Supplementary Material for DMD-Net: Deep Mesh Denoising Network

Aalok Gangopadhyay, Shashikant Verma, and Shanmuganathan Raman

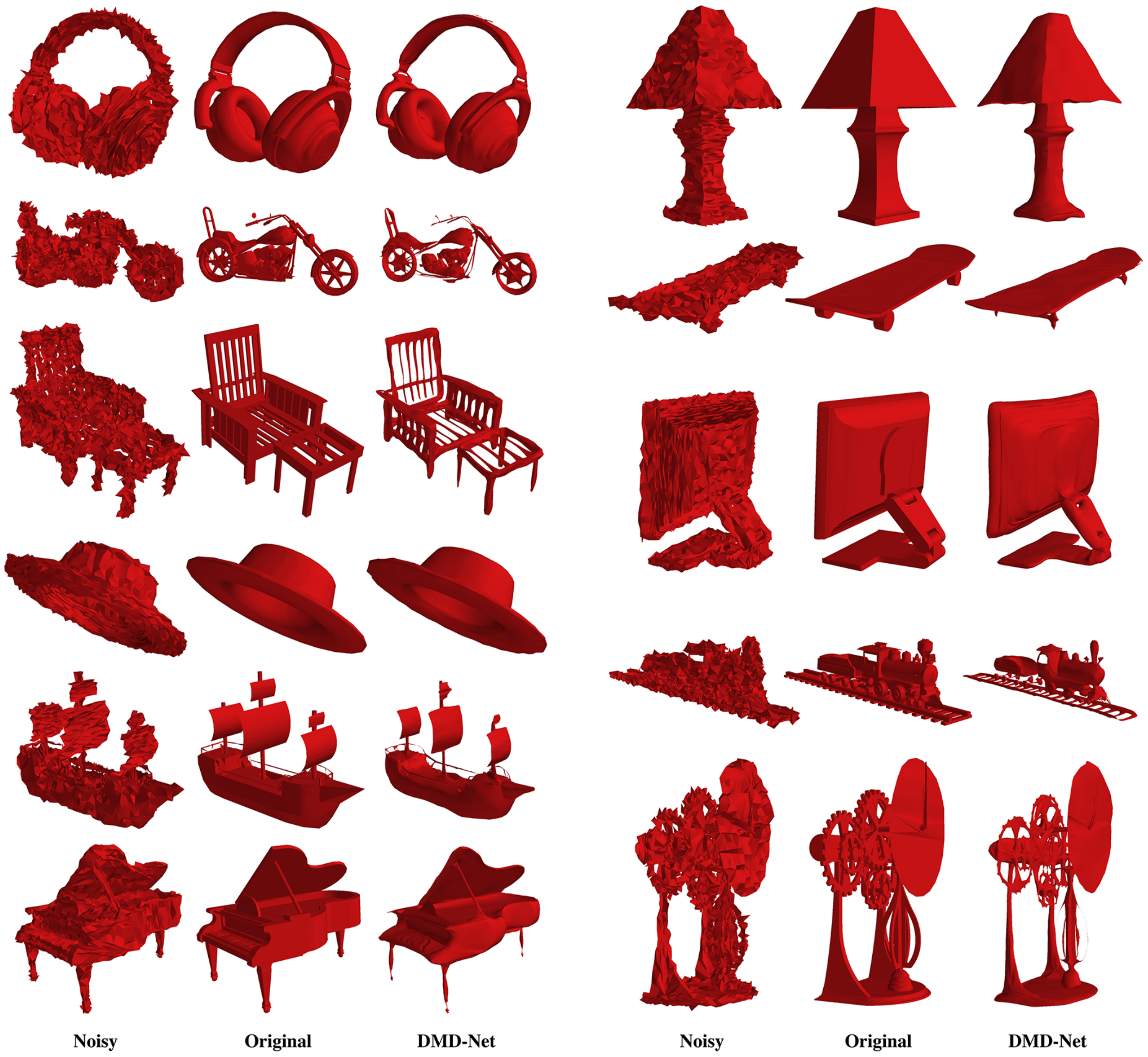

Fig. 1: Results of DMD-Net on several examples from the ShapeNet dataset.

I. DMD-Net Architecture

In this section, we provide a detailed explanation about the architecture of DMD-Net .

A. Feature Guided Transformer (FGT)

DMD-Net is based on the Feature Guided Transformer paradigm and consists of three main components: the feature extractor, the transformer, and the denoiser. The main idea behind FGT is as follows. Consider the scenario where an oracle gives us access to the local features of the original ground-truth mesh. With such an access, we could use these local features to guide the denoising process. But, in reality, we do not have such an access to the original mesh features, so we instead try to construct an artificial oracle, which we call the feature extractor network. We train the feature extractor to estimate normal vector, mean curvature, and Gaussian curvature for each vertex of the original mesh from the given noisy mesh.

These estimated local features serve as guidance for the transformer to compute a transformation matrix $\mathcal{W}_{tf}$. The noisy mesh is combined with the estimated local features through concatenation. The transformation $\mathcal{W}_{tf}$ is applied on this combination to obtain an intermediate representation. The noisy mesh and its transformed intermediate representation are both passed to the denoiser which generates the final denoised mesh. The detailed inner workings are described in the subsequent subsections.

B. Feature Extractor

The feature extractor internally contains a pair of two-stream networks which have two parallel streams, the upper one called the dual stream and the lower one called the primal stream.

C. Transformer

The objective of the transformer is to convert the noisy input mesh into an intermediate representation, which is easier to perform denoising on. For this, we first use the feature extractor to estimate the local features for each vertex, such as the normal vectors, the mean curvature and the Gaussian curvature which are of size $n\times 3$, $n\times 1$ and $n\times 1$, respectively. Thus, the combined size of the local features is $n\times 5$.

