This ICCV paper is the Open Access version, provided by the Computer Vision Foundation.
Except for this watermark, it is identical to the accepted version;
the final published version of the proceedings is available on IEEE Xplore.

Tetra-NeRF: Representing Neural Radiance Fields Using Tetrahedra

Jonas Kulhanek
Czech Technical University in Prague
jonas.kulhanek@cvut.cz

Torsten Sattler
Czech Technical University in Prague
torsten.sattler@cvut.cz

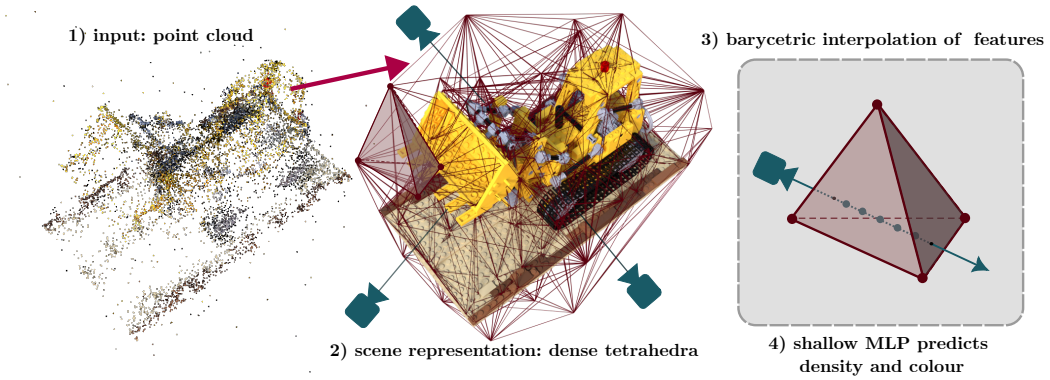


Figure 1. The input to Tetra-NeRF is a point cloud which is triangulated to get a set of tetrahedra used to represent the radiance field. Rays are sampled, and the field is queried. Barycentric interpolation is used to interpolate tetrahedra vertices, and the resulting features are passed through a shallow MLP to get the density and colours for volumetric rendering.

Abstract

Neural Radiance Fields (NeRFs) are a very recent and very popular approach for the problems of novel view synthesis and 3D reconstruction. A popular scene representation used by NeRFs is to combine a uniform, voxel-based subdivision of the scene with an MLP. Based on the observation that a (sparse) point cloud of the scene is often available, this paper proposes to use an adaptive representation based on tetrahedra obtained by Delaunay triangulation instead of uniform subdivision or point-based representations. We show that such a representation enables efficient training and leads to state-of-the-art results. Our approach elegantly combines concepts from 3D geometry processing, triangle-based rendering, and modern neural radiance fields. Compared to voxel-based representations, ours provides more detail around parts of the scene likely to be close to the surface. Compared to point-based representations, our approach achieves better performance. The source code is publicly available at: https://jkulhanek.com/tetra-nerf.

1. Introduction

Reconstructing 3D scenes from images and rendering photo-realistic novel views is a key problem in computer vision. Recently, NeRFs [3, 4, 39] became dominant in the field for their superior photo-realistic results. Originally, NeRFs used MLPs to represent the 3D scene as an implicit function. Given a set of posed images, NeRF randomly samples a batch of pixels, casts rays from the pixels into the 3D space, queries the implicit function at randomly sampled distances along the rays, and aggregates the sampled values using volumetric rendering [37, 39]. While the visual results of such methods are of high quality, the problem is that querying large MLPs at millions of points is costly. Also, once the network is trained, it is difficult to make any changes to the represented radiance field as everything is baked into the MLPs parameters, and any change has a non-local effect. Since then, there have been a lot of proposed alternatives to the large MLP field representation [9, 17, 32, 41, 57, 70, 74]. These methods combine an MLP with a voxel feature grid [41, 57], or in some cases represent the radiance field directly as a tensor [9, 10, 17]. When querying these representations, first, the containing voxel is found, and the features stored at the eight corner points of the voxel are trilinearly interpolated. The result is either passed through a shallow MLP [9, 10, 41, 57] or is used directly as the density and colour [17, 32, 57].

Having a dense tensor represent the entire scene is very inefficient, as we only need to represent a small space around surfaces. Therefore, different methods propose different ways of tackling the issue. Instant-NGP [41], for example, uses a hash grid instead of a dense tensor, where it relies on optimisation to resolve the hash collisions. How-

ever, similarly to MLPs, any change to the stored hashmap influences the field in many places. A more common direction to addressing the issue is by directly using a sparse tensor representation [9, 17]. These methods start with a low-resolution grid and, at predefined steps, subsample the representation, increasing the resolution. These approaches tend to require a careful setting of hyperparameters, such as the scene bounding box and the subdivision steps, in order for the methods to work well.

Because many of these methods use traditional structure from motion (SfM) [54, 55] methods to generate the initial poses for the captured images, we can reuse the original reconstruction in the scene representation. Inspired by classical surface reconstruction methods [22,23,29,30], we represent the scene as a dense triangulation of the input point cloud, where the scene is a set of non-overlapping tetrahedra whose union is the convex hull of the original point cloud [14]. When querying such a representation, we find to which tetrahedron the query point belongs and perform barycentric linear interpolation of the features stored in the vertices of the tetrahedron. This very simple representation can be thought of as the direct extension of the classical triangle-rendering pipelines used in graphics [40,43,45]. The representation avoids problems with the sparsity of the input point cloud as the tetrahedra fully cover the scene, resulting in a continuous rather than discrete representation.

This paper makes the following contributions: (**1**) We propose a novel radiance field representation which is initialised from a sparse or dense point cloud. This representation is naturally denser in the proximity of surfaces and, therefore, provides a higher resolution in these regions. (**2**) The proposed representation is evaluated on multiple synthetic and real-world datasets and is compared with a state-of-the-art point-cloud-based representation – Point-NeRF [70]. The presented results show that our method clearly outperforms this baseline. We further demonstrate the effectiveness of our adaptive representation by comparing it with a voxel-based representation that uses the same number of trainable parameters. (**3**) We make the source code and model checkpoints publicly available.[1]

2. Related work

Multi-view reconstruction. The problem of multi-view reconstruction has been studied extensively and tackled with a variety of structure from motion (SfM) [54, 61, 63], and multi-view stereo (MVS) [12, 18, 55, 71] methods. These methods usually output the scene represented as a point cloud [54, 55]. In most rendering approaches, the point cloud is converted into a mesh [24, 35], and novel views are rendered by reprojecting observed images into each novel viewpoint and blending them together using either heuristically-defined [8, 13, 69] or learned [20, 50, 51, 73] blending weights.

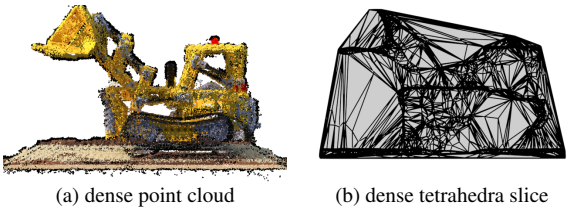
However, the process of getting the meshes is usually quite noisy, and the resulting meshes tend to have inaccurate geometry in regions with fine details or complex materials. Instead of using noisy meshes, point-based neural rendering methods [1, 26, 38, 53] perform splatting of neural features and use 2D convolutions to render them. In contrast to these methods, our approach operates and aggregates features directly in 3D and does not suffer from the noise in the point cloud or the reconstructed mesh.

