 11 (a). We can tell



(a) Noisy (b) GNF (c) NLLR (d) CNR (e) FGC (f) GCN (g) Ours

Fig. 10: Visual comparison of different methods [3], [4], [8], [9], [12] on three real-scanned scene-level models without ground truths, where our methods stands out for its feature preserving effect. Note that the learning-based methods are all conducted with network parameters pre-trained only on the *Synthetic dataset*.



(a) (b) (c)

Fig. 11: Results of ablation studies through out epochs. (a) The validation loss with two kinds of vertex learning strategies. (b) The validation loss with or without the normal regularization loss. (c) The validation loss with different numbers of residual blocks in the decoder.

that the network converges largely faster and performs better on the former. We also conducted experiments to explore the impact of normal regularization loss on vertex learning, as shown in Fig. 11 (b). The results demonstrate that incorporating the normal regularization loss into training leads to faster convergence and an increased likelihood of finding the optimal solution. This observation is further supported by a quantitative comparison between denoised results obtained with and without (w/o) the normal regularization loss in the vertex modification branch, as illustrated in Fig. 12. After applying the regularization term to the vertex-aware branch, we observed a significant reduction in mean vertex errors for both $\mathcal{M}_1'$ and $\overline{\mathcal{M}}$. This reduction not only contributes to improved normal fidelity but also highlights the effectiveness of the regularization term.

4.4.2 Number of Residual Blocks

We also tested different numbers of residual blocks for the decoder, it turns out that four blocks are enough to achieve optimal performance. More blocks make little difference and may even impair the network's performance with additional space complexity. The validation loss for normal regression throughout the first fifty epochs on the Kinect Fusion dataset is shown in Fig. 11 (c).

4.4.3 Splitting Patch Size

To evaluate the robustness of our network with respect to input graph size, we performed a series of experiments on the splitting patch size for the inference of normal regression, with batch sizes ranging from $5k$ to $20k$. The results are presented in Fig. 13 (a) (results for vertex modification can be inferred likewise). Interestingly, we observed that as the splitting patch size increases, the performance of our network on all four benchmark datasets remains stable. This suggests that our network has learned to effectively model connections and signal transmissions between graph nodes, which can be generalized to input graph patches of any size.

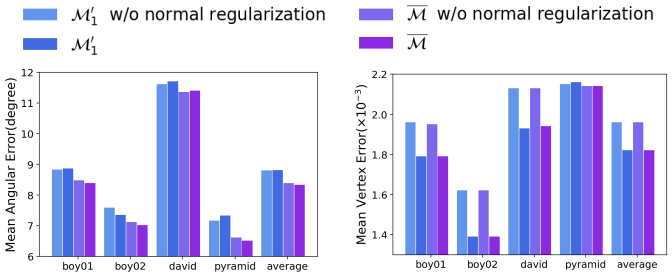
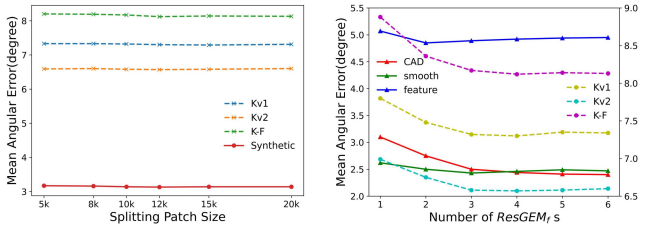


Fig. 12: Results of the ablation studies on the normal regularization, where *w/o normal regularization* denotes that the corresponding meshes in the pipeline go through the vertex-aware branch by the network trained without the normal regularization term.



(a) (b)

Fig. 13: Results of two more ablation studies. (a) Experiment results on different splitting patch sizes during the inference for normal regression. (b) Experiment results on different iteration numbers of normal regression. The synthetic data are under the scale of the left y axis and the Kinect data are under the right. In both images, the legend labels Kv1, Kv2 and K-F represent *Kinect v1 dataset*, *Kinect v2 dataset* and *Kinect Fusion dataset* respectively.

