Supplementary Material: Mesh Processing Non-Meshes via Neural Displacement Fields

1. Implementation

1.1. Training

As a preprocessing step, we uniformly scale the non-mesh surface Ω and the base mesh Ω' such that Ω' is bounded by a cube $[-1.5, 1.5]^3$. Before training for the main loss, we initialize the neural networks using Kaiming initialization [HZRS15]. Next, we train g_θ and g_ϕ so that they represent the identity. For both models, we sample $q = 32,768$ points $\mathbf{z}_1, ..., \mathbf{z}_q \in \Omega'^3$ and train the loss

$$\underset{\theta}{\operatorname{argmin}} \sum_{i=1}^{q} \|g_\theta(\mathbf{z}_i) - \mathbf{z}_i\|_2^2 \tag{1}$$

by running Adam [KB17] for 1,000 epochs (θ and ϕ are interchangeable here). After initialization, we train our loss by running Adam for 10,000 epochs. We set the sample number $s = 32,768$ and resample every ten epochs.

1.2. Code Base

We implemented our training code in PYTHON using PYTORCH [PGC*17] for standard operations on MLPs and auto differentiation, and LIBIGL [JP*18] and GPYTOOLBOX [SS*23] for standard geometry processing routines.

We implemented our remeshing algorithm using GEOMETRY CENTRAL [SC*19] for storing the intrinsic triangulation in the integer coordinate data structure [GSC21].

2. Baseline Setup

We discuss the baseline setups of the experiments we show in the main text. All baseline experiments were run on a desktop computer with an NVIDIA GeForce RTX 3070 and 8GB of RAM.

2.1. Implementation

We implemented geodesic computation and Laplacian eigenmode analysis on C++ using GEOMETRY CENTRAL [SC*19] and called it from PYTHON using PYBIND11 [JRM16]. For geodesic computation we used the MMP algorithm [MMP87].

In all of our experiments, we computed the geodesic distance from the closest point from $[1,1,1]^T$ and $[-1,-1,-1]^T$ to the mesh, and the 20 smallest eigenvalues of the Laplace-Beltrami operator. We evaluate the L^1 error of geodesic distance by querying the closest point from each vertex on the ground truth mesh to the extracted mesh, and then applying barycentric interpolation to get the value on that closest point. We used the same code in the core algorithm that computes the geodesic distance or the spectral decomposition.

The Laplace-Beltrami operator is positive semidefinite only if the triangulation is manifold and Delaunay [WMKG07]. Thus, to avoid numerical instability for meshes that do not satisfy them, we shift the eigenvalues with a small value $\sigma = 10^{-3}$.

Also, many methods do not guarantee topological identity to the ground truth; for example, Marching Cubes can easily create outliers that are not connected to other parts of the surface, leading the geodesic distances of those parts to infinity. Thus, we extract the triangles that are singly connected before we run the tasks.

2.2. Isosurfacing Methods (Figure 8)

We compared our method to Neural Dual Contouring (NDC) [CZ21], Occupancy-Based Dual Contouring (ODC) [HS24], and Marching Cubes (MC) [LC87] to measure the runtime vs. error. All the runtimes include (1) the query time of the neural network, (2) the time required for mesh extraction, (3) the time to run geodesic computation or eigenmode analysis, and (4) the time to query the solution on the ground truth vertex positions. We observed that the runtime of (4) was mostly in milliseconds and was fairly negligible.

For NDC and ODC, we used the official code release by the authors, while for MC, we used a GPU implementation on WARP [Mac22]. For NDC we used NDCx, a version that gives higher reconstruction accuracy. We ran each of these isosurfacing methods with a grid size of $h = 0.01, 0.02, ..., 0.1$, where the object fits in a bounding box of $[-1.5, 1.5]^3$. We note that NDC timed out for all the inputs with $h = 0.01$, and we simply omitted those samples from the experiment.

For our method, we extracted the mesh with a target edge length $h = 0.01, 0.014, 0.025, 0.035, 0.05, 0.07, 0.1, 0.14, 0.2$.

We also ran additional meshing after MC, using either intrinsic Delaunay refinement (iDR) [SSC19] or isotropic remeshing [BK04]. For iDR we used the code provided by the authors in GEOMETRY CENTRAL, while for isotropic remeshing, we used the C++ binding implementation from GPYTOOLBOX.