Neural radiance fields. Recently, NeRFs [3, 4, 34, 39, 77] have gained a lot of attention thanks to their high-quality rendering performance. The original NeRF method [39] was extended to better handle aliasing artefacts in [3], to better represent unbounded scenes in [4, 48, 77], or to handle real-world captured images [36, 58]. The training of the large MLPs used in these methods can be quite slow, and there has been a lot of effort on speeding up either the training [9, 17, 41] or the rendering [21, 47, 48, 74] sometimes at the cost of larger storage requirements. Other approaches tackled different aspects of NeRFs like view-dependent artefacts [62], relighting [5, 7], or proposed generative models [46, 64]. Also, a popular research direction is making the models generalize across different scenes [11, 28, 49, 66, 75]. A large area of research is dedicated to the surface reconstruction and, instead of using the radiance fields, represents the scene implicitly by modelling the signed distance function (SDF) [52, 65, 67, 72, 76]. Unlike those approaches, we focus only on the radiance field representation and consider these methods orthogonal to ours.

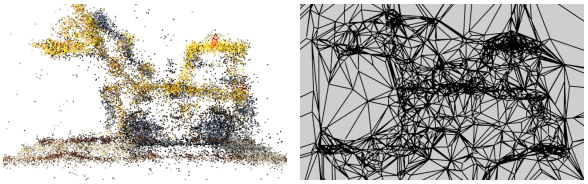
Although there are some methods that train radiance fields while fine-tuning the poses [31, 60] or without known cameras [6, 68], most methods need camera poses for the reconstruction. SfM, *e.g.*, COLMAP [54, 55], is typically used for estimating the poses, which also produces a (sparse) point cloud. Our approach makes use of this by-product of the pose recovery process instead of only using the poses themselves.

Field representations. When a single MLP is used to represent the entire scene, everything is baked into a single set of parameters which cannot be easily modified, because any change to the parameters has a non-local effect, *i.e.*, it changes the scene at multiple unrelated places. To overcome this problem, others have experimented with different representations of the radiance fields [9, 10, 17, 32, 41, 44, 57, 70]. A common practice is to represent the scene as a shallow MLP and an efficiently encoded voxel grid [9, 10, 32, 41, 44]. The encoded voxel grid can also be used represent the radiance field directly [17, 57]. The voxel grid can be encoded as a sparse tensor [17, 32, 57], a factorisation of the 4D tensor [9, 10], or a hashmap [41]. When

[1] https://github.com/jkulhanek/tetra-nerf



(a) dense point cloud (b) dense tetrahedra slice



(c) sparse point cloud (d) sparse tetrahedra slice

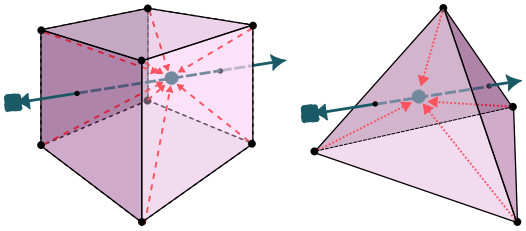
Figure 2. Input point cloud and a slice through the triangulated tetrahedra. Note that smaller tetrahedra are created closer to the surface of the scene, *i.e.*, regions close to the surface are represented with a finer resolution.

these structures are queried, trilinear interpolation is used to combine the feature vectors stored in the containing voxel corners. Unfortunately, the hashmaps [41] and hierarchical representations [10] have the same non-local effect problem, and the rest of the approaches rely on subsequent upsampling of the field and can be overly complicated. Also, feature vectors cannot be placed arbitrarily in the 3D space as they must lie on the grid with a fixed resolution. In contrast, our approach is much more flexible as it stores the feature vectors freely in 3D space.

Finally, Point-NeRF [70] represents the scene as a point cloud of features which are queried using k-nearest neighbours search. However, when the point cloud contains sparse regions, the rays do not intersect any neighbourhoods of any points and the pixels stay empty without the ability to optimise. Therefore, Point-NeRF [70] relies on gradually adding more points during training and increasing the scene complexity. Since we use the triangulation of the point cloud rather than the discrete point cloud itself, our representation is continuous and does not suffer from empty regions. Therefore, we do not have to add any points during training.

3. Method

A common strategy in the literature is to represent the scene explicitly through a voxel volume. In contrast to this uniform subdivision, we investigate using an adaptive subdivision of the scene. In many scenarios, an approximation of the scene geometry is either given, *e.g.*, when using SfM to compute the input camera poses, or can be computed, *e.g.*, via MVS or single-view depth predictions [79]. This allows us to compute an adaptive subdivision of the scene via Delaunay triangulation [14] of such a point cloud. This results in a set of non-overlapping tetrahedra, where smaller tetrahedra are created close to the surface of the scene (*c.f.* Fig. 2). In the following, we explain how this adaptive subdivision of the scene can be used instead of voxels for volume rendering and neural rendering.



(a) trilinear interpolation (b) barycentric interpolation

Figure 3. **Trilinear and barycentric intepolation.** Trilinear interpolation is a weighted combination of the eight voxel corners. Barycentric interpolation weights the four vertices of the tetrahedron vertices based on the barycentric coordinates [16].

3.1. Preliminaries

Neural Radiance Fields (NeRFs) [39] represent a scene through an implicit function $F(\mathbf{x}, \mathbf{d})$, often modelled via a neural network, that returns a colour value and a volume density prediction for a given 3D point $\mathbf{x}$ in the scene observed from a viewing direction $\mathbf{d}$. Volume rendering [37] is used to synthesise novel views: for each pixel in a virtual view, we project a ray from the camera plane into the scene and sample the radiance field to obtain the colour and density values $\mathbf{c}_i$ and σ_i at distances t_i along the ray, where $i = 1 \dots N$. The individual samples are then combined to predict the colour $\mathbf{C}$ for the pixel:

$$\begin{aligned} \mathbf{C} &= \sum_{i=1}^{N} T_i (1 - \exp(\sigma_i \delta_i)) \mathbf{c}_i \quad \text{, where} \\ T_i &= \exp(- \sum_{i=1}^{N-1} \sigma_i \delta_i) \quad , \end{aligned} \tag{1}$$

and $\delta_i = t_i - t_{i-1}$ is the distance between adjacent samples. This process is fully differentiable. Thus, after computing the mean squared error (MSE) between the predicted and ground truth colours associated with each ray, we can back-propagate the gradients to the radiance field.

Voxel-based feature fields such as NSVF [32] represent the scene as a voxel grid and an MLP. Each grid point of the voxel grid is assigned a trainable vector. There are eight feature vectors associated with a single voxel, but the vectors are shared between neighbouring voxels. For each query point sampled along the ray, its corresponding voxel is found first. A feature for the point is then computed via trilinear interpolation of the features of the voxel grid points based on the position of the query point (*c.f.* Fig. 3, left). The resulting feature vector is passed through an MLP to predict the density and appearance vector. The appearance vector is combined with the ray direction and

passed through a second MLP in order to compute a view-dependent colour.

3.2. Tetrahedra fields

Given a set of points in 3D space, we build the tetrahedra structure by triangulating the points. We apply the Delaunay triangulation [14] to obtain a set of non-overlapping tetrahedra whose union is the convex hull of the original points. Fig. 2 shows example tetrahedra obtained by triangulating dense and sparse COLMAP-reconstructed point clouds [54, 55]. Note that the resulting representation is adaptive as it uses a higher resolution (smaller tetrahedra) closer to the surface and larger tetrahedra to model regions farther away from the surface.

We associate all vertices of the tetrahedra with trainable vectors. As in the voxel grid case, vertices, and thus vectors, are shared between adjacent tetrahedra. The resulting tetrahedra field can be queried in the same way a voxel grid representation is queried: for each query point, we first find the tetrahedron containing the point. Instead of the trilinear interpolation used for voxel volumes, we use the barycentric interpolation [16] to compute a feature vector for the query point from the four feature vectors stored at the tetrahedron's vertices (*c.f*. Fig. 3). To this end, we compute the query point $\mathbf{x}$'s barycentric coordinates $\boldsymbol{\lambda}$, which express the point's 3D coordinates as a unique weighted combination of the 3D coordinates of the tetrahedron's vertices. In particular, the weight for a vertex is the volume of the tetrahedron constructed from the query point and the face opposite to the vertex divided by the volume of the full tetrahedron:

$$\boldsymbol{\lambda} = \left(\frac{V_{x234}}{V_{1234}}, \frac{V_{1x34}}{V_{1234}}, \frac{V_{12x4}}{V_{1234}}, \frac{V_{123x}}{V_{1234}} \right), \tag{2}$$

where V_{1234} is the volume of the full tetrahedron and V_{x234}, V_{1x34}, ..., are volumes of tetrahedra with 1^{st}, 2^{nd}, ..., vertex replaced by $\mathbf{x}$. The same weights $\boldsymbol{\lambda}$ are applied to the feature vectors of the vertices to obtain the query feature.