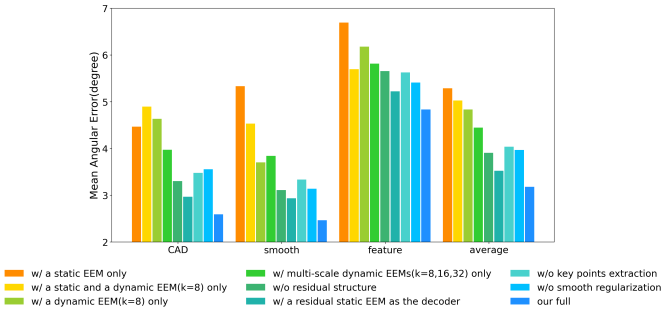


Fig. 14: Ablation studies on different network structures and training strategies.

4.4.4 Number of $ResGEM_f$ s

Moreover, different numbers of $ResGEM_f$s are experimented to prove the necessity of using multiple $ResGEM_f$s in our pipeline to regress the denoised normals in a cascaded manner. Fig. 13 (b) plots the performance on different datasets. The results on three types of models in the *Synthetic dataset* are demonstrated separately. We can tell that the performance on most datasets improves significantly during the first few iterations. However, the problem of over smoothing emerges afterwards. For models with richer features, the over-smoothed effect starts earlier. Considering both time efficiency and the denoise quality, we use three $ResGEM_f$s in our implementation.

4.4.5 Vertex Modification

To verify the necessity of the vertex-aware branch in our pipeline, vertex conditions of the *Girl03* model in the *Kinect v2 dataset* at different stages of our pipeline are compared with another two GCN-based methods in Fig. 18. The denoised result of the first iteration appears to be messy in vertex distribution because only normal information is considered, which seems almost the same as the result of GCN [9]. The vertices in the result of BGNN [10] are well-organized thanks to its simultaneous constraint on both normal and spatial domains, yet they are too well-distributed to distinguish slight features from the over-smoothed surface, such as the areas magnified at the right bottom. After performing vertex modification, the distribution of vertices becomes better arranged, and some features are recovered as well. With new rounds of normal prediction conducted, the denoised result after the final vertex updating has the surface topology the most consistent with the ground truth and maximum feature preservation. Thus the sequential training strategy for normal regression and vertex modification in our design is proven to be more effective for mesh denoising than the simultaneous scheme adopted in BGNN [10].

4.4.6 Rationality of the Topology-consistent Design

To demonstrate the superiority of our hybrid, topology-consistent approach over those incorporating graph pooling, we conduct additional experiments to assess the efficiency of our network compared to graph convolution approaches with graph pooling operations. The findings are outlined in Tab. 2. "Hybrid w/o pooling" refers to our multi-scale design utilizing two types of graph convolutions, while "*FeaSt Conv* w/ pooling" indicates a network solely built with *FeaSt Conv* layers and graph pooling operations. The interpretation of "ECC w/ pooling" follows similarly. As graph pooling operations, combined with symmetrical up-sampling processes, typically necessitate consistent graph convolution structures throughout the network to restore the original graph, hybrid structures with pooling are unnecessary to consider. As depicted, our network without pooling significantly outperforms the pooling networks in both denoising effectiveness and efficiency. This observation underscores the rationality of our hybrid and multi-scale design, which adeptly captures multi-scale geometric features and noise patterns without the need for pooling operations.

TABLE 2: Comparisons with networks that utilize pooling on the *Synthetic dataset*. We measure the mean angular errors (MAE) and the average inference times (in seconds) of the first normal regression iteration. The best results are highlighted in **bold**.

Structure	Time (s)	MAE
Hybrid w/o pooling (Ours)	**42.6**	**3.15**
FeaSt Conv w/ pooling	44.0	4.93
ECC w/ pooling	43.1	4.83

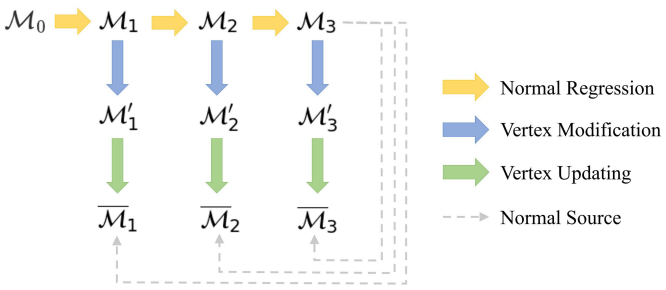


Fig. 15: An illustration of different stages of the pipeline and their corresponding notations.