In this experiment, we used NeuS [WLL*21] using the implementation of SDFSTUDIO [YCA*22] trained on the DTU dataset

[AJV*16]. To get the ground truth, we extracted the mesh with Marching Cubes [LC87] on a 300^3 grid and then applied isotropic remeshing [BK04] with $h = 0.005$.

2.2.1. Fast winding numbers

We also used the same setup to compute the isosurface of the generalized winding number (GWN) [BDS*18], which we used their implementation on LIBIGL to query the field.

All the abovementioned methods except NDC were runnable. NDC only accepts signed distance functions, which makes our measurement infeasible. Although NDC does provide a point cloud version (UNDC), we found that UNDC causes out of memory on a 10K point cloud, while their `-noisypc` option that divides point clouds into batches takes minutes to finish.

2.3. Mesh Compression Methods (Figure 10)

We compared the error vs. file size to Spectral Mesh Simplification (SMS) [LLT*20], a decimation algorithm that can preserve the spectral property well, Neural Geometry Fields for Meshes (NGF) [SLR24], a state-of-the-art mesh compression method, and Micromeshes [MMT23], a mesh compression technique based on displacement maps. For SMS, we measured the theoretical optimal file size of a mesh given by $(32 \times 3 \times |V| + 3 \times |F| \times \log_2 |V|)/8$ bytes, where $|V|$ is the number of vertices and $|F|$ is the number of faces. For NGF and Micromeshes, we measured the size of the binary file they produce, while for our neural displacement field, we measured the file size of our weights, feature vectors, and base mesh saved in a torch script file.

For both of the baseline methods, we used the source code provided by the authors. For SMS, we set the number of eigenvalues to preserve to 20 and changed the number of target vertices, and for NGF and Micromeshes, we changed the number of vertices of the base mesh.

For our method, we changed the target face number $N_{\Omega'} = 2000, 2800, 4000, ..., 32000$ with d proportional to the reciprocal of $N_{\Omega'}$, and plotted the smallest error found by changing h.

In this experiment, we randomly selected 10 models from THINGI10K [ZJ16] that have a file size of over 1MB and the whole mesh is singly connected. For all four methods, we treated the mesh as a pure SDF and did not draw any information other than the signed distance to the mesh. For NGF and Micromeshes, we extracted the mesh with Marching Cubes [LC87] on a 400^3 grid and gave this mesh as an input. For SMS, we additionally applied isotropic remeshing [BK04] with $h = 0.005$ to have the spectral properties more accurate, and then gave this mesh as input.

2.4. 3D Plot (supplemental HTML file)

In order to perform a comparison with all of the mesh extraction and mesh compresison methods, we plot all the three axes (data file size, runtime on the client side, error) on a single 3D plot. In the plot, we show that our method breaks the tradeoff between the two classes of methods, and leads to a interesting new region of the plot. Please open the HTML file for details.

In the plot, we used the same setup as Figure 8 where we compared to mesh extraction methods on NeuS trained on the DTU dataset. For additional data points for mesh compression methods, we fed the mesh we treated as the ground truth (300^3 grid Marching Cubes + isotropic remeshing) to SMS. Although mesh extraction methods do not require any additional data transfer, we plotted them on the plane of file size 100KB for clarity, which is a file size that any of the methods including ours did not achieve.

We note that NGF [SLR24] relies on the *Tri to Quad by smart triangle pairing* filter in MESHLAB [CCC*08], but we found out that this procedure is not robust enough on high genus surfaces. We observed that NGF fails to compress meshes on 43 out of 105 test meshes. Thus, in order to keep a fair comparison, we omitted NGF from this plot.

2.5. Comparison with Draco [Goo17] (Figure 19)

We compared our method's error vs. file size to Draco [Goo17], an industry standard mesh compression method. We carried out the marching cubes mesh on a 400^3 grid, varied its resolution by running mesh simplification [GH97] with different target number of faces (32K, 64K, 128K, 256K, 512K), and finally ran Draco on each of them. We compared against two different edge encoding schemes: **Draco-EB** which uses the edge encoding scheme in Edgebreaker [Ros99], and **Draco-MS** which uses a sequential connectivity encoding. For **Draco-EB** we used the default parameter `compression_level=7`, while for **Draco-MS** we have set it to `compression_level=0` to enable the sequential encoding. We note that Draco combined with mesh simplification is also the baseline method for Pentapati et al. [PPB25].

