RGB-D Mapping and Tracking in a Plenoxel Radiance Field

Andreas L. Teigen*† Norwegian University of Science and Technology

Yeonsoo Park*‡ Mobiltech

Annette Stahl § Norwegian University of Science and Technology

Rudolf Mester ¶ Norwegian University of Science and Technology

arXiv:2307.03404v1 [cs.CV] 7 Jul 2023

ABSTRACT

Building on the success of Neural Radiance Fields (NeRFs), recent years have seen significant advances in the domain of novel view synthesis. These models capture the scene's volumetric radiance field, creating highly convincing dense photorealistic models through the use of simple, differentiable rendering equations. Despite their popularity, these algorithms suffer from severe ambiguities in visual data inherent to the RGB sensor, which means that although images generated with view synthesis can visually appear very believable, the underlying 3D model will often be wrong. This considerably limits the usefulness of these models in practical applications like Robotics and Extended Reality (XR), where an accurate dense 3D reconstruction otherwise would be of significant value. In this technical report, we present the vital differences between view synthesis models and 3D reconstruction models. We also comment on why a depth sensor is essential for modeling accurate geometry in general outward-facing scenes using the current paradigm of novel view synthesis methods. Focusing on the structure-from-motion task, we practically demonstrate this need by extending the Plenoxel radiance field model: Presenting an analytical differential approach for dense mapping and tracking with radiance fields based on RGB-D data without a neural network. Our method achieves state-of-the-art results in both the mapping and tracking tasks while also being faster than competing neural network-based approaches.

Index Terms: Radiance fields—RGB-D—3D reconstruction—Mapping Tracking—Voxel grid—Novel view synthesis

In the computer vision field, dense maps can be defined as a 3D surface map generated using all observed pixels in an image set, creating a point cloud so dense as to make interpolation between points trivial or directly creating a continuous representation of the observed scene surface. Dense maps are very useful for many tasks such as path planning, collision avoidance in robotics, interaction between real-world geometry and digital objects in *extended reality* (XR) experiences, and as maps for human inspection. The RGB-D sensor is a popular sensor choice for creating dense maps, but dealing with noise and missing measurements in sensor data can be challenging. This is because nearly all image pixels are utilized for processing each frame and achieving the necessary data association, even with a known pose, can be computationally demanding. In many cases, efficient processing of such data may require the utilization of a GPU. Dense maps can also be created from only RGB images, but this requires some assumption regarding areas of the scene containing no gradients. Such an assumption can, for instance, be smooth or planar surfaces [19], or inferred from a trained neural network [6]. Although dense mapping generally requires more effort than sparse/point feature-based mapping, tracking the camera motion in a known environment with a good dense map is very accurate as all image and model information can be used in an image-to-model alignment. This is opposed to sparse tracking, which is limited to using only easily recognizable points of the image/model. Performing dense tracking is therefore positioned to result in a smoother and more accurate trajectory.

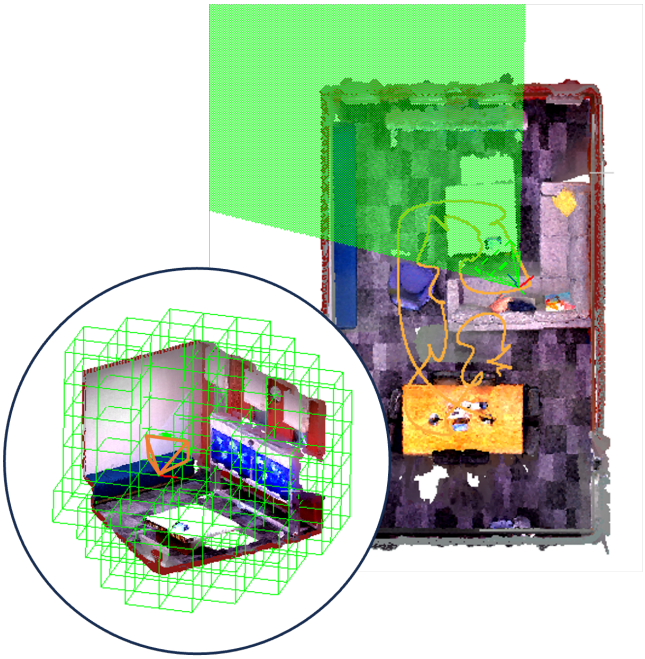

Figure 1: Visualization of generated map and estimated trajectory on Office-3 of Replica dataset. The figure illustrates the voxel grid rediance field of the map.

