

# Volumetric Surfaces: Representing Fuzzy Geometries with Layered Meshes

## Supplementary Material

In this supplementary material, we provide additional architectural and technical details (Section S1), further visualizations (Section S2), comprehensive per-scene results (Section S3), and an in-depth analysis of performance on fully solid geometries (Section S4).

### S1. Additional Technical Information

**$\beta$  scheduling details:** During training, the  $\beta$  parameter is controlled by the scheduling of  $v$  as  $\beta = e^{10v}$ . During the main surface training phase,  $v$  linearly transitions from  $v_1 = 0.3$  to  $v_2 = 0.7$ . During the training of  $k$ -SDF, it further progresses from  $v_2 = 0.7$  to  $v_3 = 1.0$ . At  $\beta_2$ , the logistic distribution standard deviation is approximately 0.001. We use this value to initialize offsets as constants ( $\Delta o$ ). By the end of implicit surface training ( $\beta_3$ ), it decreases to 0.00008, resulting in fully peaked densities.

**$k$ -SDF Architecture:** We encode 3D points using the trainable positional encoding from Rosu and Behnke [44], followed by a small MLP with three layers of 32 features each. Hidden layers employ GELU activations, while the final layer uses a linear activation to output the signed distance  $d$  (our main SDF) and a geometric feature vector  $\mathbf{z}$ . We predict relative offsets using tiny MLP heads (a single layer with 32 units) with independent parameters, taking only  $\mathbf{z}$  as input. This ensures that model complexity scales with the number of surfaces. To enforce the sign of the predicted offset, we apply a `softplus` activation multiplied by the desired sign. Finally, we compute the final ordered offsets by performing a cumulative sum over the predicted relative offsets, separately for negative and positive values.

**Volumetric Appearance Architectures:** We model RGB and transparency as two networks with identical architectures, differing only in their output dimensions (3 for RGB and 1 for transparency). Both models encode 3D points using the trainable positional encoding from Rosu and Behnke [44], followed by an MLP with three layers of 128, 128, and 64 features, respectively. Its input consists of the encoded position, a spherical harmonics encoding (with degree 3) of the view direction  $\mathbf{v}$ , the normal vector of the rendered SDF  $\mathbf{n}$  and the geometric feature vector  $\mathbf{z}$  predicted by  $k$ -SDF. Normals are computed as the normalized gradients of the SDFs; gradients are computed with finite differences,  $\epsilon = 10^{-4}$ . Hidden layers use GELU activations, while the final layer applies a Sigmoid activation to produce outputs in the range  $[0, 1]$ .

**Neural Textures Architecture:** During the mesh texturing

phase, we use separate neural texture models for RGB and transparency per mesh. We encode 2D UV coordinates using the trainable positional encoding from Müller et al. [35], followed by a small MLP with two layers of 64 features each. Hidden layers use ReLU activations, while the final layer applies a linear activation to output per-channel spherical harmonics (SH) coefficients of degree 3, which are then decoded with view direction  $\mathbf{v}$ .

**Points sampling:** During volumetric rendering the number of uniformly sampled points per ray in the foreground area of the scene is 64. On top of these, 32 points are added with importance sampling. Additionally, if a scene is unbounded (e.g. DTU), we sample 32 additional points in contracted space [3]. Rays batch size is defined w.r.t. a target number of sample points which is up to a maximum of  $512 \times 64 \times 32$  points.

**Data handling:** We use `MVDatasets` [11] to load datasets, manage training loop pixel iterators, and perform ray casting.

### S2. Additional Visualizations

We provide additional qualitative comparisons on our evaluation scenes of the DTU [21] dataset. Additionally, we provide visualizations of per-surface rendering before alpha blending (Figure S1 and Figure S3) to illustrate how each layer, based on its position and opacity, contributes with its view-dependent appearance model to the final image. Finally, we visualize results from Table 2. Figure S5 presents a qualitative comparison between a render of our 7-Mesh model, 3DGS [24], and 3DGS-75K. Figure S6 compares our 5-Mesh model to MobileNeRF [7].

#### S2.1. Transparency Attenuation

We introduced transparency attenuation in Section 4.2 to reduce visual artifacts at object boundaries. Figure S2, cropped from our ablation experiments (Section 5.1), highlights its significance in our method.

### S3. Per-scene Results

Table S1, Table S2 and Table S3 present the per-scene values that are averaged in Table 3.

