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

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

Abstract. Very recently neural implicit rendering techniques have been rapidly evolved and shown great advantages in novel view synthesis and 3D scene reconstruction. However, existing neural rendering methods for editing purposes offer limited functionality, *e.g.*, rigid transformation, or not applicable for fine-grained editing for general objects from daily lives. In this paper, we present a novel mesh-based representation by encoding the neural implicit field with disentangled geometry and texture codes on mesh vertices, which facilitates a set of editing functionalities, including mesh-guided geometry editing, designated texture editing with texture swapping, filling and painting operations. To this end, we develop several techniques including learnable sign indicators to magnify spatial distinguishability of mesh-based representation, distillation and fine-tuning mechanism to make a steady convergence, and the spatial-aware optimization strategy to realize precise texture editing. Extensive experiments and editing examples on both real and synthetic data demonstrate the superiority of our method on representation quality and editing ability. Code is available on the project webpage: https://zju3dv.github.io/neumesh/.

Keywords: neural rendering, mesh-based representation, scene editing, view synthesis, 3D deep learning

1 Introduction

Neural implicit field has achieved great success in 3D reconstruction and free-viewpoint rendering, and becomes a promising solution to take the place of traditional 3D shape and texture representation, *e.g.*, point cloud or textured mesh, due to its phenomenal rendering quality. However, for 3D modeling and CG creation, artists still prefer to use mesh-based workflow across daily works. For instance, in modern 3D CG software (*e.g.*, Blender, Maya and 3ds Max), polygon mesh-based representations can be precisely controlled and edited, *i.e.*, texturing with UV-map and changing shapes by altering vertices and faces, with

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

Fig. 1. NeuMesh. We present a novel representation for volumetric neural rendering, which encodes the neural implicit field with disentangled geometry and texture features on a mesh scaffold. With the locally separated latent codes, our representation enables a series of editing functionalities, including mesh-guided geometry deformation, designatable texture swapping, filling and painting.

all the previewed modification accurately reflected in the final rendering product. Despite great progress made to improve the flexibility of the neural implicit field, including handling dynamic scenes [34,35], becoming scene agnostic [56,45], fast rendering [43,28], and scalability improvement [51,60], the support of neural implicit field towards editing is still limited, *e.g.*, on a very specific semantic category [6,23,15] or purely rigid transformation [69,64,12]. One plausible reason is that particular network encoding structures (*e.g.*, coordinate-based MLP, voxels or scattered point cloud) are not compatible with fine-grained scene editing such as non-rigid geometry deforming and texture editing for a local region of interest, and thus cannot satisfy the broad demands of artistic creation.

In this paper, we propose a novel neural implicit representation, NeuMesh, to facilitate editing in both 3D modeling and texturing. Our representation bares the following properties to seamlessly integrate with existing common workflow for 3D editing: **1)** The neural representation encodes scene with a series of vertex-bounded codes on a mesh scaffold and MLP-based decoders, instead of a pure MLP, point clouds or voxels, and can be deformed together with the mesh. During the volume rendering, the implicit field is decoded via interpolation of these codes. By doing so, any modification to the mesh geometry or local codes would precisely reflect the rendering output. **2)** The geometry and appearance representations are disentangled, *i.e.*, encoded in two separate latent codes, such that texture can be transferred across geometry by replacing the appearance code from one another. As shown in Fig. 1, our representation supports non-rigid object deformation with a handful approach (*e.g.*, deforming with a mesh proxy), and provides various fashions of texture editing, including texture swapping of irregular mesh segments, texture filling at a specific area with pattern from a pre-captured object, and a user-friendly texture painting which reflects the philosophy of 'what you get is what you see'.

However, learning and deploying such representation for rendering and editing is non-trivial. **1)** Unlike voxel-based representation [20], naïve trilinear code interpolation is not sufficient to measure spatial variation since we dedicate to encoding the implicit field with a set of 'single layer' codes on mesh vertices, and the inner/outer queries along the direction perpendicular to the surface lacks spatial distinguishability (*i.e.*, failing to determine positive or negative direction when crossing through a mesh face). A possible workaround is to complement the network input with signed distance to the mesh surface [21], which, however, is not always available, especially on non-watertight/ill-defined geometries. To tackle this challenge, we propose to maintain a set of learnable sign indicators for mesh vertices. Then, for each query point along the ray, we compute a signed distance from nearby vertices by weighting the projected distances on the indicators. In this way, our representation is completely agnostic to arbitrary mesh typologies (*e.g.*, non-watertight or non-manifold meshes). During the training process, these sign indicators are continuously adjusted to best fit the optimization objective. **2)** Although such vertex-bounded and geometry-texture disentangled representation merits good flexibility on editing purpose, it does not preserve spatial continuity as MLP-based methods [27,55,65] and thus easily suffers from unstable training. To mitigate this problem, we employ a distillation and fine-tuning training scheme, which leverages a pre-trained implicit field to guide the optimization of our representation. In this way, we transfer a baked MLP-based implicit model into NeuMesh, the first neural rendering model that naturally inherits editable capability from the flexible mesh-based workflow. **3)** To fulfill the demand for flexible and user-friendly texture editing operations (*e.g.*, propagating 2D image painting to the 3D field), a naïve approach is fine-tuning with a single image. However, this might let the network overfit to a specific view and the rendered images from other views degrade (*e.g.*, introducing noticeable artifacts as shown in Fig. 8 (b)). In order to solve this challenge, we propose a spatial-aware optimization strategy that is naturally derived from our representation, in which we select the affected texture codes with several probing rays from painted pixels to the mesh surface, and only fine-tuning these codes during the optimization. Therefore, we can precisely transfer the painting to the desired region while maintaining other parts unchanged.

The contributions of our paper can be summarized as follows. **1)** We present a novel mesh-based neural implicit representation which aims to break the barrier between volumetric neural rendering and mesh-based 3D modeling and texturing workflow, and delivers a set of editing functionalities, including mesh-guided geometry editing, designated texture editing with texture swapping, filling and painting operations. To make the representation locally editable both on geometry and texture, we design to encode the implicit field into mesh vertices, where each vertex possesses disentangled geometry and texture features of its local space. **2)** We analyze the technical challenges and develop several techniques to enhance the spatial distinguishability with learnable sign indicators, ensure a steady training with distillation and fine-tuning mechanism, and improve the texture editing precision with spatial-aware optimization strategy. **3)** Extensive

experiments and impressive editing examples on both real and synthetic datasets demonstrate that our method achieves photo-realistic rendering quality, and is flexible and powerful at geometry and texture editing of the neural implicit field.

2 Related Works

Mesh-based representation and rendering. In computer vision and graphics, polygon mesh has been widely used in 3D scene modeling and rendering [13,1,22]. Traditional methods utilize multi-view geometry and numerical theories to reconstruct surface meshes of a captured scene [44,62,17,54]. Recently, more attention has been paid to neural network based scene reconstruction [29,49] and texture learning [52,8,59]. However, existing mesh-based rendering pipelines usually require UV-mapping to build correspondences between meshes vertices and texture maps, which limits the applicability from representing scenes with complex topology and delicate structure. Another line of methods uses MVS based mesh as a geometry proxy for image feature aggregation [41,42], but requires nearby source images to be warped back to the mesh surface and is not feasible for high-level editing operations. Instead of storing textures in a flat 2D map or warping-based view synthesis, our method directly encodes appearance information on 3D vertices, and is more flexible in representing complex objects whose UV-maps are difficult to be unwrapped.

Neural rendering. Given a set of image captures, neural rendering methods [5,66] aim to render photo-realistic images of novel views. NeRF [27] takes advantages of volume rendering to boost rendering quality, which inspires a lot of works, including surface reconstruction [32,55,65], human modeling [21,36], pose estimation [67], scene understanding [63] and relighting [48,71,2,72], *etc.*. To further increase network capacity and reduce computation, many works propose to decompose scene into local representations, such as multiple tiny networks [39], point clouds [33] and voxels [20,43]. Although these works explicitly encode scenes in a 3D spatial structure, they are not designed to be easily manipulated as polygon meshes, thus not capable of high-level applications like geometry and texture editing.

