

Volumetric Surfaces: Representing Fuzzy Geometries with Layered Meshes

Stefano Esposito¹ Anpei Chen¹ Christian Reiser¹ Samuel Rota Bulò² Lorenzo Porzi²
Katja Schwarz² Christian Richardt² Michael Zollhöfer² Peter Kotschieder² Andreas Geiger¹

¹University of Tübingen ²Meta Reality Labs



Figure 1. Our Volumetric Surfaces representation (b) consists of k lightweight, semi-transparent mesh shells that efficiently render fuzzy geometries with a limited number of samples ($3 \leq k \leq 9$) via rasterization. Our image quality surpasses that of surface-based methods (a) and approaches the quality of 3D Gaussian splatting (c). Unlike splatting methods, our sorting-free representation enables faster rendering, particularly on low-power or mobile graphics hardware. Project page: <https://autonomousvision.github.io/volsurfs>.

Abstract

High-quality view synthesis relies on volume rendering, splatting, or surface rendering. While surface rendering is typically the fastest, it struggles to accurately model fuzzy geometry like hair. In turn, alpha-blending techniques excel at representing fuzzy materials but require an unbounded number of samples per ray (P1). Further overheads are induced by empty space skipping in volume rendering (P2) and sorting input primitives in splatting (P3). We present a novel representation for real-time view synthesis where the (P1) number of sampling locations is small and bounded, (P2) sampling locations are efficiently found via rasterization, and (P3) rendering is sorting-free. We achieve this by representing objects as semi-transparent multi-layer meshes rendered in a fixed order. First, we model surface layers as signed distance function (SDF) shells with optimal spacing learned during training. Then, we bake them as meshes and fit UV textures. Unlike single-surface methods, our multi-layer representation effectively models fuzzy objects. In contrast to volume and splatting-based methods, our approach enables real-time rendering on low-power laptops and smartphones.

1. Introduction

Real-time rendering on mobile devices is challenging due to limited processing power, memory, and thermal constraints. Recent methods for real-time view synthesis can be categorized according to their rendering paradigm. On the one hand, we have surface-based methods like BakedSDF [60] or BOG [43], where rendering a pixel requires only reading appearance data from a single sampling location along the ray. On the other hand, we have volume-based methods like SMERF [10] or 3DGS [24], where rendering a pixel requires reading data from multiple sample locations along the ray. As a result, surface-based methods are generally faster than volume-based techniques, which struggle to achieve interactive frame rates on mobile devices [43]. While recent surface-based methods are also capable of representing thin structures like individual strands of grass [43, 64], they still lag behind volume-based methods, especially when it comes to reconstructing highly fuzzy geometry like hair or plush. Even if possible from a reconstruction perspective, a purely surface-based representation might be too memory-

inefficient for representing extremely fuzzy objects [4]. Towards our goal of real-time view synthesis of fuzzy objects on mobile devices, we therefore focus on finding a more efficient volumetric formulation. A key factor in the efficiency of volumetric representations is the number of samples required along a ray. State-of-the-art methods, such as SMERF and 3DGS, often require tens or even hundreds of samples per ray. Additionally, the intrinsic characteristics of the rendering algorithm impact performance. In volume rendering, traversing an extra data structure to skip empty space increases memory bandwidth usage and results in suboptimal thread coherence. For splatting, primitives must be sorted by their distance from the camera, a task that is challenging to implement efficiently on platforms with limited GPGPU capabilities. For these reasons, such approaches are not suited for current mobile hardware, e.g. low-cost smartphones. To address these limitations, we propose a differentiable representation that bounds the number of sampling points per ray to a small range (three to nine) and is sorting-free. This allows us, unlike 3DGS or SMERF, to achieve real-time rendering on low-cost smartphones.

Textured shells [23, 26, 27] have long been used in computer graphics to simulate fuzzy surfaces while minimizing geometric complexity. These are modeled as concentric, *uniformly spaced*, semi-transparent layers, and are still widely used in modern game production. Inspired by these techniques, our representation learns *adaptively spaced* layers around a reconstructed object via gradient-based optimization. This allows for rasterization in a fixed order, from outermost to innermost, without the need for expensive empty space skipping or sorting during rendering. However, it is non-trivial to learn the optimal spacing between individual layers; we tackle this by representing them as separate signed distance functions (SDFs) during training. Before training all layers, we start by training a single opaque SDF that prevents degenerate solutions. We then add the additional layers, constraining them from intersecting one another. To further increase the expressivity of our representation, while keeping a low number of layers, we make each layer’s transparency depend on the viewing direction. All layers are optimized toward smooth solutions so that each can be baked into a lightweight mesh for efficient hardware-accelerated rasterization. The simplicity of our meshes enables computing a high-quality UV parameterization, which is often problematic for highly complex, monolithic meshes [43]. Finally, we optimize UV textures of spherical harmonics (SH) coefficients on the fixed geometry defined by our meshes.

We demonstrate how our approach renders significantly faster than volume-based and splatting-based methods, enabling high frame rates on a wide range of commodity devices, while simultaneously being more capable at representing fuzzy objects than single-surface approaches.

2. Related Work

Real-Time View Synthesis: Neural radiance fields (NeRFs) achieved an impressive leap of quality by fitting a 3D scene representation via differentiable volume rendering to multi-view images [34]. NeRFs represent the scene implicitly as a multi-layer perceptron (MLP) [6, 33, 39]. Evaluating an MLP is arithmetically intensive, leading to slow inference. To overcome this, a number of works explore faster representations based on 3D grids [12, 14, 19, 29, 41, 62], triplanes [5, 10, 42], hash grids [35, 49], or explicit primitives [2, 8, 24, 45, 57]. 3D Gaussians have gained traction lately due to their fast training and rendering [24]. While these representations enhanced efficiency, rendering each pixel may still require a large number of samples per ray.

DONeRF reduces sample count by only sampling around depth values predicted by a separate MLP, which, however, requires access to ground-truth depth maps [36]. AdaNeRF explicitly introduces sampling sparsity during the course of training [25]. HybridNeRF introduces a hybrid surface–volume representation that encourages surfaces over volumetric rendering [51]. AdaptiveShells restrict sampling to a small shell around the surface [56]. This shell is represented by a triangle mesh, rasterized to find the sampling range. Unlike AdaptiveShells, which uses volume rendering, our method finds all sampling locations via rasterization, enabling the use of 2D textures instead of 3D volume textures. Additionally, while AdaptiveShells learns a single SDF with a spatially-varying kernel, we learn multiple SDFs. NDRF [53] extracts two mesh layers via marching cubes with varying density thresholds and rasterizes them to feature vectors that are decoded by a convolutional neural network (CNN). In turn, we store view-dependent opacities and colors as SH textures. Similar to us, QuadFields also aggregate opacities and view-dependent colors from multiple ray-triangle intersections [47]. However, these methods require significantly larger meshes, intersection calculations for all triangles along a ray, and a costly sorting process. In contrast, our formulation enables blending in a fixed order.

Another line of work aims for a single sample per ray. MobileNeRF [7] represents the scene with a coarse proxy mesh equipped with a binary opacity mask to increase expressivity. BakedSDF [60], NeRF2Mesh [50], NeRFMeshing [40] and BOG [43] first perform a 3D reconstruction of the scene, convert their respective training-time representation into a mesh, and then fit an appearance model to the mesh. While these single-surface approaches are faster, they struggle to reconstruct fuzzy geometries. Even with accurate reconstruction from multi-view data, representing individual hairs explicitly would require substantial memory [4].

Gaussian frosting [18] anchors Gaussians on a tight surface shell to enhance editability. Like our approach, GaussianShellMaps [1] utilize a multi-layer mesh representation.

However, unlike our approach, GaussianShellMaps uses fixed shells rather than adaptively learning them, and predicts per-shell Gaussians. Further these methods still fall short of achieving real-time rendering on budget smartphones due to their reliance on splatting techniques.

3D Reconstruction: Earlier methods for 3D reconstruction were often based on image matching [13, 46]. More recent works directly fit level-set representations via differentiable rendering [37, 38, 52, 58]. To improve convergence, many approaches convert an SDF to a volumetric density on-the-fly, which is then used for standard volume rendering [28, 44, 54, 55, 59, 63]. Recent methods demonstrate converting 2D or 3D Gaussians into high-quality surface representations, either through volumetric fusion [9, 20, 64, 65], or using a Poisson reconstruction algorithm [17].

3. Preliminaries

In this section, we provide a brief introduction to volume- and surface-based representations and related notations.