### S4. Fully Solid Scenes

Although not our targeted use case, we tested our method on the fully solid scenes of the NeRF-Synthetic dataset [34], which lacks fuzzy objects. As noted in Section 5.2, our advantages in these scenes are marginal. While we outperform PermutoSDF [44], our quality remains behind other baselines.

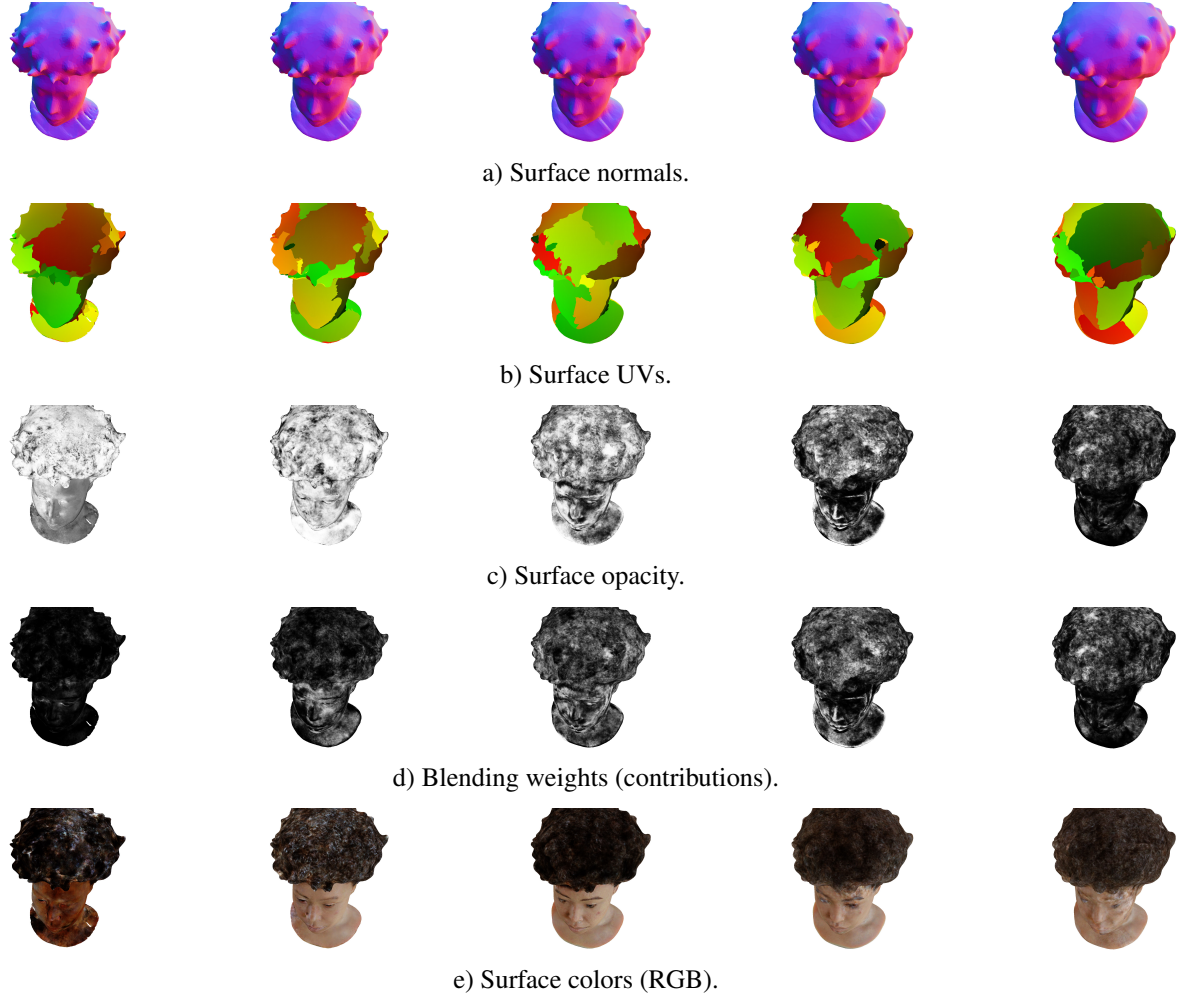


Figure S1. Visualization of render buffers from our 5-Mesh model. Layers order: left to right is inner to outer. Individual layer color and alpha buffers are blended as described in Section 4.2. Results on the *khady* scene from Shelly [56].