Neural scene editing. Scene editing is a popular topic in computer vision and photography. Early methods mainly focus on editing a single static view by inserting [16], compositing [37], moving [18,46] objects or changing lighting [26] for an existing photograph. With the development of neural rendering, many works start to edit scenes with movable [69,64,12] and deformable [68] objects, changeable colors, shapes [61,23] and textures [59]. However, existing methods are either limited to object-level rigid transformation [69,64,12], not generalize to out-of-distribution categories [6,23,31,50,4,57], restricts its representation to simple shapes [59] or orthographic projection [40], or does not support fine-grained texture editing [23,66,71,48,30,68]. By contrast, we pick up triangle mesh as a scaffold to encode the scene, since the mesh can be edited conveniently and intuitively in mature industry software, and the region of interest on the mesh can be precisely selected by vertices. Built upon this, our method delivers the capability of non-rigid geometry editing and fine-grained texture editing.

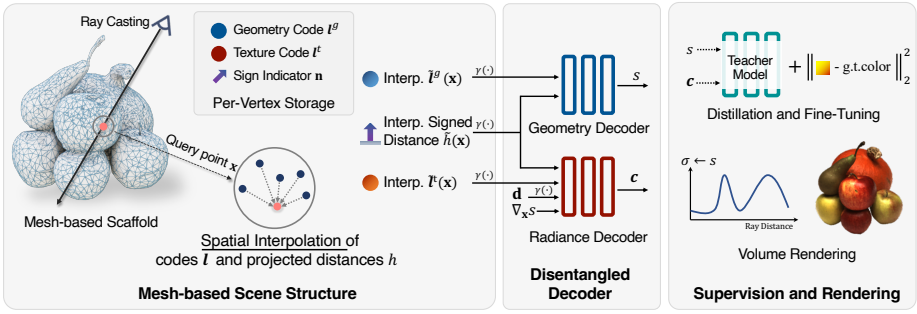


Fig. 2. Overview. We encode neural implicit field on a mesh-based scaffold, where each vertex possesses a geometry and texture code $\boldsymbol{l}^g, \boldsymbol{l}^t$, and a sign indicator $\mathbf{n}$ for computing projected distance h. For a query point $\boldsymbol{x}$ along a casted camera ray, we retrieve interpolated codes and signed distances from the nearby mesh vertices, and forward to the geometry/radiance decoder to obtain SDF value s and color $\mathbf{c}$.

3 Method

We introduce NeuMesh, a novel scene representation that encodes neural implicit field at a mesh-based scaffold. As demonstrated in Fig. 2, instead of learning the entire scene as a whole in a coordinate-based network, we leverage 3D mesh structure by decomposing the scene into a set of local-vertex-bounded implicit fields (Sec. 3.1), where each vertex stores geometry and texture information of its neighboring local space. Motivated by previous works [27,55,39], we adopt the volume rendering technique to render pixels, and employ a distillation and fine-tuning training scheme to encode the neural implicit field into the mesh surface (Sec. 3.2). During the rendering stage, we retrieve interpolated codes and learnable signed distances (*i.e.*, projected distances to the mesh vertices, which complements spatial distinguishability) from the mesh, and use two separated MLPs to decode geometry (*i.e.*, SDF values) and radiance color. In this way, the scene representation is locally aligned to the mesh, and the geometry and color are encoded in two separated latent spaces, which naturally derives the approaches of mesh-guided geometry deforming and designatable texture editing (Sec. 3.3).

3.1 Neural Mesh-based Implicit Field

Mesh-based representation. As illustrated in Fig. 2, we use a mesh-based scaffold to model the neural implicit field. First, we reconstruct the target object using out-of-box NeuS [55] and marching cubes [25], which yields a triangle mesh with about 50K∼150K vertices. Then, for each vertex $\mathbf{v}$ on the mesh, we store a set of learnable parameters, including a geometry code $\boldsymbol{l}^g$, a texture code $\boldsymbol{l}^t$ and a sign indicator $\mathbf{n}$ ($\mathbf{n}$ helps to identify relative position, and will be introduced later). In a typically volume rendering process that sample points $\mathbf{x}$ along the ray, we first find K nearest vertices $\{\mathbf{v}_k | k = 1, 2, ..., K\}$ for each point $\mathbf{x}$, and perform spatial interpolation to obtain the interpolated codes $\tilde{\boldsymbol{l}}^g(\mathbf{x})$, $\tilde{\boldsymbol{l}}^t(\mathbf{x})$ and

signed distances $\tilde{h}(\mathbf{x})$. Specifically, we adopt inverse distance weighting based interpolation [38], as:

$$\tilde{\boldsymbol{l}}(\mathbf{x}) = \frac{\sum_{k=1}^{K} w_k \boldsymbol{l}_k}{\sum_{k=1}^{K} w_k}, \quad w_k = \frac{1}{||\mathbf{v}_k - \mathbf{x}||}. \tag{1}$$

Then, we forward all these variables to geometry decoder F_G and radiance decoder F_R to obtain the SDF value $s = F_G\left(\tilde{\boldsymbol{l}}^g, \tilde{h}\right)$ and color $\boldsymbol{c} = F_R\left(\tilde{\boldsymbol{l}}^t, \tilde{h}, \mathbf{d}, \nabla_{\mathbf{x}} s\right)$ at point $\mathbf{x}$, where $\mathbf{d}$ is the viewing direction, $\nabla_{\mathbf{x}} s$ is the gradient of the SDF w.r.t query position. Different from the previous methods [66,55,65], we replace the global coordinate $\mathbf{x}$ with locally retrieved codes $\tilde{\boldsymbol{l}}^g, \tilde{\boldsymbol{l}}^t$ and sign distances $\tilde{h}$, where $\tilde{h}$ complements spatial distinguishability without hurting the locality of the representation. Note that we also apply positional encoding $\gamma(\cdot)$ [27] to the interpolated codes, distance and direction before feeding them into the MLP, but we omit it in the equations for brevity. Following the formulation of NeuS [55] and quadrature rules [27], we render the pixel $\hat{C}(\boldsymbol{r})$ with points $\{\mathbf{x}_i | i = 1, ..., N\}$ along the ray $\boldsymbol{r}$ as:

$$\hat{C}(\boldsymbol{r}) = \sum_{i=1}^{N} T_i \alpha_i \mathbf{c}_i, \; T_i = \prod_{j=1}^{i-1} (1 - \alpha_j), \; \alpha_j = \max\left(\frac{\Phi_s(s_i) - \Phi_s(s_{i+1})}{\Phi_s(s_i)}, 0\right), \tag{2}$$

where T is accumulated transmittance, Φ_s is the cumulative distribution of logistic distribution, and α is opacity derived from adjacent SDF.

Learnable sign indicator for interpolated signed distance. To complement the spatial distinguishability of the network query along the direction perpendicular to the surface (*i.e.*, inside or outside the mesh), we introduce a *learnable* sign indicator $\mathbf{n}_k$ for each vertex $\mathbf{v}_k$ that aids at computing interpolated signed distances for spatial query points. Indeed, the sign indicator plays a similar role as vertex normal (*i.e.*, initialized with vertex normal), but is continuously adjusted during the training process to best fit the target loss. The computation of interpolated signed distance $\tilde{h}(\mathbf{x})$ is defined as:

$$\tilde{h}(\mathbf{x}) = \frac{\sum_{k=1}^{K} w_k h_k}{\sum_{k=1}^{K} w_k}, \; h_k = \mathbf{p}_k \cdot \frac{\omega^n \mathbf{n}_k + \omega_k^p \mathbf{p}_k}{\omega^n + \omega_k^p}, \; \mathbf{p}_k = \mathbf{x} - \mathbf{v}_k, \tag{3}$$

where w_k is inverse distance weighting as defined in Eq. (1), ω^n and ω_k^p controls the influence between sign indicator and point-to-vertex vector $\mathbf{p}_k$, and we empirically set $\omega^n = 0.1, \omega_k^p = ||\mathbf{p}_k||$. Intuitively, when the sample points are far from the surface, $\tilde{h}(\mathbf{x})$ is numerically close to the point-to-surface distance; otherwise, when the sample points are getting close to the surface, $\tilde{h}(\mathbf{x})$ would be gradually perturbed by learnable sign indicators.