Learned radiance field algorithms, like *Neural Radiance Fields* (NeRF) [16], were introduced as a mathematically simple formulation for creating photo-realistic dense models using RGB images. This works very well for some types of scenes, especially object-centric and inward-facing scenes, and scenes with significant gradient coverage. These types of scenes almost remove the need to make assumptions in the modeling stage, reducing these algorithms' applicability to selected scenes or very restricted camera poses. Imbuing the NeRF optimization scheme to allow for full use of the RGB-D sensor will remove ambiguities due to gradient-less regions of the scene and make it more useful for practical application.

Basing our work on the Plenoxel algorithm [9]: The analytical radiance field representation which does not use a neural network but rather a voxel grid representation. We build on this representation to make it more applicable for practical tasks by augmenting the algorithm with mapping based on the RGB-D sensor. We then implement a pose optimization algorithm to track a camera throughout the voxel grid, using volumetric information for dense image-to-model alignment based on the radiance field rendering equations. We show the analytical derivations of all equations used for both optimization strategies and implement them in CUDA for fast computation times. Leveraging the inherent speed of the Plenoxel model of more than

*The two authors contributed equally to this paper.
†e-mail: andreas.l.teigen@ntnu.no
‡e-mail: yspark@mobiltech.io
§e-mail: annette.stahl@ntnu.no
¶e-mail: rudolf.mester@ntnu.no

two magnitudes faster than the original NeRF algorithm, we create a very efficient mapping algorithm and a real-time, highly accurate tracking algorithm.

Our contributions are summed up as follows:

- Comment on the differences between models for novel view synthesis and models for 3D reconstruction and why this might lead to problems directly attempting to use NeRF for many practical applications like robotics and XR.
- Optimization of a voxel-based radiance field based on RGB-D data, with the analytical derivative equations and an implementation in CUDA for efficient runtimes.
- Create an analytical derivative for tracking optimization of a camera position in a voxel-based radiance field based on RGB-D data, including CUDA implementation.
- Showing improvement in both mapping and tracking results compared to existing radiance field mapping and tracking methods given the same time constraints.

1 Related Work

1.1 Dense mapping and tracking

Despite the usefulness of dense visual mapping and the accuracy/robustness of dense tracking [8], dense mapping and tracking methods [12, 19] have received relatively little attention compared to their sparse counterparts [7, 18] This is mostly due to their technical and computational complexity as well as their reliance on either a depth sensor or an assumption for image regions with no gradients. Despite this, there have still been several noteworthy papers on the topic in the past few years, the most seminal of which is undoubtedly the DTAM algorithm [19] by Newcombe et al.. They proposed the idea of performing dense SLAM by separating the problem into alternating the tasks of updating a dense 3D model and tracking the camera pose by aligning the camera image to the model using randomly sampled image pixels. Kinect-fusion [12] built on the premise of alternating mapping and tracking, but distinct from the previous algorithm was the reliance on a depth-only sensor and representing the entire model as a truncated signed distance field, using the iterative closest point algorithm for pose optimization. A more recent RGB-D SLAM paper: Bad-SLAM [22] creates a dense map based on surfels instead of individual pixels. The surfels' position, orientation, and size are optimized with a clever implementation of a bundle adjustment variation for more efficient computation.

The popularity of dense mapping and tracking has increased through the use of deep learning-based methods [3, 26], often by the use of pixel-level depth estimators. Code-slam [3] does this by training a variational auto-encoder offline and conditioning a pixel depth estimation based on intensity images, resulting in a depth image estimation algorithm that runs in real-time. Droid-SLAM [26] is a surprisingly robust method compared to other pre-trained deep learning-based SLAM algorithms, providing good results on several different datasets, even for datasets not included in the training set.

1.2 Radiance fields

Neural Radiance Fields (NeRF) [16] is a new technology that has taken the research community by storm. It allows for the creation of photo-realistic, dense, volumetric 3D view synthesis models of real-world objects, only requiring posed RGB images of the scene. NeRF natively stores a model in an extremely compressed format as a neural network, specifically a *Multi Layer Perceptron* (MLP). The model is trained using ray-based volumetric rendering functions for training on images using multi-view consistency to produce a globally consistent model. Ever since NeRF was proposed, there has been a lot of research targeting expanding its applicability by: Targeting a wider range of scenes, performing camera pose estimation based on a pre-trained model, and even simultaneous localization and mapping (SLAM) where the model expansion and pose estimation are alternately optimized.