4.4.7 Overall Pipeline Design

In our implementation, the vertex modification process takes place after the initial iteration of normal regression. This design is motivated by the observation that the problem of over-smoothing becomes more pronounced as subsequent normal filtering processes occur, posing a challenge for the vertex-aware branch to recover features effectively. To substantiate our hypothesis, we conducted additional tests by modifying vertices at various stages of the pipeline. The quantitative results on both the *Kinect v1 dataset* and the *Kinect v2 dataset* are presented in Fig. 17. As expected, $\mathcal{M}'_1$ exhibits superior performance compared to $\mathcal{M}'_2$ and $\mathcal{M}'_3$ in terms of mean vertex errors on both datasets, achieving the most comprehensive recovery of vertex positions. However, this stage yields relatively lower performance in terms of average normal errors, as some noise is inevitably retained while simultaneously recovering the surface topology pattern. Conversely, due to the over-smoothing problem arising from exclusive normal filtering, the vertex position errors of $\mathcal{M}_i$s remain relatively high. As a solution to achieve simultaneous denoising effects on both normals and vertices, we updated the vertex positions of $\mathcal{M}'_1$ with predicted normals from $\mathcal{M}_3$, resulting in a significant reduction in errors for both metrics. More visualized comparisons can be found in the *Supplementary Material*.

4.4.8 Network and Training Design

Subsequently, we experiment on different network architectures and training strategies to verify our network design. The following conditions are taken into account:

- *w/ a static EEM only*: The encoding part only contains a static EEM.
- *w/ a static and a dynamic EEM (k=8) only*: The encoding part combines the embedding results of a static EEM and a dynamic EEM with only a small scale of neighborhood (the number of neighbors is 8).
- *w/ a dynamic EEM (k=8) only*: The encoder only contains a dynamic EEM with a small scale of neighborhood (the number of neighbors is 8).
- *w/ multi-scale dynamic EEMs (k=8,16,32) only*: The encoder combines the embedding results of three dynamic EEMs with different scales of neighborhood (the numbers of neighbors equal to 8, 16 and 32).
- *w/o residual structures*: The basic residual blocks in the decoder are all replaced by plain graph convolution layers.
- *w/ a residual static EEM as the decoder*: The graph convolution layers in the decoder are all replaced by the same ones as the static EEM in the encoder.
- *w/o smooth regularization*: The network with the full architecture is trained under the supervision of only the normal recovery loss for normal regression.
- *w/o key points extraction*: The training set is generated with randomly sampled seed points through the farthest point sampling (FPS) algorithm instead of carefully selected key points.
- *our full*: The full architecture with a four EEMs in the encoder and residual structured decoder, which is trained on the training set generated from pre-detected key points and supervised by the normal joint loss.