In eigenvalue computation on Draco meshes, we observed that after a certain file size, the error increases instead of decreasing. This is because on smaller file sizes, mesh simplification contributes to removing short edges caused by marching cubes, which would improve the eigenanalysis.

2.6. Learning Scalar Fields (Figure 21)

In this experiment, we learned the 10 eigenfunctions $u_1, ..., u_10 : \Omega \mapsto \mathbb{R}$ into a single MLP u_ξ. This MLP takes the 3D point and the number of the eigenfunction $k < 10$, and outputs the (learned) k-th eigenfunction at that point. The baseline is an MLP that takes four dimensions (xyz and the latent code) as input and has 2 layers, 32 neurons, and 8 positional encoding layers. We compare this baseline with a neural network that uses our intrinsic encoding with feature vector dimensions $d = 2, 4, 8$.

We trained the MLPs using a loss

$$\frac{1}{s} \sum_{k}^{10} \sum_{i=1}^{s} \|u_\xi(\mathbf{p}_i, k) - u_k(\mathbf{p}_i)\|_2^2, \tag{2}$$

with $s = 3276$ samples, and iterated Adam [KB17] for 40,000 epochs with a learning rate 10^{-3}.

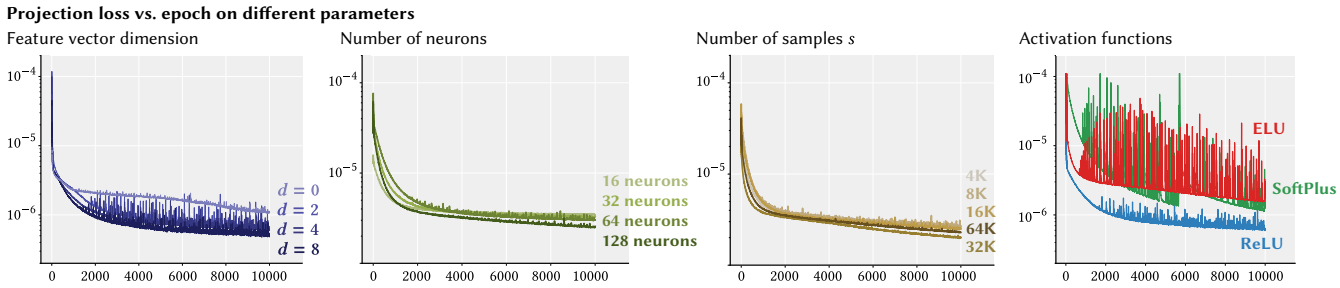


Figure 1: *We show three ablations on (left) the feature vector dimension d, (center left) the number of neurons, (center right) number of samples s, and (right) activation functions.*

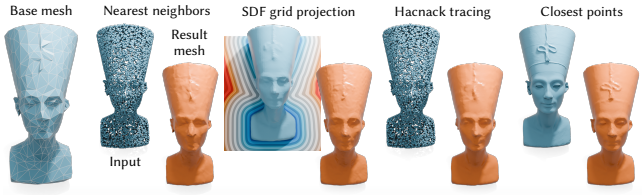


Figure 2: *Having a better approximate of the closest point to the surface leads to higher accuracy.*

3. Ablations and Measurements

3.1. Neural network settings

We show in Figure 1 the ablation studies on three parameters of the neural network: the feature vector dimension d, the number of neurons, the number of samples s, and activation functions. Given the results, we concluded that $d = 8$ and ReLU activation functions stabilize the training process the most, while the number of neurons do not make a large difference in the outcome.

3.2. Cycle Consistency

Our method encourages cycle consistency only by a soft penalty, and hence it cannot be strictly guaranteed. However, existing neural network structures that guarantee cycle consistency are not expressive and fast enough for our purpose. We report that the average $\mathcal{L}_C(\theta,\phi)$ of the final epoch when trained on 34 models from THINGI10K [ZJ16] was 4.8×10^{-7}, which was sufficiently small enough for our purpose.

3.3. Projection Operators

In Section 4.2 in the main text, we discussed different projection operators that approximates the closest point query. We observed that the visual quality of the extracted mesh highly depends on the accuracy of the closest point query (see Figure 2). In other words, the closer the projection operator becomes to give the closest point, the more accurate the final outcome.