These estimated local features are then used by the transformer to compute the transformation $\mathcal{W}_{tf}$. In order to compute this transformation, the local features are passed through a composition of several different layers, such as fully connected layer with ReLU activation, a series of densely connected aggregators (AGG), a graph aggregation layer, and a feature average pooling layer. Note that the second fully connected layer is not accompanied by a ReLU activation. This is because the distribution of scalar entries in matrix $\mathcal{W}_{tf}$ should not be skewed towards positive values.

The input to the feature average pooling layer is a matrix of size $n\times 4096$. The output obtained after pooling is a 4096-dimensional vector which is reshaped to form the transformation matrix $\mathcal{W}_{tf}$ of size 8×512. On the other hand, the noisy input mesh is concatenated with the local features to obtain a combined input of size $n\times 8$. The transformation $\mathcal{W}_{tf}$ is then applied on this combined input to obtain an $n\times 512$ matrix, which is further passed through a graph aggregation layer along with ReLU activation. This is then processed by the denoiser to obtain the output denoised mesh.

D. Denoiser

The denoiser has a structure identical to that of the feature extractor. The only difference is the number of two-stream networks used and the number of hidden units in the last fully connected layer.

E. Two-Stream Network

The two-stream network is an asymmetric module consisting of two parallel streams, the lower one for performing aggregation in the primal graph and the upper one for performing aggregation in the dual graph. It consists of the primal-to-dual layer, a cascade of aggregator layers and a primal dual fusion layer.

F. Aggregator (AGG)

The Aggregator (AGG) performs graph aggregation by pooling in the features of the neighbouring nodes. The input to AGG is $\mathcal{X}$ (feature matrix) and $\mathcal{A}$ (adjacency matrix). The input graph to AGG can be in both forms, primal as well as dual. The output of AGG is given by $g(\mathcal{X},\mathcal{A}) = \sigma(\hat{\mathcal{D}}^{-\frac{1}{2}}\hat{\mathcal{A}}\hat{\mathcal{D}}^{-\frac{1}{2}}\mathcal{X}\mathcal{W})$. Here, σ is the ReLU activation function, $\mathcal{W}$ is the learnable weight matrix, $\hat{\mathcal{A}} = \mathcal{A}+\mathcal{I}$ ($\mathcal{I}$ being the identity matrix), and $\hat{\mathcal{D}}$ is the diagonal node degree matrix of $\hat{\mathcal{A}}$. In the two-stream network, the three AGG blocks are connected via residual skip-connections to avoid node feature collision (refer section II).

G. Primal Dual Fusion (PDF)

In Primal Dual Fusion, the input from both the streams are passed into the dual average pooling (DAP) layer, which intermixes the features from both the streams at the facet level. The PDF serves as a point of communication, allowing flow of information from one stream to the other.

H. Primal to Dual (P2D)

The primal-to-dual layer appears in the dual stream of the two-stream network, where its objective is to convert the primal graph features $\mathcal{X}_{\mathcal{V}}$ into the dual graph features $\mathcal{X}_{\mathcal{F}}$. The feature of each face is represented as the centroid of the features of the vertices constituting that face. Thus, we have $\mathcal{X}_{\mathcal{F}} = \mathcal{D}_{\mathcal{F}\mathcal{V}}^{-1}\mathcal{A}_{\mathcal{F}\mathcal{V}}\mathcal{X}_{\mathcal{V}}$, where, $\mathcal{A}_{\mathcal{F}\mathcal{V}} = \mathcal{A}_{\mathcal{V}\mathcal{F}}^{T}$ and $\mathcal{D}_{\mathcal{F}\mathcal{V}}$ is the degree matrix denoting the number of vertices belonging to each face. Since our mesh is triangulated, each face is a triangle and each diagonal entry in $\mathcal{D}_{\mathcal{F}\mathcal{V}}$ is 3. Therefore, we have the simplified expression $\mathcal{X}_{\mathcal{F}} = \frac{1}{3}\mathcal{A}_{\mathcal{F}\mathcal{V}}\mathcal{X}_{\mathcal{V}}$.

I. Dual to Primal (D2P)

The dual-to-primal layer is mainly used to convert the dual features $\mathcal{X}_{\mathcal{F}}$ into a primal form $\mathcal{X}_{\mathcal{V}}$. We obtain $\mathcal{X}_{\mathcal{V}}$ by pre-multiplying $\mathcal{X}_{\mathcal{F}}$ with the degree normalized vertex-face adjacency matrix $\hat{\mathcal{A}}_{\mathcal{V}\mathcal{F}}$. Here, $\hat{\mathcal{A}}_{\mathcal{V}\mathcal{F}} = \mathcal{D}_{\mathcal{V}\mathcal{F}}^{-1}\mathcal{A}_{\mathcal{V}\mathcal{F}}$, in which $\mathcal{A}_{\mathcal{V}\mathcal{F}}$ is the vertex-face adjacency matrix and $\mathcal{D}_{\mathcal{V}\mathcal{F}}$ is the degree matrix denoting the number of faces in which a particular vertex lies.