Rendering Equation: A ray \mathbf{r} in 3D space is parameterized by its origin $\mathbf{o} \in \mathbb{R}^3$ and unit direction $\mathbf{v} \in \mathbb{S}^2$. A 3D point at distance t along ray \mathbf{r} is given by $\mathbf{r}(t) = \mathbf{o} + \mathbf{v}t$. Let σ denote a density field, which assigns a nonnegative density value $\sigma(\mathbf{x}) \in \mathbb{R}_+$ to each 3D point $\mathbf{x} \in \mathbb{R}^3$, and let ξ be a vector field providing an RGB color $\xi(\mathbf{x}, \mathbf{v})$ for each point $\mathbf{x} \in \mathbb{R}^3$ and direction $\mathbf{v} \in \mathbb{S}^2$. We can render ξ along ray \mathbf{r} for a given density field σ using the following equation [34]:

$$\mathcal{R}(\mathbf{r} \mid \sigma, \xi) = \int_0^\infty \sigma(\mathbf{r}(t)) \xi(\mathbf{r}(t), \mathbf{v}) w_r(t) dt, \quad (1)$$

where $w_r(t) = \exp\left[-\int_0^t \sigma(\mathbf{r}(s)) ds\right]$.

Volume-Based Representations (NeRF): NeRF models a scene as a volume – with absorption and emission but without scattering effects [48] – parametrized as (σ, ξ) . NeRF numerically approximates Equation (1) with quadrature [32], densely sampling rays to render each pixel (Figure 2a).

Surface-Based Representations (NeuS): Many surface-based representations have been proposed [33, 58, 59]; our work is built upon NeuS [54], as its densities weighting function – for which we refer to the original paper – is unbiased and occlusion-aware. In short, NeuS represents a surface implicitly by modeling it through a neural SDF, trained with differentiable volumetric rendering. A logistic distribution function ϕ_β maps distance values d to densities as follows:

$$\phi_\beta(d) = \beta e^{-\beta d} / (1 + e^{-\beta d})^2. \quad (2)$$

Here, the spread of densities around the surface (zero-level set) is controlled by the scalar β . A small β results in fuzzy

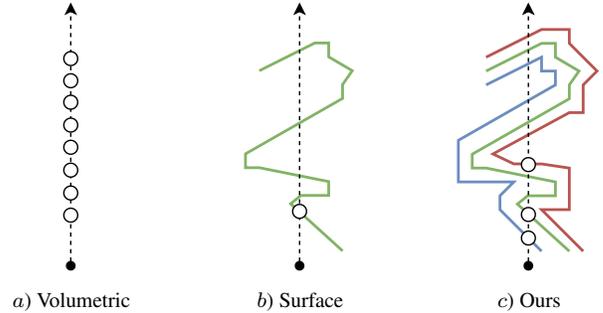


Figure 2. Sampling strategies: (a) volumetric rendering’s dense sampling; (b) single sampling point, as in surface rendering; (c) our method, only sampling the first intersection with each surface.

densities, while in the limit $\beta \rightarrow \infty$ densities are sharp impulse functions for points on the implicit surface. While NeuS regards β as a learnable parameter, Rosu and Behnke [44] showed how controlling it explicitly leads to better reconstructions. In our case, scheduling β ensures that densities shift from being widely spread to being peaked by the end of the training. When densities are peaked, the representation can be baked into a mesh suitable for efficient rendering. However, in this case, all appearance information is condensed on a single point (Figure 2b); therefore, surface-based methods are not able to model semi-transparent surfaces, strongly limiting their ability to handle mixed pixels often representing thin structures, which are notoriously missed [60].

4. Method

In this section, we describe our proposed representation, Volumetric Surfaces, in detail. We begin with its implicit geometry definition, then cover rendering, training, and baking. We also discuss and justify our design choices. Comprehensive architectural details can be found in the supplement.

4.1. k -SDF

Our new representation, called k -SDF, models k surfaces as distinct signed distance fields $\{d_1, \dots, d_k\}$. To composite the surfaces, k -SDF is endowed with an additional transparency field α , which assigns a view-dependent transparency value $\alpha(\mathbf{x}, \mathbf{v}) \in [0, 1]$ to each 3D point \mathbf{x} . The transparency field enables modeling semi-transparent surfaces. We generalize Equation (1) to render a k -SDF $(d_{1:k}, \alpha, \xi)$ as:

$$\mathcal{R}_\beta(\mathbf{r} \mid d_{1:k}, \alpha, \xi) = \sum_{i=1}^k \mathcal{R}_\beta(\mathbf{r} \mid d_i, \xi) \mathcal{R}_\beta(\mathbf{r} \mid d_i, \alpha) w_i, \quad (3)$$

where $w_i = \prod_{j=1}^i (1 - \mathcal{R}_\beta(\mathbf{r} \mid d_j, \alpha))$ and, for the sake of compactness, $\mathcal{R}_\beta(\mathbf{r} \mid d_i, \gamma)$ stands for $\mathcal{R}(\mathbf{r} \mid \phi_\beta \circ d_i, \gamma)$. Our

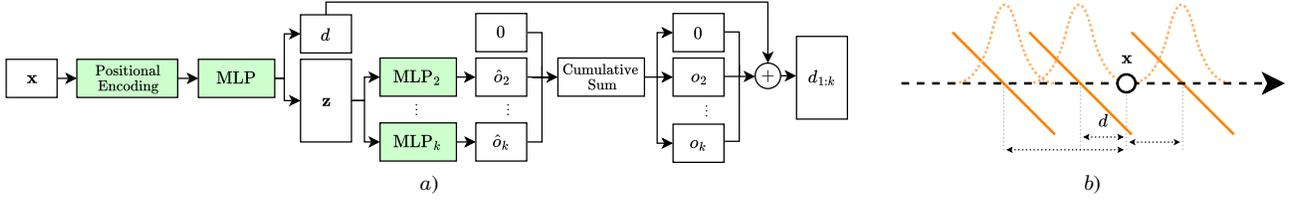


Figure 3. (a) High-level architecture of our k -SDF network, predicting k distance values as described in Section 4.1. For simplicity of visualization, all offsets are positive. We highlight trainable components. For additional architectural details, refer to the supplementary material. (b) 1D example visualization of the output $d_{1:k}$ when evaluating the network at a sample point \mathbf{x} along a ray. Signed distances are shown as solid lines, while β -controlled integration weights are represented as dotted lines.

per-surface density field is derived from the signed distance field d_i as in Equation (2).

Support Surfaces as Shells: Blending weights in Equation (3) are order-dependent, but lower-ranked densities are not necessarily positioned closer to the camera. To avoid sorting, we model the set of surfaces as shells, ensuring that layers are traversed in ray-intersection order. We model our k -SDF with a main surface represented as an SDF d , and a set of $k-1$ support surfaces represented as offset fields $\{o_2, \dots, o_k\}$ (Figure 4) from d 's zero-level set. The signed distance functions for each surface are thus given by: $d_{1:k} = (d, d + o_2, \dots, d + o_k)$. A surface defined by a positive offset is *contained* inside the main one, while a negative offset yields a surface *containing* the main one. To model multiple support surfaces while enforcing order, we perform a cumulative sum over predicted relative offsets $\{\hat{o}_2, \dots, \hat{o}_k\}$ (separately for negative and positive offsets) and use the resulting absolute displacements $\{o_2, \dots, o_k\}$ to parameterize the surfaces. Figure 3 illustrates our model.

Rendering Individual Surfaces: Equation (1) is approximated with quadrature [34] and evaluated at n discrete points $\{\mathbf{x}_1, \dots, \mathbf{x}_n\}$ sampled along a ray \mathbf{r} . For surface j , we render its color $\mathcal{C}_j(\mathbf{r})$ and transparency $\mathcal{A}_j(\mathbf{r})$ as:

$$\mathcal{C}_j(\mathbf{r}) = \sum_{i=1}^n w_{i,j} \xi(\mathbf{x}_i, \mathbf{v}, \mathbf{n}_{i,j}, \mathbf{z}_i), \quad (4)$$

$$\mathcal{A}_j(\mathbf{r}) = \sum_{i=1}^n w_{i,j} \alpha(\mathbf{x}_i, \mathbf{v}, \mathbf{n}_{i,j}, \mathbf{z}_i), \quad (5)$$

where RGB and transparency fields (ξ, α) are conditioned on sample position \mathbf{x}_i , view direction \mathbf{v} , SDF normal $\mathbf{n}_{i,j}$, and feature vector \mathbf{z}_i , an additional output of the k -SDF model. Sample weights $w_{i,j}$ are computed as in Wang et al. [54].

4.2. Surfaces Blending

We now rewrite Equation (3) as a fixed-order alpha blending of the rendered surface color and transparency:

$$\mathcal{R}(\mathbf{r}) = \sum_{i=1}^k \mathcal{C}_i(\mathbf{r}) \mathcal{A}_i(\mathbf{r}) w_i, \quad (6)$$

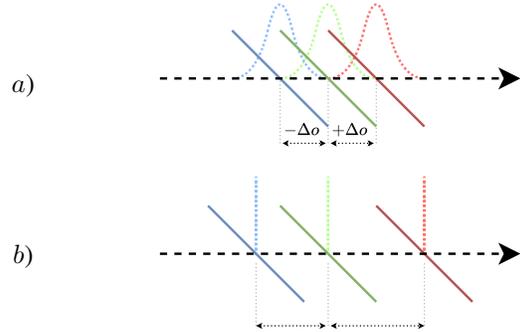


Figure 4. 1D example visualization of 3-SDF along a ray. Signed distances are shown as solid lines, while β -controlled integration weights are represented as dotted lines. (a) Initialization of the support surfaces as positive and negative constant displacements Δo from the main SDF. (b) Densities peaked at the end of training. The two support surfaces are displaced by trained offsets (o_1, o_2) , $d_1 = d - o_1$, $d_2 = d + o_2$.

where $w_i = \prod_{j=1}^i (1 - \mathcal{A}_j(\mathbf{r}))$.