3.2 Optimizing Mesh-based Implicit Field

Distillation and fine-tuning. We observe that training NeuMesh from scratch leads to artifacts and converges to sub-optimal results (see Fig. 8 and Tab. 3). Inspired by Reiser *et al.* [39], we apply a distillation and fine-tuning training

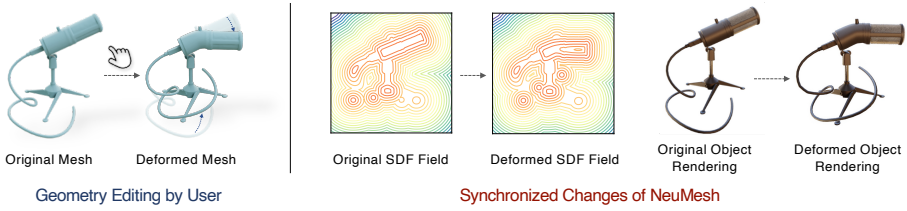


Fig. 3. Mesh-guided Geometry editing. By simply deforming the corresponding mesh, the change will synchronously take effect on the implicit field, and the rendered object will also be deformed accordingly.

scheme, *i.e.*, we supervise NeuMesh simultaneously with the output from a coordinate-based teacher model (*e.g.*, NeuS), and also the images. For a batched training rays $\boldsymbol{r} \in R$, we defined the distillation loss L_d and photometric fine-tuning loss L_f as:

$$\mathcal{L}_{\mathrm{d}} = \sum_{\boldsymbol{r}\in R}\sum_{i\in N} ||s_i - s_i^t|| + ||\boldsymbol{c}_i - \boldsymbol{c}_i^t||, \quad \mathcal{L}_{\mathrm{f}} = \sum_{\boldsymbol{r}\in R} ||\hat{C}(\boldsymbol{r}) - C(\boldsymbol{r})||_2^2 \tag{4}$$

where s_i^t and $\boldsymbol{c}_i^t$ are the SDF value and color from the teacher model, and $C(\boldsymbol{r})$ is the ground-truth pixel color from images. By leveraging distillation and fine-tuning, we smoothly transfer a pure MLP-based neural implicit model into a flexible and editable mesh-based representation, and the final model even produces better appearance details, as shown in our experiment (Sec. 4.2).

Regularization. As introduced in Sec. 3.1, we dynamically adjust a set of per-vertex sign indicators during the training process. To ensure a smooth convergence, we empirically apply a regularization to the sign indicator by slightly encouraging them being close to pre-computed vertex normal $\boldsymbol{n}^t$, as: $\mathcal{L}_{\mathrm{rs}} = \sum_k ||\boldsymbol{n}_k - \boldsymbol{n}_k^t||_2^2$. Besides, as suggested by Gropp *et al.* [10], we add an Eikonal loss to regularize the norm of the spatial gradients to 1, as: $\mathcal{L}_{\mathrm{re}} = \sum_k || \, \|\nabla_{\mathbf{x}_k} s_k\| - 1||_2^2$.

The final loss is then defined as:

$$\mathcal{L}_{\mathrm{total}} = \lambda_{\mathrm{d}}\mathcal{L}_{\mathrm{d}} + \lambda_{\mathrm{f}}\mathcal{L}_{\mathrm{f}} + \lambda_{\mathrm{rs}}\mathcal{L}_{\mathrm{rs}} + \lambda_{\mathrm{re}}\mathcal{L}_{\mathrm{re}}, \tag{5}$$

where we set $\lambda_{\mathrm{d}} = 1.0$, $\lambda_{\mathrm{f}} = 1.0$, $\lambda_{\mathrm{rs}} = 0.01$ and $\lambda_{\mathrm{re}} = 0.01$.

3.3 Mesh-guided Geometry Editing

In NeuMesh, since the neural implicit field has been tightly aligned to the mesh surface, any manipulation on mesh vertices would directly take effect on the field and the volume rendering results. Therefore, to perform geometry editing with a NeuMesh-based scene, users are only required to edit the corresponding mesh, which can be easily accomplished by interactively moving a few vertices with out-of-box mesh deforming methods (*e.g.*, as-rigid-as-possible, or ARAP [47]), or 3D modeling software like Blender. We show an example of the geometry editing in Fig. 3, where we first deform the microphone by bending its head and lifting the wire on the corresponding mesh. Then, to maintain the local consistency of

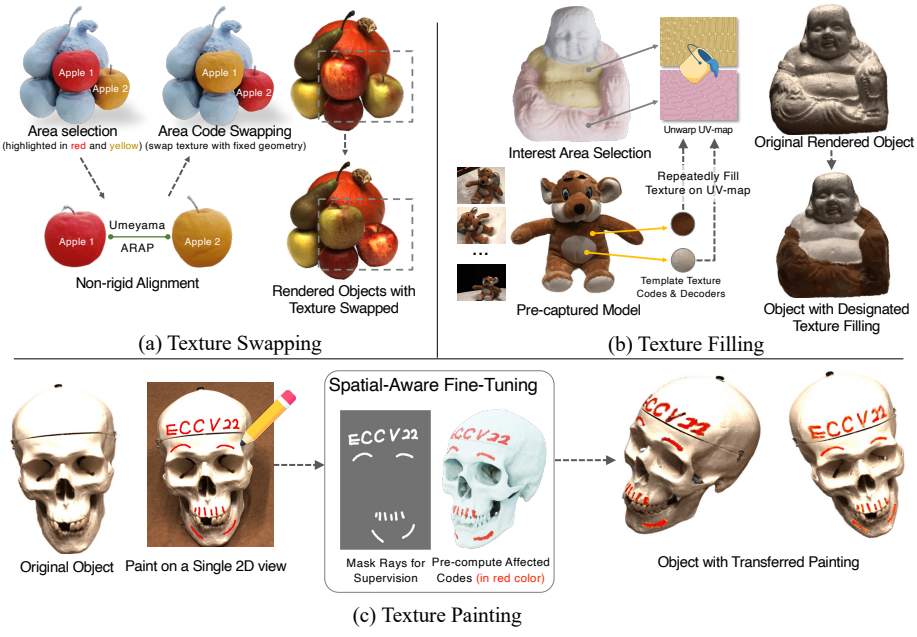


Fig. 4. Designatable Texture editing. By exchanging texture codes (and decoders), our representation delivers various texture editing pipelines on neural implicit field. [1]

signed distance (Sec 3.1), for each transformed vertex, we also compute a relative rotation of the surface normal and compensate the rotation according to its sign indicator (Sec. 3.1). Without any fine-tuning, the microphone's implicit field has been deformed in the meantime, and we can easily render the deformed view (see Fig. 3). Please refer to the supplementary materials for more details.

3.4 Designatable texture editing

Until then, texture editing on the neural implicit model is still an open problem. Previous methods tend to replace the entire materials by swapping the appearance branch [66,71,48], changing a uniformed color [23], or learning an editable UV mapping for simple and plump shapes [59]. However, in real texturing of 3D modeling software, artists are used to working with a mesh-based workflow, which allows them to select a partial region of an object and modify it with arbitrary colors and material properties. We propose to mimic such pipelines by introducing a designatable texture editing, where the selection of mesh vertices can be used to precisely guide the texture editing on the region of interest. The core step of our texture editing is that we update the latent texture code $\boldsymbol{l}^t$ ('material properties') and the binding decoder F_R ('rendering palette') at the selected region. As shown in Fig. 4, we deliver three ways of texture editing:

[1] Icon credit: Flaticon [7]

1) Texture swapping by exchanging textures between two objects through 3D geometry (*e.g.*, swapping textures of two apples in Fig. 4). Users are first asked to mark out the source and target object on the mesh, which can be done by mature 3D model software, or point-based instance segmentation [58]. Then, given a putative point matches with interactive annotation [73], we perform non-rigid 3D alignment to the source and target object with Umeyama [53] and ARAP [47]. Finally, we transfer texture codes by assigning each target vertex with code interpolated from nearby source vertices.
2) Texture filling by filling a targeting object area with repeated textures from a pre-captured model (*e.g.*, assigning part of Buddha with two furry materials from a teddy bear as shown in Fig. 4). In real applications, artists might want to try out some materials from a daily captured scene or pre-built material library, or want to fill some areas (*e.g.*, floor and walls) with uniformed materials. Therefore, we build a compatible workflow for the standard texturing operation, where we first construct a UV map for the user-interest areas, and then repeatedly fill the mapped vertices (*e.g.*, chest and cloth of Buddha) with template textures from a pre-captured model (*e.g.*, gray and brown hairs from the teddy bear).
3) Texture painting from a single 2D view to the 3D field. Users paint an arbitrary pattern or put some text on a captured image, and we can transfer these paintings into the 3D neural implicit field and freely preview in rendered novel views (*e.g.*, painting ECCV logo on a skull in Fig. 4). Compared to NeuTex [59] that might be difficult to edit on the desired position due to distorted UV-mapping, our method delivers a more natural editing way, *i.e.*, what you draw and see is what you get. However, it is not trivial to precisely control the painting transferring with only one image, since the overfitting of a single image might lead to appearance drifting at unconstrained views, which inevitably introduces artifacts in rendered novel views. To tackle this issue, we propose a *spatial-aware optimization mechanism*. Specifically, we first shoot rays through the painted pixels to obtain the surface points and find the affected texture codes of nearby vertices around the points. During the fine-tuning stage, we optimize by minimizing photometric loss of rendered pixels and painted pixels, and only backward gradient of these codes while detaching the others. Besides, to improve the training efficiency and the painting consistency across views, we restrict training rays inside a slightly dilated paint mask, and also augment with random viewing directions at the input to the radiance decoder.

