Eurographics Symposium on Geometry Processing 2025
M. Attene and S. Sellán
(Guest Editors)

COMPUTER GRAPHICS *forum*
Volume 44 (2025), Number 5

MDNF: Multi-Diffusion-Nets for Neural Fields on Meshes - Supplemental Material

Avigail Cohen Rimon[1], Tal Shnitzer[2], Mirela Ben Chen[1]

[1]Technion - Israel Institute of Technology
[2]Broad Institute of MIT and Harvard

1. Architecture and Hyper-Parameters

1.1. DiffuionNet - Details

This subsection extends the Background subsection 2.1, providing a more detailed review of the practical implementation details involved in the DiffusionNet pipeline.

The DiffusionNet architecture [SACO22] comprises of successive identical DiffusionNet *blocks*. Each block consists of three main stages: propagating information across the domain via a learned diffusion time, evaluating local spatial gradient features to model directional filters and not only radially symmetric filters, and applying multi-layer perceptrons (MLPs) at each point to model pointwise scalar functions of the feature channels.

The diffusion process is described by the heat operator H_t, which acts on an initial distribution u_0 defined on the surface to produce the diffused distribution u_t. This action is expressed as $H_t(u_0) = \exp(-t\Delta)u_0$, where exp represents the operator exponential, and Δ is the Laplace-Beltrami operator. To discretize the diffusion operator Δ, it is replaced by the Laplace matrix $\mathbf{L} \in \mathbb{R}^{n\times n}$ and mass matrix $\mathbf{M} \in \mathbb{R}^{n\times n}$, where n is the number of vertices or points. For triangle meshes, the authors employ the cotan-Laplace operator for $\mathbf{L}$ and define M as the diagonal matrix of areas associated with each vertex.

Given the feature channel $u \in \mathbb{R}^n$, the authors outline two approaches to evaluate the diffusion layer $h_t(u)$. The implicit method:

$$h_t(u) := (\mathbf{M} + t\mathbf{L})^{-1}\mathbf{M}u \tag{1}$$

and the spectral method:

$$h_t(u) := \Phi \begin{bmatrix} e^{-\lambda_1 t} \\ e^{-\lambda_2 t} \\ \vdots \end{bmatrix} \odot (\Phi^T \mathbf{M}u) \tag{2}$$

where $\odot$ denotes the Hadamard (elementwise) product. The matrices $\Phi \in \mathbb{R}^{n\times k}$, $\Lambda \in \mathbb{R}^{k\times k}$ are the matrices of first k eigenvectors and eigenvalues of the generalized eigenvalue problem $L\Phi = \Lambda M\Phi$ where Λ is the diagonal matrix with diagonal elements $[\lambda_1, \ldots, \lambda_k]$. In our work we only use the spectral method.

After diffusion, given a collection of D scalar feature channels, the spatial gradient features are obtained by first computing per-vertex gradients $\mathbf{z}_u = \mathbf{G}u \in \mathbb{C}^n$ for each channel u, where $\mathbf{G} \in \mathbb{C}^{n\times n}$ is the sparse gradient operator matrix, see [SACO22] for the exact definition. Next, the local gradients of all channels are stacked to form $\mathbf{w}_v \in \mathbb{C}^D$ for each vertex v, and inner products are computed as:

$$\mathbf{g}_v = \tanh(\mathrm{Re}(\overline{\mathbf{w}}_v \odot \mathbf{A}\mathbf{w}_v)) \tag{3}$$

where $\mathbf{A} \in \mathbb{C}^{D\times D}$ is a complex learned matrix enabling a rich space of direction-dependent filters, and $\bar{\cdot}$ denotes the complex conjugate. These diffusion and gradient features are then passed through a point-wise MLP layer to complete one DiffusionNet block. See more details in [SACO22].

1.2. Model Hyperparameters in the Code

The hyper-parameters in our architecture are detailed below.

- **k_eig**: Total number of Laplacian eigenpairs used for diffusion, corresponds to k_{eig} in the paper's notation.
- **n_block**: Number of DiffusionNet blocks in each DiffusionNet component.
- **diffusion_hidden**: Dimension of the linear layer in the MLPs of DiffusionNet components.
- **diffusion_out**: Output dimension of DiffusionNet components, denoted by F in the paper.
- **base_diffusion**: Initialization value for diffusion times in the first-level DiffusionNet component, denoted as t_{base} in the paper.
- **exp_diffusion**: Exponential factor for initializing diffusion times, denoted as t_{exp} in the paper.

