SUPPLEMENTARY MATERIAL FOR “WRAPPINGNET: MESH AUTOENCODER VIA DEEP SPHERE DEFORMATION”

Eric Lei[*][†], *Muhammad Asad Lodhi*[*], *Jiahao Pang*[*], *Junghyun Ahn*[*], *Dong Tian*[*]

[*]InterDigital, New York, NY, USA
[†]Dept. of Electrical and Systems Engineering, University of Pennsylvania, Philadelphia, PA, USA
elei@seas.upenn.edu, {muhammad.lodhi, jiahao.pang, junghyun.ahn, dong.tian}@interdigital.com

I. ADDITIONAL EXPERIMENTAL RESULTS

We first discuss additional experimental results, paralleling Sec. 4.2 (latent code for shape representation), 4.3 (reconstructed shape analysis), and 4.4 (ablation study) in the main text.

I-A. Latent Code for Shape Representation

We first provide additional t-SNE plots, followed by a new experiment demonstrating the added benefits of connectivity information for shape representation when the number of points used to represent the shape is low.

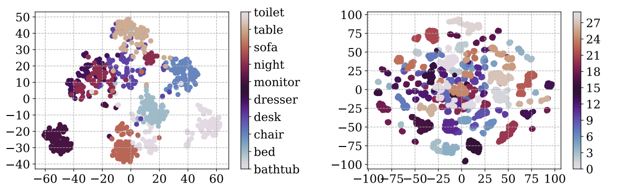
I-A.1. Additional t-SNE Visualization

We additionally show t-SNE plots for both Manifold10 and SHREC11 test sets of the learned latent codes from WrappingNet, in Fig. I, similar to the experiment in Sec. 4.2. Similar to the t-SNE plot of Manifold40, we see that the latent codes indeed cluster by class, demonstrating that WrappingNet learned global shape information in the latent codes. All figures (both t-SNE figures here and the one in the main text) use a perplexity parameter of 30.

I-A.2. Importance of Connectivity in the Sparse Regime

As shown in the Sec. 4.2 of the main text, WrappingNet achieves similar classification performance as FoldingNet and TearingNet for the full-resolution subdivision meshes as well as the original manifold meshes. At the full resolution, the subdivision-remeshed Manifold40 has, on average, 3000 points per mesh; the original manifold meshes have a density of approximately 250 points per mesh. At these resolutions, the density of points on the surface of each shape is high enough such that the high-level information of the shape (e.g., its class membership) is preserved whether or not mesh connectivity is included. However, at much lower resolutions, a lack of mesh connectivity can significantly degrade the topology information of the shape. For example, a table

Work done while the author was an intern at InterDigital.



(a) Manifold10. (b) SHREC11.

Fig. I: t-SNE plots of test latent codes extracted from subdivision WrappingNet.

top, which is flat, only needs 4 points (at the corners) with mesh connectivity to represent the flat surface; if connectivity is not available, 4 points at the corners will fail to represent the topology accurately. Since WrappingNet employs a mesh surface-based feature extraction, we expect it to maintain good performance even at lower point densities, where point cloud systems may begin to worsen in performance.

We experimentally test this by training the models at a much lower point density of 50 points, on average. This corresponds to the point density of the simplified base mesh extracted during the subdivision remeshing. We use vanilla WrappingNet since there is no subdivision structure for the base mesh. For the point cloud autoencoders, we only use the vertex positions of the lower resolution meshes, and their decoder deforms a grid of similar size. Shown in Tab. I, all models are able to maintain similar classification performance at the 250 point level (original manifold meshes) and the 300 point level (subdivision meshes). However, at 50 points, the performance of FoldingNet and TearingNet drops a significant amount by $\sim 3\%$ for Manifold40 and $\sim 4\%$ for Manifold10 when compared to the highest resolution. In comparison, WrappingNet only drops by $\sim 1\%$ for Manifold40 and $\sim 2\%$ for Manifold10. Overall, at higher resolutions, WrappingNet achieves comparable classification performance to the point cloud autoencoders, but at 50 points, WrappingNet achieves a *clearly better* classification performance, demonstrating the importance of utilizing mesh connectivity at lower resolutions.

Table I: Average classification performance from test latent representations using SVM with 5-fold cross-validation.