The interpolated features are used as the input to a small MLP in order to predict density and colour at the query point. We first pass the barycentric-interpolated features through a three-layer MLP to compute the density and appearance features. We then concatenate the appearance features with the ray direction vector encoded using Fourier features [39, 59] and pass the result through a single linear layer to get the raw RGB colour value.

Finally, to map the raw density values $\bar{\sigma}_i$ returned by the network to the volume density σ_i required by volume rendering, we apply the softplus activation function [3]. For the RGB colour value, we use the sigmoid function [17].

3.3. Efficiently querying a tetrahedra field

Determining the corresponding voxel for a given query point can be done highly efficiently via hashing [42]. In contrast, finding the corresponding tetrahedron for a query point is more complex. This in turn can significantly impact rendering, and thus training, efficiency [17, 41, 57].

In order to efficiently look up the corresponding tetrahedra, we exploit that we are not considering isolated points, but points sampled from a ray. We compute the tetrahedra that are intersected by the ray, allowing us to march through the tetrahedra rather than computing them individually per point. The relevant tetrahedra can be found efficiently using acceleration structures for fast ray-triangle intersection computations, *e.g*., via NVidia's OptiX library [43]: We first compute the intersections between the rays from a synthetic view and all faces of the tetrahedra. For each ray, we take the first 512 intersected triangles[2], and determine the corresponding tetrahedra. The tetrahedra can be ordered based on the intersections along the ray, allowing us to easily march through the tetrahedra to determine which tetrahedron to use for a given query point.

A side benefit of computing ray-triangle intersections is that we can simplify computing the barycentric coordinates: For each intersection, we compute the 2D barycentric coordinates w.r.t. the triangle. We then obtain the 3D barycentric coordinates w.r.t. the associate tetrahedron by simply adding zero for the vertex opposite to the triangle. For a query point inside a tetrahedron, we can compute its tetrahedron barycentric coordinates by linearly interpolating between the barycentric coordinates of the two intersections of the ray and the tetrahedron.

3.4. Coarse and fine sampling

We follow the common practice of having a two-stage sampling procedure [9, 39]. In the coarse stage, we sample uniformly along the ray. In the fine stage, we use the density weights from the coarse sampling stage to bias the sampling towards sampling closer to the potential surface. Following [39], we use the stratified uniform sampling for the coarse stage. The stratified uniform sampling splits the ray into equally long intervals and samples uniformly in each interval. Unlike NeRF [39], we limit the sampling to the space occupied by tetrahedra. In the fine sampling stage, we use the same network as in the coarse sampling stage.

For the fine sampling stage, we take the accumulated weights w_i from the coarse sampling:

$$\bar{w}_i = (1 - \exp(-\sigma_i \delta_i)) \exp\left(- \sum_{j=1}^{i-1} \sigma_j \delta_j \right) . \tag{3}$$

These weights are the coefficients used in Equation 1 as multipliers for the colours [39]. We obtain w_i by normalizing $\bar{w}_i$. Following [39], we sample set of fine samples using

[2]For efficiency, we only consider a fixed number of triangles. As discussed later on, this can degrade results in larger scenes / scenes with a fine-grained tetrahedralisation, where more intersections are needed. Naturally, more triangles can be considered at the cost of longer run-times.

weight w_i. To render the final colour, we merge the dense and fine samples and use all in the rendering equation [9].

4. Experiments

We compare Tetra-NeRF to relevant methods on the commonly used synthetic Blender [39], the real-world Tanks and Temples [25], and the challenging object-centric Mip-NeRF 360 [4] datasets. To show its effectiveness, we compare Tetra-NeRF to a dense-grid representation and evaluate it with reduced quality of the input point cloud. We start by describing the exact hyperparameters used.

4.1. Implementation details

Generating point cloud & triangulation. Given a set of posed images, we use the COLMAP reconstruction pipeline [54, 55] to get the point cloud used in our tetrahedra field representation. We then reduce the size of the resulting point cloud such that if the number of points is larger than 10^6, we subsample 10^6 points randomly. For the Blender dataset experiments, where the number of points is smaller, we add more randomly generated points. In that case, the number of random points is half the number of original points. The reason for adding the points is that with a low number of points, some pixels on edges may not intersect any tetrahedra, potentially producing artefacts on the edges. Each added point is sampled as follows: we sample a random point x_0 from the original point cloud, sample a random normal vector n, and a number $\alpha \sim \mathcal{N}(\bar{d}, \bar{d}^2)$, where $\bar{d}$ is the average spacing of the original point cloud, *i.e.* the average distance between each point and its six closest neighbours. We then add the point $x = x_0 + \alpha n$.

Initialization. Given the processed point cloud, we apply the Delaunay triangulation [14] to get a set of tetrahedra. To this end, we use the CGAL library [2], which in our experiments runs in the order of milliseconds. Following [41], we initialise the features of size 64 at the vertices of the tetrahedra with small values sampled uniformly in the range -10^{-4} to 10^{-4}. However, to allow the model to reuse the information contained in the point cloud, we set the first four dimensions of the feature field to the RGBA colour values (rescaled to interval $[0, 1]$) stored at the associated points in the point cloud. The alpha value of all original points is one, whereas all randomly sampled points have an alpha value of zero. For the MLP, we follow the common practice of using the Kaiming uniform initialisation [19]. The hidden sizes in all MLPs are 128.

Training. During training, we sample batches of 4,096 random rays from random training images. We use volumetric rendering to predict the colour of each ray. The gradients are computed by backpropagating the MSE loss between the predicted colour and the ground truth colour value. We use the RAdam optimizer [33] and decay the learning rate exponentially from 10^{-3} to 10^{-4} in 300k steps. The training code is built on top of the Nerfstudio framework [60], and the tetrahedra field is implemented in CUDA and uses the OptiX library [43]. We train on a single NVIDIA A100 GPU and the training speed ranges from 15k rays per second to 30k rays per second. The speed depends on how well-structured the triangulation is, how many vertices there are, and if there is empty space around the object. The full training with 300k iterations takes between 11 and 24 hours, depending on the scene complexity. However, good results are typically obtained much earlier, *e.g.*, in 100k iterations.

	PSNR ↑	SSIM ↑	LPIPS ↓
NeRF [39]	31.00	0.947	0.081
NSVF [32]	31.77	0.953	-
mip-NeRF [3]	34.51	0.961	0.043
instant-NGP [41]	33.18	-	-
Plenoxels [17]	31.71	0.958	0.049
Point-NeRF^{col} [70]	31.77	0.973	0.062
Point-NeRF^{mvs} [70]	33.31	0.978	0.049
Tetra-NeRF	32.53	0.982	0.041

Table 1. **Results on the Blender dataset** [39] averaged over all scenes in the dataset. Even though we use the same input point cloud as Point-NeRF^{col}, we outperform it greatly. We perform on par with Point-NeRF^{mvs} even though it uses many more points and densifies the point cloud during training. We highlight the best, second, and third values.