Fourier Feature Mapping:

- **base_sigma**: Base value for initializing standard deviation for Fourier mapping at the first resolution level, denoted as σ_{base} in the paper.
- **exp_sigma**: Exponential factor for initializing the standard deviation of Fourier mappings, denoted as σ_{exp} in the paper.

Sine-Activated MLP:

- **n_layer_net**: Number of resolution levels plus one, equal to $N+1$ in the paper's notations.

- **hidden_net**: Dimension of linear layers, denoted as m in the paper.
- **siren_w0**: Frequency parameter of sine layers, denoted as α_i in the paper. Although it could vary across the different linear layers, in our experiments, we use the same value across all layers.

ReLU-activated MLP:

- **n_layer_back**: Number of linear layers.
- **hidden_back**: Dimension of linear layers.
- **backbone_activation**: Activation function, default is ReLU.

We note that before inputting the mesh into the network, its vertices are normalized and centered.

An important note is that we carefully tune parameters to optimize the results for each experiment and each model. This ensures a fair comparison and highlights the maximal representational potential of each method, even when the improvement is marginal (e.g., reducing the error from approximately 10^{-7} to 10^{-13}). In general, the main parameters we tune per experiment are: **base_diffusion**, **exp_diffusion**, **base_sigma**, **exp_sigma**, **n_layer_net**, and **siren_w0**. Below, we provide some guidelines for setting these parameters:

- **t_base**: As noted in the paper, we typically set this to the squared mean edge length of the mesh, usually in the range of 10^{-4} to 10^{-3}.
- **t_exp**: Depends on the number of resolution levels and generally falls within the range of [0.5, 0.9].
- **sigma_base**: Typically within the range of [1, 10].
- **sigma_exp**: Generally in the range of [1.2, 2].
- **N**: The number of layers depends on the complexity of the learned signal. We commonly start with $N = 2$ or $N = 3$ and increase it if needed.
- **siren_w0**: The choice of this parameter depends on the complexity of the learned signal. The model is relatively robust to small variations in **siren_w0**, with significant changes observed only for large differences (e.g., between 10 and 100). The typical range is [10, 200]. In our experiments, we use the same α for all levels.

Based on our experience, the training process exhibits robustness to moderate changes in most of these parameters.

1.3. Model Complexity

The number of parameters across the compared models is generally of the same order of magnitude, ensuring a fair basis for comparison. While model complexity can influence performance, our primary focus is on evaluating the potential of different architectural designs rather than the impact of model size.

In certain cases, we increased the number of parameters to enable fair comparisons, ensuring that performance differences are attributed to architectural improvements rather than variations in model capacity. For example, in the case of the Lion model used in the UV learning experiment (Section 4.2.2 in the paper), the number of parameters used for the diffusion-net baseline is much higher than in our models.

The computational efficiency of our models remains practical: inference time is less than one second, while training time ranges from 10 minutes to several hours, depending on the mesh size and experimental setup.

Tables 1, 2, 3, and 4 provide details regarding the number of parameters and runtime performance for all models employed in our Experimental Results section, corresponding to Sections 4.1, 4.2.1, 4.2.2, and 4.3, respectively.

Model	# Parameters	Training Time	Inference Time
DiffusionNet	4,599,555	2h 13m 12s	1.05s
One-Level	1,928,773	1h 21m 57s	0.71s
N-Level	5,024,333	2h 35m 6s	0.92s

Table 1: *Performance and model complexity comparison of models corresponding to the Synthetic Example experiment (Section 4.1) on a Chinese lion mesh with 50K vertices.*

Model	# Parameters	Training Time	Inference Time
DiffusionNet	3,679,746	22m 24s	0.56s
One-Level	5,147,140	24m 45s	0.73s
N-Level	5,024,268	30m 40s	0.60s

Table 2: *Performance and model complexity comparison of models corresponding to the Discontinuity of Mesh and UV Coordinates experiment (Section 4.2.1) on a kangaroo mesh with 10K vertices.*

Model	# Parameters	Training Time	Inference Time
DiffusionNet	11,495,938	16m 27s	0.66s
One-Level	3,796,740	8m 37s	0.58s
N-Level	4,020,490	9m 43s	0.58s

Table 3: *Performance and model complexity comparison of models corresponding to the Exponential Scale Variations experiment (Section 4.2.2) on a lion mesh with 8K vertices.*

