NeuMesh: Learning Flexible and Disentangled Neural Mesh-based Implicit Field for Geometry and Texture Editing Supplementary Material

Bangbang Yang[1*], Chong Bao[1*], Junyi Zeng[1], Hujun Bao[1], Yinda Zhang[2†], Zhaopeng Cui[1†], and Guofeng Zhang[1†]

[1] State Key Lab of CAD&CG, Zhejiang University
[2] Google

In this supplementary material, we describe more details of our method, including model architecture in Sec. A, geometry editing in Sec. B.2, and texture editing in Sec. B.3. Besides, we also provide more discussions including limitations in Sec. C and experiment results in Sec. D.

A Model Architecture

The detailed model architecture is shown in Fig.A. To begin with, we first extract a triangle mesh with marching cubes [7] from NeuS's [14] SDF field, where we set the voxel resolution as 256 and the spatial range as $[-1, 1]$. Then, for each query point $\mathbf{x}$, we find K nearest vertices (with $K = 8$ in our experiments) and obtain the interpolated geometry code (32 dimensions), texture code (32 dimensions) and learnable signed distance (scalar) from these vertices. Before feeding into the network, we apply positional encoding to the signed distances (with 8 frequencies), interpolated codes (with 2 frequencies) and viewing directions (with 4 frequencies). The geometry decoder and the radiance decoder are constructed with a MLP of 3 / 4 hidden layers and 256 hidden sizes, and we use SoftPlus / ReLU activation, respectively. During the rendering stage, we first sample 64 coarse points along the ray and adopt a progressive up-sampling strategy from Wang *et al*. [14] to guide the sampling of 64 fine points, which yields 128 samples for each ray. Besides, to accelerate rendering and training, we pre-compute near-far bound for each ray by counting the minimum and maximum distances of ray-to-mesh intersections.

B Implementation Details

B.1 Training Details

As introduced in our main paper, we adopt a distillation and fine-tuning training scheme. Practically, for each object, we first train a teacher model (*i.e.*,

*Bangbang Yang and Chong Bao contributed equally to this work.
†Corresponding authors.

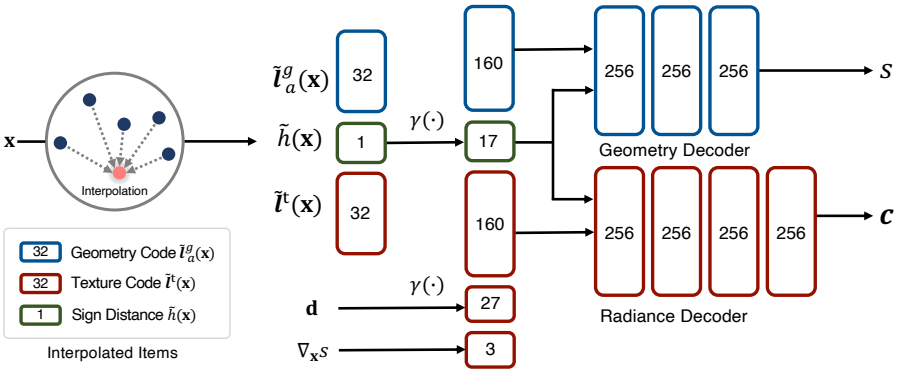


Fig. A. The model architecture of NeuMesh.

NeuS [14]). Then, we optimize codes and decoders with output from the teacher model and the images. During the training process, we use a batch size of 512 rays on a single Nvidia RTX3090-24G GPU, where the queried color and SDF value for each sample point will also be supervised with the output from the teacher model (a.k.a distillation loss in Sec. 3.2 Eq.(4)). We adopt the Adam optimizer with an initial learning rate of 0.0005 and a cosine annealing scheduler with 5000 warm-up steps. The training process takes about 16 hours for each model. Besides, to train on the DTU dataset that contains unbounded background, we follow previous works [16,14] by taking foreground masks into the supervision with a binary cross-entropy loss.

B.2 Details of Geometry Editing

With our mesh-based representation, deforming a neural implicit field is as simple as deforming the corresponding mesh scaffold. The only thing to note is to keep the local consistency of the learnable signed distances (Sec. 3.1), *i.e.*, the interpolated signed distance of the locally deformed or rotated region should keep the same. To achieve this goal, we simply compensate the rotation of the surface normal to the learnable signed indicator $\tilde{\boldsymbol{h}}(\mathbf{x})$, as: $\tilde{\boldsymbol{h}}'(\mathbf{x}) = \tilde{\boldsymbol{h}}(\mathbf{x}) + \delta h_x$, where δh_x is the relative difference of vertex normal (averaged from the nearby surface normal) from the original mesh to the deformed mesh, and $\tilde{\boldsymbol{h}}'(\mathbf{x})$ is the compensated signed indicator.