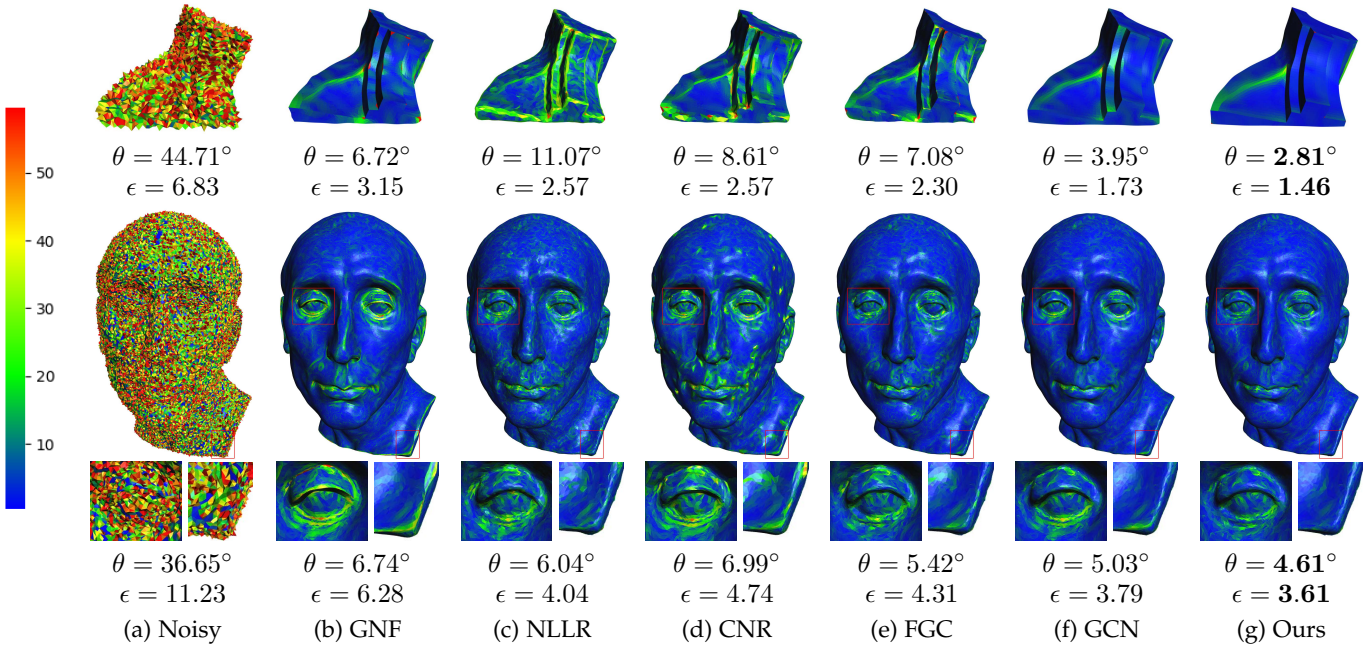
Fig. 14 plots the normal regression results of different design choices of our network training on the *Synthetic dataset*. As is demonstrated, each part of our network training design contributes to the mesh denoising performance. Particularly, the decoder utilizing the *FeaSt Conv* outperforms the decoder that employs the same convolution unit as the encoder. This observation highlights the carefully designed network structure, featuring distinct modules in both the encoder and decoder. Moreover, as the key points extraction strategy has been replaced by the simple and random FPS algorithm, the denoising ability of the network has been significantly diminished. As a result, comprehensive training using the full network architecture, along with the original training schemes involving key points extraction and smooth regularization, achieves the best performance. This underscores the superiority and rationality of our design.

5 Limitations

Although our method enables high-quality solutions for mesh denosing on various models, there still exist limitations that can be further optimized to enhance its performance. One limitation of our proposed framework is that the smooth regularization term can cause the network to prioritize smoothness over fine-scale or sharp features, resulting in a tendency to over-smooth. Additionally, our connection-based graph extraction strategy can only generate graphs from connected surfaces, and the number of nodes in each input graph must exceed the number of neighbors required for the feature embedding module at the largest scale, which is 32 in our implementation. Therefore, it is worth exploring alternative methods for constructing graphs that can capture surface information while relying less on local connectivity.

