Mesh Denoising with Facet Graph Convolutions

Matthieu Armando, Jean-Sébastien Franco, Edmond Boyer

To cite this version:

Matthieu Armando, Jean-Sébastien Franco, Edmond Boyer. Mesh Denoising with Facet Graph Convolutions. IEEE Transactions on Visualization and Computer Graphics, 2022, 28 (8), pp.2999-3012. 10.1109/TVCG.2020.3045490 . hal-03066322

HAL Id: hal-03066322
https://inria.hal.science/hal-03066322

Submitted on 15 Dec 2020

Mesh Denoising with Facet Graph Convolutions

Matthieu Armando, Jean-Sébastien Franco, and Edmond Boyer

Abstract—We examine the problem of mesh denoising, which consists of removing noise from corrupted 3D meshes while preserving existing geometric features. Most mesh denoising methods require a lot of mesh-specific parameter fine-tuning, to account for specific features and noise types. In recent years, data-driven methods have demonstrated their robustness and effectiveness with respect to noise and feature properties on a wide variety of geometry and image problems. Most existing mesh denoising methods still use hand-crafted features, and locally denoise facets rather than examine the mesh globally. In this work, we propose the use of a fully end-to-end learning strategy based on graph convolutions, where meaningful features are learned directly by our network. It operates on a graph of facets, directly on the existing topology of the mesh, without resampling, and follows a multi-scale design to extract geometric features at different resolution levels. Similar to most recent pipelines, given a noisy mesh, we first denoise face normals with our novel approach, then update vertex positions accordingly. Our method performs significantly better than the current state-of-the-art learning-based methods. Additionally, we show that it can be trained on noisy data, without explicit correspondence between noisy and ground-truth facets. We also propose a multi-scale denoising strategy, better suited to correct noise with a low spatial frequency.

Index Terms—Mesh denoising, normal filtering, graph convolution, feature preserving, geometric deep learning

1 Introduction

Polygon meshes are extensively used to represent geometry models of 3D shapes. This applies in particular to real shapes that can be digitized into 3D meshes using capture devices and reconstruction methods. The mesh models resulting from such acquisition processes are perturbed by noise originating from various sources, including capture sensor imprecision and numerical issues. *Mesh denoising* aims at correcting or reducing such noise perturbations on 3D mesh models.

Mesh denoising is, in essence, an ill-posed problem since differentiating noise from the original geometric features requires prior knowledge on the noise, the shape, or both. A common strategy in that respect is to assume known distributions, typically Gaussian noise or smooth shapes, with nevertheless severe limitations. Chosen distributions are only a coarse approximation of the true distributions, and they are usually hand-picked depending on the application. As a result, they do not generalize well. Actually, in most capture scenarios, it proves difficult to provide hand-crafted prior models for noise or shape that cover a reasonable part of the spectrum of possible distributions. Consequently, data-driven strategies for mesh denoising have gained interest over the last decade, boosted by the success of deep learning in various domains, in particular image denoising *e.g*. FFDNet [1]. Related approaches learn distributions from training examples and have already shown promising results with meshes, as in [2], [3]. For instance, in [2], a neural network is trained to denoise mesh normals. For that purpose, hand-crafted features are pre-computed for each face and fed into the network individually. While not end to end, the results obtained demonstrate the ability to learn local denoising patterns. Our work follows this line of research, with the objective to further exploit learning methods, and propose a fully end-to-end solution. We want to build a network that can learn relevant features over a large receptive field, and retain connectivity information throughout the network, so as to ensure spatial consistency.

Convolutional neural networks excel at learning spatially-varying features at different scales, with a limited number of parameters, notably through the use of pooling and unpooling layers. For these reasons, CNNs have proven very successful in most image processing tasks, and image denoising is no exception. For the recent super-resolution challenge NTIRE2017 [4], the top three methods all include convolutional layers. The ability of CNNs to model complex features at different scales is obviously useful for denoising. Our goal is to check if this holds for geometric data as well

A significant challenge is that traditional CNNs are restricted to regular grid structures, such as images. Several works have tried to circumvent this limitation, and extend convolutional layers to graph-like structures, such that graph convolutional networks (GCNs) has become a whole new field of research. We build on FeaStNet [5] which exhibits two key characteristics for our problem. First it generalizes convolution layers of standard CNNs to graphs in a natural way. Second, it allows to express pooling and unpooling layers over graphs, a key feature which we use to increase the receptive field of our network.

We contribute therefore with an end-to-end learning framework for mesh denoising. Our network considers the graph of faces of a mesh and uses the layer defined in [5], but for vertex graphs, as a building block, with an architecture, pooling strategy and graph connectivity that is adapted to the mesh denoising problem. Such a learning framework can exploit spatial organization as an additional feature with respect to recent works that consider spatial distance or patch similarity. This strategy proves to be successful and outperforms the current state of the art on the benchmark of [2]. Moreover, it opens research opportunities to denoise

- *M. Armando, J.-S. Franco and E. Boyer are with Univ. Grenoble Alpes, Inria, CNRS, Grenoble INP, LJK, 38000 Grenoble, France.*
  E-mails: firstname.name@inria.fr
- *M. Armando is funded by the MSR-Inria Joint Center.*

graph signals other than normals on meshes.

The remaining of the article is as follows. Section 3 gives an overview of our approach, which is then detailed in sections 4 and 5. Section 6 describes our implementation choices as well as two extensions to our baseline method. Finally, section 7 is dedicated to different experiments, both on synthetic and real data.

2 Related Work

We give here a general review of previous works on mesh denoising. This section does not aim to be exhaustive, given the prolific nature of the field, but merely tries to present the general trends, through seminal works and influencial papers. These categories are not mutually exclusive.

2.1 Early works

Early mesh denoising methods were isotropic, based on laplacian smoothing [6], [7], [8]. They smooth sharp features as well as noise. As such, they are *fairing* rather than *denoising* methods. In an attempt to discriminate between high-frequency noise and sharp features, subsequent methods introduce anisotropic filtering usually inspired by image denoising techniques, such as bilateral filtering [9], or scale space and anisotropic diffusion [10], [11].

2.2 Normal filtering methods

A more recent trend has seen the emergence of a two-steps framework, first introduced by Taubin in 2001 [12]. First, some local, non-linear denoising filter is applied to the facets' normals rather than the vertex positions. Then, as a second stage, vertex positions are updated according to the filtered face normals. This is usually repeated in an iterative manner. It yields better results than applying similar filters directly to the vertex positions. Such methods can be characterised by three distinct parts:

- **Normal filtering**, which is the central component: Yagou *et al*. suggest mean and median filtering [13] or alpha-trimming filtering [14]. Most methods are based on bilateral filtering [15] or some of its derivatives, such as joint bilateral filtering [16]. Different approaches include that of Yadav *et al*. [17], that uses a element-based normal voting tensor: it filters the eigenvalues of a local covariance matrix on each facet. In [18], they propose a similarity function that is more robust to outliers.
- **Vertex updating**: This step is less critical and more consensual than normal filtering. Sun *et al*. give a thorough review on the subject in [19]. Most methods follow the original proposition of Taubin [12], or some derivative: it sets a constraint of orthogonality between the estimated normal of each face and its adjacent edges, and proposes to solve it in the least squares sense via a gradient descent optimization.
- **Neighborhood choice**: Some methods follow a more combinatorial approach, *i.e.* they select neighbours based on local connectivity (as in [19]). Others use a simple spatial kernel, *i.e.* they select neighboring facets and weigh their contribution based on euclidian distance in 3D (and normals). As Liu *et al*. [20] point out, this can lead to cross-region mixing. Instead of weighing contributions based on spatial distance and signal difference, they consider all values of the signal of interest along the geodesic path.

These are good general-purpose methods that work reasonably well for most surfaces and random noise distributions. However, for a given application, their parameters need to be carefully tuned, to strike the good balance between noise removal and features preservation. In particular, this includes at least the number of iterations, which can only be set through trial and error.

We follow the same general framework, but our model returns denoised normals in a single regression step. Thus, we avoid the usual trade-off between removing noise and preserving original features: optimal parameters are learned directly from training data.

2.3 Global optimization

Other recent works follow a global optimization approach, like L_0-minimization, introduced by He *et al*. on vertex positions [21], and improved by Zhao *et al*. to include face normals information as well [22]. Instead of processing faces (or vertices) independently, these methods try to compute a global solution that minimizes some energy term on the whole mesh, based on some prior on surfaces. They usually rely on the assumption that real surfaces are made of smooth regions punctuated by sparse sharp features. These examples work well for CAD-like models, but they are less successful on meshes with dense features. Besides, they are quite slow to compute. Instead, we propose to learn a prior from training data.

2.4 Spectral methods

Spectral methods have been used for a wide range of mesh processing applications. Zhang *et al*. provide a thorough survey on the subject [23].

Early denoising methods based on Laplacian filtering [6] fall into this category, even though they can be carried out in the spatial domain via convolutions. Some subsequent methods require the explicit computation of eigenvectors, which comes at a high computational cost. To remedy this problem, most approaches partition a given surface into smaller patches that are processed separately (*e.g.* [24]). More recent works include TSGSP [25], a two-stage approach, also applied on a per-patch basis. They estimate the noise level for a given patch, and perform low-pass spectral filtering on it. Then, they use some sort of guided normal filtering.

Generally speaking, spectral methods have trouble recovering sharp edges, which yield harmonics of many different frequencies [26].

2.5 Nonlocal similarity methods

Other works rely on the assumptions that similar patches can be found on a real surface, and their aim is to make use of this redundancy of information. Like most mesh denoising approaches, these are inspired by successful image processing concepts. For example, Yoshizawa *et al*. [27] extend the *Non-Local means* concept of Buades *et al*. [28] to geometry processing. More recently, Wei *et al*. [29] or Li *et al*. [30] co-filter similar patches using low-rank matrix recovery.

While we do not explicitly consider non-local similarity in our method, because of the large receptive field of our network, it could theoretically leverage such redundancy.

2.6 Data-driven methods

Data-driven methods, that try to learn from examples, have been gaining increasing popularity for mesh denoising. An early work in this category [3] formulates the whole mesh denoising problem in a Bayesian way, with a generative model of the noisy surface. The prior on surface shapes is expressed as a potential between normals of adjacent faces. Different potential functions are compared and, interestingly, the prior parameters are determined through supervised learning. This is a first step towards application-specific denoising techniques, without the cumbersome hand-tweaking of parameters. However, the prior is constrained with a limited number of parameters and while the shape prior is learned, the noise model parameters are still hand-picked.

