Adaptive Shells for Efficient Neural Radiance Field Rendering

ZIAN WANG*, NVIDIA, University of Toronto, Vector Institute, Canada
TIANCHANG SHEN*, NVIDIA, University of Toronto, Vector Institute, Canada
MERLIN NIMIER-DAVID*, NVIDIA, Switzerland
NICHOLAS SHARP, NVIDIA, USA
JUN GAO, NVIDIA, University of Toronto, Vector Institute, Canada
ALEXANDER KELLER, NVIDIA, Germany
SANJA FIDLER, NVIDIA, University of Toronto, Vector Institute, Canada
THOMAS MÜLLER, NVIDIA, Switzerland
ZAN GOJCIC, NVIDIA, Switzerland

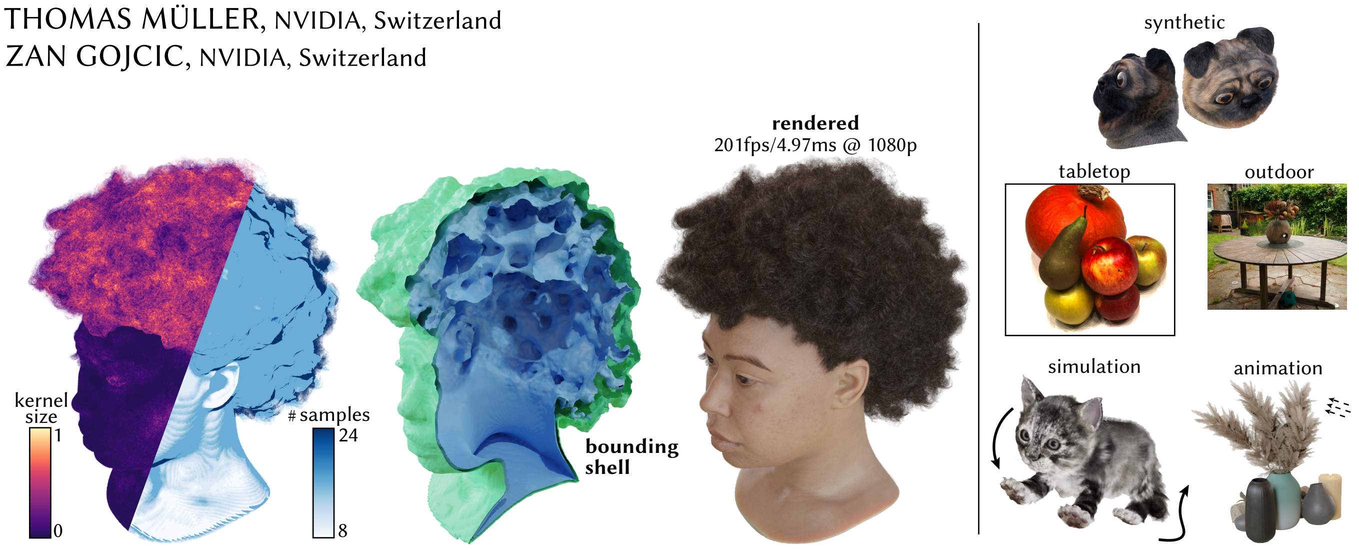


Fig. 1. This work presents an approach for efficiently rendering neural radiance fields by restricting volumetric rendering to a narrow band around the object. *Left:* We first fit a dense neural volume using a new spatially-varying kernel that automatically adapts to be large in volumetric regions such as hair or grass, and small in sharp-surface regions such as skin or furniture. We then extract an explicit bounding mesh of the region to be rendered whose width is determined by the kernel, and render at real-time rates. *Right:* the proposed method is general and effective across a wide range of data and well-suited for downstream applications such as simulation and animation. The face model of the Khady synthetic human shown *left* is courtesy of texturing.xyz.

Neural radiance fields achieve unprecedented quality for novel view synthesis, but their volumetric formulation remains expensive, requiring a huge number of samples to render high-resolution images. Volumetric encodings are essential to represent fuzzy geometry such as foliage and hair, and they are well-suited for stochastic optimization. Yet, many scenes ultimately consist largely of solid surfaces which can be accurately rendered by a single sample per pixel. Based on this insight, we propose a neural radiance formulation that smoothly transitions between volumetric- and surface-based rendering, greatly accelerating rendering speed and even improving visual fidelity. Our method constructs an explicit mesh envelope which spatially bounds a neural volumetric representation. In solid regions, the envelope nearly converges to a surface and can often be rendered with a single sample. To this end, we generalize the NeuS [Wang et al. 2021] formulation with a learned spatially-varying kernel size which encodes the spread of the density, fitting a wide kernel to volume-like regions and a tight kernel to surface-like regions. We then extract an explicit mesh of a narrow band around the surface, with width determined by the kernel size, and fine-tune the radiance field within this band. At inference time, we cast rays against the mesh and evaluate the radiance field only within the enclosed region, greatly reducing the number of samples required. Experiments show that our approach enables efficient rendering at very high fidelity. We also demonstrate that the extracted envelope enables downstream applications such as animation and simulation.

CCS Concepts: • **Computing methodologies** → **Rendering**; **Shape representations**; **Reconstruction**.

Additional Key Words and Phrases: Neural Radiance Fields, Fast Rendering, Level Set Methods, Novel View Synthesis

*Authors contributed equally.

Authors' addresses: Zian Wang, NVIDIA, University of Toronto, Vector Institute, Toronto, Canada, zianw@nvidia.com; Tianchang Shen, NVIDIA, University of Toronto, Vector Institute, Toronto, Canada, frshen@nvidia.com; Merlin Nimier-David, NVIDIA, Zürich, Switzerland, mnimierdavid@nvidia.com; Nicholas Sharp, NVIDIA, Seattle, USA, nsharp@nvidia.com; Jun Gao, NVIDIA, University of Toronto, Vector Institute, Toronto, Canada, jung@nvidia.com; Alexander Keller, NVIDIA, Berlin, Germany, akeller@nvidia.com; Sanja Fidler, NVIDIA, University of Toronto, Vector Institute, Toronto, Canada, sfidler@nvidia.com; Thomas Müller, NVIDIA, Zürich, Switzerland, tmueller@nvidia.com; Zan Gojcic, NVIDIA, Zürich, Switzerland, zgojcic@nvidia.com.

1 INTRODUCTION

Neural radiance fields, which we will refer to as *NeRFs*, have recently emerged as a powerful 3D representation enabling photorealistic novel-view synthesis and reconstruction. Unlike traditional explicit methods for novel-view synthesis, NeRFs forego high-quality mesh reconstruction and explicit surface geometry in favor of neural networks, which encode the volumetric density and appearance of a scene as a function of 3D spatial coordinates and viewing direction. However, the high visual fidelity of NeRFs comes at a great computational cost, as the volume rendering formulation requires a large number of samples along each ray and ultimately prevents real-time synthesis of high-resolution novel views. In tandem, explicit reconstruction and novel-view synthesis have continued to make great progress by leveraging advances in inverse rendering and data-driven priors, but a fidelity gap remains. The goal of this work is to close this gap by developing a neural volumetric formulation that leverages explicit geometry to accelerate performance, without sacrificing quality.

Much recent and concurrent work has likewise sought to improve the efficiency of NeRF representations and volume rendering. An important step towards this goal was the evolution from a global large multi-layer perceptron (MLP) representation [Mildenhall et al. 2020] to local sparse feature fields combined with shallow MLP decoders [Müller et al. 2022; Sara Fridovich-Keil and Alex Yu et al. 2022]. This resulted in several orders-of-magnitude speed-ups. Complementary research to improve the efficiency of NeRFs proposed replacing the neural networks by simpler functions such as spherical harmonics, or baking the volumetric representation onto proxy geometry that accelerates rendering [Chen et al. 2023; Yariv et al. 2023]. The latter formulation enables especially large speedups and facilitates real-time rendering even on commodity devices [Chen et al. 2023]. Yet, doing so compromises the quality as the scene content is projected onto proxy geometry.

In this work, we instead aim to make NeRF rendering more efficient while maintaining or even improving the perceptual quality. To this end, we propose a narrow-band rendering formulation that enables efficient novel-view synthesis, while enjoying the desirable properties of the volumetric representation (Figure 1 *left*). Our method is inspired by the insight that different regions of the scene benefit from different styles of rendering. Indeed, fuzzy surfaces with intricate geometry and complex transparency patterns benefit greatly from exhaustive volume rendering, while conversely, opaque smooth surfaces can be well—or potentially even better—represented by a single sample where the ray intersects the surface. This observation allows us to better distribute the computational cost across the rays by assigning as many samples as needed to faithfully represent the ground-truth appearance.

With the introduction of auxiliary acceleration data structures that promote empty space skipping [Müller et al. 2022], NeRFs can already render images with a varying number of samples per ray. Still, there remain many challenges that prevent the current formulations from efficiently adapting to the local complexity of the scene (Figure 2). First, the memory footprint of grid-based acceleration structures scales poorly with resolution. Second, the smooth inductive bias of MLPs hinders learning a sharp impulse or step function

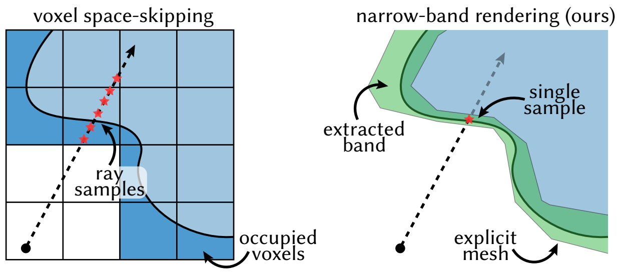


Fig. 2. One state-of-the-art approach to accelerate volumetric rendering is to skip empty voxels, however this still requires multiple samples within occupied voxels (*left*). Our approach extracts a narrow band mesh, for which a single sample at the midpoint is a very good approximation of the surface (*right*).

for volume density, and even if such an impulse was learned it would be difficult to sample it efficiently. Finally, due to the lack of constraints, the implicit volume density field fails to accurately represent the underlying surfaces [Wang et al. 2021], which often limits their application in downstream tasks that rely on mesh extraction.

To remedy the last point, [Wang et al. 2021, 2022a; Yariv et al. 2021] propose to optimize a signed distance function (SDF) along with a kernel size encoding the spread of the density, rather than optimizing density directly. While this is effective for improving surface representations, the use of a global kernel size contradicts the observation that different regions of the scene demand adaptive treatment.

To address the above challenges, we propose a new volumetric neural radiance field representation. In particular: **i**) We generalize the NeuS [Wang et al. 2021] formulation with a *spatially-varying* kernel width that converges to a wide kernel for fuzzy surfaces, while collapsing to an impulse function for solid opaque surfaces without additional supervision. This improvement alone results in an increased rendering quality across all scenes in our experiments. **ii**) We use the learned spatially-varying kernel width to extract a mesh envelope of a narrow band around the surface. The width of the extracted envelope adapts itself to the complexity of the scene and serves as an efficient auxiliary acceleration data structure. **iii**) At inference time, we cast rays against the envelope in order to skip empty space and sample the radiance field only in regions which contribute significantly to the rendering. In surface-like regions, the narrow band enables rendering from a single sample, while progressing to a wider kernel and local volumetric rendering for fuzzy surfaces.

The experiments of Section 4 validate the effectiveness of our formulation across several data sets. In addition, the applications of Section 5 demonstrate the benefits of our representation.