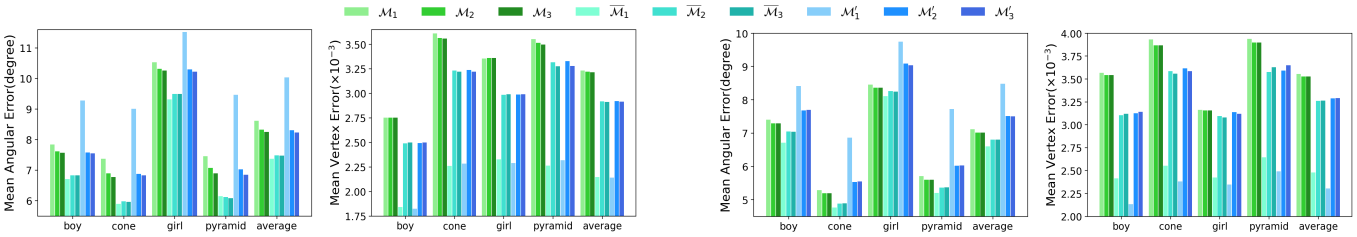
6 Conclusions and Future Directions

We presented a novel graph convolution network termed ResGEM for efficient *patch-to-patch* mesh denoising. Our network employs a structured encoder-decoder architecture, incorporating hybrid graph convolutions to balance denoising effectiveness and computational complexity. In the encoder, multi-scale edge-conditioned convolutions efficiently



(a) Noisy (b) GNF (c) NLLR (d) CNR (e) FGC (f) GCN (g) Ours

Fig. 16: Visualized normal angular error heatmaps of different methods [3], [4], [8], [9], [12] on unseen extreme noise, including model (top) *Fandisk* with the Gaussian noise of level $0.6l_e$ and (bottom) *Nicolo* with the impulsive noise of level 0.6 (both in percentage and strength). The θs represent the MAE values and the ϵs represent 10^3 times and 10^4 times of MVE values of the *Fandisk* models and the *Nicolo* models respectively.



(a) Kinect v1 (b) Kinect v2

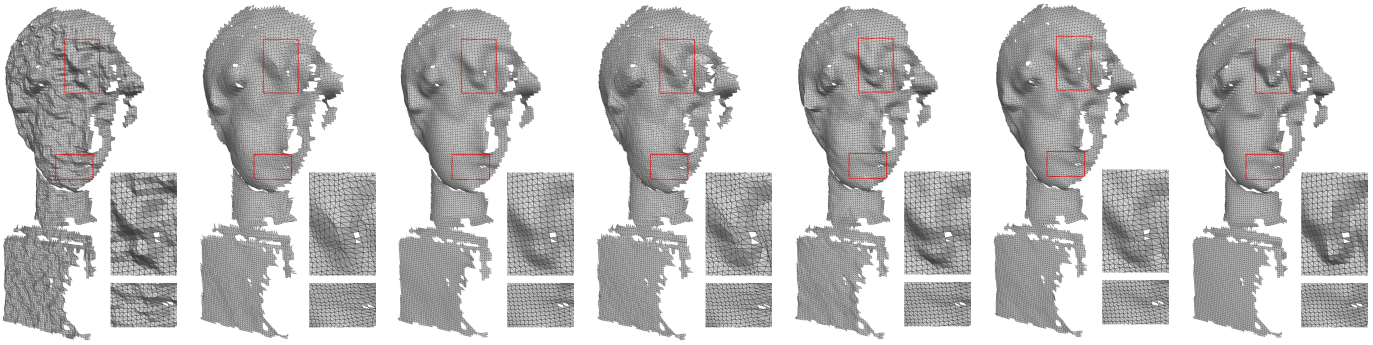
Fig. 17: The quantitative comparison between the denoised results on the *Kinect v1 dataset* and the *Kinect v2 dataset* from different pipeline stages, where $\mathcal{M}_i$ denotes the denoised mesh after the i-th iteration of normal regression, $\mathcal{M}'_i$ denotes the vertex modification result of mesh $\mathcal{M}_i$, and $\overline{\mathcal{M}}_i$ denotes the final updated mesh from $\mathcal{M}'_i$ with the predicted normals of $\mathcal{M}_3$. A more intuitive map is illustrated in Fig. 15 to exhibit the relations between the above notations. Note that the number of normal iterations is fixed to 3 with respect to the ablation study demonstrated in section 4.4.4.

extract multi-level features in a topology-consistent manner, while the decoder utilizes more intricate convolutions to reconstruct the noise-free surface at manageable computational costs. Additionally, unlike previous dual-domain methods [10], [11] that perform simultaneous training on both normal and facial graph representations, we propose a two-branch pipeline to sequentially facilitate the recovery of local geometric information and correct vertex distortion, leading to significant improvements in denoising quality. Furthermore, the integration of regularization terms for both normal regression and vertex modification enhances convergence and denoising accuracy by constraining local normal consistency and fidelity. Exhaustive qualitative and quantitative experimental results demonstrate that our method outperforms state-of-the-art approaches in denoising effects and generalization ability.