More recently, Wang *et al*. train neural networks to denoise facets' normals [2]. Hand-crafted local geometry descriptors called FND for *filtered facet normal descriptor* are taken as inputs. They are based on the bilateral and joint bilateral filters. The approach uses then single-hidden layer feed forward networks (SLFN) in a cascaded way. The method is fast and effective, yet still far from end-to-end learning: (1) The descriptors fed to the network are hand-crafted and computationally expensive; (2) It divides input data into clusters and trains separate networks for each one; (3) It applies the cluster-based regression in a cascaded way and the vertex-updating steps taking place in-between regression steps are highly constrained (and hand-crafted). This results in additional hyperparameters that need to be set manually. Furthermore, we believe that features learned specifically for the task at hand might be more effective than FNDs at conveying relevant information. [31] use a similar design with iterative per-face learning using FNDs, but with a two-steps framework, where the second normal estimation is supposed to recover features lost during the first step.

Also [32] propose a learning framework based on a non-local similarity approach: Patch vectors based on a similarity criterion are grouped and fed into a convolution network. In contrast, our convolutions have a spatial support, and can extract meaningful local features at different scales.

Finally, [33] propose a CNN-based denoising technique, *NormalNet*. For each face, the normals of neighbouring facets are projected into a locally-defined voxel grid and 3D convolutions are performed in this new regular structure, in order to regress refined normals. This approach is based on a cascaded structure similar to that of [2], alternating normals regression and vertex updating. Relevant features are actually learned by the network. However, each normal regression is performed in a separate locally-defined space. Thus, the potential of convolutional networks for spatial consistency is not fully exploited. Besides, the proposed solution seems to rely on a computationally heavy preprocessing step per face whereas our network operates directly on the whole mesh, and does not resort to some local space transform and resampling.

3 Method Overview

We consider a surface $\mathcal{S}$ described by a mesh $\mathcal{M} = \{V, \mathcal{E}\}$, where V is the set of vertices and $\mathcal{E}$ the set of topological edges connecting them. Suppose that a noisy observation $\hat{\mathcal{M}} = \{\hat{V}, \hat{\mathcal{E}}\}$ is available. We assume that $\hat{V}$ is obtained through a generative process $\hat{V} = V + \mathcal{N}$ where $\mathcal{N}$ is the acquisition noise. In this work we do not address topological noise and, hence, we assume that the observed topology and connectivity is correct, *i.e.* $\hat{\mathcal{E}} = \mathcal{E}$.

In order to denoise a mesh, a common practice among the most efficient methods is to first denoise the mesh normals before updating the vertex positions accordingly, hence benefiting from the scale invariance of the normals. We adopt the same strategy and consider the face normals as they proved more efficient than vertex locations in our experiments. Following Wang *et al*. [2] we assume that local noise patterns can be learned from examples in training datasets. However, differently from [2], we investigate the ability to learn directly from the normal information, without relying on intermediate handcrafted descriptors. To this purpose, we build on image denoising techniques with CNNs and extend them to meshes and their associated irregular graphs using a graph convolutional network (GCN) approach [5].

We therefore train a network to regress faces normals, given a graph of faces with noisy positions and noisy normals. We implement a multi-scale architecture, with pooling and unpooling layers [5], that allows for noise patterns at different scales.

In a final step, we update the vertex locations given the corrected normals. Without loss of generality, we make use of the differentiable solution presented in [19] that iteratively optimizes vertex locations so that the faces they define are orthogonal to the predicted normals (see [19] for details).

To sum up, as depicted in Figure 1, our method takes as input a noisy mesh $\{\hat{V}, \mathcal{E}, \hat{N}\}$, where $\hat{N}$ are the noisy normals, and outputs a denoised mesh $\{\tilde{V}, \mathcal{E}, \tilde{N}\}$ with a vertex updating scheme based on normal predictions $\tilde{N}$. Such denoised normal predictions are obtained with multi-scale graph convolutions applied on face normals over the mesh. Details on the network architecture, its training and the evaluation follow.

4 Neural Network

This section describes the neural network architecture of our approach. First, we give a general view of the architecture and then describe the convolution layers in more details. The method used for pooling operations is detailed in section 5.2.

4.1 Architecture

Our graph convolution network builds on the popular U-net architecture [34], originally introduced for image segmentation. U-net takes as input a signal defined over

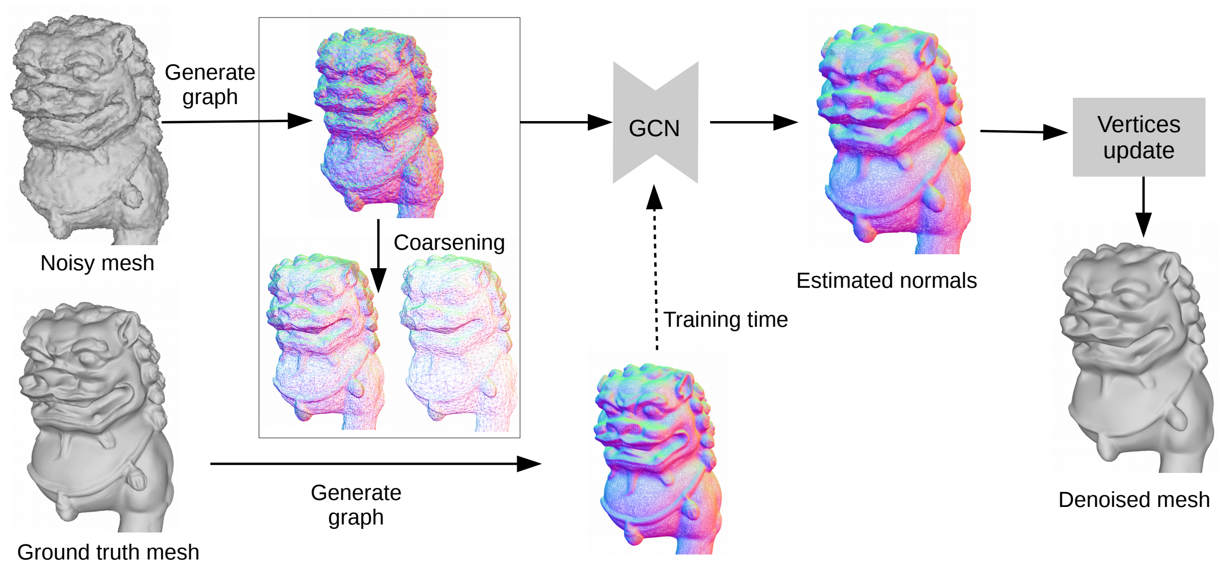


Fig. 1. Full pipeline. Given an noisy mesh, we build a graph of faces with normal information (sec. 5.1) and precompute coarser representations for pooling (sec. 5.2). This is fed into our GCN (4) that is trained to regress the denoised normal for each face. During inference, the estimated normals are used to update the vertex positions through an iterative process (sec. 6.3).

a 2D domain and essentially consists of two consecutive subnetworks: (i) First a contracting path, which is a succession of convolution and pooling layers, that can extract global context from the signal, but loses small scale features on the way; (ii) Then an expanding path, roughly symmetrical, consisting of up-convolutions and convolutions, where the final output has the same spatial size as the input. By up-convolutions, it is meant upsampling operation, followed by a convolution that halves the number of channels ([34]).

A key property of the architecture lies in the so-called skip-connections: features from the contracting path are concatenated to corresponding features in the expanding path. This way, small-scale features are not lost through the successive pooling operations, and the final output depends on both large-scale context and small-scale features. Besides image segmentation, this design has already proven successfull in image denoising [35]. We investigate therefore a similar multi-scale architecture with our approach (see Figure 2). U-net is primarily designed for regular image grids, we thus need to adapt it to the irregular grid structure of meshes. This implies redefining local convolutions as well as the pooling and unpooling operations.

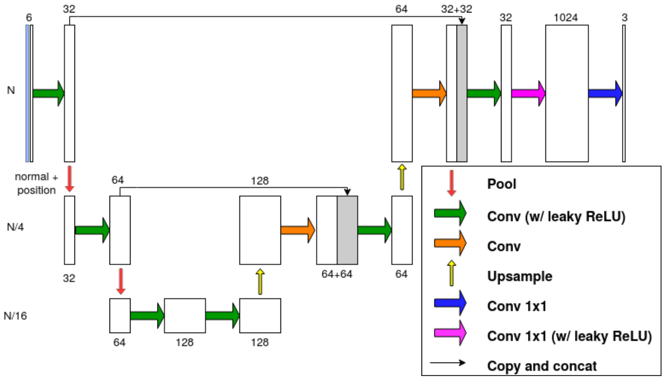


Fig. 2. The network architecture: Inputs are 6D vectors composed of normal and position information and outputs are 3D corrected normals; 3 different scales are taken into account.

Geometric deep learning has been actively researched recently. Several works have proposed solutions to extend regular convolutional layers to irregular graphs by applying filters directly to the manifold surface (*e.g.* [36], [37]). We choose to exploit the FeaStNet graph convolutional layer of [5], originally introduced for mesh registration. Contrary to the approaches cited above, it does not require a resampling of the original data. Besides, it can be naturally integrated in a multi-scale architecture. Even though it was used in a classification context only, and with a different input, it can be adapted to our regression task given the proper loss. Thus, we perform multi-scale convolutions on decimated coarser graphs.

4.2 Convolutional layers

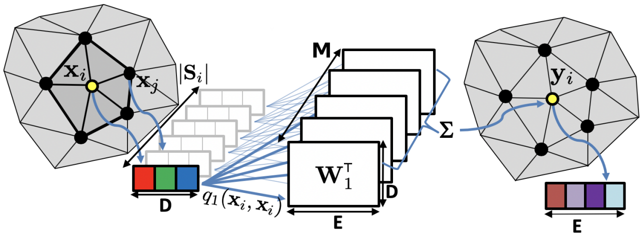


Fig. 3. The graph convolutional layer from [5], which we adapt to faces instead of vertices.