Transparency Attenuation: Blending different hard surfaces might result in clear cut-offs at their borders, especially noticeable from test views. We prefer smoother transitions towards full transparency. We achieve this by multiplying predicted transparencies with a weight α_w that depends on the angle between the view direction and the surface normal, downweighting the contribution from grazing angles. Specifically, we use: $\alpha_w = 2 \cdot \text{Sigmoid}(10 \cdot |\mathbf{v} \cdot \mathbf{n}|) - 1$.

4.3. Training and Baking

Our method is composed of two main stages. First, we optimize an implicit representation of k surfaces and their appearances. Then, we fine-tune resolution-bounded neural textures over their explicit (meshes) representation. Our hyperparameters are robust, ensuring consistent performance across all scenes and datasets. In the following, we describe both phases in detail.

Implicit Surfaces: We begin by training a standard NeuS-like [54] model for 100k iterations, during which β is ex-

ponentially scheduled from large densities ϕ_{β_1} to thinner densities ϕ_{β_2} . In practice, training the k -SDF model from scratch with predicted transparencies often introduces fully transparent additional geometry in the reconstruction. This opaque pre-training step helps prevent such artifacts. The reconstructed surface serves as an anchor for initializing the remaining $k-1$ support surfaces, which are represented as shells uniformly spaced from the main surface by Δo . The mathematical formulation of k -SDF (Section 4.1) allows for initializing support surfaces both inside and outside the main SDF. However, we find that initializing all support surfaces on the inside increases model capacity, leading to better results (Section 5.1).

While we have observed the optimization to be robust under different values of Δo , we found a good balance by setting it by the logistic distribution function ϕ_{β_2} standard deviation: $\Delta o = (1/\beta_2)\pi/\sqrt{3}$. This ensures that surface densities only partially overlap (e.g., Figure 4). The k -SDF model is trained for 50k iterations starting from ϕ_{β_2} , until a ϕ_{β_3} value for which all surfaces are modeled as peaked densities. When this happens, our rendering process collapses to a simple blending of hard surfaces (Figure 2c), making the reconstructed set of implicit surfaces optimal for the later steps. During both training phases, we apply two additional losses to all surfaces. The Eikonal loss \mathcal{L}_e [16], calculated on points in the vicinity of the zero-level sets and on randomly sampled points, and a curvature loss \mathcal{L}_s [44] on near-surfaces points, to push the optimization toward smooth solutions. We minimize $\mathcal{L} = \mathcal{L}_c + \lambda_e \mathcal{L}_e + \lambda_s \mathcal{L}_s$, where \mathcal{L}_c is the standard L_1 pixel-wise color loss, $\lambda_e = 0.04$ and $\lambda_s = 0.65$.

Occupancy Grid: As implicit surfaces training progresses and densities peak, our volumetric rendering samples are gradually positioned closer to the zero-level sets of the signed distance functions, since points farther away would be in empty space. To do so, we compute per-surface binary occupancy values by describing as occupied voxels whose center point’s $|d|$ value, together with the current β and the voxel’s space diagonal, would allow any point in the voxel volume to have density above a 10^{-4} threshold. Per-surface occupancy values are then aggregated with an OR operation, resulting in a single binary occupancy grid. Rays traverse the grid to sample n points uniformly in occupied space. In our experiments, we used a grid of resolution 256^3 .

Importance Sampling: During implicit surfaces training, we adopt hierarchical sampling from NeuS [54], extending it to the k -surfaces case. Starting from the n uniform samples, each SDF is evaluated to compute k CDFs; these are summed together and normalized. The resulting probability distribution is used to sample $m/2$ additional points that are added to the previous n . The whole operation is then repeated on the expanded set of points to add $m/2$ more samples. The resulting number of samples per ray is then

$n + m$. In the first iteration, the kernel size β is half of that used during rendering, while in the second, it matches it.

Baking Meshes: After the implicit surfaces optimization, we extract k -SDF zero-level sets as high-resolution meshes using marching cubes [30] and simplify them significantly [15, 31] to 0.02% of the original triangle count (approximately 2 MB per mesh for synthetic scenes), meeting the strict computing and memory constraints of mobile hardware. Finally, we generate UV atlases for these lightweight meshes using `xatlas` [61]. This step is essential for training per-mesh neural SH textures, which are later baked into explicit textures.

Texturing Meshes: Finally, we train a per-surface view-dependent appearance model for color and transparency using neural textures. A neural texture is implemented as a hash-grid with input dimension 2 and a small decoder MLP with output dimension dependent on the SH degree it models. The i -th neural texture predicts $\overline{\text{sh}}_i \in \mathbb{R}^{4 \times c_{\text{deg}}}$, with $c = \{1, 3, 5, 7\}$. The outputs of all neural textures are concatenated over the coefficient dimension ($\text{sh} \in \mathbb{R}^{4 \times (\text{deg}+1)^2}$) and decoded with view direction \mathbf{v} to predict RGBA. Specifically, we discretize our neural textures by grounding them to a fixed target baking resolution. In this setting, each point $\mathbf{x} \in \mathbb{R}^3$ on the surface of a mesh is mapped to a point $\overline{\mathbf{x}} \in [0, 1]^2$ in UV space. SH coefficients at $\overline{\mathbf{x}}$ are predicted as the result of a bilinear interpolation of predictions at neighboring texel centers $\{\overline{\mathbf{x}}_i\}_{1,\dots,4}$, as visualized in Figure 5. This effectively mimics how OpenGL fragment shaders access and interpolate texture values; by doing so, we optimize our texture values such that when displayed in our real-time renderer, images match – up to numerical precision – those synthesized by our differentiable renderer. This final phase is trained for 15k iterations using the L_1 pixel-wise color loss.

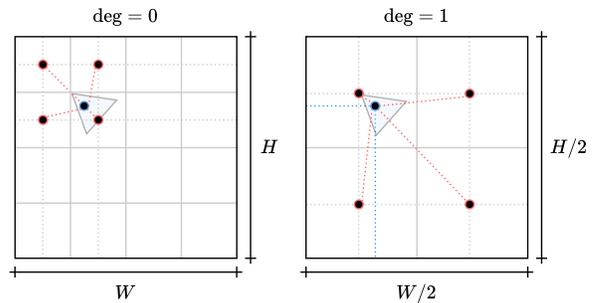


Figure 5. Bilinear interpolation in our mixed-resolution textures. Instead of querying the blue 2D point directly, we predict the values at its surrounding texel centers (red points) and bilinearly interpolate them. This anchors the neural texture to a predefined target resolution (W, H) .

Table 1. Sweep over the number of meshes and target neural textures resolution. Mixed resolutions are described in Section 4.3. Increasing neural texture resolution improves reconstruction quality up to 1024^2 , but performance declines beyond that. Modeling all textures at the maximum resolution (2048^2) is counterproductive, as results indicate that a mixed-resolution approach yields better image quality while reducing memory usage. Results on the *khady* scene from Shelly [56]; we highlight the **best**, **second best** and **third best**.

Method	PSNR \uparrow				
	mixed	2048^2	1024^2	512^2	256^2
5-Mesh	29.88	29.88	29.91	29.82	29.54
7-Mesh	29.97	29.97	30.02	29.94	29.65
9-Mesh	29.96	29.93	29.97	28.91	29.62

Mixed Resolutions: Storing all textures at full resolution (2048^2) is impractical, as it would require approximately 0.5 GB per mesh. We propose scaling texture resolution proportionally to the spherical harmonics (SH) coefficient degree: the base color is modeled at the highest resolution (2048^2), while the highest SH degree coefficients are modeled at the lowest resolution (256^2). This approach significantly reduces the memory footprint to approximately 14 MB per mesh without compromising image quality (Table 1).

Squeezing and Quantization: Neural textures predicted values are continuous and unbounded. Before storing them, they must be squeezed to the unit range via $\hat{\mathbf{sh}} = \text{Sigmoid}(\mathbf{sh})$. Training must account for quantization to the discrete $[0, 255]$ range when storing textures as images. Following MERF [42], we apply the function $q(x) = \text{round}(255x)/255$ to the predicted and squeezed coefficients. Before rendering, we re-scale value to a hyper-parameters controlled range $[v_{\min}, v_{\max}]$: $\mathbf{sh} = v_{\min} + (v_{\max} - v_{\min})q(\hat{\mathbf{sh}})$. We observed a range of $[-15, 15]$ to work well on all scenes.