In the future, there are two main research directions that are worthy of exploration. Firstly, the development of unsupervised or semi-supervised learning methods is necessary, since noisy models usually lack corresponding ground truth, especially for real-world scanned ones. Secondly, most existing methods perform normal prediction and vertex updating sequentially and separately, which is not an end-to-end trainable pipeline. How to integrate normal prediction and vertex updating into a fully end-to-end architecture is still an open yet quite valuable issue.

ACKNOWLEDGEMENT

This work is partially funded by the Strategic Priority Research Program of the Chinese Academy of Sciences (XDB0640000), National Natural Science Foundation of China (62172415,62102414,62172416), and the Guangdong Science and Technology Program (2023B1515120026).



(a) Noisy (b) GCN (c) BGNN (d) Ours-$\mathcal{M}_1$ (e) Ours-$\mathcal{M}_1'$ (f) Ours-$\overline{\mathcal{M}}$ (g) GT

Fig. 18: Visual comparison of vertex conditions in different stages of our pipeline and two other GCN-based methods [9], [10], where *Ours*-$\mathcal{M}_1$ is the denoise result after the first iteration of normal prediction, *Ours*-$\mathcal{M}_1'$ represents the adjusted mesh after vertex modification and *Ours*-$\overline{\mathcal{M}}$ denotes the vertex updating result of *Ours*-$\mathcal{M}_1'$ by the predicted normals of the third iteration.

REFERENCES

[1] Y. Zheng, H. Fu, O. K.-C. Au, and C.-L. Tai, "Bilateral normal filtering for mesh denoising," *IEEE Trans. Vis. Comput. Graph.*, vol. 17, no. 10, pp. 1521–1530, 2010.

[2] K.-W. Lee and W.-P. Wang, "Feature-preserving mesh denoising via bilateral normal filtering," in *IEEE Int. Conf. Comput. Aided Des. Comput. Graph.*, 2005, pp. 1–6.

[3] 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.

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

[5] W. Zhao, X. Liu, Y. Zhao, X. Fan, and D. Zhao, "NormalNet: Learning-based normal filtering for mesh denoising," *IEEE Trans. Circuit Syst. Video Technol.*, vol. 31, no. 12, pp. 4697–4710, 2021.

[6] 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, p. 102861, 2020.

[7] X. Li, R. Li, L. Zhu, C.-W. Fu, and P.-A. Heng, "Dnf-net: A deep normal filtering network for mesh denoising," *IEEE Trans. Vis. Comput. Graph.*, vol. 27, no. 10, pp. 4060–4072, 2020.

[8] M. Armando, J. Franco, and E. Boyer, "Mesh denoising with facet graph convolutions," *IEEE Trans. Vis. Comput. Graph.*, 2021.

[9] Y. Shen, H. Fu, Z. Du, X. Chen, E. Burnaev, D. Zorin, K. Zhou, and Y. Zheng, "GCN-denoiser: Mesh denoising with graph convolutional networks," *ACM Trans. Graph.*, vol. 41, no. 1, pp. 1–14, 2022.

[10] Y. Zhang, G. Shen, Q. Wang, Y. Qian, M. Wei, and J. Qin, "GeoBi-GNN: Geometry-aware bi-domain mesh denoising via graph neural networks," *Comput. Aided Des.*, vol. 144, p. 103154, 2022.

[11] S. Hattori, T. Yatagawa, Y. Ohtake, and H. Suzuki, "Learning self-prior for mesh denoising using dual graph convolutional networks," in *Proc. Eur. Conf. Comput. Vis.*, 2022.

[12] 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, 2018.

[13] M. Wei, J. Huang, X. Xie, L. Liu, J. Wang, and J. Qin, "Mesh denoising guided by patch normal co-filtering via kernel low-rank recovery," *IEEE Trans. Vis. Comput. Graph.*, vol. 25, no. 10, pp. 2910–2926, 2018.

