# GCN-Denoiser: Mesh Denoising with Graph Convolutional Networks

YUEFAN SHEN, The State Key Lab of CAD&CG, Zhejiang University
HONGBO FU, The School of Creative Media, City University of Hong Kong
ZHONGSHUO DU and XIANG CHEN, The State Key Lab of CAD&CG, Zhejiang University
EVGENY BURNAEV, Skolkovo Institute of Science and Technology
DENIS ZORIN, New York University
KUN ZHOU, The State Key Lab of CAD&CG, Zhejiang University
YOUYI ZHENG*, The State Key Lab of CAD&CG, Zhejiang University

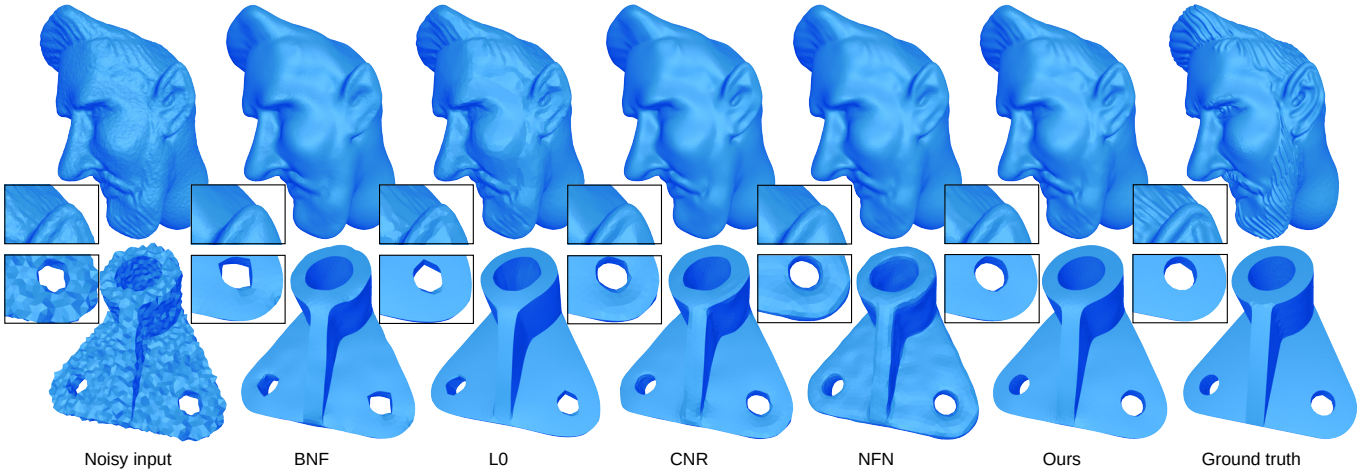


Fig. 1. Comparisons of our method to the state-of-the-art mesh denoising methods: Bilateral normal filtering (BNF) [Zheng et al. 2011], $L0$ Smoothing (L0) [He and Schaefer 2013], Cascaded normal regression (CNR) [Wang et al. 2016], and NormalF-Net (NFN) [Li et al. 2020c]. Our method consistently achieves the best results for both smooth features and sharp features. The first row shows the denoising results on a real scan model and the second row shows the results on a model with synthetic noise (i.e., Gaussian noise with the level of 0.3 mean edge length). The average normal angular errors (from left to right) are: ($1^{st}$ row) 9.68°, 9.03°, 8.17°, 8.29°, 8.35°, and **7.62**°; ($2^{nd}$ row) 25.85°, 3.53°, 5.69°, 3.36°, 7.79°, and **1.84**°.

In this paper, we present GCN-Denoiser, a novel feature-preserving mesh denoising method based on graph convolutional networks (GCNs). Unlike previous learning-based mesh denoising methods that exploit hand-crafted or voxel-based representations for feature learning, our method explores the structure of a triangular mesh itself and introduces a graph representation followed by graph convolution operations in the dual space of triangles. We show such a graph representation naturally captures the geometry features while being lightweight for both training and inference. To facilitate effective feature learning, our network exploits both static and dynamic edge convolutions, which allow us to learn information from both the explicit mesh structure and potential implicit relations among unconnected neighbors. To better approximate an unknown noise function, we introduce a cascaded optimization paradigm to progressively regress the noise-free facet normals with multiple GCNs. GCN-Denoiser achieves the new state-of-the-art results in multiple noise datasets, including CAD models often containing sharp features and raw scan models with real noise captured from different devices. We also create a new dataset called PrintData containing 20 real scans with their corresponding ground-truth meshes for the research community. Our code and data are available in https://github.com/Jhonve/GCN-Denoiser.

CCS Concepts: • **Computing methodologies** → **Shape analysis**; **Mesh models**.

Additional Key Words and Phrases: Mesh denoising, graph convolutional networks, cascaded optimization.

**ACM Reference Format:**
Yuefan Shen, Hongbo Fu, Zhongshuo Du, Xiang Chen, Evgeny Burnaev, Denis Zorin, Kun Zhou, and Youyi Zheng. 2021. GCN-Denoiser: Mesh Denoising with Graph Convolutional Networks. *ACM Trans. Graph.* 40, 4, Article 111 (August 2021), 14 pages. https://doi.org/10.1145/3480168

*corresponding author

Authors' addresses: Yuefan Shen, jhonve@zju.edu.cn, The State Key Lab of CAD&CG, Zhejiang University; Hongbo Fu, The School of Creative Media, City University of Hong Kong; Zhongshuo Du; Xiang Chen, The State Key Lab of CAD&CG, Zhejiang University; Evgeny Burnaev, Skolkovo Institute of Science and Technology; Denis Zorin, New York University; Kun Zhou, The State Key Lab of CAD&CG, Zhejiang University; Youyi Zheng, The State Key Lab of CAD&CG, Zhejiang University.


0730-0301/2021/8-ART111 $15.00
https://doi.org/10.1145/3480168

## 1 INTRODUCTION

With the increasing popularity of consumer-level depth cameras and 3D scanners, it has become easier to acquire 3D models by 3D-scanning real-world objects. However, even with advanced camera technologies, scanned models are inevitably corrupted by noise mainly due to imperfect measurements, making them not immediately usable in the subsequent graphics pipelines.

Feature-preserving mesh denoising aims to recover the underlying surface signal (e.g., position, normal) from a measurement with noise $\Delta^*$, essentially, to remove the noise $\epsilon = \Delta^* - \Delta$ while keeping the underlying features preserved. This problem is inherently ill-posed since both $\Delta$ and $\epsilon$ are unknown. To make it tractable, priors and assumptions on $\Delta$ or $\epsilon$ are often made. For example, the pattern of $\epsilon$ is assumed to follow a Gaussian distribution or to be independent and identically distributed. Unfortunately, these assumptions are often invalid for real scanners [Wang et al. 2012], thus challenging a variety of traditional filter-based and optimization-based denoising approaches [Fleishman et al. 2003; He and Schaefer 2013; Li et al. 2018; Wei et al. 2019b; Zhang et al. 2015; Zheng et al. 2011].

Data-driven methods have attracted a lot of attention lately and several approaches have been introduced for mesh denoising [Li et al. 2020c; Wang et al. 2016; Zhao et al. 2019b]. Without making any specific assumption on the data, these methods estimate $\epsilon$ from massive noisy meshes and their ground-truth counterparts, and achieve impressive results. However, their learning paradigms still suffer from several aspects. First, since $\epsilon$ can be a highly complex function eroded over various geometric features, globally estimating $\epsilon$ is often intractable and thus surface patches are usually exploited to regress $\Delta$ locally. This makes the design of a representation of local patches an essential issue for learning-based algorithms. Existing data-driven methods either use hand-crafted features [Li et al. 2020c; Wang et al. 2016] or exploit a voxel representation [Zhao et al. 2019b], leading to either insufficient or redundant information flow into the subsequent learning module. In addition, methods that rely on convolution operations on voxels are known to be slow [Wang et al. 2017], causing an additional efficiency issue.

To seek the balance between efficacy (geometry representation) and efficiency, we present *GCN-Denoiser*, a novel approach for feature-preserving denoising of mesh surfaces (triangular meshes in our implementation) based on the extension of graph convolutional networks (GCNs), which have been proved effective for various applications [Schult et al. 2020] but not for feature-preserving mesh denoising. GCN-Denoiser exploits a novel graph-based representation for local surface patches and employs graph convolution operations to effectively learn the relations between $\epsilon$ and $\Delta$. The graph representation naturally fits into the local structure surrounding a surface facet and thus has better capability in capturing the local geometric information than hand-crafted features and voxel-based features, enabling a more accurate estimation of $\epsilon$ while being also efficient at runtime.

Like previous works [He and Schaefer 2013; Wang et al. 2016], we model $\Delta$ as surface normal signals. Our goal is to regress for each facet a normal from a local patch surrounding the facet. To this end, we build our graph structure in the dual space of mesh facets, where each facet of the mesh forms a graph node and adjacent facets are connected with graph edges. We then define graph convolution operations that directly operate on mesh facets to deal with varying local graph structures. To effectively learn features, our network architecture employs both static and dynamic edge convolution operations to exploit both the explicit and implicit graph structures, enabling information flow from both the neighboring facets and the unconnected ones in a patch. We further introduce a normal tensor voting strategy to make our patch representation rotation-invariant. Finally, cascaded optimization is employed to progressively regress $\Delta$ using multiple GCNs.

We test our method extensively on both synthetic and real-scan models (including a newly created dataset: PrintData) and make comparisons with the state-of-the-art algorithms. Various experiments demonstrate that our method produces superior results to the compared approaches. In summary, the main contributions of this paper are:

- We introduce the first GCN-based method for feature-preserving mesh denoising, which achieves the new state-of-the-art results while being well-balanced between efficacy and efficiency;
- We present a rotation-invariant graph representation on the dual space of mesh faces, enabling effective feature learning with graph convolutions;
- We create a new feature-preserving mesh denoising dataset consisting of 20 real scans with corresponding ground-truth meshes by using a high-end 3D printer and a high-resolution scanner.

## 2 RELATED WORK

We review the literature of feature-preserving mesh denoising and some recent advances in graph convolutional networks.

### 2.1 Mesh Denoising

The task of denoising on mesh surfaces is akin to that on 2D images, since vertex positions or face normals are essentially signals in 3D. Hence, mesh denoising techniques have been heavily inspired by denoising techniques in images. Various low-pass and feature-preserving filters have been introduced for mesh denoising [Bajaj and Xu 2003; Clarenz et al. 2000; Shen and Barner 2004; Tasdizen et al. 2002; Yagou et al. 2002, 2003], based on the assumption that the noise over a surface is often of high frequency. Among them, a bilateral filter is one of the most widely applied filters [Adams et al. 2009; Fan et al. 2010; Fleishman et al. 2003; Jones et al. 2003; Shimizu et al. 2005; Wang et al. 2012; Wei et al. 2015; Yadav et al. 2019; Yoshizawa et al. 2006; Zheng et al. 2011] due to its simplicity and feature-preserving property. It is reported in [Lee and Wang 2005; Zheng et al. 2011] that normals are often more discriminative in describing local geometry than vertex positions. In light of this, a series of research works improve the denoising performance by first updating facet normals and then vertex positions [Sun et al. 2007]. Extensions to these works have been made later on to find more robust filtering schemes on facet normals [Zhang et al. 2015; Zhao et al. 2019a]. A common drawback in filter-based methods is that once the features are highly corrupted by noise, they, especially weak ones, are difficult to be recovered by these methods.