Table 1: *We report the training runtime on different representations.*

Surface representation	Runtime
Mesh treated as an SDF	18 minutes
NeuS	32 minutes
Oriented point cloud	7 hours 12 minutes

Table 2: *We report the hyperparameters we used on different representations.*

Surface representation	λ_C	λ_P	learning rate
Mesh treated as SDFs	10^2	10^4	1×10^{-3}
Neural implicits / NeuS	10^2	10^4	1×10^{-3}
Non-oriented point clouds	10^5	10^4	1×10^{-3}
Oriented point clouds	10^4	10^4	1×10^{-3}
NeRFs	10^4	10^4	1×10^{-3}
3D Gaussian splats	10^5	10^4	1×10^{-3}

3.4. Runtime

We report the runtime of our training in Table 1. As we used a relatively low-budget desktop computer with NVIDIA GeForce RTX 3070 and only 8GB of RAM, this could potentially be sped up on a stronger one.

3.5. Hyperparameters

The hyperparameters of our training are λ_C, λ_P and the learning rate. We report the choice of our hyperparameters in Table 2.

4. Additional Losses

We provide additional losses that can be added to our vanilla loss during our training.

4.1. Anchor Point Loss

To guide the training of the map in Figure 24 in the main text, we set a pair of anchor points $\mathbf{p} = [\mathbf{p}_1, ..., \mathbf{p}_m] \in \Omega'$ and $\mathbf{q} = [\mathbf{q}_1, ..., \mathbf{q}_m] \in$

Ω such that $g_\theta(\mathbf{p}_i)$ matches $\mathbf{q}_i$. This can be done by adding a loss

$$\lambda_A \sum_{i=1}^{m} \|g_\theta(\mathbf{p}_i) - g_\theta(\mathbf{q}_i)\|^2 \tag{3}$$

which we set $\lambda_A = 0.1\lambda_P$.

4.2. Normal Alignment Loss

In order to improve the visual quality of the mapped surface, one may encourage the normal direction $\mathbf{n}_\theta(g_\theta(\mathbf{x}_i))$ on $g_\theta(\mathbf{x}_i) \in \Omega_\theta$ to align with the normal direction $\mathbf{n}_\Omega(\mathbf{y}_\Omega(\mathbf{x}_i))$ on the projected point $\mathbf{y}_\Omega(\mathbf{x}_i) = \mathcal{P}(g_\theta(\mathbf{x}), \mathbf{n}_\theta(g_\theta(\mathbf{x}_i))) \in \Omega$ by adding a loss

$$\mathcal{L}_N(\theta) = \frac{1}{s} \sum_{i=1}^{s} \|\mathbf{n}_\theta(g_\theta(\mathbf{x}_i)) - \mathbf{n}_\Omega(\mathbf{y}_\Omega(\mathbf{x}_i))\|^2 . \tag{4}$$

4.3. Local Conformality Loss

In order to obtain a map f with cycle consistency, it must be free of *foldovers*, i.e., points where the local Jacobian $\mathbf{J}_f(\mathbf{x})$ is negative or indefinite [GKK*21]. Also, the map f should not have large anisotropic distortions if the user wants to use our map for, e.g., texture mapping. Thus, the user may wish it to be as conformal as possible.

A typical approach to encourage foldover-free and conformal maps [HG00, Gar00] is to use a barrier-like loss

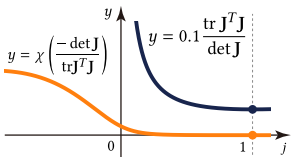
$$\sum_{i=1}^{s} \frac{\operatorname{tr} \mathbf{J}_i^T \mathbf{J}_i}{\det \mathbf{J}_i}, \tag{5}$$

where $\mathbf{J}_i = \mathbf{J}_f(\mathbf{x}_i)$. However, our map is prone to having negative Jacobians during the training, leading this loss to approach infinity. Another tempting approach is to wrap $\det \mathbf{J}_i$ with a function that always gives a positive value [GKK*21], but this still caused instabilities due to orders of magnitudes of larger losses for negative Jacobians.