J. Dual Average Pooling (DAP)

Let $\mathcal{X}_{\mathcal{V}}$ and $\mathcal{X}_{\mathcal{F}}$ be the input to the Dual Average Pooling layer arriving via the primal stream and the dual stream, respectively. The sizes of $\mathcal{X}_{\mathcal{V}}$ and $\mathcal{X}_{\mathcal{F}}$ are $n \times k$ and $f \times k$, respectively. Let $\mathcal{F}$ denote the collection of faces of the primal graph. For each face in $\mathcal{F}$, we perform the following: gather the features of the three vertices belonging to that face from $\mathcal{X}_{\mathcal{V}}$ and gather the feature of that face from $\mathcal{X}_{\mathcal{F}}$. Let p_1, p_2, and p_3 denote the features of the three vertices and d denote the feature of the face. Note that, in general, d need not be the centroid of p_1, p_2, and p_3. We measure the distance of d from the three vertices in each of the k dimensions by evaluating the following three quantities: $l^a = |p_1 - d|$, $l^b = |p_2 - d|$, and $l^c = |p_3 - d|$, where $|\cdot|$ denotes the element-wise absolute value. l^a, l^b, and l^c are all k-dimensional vectors. We thus obtain $l_i = (l_i^a, l_i^b, l_i^c)$, the fused feature descriptor for i^{th} face. Collecting the fused descriptor for all the faces gives us $L = (l_1, l_2, \cdots, l_i, \cdots, l_f)$, a tensor with three axes having size $f \times 3 \times k$. In order to keep our network permutation-invariant, we devise the following strategy to pool the fused features. We pool by using average pooling across the second axis, which gives an output u of size $f \times k$. Let $L^a = (l_1^a, l_2^a, \cdots, l_f^a)$, $L^b = (l_1^b, l_2^b, \cdots, l_f^b)$, and $L^c = (l_1^c, l_2^c, \cdots, l_f^c)$, each of size $f \times k$. Then, $u = \frac{1}{3}\sum_{i \in \{a,b,c\}} L^i$ is the output of DAP. Figure 3(c) in main paper illustrates the pooling mechanism of DAP. In the figure, the outer tetrahedron with red nodes is the primal mesh, whereas the inner tetrahedron with blue nodes is the dual mesh. These meshes are outputs of the aggregation block, which separately processes the primal and the dual meshes. Hence, the blue nodes are not centroids of the red nodes, in general.

K. Feature Average Pooling (FAP)

The Feature Average Pooling layer is used inside the transformer to pool the features across all the vertices and output the transformation. Given $H = \{h_1, h_2, \cdots, h_i, \cdots, h_n\}$ as the input to the FAP layer, it outputs $z = \frac{1}{n}\sum_{i=1}^{n} h_i$.

II. Node Feature Collision

We explain in detail the concept of *node feature collision*, an undesirable artifact of the vanilla Graph-CNN network. We also discuss the strategy adopted by us to eliminate this problem.

Let $g(\mathcal{X}, \mathcal{A}) = \sigma(\hat{\mathcal{D}}^{-\frac{1}{2}}\hat{\mathcal{A}}\hat{\mathcal{D}}^{-\frac{1}{2}}\mathcal{X}\mathcal{W})$ denote the output of the aggregator layer, for the inputs $\mathcal{X}$ (node feature matrix) and $\mathcal{A}$ (adjacency matrix). In the transformer as well as the two-stream network, there are three AGG blocks used in series along with residual skip connections. We now justify the use of residual skip connections. Assume the case where we do not use these skip connections. Then the output would be $\mathcal{X}' = g(g(g(\mathcal{X}, \mathcal{A}), \mathcal{A}), \mathcal{A})$. The series of AGG blocks in this form encounters a peculiar issue in some cases. If there exist two adjacent vertices in the graph whose neighbourhood is exactly the same, or in other words, if two rows of $\hat{\mathcal{A}}$ are identical, then the two vertices would get mapped on to the same output point in the feature space by g. For instance, if the input is a tetrahedron, then in the normalized adjacency matrix, all the four vertices are adjacent to each other and all four have exactly the same neighbourhood (note that $\hat{\mathcal{A}}$ contains self loop for each vertex). Thus, all four points of the tetrahedron get mapped to the same output after aggregation. This effect is highly undesirable and we refer to it as *node feature collision*. We solve this problem by making use of residual skip connections from the input to the output. Thus, the modification now becomes: $h(\mathcal{X}, \mathcal{A}) = g(\mathcal{X}, \mathcal{A}) + \mathcal{X}$. With the introduction of the skip connections we thus have the output $\mathcal{X}' = h(h(h(\mathcal{X}, \mathcal{A}), \mathcal{A}), \mathcal{A})$.

III. Loss Functions

Let $\mathcal{G}_{gt} = (\mathcal{V}, \mathcal{E}, \mathcal{F}, \mathcal{P}_{gt})$, $\mathcal{G}_{noisy} = (\mathcal{V}, \mathcal{E}, \mathcal{F}, \mathcal{P}_{noisy})$, and $\mathcal{G}_{out} = (\mathcal{V}, \mathcal{E}, \mathcal{F}, \mathcal{P}_{out})$ denote the original ground truth mesh, the noisy mesh, and the denoised mesh obtained using DMD-Net, respectively. Let θ_{gt}, θ_{noisy}, and θ_{out} denote the interior angles of the vertices on their respective faces. If vertex v is contained in face u, then $\theta(v, u)$ denotes the interior angle of v on triangular face u. We define several loss functions whose objective is to make $\mathcal{G}_{out}$ as close to $\mathcal{G}_{gt}$ as possible.

A. Vertex Loss

The vertex loss computes the mean Euclidean distance between the corresponding vertices. It is defined in Equation 1, where, $\mathcal{P}_{out}(v)$ and $\mathcal{P}_{gt}(v)$ denote the feature vectors of the vertex v.

$$\mathcal{L}_{vertex} = \frac{1}{|\mathcal{V}|}\sum_{v \in \mathcal{V}} \|\mathcal{P}_{out}(v) - \mathcal{P}_{gt}(v)\|_2^2 \tag{1}$$

B. Normal Loss

The normal loss computes the mean angular deviation between the normals of the corresponding faces. Let $\widehat{\mathcal{N}}_{out}(s)$ and $\widehat{\mathcal{N}}_{gt}(s)$ denote the unit normal of the face s, where the normal direction of a face is computed by taking the cross product between two edges of the triangular face. The normal loss is defined in Equation 2, where, $\langle\cdot,\cdot\rangle$ denotes inner-product.

$$\mathcal{L}_{normal} = \frac{1}{|\mathcal{F}|}\sum_{s \in \mathcal{F}} \cos^{-1}\left(\langle\widehat{\mathcal{N}}_{out}(s), \widehat{\mathcal{N}}_{gt}(s)\rangle\right) \tag{2}$$