Dataset	Manifold10			Manifold40		
No. of Points	50	250	3000	50	250	3000
FoldingNet	86.8%	89.8%	91.0%	79.4%	82.5%	82.5%
TearingNet	86.7%	90.2%	90.8%	79.4%	82.9%	82.9%
WrappingNet	89.0%	90.8%	90.9%	82.2%	83.0%	83.3%

I-B. Reconstructed Shape Analysis

We describe in detail some of the metrics used to evaluate the reconstruction performance, and provide additional reconstruction examples following the latent interpolation and varying base graph experiment in Sec. 4.3. Then, we display a gallery of reconstructed shapes generated from WrappingNet.

I-B.1. Reconstruction Metrics

To evaluate reconstruction performance with point cloud autoencoders, we use Chamfer distance (CD), normals error (NE), and curvature preservation (CP). The latter two are inspired from surface-aware losses used in [1]. All metrics are used to compare two sets of points S_1, S_2. In our setting, one represents the ground-truth vertex positions, and the other represents the reconstructed vertex positions. As mentioned in the main text, these metrics are all point-based (i.e., they do not consider connectivity information) for the purposes of comparing to the point cloud autoencoders. CD is computed as

$$\begin{aligned}\mathsf{CD}(S_1, S_2) := \frac{1}{|S_1|}\sum_{\mathbf{x}\in S_1}\min_{\mathbf{x}'\in S_2}\|\mathbf{x}-\mathbf{x}'\|_2^2 \\ + \frac{1}{|S_2|}\sum_{\mathbf{x}\in S_2}\min_{\mathbf{x}'\in S_1}\|\mathbf{x}-\mathbf{x}'\|_2^2.\end{aligned} \tag{i}$$

NE metric is computed as

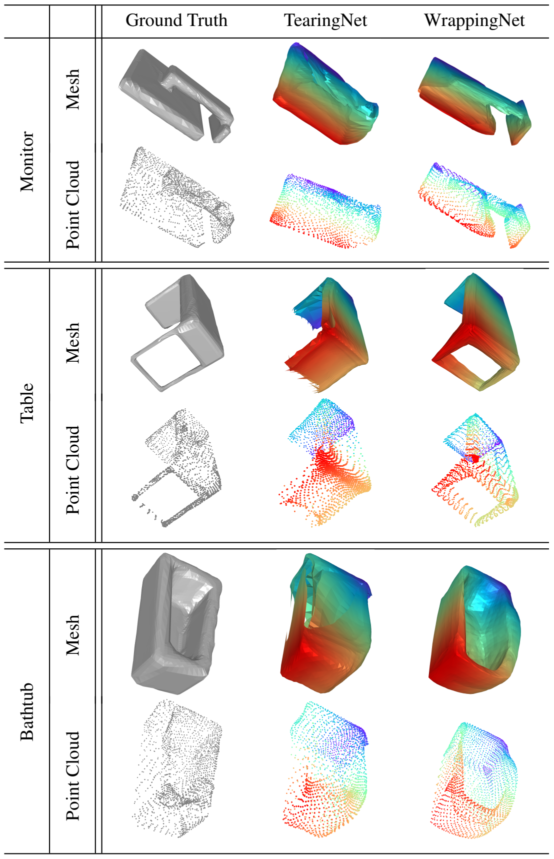
$$\begin{aligned}\mathsf{NE}(S_1, S_2) := \frac{1}{|S_1|}\sum_{\substack{\mathbf{x}\in S_1 \\ \mathbf{x}'=\arg\min_{\mathbf{y}\in S_2}\|\mathbf{x}-\mathbf{y}\|_2^2}} 1-\frac{\mathbf{n}_\mathbf{x}\cdot\mathbf{n}_{\mathbf{x}'}}{\|\mathbf{n}_\mathbf{x}\|_2\|\mathbf{n}_{\mathbf{x}'}\|_2} \\ + \frac{1}{|S_2|}\sum_{\substack{\mathbf{x}\in S_2 \\ \mathbf{x}'=\arg\min_{\mathbf{y}\in S_1}\|\mathbf{x}-\mathbf{y}\|_2^2}} 1-\frac{\mathbf{n}_\mathbf{x}\cdot\mathbf{n}_{\mathbf{x}'}}{\|\mathbf{n}_\mathbf{x}\|_2\|\mathbf{n}_{\mathbf{x}'}\|_2},\end{aligned} \tag{ii}$$

where $\mathbf{n}_\mathbf{x}$ is the normal vector at point $\mathbf{x}$. Normals are estimated via covariance methods if they do not exist. CP uses the covariance-based curvature of a k-nearest neighborhood around each point defined in [2, Sec. 3.1]. In our experiments, we set $k = 15$ for calculating the CP metric.