A convolutional layer within a neural network takes as input a signal $\mathbf{x}$ of dimension D and outputs a feature signal $\mathbf{y}$ of dimension E that is the result of local convolutions of

$\mathbf{x}$ by some filter with weights to be learned. With a regular 2D grid, the local support of the filter is a neighborhood on the grid, typically 8 pixels around a central pixel in an image, and the convolution boils down to local weighted sums of the input signal multiplied by some feature transformation matrices. Such a local support being constant, filter weights can be shared over nodes within the grid, hence drastically reducing the number of parameters to be learned. Over an arbitrary mesh, this property does not hold since neighborhoods differ from one vertex to another. In order to enable shared convolution operators over graphs, Verma *et al.* [5] suggest assigning a weighted sum of a fixed number M of feature transformations to each node inside the support region, where the assignment is a function of $\mathbf{x}$ which parameters are learned by the network, along with the transformation weights. They formulate the GCN convolutional layer as (see figure 3):

$$\mathbf{y}_i = \mathbf{b} + \sum_{m=1}^{M} \frac{1}{|\mathcal{S}_i|} \sum_{j \in \mathcal{S}_i} q_m(\mathbf{x}_i, \mathbf{x}_j) \mathbf{W}_m \mathbf{x}_j, \tag{1}$$

where $\mathbf{b}$ is a bias term, $\mathcal{S}_i$ the support region of $\mathbf{x}_i$ on $\mathcal{M}$, $\mathbf{W}_m$ the $E \times D$ weight matrix of the m^{th} feature transformation and $q_m(\mathbf{x}_i, \mathbf{x}_j)$ is the assignment function of that transformation:

$$q_m(\mathbf{x}_i, \mathbf{x}_j) \propto \exp(\mathbf{u}_m^\top \mathbf{x}_i + \mathbf{v}_m^\top \mathbf{x}_j + \mathbf{c}_m), \tag{2}$$

with $\mathbf{u}_m$, $\mathbf{v}_m$ and $\mathbf{c}_m$ the parameters to be learned in addition to the transformation weights ($\mathbf{W}$, $\mathbf{b}$). See [5] for more details. Finally, the number of learned parameters for each convolution is MDE for weights $\mathbf{W}$, E for the bias $\mathbf{b}$, MD for the assignment variables $\mathbf{u}$ and $\mathbf{v}$, and M for $\mathbf{c}$. In total, this yields $M(D+1)$ assignment weights. Such convolution is applied at each node in the input graph and the outputs $\mathbf{y}_i$ obtained over the mesh are used to feed the next convolutional layer in the network.

In this work, we use a similar GCN formulation, but with noticeable differences: (1) the architectural design presented in section (4.1) is more similar to the original design of U-Net, with the up-convolution pattern. (2) Our network operates on a different input graph with a higher connectivity (section 5.1) and a different input signal (normals + position). (3) In contrast to the classification problem of FeaStNet with a cross-entropy loss, we tackle a regression problem. (Losses used are detailed in section 6). (4) Graph coarsening is based on spatial and normal proximity, contrary to the random coarsening used in FeaStNet (see section 5.2), and we perform two iterations of the coarsening algorithm per pooling layer (section 6), in order to increase the receptive field of the network.

5 Data Representation

The previous section presents the neural network architecture we use to process information defined over a graph. We discuss in this section how to apply it in our specific mesh denoising context. In particular we precise the input graph we consider as well as the multi-scale strategy with meshes.

5.1 Input Graph

The noisy meshes we want to denoise come with a natural graph structure that is their vertex connectivities. While a reasonable choice when the input signal is defined at the vertices, *e.g.* the vertex locations, this graph appears less adapted with the normal information we want to process that is defined at the face level. A simple solution would be to merge the face normals around a vertex, but this would loose local information. Thus, we cannot operate directly on the graph of vertices like FeaStNet. Other graph structures with mesh faces as nodes can be considered, for instance (see figure 4):

- The dual representation of the mesh: each face is connected to exactly 3 neighboring faces, with which it shares an edge with, *i.e.* each node has degree 3.
- An extended dual representation where each face is connected to all faces in its 1-ring vertex neighborhood, *i.e.* all the faces it shares a vertex with.

We adopt the mentioned extended dual representation that increases the receptive field of our neural network. A higher degree at each node in the graph favors quicker propagation of the signal through convolutions. In practice, in our experiments, each face has more than 10 neighbors on average in this graph representation.

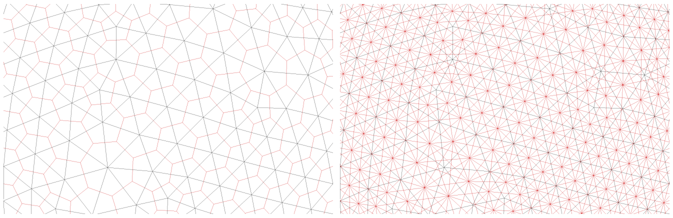


Fig. 4. Input face graphs in red (the vertex mesh appears in black): (left) The dual representation of the vertex mesh with valence 3; (right) The extended graph where each face is connected to its 1-ring neighborhood resulting in a much denser representation.

5.2 Multi-scale representation

As stated previously, receptive fields of different sizes enable the neural network to capture contexts at different scales and, hence, to learn denoising patterns at different scales as well. By reducing the sampling frequency of the signal, pooling is a good way of enlarging the receptive field, without for that matter increasing the complexity with a larger depth of the network. Besides, pooling adds implicit spatial regularization by sharing features between neighbouring points. However, while downsampling a regular grid is a straightforward operation, the case of meshes with arbitrary graph structures appears more challenging.

To address this issue and build a multi-scale mesh representation, we choose the graph coarsening solution of Defferrard *et al.* [38]. It is a greedy multi-level coarsening technique with, at its core, the max-cut coarsening strategy used by the Graclus clustering algorithm [39]. At each coarsening level, graph nodes are grouped into pairs, except for a few remaining singletons. To this aim, Graclus iteratively picks an unmarked node x, and pairs it with its unmarked neighbor y that maximizes:

$$\frac{e(x,y)}{d_x} + \frac{e(x,y)}{d_y}, \tag{3}$$

where $e(x,y)$ is the edge weight between nodes x and y, and d_x is the degree of node x, *i.e.* the sum of the edge weights between x and its neighbors. The process is iterated until all nodes have been visited. Then, at each level, Defferrard *et al.* introduce fake nodes to be paired with the singletons, in order to form a binary tree where each node has two children in the next finer level. Finally, nodes are reordered so that the tree structure is implicitly encoded in the indexing of the nodes. This makes pooling and unpooling operations as simple and efficient as for a regular 1D signal.

For neighboring faces x and y with normals n_x, n_y and barycenter positions c_x, c_y, we set the edge weight as:

$$e(x,y) = max(n_x.n_y, \epsilon) \times \exp - \frac{||c_x - c_y||^2}{2 \times l_e^2} \tag{4}$$

where l_e is the average edge length in the graph. Intuitively, this weight favors close faces with similar normals to be grouped together. ϵ is set close to zero, and ensures that neighboring faces stay connected even around an extreme bend. For subsequent coarsening operations, we sum the edge weights of all edges grouped together by pairing operations. Figure 5 illustrates the mesh coarsening approach. This weighting differs from FeaStNet, that uses constant edge weights, which makes the coarsening operations purely random.

6 Implementation

In this section we provide implementation details. In addition to network and training settings, we precise the vertex updating scheme which re-estimates a mesh given corrected face normals as predicted by our network. In relation to this step, we also present two extensions we investigated: First the ability to train with unregistered data by using a loss function, based on a mesh to mesh distance, which applies to the meshes obtained after the vertex updating step; Second a multi-scale approach for vertex updating that corrects vertex positions in a coarse to fine manner using intermediate normal outputs of the network at different scales.

6.1 Network Setting

The network takes as input 6D vectors composed of face barycenter positions and face normals and outputs 3D face normals. Adding the face position gives better results in our experiments (see the supplemental for input comparisons). The following implementation choices were made for all the experiments reported in this document:

- We choose leaky ReLU [40] as the activation function throughout the network since it demonstrated better convergence properties than ReLU.
- In contrast to the original U-net architecture [34], we use only 3 different levels or scales, *i.e.* 2 pooling layers, and we perform only 1, instead of 2, convolutions at a time in-between other layers. The motivation for these changes is to reduce the network complexity while keeping the same general design.
- Contrary to FeaStNet, we perform two coarsening steps between two consecutive levels, *i.e.* the number of nodes is approximately divided by 4 in-between levels. (See figure 5).
- For all convolution layers, we set the number of filters (see equation 1) to $M = 9$. We perform max pooling on the features for all pooling layers, which provides similar results to average pooling but accelerates back-propagation in practice.

6.2 Training

The network is trained with a L1 loss on the angular difference between the ground-truth facet normals, and the estimated normals, without regularization, using Adam optimization [41]. All meshes are centered on the origin for data normalization, and scaled so that the diagonal of the bounding box is set to unit length. Each noisy face makes up a single training example for our network (along with its noisy neighbourhood and ground-truth normal). Thus, meshes in the training set form natural batches for training. For meshes with less than 100k faces, we use them as training batches. For meshes with more than 100k we generate several patches, of 100k faces each, that we grow from a random seed facet, until every face is present in at least one patch. We use these patches as training batches. In both cases, we only perform back-propagation on 10k faces randomly sampled, which is hence our real batch size. The whole patch is still needed however, since other faces appear in the convolutions.

Our network is not intrinsically invariant with respect to rigid transformations of the input data. We choose to let the network learn this invariance from the data, and encourage this through data augmentation by applying at each training step a random rotation to the input mesh or patch. In our experiments, it proved better than to have this invariance built into the model, with equal performance and faster training.