2 RELATED WORK

Synthesizing novel views from a set of images is a longstanding problem in the fields of computer vision and graphics. The classical approaches to novel-view synthesis can be roughly categorized based on the coverage density of the input images. In particular,

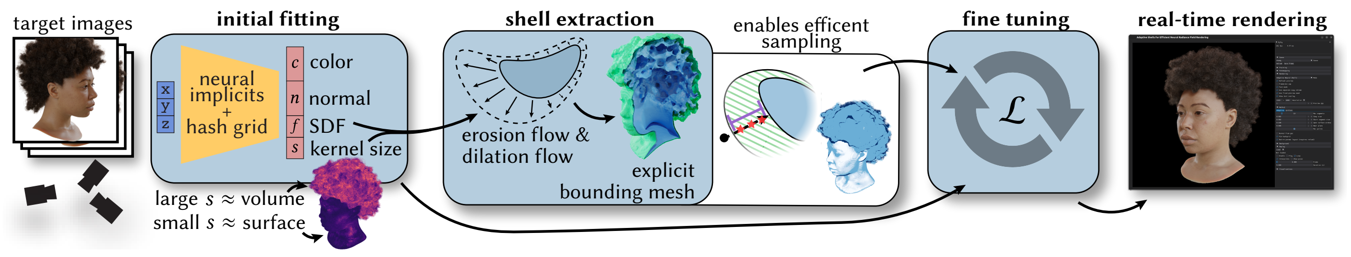


Fig. 3. Overview of the proposed approach. We demonstrate high-fidelity, efficient neural implicit scene reconstruction by efficiently-sampling volumetric rendering inside of an explicit thin shell, which is automatically fit from visual objectives.

light field interpolation methods [Davis et al. 2012; Gortler et al. 1996; Levoy and Hanrahan 1996] assume that the input views are sampled densely and close to the target view. When the input views are sparse, classical methods usually follow a two-stage approach: In the first stage, they construct a proxy geometry from the images using a combination of a multi-view stereo pipeline [Schönberger and Frahm 2016; Schönberger et al. 2016] and point cloud reconstruction methods [Kazhdan et al. 2006; Kazhdan and Hoppe 2013]. In the second stage, the input images are then unprojected onto the geometry either directly in terms of RGB colors [Buehler et al. 2001; Debevec et al. 1996; Waechter et al. 2014; Wood et al. 2000] or, more recently, latent features [Riegler and Koltun 2020, 2021]. Other lines of research have developed specialized methods for certain classes of objects, such as faces (*e.g.* Bi et al. [2021])—although we show results on synthetic human and animal data (Figure 1), the approach presented here is entirely general.

Neural Radiance Fields (NeRFs). NeRF [Mildenhall et al. 2020] have revolutionized the prevailing paradigm of novel-view synthesis, by using a neural network to represent the scenes as a volumetric (radiance) field that may be queried at any location to return the view-dependent radiance and volume density. Mildenhall et al. [2020] synthesize novel views by querying the radiance field along the image rays and accumulating the appearance using volume rendering. The photorealistic quality of NeRF has inspired a large body of follow-up work. NeRF++ [Zhang et al. 2020] analyzed the difficulties of NeRF to represent unbounded scenes and proposed a background formulation based on the inverted sphere representation. MipNeRF [Barron et al. 2021] addressed the aliasing effects with an integrated positional encoding. This work was later extended to unbounded scenes [Barron et al. 2022] by contracting the volume and using an additional proposal network. [Deng et al. 2022] and [Niemeyer et al. 2022] tackled the challenging setting with sparse input views and proposed to regularize the volumetric representation using depth supervision or smoothness constraints and data priors based on normalizing flows, respectively. NeRF-W [Martin-Brualla et al. 2021] has shown how NeRF can be extended to unstructured collections of images captured in-the-wild, by using per-frame learnable latent codes to compensate for appearance differences and a transient embedding to remove dynamic objects. Alternative representations to neural fields include point clouds [Kopanas et al. 2021; Rückert et al. 2021], spheres [Lassner and Zollhöfer 2021], and 3D Gaussians [Kerbl et al. 2023].

Implicit surface representation. The NeRF formulation has two main shortcomings when it comes to modeling surfaces: i) Besides a lack of regularization of the density field, ii) surface extraction has to be performed at an arbitrary level-set of the density field. In combination, these lead to noisy and low-fidelity surface reconstruction. However, with small changes in the formulation, implicit representations combined with volume rendering [Oechsle et al. 2021; Wang et al. 2021, 2022b; Yariv et al. 2021, 2020; Zhang et al. 2021] still appear as a promising alternative to classical surface reconstruction approaches from image data [Schönberger et al. 2016]. For example, instead of directly optimizing the density field, [Wang et al. 2021; Yariv et al. 2021] proposed to decompose it into an SDF and a global kernel size that defines the spread of the density. This allows for extracting accurate surfaces from the zero-level set of the SDF, which can also be regularized using the Eikonal constraint. Similar to NeRFs, implicit surface representations were also combined with local feature fields and auxiliary acceleration data structures [Li et al. 2023; Rosu and Behnke 2023; Tang et al. 2023; Wang et al. 2022a; Zhao et al. 2022] with the goal of improved efficiency and representation capacity. While our method is built on the NeuS [Wang et al. 2021] formulation, our main goal is not to improve the accuracy of the extracted surface. Instead, we utilize the SDF to extract a narrow shell that allows us to adapt the representation to the local complexity of the scene and in turn to accelerate rendering.

Accelerating neural volume rendering. One of the main limitations of NeRFs is the computational complexity of neural volume rendering which slows down both training and inference. Recently, various different directions to accelerate NeRFs have been explored. For example, replacing a global MLP with a (sparse) local feature field combined with a shallow MLP [Chen et al. 2022; Liu et al. 2020; Müller et al. 2022; Sun et al. 2022; Yu et al. 2021] or the spherical harmonics embedding [Chen et al. 2022; Karnewar et al. 2022; Sara Fridovich-Keil and Alex Yu et al. 2022], partitioning the volume into a large number of local (shallow) MLPs [Rebain et al. 2020; Reiser et al. 2021], or using efficient sampling strategies [Hu et al. 2022; Kurz et al. 2022; Lin et al. 2022; Neff et al. 2021], or image-space convolutions [Cao et al. 2022; Wan et al. 2023]. However, even the most optimized volumetric representations [Müller et al. 2022] are still much slower than pure surface-based approaches such as NvDiffRec [Munkberg et al. 2022].

To further increase the efficiency of the inference phase, volumetric representations can be baked onto a proxy surface geometry [Chen et al. 2023; Wan et al. 2023; Yariv et al. 2023] that can be efficiently rendered using high-performance rasterization pipelines. An alternative *"baking"* strategy is to precompute the outputs of the neural network and store them on a (sparse) discrete grid that acts as a lookup during inference [Hedman et al. 2021; Reiser et al. 2023]. In this work, we investigate an alternate approach to speeding up the (volumetric) rendering, by adapting the number of samples required to render each pixel to the underlying local complexity of the scene. Note that our formulation is complementary to the *"baking"* approaches and we consider the combination of both an interesting avenue for future research.

3 METHOD

Our method (see Figure 3) builds on NeRF [Mildenhall et al. 2020] and NeuS [Wang et al. 2021]. Specifically, we generalize NeuS [Wang et al. 2021] with a new spatially-varying kernel (Section 3.2), which improves the quality and guides the extraction of a narrow-band shell (Section 3.3). Then, the neural representation is fine-tuned (Section 3.5) within the shell that significantly accelerates rendering (Section 3.4).

3.1 Preliminaries

NeRF [Mildenhall et al. 2020] represents a scene as a volumetric radiance field that maps a 3D point $\mathbf{x} \in \mathbb{R}^3$ and a viewing direction $\mathbf{d} \in \mathbb{R}^3$ to the volume density σ and the emitted view-dependent color $\mathbf{c} \in \mathbb{R}^3$. This volumetric field is represented by a neural network $\mathrm{NN}_\theta(\cdot)$ with parameters θ, such that $(\mathbf{c}, \sigma) = \mathrm{NN}_\theta(\mathbf{x}, \mathbf{d})$. The scene can then be rendered along a ray $\mathbf{r} = \mathbf{o} + \tau\mathbf{d}$ with origin $\mathbf{o} \in \mathbb{R}^3$ and direction $\mathbf{d} \in \mathbb{R}^3$ from τ_n to τ_f via standard volumetric rendering

$$\mathbf{c}(\mathbf{r}) = \int_{\tau_n}^{\tau_f} \exp\left[\int_{\tau_n}^{\tau} -\sigma(\mathbf{r}(z))dz\right]\sigma(\mathbf{r}(\tau))\mathbf{c}(\mathbf{r}(\tau), \mathbf{d})d\tau\,, \quad (1)$$

which is approximated by numerical integration

$$\mathbf{c}(\mathbf{r}) = \sum_{i=1}^{N_r} \exp\left[-\sum_{j=1}^{i-1}\sigma_j\delta_j\right](1-\exp(-\sigma_i\delta_i))\mathbf{c}(\mathbf{r}, \mathbf{d})_i\,, \quad (2)$$

where N_r denotes the number of samples along the ray $\mathbf{r}$ and δ_i is the distance between two adjacent samples.

To improve geometric surface quality in NeRF-like scene reconstructions, NeuS [Wang et al. 2021] and VolSDF [Yariv et al. 2021] propose to replace the learned density σ by a learned signed distance field f, and then transform f to σ for rendering via a sigmoid-shaped map. The formulation of NeuS optimizes an SDF $(\mathbf{c}, f) = \mathrm{NN}_\theta(\mathbf{x}, \mathbf{d})$ along with a global kernel size s that controls the sharpness of the implied density. To evaluate volume rendering (Equation 2) the SDF value f at $\mathbf{x}$ is transformed to a density σ by

$$\sigma = \max\left(-\frac{\frac{d\Phi_s}{d\tau}(f)}{\Phi_s(f)}, 0\right), \qquad \Phi_s(f) = (1+\exp(-f/s))^{-1}, \quad (3)$$

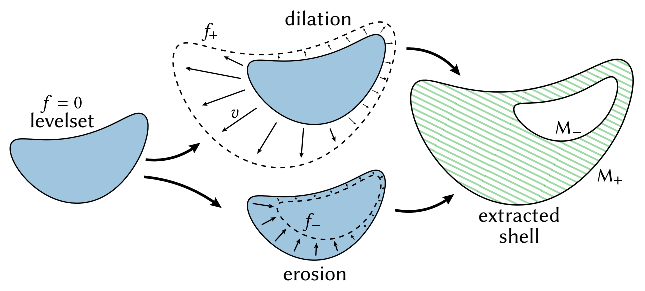
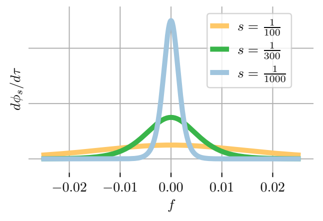


Fig. 4. After fitting an initial SDF and spatially varying kernel, we apply level set flows to extract an adaptive shell via dilation and erosion. For illustrative purposes, the adaptive shell is enlarged; in practice it very tightly encloses sharp surfaces.



where f is implicitly $f(\mathbf{r}(\tau))$ along a ray. Intuitively, a small s results in a wide kernel with a fuzzy density, while in the limit $\lim_{s\to 0} d\Phi_s/d\tau$ approximates a sharp impulse function (see inset). This SDF-based formulation allows for the use of an Eikonal regularizer during training, which encourages the learned f to be an actual distance function, resulting in a more accurate surface reconstruction. The relevant losses are discussed in Section 3.5.

3.2 Spatially-Varying Kernel Size

The NeuS SDF formulation is highly effective, yet, it relies on one *global* kernel size. In combination with the Eikonal regularization this implies a constant spread of the volume density across the whole scene. However, a one-size-fits-all approach does not adapt well to scenes that contain a mixture of "sharp" surfaces (*e.g.* furniture or cars) and "fuzzy" volumetric regions (*e.g.* hair or grass).

Our first contribution is to augment the NeuS formulation with a spatially-varying, *locally* learned kernel size s as an additional neural output that is dependent on the input 3D position $\mathbf{x}$. The extended network becomes $(\mathbf{c}, f, s) = \mathrm{NN}_\theta(\mathbf{x}, \mathbf{d})$ (see the implementation details in Section 4.1). During training, we additionally include a regularizer that promotes the smoothness of the kernel size field (Section 3.5). This neural field can still be fit from only color image supervision, and the resulting spatially-varying kernel size automatically adapts to the sharpness of the scene content (Figure 7). This enhanced representation is independently valuable, improving reconstruction quality in difficult scenes, but importantly it will serve to guide our explicit shell extraction in Section 3.3, which greatly accelerates rendering.

3.3 Extracting an Explicit Shell

The adaptive shell delimits the region of space which contributes significantly to the rendered appearance, and is represented by two explicit triangle meshes. When s is large the shell is thick, corresponding to volumetric scene content, and when s is small the shell is thin, corresponding to surfaces. After the implicit fields

s and f have been fit as described in Section 3.2, we extract this adaptive shell once as a post-process.