B.3 Details of Texture Editing

Since our representation disentangles textures into locally bounded texture codes saved on mesh vertices, texture editing for a neural implicit field can be accomplished by updating or optimizing texture codes (and the binding encoders) for the region of interest.

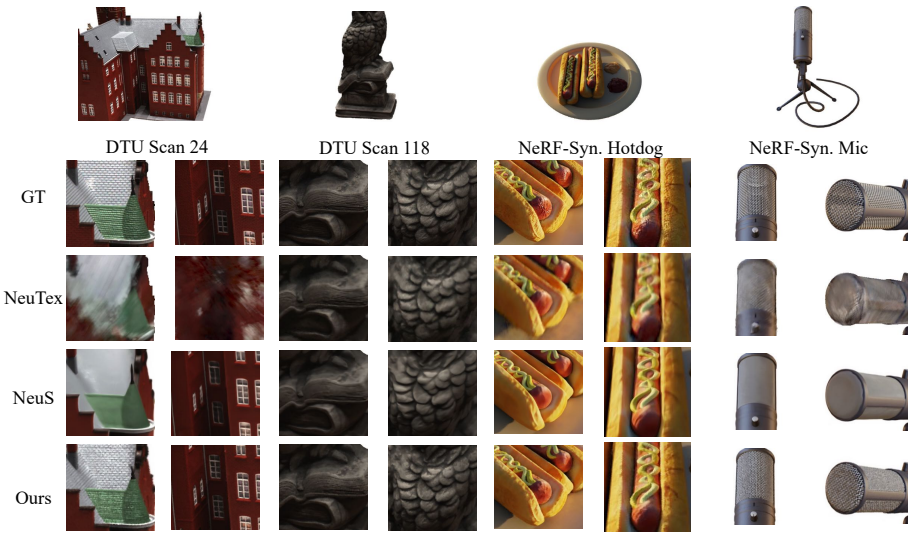


Fig. B. We show more comparison of rendering quality on the DTU dataset and the NeRF 360° Synthetic dataset. Our rendering results show better appearance details than NeuS and NeuTex (*e.g.*, the roof at DTU Scan 37, and the metal grids at NeRF-Synthetic Mic).

Texture swapping. We can easily swap textures of two areas by swapping texture codes on the surface, as long as we find the correspondence from the source area's vertices to the target area's vertices. To this end, we provide a solution to perform texture swapping on two areas that can be reasonably aligned but with slightly different shapes (*e.g.*, two apples in Fig.4 (a) from the main paper). In practice, we first choose source and target areas by selecting mesh vertices with Blender, and annotate 4∼9 point correspondences with our scripts between these two areas. Note that this can also be automated with point cloud or image segmentation tools when deploying to user-friendly applications. Then, we perform non-rigid mesh alignment by solving scaled transformation with Umeyama [13] between point correspondences, and then feed the point residual to ARAP [12], so as to deform the source area to the target area. Finally, we update the texture codes on the target area by assigning interpolated code (with inverse distance weighting) from 4 nearest deformed source vertices.

Texture filling. By leveraging NeuMesh, our model supports filling of the user-selected area on a neural implicit field with a texture template (*e.g.*, furry hair or golden metal in Fig. 7 (c)) from a pre-captured object model. First, we need to obtain the target UV-map of the selected area, *i.e.*, utilizing Blender to unwrap the UV-map of the selected vertices. Then, we select a texture template from a pre-trained NeuMesh model, *e.g.*, a small squared patch with ∼10 vertices, and repeatedly fill the target UV-map with the template in a sliding-window manner. Practically, we assign texture codes in the target vertices with interpolated codes

Fig. C. We show the comparison between textured mesh editing and our NeuMesh editing on a statue. This proves that direct editing texture meshes with template patterns without lighting and material property estimation cannot provide satisfactory results.

from the template and also bind the radiance decoder to the one from the pre-trained model, as the target texture code and the template texture code do not share the same latent radiance space. Besides, to make a smooth transition near the area boundary (*e.g.*, naturally transiting from the edited appearance to the original appearance), for each query point that has texture codes/decoders from different sources, we fuse the color contribution from different decoders with inverse distance weighting.