Speed and efficiency. Despite the original NeRF's remarkable capabilities [16], early versions were hindered by limitations of slow convergence rates. Numerous strategies [2, 20, 31] have been proposed to enhance its efficiency for both training and rendering of view synthesis models. Recently there have emerged papers [9, 17] which show that it is possible to improve the training and/or rendering speed by several orders of magnitude by augmentation or completely changing the model representation. Muller et al. [17] do this by utilizing a multi-resolution hash encoding with trainable parameters that allows disambiguation of hash collisions and consequently allows a smaller MLP to represent a larger scene size. Fridovich et al. [9] completely discards the MLP, rather favoring a voxel grid where each vertex is modeled by one density value and 27 color values (nine per color channel). All information is directly accessed instead of requiring a forward pass through a neural network, and samples at arbitrary locations are retrieved by linear interpolation of the eight closest voxels. This allows for greatly improving its efficiency at the cost of increased memory consumption.

Pose optimization. Wang et al. [27] have demonstrated the possibility to optimize intrinsic camera parameters along with the neural radiance fields. Followed by Yen et al. [29] who proposed camera pose optimization based on a trained NeRF model, using interest points to make sure the optimization is performed on the areas of gradient in the image. Lin et al. [15] went further and showed that given a coarse initialization of the frame poses, the pose optimization for all training images could be done simultaneously during the model's training. However, to improve both the radius of convergence and performance, gradually, more layers of the frequency encodings were introduced during the optimization process, which required human supervision due to scene-specific variations.

Mapping and tracking. Attributed to NeRF's simple formulation of dense mapping and its small storage size, several authors have attempted to use NeRF as a map representation in dense mapping and tracking problems, normally in the form of Simultaneous Localization And Mapping (SLAM) algorithms [25, 28, 33]. The first to attempt this for real-time processing was Sucar et al. [25], who used the original NeRF model [16] and an RGB-D sensor both for speed and to solve the geometric ambiguity problem covered in Sec. 2. Although NeRFs are slow to train, the significant convergence speed of the earlier epochs is a great help in attaining passable real-time performance. They showed good results in a room-scale scene, but an increase in scene size would lead to catastrophic forgetting [10]. Zhu et al. [33] addressed the weakness of catastrophic forgetting by storing values from the learned model in a grid which then would be queried by pre-trained decoder networks. Their follow-up work in [32] continued this trend of offloading tasks to pre-trained networks, replacing the depth sensor with two convolutional networks predicting a depth map and a normal map, respectively. Vox-Fusion [28] adopts a similar concept but achieves significantly reduced memory consumption by dynamically allocating sparse voxels based on an octree structure. It trains network from scratch during the process, diverging from using pre-trained models, which makes generalization poorly to different types of scenes, making them less useful in practical scenarios. Both [28, 32] integrate the signed distance function (SDF) to better fit the network to the surface.

Less relevant but noteworthy papers use NeRF models as a visually pleasing back-end on existing sparse SLAM algorithms [4, 21].

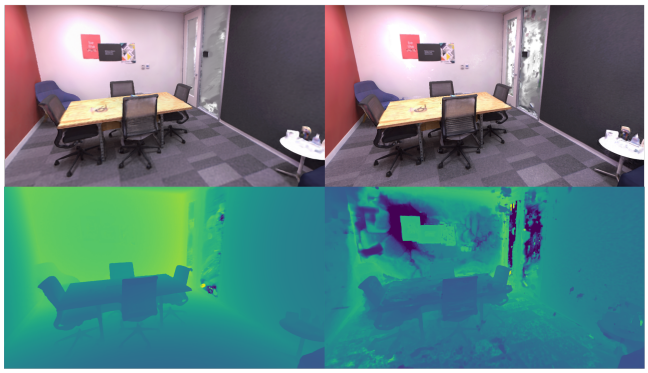

Figure 2: Color and depth rendering from two different radiance fields trained on office-2 in the Replica dataset: One trained with an RGB-D sensor(left column) and the other one trained with only an RGB sensor(right column), both trained on a large set of images viewing the scene from several different angles.