6.3 Vertex updating

In order to update the vertex positions $\mathbf{x}$ on $\mathcal{M}$ given the corrected normals $\tilde{\mathbf{n}}$, we follow the iterative approach of Xianfang *et al.* [19]. This approach iteratively moves vertices in order to make the mesh edges as orthogonal as possible to the estimated normals of the edge neighboring faces. First, let us define ∂F_k as the set of edges that constitute the boundary of face k, and $F_v(i)$ as the set of faces that share vertex i. The chosen strategy optimizes:

$$\underset{\mathbf{x}}{min}\, E(\mathbf{x}, \tilde{\mathbf{n}}) = \sum_{k \in F} \sum_{(i,j) \in \partial F_k} (\tilde{\mathbf{n}}_k \cdot (\mathbf{x}_j - \mathbf{x}_i))^2, \tag{5}$$

with a gradient descent. At each iteration, it updates the position $\mathbf{x}_i$ of each vertex with the new position $\tilde{\mathbf{x}_i}$ given by:

$$\tilde{\mathbf{x}}_i = \mathbf{x}_i + \frac{1}{|F_v(i)|} \sum_{k \in F_v(i)} \tilde{\mathbf{n}}_k (\tilde{\mathbf{n}}_k \cdot (\mathbf{c}_k - \mathbf{x}_i)), \tag{6}$$

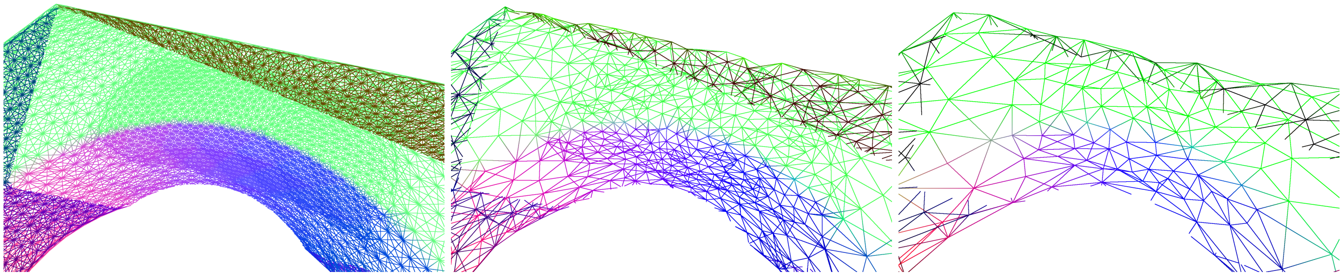


Fig. 5. Graph coarsening example. The input graph (left) is shown with every node (facet) colored according to the input normal orientation. Subsequent images show the graph at coarser levels. For illustration purposes, normals and positions are here propagated to coarser levels with average pooling. In reality, these quantities are not retained beyond the first layer of the network, and thus, they are not defined for coarser levels of the graph.

where $\mathbf{c}_k$ is the barycenter of face k, and $\tilde{\mathbf{n}}_k$ is the estimated normal for face k. We perform 60 such iterations in our experiments. This number was chosen empirically as increasing it further has no visible effect for most use cases.

6.4 Learning from unregistered data

In order to train with datasets that do not provide exact associations between ground-truth and noisy normals, we propose a specific training scheme. The interest arises with real datasets, *e.g.* the Kinect dataset in [2], for which the correspondences between noisy and unnoisy meshes can only be estimated. Our scheme integrates the vertex updating step into the back-propagation and defines a mesh to mesh distance loss that applies directly on the vertex positions. This loss combines an accuracy term $\mathcal{L}_{acc}$ and a completeness term $\mathcal{L}_{comp}$. They are respectively the average distance from points of $\tilde{V}$ to V, and the average distance from points of V to $\tilde{V}$:

$$\mathcal{L}_{acc} = \frac{1}{|\tilde{V}|} \sum_{\tilde{\mathbf{x}}_i \in \tilde{V}} \min_{\mathbf{x}_j \in V} (\|\tilde{\mathbf{x}}_i - \mathbf{x}_j\|), \tag{7}$$

$$\mathcal{L}_{comp} = \frac{1}{|V|} \sum_{\mathbf{x}_i \in V} \min_{\tilde{\mathbf{x}}_j \in \tilde{V}} (\|\mathbf{x}_i - \tilde{\mathbf{x}}_j\|). \tag{8}$$

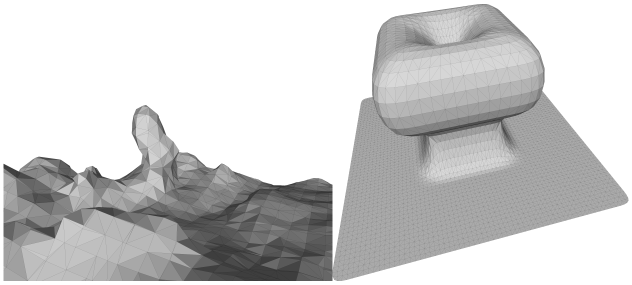
Note that this formulation of a global loss over meshes is made possible since our network considers complete meshes as input, and not individual facets with precomputed descriptors. It allows the network to converge to possibly better associations on the training data than the provided estimated ones. Moreover, our results on the Kinect datasets have shown that the vertex updating step is imperfect. On some occasions it actually increases the angular error on normals. By integrating this loss in the training, we allow the network to optimize its predictions with respect to the final output mesh rather than the predicted normals. We validate this strategy on the synthetic dataset of [2] in section 7.4.

6.5 Multi-scale vertex updating

The vertex updating method presented in section 6.3 implicitly assumes independent and zero mean noise at each vertex. This is not always true with real data. In particular when performing Poisson reconstructions from noisy point clouds, large artifacts can appear. Because space is resampled during the Poisson reconstruction, vertices are not raw independent measurements. In fact, the sampling frequency in the final mesh can be significantly higher than that of the captured point cloud. Thus, a single noisy measurement can lead to an artifact spanning several triangles. Figure 6 shows examples of such artifacts. Applying the vertex updating scheme on these examples results in very slow convergence since the artifact scale is significantly larger than the receptive field (*i.e.* the first vertex ring on the mesh) of the approach. Consequently, it takes a considerable number of iterations to converge to a planar result, even when the estimated normals are correct (see figure 7). This impacts the training too, if we include this step into the back-propagation as proposed in section 6.4.

To address this issue, we propose a multi-scale vertex updating scheme that allows for faster convergence. We generalize equation 6 to work in a multi-scale fashion and apply it successively at different scales, from coarsest to finest. For a given coarsening level p and face f_k, we note $\mathbf{c}_k^p$ the barycenter of graph node C that contains f_k, (at the finer level, $C = f_k$. Otherwise, C is a set of faces pooled together):

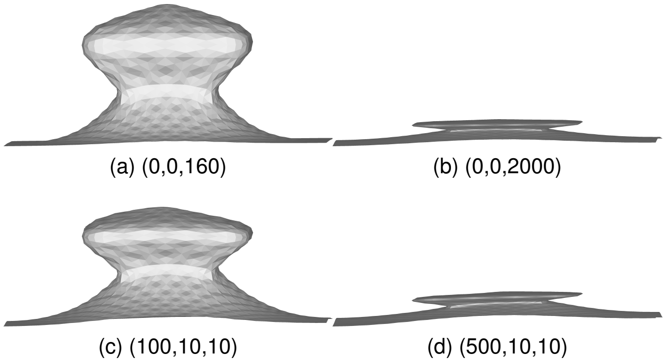
$$\text{if } f_k \in C, \mathbf{c}_k^p = \frac{1}{|C|} \sum_{f_j \in C} \mathbf{c}_j. \tag{9}$$



(a) Real-life example

(b) Toy example

Fig. 6. Left: A real-life example of reconstruction artifact: close-up of a reconstruction of a planar surface from the DTU dataset [42]. Right: Our toy example for testing, where an originally square planar mesh presents an unwanted bump that spans a large number of faces. we purposefully emphasize the problem to test the limits of the vertex-updating framework.



(a) (0,0,160)

(b) (0,0,2000)

(c) (100,10,10)

(d) (500,10,10)

Fig. 7. Toy example correction with constant estimated normals facing up. The algorithm should ultimately converge to a flat plane with folds. **Top** Single scale: (a): after 160 iterations. (b): after 2000 iterations **Bottom** Multi-scale: (c): after (100,10,10) iterations (from coarsest level to finest). (d): after (500,10,10) iterations.

For a given scale p, each vertex updating step is given by:

$$\tilde{\mathbf{x}}_i = \mathbf{x}_i + \frac{1}{|F_v(i)|} \sum_{f_k \in F_v(i)} \tilde{\mathbf{n}}_k^p (\tilde{\mathbf{n}}_k^p . (\mathbf{c}_k^p - \mathbf{x}_i)). \qquad (10)$$

This is in essence similar to the previous scheme (equation 6) with however a larger neighborhood around each vertex. Figure 7 demonstrates the faster convergence of our method on the toy example. For the choice of $\tilde{\mathbf{n}}_k^p$, the face normals in the cluster could be averaged. We choose instead to trust the network ability to better behave than an averaging filter and we train it to predict normals at each coarsening level during the expanding path, with the cost defined in section 6.4. Please note that barycenters defined in equation 9 are only used in equation 10 and they do not flow through the network (see figure 5).

7 Experiments

7.1 Evaluation Strategy

In order to evaluate the benefit of our end-to-end learning architecture, we first compare to current state-of-the-art learning-based approaches for mesh denoising which are the *Cascaded Normal Regression* (CNR) method of [2] and *NormalF-Net* [32]. Comparisons on synthetic and real data are presented in sections 7.2 and 7.3 respectively. We follow the experimental setup of CNR [2] as the authors provided all the necessary data for that purpose. They perform four different experiments, on four datasets (one synthetic dataset, and three obtained from Kinect scans). The authors of *NormalF-Net* provided us with all their results on these datasets.

As a baseline evalutation, we also report results from other parametric methods when available. In particular, results for *Non-Local Low-Rank Normal Filtering* (NLLR) [30] were computed using the executable file released by the authors. We try four sets of parameters everytime $((\sigma_M, v_{iter}, N_k) \in \{(0.25, 7, 7), (0.39, 7, 7), (0.39, 10, 10), (0.6, 10, 10)\})$ and keep the best results only.

In addition, we evaluate the benefit of the extensions presented in section 6 by comparing them with our baseline approach (7.4). Finally, we also test the generalization ability of our network in a practical real scenario with MultiView Stereo data (7.5).

7.1.0.1 Metric.: For the numerical evaluations, we consider the average angular difference between denoised normals and ground truth normals. This metric can be computed using either the raw output normals estimated by the network, or the refined normals of the denoised meshes as obtained after a vertex location optimization. We believe that the raw normals should be used for the evaluation since it is the critical part for the denoising problem and also where our main contribution lies. The vertex updating step is a process that can alter the raw measurement, as will be shown in our experiments. However, we note that [2] and [32] while also using the angular difference with respect to ground truth normals, consider the refined normals after the vertex updating step. This seems legitimate in their case since they follow an iterative scheme where vertex updating is an intrisic part of the method. Therefore, we also use refined normals to guarantee a fair comparison.

7.2 Comparison on Synthetic Data

7.2.1 Dataset

We first validate our method on the publicly available synthetic dataset of [2]. It is composed of 50 meshes divided into 3 categories: CAD-like models with flat areas and angular features, smooth models with low frequency features and complex models with multi-scale features. 21 are used for training, and 29 for testing. For each mesh, three noisy versions are provided, obtained by adding Gaussian noise with different standard deviations to the vertex positions.