Texture painting. As introduced in our main paper (Sec.3.4), we propose a spatial-aware optimization to precisely transfer the painting from 2D image to 3D field, while keeping geometry and appearance of other parts unchanged. In detail, we first shoot probing rays from the painted pixels to the mesh scaffold, and find the affected texture codes by collecting the vertices of the hit faces. During optimization, we adopt Adam optimizer with the fixed learning rate of 0.01, and only allow these codes to be changed. The whole texture painting optimization takes about ∼1 hours with 8000 iterations.

C More Discussions

Using neural implicit representation instead of traditional textured mesh. Neural implicit representation merits easy-reconstruction with photo-realistic volumetric rendering and view-dependent effects (*e.g.*, shiny golden materials) on both real-world and synthetic data, and flexibility to accomplish some fine-grained editing demands (*e.g.*, material editing or appearance variations) on the real-world scene with latent space operations. While the rendering quality of the textured mesh is bounded by the MVS reconstruction and texturing. It is not feasible for the textured 3D mesh to achieve such effects (see Fig. C) without BRDF material properties and lighting estimation.

Using learnable signed distance. Unlike voxel-based [4,11] or point-cloud-based [9] methods that possess spatially scattered features, we only learn a set of 'single layer' features on mesh surfaces as we want to build a surface-aligned implicit field. Therefore, a bare code interpolation is not sufficient to coordinate

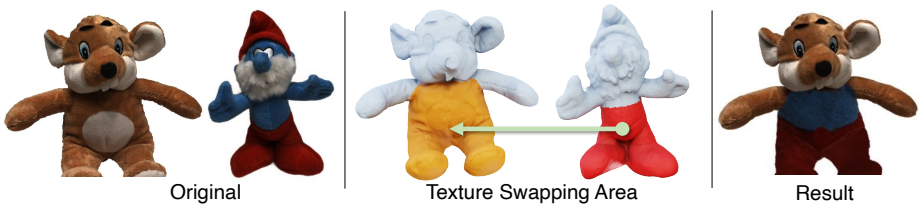


Fig. D. Texture swapping with different geometry.

the query relative position for our mesh-based representation, (*i.e.*, the inner and outer point queries along the direction perpendicular to the surface still lack spatial distinguishability). One plausible solution is to use a physically computed signed distance to the surface as Liu *et al.* [5] does, but it is not applicable for general object meshes because the geometry is not always well-defined (*e.g.*, watertight or even predefined skinning weights) as a human-body model (SMPL) [6], which confuses ray-to-mesh intersection counting and the sign of the distance might be unexpectedly reversed. Therefore, we propose to use a learnable sign indicator to compute interpolated signed distances for spatial query points, as described in Sec. 3.1.

Using distillation instead of training from scratch. As explained in Sec. 3.2, we exploit the teacher NeuS model with distillation and fine-tuning training scheme instead of training from scratch. The teacher NeuS model serves two purposes: **1)** it provides an SDF field where we could extract a mesh scaffold; **2)** the locally embedded geometry and appearance features in our model facilitate region-based editing but may lead the training to fall into a local minimum (as shown in our ablation studies), and the use of distillation helps to alleviate such training issue.

Texture swapping with different geometry/topology. Our method can be applied to objects with a moderate geometry difference (see Fig. C). If there is a significant topology difference between two objects, we suggest using texture filling (Sec. 3.4 (2)) that swaps textures in UV spaces regardless of object geometries.

Limitations for real-world applications. Currently, our rendering speed (about 30s for each view) is bounded by the intensive network queries and nearest neighboring searching operations. When deploying to real-world applications, we might consider accelerating the inference speed to fulfill the real-time rendering demand with recently proposed coefficient caching techniques [11,3] , multiresolution hash encoding [8] or MVS priors [2]. Besides, we rely on 3D modeling software to select vertices for the region of interest, which can be replaced with some semantic annotation approaches [15] to facilitate broaden users.