	RGB PSNR↑	Depth diff. (m/pixel)↓
RGB	30.612	0.6971
RGB-D	28.570	0.0090

Table 1: Comparison of map geometry accuracy between RGB and RGB-D versions of Plenoxel on Office-2 of Replica dataset.

2 Novel view synthesis vs 3D reconstruction

2.1 Novel view synthesis is not 3D reconstruction

We want to emphasize the difference between a model for novel view synthesis and a 3D reconstruction. A 3D reconstruction is a task that aims at recreating an accurate representation of the geometry of the target object/scene. View synthesis, on the other hand, is for recreating an accurate representation of the appearance of the target object/scene. While these two statements sound similar, they are not the same. In their current form [1,9,16], novel view synthesis models are trained based on the RGB data from a limited number of views and will only produce reasonable novel appearance renderings from inside the distribution of camera poses that were used for the training of the model. In the case of an image rendered from a camera pose outside of the training pose distribution, the image will most likely not correspond to the expected result. This is because the underlying geometry from a series of images is ambiguous for areas of the images with no gradients, something which has been well-known in the photogrammetry community for decades. However, although the spatial geometry of such a region is unknown, the appearance of the same image region is known, at least locally in the training pose distribution, allowing for the production of a model which can produce highly convincing images with incorrect underlying geometry.

This is exactly the reason why radiance models can be so deceiving, but it also gives us a glimpse of our own faulty assumptions when determining scene geometry. If we are presented with an image from a novel view synthesis model that produces an accurate appearance but contains an incorrect underlying geometry, we will not perceive the incorrect model geometry but rather assume a geometry closer to the true scene geometry because that intuitively makes more sense to us. A good example of this can be seen in Fig. 2 Although the model trained with the RGB sensor looks good in the color rendering, we can see that the underlying geometry is very different from what we would expect compared to the radiance field trained with the RGB-D sensor.

The problem of ambiguous geometry can be mitigated either by training on a scene with color gradients covering significant parts of the scene, thereby reducing the total possible ambiguity, or by training the model with training images covering a 360-degree sphere around the model (inward-facing scene), effectively performing space carving [14]. This will encourage the model to approach the true scene geometry. For many practical applications like XR and robotics, both of these mitigation strategies are often infeasible and/or impossible. Consider, for instance, an example of a room with mono-colored walls (outwards-facing scene). There will be little to no color gradient on the walls, and capturing all objects in the room from every angle puts significant requirements on the data capture process.

However, if a depth sensor is adopted, this problem can be eradicated as this allows for the estimation of a geometrically correct model with just a very limited coverage of the scene. A second, less ideal solution is to adopt an assumption on the geometry. For instance, assume areas with no color gradient are smooth, planar, or other more advanced solutions like inference from a pre-trained neural network. This would reduce the ambiguity of the geometry but also induce a bias that might either help or hinder the 3D reconstruction process depending on the assumption's correctness for any particular scene.

The RGB-D sensor is becoming more ubiquitous, providing an unbiased representation of the geometry and allowing easy integration. Additionally, the depth signal is often a lower frequency signal than the color signal, and therefore, it also helps increase the radius of convergence when using dense image-to-model alignment. For these two reasons, we focus on the use of the RGB-D sensor in this work.

3 Method

3.1 Overview

Our proposed algorithm builds on the voxel grid representation for radiance field optimization [9,30] and is split into two separate parts: One offline mapping algorithm using RGB-D data with known poses to create a model of the scene and one online tracking algorithm using the map in an image-to-model alignment scheme for pose optimization. All the partial derivatives of the loss function needed for both tasks are analytically calculated and implemented in CUDA for fast processing on GPUs.

3.2 Volumetric Rendering

All radiance field learning approaches share the volumetric rendering equations which were initially outlined in [13]. They explain how a pixel color is rendered based on a volume containing a continuously valued implicit density and color functions. This is done by integrating over the pixel ray from the rendering view-port into the volume, where for any infinitesimal point in space, a larger density corresponds to returning more of the color of that point. The total amount of color that can be returned gradually diminishes based on a saturation quota. This work presents a practical implementation of these formulas, requiring a discrete sampling of the volume, and therefore we only present the discretized equations for the volume rendering.