C. Curvature Loss

We use two types of curvature loss: the mean curvature loss and the Gaussian curvature loss. The mean curvature κ^H is computed using the mean curvature normal operator and the Gaussian curvature κ^G is computed using the Gaussian curvature operator. Let $\mathcal{N}(v)$ denote the neighbourhood of vertex v in the graph and let $\mathcal{N}_{\mathcal{F}}(v)$ denote the set of faces that contain the vertex v. Then the curvature values at v for the two graphs $\mathcal{G}_{gt}$ and $\mathcal{G}_{out}$ are denoted by $\kappa^H_{out}(v)$, $\kappa^H_{gt}(v)$, $\kappa^G_{out}(v)$, and $\kappa^G_{gt}(v)$. The mean curvatures are defined in Equations 3 and 4.

$$\kappa^H_{out}(v) = \frac{1}{4\mathcal{A}_{Mixed}(v)} \left\| \sum_{u\in\mathcal{N}(v)} w_{vu}(\mathcal{P}_{out}(v) - \mathcal{P}_{out}(u)) \right\|_2 \tag{3}$$

$$\kappa^H_{gt}(v) = \frac{1}{4\mathcal{A}_{Mixed}(v)} \left\| \sum_{u\in\mathcal{N}(v)} w_{vu}(\mathcal{P}_{gt}(v) - \mathcal{P}_{gt}(u)) \right\|_2 \tag{4}$$

Here, $w_{vu} = \sum_{f\in\mathcal{F}_{vu}} \cot \alpha^f_{vu}$ denotes the cotangent weights, in which, α^f_{vu} is the interior angle of the vertex opposite to edge vu in face f and $\mathcal{F}_{vu}$ is the collection of all faces that contain edge vu. $\mathcal{A}_{Mixed}(v)$ denotes the augmented version of the Voronoi region area of a vertex v. This augmentation takes into account the case where the triangular faces of the mesh are obtuse. The Gaussian curvatures are defined in Equations 5 and 6.

$$\kappa^G_{out}(v) = \frac{1}{\mathcal{A}_{Mixed}(v)}(2\pi - \sum_{u\in\mathcal{N}_{\mathcal{F}}(v)} \theta_{out}(v,u)) \tag{5}$$

$$\kappa^G_{gt}(v) = \frac{1}{\mathcal{A}_{Mixed}(v)}(2\pi - \sum_{u\in\mathcal{N}_{\mathcal{F}}(v)} \theta_{gt}(v,u)) \tag{6}$$

The two components of curvature loss are then defined in Equation 7 and 8.

$$\mathcal{L}_H = \frac{1}{|\mathcal{V}|} \sum_{v\in\mathcal{V}} \left\| \kappa^H_{out}(v) - \kappa^H_{gt}(v) \right\|_1 \left(\kappa^H_{gt}(v) \right)^2 \tag{7}$$

$$\mathcal{L}_G = \frac{1}{|\mathcal{V}|} \sum_{v\in\mathcal{V}} \left\| \kappa^G_{out}(v) - \kappa^G_{gt}(v) \right\|_1 \left(\kappa^G_{gt}(v) \right)^2 \tag{8}$$

Here, $\|\cdot\|_1$ is the absolute value operator. Finally we define the total curvature loss as the average of the two curvature losses in Equation 9.

$$\mathcal{L}_{curvature} = (\gamma_H \mathcal{L}_H + \gamma_G \mathcal{L}_G) \tag{9}$$

Here, γ_H and γ_G are weights used for combining the two curvatures. Based on distribution of objects in our dataset, we choose $\gamma_H = 10^{-6}$ and $\gamma_G = 1$.

D. Chamfer Loss

Let $\mathcal{V}_{out}$ and $\mathcal{V}_{gt}$ denote the vertex sets of the two graphs and let $\mathcal{P}_{out}(v)$ and $\mathcal{P}_{gt}(v)$ denote the features of the vertex v in the two graphs. Then the Chamfer loss is defined in Equation 10

$$\mathcal{L}_{chamfer} = \frac{1}{|\mathcal{V}_{out}|} \sum_{u\in\mathcal{V}_{out}} \min_{v\in\mathcal{V}_{gt}} \|\mathcal{P}_{out}(u) - \mathcal{P}_{gt}(v)\|_2^2 + \frac{1}{|\mathcal{V}_{gt}|} \sum_{v\in\mathcal{V}_{gt}} \min_{u\in\mathcal{V}_{out}} \|\mathcal{P}_{out}(u) - \mathcal{P}_{gt}(v)\|_2^2 \tag{10}$$

E. Feature Extractor Loss

The output of the feature extractor block is a matrix of size $n\times 5$ which contains the estimate of normals and curvatures for each vertex. Let $\widehat{\mathcal{N}}_{fe}(v)$, $\kappa^H_{fe}(v)$, and $\kappa^G_{fe}(v)$ be the normal, the mean curvature and the Gaussian curvature respectively estimated by the feature extractor for vertex v. Let $\widehat{\mathcal{N}}_{gt}(v)$, $\kappa^H_{gt}(v)(v)$, and $\kappa^G_{gt}(v)$ be the normal, the mean curvature and the Gaussian curvature respectively of vertex v in the original ground-truth mesh. The feature extractor loss is then given in Equation 11

$$\mathcal{L}_{FE} = \frac{1}{|\mathcal{V}|} \sum_{v\in\mathcal{V}} \Bigg(\left\| \widehat{\mathcal{N}}_{fe}(v) - \widehat{\mathcal{N}}_{gt}(v) \right\|_2^2 + \left\| \kappa^H_{fe}(v) - \kappa^H_{gt}(v) \right\|_2^2 + \left\| \kappa^G_{fe}(v) - \kappa^G_{gt}(v) \right\|_2^2 \Bigg) \tag{11}$$

F. Loss function weights

For training our network, we use a linear combination of the loss functions described above.The weights used for vertex loss, normal loss, curvature loss, chamfer loss and the feature extractor loss are $\lambda_V = 1$, $\lambda_N = 0.2$, $\lambda_\kappa = 0.01$, $\lambda_C = 0.05$ and $\lambda_{FE} = 1$, respectively.