4.2. Results

Blender dataset [39] results. We compare to relevant baselines on the standard Blender dataset. We used the same split and evaluation procedure as in the original NeRF paper [39] and the same SSIM implementation as Point-NeRF [70]. In order to ensure a fair comparison with Point-NeRF [70] when COLMAP points were used, we use the exact same COLMAP reconstruction as Point-NeRF. We report the PSNR, SSIM, and LPIPS (VGG) [78] metrics. Tab. 1 shows averaged results, Fig. 4 shows qualitative results, and the results for individual scenes are given in *Supp. Mat.*

When we use the exact same COLMAP points as Point-NeRF (row *Point-NeRF^{col}*), we outperform it in all three metrics. We score comparably with *Point-NeRF^{mvs}*, even though it starts from a much denser and higher-quality initial point cloud, which it generates from a jointly trained model. Also note that both of these Point-NeRF configurations grow the point cloud during training and, therefore, the complexity of the scene representation grows. For us, the points are fixed, and the number of parameters stays the same. We also outperform Plenoxels [17], which uses a sparse grid. Note that same as Point-NeRF, Plenoxels also gradually increases the representation complexity by subdividing the grid resolution at predefined training epochs. Even though both Mip-NeRF and instant-NGP outperform

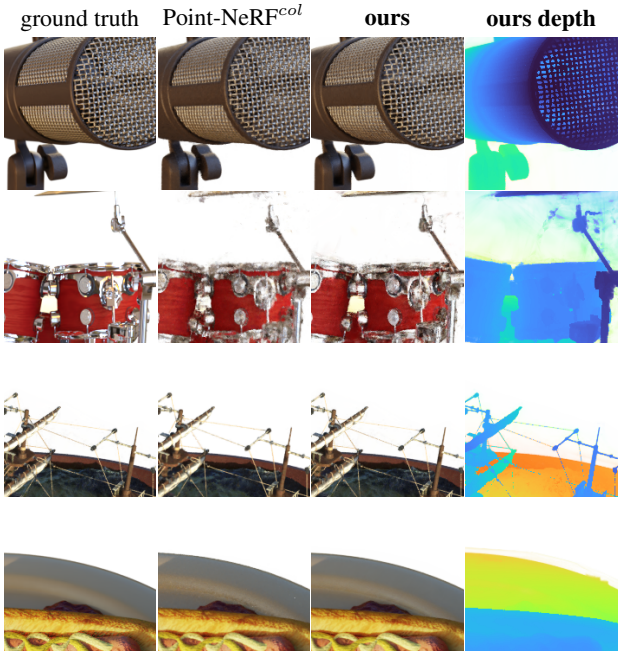


Figure 4. **Qualitative results on the Blender dataset.** We compare with Point-NeRF^{col} as we use the same input point cloud. In the **top**-row picture, we can see that Tetra-NeRF is able to represent fine details well on the *mic* scene. On the *drums* scene (**2nd** row), both methods struggle with shiny materials, but our method performs slightly better. **3rd** row: Tetra-NeRF can render thin ropes. **Bottom** row: accumulation is nonzero in areas close to surface.

	hotdog		*ship*	
	PSNR ↑	SSIM ↑	PSNR ↑	SSIM ↑
Point-NeRF^{static}	29.91	0.978	19.35	0.905
Tetra-NeRF	33.31	0.989	31.13	0.994

Table 2. **Comparison with Point-NeRF with disabled point cloud growing and prunning** shows that Point-NeRF performs significantly worse in all measured metrics because it struggles to handle the sparse point clouds.

our approach in terms of PSNR, our method is slightly better in terms of SSIM, and on par with Mip-NeRF in terms of LPIPS.

To analyze the tetrahedra field representation, we compare it with the point cloud field representation used in Point-NeRF. In Table 2, we compare our method to Point-NeRF when we disable point cloud growth and pruning. We show the results on two scenes from the Blender dataset [39] which were selected in the Point-NeRF paper. With this setup, we vastly outperform Point-NeRF in all metrics. The reason is that Point-NeRF requires the point cloud to be dense such that all rays have a chance to intersect a neighbourhood of a point. Since we use a continuous representation (tetrahedra field) rather than a discrete one, we can achieve good results even for sparser point clouds.

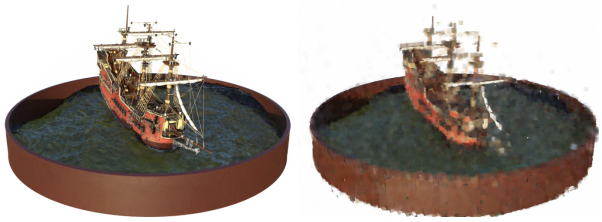


Figure 5. **Comparison with dense grid field representation.** Tetra-NeRF on the **left**, dense grid on the **right**. We used a comparable number of parameters for both methods (the dense grid having slightly more parameters). Due to the adaptive nature of tetrahedra fields, Tetra-NeRF produces significantly better rendering as it is able to focus on relevant parts of the scene.

	PSNR ↑	SSIM ↑	LPIPS ↓
NV [34]	23.70	0.848	0.260
NeRF [39]	25.78	0.864	0.198
NSVF [32]	28.40	0.900	0.153
Point-NeRF [70]*	28.35	0.942	0.090
Tetra-NeRF	28.90	0.957	0.059

Table 3. **Result on the Tanks and Temples dataset [25]** as processed by NSVF [32] averaged over all scenes. We show the PSNR, SSIM, and LPIPS (Alex) [78] metrics. We highlight the best, second, and third values. **Note, that Point-NeRF [70] results differ from the paper as they were recomputed with the resolution used in other methods.*

Comparison with the dense grid representation. In order to show the utility of the adaptive tetrahedra field representation, we compare it to a dense grid representation. Similarly to NSVF [32], we split the 3D scene bounding box into equally-sized voxels. When querying the field, we find the voxel to which the query point belongs and perform trilinear interpolation of the eight corners of the voxel. We choose the grid resolution such that the number of grid points is the lowest cube number larger than the number of points of the original point cloud. This ensures a fair comparison as the baseline uses a comparable (but larger) number of features than our approach. All other hyperparameters are kept the same as for Tetra-NeRF. We evaluate both approaches on two scenes from the Blender dataset [39]. The dense grid representation only scores PSNR 18.81 and 18.91 on the *lego* and *ship* scenes, respectively, whereas Tetra-NeRF scores PSNR 33.79 and 30.69, respectively. The results can be seen in Figure 5. Note that in this experiment, we only trained for $100,000$ iterations to save computation time. From the numbers and the figure, we can clearly see that the dense grid resolution is not sufficient to reconstruct the scene in enough detail. Because Tetra-NeRF uses an adaptive subdivision, which is more detailed around the surface, we are better able to focus on relevant scene parts. With a similar number of trainable parameters, we thus obtain better results.

Tanks and Temples [25] dataset. To be able to compare

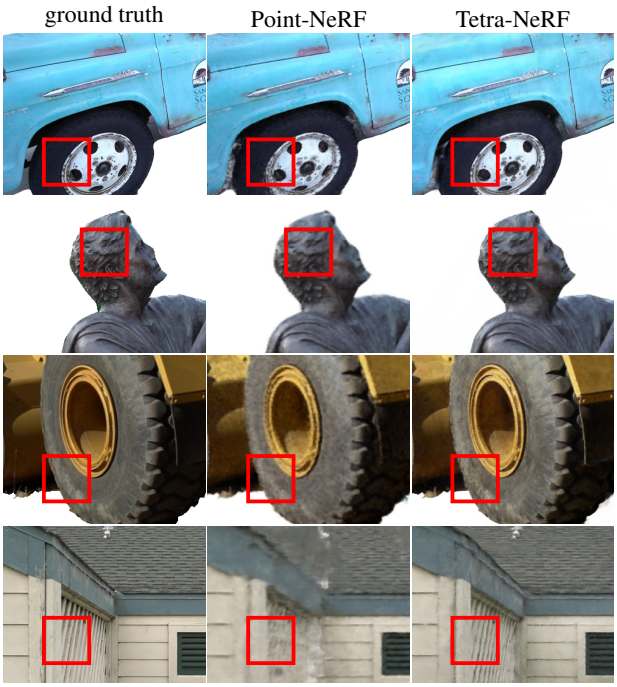


Figure 6. **Results on Tanks and Temples dataset.** In the **top** row, we can see that we are able to represent the rim of the wheel better than Point-NeRF. Similarly, we can represent the tyre better in the **third row**. Finally, in the **bottom**, Point-NeRF fails on the wall and the roof, whereas Tetra-NeRF can render these parts well.

with Point-NeRF [70] on real-world data, we have evaluated Tetra-NeRF on the Tanks and Temples [25] dataset. We use the same setup as in NSVF [32], where the object is masked. We report the usual metrics: PSNR, SSIM, and LPIPS (Alex).[3] We used the dense COLMAP reconstruction to get the point cloud used in the tetrahedra field. Quantitative results are shown in Table 3, qualitative results are presented in Figure 6, and the per-scene results are given in the *Supp. Mat.* Note that the results originally reported in the Point-NeRF paper [70] were evaluated with a resolution different from the compared methods. Therefore, to ensure a fair comparison, we have recomputed the metrics in the publicly available Point-NeRF's predictions with the full resolution of 1920×1080 as used in NSVF [32]. In the public dataset, one of the scenes had corrupted camera parameters and we had to reconstruct the poses again for the Ignatius scene. With the reconstructed poses, our method performs slightly worse on that scene compared to others.