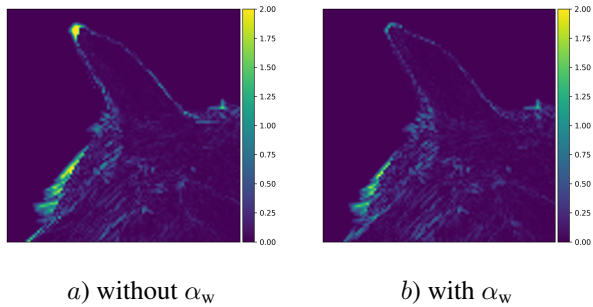


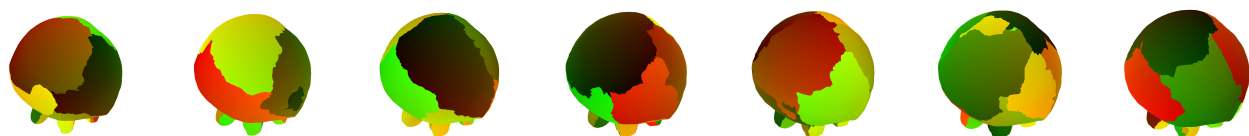
Figure S2. (a) Rendering error crop (averaged over color channels) without and (b) with transparency-decay, resulting in a 2.13 dB PSNR gain. Scene from the Shelly [56].

as spatial sampling is key to accurately capturing these effects. By favoring smoother surfaces, our method tends to reconstruct overly simplified geometry in under-observed areas (Figure S7). Fully solid scenes can be optimally modeled as a single surface. However, SDF-based methods struggle in handling thin structures, as optimization often fails to reconstruct them reliably (e.g., BakedSDF [60], BOG [43]). Our surface smoothing, combined with view-dependent transparency, often leads to thin structures being reconstructed as view-dependent effects (Figure S8). As a result, our model tends to overfit training views, leading to a larger quality gap between training and test views (Table S4).

We model fuzzy surfaces by optimizing sample distribution rather than reconstructing high-frequency geometric details,



a) Surface normals.



b) Surface UVs.



c) Surface opacity.



d) Blending weights (contributions).



e) Surface colors (RGB).

Figure S3. Visualization of render buffers from our 7-Mesh model. Layers order: left to right is inner to outer. Individual layer colors and alpha buffers are blended as described in [Section 4.2](#). Results our custom *plushy* scene.



Figure S4. Qualitative comparison of our 7-Mesh model with PermutoSDF [44] and 3DGS [24]. Scenes from the DTU dataset [21].

Table S1. Per-scene results for baselines. Refer to Table 3 for averaged results. Metrics not provided are denoted with “—”. PermutoSDF trained until densities are fully peaked ( $\phi_{\beta_3}$ ). The *hairy monkey* scene is from Sharma et al. [47].

Dataset	Scene	PermutoSDF [44]			3DGS [24]			MobileNeRF [7]		
		PSNR $\uparrow$	SSIM $\uparrow$	LPIPS $\downarrow$	PSNR $\uparrow$	SSIM $\uparrow$	LPIPS $\downarrow$	PSNR $\uparrow$	SSIM $\uparrow$	LPIPS $\downarrow$
Shelly [56]	<i>fernvas</i>	28.42	0.953	0.078	34.82	0.986	0.040	29.06	0.957	0.088
	<i>horse</i>	34.68	0.993	0.040	41.45	0.997	0.038	33.31	0.988	0.065
	<i>khady</i>	26.22	0.879	0.226	30.54	0.924	0.187	26.42	0.877	0.228
	<i>kitten</i>	30.91	0.971	0.093	38.17	0.991	0.050	30.22	0.968	0.098
	<i>pug</i>	29.48	0.953	0.168	35.96	0.983	0.089	28.57	0.927	0.197
	<i>woolly</i>	29.39	0.949	0.167	31.71	0.969	0.130	28.20	0.919	0.221
Custom	<i>hairy monkey</i>	33.67	0.977	0.194	37.67	0.990	0.142	30.25	0.949	0.200
	<i>plushy</i>	32.94	0.945	0.192	37.02	0.975	0.153	31.53	0.934	0.190
DTU [21]	<i>scan 105</i>	34.78	0.985	0.124	35.50	0.984	0.102	—	—	—
	<i>scan 83</i>	37.84	0.991	0.072	40.61	0.994	0.070	—	—	—



