Binary Opacity Grids: Capturing Fine Geometric Detail for Mesh-Based View Synthesis

CHRISTIAN REISER, University of Tübingen, Tübingen AI Center, Google Research, Germany
STEPHAN GARBIN, Google Research, United Kingdom
PRATUL P. SRINIVASAN, Google Research, United States of America
DOR VERBIN, Google Research, United States of America
RICHARD SZELISKI, Google Research, United States of America
BEN MILDENHALL, Google Research, United States of America
JONATHAN T. BARRON, Google Research, United States of America
PETER HEDMAN*, Google Research, United Kingdom
ANDREAS GEIGER*, University of Tübingen, Tübingen AI Center, Germany

Fig. 1. Our method reconstructs triangle meshes from multi-view images and is able to capture fine geometric detail such as leaves, branches and grass (left). At the same time our meshes are compact enough for real-time view synthesis on a Google Pixel 8 Pro (right).

While surface-based view synthesis algorithms are appealing due to their low computational requirements, they often struggle to reproduce thin structures. In contrast, more expensive methods that model the scene's geometry as a volumetric density field (e.g. NeRF) excel at reconstructing fine geometric detail. However, density fields often represent geometry in a "fuzzy" manner, which hinders exact localization of the surface. In this work, we modify density fields to encourage them to converge towards surfaces, without compromising their ability to reconstruct thin structures. First, we employ a discrete opacity grid representation instead of a continuous density field, which allows opacity values to discontinuously transition from zero to one at the surface. Second, we anti-alias by casting multiple rays per pixel, which allows occlusion boundaries and subpixel structures to be modelled without using semi-transparent voxels. Third, we minimize the binary entropy of the opacity values, which facilitates the extraction of surface geometry by encouraging opacity values to binarize towards the end of training. Lastly, we develop a fusion-based meshing strategy followed by mesh simplification and appearance model fitting. The compact meshes produced by our model can be rendered in real-time on mobile devices and achieve significantly higher view synthesis quality compared to existing mesh-based approaches. Our interactive webdemo is available at https://binary-opacity-grid.github.io.

CCS Concepts: • **Computing methodologies** → **Reconstruction**; *Appearance and texture representations*; Rasterization.

Additional Key Words and Phrases: Novel View Synthesis, Differentiable Rendering, Neural Radiance Fields, Multiview-to-3D, Real-Time Rendering

*Equal advising.

Authors' addresses: Christian Reiser, University of Tübingen, Tübingen AI Center, Google Research, Germany, Tübingen, christian.j.reiser@gmail.com; Stephan Garbin, Google Research, United Kingdom, London, stephangarbin@outlook.com; Pratul P. Srinivasan, Google Research, United States of America, San Francisco, pratuls@google.com; Dor Verbin, Google Research, United States of America, Boston, dorverbin@google.com; Richard Szeliski, Google Research, United States of America, Seattle, szeliski@google.com; Ben Mildenhall, Google Research, United States of America, San Francisco, bmild@google.com; Jonathan T. Barron, Google Research, United States of America, San Francisco, barron@google.com; Peter Hedman, Google Research, United Kingdom, London, hedman@google.com; Andreas Geiger, University of Tübingen, Tübingen AI Center, Germany, Tübingen, a.geiger@uni-tuebingen.de.

1 INTRODUCTION

Surface rendering is generally considered to be more efficient than volume rendering, as surface rendering ideally only requires reading appearance data from a single 3D location, while volume rendering requires aggregating colors and densities across multiple points along each ray. Nevertheless, the current highest-quality view synthesis algorithms [Barron et al. 2023; Duckworth et al. 2023; Kerbl et al. 2023] all use volume rendering. These algorithms tend to represent even hard surfaces as "fuzzy" volumes, which leads to their high computational cost. This also holds when applying surface-promoting regularizers [Barron et al. 2022], see Figure 2.

Recently, BakedSDF [Yariv et al. 2023] has demonstrated that accurate view synthesis is also possible with a surface-based approach. However, in contrast to volumetric methods, BakedSDF struggles with recovering fine geometric detail. One reason for this is that BakedSDF adopts the currently dominant paradigm for 3D reconstruction, where the SDF is converted to a fuzzy volume during training [Li et al. 2023; Wang et al. 2021; Yariv et al. 2021; Yu et al. 2022a,b]. This soft conversion from SDF to volumetric density allows the model to "cheat" by representing thin structures in a fuzzy manner. As a result, during meshing, thin structures often vanish. Furthermore, during training, the validity of the recovered SDF must be ensured using an Eikonal loss, which acts as a smoothness prior and thereby tends to remove fine geometric detail.

To avoid these weaknesses of SDF-based approches, we investigate an alternative strategy that does not require an Eikonal loss or soft density conversion. We use a volume-based representation whose geometry we successively "sharpen" during training. We achieve this surface convergence by applying the following three modifications to an existing state-of-the-art radiance field model [Barron et al. 2023]. First, we employ a discrete opacity grid instead of a continuous density field, which enables opacity values to discontinuously transition from zero to one at the surface [Chen et al. 2023]. Second, we cast multiple rays per pixel to allow our model to accurately reproduce anti-aliased occlusion boundaries without using semi-transparent voxels. Third, we explicitly encourage hard surfaces by enforcing a binary entropy loss on the opacity values. As shown in Figure 2, this causes opacity values to binarize to zero or one as training converges, which enables the extraction of surface geometry. We demonstrate that all three of these elements are required for accurate reconstruction of subpixel structures.

Furthermore, we present a fusion-based meshing strategy for converting our recovered binary opacity grid (BOG) into a triangle mesh after training. The resulting mesh can then be simplified with off-the-shelf tools to a complexity that is adequate for real-time rendering while still preserving thin structures. Finally, we equip that mesh with a lightweight view-dependent appearance model that is well-suited for real-time viewer applications. Because the standard approach of UV mapping is problematic for our highly detailed meshes, we systematically evaluate alternative appearance representations and find the combination of triplanes with a low-resolution voxel grid as the preferred method [Reiser et al. 2023]. Our triangle mesh and appearance representation are compact enough to be rendered in real-time on mobile devices and achieve significantly higher view synthesis quality compared to existing mesh-based models. As such, our work represents a step towards closing the gap between surface-based view synthesis methods and volume-based ones.

2 RELATED WORK

Here we review real-time view synthesis methods that fit a 3D scene representation to calibrated multi-view images with differentiable rendering. These can be subdivided according to their rendering formulation into volume-based, surface-based and hybrid methods.
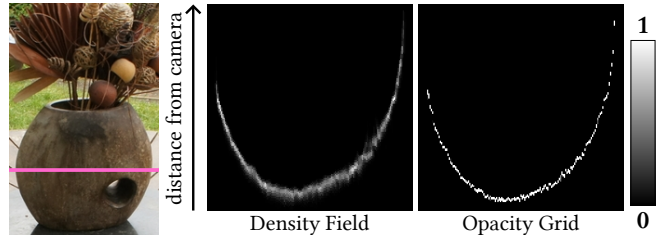

Fig. 2. Volume rendering weights for rays along a row of pixels (left, in pink). Density fields like Zip-NeRF [Barron et al. 2023] tend to represent hard surfaces as semi-transparent volumes despite using surface-promoting regularizers. In contrast, our opacity grid converges to a hard surface. Note that, as each column visualizes volume rendering weights of a single ray, gaps in this visualization do not indicate holes in the underlying representation.

Volume-based Methods. The highest-quality view synthesis methods use volume rendering during optimization and inference [Barron et al. 2023]. Many techniques in this category follow the neural radiance fields (NeRFs) paradigm [Mildenhall et al. 2020]. NeRFs associate a density and view-dependent color value with each 3D point and can be fitted using differentiable volume rendering to multi-view images with a re-rendering objective. The original NeRF uses an MLP to represent the scene, which results in slow rendering. Follow-up works speed-up rendering by using alternative representations such as voxel grids [Garbin et al. 2021; Hedman et al. 2021; Yan et al. 2023; Yu et al. 2021], triplanes [Chen et al. 2022; Duckworth et al. 2023; Reiser et al. 2023], or point-based representations [Kerbl et al. 2023; Kopanas et al. 2021; Rückert et al. 2022a,b; Xu et al. 2022].

Surface-based Methods. Although recent volume-based methods such as SMERF or 3DGS are capable of real-time rendering, they are slower than surface-based alternatives such as BakedSDF, as demonstrated in Duckworth et al. [2023]. In volume rendering, computing a pixel's value requires compositing colors from *multiple* sampling locations (SMERF) or primitives (3DGS), while surface rendering (under typical conditions) only requires reading appearance data from a *single* surface location. This is the case for surface-based view synthesis methods that employ a triangle mesh, which can be efficiently rendered using hardware-accelerated rasterization.

Early view-synthesis methods use surface geometry from multi-view stereo [Jancosek and Pajdla 2011; Schönberger et al. 2016] and model appearance by blending between input images [Debevec et al. 1998; Waechter et al. 2014; Wood et al. 2000]. Later methods improve visual quality by predicting the appearance of the surface with a trained neural network [Chen et al. 2018; Philip et al. 2021; Riegler and Koltun 2021; Thies et al. 2019]. Most of these approaches are constrained by the quality of the reconstructed surface, as they did not jointly optimize for appearance and geometry. For objects whose topology is close to an initial template mesh Worchel et al. [2022] optimize a triangle mesh together with appearance.