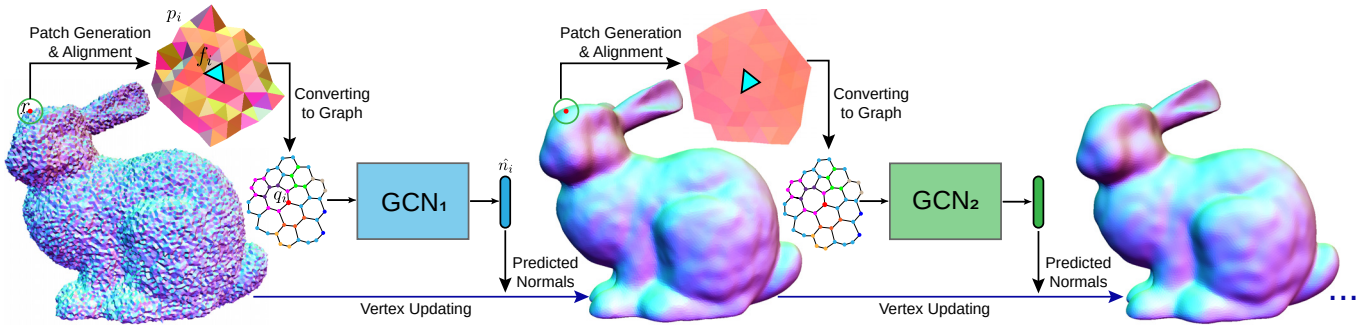


Fig. 2. The pipeline of GCN-Denoiser. For each mesh facet, a graph patch is generated in the dual space of triangles and subsequently fed into multiple GCNs to regress the final noise-free normal for the facet. The output normals of each GCN are used to update the mesh geometry via vertex updating. Then each updated graph patch is fed into the next GCN. In practice, 2 GCNs are sufficient to achieve sufficiently denoised results.

Optimization-based mesh denoising methods take from another perspective by seeking a denoised mesh to approximate an input mesh conditional on a set of priors imposed on the ground-truth geometry or noise patterns. They formulate the denoising procedure as optimization problems and solve them via techniques such as Bayesian [Diebel et al. 2006], $L_0$ minimization [He and Schaefer 2013], compressed sensing [Wang et al. 2014], or low-rank recovery [Li et al. 2018; Wei et al. 2019b]. These methods, however, only work well for meshes where their assumptions are satisfied, and often do not generalize well for meshes with different geometry features and noise patterns.

In contrast, learning-based methods do not make specific assumptions about the underlying features or noise patterns, and have been successfully applied to image denoising [Agostinelli et al. 2013; Burger et al. 2012; Jin et al. 2017; Sterzentsenko et al. 2019; Xie et al. 2012; Yan et al. 2018; Zhang et al. 2017]. However, unlike images, 3D meshes are usually irregular, thus making image-based convolutional operations not directly applicable. To address this issue, Wang et al. [2016] introduce a filtered facet normal descriptor (FND), which is extracted from a local region by bilateral filtering with multiple kernels, and utilize simple multilayer perceptrons (MLPs) to regress noise-free normals from the descriptor. Later on, Zhao et al. [2019b] employ a voxel-based representation and apply 3D convolution to regress noise-free normals. In [2020c], Li et al. adopt a non-local patch matrix proposed by [Li et al. 2018] as a regular representation of mesh patches and use 2D convolutional networks to learn noise-free normals. Wei et al. [2019a] introduce a learning-based de-filtering strategy to recovery over-smoothed weak features, but their results depend on the quality of pre-denoised meshes. In a concurrent work, Li et al. [2020b] omit the adjacent relationship between mesh faces and use a similar architecture of PointNet++ [Qi et al. 2017b] to regress a new normal vector from a patch of facet normals. Different from these works, our method directly feeds an irregular mesh patch data into a graph convolutional network rather than seeking an intermediate hand-crafted representation. We show that our graph-based representation of local surface geometry with graph convolution operations enables our network to have better capability in capturing the inherent geometry features of a noisy patch than the above-discussed methods.

## 2.2 Graph Convolutional Networks

Graph convolutional networks (GCNs) have been introduced for handling non-Euclidean structures. Early works of GCNs require a static graph structure [Bruna et al. 2014; Defferrard et al. 2016] and thus cannot be extended for meshes with varying topology. Recent studies on dynamic graph convolution show that changeable edges can perform better. For example, Simonovsky et al. [2017] generalize the convolution operator to irregular graphs with filter weights. Wang et al. [2019] dynamically construct graph structures by connecting neighboring nodes in each layer. Valsesia et al. [2019] construct node neighbors using K-nearest-neighbors (KNN). Furthermore, Li et al. [2019] show that very deep GCNs with dynamic graph convolutions can further boost the performance in applications such as point cloud recognition and segmentation. Our method exploits both the static graph structure in patches and dynamic graph structures that are constructed during convolution to effectively learn the geometry features in patches.

There are some other convolution operators developed for meshes lately. In [Masci et al. 2015; Monti et al. 2017], local coordinate systems are defined to confine the convolution operations on regular grids. MeshCNN [Hanocka et al. 2019] executes convolution or pooling on mesh edges while [Schult et al. 2020] performs graph convolution and vertex pooling on mesh vertices. In [2019], Feng et al. introduce a convolution on mesh facets and separate mesh features into spatial and structure levels manually. These methods are mainly designed for understanding whole objects or large scenes and require very deep architectures. Instead, we pay more attention to local patches and employ the convolution in the dual space of mesh facets.

# 3 ALGORITHM OVERVIEW

Fig. 2 illustrates the pipeline of GCN-Denoiser. Since $\epsilon$ is usually a highly complex function over the underlying surface, we take a local approach to approximate it. Our aim is to predict a noise-free normal $n_f$ for each individual facet $f$ surrounded by a noisy local surface patch $p$ in the dual domain of mesh triangles. Given the denoised facet normals, we update the vertex positions according to the method of [Zheng et al. 2011] to get a denoised surface.

Given a noisy triangular mesh, we first generate a local patch for each facet. To eliminate the global spatial transformations among the local patches, we apply normal tensor voting to the collected patches to align them into a common embedding (Section 4.2). Next, we convert these aligned patches into graph representations (Section 4.3), then feed these patch graphs into our GCN network to progressively compute the noise-reduced normals (Section 5.1), and then obtain a noise-reduced mesh via vertex updating (Section 6). Even when we approximate the noise function $\epsilon$ in local patches, the underlying noise patterns there could still be complex as they are eroded by different geometric features. Hence, as a common practice for approximating a highly nonlinear function [Li et al. 2020c; Wang et al. 2016], we employ a cascaded optimization to train multiple GCNs to progressively regress the final noise-free normals. In each GCN, pairs of $(p, n_f)$ are collected for training (Section 5.2). The overall workflow is similar among all the cascaded stages, and the only difference is that the noise levels of $\{p\}$-s are different.

## 4 PATCH GENERATION AND ALIGNMENT

We define an input triangular mesh as $M = \{V, F\}$, with $V = \{v_i\}_1^{N_v}$ the set all vertices and $F = \{f_i\}_1^{N_f}$ the set all facets. $N_v$ and $N_f$ are respectively the number of vertices and the number of faces. For each facet $f_i$ in $F$, we will generate its local patch data $p_i$. The set of all patches in $M$ is defined as $P = \{p_i\}_1^{N_f}$. Also, we denote the normal of a facet $f_i$ as $n_i$, its centroid as $c_i$, and its area as $a_i$.

### 4.1 Patch Selection

A patch $p_i$ in our context refers to all facets (including $f_i$) within a sphere of radius $r$ located at the centroid $c_i$ of facet $f_i$, i.e., $p_i$ is supposed to satisfy:

$$\underset{f_j \in p_i}{\forall} \underset{v_k \in f_j}{\exists} ||v_k - c_i|| < r. \tag{1}$$

We use $r = k\bar{a}_i^{\frac{1}{2}}$, where $\bar{a}_i$ is the average area of 2-ring neighboring facets of $f_i$ and $k$ is a parameter relevant to the resolution of an input mesh (Section 7.4). We incorporate the facet area in defining our patch to accommodate irregular sampling (see an example in Fig. 8).

### 4.2 Patch Alignment via Tensor Voting

Patches at different spatial locations would cause troubles for neural networks, since learning spatial transformations is known to be difficult for deep methods [Li et al. 2020a]. To address this issue, we resort to a normal tensor voting strategy to explicitly align the patches into a common coordinate system.

As in [Zhao et al. 2019b], we first translate $p_i$ to the origin $[0, 0, 0]$ and also scale it into a unit bounding box. The normal voting tensor $T_i$ [Shimizu et al. 2005] for facet $f_i$ is then defined by:

$$T_i = \sum_{f_j \in p_i} \mu_j n_j{}' n_j{}'^T, \tag{2}$$

where $\mu_j = (a_j/a_m) \exp(-||c_j - c_i||/\sigma)$, with $a_m$ being the maximum triangle area in $p_i$, and $n_j{}'$ is the voted normal of $f_j$: $n_j{}' = 2(n_j \cdot w_j)w_j - n_j$, with $w_j = normalize\{[(c_j - c_i) \times n_j] \times (c_j - c_i)\}$.

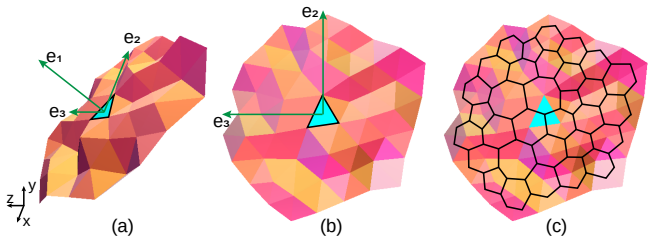


Fig. 3. (a) original patch, (b) patch after applying our normal tensor voting to (a), and (c) the graph structure of the patch.

Since $T_i$ is a positive semi-definite matrix and can be represented by its spectral decomposition as:

$$T_i = \lambda_1 e_1 e_1{}^T + \lambda_2 e_2 e_2{}^T + \lambda_3 e_3 e_3{}^T, \tag{3}$$

where $\lambda_1 \geq \lambda_2 \geq \lambda_3$ are its eigen values, and $e_1, e_2$ and $e_3$ are the corresponding unit eigen vectors, which form a group of orthonormal basis. Then we construct a rotation matrix $R_i = [e_1, e_2, e_3]$ and multiply each facet's centroid and normal in $p_i$ with $R_i^{-1}$ to generate new patch data $\bar{p}_i$. We show the comparison between the rendered patch data before and after alignment in Fig. 3.