Relation to point-based methods. From a high-level perspective, both ours and point-based methods can be regarded as building upon local feature-based representations, while the main differences include: **1)** Our model encodes fea-

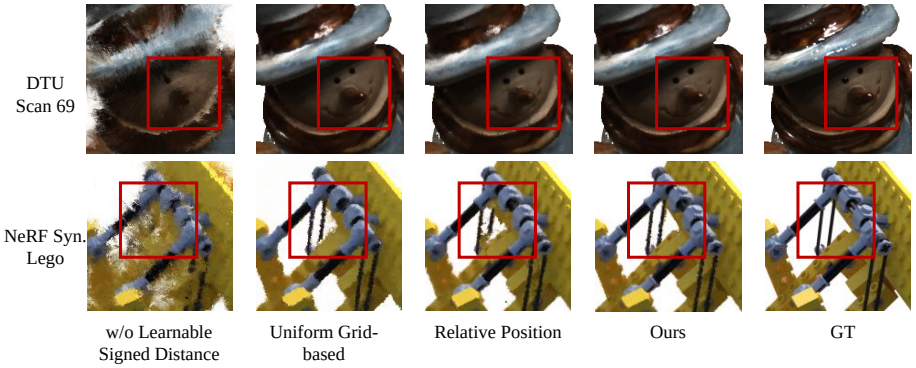


Fig. E. We present visual comparison to alternative designs.

Config.	DTU 69			NeRF 360° Synthetic Lego		
	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓
w/o Learnable Signed Distance	23.622	0.865	0.210	20.827	0.827	0.240
Uniform Grid	26.931	0.943	0.117	25.866	0.898	0.094
Relative Position	26.308	0.937	0.128	27.270	0.918	0.055
Ours	**27.254**	**0.946**	**0.113**	**27.881**	**0.926**	**0.046**

Table A. We perform more experiments to analyze the model design with DTU Scan 69 and NeRF 360° Synthetic Lego.

tures on mesh vertices, so we can easily deform objects with a mesh proxy or modify textures through a UV space. Point-based methods use scattered point features, so it is non-trivial to perform mesh-based editing like ours, *i.e.*, each point that is projected at the pixel (NPBG) or lying nearby ray samples (Point-NeRF) would contribute to the appearance, making it hard to distinguish which point features should be edited. **2)** We embed surface normal (similar to IDR/NeuS) to realize view-dependent effects of texture filling, which cannot be directly inherited by point-based methods.

D More Experiment Results

Rendering quality comparison. We present more results of rendering quality comparison in Fig. B. It is clear that our method renders more details than other competitors, especially when reconstructing with complex shapes and textures (*e.g.*, the roof at DTU Scan 37, and the metal grids at NeRF-Synthetic Mic in Fig B).

Rendering quality with varying mesh vertex numbers. We analyze the impact of varying mesh vertex numbers on rendering quality. Specifically, we train on DTU Scan 114 with 3 sets of mesh vertices (10K, 50K, and 100K). As shown in Fig. F, the metric quality of rendered images are slightly affected

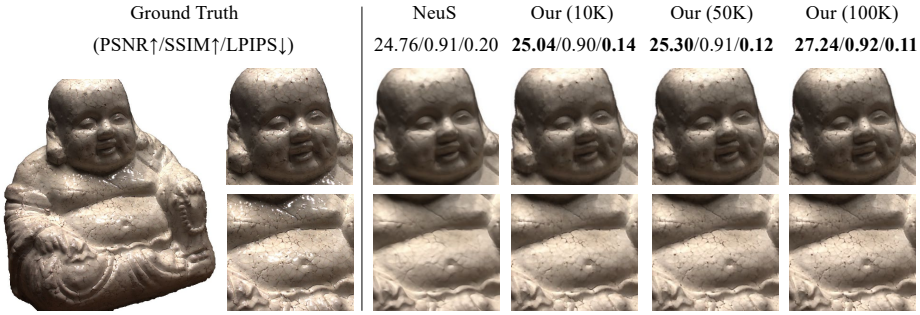


Fig. F. We analyze the impact of vertex numbers on the rendering quality by training with 10K / 50K / 100K vertices.

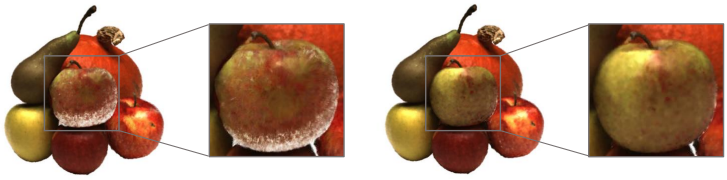
when decreasing vertex numbers, but still outperform NeuS even with only 10K vertices, which demonstrate the robustness and advantages of our representation.

Learnable signed distance. We report the training results without learnable signed distance as network input in Fig. E (first column) and Tab. A (first row), which proves the necessity of this design in our mesh-based representation, as it complements spatial distinguishability on the direction perpendicular to the surface (Sec. 3.1).