The results show that our method outperforms the baselines in all compared metrics. This indicates that even though Point-NeRF relies on the ability to grow the point cloud density during training, this is not needed when using a continuous instead of a discrete representation.

Sparse and dense point cloud comparison. Previous experiments have used dense COLMAP reconstructions. However, dense point clouds may not be required to achieve good reconstructions. In this section, we compare models trained on dense reconstructions and sparse reconstructions. Table 5 compares PSNR, and SSIM metrics on two synthetic scenes from the Blender dataset [39] and two real-world scenes from the Mip-NeRF 360 dataset [4]. We also show the number of vertices of the tetrahedra, which is the size of the input point cloud, including the random points added at the beginning. Note that in this experiment, we only trained for $100,000$ iterations to save computation time.

[3] We always choose the type of LPIPS (Alex [27]/VGG [56]) such that we can compare with more methods as some only evaluate using one type.

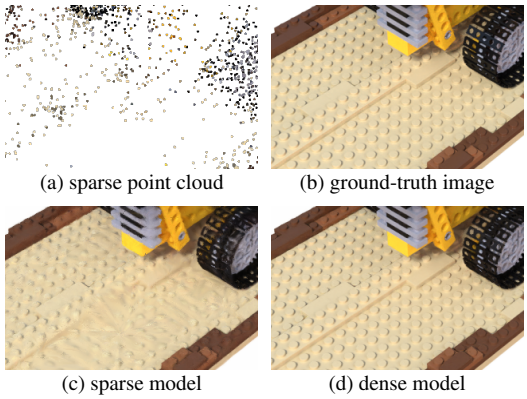


Figure 7. **Sparse *vs.* dense model comparison detail.** Even though the area in the bottom contains few points, we are still able to reconstruct at least low-frequency data.

From the results, we can see that for the Blender dataset [39] case, the dense model is much better. This is to be expected since the number of points obtained from the sparse reconstruction on the Blender dataset is very small and our MLP does not have enough capacity to represent fine details. However, even in regions with zero point coverage, we are still able to provide at least some low-frequency data (*c.f.* Fig. 7). On real-world scenes, the sparse point cloud model is almost on par with the dense one. The sparse reconstructions on the real-world dataset provide many more points compared to the synthetic dataset case, and the coverage is sufficient. From these results, we conclude that on real-world data the sparse model can be sufficient to achieve good performance.

Mip-NeRF 360 [4] dataset. We have further evaluated our method on the Mip-NeRF 360 [4] dataset. In order to ensure a fair comparison with Mip-NeRF 360 [4], we trained and evaluated on four times downsized images for the outdoor scenes and two times downsized images for the indoor scenes. The quantitative results are presented in Table 4, and the qualitative results are shown in Figure 8.

From the results, we can see that Tetra-NeRF is able to outperform both the vanilla NeRF [39] and Mip-NeRF [3]. We also outperform Stable View Synthesis [51] in terms of PSNR and score comparably in terms of LPIPS. This is pos-

	Outdoor			*Indoor*			*Mean*		
	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓
NeRF [15, 39]	21.46	0.458	0.515	26.84	0.790	0.370	23.85	0.605	0.451
mip-NeRF [3]	21.69	0.471	0.505	26.98	0.798	0.361	24.04	0.616	0.441
NeRF++ [77]	22.76	0.548	0.427	28.05	0.836	0.309	25.11	0.676	0.375
Deep Blending [20]	21.54	0.524	0.364	26.39	0.844	0.261	23.70	0.666	0.318
Point-Based Neural Rendering [26]	21.66	0.612	0.302	26.28	0.887	0.191	23.71	0.734	0.253
Stable View Synthesis [51]	23.01	0.662	0.253	28.22	0.907	0.160	25.33	0.771	0.211
mip-NeRF 360 [4]	23.72	0.687	0.282	29.43	0.911	0.181	26.26	0.786	0.237
Tetra-NeRF	23.17	0.586	0.298	30.21	0.881	0.103	26.30	0.717	0.211

Table 4. **Mip-NeRF 360 dataset results.** We show the PSNR, SSIM, and LPIPS (Alex) [78] on two categories of Mip-NeRF 360 [4] scenes: *outdoor*, and *indoor*. On the *outdoor* scenes, we outperform the Stable View Synthesis [51] and are comparable to Mip-NeRF 360 [4], even though our method does not implement the improvements suggested in NeRF++ [77] and Mip-NeRF 360 [4], and is more comparable to vanilla NeRF [39]. We highlight the best, second, and third values.

	sparse		*dense*	
	PSNR/SSIM	#points	PSNR/SSIM	#points
Blender/lego	29.77/0.959	25,784	33.79/0.985	302,781
Blender/ship	26.90/0.899	7,152	30.69/0.942	321,861
360/bonsai	28.12/0.902	413,226	28.34/0.902	1,000,000
360/garden	24.79/0.806	227,532	25.41/0.838	1,000,000

Table 5. **Dense and sparse point cloud comparison.** We present results on two scenes from the Blender [39] and Mip-NeRF 360 [4] datasets. We also show the number of vertices of the tetrahedra field – including the randomly sampled points.