MobileNeRF is similar to our method in that it also uses an opacity-based representation during training [Chen et al. 2023]. The key difference is that MobileNeRF outputs a coarse proxy mesh equipped with binary alpha masks, whereas we aim for a traditional mesh,

which enables wider compatibility. To this end, we use a significantly higher voxel grid resolution than MobileNeRF and rely on simplification to obtain a compact mesh. Our fine grid is more geometrically expressive than MobileNeRF's alpha-textured coarse mesh, which is unable to represent multiple close-by sheets of geometry. During training, we achieve surface convergence using a combination of supersampling and entropy regularization, while MobileNeRF differentiably quantizes opacity values.

UNISURF also uses an opacity-based representation [Oechsle et al. 2021]. In addition to volume rendering, UNISURF uses a second rendering formulation where the 0.5 level set defines the surface [Niemeyer et al. 2020; Yariv et al. 2020]. We instead obtain hard surfaces by regularizing opacity values.

BakedSDF optimizes a signed distance function (SDF) that can be converted into a mesh after training and encodes appearance as vertex attributes [Yariv et al. 2023]. Similar to VolSDF [Yariv et al. 2021], NeuS [Wang et al. 2021, 2023a], or NeuralAngelo [Li et al. 2023], BakedSDF converts signed distances to density values during optimization, and those densities are used for volume rendering. Valid SDFs are encouraged through the use of a loss to enforce the Eikonal constraint, but this constraint is sometimes violated in favor of reconstructing fine geometric detail in a fuzzy manner. As a result, thin structures often vanish when "baking" these SDFs into meshes. In contrast, in our method, there is high agreement between optimized and extracted geometry, since opacity values mostly become binary towards the end of the training.

A number of recent papers also focus on extraction of fine geometric detail from volumetric density fields. NeRFMeshing and NeRF2Mesh both convert a density field into a triangle mesh [Rakotosaona et al. 2023; Tang et al. 2023]. Since density fields do not have a clearly defined surface, these methods compensate for lossy mesh conversion with an additional optimization stage. LoD-NeuS uses an error-guided SDF growth strategy to featurize conical frusta along each ray [Zhuang et al. 2023].

DMTet and FlexiCubes differentiably convert an implicit representation to a triangle mesh during training. Similar to our method, any mismatch between optimized and baked geometry is avoided, but these methods do not scale to high resolutions because they require that the full grid be processed during each forward pass [Liao et al. 2018; Munkberg et al. 2022; Shen et al. 2021, 2023].

Several concurrent preprints extract meshes from 3DGS [Guédon and Lepetit 2023; Huang et al. 2024; Lyu et al. 2024; Wolf et al. 2024; Yu et al. 2024a,b], but do not reason about subpixel structures and do not evaluate baked meshes for view synthesis.

Hybrid Methods. Recently, some methods have emerged that use a combination of surface and volume rendering during inference [Guo et al. 2023; Turki et al. 2023; Wang et al. 2023b]. These models aim to model the majority of the scene as surface geometry, while modeling whatever small subsets of the scene that happen to look "fuzzy" as volumes. Our goal is to expand the portion of the scene that can be represented as a surface, aiming to use volume rendering as sparingly as possible to ensure optimal performance.

3 BINARY OPACITY GRIDS

To capture thin structures with a surface-based representation, our model first uses an opacity-based voxel grid representation. Through the use of an entropy regularizer and supersampling, our opacity values become binary (either zero or one) towards the end of training. This enables us to exactly locate the surface, which is essential for the conversion of our recovered model into a triangle mesh.

3.1 Representation

During training, we represent the scene with an $R \times R \times R$ voxel grid, using a 3D contraction function, as described in Appendix C. With each voxel, we associate an opacity value $\alpha \in [0, 1]$ and a color value $\mathbf{c} \in [0, 1]^3$, which additionally depends on the view direction. To render a pixel, we cast a ray from the camera origin through the center of the pixel, and this ray is then intersected with all of the voxels along its path. For each intersected voxel, we query its opacity value α_k and its color value $\mathbf{c}_k$. The final pixel value $\mathbf{C}$ is computed using front-to-back alpha compositing:

$$\mathbf{C} = \sum_k \alpha_k \left(\prod_{j=1}^{k-1} \alpha_j \right) \mathbf{c}_k . \tag{1}$$

Following MobileNeRF, we directly parameterize opacity values in $[0, 1]$, unlike NeRF, which parameterizes density values that are later converted to opacity values using the distance between sampling points [Mildenhall et al. 2020]. In contrast to density-based volume rendering, our formulation does not involve any approximation since it is a finite sum over values associated with the voxels along the ray. One advantage of our formulation is that, when all opacity values are binary, the surface must be located at the first voxel along the ray with an opacity value of one [Chen et al. 2023].

To represent thin structures, we require a high voxel grid resolution R on the order of 2^{13}. Directly optimizing a voxel grid of this size is not feasible, as this would require > 2 terabytes of memory to store opacity values alone. Instead, we predict the grid values using an MLP equipped with a multi-resolution hash encoding as in Müller et al. [2022]. Note that our overall representation is still discrete in nature because the MLP is only queried at quantized positions [Reiser et al. 2023].

The number of voxels that intersect a ray is proportional to the grid resolution R. Therefore, with a high resolution, it becomes computationally intractable to query the representation at all intersected voxels. To address this, prior work adopted a coarse-to-fine strategy in combination with empty space skipping, but it has been observed that this can cause thin structures to be lost during early training iterations [Liu et al. 2020; Müller et al. 2022]. For standard density-based NeRFs, this issue can be circumvented with hierarchical sampling using a "proposal" MLP [Barron et al. 2022; Mildenhall et al. 2020]. However, this strategy relies on the assumption that, at the beginning of training, the volume rendering integral can be well-approximated with randomly placed samples due to the initial volume being somewhat smooth. Since opacity-based rendering does not incorporate the distance between sample points, the finite sum in Equation (1) can only be poorly estimated with a small number of randomly placed samples. To circumvent this, we first train a Zip-NeRF to produce a converged proposal MLP, which encodes the

coarse geometry of the scene [Barron et al. 2023]. When training our model, we query our representation only at a fixed number of samples from the distribution predicted by the pre-trained proposal MLP, whose weights are kept fixed. These samples represent a superset of the actual surface locations, which entails that the finite sum in Equation (1) is computed accurately. We quantize the sampled positions by rounding down each coordinate to the nearest multiple of the voxel size. If more than one sampled position falls within the same voxel, only the first position is used, which ensures that each voxel only contributes at most once.

3.2 Training strategy

Locating a surface and extracting a triangle mesh from an opacity grid requires binary opacity values, but optimizing the opacity values of our grid with no additional regularization does not naturally result in binarized values at the end of training. To encourage binary opacity values, we use an entropy loss that pulls opacity values smaller than 0.5 towards 0 and opacity values larger than 0.5 towards 1 [Lombardi et al. 2019; Xu et al. 2022]. We apply this loss to the opacity values α_k of all voxels sampled along each ray:

$$\mathcal{L}_{\text{ent}} = \frac{1}{k} \sum_k \text{H}(\alpha_k), \tag{2}$$

where H is the binary entropy function:

$$\text{H}(p) = -p \log_2(p) - (1-p) \log_2(1-p). \tag{3}$$

This alone, however, is not sufficient to accurately reconstruct fine geometric detail. This is because, in a properly anti-aliased image (such as the photographs we use as inputs), each pixel's value is the integral of all light within the cone associated with that pixel.

Consider the case of a "mixed pixel" at an occlusion boundary, where a pixel's value depends on light emitted from both a foreground object and a background object. In volumetric methods such as NeRF or 3DGS, such a pixel will be modeled by reconstructing a semi-transparent region of the foreground object, such that the ray being cast partially penetrates it and proceeds to the background object. This correctly yields a reconstructed pixel value that contains contributions from both the foreground and background objects. But this use of semi-transparency violates the binary entropy assumption required by our model: if opacity values are all binary, casting a single ray through the center of a pixel will result in *either* the foreground or the background object being struck, and will therefore yield an incorrect and aliased pixel intensity (i.e., "jaggies"). It is therefore infeasible to accurately reconstruct these mixed pixels using binary opacity values, assuming a single ray is cast for each pixel. To correctly disambiguate the contributions from multiple surfaces, we therefore cast *multiple* rays per pixel during training. More specifically, we uniformly sample 16 sub-rays within the footprint of each pixel. After rendering each sub-ray, the final pixel value is computed as the arithmetic mean of the subpixel values. We observe that supersampling produces a significant improvement in geometric quality, especially regarding the reconstruction of thin structures, which often cover less than a single pixel.
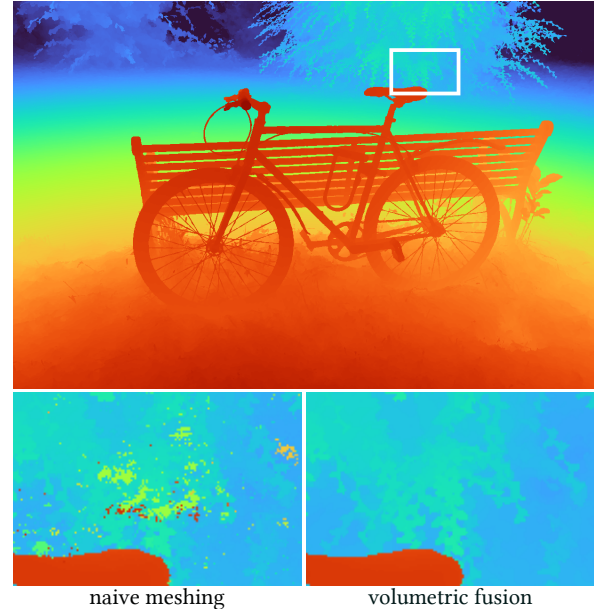

naive meshing   volumetric fusion

Fig. 3. **Different meshing strategies.** The bottom left image shows a depth map rendered from a mesh that was obtained by applying the meshing strategy from Yariv et al. [2023] to our representation. Geometry is instantiated at all visible voxels with an opacity value of 1 that are sampled by the proposal MLP in *any* training view. This leads to numerous floating artifacts, as infrequently sampled voxels in free space are severely underconstrained by the training loss. The bottom right shows that these underconstrained voxels can be effectively filtered by running volumetric fusion on depth maps rendered from our model. This filtering step also fully preserves thin structures, as can be seen in the top image.

