Volumetric Rendering with Baked Quadrature Fields

Gopal Sharma[1], Daniel Rebain[1], Kwang Moo Yi[1], Andrea Tagliasacchi[2, 3, 4]

[1] University of British Columbia, [2] Google DeepMind,
[3] Simon Fraser University, [4] University of Toronto
https://quadraturefields.github.io

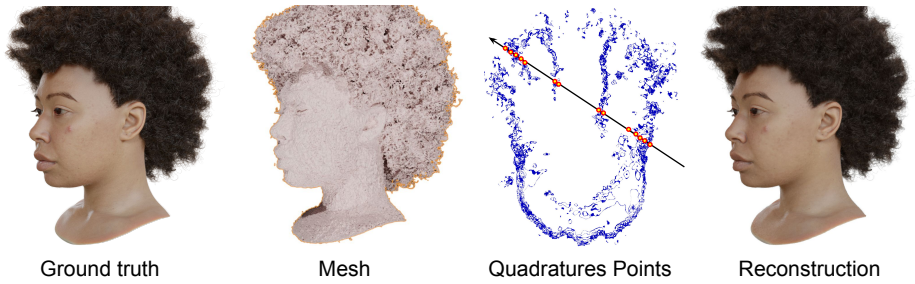

Fig. 1: We propose using textured polygons with NeRF to efficiently render non-opaque scenes, combining high-quality rendering with modern graphics hardware. To model a scene, we produce a mesh that gives quadrature points along a ray (shown as points on the intersection with the cross-section of the mesh) required in volumetric rendering.

Abstract. We propose a novel Neural Radiance Field (NeRF) representation for non-opaque scenes that enables fast inference by utilizing textured polygons. Despite the high-quality novel view rendering that NeRF provides, a critical limitation is that it relies on volume rendering that can be computationally expensive and does not utilize the advancements in modern graphics hardware. Many existing methods fall short when it comes to modelling volumetric effects as they rely purely on surface rendering. We thus propose to model the scene with polygons, which can then be used to obtain the quadrature points required to model volumetric effects, and also their opacity and colour from the texture. To obtain such polygonal mesh, we train a specialized field whose zero-crossings would correspond to the quadrature points when volume rendering, and perform marching cubes on this field. We then perform ray-tracing and utilize the ray-tracing shader to obtain the final colour image. Our method allows an easy integration with existing graphics frameworks allowing rendering speed of over 100 frames-per-second for a 1920×1080 image, while still being able to represent non-opaque objects.

Keywords: Novel View Synthesis · Neural Rendering · Neural Fields.

1 Introduction

Neural Radiance Fields (NeRFs) [30] have gained popularity by demonstrating impressive capabilities in generating photo-realistic novel views. They use a continuous volumetric function to represent a scene with a 5D implicit function that estimates the density and radiance for any position and direction. NeRF representations are trained to achieve multi-view colour consistency for a set of posed images. One of the main challenges to the widespread adoption of NeRF is the high computation cost. For example, the traditional implementation of NeRF involves a volumetric rendering algorithm that calculates the density and radiance by evaluating a large MLP at hundreds of sample positions along the ray for each pixel. This rendering process is too slow for interactive visualization without powerful GPUs.

Researchers have thus been exploring real-time rendering methods using voxel grids [21] and polygonal meshes [7, 54] to address the challenge of representing scene geometry in neural volumetric rendering. While MobileNeRF [7] and BakedSDF [54] have made progress in using *binary* opacities to restrict volumetric content to polygonal meshes they cannot represent *transparent* surfaces such as glass or clouds as they rely on a *single* point to render each ray, thus unable to represent anything other than hard surfaces. To overcome this limitation, multiple quadrature points need to be sampled along a ray within the neural volumetric rendering setup.[1] Towards that end, MERF [40] allows for fast rendering of large-scale scenes with a small memory footprint by utilizing a sparse feature grid and high-resolution 2D feature planes. They use a hybrid of planar and volumetric representations to model the scene but do not extract an explicit geometry that can enable downstream tasks in graphics and simulation. Thus, precisely and efficiently storing quadrature points needed for volumetric rendering is difficult and still an open problem. Partially circumventing this problem, Adaptive-Shells [50] enables the rendering of fur and hair by performing volumetric rendering within a shell defined around the surface extracted from a signed distance field. However, their inherent representation based on SDF can be limiting in glassy media where SDF fails to recover a correct surface.

In this work, our goal is to "bake" an existing NeRF model capable of approximating both solid and transparent objects, while still being capable of running in real-time. Toward this objective, we start from the key insight that mesh intersection-finding algorithms are highly efficient in modern graphics hardware. Thus, if we were to have a mesh that, for each ray, the intersection points of the ray with this mesh correspond to the points that are supposed to be used for volume rendering, then the rendering process can be made highly efficient. More specifically, we aim for a mesh that on hard surfaces creates a single intersection point with the ray thus imitating surface rendering, and at transparent surfaces that require volume rendering creates multiple intersection points, *e.g.*, through

[1] Note that simply using multiple quadrature points does not enable modeling complicated physics-based rendering such as refraction, but we limit ourselves in this work to what is possible strictly with volume rendering.

wrinkles. While the insight is straightforward, implementing a method that can find such a surface is non-trivial.

To implement this insight we propose learning an auxiliary neural field whose zero crossing surfaces induce a set of quadrature points for NeRF volumetric rendering – we name this field the *quadrature field*. We train the quadrature field so that it aligns with the *surface-field* [16] of the scene being represented; see Fig. 3. [2] With this field, we use marching cubes to extract the polygonal mesh, and for each ray, we use the intersection points with the mesh as quadrature points for volume rendering. To train this field, we use its gradients to encourage quadrature points to occur near the surfaces, as shown in the Fig. 1.

We evaluate our method on several datasets, including the Shelly dataset consisting of non-opaque shapes [50] and the MipNeRF-360 dataset of real scenes. To summarize, our contributions are as follows:

- our approach produces volumetric effects and successfully reconstructs non-solid objects while using only triangular mesh to place quadrature points;
- we show that by doing so, one can take full leverage of modern graphics hardware, achieving over 100 frames-per-second for rendering 1920×1080 images;
- we introduce a quadrature field and use it to train a neural field to extract quadrature points for both solid and transparent objects;
- our approach produces a compact representation of the scene and produces comparable results to the originally trained NeRF models.

2 Related works

Novel view synthesis has been studied extensively in the literature [45]. In this section, we review previous works with a focus on real-time rendering.

► **Light fields** When viewpoints are densely sampled, novel-view synthesis can be achieved through light field rendering [26]. Lumigraph [18] performs interpolation between observed rays for rendering novel views, but this approach demands significant memory and restricts camera movements. These challenges can be mitigated by utilizing optical flow [4] for image interpolation or by employing neural networks to represent light fields [1]. Multi-plane [11, 29, 37] and multi-sphere image representations [2] have demonstrated usefulness, although they still limit camera movement. However, in practical settings where observed viewpoints are not densely captured, reconstructing a 3D representation of the scene is crucial for rendering convincing novel views.

► **Mesh rendering (classical)** Traditional approaches to generating novel views utilize triangle meshes, typically reconstructed from point clouds via a multi-step process involving multi-view stereo [13], Poisson surface reconstruction [24], and marching cubes [28]. To create novel views, observed images are reprojected into each desired viewpoint and merged using either predetermined [5] or learned blending weights [22, 42, 43]. Although mesh-based representations are

[2] Surface fields apply to both solid and transparent objects.

suitable for real-time rendering, they often exhibit inaccurate geometry in regions with intricate details or complex materials, resulting in visible imperfections.

▶ **Mesh rendering (differentiable)** It is also possible to compute explicit triangle meshes through differentiable inverse rendering. For instance, DefTet [14] differentiably renders a tetrahedral grid, considering occupancy and colour at each vertex, and then composes the interpolated values along a ray. NVDiffRec [32] combines differentiable marching tetrahedra [44] with differentiable rasterization to perform full inverse rendering, allowing extraction of triangle meshes, materials, and lighting from images. While these approaches enable scene editing and relighting, they tend to compromise view synthesis quality.