I-B.2. Additional Qualitative Reconstruction Comparison

We show additional qualitative topology comparisons with TearingNet reconstructed point clouds and meshes in Tab. II.

Table II: Topology comparison of reconstructed shapes. Both a point cloud and mesh rendering are provided for each object.



Again, as in the main text, WrappingNet clearly does a better job in preserving the mesh topology, which supports the lower reconstruction loss for the surface-aware metrics shown in Tab. 3 of the main text.

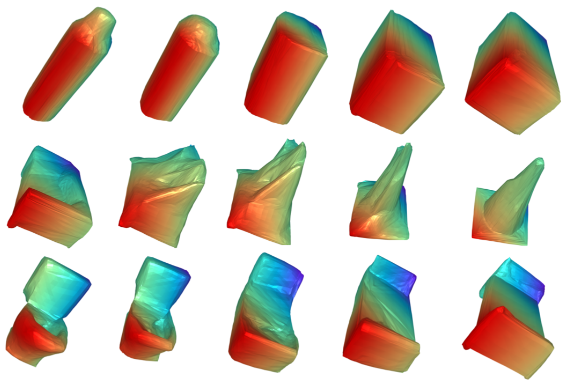
I-B.3. Latent Interpolation

The additional reconstructed shapes generated via interpolating the latent codes can be seen in Fig. II, which follows the procedure described in Sec. 4.3.

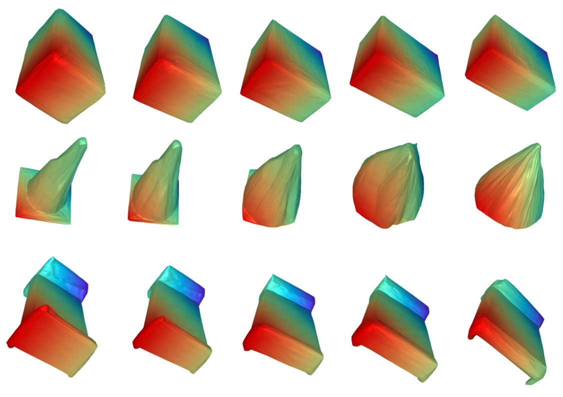
I-B.4. Reconstruction from Varying Base Graphs

The additional results demonstrating the reconstruction of shapes from varying the base graphs can be found in Tab. III. Again, we see that the codeword maintains global shape information, whereas the base graph influences the topology of the reconstructed shape. When the genus is mis-matched between the codeword and the base graph (gray-scale reconstructions), the decoder tries its best to fit the mis-matched topology to the global shape embedded in the codeword.

In particular, using the codeword from a higher genus object and the base graph from a lower genus, the reconstructed meshes still appear pleasant (the two gray meshes in the 3rd



(a) *Inter*-class. 1st row: bottle → glass box. 2nd row: sofa → cone. 3rd row: toilet → bed.



(b) *Intra*-class. 1st row: glass box. 2nd row: cone. 3rd row: bed.

Fig. II: Latent space interpolation. Base graph in use is generated from WrappingNet encoder on meshes from the left side.

row in Tab. III). On the contrary, if the codeword is from a lower genus object and the base graph is from a higher genus, the reconstruction is much less meaningful (the two gray meshes in the 3rd column in Tab. III). This provides additional evidence that the codeword is more dominant (than the base graph) in rebuilding the global shape, that is aligned with the intention of our autoencoder design.