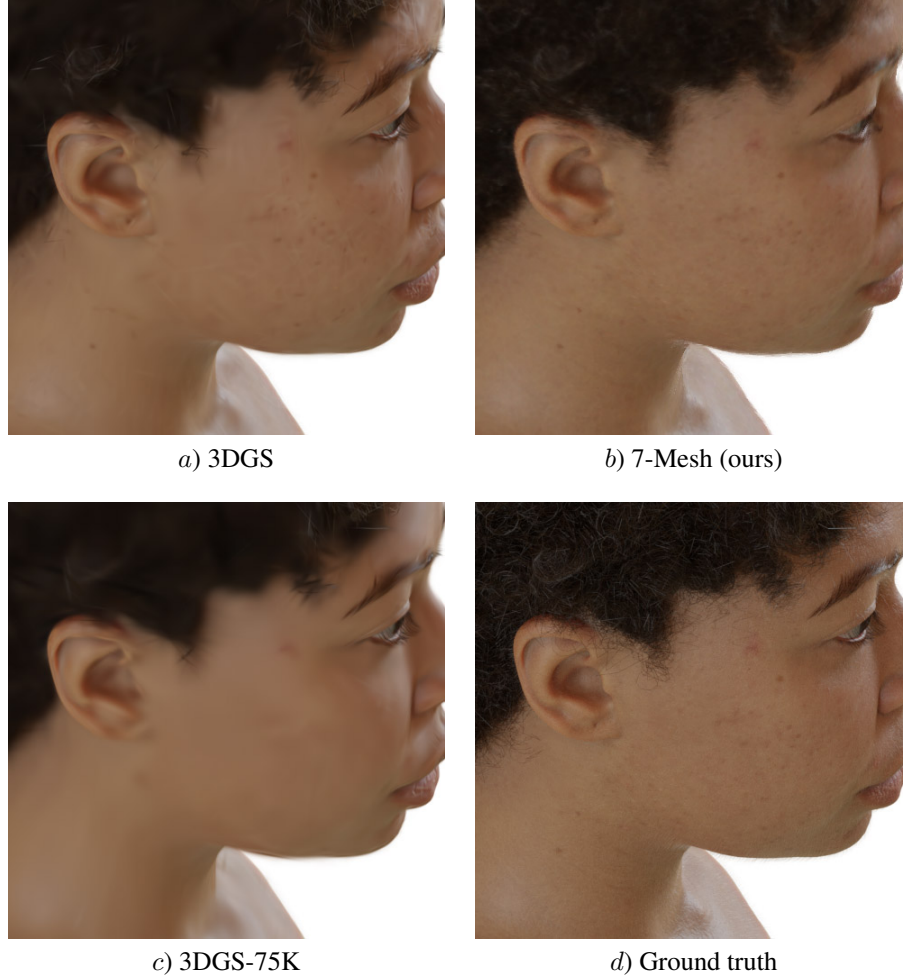


Figure S5. 3DGS [24] demonstrates superior performance in modeling thin structures but is significantly less effective in representing large, textured areas. Our method renders faster than 3DGS-75K on mobile devices. Results on the *khady* scene from Shelly [56]. Quantitative results are in Table 2.

Table S2. Our per-scene results. The *hairy monkey* scene is from Sharma et al. [47]. Refer to Table 3 for averaged results.

Dataset	Scene	3-Mesh			5-Mesh		
		PSNR $\uparrow$	SSIM $\uparrow$	LPIPS $\downarrow$	PSNR $\uparrow$	SSIM $\uparrow$	LPIPS $\downarrow$
Shelly [56]	<i>fern vase</i>	32.41	0.985	0.066	33.63	0.988	0.064
	<i>horse</i>	38.34	0.998	0.038	39.78	0.998	0.034
	<i>khady</i>	29.78	0.938	0.193	29.88	0.941	0.194
	<i>kitten</i>	35.84	0.991	0.078	36.85	0.992	0.076
	<i>pug</i>	33.72	0.983	0.138	34.25	0.985	0.132
	<i>woolly</i>	30.26	0.973	0.175	31.12	0.978	0.162
Custom	<i>hairy monkey</i>	35.59	0.984	0.178	35.90	0.985	0.179
	<i>plushy</i>	34.41	0.957	0.164	34.99	0.965	0.164
DTU [21]	<i>scan 105</i>	34.77	0.980	0.120	35.40	0.982	0.106
	<i>scan 83</i>	38.05	0.990	0.064	38.34	0.990	0.063



Figure S6. Our method surpasses MobileNeRF [7] in modeling volumetric hair while also achieving superior performance on flat surfaces. Results on *hairy monkey* from QuadFields [47], our custom *plushy* scene and *khady* from Shelly [56]. Quantitative results are in Table 2.