▶ **Neural radiance fields (NeRF)** Neural radiance fields [30] learn 3D consistent scene representation using continuous opacities and radiance with the help of an MLP. This approach has produced excellent results for the novel-view synthesis of 3D objects with reflections [47], outdoor bounded scenes [3] and unbounded scenes [41]. However, as volumetric rendering produces pixel colour by evaluating the MLP over hundreds of points per ray, rendering speed is limited.

▶ **Efficient rendering of NeRFs** There have been several ways to speed up training and inference of NeRF. Early attempts include AutoInt [51], which models the radiance and opacity of a segment of ray instead of individual points and alpha composite the values over segments to get pixel colour, and DeRF [38], which combines multiple small NeRFs trained to represent disjoint spaces. Garbin *et al*. [15] introduced caching a factorized representation of neural radiance field for fast inference albeit at higher memory requirement for the cache. More recently, this problem has most commonly been addressed by trading off compute vs. storage by storing features into grids. These feature grids can be dense voxel grids [12], sparse voxel grids [21], multi-resolution hash grids [31], small MLPs distributed spatially [39] and low-rank tensor approximations of dense grids [6]. While in these methods features are converted into radiance/density by a small MLP, diffuse colours can also be stored on the grid and view-dependent radiance be represented by spherical harmonics [12,23,56]. Recently, Kerbl *et al*. [25] proposed learning a sparse set of 3D Gaussians to represent the scene, which allows skipping the empty regions of the scene easily and gives real-time rendering. While their approach can also reconstruct volumetric media, their primitives are disconnected, hindering several applications in graphics and simulations.

▶ **Baking neural features** Rather than accelerating the NeRF directly, one can also "bake" the neural features into polygonal meshes or volumetric textures. SNERG [21] proposed to store features in sparse volumetric textures, and volumetric ray marching combined with deferred rendering to generate pixel colours. However, this approach requires a large amount of GPU memory to store volumetric data. Recently, MERF [40] allows for fast rendering of large-scale scenes while utilizing smaller memory in comparison to SNERG by utilizing a sparse feature grid and high-resolution 2D feature planes. They use a hybrid of planar and volumetric representations to model the scene, which is orthogonal to our approach which is purely surface-based. In contrast, MobileNeRF [7] proposed using a classical triangular mesh for baking the neural features into a texture

map, and used *binary* opacities to optimize for a rasterized representation via volumetric rendering. BakedSDF [54] starts with VolSDF to learn a surface-based neural radiance field and use learned features baked at mesh vertices for real-time rendering. This produces smoother and cleaner meshes in comparison to MobileNeRF owing to SDF priors via Eikonal loss and leads to better performance. However, both MobileNeRF and BakedSDF are unable to represent transparent objects faithfully as both representations assume rays to terminate at the first intersection with a surface.

▶ **Modeling transparent scenes** αSurf [52] reconstructs 3D geometry of semi-transparent objects such as glass. Their approach is based on a field that is initialized using normalized values of volumetric density, in comparison to our approach which is based on the surface field derived from volumetric weights. Their approach extracts faithful surfaces for transparent scenes but they do not explore the accurate placement of these surfaces for novel-view synthesis applications. NEMTO [48] models transparent objects by extracting the geometry of the object using a DeepSDF [34] based approach and using another network to model bending of the ray through transparent media. Their reliance on the SDF-based representation prevents them from modeling volumetric effects like hair and fur, which our models can model easily as shown in Sec. 4. Most recently, Adaptive-shells [50] performs sample efficient volumetric rendering by modifying the NeUS [49] representation by predicting the scale parameter to adapt to the local volumetric media. The scale is used to extract a shell of triangular mesh which constrains the region where volumetric rendering is done with the help of an Instant-NGP [31] backbone. Their reliance on an SDF as the intrinsic representation can be limiting in approximating a glass-like media. In contrast, we extract the mesh using a general quadrature field (see Sec. 3.2) and define volumetric rendering by finding the quadrature points via ray-mesh intersection. This allows our approach to represent glass-like objects as shown in the Fig. 5.

3 Method

Given a set of posed images, our goal is to create a compact 3D representation of the scene that allows fast rendering. Similarly to MobileNeRF [7], the representation consists of a triangular mesh and a texture map consisting of neural features and continuous opacities. Our rendering process consists of two steps:

1. We use ray tracing to render a transparent scene (i.e. continuous opacities) by alpha compositing colour at points of intersection of the ray and the triangular mesh; see 3.4.
2. We render view-dependent effects via spherical Gaussian lobes stored in the texture-map;

This representation is created in four-stages (also illustrated in the Fig. 2):

1. *Training the NeRF.* We train a NeRF model with continuous opacities in which quadrature points are sampled using importance sampling (Sec. 3.1).

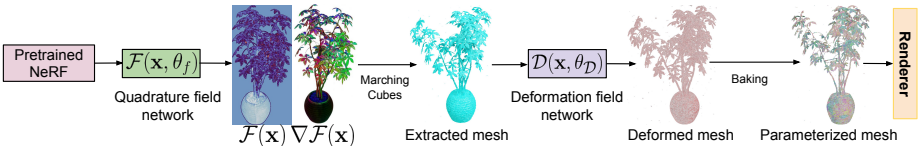

Fig. 2: Overview of our pipeline. We start with a pre-trained network to train a quadrature field that learns the placement of quadrature points. The extracted mesh from the quadrature field is fine-tuned using a deformation field (deformation is shown using red colour on the deformed mesh). Lastly, the neural features are baked into a texture map and the mesh, which can be rendered using ray-tracing.

2. *Training the quadrature-field.* We train the quadrature field network with the help of NeRF. We use the trained quadrature field to extract a mesh (Sec. 3.2).
3. *Fine-tuning.* We fine-tune the mesh vertices and NeRF with a network that produces the deformation field (3.3).
4. *Baking.* We extract the neural features on the surface of the mesh and bake these features into a texture-map (Sec. 3.4).

3.1 Training the NeRF

Mildenhall *et al.* [30] introduced a 3D scene representation consisting of an MLP with trainable parameters $\boldsymbol{\theta}$ that takes a position $\mathbf{x} \in \mathbb{R}^3$ and a direction vector $\mathbf{d}$ and outputs radiance $\mathbf{c}(\mathbf{x}, \mathbf{d})$ and density $\sigma(\mathbf{x})$. Given a camera pose, the pixel colour is computed using volumetric integration along a ray $\mathbf{r} = (\mathbf{o}, \mathbf{d})$ which is sampled at quadratures $\{t_i\}$ inducing a set of spatial samples $\mathbf{x}_i = \mathbf{o} + t_i\mathbf{d}$:

$$\mathbf{C}(\mathbf{r}; \boldsymbol{\theta}) = \sum_i w_i \cdot c_i \tag{1}$$

where $w_i = \alpha_i \Pi_{j<i} \exp(1 - \alpha_i)$ is the weight given to a sampled point, $\alpha_i = 1 - \exp(-\sigma(\mathbf{x}_i)\delta_i)$ is the opacity and $\delta_i = t_{i+1} - t_i$. The MLP parameters are optimized by minimizing the difference between the predicted ray colour and the ground truth ray colour, specifically:

$$\mathcal{L}(\mathbf{r}; \boldsymbol{\theta}) = \|\mathbf{C}(\mathbf{r}; \boldsymbol{\theta}) - \mathbf{C}_{gt}(\mathbf{r})\|^2 \tag{2}$$

In this work, we employ the Instant-NGP [31] variant of NeRF for training.

3.2 Training the quadrature field

The value w_i in Eq. (1) represents the weight given to the quadrature point i; the higher the weight, the more likely light traveling along the direction d hits the point $\mathbf{x}_i$. In this sense, w_i can be seen as (view-dependent) *surfaceness*, as described by Goli *et al.* [16]. While for solid surfaces a single quadrature point should be sufficient to approximate the rendering equation, for non-solid objects more than one quadrature point is needed. Traditional NeRF models sample a

fixed number of quadrature points from the probability density function w to approximate integration via Eq. (1). However, it is not clear how to determine these quadrature points deterministically. In this work, we propose a deterministic way to find quadrature points for *all surface types*. We emphasize here that we are interested in *multiple* quadrature points along the ray to enable volumetric effects, unlike those that only allow a single point [7, 54].