Please refer to the supplementary materials for more details.

4 Experiments

4.1 Datasets

We evaluate our method on the real captured DTU [14] dataset and NeRF 360° Synthetic dataset. For the DTU dataset, we follow the setting of IDR [66] by using 15 scenes with images of 1600×1200 resolution and foreground masks for experiments. To facilitate the metric evaluation for both rendering and mesh quality, we randomly select 10% images as test split and use the remaining images for training. For NeRF 360° Synthetic dataset, we follow the official split and

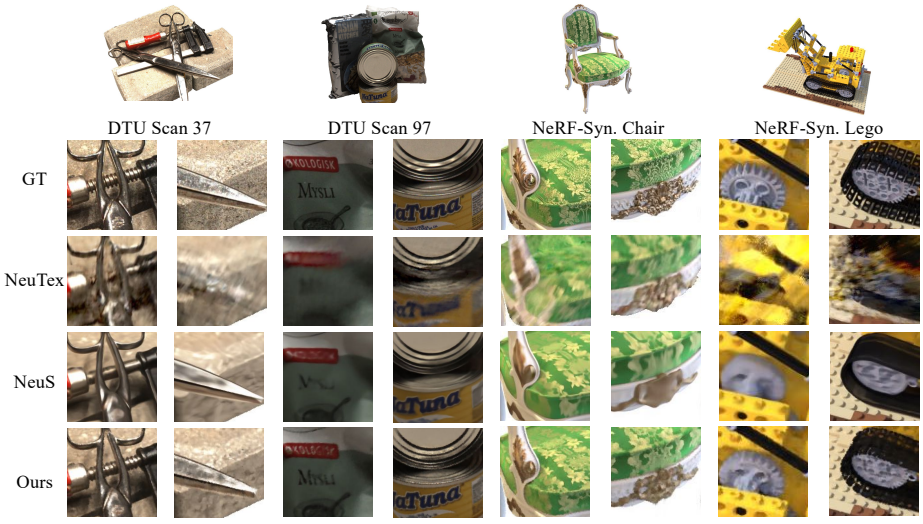


Fig. 5. We show rendering examples of NeuTex [59], NeuS [55] and our method on the DTU dataset and the NeRF 360° Synthetic dataset.

Methods	DTU			NeRF 360° Synthetic		
	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓
NeuTex [59]	26.080	0.893	0.196	25.718	0.914	0.109
NeuS [55]	26.352	0.909	0.176	30.588	**0.960**	0.058
Ours	**28.289**	**0.921**	**0.117**	**30.945**	0.951	**0.043**

Table 1. We compare rendering quality with NeuS [55] and NeuTex [59] on the DTU dataset and the NeRF 360° Synthetic dataset.

choose 4 representative scenes for evaluation, including thin structures (Mic), complex shapes (Lego) and rich textures (Chair and Hotdog).

4.2 Comparison of Rendering and Mesh Quality

We first compare the rendering and mesh reconstruction quality of our representation with the baseline method NeuS [55] and the SOTA texture-editable implicit neural rendering method NeuTex [59]. Following previous works [27,55,66], we use PSNR, SSIM and LPIPS to measure the rendering quality, and use Chamfer distance to measure the reconstructed mesh quality. Please note that for mesh quality comparison, we use a subset (training split) of images, while NeuS takes all images for training in their paper, so the result is slightly different. As demonstrated in Fig. 5 and Tab. 1, our method is comparable or even better than NeuS and NeuTex on rendering quality. To achieve texture editing, NeuTex attempts to encode all the textures in a single continuous UV space, which works for plump objects (*e.g.*, plush toys or Buddah as shown in its paper) but struggles to reconstruct objects with complex shapes (*e.g.*, scissors in DTU Scan 37 and gears in NeRF-Synthetic Lego as shown in Fig. 5). We consider that because

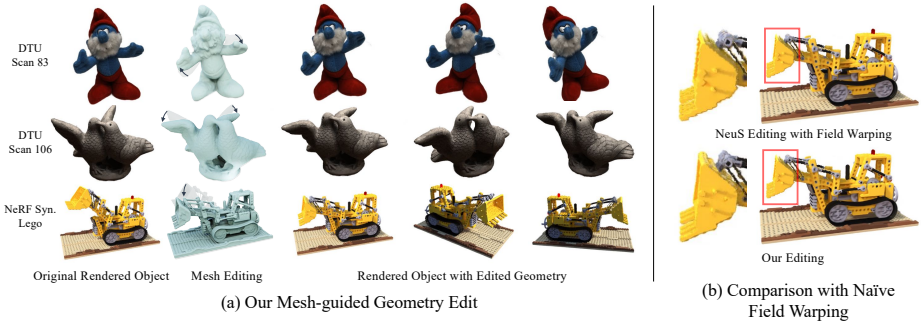


Fig. 6. We show examples of mesh-guided geometry editing in (a) and also compare with naïve field warping solution in (b).

	DTU Scan ID															
Method	24	37	40	55	63	65	69	83	97	105	106	110	114	118	122	Avg.
NeuTex [59]	2.078	5.038	3.477	1.039	3.744	2.078	3.201	2.163	5.104	1.828	1.951	4.319	1.177	3.100	1.921	2.815
NeuS* [55]	1.544	**1.224**	1.065	**0.665**	1.286	**0.825**	0.904	1.350	1.320	0.855	0.987	1.328	**0.487**	**0.636**	**0.678**	1.010
Ours	**1.112**	1.262	**0.988**	0.674	**1.224**	0.835	**0.878**	**1.232**	**1.304**	**0.741**	**0.963**	**1.239**	0.558	0.645	0.739	**0.960**

Table 2. We compare mesh quality (Chamfer distance) with NeuS [55] on the DTU dataset. Note that we use training split of images instead of full images, so the result of NeuS [55] is different from the original paper.

NeuTex tries to memorize all textures in the single continuous UV-map by using a simplified Atlas-Net [11] (*i.e.*, one atlas), which limits its representation of complex shapes. For NeuS, as it pursues better mesh reconstruction quality than novel view synthesis, the details of rendered images are slightly blurred and smoothed. By contrast, our representation not only delivers the capability of geometry and texture editing, but also shows clear appearance detail (see Fig. 5) and maintains mesh quality on par with NeuS (see Tab. 2).

4.3 Experiment on Geometry Editing

We now show the result of mesh-guided geometry editing in Fig. 6 (a), where we simply deform meshes with Blender, and the rendered objects are deformed simultaneously. Since an implicit field can be trivially deformed with non-rigid warping, we also compare our editing with a naïve field warping solution that is directly applied to NeuS, which bends the query points from the deformed space to the original space by computing interpolated warping with the offsets from 3 nearest vertices of the extracted mesh. As shown in Fig. 6 (b), the object boundary of the field warping results is much jaggier than ours, which proves the necessity of our mesh-based representation on this task.