4 MESH CONVERSION

After optimization, we convert the recovered BOG into a triangular mesh — the most ubiquitous and practical representation for geometry in computer graphics. If done naïvely, this conversion leads to a mesh consisting of billions of tiny cubes, which is prohibitively large for real-time rendering. To mitigate this, we design a simple and scalable baking pipeline that outputs a mesh that can be simplified using off-the-shelf tools.

4.1 Volumetric Fusion for Outlier Removal

The most basic strategy for converting our binary occupancy grid representation into a triangle mesh is to simply instantiate a surface quad between every pair of voxels with opposing opacity values. This works poorly, because the opacity values of voxels in free space are completely unconstrained, as these voxels are never sampled during training. Similarly, occluded space is not constrained by the re-rendering objective, which leads to arbitrary opacity values in the interior of objects. Therefore this strategy results in the creation of many random surfaces, which are either distracting floating artifacts in front of objects or invisible but computationally-wasteful pseudo-geometry in the interior of objects.

A better strategy is to incorporate the proposal MLP that encodes which opacity values are constrained by the training objective. Prior

work does this by rendering all training views using the proposal MLP [Reiser et al. 2023; Yariv et al. 2023] and then only instantiating surfaces in the vicinity of voxels that contribute to rendering of any pixel. This leads to the filtering of unconstrained areas, since only parts of the scene that are not occluded and sampled by the proposal MLP and thus receive supervision are considered for meshing. However, in our model, this strategy still produces a significant number of floating artifacts, as can be seen in the bottom left image of Figure 3. This is because some voxels are severely underconstrained, as they are not consistently sampled by the proposal MLP during training. In other words, during training, some voxels are only sampled in a fraction of the training views that observe a voxel. These underconstrained voxels may get erroneously assigned an opacity value of 1 despite being far from any surface. Since these voxels are still sampled in some of the training views, they are incorrectly appended to the final mesh. To filter these false positives, we employ volumetric fusion [Curless and Levoy 1996] as described in Appendix A. The bottom right image of Figure 3 demonstrates the effectiveness of volumetric fusion for removing outliers.

Another important motivation for using volumetric fusion is that it outputs a dense implicit representation of the scene. As shown in Curless and Levoy [1996], this implicit representation can be converted into a hole-free mesh, which is the preferred input for most mesh simplification algorithms. Before conversion to a mesh with marching cubes, we filter the implicit representation with a small Gaussian blur with $\sigma = 1$ to remove geometric noise in the underconstrained outer parts of the scene.

4.2 Simplification and Visibility Culling

To produce a more compact representation, we simplify the mesh with an off-the-shelf tool based on quadric edge collapse decimation [Garland and Heckbert 1997]. We found this approach to dramatically simplify our meshes while still preserving thin structures. We explicitly simplify the mesh in far-away regions more aggressively, as described in Appendix C. After simplification, we cull triangles that are not visible from any training camera, which leads to another significant reduction in the number of triangles. Only using the training cameras' poses for visibility estimation leads to holes in the mesh that become apparent during novel view synthesis. To combat this, we augment the set of camera poses used for visibility estimation: we create additional poses by adding randomly sampled offsets and rotations to the poses of the training cameras, as described in Appendix C. We find that it is crucial to perform culling *after* simplification, as mesh simplification methods tend to not be robust to the numerous small holes introduced by culling.

5 VIEW-DEPENDENT APPEARANCE FOR MESHES

To enable view synthesis we need a view-dependent appearance model for our reconstructed mesh. To this end, we evaluate a number of potential representations and encodings for view-dependent color, with a focus on options that are suited for real-time rendering.

5.1 Spatial Parameterization

We begin by exploring parameterizations which efficiently map positions on our mesh to coefficients that encode appearance.

UV mapping. UV texture maps are the most ubiquitous representation for appearance. However, we found that current UV mapping tools cannot deal well with the complexity of our input mesh, which contains a lot of fine geometric detail (though concurrent work such as Srinivasan et al. [2023] may provide a viable path).

Vertex Attributes. Prior mesh-based view synthesis methods like Yariv et al. [2023] store appearance coefficients at vertex attributes on the mesh and then interpolatie them across each face. Unfortunately, this requires the vertex density to be higher than the desired texture density, which results in prohibitively large and expensive meshes. This is not the case for us, as our meshes are drastically simplified, leading to large triangles in geometrically simple regions.

Volume Textures. We can directly associate a color value with each 3D position using a 3D volume texture. A simple way to encode a volume sparsely is to subdivide the volume into blocks of D^3 voxels and store only the nonempty blocks [Hedman et al. 2021]. The choice of the block size D involves a trade-off: A small block size yields high compactness, but results in poor data locality, which leads to slow rendering. A large block size comes with high memory consumption, since any block that contains a single surface-adjacent voxel must be allocated. This also holds for alternative sparse data structures such as octrees [Benson and Davis 2002] or spatial hashing [Lefebvre and Hoppe 2006], since they equally depend on blocking for fast access.

Triplanes and Low-resolution Voxel Grid. Recently, it has been shown that volume textures can be encoded compactly with a combination of triplanes and a low-resolution voxel grid [Reiser et al. 2023]. Both the triplanes and the low-resolution voxel grid are cache-friendly, leading to fast random access.

Table 1 shows that volume textures yield the highest quality, followed by the combination of triplanes and low-resolution voxel grid. The gap to vertex attributes is more pronounced, which you can also see in Figure 4: vertex attributes look blurry in geometrically simple regions. Finally, while the triplane and grid combination uses less memory than and is much faster than volume textures, these representations look nearly identical in Figure 4.

5.2 View-Dependence

We also investigate several encodings for view-dependent color: **spherical harmonics** and **spherical Gaussians**, which are established formats for view-dependent colors in real-time view synthesis systems [Fridovich-Keil et al. 2022; Yariv et al. 2023; Yu et al. 2021],

Table 1. Different representations for mesh appearance in gardenvase. Replacing vertex attributes with a 3D grid leads to higher quality at the cost of higher memory consumption (VRAM). At a slight quality loss, the "triplane + voxel" option is more compact, while rendering faster than the alternatives. All rows except the last one use spherical Gaussians to model view-dependence. The last row (offline) is an upper bound on quality and uses an expensive appearance network.

	PSNR ↑	SSIM ↑	LPIPS ↓	VRAM ↓	FPS ↑
vertex attributes	25.58	0.771	0.211	**97**	261
volume textures	**26.25**	**0.820**	**0.143**	4513	169
triplane + voxel	26.02	0.807	0.157	629	**477**
offline	26.86	0.830	0.135	–	–

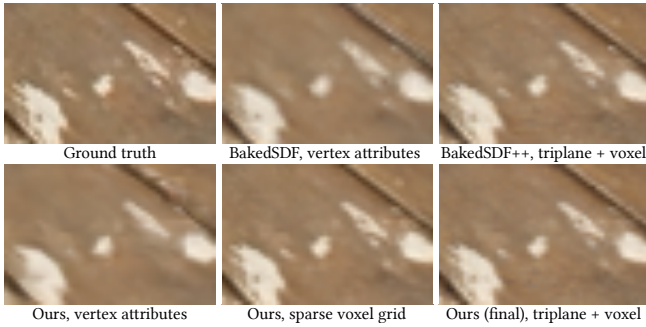| Ground truth | BakedSDF, vertex attributes | BakedSDF++, triplane + voxel |
| Ours, vertex attributes | Ours, sparse voxel grid | Ours (final), triplane + voxel |

Fig. 4. **Different representations for mesh appearance.** Replacing vertex attributes with a grid representation leads to sharper textures. The cheaper alternative "triplane + voxel" is very similar to "voxel grid".

Table 2. View-dependency encodings on gardenvase using our combination of triplanes and a low-resolution voxel grid.

	PSNR ↑	SSIM ↑	LPIPS ↓	bytes ↓
Spherical Gaussians	**26.02**	**0.807**	**0.157**	24
Spherical Harmonics	25.65	0.797	0.166	27
8-dim. Neural Feature	25.18	0.781	0.179	**8**
24-dim. Neural Feature	25.72	0.798	0.164	24

and **neural feature vectors** that get decoded to a view-dependent color with a small MLP [Chen et al. 2018; Hedman et al. 2021]. As can be seen in Table 2, spherical Gaussians deliver the highest quality, while only requiring 24 bytes instead of 27 bytes per texel compared to spherical harmonics. To fairly compare neural feature vectors with spherical Gaussians, we choose a 24-dimensional neural feature vector to match memory consumption and bandwidth requirements. Our decoder uses the same architecture as other recent real-time view synthesis systems [Chen et al. 2023; Duckworth et al. 2023; Hedman et al. 2021; Reiser et al. 2023]. Even with a large neural feature vector, this underperforms the non-neural baselines. As such, in our model, we use spherical Gaussians with the triplane and low-resolution grid combination, which gives the best trade-off between rendering speed, quality and memory consumption.