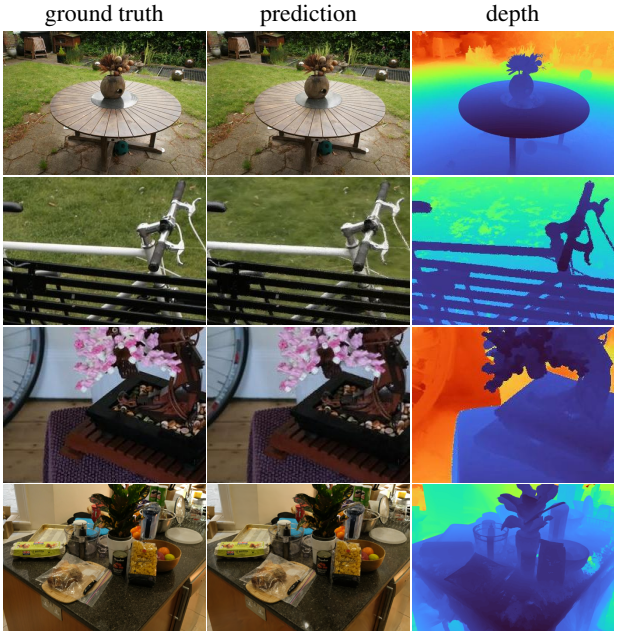


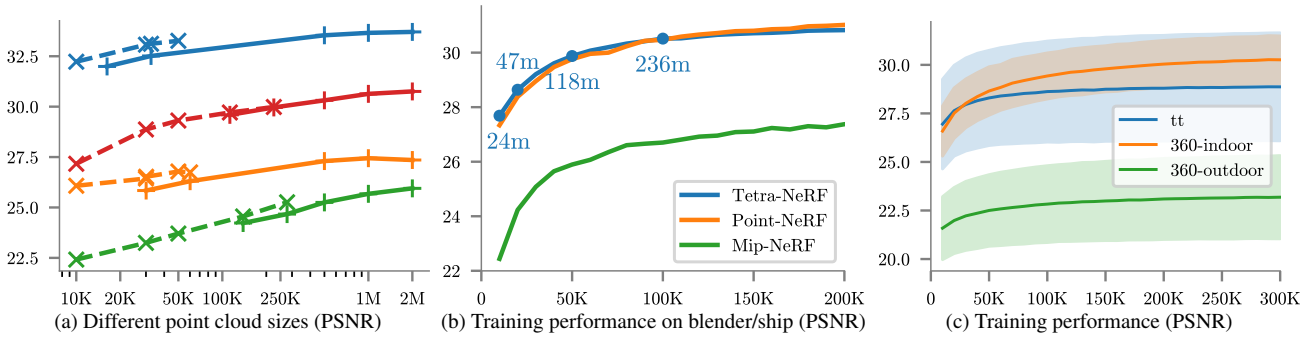
Figure 8. **Mip-NeRF 360 results.** **1st row**: we show the results on the *garden* scene. **2nd**: we can see that Tetra-NeRF has some problems rendering grass. **3rd**: Tetra-NeRF is able to represent the delicate texture of the tablecloth well. **4th**: Tetra-NeRF is able to recover fine geometry.

sible because outdoor scenes contain a lot of high-resolution geometries, such as leaves and Stable View Synthesis is not able to overcome noise in the approximate geometry. Tetra-NeRF does not suffer from these problems as we aggregate features in 3D along rays rather than on the surface. Mip-NeRF 360 scores comparably to Tetra-NeRF in terms of PSNR and LPIPS, but it outperforms Tetra-NeRF in terms of SSIM. However, Mip-NeRF 360 implements some tricks designed to boost its performance. On the other hand, our approach is based on vanilla NeRF, and we believe that the performance can be boosted similarly.

Varying the number of input points We conducted a study on the effect of the input point cloud size on the reconstruction quality. We consider both the sparse and the dense point clouds and analyse the performance as we subsample the points or add more points randomly. We evaluate on the *family* and the *truck* scenes from the Tanks and Temples [25] (tt) dataset, and the *room* and the *garden* scenes from the Mip-NeRF 360 [4] dataset. The sizes of sparse point clouds were roughly 16K, 30K, 113K, and 139K respectively and the sizes of dense point clouds were larger than 5M for all scenes. When the size of the original point cloud was larger than the desired size, we selected points uniformly at random, and when smaller, we added new random points as described Section 4.1. We include detailed results in the *Supp. Mat.* In order to save computational resources, we only train the method for 100k iterations. The results are visualised in Figure 9a.

As expected, the performance improves with the number of points used, as it leads to a finer subdivision of the scene around the surface. We can also observe (*e.g.*, from *tt/family*) that the sparse reconstruction leads to a better reconstruction quality at the same point cloud size. The likely explanation is that sampling uniformly at random will be biased towards regions with high density and some regions may be missed or covered extremely sparsely.

Speed of convergence Figures 9b, 9c show the rate of convergence on the *ship* scene from the Blender dataset and aggregated results on the Tanks and Temples [25], and Mip-NeRF 360 (indoor/outdoor) [4] datasets. Similar to Point-NeRF, good results can be achieved quite early in the train-



(a) Different point cloud sizes (PSNR) (b) Training performance on blender/ship (PSNR) (c) Training performance (PSNR)

Figure 9. **a)** shows the PSNR with different sizes of input point clouds. The solid and dashed lines represent the dense and coarse COLMAP reconstructions. The following scenes were evaluated (best to worst): *tt/family*, *360/room*, *tt/truck*, and *360/garden* from Tanks and Temples [25] (tt) and Mip-NeRF 360 [4] (360-indoor/360-outdoor) datasets. As expected, the quality increases with the number of points. **b)** visualizes the training speed (PSNR at different training iterations) on the *blender/ship* scene [39]. It shows Tetra-NeRF training times for various timesteps and compares the rate of convergence with Point-NeRF [70] and Mip-NeRF [3]. Similarly, **c)** shows the rate of convergence aggregated over Tanks and Temples [25] (tt) and Mip-NeRF 360 [4] (360-indoor/360-outdoor) datasets. It shows the average PSNR, and the highlighted area represents the standard deviation.

(a) **Failure case 1**: low point cloud density on the ground (b) **Failure case 2**: too many intersected tetrahedra

Figure 10. **Failure cases. Left**: *barn* scene from [25]. **Right**: *tree hill* scene from [4].

ing process. *E.g.*, on the *blender/ship* scene, in 24 minutes, our method achieves performance similar to Mip-NeRF [3] after several hours of training. The efficiency of the implementation is a large factor in the speed of different methods. *E.g.*, in Instant-NGP [41], the authors invested a significant effort in optimizing GPU memory accesses, and the result is highly optimised efficient code (a main contribution of [41]). There are several places where our implementation can easily be made more efficient. *E.g.*, changing the memory layout (row-major *vs*. column-major) of the tetrahedra field leads to a 10% speedup.

5. Limitations

One drawback of our approach is that the quality of different regions of the rendered scene depends on the density of the point cloud in these regions. If the original point cloud was constructed using 2D feature matching, there can be regions with a very low number of 3D points. We show such an example in Fig. 10a. There are few 3D points on the ground, resulting in blurry renderings for that region.

Another issue is that the current implementation has a limit on the number of intersected tetrahedra per ray. For large or badly structured scenes, where rays intersect many tetrahedra, this can limit the reconstruction quality. Fig. 10b shows such a case on a Mip-NeRF 360 scene [4]. This problem can be addressed by increasing the limit on the number of visited tetrahedra at the cost of higher memory requirements and run-time. Alternatively, coarse-to-fine schemes that start with a coarser tetrahedralisation and prune the space could potentially handle this issue.

6. Conclusion

This paper proposes a novel radiance field representation that, compared to standard voxel-based representations, is easily able to adapt to 3D geometry priors given in the form of a (sparse) point cloud. Our approach elegantly combines concepts from geometry processing (Delaunay triangulation) and triangle-based rendering (ray-triangle intersections) with modern neural rendering approaches. The representation has a naturally higher resolution in the space near surfaces, and the input point cloud provides a straightforward way to initialise the radiance field. Compared to Point-NeRF, a state-of-the-art point cloud-based radiance field representation which uses the same input as our approach, Tetra-NeRF shows clearly better results. Our method performs comparably to state-of-the-art MLP-based methods. The results demonstrate that Tetra-NeRF is an interesting alternative to existing radiance field representations that is worth further investigation. Interesting research directions include adaptive refinement and pruning of the tetrahedralisation and exploiting the fact that the surface of the scene is likely close to some of the triangles in the scene.

Acknowledgements This work was supported by the Czech Science Foundation (GAČR) EXPRO (grant no. 23-07973X), the Grant Agency of the Czech Technical University in Prague (grant no. SGS22/112/OHK3/2T/13), and by the Ministry of Education, Youth and Sports of the Czech Republic through the e-INFRA CZ (ID:90254).