[14] X. Lu, S. Schaefer, J. Luo, L. Ma, and H. Ying, "Low rank matrix approximation for 3D geometry filtering," *Comput. Graph. Forum*, vol. 38, no. 5, pp. 75–83, 2019.

[15] L. He and S. Schaefer, "Mesh denoising via l_0 minimization," *ACM Trans. Graph.*, vol. 32, no. 4, pp. 1–8, 2013.

[16] X. Lu, W. Chen, and S. Schaefer, "Robust mesh denoising via vertex pre-filtering and l_1-median normal filtering," *Comput. Aided Geom. Des.*, vol. 54, pp. 49–60, 2017.

[17] Y. Zhao, H. Qin, X. Zeng, J. Xu, and J. Dong, "Robust and effective mesh denoising using l_0 sparse regularization," *Comput. Aided Des.*, vol. 101, pp. 82–97, 2018.

[18] L. Zhu, M. Wei, J. Yu, W. Wang, J. Qin, and P.-A. Heng, "Coarse-to-fine normal filtering for feature-preserving mesh denoising based on isotropic subneighborhoods," *Comput. Graph. Forum*, vol. 32, no. 7, pp. 371–380, 2013.

[19] M. Wei, L. Liang, W.-M. Pang, J. Wang, W. Li, and H. Wu, "Tensor voting guided mesh denoising," *IEEE Trans. Autom. Sci. Eng.*, vol. 14, no. 2, pp. 931–945, 2017.

[20] S. K. Yadav, U. Reitebuch, and K. Polthier, "Mesh denoising based on normal voting tensor and binary optimization," *IEEE Trans. Vis. Comput. Graph.*, vol. 24, no. 8, pp. 2366–2379, 2018.

[21] K. Li, M. Zhao, H. Wu, D.-M. Yan, Z. Shen, F.-Y. Wang, and G. Xiong, "Graphfit: Learning multi-scale graph-convolutional representation for point cloud normal estimation," in *Proc. Eur. Conf. Comput. Vis.* Springer, 2022, pp. 651–667.

[22] M. Zhao, L. Ma, X. Jia, D.-M. Yan, and T. Huang, "Graphreg: Dynamical point cloud registration with geometry-aware graph signal processing," *IEEE Trans. Image Process.*, vol. 31, pp. 7449–7464, 2022.

[23] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and deep locally connected networks on graphs," in *Proc. Int. Conf. Learn. Represent.*, 2014.

[24] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2016, pp. 3844–3852.

[25] D. Valsesia, G. Fracastoro, and E. Magli, "Learning localized generative models for 3D point clouds via graph convolution," in *Proc. Int. Conf. Learn. Represent.*, 2018, pp. 1–15.

[26] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon, "Dynamic graph CNN for learning on point clouds," *ACM Trans. Graph.*, vol. 38, no. 5, pp. 1–12, 2019.

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

[28] R. Hanocka, A. Hertz, N. Fish, R. Giryes, S. Fleishman, and D. Cohen-Or, "MeshCNN: a network with an edge," *ACM Trans. Graph.*, vol. 38, no. 4, pp. 1–12, 2019.

[29] J. Schult, F. Engelmann, T. Kontogianni, and B. Leibe, "DualConvMesh-Net: Joint geodesic and euclidean convolutions on 3D meshes," in *Proc. IEEE Conf. Comput. Vis. Pattern Recog.*, 2020, pp. 8612–8622.

[30] N. Verma, E. Boyer, and J. Verbeek, "FeastNet: Feature-steered graph convolutions for 3D shape analysis," in *Proc. IEEE Conf. Comput. Vis. Pattern Recog.*, 2018, pp. 2598–2606.

[31] I. Sipiran and B. Bustos, "Harris 3D: a robust extension of the harris operator for interest point detection on 3D meshes," *Visual Comput.*, vol. 27, no. 11, pp. 963–976, 2011.

[32] C. Harris, M. Stephens *et al.*, "A combined corner and edge detector," in *Alvey Vis. Conf.*, 1988, pp. 1–6.