5.3 Real-Time Implementation

We implement a prototype web viewer for our representation based on Three.js. Since our meshes contain many tiny structures, anti-aliasing (AA) is critical during rendering. To avoid the computational expense of supersampling anti-aliasing (SSAA), we implement temporal anti-aliasing (TAA) [Yang et al. 2020], which amortizes sampling across time and only needs a single sample per pixel per frame, as described in Appendix D. We find that the quality of our TAA is on par with the significantly more expensive SSAA, even when capturing frames under motion.

6 EXPERIMENTS

We conduct experiments on the indoor and outdoor scenes of the challenging dataset from Mip-NeRF 360 [Barron et al. 2022], where

Table 3. Test-time anti-aliasing algorithms on the outdoor scenes from the mip-NeRF 360 dataset [Barron et al. 2022]. TAA achieves nearly the same fidelity as significantly more expensive SSAA.

	PSNR ↑	SSIM ↑	LPIPS ↓	FPS ↓
SSAA	23.94	**0.680**	**0.263**	50
TAA, stationary	**24.00**	**0.680**	0.266	448
TAA, under motion	23.92	0.676	0.270	448
No AA	23.26	0.652	0.287	**477**

we compare our method in terms of quality, rendering speed, memory consumption, and storage impact to a range of volume-based and surface-based alternatives. We additionally conduct an ablation study to illustrate how our individual components contribute towards accurately reconstructing thin structures.

Test-time Anti-Aliasing. Using anti-aliasing for test-time rendering is crucial for high quality, as evidenced by Table 3. To this end, we study in our setting whether SSAA can be replaced by cheaper TAA without negatively affecting view synthesis quality. For this comparison, we always employ our previously-detailed 16× supersampling strategy during training and only vary the anti-aliasing algorithm used for test-time rendering. Since TAA may introduce blur under motion, we also measure the quality of images that were captured after moving the camera over a fixed number of frames to the target pose. As Table 3 shows, TAA has comparable quality with the more expensive SSAA even when the camera is moving, making it a viable choice for our viewer.

Comparison with BakedSDF. We compare with BakedSDF — the state-of-the-art for real-time, mesh-based view synthesis — in all scenes from Mip-NeRF 360 [Barron et al. 2022]. To disentangle whether differences in rendering quality between BakedSDF and our method stem from geometry or appearance, we fit our best-performing appearance model (Section 5) to the meshes from BakedSDF, i.e., we encode appearance using a combination of triplanes and low-resolution voxel grid instead of vertex attributes. We also use the same supersampling strategy for BakedSDF and our method. We call this improved version BakedSDF++.

As can be seen in Figure 4, fitting our own appearance model to the meshes from BakedSDF leads to sharper textures, since with the combination of triplanes and low-resolution voxel grid, texture resolution is not bounded by vertex density. This indicates that this representation might be a viable alternative to vertex attributes even for dense meshes as produced by BakedSDF.

More importantly, as Figure 7 shows, our method is significantly better than BakedSDF at reconstructing thin structures, which are often absent for this baseline. Furthermore, in Table 6 our method outperforms BakedSDF++ across all key metrics, highlighting that our meshes are better for view synthesis than those from BakedSDF.

Comparison with other Baselines. We also compare our method with a broader set of baselines in terms of quality and rendering speed. We benchmark the volume-based baselines MERF and 3DGS and the surface-based method BakedSDF on a Google Pixel 8 Pro smartphone, a MacBook M1 Pro (2022) laptop and a desktop equipped with an NVIDIA RTX 3090 graphics card. We report the harmonic mean of frames per second (FPS) on the outdoor scenes of

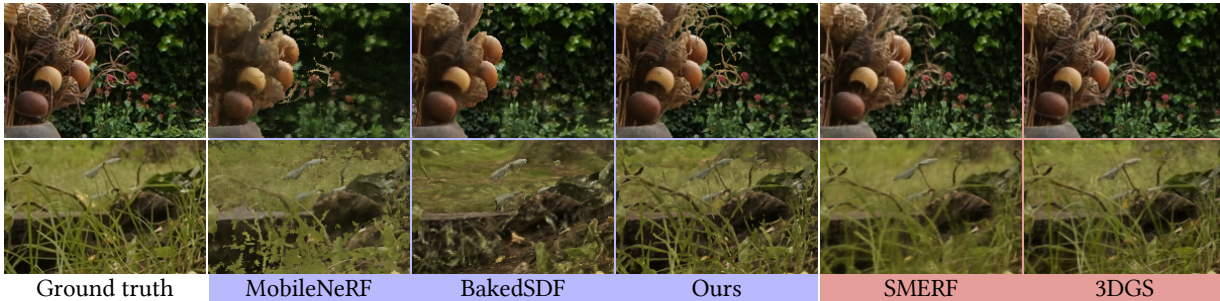| Ground truth | MobileNeRF | BakedSDF | Ours | SMERF | 3DGS |

Fig. 5. Our method narrows the quality gap between **surface-based** and **volume-based** methods when it comes to the reconstruction of thin structures.

Table 4. Results on the outdoor and indoor scenes from mip-NeRF 360 [Barron et al. 2022], split into **volume-based** and **surface-based** methods. The best metric in each family shown in **bold** and "−" denotes missing metrics.

	Outdoor Scenes			Indoor Scenes		
	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓
Instant-NGP [2022]	22.90	0.566	0.371	29.15	0.880	0.216
MERF [2023]	23.19	0.616	0.343	27.80	0.855	0.271
3DGS [2023]	24.64	0.731	0.234	30.41	0.920	0.189
Zip-NeRF [2023]	**25.68**	**0.761**	**0.208**	**32.65**	**0.929**	**0.168**
Shells [2023b]	23.17	0.606	0.389	29.19	0.872	0.285
SMERF [2023]	25.32	0.739	0.232	31.32	0.917	0.186
Mobile-NeRF [2023]	21.95	0.470	0.470	–	–	–
BakedSDF [2023]	22.47	0.585	0.349	27.06	0.836	0.258
Ours (SSAA)	**23.94**	**0.680**	**0.263**	**27.71**	**0.873**	**0.227**

Table 5. Rendering speed comparison in frames per second. Our method is significantly faster than **volume-based** and **surface-based** baselines and is the only method capable of real-time rendering on our test smartphone.

Device	Smartphone	Laptop	Desktop
Resolution	400 × 750	1280 × 720	1920 × 1080
MERF [2023]	10	21	113
3DGS [2023]	–	–	176
BakedSDF [2023]	19	81	412
Ours (TAA)	**67**	**448**	**927**

Table 6. Comparison between BakedSDF, an improved version of BakedSDF (BakedSDF++) and our method on the outdoor scenes from mip-NeRF 360 [Barron et al. 2022].

	PSNR ↑	SSIM ↑	LPIPS ↓	#faces ↓
BakedSDF	22.47	0.585	0.349	40M
BakedSDF++	22.50	0.612	0.315	40M
Ours (SSAA)	**23.94**	**0.680**	**0.263**	**13M**

the Mip-NeRF 360 dataset [Barron et al. 2022]. In terms of rendering speed, our mesh-based representation outperforms all volume-based baselines, see Table 5. In terms of quality metrics, our method still lags behind the most recent volume-based baselines, as can be seen in Table 4. However, the quality gap between surface-based and volume-based methods is significantly reduced, especially for thin structures as shown by Figure 5.

Table 7. Geometric ablations on the outdoor scenes from the mip-NeRF 360 dataset [Barron et al. 2022].

	PSNR ↑	SSIM ↑	LPIPS ↓
(a) No supersampling	23.38	0.645	0.292
(b) No entropy loss	23.21	0.635	0.293
(c) R = 2048 instead of R = 8192	22.44	0.582	0.343
Ours (SSAA)	**23.94**	**0.680**	**0.263**

Geometry Ablations. We conduct an ablation study on the outdoor scenes from mip-NeRF 360 [Barron et al. 2022] where we focus on the initial opacity grid, which determines the quality of the mesh.

We train a variant of our model without supersampling (a). In this case, we only disable supersampling during training of the binary occupancy grid, but we still use supersampling for fitting the mesh appearance model and for computing quality metrics. This isolates the effect supersampling has on the quality of the obtained mesh. As shown in Figure 6, thin structures are hard to recover without casting multiple rays per pixel during training.

Next, we train a variant of our model without the entropy loss (b). Since for this model, many opacity values do not become binary during the course of training, we define depth as the distance to the first voxel along the ray with an opacity value greater than 0.5. Similar to (a), the effect of this is most pronounced for very thin structures such as the ones shown in Figure 6.

Finally, we decrease the resolution R of the BOG (c). For this experiment, we only decrease the resolution of the initial BOG, but we use the same resolution of triplanes and the low-resolution voxel grid during mesh appearance fitting as for the full model. This isolates the effect geometric resolution has on mesh quality. As can be seen in Figure 6, a high resolution is crucial for reconstructing thin structures. Quantitative results for these ablations are given in Table 7.