7.2.2 Results

In addition to the learning methods mentioned above, we add results obtained with *Bilateral Mesh Denoising* [9], *Bilateral Normal Filtering* (BNF) [15], *Guided Mesh Normal Filtering* (GMNF) [16] , *L_0 Minimization* [21] and the *bayesian method* [3], all provided by the authors of [2]. In each case, we only show results for the best set of parameters tested by [2]. We found that the vertex updating step plays a significant role in smoothing out some of the remaining noise in this evaluation (see supplementary). Figure 8 shows quantitative results. Even before the regularization provided by the vertex updating step, our approach performs better on average than all the others and clearly outperforms them in all data categories after that step. Figure 15 shows qualitative results on the test set. Additional comparisons, including results on other synthetic data, also appear in the supplementary material.

Finally, we also compare our appraoch to the spectral method TSGSP [25], since it demonstrates competitive results with respect to CNR on this test set. Numerical results are shown in table 1: Our method outperforms all the others by a fair margin on average, which validates the learning framework we propose.

7.2.3 Runtime experiments

We performed runtime experiments on a desktop computer with a 2.40GHz Intel(R) Xeon(R) CPU E5-2630 v3, 32GB of memory, and a NVIDIA Titan XP GPU. This configuration is chosen to be as close as possible to the experimental

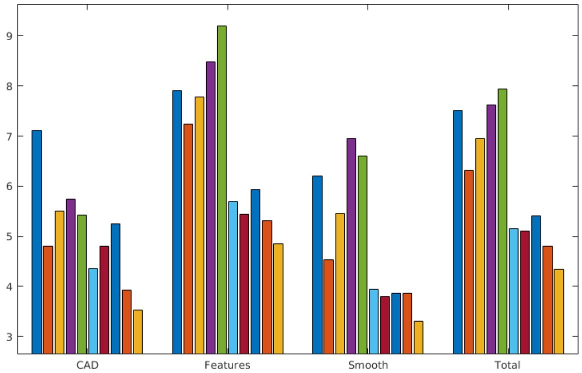


Fig. 8. Average angular error in degrees, on the synthetic benchmark dataset of [2], per category. From left to right: 🟦 BMD [9], 🟧 BNF [15], 🟨 GMNF [16], [21] 🟪 L_0 *Minimization*, 🟩 *bayesian method* [3], 🟦 CNR [2], 🟥 NormalF-Net [32], 🟦 NLLR [30], 🟧 Ours (raw normals estimated by our network), 🟨 Ours (final result).

TABLE 1
Average angular error (in degrees) over some test meshes of the synthetic dataset of [2] with intermediate noise level. Values for TSGSP and CNR are taken from [25].

Name of Model	TSGSP [25]	CNR [2]	Ours
Block	**2.3414**	2.3436	2.3890
Bumpy torus	3.9688	4.0464	**3.1715**
Bunny hi	5.3268	5.1152	**5.0205**
Carter100K	6.8722	7.8415	**5.6258**
Child	6.2145	6.9801	**5.3627**
Chinese lion	7.1285	7.6701	**6.2854**
Cube	**0.7747**	0.8656	0.9172
Eight	6.0995	**5.8017**	6.0118
Eros100K	8.0866	8.3486	**7.2148**
Fertility	3.8456	3.6379	**3.1902**
Genus3	2.4576	2.5751	**1.8895**
Joint	**1.6837**	1.6970	1.8098
Kitten	2.8386	2.8195	**2.4857**
Nicolo	4.5729	4.4868	**4.0940**
Part Lp	2.5046	2.5422	**2.3340**
Plane sphere	1.3816	1.2606	**1.2193**
Pulley	4.7630	4.5899	**3.7931**
Pyramid	**0.9446**	0.9912	1.1349
Rolling stage	4.5187	4.1767	**3.4522**
Screwdriver	3.7645	**2.9652**	3.1492
Smooth feature	**0.9847**	1.0085	1.0418
Sphere	2.5285	2.3076	**2.0047**
Star	1.5895	1.6502	**1.3793**
Trim star	6.4181	**4.1866**	4.7672
Turbine Lp	3.7025	2.7707	**2.5397**
Average over faces	4.7989	4.7945	**4.0840**

setup of NormalF-Net [32], for a meaningful comparison. Their experiments are run on a PC with a CPU of the same generation (2.2GHz Intel Xeon E5-2650), 64GB RAM and a NVIDIA GTX-1080Ti. The computation time for different test meshes with various sizes are given in table 2, and results from [32] for other learning methods are also reported. In our case, the preprocessing step is performed on the CPU only, and could certainly be accelerated with a proper parallel implementation. The most time consuming parts are the coarsening of the graph and, for large meshes only, the input subdivision into separate mesh patches that are processed separately. The inference step by our network, including the vertex updating step, is run on the GPU.

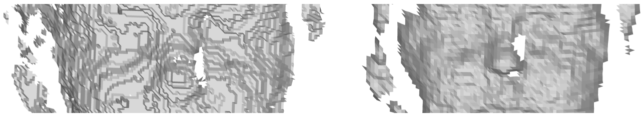


Fig. 9. Close-ups of meshes from the *Kinect v1* (left) and *Kinect v2* (right) datasets [2] showing holes and disjoint parts, as common in both datasets.

Our approach is slower than CNR [2] on small meshes, though this might be partly due to some overhead cost, given that our inference time scales really well with mesh sizes on the test data. Nevertheless, an efficient GPU implementation for the preprocessing step will be required to be competitive with CNR in terms of running time.
On the other hand, our method is significantly faster than NormalF-Net [32], and several orders of magnitude faster than NormalNet [33], where a local support is computed for each facet independently. This validates our motivation for using graph convolutions on the mesh.

TABLE 2
Running time of our method (in seconds) on some test meshes of the synthetic dataset of [2]. Results for competitors are taken from [32] and obtained with a PC that presents slightly different specifications, though minor enough to legitimate orders of magnitude comparisons.

Model	SharpSphere	Fertility	Grayloc	Eros	Gargoyle
Faces	20882	27954	68580	100000	171112
Ours					
Preprocessing	16	25	54	82	149
Inference	7	8	9	10	11
Total runtime	24	33	63	91	160
Competitors					
CNR [2]	1	2	3	5	10
NormalF-Net [32]	97	110	345	481	975
NormalNet [33]	836	1132	5418	9163	20763

7.3 Comparison on Real Data

7.3.1 Datasets

CNR [2] also provides three Kinect datasets, obtained respectively from *Microsoft Kinect v1* scans, *Microsoft Kinect v2* scans, and reconstructions of *Microsoft Kinect v1* scans, using KinectFusion [43]. We use a similar experimental setup on these datasets, with two major differences regarding the *Kinect v1* and *v2* experiments.

First, we note that meshes in these datasets suffer from topological noise, which violates our central premise. They present numerous holes and disjoint parts, as shown in figure 9. Since our convolutional layers are based on local connectivity, this means each disjoint part of a given mesh will be processed independently by our network. This is one limit of our approach. To deal with such data, filters based on spatial distance (as in CNR) or patch similarity (as in NormalF-Net) can be better equipped than filters based on local connectivity. Nevertheless, in order to improve the performance of our network, we add a new – binary – channel to our input, that differentiate between faces that lie on a border of the mesh, and faces that do not. This results in faster convergence during training, and slightly improves our results.

Second, since meshes in the *Kinect v1* and *v2* datasets are obtained from depthmaps, we constrain vertices to move only along the depth direction in this case. These two changes are not applied to the *KinectFusion* dataset.

7.3.2 Results

Numerical results are shown in figure 10, and qualitative results in figure 11. We perform on par with or better than competitors on all the Kinect datasets. Interestingly, we notice that the angular error is actually increased by the vertex updating step for the *Kinect-Fusion* dataset. This is presumably due to the very specific sampling of those meshes, with many thin triangles. The reader can refer to the supplemental material for more illustrations. This supports the argument made before that comparisons before refinement steps are more pertinent. Besides, this provides an extra motivation for the vertex loss we propose in 6.4.

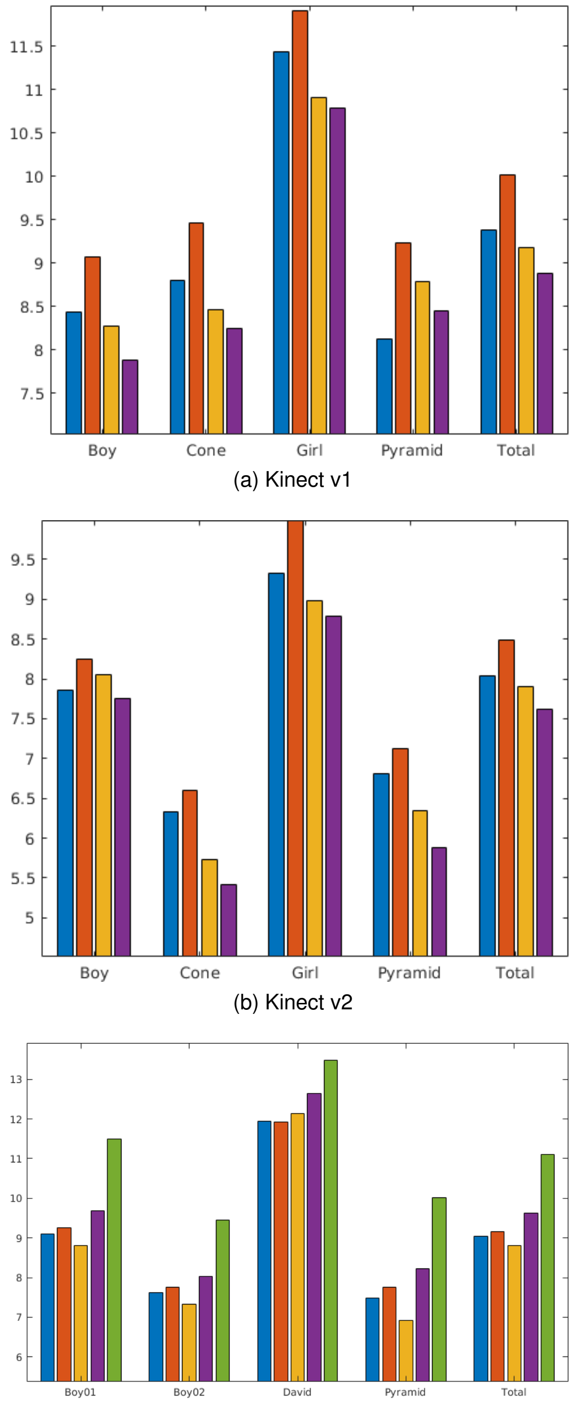
7.4 Method Extensions

In this section, we present results obtained using the 2 extensions exposed in sections 6.4 and 6.5 and compared to our standard approach. For the multi-scale vertex updating, the iteration numbers are set to $(80, 20, 20)$ from coarsest to finest (see figure 7). Figure 12 shows numerical results on the synthetic dataset of [2]. It shows that the extensions benefits to the vertex location estimation but not to the normal estimation, which can be expected since the extensions apply a loss on vertices.