[33] Y. Eldar, M. Lindenbaum, M. Porat, and Y. Zeevi, "The farthest

point strategy for progressive image sampling," *IEEE Trans. Image Process.*, vol. 6, no. 9, pp. 1305–1315, 1997.
[34] M. Simonovsky and N. Komodakis, "Dynamic edge-conditioned filters in convolutional neural networks on graphs," in *Proc. IEEE Conf. Comput. Vis. Pattern Recog.*, 2017, pp. 3693–3702.
[35] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recog.*, June 2016, pp. 770–778.
[36] L. I. Rudin, S. Osher, and E. Fatemi, "Nonlinear total variation based noise removal algorithms," *Phys. D: Nonlinear Phenom.*, vol. 60, no. 1-4, pp. 259–268, 1992.
[37] X. Sun, P. L. Rosin, R. Martin, and F. Langbein, "Fast and effective feature-preserving mesh denoising," *IEEE Trans. Vis. Comput. Graph.*, vol. 13, no. 5, pp. 925–938, 2007.
[38] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohi, J. Shotton, S. Hodges, and A. Fitzgibbon, "Kinectfusion: Real-time dense surface mapping and tracking," in *IEEE Int. Symp. Mix. Augment. Real.*, 2011, pp. 127–136.
[39] M. Fey and J. E. Lenssen, "Fast graph representation learning with PyTorch Geometric," in *Proc. Int. Conf. Learn. Represent. Worksh.*, 2019.
[40] L. Nan, "Easy3D: a lightweight, easy-to-use, and efficient C++ library for processing and rendering 3D data," *J. Open Source Softw.*, vol. 6, no. 64, p. 3255, 2021.

Ziqi Zhou is currently a graduate student at the School of Artificial Intelligence, University of Chinese Academy of Sciences (UCAS). She is also a research student at the State Key Laboratory of Multimodal Artificial Intelligence Systems (MAIS) & National Laboratory of Pattern Recognition (NLPR), Institute of Automation, Chinese Academy of Sciences (CAS). Her research interests include computer graphics, computer vision, and multimodal information processing.

Mengke Yuan received the bachelor's degree in applied mathematics and the master's degree in computational mathematics from Zhengzhou University, Zhengzhou, China, in 2012 and 2015, respectively, and the Ph.D. degree in computer sciences from the State Key Laboratory of Multimodal Artificial Intelligence Systems (MAIS) & National Laboratory of Pattern Recognition (NLPR), Institute of Automation, Chinese Academy of Sciences (CASIA), Beijing, China, in 2019. He is currently a researcher at PIESAT Information Technology Co Ltd. His research interests include image processing, computer vision, and computer graphics.

Mingyang Zhao is currently a research assistant professor at CAIR, Hong Kong Institute of Science & Innovation, Chinese Academy of Science (CAS). He received the PhD degree from Academy of Mathematics and Systems Science, CAS, in 2021. He now focuses his research and development on robust estimation, geometric problems, and Gaussian processes. He is awarded the grand Phd scholarship of Saudi Arabian, and the excellent student prize of president fellowships, Chinese Academy of Sciences in 2020 and 2021, respectively.

Jianwei Guo is an Associate Professor at the State Key Laboratory of Multimodal Artificial Intelligence Systems (MAIS) & National Laboratory of Pattern Recognition (NLPR), Institute of Automation, Chinese Academy of Sciences (CASIA). He received his Ph.D. degree in computer science from CASIA in 2016, and bachelor's degree from Shandong University in 2011. His research interests include computer graphics, geometric processing and 3D vision.

Dong-Ming Yan (Member, IEEE) is a professor at the State Key Laboratory of Multimodal Artificial Intelligence Systems (MAIS) & National Laboratory of Pattern Recognition (NLPR), Institute of Automation, Chinese Academy of Sciences (CAS). He received his Ph.D. from Hong Kong University in 2010 and his Master's and Bachelor's degrees from Tsinghua University in 2005 and 2002, respectively. His research interests include computer graphics, computer vision, geometric processing and pattern recognition.