We define several loss functions whose objective is to make $\mathcal{G}_{out}$ as close to $\mathcal{G}_{gt}$ as possible. In order to get $\mathcal{P}_{out}$ close to $\mathcal{P}_{gt}$, the natural loss function to use would be a metric that compares the distance between the corresponding pair of vertices in the two graphs. In cases where the correspondence between vertices is known, we can directly use the vertex-wise Euclidean distance and then average them over all the vertices. We use this loss and call it vertex loss. However, in cases where the correspondence between the vertices of the two graphs is not known, it is appropriate to use distance metric defined between two sets as the loss function, such as, Hausdorff distance, Chamfer distance, Earth mover's distance, etc. We include the Chamfer loss in our framework.

It might be argued that, since in DMD-Net the correspondence between vertices is preserved throughout, what is the need for using Chamfer distance? A response to this argument might be to note that while performing denoising using a deep learning framework, the network can contain components that may destroy the correspondence. For instance, one can conceive of an autoencoder which obtains a latent code for the entire graph in the encoding stage, thereby destroying all the structure, and while decoding, it might output a fixed number of vertices thereby changing the number of vertices in the two graphs. There could be many other such possibilities that destroy the correspondence. In such cases, vertex loss would fail. Hence, for the sake of making our framework more general and robust to these possibilities, we allow the use of both Chamfer loss and Vertex loss.

In the absence of original correspondence, we can still obtain a correspondence by solving the assignment problem through cost minimization. Once a correspondence is obtained, many other loss functions can further be used. For instance, comparing the length of edges, area of faces, location of centroids, direction of normal of faces, laplacian of vertices,

TABLE I
TRAINING SCHEME - ABLATION STUDY.

Model	test-intra			test-inter		
	Vertex ($\times 10^{-4}$)	Normal (degrees)	Chamfer ($\times 10^{-4}$)	Vertex ($\times 10^{-4}$)	Normal (degrees)	Chamfer ($\times 10^{-4}$)
Joint	**3.917**	**25.458**	**2.028**	**3.717**	**25.184**	**2.035**
Alter	5.112	26.458	2.46	4.882	26.056	2.459
2-Phase	3.987	26.216	2.097	3.937	25.722	2.155
U-FE	4.768	25.767	2.399	5.007	25.387	2.544

TABLE II
DROPOUT RATE - ABLATION STUDY.

Model	test-intra			test-inter		
	Vertex ($\times 10^{-4}$)	Normal (degrees)	Chamfer ($\times 10^{-4}$)	Vertex ($\times 10^{-4}$)	Normal (degrees)	Chamfer ($\times 10^{-4}$)
$D_r = 0$	**3.917**	**25.458**	**2.028**	**3.717**	**25.184**	**2.035**
$D_r = 0.2$	101.777	45.86	17.912	98.108	46.778	18.743
$D_r = 0.35$	94.623	47.578	19.767	95.461	47.836	21.056
$D_r = 0.5$	220.129	47.905	33.388	242.294	49.462	37.828

TABLE III
GRAPH ATTENTION NETWORKS - ABLATION STUDY.

Model	test-intra			test-inter		
	Vertex ($\times 10^{-4}$)	Normal (degrees)	Chamfer ($\times 10^{-4}$)	Vertex ($\times 10^{-4}$)	Normal (degrees)	Chamfer ($\times 10^{-4}$)
With Attention	27.881	28.689	6.845	31.399	28.493	7.974
Without Attention	**3.917**	**25.458**	**2.028**	**3.717**	**25.184**	**2.035**

curvature of vertices, etc. However, in spite of these several possibilities, we use only the loss functions that enforce the vertex locations, face normals, and curvatures of the output denoised mesh to be aligned with that of the ground-truth mesh.

IV. ABLATION STUDIES

We conduct several ablation studies, where, we devise several variants of the proposed approach and show that the proposed approach outperforms all the variants. This establishes the importance of each component in our network. In all of our ablation studies, we train the networks on mixed noise, that is, we randomly choose the noise type and the noise level in each iteration. For the depth ablation we train all the variants for 200 epochs. For the rest of the ablation studies, we train all the variants for 60 epochs. Except the loss ablation study and the training scheme ablation study, all other ablation studies use a linear combination of loss functions with the following weights: $\lambda_V = \lambda_N = \lambda_\kappa = \lambda_{FE} = \lambda_C = 1$

A. Training Scheme Ablation

The loss functions discussed in Section III can be segregated into two categories. The first category consists of the feature extractor loss which provides feedback solely to the feature extractor. The second category consists of the rest of the loss functions which provide feedback to the entire network. Thus, this segregation, naturally opens the possibility of various different competing training schemes as below.

• *Joint Optimization (Joint).* The final loss function is a linear combination of all the loss functions. All the components of the network are jointly trained using this final loss function. In this case $\lambda_V = \lambda_N = \lambda_\kappa = \lambda_{FE} = \lambda_C = 1$.

• *Alternating Optimization (Alter).* The training alternates between training the feature extractor and training the rest of the network. In odd iterations, the feature extractor is trained using the feature extractor loss while the rest of the network is frozen. We thus have, $\lambda_{FE} = 1$ and $\lambda_V = \lambda_N = \lambda_\kappa = \lambda_C = 0$. In the even iterations, the feature extractor is frozen while the rest of the network is trained using the loss functions in the second category. We thus have $\lambda_{FE} = 0$ and $\lambda_V = \lambda_N = \lambda_\kappa = \lambda_C = 1$.

• *Two Phase Optimization (2-Phase).* The training process is divided into two phases: the pre-training phase and the post-training phase. In the pre-training phase the feature extractor is trained for several epochs keeping the rest of the network frozen, with the following weights of loss functions $\lambda_{FE} = 1$ and $\lambda_V = \lambda_N = \lambda_\kappa = \lambda_C = 0$. In the post-training phase the rest of the network is trained for the remaining epochs keeping the feature extractor frozen, in which case, the loss function weights are $\lambda_{FE} = 0$ and $\lambda_V = \lambda_N = \lambda_\kappa = \lambda_C = 1$.