2. Measuring Function Smoothness (Section 3)

Let $L \in \mathbb{R}^{n\times n}$ be the cotan-Laplace operator and $M \in \mathbb{R}^{n\times n}$ be the mass matrix, where n denotes the number of mesh vertices. Inspired by the strong association and extensive use of the mesh Laplacian for smoothing operations [NISA06], we define the following normalized smoothness measure for a function $x \in \mathbb{R}^n$ defined on the mesh vertices:

$$\frac{\langle Lx, Lx\rangle_M}{\langle x, x\rangle_M} \tag{4}$$

where $\langle\cdot,\cdot\rangle_{\mathbf{M}}$ denotes the inner product with respect to $\mathbf{M}$. The normalization ensures a scale-invariant measure for the function x.

3. RGB Neural Field (Sections 3 and 4.1)

To constrain the output neural field to valid RGB values, we add either a clamp layer or a sigmoid activation at the end of the ReLU-activated MLP. For the illustrative example (Section 3), we employed a standard clamp layer to restrict the values between [0, 1], simplifying the architecture for explanation purposes. Since the

Model	# Parameters	Training Time	Inference Time
DiffusionNet	2,760,451	3h 9m 52s	0.68s
One-Level	1,471,301	3h 42m 9s	0.63s
N-Level	2,830,165	5h 44m 6s	0.94s

Table 4: *Performance and model complexity comparison of models corresponding to the Mesh Generalization experiment (Section 4.3) on ogre meshes with 20K-33K vertices.*

DiffusionNet with a clamp layer resulted in poor performance, and switching to sigmoid activation resolved this issue, in the first experiment (Section 4.1), we adopted sigmoid activation for all models to ensure a fair comparison.

3.1. Chinese Dragon (Section 3)

In this example, we configure a network with three resolution levels ($N = 3$), each featuring a DiffusionNet component with two DiffusionNet blocks of width 128. We set t_{base} to 0.00025, t_{exp} to 0.5, σ_{base} to 5, σ_{exp} to 1.3, and α_i to 105 for all resolution levels. The sine-activated MLP has a width of 128, while the ReLU-activated MLP features a width of 64 and a depth of 2. The network is trained over 10K epochs.

3.2. Chinese Lion (Section 4.1)

To generate the highest frequency function (corresponding to the Blue vertices group) in this example, we produce Perlin noise with the *shape* parameter set to

$$\text{int}(2/\text{mean\_edge\_length}) \times 3$$

and the *res* parameter set to

$$\text{int}(\text{int}(2/\text{mean\_edge\_length})/6) \times 3$$

where $\times$ denotes concatenation, mean_edge_length represents the mean edge length in the mesh, and $\text{int}(\cdot)$ denotes conversion to an integer. In the following, we detail the configuration settings for each result presented in Section 4.1. All models are trained for 10K epochs.

DiffusionNet Model We define a DiffusionNet network consisting of 10 DiffusionNet blocks, each featuring linear layers with a width of 256.

One-Level We define a network with a single resolution level, featuring a DiffusionNet component that includes 4 DiffusionNet blocks, each with linear layers of width 256. We set t_{base} to 0.00016, σ_{base} to 5, and α_1 to 185. The sine-activated MLP has a width of 256, while the ReLU-activated MLP has a width of 64 and a depth of 2.

N-Level We define a network with 5 resolution levels, each featuring a DiffusionNet component with 2 DiffusionNet blocks of width 256. We set t_{base} to 0.00016 and t_{exp} to 0.7, σ_{base} to 5, σ_{exp} to 1.5, and α_i to 165 for all resolution levels. The sine-activated MLP has a width of 256, while the ReLU-activated MLP has a width of 64 and a depth of 2.

4. UV Neural Field (Section 4.2)

4.1. Kangaroo (Section 4.2.1)

We train all models for 5K epochs.

DiffusionNet Model We define a DiffusionNet network consisting of 8 DiffusionNet blocks, each featuring linear layers with a width of 256.

One-Leve Model We define a network with a single resolution level, featuring a DiffusionNet component that includes 11 DiffusionNet blocks, each with linear layers of width 256. We set t_{base} to 0.002, σ_{base} to 10, and α_1 to 105. The sine-activated MLP has a width of 256, while the ReLU-activated MLP has a width of 64 and a depth of 2.