To solve these issues, we instead use a softer penalty loss

$$\mathcal{L}_F(\theta) = \frac{1}{s} \sum_{i=1}^{s} \chi\left(-\frac{\det \mathbf{J}_i}{\operatorname{tr} \mathbf{J}_i^T \mathbf{J}_i}\right), \tag{6}$$

where $\chi(t) = \frac{1}{10}\log(1 + \exp(10t))$ is the SoftPlus function. The $\mathbf{J}_i$ that gives the global minimum of Eq. 6 also gives the global minimum for Eq. 5; in the inset we show the case where $\mathbf{J} = \begin{bmatrix} 1 & 0 \\ 0 & j \end{bmatrix}$.



References

[AJV*16] AANÆS H., JENSEN R. R., VOGIATZIS G., TOLA E., DAHL A. B.: Large-scale data for multiple-view stereopsis. *International Journal of Computer Vision* (2016), 1–16. 2

[BDS*18] BARILL G., DICKSON N. G., SCHMIDT R., LEVIN D. I. W., JACOBSON A.: Fast winding numbers for soups and clouds. *ACM Trans. Graph. 37*, 4 (jul 2018). URL: https://doi.org/10.1145/3197517.3201337, doi:10.1145/3197517.3201337. 2

[BK04] BOTSCH M., KOBBELT L.: A remeshing approach to multiresolution modeling. In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing* (New York, NY, USA, 2004), SGP '04, Association for Computing Machinery, p. 185–192. URL: https://doi.org/10.1145/1057432.1057457, doi:10.1145/1057432.1057457. 1, 2

[CCC*08] CIGNONI P., CALLIERI M., CORSINI M., DELLEPIANE M., GANOVELLI F., RANZUGLIA G.: MeshLab: an Open-Source Mesh Processing Tool. In *Eurographics Italian Chapter Conference* (2008), Scarano V., Chiara R. D., Erra U., (Eds.), The Eurographics Association. doi:10.2312/LocalChapterEvents/ItalChap/ItalianChapConf2008/129-136. 2

[CZ21] CHEN Z., ZHANG H.: Neural marching cubes. *ACM Trans. Graph. 40*, 6 (Dec. 2021). URL: https://doi.org/10.1145/3478513.3480518, doi:10.1145/3478513.3480518. 1

[Gar00] GARANZHA V.: The barrier method for constructing quasi-isometric grids. *Computational Mathematics and Mathematical Physics 40* (2000), 1617–1637. 4

[GH97] GARLAND M., HECKBERT P. S.: Surface simplification using quadric error metrics. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques* (USA, 1997), SIGGRAPH '97, ACM Press/Addison-Wesley Publishing Co., p. 209–216. URL: https://doi.org/10.1145/258734.258849, doi:10.1145/258734.258849. 2

[GKK*21] GARANZHA V., KAPORIN I., KUDRYAVTSEVA L., PROTAIS F., RAY N., SOKOLOV D.: Foldover-free maps in 50 lines of code. *ACM Trans. Graph. 40*, 4 (jul 2021). URL: https://doi.org/10.1145/3450626.3459847, doi:10.1145/3450626.3459847. 4

[Goo17] GOOGLE: Draco: 3d data compression, 2017. URL: https://github.com/google/draco. 2

[GSC21] GILLESPIE M., SHARP N., CRANE K.: Integer coordinates for intrinsic geometry processing. *ACM Trans. Graph. 40*, 6 (Dec. 2021). URL: https://doi.org/10.1145/3478513.3480522, doi:10.1145/3478513.3480522. 1

[HG00] HORMANN K., GREINER G.: Mips: An efficient global parametrization method. *Curve and Surface Design: Saint-Malo 1999* (2000), 153–162. 4

[HS24] HWANG J., SUNG M.: Occupancy-based dual contouring. In *SIGGRAPH Asia 2024 Conference Papers* (New York, NY, USA, 2024), SA '24, Association for Computing Machinery. URL: https://doi.org/10.1145/3680528.3687581, doi:10.1145/3680528.3687581. 1

[HZRS15] HE K., ZHANG X., REN S., SUN J.: Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *2015 IEEE International Conference on Computer Vision (ICCV)* (2015), pp. 1026–1034. doi:10.1109/ICCV.2015.123. 1

[JP*18] JACOBSON A., PANOZZO D., ET AL.: libigl: A simple C++ geometry processing library, 2018. https://libigl.github.io/. 1

[JRM16] JAKOB W., RHINELANDER J., MOLDOVAN D.: pybind11 — seamless operability between c++11 and python, 2016. https://github.com/pybind/pybind11. 1