4.4 Experiment on Texture Editing

To the best of our knowledge, prior to our work, only NeuTex [59] supports texture editing of the neural implicit field by painting on 2D UV texture. How-

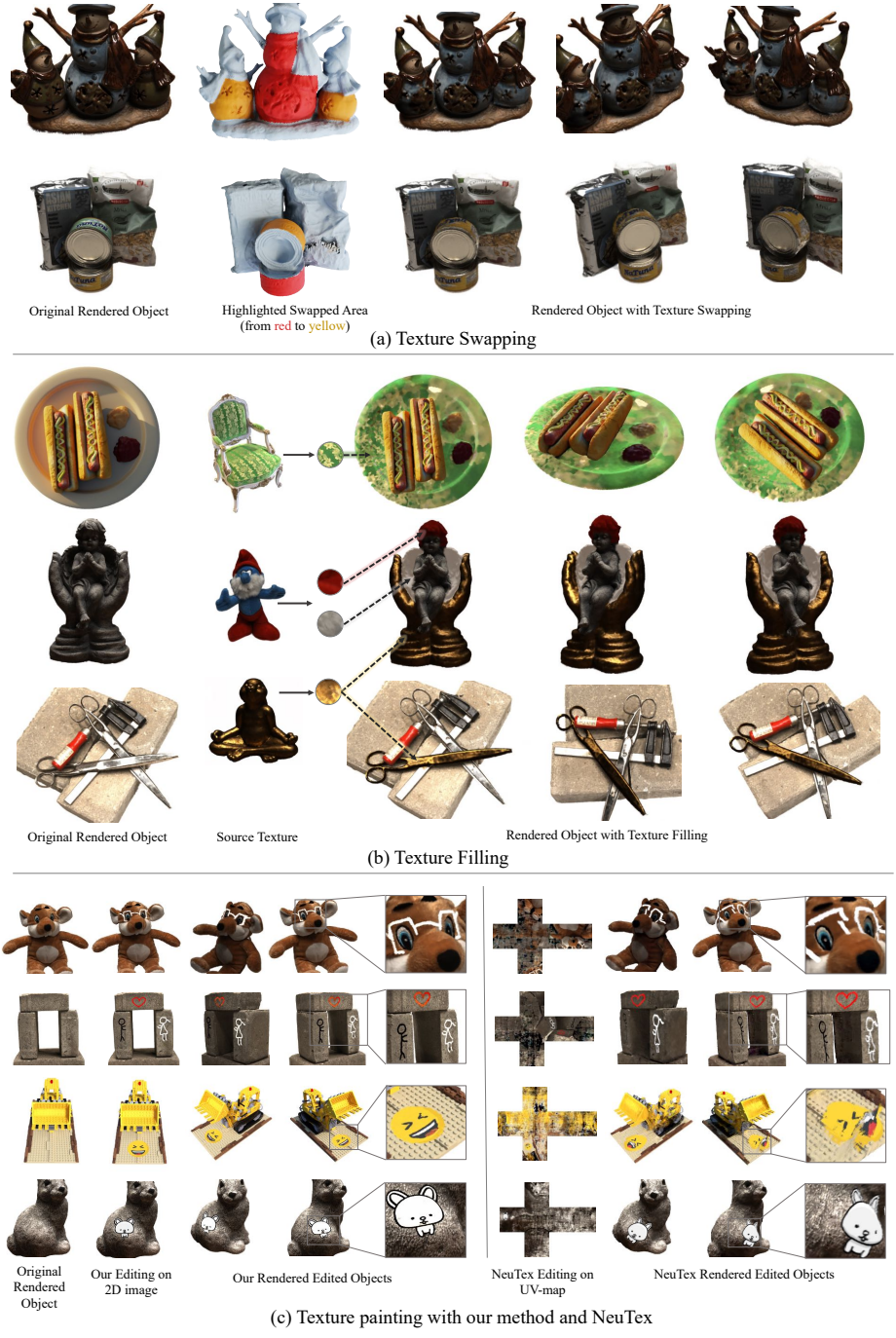


Fig. 7. We show texture editing examples on the DTU dataset and the NeRF 360° Synthetic dataset.

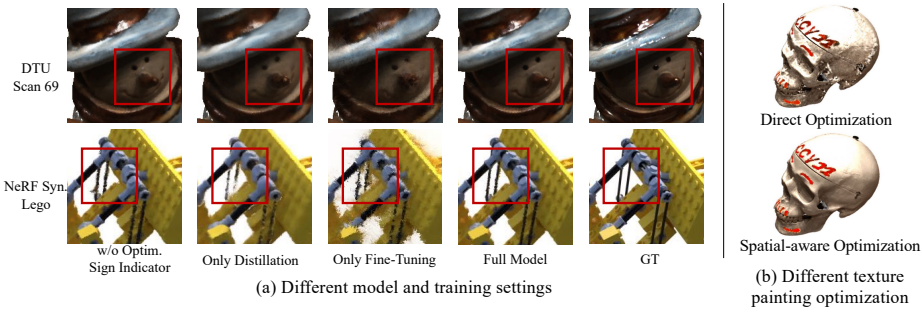


Fig. 8. We show the rendering results with different settings in (a) and also show the effectiveness of spatial-aware optimization for texture painting in (b).

ever, due to the distorted UV mapping, we find NeuTex hard to perform all the editing operations like ours, so we only compare it on the texture painting task.

Texture swapping. We present 2 examples of texture swapping in Fig. 7 (a), where the textures of the snowman's body and the packaging of cans have been seamlessly swapped, and even the details (*e.g.*, texts on the cans) have been clearly transferred into the target object, while the geometry is kept unchanged. This demonstrates that our representation successfully disentangles geometry and texture in two spaces, and the disentangled texture representation can be seamlessly integrated into new shapes.

Texture filling. We show 3 examples of texture filling in Fig. 7 (b), in which the targeting areas are repeatedly filled with template texture code and decoder from previously captured source models. It is worth noting that even though the source template only covers a small area of texture codes, we can still observe view-dependent effects (*e.g.*, golden metal naturally exhibits specular reflections at different views).

Texture painting. In Fig. 7 (c), we exhibit 4 examples of texture painting, and also conduct similar editing with NeuTex [59] by painting on the unwrapped UV-map. For NeuTex, as the learned texture mapping is somehow irregular and distorted (*e.g.*, the head of the teddy bear is separated in UV-map), we find it hard to paint at the desired location. In the second row (painting on bricks), we have to adjust the painting position back and forth to get a reasonable editing result. Besides, due to the mapping issue explained in Sec. 4.2, NeuTex cannot picture a clear result when editing on complex shapes (Lego in the second row). On the contrary, our method offers a user-friendly editing pipeline by directly painting on 2D images and then transferring the painting into the 3D implicit field.

4.5 Ablation Studies

Learnable sign indicator. We first inspect the effectiveness of the proposed learnable sign indicator in each vertex. Specifically, we set sign indicators as constant vertex normal without adjusting during the training process and evaluate

Config.	DTU 69			NeRF 360° Synthetic Lego		
	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓
w/o Optim. Sign Indicator	26.633	0.940	0.119	27.631	0.922	0.053
Only Distillation	26.599	0.936	0.144	25.606	0.901	0.081
Only Fine-Tuning	26.258	0.926	0.132	23.583	0.876	0.135
Full Model	**27.254**	**0.946**	**0.113**	**27.881**	**0.926**	**0.046**

Table 3. We perform ablation studies on the model design and training strategy with DTU Scan 69 and NeRF 360° Synthetic Lego.

the model both qualitatively and quantitatively. As demonstrated in Fig. 8 (a) and Tab. 3 (first row), online adjusting sign indicators consistently improves the image quality. By the way, we notice that the PSNR improvement on real data (DTU Scan 69) is more significant than the synthetic one (Lego). We consider that the mesh quality (and vertex normal) of real data is worse than the synthetic data due to sensor noises, which degrades the rendering quality, while the learnable sign indicator helps to mitigate this issue.

Distillation and fine-tuning training scheme. We then study the necessity of distillation and fine-tuning training scheme by ablating one of them during model training. As shown in Fig. 8 (a) and Tab. 3 (second and third row), by enabling distillation only, the rendered image is blurry than the full model ones. When using fine-tuning without distillation, the rendering result ends up with noticeable artifacts. These results suggest that both distillation and fine-tuning are indispensable when training our mesh-based representation.

Spatial-aware optimization in texture painting. We also evaluate the proposed spatial-aware optimization in the texture painting task and visualize the comparison in Fig. 8 (b). It is clear that when naïvely optimizing painting with a single image, the model will overfit to the specific viewpoint, and the change to the texture codes might break the appearance consistency, which results in visual artifacts when rendering the implicit field from the side view. By introducing a spatial-aware optimization mechanism, we successfully avoid such artifacts and obtain the modified field while maintaining other parts untouched.

5 Conclusion

We have proposed a novel mesh-based neural representation, which supports high-fidelity volume rendering, and flexible geometry and texture editing. Specifically, we encode the neural implicit field into a mesh scaffold, where each mesh vertex possesses learnable geometry and texture code for its neighboring local space. One limitation of our method is that we do not model fine-grained lighting effects such as shadowing and specular reflection of a certain lighting environment, which can be improved by introducing material and lighting estimation in future works. Besides, due to the reliance on mesh scaffold, we cannot represent objects that fails during reconstruction (*e.g.*, smoke or liquid).

Acknowledgment. This work was partially supported by NSF of China (No. 61932003, No. 62102356).