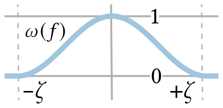
In Equation 3 the magnitude of the quantity f/s in the sigmoid exponent determines the rendering contribution along a ray (see the inset figure in Section 3.1). It is tempting to simply extract a band where $|f/s| < \eta$ for some η as the region that makes a significant contribution to the rendering. However, the learned functions quickly become noisy away from the $f = 0$ level set, and cannot be sufficiently regularized without destroying fine details. Our solution is to separately extract an inner boundary as an erosion of the $f = 0$ level set, and an outer boundary as its dilation (Figure 4), both implemented via a regularized, constrained level set evolution tailored to the task.

In detail, we first sample the fields f and s at the vertices of a regular grid. We then apply a level set evolution to f, producing a new eroded field SDF$_-$, and extract the SDF$_- = 0$ level set as the inner shell boundary via marching cubes. A separate, similar evolution yields the dilated field SDF$_+$, and the SDF$_+ = 0$ level set forms the outer shell boundary. We define both level sets separately: the dilated outer surface should be smooth to avoid visible boundary artifacts, while the eroded inner surface needs not be smooth, but must only exclude regions which certainly do not contribute to the rendered appearance.

Recall that a basic level set evolution of a field a is given by $\partial a/\partial t = -|\nabla a|\, v$, where v is the desired scalar outward-normal velocity of the level set. Our constrained, regularized flow on f is then

$$\frac{\partial f}{\partial t} = -|\nabla f| \left(v(f_0, s) + \lambda_{\text{curv}} \nabla \cdot \frac{\nabla f}{|\nabla f|} \right) \omega(f), \tag{4}$$

where f_0 here denotes the initial learned SDF, the divergence term is a curvature smoothness regularizer with weight λ_{curv}. The soft falloff ω (see inset) limits the flow to a window around the level set:



$$\omega(f) = \tfrac{1}{2}\left(1 + \cos(\pi \operatorname{clamp}(f/\zeta, -1., 1.)\right), \tag{5}$$

with window width ζ. To dilate the level set, the velocity is chosen to fill all regions with density $\sigma > \sigma_{\min}$ for a ray incoming in the normal direction

$$v_{\text{dilate}}(f_0, s) = \begin{cases} \beta_d \sigma(f_0, s) & \sigma(f_0, s) > \sigma_{\min} \\ 0 & \sigma(f_0, s) \le \sigma_{\min} \end{cases}, \tag{6}$$

with β_d as a scaling coefficient. We use $\zeta = 0.1$, and $\lambda_{\text{curv}} = 0.01$. To erode the level set, the velocity is inversely-proportional to density, so that the shell expands inward quickly for low density regions and slowly for high density regions

$$v_{\text{erode}}(f_0, s) = \min\left(v_{\max}, \beta_e \frac{1}{\sigma(f_0, s)}\right), \tag{7}$$

where here we use $\zeta = 0.05$, and $\lambda_{\text{curv}} = 0$. These velocities lead to a short-distance flow, and thus a narrow shell where s is small and the content is surface-like. They lead to a long-distance flow and hence a wide shell where s is large and the content is volume-like.

We compute the dilated field SDF$_+$ and eroded field SDF$_-$ respectively by forward-Euler integrating this flow on the grid for 50 steps of integration, computing derivatives via spatial finite differences. We do not find numerical redistancing to be necessary. Finally, we clamp the results SDF$_- \leftarrow \max(f_0, \text{SDF}_-)$ and SDF$_+ \leftarrow \min(f_0, \text{SDF}_+)$, to ensure that the eroded field only shrinks the level set, and the dilated flow only grows the level set. The SDF$_+ = 0$ and SDF$_- = 0$ level sets are extracted via marching cubes as the outer and inner shell boundary meshes M$_+$ and M$_-$, respectively. Figure 5 visualizes the resulting fields. Further details are provided in Procedure 1 and 2 of the Appendix.

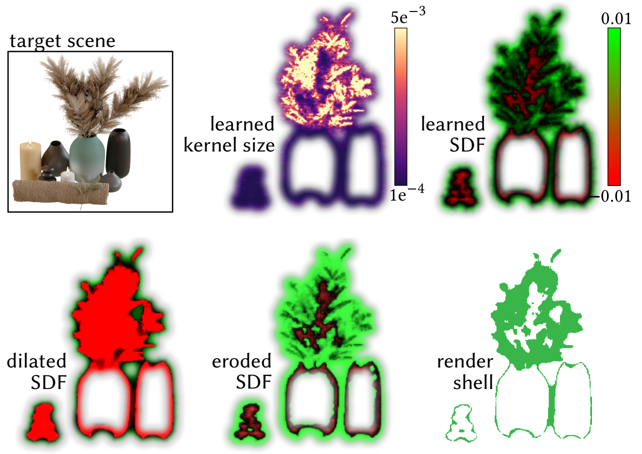
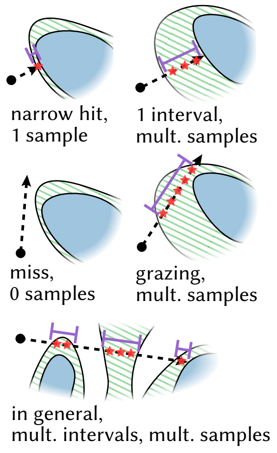


Fig. 5. Given kernel size s and SDF f learned from Section 3.2 (*top*), we apply a erosion and dilation flows to f (*bottom middle and left*) to extract a narrow shell in which we efficiently render (*bottom right*). Here, we visualize each quantity on a 2D slice through a scene. For clarity, we show the fields only nearby the adaptive shell that is ultimately rendered.

3.4 Narrow-Band Rendering

The extracted adaptive shell serves as an auxiliary acceleration data structure to guide the sampling of points along a ray (Equation 2), enabling us to efficiently skip empty space and sample points only where necessary for high perceptual quality. For each ray, we use hardware-accelerated ray tracing to efficiently enumerate the ordered intervals defined by the intersection of the ray and the adaptive shell. Within each interval we query equally-spaced samples. Our renderer does not require any dynamic adaptive sampling or sample-dependent termination criteria, which facilitates high-performance parallel evaluation.

In detail, we first build ray tracing acceleration data structures for both the outer mesh M$_+$ and inner mesh M$_-$ We then cast each ray against the meshes, yielding a series of intersections where the ray enters or exits the mesh, partitioning the ray into zero or more intervals contained in the shell (see inset). For each interval



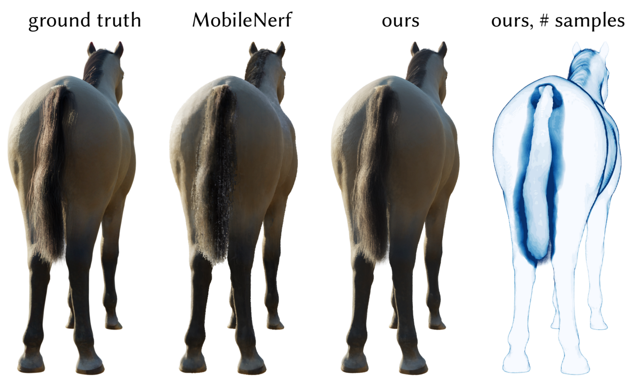


Fig. 6. Pure surface-based representations struggle to represent fuzzy surfaces such as the tail. On the other hand, our method adapts the narrow shell to the local complexity of the scene, using a single sample ○ for the sharp skin surface and up to 16 samples ● for the tail.

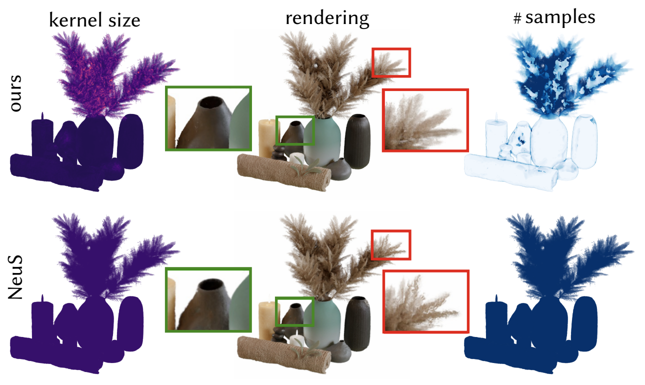


Fig. 7. The original NeuS [Wang et al. 2021] formulation uses a single global kernel size s. On complex scenes with varying content, the global s value converges to an average which is too small for volumetric parts 🟥, and too large for sharp surfaces 🟩. Instead, our locally varying kernel size adapts to the scene, in-turn allowing us to reduce the number of samples to a single sample ○ for sharp surfaces and up to 32 samples ● for the fern (*top right*). NeuS uses a constant 384 samples per pixel (*bottom right*).

with width w, a target inter-sample spacing δ_s, and a single-sample threshold w_s, we compute the number of samples as $\text{ceil}(\max(w - w_s, 0)/\delta_s) + 1$. We cap the maximum number of samples to $N_{\max}$, and equidistantly sample the interval. Note that if the interval has $w < w_s$, a single sample is taken at the center of the interval. When an interval ends because the ray hits the inner mesh M_-, we do not process any subsequent samples, as this represents the interior of a solid object. Otherwise, we process intervals until the ray exits the scene or we hit a maximum cap, accumulating the contributions as in Equation 2.

Note that this procedure can be implemented by first generating all samples within all intervals, and then performing a single batched MLP inference pass, which improves throughput. For surfaces, our narrow-band sampling often amounts to just a hardware-accelerated ray tracing, followed by a single network evaluation, while for fuzzy regions it densely samples only where necessary—in either case, performance is greatly accelerated (Table 1). More algorithmic details are included in Procedure 3 of the Appendix.

3.5 Losses and Training

We optimize the parameters of our representation in two stages. In the first stage, we use the fully volumetric formulation described in Sections 3.1 and 3.2 and minimize the following objective

$$\mathcal{L} = \mathcal{L}_{\mathbf{c}} + \lambda_e \mathcal{L}_e + \lambda_s \mathcal{L}_s + \lambda_{\mathbf{n}} \mathcal{L}_{\mathbf{n}} \tag{8}$$

with the weights $\lambda_{\mathbf{c}} = 1$, $\lambda_e = 0.1$, $\lambda_{\mathbf{n}} = 0.1$, and $\lambda_s = 0.01$ for all experiments. Here $\mathcal{L}_{\mathbf{c}}$ is the standard pixel-wise color loss against calibrated ground-truth images

$$\mathcal{L}_{\mathbf{c}} = \frac{1}{|\mathcal{R}|} \sum_{\mathbf{r} \in \mathcal{R}} |\mathbf{c}(\mathbf{r}) - \mathbf{c}_{\text{gt}}(\mathbf{r})| \tag{9}$$

and $\mathcal{L}_e$ is the Eikonal regularizer as in [Wang et al. 2021]

$$\mathcal{L}_e = \frac{1}{|\mathcal{X}|} \sum_{\mathbf{x} \in \mathcal{X}} (||\nabla f(\mathbf{x})||_2 - 1)^2, \tag{10}$$

where $\mathcal{R}$ and $\mathcal{X}$ denote the set of rays and samples along the rays, respectively. $\nabla f(\mathbf{x})$ can be obtained either analytically [Wang et al. 2021, 2022a; Yariv et al. 2021] or through finite differences [Li et al. 2023; Wang et al. 2023]. We use the latter approach. The loss $\mathcal{L}_s$ regularizes the spatially varying kernel size introduced in our formulation for smoothness

$$\mathcal{L}_s = \frac{1}{\mathcal{X}} \sum_{\mathbf{x} \in \mathcal{X}} || \log \left[s(\mathbf{x}) \right] - \log \left[s(\mathbf{x} + \mathcal{N}(0, \varepsilon^2)) \right] ||_2, \tag{11}$$

where $\mathcal{N}(0, \varepsilon^2)$ denotes samples from the normal distribution with standard deviation ε. Lastly, we incorporate the loss

$$\mathcal{L}_{\mathbf{n}} = \frac{1}{|\mathcal{X}|} \left\| \mathbf{n}(\mathbf{x}) - \frac{\nabla f(\mathbf{x})}{||\nabla f(\mathbf{x})||_2} \right\|_2, \tag{12}$$

internal to our network architecture (Section 4.1). Like NeuS, we will leverage geometric normals as an input to a shading subnetwork, but we find that predicting these normals internally improves inference performance *vs.* gradient evaluation. $\mathcal{L}_{\mathbf{n}}$ serves to train these predicted normals to remain approximately faithful to ones obtained through the finite differences of the underlying SDF field $\nabla f(\mathbf{x})$ [Li et al. 2023; Wang et al. 2023].