• *Unsupervised Feature Extraction (U-FE).* The feature extractor loss is switched off in this case. The feature extractor receives feedback from rest of the loss functions. Since the feature extractor loss is disabled, the supervised training with local features does not occur. Hence, the feature extraction step becomes an unsupervised process. The weights of the loss functions are $\lambda_{FE} = 0$ and $\lambda_V = \lambda_N = \lambda_\kappa = \lambda_C = 1$.

In this ablation study, we use a pair of two-stream networks in both the feature extractor as well as the denoiser. As visible in Table I, the joint optimization scheme outperforms the rest of the training schemes. From now on we use joint optimization for the remaining ablation studies.

B. Dropout Rate Ablation

In this study we include dropout layer in our network after each ReLU function. We experiment with four different dropout rate D_r: (a) $D_r = 0.5$, (b) $D_r = 0.35$, (c) $D_r = 0.2$, and (d) $D_r = 0$. In Table II, we find that including dropout layer worsens the performance. Therefore, we do not include dropout layers in our network.

C. Graph Attention Networks Ablation

We test the effect of including graph attention network in the graph aggregation layer. We find in Table III, that, the inclusion of graph attention network degrades the performance. Hence, we exclude graph attention network from the proposed approach.

D. Structure Ablation

In this study we modify the structure of our network in various ways and show that our network in its current form performs the best. We devise the following variants:

• *FGT-8.* This is the architecture we propose in our work. In this method, the noisy input mesh is combined with the estimated local features before being transformed by the transformer.

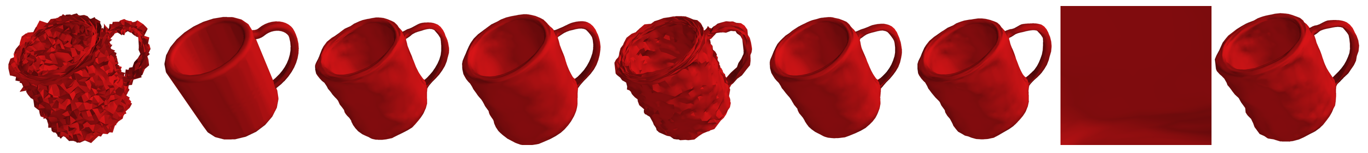

Fig. 2: Output of loss ablation variants on the cup model. (a) Noisy, (b) Original, (c) With All, (d) Without Vertex, (e) Without Normal, (f) Without Curvature, (g) Without Chamfer, (h1) Without Vertex and Chamfer, and (h2) Without Vertex and Chamfer (zoomed-out). The figures from (a)-(h1) are all under the same camera setting. (h2) is the zoomed out version of (h1).

TABLE IV
NETWORK STRUCTURE - ABLATION STUDY.

Model	**test-intra**			**test-inter**		
	Vertex ($\times 10^{-4}$)	Normal (degrees)	Chamfer ($\times 10^{-4}$)	Vertex ($\times 10^{-4}$)	Normal (degrees)	Chamfer ($\times 10^{-4}$)
FGT-8	**3.917**	**25.458**	**2.028**	**3.717**	**25.184**	**2.035**
FGT-3	4.506	26.396	2.224	4.306	25.917	2.14
W/O T	6.112	26.328	2.849	5.588	25.994	2.696
MT	5.586	26.366	2.653	5.521	26.097	2.643
DO-2S	5.398	26.224	2.649	4.967	25.878	2.548
DO-1S	6.097	27.356	2.827	6.02	26.801	2.888
FGT-1S	9.62	28.368	3.969	9.101	27.963	3.654

• *FGT-3*. FGT-3 is similar in structure to FGT-8 except that, in this method, the noisy input mesh is directly passed to the transformer without being combined with the estimated local features.
• *Without Transformer (W/O T)*. Here we eliminate the transformer. The combination of the local features and the noisy mesh are directly passed to the primal stream of the denoiser.
• *Multiple Transformer (MT)*. Here instead of computing a single transformation matrix, the transformer computes a cascade of transformations which is then applied to the combination of noisy mesh and the estimated features.
• *Denoiser Only - Two Stream (DO-2S)*. Here we eliminate both the feature extractor as well as the transformer. The noisy mesh is directly fed to both the streams of the denoiser. The purpose of this network is to validate the use of the feature extractor and transformer in the FGT paradigm.
• *Denoiser Only - Single Stream (DO-1S)*. This is same as the Denoiser Only (Two Stream) except that now the denoiser contains only a single stream. This also implies that the two-stream network does not contain the primal dual fusion block. The purpose of this network is to test how well a vanilla graph neural network works on the mesh denoising task.
• *FGT - Single Stream (FGT-1S)*. This is similar in structure to the proposed approach FGT-8, the only difference being that both the denoiser and the feature extractor contain a single stream. The purpose of this network is to validate the use of doing aggregation in both primal and dual graph.

The comparison is shown in Table IV. We find that the proposed network FGT-8 performs the best. We use this architecture in the consequent ablation studies.

E. Loss Ablation

In this study we experiment with the weights of several loss functions. This study is conducted to find out the relevance of each loss function. Table V shows the results for various different configurations. The first row (With All) refers to the case where all the loss functions are used. In the second row (Without Vertex) when we drop only the vertex loss, we observe that all the three metrics degrade, thereby indicating the significance of vertex loss. When we drop only the normal loss (Without Normal), we observe significant improvement in two metrics but large degradation in the normal metric. When we drop curvature loss (Without Curvature) or chamfer loss (Without Chamfer), we do not see significant change in the three metrics. This indicates that chamfer loss and curvature loss are not of very high significance. When we drop both vertex loss and chamfer loss, we see a very huge jump in the vertex and chamfer metric, indicating that normal loss alone is not sufficient. Based on these results and some further experimentation, we arrive at the final weights of the loss functions: $\lambda_V = 1$, $\lambda_N = 0.2$, $\lambda_\kappa = 0.01$, $\lambda_{FE} = 1$, and $\lambda_C = 0.05$. The visual comparison of these loss ablation variants are made in Figure 2. In Figure 2(a)-(h1) the camera setting are kept same during rendering. Figure 2(h2) is the zoomed out image of 2(h1) to fit the entire object inside the field of view of the camera. As visible in Figure 2(h2), in the absence of both vertex loss and chamfer loss the output model achieves shape similar to the ground truth. But the object is a highly scaled up version hence the red patch in Figure 2(h1). This scaling up of the object is the reason why the vertex loss metric and the chamfer loss metric is very high in last row of Table V. As can be seen in the Figure 2(e), DMD-Net without normal loss performs very poorly, thereby signifying the importance of normal loss though this variant achieves the best vertex loss (Table V)