### 4.3 Graph Representation

We build a graph structure for each of our generated patches after alignment in order to shape it to fit our subsequent graph convolutional networks. An undirected graph $\mathcal{G} = (Q, E, \Phi)$ is built, where a graph node $q_i \in Q$ is created for each facet $f_i$ in patch $\bar{p}$ and an edge $e = (q_i, q_j) \in E$ is created if the corresponding faces $f_i$ and $f_j$ are adjacent. Fig. 3 (c) shows an example. $\Phi$ contains a set of node attributes, i.e., feature tuples. For each $\phi_i \in \Phi$, which corresponds to facet $f_i$, we set $\phi_i = (\bar{c}_i, \bar{n}_i, a_i, d_i)$. Here $\bar{c}_i$ and $\bar{n}_i$ respectively indicate the centroid and normal of facet $f_i$ after alignment. $d_i$ is the number of neighboring facets in the 1-ring neighborhood of $f_i$, which helps distinguish boundary faces.

## 5 NORMAL REGRESSION

With the graph representation $\mathcal{G}_i$ extracted for each patch $p_i$, we now detail our multiple graph convolutional networks that take $\mathcal{G}_i$ as input and output a denoised normal vector $\hat{n}_i$ for facet $f_i$.

### 5.1 Graph Convolutional Network

Our GCN network consists of multiple convolutional layers. In each layer, similar to traditional convolutional networks, our GCN aggregates features from neighboring nodes of each node and updates its features. The aggregating and updating are also named as a convolutional operation.

*Graph Convolution.* Although each graph contains necessary connectivity among facets, their structures vary across patches. Thus, we adopt an Edge-Conditioned Convolution (ECC) strategy proposed in [Simonovsky and Komodakis 2017] to deal with convolutions among varying structures. Specifically, we use *EdgeConv* [Wang et al. 2019] as described below.

Let $\mathcal{G}_l = (Q_l, E_l, \Phi_l)$ be the $l$-th layer in our graph convolutional network and $\mathrm{F}_l^i$ is the feature vector of the $i$-th node in $\mathcal{G}_l$. EdgeConv

updates nodes features by:

$$\mathrm{F}_{l+1}^{i} = \underset{j:(i,j)\in E_l}{\Psi} h_{\Theta}^{l}(\mathrm{F}_{l}^{i}, \mathrm{F}_{l}^{j}), \tag{4}$$

where $\Psi$ is a max aggregation operation and $h_{\Theta}^{l} = \texttt{Linear}_{\Theta}^{l}(\mathrm{F}_{l}^{i}, \mathrm{F}_{l}^{j} - \mathrm{F}_{l}^{i})$. Each graph convolutional layer of our network shares the same $\texttt{Linear}_{\Theta}$, which is a multi-layer perceptron (MLP) including batch normalization and activation function `LeakyReLU`.

By only utilizing the original graph structure might lead to some missing information during convolution since the mapping from geometry to connectivity is not a one-to-one function. To enrich the receptive field of a graph node, we further allow non-adjacent graph nodes to be connected during convolution. This corresponds to a dynamic graph construction procedure [Simonovsky and Komodakis 2017; Wang et al. 2019]. We call graph convolution with this scheme *dynamic EdgeConv*. For this scheme, the neighbors of each node are dynamically calculated by KNN (K = 8 in our implementation) based on the Euclidean distance of node features.

*Network Architecture and Training.* As shown in Fig. 4, our network architecture consists of $L_e$ layers of EdgeConv, $L_d$ layers of dynamic EdgeConv, and $L_l$ layers of fully connected (FC) layers. After the layers of graph convolution, the learned features are concatenated together for pooling. We use both average pooling and max pooling as symmetric functions, which are able to select the most important features. Finally, the FC layers follow to regress a 3D vector, which is our predicted normal. Every layer in our architecture except the last FC layer is followed by batch normalization and activation function `LeakyReLU`.

As discussed in Section 3, we use cascaded GCNs ($\mathrm{GCN}_1$, ..., $\mathrm{GCN}_l$) to progressively regress the noise-free normals. All GCNs used in the cascaded optimization share the same architecture but with different numbers of EdgeConv, dynamic EdgeConv, and FC layers and are trained iteratively as in [Wang et al. 2016]. In our experiments, we use $L_e = 3, L_d = 3$, and $L_l = 4$ for the first GCN, and $L_e = 2, L_d = 2$, and $L_l = 3$ for the rest, since we wish the first one to recover the normals coarsely while the rest ones to refine the details. The loss function is MSE between the network output and the ground truth normal, which is $R^{-1}\tilde{n}_f$. Here, $\tilde{n}_f$ is the normal of face $f_i$ in the ground-truth noise-free mesh and $R$ is the corresponding rotation matrix in Section 4.2. We use the MSE loss instead of cosine similarity since it leads to more stable training as MSE imposes hard constraints on the values to be within the range of (0,1). Note that we map the normalized normals from (-1, 1) into (0, 1).

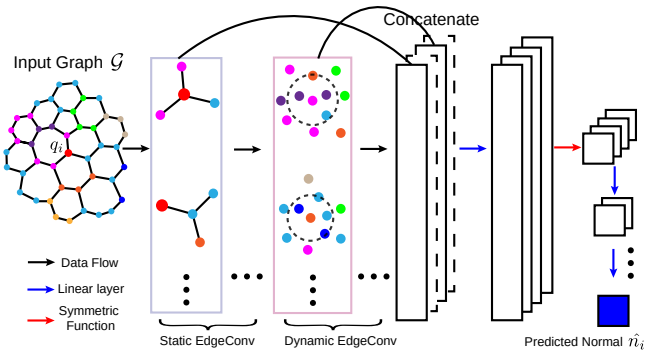


Fig. 4. Our GCN architecture. Both static EgdeConv and dynamic EdgeConv are applied and their aggregated information is combined to flow into subsequent MLPs to regress a noise-reduced normal.

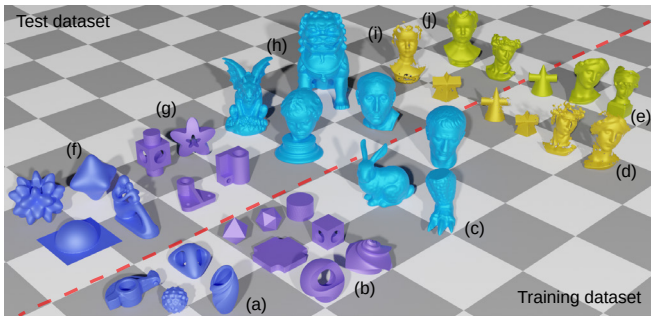


Fig. 5. Representative mesh models including (a, b, c, f, g, h) those with synthetic noise (SysData), (d, i) Kinect v1/v2 real scan models (Kv1Data, Kv2Data), and (e, j) Kinect v1 fused models (K-FData). Following [Wang et al. 2012], we consider (a, f) as smooth models, (b, g) CAD models, and (c, h) feature models. The bottom-right set of models consists of all synthetic models and sampled real scan models for training, and the top-left set of models consists of sampled test models.

In an offline training step, we use the output of $\mathrm{GCN}_i$ to denoise (with both normal updating and vertex updating) a noisy mesh in our training set, and new graphs are generated from these updated meshes to train the next $\mathrm{GCN}_{i+1}$. We stop cascading our GCNs when the validation error is not decreasing. In our experiments, we find that from $\mathrm{GCN}_2$, the accuracy of $\mathrm{GCN}_i$ improves very slightly (see in Fig. 18), thus, unless explicitly indicated, we use two cascaded GCNs in our pipeline.

## 5.2 Data Generation

Our training set contains mesh models with both synthetic or real noise (Fig. 5). For each 3D model, we generate different levels and types (Gaussian and Impulsive) of noise for training.

We observe that although mesh surfaces appear rather different from each other, their local patches are less diverse. For example, in most of the CAD models, there are a redundant number of flat patches, and edge and corner features, which would cause an imbalanced data distribution during training. In light of this, we apply the tensor voting strategy described in Section 4.2 on each facet of noise-free models in our dataset and get three eigenvalues $\lambda_1, \lambda_2$, and $\lambda_3$. For each model we first classify the facets into four groups: those with $\{f_i | \lambda_2^i < 0.01 \wedge \lambda_3^i < 0.001\}$ as flat facets, those with $\{f_i | \lambda_2^i > 0.01 \wedge \lambda_3^i < 0.1\}$ as edge facets, those with $\{f_i | \lambda_3^i > 0.1\}$ as corner facets, and the rest as transitional facets. Several examples of classified faces are shown in Fig. 7 with different colors. Due to the numbers of edge and corner facets are smaller compared with those of the other two types of features, we further arrange them into two groups: a group of non-feature facets composed of flat and transitional facets, and a group of feature facets composed of

corner and edge facets. We then sample among the two groups to collect our training patches. This data-balancing strategy is used for training all our GCNs.

## 6 SURFACE DENOISING WITH PREDICTED NORMALS

Given the predicted normals of a noisy input mesh, we first refine them to handle possible discontinuities between adjacent faces and then reconstruct its denoised version under their guidance.

### 6.1 Normal Refinement

The method based on local regression of each face's normal independently would lead to a problem of discontinuities between adjacent faces. This results in small perturbations on the denoised surface especially for some CAD models (Fig. 8). To alleviate this issue, we apply a bilateral filter [Zheng et al. 2011] with small kernels to refine


Fig. 6. Mesh models and printed results in the proposed PrintData dataset, consisting of 20 models in total. The close up shows some print errors like the vertical lines.

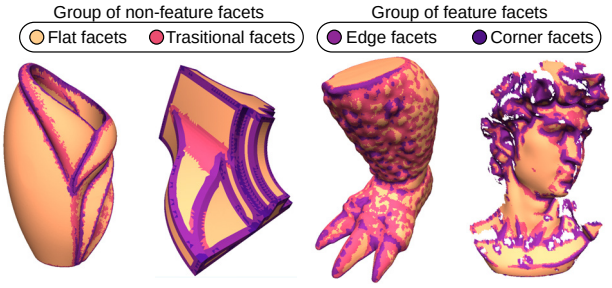


Fig. 7. We classify the facets into two main groups: a group of non-feature facets (flat facets and transitional facets) and a group of feature facets (edge facets and corner facets), and select samples from these two groups for a balanced training.