After the implicit field has been fit, we extract the adaptive shell as in Section 3.3. While initial training requires dense sampling along rays, our explicit shell now allows narrow-band rendering to concentrate samples only in significant regions. We therefore fine-tune the representation within the narrow band, now with only $\mathcal{L}_{\mathbf{c}}$—it is no longer necessary to encourage a geometrically-nice representation, as we have already extracted the shell and restricted the sampling to a small band around the scene content. Disabling regularization enables the network to devote its whole capacity to fit the visual appearance, which leads to improved visual fidelity (Table 4). In Procedure 4 of the Appendix, we also present the training pipeline with algorithm details.

4 EXPERIMENTS

In this section, we provide low-level details of our implementation and evaluate our method in terms of rendering quality and efficiency on four data sets that range from synthetic object-level *Shelly*, *NeRF-Synthetic* [Mildenhall et al. 2020] and tabletop *DTU* [Jensen et al. 2014] data, to real-world large outbound scenes *MipNeRF360* [Barron et al. 2022]. For comparisons, we treat Instant NGP [Müller et al. 2022] as our volumetric baseline, due to its balance between high fidelity and efficiency. In addition, we compare to prior methods that were optimized either for fidelity [Barron et al. 2021, 2022; Wang et al. 2021; Yuan et al. 2022] or rendering efficiency [Chen et al. 2023; Guo et al. 2023; Yariv et al. 2023].

When running NeRF [Mildenhall et al. 2020] and Mip-NeRF [Barron et al. 2021] on *DTU* and *Shelly*, we use the implementation from Nerfstudio [Tancik et al. 2023]. For other methods and experiment settings, we use their official implementations.

4.1 Architecture Details

Following the state of the art in neural volumetric rendering, we represent our neural field as a combination of a feature field and a small (decoder) neural network. Specifically, we use a multi-resolution hash encoding [Müller et al. 2022] $\Psi(\cdot)$ with 14 levels, where each level is represented by a hash-table with 2^{22} two-dimensional features. The voxel grid resolution of our feature field grows from $16^3 \longrightarrow 4096^3$ for *Shelly* and *NeRFSynthetic*, and from $16^3 \longrightarrow 8192^3$ for the other data sets. The features at each level are obtained through tri-linear interpolation before being concatenated to form the feature embedding $\Psi(\cdot) \in \mathbb{R}^{28}$. This is further concatenated with the sample coordinates and input to the geometry network $(f, 1/s, \mathbf{f}_{\text{geo}}, \mathbf{n}) = \text{NN}_\theta^{\text{geo}}([\Psi(\mathbf{x}), \mathbf{x}])$ which is an MLP with a single hidden layer and $31 \longrightarrow 64 \longrightarrow 31$ dimensions. Apart from the SDF value f and kernel size s, $\text{NN}_\theta^{\text{geo}}$ also outputs a geometry latent feature $\mathbf{f}_{\text{geo}} \in \mathbb{R}^{26}$ and the normal vector $\mathbf{n} \in \mathbb{R}^3$ which are combined with $\mathbf{x}$ and the encoded view direction $\mathbf{d}$ as input to the radiance network $\mathbf{c} = \text{NN}_\theta^{\text{rad}}([\gamma(\mathbf{d}), \mathbf{f}_{\text{geo}}, \mathbf{n}, \mathbf{x}])$ that predicts the emitted color. Here, $\text{NN}_\theta^{\text{rad}}$ is an MLP with two hidden layers and dimensions $48 \longrightarrow 64 \longrightarrow 64 \longrightarrow 3$. To reduce the computational cost, we directly predict the normal vector $\mathbf{n}$ with an MLP rather than computing it as the gradient of the underlying SDF field as done in NeuS [Wang et al. 2021]. Finally, to encode the input direction $\mathbf{d}$, we use the spherical harmonic basis up to degree 4, such that $\gamma(\mathbf{d}) \in \mathbb{R}^{16}$. The dimensions of all layers in both networks and the feature field were selected for high throughput on modern GPU devices.

4.2 Implementation

The training stage of our method is implemented in PyTorch [Paszke et al. 2017], while the inference stage is implemented in Dr.Jit [Jakob et al. 2022]. To achieve real-time inference rates, we rely on the automatic kernel fusion performed by Dr.Jit as well as GPU-accelerated ray-mesh intersection provided by OptiX [Parker et al. 2010]. While the inference pass is implemented with high-level Python code, the asynchronous execution of large fused kernels hides virtually all of the interpreter's overhead. Combined with the algorithmic improvements described above, we achieve frame rates from 40 fps (25 ms/frame) on complex outdoor scenes to 300 fps (3.33 ms/frame) on object-level scenes, at 1080p resolution on a single RTX 4090 GPU. A performance comparison to Instant NGP [Müller et al. 2022] on four data sets is given in Table 1. Note that in this work, we focused on inference performance only, and have not yet applied these performance optimizations to the training procedure. Detailed pseudo-code is given in Procedures 1, 2, 3 and 4 of the Appendix.

4.3 Evaluation Metrics

In order to evaluate the rendering quality, we report the commonly used peak signal-to-noise ratio (*PSNR*), learned perceptual image patch similarity (*LPIPS*), and structural similarity (*SSIM*) metrics. Unfortunately, evaluating the efficiency of the methods is less straightforward as the complexity of the method is often intertwined with the selected hardware and low-level implementation details. Indeed, reporting only the number of frames-per-second (FPS) or the time needed to render a single frame may paint an incomplete picture. We therefore additionally report the number of samples along the ray that are required to render each pixel. While the number of samples along the ray also does not tell the whole story as the per-sample evaluation can have different computational complexity, combining all metrics provides a good assessment of the computational complexity of the individual methods.

4.4 *Shelly* Data Set

The *NeRFSynthetic* data set that was introduced in [Mildenhall et al. 2020] is still one of the most widely used data sets to evaluate novel-view synthesis methods. Yet, it mainly consists of objects with sharp surfaces that can be well-represented by surface rendering methods [Munkberg et al. 2022], and thus does not represent the challenge of general scene reconstruction. This motivated us to introduce a new synthetic data set, which we name *Shelly*. It covers a wider variety of appearance including fuzzy surfaces such as hair, fur, and foliage. *Shelly* contains six object-level scenes: Khady, Pug, Kitty, Horse, Fernvase and Woolly. For each scene, we have rendered 128 training and 32 test views from random camera positions distributed on a sphere with a fixed radius. We are grateful to the original artists of these models: jhon maycon, Pierre-Louis Baril, abdoubouam, ckat609, the BlenderKit team, and texturing.xyz.

Table 2 shows quantitative results, while the novel views are qualitatively compared in Figure 8. Our method significantly outperforms prior methods across all quality metrics achieving more than 2dB higher PSNR than Instant NGP. Figure 8 demonstrates that surface-based rendering methods (MobileNerf) struggle to represent fuzzy surfaces. On the other hand, our method aligns its representation to the complexity of the scene. For example, Figure 6 shows that our method represents the skin of the horse as a sharp surface, while using a wider kernel for its tail, which benefits from volumetric rendering.

4.5 *DTU* Data Set

We consider 15 tabletop scenes from the *DTU* data set [Jensen et al. 2014]. These scenes were captured by a robot-held monocular RGB camera, and are commonly used to evaluate implicit surface representations. We follow prior work [Wang et al. 2021; Yariv et al. 2021] and task the methods to represent the full scene, but evaluate the performance only within the provided object masks.

Table 2 depicts that our method outperforms all baselines across all evaluation metrics. Qualitative results are provided in Figure 9.

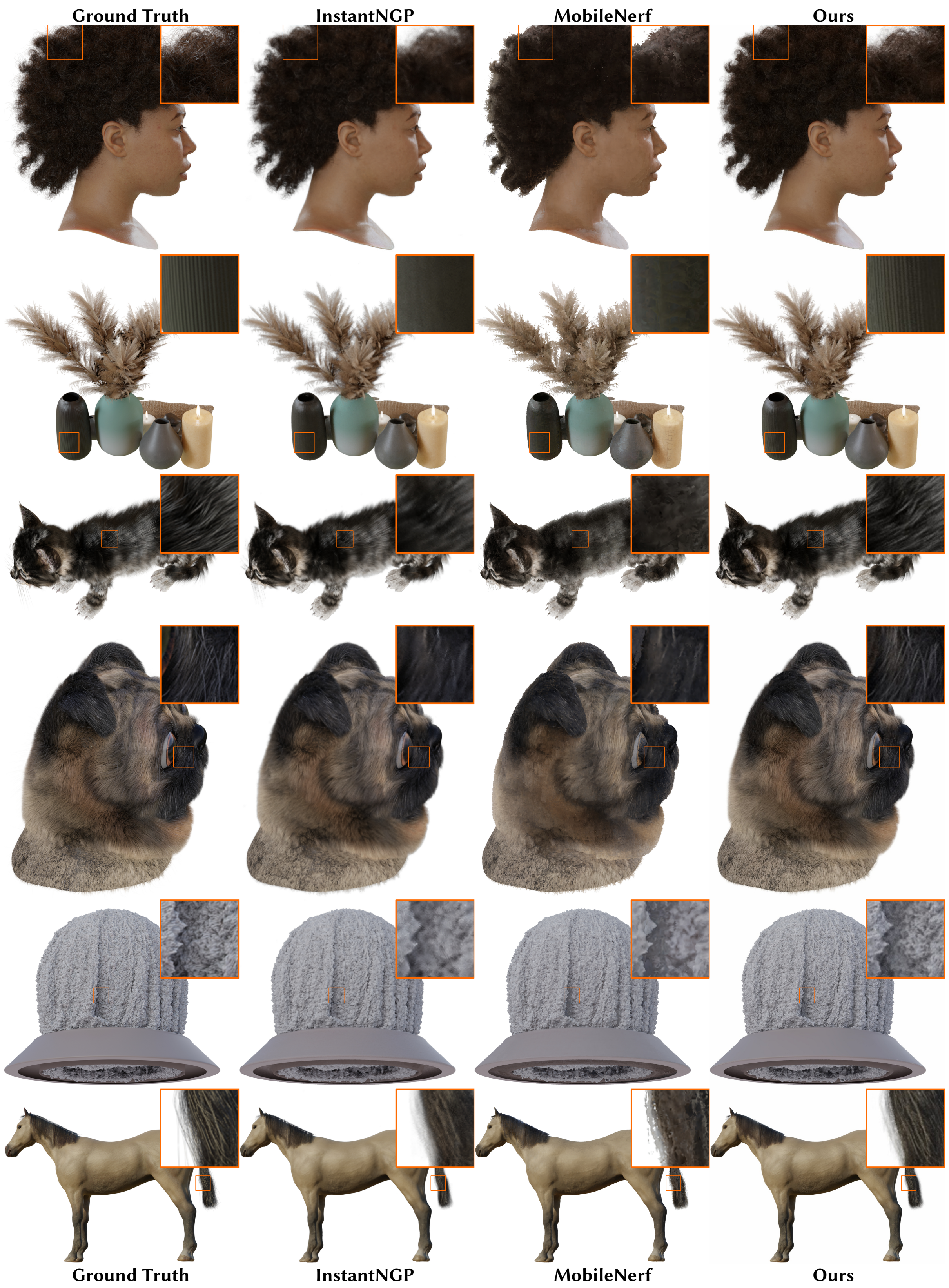


Fig. 8. A gallery of results on the test-views of our *Shelly* data set.

Table 1. Performance comparisons on all four data sets, measured at 1080p without GUI overhead using an RTX 4090 GPU. Our adaptive sample placement and mesh-based empty-space skipping technique allows us to outperform Instant NGP without compromising visual fidelity. *Note that Instant NGP's performance on the DTU data set was hindered by a large number of background samples, and is therefore not necessarily indicative of a real use case: the user may specify a tighter scene bounding box to focus the samples on the main scene contents.*

	Ours			Instant NGP [Müller et al. 2022]		
	Sample count ↓	ms/frame ↓	FPS ↑	Sample count ↓	ms/frame ↓	FPS ↑
Shelly	2.07	3.81	262.69	2.89	11.74	85.16
DTU	5.11	6.37	157.00	56.10	123.31	8.10
NeRFSynthetic	1.98	3.56	280.68	3.19	14.20	70.40
MipNeRF360	17.05	27.64	36.18	45.62	93.57	10.69

Table 2. Quantitative results on *Shelly* data set, *DTU* data set and *NeRFSynthetic* data set. We report PSNR, LPIPS and SSIM. Our method achieves better results across all metrics on *Shelly* and *DTU* and comparable results on *NeRFSynthetic*. *Real-time* denotes methods that achieve >30FPS at 1080p. On *Shelly* and *DTU*, we run NeRF and Mip-NeRF with Nerfstudio [Tancik et al. 2023], and use official implementation for other methods. Baselines of *NeRFSynthetic* are from the original papers. Detailed results for each object/scene are provided in the Supplement.