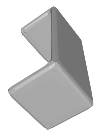
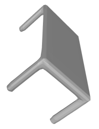
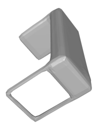
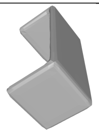
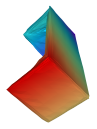
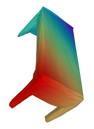
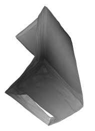
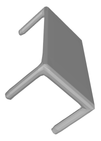
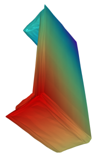
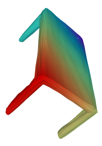
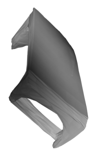
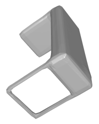
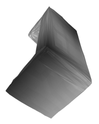
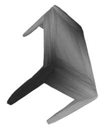
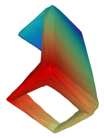
I-B.5. Gallery of reconstructions

In Fig. III, we display a gallery of reconstructed meshes from WrappingNet, on the Manifold40 test set. WrappingNet is able to learn a *shared latent space across all such meshes*, despite the heterogeneity of the dataset.

I-C. Ablation Study

We demonstrate t-SNE figures of extracted latent codes of the first two configurations in the ablation study (Sec. 4.4), shown in Fig. IV. These figures further support the benefit of wrapping operations in pushing the global shape information into the latent code. As shown, the first configuration (sending the base mesh) fails to capture any class-related clustering phenomenon in the latent codes, since the latent code and base mesh are entangled. In this setting, the latent code is merely used to upsample the base mesh. For the second configuration (sending M_{AS}), where we use wrapping operations but no matching, there is more disentanglement. The full WrappingNet model with both wrapping operations and matching (Fig. 5 of main text) displays the most disentanglement achieved.

Table III: Reconstructions from various base graphs & latent codes.

		Base graph comes from		
				
Latent codes comes from				
				
				

II. WRAPPINGNET IMPLEMENTATION DETAILS

The code will be made public upon publication.

II-A. Sphere Matching Details

Recall that the sphere grid of points on the sphere $\{(\bar{x}_i, \bar{y}_i, \bar{z}_i)\}_{i=1}^N$ is defined by the Fibonacci lattice [3]. Let $\bar{\mathbf{s}}_i = (\bar{x}_i, \bar{y}_i, \bar{z}_i)$ be the i-th point on the sphere grid, and let $\mathbf{s}_i = (x_i, y_i, z_i)$ be the i-th point of the output of the UnWrapping layers. Since UnWrapping has been pretrained to output near unit spheres, we expect the $\mathbf{s}_i$'s to approximately lie on the surface of a sphere. However, if we compute nearest neighbors between $\bar{\mathbf{s}}_i$'s and $\mathbf{s}_i$'s, they may not be in correspondence with each other. To resolve this, we use the assignment problem to compute a correspondence between the two sets of points.

Let $\mathbf{C}_{i,j} \triangleq \|\mathbf{s}_i - \bar{\mathbf{s}}_j\|_2^2$ be the pairwise squared Euclidean distance between the i-th and j-th points in the UnWrapping output and sphere grid, respectively. Since the number of points in the UnWrapped and sphere grid do not match, we solve the unbalanced variant of the assignment problem [4]

$$\hat{\sigma} = \underset{\sigma:\{1,\dots,N'\}\to\{1,\dots,N\}}{\arg\min} \sum_{i=1}^{N'} \mathbf{C}_{i,\sigma(i)}, \tag{iii}$$