Figure 13 shows qualitative results. The approach without extensions yields smoother results that are visually pleasing, however it tends to lose more small scale features from the original mesh. Figure 14 illustrates the contribution of each scale to the final output: The coarsest normals smooth out noise and small scale features. Subsequent steps better reconstruct those features. Note that the orientation of estimated normals has no influence on the vertex updating step in equation 10. Depending on the initialization of the network weights, normals can therefore be flipped. This happens in this example, for the intermediate level.

7.5 Generalisation to Other Data

The ability of our learned model to generalize to unseen data is a primary concern, in particular with real data as produced by digitalization apparatus. This appears challenging for a model trained on synthetic data only, that are intrinsically less diverse than real data. To evaluate the generalization capability of our method, we tested it on surfaces obtained by multi-view stereo reconstructions from RGB images, using the reconstruction method of [44]. Such surfaces exhibit various noise types, such as missing concavities, holes, topological noise, flipped faces, among other acquisition imperfections. While correcting all of them is beyond the scope of this work, the question that arises is whether our framework can improve the reconstruction results by exploiting the learned local noise and shape patterns.



(a) Kinect v1

(b) Kinect v2

(c) Kinect Fusion

Fig. 10. Average angular errors in degrees, on the 3 Kinect datasets of [2], per scanned model. 🟦 CNR. 🟥 NormalF-Net. 🟨 Ours (raw estimated normals). 🟪 Ours (refined normals). 🟩 NLLR.

Figure 16 shows qualitative results of the model trained on the synthetic dataset (from 7.2). We compare our method to CNR [2] and HC laplacian smoothing [7] that form both a general purpose baseline. Compared to this baseline, our model appears to better preserve the recovered features, *e.g.* the shirt folds in the back, and to better filters out random noise. On the other hand, CNR produce a smoother results, removing some large noise patterns, however losing some features along the way, *e.g.* the jawline, fingers, or shirt folds.

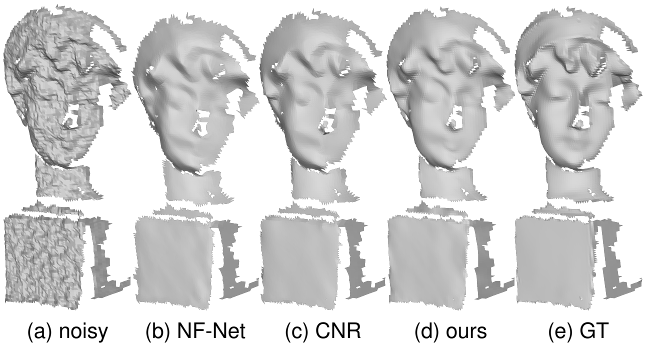
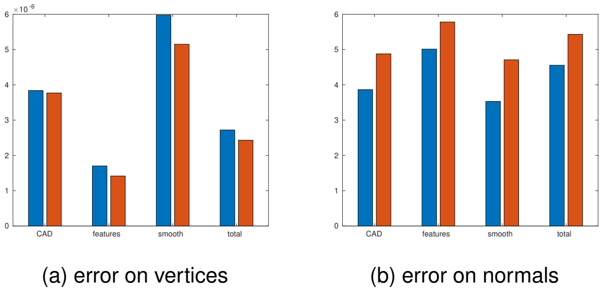

(a) noisy (b) NF-Net (c) CNR (d) ours (e) GT

Fig. 11. An example from the *Kinect v2* dataset of [2]. NF-Net stands for NormalF-Net



(a) error on vertices (b) error on normals

Fig. 12. Extension (Sec. 6.4 and 6.5) evaluation: Comparison of the average error on the test set from the synthetic dataset of [2]. **Blue**: Network trained with our standard approach. **Red**: Network trained using both extensions. (a): Average distance of each vertex of the denoised mesh to the closest vertex of the ground-truth, normalized by the diagonal length of the mesh. (b): Average angular error on facet normals.

8 Conclusion

In this work, we have presented a novel end-to-end learning approach for normal denoising on a mesh surface. It demonstrates that a graph convolutional network architecture can learn meaningful features with respect to local shape and noise patterns and, thanks to its convolutional nature, it can also learn spatial consistency without the need for explicit constraints. As a result, the approach presents better results when compared to the state of the art methods for mesh denoising. We have also investigated 2 extensions of this framework. First, building on the observation that the vertex updating step is fully differentiable, we have proposed a new learning framework that can use unregistered noisy data. Second, within this framework, we have extended the standard vertex-updating scheme to work in a multi-scale manner, so as to alleviate its constraint on the vertex displacement. These extensions provide additional constraints on the vertex locations that can help recovering small features on a noisy mesh, this in comparison to a training loss based on local orientations of the surface, as without the extensions. An interesting direction to explore here is how to benefit from both modalities, orientation and location, through the training loss.

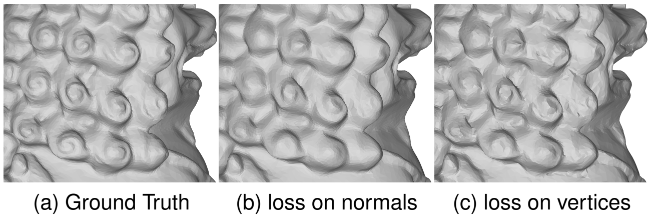
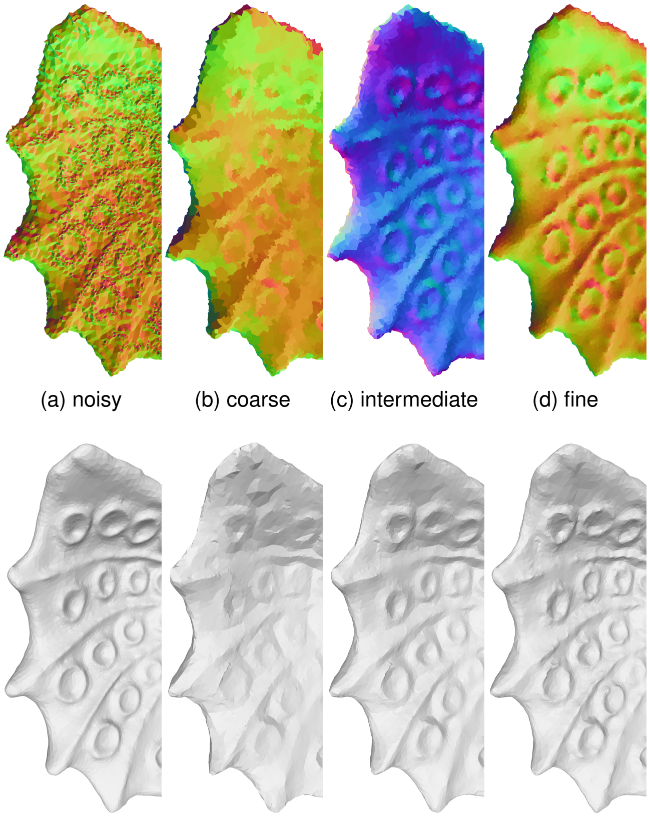

(a) Ground Truth (b) loss on normals (c) loss on vertices

Fig. 13. Extension evaluation: Close-ups of the ”chinese lion” mesh from the synthetic dataset of [2]. (b): Network trained with our standard approach. (c): Network trained using the 2 extensions described in section 6.


(a) noisy (b) coarse (c) intermediate (d) fine

(e) Ground Truth (f) 1st step (g) 2nd step (h) final output

Fig. 14. Multi-scale vertex updating extension: Close-ups of the ”gargoyle” mesh. Top row: Normals estimated by the network at different resolution levels (in (c) normals are flipped by the network, see text for detail). Bottom row: Mesh obtained after the contribution on each set of normals.

One of the limitation of the framework we propose is that it does not handle topological noise, assuming the connectivity of the input mesh to be correct. While this assumption is valid in many cases, there are situations where spatial relation can be more meaningful than local connectivity (*e.g.* with incorrectly disjoint components). This is true with, for instance, the *Kinect* datasets, as shown in our results. As many recent learning-based methods have been proposed for point cloud denoising, it would be interesting to investigate how they could contribute in our framework by handling purely spatial information.