Color rendering. Let C represent the RGB color value of a single pixel, and $\hat{C}$ denote the rendered color of that pixel from the radiance field model. $\hat{C}$ is obtained by accumulating N rendered sample values computed by the sampled densities σ_i and color values c_i for sample points $\vec{p}_i$ along the ray $\vec{r} = \vec{o} + t_i \cdot \vec{d}$, $i \in [0, 1, ..., N]$, where $\vec{o}$ is the camera center and $\vec{d}$ is the ray direction from camera center to the pixel on the image plane and t_i is the distance from the camera center. The distances between samples are denoted as δ_i. The discrete rendering equations can then be expressed:

Figure 3: Rendered RGB images on the same camera position from trained radiance field under Scan-net scene0207. From left to right, the order is as follows: ground truth RGB image, from ours, from Vox-Fusion, and from NICE-SLAM.

$$\hat{C}(\vec{\sigma},\vec{c}) = \sum_{i=1}^{N} T_i(1-\exp(-\sigma_i\delta_i))c_i, \tag{1}$$

where,

$$T_i = \exp\left(-\sum_{j=1}^{i-1}\sigma_j\delta_j\right). \tag{2}$$

The value T_i represents the amount of light transmitted to sample i along the ray $\vec{r}$. This is essentially the remaining light of a quota that are not reflected by all the previous samples $j = [1,..,i-1]$.

Depth rendering. Let D be the measured distance from the camera center to the scene surface along a ray starting from the camera center and passing through the pixel, and $\hat{D}$ be the expected distance attained by volumetrically sampling density values along the ray $\vec{r}$ in the radiance field. The distance $\hat{D}$ can be estimated in an analogous way to the color rendering by treating the sample distance t_i from the camera as a sample color value c_i in Eq. (1), resulting in:

$$\hat{D}(\vec{\sigma}) = \sum_{i=1}^{N} T_i(\vec{\sigma})(1-\exp(-\sigma_i\delta_i))t_i, \tag{3}$$

with $T_i(\vec{\sigma})$ as defined in Eq. (2).

3.3 Mapping

Model representation. Instead of using a neural network as a model representation of the radiance field as was proposed in [16], we utilize the Plenoxel representation [9], which represents the radiance field as a sparse voxel grid utilizing tri-linear interpolation to produce a continuously valued implicit density and color functions, see Fig. 4. Each vertex in the voxel grid is represented by 28 scalar values: one density value and nine values per color channel. The multiple values per color channel make up the coefficients of spherical harmonics functions, allowing for modeling view dependencies caused by Lambertian surfaces and specular reflections. The main advantage of using a voxel grid compared to a single neural network is the significant reduction of computational effort needed in both training and inference of the model. This speedup mainly comes from needing only to reference the vertices that directly influence a sample ray. This is compared to the neural network that needs to reference all values in the neural network for sampling every single sample point on the ray, which incurs a heavy time penalty. Practically the voxel grid representation reduces the training time by two orders of magnitude and allows for real-time image rendering [9]. Upscaling the resolution of the voxel grid during training while performing dynamic sparsification of the empty areas increases the model's fidelity while simultaneously constraining memory usage [30].

RGB-D Mapping. Mapping based on RGB-D data involves simultaneous optimization based on color value C and depth value D. We convey the color and depth residuals rendered for M sample pixels as photometric and geometric losses, computed using mean squared error (MSE), which is described as follows:

$$\mathscr{L}_p(\hat{C}(\sigma,c)) = \frac{1}{M}\sum_{i=1}^{M}||\hat{C}(\sigma_i,c_i)-C||^2, \tag{4}$$

$$\mathscr{L}_g(\hat{D}(\sigma)) = \frac{1}{M}\sum_{i=1}^{M}||\hat{D}(\sigma)-D||^2. \tag{5}$$

This is done by non-linear optimization based on the differentiation of Eq. (1) and Eq. (3) with respect to all color $\vec{c}$ and density $\vec{\sigma}$ values comprising every sampled vertex of radiance field grid. We continue by showing the analytical derivatives needed for optimizing these functions: Partial derivatives of the photometric loss L_p, and geometric loss L_g with respect to the variables (σ_i, c_i) gives us:

$$\frac{\partial\mathscr{L}_p(\hat{C}(\sigma,c))}{\partial(\sigma,c)} = \frac{\partial\mathscr{L}_p(\hat{C})}{\partial\hat{C}}\left(\frac{\partial\hat{C}(\sigma,c)}{\partial c_i}+\frac{\partial\hat{C}(\sigma,c)}{\partial\sigma_i}\right), \tag{6}$$

$$\frac{\partial\mathscr{L}_g(\hat{D}(\sigma))}{\partial(\sigma)} = \frac{\partial\mathscr{L}_p(\hat{D})}{\partial\hat{D}}\frac{\partial\hat{C}(\sigma)}{\partial\sigma}. \tag{7}$$