N-Level Model We define a network with 5 resolution levels, each including a DiffusionNet component with 2 DiffusionNet blocks of width 256. We set t_{base} to 0.002 and t_{exp} to 0.8, σ_{base} to 10, σ_{exp} to 1.2, and α_i to 220 for all resolution levels. The sine-activated MLP has a width of 256, while the ReLU-activated MLP has a width of 64 and a depth of 2.

4.2. Lion (Section 4.2.2)

We train all models for 2K epochs.

DiffusionNet Model We define a DiffusionNet network consisting of 25 DiffusionNet blocks, each featuring linear layers with a width of 256.

One-Level Model We define a network with a single resolution level, featuring a DiffusionNet component that includes 8 DiffusionNet blocks, each with linear layers of width 256. We set time t_{base} to 0.002, σ_{base} to 5, and α_1 to 220. The sine-activated MLP has a width of 256, while the ReLU-activated MLP has a width of 128 and a depth of 2.

N-Level Model We define a network with 4 resolution levels, each including a DiffusionNet component with 2 DiffusionNet blocks of width 256. We set t_{base} to 0.002, t_{exp} to 0.8, σ_{base} to 5, σ_{exp} to 1.2, and α_i to 220 for all resolution levels. The sine-activated MLP has a width of 256, while the ReLU-activated MLP has a width of 64 and a depth of 2.

4.3. Dennis - Additional Results

Here, we present an additional result from applying our network to learn the UV coordinates of the *Dennis* mesh, which consists of 15K vertices and 30K faces, and features discontinuous coordinates. The texture image can be seen in Figure 1. As noted in the paper, we also evaluated the NFFB model [WJY23] on this challenging example. We note that NFFB demonstrates superior performance compared to methods such as InstantNGP [MESK22], SIREN [SMB*20], and ModSine [MGB*21]. We adapted the NFFB implementation with minor modifications to their 2D image fitting experiment, enabling it to accept 3D coordinates as input and produce 2D UV coordinates as output instead of RGB values.

Figure 1: ***UV Learning - Additional Results.*** *The texture image of the Dennis mesh exhibits discontinuous UV coordinates, making it a challenging test case.*

The model was trained on the mesh vertices without sampling additional points in the space.

Figure 2 compares the ground truth (GT) texture with the textures learned by our N-Level model and NFFB. In each subfigure (a), (b), and (c), we present, from left to right: the GT texture, the result from our model, and the result from NFFB. Subfigure (a) shows the original mesh texture, while (b) and (c) depict the mesh in two different poses with a standard grid texture to emphasize discontinuities. NFFB exhibits noticeable artifacts near UV discontinuities, and in addition exhibits noise even in continuous regions (e.g. the foot and the back of the head). In contrast, our model accurately captures the corresponding neural field, achieving an exceptionally low error of 8×10^{-13}.

We train the model for 5K epochs. We define a network with 4 resolution levels, each including a DiffusionNet component with 2 DiffusionNet blocks of width 256. We set t_{base} to 0.002, t_{exp} to 0.8, σ_{base} to 5, σ_{exp} to 1.2, and α_i to 200 for all resolution levels. The sine-activated MLP has a width of 256, while the ReLU-activated MLP has a width of 64 and a depth of 2.

5. Vertex Normals Neural Field (Section 4.3)

To generate the dataset for this experiment, we used the Loop subdivision algorithm implemented by MeshLab with default parameters. We train all models for 10K epochs.

DiffusionNet Model We define a DiffusionNet network consisting of 6 DiffusionNet blocks, each featuring linear layers with a width of 256.

One-Level Model We define a network with a single resolution level, featuring a DiffusionNet component that includes 12 DiffusionNet blocks, each with linear layers of width 128. We set t_{base} to 0.1, σ_{base} to 5, and α_1 to 75. The sine-activated MLP has a width of 256, while the ReLU-activated MLP has a width of 64 and a depth of 2.

N-Level Model We define a network with 9 resolution levels, each including a DiffusionNet component with 2 DiffusionNet blocks of width 256. We set t_{base} to 0.1, t_{exp} to 0.55, σ_{base} to 5, σ_{exp} to 1.3, and α_i to 100 for all resolution levels. The sine-activated MLP has a width of 256, while the ReLU-activated MLP has a width of 64 and a depth of 2.