Baking Textures: Lastly, we bake our resolution-grounded neural textures, which is straightforward as it only requires predicting values at texel centers and storing them. Baking results in $(\text{deg} + 1)^2$ PNG images, where the i -th image stores the i -th coefficient of RGBA channels. The fully baked representation can finally be visualized in our WebGL renderer, which rasterizes all meshes in a fixed order and blends them in the final frame buffer before displaying it on screen.

5. Evaluation

Our key baselines focus on real-time rendering, with 3DGS and MobileNeRF as primary competitors. 3DGS represents the fastest volumetric approach, while MobileNeRF pioneers surface-based neural graphics for mobile devices. We compare to these methods in terms of quality, speed, and memory footprint on a low-power laptop (Dell XPS 13 i5)

Table 2. Framerate is measured on *close-up* views at HD (720p) resolution on low-power smartphone (Samsung A52s) (marked with \diamond) and laptop (Dell XPS 13 i5) (marked with \star), on respective WebGL renderers; memory footprint as stored on disk. 3DGS [24] with spherical harmonics of degree 2, ours of degree 3. Metrics averaged over scenes of the Shelly [56] dataset. We present further qualitative comparisons in the supplementary material.

Method	FPS $\diamond \uparrow$	FPS $\star \uparrow$	PSNR \uparrow	MB \downarrow
MobileNeRF [7]	24	35	29.30	194
3DGS-50K	20	160	32.73	12
3DGS-75K	13	115	33.05	18
3DGS [24]	8	18	35.44	57
3-Mesh	65	145	33.39	46
5-Mesh	55	90	34.25	77
7-Mesh	42	70	34.50	110
9-Mesh	35	55	34.38	140

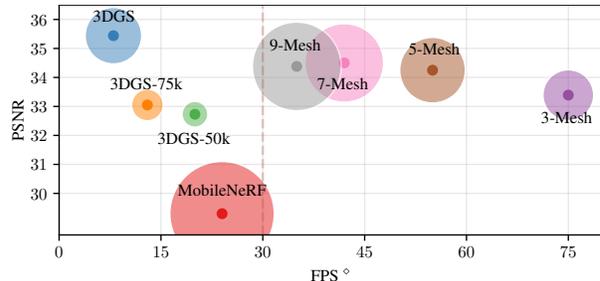


Figure 6. Frame rate vs. image quality comparison (smartphone results \diamond from Table 2). The radius of each circle represents the memory footprint as stored on disk. The vertical dashed line marks the required frame rate for real-time rendering (30 FPS).

and a smartphone (Samsung A52s) (Table 2). Additionally, we compare with other baselines not designed for general-purpose hardware rendering to provide a broader overview of current methods (Table 3). Aiming at high-quality representation of fuzzy geometries, we focus our evaluation on object-centric datasets with prominent fuzzy structures. We present results from synthetic benchmark datasets like Shelly [56], plush objects from real-world tabletop scenes (DTU [21]), and additional synthetic custom scenes (our *plushy* and *hairy monkey* from Sharma et al. [47]).

Our method, tested on renderings from our real-time WebGL renderer, consistently delivers higher image quality than surface-based competitors and renders faster than 3DGS. We observe that our adaptive shell spacing clusters surfaces around solid structures while maintaining greater separation in volumetric regions (Figure 7). Using seven layers offers a good balance between image quality, model size, and rendering speed, as quality tends to degrade with nine meshes under the same number of training iterations. This happens as deeper surfaces contribute less to pixel color, reducing gradient magnitudes and slowing optimization. Figure 6 il-

Table 3. Results are averaged across all testing scenes; we highlight the **best**, **second best** and **third best** results. Highlighted **baselines** meet the compute and/or memory requirements to run on general-purpose hardware (via WebGL) without modifications, as discussed in Section 2. QuadFields [47] fails to meet real-time requirements due to its reliance on specialized ray-tracing acceleration hardware and its high memory consumption (e.g., Shelly [56] scenes require an average of 1213 MB). **Our results are directly computed from our WebGL real-time render on our final baked representation.** Methods marked with a “*” show results from original papers, as code is unavailable at the time of writing. Instant-NGP [35] results on Shelly from Wang et al. [56]. DTU [21] is evaluated on masked foreground. PermutoSDF [44] trained until densities are fully peaked (ϕ_{β_3}). Metrics not provided by a baseline are denoted with “—”. Please note that the Shelly dataset, as released by the authors, has a large overlap of views between test and training sets. All our experiments were conducted on a cleaned version of the dataset, free of this problem. However, a fair comparison with baselines whose code is not public remains difficult. Our per-scene metrics are reported in the supplementary material.

Method	Shelly [56]			Custom (<i>plushy</i> + [47])			DTU [21] (<i>scans 83, 105</i>)		
	PSNR \uparrow	SSIM \uparrow	LPIPS \downarrow	PSNR \uparrow	SSIM \uparrow	LPIPS \downarrow	PSNR \uparrow	SSIM \uparrow	LPIPS \downarrow
3DGS [24]	35.44	0.975	0.089	37.34	0.982	0.147	38.06	0.989	0.086
Instant-NGP [35]	33.22	0.922	0.125	31.13	0.935	0.132	38.24	—	—
PermutoSDF [44]	29.85	0.950	0.129	33.31	0.961	0.193	36.31	0.988	0.098
AdaptiveShells* [56]	36.02	0.954	0.079	—	—	—	—	—	—
QuadFields* [47]	35.13	0.954	0.073	—	—	—	—	—	—
MobileNeRF [7]	29.30	0.939	0.150	30.89	0.942	0.195	—	—	—
3-Mesh	33.39	0.978	0.115	35.00	0.970	0.171	36.41	0.985	0.092
5-Mesh	34.25	0.980	0.110	35.45	0.975	0.172	36.87	0.986	0.085
7-Mesh	34.50	0.981	0.109	35.63	0.977	0.169	36.77	0.987	0.084
9-Mesh	34.38	0.981	0.110	35.74	0.978	0.167	37.17	0.987	0.083

illustrates the trade-offs between frame rate on a low-cost smartphone and model size. Notably, 3DGS fails to meet real-time requirements even when the number of Gaussians is capped during optimization, which leads to a substantial loss in quality.

For comparability, the frame rate of 3DGS is measured using a widely adopted web viewer [22]. This implementation skips per-frame sorting to prevent slowdowns, using occasional CPU sorting instead. This results in noticeable popping artifacts during rapid camera rotation and suboptimal performance on mobile devices. Our sorting-free method avoids these issues. While our method falls short of the latest volume-based baselines in image quality, our baked representation provides a favorable balance between quality and speed, rendering much faster on non-specialized hardware. We refer to our supplementary material for additional results visualizations.

5.1. Ablations

We ablate all crucial aspects of our method and show results on both implicit geometry and fully baked phases in Table 4. We run our full model: 1) Without view-dependent surface transparency, we observe increased model expressivity as shown by the improved image quality metrics. 2) Without curvature loss during k -SDF training ($\lambda_s = 0$), surfaces can reconstruct high-frequency details, but the image quality of the baked representation worsens. This happens because the post-baking mesh poorly aligns with the implicit one. En-

Table 4. Ablation studies over intermediate results (5-SDF, implicit representation, before meshes baking) and on final results (5-Mesh, baked, real-time rendering assets). See Section 5.1 for explanations. 5-Mesh achieves higher quality than the 5-SDF due to its fixed geometry and surface-constrained appearance model; the SDF representation suffers from the stochastic nature of ray sampling, forcing the appearance model to allocate capacity to off-surface elements. Results averaged over Shelly [56].

Ablation	5-SDF			5-Mesh		
	PSNR \uparrow	SSIM \uparrow	LPIPS \downarrow	PSNR \uparrow	SSIM \uparrow	LPIPS \downarrow
Full	32.05	0.964	0.130	34.25	0.980	0.110
1) w/o view-dep. α	31.75	0.962	0.131	32.71	0.975	0.120
2) w/o curvature \mathcal{L}_s	33.02	0.971	0.117	33.41	0.980	0.110
3) w/o α_w	32.11	0.966	0.130	33.96	0.980	0.113
4) w. const. Δo	—	—	—	30.09	0.950	0.129
5) w. outer init.	—	—	—	30.85	0.955	0.121

abling curvature loss pushes k -SDF to reconstruct smoother surfaces with meshes that better match their implicit counterparts while keeping the triangle cost low. Moreover, this highlights how our high-capacity appearance model compensates for missing geometric details. 3) Without transparency attenuation (Section 4.2) at grazing angles (no α_w), rendering errors increase, especially at object boundaries. 4) With fixed (not trained) support surfaces offsets (Δo). We observe a significant decline in results, as training offsets is essential for adapting spacing optimally to each scene. 5) Initializing support surfaces outside the main SDF. We observe how this leads geometry to extend beyond the object silhouette.