▶ **Defining the quadrature field** We seek to find a field that *concentrates* quadrature points in regions where surfaces are more likely to occur.[3] We take inspiration from parameterization literature [20], as well as from methods that exploit the zero crossing of a signed distance field to define quadrature points [33, 49, 53], and define our *quadrature field* as:

$$\mathcal{Q}(\mathbf{x}) = \sin(\omega\, \mathcal{F}(\mathbf{x}; \boldsymbol{\theta}_{\mathcal{F}})) \tag{3}$$

$$\text{where } \mathcal{F}(\mathbf{x}; \boldsymbol{\theta}_{\mathcal{F}}) : \mathbb{R}^3 \to \mathbb{R} \tag{4}$$

and ω is a hyperparameter that controls the frequency of zero-crossings as shown in Fig. 3. Our quadrature points are then defined by the intersection of a ray and the zero crossings of $\mathcal{Q}$. Note the field $\mathcal{Q}$ is only a function of position, as the quadrature points will be represented as a surface mesh, whose geometry does not change according to a viewpoint. To train the quadrature field, we optimize the parameters $\boldsymbol{\theta}_{\mathcal{F}}$ of the function $\mathcal{F}$, and create quadrature points that approximate the volume-rendering integral along a given ray.

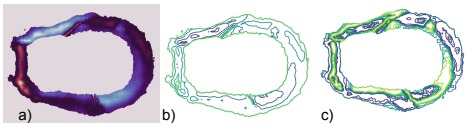

Fig. 3: Effect of omega on quadrature field. a) quadrature field along a cross-section of a shape, b) zero-crossings of the quadrature field at $\omega = 1$ and c) at $\omega = 50$. The higher values of omega leads to more zero-crossings.

▶ **Training the quadrature field** For quality rendering, the field q should have more zero-crossings in the region where the weight function w attains higher values. To fulfill this objective, we make two simple observations:

- Assuming local linearity, the number of zero-crossings of Eq. (3) will be proportional to the gradient of Eq. (4);
- As the weight function w in Eq. (1) is a *view-dependent* quantity, we can only supervise the *directional* derivative of Eq. (4).

Putting these two observations together, the constraint $\nabla f(\mathbf{x}; \boldsymbol{\theta}_{\mathcal{F}}) \cdot \mathbf{d} \approx w$ emerges, which we approximately satisfy via the following loss:

$$\mathcal{L}_f(\mathbf{x}; \boldsymbol{\theta}_{\mathcal{F}}) = ||\nabla \mathcal{F}(\mathbf{x}; \boldsymbol{\theta}_{\mathcal{F}}) \cdot \mathbf{d}| - \max(w(\mathbf{x}, \mathbf{d}), w(\mathbf{x}, -\mathbf{d}))| \tag{5}$$

where note the function w is non-optimized and instead kept fixed.

[3] Note that this is not restricted to solids, but also transparent surfaces that require many quadrature points per ray to be accurately represented.

As $w(\mathbf{x}, d)$ is a function of direction, whereas $\nabla f(\mathbf{x})$ is direction independent, we force the field f to vary most in the direction for which $w(\mathbf{x}, \pm d)$ is the largest; see Fig. 4. We implement $\mathcal{F}$ as an MLP on top of a hash-grid [31].

An alternative would be to take maximum along all directions as is suggested in nerf2nerf [17, Eq.10], but it would require an impractical amount of compute for training. MobileNeRF instead initializes a pruning grid to zero in [7, Eq.10] and uses surface-field as a lower-bound. In our case, this entails initializing the network as $\nabla f(\mathbf{x}) = 0 \; \forall \mathbf{x}$, which is harder to achieve. Our formulation, Eq. (5), provides simple and stable optimization.

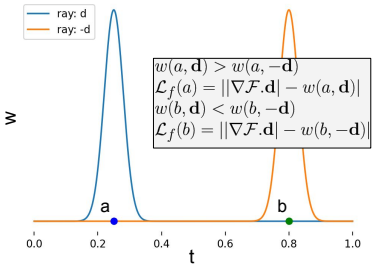

Fig. 4: Quadrature field loss. For a particular point, the quad field is supervised to predict the directional gradient to be equal to the maximum weight between the bi-directions (Eq. (5)).

Finally, to encourage *sparse* creation of surfaces, thus fewer quadratures to rasterize, we have explored the use of an additional ℓ_1 regularization. We observed that the pruning data structure in Instant-NGP [31] is sufficient to remove surfaces from low-density regions. After learning the quadrature field, we extract the quadrature mesh $\mathcal{M}$ via marching cubes on the function $\mathcal{Q}$.

3.3 Fine-tuning

While Eq. (5) can supervise the density of quadrature points, their precise location should be derived by directly optimizing for photometric reconstruction losses. One way to achieve this would be to fine-tune the vertices of the mesh and the original NeRF directly, as in [7]. However this saturates GPU memory and, in our experience, leads to non-smooth optimization. Instead, we employ a vector field to represent deformations continuously in space:

$$\mathcal{D}(\mathbf{x}; \boldsymbol{\theta}_{\mathcal{D}}) : \mathbb{R}^3 \rightarrow \mathbb{R}^3 \tag{6}$$

and use a perturbed set of quadrature positions $\{\tilde{\mathbf{x}}_i\}$ to evaluate photometric reconstruction via Eq. (1), where perturbation is restricted to happen *along* the ray direction $\mathbf{d}$:

$$\tilde{\mathbf{x}}_i = \mathbf{x}_i + \delta(\mathbf{x}_i) \quad \delta(\mathbf{x}_i) = \kappa \, \mathbf{d} \cdot \mathcal{D}(\mathbf{x}_i; \boldsymbol{\theta}_{\mathcal{D}}) \, \mathbf{d} \tag{7}$$

where a hyperbolic tangent is used as the final activation function in the $\mathcal{D}$, together with κ, to limit the perturbations within the spatial support of the marching cube mesh and thus stabilize training. We implement the deformation field using an MLP on top of a hash-grid [31]. Given the deformation field, a mesh vertex $\mathcal{V}_i$ is updated as:

$$\mathcal{V}_i \leftarrow \mathcal{V}_i + \mathbb{E}_c[w_{i,c}(\kappa \mathbf{d}_c \cdot \mathcal{D}(\mathcal{V}_i; \boldsymbol{\theta}_{\mathcal{D}}))\, \mathbf{d}_c] \; / \; \mathbb{E}_c[w_{i,c}] \tag{8}$$

where c indices over training views, and the perturbation is weighted by the weight $w_{i,c}$ on the basis of how much a view c affects the given vertex $\mathcal{V}_i$.

► **Training loss and regularization** Training is done by defining photometric loss with perturbed quadrature points as:

$$\mathcal{L}_{\text{def}}(\mathbf{r}, \boldsymbol{\theta}, \boldsymbol{\theta}_{\mathcal{D}}) = \| \sum_i w(\tilde{\mathbf{x}}_i, \mathbf{r}) c(\tilde{\mathbf{x}}_i) - \mathbf{C}_{gt}(\mathbf{r}) \|^2 \tag{9}$$

We jointly optimize the parameters of NeRF ($\boldsymbol{\theta}$) and the deformation field ($\boldsymbol{\theta}_{\mathcal{D}}$). We also encourage $\boldsymbol{\theta}_{\mathcal{D}}$ to be smooth by minimizing the norm of deformation, and by encouraging the deformation to be smooth for each triangle $\mathcal{T}$:

$$\mathcal{L}_{\text{reg}}(\boldsymbol{\theta}_{\mathcal{D}}) = \mathbb{E}_{\mathcal{T} \in \mathcal{M}} \mathbb{E}_{(\mathbf{x}_a, \mathbf{x}_b) \in \mathcal{T}} \|\mathcal{D}(\mathbf{x}_a; \boldsymbol{\theta}_{\mathcal{D}})\|_2^2 + \|\mathcal{D}(\mathbf{x}_a; \boldsymbol{\theta}_{\mathcal{D}}) - \mathcal{D}(\mathbf{x}_b; \boldsymbol{\theta}_{\mathcal{D}})\|_2^2$$

► **Training implementation** We employ block coordinate descent, where we first optimize the deformation field till convergence, and then we update the mesh vertices. In order to update vertices in Eq. (8), we perform a sweep over all training views to compute w_i. We do not change the topology of the mesh during the above process, as we assume that the extracted mesh from the quadrature field is a good approximation. Note that fine-tuning allows our approach to adapt to fewer quadrature points per ray while retaining higher reconstruction quality, as shown in the Tab. 5.