6. Illustrative Application (Section 5)

6.1. Prompt "A 3D rendering of a ninja in unreal engine" (Figure 13a in the paper)

In this experiment, we maintain the original parameters specified in [MBOL*22] while exploring various parameter configurations for our network module. In the following, we detail the parameters used for each "Modified Architecture" result, presented from left to right in the paper figure 13a:

- **n_block** = 4, **n_layer_net** = 3, **hidden_net** = 256, **base_diffusion** = 0.0001, **exp_sigma** = 4, **siren_w0** = 30
- **n_block** = 2, **n_layer_net** = 3, **hidden_net** = 128, **base_diffusion** = 0.0001, **exp_sigma** = 3, **siren_w0** = 50
- **n_block** = 2, **n_layer_net** = 4, **hidden_net** = 128, **base_diffusion** = 0.0001, **exp_sigma** = 10, **siren_w0** = 50
- **n_block** = 2, **n_layer_net** = 3, **hidden_net** = 128, **base_diffusion** = 0.0001, **exp_sigma** = 3, **siren_w0** = 50

Additionally, the following parameters remain fixed across all experiments: **k_eig** = 500, **diffusion_out** = 2, **exp_diffusion** = 0.7, **base_sigma** – 3.0, **diffusion_hidden** = 128.

6.2. Prompt "a 3D rendering of the Hulk in unreal engine" (Figure 13b in the paper)

For this experiment we slightly modified the original architecture setting. Inspired by the Nerf experiment in the original NFFB paper [WJY23], instead of using the output of our MDNF network directly as the input to the two other MLPs (predicting color and displacement), we feed the MLP network predicting the displacement with our MDNF output and predict both displacement and low-dimensional features vector which is then fed as the input to the MLP predicting color. We did the same for the original [MBOL*22] architecture to make a fair comparison. Figure 3 shows the results obtained with the original architecture with and without this modification. The parameters for each "Modified Architecture" results presented from left to right in the paper figure 13b:

- **n_block** = 3, **n_layer_net** = 6, **hidden_net** = 128, **base_diffusion** = 0.001, **base_sigma** = 8, **exp_sigma** = 1.5, **siren_w0** = 15
- **n_block** = 3, **n_layer_net** = 6, **hidden_net** = 128, **base_diffusion** = 0.001, **base_sigma** = 8, **exp_sigma** = 3, **siren_w0** = 15
- **n_block** = 4, **n_layer_net** = 3, **hidden_net** = 256, **base_diffusion** = 0.0001, **base_sigma** = 3, **exp_sigma** = 4, **siren_w0** = 30