Mesh-based representation vs. uniform grid-based representation. We first compare our 'single layer' mesh-based representation with a uniform grid-based representation (*i.e.*, similar to NSVF [4] or Plenoxel [11]). Specifically, we thicken the mesh vertices to uniform grids, so the interpolated codes can be fully aware of the spatial coordinates, and the signed distance can be omitted. Note that this also loses some flexibility for fine-grained editing. As shown in Fig. E (second column) and Tab. A (second row), even with only a single slice of spatial features, our method shows on par visual quality with these uniform grid-based representation, but enables the functionalities of geometry and texture editing.

Learnable signed distance vs. relative position. We then compare the encoding of our learnable signed distance with an alternative design, *i.e.*, relative position encoding from PointNet [10]. Specifically, for each query point, we first concatenate codes (from nearby vertices) and relative coordinate offsets (from query to vertex), and encode with a shallow MLP (with 2 hidden layers and 64 hidden sizes). Then, we use the same inverse distance weighting to obtain the final interpolated embedding for the query. As demonstrated in Fig. E (third column) and Tab. A (third row), our learnable signed distance encoding shows better rendering quality when incorporated with such 'single layer' surface features and is a better choice for mesh-based representation.

Our texture editing vs. radiance field warping. One possible workaround of texture editing is to warp the radiance field from the original space to the aligned space according to the non-rigid mesh alignment (Sec. 3.4). So, we compare our code updating based texture editing with such naïve radiance field warping on DTU Scan 63. As shown in Fig. G, the rendered apple of the naïve



(a) Radiance Field Warping (b) Our Texture Code Swapping

Fig. G. We show the comparison of our texture editing to the field warping.

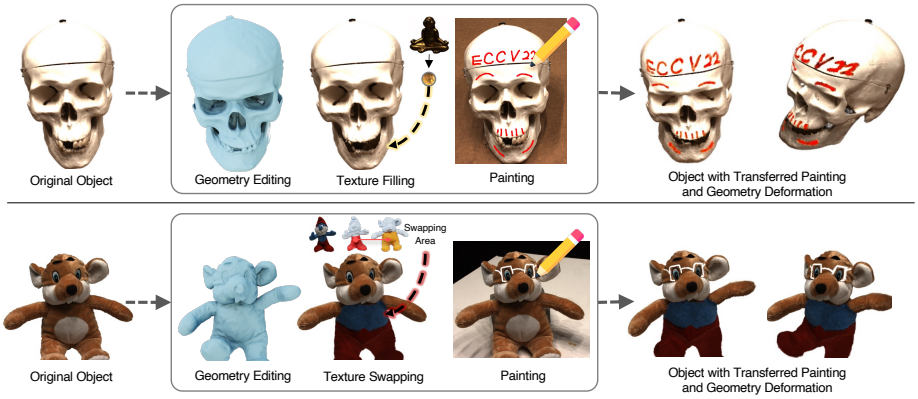


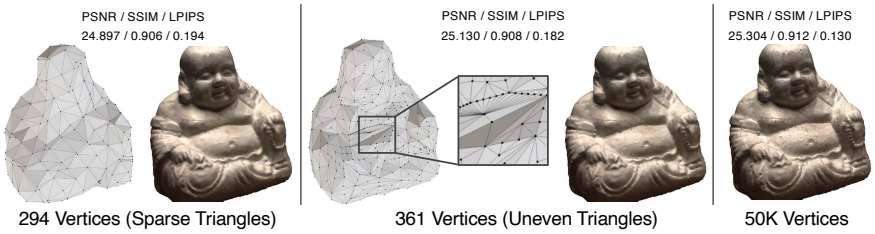
Fig. H. We show examples of hybrid object editing by combining multiple editing operations.

approach contains noticeable artifacts, while our editing result is visually much more natural. We believe that this is mainly due to the fact that the warped texture field might not be compatible with the geometry (SDF field), which leads to spatial misalignment (*e.g.*, SDF field is close to the surface while radiance field is not) during the volume rendering process and produces erroneous color. In contrast, since our method exchanges textures through code swapping, the edited texture field is tightly fit to the geometry, which yields a better rendering quality.

Hybrid object editing. To demonstrate the editing flexibility of our method, we show examples of hybrid object editing in Fig. H by combining geometry/texture editing operations, which sheds light on integrating our representation into modern 3D modeling workflow.