3.4 Baking and rendering neural features

After fine-tuning, we now prepare the triangular mesh and the texture maps that we ultimately use to render. We start by post-processing the mesh to remove surfaces that are not visible from training views – these are likely artifacts as they were never "seen". We further remove the surfaces for which the maximum volumetric weight w across all training views is below a threshold. We then construct the texture map by first parameterizing the mesh using a publicly available library [55]. We provide more detail on parameterization in the supplementary material.

We implement our real-time rendering using highly optimized Nvidia Optix [35] library for ray-tracing on an RTX GPU. For compatibility with Optix ray-tracing pipeline, as well as general efficiency, we compress the representation of colour, alpha, and spherical Gaussian parameters to 8-bit texture maps. Specifically, we use a sigmoid transformation to bring unbounded RGB coefficients to a $[0, 1]$ range, represent spherical Gaussian lobe axes as 8-bit azimuth and elevation angles, and compress lambda values using a logarithmic mapping. These quantization results in minimal loss in performance as shown in Tab. 5.

3.5 Implementation details

We implement our NeRF using Instant-NGP, where we use $3-6$ spherical Gaussians depending on the dataset for view-dependent effects. We use the Nerfacc [27] library based on Pytorch [36], as it provides stable training at mixed

Fig. 5: Our approach can represent the transparency of a glassy object (Objaverse dataset [9]) achieving 30.6 PSNR whereas MobileNeRF fails with only 24.6 PSNR. To effectively test transparency, the shape is also rendered against a textured background. Please zoom-in to see details.

Table 1: Quantitative performance on Shelly dataset [50]. We compare our approach with various real-time approaches based on baking neural features. We also report average number of samples used per-ray to render a pixel.

Methods	Avg. # samples	Avg. PSNR	Fernvase	Pug	Woolly	Horse	Khady	Kitten
Instant-NGP [31]	7.13	40.13	40.99	38.75	38.66	47.05	33.00	42.31
GS [25]	-	38.30	39.05	38.63	34.39	45.16	31.73	40.82
Adaptive shells [50]	1.74	36.02	36.47	35.83	34.19	40.57	31.22	37.82
Ours (fine-tuned)	2.40	37.29	37.56	36.35	35.21	42.04	32.45	40.15
MobileNeRF [7]	<1	31.62	31.38	31.50	31.61	36.48	26.84	31.93
Ours (baked)	2.40	35.13	35.85	34.42	31.96	38.67	31.64	38.25

precision with comparable performance as the original Instant-NGP paper. For real scenes, we use contraction mapping [3] to train the NeRF. For extracting meshes, we use a 1024 sized voxel grid for synthetic scenes and a 2048 sized voxel grid for real scenes. We further add meshes extracted from the density field of the NeRF with the mesh extracted from our quadrature field, which helps in avoiding holes like artifacts. We use $\omega = 100$ for synthetic scenes and $\omega = 10$ for real scenes. We allow maximum of 25 ray-triangle intersection for synthetic scenes and 15 for real scenes. We use super-sampling at twice the resolution to achieve anti-aliasing. We ablate this choice in the Tab. 5 and provide more information in the Supplementary material. Our code is available at https://quadraturefields.github.io.

4 Experiments

4.1 Novel-view synthesis

We design our experiments to showcase superior reconstruction quality in real-time on a variety of scenes. To showcase the ability of our approach to reconstruct non-opaque shapes we experiment with the Shelly dataset [50] with 6 360-degree scenes with emphasis on fuzzy surfaces with complex geometry and transparency. We also use the NeRF-Synthetic dataset consisting of 8 synthetic 360-degree scenes [30]. We also experiment with a more challenging real-world dataset consisting of 7 scenes from MipNeRF 360 [3]. We focus on comparing with previous works that extract meshes to constrain quadrature points used in volumetric rendering. We compare with Mobile-NeRF and Baked-SDF that bake neural features into a mesh. We also compare with Adaptive-shells [50] which produces volumetric effects by sampling quadrature points within a shell defined

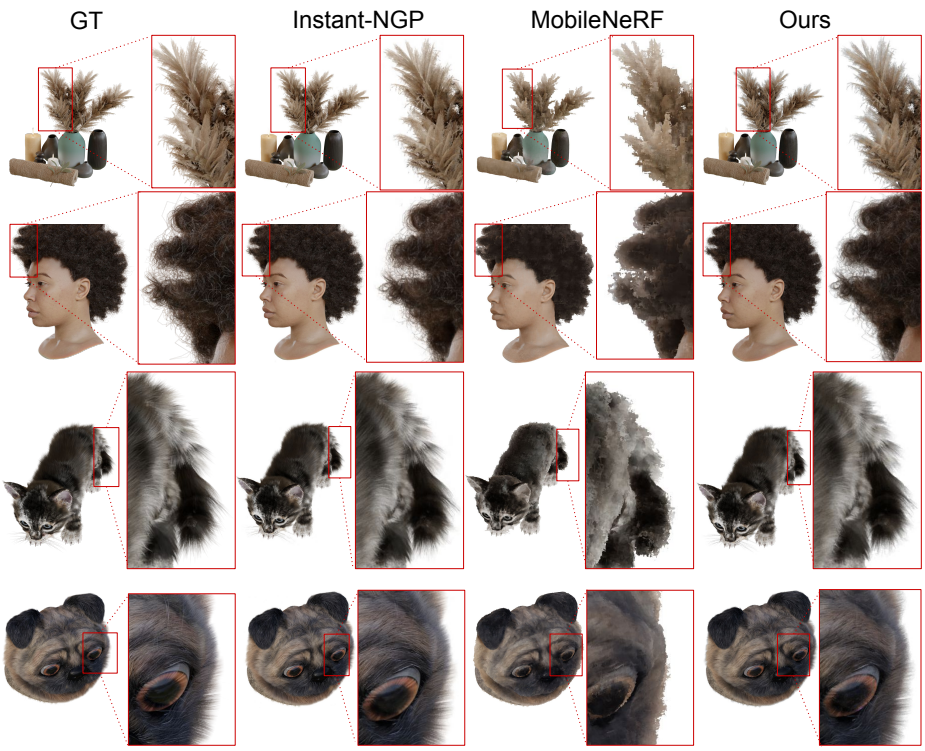

Fig. 6: Qualitative results on Shelly-dataset. Our real-time rendering approach results in a detailed reconstruction of non-opaque objects. Please zoom in to see details.

by a mesh. We perform qualitative comparisons with the baselines if their final renderings or code-base are publicly available. We evaluate the performance of our method using Peak Signal to Noise Ratio (PSNR). Other metrics such as Learned Perceptual Image Patch Similarity (LPIPS) and Structural Similarity Index (SSIM) are provided in the Supp. material. We also report the average number of samples per ray to highlight the efficiency of a method in rendering a pixel. We provide ablations showing the effect of our design choices in Tab. 5.

► **Experiments on non-opaque Shapes** We experiment with the Shelly dataset [50] introduced by Wang *et al.* It covers a wider variety of appearances including fuzzy surfaces such as hair, fur, and foliage rendered at 1920×1080 resolution. Our approach faithfully reconstructs thin structures and transparency as shown in Fig. 6. Though our approach loses performance because of the baking process, it still produces quantitatively similar results to Adaptive shells as shown in Tab. 1. To further stress test our approach, we experiment with a transparent object from the Objaverse dataset [9] in the Fig. 5. MobileNeRF fails to reconstruct thin structures and completely fails to reconstruct the transparent shape, shown in the Fig. 6 and Fig. 5.