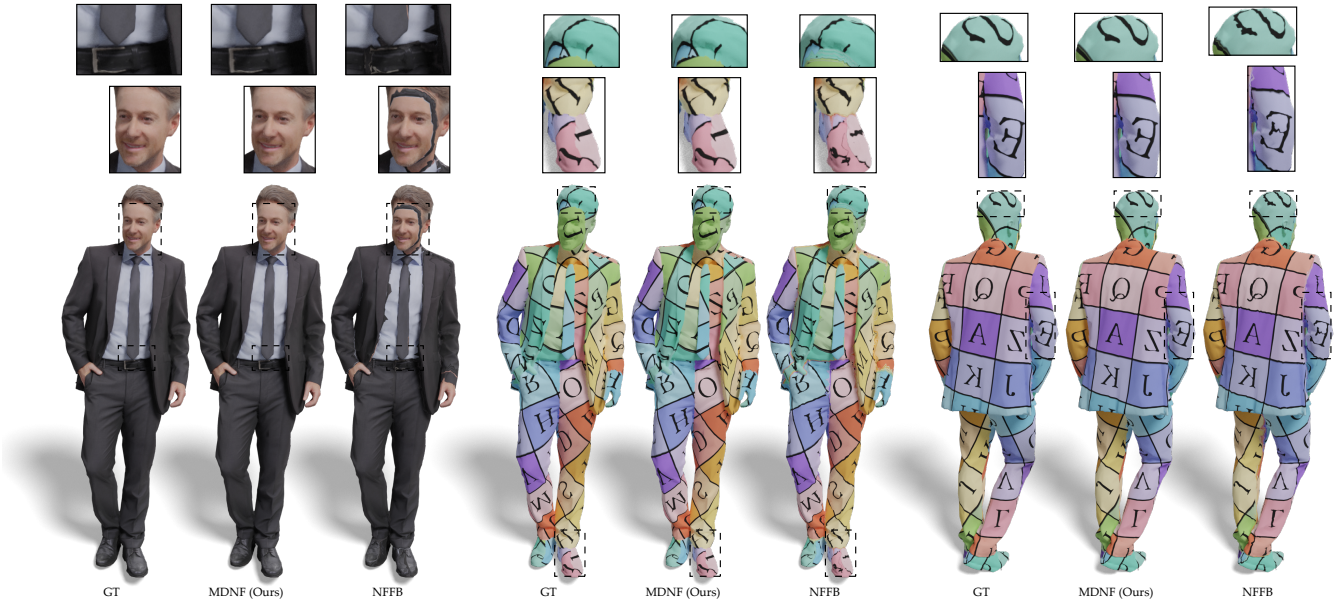

Figure 2: ***UV Learning - Additional Results.*** *Comparison between ground truth (GT) UV coordinates and those learned by our N-Level model and NFFB [WJY23]. (a) Original texture mapping. (b, c) Two poses with grid texture pattern highlighting mapping discontinuities. Each triplet shows, from left to right: GT, our N-Level model results, and NFFB results. Note that our method better preserves the mapping accuracy across discontinuous regions.*

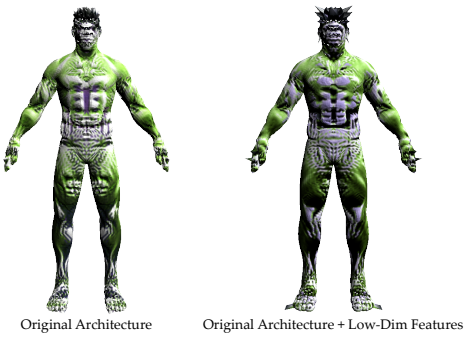

Figure 3: ***Illustrative Application.*** *The textured mesh generated for the prompt* "a 3D rendering of the Hulk in unreal engine" *(Figure 13b in the paper) by (left) original architecture and parameters from [MBOL*22], (right) same architecture and parameters where the input to the color-predicting MLP uses low-dimensional features predicted by the displacement-predicting MLP and the color MLP depth is increased by 1.*

- **n_block** = 4, **n_layer_net** = 3, **hidden_net** = 256, **base_diffusion** = 0.0001, **base_sigma** = 3, **exp_sigma** = 4, **siren_w0** = 50

Additionally, the following parameters remain fixed across all experiments: **k_eig** = 500, **diffusion_out** = 2, **exp_diffusion** = 0.7, **diffusion_hidden** = 128. The parameters changed between experiments which belong to the original architecture are:

- **width=64**, **normratio=0.5**
- **width=64**, **normratio=0.5**
- **width=256**, **normratio=0.1**
- **width=256**, **normratio=0.1**

We note that due to the inherent stochasticity of CLIP's architecture, the results may vary slightly when run in different environments.

References

[MBOL*22] MICHEL O., BAR-ON R., LIU R., BENAIM S., HANOCKA R.: Text2mesh: Text-driven neural stylization for meshes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2022), pp. 13492–13502. 4, 5

[MESK22] MÜLLER T., EVANS A., SCHIED C., KELLER A.: Instant neural graphics primitives with a multiresolution hash encoding. *ACM transactions on graphics (TOG) 41*, 4 (2022), 1–15. 3

[MGB*21] MEHTA I., GHARBI M., BARNES C., SHECHTMAN E., RAMAMOORTHI R., CHANDRAKER M.: Modulated periodic activations for generalizable local functional representations. In *Proceedings of the IEEE/CVF International Conference on Computer Vision* (2021), pp. 14214–14223. 3

[NISA06] NEALEN A., IGARASHI T., SORKINE O., ALEXA M.: Laplacian mesh optimization. In *Proceedings of the 4th international conference on Computer graphics and interactive techniques in Australasia and Southeast Asia* (2006), pp. 381–389. 2

[SACO22] SHARP N., ATTAIKI S., CRANE K., OVSJANIKOV M.: Diffusionnet: Discretization agnostic learning on surfaces. *ACM Transactions on Graphics (TOG) 41*, 3 (2022), 1–16. 1

[SMB*20] SITZMANN V., MARTEL J., BERGMAN A., LINDELL D.,

WETZSTEIN G.: Implicit neural representations with periodic activation functions. *Advances in neural information processing systems 33* (2020), 7462–7473. 3

[WJY23] WU Z., JIN Y., YI K. M.: Neural fourier filter bank. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2023), pp. 14153–14163. 3, 4, 5