Storage Analysis. Next, we study how the individual components of our representation contribute to disk storage and memory consumption. We split our representation into a mesh and an appearance model. As we have shown experimentally, reconstructing thin structures requires a high grid resolution. As can be seen in Table 8, without any further processing, this leads to meshes with billions of faces, resulting in an impractical storage requirement of over 20 GiB. However, using simplification and culling, the size of the mesh

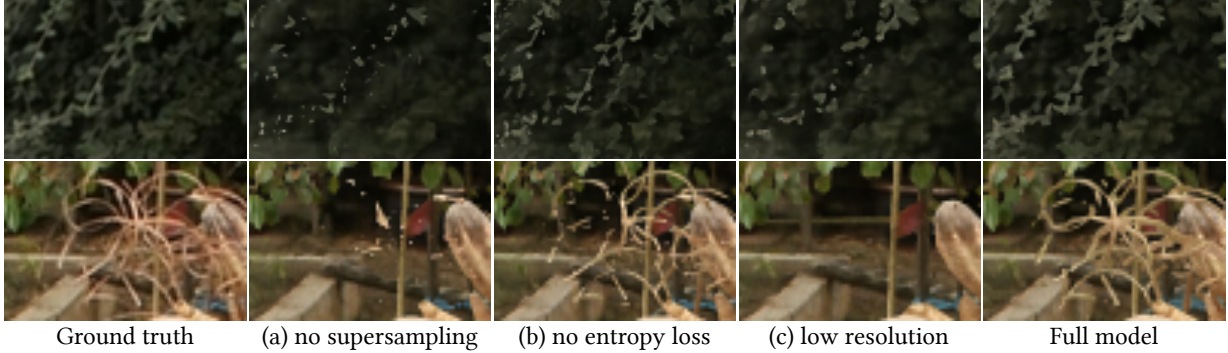| Ground truth | (a) no supersampling | (b) no entropy loss | (c) low resolution | Full model |

Fig. 6. Geometric ablations. Recovering challenging thin structures requires (a) casting multiple rays per pixel, (b) the entropy loss, and (c) the grid resolution.

Table 8. Memory consumption and on-disk size. (a) The dense mesh before simplification, (b) after simplification, (c) after culling. Results are averaged over all scenes from mipNeRF-360 [Barron et al. 2022].

		(a) Dense Mesh	(b) + Simpl.	(c) + Culling
Mesh	#vertices	606M	9M	**7M**
Mesh	#faces	1208M	18M	**10M**
Mesh	VRAM	20.28 GiB	0.30 GiB	**0.19 GiB**
Mesh	DISK	21.40 GiB	0.32 GiB	**0.20 GiB**
Appearance	VRAM	0.75 GiB	0.75 GiB	**0.75 GiB**
Appearance	DISK	0.65 GiB	0.65 GiB	**0.65 GiB**
Total	VRAM	21.02 GiB	1.05 GiB	**0.94 GiB**
Total	DISK	22.05 GiB	0.97 GiB	**0.85 GiB**

Table 10. Processing times for the stages of our pipeline.

Stage	hours
Zip-NeRF Optimization	2.4
BOG Optimization	7.1
Volumetric Fusion	20.7
Simplification	3.7
Mesh Appearance Optimization	26.2
Total	60.1

Processing Time. Table 10 shows processing times for each stage of our pipeline. All stages use 8×V100 GPUs, except for volumetric fusion (single V100) and simplification (CPU only). Our implementation of volumetric fusion is highly inefficient, offering potential for significant speed-ups with, e.g., a CUDA implementation. Additionally, using 16x supersampling results in roughly 3x slower training than Zip-NeRF, which uses 6x multisampling [Barron et al. 2023]. Exploring alternatives to supersampling or limiting it to fewer pixels is promising future work.

can be reduced by a factor of 100 to around 200 MiB. This results in the size of the representation being dominated by the appearance model, which occupies around 76% of the overall storage.

Quality Loss Analysis. Finally, we analyze the quality loss from each concession we make to achieve real-time rendering. We start from the state-of-the-art offline method Zip-NeRF, which we modify to extract surface geometry. Since our BOG uses the same architecture as Zip-NeRF, any loss in rendering quality before meshing is from our surface constraints (entropy regularization). Row 2 in Table 9 shows that this leads to a 1.28 dB drop in PSNR. Next, we convert our opacity grid into a triangle mesh, which only decreases PSNR by 0.06 dB (Table 9, row 3). To isolate the quality loss incurred by meshing, we equipped the mesh with the same offline appearance model that was used during BOG optimization. Comparing row 3 and 4 in Table 9, we see that switching to our lightweight appearance model incurs a quality loss of 0.39 dB.

Table 9. Quality loss incurred by real-time concessions. Results are averaged over the outdoor scenes from mip-NeRF 360 [Barron et al. 2022].

	PSNR ↑	SSIM ↑	LPIPS ↓
(1) Zip-NeRF	**25.68**	**0.761**	**0.208**
(2) BOG before meshing	24.40	0.698	0.263
(3) Our mesh: offline appearance	24.34	0.699	0.239
(4) Our mesh: real-time appearance	23.94	0.680	0.263

7 LIMITATIONS AND FUTURE WORK

Like other mesh-based view synthesis methods, ours does not handle semi-transparent objects [Chen et al. 2023; Yariv et al. 2023]. While our meshes render quickly, the processing time for our reconstruction pipeline is substantial, as discussed in the previous paragraph. The reconstruction of the underconstrained background of the scene is often highly noisy, which significantly increases the size of our meshes. This could potentially be mitigated with a smoothness regularizer. The concurrent work Nuvo presents a UV mapping method that is suited for high-detail meshes such as ours [Srinivasan et al. 2023]. Representing appearance with UV textures and obtaining more compact meshes using smoothness regularization could significantly speed-up rendering and save memory. Another consequence of the lack of smoothness is that our meshes' normals are too noisy for relighting. We found the quality difference between our approach and volume-based methods larger in the indoor scenes. We attribute this to changes in illumination (e.g. shadows from the photographer) that are difficult to capture on surfaces with a low-capacity view-dependence model. Indeed, we
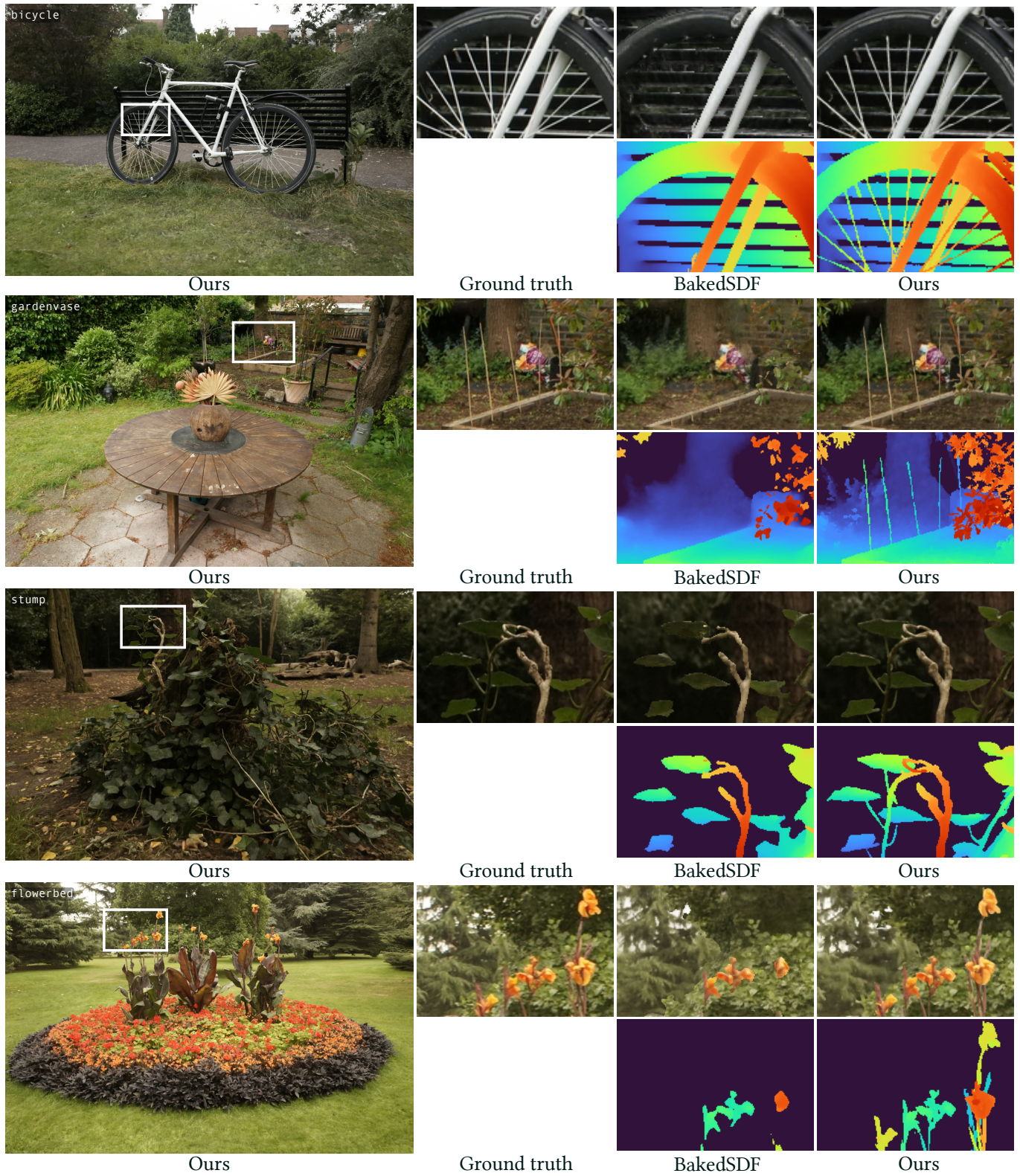
Fig. 7. Our method retains more geometric detail than BakedSDF. Bottom rows are visualizations of depth maps, for which we do not have ground-truth.

found a large offline view-dependence network to yield significantly higher quality in these scenes. In general, a quality gap remains between our approach and volume-based methods. We hypothesize that disturbances during capture, like inaccurate poses, wind, motion blur, or depth of field, present a greater challenge for surface-based approaches since volumetric approaches can fuzzily resolve these disturbances. Furthermore, volumetric approaches excel at modeling reflections by exploiting semi-transparency to create "mirror worlds".