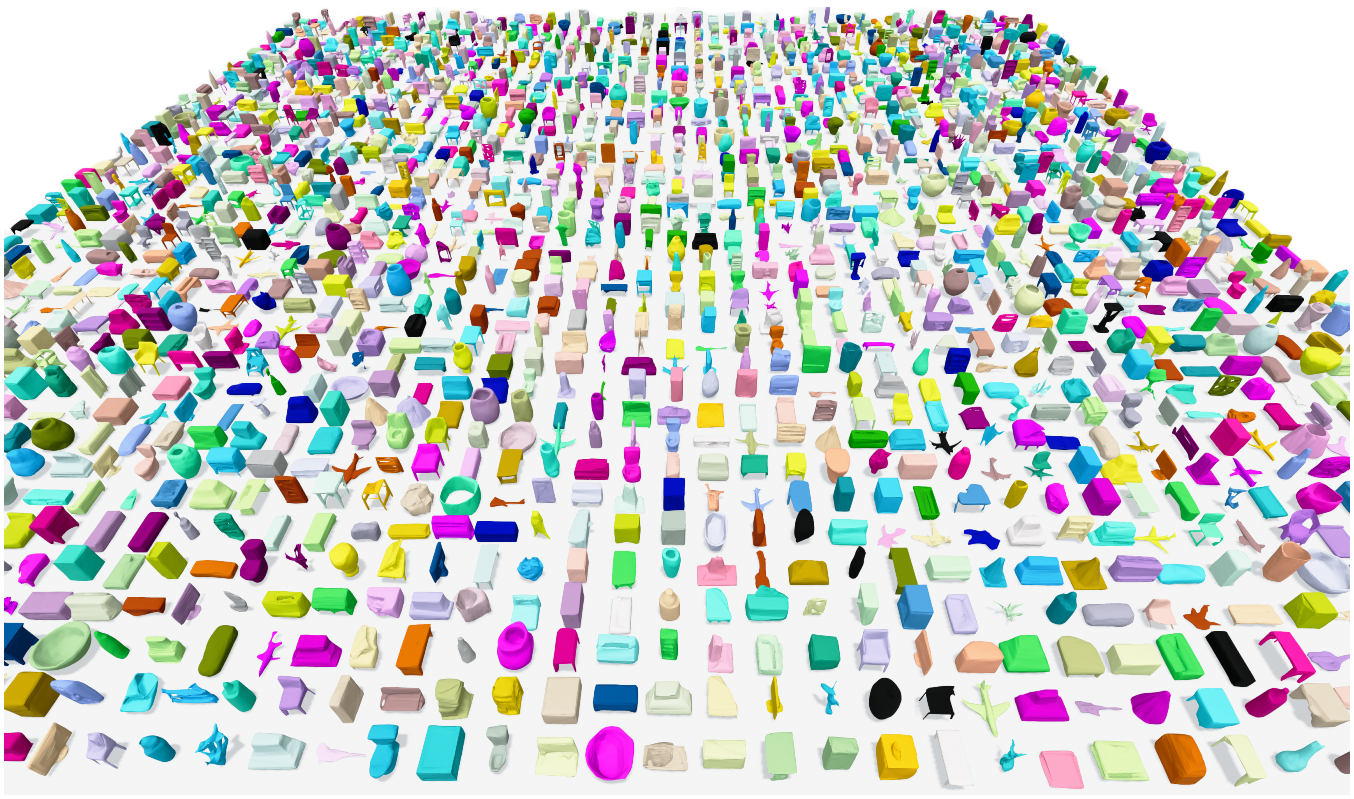
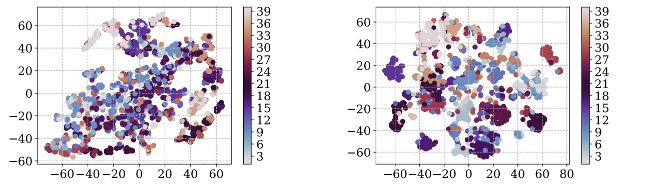


Fig. III: Gallery of reconstructed meshes. WrappingNet extracts a *shared latent space* across all such heterogeneous meshes.



(a) Send base mesh: no wrapping and no sphere matching.

(b) Send M_{AS}: with wrapping but without sphere matching.

Fig. IV: t-SNE plots of latent codes extracted from Manifold40 on different configurations (ablation study) of WrappingNet. Configuration with both wrapping/unwrapping and matching can be found in Fig. 5 of the main text.

where N' is the number of points the UnWrapping output. This can be solved using the Hungarian algorithm [5], which will return a bijection $\hat{\sigma}$.

Once the UnWrapping output gets matched with the sphere grid, the base graph (base mesh connectivity) gets re-indexed to be in the ordering of the sphere grid via $\hat{\sigma}$, and sent to the decoder. To enforce the vertex-to-vertex loss between the reconstructed mesh and ground-truth mesh, the matching $\hat{\sigma}$ can be used to ensure the input mesh and reconstructed mesh are in the same node ordering (either that of the original mesh or of the sphere grid). We clarify that the use of the matching $\hat{\sigma}$ to align the input mesh and reconstructed mesh is only needed to enforce the loss during training; during inference, all that is needed is to use the matching to re-index the base graph before sending it to the decoder.

For subdivision meshes, the sphere matching is performed at the lowest level of subdivision (the base mesh). Since subdivision meshes' face and vertex indexing is deterministic and completely determined by the base mesh's indexing, the bijection $\hat{\sigma}$ at the base mesh level is sufficient to compute the corresponding bijection at any level of subdivision.