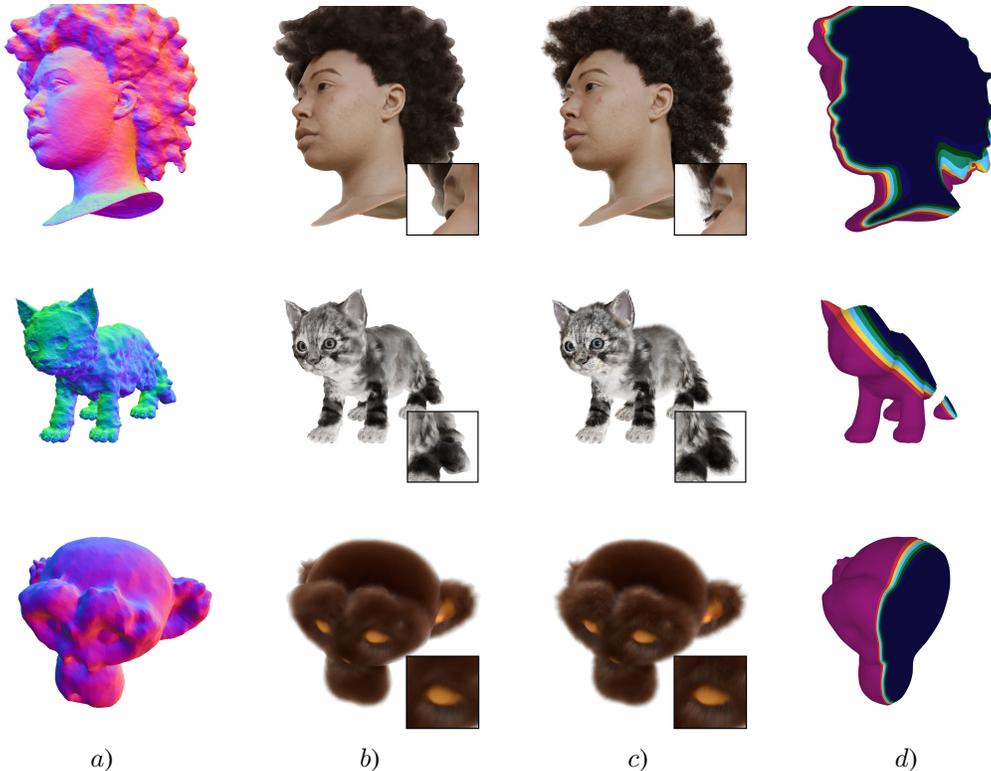


Figure 7. Qualitative comparisons between (b) PermutoSDF [44], a state-of-the-art implicit surface-based method, and (c) our Volumetric Surfaces demonstrate that our approach convincingly represents fuzzy objects. This is achieved by trading (a) high-frequency geometry for the number of integration points, which are found by rasterizing smooth, lightweight meshes defined as (d) shells around the object and traversed in a fixed order. Volumetric Surfaces enable fast rendering of fuzzy geometries on general-purpose hardware with image quality approaching the latest volumetric representations (Table 3). Scenes from Shelly [56] and QuadFields [47].

While training views can compensate through learned view-dependent transparency, test views suffer from degraded generalization. In contrast, initializing surfaces inside biases them to be tighter, preventing unwanted expansion. This happens because the reconstructed main surface is typically conservative, forming an outer shell that encloses the scene content, including fuzzy geometry (Figure 1).

5.2. Limitations

Textured shells [23, 26, 27] exhibit artifacts at grazing angles, especially when test views fall outside training coverage or model capacity is limited. Increasing shell count mitigates this but raises memory and computation costs, particularly during reconstruction. Artist-designed extruded textured fins [26] can address these artifacts, though learning this component remains challenging and is left for future work. Our model performs well in densely observed scenes but struggles in sparsely sampled ones. When under-constrained, it tends to explain observations through view-dependency rather than multi-view consistent geometry, leading to poorer generalization in test views. A solution explored by Wan et al. [53] trains a robust model (e.g., NeRF-like) and distills its

reconstruction using renderings from randomly generated cameras as the training set. Handling thin structures remains challenging due to the limitations of the underlying SDF geometry representation. Advantages on fully solid surfaces are also marginal; see the supplement for further details.

6. Conclusion

We presented Volumetric Surfaces, a multi-layer mesh representation for real-time view synthesis of fuzzy objects on a low-power laptops and smartphones. Our method renders faster than state-of-the-art volume-based approaches, while being significantly more capable at reproducing fuzzy objects than single-surface methods. For future work, we aim to develop a single-stage, end-to-end training procedure that directly generates real-time renderable assets.

Acknowledgments

Plushy Blender object by *kaizentutorials*. Stefano Esposito acknowledges travel support from the European Union’s Horizon 2020 research and innovation program under ELISE Grant Agreement No. 951847.

References

- [1] Rameen Abdal, Wang Yifan, Zifan Shi, Yinghao Xu, Ryan Po, Zhengfei Kuang, Qifeng Chen, Dit-Yan Yeung, and Gordon Wetzstein. Gaussian Shell Maps for Efficient 3D Human Generation. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 9441–9451, 2024. 2
- [2] Kara-Ali Aliev, Artem Sevastopolsky, Maria Kolos, Dmitry Ulyanov, and Victor S. Lempitsky. Neural point-based graphics. In *Proc. of the European Conf. on Computer Vision (ECCV)*, 2020. 2
- [3] Jonathan T. Barron, Ben Mildenhall, Dor Verbin, Pratul P. Srinivasan, and Peter Hedman. Mip-NeRF 360: Unbounded Anti-Aliased Neural Radiance Fields. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2022. 1
- [4] Gaurav Bhokare, Eisen Montalvo, Elie Diaz, and Cem Yuksel. Real-Time Hair Rendering with Hair Meshes. In *ACM Trans. on Graphics*, 2024. 2
- [5] Anpei Chen, Zexiang Xu, Andreas Geiger, Jingyi Yu, and Hao Su. TensorRF: Tensorial radiance fields. In *Proc. of the European Conf. on Computer Vision (ECCV)*, 2022. 2
- [6] Zhiqin Chen and Hao Zhang. Learning implicit fields for generative shape modeling. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2019. 2
- [7] Zhiqin Chen, Thomas Funkhouser, Peter Hedman, and Andrea Tagliasacchi. MobileNeRF: Exploiting the polygon rasterization pipeline for efficient neural field rendering on mobile architectures. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2023. 2, 6, 7, 1, 4
- [8] Zhang Chen, Zhong Li, Liangchen Song, Lele Chen, Jingyi Yu, Junsong Yuan, and Yi Xu. NeuRBF: A Neural Fields Representation with Adaptive Radial Basis Functions. In *Proc. of the IEEE International Conf. on Computer Vision (ICCV)*, 2023. 2
- [9] Pinxuan Dai, Jiamin Xu, Wenxiang Xie, Xinguo Liu, Huamin Wang, and Weiwei Xu. High-quality Surface Reconstruction using Gaussian Surfels. In *ACM Trans. on Graphics*. Association for Computing Machinery, 2024. 3
- [10] Daniel Duckworth, Peter Hedman, Christian Reiser, Peter Zhizhin, Jean-François Thibert, Mario Lučić, Richard Szeliski, and Jonathan T Barron. SMERF: Streamable Memory Efficient Radiance Fields for Real-Time Large-Scene Exploration. *ACM Trans. on Graphics*, 43(4):1–13, 2024. 1, 2
- [11] Stefano Esposito and Andreas Geiger. MVDatasets: Standardized dataloaders for 3D computer vision, 2025. <https://github.com/autonomousvision/mvdatasets>. 1
- [12] Stefano Esposito, Daniele Baieri, Stefan Zellmann, André Hinkenjann, and Emanuele Rodolà. KiloNeuS: A Versatile Neural Implicit Surface Representation for Real-Time Rendering. *arXiv.org*, 2206.10885, 2022. 2
- [13] Yasutaka Furukawa, Brian Curless, Steven M. Seitz, Richard Szeliski, and Google Inc. Towards internet-scale multiview stereo. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2010. 3
- [14] Stephan J. Garbin, Marek Kowalski, Matthew Johnson, Jamie Shotton, and Julien Valentin. FastNeRF: High-fidelity neural rendering at 200fps. In *Proc. of the IEEE International Conf. on Computer Vision (ICCV)*, pages 14346–14355, 2021. 2
- [15] Michael Garland and Paul S. Heckbert. Surface simplification using quadric error metrics. In *ACM Trans. on Graphics*, pages 209–216, 1997. 5
- [16] Amos Gropp, Lior Yariv, Niv Haim, Matan Atzmon, and Yaron Lipman. Implicit geometric regularization for learning shapes. In *Proc. of the International Conf. on Machine Learning (ICML)*, 2020. 5
- [17] Antoine Guédon and Vincent Lepetit. SuGaR: Surface-Aligned Gaussian Splatting for Efficient 3D Mesh Reconstruction and High-Quality Mesh Rendering. *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2024. 3
- [18] Antoine Guédon and Vincent Lepetit. Gaussian Frosting: Editable Complex Radiance Fields with Real-Time Rendering. *Proc. of the European Conf. on Computer Vision (ECCV)*, 2024. 2
- [19] Peter Hedman, Pratul P. Srinivasan, Ben Mildenhall, Jonathan T. Barron, and Paul Debevec. Baking neural radiance fields for real-time view synthesis. *Proc. of the IEEE International Conf. on Computer Vision (ICCV)*, 2021. 2
- [20] Binbin Huang, Zehao Yu, Anpei Chen, Andreas Geiger, and Shenghua Gao. 2D Gaussian splatting for geometrically accurate radiance fields. In *ACM Trans. on Graphics*, 2024. 3
- [21] Rasmus Jensen, Anders Dahl, George Vogiatzis, Engin Tola, and Henrik Aanæs. Large scale multi-view stereopsis evaluation. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2014. 6, 7, 1, 4, 5
- [22] Mark Kellogg. 3D Gaussian Splatting for Three.js, 2023. <https://github.com/mkkellogg/GaussianSplats3D>. 7
- [23] Benjamin Kenwright. A Practical Guide to Generating Real-Time Dynamic Fur and Hair using Shells, 2014. Technical course notes. 2, 8
- [24] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3D Gaussian Splatting for Real-Time Radiance Field Rendering. *ACM Trans. on Graphics*, 2023. 1, 2, 6, 7, 4, 5
- [25] Andreas Kurz, Thomas Neff, Zhaoyang Lv, Michael Zollhöfer, and Markus Steinberger. AdaNeRF: Adaptive Sampling for Real-time Rendering of Neural Radiance Fields. In *Proc. of the European Conf. on Computer Vision (ECCV)*, 2022. 2
- [26] Jerome Lengyel, Emil Praun, Adam Finkelstein, and Hugues Hoppe. Real-time fur over arbitrary surfaces. In *Proc. of the Symposium on Interactive 3D Graphics (SI3D)*, pages 227–232, 2001. 2, 8
- [27] Jerome Edward Lengyel. Real-Time Fur. In *Rendering Techniques 2000: Proceedings of the Eurographics Workshop in Brno, Czech Republic, June 26–28, 2000 11*, pages 243–256. Springer, 2000. 2, 8
- [28] Zhaoshuo Li, Thomas Müller, Alex Evans, Russell H Taylor, Mathias Unberath, Ming-Yu Liu, and Chen-Hsuan Lin. Neuralangelo: High-Fidelity Neural Surface Reconstruction. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2023. 3