8 CONCLUSION

We have presented the first mesh-based view synthesis algorithm that is capable of reproducing subpixel structures in the input images by employing a high-resolution opacity grid combined with supersampling and a binary entropy loss. In contrast to volume-based alternatives, our method renders in real-time on affordable smartphones. Compared to BakedSDF, the previous state-of-the-art in mesh-based view synthesis, our method yields 3 times more compact meshes and achieves 1.46 dB higher PSNR in outdoor scenes.

ACKNOWLEDGMENTS

REFERENCES

Jonathan T. Barron, Ben Mildenhall, Dor Verbin, Pratul P. Srinivasan, and Peter Hedman. 2022. Mip-NeRF 360: Unbounded Anti-Aliased Neural Radiance Fields. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 5460–5469. https://doi.org/10.1109/CVPR52688.2022.00539

Jonathan T. Barron, Ben Mildenhall, Dor Verbin, Pratul P. Srinivasan, and Peter Hedman. 2023. Zip-NeRF: Anti-Aliased Grid-Based Neural Radiance Fields. In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*. 19640–19648. https://doi.org/10.1109/ICCV51070.2023.01804

David Benson and Joel Davis. 2002. Octree textures. *ACM Trans. Graph.* 21, 3 (jul 2002), 785–790. https://doi.org/10.1145/566654.566652

Anpei Chen, Minye Wu, Yingliang Zhang, Nianyi Li, Jie Lu, Shenghua Gao, and Jingyi Yu. 2018. Deep Surface Light Fields. *Proc. ACM Comput. Graph. Interact. Tech.* 1, 1, Article 14 (jul 2018), 17 pages. https://doi.org/10.1145/3203192

Anpei Chen, Zexiang Xu, Andreas Geiger, Jingyi Yu, and Hao Su. 2022. TensoRF: Tensorial Radiance Fields. In *Computer Vision – ECCV 2022*, Shai Avidan, Gabriel Brostow, Moustapha Cissé, Giovanni Maria Farinella, and Tal Hassner (Eds.). Springer Nature Switzerland, Cham, 333–350.

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 *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 16569–16578. https://doi.org/10.1109/CVPR52729.2023.01590

Brian Curless and Marc Levoy. 1996. A volumetric method for building complex models from range images. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '96)*. Association for Computing Machinery, New York, NY, USA, 303–312. https://doi.org/10.1145/237170.237269

Paul Debevec, Yizhou Yu, and George Borshukov. 1998. Efficient View-Dependent Image-Based Rendering with Projective Texture-Mapping. In *Rendering Techniques '98*, George Drettakis and Nelson Max (Eds.). Springer Vienna, Vienna, 105–116.

Claude E. Duchon. 1979. Lanczos Filtering in One and Two Dimensions. *Journal of Applied Meteorology and Climatology* 18, 8 (1979), 1016 – 1022. https://doi.org/10.1175/1520-0450(1979)018<1016:LFIOAT>2.0.CO;2

Daniel Duckworth, Peter Hedman, Christian Reiser, Peter Zhizhin, Jean-François Thibert, Mario Lučić, Richard Szeliski, and Jonathan T. Barron. 2023. SMERF: Streamable Memory Efficient Radiance Fields for Real-Time Large-Scene Exploration. arXiv:cs.CV/2312.07541

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

In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 5491–5500. https://doi.org/10.1109/CVPR52688.2022.00542

Stephan J. Garbin, Marek Kowalski, Matthew Johnson, Jamie Shotton, and Julien Valentin. 2021. FastNeRF: High-Fidelity Neural Rendering at 200FPS. In *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*. 14326–14335. https://doi.org/10.1109/ICCV48922.2021.01408

Michael Garland and Paul S. Heckbert. 1997. Surface simplification using quadric error metrics. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '97)*. ACM Press/Addison-Wesley Publishing Co., USA, 209–216. https://doi.org/10.1145/258734.258849

Yuan-Chen Guo, Yan-Pei Cao, Chen Wang, Yu He, Ying Shan, and Song-Hai Zhang. 2023. VMesh: Hybrid Volume-Mesh Representation for Efficient View Synthesis. In *SIGGRAPH Asia 2023 Conference Papers (SA '23)*. Association for Computing Machinery, New York, NY, USA, Article 17, 11 pages. https://doi.org/10.1145/3610548.3618161

Antoine Guédon and Vincent Lepetit. 2023. SuGaR: Surface-Aligned Gaussian Splatting for Efficient 3D Mesh Reconstruction and High-Quality Mesh Rendering. arXiv:cs.GR/2311.12775

Peter Hedman, Pratul P. Srinivasan, Ben Mildenhall, Jonathan T. Barron, and Paul Debevec. 2021. Baking Neural Radiance Fields for Real-Time View Synthesis. In *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*. 5855–5864. https://doi.org/10.1109/ICCV48922.2021.00582

Binbin Huang, Zehao Yu, Anpei Chen, Andreas Geiger, and Shenghua Gao. 2024. 2D Gaussian Splatting for Geometrically Accurate Radiance Fields. arXiv:cs.CV/2403.17888

Michal Jancosek and Tomas Pajdla. 2011. Multi-view reconstruction preserving weakly-supported surfaces. In *CVPR 2011*. 3121–3128. https://doi.org/10.1109/CVPR.2011.5995693

Brian Karis. 2014. High Quality Temporal Supersampling. ACM SIGGRAPH Courses: Advances in Real-Time Rendering in Games.

Bernhard Kerbl, Georgios Kopanas, Thomas Leimkuehler, and George Drettakis. 2023. 3D Gaussian Splatting for Real-Time Radiance Field Rendering. *ACM Trans. Graph.* 42, 4, Article 139 (jul 2023), 14 pages. https://doi.org/10.1145/3592433

Diederik P. Kingma and Jimmy Ba. 2017. Adam: A Method for Stochastic Optimization. arXiv:cs.LG/1412.6980

Georgios Kopanas, Julien Philip, Thomas Leimkühler, and George Drettakis. 2021. Point-Based Neural Rendering with Per-View Optimization. *Computer Graphics Forum* 40, 4 (2021), 29–43. https://doi.org/10.1111/cgf.14339 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.14339

Sylvain Lefebvre and Hugues Hoppe. 2006. Perfect spatial hashing. *ACM Trans. Graph.* 25, 3 (jul 2006), 579–588. https://doi.org/10.1145/1141911.1141926

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 *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 8456–8465. https://doi.org/10.1109/CVPR52729.2023.00817

Yiyi Liao, Simon Donné, and Andreas Geiger. 2018. Deep Marching Cubes: Learning Explicit Surface Representations. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2916–2925. https://doi.org/10.1109/CVPR.2018.00308

Lingjie Liu, Jiatao Gu, Kyaw Zaw Lin, Tat-Seng Chua, and Christian Theobalt. 2020. Neural Sparse Voxel Fields. In *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (Eds.), Vol. 33. Curran Associates, Inc., 15651–15663. https://proceedings.neurips.cc/paper_files/paper/2020/file/b4b758962f17808746e9bb832a6fa4b8-Paper.pdf

Stephen Lombardi, Tomas Simon, Jason Saragih, Gabriel Schwartz, Andreas Lehrmann, and Yaser Sheikh. 2019. Neural volumes: learning dynamic renderable volumes from images. *ACM Trans. Graph.* 38, 4, Article 65 (jul 2019), 14 pages. https://doi.org/10.1145/3306346.3323020

Xiaoyang Lyu, Yang-Tian Sun, Yi-Hua Huang, Xiuzhe Wu, Ziyi Yang, Yilun Chen, Jiangmiao Pang, and Xiaojuan Qi. 2024. 3DGSR: Implicit Surface Reconstruction with 3D Gaussian Splatting. arXiv:cs.CV/2404.00409

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 *Computer Vision – ECCV 2020*, Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm (Eds.). Springer International Publishing, Cham, 405–421.

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 (jul 2022), 15 pages. https://doi.org/10.1145/3528223.3530127

Jacob Munkberg, Wenzheng Chen, Jon Hasselgren, Alex Evans, Tianchang Shen, Thomas Müller, Jun Gao, and Sanja Fidler. 2022. Extracting Triangular 3D Models, Materials, and Lighting From Images. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 8270–8280. https://doi.org/10.1109/CVPR52688.2022.00810

Michael Niemeyer, Lars Mescheder, Michael Oechsle, and Andreas Geiger. 2020. Differentiable Volumetric Rendering: Learning Implicit 3D Representations Without 3D Supervision. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition*

(CVPR). 3501–3512. https://doi.org/10.1109/CVPR42600.2020.00356

Michael Oechsle, Songyou Peng, and Andreas Geiger. 2021. UNISURF: Unifying Neural Implicit Surfaces and Radiance Fields for Multi-View Reconstruction. In *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*. 5569–5579. https://doi.org/10.1109/ICCV48922.2021.00554

Julien Philip, Sébastien Morgenthaler, Michaël Gharbi, and George Drettakis. 2021. Free-viewpoint Indoor Neural Relighting from Multi-view Stereo. *ACM Trans. Graph.* 40, 5, Article 194 (sep 2021), 18 pages. https://doi.org/10.1145/3469842

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. *3DV* (2023).