► **NeRF synthetic dataset** Our work produces triangular meshes using the

Table 2: Quantitative performance on NeRF synthetic dataset. We compare our approach with various real-time approaches that are based on baking neural features using PSNR metric. We also report average # samples to render a pixel.

Methods	Avg. # samples	Avg. PSNR	Lego	Chair	Ship	Mic	Drums	Mat.	Ficus	Hotd.
Instant-NGP [31]	7.95	33.16	35.84	35.54	30.71	36.58	25.60	29.58	33.98	37.48
GS [25]	-	33.31	35.78	35.83	30.80	35.36	26.15	30.00	34.87	37.72
SNeRG [21]	-	30.38	33.82	33.24	27.97	32.60	24.57	27.21	29.32	34.33
VMesh [19]	-	30.70	-	-	-	-	-	-	-	-
MobileNeRF [7]	<1	30.90 ●	34.18	34.09	29.06	32.48	25.02	26.72	30.20	35.46
Adaptive-shells [50]	3.53	31.84 ●	33.49	34.94	29.54	33.91	25.19	27.82	33.63	36.21
Ours	1.69	31.00 ●	32.89	33.48	28.83	33.70	25.33	27.91	32.24	33.57

Table 3: Qualitative evaluation on mip-NeRF 360 dataset. We report reconstruction quality (PSNR metric). We also report average # samples to render a pixel.

Method	Avg.# samples	Mean-indoor	Kitchen	Room	Bonsai	Counter	Mean-outdoor	Garden	Bicycle	Stump
Instant-NGP [31]	13.03	29.67	29.67	30.60	31.14	27.26	24.58	25.68	23.32	24.75
GS [25]	-	30.41	30.32	30.63	31.98	28.70	26.40	27.41	25.25	26.55
MobileNeRF [7]	<1	-	-	-	-	-	23.06	23.54	21.70	23.95
BakedSDF [54]	<1	27.06 ●	26.72	28.68	27.17	25.69	23.52 ●	24.94	22.04	23.59
Adaptive Shells [50]	17.41	29.19 ●	28.43	30.63	32.47	25.24	23.17 ●	25.35	22.19	21.96
Ours	5.15	28.13 ●	28.52	28.81	28.87	26.30	24.12 ●	25.54	22.93	23.90

quadrature field that is based on the surface field (Sec. 3.2). An alternative approach to generate meshes would be to use Delaunay Triangulation on the surface field. This approach produces more vertices of tetrahedra in regions where the surface is likely to occur. The mesh reconstructed using Delaunay triangulation results in a bad representation of the underlying surface as is shown in the Supplementary material. We further compare with other methods that propose baking neural features into a volumetric representation such as voxels (SNeRG [21]), and meshes (MobileNeRF [7]) and hybrid volumetric and mesh representation (VMesh [19]). We report the results on the NeRF synthetic dataset in the Tab. 2.

▶ **MipNeRF 360 dataset** We further evaluate our approach with a real-world dataset and compare it with surface-based representation (MobileNeRF and BakedSDF) and hybrid volumetric rendering (Adaptive-shells [50]). Tab. 3 shows the quantitative performance of our approach and comparison with the baselines. Fig. 7 shows the qualitative performance of our approach for different real scenes. Our approach outperforms the surface-based baselines in outdoor scenes and is competitive with the baselines in indoor scenes. Our approach performs similarly to Adaptive-shells while using only a third of the samples to render images. Fig. 8 shows a comparison with surface-based methods – BakedSDF and MobileNeRF produce incomplete reconstruction of transparent objects.

4.2 Run-time performance

We evaluate our run-time performance using the CUDA Optix library running on an Nvidia RTX 3090 GPU in the Tab. 4. We render the images at 800×800 resolution for NeRF synthetic dataset and 1920×1080 for MipNeRF and Shelly dataset. Our approach runs at an interactive speed or better across all datasets despite using more than one samples per ray. We also implement rendering using the depth peeling algorithm [10] to efficiently render in a browser. These experiments are discussed in the Supplementary material.

Fig. 7: Visualization of Mip-NeRF 360 dataset. Top: GT, **Bottom:** Ours.

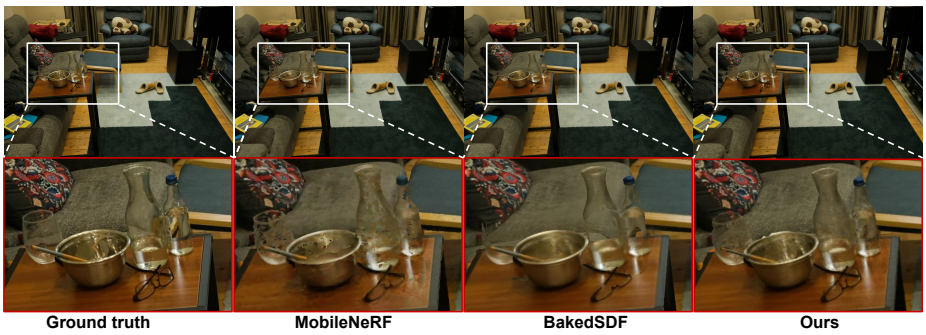

Fig. 8: Comparison with BakedSDF [54] and MobileNeRF [7]. The bottom row shows a zoomed-in reconstruction of transparent objects.

4.3 Ablations

We ablate our algorithms in Tab. 5 and observe the following:

- **Number of spherical-gaussian lobes:** Increasing spherical lobes improves reconstruction of view-dependent effects, albeit at the cost of rendering.
- **Include mesh from density field:** Often including coarse mesh extracted from the density fields complements the mesh extracted from the quadrature fields and helps fill the holes. We provide visualization in the Supp. material.
- **Effect of omega:** Larger omega leads to more quadrature points, which better captures the volumetric effects.
- **Effect of fine-tuning:** Fine-tuning aligns the mesh with the NeRF density field improving the reconstruction.
- **Size of texture map:** Increasing texels per triangle improves reconstruction but at a slight cost of speed.
- **Size of mesh:** A higher resolution mesh gives better reconstruction quality at the cost of rendering speed.
- **Super-sampling:** We can achieve anti-aliasing using super-sampling of each pixel resulting in better reconstruction quality at the cost of rendering speed.

Table 4: Run-time comparison across different datasets. We report our run-time at 1 and 2× super-sampling.

		NeRF synthetic		**Shelly**		**MipNeRF 360**	
Method	Framework	FPS	Watts	FPS	Watts	FPS	Watts
GS	CUDA	393	340	403	340	134	300
BakedSDF	WebGL	-	-	-	-	72	85
MobileNeRF	WebGL	744.5	250	455	340	280	250
Adaptive Shells	CUDA	281	450	263	450	36	450
Ours 1× (Nvidia 3090)	CUDA	570	340	323	340	140	340
Ours 2× (Nvidia 3090)	CUDA	167	340	93	340	35	340

Table 5: Ablations. We ablate our proposed approach at three stages using 1) a different number of spherical Gaussian (SG) lobes, 2) whether we use density mesh along with mesh extracted from the quadrature field, 3) the effect of fine-tuning, 4) the use of continuous rendering loss, 5) effect of ω used for mesh extraction, 6) effect of quantization (baking), 7) size of the mesh, 9) finally the different sizes of texture map used for baking neural features and the effect of super-sampling on reconstruction quality and speed. We use the Mic scene from the synthetic dataset to ablate.

			PSNR				**PSNR**	**FPS**
I	Full model	Instant-NGP	36.58	III	Quantization	Ours w/o quant.	34.06	-
II	Finetune	No finetune	31.05			Ours w/ quant. (baked)	33.70	210
	SG lobes	Finetune (w/ 3 SG)	33.68		Mesh size	Mesh w/ 880k faces	33.70	210
		Finetune (w/ 6 SG)	34.06			Mesh w/ 3.3 mil faces	33.92	180
	Mesh Extraction	Finetune w/ density mesh	30.37		Texture size	Baked w/ 4096 text. map	33.11	230
		Finetune with quad mesh	33.40			Baked w/ 8192 text. map	33.70	210
		Finetune with omega=10	31.09		Super-sampling	1x	32.05	750
	Loss	Finetune w/o cont. loss	34.00			2x	33.70	210

5 Conclusion

Our research addresses a critical limitation of the NeRF representation by introducing a novel approach that leverages textured polygons with continuous opacity and encodes feature vectors, enabling rapid rendering and integration into standard graphics pipelines. By training a specialized field to identify quadrature points and utilizing a novel gradient-based loss function, we achieve a quality mesh suitable for interactive rendering on desktops. Our method retains the ability to handle scenes featuring transparent objects, enhancing its practical applicability and potential impact in computer vision and graphics domains.