		Shelly			*DTU*			*NeRFSynthetic*		
		PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓
offline	NeRF [Mildenhall et al. 2020]	31.27	0.893	0.157	28.51	0.894	0.183	31.01	0.947	0.081
	NeuS [Wang et al. 2021]	29.98	0.893	0.158	28.92	0.913 ⚪	0.168	/	/	/
	Mip-NeRF [Barron et al. 2021]	32.59	0.899	0.148	28.90	0.900	0.179	33.09 ⚪	0.961 ⚪	0.043 🟡
	Ours (full ray)	34.26 ⚪	0.932 ⚪	0.104 ⚪	33.51 🟡	0.901	0.081 ⚪	32.51 🟤	0.962 🟡	0.048 ⚪
real-time	I-NGP [Müller et al. 2022]	33.22 🟤	0.922 🟤	0.125 🟤	31.37 🟤	0.932 ⚪	0.139 🟤	33.18 🟡	/	/
	MobileNeRF [Chen et al. 2023]	31.62	0.911	0.129	/	/	/	30.90	0.947	0.062
	VMesh [Guo et al. 2023]	/	/	/	/	/	/	30.70	0.947	0.060
	Ours	36.02 🟡	0.954 🟡	0.079 🟡	33.37 ⚪	0.964 🟡	0.077 🟡	31.84	0.957 🟤	0.056 🟤

Different from the *Shelly* data set, the performance of *Ours* on the *DTU* data set is slightly lower than that of *Ours (full ray)* in terms of PSNR. We hypothesize that this is due to the distribution of the camera poses that observe the scene only from a single direction. This hinders constraining the neural field and hence also the adaptive shell extraction. The same reason also contributes to a significant increase in the sample count for Instant NGP (see Table 1).

4.6 *NeRFSynthetic* Data Set

The *NeRFSynthetic* data set introduced in [Mildenhall et al. 2020] contains 8 synthetic objects rendered in Blender and is widely adopted to evaluate the quality of novel view synthesis methods.

As shown in Table 2, our method can achieve comparable quality to the state of the art methods Mip-NeRF [Barron et al. 2021] and I-NGP [Müller et al. 2022], but with a much faster runtime performance (Table 1). Our method also achieves better image quality compared to recent works optimized for rendering efficiency [Chen et al. 2023; Guo et al. 2023][1].

4.7 *MipNeRF360* Data Set

The MipNeRF-360 data set [Barron et al. 2022] is a challenging real-world data set that contains large indoor and outdoor scenes captured from 360° camera views[2]. The scenes feature a complex central object accompanied by a highly detailed background. To better represent the background, we follow [Yariv et al. 2023] and extend our method with the scene contraction proposed in [Barron et al. 2022] (more details are provided in the supplemental document).

Table 3 provides the quantitative results and the qualitative comparison is depicted in Figure 10. Our method achieves comparable performance to other *interactive* methods. Directly compared to I-NGP, our proposed narrow-band formulation can reduce the number of samples by a factor of three, resulting in fivetimes higher average frame rates at comparable rendering quality. We note that on this data set, performance and quality depend significantly on the background, which our approach is not specialized to handle.

4.8 Performance Evaluation

We compare the performance of our method to the most efficient volumetric baseline, Instant NGP [Müller et al. 2022], in Table 1. To ensure a fair comparison, we render the same test views for both methods at 1080p resolution and remove the GUI overhead. The comparison was run on a single RTX 4090 GPU. Our narrow-band rendering formulation can efficiently reduce the number of samples along the ray (up to 10 times) which results in significantly reduced inference time per frame. On the challenging outbound 360 scenes, our method already runs at real-time rates. Yet, additional speedups could be achieved by further studying the interaction of our adaptive sample placement with the spatial remapping employed in these scenes.

[1]Note that VMesh [Guo et al. 2023] also optimizes for disk storage, which is an orthogonal direction to this paper.
[2]In our evaluation, we exclude the two scenes with license issues: *Flowers* and *Treehill*.

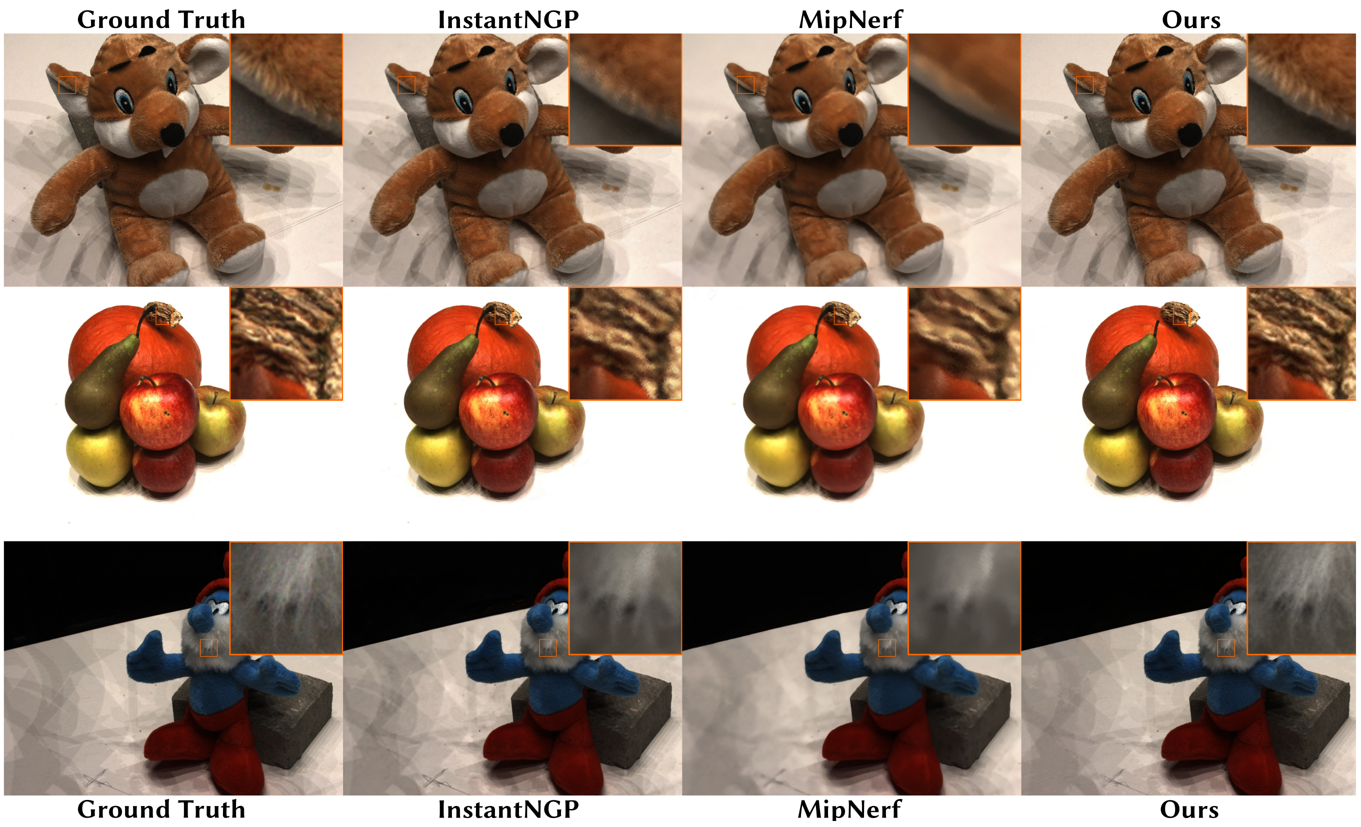


Fig. 9. A gallery of results on the *DTU* data set.

Table 3. Quantitative results on the *MipNeRF360* data set. We report the PSNR, LPIPS and SSIM results for each object and compare them to baselines. Our method achieves a performance comparable to the baselines while being significantly faster during inference (see Table 1). In our comparison, we exclude the two scenes with license issues: Flowers, Treehill.

		Outdoor scenes			Indoor scenes		
		PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓
offline	NeRF [Mildenhall et al. 2020]	22.20	0.485	0.501	26.84	0.790	0.370
	Mip-NeRF [Barron et al. 2021]	22.02	0.505	0.484	26.98	0.798	0.361
	Mip-NeRF 360 [Barron et al. 2022]	25.92 🟡	0.747 🟡	0.244 🟡	31.72 🟡	0.917 🟡	0.179 🟡
	Ours (full ray)	24.30 ⚪	0.703 ⚪	0.316 ⚪	29.04	0.900 ⚪	0.239 ⚪
interactive	I-NGP [Müller et al. 2022]	23.90 🟤	0.648 🟤	0.369	29.47 ⚪	0.877 🟤	0.273 🟤
	MobileNeRF [Chen et al. 2023]	22.90	0.524	0.463	25.74	0.757	0.453
	BakedSDF [Yariv et al. 2023]	23.40	0.577	0.351 🟤	27.20	0.845	0.300
	Ours	23.17	0.606	0.389	29.19 🟤	0.872	0.285

4.9 Ablation Study

We ablate our design choices on the *Shelly* data set in Table 4. In line with our motivation in Section 3.2, the spatially-varying kernel size provides the required flexibility to adapt to the local complexity of the scene which results in improvement across all metrics. Using a fixed SDF threshold to extract the band requires seeking a compromise between an adaptive shell that is too narrow to represent fuzzy surfaces (threshold 0.02) or an increased sample count (threshold 0.05). Instead, our formulation can automatically adapt to the local complexity of the scene leading to higher quality metrics and lower sample count. As described in Section 3.5, we disable the regularization terms after shell extraction to devote more capacity to fit the visual appearance. Comparing *Ours (keep regularization)* with *Ours*, this leads to improved visual fidelity.

Table 4. Ablating our method on the *Shelly* data set. SV Kernel denotes the spatially varying kernel as introduced in Section 3.2. *Band, fixed* denotes the shell is not adaptive but extracted for a given SDF threshold.

Model	PSNR ↑	LPIPS ↓	SSIM ↑	Sample ↓
Ours (full ray, w/o SV kernel)	32.99	0.115	0.921	384
Ours (full ray)	34.26	0.104	0.932	384
Ours (band, fixed ±0.05)	33.83	0.110	0.928	4.51
Ours (band, fixed ±0.02)	31.14	0.136	0.913	2.29
Ours (keep regularization)	34.22	0.085	0.948	1.74
Ours	36.02	0.079	0.954	1.74

In Figure 11, we ablate our method and study how image quality and runtime change with different sample counts. We vary the sample step size δ_s in narrow-band rendering (Section 3.4) to produce varying sample counts, and keep other hyperparameters unchanged. The PSNR is sensitive to sample counts when the samples are insufficient (0.25×-1×), and the image quality starts to saturate as the sample counts go higher (1×-4×). In most scenes, the runtime performance is approximately linear w.r.t. the sample count. For simpler