Christian Reiser, Rick Szeliski, Dor Verbin, Pratul Srinivasan, Ben Mildenhall, Andreas Geiger, Jon Barron, and Peter Hedman. 2023. MERF: Memory-Efficient Radiance Fields for Real-time View Synthesis in Unbounded Scenes. *ACM Trans. Graph.* 42, 4, Article 89 (jul 2023), 12 pages. https://doi.org/10.1145/3592426

G. Riegler and V. Koltun. 2021. Stable View Synthesis. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE Computer Society, Los Alamitos, CA, USA, 12211–12220. https://doi.org/10.1109/CVPR46437.2021.01204

Darius Rückert, Linus Franke, and Marc Stamminger. 2022a. ADOP: approximate differentiable one-pixel point rendering. *ACM Trans. Graph.* 41, 4, Article 99 (jul 2022), 14 pages. https://doi.org/10.1145/3528223.3530122

Darius Rückert, Yuanhao Wang, Rui Li, Ramzi Idoughi, and Wolfgang Heidrich. 2022b. NeAT: neural adaptive tomography. *ACM Trans. Graph.* 41, 4, Article 55 (jul 2022), 13 pages. https://doi.org/10.1145/3528223.3530121

Marco Salvi. 2016. An excursion in temporal supersampling. Game Developers Conference.

Johannes L. Schönberger, Enliang Zheng, Jan-Michael Frahm, and Marc Pollefeys. 2016. Pixelwise View Selection for Unstructured Multi-View Stereo. In *Computer Vision – ECCV 2016*, Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling (Eds.). Springer International Publishing, Cham, 501–518.

Tianchang Shen, Jun Gao, Kangxue Yin, Ming-Yu Liu, and Sanja Fidler. 2021. Deep Marching Tetrahedra: a Hybrid Representation for High-Resolution 3D Shape Synthesis. In *Advances in Neural Information Processing Systems*, M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan (Eds.), Vol. 34. Curran Associates, Inc., 6087–6101. https://proceedings.neurips.cc/paper_files/paper/2021/file/30a237d18c50f563cba4531f1db44acf-Paper.pdf

Tianchang Shen, Jacob Munkberg, Jon Hasselgren, Kangxue Yin, Zian Wang, Wenzheng Chen, Zan Gojcic, Sanja Fidler, Nicholas Sharp, and Jun Gao. 2023. Flexible Isosurface Extraction for Gradient-Based Mesh Optimization. *ACM Trans. Graph.* 42, 4, Article 37 (jul 2023), 16 pages. https://doi.org/10.1145/3592430

Pratul P. Srinivasan, Stephan J. Garbin, Dor Verbin, Jonathan T. Barron, and Ben Mildenhall. 2023. Nuvo: Neural UV Mapping for Unruly 3D Representations. arXiv:cs.CV/2312.05283

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. In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*. 17693–17703. https://doi.org/10.1109/ICCV51070.2023.01626

Justus Thies, Michael Zollhöfer, and Matthias Nießner. 2019. Deferred neural rendering: image synthesis using neural textures. *ACM Trans. Graph.* 38, 4, Article 66 (jul 2019), 12 pages. https://doi.org/10.1145/3306346.3323035

Haithem Turki, Vasu Agrawal, Samuel Rota Bulò, Lorenzo Porzi, Peter Kontschieder, Deva Ramanan, Michael Zollhöfer, and Christian Richardt. 2023. HybridNeRF: Efficient Neural Rendering via Adaptive Volumetric Surfaces. arXiv:cs.CV/2312.03160

Oliver T. Unke and Hartmut Maennel. 2024. E3x: E(3)-Equivariant Deep Learning Made Easy. *arXiv* (2024).

Michael Waechter, Nils Moehrle, and Michael Goesele. 2014. Let There Be Color! Large-Scale Texturing of 3D Reconstructions. In *Computer Vision – ECCV 2014*, David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars (Eds.). Springer International Publishing, Cham, 836–850.

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. In *Advances in Neural Information Processing Systems*, M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan (Eds.), Vol. 34. Curran Associates, Inc., 27171–27183. https://proceedings.neurips.cc/paper_files/paper/2021/file/e41e164f7485ec4a28741a2d0ea41c74-Paper.pdf

Yiming Wang, Qin Han, Marc Habermann, Kostas Daniilidis, Christian Theobalt, and Lingjie Liu. 2023a. NeuS2: Fast Learning of Neural Implicit Surfaces for Multi-view Reconstruction. In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*. 3272–3283. https://doi.org/10.1109/ICCV51070.2023.00305

Zian Wang, Tianchang Shen, Merlin Nimier-David, Nicholas Sharp, Jun Gao, Alexander Keller, Sanja Fidler, Thomas Müller, and Zan Gojcic. 2023b. Adaptive Shells for Efficient Neural Radiance Field Rendering. *ACM Trans. Graph.* 42, 6, Article 260 (dec 2023), 15 pages. https://doi.org/10.1145/3618390

Yaniv Wolf, Amit Bracha, and Ron Kimmel. 2024. Surface Reconstruction from Gaussian Splatting via Novel Stereo Views. arXiv:cs.CV/2404.01810

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., USA, 287–296. https://doi.org/10.1145/344779.344925

Markus Worchel, Rodrigo Diaz, Weiwen Hu, Oliver Schreer, Ingo Feldmann, and Peter Eisert. 2022. Multi-View Mesh Reconstruction with Neural Deferred Shading. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 6177–6187. https://doi.org/10.1109/CVPR52688.2022.00609

Qiangeng Xu, Zexiang Xu, Julien Philip, Sai Bi, Zhixin Shu, Kalyan Sunkavalli, and Ulrich Neumann. 2022. Point-NeRF: Point-based Neural Radiance Fields. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 5428–5438. https://doi.org/10.1109/CVPR52688.2022.00536

Han Yan, Celong Liu, Chao Ma, and Xing Mei. 2023. Plen-VDB: Memory Efficient VDB-Based Radiance Fields for Fast Training and Rendering. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 88–96. https://doi.org/10.1109/CVPR52729.2023.00017

Lei Yang, Shiqiu Liu, and Marco Salvi. 2020. A Survey of Temporal Antialiasing Techniques. *Computer Graphics Forum* 39, 2 (2020), 607–621. https://doi.org/10.1111/cgf.14018 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.14018

Lior Yariv, Jiatao Gu, Yoni Kasten, and Yaron Lipman. 2021. Volume Rendering of Neural Implicit Surfaces. In *Advances in Neural Information Processing Systems*, M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan (Eds.), Vol. 34. Curran Associates, Inc., 4805–4815. https://proceedings.neurips.cc/paper_files/paper/2021/file/25e2a30f44898b9f3e978b1786dcd85c-Paper.pdf

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. In *ACM SIGGRAPH 2023 Conference Proceedings (SIGGRAPH '23)*. Association for Computing Machinery, New York, NY, USA, Article 46, 9 pages. https://doi.org/10.1145/3588432.3591536

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. In *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (Eds.), Vol. 33. Curran Associates, Inc., 2492–2502. https://proceedings.neurips.cc/paper_files/paper/2020/file/1a77befc3b608d6ed363567685f70e1e-Paper.pdf

Alex Yu, Ruilong Li, Matthew Tancik, Hao Li, Ren Ng, and Angjoo Kanazawa. 2021. PlenOctrees for Real-time Rendering of Neural Radiance Fields. In *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*. 5732–5741. https://doi.org/10.1109/ICCV48922.2021.00570

Mulin Yu, Tao Lu, Linning Xu, Lihan Jiang, Yuanbo Xiangli, and Bo Dai. 2024a. GSDF: 3DGS Meets SDF for Improved Rendering and Reconstruction. arXiv:cs.CV/2403.16964

Zehao Yu, Anpei Chen, Bozidar Antic, Songyou Peng, Apratim Bhattacharyya, Michael Niemeyer, Siyu Tang, Torsten Sattler, and Andreas Geiger. 2022a. SDFStudio: A Unified Framework for Surface Reconstruction. https://github.com/autonomousvision/sdfstudio

Zehao Yu, Songyou Peng, Michael Niemeyer, Torsten Sattler, and Andreas Geiger. 2022b. MonoSDF: Exploring Monocular Geometric Cues for Neural Implicit Surface Reconstruction. In *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (Eds.), Vol. 35. Curran Associates, Inc., 25018–25032. https://proceedings.neurips.cc/paper_files/paper/2022/file/9f0b1220028dfa2ee82ca0a0e0fc52d1-Paper-Conference.pdf

Zehao Yu, Torsten Sattler, and Andreas Geiger. 2024b. Gaussian Opacity Fields: Efficient and Compact Surface Reconstruction in Unbounded Scenes. arXiv:cs.CV/2404.10772

Yiyu Zhuang, Qi Zhang, Ying Feng, Hao Zhu, Yao Yao, Xiaoyu Li, Yan-Pei Cao, Ying Shan, and Xun Cao. 2023. Anti-Aliased Neural Implicit Surfaces with Encoding Level of Detail. In *SIGGRAPH Asia 2023 Conference Papers (SA '23)*. Association for Computing Machinery, New York, NY, USA, Article 119, 10 pages. https://doi.org/10.1145/3610548.3618197

A FUSION ALGORITHM

When converting our trained binary opacity grid into a triangle mesh, it is crucial to only instantiate geometry in regions that were sampled by the proposal MLP, since only these regions were supervised during training. This can be achieved by rendering depth maps from the training viewpoints using the proposal MLP and creating surface voxels via unprojection. However, during training, some voxels may only be sampled in a fraction of the views and thus have incorrect opacity values. This leads to floating artifacts in the resulting mesh (see Figure 3).

We use volumetric fusion to filter these underconstrained voxels. Specifically, for each voxel, we count for how many depth maps the voxel is observed 1) in free space and 2) on the surface. As underconstrained voxels appear in very few training views, we can detect and discard them by only keeping voxels that are more frequently observed on the surface rather than in free space.