On the application side, the interest of a learning-based strategy over traditional hand-crafted methods is its capacity to better model complex types of real noise. One typical example being the multi-view stereo reconstruction pipeline, from capture to reconstruction. In the future, we plan to investigate if our method is able to improve a

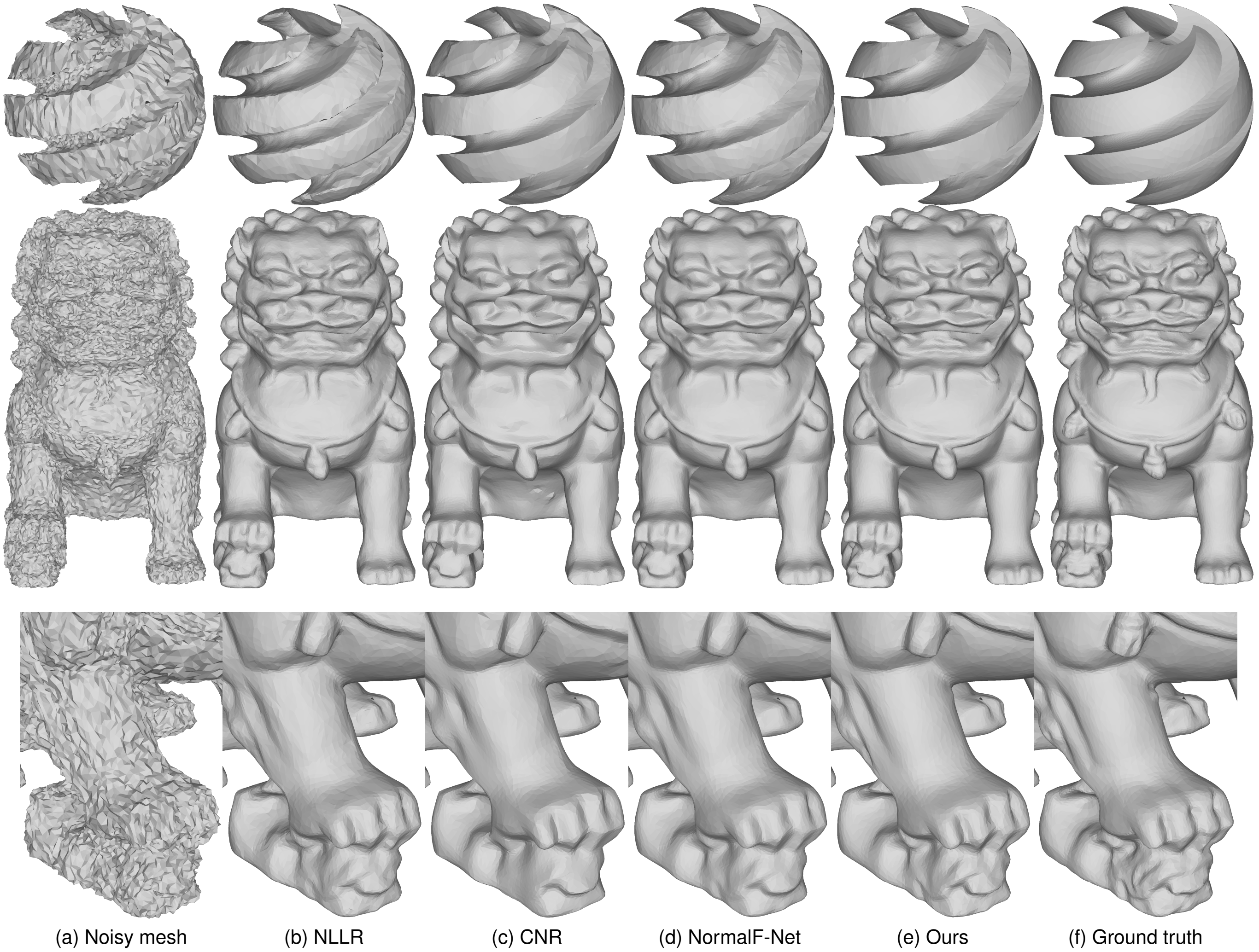


Fig. 15. Qualitative results from Wang *et al*. synthetic database, with the highest noise level. The last row shows close-up views of the *chinese lion* model. Note how sharp and complex features are handled by the different methods.

given reconstruction method by learning its intrinsic noise distribution. Finally, our approach is not fully end-to-end since it still requires an additional vertex updating step. Whether this additional step can be integrated in the network architecture is certainly a future work.

References

[1] K. Zhang, W. Zuo, and L. Zhang, "FFDNet: Toward a fast and flexible solution for CNN-Based image denoising," *IEEE Transactions on Image Processing*, vol. 27, no. 9, pp. 4608–4622, 2018.

[2] P.-S. Wang, Y. Liu, and X. Tong, "Mesh denoising via cascaded normal regression," *ACM Transactions on Graphics*, vol. 35, no. 6, pp. 1–12, 2016. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2980179.2980232

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

[4] R. Timofte, E. Agustsson, L. Gool, M.-H. Yang, L. Zhang, B. Lim, S. Son, H. Kim, S. Nah, K. Lee, X. Wang, Y. Tian, K. Yu, Y. Zhang, S. Wu, C. Dong, L. Lin, Y. Qiao, C. Loy, W. Bae, J. Yoo, Y. Han, J. Ye, J.-S. Choi, M. Kim, Y. Fan, J. Yu, W. Han, D. Liu, H. Yu, Z. Wang, H. Shi, X. Wang, T. Huang, Y. Chen, K. Zhang, W. Zuo, Z. Tang, L. Luo, S. Li, M. Fu, L. Cao, W. Heng, G. Bui, T. Le, Y. Duan, D. Tao, R. Wang, X. Lin, J. Pang, J. Xu, Y. Zhao, X. Xu, J. Pan, D. Sun, Y. Zhang, X. Song, Y. Dai, X. Qin, X.-P. Huynh, T. Guo, H. Mousavi, T. Vu, V. Monga, C. Cruz, K. Egiazarian, V. Katkovnik, R. Mehta, A. Jain, A. Agarwalla, C. Praveen, R. Zhou, H. Wen, C. Zhu, Z. Xia, Z. Wang, and Q. Guo, "NTIRE 2017 Challenge on Single Image Super-Resolution: Methods and Results," *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, vol. 2017-July, 2017.

[5] N. Verma, E. Boyer, and J. Verbeek, "FeaStNet: Feature-Steered Graph Convolutions for 3D Shape Analysis," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. IEEE, jun 2018, pp. 2598–2606. [Online]. Available: https://ieeexplore.ieee.org/document/8578373/

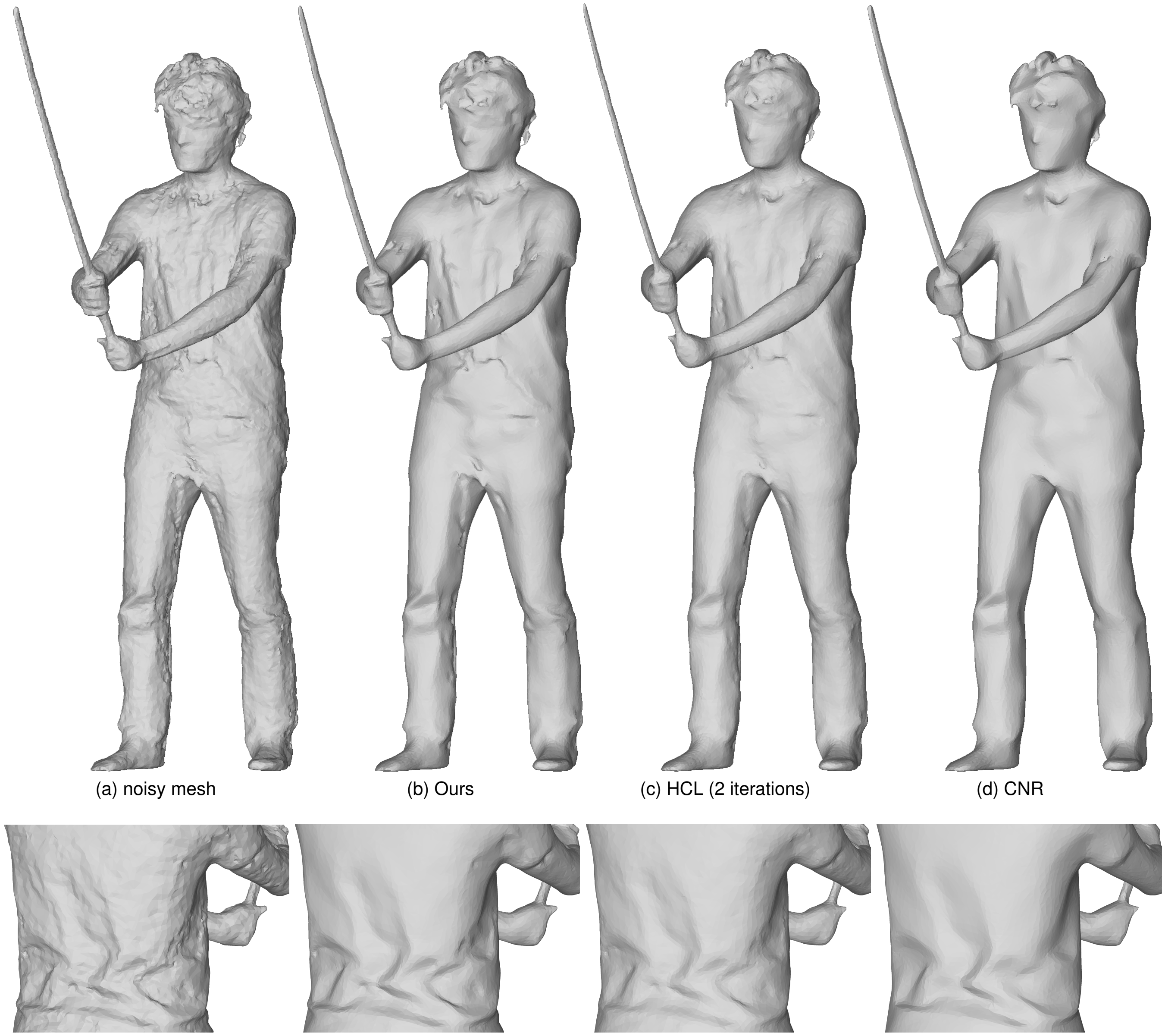
[6] G. Taubin, "A signal processing approach to fair surface design," *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques - SIGGRAPH '95*, pp. 351–358, 1995. [Online]. Available: http://portal.acm.org/citation.cfm?doid=218380.218473

[7] J. Vollmer, R. Mencl, and H. Muller, "Improved Laplacian Smoothing of Noisy Surface Meshes," *Computer Graphics Forum*, vol. 18, no. 3, pp. 131–138, 1999. [Online]. Available: http://doi.wiley.com/10.1111/1467-8659.00334

[8] M. Desbrun, M. Meyer, P. Schröder, and A. H. Barr, "Implicit fairing of irregular meshes using diffusion and curvature flow," in *Proceedings of the 26th annual conference on Computer graphics and interactive techniques - SIGGRAPH '99*. New York, New York, USA: ACM Press, 1999, pp. 317–324. [Online]. Available: http://portal.acm.org/citation.cfm?doid=311535.311576

[9] S. Fleishman, I. Drori, and D. Cohen-Or, "Bilateral mesh denoising," in *ACM Transactions on Graphics*, vol. 22, no. 3. New York, New York, USA: ACM Press, 2003, p. 950. [Online]. Available: http://portal.acm.org/citation.cfm?doid=882262.882368

[10] U. Clarenz, U. Diewald, and M. Rumpf, "Anisotropic geometric



(a) noisy mesh

(b) Ours

(c) HCL (2 iterations)

(d) CNR

(e) same comparison, zoomed-in back view

Fig. 16. Qualitative results on a mesh captured from a real scene, using the multi-view Kinovis platform [45] and a MVS approach [44]. Our network trained on synthetic data tends to better preserve sharp or large scale features than a traditional HC laplacian smoothing (HCL) or than CNR [2] trained on the same data. This can be seen for instance on the person's facial features or fingers or on the folds in the shirt. Best viewed digitally.

diffusion in surface processing," in *Proceedings Visualization 2000. VIS 2000 (Cat. No.00CH37145)*. IEEE, 2002, pp. 397–405,. [Online]. Available: http://ieeexplore.ieee.org/document/885721/

[11] T. Tasdizen, R. Whitaker, P. Burchard, and S. Osher, "Geometric surface smoothing via anisotropic diffusion of normals," *Proceedings of the IEEE Visualization Conference*, pp. 125–132, 2002.