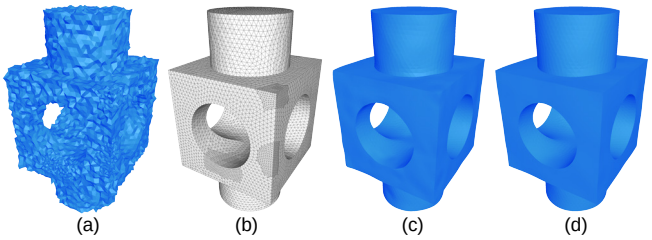


Fig. 8. Normal refinement further improves the output normals from our GCNs output (c), and leads to a slightly better denoised result (d). (b) is the wireframe of the original noise-free mesh showing the irregular sampling. The average angular errors $E_a$ of (c) and (d) are 2.67° and **2.16**°, respectively.

the normals predicted by cascaded GCNs:

$$\dot{n}_i = normalize(\sum_{f_j \in \Omega_i} a_j W_s(||c_j - c_i||) W_r(||\hat{n}_j - \hat{n}_i||)\hat{n}_j), \quad (5)$$

where $\dot{n}_i$ is the final refined normal, $\Omega_i$ is a set of neighbors of $f_i$ defined in [Zhang et al. 2015], $W_s$ and $W_r$ are Gaussian functions with kernels $\sigma_s$ (spatial variance) and $\sigma_r$ (range variance) respectively, and $\hat{n}$ represents the predicted normal outputted by our GCNs. The refinement is applied iteratively with $m$ steps and fixed kernels ($\sigma_s = \bar{l}_e$, where $\bar{l}_e$ is the average distance between neighboring facet centroids across the whole mesh and $\sigma_r = 0.3$). Note that this normal refinement is only applied to the output normals of the last cascaded GCN.

In our experiments, we find that meshes consisting of many large flat areas of features or corrupted by high noise can be refined well after $m \in [8, 16]$ iterations of normal refinement, while meshes with fine features and small noise do not necessarily need normal refinement. Thus, in our experiments, we set $m = 12$ for CAD models and Kinect real-scan models, which often contain high noise, and $m = 1$ for the others. Fig. 9 and 10 show the effect of normal refinement.

### 6.2 Vertex Updating

The vertex updating scheme in our method is the same as [Sun et al. 2007; Zheng et al. 2011], and defined by:

$$v_i^{k+1} = v_i^k + \frac{1}{3|\Omega_i'|} \sum_{f_j \in \Omega_i'} \sum_{e_{ij} \in \partial f_j} n_j^g [n_j^g \cdot (v_j^k - v_i^k)], \quad (6)$$

where $\Omega_i'$ is a set of one-ring neighboring faces for $v_i$ and $n_j^g$ is the denoised normal of facet $f_j$. In our experiments, we find that 15 iterations are sufficient for all our results.

## 7 EXPERIMENTS

We have conducted extensive experiments to evaluate our proposed method with both synthetic and real-scan datasets. All of our experimental tests are conducted on a PC with Intel(R) Core(TM) i7-8770 3.20GHz CPU, 16GB memory, and one GeForce GTX 1080Ti GPU. Our GCNs are implemented using the PyTorch framework and integrated with the PyTorch C++ Library.

### 7.1 Dataset

We build our training datasets on the models from [Wang et al. 2016] (see representative models in Fig. 5), consisting of:

- SysData: There are 14 models as shown in (a), (b) and (c) of Fig. 5. For each mesh, we synthesize three levels of Gaussian (with their deviations set to 0.1, 0.2, and 0.3 of each mesh average edge length) and impulsive noise (the numbers of impulsive vertices are 10%, 20%, and 30% of the mesh vertex numbers) for training. After data balancing (Section 5.2), there are about 2.4M patches in this dataset. This dataset contains three types of mesh models: CAD models, smooth models, and models with rich fine-scale features as shown in Fig. 5.
- Kv1Data: This set contains meshes scanned by Microsoft Kinect v1. We select 48 frames from four scanned models (12 frames for each model) and there are about 910K patches in total after data balancing.
- Kv2Data: These are meshes scanned by Microsoft Kinect v2 and we select 48 frames from four scanned models (12 frames for each model). In total, there are about 560K patches after data balancing.
- K-FData: This set contains meshes scanned by Microsoft Kinect v1 via the Kinect-Fusion technique [Newcombe et al. 2011]. We generate about 200K patches from 3 models (see in Fig. 5 e).

For the real scan datasets (i.e., Kv1Data, Kv2Data, and K-FData), their ground-truth counterparts are provided by [Wang et al. 2016].

For testing, four benchmark datasets are paired with the training datasets accordingly. For SysData, similarly, three levels of Gaussian noise are added onto 29 meshes including 14 CAD-like mesh models, 7 smooth mesh models, and 8 mesh models with rich features, so there are 87 models in total. For real-scan datasets, there are 73 scanned frames for the Kv1Data benchmark, 72 scanned frames for the Kv2Data benchmark, and 4 models for the K-FData benchmark. Apart from these, we build a new real-scan dataset, consisting of 20 scans and their corresponding ground-truth models:

- PrintData: To create this dataset, we first select 20 3D models from an online 3D model repository (3dmag.org). These original 3D models serve as the ground truth. We then prepare physical models by 3D-printing these digital models using a high-end 3D printer (Stratasys Eden260v) and scan the printed models using high-resolution scanners (Artec Spider$^{TM}$ and SHINNING 3D EinScan Pro 2x).

### 7.2 Error Metrics

To evaluate our results and quantitatively compare our method with the state-of-the-art methods, we use two commonly adopted types of metric defined as follows. $E_a$ measures the average normal angular difference between a denoised mesh and the original noise-free mesh.

$$E_a = \frac{1}{N_f} \sum_{f_i^r \in F^r} acos(n_i^r \cdot \tilde{n}_i), \tag{7}$$

where $n_i^r$ and $\tilde{n}_i$ are a normalized normal of the $i$-th facet in the denoised mesh and the normalized normal of the corresponding facet in the ground-truth mesh. $F^r$ is the set of all facets in the denoised mesh.

$$E_v = \frac{1}{N_v L_d} \sum_{v_i^r \in V_M^r} \min_{\tilde{v}_j \in \tilde{V}_M} ||v_i^r - \tilde{v}_j||. \tag{8}$$

Here $E_v$ is the normalized average Hausdorff distance from a denoised mesh to the corresponding ground-truth mesh [Wang et al. 2016], where $L_d$ is the diagonal length of the mesh's bounding box, and $V_M^r$ and $\tilde{V}_M$ are vertices sets of these paired meshes after Monte Carlo sampling.

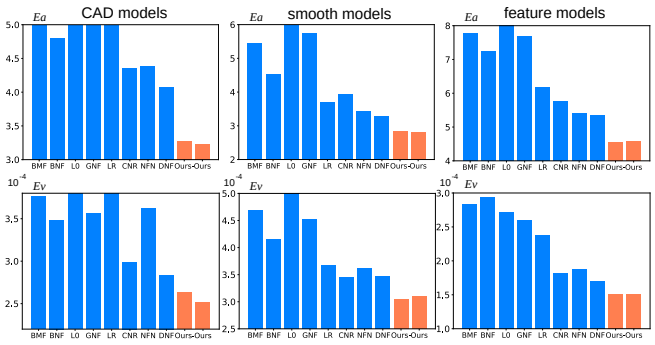


Fig. 9. Quantitative comparisons with the state-of-the-art methods on the SysData benchmark dataset (i.e., CAD models, smooth models, and feature models), using the error metrics defined in Equations 7 and 8. Higher bars are truncated for better illustration. "Ours-" denotes our method without normal refinement.

### 7.3 Results and Comparisons

We compare both qualitatively and quantitatively of our method with the state-of-the-art mesh denoising methods including bilateral mesh filtering (BMF) [Fleishman et al. 2003], bilateral normal filtering (BNF) [Zheng et al. 2011], the $L_0$ smoothing (L0) [He and Schaefer 2013], guided normal filtering (GNF) [Zhang et al. 2015], non-local low-rank based method (LR) [Li et al. 2018], cascaded normal regression (CNR) [Wang et al. 2016], deep normal filtering (DNF) [Li et al. 2020b], and NormalF-Net (NFN) [Li et al. 2020c].

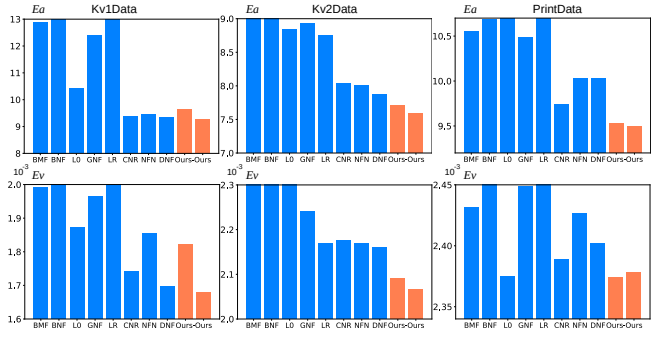


Fig. 10. Comparisons with the state-of-the-art methods on the benchmark datasets: Kv1Data, Kv2Data, and PrintData. "Ours-" denotes our method without normal refinement.

*Synthetic Models.* For the comparisons with the learning-based methods [Li et al. 2020c; Wang et al. 2016], we use the results directly obtained from the original authors (since the authors of [Zhao et al. 2019b] do not provide their source code, we refer to [Li et al. 2020c] for detailed comparisons therein). To compare with BMF, BNF, L0, GNF, and LR, similar to [Wang et al. 2016], we select the best results with fine-tuned groups of parameters for each of them as our competitors. Specifically, we use $\sigma_s = \bar{l}_e$, $\sigma_r = 0.35$, $n_n = 20$, and $n_v = 20$ (20 times normal iteration and vertex updating)] in BNF, $\lambda = 0.02\bar{l}_e^2\bar{\gamma}$ ($\bar{\gamma}$ is the average dihedral angle of a mesh) in L0, $\sigma_s = \bar{l}_e$, $\sigma_r = 0.35$, $n_n = 20$, and $n_v = 20$ in GNF, $\sigma_M = 0.2$, $n_n = 10$, and $n_v = 10$ in LR.

We quantitatively compare the normal angular error $E_a$ and the vertex distance error $E_v$ in the synthetic benchmark dataset SysData for all the competing methods, including ours. Fig. 9 shows the results. In this benchmark, our method consistently outperforms the compared methods in all the categories, including CAD models, smooth models, and models with rich features. Several representative visual comparisons are shown in Fig. 11. In this test, the performances of NormalF-Net (NFN) [Li et al. 2020c] and the non-local recovery method (LR) [Li et al. 2018] are similar because they both build upon the non-local representations. However, neither of them can handle well the CAD models with sharp features. This is a nature of the optimization-based methods, which tend to smooth out sharp features during minimization. As for DNF ([Li et al. 2020b]), lacking mesh topological information in the normal prediction step limits its ability to remove noise. The method of cascaded regression (CNR) [Wang et al. 2016] performs moderately well in both types of models with sharp features and fine features. Nevertheless, our method is able to recover sharp features better even from high noise, as shown in the top two rows of Fig. 11 while also preserving the fine-scale features as shown in the bottom two rows, clearly demonstrating the efficacy of our algorithm.