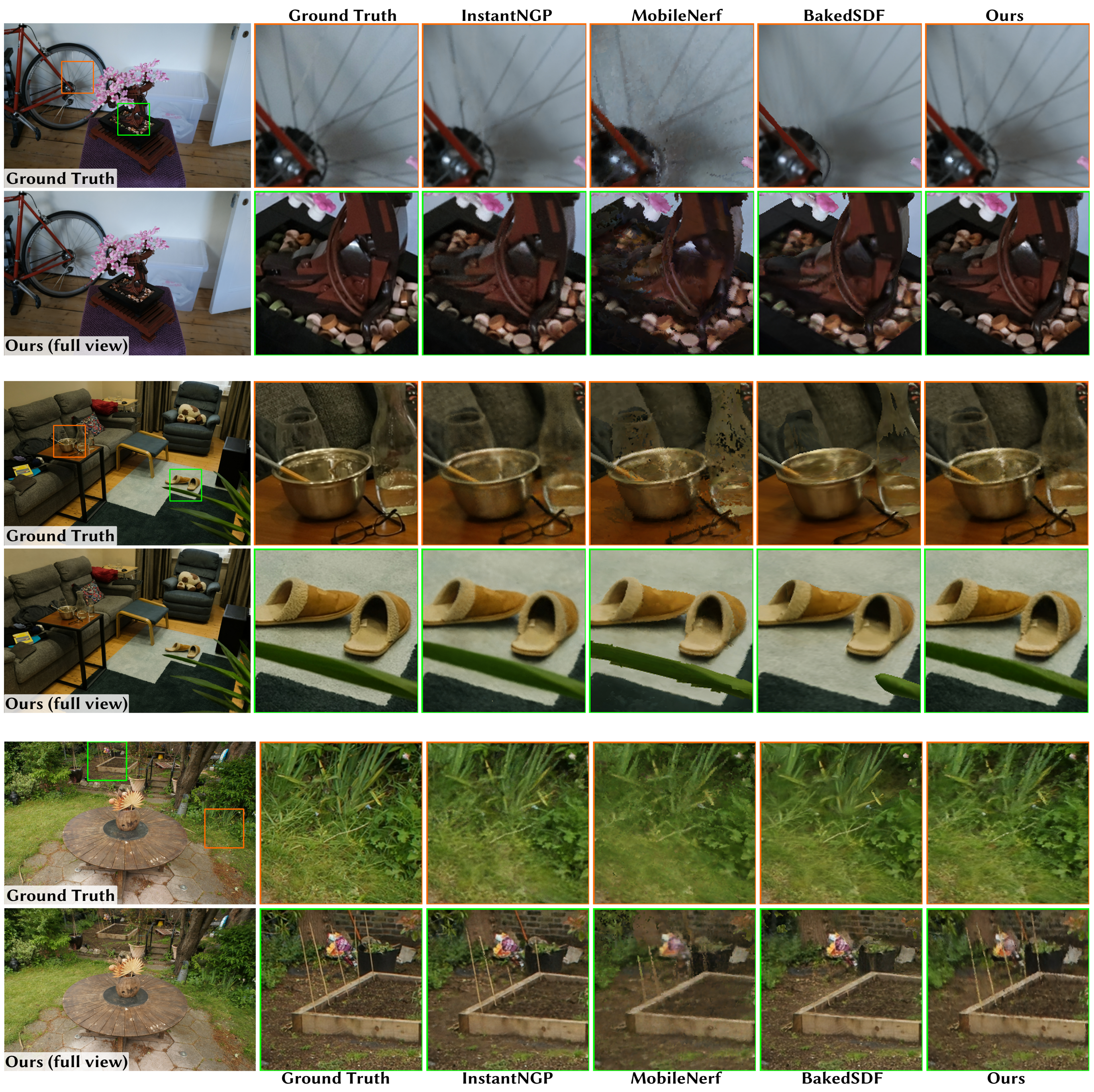
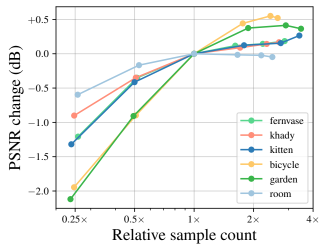


Fig. 10. A gallery of results on the test-views of the MipNerf360 data set.

scenes such as Kitten and Fernvase, smaller sample counts (0.25×-1×) do not further reduce the runtime due to a mixture of fixed overheads (e.g. Python interpreter and Dr.Jit tracing) and under-utilization of the GPU.

5 APPLICATIONS

Our method directly constructs an explicit outer shell mesh M_+ which by definition contains all regions of space that contribute to the rendered appearance. This property has great utility for use in downstream applications.



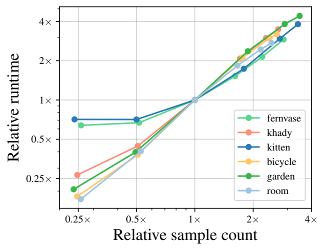


Fig. 11. Ablating the effect of sample count on image quality and runtime performance. We vary the sample count, and plot the PSNR change (left) and relative runtime performance (right) compared to the default hyper-parameters denoted as "1× sample count". We experiment with six scenes from the *Shelly* (fernvase, khady, kitten) and *MipNeRF360* (bicycle, garden, room) data sets.

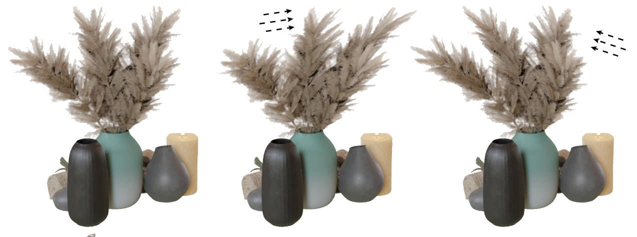
So far our scenes have represented entirely static content, yet, the world is full of motion. Cage-based deformation methods have shown promise for enabling dynamic, non-rigid motion in NeRF and other volumetric representations [Garbin et al. 2022; Joshi et al. 2007; Lee et al. 2018; Xu and Harada 2022; Yuan et al. 2022]. The basic idea is to construct a coarse tetrahedral cage around a neural volume, deform the cage, and use it to render the deformed appearance of the underlying volume. Our approach perfectly supports such techniques, as the outer shell mesh M_+ guides the construction of a cage which will surely contain the content.

We first dilate and tetrahedralize the outer mesh M_+ with Fast-TetWild [Hu et al. 2020] to produce a tetrahedral mesh that encapsulates the scene. This mesh acts as a proxy for performing physics simulations, animations, editing, and other operations. To render our representation after deforming the tetrahedral cage, any deformation is transferred to M_+ and M_- via barycentric interpolation, using precomputed barycentric coordinates generated as a preprocess. Ray directions are likewise transformed via finite differences. After the transformation, we proceed with rendering as usual in the reference space of our representation, as described in Section 3.4. Note that even in the presence of deformations, the rendering process still benefits from our efficient adaptive shell representation, and is able to efficiently sample the underlying neural volume.

We show two examples of applying physical simulation and animation to the reconstructed objects in Figure 12; please see the supplemental video for dynamic motion. In the animation example, we manually drive the motion of plants in a vase according to an analytical wind-like spatial function. Other animation schemes, such as blend shapes or character rigs could potentially be substituted to drive the motion. In the physical simulation example, we simulate the reconstructed asset via finite-element elastic simulation on the cage tetrahedra including collision penalties [Jatavallabhula et al. 2021].

6 DISCUSSION

Recent work has developed schemes to accelerate and improve the quality of NeRF-like scene representations. Section 4 provides comparisons to selected, particularly relevant methods. Note that due to the high research activity in the field, it is impossible to compare



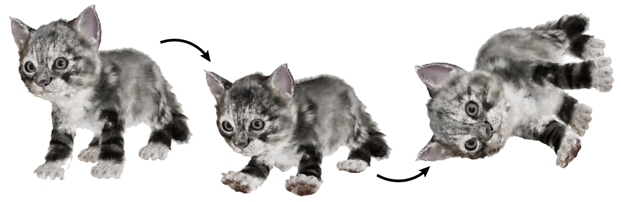


Fig. 12. Our representation is well-suited for animation (*top*) and physical simulation (bottom). The visual quality is preserved under deformation: the original shape is shown in leftmost column, with deformations in the middle and rightmost column. For details, please zoom into the fuzzy regions (e.g. fur, leaves), and refer to the supplemental video.

to all techniques and for many approaches implementations are not available. Hence, we offer additional comments on some related work:

- MobileNeRF [Chen et al. 2023], BakedSDF [Yariv et al. 2023], NeRFMeshing [Rakotosaona et al. 2023], and nerf2mesh [Tang et al. 2023] post-process NeRF-like models and extract meshes to accelerate inference, similar to this work. However, these approaches constrain appearance to surfaces, sacrificing quality. Our method instead retains a full volumetric representation and nearly full-NeRF quality, at the cost of moderately more expensive inference (though still real-time on modern hardware).
- DuplexRF [Wan et al. 2023] also extracts an explicit shell from the underlying neural field and uses it to accelerate rendering, although it does so with a very different neural representation, prioritizing performance. Their shell is directly extracted from two thresholds of the radiance field, which requires the careful selection of the thresholds and results in a noisy shell that is not adapted to the local complexity of the scene in contrast to our approach.
- VMesh [Guo et al. 2023] builds upon the similar insight that different parts of the scene require different treatment. However, their formulation assumes an additional voxel-grid data structure to mark the volumetric areas that contribute to the final rendering. This approach suffers from poor complexity scaling as with the auxiliary acceleration data structure of [Müller et al. 2022]. Instead, our method uses an explicit, adaptive shell to delimit the areas that contribute to the rendering. Apart from lower complexity, our formulation seamlessly enables further applications as discussed in Section 5.

7 CONCLUSION AND FUTURE WORK

In this work we focus on efficiently rendering NeRFs. Our first stage of training (Section 3.2) is largely similar to that of [Li et al. 2023], and likely can be accelerated by algorithmic advancements and low-level tuning similar to our inference pipeline [Wang et al. 2022a].

Although our method offers large speedups for high-fidelity neural rendering and runs at real-time rates on modern hardware (Table 1, it is still significantly more expensive than approaches such as MeRF [Reiser et al. 2023] that precompute the neural field outputs and bake them onto a discrete grid representation. Our formulation is complimentary to that of MeRF [Reiser et al. 2023] and we hypothesize that combining both approaches will lead to further speedups, potentially reaching the performance—at high quality—of the methods that bake the volumetric representation to explicit meshes and can run in real-time even on commodity hardware (*e.g.* [Chen et al. 2023]).

Our method does not guarantee to capture thin structures—if the extracted adaptive shell omits a geometric region, it can never be recovered during fine-tuning and will always be absent from the reconstruction. Artifacts of this form are visible in some *MipNeRF360* scenes. Future work will explore an iterative procedure, in which we alternately tune our reconstruction and adapt the shell to ensure that no significant geometry is missed. Other artifacts occasionally present in our reconstructions include spurious floating geometry and poorly-resolved backgrounds; both are common challenges in neural reconstructions and our approach may borrow solutions from other work across the field (*e.g.* [Niemeyer et al. 2022]).

More broadly, there is great potential in combining recent neural representations with high-performance techniques honed for real-time performance in computer graphics. Here, we have shown how ray tracing and adaptive shells can be used to greatly improve performance.

ACKNOWLEDGMENTS

The authors are grateful to Alex Evans for the insightful brainstorming sessions and his explorations in the early stages of this project. We also appreciate the feedback received from Jacob Munkberg, Jon Hasselgren, Chen-Hsuan Lin, and Wenzheng Chen during the project. Finally, we would like to thank Lior Yariv for providing the results of BakedSDF.

REFERENCES

Jonathan T. Barron, Ben Mildenhall, Matthew Tancik, Peter Hedman, Ricardo Martin-Brualla, and Pratul P. Srinivasan. 2021. Mip-NeRF: A Multiscale Representation for Anti-Aliasing Neural Radiance Fields. *ICCV* (2021).

Jonathan T. Barron, Ben Mildenhall, Dor Verbin, Pratul P. Srinivasan, and Peter Hedman. 2022. Mip-NeRF 360: Unbounded Anti-Aliased Neural Radiance Fields. *CVPR* (2022).

Sai Bi, Stephen Lombardi, Shunsuke Saito, Tomas Simon, Shih-En Wei, Kevyn Mcphail, Ravi Ramamoorthi, Yaser Sheikh, and Jason Saragih. 2021. Deep relightable appearance models for animatable faces. *ACM Transactions on Graphics (TOG)* 40, 4 (2021), 1–15.

Chris Buehler, Michael Bosse, Leonard McMillan, Steven Gortler, and Michael Cohen. 2001. Unstructured Lumigraph Rendering. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '01)*. Association for Computing Machinery, New York, NY, USA, 425–432.

Junli Cao, Huan Wang, Pavlo Chemerys, Vladislav Shakhrai, Ju Hu, Yun Fu, Denys Makoviichuk, Sergey Tulyakov, and Jian Ren. 2022. Real-Time Neural Light Field on Mobile Devices. *arXiv preprint arXiv:2212.08057* (2022).

Anpei Chen, Zexiang Xu, Andreas Geiger, Jingyi Yu, and Hao Su. 2022. TensoRF: Tensorial Radiance Fields. In *European Conference on Computer Vision (ECCV)*.