F. Depth Ablation

Depth is defined as the number of two-stream networks in the denoiser. We perform this experiment with four different variants: (a) *DMD-Net 1*. contains only one two-stream networks (b) *DMD-Net 2*. contains two two-stream networks (c) *DMD-Net 3*. contains three two-stream networks (d) *DMD-Net 4*. contains four two-stream networks In Table VI we find out that DMD-Net 2 performs the best. Thus, based on these results, we include only two two-stream networks in the denoiser in our proposed approach. Note that in depth ablation study all variant are trained for 200 epochs.

TABLE V
LOSS ABLATION - STUDY.

Model					test-intra			test-inter		
Loss	λ_V	λ_N	λ_κ	λ_C	Vertex ($\times 10^{-4}$)	Normal (degrees)	Chamfer ($\times 10^{-4}$)	Vertex ($\times 10^{-4}$)	Normal (degrees)	Chamfer ($\times 10^{-4}$)
With All	1	1	1	1	4.624	25.98	2.337	4.353	25.725	2.264
Without Vertex	0	1	1	1	17.291	26.214	5.98	14.282	25.961	5.188
Without Normal	1	0	1	1	**1.923**	40.37	**1.037**	**1.907**	39.985	**1.054**
Without Curvature	1	1	0	1	4.585	26.493	2.342	4.551	25.966	2.379
Without Chamfer	1	1	1	0	4.601	26.086	2.359	4.406	25.844	2.286
Without Vertex and Chamfer	0	1	1	0	1.039×10^{16}	**25.302**	2.938×10^{15}	9.449×10^{15}	**24.979**	3.113×10^{15}

TABLE VI
DEPTH ABLATION - STUDY.

Model	test-intra			test-inter		
	Vertex ($\times 10^{-4}$)	Normal (degrees)	Chamfer ($\times 10^{-4}$)	Vertex ($\times 10^{-4}$)	Normal (degrees)	Chamfer ($\times 10^{-4}$)
DMD-Net 1	5.498	26.12	2.487	4.968	25.8	2.369
DMD-Net 2	**3.301**	**25.37**	**1.786**	**3.271**	**25.053**	**1.815**
DMD-Net 3	4.968	26.004	2.393	4.755	25.509	2.362
DMD-Net 4	5.161	25.98	2.493	4.812	25.575	2.246

TABLE VII
COMPARISON OF THE PROPOSED APPROACH WITH AND WITHOUT SPATIAL TRANSFORMER NETWORKS.

Model	test-intra			test-inter		
	Vertex ($\times 10^{-4}$)	Normal (degrees)	Chamfer ($\times 10^{-4}$)	Vertex ($\times 10^{-4}$)	Normal (degrees)	Chamfer ($\times 10^{-4}$)
W STN	5.233	26.238	2.471	4.692	25.927	2.189
W/O STN	**3.301**	**25.37**	**1.786**	**3.271**	**25.053**	**1.815**

V. TRANSFORMATION EQUIVARIANCE

First we define what we mean by transformation equivariance. Let $\mathcal{T}$ be a transformation operator, let $\mathcal{D}$ be a mesh denoising algorithm and let $\mathcal{G}$ denote a mesh. We say that $\mathcal{D}$ has $\mathcal{T}$-equivariance, if for any given mesh, $\mathcal{G}$ we have $\mathcal{T}(\mathcal{D}(\mathcal{G})) = \mathcal{D}(\mathcal{T}(\mathcal{G}))$. That is, $\mathcal{D}$ is $\mathcal{T}$-equivariant, if $\mathcal{D}$ and $\mathcal{T}$ commute. We now discuss whether DMD-Net is equivariant with respect to the following mentioned transformations.

1) Scaling Equivariance: Given a mesh, we first normalize it to fit inside a unit cube. We then denoise it using DMD-Net and unnormalize it back to it's original scale. Thus DMD-Net is scale equivariant as it first converts the mesh into a cannonical scale before denoising.

A. Translation Equivariance

Before denoising, we shift the mesh to the origin and after denoising, we shift it back to its original location. Thus, DMD-Net is translation equivariant as the object is always aligned to the origin.

DMD-Net is not rotation equivariant in the theoretical sense. However, we try to establish rotation equivariance by introducing concepts like rotation augmentation in the dataset and the use of spatial transformer networks (STN). In rotation augmentation, we augment the data during training by randomly rotating the ground truth mesh before adding noise. Spatial transformer networks (STN) was used by to canonicalize the orientation of the object to establish rotation invariance/equivariance in a deep learning framework that processes point clouds. We experiment with the idea of STN to check whether it establishes equivariance in our case. Through the use of a rotation equivariance test we evaluate the degree to which rotation equivariance is established in DMD-Net by these concepts. We first combine the test-intra and the test-inter sets to obtain a single test set $\mathcal{S}$. For the i^{th} mesh $\mathcal{G}_i$ in this set we choose $\gamma = 5$ random rotation operators, the j^{th} rotation operator denoted as $\mathcal{R}_{ij}$. We then compute the following two meshes: $\mathcal{G}_{ij}^{\mathcal{RD}} = \mathcal{R}_{ij}(\mathcal{D}(\mathcal{G}_i))$ and $\mathcal{G}_{ij}^{\mathcal{DR}} = \mathcal{D}(\mathcal{R}_{ij}(\mathcal{G}_i))$. We then find the distance between $\mathcal{G}_{ij}^{\mathcal{RD}}$ and $\mathcal{G}_{ij}^{\mathcal{DR}}$ by evaluating the following loss metrics: $\mathcal{L}_{vertex}(\mathcal{G}_{ij}^{\mathcal{RD}}, \mathcal{G}_{ij}^{\mathcal{DR}})$, $\mathcal{L}_{normal}(\mathcal{G}_{ij}^{\mathcal{RD}}, \mathcal{G}_{ij}^{\mathcal{DR}})$, and $\mathcal{L}_{chamfer}(\mathcal{G}_{ij}^{\mathcal{RD}}, \mathcal{G}_{ij}^{\mathcal{DR}})$. We then sum these loss metrics over the entire test set $\mathcal{S}$ as mentioned in Equations 12, 13, and 14.