[12] G. Taubin and G. Taubin, "Linear anisotropic mesh filtering," in *IBM Research Report RC22213(W0110-051), IBM T.J. Watson Research*, 2001.

[13] H. Yagou, Y. Ohtake, and A. Belyaev, "Mesh smoothing via mean and median filtering applied to face normals," *Proceedings - Geometric Modeling and Processing: Theory and Applications, GMP 2002*, pp. 124–131, 2002.

[14] H. Yagou, Y. Ohtake, and A. G. Belyaev, "Mesh denoising via iterative alpha-trimming and nonlinear diffusion of normals with automatic thresholding," *Proceedings of Computer Graphics International Conference, CGI*, vol. 2003-Janua, pp. 28–33, 2003.

[15] Y. Zheng, H. Fu, O. K. C. Au, and C. L. Tai, "Bilateral normal filtering for mesh denoising," *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, no. 10, pp. 1521–1530, 2011.

[16] W. Zhang, B. Deng, J. Zhang, S. Bouaziz, and L. Liu, "Guided Mesh Normal Filtering," *Computer Graphics Forum*, vol. 34, no. 7, pp. 23–34, 2015.

[17] S. K. Yadav, U. Reitebuch, and K. Polthier, "Mesh Denoising Based on Normal Voting Tensor and Binary Optimization," *IEEE Transactions on Visualization and Computer Graphics*, vol. 24, no. 8, pp. 2366–2379, 2018.

[18] ——, "Robust and High Fidelity Mesh Denoising," *IEEE Transactions on Visualization and Computer Graphics*, vol. 2626, no. c, pp. 1–9, 2018.

[19] S. Xianfang, P. L. Rosin, R. R. Martin, and F. C. Langbein, "Fast and effective feature-preserving mesh denoising," *IEEE Transactions on Visualization and Computer Graphics*, vol. 13, no. 5, pp. 925–938, 2007.

[20] B. Liu, J. Cao, W. Wang, N. Ma, B. Li, L. Liu, and X. Liu, "Propagated mesh normal filtering," *Computers and Graphics (Pergamon)*, vol. 74, pp. 119–125, 2018.

[21] L. He and S. Schaefer, "Mesh Denoising via L0 Minimization,"

ACM Trans. Graph., vol. 32, no. 4, pp. 64:1—-64:8, 2013. [Online]. Available: http://doi.acm.org/10.1145/2461912.2461965

[22] Y. Zhao, H. Qin, X. Zeng, J. Xu, and J. Dong, "Robust and effective mesh denoising using L0sparse regularization," *CAD Computer Aided Design*, vol. 101, pp. 82–97, 2018. [Online]. Available: https://doi.org/10.1016/j.cad.2018.04.001

[23] H. Zhang, O. Van Kaick, and R. Dyer, "Spectral Mesh Processing," *Computer Graphics Forum*, vol. 29, no. 6, pp. 1865–1894, sep 2010. [Online]. Available: http://doi.wiley.com/10.1111/j.1467-8659.2010.01655.x

[24] Z. Karni and C. Gotsman, "Spectral compression of mesh geometry," *Proceedings of the ACM SIGGRAPH Conference on Computer Graphics*, pp. 279–286, 2000.

[25] G. Arvanitis, A. S. Lalos, K. Moustakas, and N. Fakotakis, "Feature preserving mesh denoising based on graph spectral processing," *IEEE Transactions on Visualization and Computer Graphics*, vol. 25, no. 3, pp. 1513–1527, 2019.

[26] B. Vallet and B. Lévy, "Spectral Geometry Processing with Manifold Harmonics," *Computer Graphics Forum*, vol. 27, no. 2, pp. 251–260, apr 2008. [Online]. Available: http://doi.wiley.com/10.1111/j.1467-8659.2008.01122.x

[27] S. Yoshizawa, A. Belyaev, and H. P. Seidel, "Smoothing by example: Mesh denoising by averaging with similarity-based weights," *Proceedings - IEEE International Conference on Shape Modeling and Applications 2006, SMI 2006*, vol. 2006, p. 9, 2006.

[28] A. Buades, B. Coll, and J.-M. Morel, "A Non-Local Algorithm for Image Denoising," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 2, no. 0. IEEE, 2006, pp. 60–65. [Online]. Available: http://ieeexplore.ieee.org/document/1467423/

[29] 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 Transactions on Visualization and Computer Graphics*, vol. PP, no. c, p. 1, 2018.

[30] X. Li, L. Zhu, C. W. Fu, and P. A. Heng, "Non-Local Low-Rank Normal Filtering for Mesh Denoising," *Computer Graphics Forum*, vol. 37, no. 7, pp. 155–166, 2018.

[31] J. Wang, J. Huang, F. L. Wang, M. Wei, H. Xie, and J. Qin, "Data-driven Geometry-recovering Mesh Denoising," *CAD Computer Aided Design*, vol. 114, pp. 133–142, 2019. [Online]. Available: https://doi.org/10.1016/j.cad.2019.05.027

[32] 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," *Computer-Aided Design*, p. 102861, may 2020. [Online]. Available: https://doi.org/10.1016/j.cad.2020.102861 https://linkinghub.elsevier.com/retrieve/pii/S0010448520300543

[33] W. Zhao, X. Liu, Y. Zhao, X. Fan, and D. Zhao, "NormalNet: Learning-based Normal Filtering for Mesh Denoising," pp. 1–11, mar 2019. [Online]. Available: http://arxiv.org/abs/1903.04015

[34] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 9351, pp. 234–241, 2015.

[35] D. Liu, B. Wen, X. Liu, Z. Wang, and T. S. Huang, "When image denoising meets high-level vision tasks: A deep learning approach," *IJCAI International Joint Conference on Artificial Intelligence*, vol. 2018-July, pp. 842–848, 2018.

[36] J. Masci, D. Boscaini, M. M. Bronstein, and P. Vandergheynst, "Geodesic Convolutional Neural Networks on Riemannian Manifolds," *Proceedings of the IEEE International Conference on Computer Vision*, vol. 2015-Febru, pp. 832–840, 2015.

[37] D. Boscaini, J. Masci, E. Rodolà, and M. Bronstein, "Learning shape correspondence with anisotropic convolutional neural networks," in *Advances in Neural Information Processing Systems 29*, D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, Eds. Curran Associates, Inc., 2016, pp. 3189–3197. [Online]. Available: http://papers.nips.cc/paper/6045-learning-shape-correspondence-with-anisotropic-convolutional-neural-networks.pdf

[38] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Advances in Neural Information Processing Systems 29*, D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, Eds. Curran Associates, Inc., 2016, pp. 3844–3852. [Online]. Available: http://papers.nips.cc/paper/6081-convolutional-neural-networks-on-graphs-with-fast-localized-spectral-filtering.pdf

[39] I. S. Dhillon, Y. Guan, and B. Kulis, "Weighted graph cuts without eigenvectors a multilevel approach," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 11, pp. 1944–1957, 2007.

[40] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," *in ICML Workshop on Deep Learning for Audio, Speech and Language Processing*, vol. 28, 2013.

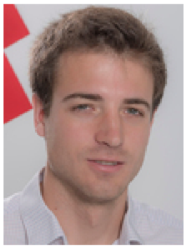
[41] D. P. Kingma and J. L. Ba, "Adam: A method for stochastic optimization," *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, pp. 1–15, 2015.

[42] H. Aanæs, R. Ramsbøl, J. George, V. Engin, T. Anders, and B. Dahl, "Large-Scale Data for Multiple-View Stereopsis," *IJCV*, 2016.

[43] S. Izadi, D. Kim, O. Hilliges, D. Molyneaux, R. Newcombe, P. Kohli, J. Shotton, S. Hodges, D. Freeman, A. Davison, and A. Fitzgibbon, "KinectFusion: Real-time 3D reconstruction and interaction using a moving depth camera," *UIST'11 - Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, pp. 559–568, 2011.

[44] V. Leroy, J.-S. Franco, and E. Boyer, "Shape reconstruction using volume sweeping and learned photoconsistency," in *The European Conference on Computer Vision (ECCV)*, September 2018.

[45] "Kinovis inria platform," https://kinovis.inria.fr/inria-platform/, accessed: 2020-02-26.



Matthieu Armando received the MSc degree in computer graphics, vision and imaging from the University College of London in 2013, and graduated from Supélec in 2014. He is working towards the PhD degree at INRIA Grenoble, within the Morpheo team. His interests include surface and appearance reconstruction from images and geometric deep learning.

Jean-Sébastien Franco is an associate professor of Computer Science at the Ensimag (School of Computer Science and Applied Mathematics, INP Grenoble University), and a researcher at the INRIA Grenoble Rhône-Alpes, France, with the Morpheo team since 2011.

He obtained his PhD from the Institut National Polytechnique de Grenoble in 2005 with the INRIA MOVI team. His expertise and interest is in the field of computer vision, dynamic 3D modeling and 4D spatio-temporal modeling from multiple views, 3D interaction.

Edmond Boyer is senior researcher at INRIA Grenoble Rhône-Alpes (France) where he leads the Morpheo research team on the capture and the analysis of moving shapes using visual cues. His fields of competence cover computer vision, computational geometry and virtual reality. His current research interests are on 3D dynamic modelling from images and videos, motion perception and analysis from videos, and immersive and interactive environments.

Edmond Boyer obtained a PhD in computer science from the Institut National Polytechnique de Lorraine in 1996. He started his professional career as a research assistant at the University of Cambridge (UK) in the Department of Engineering. He joined the INRIA Grenoble Rhône-Alpes in 1998. From 1998 to 2010, he was associate professor of computer science at the university Joseph Fourier part of Grenoble universities. Edmond Boyer is co-founder of the 4DViews company that is specialized in 4D acquisition.