References

1. Akenine-Moller, T., Haines, E., Hoffman, N.: Real-Time Rendering. AK Peters/crc Press (2019) 4
2. Boss, M., Braun, R., Jampani, V., Barron, J.T., Liu, C., Lensch, H.: NeRD: Neural Reflectance Decomposition from Image Collections. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 12684–12694 (2021) 4
3. 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) 28
4. 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) 4, 24
5. Dellaert, F., Yen-Chen, L.: Neural Volume Rendering: NeRF And Beyond. arXiv preprint arXiv:2101.05204 (2020) 4
6. Deng, Y., Yang, J., Tong, X.: Deformed Implicit Field: Modeling 3D Shapes with Learned Dense Correspondence. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 10286–10296 (2021) 2, 4
7. Freepik: Flaticon. https://www.flaticon.com/ (2022), Accessed: 2022-07-19 8
8. Gao, D., Chen, G., Dong, Y., Peers, P., Xu, K., Tong, X.: Deferred Neural Lighting: Free-Viewpoint Relighting from Unstructured Photographs. ACM Transactions on Graphics (TOG) **39**(6), 1–15 (2020) 4
9. 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) 24
10. Gropp, A., Yariv, L., Haim, N., Atzmon, M., Lipman, Y.: Implicit Geometric Regularization for Learning Shapes. In: Proceedings of Machine Learning and Systems 2020, pp. 3569–3579 (2020) 7
11. Groueix, T., Fisher, M., Kim, V.G., Russell, B.C., Aubry, M.: A Papier-mâché Approach to Learning 3D Surface Generation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 216–224 (2018) 11
12. Guo, M., Fathi, A., Wu, J., Funkhouser, T.: Object-Centric Neural Scene Rendering. arXiv preprint arXiv:2012.08503 (2020) 2, 4
13. Izadi, S., Kim, D., Hilliges, O., Molyneaux, D., Newcombe, R., Kohli, P., Shotton, J., Hodges, S., Freeman, D., Davison, A., et al.: Kinectfusion: real-time 3D reconstruction and interaction using a moving depth camera. In: Proceedings of the 24th annual ACM symposium on User interface software and technology. pp. 559–568 (2011) 4
14. Jensen, R., Dahl, A., Vogiatzis, G., Tola, E., Aanæs, H.: Large Scale Multi-view Stereopsis Evaluation. In: 2014 IEEE Conference on Computer Vision and Pattern Recognition. pp. 406–413. IEEE (2014) 9
15. Kania, K., Yi, K.M., Kowalski, M., Trzciński, T., Tagliasacchi, A.: CoNeRF: Controllable Neural Radiance Fields. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 18623–18632 (2022) 2
16. Karsch, K., Hedau, V., Forsyth, D.A., Hoiem, D.: Rendering Synthetic Objects into Legacy Photographs. ACM Trans. Graph. **30**(6), 157 (2011) 4
17. Kazhdan, M.M., Bolitho, M., Hoppe, H.: Poisson Surface Reconstruction. In: Proceedings of Eurographics Symposium on Geometry Processing. pp. 61–70 (2006) 4

18. Kholgade, N., Simon, T., Efros, A., Sheikh, Y.: 3D Object Manipulation in a Single Photograph Using Stock 3D Models. ACM Transactions on Graphics (TOG) **33**(4), 1–12 (2014) 4
19. Knapitsch, A., Park, J., Zhou, Q.Y., Koltun, V.: Tanks and Temples: Benchmarking Large-Scale Scene Reconstruction. ACM Transactions on Graphics **36**(4) (2017) 27, 28
20. 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) 3, 4, 23, 26, 28
21. 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) 3, 4, 23
22. Liu, S., Li, T., Chen, W., Li, H.: Soft Rasterizer: A Differentiable Renderer for Image-based 3D Reasoning. The IEEE International Conference on Computer Vision (ICCV) (Oct 2019) 4
23. Liu, S., Zhang, X., Zhang, Z., Zhang, R., Zhu, J.Y., Russell, B.: Editing Conditional Radiance Fields. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 5773–5783 (2021) 2, 4, 8
24. 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) 23
25. Lorensen, W.E., Cline, H.E.: Marching Cubes: A High Resolution 3D Surface Construction Algorithm. ACM SIGGRAPH Computer Graphics **21**(4), 163–169 (1987) 5, 20
26. Luo, J., Huang, Z., Li, Y., Zhou, X., Zhang, G., Bao, H.: NIID-Net: Adapting Surface Normal Knowledge for Intrinsic Image Decomposition in Indoor Scenes. IEEE Transactions on Visualization and Computer Graphics **26**(12), 3434–3445 (2020) 4
27. Mildenhall, B., Srinivasan, P.P., Tancik, M., Barron, J.T., Ramamoorthi, R., Ng, R.: NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis. In: European Conference on Computer Vision. pp. 405–421. Springer (2020) 3, 4, 5, 6, 10
28. 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) 2, 24
29. Murez, Z., As, T.v., Bartolozzi, J., Sinha, A., Badrinarayanan, V., Rabinovich, A.: Atlas: End-to-End 3D Scene Reconstruction from Posed Images. In: European Conference on Computer Vision. pp. 414–431. Springer (2020) 4
30. Niemeyer, M., Geiger, A.: Giraffe: Representing Scenes as Compositional Generative Neural Feature Fields. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 11453–11464 (2021) 4
31. Oechsle, M., Mescheder, L., Niemeyer, M., Strauss, T., Geiger, A.: Texture Fields: Learning Texture Representations in Function Space. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 4531–4540 (2019) 4
32. Oechsle, M., Peng, S., Geiger, A.: UNISURF: Unifying Neural Implicit Surfaces and Radiance Fields for Multi-View Reconstruction. In: International Conference on Computer Vision (ICCV) (2021) 4
33. Ost, J., Laradji, I., Newell, A., Bahat, Y., Heide, F.: Neural Point Light Fields. arXiv preprint arXiv:2112.01473 (2021) 4, 23

34. Park, K., Sinha, U., Barron, J.T., Bouaziz, S., Goldman, D.B., Seitz, S.M., Martin-Brualla, R.: Nerfies: Deformable Neural Radiance Fields. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 5865–5874 (2021) 2
35. Park, K., Sinha, U., Hedman, P., Barron, J.T., Bouaziz, S., Goldman, D.B., Martin-Brualla, R., Seitz, S.M.: HyperNeRF: A Higher-Dimensional Representation for Topologically Varying Neural Radiance Fields. ACM Trans. Graph. **40**(6) (dec 2021) 2
36. Peng, S., Zhang, Y., Xu, Y., Wang, Q., Shuai, Q., Bao, H., Zhou, X.: Neural body: Implicit Neural Representations with Structured Latent Codes for Novel View Synthesis of Dynamic Humans. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 9054–9063 (2021) 4
37. Pérez, P., Gangnet, M., Blake, A.: Poisson Image Editing. ACM Trans. Graph. **22**(3), 313–318 (2003) 4
38. 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) 6, 26
39. Reiser, C., Peng, S., Liao, Y., Geiger, A.: KiloNeRF: Speeding up Neural Radiance Fields with Thousands of Tiny MLPs. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 14335–14345 (2021) 4, 5, 6
40. Rematas, K., Ferrari, V.: Neural Voxel Renderer: Learning an Accurate and Controllable Rendering Tool. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 5417–5427 (2020) 4
41. Riegler, G., Koltun, V.: Free View Synthesis. In: European Conference on Computer Vision. pp. 623–640. Springer (2020) 4
42. Riegler, G., Koltun, V.: Stable View Synthesis. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 12216–12225 (2021) 4
43. Sara Fridovich-Keil and Alex Yu, Tancik, M., Chen, Q., Recht, B., Kanazawa, A.: Plenoxels: Radiance Fields without Neural Networks. In: CVPR (2022) 2, 4, 23, 24, 26
44. Schönberger, J.L., Frahm, J.: Structure-from-Motion Revisited. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition. pp. 4104–4113. IEEE Computer Society (2016) 4
45. Schwarz, K., Liao, Y., Niemeyer, M., Geiger, A.: GRAF: Generative Radiance Fields for 3D-Aware Image Synthesis. Advances in Neural Information Processing Systems **33**, 20154–20166 (2020) 2
46. Shetty, R.R., Fritz, M., Schiele, B.: Adversarial Scene Editing: Automatic Object Removal from Weak Supervision. Advances in Neural Information Processing Systems **31** (2018) 4
47. Sorkine, O., Alexa, M.: As-rigid-as-possible Surface Modeling. In: Symposium on Geometry Processing. vol. 4, pp. 109–116 (2007) 7, 9, 22
48. Srinivasan, P.P., Deng, B., Zhang, X., Tancik, M., Mildenhall, B., Barron, J.T.: NeRV: Neural Reflectance and Visibility Fields for Relighting and View Synthesis. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 7495–7504 (2021) 4, 8
49. Sun, J., Xie, Y., Chen, L., Zhou, X., Bao, H.: NeuralRecon: Real-time Coherent 3D Reconstruction from Monocular Video. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 15598–15607 (2021) 4