Fig. 12 shows two examples of denoised results of our method on meshes with extremely high-level Gaussian noise (level 0.6) and impulsive noise (60%, both in percentage and strength). The noise level in those models is significantly higher than that of any model in our training sets. Still, our method is able to denoise such noisy models and consistently performs better than the compared methods. In these examples, CNR cannot handle large positional bias between the noisy meshes and ground-truth meshes (as admitted by the authors of CNR), and thus it treats some of the impulsive and high noise as features and preserves them (e.g., see the face of the Nicolo model and the edge features on the Fandisk). In both examples, GNF [Zhang et al. 2015] tends to smooth out small features and preserve only the dominant features. On the other hand, NFN [Li et al. 2020c] performs better than CNR and GNF on the Nicolo model with fine-scale features. However, it performs worse than GNF on the Fandisk model with sharp features. This agrees with the findings above. Besides, it can be observed that DNF performs worse for high-level noises. To show the stability of our method, we also offer an example with the irregular face resolution corrupted by two different types of noise with three different levels (first by Gaussian noise of levels 0.1 and 0.3 in different regions and then by the impulsive noise of level 0.5) in Fig. 8. It can be seen that our GCNs produce a high-quality denoising result, which is further improved by the normal refinement step.

*Real-scan Models.* For the comparisons on the real scans, we also select the best results with fine-tuned groups of parameters for BNF, $L_0$, GNF, and LR. Specifically, we set $\sigma_s = \bar{l}_e$, $\sigma_r = 0.45$, $n_n = 20$, and $n_v = 20$ in BNF, $\lambda = 0.06\bar{l}_e^2\bar{\gamma}$ in $L_0$, $\sigma_s = \bar{l}_e$, $\sigma_r = 0.45$, $n_n = 20$, and $n_v = 20$ in GNF, and $\sigma_M = 0.4$, $n_n = 10$, and $n_v = 10$ in LR.

The quantitative comparisons are shown in Fig. 10. Our method also consistently achieves the best results in all Kv1Data, Kv2Data, and PrintData real-scan benchmarks. The representative results of qualitative comparisons on Kv2Data and K-FData are shown in Fig. 13. Note that in these real scans, due to the low capture quality, the fine features of the acquired models are often highly corrupted or even destroyed by noise (e.g., the eyes and the mouth in the statue of Fig. 13 have been somehow erased and are difficult to perceive even for humans). Thus it is difficult for most of the denoising methods to faithfully recover these features, so is ours. Nonetheless, our method still receives the lowest reconstruction errors while also preserving the features better than the previous methods. Moreover, the models in Kv1Data, Kv2Data, and K-FData often do not exhibit complex geometric features and are not very suitable for testing the limit of the feature-preserving ability of different methods.

To further demonstrate the performance of our method, we apply denoising on the proposed PrintData. As shown in Fig. 14, the scans in our dataset often involve more geometric details than the scans by Kinect but still suffer from noise of small scales. The Kv1Data, Kv2Data, and K-FData use high-resolution scanned models (by an Artec Spider$^{TM}$ scanner) as ground truth and low-precision Kinect scanned models as noisy input. In contrast, we take existing 3D digital models as our ground truth and take the scanned models as noisy input. Due to the quality of 3D printing (it may introduce larger errors than 3D scanning), there are naturally small differences between the printed models and the ground truth, e.g., the vertical lines in the first row of Fig. 14 (the corresponding printed result is shown in Fig. 6). Our results preserve the original details and meanwhile have the lower errors. Also, our results have better performance around the sharp features especially in CAD models (the second row in Fig.14). Again, our approach consistently preserves the features (e.g., in the ear of the Goblin and the bottom letters shown in the third row of Fig. 14 and the head of the Stitch and the guitar in the fourth frow of Fig. 14) better than the compared methods.

*Timing.* Following [Wang et al. 2016], we train our network for synthetic and real scans separately. Our method takes about 6-8 hours for training the network of synthetic models and 2-3 hours for training the network of real scan models.

Our learning-based method is also efficient at runtime and its performance comparisons with the previous learning-based methods on typical models are summarized in Table 1. The runtime largely depends on the complexity of network architectures. It can be seen that our method runs one magnitude faster than DNF [Li et al. 2020b] and NormalF-Net [Li et al. 2020c] and two magnitudes faster than NormalNet [Zhao et al. 2019b], which is based on 3D convolutions on voxels. The MLP-based method [Wang et al. 2016]

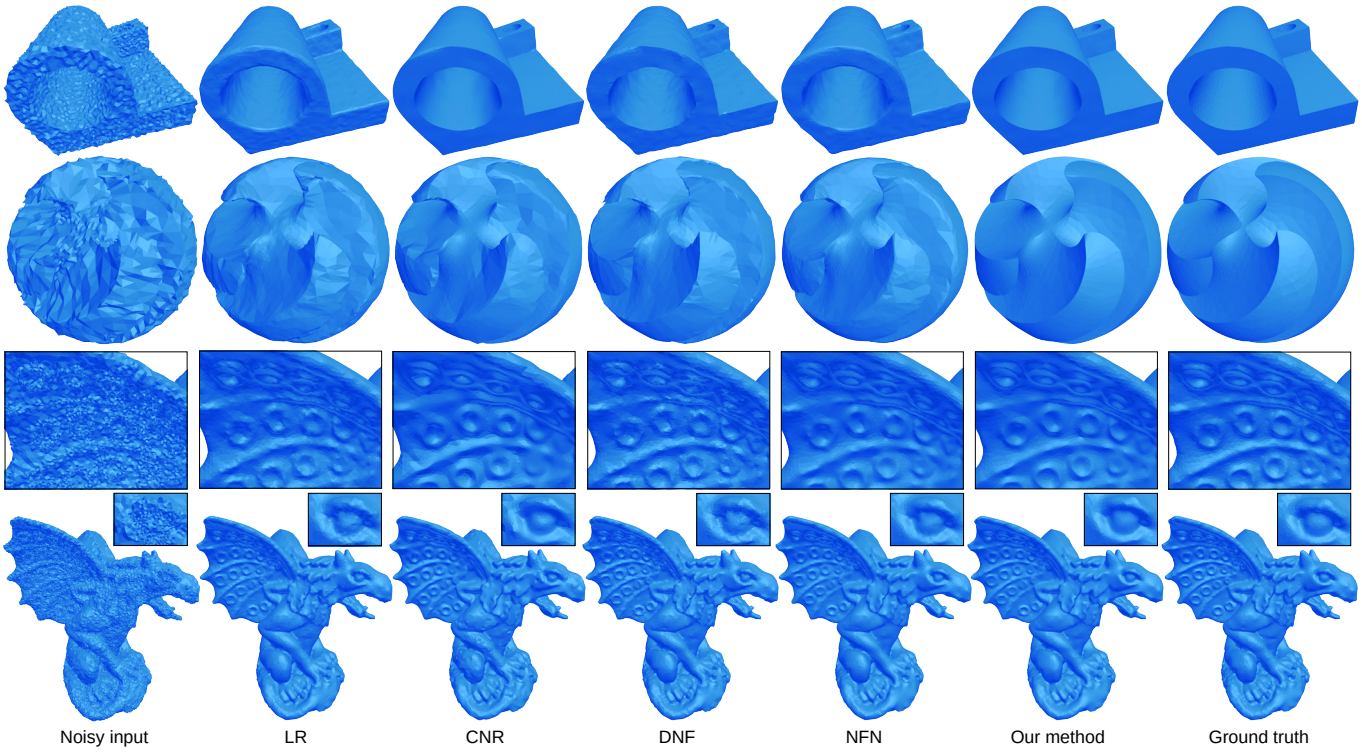


Fig. 11. Visual comparisons of various methods on the SysData benchmark dataset. Models: Joint, Sharp-Shpere, Carter, and Gargoyle with the Gaussian noise of level 0.3, 0.3 and 0.3 (mean edge length), respectively. The average normal angular errors $E_a$ (from left to right) are: (top example) 28.65°, 6.52°, 2.22°, 4.50°, 5.02°, and **1.86**°; (middle example) 33.17°, 12.17°, 8.03°, 8.34°, 8.84°, and **4.39**°; (bottom example) 31.78°, 7.56°, 8.72°, 7.98°, 6.89°, and **5.25**°.

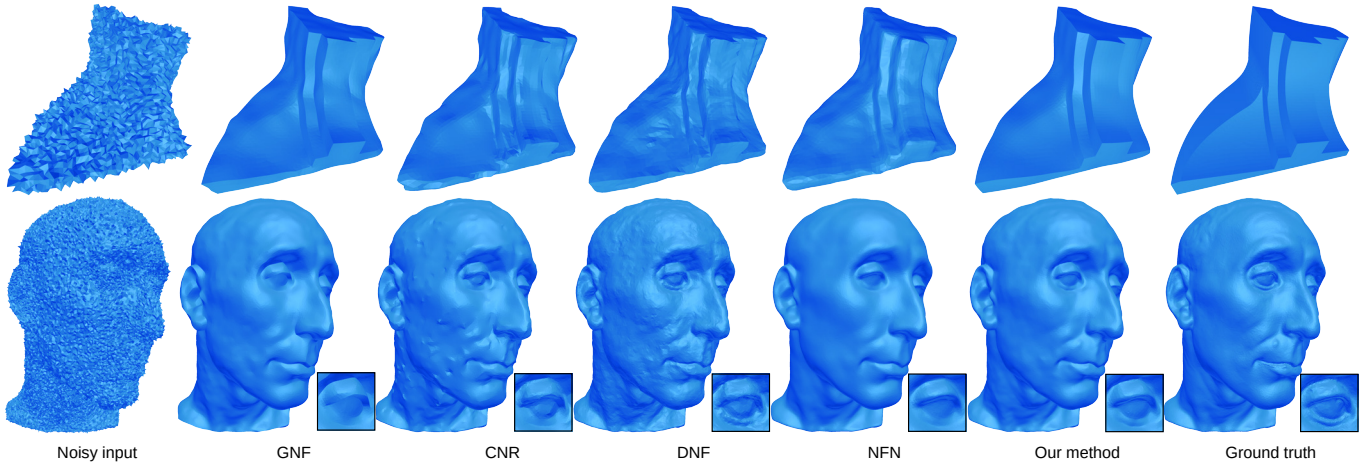


Fig. 12. Visual comparisons of denoised results on models with extreme Gaussian or impulsive noise. Models: Fandisk with the Gaussian noise of level 0.6, Nicolo with the impulsive noise of level 0.6 (both in percentage and strength). The average normal angular errors (from left to right) are: ($1^{st}$ row) 44.71°, 6.72°, 8.60°, 7.52°, 11.24°, and **3.87**°; ($2^{nd}$ row) 36.65°, 6.74°, 6.99°, 6.82°, 5.35°, and **5.03**°.

receives the best performance due to its simplicity in the network architecture (with only a few fully connected layers). Nonetheless, our method achieves the best denoising quality while being well balanced between the efficacy and efficiency among the learning-based methods.