Another important motivation for volumetric fusion is that it outputs a dense implicit representation which can easily be converted into a hole-free mesh — the preferred input for most simplification algorithms. Consequently, we would like our fusion algorithm to also label unobserved voxels as "inside" or "outside". A simple heuristic is to lo label unobserved voxels as "inside" [Curless and Levoy 1996]. However, in rare cases, the proposal MLP does not sample the surface of an object, which carves large holes into the resulting mesh. We address this by requiring several views to observe a voxel in free space before it can be labelled as outside.

This leads to our fusion algorithm. For each voxel, we count:

- S: the number of views where it is observed on the surface.
- $\mathcal{F}$: the number of views where it is observed in front of the surface, i.e. in free space.
- O: the number of views in which it is observed at all.

Specifically, we project the voxel into each training view where we obtain a depth value d. Then, we increment S if the voxel's depth is approximately the same as d. We increment $\mathcal{F}$ if the voxels' depth is smaller than d. If the voxel is within the training camera's frustum, we increment O. Finally, we label the voxel as "inside" if any of the following conditions is met:

(1) $sS > \mathcal{F}$
(2) $\mathcal{F} < 4 \cdot \mathbb{1}[O > 40] + \mathbb{1}[40 \geq O > 6]$
(3) $S = 0$ and $\mathcal{F} = 0$
(4) $O < 2$

We multiply S with a number $s > 1$ to bias the reconstruction towards surfaces, which helps preserve thin structures. Weighing S and $\mathcal{F}$ equally leads to the erosion of objects, as voxels at object boundaries are not consistently sampled by the proposal MLP. Rule (2) is motivated by the fact that sometimes the proposal MLP misses an entire object altogether. In the unobserved interior of an object, S will be equal to zero in this case, but since $\mathcal{F}$ is nonzero, this leads to incorrectly labeling the voxel as outside. Rule (2) fixes this by requiring a minimum number of views that need to observe a voxel to lie in free space for it to labelled as outside. This threshold needs to be dependent on then number of views that observe a voxel, since otherwise sparsely observed voxels are always labelled as inside. Rule (3) ensures that completely unobserved parts of the scene are labelled as inside. The optional rule (4) ensures that only voxels are considered that are observed in at least two views, since parts of the scene that are only observed by a single camera are inherently underconstrained and thus only contribute geometric noise. As can be seen in Figure 3 of the main paper, this algorithm effectively removes floating artifacts, while preserving thin structures.

B SCALABLE MESH CONVERSION VIA CHUNKING

To capture thin structures with a surface-based approach, a very high grid resolution of 8192^3 is required. Processing such a high resolution grid in one pass requires too much memory. Fortunately, all of the steps (volumetric fusion, filtering, marching cubes, and simplification) in our pipeline can be executed in 1024^3 chunks, which allows scaling to arbitrary resolutions. To avoid discontinuities at chunk boundaries, we configure the mesh simplification algorithm to keep boundary vertices intact. The final mesh can then be computed by concatenating sub-meshes and merging the duplicate boundary vertices. To speed up volumetric fusion, we only process voxels that are sufficiently close to an initial estimate of the surface. We obtain this initial estimate of the surface by unprojecting the depth values contained in the depth maps of all input views. We then quantize the resulting 3D points based on grid resolution (8192^3), which gives us a list of surface voxels. We then subdivide the scene into 16^3 blocks and determine for each block whether it contains a voxel that is maximally $D = 64$ voxels apart from a voxel in the previously computed list of observed surface voxels. During fusion we skip blocks that are not marked as alive. Since we are no longer densely computing the implicit representation, it is no longer guaranteed that running marching cubes results in a hole-free mesh, which is the preferred input for most simplification algorithm. However, we find that with our choice of $D = 64$, only a moderate number of holes are introduced. These holes are usually not observed and therefore the quality of the reconstruction does not suffer. This technique also leads to slightly smaller meshes.

C IMPLEMENTATION DETAILS

Architecture. For the proposal MLPs and the MLP that predicts binary opacity values and view-dependent colors, we closely follow Zip-NeRF's [Barron et al. 2023] architecture based on a multi-resolution hash encoding [Müller et al. 2022]. To bound opacity and color values between 0 and 1, we use a sigmoid activation function. Following Reiser et al. [2023], during mesh appearance fitting, we predict the values of the triplanes and low-resolution voxel grid with a hash grid-equipped MLP. For this MLP, we use the same architecture as the MLP that parameterizes the binary opacity grid.

In unbounded scenes, regions that are only observed from far away can be represented with a low resolution. To achieve a resolution that smoothly decreases with the distance from the scene's center, we apply MERF's contraction function to each position $\mathbf{x}$ before querying the MLP:

$$\text{contract}(\mathbf{x})_j = \begin{cases} x_j & \text{if } \|\mathbf{x}\|_\infty \leq 1 \\ \frac{x_j}{\|\mathbf{x}\|_\infty} & \text{if } x_j \neq \|\mathbf{x}\|_\infty > 1 \\ \left(2 - \frac{1}{|x_j|}\right) \frac{x_j}{|x_j|} & \text{if } x_j = \|\mathbf{x}\|_\infty > 1 \end{cases} \quad (4)$$

where $\|\cdot\|_\infty$ is the L_∞ norm ($\|\mathbf{x}\|_\infty = \max_j |x_j|$). Before applying the contraction function, we scale input coordinates by a factor of 2.5 to allocate more representation power to the foreground. For the standalone voxel grid, we use a resolution of 2048^3. For the combination of triplane and low-resolution voxel grid, we use a resolution of 2048^2 and 512^3, respectively.

Optimization. For binary opacity grids, we use Adam [Kingma and Ba 2017] with an initial learning rate of 0.01, a final learning rate of 0.001 and 25K steps. For mesh appearance, we use Adam with an initial learning rate of 0.0005, a final learning rate of 0.00005 and

100K steps. The learning rate is warmed up for 2500 steps. For the binary entropy loss, we use a weight of 0.05.

Simplification. For mesh simplification, we need to specify a ratio R that controls what fraction of original triangles should be kept. We want to simplify the background more aggressively than the foreground, as we find it less important for accurate view synthesis. As detailed in the previous section, simplification is executed on a chunk-by-chunk basis. The scene is subdivided into an 8^3 grid of chunks. We define a chunk as lying in the background if its center lies outside of the $[-1, 1]^3$ unit cube. For foreground chunks, we set R to 0.03. We simplify backgrounds chunks twice as aggressively by setting R to 0.015. In addition, we make sure that background chunks contain at most 0.5M faces by adjusting R accordingly.

Visibility Culling. For visibility culling, we not only use the camera poses of the training images, but also generate 6 additional poses for each training pose by adding random offsets and rotations to the original pose. Let $\mathbf{o}$ be the origin of the training camera let $\mathbf{d}$ be the direction the training camera faces to. We obtain a new origin $\mathbf{o}$ by applying isotropic Gaussian noise. To obtain a new direction $\hat{\mathbf{d}}$, we use the E3X library [Unke and Maennel 2024] and draw a a uniform sample from an ϵ-neighborhood of the the direction vector:

$$\hat{\mathbf{o}} \sim \mathcal{N}(\mathbf{o}, \sigma^2 \mathbf{I})), \tag{5}$$

$$\hat{\mathbf{d}} \sim \mathcal{U}(\{\mathbf{v} \in \mathbb{R}^3 : ||\mathbf{v} - \mathbf{d}||_2 < \epsilon, ||\mathbf{v}||_2 = 1\}). \tag{6}$$

We find that the additional poses are crucial for avoiding visible holes in the final mesh.

D TEMPORAL ANTI-ALIASING

We implement our temporal anti-aliasing strategy [Yang et al. 2020] following industry best practices [Karis 2014]. Namely, we jitter the projection matrix with a Halton$(2, 3)$ sequence of length 16 and reproject the previous frame's color using the current depth buffer. We then average the reprojected color with the current frame's color using an exponentially moving average with a blend factor of 0.05. To reduce blur from repeated resampling, we use a Lanczos kernel [Duchon 1979] with a radius of 3 for reprojection. Finally, to limit ghosting artifacts for disoccluded content, we clip the reprojected color using variance-box [Salvi 2016] neighborhood clamping in the YCoCg color space [Karis 2014].

Since TAA is known to cause blur under motion, we evaluate the quality of our test set images with a moving camera. Given a target camera pose, we first extract its "up" vector $\mathbf{u}$ and its "left" vector $\mathbf{l}$. We then translate the camera from an initial position $\mathbf{p} + c(\mathbf{u} + \mathbf{l})$ to the target position $\mathbf{p}$ over a fixed number of frames $T = 100$:

$$\mathbf{p}(t) = \mathbf{p} + \lambda c(\mathbf{u} + \mathbf{l}) \tag{7}$$

where $\lambda = 1 - \frac{t}{T-1}$ and the time step t ranges from 0 to $T - 1$. The factor $c = 0.05$ controls how far the initial position is from the target position. The camera's rotation is kept fixed during the entire trajectory. The frame used for computing quality metrics is captured when the camera has arrived at the target position.

E FRAME RATE BENCHMARKING

We follow the evaluation protocol from Duckworth et al. [2023] and measure the average frame rate over the scene's test set camera poses. Following Duckworth et al. [2023], we render for each camera pose 100 frames and compute the average frame time. Similar to SMERF, for the browser-based viewer applications (BakedSDF [Yariv et al. 2023], MERF [Reiser et al. 2023] and our method), we measure frame rates that exceed the browser's frame rate limit by drawing each frame k times before scheduling it for display. Following SMERF, we measure frame times with three different values of k, where we choose the initial value for k to ensure that frame rate lies below 60 FPS. We then perform two additional measurements with larger values for k. Finally, for each frame, we report the minimum average frame time over k.