50. Sun, J., Wang, X., Zhang, Y., Li, X., Zhang, Q., Liu, Y., Wang, J.: FENeRF: Face Editing in Neural Radiance Fields. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 7672–7682 (2022) 4
51. Tancik, M., Casser, V., Yan, X., Pradhan, S., Mildenhall, B., Srinivasan, P.P., Barron, J.T., Kretzschmar, H.: Block-NeRF: Scalable Large Scene Neural View Synthesis. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 8248–8258 (2022) 2
52. Thies, J., Zollhöfer, M., Nießner, M.: Deferred Neural Rendering: Image Synthesis Using Neural Textures. ACM Transactions on Graphics (TOG) **38**(4), 1–12 (2019) 4
53. Umeyama, S.: Least-Squares Estimation of Transformation Parameters Between Two Point Patterns. IEEE Transactions on Pattern Analysis & Machine Intelligence **13**(04), 376–380 (1991) 9, 22
54. Waechter, M., Moehrle, N., Goesele, M.: Let there be color! Large-Scale Texturing of 3D Reconstructions. In: European Conference on Computer Vision. pp. 836–850. Springer (2014) 4
55. 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) 3, 4, 5, 6, 10, 11, 20, 21
56. Wang, Q., Wang, Z., Genova, K., Srinivasan, P.P., Zhou, H., Barron, J.T., Martin-Brualla, R., Snavely, N., Funkhouser, T.: IBRNet: Learning Multi-view Image-based Rendering. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 4690–4699 (2021) 2
57. Wang, T.Y., Su, H., Huang, Q., Huang, J., Guibas, L.J., Mitra, N.J.: Unsupervised Texture Transfer from Images to Model Collections. ACM Trans. Graph. **35**(6), 177–1 (2016) 4
58. 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) 9, 24
59. Xiang, F., Xu, Z., Hasan, M., Hold-Geoffroy, Y., Sunkavalli, K., Su, H.: NeuTex: Neural Texture Mapping for Volumetric Neural Rendering. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 7119–7128 (2021) 4, 8, 9, 10, 11, 13
60. Xiangli, Y., Xu, L., Pan, X., Zhao, N., Rao, A., Theobalt, C., Dai, B., Lin, D.: CityNeRF: Building NeRF at City Scale. arXiv preprint arXiv:2112.05504 (2021) 2
61. Xie, C., Park, K., Martin-Brualla, R., Brown, M.: FiG-NeRF: Figure-ground Neural Radiance Fields for 3D Object Category Modelling. In: 2021 International Conference on 3D Vision (3DV). pp. 962–971. IEEE (2021) 4
62. Xu, Q., Tao, W.: Multi-Scale Geometric Consistency Guided Multi-View Stereo. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition. pp. 5483–5492 (2019) 4
63. Yang, B., Zhang, Y., Li, Y., Cui, Z., Fanello, S., Bao, H., Zhang, G.: Neural Rendering in a Room: Amodal 3D Understanding and Free-Viewpoint Rendering for the Closed Scene Composed of Pre-Captured Objects. ACM Trans. Graph. **41**(4), 101:1–101:10 (Jul 2022) 4
64. Yang, B., Zhang, Y., Xu, Y., Li, Y., Zhou, H., Bao, H., Zhang, G., Cui, Z.: Learning Object-Compositional Neural Radiance Field for Editable Scene Rendering. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 13779–13788 (2021) 2, 4

65. Yariv, L., Gu, J., Kasten, Y., Lipman, Y.: Volume Rendering of Neural Implicit Surfaces (2021) 3, 4, 6
66. 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) 4, 6, 8, 9, 10, 21
67. Yen-Chen, L., Florence, P., Barron, J.T., Rodriguez, A., Isola, P., Lin, T.Y.: iNeRF: Inverting Neural Radiance Fields for Pose Estimation. In: 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). pp. 1323–1330. IEEE (2021) 4
68. Yuan, Y.J., Sun, Y.T., Lai, Y.K., Ma, Y., Jia, R., Gao, L.: NeRF-Editing: Geometry Editing of Neural Radiance Fields. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 18353–18364 (June 2022) 4
69. Zhang, J., Liu, X., Ye, X., Zhao, F., Zhang, Y., Wu, M., Zhang, Y., Xu, L., Yu, J.: Editable Free-viewpoint Video Using a Layered Neural Representation. ACM Transactions on Graphics (TOG) **40**(4), 1–18 (2021) 2, 4
70. Zhang, K., Riegler, G., Snavely, N., Koltun, V.: NeRF++: Analyzing and Improving Neural Radiance Fields. arXiv preprint arXiv:2010.07492 (2020) 28
71. Zhang, X., Srinivasan, P.P., Deng, B., Debevec, P., Freeman, W.T., Barron, J.T.: NeRFactor: Neural Factorization of Shape and Reflectance Under an Unknown Illumination. ACM Transactions on Graphics (TOG) **40**(6), 1–18 (2021) 4, 8
72. Zhao, B., Yang, B., Li, Z., Li, Z., Zhang, G., Zhao, J., Yin, D., Cui, Z., Bao, H.: Factorized and Controllable Neural Re-Rendering of Outdoor Scene for Photo Extrapolation. In: Proceedings of the 30th ACM International Conference on Multimedia (2022) 4
73. Zhou, Q.Y., Park, J., Koltun, V.: Open3D: A Modern Library for 3D Data Processing. arXiv preprint arXiv:1801.09847 (2018) 9

Supplementary Material

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.I. To begin with, we first extract a triangle mesh with marching cubes [25] from NeuS's [55] 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.* [55] 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.

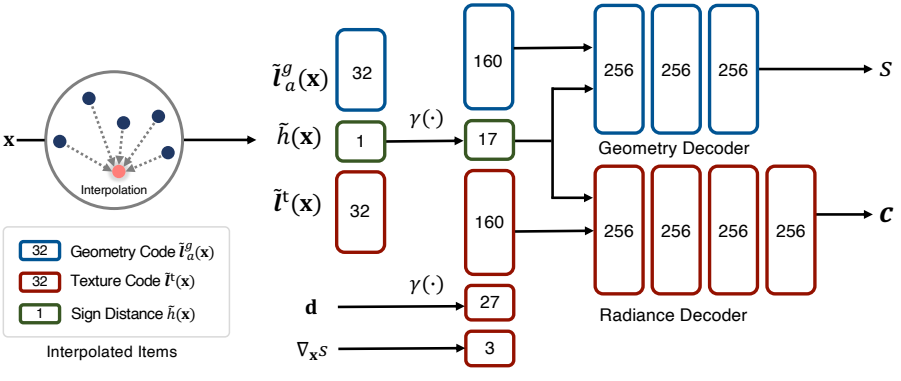


Fig. I. The model architecture of NeuMesh.

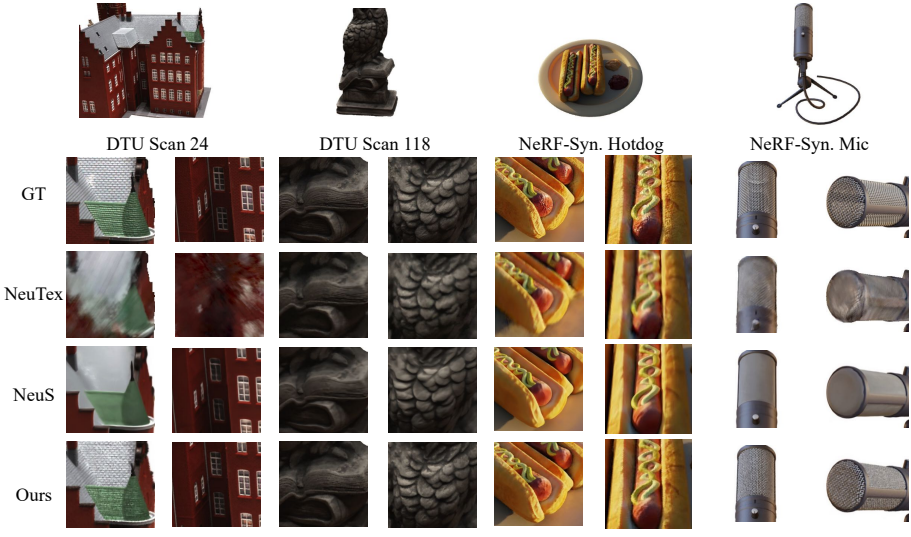


Fig. J. We show more comparison of rendering quality on the DTU dataset and the NeRF 360° Synthetic dataset. Our rendering results show better appearance details than NeuS and NeuTex (*e.g.*, the roof at DTU Scan 37, and the metal grids at NeRF-Synthetic Mic).