Large-scale scenes. The modeling ability of our method depends mainly on the teacher SDF model. As long as the scaffold mesh is available, our method can be freely scaled-up thanks to the locally embedded features. For large scenes with complicated backgrounds, we can adopt NeRF++ [17]'s parameterization to handle unbounded backgrounds, or use pre-computed segmentation masks like

Fig. I. Texture editing of large-scale scenes on the Tanks&Temple dataset.



in NSVF and IDR. Here are two texture editing examples on the Tanks&Temple dataset with foreground segmentation provided by NSVF.

Influence of triangle quality. Our method can still deliver reasonable rendering quality with locally sparse/uneven triangulation (see below). In fact, as the mesh scaffold is created based on the SDF from teacher NeuS, we can handily guarantee a uniformed distribution of vertices with off-the-shelf mesh regularization algorithms (*e.g.*, isotropic remeshing by Botsch *et al.* [1]).

References

1. Botsch, M., Kobbelt, L.: A Remeshing Approach to Multiresolution Modeling. In: Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing. pp. 185–192 (2004) 9
2. Chen, A., Xu, Z., Zhao, F., Zhang, X., Xiang, F., Yu, J., Su, H.: Mvsnerf: Fast generalizable radiance field reconstruction from multi-view stereo. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 14124–14133 (2021) 5
3. Garbin, S.J., Kowalski, M., Johnson, M., Shotton, J., Valentin, J.: FastNeRF: High-fidelity Neural Rendering at 200FPS. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 14346–14355 (2021) 5
4. Liu, L., Gu, J., Zaw Lin, K., Chua, T.S., Theobalt, C.: Neural Sparse Voxel Fields. Advances in Neural Information Processing Systems **33**, 15651–15663 (2020) 4, 7
5. Liu, L., Habermann, M., Rudnev, V., Sarkar, K., Gu, J., Theobalt, C.: Neural Actor: Neural Free-view Synthesis of Human Actors with Pose Control. ACM Transactions on Graphics (TOG) **40**(6), 1–16 (2021) 5
6. Loper, M., Mahmood, N., Romero, J., Pons-Moll, G., Black, M.J.: SMPL: A Skinned Multi-person Linear Model. ACM Transactions on Graphics (TOG) **34**(6), 1–16 (2015) 5
7. Lorensen, W.E., Cline, H.E.: Marching Cubes: A High Resolution 3D Surface Construction Algorithm. ACM SIGGRAPH Computer Graphics **21**(4), 163–169 (1987) 1
8. Müller, T., Evans, A., Schied, C., Keller, A.: Instant Neural Graphics Primitives with a Multiresolution Hash Encoding. ACM Trans. Graph. **41**(4), 102:1–102:15 (Jul 2022) 5
9. Ost, J., Laradji, I., Newell, A., Bahat, Y., Heide, F.: Neural Point Light Fields. arXiv preprint arXiv:2112.01473 (2021) 4
10. Qi, C.R., Su, H., Mo, K., Guibas, L.J.: PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. In: Proceedings of the IEEE conference on Computer Vision and Pattern Recognition. pp. 652–660 (2017) 7
11. Sara Fridovich-Keil and Alex Yu, Tancik, M., Chen, Q., Recht, B., Kanazawa, A.: Plenoxels: Radiance Fields without Neural Networks. In: CVPR (2022) 4, 5, 7
12. Sorkine, O., Alexa, M.: As-rigid-as-possible Surface Modeling. In: Symposium on Geometry Processing. vol. 4, pp. 109–116 (2007) 3
13. Umeyama, S.: Least-Squares Estimation of Transformation Parameters Between Two Point Patterns. IEEE Transactions on Pattern Analysis & Machine Intelligence **13**(04), 376–380 (1991) 3
14. Wang, P., Liu, L., Liu, Y., Theobalt, C., Komura, T., Wang, W.: NeuS: Learning Neural Implicit Surfaces by Volume Rendering for Multi-view Reconstruction. NeurIPS (2021) 1, 2
15. Wang, W., Yu, R., Huang, Q., Neumann, U.: SGPN: Similarity Group Proposal Network for 3D Point Cloud Instance Segmentation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 2569–2578 (2018) 5
16. Yariv, L., Kasten, Y., Moran, D., Galun, M., Atzmon, M., Ronen, B., Lipman, Y.: Multiview Neural Surface Reconstruction by Disentangling Geometry and Appearance. Advances in Neural Information Processing Systems **33**, 2492–2502 (2020) 2
17. Zhang, K., Riegler, G., Snavely, N., Koltun, V.: NeRF++: Analyzing and Improving Neural Radiance Fields. arXiv preprint arXiv:2010.07492 (2020) 8