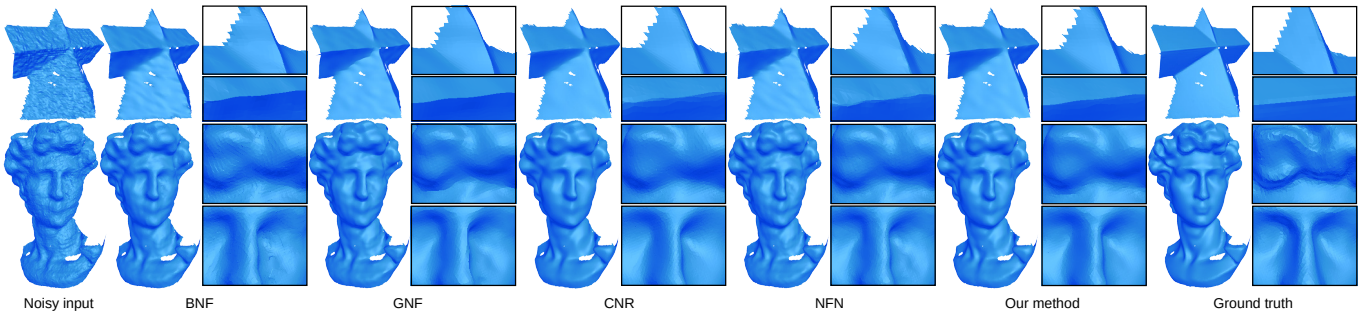


Fig. 13. The denoised Kinect v2 ($1^{st}$ row) single-frame meshes and Kinect Fusion models ($2^{nd}$ row). From left to right: noisy input, denoised results of BNF [Zheng et al. 2011], GNF [Zhang et al. 2015], CNR [Wang et al. 2016], NFN [Li et al. 2020c], ours and the GT. The average normal angular errors (from left to right) are: ($1^{st}$ row) 20.85°, 9.25°, 7.96°, 6.93°, 7.51°, and **6.58**°; ($2^{nd}$ row) 17.89°, 12.99°, 11.95°, 11.94°, 12.15°, and **11.61**°.

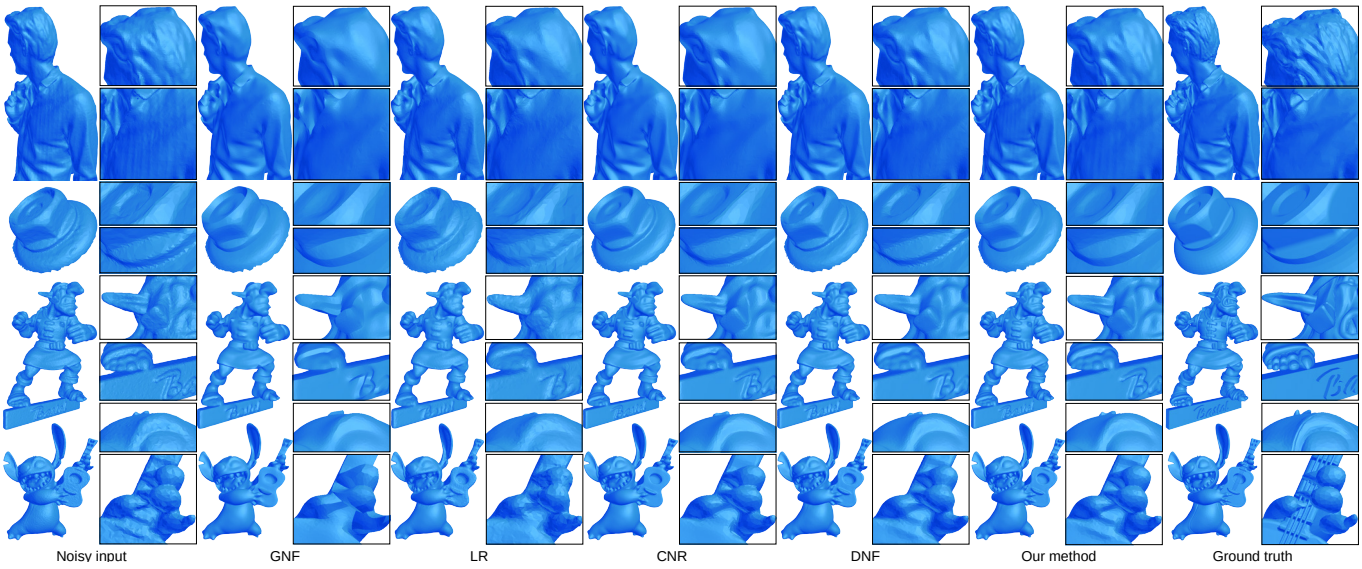


Fig. 14. Denoising real-scan meshes in PrintData. From left to right: noisy input, denoised results of GNF [Zhang et al. 2015], LR [Li et al. 2018], CNR [Wang et al. 2016], DNF [Li et al. 2020b], ours, and the GT. The average normal angular errors (from left to right) are: ($1^{st}$ row) 10.88°, 12.06°, 12.27°, 11.02°, 10.74°, and **10.54**°; ($2^{nd}$ row) 18.08°, 18.68°, 17.96°, 17.23°, 17.67°, and **17.08**°, ($3^{rd}$ row) 16.16°, 16.96°, 18.80°, 15.53°, 15.96°, and **14.95**°, ($4^{th}$ row) 8.08°, 7.50°, 7.97°, 6.11°, 6.68°, and **5.84**°.

## 7.4 Ablation Studies

In this subsection, we introduce the ablation studies to show the impact of various algorithmic components of our method.

*Patch Size.* The size of patches $r$ in Eqn. (1) determines the receptive filed of our GCNs, i.e., how much local geometry we can see in a patch w.r.t. an entire mesh. This is an important hyper-parameter. Essentially, $r$ should be set to a fixed value, e.g., $5\%$ of the diagonal length of the mesh's bounding box to make the patch sizes consistent. In our implementation, we set $r = ka_i^{\frac{1}{2}}$ as a value relevant to the triangle area to accommodate different surface samplings. Besides, the input to our GCN should be padded into a fixed size of nodes since we train the GCN with batches.

We test over six scales of patches, i.e., $k = 2, 3, 4, 6, 8$, and $10$ ($k$ is defined in Sec. 4.1). For training in batches, we fix the number of graph nodes to 16, 32, 64, 128, 256, and 512 according to the values of $k$ (in case the number of facets within the patch does not equal to the defined number, we perform random shrinkage or extension to the local patch). Fig. 15 shows the corresponding performance of these choices. We find that setting $k = 4$ makes our GCN regress well enough for the synthetic data, while setting $k = 8$ is sufficient for the real-scan Kinect data. A desired value of $k$ for the real-scan Kinect data is larger, mainly because the surface resolution of the real-scan data is often much larger than that of the synthetic ones and the low-precision scans often bring in large-scale noise. The experiments show that as long as the patch sufficiently covers local regions, the performance is stable. A visualization effect of meshes denoised with different patch sizes is shown in Fig. 19.

Since our patches are generated within a sphere, some noisy, disconnected facets might be included in our patch graphs if the

Table 1. Time comparisons with the state-of-the-art learning-based denoising methods: NormalNet (NN) [Zhao et al. 2019b], CNR [Wang et al. 2016], DNF [Li et al. 2020b], and NormalF-Net (NFN) [Li et al. 2020c]. The running time is in seconds. Please find the corresponding models in the $2^{nd}$ in Fig. 11, the rightmost one in Fig. 5 (f), the bottom one in Fig. 5 (h), and the $4^{th}$ in Fig. 11.

| **Model** | Sharp-Sphere | Fertility | Eros | Gragoyle |
|---|---|---|---|---|
| **Faces** | 20882 | 27954 | 100000 | 171112 |
| **NN** | 836.12s | 1132.42s | 9163.24s | 20763.28s |
| **CNR** | 1.27s | 1.71s | 5.16s | 10.04s |
| **DNF** | 352.39s | 494.94s | 1912.74s | 3376.69s |
| **NFN** | 97.37s | 109.63s | 480.55s | 975.05s |
| **Ours** | 11.88s | 12.97s | 59.95s | 145.96s |

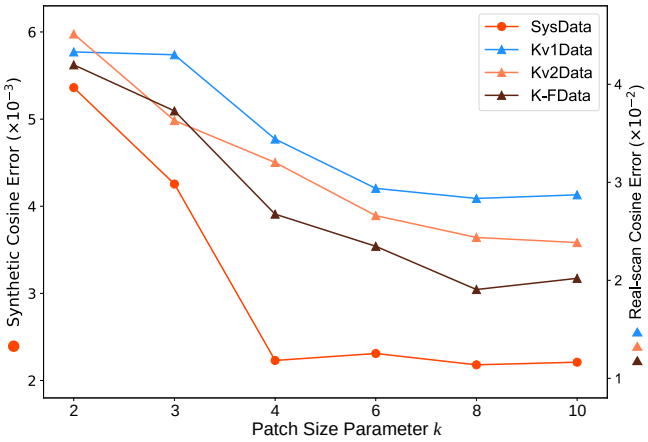


Fig. 15. Impact of patch size parameter $k$ on different datasets. For the synthetic data (SysData), a value of 4 leads to satisfactory results while for the real scan data (i.e., Kv1Data, Kv2Data, K-FData), $k = 8$ leads to sufficient good results.

geometry around a facet has a broken or thin structure. Fig. 16 shows such an example, where facets in the disconnect regions and on the other side of the lens are presented in the same patch graph. Here $k = 8$ is used. Nevertheless, unlike DNF [Li et al. 2020b], our method is not sensitive to such thin or broken structures and produces satisfactory results, as shown in Fig. 16. This is due to that our static EdgeConv inherently exploits a mesh's original structure and our dynamic EdgeConv distinguishes which faces features are helpful, thus making our method robust to such noisy representations.

*Number of Graph Convolution Layers.* To examine how the number of layers (i.e., how deep) of our GCN affects the results, we conduct the following experiments. For the number of layers of graph convolution, i.e., $L_e + L_d$, we test the following numbers: 4 ($L_e, L_d = 2$), 6 ($L_e, L_d = 3$), and 8 ($L_e, L_d = 4$). We set $L_e = L_d$ for combinatorial simplicity. Fig. 17 plots the performance. We find that the performance does not increase much after 6, showing that a moderate size of GCN is sufficient in learning the local geometry

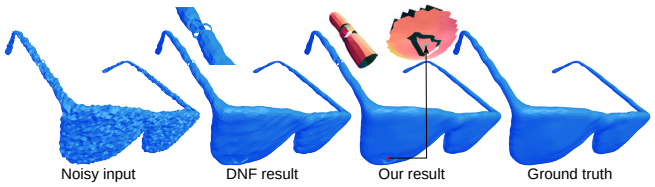


Fig. 16. A denoising result of a broken thin-structured model. Our method is able to learn features well even from patches with disconnected graph structures. The average normal angular errors $E_a$ are: $28.114^\circ$ (input), $7.35^\circ$ (DNF's result), and $\mathbf{4.70}^\circ$ (our result).