[KB17] KINGMA D. P., BA J.: Adam: A method for stochastic optimization, 2017. arXiv:1412.6980. 1, 2

[LC87] LORENSEN W. E., CLINE H. E.: Marching cubes: A high resolution 3d surface construction algorithm. *SIGGRAPH Comput. Graph. 21*, 4 (Aug. 1987), 163–169. URL: https://doi.org/10.1145/37402.37422, doi:10.1145/37402.37422. 1, 2

[LLT*20] LESCOAT T., LIU H.-T. D., THIERY J.-M., JACOBSON A., BOUBEKEUR T., OVSJANIKOV M.: Spectral mesh simplification. *Computer Graphics Forum 39*, 2 (2020), 315–324. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.13932, arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.13932, doi:https://doi.org/10.1111/cgf.13932. 2

[Mac22] MACKLIN M.: Warp: A high-performance python framework for gpu simulation and graphics. https://github.com/nvidia/warp, March 2022. NVIDIA GPU Technology Conference (GTC). 1

[MMP87] MITCHELL J. S. B., MOUNT D. M., PAPADIMITRIOU C. H.: The discrete geodesic problem. *SIAM Journal on Computing 16*, 4 (1987), 647–668. URL: https://doi.org/10.1137/0216045, arXiv:https://doi.org/10.1137/0216045, doi:10.1137/0216045. 1

[MMT23] MAGGIORDOMO A., MORETON H., TARINI M.: Micro-mesh construction. *ACM Trans. Graph. 42*, 4 (July 2023). URL: https://doi.org/10.1145/3592440, doi:10.1145/3592440. 2

[PGC*17] PASZKE A., GROSS S., CHINTALA S., CHANAN G., YANG E., DEVITO Z., LIN Z., DESMAISON A., ANTIGA L., LERER A.: Automatic differentiation in pytorch. 1

[PPB25] PENTAPATI S. K., PHILLIPS G., BOVIK A. C.: Mesh compression with quantized neural displacement fields, 2025. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.70074, arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.70074, doi:https://doi.org/10.1111/cgf.70074. 2

[Ros99] ROSSIGNAC J.: Edgebreaker: connectivity compression for triangle meshes. *IEEE Transactions on Visualization and Computer Graphics 5*, 1 (1999), 47–61. doi:10.1109/2945.764870. 2

[SC*19] SHARP N., CRANE K., ET AL.: Geometrycentral: A modern c++ library of data structures and algorithms for geometry processing. 1

[SLR24] SIVARAM V. E., LI T.-M., RAMAMOORTHI R.: Neural geometry fields for meshes. In *ACM SIGGRAPH 2024 Conference Papers* (New York, NY, USA, 2024), SIGGRAPH '24, Association for Computing Machinery. URL: https://doi.org/10.1145/3641519.3657399, doi:10.1145/3641519.3657399. 2

[SS*23] SELLÁN S., STEIN O., ET AL.: gptyoolbox: A python geometry processing toolbox, 2023. https://gpytoolbox.org/. 1

[SSC19] SHARP N., SOLIMAN Y., CRANE K.: Navigating intrinsic triangulations. *ACM Trans. Graph. 38*, 4 (July 2019). URL: https://doi.org/10.1145/3306346.3322979, doi:10.1145/3306346.3322979. 1

[WLL*21] WANG P., LIU L., LIU Y., THEOBALT C., KOMURA T., WANG W.: Neus: Learning neural implicit surfaces by volume rendering for multi-view reconstruction. *arXiv preprint arXiv:2106.10689* (2021). 1

[WMKG07] WARDETZKY M., MATHUR S., KÄLBERER F., GRINSPUN E.: Discrete laplace operators: no free lunch. In *Proceedings of the Fifth Eurographics Symposium on Geometry Processing* (Goslar, DEU, 2007), SGP '07, Eurographics Association, p. 33–37. 1

[YCA*22] YU Z., CHEN A., ANTIC B., PENG S., BHATTACHARYYA A., NIEMEYER M., TANG S., SATTLER T., GEIGER A.: Sdfstudio: A unified framework for surface reconstruction, 2022. URL: https://github.com/autonomousvision/sdfstudio. 1

[ZJ16] ZHOU Q., JACOBSON A.: Thingi10k: A dataset of 10,000 3d-printing models, 2016. URL: https://arxiv.org/abs/1605.04797, arXiv:1605.04797. 2, 3