Zhiqin Chen, Thomas Funkhouser, Peter Hedman, and Andrea Tagliasacchi. 2023. MobileNeRF: Exploiting the Polygon Rasterization Pipeline for Efficient Neural Field Rendering on Mobile Architectures. In *The Conference on Computer Vision and Pattern Recognition (CVPR)*.

Abe Davis, Marc Levoy, and Fredo Durand. 2012. Unstructured Light Fields. *Comput. Graph. Forum* 31, 2pt1 (2012), 305–314.

Paul E. Debevec, Camillo J. Taylor, and Jitendra Malik. 1996. Modeling and Rendering Architecture from Photographs: A Hybrid Geometry- and Image-Based Approach. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '96)*. Association for Computing Machinery, 11–20.

Kangle Deng, Andrew Liu, Jun-Yan Zhu, and Deva Ramanan. 2022. Depth-supervised NeRF: Fewer Views and Faster Training for Free. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Stephan J Garbin, Marek Kowalski, Virginia Estellers, Stanislaw Szymanowicz, Shideh Rezaeifar, Jingjing Shen, Matthew Johnson, and Julien Valentin. 2022. VolTeMorph: Realtime, Controllable and Generalisable Animation of Volumetric Representations. *arXiv preprint arXiv:2208.00949* (2022).

Steven J. Gortler, Radek Grzeszczuk, Richard Szeliski, and Michael F. Cohen. 1996. The Lumigraph. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '96)*. Association for Computing Machinery, 43–54.

Yuan-Chen Guo, Yan-Pei Cao, Chen Wang, Yu He, Ying Shan, Xiaohu Qie, and Song-Hai Zhang. 2023. VMesh: Hybrid Volume-Mesh Representation for Efficient View Synthesis. *arXiv preprint arXiv:2303.16184* (2023).

Peter Hedman, Pratul P. Srinivasan, Ben Mildenhall, Jonathan T. Barron, and Paul Debevec. 2021. Baking Neural Radiance Fields for Real-Time View Synthesis. *ICCV* (2021).

Tao Hu, Shu Liu, Yilun Chen, Tiancheng Shen, and Jiaya Jia. 2022. EfficientNeRF Efficient Neural Radiance Fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 12902–12911.

Yixin Hu, Teseo Schneider, Bolun Wang, Denis Zorin, and Daniele Panozzo. 2020. Fast tetrahedral meshing in the wild. *ACM Transactions on Graphics (TOG)* 39, 4 (2020), 117–1.

Wenzel Jakob, Sébastien Speierer, Nicolas Roussel, and Delio Vicini. 2022. DR. JIT: a just-in-time compiler for differentiable rendering. *ACM Transactions on Graphics (TOG)* 41, 4 (2022), 1–19.

Krishna Murthy Jatavallabhula, Miles Macklin, Florian Golemo, Vikram Voleti, Linda Petrini, Martin Weiss, Breandan Considine, Jérôme Parent-Lévesque, Kevin Xie, Kenny Erleben, et al. 2021. gradsim: Differentiable simulation for system identification and visuomotor control. *arXiv preprint arXiv:2104.02646* (2021).

Rasmus Jensen, Anders Dahl, George Vogiatzis, Engil Tola, and Henrik Aanæs. 2014. Large scale multi-view stereopsis evaluation. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 406–413.

Pushkar Joshi, Mark Meyer, Tony DeRose, Brian Green, and Tom Sanocki. 2007. Harmonic Coordinates for Character Articulation. *ACM Trans. Graph.* 26, 3 (jul 2007), 71–es. https://doi.org/10.1145/1276377.1276466

Animesh Karnewar, Tobias Ritschel, Oliver Wang, and Niloy Mitra. 2022. ReLU Fields: The Little Non-Linearity That Could. In *ACM SIGGRAPH 2022 Conference Proceedings (SIGGRAPH '22)*. Association for Computing Machinery, New York, NY, USA, Article 27, 9 pages. https://doi.org/10.1145/3528233.3530707

Michael M. Kazhdan, Matthew Bolitho, and Hugues Hoppe. 2006. Poisson Surface Reconstruction. In *Proceedings of the Fourth Eurographics Symposium on Geometry Processing (SGP '06, Vol. 256)*. Eurographics Association, 61–70.

Michael M. Kazhdan and Hugues Hoppe. 2013. Screened poisson surface reconstruction. *ACM Trans. Graph.* 32, 3 (2013), 29:1–29:13.

Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 2023. 3D Gaussian Splatting for Real-Time Radiance Field Rendering. *ACM Transactions on Graphics* 42, 4 (July 2023). https://repo-sam.inria.fr/fungraph/3d-gaussian-splatting/

Georgios Kopanas, Julien Philip, Thomas Leimkühler, and George Drettakis. 2021. Point-Based Neural Rendering with Per-View Optimization. *Computer Graphics Forum (Proceedings of the Eurographics Symposium on Rendering)* 40, 4 (June 2021). http://www-sop.inria.fr/reves/Basilic/2021/KPLD21

Andreas Kurz, Thomas Neff, Zhaoyang Lv, Michael Zollhöfer, and Markus Steinberger. 2022. AdaNeRF: Adaptive Sampling for Real-time Rendering of Neural Radiance Fields. In *European Conference on Computer Vision (ECCV)*.

Christoph Lassner and Michael Zollhöfer. 2021. Pulsar: Efficient Sphere-based Neural Rendering. *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2021), 1440–1449.

Minjae Lee, David Hyde, Michael Bao, and Ronald Fedkiw. 2018. A skinned tetrahedral mesh for hair animation and hair-water interaction. *IEEE transactions on visualization and computer graphics* 25, 3 (2018), 1449–1459.

Marc Levoy and Pat Hanrahan. 1996. Light Field Rendering. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '96)*. Association for Computing Machinery, 31–42.

Max Zhaoshuo Li, Thomas Müller, Alex Evans, Russell H. Taylor, Mathias Unberath, Ming-Yu Liu, and Chen-Hsuan Lin. 2023. Neuralangelo: High-Fidelity Neural Surface Reconstruction. In *Conference on Computer Vision and Pattern Recognition (CVPR)*.

Haotong Lin, Sida Peng, Zhen Xu, Yunzhi Yan, Qing Shuai, Hujun Bao, and Xiaowei Zhou. 2022. Efficient Neural Radiance Fields with Learned Depth-Guided Sampling. In *SIGGRAPH Asia Conference Proceedings*.

Lingjie Liu, Jiatao Gu, Kyaw Zaw Lin, Tat-Seng Chua, and Christian Theobalt. 2020. Neural Sparse Voxel Fields. *NeurIPS* (2020).

Ricardo Martin-Brualla, Noha Radwan, Mehdi S. M. Sajjadi, Jonathan T. Barron, Alexey Dosovitskiy, and Daniel Duckworth. 2021. NeRF in the Wild: Neural Radiance Fields for Unconstrained Photo Collections. In *CVPR*.

Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. 2020. NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis. In *ECCV*.

Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. 2022. Instant Neural Graphics Primitives with a Multiresolution Hash Encoding. *ACM Trans. Graph.* 41, 4, Article 102 (July 2022), 15 pages. https://doi.org/10.1145/3528223.3530127

Jacob Munkberg, Jon Hasselgren, Tianchang Shen, Jun Gao, Wenzheng Chen, Alex Evans, Thomas Müller, and Sanja Fidler. 2022. Extracting Triangular 3D Models, Materials, and Lighting From Images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 8280–8290.

Thomas Neff, Pascal Stadlbauer, Mathias Parger, Andreas Kurz, Joerg H. Mueller, Chakravarty R. Alla Chaitanya, Anton S. Kaplanyan, and Markus Steinberger. 2021. DONeRF: Towards Real-Time Rendering of Compact Neural Radiance Fields using Depth Oracle Networks. *Computer Graphics Forum* 40, 4 (2021).

Michael Niemeyer, Jonathan T. Barron, Ben Mildenhall, Mehdi S. M. Sajjadi, Andreas Geiger, and Noha Radwan. 2022. RegNeRF: Regularizing Neural Radiance Fields for View Synthesis from Sparse Inputs. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*.

Michael Oechsle, Songyou Peng, and Andreas Geiger. 2021. UNISURF: Unifying Neural Implicit Surfaces and Radiance Fields for Multi-View Reconstruction. In *International Conference on Computer Vision (ICCV)*.

Steven G. Parker, James Bigler, Andreas Dietrich, Heiko Friedrich, Jared Hoberock, David Luebke, David McAllister, Morgan McGuire, Keith Morley, Austin Robison, and Martin Stich. 2010. OptiX: A General Purpose Ray Tracing Engine. *ACM Trans. Graph.* 29, 4, Article 66 (jul 2010), 13 pages. https://doi.org/10.1145/1778765.1778803

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in PyTorch. *NeurIPS Workshop on Autodiff* (2017).

Marie-Julie Rakotosaona, Fabian Manhardt, Diego Martin Arroyo, Michael Niemeyer, Abhijit Kundu, and Federico Tombari. 2023. NeRFMeshing: Distilling Neural Radiance Fields into Geometrically-Accurate 3D Meshes. *arXiv preprint arXiv:2303.09431* (2023).

Daniel Rebain, Wei Jiang, Soroosh Yazdani, Ke Li, Kwang Moo Yi, and Andrea Tagliasacchi. 2020. DeRF: Decomposed Radiance Fields. *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2020), 14148–14156.

Christian Reiser, Songyou Peng, Yiyi Liao, and Andreas Geiger. 2021. KiloNeRF: Speeding up Neural Radiance Fields with Thousands of Tiny MLPs. In *International Conference on Computer Vision (ICCV)*.

Christian Reiser, Richard Szeliski, Dor Verbin, Pratul P Srinivasan, Ben Mildenhall, Andreas Geiger, Jonathan T Barron, and Peter Hedman. 2023. Merf: Memory-efficient radiance fields for real-time view synthesis in unbounded scenes. *arXiv preprint arXiv:2302.12249* (2023).

Gernot Riegler and Vladlen Koltun. 2020. Free View Synthesis. In *European Conference on Computer Vision*.

Gernot Riegler and Vladlen Koltun. 2021. Stable View Synthesis. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

Radu Alexandru Rosu and Sven Behnke. 2023. PermutoSDF: Fast Multi-View Reconstruction with Implicit Surfaces using Permutohedral Lattices. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Vol. 2.

Darius Rückert, Linus Franke, and Marc Stamminger. 2021. Adop: Approximate differentiable one-pixel point rendering. *arXiv preprint arXiv:2110.06635* (2021).

Sara Fridovich-Keil and Alex Yu, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. 2022. Plenoxels: Radiance Fields without Neural Networks. In *CVPR*.

Johannes Lutz Schönberger and Jan-Michael Frahm. 2016. Structure-from-Motion Revisited. In *Conference on Computer Vision and Pattern Recognition (CVPR)*.

Johannes Lutz Schönberger, Enliang Zheng, Marc Pollefeys, and Jan-Michael Frahm. 2016. Pixelwise View Selection for Unstructured Multi-View Stereo. In *European Conference on Computer Vision (ECCV)*.

Cheng Sun, Min Sun, and Hwann-Tzong Chen. 2022. Direct Voxel Grid Optimization: Super-fast Convergence for Radiance Fields Reconstruction. In *CVPR*.

Matthew Tancik, Ethan Weber, Evonne Ng, Ruilong Li, Brent Yi, Justin Kerr, Terrance Wang, Alexander Kristoffersen, Jake Austin, Kamyar Salahi, Abhik Ahuja, David McAllister, and Angjoo Kanazawa. 2023. Nerfstudio: A Modular Framework for Neural Radiance Field Development. In *ACM SIGGRAPH 2023 Conference Proceedings (SIGGRAPH '23)*.

Jiaxiang Tang, Hang Zhou, Xiaokang Chen, Tianshu Hu, Errui Ding, Jingdong Wang, and Gang Zeng. 2023. Delicate textured mesh recovery from nerf via adaptive surface refinement. *arXiv preprint arXiv:2303.02091* (2023).

Michael Waechter, Nils Moehrle, and Michael Goesele. 2014. Let There Be Color! Large-Scale Texturing of 3D Reconstructions. In *Computer Vision – ECCV 2014* (Heidelberg) *(Lecture Notes in Computer Science, Vol. 8693)*, David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars (Eds.). Springer, 836–850. https://doi.org/10.1007/978-3-319-10602-1_54

Ziyu Wan, Christian Richardt, Aljaž Božič, Chao Li, Vijay Rengarajan, Seonghyeon Nam, Xiaoyu Xiang, Tuotuo Li, Bo Zhu, Rakesh Ranjan, et al. 2023. Learning Neural Duplex Radiance Fields for Real-Time View Synthesis. *arXiv preprint arXiv:2304.10537* (2023).