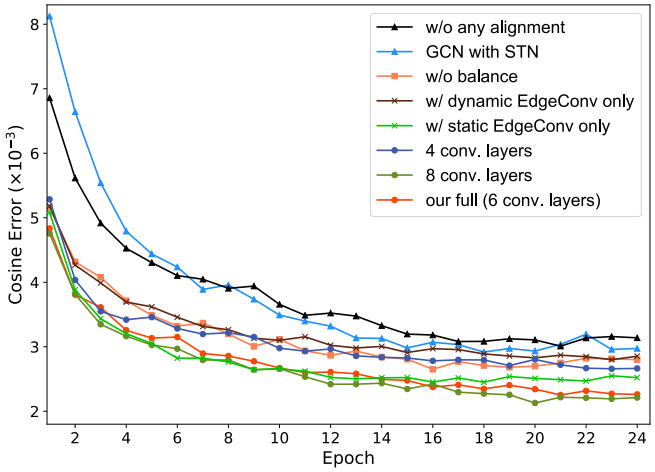


Fig. 17. Ablation studies on various design choices of our algorithm.

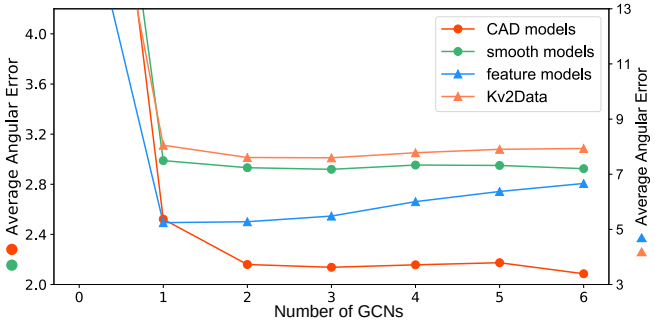


Fig. 18. Test on the number of GCNs. Cascading a larger number of GCNs does not necessarily increase the performance. In fact, it might over-smooth certain fine-scale features when this number increases.

details of our aligned patches. Thus we use $L_e = 3$ and $L_d = 3$ for our first GCN and $L_e = 2$ and $L_d = 2$ for the rest.

*Number of GCNs.* In this test, we show the necessity of multiple GCNs. To do so, we randomly sample a set of representative models in each category of the SysData benchmark dataset, including 4 CAD models, 4 smooth models, and 4 models with rich features (0.1–0.3 levels of Gaussian noise are added to each model). We also sample a similar set of models in the Kv2Data benchmark dataset.

We then run the test on these sampled sets using 1 – 5 GCNs for denoising. The quantitative comparisons of the performance are shown in Fig. 18. It can be seen that the performance of adding more GCNs stops improving when the number is greater than 2 for the CAD models, smooth models, and the Kinect v2 models with low-frequency features. For the models with rich fine features, adding more GCNs might result in an over-smoothed effect since under such circumstances, it is hard to distinguish those fine features from noise. Hence, we use 2 GCNs for denoising in all our experiments. Visual results of using 1 and 2 GCNs are shown in Fig. 19.

*Patch Alignment.* To show the influence of our patch alignment, we replace our tensor voting with a spatial transform network (STN) [Wang et al. 2019] module implemented in our GCN. STN is widely used in point cloud processing works [Qi et al. 2017a; Wang et al. 2019] to eliminate spatial variations among the inputs. Fig. 17 (the curve of "GCN with STN") shows that STN does not work as well as our normal tensor voting. This agrees with the finding that adding a spatial transformation module does not improve the performance much in [Qi et al. 2017a] and partially proves that spatial transformation is indeed not easy to learn by neural networks. On the other hand, if we do not use any alignment scheme, the performance decreases (see the curve of "w/o any alignment" in Fig. 17). Patch alignment may be influenced by the noise level, however, our cascaded optimization may help progressively correct the errors (Fig. 19).

*Data Balancing.* We also examine the influence of our data balancing strategy. Fig. 17 shows that the performance of our network increases slightly due to the adopted data balancing strategy. Let us denote $r$ as the ratio of the number of featured facets compared to that of non-feature ones. Without data balancing, the value of $r$ is approximately 0.1. We test over $r = \{0.5, 1, 1.5, 2, 5, 10\}$ and find that the performance increases very slightly after $k = 1.5$ and starts to decrease after $r = 10$. Hence, throughout our experiments, we use $r = 1.5$ for data balancing. The data balancing allows more effective learning of the underlying features.

*Static and Dynamic Graph Convolutions.* We train our GCNs with both static and dynamic EdgeConv. To prove their effectiveness, we examine the following alternatives: a network with static EdgeConv only and a network with dynamic EdgeConv only. Again, it is witnessed that a combination of the two allows more information flow from both neighboring graph nodes and these potentially unconnected ones, thus leading to a more effective learning of the features, as shown in Fig. 17. On the other hand, the network with only EdgeConv leads to a better performance than the one with only dynamic EdgeConv. This shows that the original graph structure in the mesh is already very informative.

*KNN.* We use KNN to dynamically construct graph structures in dynamic EdgeConv. We examine the influence of the performance on different values of $K$. We test $K = 4, 8, 12$, and 16. The performance of $K = 4$ is similar to that of using static EdgeConv only and does not improve after $K = 8$. Thus we use $K = 8$ throughout our experiments.

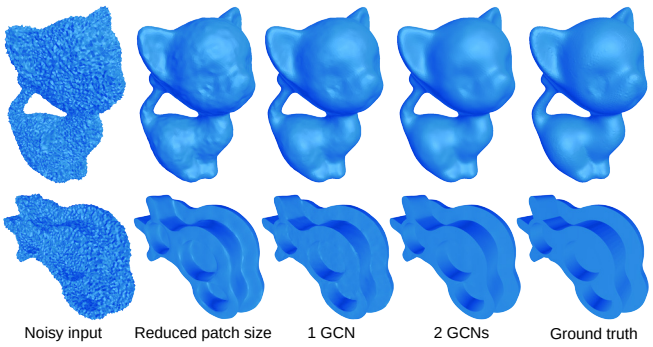


Fig. 19. Comparative denoised results of different patch sizes ($2^{nd}$ column: $k = 2$) and different numbers of cascaded GCNs. The average angular errors (from left to right) are: ($1^{st}$ row) 40.52°, 5.58°, 3.84°, and **3.29**°; ($2^{nd}$ row) 33.41°, 5.61°, 2.47°, and **2.01**°.

### 7.5 Implementation Details

As mentioned before, our GCNs have different numbers of convolution and MLP layers. In our experiments, the numbers of feature channels are set as (64, 128, 128, 256, 256, 256, 1024, 512, 256, 64) in the first GCN and (64, 128, 256, 256, 512, 256, 64) in the other GCNs. In the training stage, we use Adam ($\beta_1 = 0.9$ and $\beta_2 = 0.999$) for optimization with the base learning rate 0.0001. We set the batch size as 128 and train 24 epochs for the first GCN, and 16 epochs for the rest. At runtime, we also regress normals in batches. Due to the limited GPU memory, we set the batch size as 720 for patch size $k = 4$ and set the batch size as 160 for patch size $k = 8$.

## 8 LIMITATIONS

Our method has several limitations. First, since we learn the unknown noise patterns from massive data, the capacity of our method is limited by the training data. Second, although our method is able to generalize to unseen noise levels, it could still fail to recover the underlying features once they are deeply corrupted by noise. Such effects have been demonstrated in examples of this paper with extremely high noise (Fig. 12) or low quality noisy input from low-end depth cameras (Fig. 13). Third, one assumption of our method is that the geometry variation of the noise and that of the underlying features are different so that both variations can be well modeled by our GCNs. If this assumption is broken, our method would fail to distinguish features from noise and tend to either smooth out the features or preserve the noise. This is often the case with the meshes containing many fine-scale features (Fig. 1). A more robust and fine-grained classification module might be helpful but cannot completely solve this ill-posed problem due to the inherent ambiguity. Fourth, like most of the existing feature-preserving denoising methods, our method does not change the mesh connectivity, thus we cannot remove topological noise. Utilizing dynamic graph convolution that explicitly updates the mesh connectivity may be helpful to resolve this issue. We consider it as an orthogonal future work.

## 9 CONCLUSION AND DISCUSSION

In this paper, we have presented the first GCN-based approach for feature-preserving mesh denoising. Our method takes a triangular mesh as input and employs multiple GCNs to progressively regress the noise-free normals of the underlying surface patches. An essential ingredient of our method is to represent the local surface patches as graphs in the dual space of triangles. We show such an intact representation allows convolution operations to be performed directly on the mesh surface to effectively learn geometric features. We employ both static and dynamic graph convolutions to aggregate features from both connected neighbors and unconnected ones, enabling a more effective feature learning. Extensive experimental results show that our GCN models achieve the new state-of-the-art results while being well balanced between efficacy and efficiency.

Although our current implementation relies on triangular meshes, it can be easily adapted to other representations, for instance, quad meshes. We are also interested in extending our method for denoising unorganized point clouds or non-manifold meshes. For point clouds, it might be tricky if we directly perform denoising on normals since the subsequent vertex updating step is not feasible if the connectivity is unknown. One possibility is to regress the point positions with dynamic EdgeConv [Rakotosaona et al. 2019], but in a local and progressive manner. For non-manifold meshes, we may require additional such training data and a new vertex updating scheme. Moreover, we believe that our framework can be extended for applications such as geometric texture synthesis, mesh feature enhancement, mesh topological noise removal, and shape deformation. We also believe that the general framework of regressing a complex function over a 3D surface in a cascaded and local manner could be inspiring for various geometry tasks, such as surface reconstruction [Jiang et al. 2020] and super-resolution.

## ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers for their constructive comments. This work was supported in part by the National Key Research Development Program of China (2018YFE0100900) and the NSF China (No. 61890954, 61772024, 61732016).

## REFERENCES

Andrew Adams, Natasha Gelfand, Jennifer Dolson, and Marc Levoy. 2009. Gaussian kd-trees for fast high-dimensional filtering. In *ACM SIGGRAPH 2009 papers*. 1–12.

Forest Agostinelli, Michael R Anderson, and Honglak Lee. 2013. Adaptive multi-column deep neural networks with application to robust image denoising. In *Advances in Neural Information Processing Systems*. 1493–1501.

Chandrajit L. Bajaj and Guoliang Xu. 2003. Anisotropic Diffusion of Surfaces and Functions on Surfaces. *ACM Trans. Graph.* 22, 1 (2003), 4–32.

Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. 2014. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203* (2014).

Harold C Burger, Christian J Schuler, and Stefan Harmeling. 2012. Image denoising: Can plain neural networks compete with BM3D?. In *2012 IEEE conference on computer vision and pattern recognition*. 2392–2399.

U. Clarenz, U. Diewald, and M. Rumpf. 2000. Anisotropic Geometric Diffusion in Surface Processing. In *Proceedings of the Conference on Visualization '00*. 397–405.

Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in neural information processing systems*. 3844–3852.

James R. Diebel, Sebastian Thrun, and Michael Brünig. 2006. A Bayesian Method for Probable Surface Reconstruction and Decimation. *ACM Trans. Graph.* 25, 1 (2006), 39–59.