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.*, NeuS [55]). 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 [66,55] 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.

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 [53] between point correspondences, and then feed the point residual to ARAP [47], 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 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

Fig. K. 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.

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 [20,43] or point-cloud-based [33] 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 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.* [21] 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) [24], 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

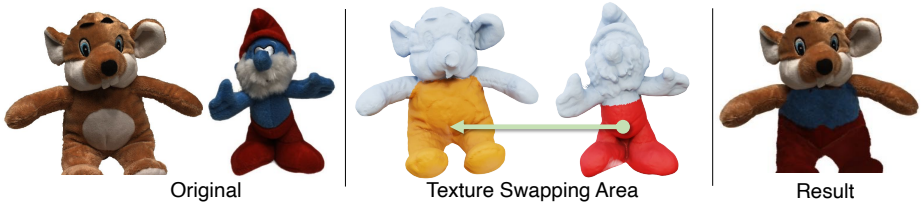


Fig. L. Texture swapping with different geometry.

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 [43,9] , multiresolution hash encoding [28] or MVS priors [4]. Besides, we rely on 3D modeling software to select vertices for the region of interest, which can be replaced with some semantic annotation approaches [58] 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 features 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. J. 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 J).

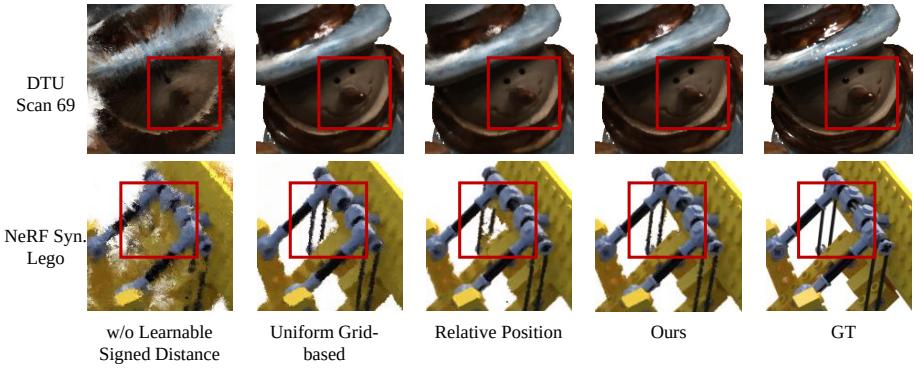


Fig. M. We present visual comparison to alternative designs.

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

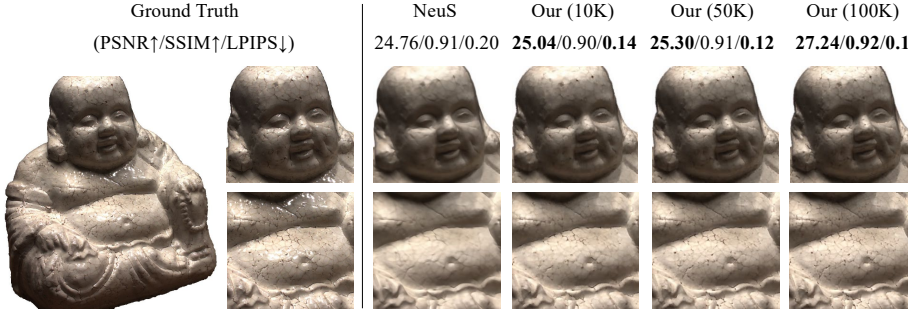
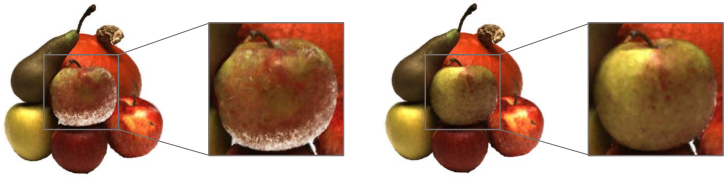


Fig. N. We analyze the impact of vertex numbers on the rendering quality by training with 10K / 50K / 100K vertices.

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. N, the metric quality of rendered images are slightly affected 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. M (first column) and Tab. D (first row),



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

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

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 [20] or Plenoxel [43]). 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. M (second column) and Tab. D (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 [38]. 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. M (third column) and Tab. D (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. O, the rendered apple of the naïve 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.

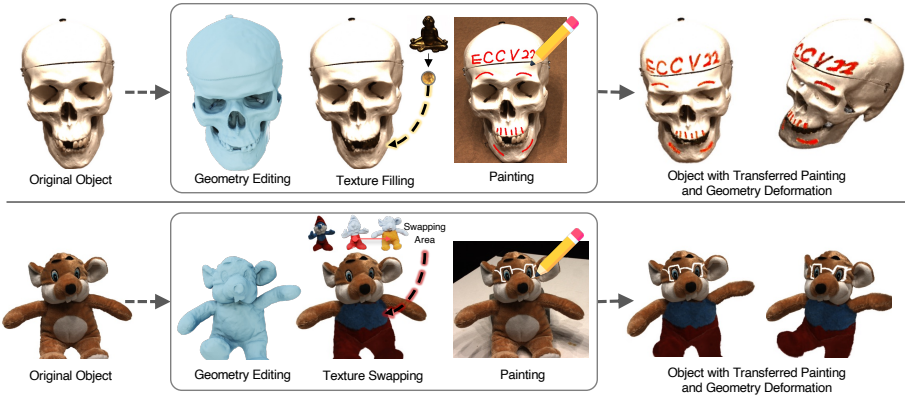


Fig. P. We show examples of hybrid object editing by combining multiple editing operations.

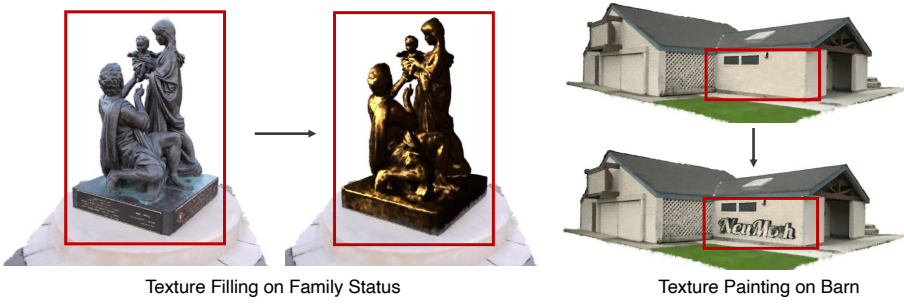


Fig. Q. Texture editing of large-scale scenes on the Tanks&Temple [19] dataset.

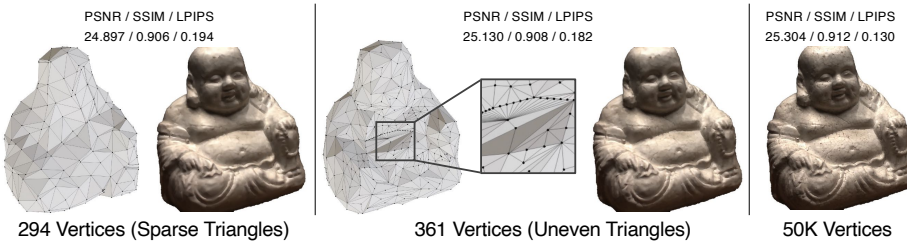


Fig. R. Rendering quality under sparse and uneven triangulation.

Hybrid object editing. To demonstrate the editing flexibility of our method, we show examples of hybrid object editing in Fig. P 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++ [70]'s parameterization to handle unbounded backgrounds, or use pre-computed segmentation masks like in NSVF and IDR. Here we show two texture editing examples (see Fig. Q) on the Tanks&Temple dataset [19] with foreground segmentation provided by NSVF [20].

Influence of triangle quality. Our method can still deliver reasonable rendering quality with locally sparse/uneven triangulation (see Fig. R). 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.* [3]).