► **Limitations** Our method is bound by the limitations of NeRF; extending quadrature fields to more general rendering techniques would be interesting. Dealing with thin surfaces poses a challenge, as rarely, our method may miss thin surfaces when limited in capacity; recent developments for better quadrature for NeRFs [46] might be helpful. For large-scenes, reducing the memory footprint could be interesting to enable extremely low-end devices.

Acknowledgements

The authors would like to thank Sara Sabour for helping to create a dataset. This work was supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC) Discovery Grant, NSERC Collaborative Research

and Development Grant, Google, Digital Research Alliance of Canada, and Advanced Research Computing at the University of British Columbia.

References

1. Attal, B., Huang, J.B., Zollhöfer, M., Kopf, J., Kim, C.: Learning neural light fields with ray-space embedding networks. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (2022) 3
2. Attal, B., Ling, S., Gokaslan, A., Richardt, C., Tompkin, J.: MatryODShka: Real-time 6DoF video view synthesis using multi-sphere images. In: European Conference on Computer Vision (ECCV) (Aug 2020), https://visual.cs.brown.edu/matryodshka 3
3. Barron, J.T., Mildenhall, B., Verbin, D., Srinivasan, P.P., Hedman, P.: Mip-nerf 360: Unbounded anti-aliased neural radiance fields. CVPR (2022) 4, 10, 22
4. Bertel, T., Yuan, M., Lindroos, R., Richardt, C.: Omniphotos: Casual 360° vr photography. ACM Trans. Graph. (2020) 3
5. Buehler, C., Bosse, M., McMillan, L., Gortler, S., Cohen, M.: Unstructured Lumigraph Rendering (2023) 3
6. Chen, A., Xu, Z., Geiger, A., Yu, J., Su, H.: Tensorf: Tensorial radiance fields. In: European Conference on Computer Vision (ECCV) (2022) 4
7. Chen, Z., Funkhouser, T., Hedman, P., Tagliasacchi, A.: Mobilenerf: Exploiting the polygon rasterization pipeline for efficient neural field rendering on mobile architectures. arXiv preprint arXiv:2208.00277 (2022) 2, 4, 5, 7, 8, 10, 12, 13, 21, 22
8. Clevert, D.A., Unterthiner, T., Hochreiter, S.: Fast and accurate deep network learning by exponential linear units (elus). Under Review of ICLR2016 (1997) (11 2015) 22
9. Deitke, M., Liu, R., Wallingford, M., Ngo, H., Michel, O., Kusupati, A., Fan, A., Laforte, C., Voleti, V., Gadre, S.Y., VanderBilt, E., Kembhavi, A., Vondrick, C., Gkioxari, G., Ehsani, K., Schmidt, L., Farhadi, A.: Objaverse-xl: A universe of 10m+ 3d objects. arXiv preprint arXiv:2307.05663 (2023) 10, 11
10. Everitt, C.: Interactive order-independent transparency (2001), https://www.gamedevs.org/uploads/interactive-order-independent-transparency.pdf, accessed on November 12, 2023 12, 19
11. Flynn, J., Broxton, M., Debevec, P., DuVall, M., Fyffe, G., Overbeck, R., Snavely, N., Tucker, R.: Deepview: View synthesis with learned gradient descent. In: 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 2362–2371 (2019). https://doi.org/10.1109/CVPR.2019.00247 3
12. Fridovich-Keil and Yu, Tancik, M., Chen, Q., Recht, B., Kanazawa, A.: Plenoxels: Radiance fields without neural networks. In: CVPR (2022) 4
13. Furukawa, Y., Hernández, C.: Multi-view stereo: A tutorial. Found. Trends. Comput. Graph. Vis. p. 1–148 (jun 2015) 3
14. Gao, J., Chen, W., Xiang, T., Tsang, C.F., Jacobson, A., McGuire, M., Fidler, S.: Learning deformable tetrahedral meshes for 3d reconstruction. In: Advances In Neural Information Processing Systems (2020) 4
15. Garbin, S.J., Kowalski, M., Johnson, M., Shotton, J., Valentin, J.P.C.: Fastnerf: High-fidelity neural rendering at 200fps. 2021 IEEE/CVF International Conference on Computer Vision (ICCV) pp. 14326–14335 (2021), https://api.semanticscholar.org/CorpusID:232270138 4

16. Goli, L., Rebain, D., Sabour, S., Garg, A., Tagliasacchi, A.: nerf2nerf: Pairwise registration of neural radiance fields (2022) 3, 6
17. Goli, L., Rebain, D., Sabour, S., Garg, A., Tagliasacchi, A.: nerf2nerf: Pairwise registration of neural radiance fields. In: International Conference on Robotics and Automation (ICRA) (2022) 8
18. Gortler, S.J., Grzeszczuk, R., Szeliski, R., Cohen, M.F.: The lumigraph. SIGGRAPH '96 (1996) 3
19. Guo, Y.C., Cao, Y.P., Wang, C., He, Y., Shan, Y., Qie, X., Zhang, S.H.: Vmesh: Hybrid volume-mesh representation for efficient view synthesis (2023) 12
20. He, L., Schaefer, S., Hormann, K.: Parameterizing subdivision surfaces. ACM Trans. Graph. (2010) 7
21. Hedman, P., Srinivasan, P.P., Mildenhall, B., Barron, J.T., Debevec, P.: Baking neural radiance fields for real-time view synthesis (2021) 2, 4, 12
22. Hedman, P., Philip, J., Price, T., Frahm, J.M., Drettakis, G., Brostow, G.: Deep blending for free-viewpoint image-based rendering. ACM Trans. Graph. (2018) 3
23. Karnewar, A., Ritschel, T., Wang, O., Mitra, N.: Relu fields: The little non-linearity that could. In: ACM SIGGRAPH 2022 Conference Proceedings. SIGGRAPH '22, Association for Computing Machinery, New York, NY, USA (2022). https://doi.org/10.1145/3528233.3530707, https://doi.org/10.1145/3528233.3530707 4
24. Kazhdan, M., Bolitho, M., Hoppe, H.: Poisson Surface Reconstruction. In: Sheffer, A., Polthier, K. (eds.) Symposium on Geometry Processing. The Eurographics Association (2006). https://doi.org/10.2312/SGP/SGP06/061-070 3
25. Kerbl, B., Kopanas, G., Leimkühler, T., Drettakis, G.: 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/ 4, 10, 12, 21, 22
26. Levoy, M., Hanrahan, P.: Light field rendering. In: Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques. SIGGRAPH '96 (1996) 3
27. Li, R., Gao, H., Tancik, M., Kanazawa, A.: Nerfacc: Efficient sampling accelerates nerfs. arXiv preprint arXiv:2305.04966 (2023) 9
28. Lorensen, W.E., Cline, H.E.: Marching cubes: A high resolution 3d surface construction algorithm. SIGGRAPH Comput. Graph. p. 163–169 (aug 1987) 3
29. Mildenhall, B., Srinivasan, P.P., Ortiz-Cayon, R., Kalantari, N.K., Ramamoorthi, R., Ng, R., Kar, A.: Local light field fusion: Practical view synthesis with prescriptive sampling guidelines. ACM Transactions on Graphics (TOG) (2019) 3
30. Mildenhall, B., Srinivasan, P.P., Tancik, M., Ramamoorthi, R., Ng, R.: Nerf: Representing scenes as neural radiance fields for view synthesis. In: ECCV (2020) 2, 4, 6, 10
31. Müller, T., Evans, A., Schied, C., Keller, A.: Instant neural graphics primitives with a multiresolution hash encoding. ACM TOG (2022) 4, 5, 6, 8, 10, 12, 21, 22
32. Munkberg, J., Hasselgren, J., Shen, T., Gao, J., Chen, W., Evans, A., Müller, T., Fidler, S.: Extracting Triangular 3D Models, Materials, and Lighting From Images. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 8280–8290 (June 2022) 4
33. Oechsle, M., Peng, S., Geiger, A.: Unisurf: Unifying neural implicit surfaces and radiance fields for multi-view reconstruction. In: International Conference on Computer Vision (ICCV) (2021) 7
34. Park, J.J., Florence, P., Straub, J., Newcombe, R., Lovegrove, S.: Deepsdf: Learning continuous signed distance functions for shape representation. In: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (June 2019) 5