References

[1] Kara-Ali Aliev, Artem Sevastopolsky, Maria Kolos, Dmitry Ulyanov, and Victor Lempitsky. Neural point-based graphics. In *ECCV*, 2020. 2

[2] Pierre Alliez, Simon Giraudot, Clément Jamin, Florent Lafarge, Quentin Mérigot, Jocelyn Meyron, Laurent Saboret, Nader Salman, Shihao Wu, and Necip Fazil Yildiran. Point set processing. In *CGAL User and Reference Manual*. CGAL Editorial Board, 5.5.1 edition, 2022. 5

[3] Jonathan T Barron, Ben Mildenhall, Matthew Tancik, Peter Hedman, Ricardo Martin-Brualla, and Pratul P Srinivasan. Mip-NeRF: A multiscale representation for anti-aliasing neural radiance fields. In *ICCV*, 2021. 1, 2, 4, 5, 7, 8, 9

[4] Jonathan T Barron, Ben Mildenhall, Dor Verbin, Pratul P Srinivasan, and Peter Hedman. Mip-NeRF 360: Unbounded anti-aliased neural radiance fields. In *CVPR*, 2022. 1, 2, 5, 7, 8, 9

[5] Sai Bi, Zexiang Xu, Pratul Srinivasan, Ben Mildenhall, Kalyan Sunkavalli, Miloš Hašan, Yannick Hold-Geoffroy, David Kriegman, and Ravi Ramamoorthi. Neural reflectance fields for appearance acquisition. *arXiv:2008.03824*, 2020. 2

[6] Wenjing Bian, Zirui Wang, Kejie Li, Jia-Wang Bian, and Victor Adrian Prisacariu. NoPe-NeRF: Optimising neural radiance field with no pose prior. *arXiv:2212.07388*, 2022. 2

[7] Mark Boss, Raphael Braun, Varun Jampani, Jonathan T Barron, Ce Liu, and Hendrik Lensch. NeRD: Neural reflectance decomposition from image collections. In *ICCV*, 2021. 2

[8] Chris Buehler, Michael Bosse, Leonard McMillan, Steven Gortler, and Michael Cohen. Unstructured lumigraph rendering. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, 2001. 2

[9] Anpei Chen, Zexiang Xu, Andreas Geiger, Jingyi Yu, and Hao Su. TensoRF: Tensorial radiance fields. In *ECCV*, 2022. 1, 2, 4, 5

[10] Anpei Chen, Zexiang Xu, Xinyue Wei, Siyu Tang, Hao Su, and Andreas Geiger. Factor fields: A unified framework for neural fields and beyond. *arXiv:2302.01226*, 2023. 1, 2, 3

[11] Anpei Chen, Zexiang Xu, Fuqiang Zhao, Xiaoshuai Zhang, Fanbo Xiang, Jingyi Yu, and Hao Su. MVSNeRF: Fast generalizable radiance field reconstruction from multi-view stereo. In *ICCV*, 2021. 2

[12] Shuo Cheng, Zexiang Xu, Shilin Zhu, Zhuwen Li, Li Erran Li, Ravi Ramamoorthi, and Hao Su. Deep stereo using adaptive thin volume representation with uncertainty awareness. In *CVPR*, 2020. 2

[13] Paul E Debevec, Camillo J Taylor, and Jitendra Malik. Modeling and rendering architecture from photographs: A hybrid geometry-and image-based approach. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, 1996. 2

[14] B Delaunay, S Vide, A Lamémoire, and V De Georges. Bulletin de l'academie des sciences de l'URSS. *Classe des sciences mathématiques et naturelles*, 6:793–800, 1934. 2, 3, 4, 5

[15] Boyang Deng, Jonathan T. Barron, and Pratul P. Srinivasan. JaxNeRF: an efficient JAX implementation of NeRF, 2020. 8

[16] Michael S Floater. Generalized barycentric coordinates and applications. *Acta Numerica*, 24:161–214, 2015. 3, 4

[17] Sara Fridovich-Keil, Alex Yu, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. Plenoxels: Radiance fields without neural networks. In *CVPR*, 2022. 1, 2, 4, 5

[18] Yasutaka Furukawa and Jean Ponce. Accurate, dense, and robust multiview stereopsis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32:1362–1376, 2009. 2

[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *ICCV*, 2015. 5

[20] Peter Hedman, Julien Philip, True Price, Jan-Michael Frahm, George Drettakis, and Gabriel Brostow. Deep blending for free-viewpoint image-based rendering. *SIGGRAPH, ToG*, 37(6):1–15, 2018. 2, 8

[21] Peter Hedman, Pratul P Srinivasan, Ben Mildenhall, Jonathan T Barron, and Paul Debevec. Baking neural radiance fields for real-time view synthesis. In *ICCV*, 2021. 2

[22] Vu Hoang Hiep, Renaud Keriven, Patrick Labatut, and Jean-Philippe Pons. Towards high-resolution large-scale multi-view stereo. In *CVPR*, 2009. 2

[23] Michal Jancosek and Tomas Pajdla. Multi-view reconstruction preserving weakly-supported surfaces. In *CVPR*, 2011. 2

[24] Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. Poisson surface reconstruction. In *SGP*, 2006. 2

[25] Arno Knapitsch, Jaesik Park, Qian-Yi Zhou, and Vladlen Koltun. Tanks and temples: Benchmarking large-scale scene reconstruction. *SIGGRAPH, ToG*, 2017. 5, 6, 7, 8, 9

[26] Georgios Kopanas, Julien Philip, Thomas Leimkühler, and George Drettakis. Point-based neural rendering with per-view optimization. In *Computer Graphics Forum*, volume 40, pages 29–43. Wiley Online Library, 2021. 2, 8

[27] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems*, 25, 2012. 7

[28] Jonáš Kulhánek, Erik Derner, Torsten Sattler, and Robert Babuška. ViewFormer: Nerf-free neural rendering from few images using transformers. In *ECCV*, 2022. 2

[29] Patrick Labatut, Jean-Philippe Pons, and Renaud Keriven. Efficient multi-view reconstruction of large-scale scenes using interest points, Delaunay triangulation and graph cuts. In *ICCV*, 2007. 2

[30] Patrick Labatut, J-P Pons, and Renaud Keriven. Robust and efficient surface reconstruction from range data. In *Computer graphics forum*, volume 28, pages 2275–2290. Wiley Online Library, 2009. 2

[31] Chen-Hsuan Lin, Wei-Chiu Ma, Antonio Torralba, and Simon Lucey. BARF: Bundle-adjusting neural radiance fields. In *ICCV*, 2021. 2

[32] Lingjie Liu, Jiatao Gu, Kyaw Zaw Lin, Tat-Seng Chua, and Christian Theobalt. Neural sparse voxel fields. *Advances in Neural Information Processing Systems*, 33:15651–15663, 2020. 1, 2, 3, 5, 6, 7

[33] Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. On the variance of the adaptive learning rate and beyond. In *International Conference on Learning Representations*, 2020. 5

[34] Stephen Lombardi, Tomas Simon, Jason Saragih, Gabriel Schwartz, Andreas Lehrmann, and Yaser Sheikh. Neural volumes: learning dynamic renderable volumes from images. *SIGGRAPH, ToG*, 38(4):1–14, 2019. 2, 6

[35] William E Lorensen and Harvey E Cline. Marching cubes: A high resolution 3D surface construction algorithm. *SIGGRAPH, ToG*, 1987. 2

[36] Ricardo Martin-Brualla, Noha Radwan, Mehdi SM Sajjadi, Jonathan T Barron, Alexey Dosovitskiy, and Daniel Duckworth. NeRF in the wild: Neural radiance fields for unconstrained photo collections. In *CVPR*, 2021. 2

[37] Nelson Max. Optical models for direct volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 1(2):99–108, 1995. 1, 3

[38] Moustafa Meshry, Dan B. Goldman, Sameh Khamis, Hugues Hoppe, Rohit Pandey, Noah Snavely, and Ricardo Martin-Brualla. Neural rerendering in the wild. In *CVPR*, June 2019. 2

[39] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. NeRF: Representing scenes as neural radiance fields for view synthesis. In *ECCV*, 2020. 1, 2, 3, 4, 5, 6, 7, 8, 9

[40] Tomas Möller. A fast triangle-triangle intersection test. *Journal of Graphics Tools*, 2(2):25–30, 1997. 2

[41] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *SIGGRAPH, ToG*, 2022. 1, 2, 3, 4, 5, 9

[42] M. Nießner, M. Zollhöfer, S. Izadi, and M. Stamminger. Real-time 3D reconstruction at scale using voxel hashing. *SIGGRAPH, ToG*, 2013. 4

[43] Steven G Parker, James Bigler, Andreas Dietrich, Heiko Friedrich, Jared Hoberock, David Luebke, David McAllister, Morgan McGuire, Keith Morley, Austin Robison, et al. OptiX: a general purpose ray tracing engine. *ACM Transactions on Graphics (ToG)*, 29(4):1–13, 2010. 2, 4, 5