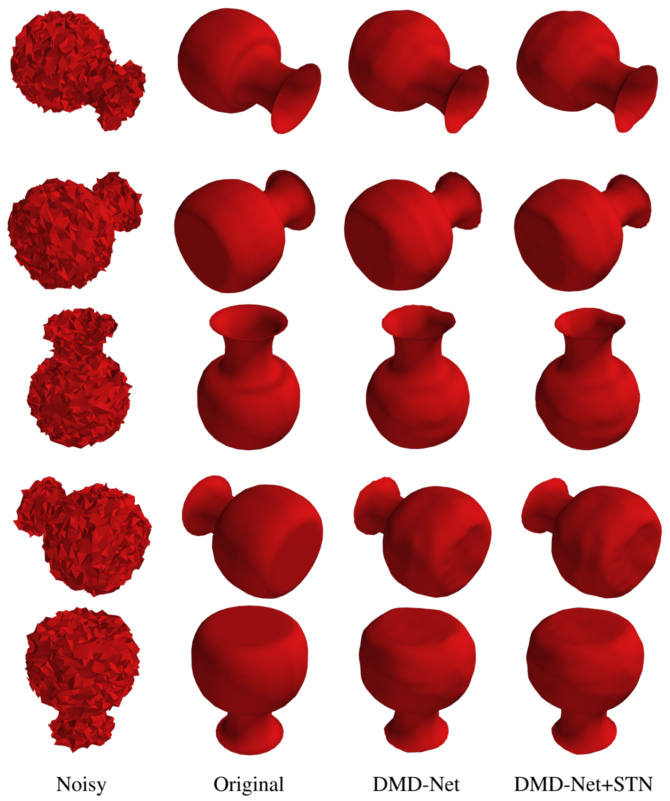

Fig. 3: Visualization of the degree to which rotation equivariance is achieved by DMD-Net and DMD-Net+STN. Objects in the first column are rotated versions of each other. The second column refers to the corresponding ground truth objects. Third column and the fourth column are the outputs of the DMD-Net and DMD-Net+STN.

TABLE VIII
ROTATION EQUIVARIANCE TEST FOR MEASURING THE DEGREE TO WHICH ROTATION EQUIVARIANCE IS ESTABLISHED.

Model	test intra + test inter		
	Vertex ($\times 10^{-4}$)	Normal (degrees)	Chamfer ($\times 10^{-4}$)
Rotation Augmentation	**0.704**	5.998	**0.501**
Rotation Augmentation + STN	3.723	**5.711**	1.796

$$\mathcal{L}_{\mathcal{V}}^{\mathcal{RE}} = \frac{1}{\gamma|S|}\sum_{i=1}^{|S|}\sum_{j=1}^{\gamma}\mathcal{L}_{vertex}(\mathcal{G}_{ij}^{\mathcal{RD}}, \mathcal{G}_{ij}^{\mathcal{DR}}) \tag{12}$$

$$\mathcal{L}_{\mathcal{N}}^{\mathcal{RE}} = \frac{1}{\gamma|S|}\sum_{i=1}^{|S|}\sum_{j=1}^{\gamma}\mathcal{L}_{normal}(\mathcal{G}_{ij}^{\mathcal{RD}}, \mathcal{G}_{ij}^{\mathcal{DR}}) \tag{13}$$

$$\mathcal{L}_{\mathcal{C}}^{\mathcal{RE}} = \frac{1}{\gamma|S|}\sum_{i=1}^{|S|}\sum_{j=1}^{\gamma}\mathcal{L}_{chamfer}(\mathcal{G}_{ij}^{\mathcal{RD}}, \mathcal{G}_{ij}^{\mathcal{DR}}) \tag{14}$$

We conduct rotation equivariance test in two cases: (a) Rotation Augmentation and (b) Rotation Augmentation with Spatial Transformer Network. The results are depicted in Table VIII. In terms of $\mathcal{L}_{\mathcal{V}}^{\mathcal{RE}}$ and $\mathcal{L}_{\mathcal{C}}^{\mathcal{RE}}$, case (a) performs far better but in terms of $\mathcal{L}_{\mathcal{N}}^{\mathcal{RE}}$, case(b) performs slightly better. Thus, we discover that introducing STN does not provide significant improvement but rather degrades the overall equivariance metric. Moreover, in Table VII we assess the effect of adding the STN by comparing the results on the mesh denoising task. This table has a similar setting as that of the ablation studies mentioned in Section IV. We conclude that introducing STN does not establish rotation equivariance to a larger degree. Moreover, it also degrades the performance on mesh denoising. Thus, we do not include STN in the proposed approach and only use rotation augmentation for rotation equivariance. In Figure 3, we visually show that DMD-Net has rotation equivariance to a large degree. The five rows in the figure correspond to five different rotations. All the images are rendered using a fixed camera position and orientation.

VI. Supplementary Video

A brief audio-visual explanation of DMD-Net is included in the supplementary video, which has been uploaded online as an unlisted video on youtube. The video can be accessed through the following link: https://youtu.be/wPJvUYqiL94