Peng Wang, Lingjie Liu, Yuan Liu, Christian Theobalt, Taku Komura, and Wenping Wang. 2021. NeuS: Learning Neural Implicit Surfaces by Volume Rendering for Multi-view Reconstruction. *NeurIPS* (2021).

Yiming Wang, Qin Han, Marc Habermann, Kostas Daniilidis, Christian Theobalt, and Lingjie Liu. 2022a. NeuS2: Fast Learning of Neural Implicit Surfaces for Multi-view Reconstruction.

Yiqun Wang, Ivan Skorokhodov, and Peter Wonka. 2022b. HF-NeuS: Improved Surface Reconstruction Using High-Frequency Details. *arXiv preprint arXiv:2206.07850* (2022).

Zian Wang, Tianchang Shen, Jun Gao, Shengyu Huang, Jacob Munkberg, Jon Hasselgren, Zan Gojcic, Wenzheng Chen, and Sanja Fidler. 2023. Neural Fields meet Explicit Geometric Representations for Inverse Rendering of Urban Scenes. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Daniel N. Wood, Daniel I. Azuma, Ken Aldinger, Brian Curless, Tom Duchamp, David H. Salesin, and Werner Stuetzle. 2000. Surface Light Fields for 3D Photography. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '00)*. ACM Press/Addison-Wesley Publishing Co., 287–296.

Tianhan Xu and Tatsuya Harada. 2022. Deforming radiance fields with cages. In *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXXIII*. Springer, 159–175.

Lior Yariv, Jiatao Gu, Yoni Kasten, and Yaron Lipman. 2021. Volume rendering of neural implicit surfaces. In *Thirty-Fifth Conference on Neural Information Processing Systems*.

Lior Yariv, Peter Hedman, Christian Reiser, Dor Verbin, Pratul P Srinivasan, Richard Szeliski, Jonathan T Barron, and Ben Mildenhall. 2023. BakedSDF: Meshing Neural SDFs for Real-Time View Synthesis. *arXiv preprint arXiv:2302.14859* (2023).

Lior Yariv, Yoni Kasten, Dror Moran, Meirav Galun, Matan Atzmon, Basri Ronen, and Yaron Lipman. 2020. Multiview Neural Surface Reconstruction by Disentangling Geometry and Appearance. *Advances in Neural Information Processing Systems* 33 (2020).

Alex Yu, Ruilong Li, Matthew Tancik, Hao Li, Ren Ng, and Angjoo Kanazawa. 2021. PlenOctrees for Real-time Rendering of Neural Radiance Fields. In *ICCV*.

Yu-Jie Yuan, Yang-Tian Sun, Yu-Kun Lai, Yuewen Ma, Rongfei Jia, and Lin Gao. 2022. NeRF-editing: geometry editing of neural radiance fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 18353–18364.

Kai Zhang, Fujun Luan, Qianqian Wang, Kavita Bala, and Noah Snavely. 2021. PhySG: Inverse Rendering with Spherical Gaussians for Physics-based Material Editing and Relighting. In *The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Kai Zhang, Gernot Riegler, Noah Snavely, and Vladlen Koltun. 2020. NeRF++: Analyzing and Improving Neural Radiance Fields. *arXiv:2010.07492* (2020).

Fuqiang Zhao, Yuheng Jiang, Kaixin Yao, Jiakai Zhang, Liao Wang, Haizhao Dai, Yuhui Zhong, Yingliang Zhang, Minye Wu, Lan Xu, and Jingyi Yu. 2022. Human Performance Modeling and Rendering via Neural Animated Mesh. *ACM Trans. Graph.* 41, 6, Article 235 (2022), 17 pages.

APPENDIX

We provide additional algorithmic details and pseudocode. In the first phase of training, our method adaptively extracts an explicit mesh envelope which spatially bounds the neural volumetric representation: level set evolution and shell extraction are shown in Procedures 1 and 2. In the second phase of training, as well as inference, we leverage the extracted shells to sample query points only where they are needed. We cast rays against the shell meshes and compute query locations in the narrow band between the outer and inner shell. This is detailed in Procedure 3. Note that one ray may intersect with multiple narrow bands, however it always terminates when encountering an inner shell. Finally, we include the overall training pipeline in Procedure 4.

Procedure 1 LevelSetEvolution($f, v_{\text{evolve}}, T, \Delta t, \zeta, \lambda_{\text{curv}}$)

Input: level set field $f \in \mathbb{R}^{X\times Y\times Z}$, velocity $v_{\text{evolve}} \in \mathbb{R}^{X\times Y\times Z}$, evolution steps T, timestep Δt, soft falloff threshold ζ, curvature regularization weight λ_{curv}

Output: Evolved level set field f_T

```
1: $f_0 \leftarrow f$                                                    ▷Initialize level set field
2: for $i \leftarrow 0$ to $T-1$ do
3:     $\left[\frac{\partial f_i}{\partial t}\right]_{\text{mot}} \leftarrow -|\nabla f_i|\, v_{\text{evolve}}$                    ▷Motion term
4:     $\left[\frac{\partial f_i}{\partial t}\right]_{\text{curv}} \leftarrow -\lambda_{\text{curv}}|\nabla f_i|\left(\nabla \cdot \frac{\nabla f}{|\nabla f|}\right)$      ▷Curvature term
5:     $\omega_i \leftarrow \frac{1}{2}\left(1+\cos(\pi\ \text{clamp}(f_i/\zeta, -1., 1.)\right)$   ▷Soft falloff (Eq. 5)
6:     $\frac{\partial f_i}{\partial t} \leftarrow \omega_i \left(\left[\frac{\partial f_i}{\partial t}\right]_{\text{mot}} + \left[\frac{\partial f_i}{\partial t}\right]_{\text{curv}}\right)$
7:     $f_{i+1} = f_i + \Delta t \frac{\partial f_i}{\partial t}$                           ▷Update level set field
8: Return $f_T$
```

Procedure 2 ShellExtraction($f, s, \tau_d, \beta_d, \sigma_{\min}, \beta_e, v_{\max}$)

Input: signed distance field $f \in \mathbb{R}^{X\times Y\times Z}$, spatially-varying kernel size $s \in \mathbb{R}^{X\times Y\times Z}$, grid size τ_d, dilation hyperparameters $\beta_d, \sigma_{\min}$, erosion hyperparameters $\beta_e, v_{\max}$

Output: outer mesh M_+ and inner mesh M_-

```
1: $\alpha \leftarrow \frac{\text{Sigmoid}\left((f-\tau_d/2)/s\right)-\text{Sigmoid}\left((f+\tau_d/2)/s\right)}{\text{Sigmoid}\left((f-\tau_d/2)/s\right)}$   ▷Eq. 3, as in NeuS
2:
3: ▷Level set dilation for outer mesh
4: $v_{\text{dilate}} \leftarrow \text{Where}(\alpha > \sigma_{\min}, \beta_d\alpha, 0)$                   ▷Eq. 6
5: $f_{\text{dilate}} \leftarrow \text{LevelSetEvolution}(f, v_{\text{dilate}}, T=50, \Delta t=0.1,$
6:                                          $\zeta = 0.1, \lambda_{\text{curv}} = 0.01)$
7: $f_{\text{dilate}} \leftarrow \min(f_0, f_{\text{dilate}})$                ▷Clip SDF for a strict dilation
8: $M_+ \leftarrow \text{MarchingCubes}(f_{\text{dilate}})$
9:
10: ▷Level set erosion for inner mesh
11: $v_{\text{erode}} \leftarrow \min(v_{\max}, \beta_e/\alpha)$                              ▷Eq. 7
12: $f_{\text{erode}} \leftarrow \text{LevelSetEvolution}(f, v_{\text{erode}}, T=50, \Delta t=0.1,$
13:                                          $\zeta = 0.05, \lambda_{\text{curv}} = 0)$
14: $f_{\text{erode}} \leftarrow \max(f_0, f_{\text{erode}})$                 ▷Clip SDF for a strict erosion
15: $M_- \leftarrow \text{MarchingCubes}(f_{\text{erode}})$
16:
17: Return $M_+, M_-$
```

Procedure 3 NarrowBandSampling($M_+, M_-, r, o, w_s, \delta_s, N_{max}, dp_{max}$)

Input: outer mesh M_+ and inner mesh M_-, ray origin $\mathbf{o} \in \mathbb{R}^3$ and direction $\mathbf{r} \in \mathbb{R}^3$, target inter-sample spacing δ_s, single-sample threshold w_s, maximum number of samples per interval N_{max}, and a maximum cap for depth peeling dp_{max}

Output: a list of distances τ_s to sampled points along the ray

```
1: $\tau_s \leftarrow$ empty list, $n_{hits} \leftarrow 0$                        ▷Initialize Steps
2: ▷Find hits of the inner mesh
3: $rayHits_{M_-} \leftarrow \text{CastRay}(M_-, \mathbf{o}, \mathbf{r})$
4: if HasNext($rayHits_{M_-}$) then
5:     $(d_{M_-}, flag) \leftarrow \text{GetNextHit}(rayHits_{M_-})$
6: else
7:     $d_{M_-} \leftarrow \infty$
8: ▷Loop through the hits of the outer mesh
9: $rayHits_{M_+} \leftarrow \text{CastRay}(M_+, \mathbf{o}, \mathbf{r})$
10: while HasNext($rayHits_{M_+}$) AND $n_{hits} < dp_{max}$ do
11:     $(hitDistance, flag) \leftarrow \text{GetNextHit}(rayHits_{M_+})$
12:     $n_{hits} \leftarrow n_{hits} + 1$
13:     if $flag$ = ENTERING then                      ▷Ray enters the mesh band
14:         $d_{enter} \leftarrow hitDistance$
15:     else if $flag$ = EXITING then                    ▷Ray exits the mesh band
16:         ▷Compute samples between enter and exit
17:         $d_{exit} \leftarrow \text{Min}(hitDistance, d_{M_-})$
18:         $w \leftarrow d_{exit} - d_{enter}$
19:         $N \leftarrow \text{Min}(\text{Ceil}(\text{Max}(w - w_s, 0)/\delta_s)+1, N_{max})$
20:         $\tau_s \leftarrow \tau_s + \text{Linspace}(d_{enter}, d_{exit}, N+2)[1:-1]$
21:         if $hitDistance > d_{M_-}$ then
22:             break                      ▷Terminate if beyond the inner mesh
23: end while
24: Return $\tau_s$
```

Procedure 4 Training Pipeline

Input: rays $\mathcal{R}$, ground-truth pixel colors C, training iterations for the first stage N_1 and second stage N_2, network NN_θ

Output: optimized network parameters θ, shell M_+, M_-

```
    ▷First stage training with full ray volume rendering
1: for $i \leftarrow 0$ to $N_1 - 1$ do
2:     Sample data $\mathbf{r}_i \in \mathcal{R}, \mathbf{c}_i \in C$
3:     $output \leftarrow \text{VolumeRendering}(\text{NN}_\theta, \mathbf{r}_i)$
4:     $\mathcal{L} \leftarrow \text{Loss}(output, \mathbf{c}_i)$                    ▷Compute loss with Eq. 8
5:     Update network: $\theta \leftarrow \theta - \eta \frac{\partial \mathcal{L}}{\partial \theta}$
6: ▷Extract adaptive shells
7: $M_+, M_- \leftarrow \text{ShellExtraction}(\text{NN}_\theta)$                         ▷Section 3.3
8: ▷Second stage training with narrow-band rendering
9: for $i \leftarrow 0$ to $N_2 - 1$ do
10:     Sample data $\mathbf{r}_i \in \mathcal{R}, \mathbf{c}_i \in C$
11:     $output \leftarrow \text{NarrowBandRendering}(\text{NN}_\theta, \mathbf{r}_i, M_+, M_-)$
12:     $\mathcal{L}_c \leftarrow \text{ColorLoss}(output, \mathbf{c}_i)$            ▷Compute loss with Eq. 9
13:     Update network: $\theta \leftarrow \theta - \eta \frac{\partial \mathcal{L}_c}{\partial \theta}$
14: Return $\theta, M_+, M_-$
```