- [29] Lingjie Liu, Jiatao Gu, Kyaw Zaw Lin, Tat-Seng Chua, and Christian Theobalt. Neural sparse voxel fields. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020. 2
- [30] William E. Lorensen and Harvey E. Cline. Marching Cubes: A high resolution 3D surface construction algorithm. In *ACM Trans. on Graphics*, 1987. 5
- [31] Filippo Maggioli, Daniele Baieri, Emanuele Rodolà, and Simone Melzi. ReMatching: Low-Resolution Representations for Scalable Shape Correspondence. *Proc. of the European Conf. on Computer Vision (ECCV)*, 2024. 5
- [32] Nelson Max. Optical models for direct volume rendering. *IEEE Transactions on Visualization and Computer Graphic (TVCG)*, 1995. 3
- [33] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy Networks: Learning 3D reconstruction in function space. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2019. 2, 3
- [34] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. NeRF: Representing scenes as neural radiance fields for view synthesis. In *Proc. of the European Conf. on Computer Vision (ECCV)*, 2020. 2, 3, 4, 1, 6, 7
- [35] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Trans. on Graphics*, 2022. 2, 7, 1
- [36] Thomas Neff, Pascal Stadlbauer, Mathias Parger, Andreas Kurz, Joerg H Mueller, Chakravarty R Alla Chaitanya, Anton Kaplanyan, and Markus Steinberger. DONeRF: Towards real-time rendering of neural radiance fields using depth oracle networks. In *Computer Graphics Forum*, pages 45–59. Wiley Online Library, 2021. 2
- [37] Michael Niemeyer, Lars Mescheder, Michael Oechsle, and Andreas Geiger. Differentiable Volumetric Rendering: Learning implicit 3D representations without 3D supervision. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2020. 3
- [38] Michael Oechsle, Songyou Peng, and Andreas Geiger. UNISURF: Unifying neural implicit surfaces and radiance fields for multi-view reconstruction. In *Proc. of the IEEE International Conf. on Computer Vision (ICCV)*, 2021. 3
- [39] Jeong Joon Park, Peter Florence, Julian Straub, Richard A. Newcombe, and Steven Lovegrove. DeepSDF: Learning continuous signed distance functions for shape representation. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2019. 2
- [40] Marie-Julie Rakotosaona, Fabian Manhardt, Diego Martin Arroyo, Michael Niemeyer, Abhijit Kundu, and Federico Tombari. NeRFMeshing: Distilling Neural Radiance Fields into Geometrically-Accurate 3D Meshes. In *Proc. of the International Conf. on 3D Vision (3DV)*, 2023. 2
- [41] Christian Reiser, Songyou Peng, Yiyi Liao, and Andreas Geiger. KiloNeRF: Speeding up neural radiance fields with thousands of tiny mlps. In *Proc. of the IEEE International Conf. on Computer Vision (ICCV)*, 2021. 2
- [42] Christian Reiser, Richard Szeliski, Dor Verbin, Pratul P. Srinivasan, Ben Mildenhall, Andreas Geiger, Jonathan T. Barron, and Peter Hedman. MERF: Memory-Efficient Radiance Fields for Real-time View Synthesis in Unbounded Scenes. *ACM Trans. on Graphics*, 2023. 2, 6
- [43] Christian Reiser, Stephan Garbin, Pratul P. Srinivasan, Dor Verbin, Richard Szeliski, Ben Mildenhall, Jonathan T. Barron, Peter Hedman, and Andreas Geiger. Binary Opacity Grids: Capturing Fine Geometric Detail for Mesh-Based View Synthesis. *ACM Trans. on Graphics*, 2024. 1, 2
- [44] Radu Alexandru Rosu and Sven Behnke. PermutoSDF: Fast Multi-View Reconstruction with Implicit Surfaces using Permutohedral Lattices. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2023. 3, 5, 7, 8, 1, 4
- [45] Darius Rückert, Linus Franke, and Marc Stamminger. ADOP: Approximate differentiable one-pixel point rendering. *ACM Trans. on Graphics*, 2022. 2
- [46] Johannes L. Schönberger and Jan-Michael Frahm. Structure-from-motion revisited. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2016. 3
- [47] Gopal Sharma, Daniel Rebain, Andrea Tagliasacchi, and Kwang Moo Yi. Volumetric Rendering with Baked Quadrature Fields. In *Proc. of the European Conf. on Computer Vision (ECCV)*, 2024. 2, 6, 7, 8, 4, 5
- [48] Andrea Tagliasacchi and Ben Mildenhall. Volume rendering digest (for NeRF). *arXiv.org*, 2209.02417, 2022. 3
- [49] Towaki Takikawa, Thomas Müller, Merlin Nimier-David, Alex Evans, Sanja Fidler, Alec Jacobson, and Alexander Keller. Compact Neural Graphics Primitives with Learned Hash Probing. In *ACM Trans. on Graphics*, 2023. 2
- [50] Jiaxiang Tang, Hang Zhou, Xiaokang Chen, Tianshu Hu, Errui Ding, Jingdong Wang, and Gang Zeng. Delicate Textured Mesh Recovery from NeRF via Adaptive Surface Refinement. *Proc. of the IEEE International Conf. on Computer Vision (ICCV)*, 2023. 2
- [51] Haithem Turki, Vasu Agrawal, Samuel Rota Bulò, Lorenzo Porzi, Peter Kotschieder, Deva Ramanan, Michael Zollhöfer, and Christian Richardt. HybridNeRF: Efficient Neural Rendering via Adaptive Volumetric Surfaces. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2024. 2
- [52] Delio Vicini, Sébastien Speierer, and Wenzel Jakob. Differentiable signed distance function rendering. *ACM Trans. on Graphics*, 2022. 3
- [53] Ziyu Wan, Christian Richardt, Aljaž Božič, Chao Li, Vijay Rengarajan, Seonghyeon Nam, Xiaoyu Xiang, Tuotuo Li, Bo Zhu, Rakesh Ranjan, and Jing Liao. Learning neural duplex radiance fields for real-time view synthesis. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2023. 2, 8
- [54] Peng Wang, Lingjie Liu, Yuan Liu, Christian Theobalt, Taku Komura, and Wenping Wang. NeuS: Learning Neural Implicit Surfaces by Volume Rendering for Multi-view Reconstruction. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021. 3, 4, 5
- [55] Yiming Wang, Qin Han, Marc Habermann, Kostas Daniilidis, Christian Theobalt, and Lingjie Liu. NeuS2: Fast Learning