35. Parker, S.G., Bigler, J., Dietrich, A., Friedrich, H., Hoberock, J., Luebke, D., McAllister, D., McGuire, M., Morley, K., Robison, A., Stich, M.: Optix: a general purpose ray tracing engine. ACM Trans. Graph. **29**(4) (jul 2010). https://doi.org/10.1145/1778765.1778803, https://doi.org/10.1145/1778765.1778803 9
36. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S.: Pytorch: An imperative style, high-performance deep learning library. In: Advances in Neural Information Processing Systems 32 (2019), http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf 9
37. Penner, E., Zhang, L.: Soft 3d reconstruction for view synthesis **36**(6) (2017) 3
38. Rebain, D., Jiang, W., Yazdani, S., Li, K., Yi, K.M., Tagliasacchi, A.: Derf: Decomposed radiance fields. In: 2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 14148–14156 (2021). https://doi.org/10.1109/CVPR46437.2021.01393 4
39. Reiser, C., Peng, S., Liao, Y., Geiger, A.: Kilonerf: Speeding up neural radiance fields with thousands of tiny mlps. In: International Conference on Computer Vision (ICCV) (2021) 4
40. Reiser, C., Szeliski, R., Verbin, D., Srinivasan, P.P., Mildenhall, B., Geiger, A., Barron, J.T., Hedman, P.: Merf: Memory-efficient radiance fields for real-time view synthesis in unbounded scenes. arXiv preprint arXiv:2302.12249 (2023) 2, 4
41. Rematas, K., Liu, A., Srinivasan, P.P., Barron, J.T., Tagliasacchi, A., Funkhouser, T., Ferrari, V.: Urban radiance fields. CVPR (2022) 4
42. Riegler, G., Koltun, V.: Free view synthesis. In: European Conference on Computer Vision (2020) 3
43. Riegler, G., Koltun, V.: Stable view synthesis. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (2021) 3
44. Shen, T., Gao, J., Yin, K., Liu, M.Y., Fidler, S.: Deep marching tetrahedra: a hybrid representation for high-resolution 3d shape synthesis. In: Advances in Neural Information Processing Systems (NeurIPS) (2021) 4
45. Tewari, A., Fried, O., Thies, J., Sitzmann, V., Lombardi, S., Sunkavalli, K., Martin Brualla, R., Simon, T., Saragih, J., Nießner, M., Pandey, R., Fanello, S., Wetzstein, G., Zhu, J.Y., Theobalt, C., Agrawala, M., Shechtman, E., Goldman, D., Zollhöfer, M.: State of the art on neural rendering. Computer Graphics Forum (2020) 3
46. Uy, M.A., Nakayama, G.K., Yang, G., Thomas, R.K., Guibas, L., Li, K.: Nerf revisited: Fixing quadrature instability in volume rendering. In: NeurIPS (2023) 14
47. Verbin, D., Hedman, P., Mildenhall, B., Zickler, T., Barron, J.T., Srinivasan, P.P.: Ref-NeRF: Structured view-dependent appearance for neural radiance fields. CVPR (2022) 4
48. Wang, D., Zhang, T., Süsstrunk, S.: NEMTO: Neural Environment Matting for Novel View and Relighting Synthesis of Transparent Objects. In: Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV) (October 2023) 5
49. Wang, P., Liu, L., Liu, Y., Theobalt, C., Komura, T., Wang, W.: Neus: Learning neural implicit surfaces by volume rendering for multi-view reconstruction. arXiv preprint arXiv:2106.10689 (2021) 5, 7

50. Wang, Z., Shen, T., Nimier-David, M., Sharp, N., Gao, J., Keller, A., Fidler, S., Müller, T., Gojcic, Z.: Adaptive shells for efficient neural radiance field rendering. ACM Trans. Graph. **42**(6) (2023). https://doi.org/10.1145/3618390, https://doi.org/10.1145/3618390 2, 3, 5, 10, 11, 12, 21, 22
51. Weiping, S., Chence, S., Zhiping, X., Zhijian, D., Yewen, X., Ming, Z., Jian, T.: Autoint: Automatic feature interaction learning via self-attentive neural networks. arXiv preprint arXiv:1810.11921 (2018) 4
52. Wu, T., Liang, H., Zhong, F., Riegler, G., Vainer, S., Oztireli, C.: αsurf: Implicit surface reconstruction for semi-transparent and thin objects with decoupled geometry and opacity (2023) 5
53. Yariv, L., Gu, J., Kasten, Y., Lipman, Y.: Volume rendering of neural implicit surfaces. In: Thirty-Fifth Conference on Neural Information Processing Systems (2021) 7
54. Yariv, L., Hedman, P., Reiser, C., Verbin, D., Srinivasan, P.P., Szeliski, R., Barron, J.T., Mildenhall, B.: Bakedsdf: Meshing neural sdfs for real-time view synthesis. arXiv (2023) 2, 5, 7, 12, 13, 22
55. Young, J.: Xatlas. Available at: https://github.com/jpcy/xatlas (2023) 9
56. Yu, A., Li, R., Tancik, M., Li, H., Ng, R., Kanazawa, A.: PlenOctrees for real-time rendering of neural radiance fields. In: ICCV (2021) 4

6 Supplementary Material

In this section, we provide information about the following:

- Surface parameterization
- Spherical Gaussian fitting
- Run-time performance on mipNeRF 360 scenes
- Visualization of ablations (quad mesh)
- Visualization of mesh extracted using different approaches
- Evaluation using more metrics
- Storage requirements of rendering
- Visualization of the geometry of scenes

6.1 Spherical Gaussian fitting

To achieve high rendering speed, we parameterize radiance using Spherical Gaussians (SG). We start with a fine-tuned NeRF (see Sec. 3.3) in which view-dependent radiance is represented using an MLP. We then train an instant-NGP back-bone to predict radiance on the surface of the mesh using a fixed number of SG. This network is supervised using rendering loss (Eq. 2). We use 6 SG lobes for NeRF synthetic dataset that consists of shapes with large view-dependent effects. We use 3 SG lobes for Shelly dataset and mipNeRF 360 scenes, which provides a good tradeoff between rendering speed and reconstruction quality. This trained network is used to bake SG parameters into a texture map (see Sec 3.4). We choose this two-step process instead of directly using SGs to represent radiance in the pre-training of NeRF because smaller number of SG lobes have less representation power in comparison to MLP which can lead the density branch predicting fuzzy geometry to approximate the view-dependent effects.

6.2 Surface parameterization

Before the surface parameterization we simplify the mesh using vertex clustering algorithm. We further remove the mesh faces that are invisible from the training views. We also remove the regions of the faces where maximum volumetric weights across all training views are below a certain threshold, as this region belongs to the internal part of an object and does not contribute significantly to rendering. We then segment the mesh into smaller patches using graph cuts. Each segmented patch is parameterized separately and then all patches are packed into an atlas. For mipNeRF 360 scenes, we parameterize the mesh in contracted space such that the triangles that are farther away use fewer texels per unit.

6.3 Run-time performance

We also implement rendering using the depth peeling algorithm [10] to efficiently render in a browser while taking full advantage of hardware-accelerated rasterization, as shown in the Tab. 6.

6.4 Effect of different mesh extraction techniques

Mesh extracted from our quadrature field can result in holes. This can be alleviated by adding a coarse mesh extracted from the density field as shown in Fig. 9. This comes at a little additional cost of more triangles per scene.

Fig. 9: Mesh extracted from our quadrature field can result in holes. This can be alleviated by adding a coarse mesh extracted from the density field.