II-B. Face Feature Initialization

We wish to ensure that the input features are invariant to the ordering of nodes and faces, and the global position or orientation of the face. Hence, we choose the input face features to be the normal vector of the face, the face area, and a vector containing curvature information of the face, which is defined as follows. For face i, let j_0, j_1, j_2 denote the face indices of its 3 neighbors. The curvature vector is simply $\mathbf{b}_i^c - \frac{1}{3}(\mathbf{b}_{j_0}^c + \mathbf{b}_{j_1}^c + \mathbf{b}_{j_2}^c)$ where $\mathbf{b}_i^c, \mathbf{b}_{j_0}^c, \mathbf{b}_{j_1}^c, \mathbf{b}_{j_2}^c$ are the centroids of the faces respectively. Thus, we have a total of 7 input features. These are used for the meshes that are inputs to the Wrapping and UnWrapping modules. For the Wrapping modules at the decoder, they are concatenated with the copied codeword.

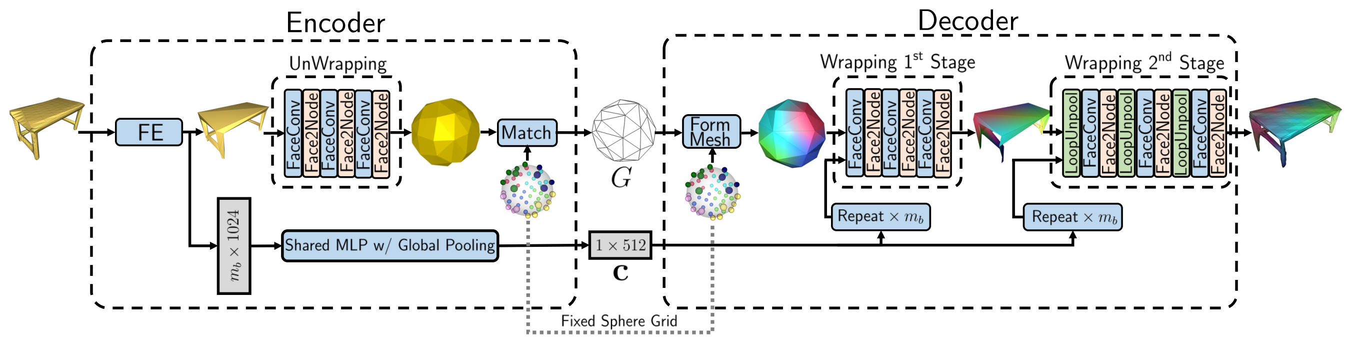


Fig. V: Subdivision-enhanced WrappingNet architecture. The input mesh is assumed to be a subdivision mesh with base mesh containing m_b faces. Loop pooling and subdivision are incorporated into the feature extractor (FE) at the encoder, and 2nd Wrapping stage at the decoder.

II-C. Model Architecture

All FaceConv layers use a kernel size of 3 and dilation of 1 defined in [6]. The convolution operation, defined there as well, is order-invariant, since it uses 4 learnable weights $w_0, \dots, w_3$ to apply the convolution, which is given by

$$\mathbf{F}'_i = w_0 \mathbf{F}_i + w_1 \sum_{j \in \mathcal{N}_i} \mathbf{F}_j + w_2 \sum_{j \in \mathcal{N}_i} |\mathbf{F}_{j+1} - \mathbf{F}_j| + w_3 \sum_{j \in \mathcal{N}_i} |\mathbf{F}_i - \mathbf{F}_j|, \qquad \text{(iv)}$$

where $\mathbf{F}_i$ is the face feature on face i and $\mathcal{N}_i$ is the set of three face indices adjacent to face i corresponding to its edges. This is done for each input-output channel pair.

We use the PyTorch framework [7] for the implementation of WrappingNet, along with PyTorch Geometric [8] to implement sparse batching of meshes. The full parameter list is found in Tab. IV. Multi-layer perceptron (MLP) layers are applied face-wise. All MLP and FaceConv layers use biases. ReLU activations exist after each fully connected layer in MLPs except for the last layer. MLPs in the Face2Node modules have 2 layers, and follow the same style as other MLPs. At the encoder, the feature extractor uses 4 FaceConv layers with hidden dimension of 128, and the shared MLP has 4 hidden layers of dimension 1024. The Wrapping module uses a hidden dimension of 64. At the decoder, the UnWrapping module uses a hidden dimension of 128.