- of Neural Implicit Surfaces for Multi-view Reconstruction. In *Proc. of the IEEE International Conf. on Computer Vision (ICCV)*, 2023. 3
- [56] Zian Wang, Tianchang Shen, Merlin Nimier-David, Nicholas Sharp, Jun Gao, Alexander Keller, Sanja Fidler, Thomas Müller, and Zan Gojcic. Adaptive Shells for Efficient Neural Radiance Field Rendering. In *ACM Trans. on Graphics*, 2023. 2, 6, 7, 8, 4, 5
- [57] Qiangeng Xu, Zexiang Xu, Julien Philip, Sai Bi, Zhixin Shu, Kalyan Sunkavalli, and Ulrich Neumann. Point-NeRF: Point-based neural radiance fields. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2022. 2
- [58] Lior Yariv, Yoni Kasten, Dror Moran, Meirav Galun, Matan Atzmon, Basri Ronen, and Yaron Lipman. Multiview Neural Surface Reconstruction by Disentangling Geometry and Appearance. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020. 3
- [59] Lior Yariv, Jiatao Gu, Yoni Kasten, and Yaron Lipman. Volume rendering of neural implicit surfaces. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021. 3
- [60] Lior Yariv, Peter Hedman, Christian Reiser, Dor Verbin, Pratul P. Srinivasan, Richard Szeliski, Jonathan T. Barron, and Ben Mildenhall. BakedSDF: Meshing Neural SDFs for Real-Time View Synthesis. In *ACM Trans. on Graphics*, 2023. 1, 2, 3
- [61] Jonathan Young. xatlas, 2022. <https://github.com/jpcy/xatlas>. 5
- [62] Alex Yu, Ruilong Li, Matthew Tancik, Hao Li, Ren Ng, and Angjoo Kanazawa. PlenOctrees for real-time rendering of neural radiance fields. In *Proc. of the IEEE International Conf. on Computer Vision (ICCV)*, 2021. 2
- [63] Zehao Yu, Anpei Chen, Bozidar Antic, Songyou Peng, Apratim Bhattacharyya, Michael Niemeyer, Siyu Tang, Torsten Sattler, and Andreas Geiger. SDFStudio: A Unified Framework for Surface Reconstruction, 2022. GitHub repository. 3
- [64] Zehao Yu, Torsten Sattler, and Andreas Geiger. Gaussian Opacity Fields: Efficient Adaptive Surface Reconstruction in Unbounded Scenes. *ACM Trans. on Graphics*, 2024. 1, 3
- [65] Baowen Zhang, Chuan Fang, Rakesh Shrestha, Yixun Liang, Xiaoxiao Long, and Ping Tan. RaDe-GS: Rasterizing Depth in Gaussian Splatting. *arXiv.org*, 2406.01467, 2024. 3

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

β scheduling details: During training, the β 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 β_2 , the logistic distribution standard deviation is approximately 0.001. We use this value to initialize offsets as constants (Δo). By the end of implicit surface training (β_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.

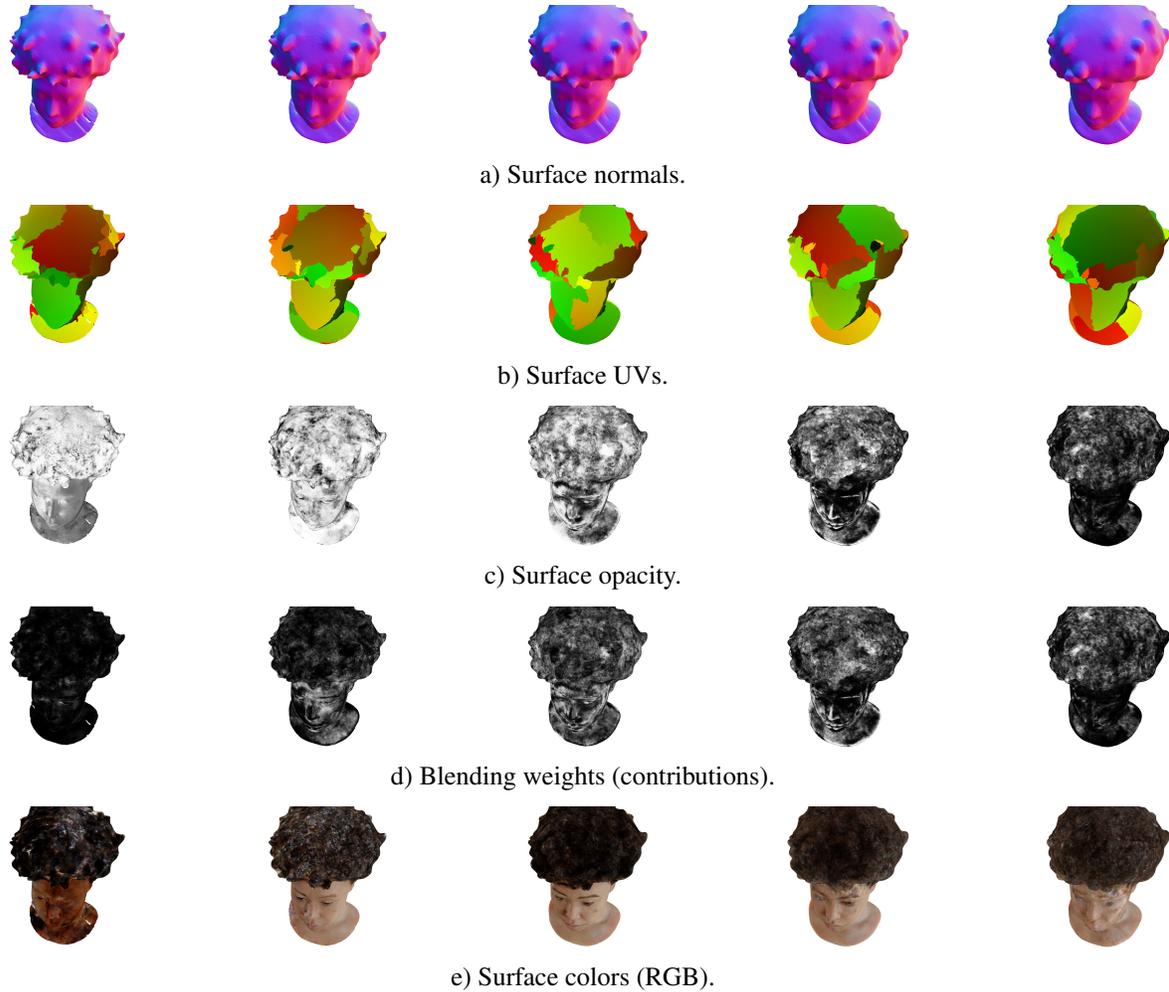


Figure S1. Visualization of render buffers from our 5-Mesh model. Layers order: left to right is inner to outer. Individual layer color and alpha buffers are blended as described in Section 4.2. Results on the *khady* scene from Shelly [56].

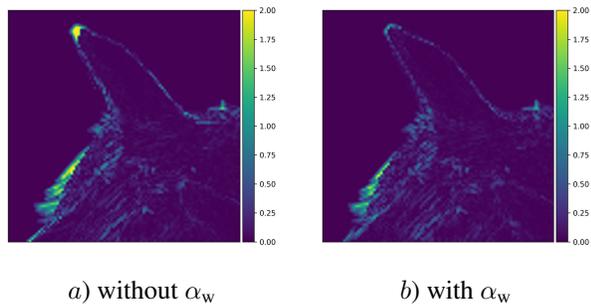


Figure S2. (a) Rendering error crop (averaged over color channels) without and (b) with transparency-decay, resulting in a 2.13 dB PSNR gain. Scene from the Shelly [56].

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.