	FPS (depth-peeling)	FPS (ray-tracing)
Desktop Nvidia-3090	15	140

Table 6: Comparison of rendering speed between depth-peeling and ray-tracing implementations using mic scene of NeRF synthetic dataset at 2× super-resolution.

6.5 Evaluation metrics

We provide more evaluation using SSIM metric in Tab. 7, Tab. 9, and Tab. 11 and LPIPS metric Tab. 8, Tab. 10 and Tab. 12 respectively on Shelly, NeRF synthetic and mipNeRF 360 dataset.

6.6 Architecture design

We implement a quadrature field using the hash grid with an MLP with two hidden layers, each of width 16. The input coordinates are not only input to the hash

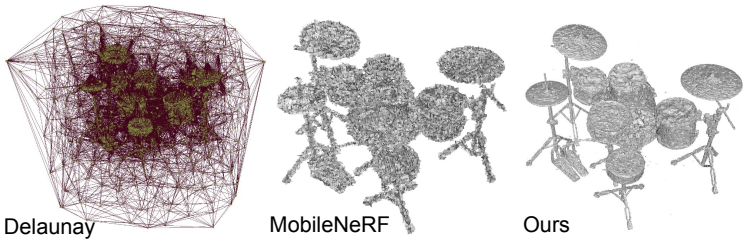

Fig. 10: Comparison of reconstructed geometry. 1) Mesh reconstructed using Delaunay Triangulation on the surface field. This approach produces more vertices of tetrahedra in regions where the surface is likely to occur. The mesh reconstructed using Delaunay triangulation results in a bad representation of the underlying surface. 2) Mesh reconstructed using MobileNeRF consists of polygonal soup. 3) Mesh reconstructed using our approach.

Table 7: Evaluation of Shelly dataset using SSIM metric.

SSIM	Mean	Fernvase	Pug	Woolly	Horse	Khady	Kitten
Instant-NGP [31]	0.9675	0.9870	0.9634	0.9778	0.9947	0.8955	0.9870
GS [25]	0.9591	0.9871	0.9627	0.9470	0.9942	0.8793	0.9841
MobileNeRF [7]	0.9108	0.9440	0.8850	0.8910	0.9800	0.8230	0.9420
Adaptive shells [50]	0.9542	0.9760	0.9470	0.9500	0.9920	0.8810	0.9790
Ours	0.9545	0.9778	0.9385	0.9503	0.9901	0.8893	0.9813

Table 8: Evaluation of Shelly dataset using LPIPS metric.

LPIPS	Mean	Fernvase	Pug	Woolly	Horse	Khady	Kitten
Instant-NGP [31]	0.0584	0.0256	0.0672	0.0545	0.0222	0.1475	0.0333
GS [25]	0.0670	0.0235	0.0737	0.0855	0.0244	0.1606	0.0345
MobileNeRF [7]	0.1288	0.0740	0.1670	0.1630	0.0570	0.2180	0.0940
Adaptive shells [50]	0.0788	0.0460	0.0930	0.0890	0.0290	0.1600	0.0560
Ours	0.0730	0.0369	0.0905	0.0950	0.0302	0.1400	0.0454

Table 9: Evaluation on NeRF synthetic dataset using SSIM metric.

SSIM	Mean	lego	chair	ship	mic	drums	materials	ficus	hotdog
Instant-NGP [31]	0.9617	0.9803	0.9849	0.8943	0.9907	0.9328	0.9470	0.9819	0.9820
Adaptive shells [50]	0.9571	0.9730	0.9850	0.8770	0.9880	0.9370	0.9350	0.9810	0.9810
MobileNeRF [7]	0.9471	0.9750	0.9780	0.8670	0.9790	0.9270	0.9130	0.9650	0.9730
Ours	0.9522	0.9687	0.9745	0.8787	0.9855	0.9274	0.9347	0.9764	0.9715

Table 10: Evaluation on NeRF synthetic dataset using LPIPS metric.

LPIPS	Mean	lego	chair	ship	mic	drums	materials	ficus	hotdog
Instant-NGP [31]	0.0518	0.0228	0.0215	0.1331	0.0152	0.0852	0.0736	0.0279	0.0349
Adaptive shells [50]	0.0563	0.0310	0.0230	0.1410	0.0150	0.0860	0.0860	0.0330	0.0350
MobileNeRF [3]	0.0618	0.0250	0.0250	0.1450	0.0320	0.0770	0.0920	0.0480	0.0500
Ours	0.0691	0.0476	0.0405	0.1504	0.0262	0.0939	0.0871	0.0380	0.0690

Table 11: Evaluation of MipNeRF 360 dataset using SSIM metric.

SSIM	Mean indoors	KitchenLego	Room	Bonsai	Kitchencounter	Mean outdoors	Garden	Bicycle	Stump
Instant-NGP [31]	0.8663	0.8607	0.8839	0.9025	0.8182	0.6346	0.7270	0.5524	0.6245
GS [25]	0.9198	0.9220	0.9140	0.9380	0.9050	0.8047	0.8680	0.7710	0.7750
MobileNeRF [7]	-	-	-		-	0.5270	0.5990	0.4260	0.5560
BakedSDF [54]	0.8365	0.8170	0.8700	0.8510	0.8080	0.6387	0.7510	0.5700	0.5950
Adaptive Shells [50]	0.8723	0.8660	0.8950	0.9330	0.7950	0.6057	0.7570	0.5440	0.5160
Ours	0.8598	0.8660	0.8763	0.8896	0.8075	0.6496	0.7520	0.5673	0.6296

grid but also concatenated with the output of the hash grid. We use Exponential Linear Units [8] in the MLP which allows double derivatives used to supervise our quadrature field. The experiments on NeRF synthetic and Shelly dataset are done with the hash grid with a maximum resolution of 512, minimum resolution of 16 and codebook size of 2^{30}. For the real dataset, we use the hash grid with a maximum resolution of 4096, a minimum resolution of 16 and a codebook size of 2^{25}. We use a similar architecture design to implement our deformation network, except we use relu non-linearity in the MLP. The choice of these parameters is dependent on available GPU memory and validation performance.

Table 12: Evaluation on MipNeRF 360 dataset using LPIPS metric.

LPIPS	Mean indoors	KitchenLego	Room	Bonsai	Kitchencounter	Mean outdoors	Garden	Bicycle	Stump
Instant-NGP [31]	0.2997	0.2328	0.3291	0.2801	0.3569	0.3849	0.2725	0.4602	0.4221
GS [25]	0.1895	0.1290	0.2200	0.2050	0.2040	0.1727	0.1030	0.2050	0.2100
MobileNeRF [7]	-	-	-	-	-	0.4337	0.3580	0.5130	0.4300
BakedSDF [54]	0.2583	0.2370	0.2510	0.2590	0.2860	0.3173	0.2130	0.3680	0.3710
Adaptive Shells [50]	0.2848	0.2190	0.3000	0.2250	0.3950	0.3890	0.2470	0.4380	0.4820
Ours	0.3032	0.2315	0.3229	0.3023	0.3561	0.3513	0.2414	0.4220	0.3905

	NeRF Synthetic	Shelly	MipNeRF 360
Disk (MB)	328	1213	4331
VRAM (MB)	803	2764	9066

Table 13: Average memory requirement on different datasets.

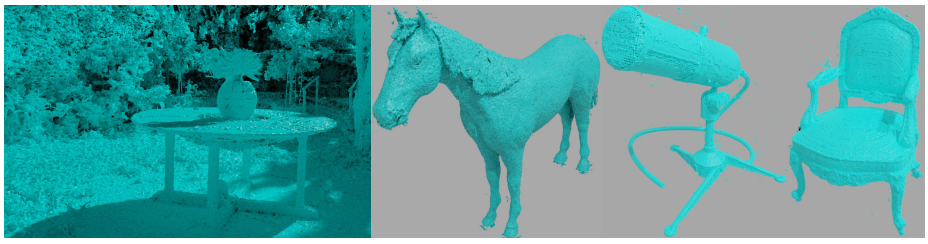

Fig. 11: Geometry from our approach on different datasets.