[44] Songyou Peng, Michael Niemeyer, Lars Mescheder, Marc Pollefeys, and Andreas Geiger. Convolutional occupancy networks. In *ECCV*, 2020. 2

[45] Bui Tuong Phong. Illumination for computer generated pictures. *Communications of the ACM*, 18(6):311–317, 1975. 2

[46] Ben Poole, Ajay Jain, Jonathan T Barron, and Ben Mildenhall. DreamFusion: Text-to-3D using 2D diffusion. *arXiv:2209.14988*, 2022. 2

[47] Christian Reiser, Songyou Peng, Yiyi Liao, and Andreas Geiger. KiloNeRF: Speeding up neural radiance fields with thousands of tiny mlps. In *ICCV*, 2021. 2

[48] Christian Reiser, Richard Szeliski, Dor Verbin, Pratul P Srinivasan, Ben Mildenhall, Andreas Geiger, Jonathan T Barron, and Peter Hedman. MERF: Memory-efficient radiance fields for real-time view synthesis in unbounded scenes. *arXiv:2302.12249*, 2023. 2

[49] Jeremy Reizenstein, Roman Shapovalov, Philipp Henzler, Luca Sbordone, Patrick Labatut, and David Novotny. Common objects in 3D: Large-scale learning and evaluation of real-life 3D category reconstruction. In *ICCV*, 2021. 2

[50] Gernot Riegler and Vladlen Koltun. Free view synthesis. In *ECCV*, 2020. 2

[51] Gernot Riegler and Vladlen Koltun. Stable view synthesis. In *CVPR*, 2021. 2, 7, 8

[52] Radu Alexandru Rosu and Sven Behnke. Permutosdf: Fast multi-view reconstruction with implicit surfaces using permutohedral lattices. In *CVPR*, 2023. 2

[53] Darius Rückert, Linus Franke, and Marc Stamminger. ADOP: Approximate differentiable one-pixel point rendering. *SIGGRAPH, ToG*, 41(4):1–14, 2022. 2

[54] Johannes Lutz Schönberger and Jan-Michael Frahm. Structure-from-motion revisited. In *CVPR*, 2016. 2, 4, 5

[55] Johannes Lutz Schönberger, Enliang Zheng, Marc Pollefeys, and Jan-Michael Frahm. Pixelwise view selection for unstructured multi-view stereo. In *ECCV*, 2016. 2, 4, 5

[56] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv:1409.1556*, 2014. 7

[57] Cheng Sun, Min Sun, and Hwann-Tzong Chen. Direct voxel grid optimization: Super-fast convergence for radiance fields reconstruction. In *CVPR*, 2022. 1, 2, 4

[58] Matthew Tancik, Vincent Casser, Xinchen Yan, Sabeek Pradhan, Ben Mildenhall, Pratul P Srinivasan, Jonathan T Barron, and Henrik Kretzschmar. Block-NeRF: Scalable large scene neural view synthesis. In *CVPR*, 2022. 2

[59] Matthew Tancik, Pratul Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan Barron, and Ren Ng. Fourier features let networks learn high frequency functions in low dimensional domains. *Advances in Neural Information Processing Systems*, 33:7537–7547, 2020. 4

[60] Matthew Tancik, Ethan Weber, Evonne Ng, Ruilong Li, Brent Yi, Justin Kerr, Terrance Wang, Alexander Kristoffersen, Jake Austin, Kamyar Salahi, et al. Nerfstudio: A modular framework for neural radiance field development. *arXiv:2302.04264*, 2023. 2, 5

[61] Chengzhou Tang and Ping Tan. BA-net: Dense bundle adjustment networks. In *International Conference on Learning Representations*, 2019. 2

[62] Dor Verbin, Peter Hedman, Ben Mildenhall, Todd Zickler, Jonathan T Barron, and Pratul P Srinivasan. Ref-NeRF: Structured view-dependent appearance for neural radiance fields. In *CVPR*, 2022. 2

[63] Sudheendra Vijayanarasimhan, Susanna Ricco, Cordelia Schmid, Rahul Sukthankar, and Katerina Fragkiadaki. SfM-Net: Learning of structure and motion from video. *arXiv:1704.07804*, 2017. 2

[64] Can Wang, Menglei Chai, Mingming He, Dongdong Chen, and Jing Liao. Clip-NeRF: Text-and-image driven manipulation of neural radiance fields. In *CVPR*, 2022. 2

[65] Peng Wang, Lingjie Liu, Yuan Liu, Christian Theobalt, Taku Komura, and Wenping Wang. NeuS: Learning neural implicit surfaces by volume rendering for multi-view reconstruction. *Advances in Neural Information Processing Systems*, 34:27171–27183, 2021. 2

[66] Qianqian Wang, Zhicheng Wang, Kyle Genova, Pratul P. Srinivasan, Howard Zhou, Jonathan T. Barron, Ricardo Martin-Brualla, Noah Snavely, and Thomas Funkhouser. IBRNet: Learning multi-view image-based rendering. In *CVPR*, 2021. 2

[67] Yiming Wang, Qin Han, Marc Habermann, Kostas Daniilidis, Christian Theobalt, and Lingjie Liu. NeuS2: Fast learning of neural implicit surfaces for multi-view reconstruction. *arXiv:2212.05231*, 2022. 2

[68] Zirui Wang, Shangzhe Wu, Weidi Xie, Min Chen, and Victor Adrian Prisacariu. NeRF–: Neural radiance fields without known camera parameters. *arXiv:2102.07064*, 2021. 2

[69] Daniel N Wood, Daniel I Azuma, Ken Aldinger, Brian Curless, Tom Duchamp, David H Salesin, and Werner Stuetzle. Surface light fields for 3D photography. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, 2000. 2

[70] Qiangeng Xu, Zexiang Xu, Julien Philip, Sai Bi, Zhixin Shu, Kalyan Sunkavalli, and Ulrich Neumann. Point-NeRF: Point-based neural radiance fields. In *CVPR*, 2022. 1, 2, 3, 5, 6, 7, 9

[71] Yao Yao, Zixin Luo, Shiwei Li, Tian Fang, and Long Quan. MVSNet: Depth inference for unstructured multi-view stereo. In *ECCV*, 2018. 2

[72] Lior Yariv, Jiatao Gu, Yoni Kasten, and Yaron Lipman. Volume rendering of neural implicit surfaces. *Advances in Neural Information Processing Systems*, 34:4805–4815, 2021. 2

[73] Lior Yariv, Peter Hedman, Christian Reiser, Dor Verbin, Pratul P Srinivasan, Richard Szeliski, Jonathan T Barron, and Ben Mildenhall. BakedSDF: Meshing neural SDFs for real-time view synthesis. *arXiv:2302.14859*, 2023. 2

[74] Alex Yu, Ruilong Li, Matthew Tancik, Hao Li, Ren Ng, and Angjoo Kanazawa. PlenOctrees for real-time rendering of neural radiance fields. In *ICCV*, 2021. 1, 2

[75] Alex Yu, Vickie Ye, Matthew Tancik, and Angjoo Kanazawa. PixelNeRF: Neural radiance fields from one or few images. In *CVPR*, 2021. 2

[76] Zehao Yu, Songyou Peng, Michael Niemeyer, Torsten Sattler, and Andreas Geiger. MonoSDF: Exploring monocular geometric cues for neural implicit surface reconstruction. In *Advances in Neural Information Processing Systems*, 2022. 2

[77] Kai Zhang, Gernot Riegler, Noah Snavely, and Vladlen Koltun. NeRF++: Analyzing and improving neural radiance fields. *arXiv:2010.07492*, 2020. 2, 8

[78] Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *CVPR*, June 2018. 5, 6, 8

[79] Chaoqiang Zhao, Qiyu Sun, Chongzhen Zhang, Yang Tang, and Feng Qian. Monocular depth estimation based on deep learning: An overview. *Science China Technological Sciences*, 63(9):1612–1627, 2020. 3