Figure S7. Visualization of surface normals from our 7-Mesh model. Scene from NeRF-Synthetic [34].

Table S3. Our per-scene results. The *hairy monkey* scene is from Sharma et al. [47]. Refer to Table 3 for averaged results.

Dataset	Scene	7-Mesh			9-Mesh		
		PSNR $\uparrow$	SSIM $\uparrow$	LPIPS $\downarrow$	PSNR $\uparrow$	SSIM $\uparrow$	LPIPS $\downarrow$
Shelly [56]	<i>fern vase</i>	34.55	0.990	0.062	34.64	0.991	0.062
	<i>horse</i>	40.05	0.998	0.033	39.32	0.998	0.034
	<i>khady</i>	29.97	0.942	0.194	29.96	0.943	0.195
	<i>kitten</i>	37.11	0.993	0.074	37.05	0.993	0.074
	<i>pug</i>	34.25	0.985	0.132	34.24	0.985	0.133
	<i>woolly</i>	31.04	0.978	0.158	31.05	0.978	0.160
Custom	<i>hairy monkey</i>	36.09	0.987	0.177	36.14	0.987	0.175
	<i>plushy</i>	35.18	0.967	0.162	35.35	0.969	0.160
DTU [21]	<i>scan 105</i>	35.50	0.982	0.106	35.54	0.983	0.105
	<i>scan 83</i>	38.04	0.991	0.062	38.81	0.991	0.062

Table S4. Results averaged across test scenes. In solid scenes, our method outperforms PermutoSDF [44] (see Figure S8) but lags behind other baselines. We explain this behavior in Section S4. Methods marked with a  $\star$  show results taken from original papers. PermutoSDF trained until densities are fully peaked ( $\phi_{\beta_3}$ ). Metrics not provided by a baseline are denoted with “—”.

Method	NeRF-Synthetic [34]					
	Training			Test		
	PSNR $\uparrow$	SSIM $\uparrow$	LPIPS $\downarrow$	PSNR $\uparrow$	SSIM $\uparrow$	LPIPS $\downarrow$
3DGS [24]	36.76	0.991	0.030	33.23	0.981	0.037
Instant-NGP [35] $\star$	—	—	—	33.18	—	—
PermutoSDF [44]	29.31	0.975	0.057	28.05	0.966	0.065
AdaptiveShells [56] $\star$	—	—	—	31.84	0.957	0.056
QuadFields [47] $\star$	—	—	—	31.00	0.952	0.069
MobileNeRF [7] $\star$	—	—	—	30.90	0.947	0.060
3-Mesh	32.40	0.983	0.060	28.50	0.958	0.083
5-Mesh	33.23	0.986	0.055	28.77	0.959	0.081
7-Mesh	33.31	0.986	0.056	28.88	0.960	0.081
9-Mesh	33.19	0.986	0.057	28.79	0.960	0.082

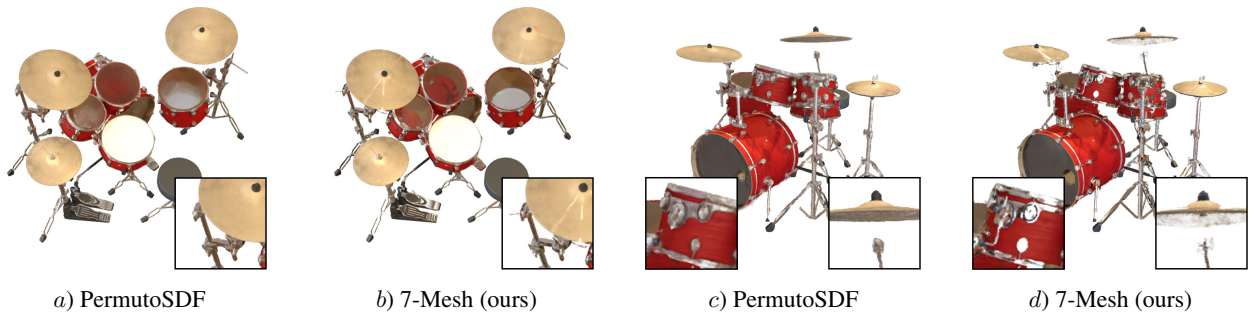


Figure S8. Qualitative comparison of our method with PermutoSDF [44]. Renderings (a) and (b) are from a training view. Renderings (c) and (d) are from a test view. Scene from NeRF-Synthetic [34].