Hanqi Fan, Yizhou Yu, and Qunsheng Peng. 2010. Robust feature-preserving mesh denoising based on consistent subneighborhoods. *IEEE Transactions on Visualization and Computer Graphics* 16, 2 (2010), 312–324.

Yutong Feng, Yifan Feng, Haoxuan You, Xibin Zhao, and Yue Gao. 2019. MeshNet: Mesh neural network for 3D shape representation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 8279–8286.

Shachar Fleishman, Iddo Drori, and Daniel Cohen-Or. 2003. Bilateral Mesh Denoising. In *ACM SIGGRAPH 2003 Papers*. 950–953.

Rana Hanocka, Amir Hertz, Noa Fish, Raja Giryes, Shachar Fleishman, and Daniel Cohen-Or. 2019. MeshCNN: A Network with an Edge. *ACM Trans. Graph.* 38, 4 (2019).

Lei He and Scott Schaefer. 2013. Mesh Denoising via L0 Minimization. *ACM Trans. Graph.* 32, 4 (2013).

Chiyu Max Jiang, Avneesh Sud, Ameesh Makadia, Jingwei Huang, Matthias Nießner, and Thomas Funkhouser. 2020. Local Implicit Grid Representations for 3D Scenes.

Kyong Hwan Jin, Michael T McCann, Emmanuel Froustey, and Michael Unser. 2017. Deep convolutional neural network for inverse problems in imaging. *IEEE Transactions on Image Processing* 26, 9 (2017), 4509–4522.

Thouis R Jones, Frédo Durand, and Mathieu Desbrun. 2003. Non-iterative, feature-preserving mesh smoothing. In *ACM SIGGRAPH 2003 Papers*. 943–949.

Kai-Wah Lee and Wen-Ping Wang. 2005. Feature-preserving mesh denoising via bilateral normal filtering. In *Ninth International Conference on Computer Aided Design and Computer Graphics (CAD-CG'05)*. 6–pp.

Guohao Li, Matthias Muller, Ali Thabet, and Bernard Ghanem. 2019. Deepgcns: Can gcns go as deep as cnns?. In *Proceedings of the IEEE International Conference on Computer Vision*. 9267–9276.

Xianzhi Li, Ruihui Li, Guangyong Chen, Chi-Wing Fu, Daniel Cohen-Or, and Pheng-Ann Heng. 2020a. A Rotation-Invariant Framework for Deep Point Cloud Analysis. *CoRR* abs/2003.07238 (2020).

Xianzhi Li, Ruihui Li, Lei Zhu, Chi-Wing Fu, and Pheng-Ann Heng. 2020b. DNF-Net: a Deep Normal Filtering Network for Mesh Denoising. *IEEE Transactions on Visualization and Computer Graphics* (2020).

Xianzhi Li, Lei Zhu, Chi-Wing Fu, and Pheng-Ann Heng. 2018. Non-Local Low-Rank Normal Filtering for Mesh Denoising. *Comput. Graph. Forum* 37, 7 (2018), 155–166.

Zhiqi Li, Yingkui Zhang, Yidan Feng, Xingyu Xie, Qiong Wang, Mingqiang Wei, and Pheng-Ann Heng. 2020c. NormalF-Net: Normal filtering neural network for feature-preserving mesh denoising. *Computer-Aided Design* (2020), 102861.

Jonathan Masci, Davide Boscaini, Michael Bronstein, and Pierre Vandergheynst. 2015. Geodesic convolutional neural networks on riemannian manifolds. In *Proceedings of the IEEE international conference on computer vision workshops*. 37–45.

Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodola, Jan Svoboda, and Michael M Bronstein. 2017. Geometric deep learning on graphs and manifolds using mixture model cnns. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 5115–5124.

Richard A. Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J. Davison, Pushmeet Kohli, Jamie Shotton, Steve Hodges, and Andrew Fitzgibbon. 2011. KinectFusion: Real-Time Dense Surface Mapping and Tracking. In *Proceedings of the 2011 10th IEEE International Symposium on Mixed and Augmented Reality (ISMAR '11)*. 127–136.

Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. 2017a. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J. Guibas. 2017b. PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*. 5099–5108.

Marie-Julie Rakotosaona, Vittorio La Barbera, Paul Guerrero, Niloy J. Mitra, and Maks Ovsjanikov. 2019. POINTCLEANNET: Learning to Denoise and Remove Outliers from Dense Point Clouds. *CoRR* abs/1901.01060 (2019).

Jonas Schult, Francis Engelmann, Theodora Kontogianni, and Bastian Leibe. 2020. DualConvMesh-Net: Joint Geodesic and Euclidean Convolutions on 3D Meshes. *CoRR* abs/2004.01002 (2020).

Yuzhong Shen and Kenneth E Barner. 2004. Fuzzy vector median-based surface smoothing. *IEEE Transactions on Visualization and Computer Graphics* 10, 3 (2004), 252–265.

Takafumi Shimizu, Hiroaki Date, Satoshi Kanai, and Takeshi Kishinami. 2005. A new bilateral mesh smoothing method by recognizing features. In *Ninth International Conference on Computer Aided Design and Computer Graphics (CAD-CG'05)*. 6–pp.

Martin Simonovsky and Nikos Komodakis. 2017. Dynamic edge-conditioned filters in convolutional neural networks on graphs. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 29–38.

Vladimiros Sterzentsenko, Leonidas Saroglou, Anargyros Chatzitofis, Spyridon Thermos, Nikolaos Zioulis, Alexandros Doumanoglou, Dimitrios Zarpalas, and Petros Daras. 2019. Self-supervised deep depth denoising. In *Proceedings of the IEEE International Conference on Computer Vision*. 1242–1251.

Xianfang Sun, Paul L Rosin, Ralph Martin, and Frank Langbein. 2007. Fast and effective feature-preserving mesh denoising. *IEEE Transactions on Visualization and Computer Graphics* 13, 5 (2007), 925–938.

Tolga Tasdizen, Ross Whitaker, Paul Burchard, and Stanley Osher. 2002. Geometric Surface Smoothing via Anisotropic Diffusion of Normals. In *Proceedings of the Conference on Visualization '02*. 125–132.

Diego Valsesia, Giulia Fracastoro, and Enrico Magli. 2019. Learning Localized Generative Models for 3D Point Clouds via Graph Convolution. In *International Conference on Learning Representations*.

Jun Wang, Xi Zhang, and Zeyun Yu. 2012. A cascaded approach for feature-preserving surface mesh denoising. *Computer-Aided Design* 44, 7 (2012), 597–610.

Peng-Shuai Wang, Yang Liu, Yu-Xiao Guo, Chun-Yu Sun, and Xin Tong. 2017. O-CNN: Octree-Based Convolutional Neural Networks for 3D Shape Analysis. *ACM Trans. Graph.* 36, 4, Article 72 (2017), 11 pages.

Peng-Shuai Wang, Yang Liu, and Xin Tong. 2016. Mesh Denoising via Cascaded Normal Regression. *ACM Trans. Graph.* 35, 6 (2016).

Ruimin Wang, Zhouwang Yang, Ligang Liu, Jiansong Deng, and Falai Chen. 2014. Decoupling Noise and Features via Weighted L1-Analysis Compressed Sensing. *ACM Trans. Graph.* 33, 2 (2014).

Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E. Sarma, Michael M. Bronstein, and Justin M. Solomon. 2019. Dynamic Graph CNN for Learning on Point Clouds. *ACM Trans. Graph.* 38, 5 (2019).

M Wei, X Guo, J Huang, Haoran Xie, H Zong, R Kwan, FL Wang, and J Qin. 2019a. Mesh defiltering via cascaded geometry recovery. *Comput. Graph. Forum* 38, 7 (2019), 591–605.

Mingqiang Wei, Jin Huang, Xingyu Xie, Ligang Liu, Jun Wang, and Jing Qin. 2019b. Mesh denoising guided by patch normal co-filtering via kernel low-rank recovery. *IEEE transactions on visualization and computer graphics* 25, 10 (2019), 2910–2926.

Mingqiang Wei, Jinze Yu, Wai-Man Pang, Jun Wang, Jing Qin, Ligang Liu, and Pheng-Ann Heng. 2015. Bi-normal filtering for mesh denoising. *IEEE transactions on visualization and computer graphics* 21, 1 (2015), 43–55.

Junyuan Xie, Linli Xu, and Enhong Chen. 2012. Image denoising and inpainting with deep neural networks. In *Advances in neural information processing systems*. 341–349.

Sunil Kumar Yadav, Ulrich Reitebuch, and Konrad Polthier. 2019. Robust and high fidelity mesh denoising. *IEEE transactions on visualization and computer graphics* 25, 6 (2019), 2304–2310.

Hirokazu Yagou, Yutaka Ohtake, and Alexander Belyaev. 2002. Mesh smoothing via mean and median filtering applied to face normals. In *Geometric Modeling and Processing. Theory and Applications. GMP 2002. Proceedings*. 124–131.

Hirokazu Yagou, Yutaka Ohtake, and Alexander G Belyaev. 2003. Mesh denoising via iterative alpha-trimming and nonlinear diffusion of normals with automatic thresholding. In *Proceedings Computer Graphics International 2003*. 28–33.

Shi Yan, Chenglei Wu, Lizhen Wang, Feng Xu, Liang An, Kaiwen Guo, and Yebin Liu. 2018. Ddrnet: Depth map denoising and refinement for consumer depth cameras using cascaded cnns. In *Proceedings of the European conference on computer vision (ECCV)*. 151–167.

Shin Yoshizawa, A. Belyaev, and H. . Seidel. 2006. Smoothing by Example: Mesh Denoising by Averaging with Similarity-Based Weights. In *IEEE International Conference on Shape Modeling and Applications 2006 (SMI'06)*. 9–9. https://doi.org/10.1109/SMI.2006.38

Kai Zhang, Wangmeng Zuo, Yunjin Chen, Deyu Meng, and Lei Zhang. 2017. Beyond a gaussian denoiser: Residual learning of deep cnn for image denoising. *IEEE Transactions on Image Processing* 26, 7 (2017), 3142–3155.

Wangyu Zhang, Bailin Deng, Juyong Zhang, Sofien Bouaziz, and Ligang Liu. 2015. Guided mesh normal filtering. *Comput. Graph. Forum* 34, 7 (2015), 23–34.

Wenbo Zhao, Xianming Liu, Shiqi Wang, Xiaopeng Fan, and Debin Zhao. 2019a. Graph-based Feature-Preserving Mesh Normal Filtering. *IEEE Transactions on Visualization and Computer Graphics* (2019).

Wenbo Zhao, Xianming Liu, Yongsen Zhao, Xiaopeng Fan, and Debin Zhao. 2019b. Normalnet: Learning based guided normal filtering for mesh denoising. *CoRR* abs/1903.04015 (2019).

Youyi Zheng, Hongbo Fu, Oscar Kin-Chung Au, and Chiew-Lan Tai. 2011. Bilateral normal filtering for mesh denoising. *IEEE Transactions on Visualization and Computer Graphics* 17, 10 (2011), 1521–1530.