II-D. Subdivision Enhanced WrappingNet

In the subdivision-enhanced version of WrappingNet, the primary difference is that Loop pooling and subdivision is applied at the encoder and decoder in order to incorporate hierarchical feature extraction, as well as perform the matching on a smaller base mesh. The feature extractor (FE) now applies Loop pooling after each of its face convolutions layers, and outputs a feature map on the base mesh, which contains m_b faces. UnWrapping and codeword pooling now takes place on the base mesh. At the decoder, we now have two Wrapping modules. The first stage is the same as before, with no Loop subdivision; this recovers an approximate base mesh. Then, a second Wrapping stage is applied, which interleaves Loop subdivision into the layers[1]. In all experiments, we use 3 levels of subdivision, which is reflected in our architecture; the FE uses 3 Loop pooling layers, and Wrapping 2nd stage uses 3 Loop subdivision (unpooling) layers.

II-E. Model Size and Efficiency

A forward pass during evaluation (without accumulating gradients), consisting of encoding and decoding, using WrappingNet takes ≈ 254 ms per mesh on the Manifold40 test set. This is evaluated on an NVIDIA GeForce RTX 2080 Ti GPU. Shown in Tab. IV, the total number of parameters is ≈ 4.6M, which is on the same order of magnitude with [9, 10].

II-F. Remeshing Details for Generating Subdivision Meshes

In this section, we elaborate on the remeshing methods used for SHREC11 and Manifold40. All remeshing techniques that generate subdivision meshes have the same overall process:

(1) Apply a series of edge collapses to the original mesh in order to generate a low-resolution base mesh.

(2) Subdivide the base mesh L times, with new vertex positions on the midpoints of edges.

(3) Project all vertex positions onto the original mesh to approximately recover the shape at subdivision level L.

The primary differences among remeshing methods are in steps (1) and (3). For SHREC11, we use the subdivision meshes generated from [6], who use the MAPS [11] method. For Manifold40, we use the original Manifold40 meshes from [6] (not subdivision), and remesh them using the improved version of MAPS from [12]. In this case, for step (1) we use quadric error simplification [13] to simplify down to a base mesh containing approximately 50 vertices, and for step (3)

[1]The subdivided points' vertex positions take the midpoints of the edges they lie on.

we use the parametrization between the subdivided mesh and the original mesh explained in [12].

III. REFERENCES

[1] Rolandos Alexandros Potamias, Stylianos Ploumpis, and Stefanos Zafeiriou, "Neural mesh simplification," in *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022, pp. 18562–18571.

[2] Rolandos Alexandros Potamias, Giorgos Bouritsas, and Stefanos Zafeiriou, "Revisiting point cloud simplification: A learnable feature preserving approach," in *Computer Vision – ECCV 2022*, Shai Avidan, Gabriel Brostow, Moustapha Cissé, Giovanni Maria Farinella, and Tal Hassner, Eds., Cham, 2022, pp. 586–603, Springer Nature Switzerland.

[3] Álvaro González, "Measurement of areas on a sphere using fibonacci and latitude–longitude lattices," *Mathematical Geosciences*, vol. 42, no. 1, pp. 49–64, 2010.

[4] Gabriel Peyré, Marco Cuturi, et al., "Computational optimal transport: With applications to data science," *Foundations and Trends® in Machine Learning*, vol. 11, no. 5-6, pp. 355–607, 2019.

[5] Harold W. Kuhn, "The Hungarian Method for the Assignment Problem," *Naval Research Logistics Quarterly*, vol. 2, no. 1–2, pp. 83–97, March 1955.

[6] Shi-Min Hu, Zheng-Ning Liu, Meng-Hao Guo, Jun-Xiong Cai, Jiahui Huang, Tai-Jiang Mu, and Ralph R Martin, "Subdivision-based mesh convolution networks," *ACM Transactions on Graphics (TOG)*, vol. 41, no. 3, pp. 1–16, 2022.

[7] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al., "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.