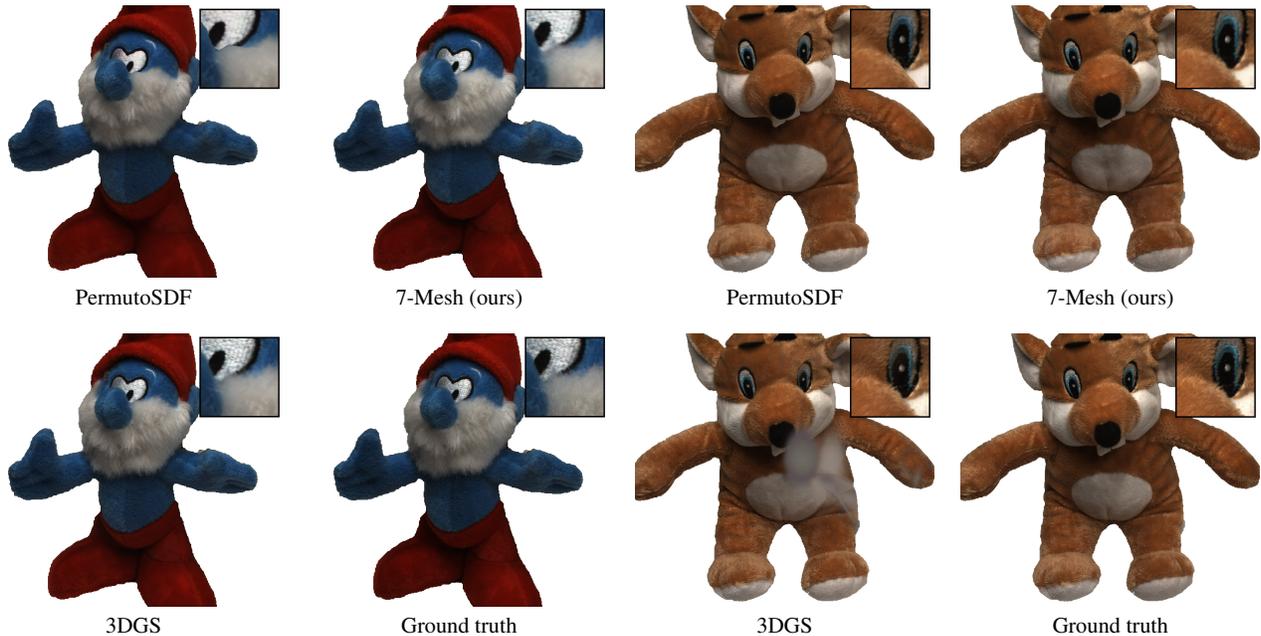


Figure S4. Qualitative comparison of our 7-Mesh model with PermutoSDF [44] and 3DGS [24]. Scenes from the DTU dataset [21].

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 (ϕ_{β_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>fern vase</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	—	—	—

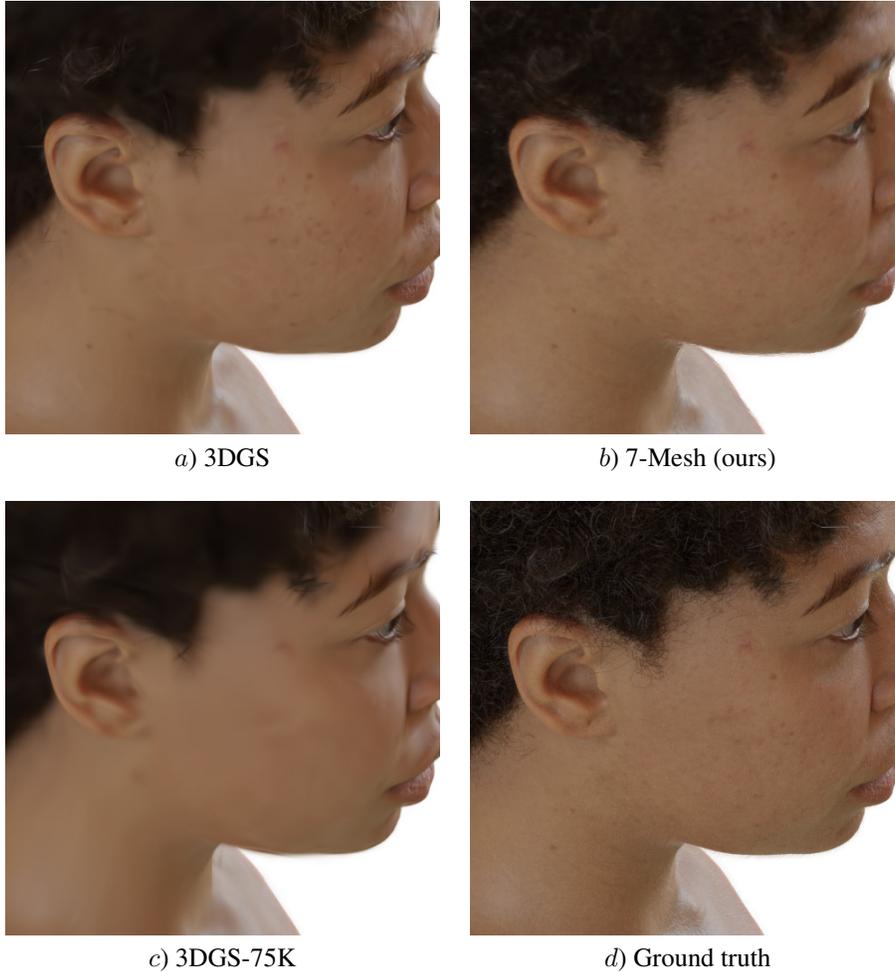


Figure S5. 3DGS [24] demonstrates superior performance in modeling thin structures but is significantly less effective in representing large, textured areas. Our method renders faster than 3DGS-75K on mobile devices. Results on the *khady* scene from Shelly [56]. Quantitative results are in Table 2.

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>fernvase</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

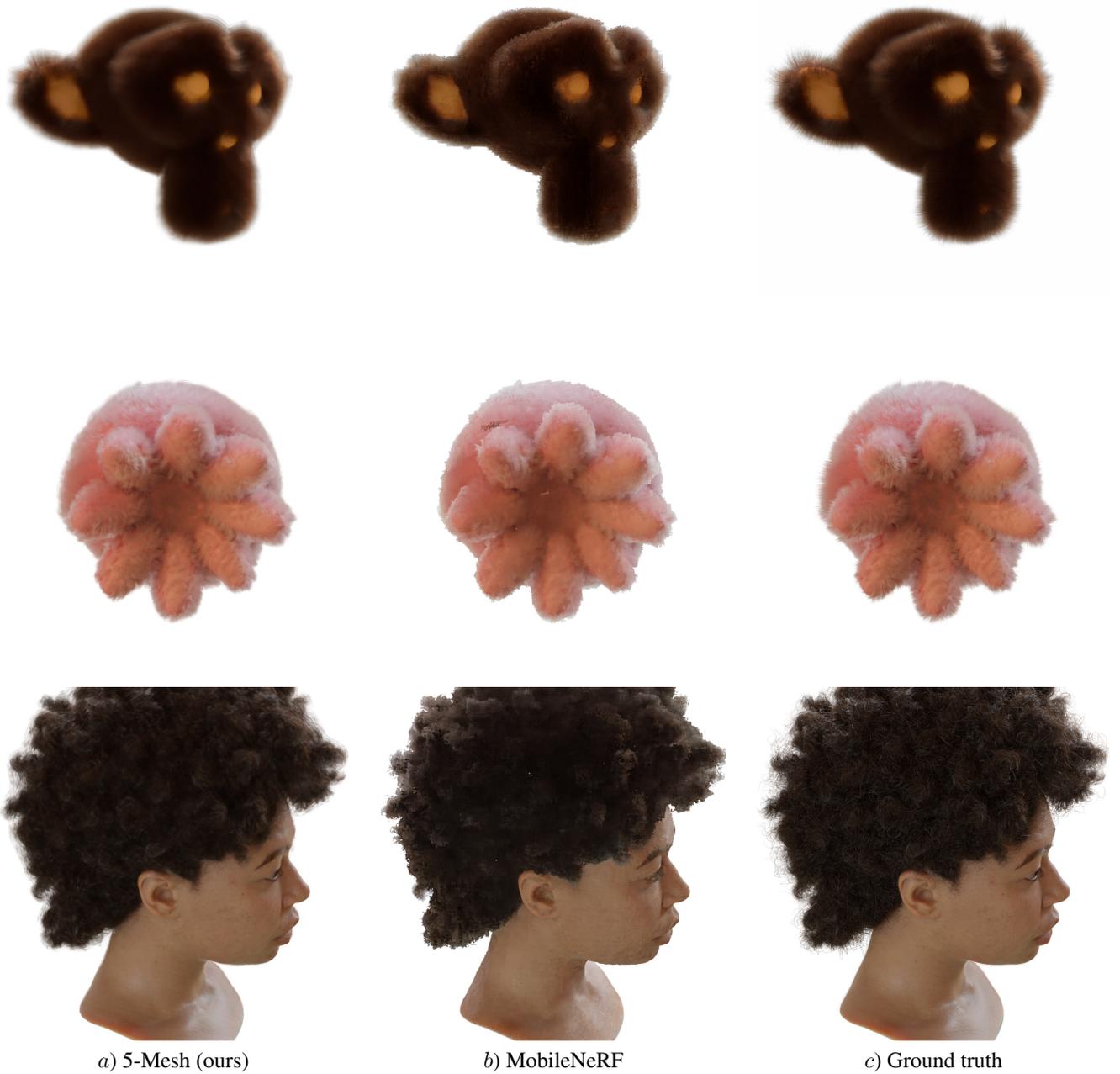


Figure S6. Our method surpasses MobileNeRF [7] in modeling volumetric hair while also achieving superior performance on flat surfaces. Results on *hairy monkey* from QuadFields [47], our custom *plushy* scene and *khady* from Shelly [56]. Quantitative results are in Table 2.



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>fermvase</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 (ϕ_{β_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].