[8] Matthias Fey and Jan E. Lenssen, "Fast graph representation learning with PyTorch Geometric," in *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[9] Jiahao Pang, Duanshun Li, and Dong Tian, "TearingNet: Point cloud autoencoder to learn topology-friendly representations," in *IEEE Conference on Computer Vision and Pattern Recognition*, Nashville, Tennessee, USA, 2021, pp. 7453–7462, IEEE.

[10] Yaoqing Yang, Chen Feng, Yiru Shen, and Dong Tian, "FoldingNet: Point cloud auto-encoder via deep grid deformation," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 206–215.

[11] A. W. F. Lee, W. Sweldens, P. Schröder, L. Cowsar, and D. Dobkin, "Maps: Multiresolution adaptive parameterization of surfaces," *Computer Graphics Proceedings (SIGGRAPH 98)*, pp. 95–104, 1998.

[12] Hsueh-Ti Derek Liu, Vladimir G. Kim, Siddhartha Chaudhuri, Noam Aigerman, and Alec Jacobson, "Neural subdivision," *ACM Trans. Graph.*, vol. 39, no. 4, aug 2020.

[13] Michael Garland and Paul S. Heckbert, "Surface simplification using quadric error metrics," in *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, USA, 1997, SIGGRAPH '97, p. 209–216, ACM Press/Addison-Wesley Publishing Co.

Table IV: Layers in each module in subdivision-enhanced WrappingNet (with 3 levels of subdivision). Shape column for each layer contains (# input faces, # input channels, # output channels).

Layer	Shape	# Param.
FaceConv w/ ReLU	(n_f, 7, 128)	3684
SubdivPool	(n_f, 128, 128)	0
FaceConv w/ ReLU	(n_f/4, 128, 128)	65664
SubdivPool	(n_f/4, 128, 128)	0
FaceConv w/ ReLU	(n_f/16, 128, 128)	65664
SubdivPool	(n_f/16, 128, 128)	0
FaceConv w/ ReLU	(n_f/64, 128, 1024)	524416

(a) SubdivNet backbone feature extractor in WrappingNet encoder.

Layer	Shape	# Param.
MLP (2-layer)	(n_f/64, 1024, 1024)	2099200
MaxPool	(n_f/64, 1024, 1024)	0
MLP (3-layer)	(n_f/64, 1024, 512)	1050112

(b) Shared MLP with global pooling module in WrappingNet encoder. First MLP's hidden layer dim: 1024; second MLP's hidden layer dim: 512

Layer	Shape	# Param.
FaceConv w/ ReLU	(n_f/64, 7, 64)	576
FaceConv w/ ReLU	(n_f/64, 64, 64)	4224
Face2Node	(n_f/64, 64+9, 64+3)	8899
FaceConv w/ ReLU	(n_f/64, 64, 64)	4224
FaceConv w/ ReLU	(n_f/64, 64, 64)	4224
Face2Node	(n_f/64, 64+9, 64+3)	8899
FaceConv w/ ReLU	(n_f/64, 64, 64)	4224
FaceConv w/ ReLU	(n_f/64, 64, 64)	4224
Face2Node	(n_f/64, 64+9, 3)	4739

(c) UnWrapping module in WrappingNet encoder.

Layer	Shape	# Param.
FaceConv w/ ReLU	(n_f/64, 512+7, 128)	265856
Face2Node	(n_f/64, 128+9, 128+3)	34179
FaceConv w/ ReLU	(n_f/64, 128, 128)	65664
Face2Node	(n_f/64, 128+9, 128+3)	34179
FaceConv w/ ReLU	(n_f/64, 128, 128)	65664
Face2Node	(n_f/64, 128+9, 128+3)	34179

(d) 1st Wrapping module in WrappingNet decoder.

Layer	Shape	# Param.
SubdivUnpool	(n_f/64, 128, 128)	0
FaceConv w/ ReLU	(n_f/16, 128, 128)	65664
Face2Node	(n_f/16, 128+9, 128+3)	34179
SubdivUnpool	(n_f/16, 128, 128)	0
FaceConv w/ ReLU	(n_f/4, 128, 128)	65664
Face2Node	(n_f/4, 128+9, 128+3)	34179
SubdivUnpool	(n_f/4, 128, 128)	0
FaceConv w/ ReLU	(n_f, 128, 128)	65664
Face2Node	(n_f, 128+9, 3)	17667

(e) 2nd Wrapping module in WrappingNet decoder.