The derivatives of the rendering color $\hat{C}$ are calculated in [30], but for the sake of completeness, we include them here:

$$\frac{\partial\hat{C}}{\partial c_i}(\sigma,c) = T_i(\sigma)(1-\exp(-\sigma_i\delta_i)), \tag{8}$$

$$\frac{\partial\hat{C}}{\partial\sigma_i}(\sigma,c) = \delta_i\left[c_iT_{i+1}(\sigma)-\hat{C}+\sum_{j=0}^{i}c_jT_j(1-\exp(-\sigma_j\delta_j))\right]. \tag{9}$$

The derivative of the depth $\hat{D}$ follows an approach analogous to that in Eq. (9). The only difference is that the color value c_i is replaced with the distance value t_i from the camera center to the sample point:

$$\frac{\partial\hat{D}}{\partial\sigma_i}(\sigma) = \delta_i\left[t_iT_{i+1}(\sigma)-\hat{D}+\sum_{j=0}^{i}t_j(T_j(1-\exp(-\sigma_j\delta_j))\right]. \tag{10}$$

As the partial derivatives for equations are all analytically defined, it allows us to implement them directly in a custom CUDA kernel. Building on the code developed for Plenoxel [9] to boost computational speed.

The final loss function we employ during mapping is represented as follows:

$$\mathscr{L} = \mathscr{L}_p + w_d\mathscr{L}_g, \tag{11}$$

where w_d is the scaling factor of the geometric loss. By performing simultaneous and iterative optimization based on the loss function of a sufficiently large random set of rays, the entire scene can be effectively optimized based on an RGB-D image set with sufficient coverage of the scene.

Methods	Metric	Room-0	Room-1	Room-2	Office-0	Office-1	Office-2	Office-3	Office-4	Avg.	time(avg.)(ms)
Vox-Fusion [28]	**ATE**[m]↓	0.0042	0.0036	0.0090	0.6539	0.0029	0.0038	0.0042	0.0046	0.0046	
	RPE$_t$[m]↓	0.00391	**0.0039**	0.0088	0.0500	0.0032	0.0036	0.0039	0.0050	0.0046	581.35
	RPE$_a$[°]↓	0.1076	0.1275	0.4034	2.6541	0.1501	0.1331	0.1316	0.1471	0.1715	
Ours[1]	**ATE**[m]↓	**0.0017**	0.0036	**0.0020**	**0.0077**	**0.0020**	**0.0027**	**0.0022**	**0.0027**	**0.0031**	
	RPE$_t$[m]↓	**0.0008**	0.0059	**0.0013**	**0.0034**	**0.0012**	**0.0009**	**0.0008**	**0.0011**	**0.0019**	419.86
	RPE$_a$[°]↓	**0.0191**	**0.0736**	**0.0483**	**0.1564**	**0.0440**	**0.0282**	**0.0199**	**0.0278**	**0.0522**	
NICE-SLAM [33]	**ATE**[m]↓	0.0119	0.0220	0.0334	**0.0100**	0.0042	0.0093	0.0862	0.0573	0.0293	
	RPE$_t$[m]↓	0.0170	0.0229	0.0215	0.0143	0.0059	0.0131	0.0445	0.0305	0.0212	149.57
	RPE$_a$[°]↓	0.3267	0.5103	1.4032	0.2700	0.2037	0.3247	1.0076	0.5058	0.5690	
Ours[2]	**ATE**[m]↓	**0.0093**	**0.0061**	**0.0179**	0.0117	**0.0027**	**0.0056**	**0.0072**	**0.0061**	**0.0083**	
	RPE$_t$[m]↓	**0.0065**	**0.0090**	**0.0108**	**0.0055**	**0.0025**	**0.0033**	**0.0046**	**0.0039**	**0.0058**	145.20
	RPE$_a$[°]↓	**0.1494**	**0.1903**	**0.4833**	**0.2034**	**0.1196**	**0.1116**	**0.1201**	**0.1113**	**0.1861**	

Table 2: Trajectory estimation results on the Replica dataset. The lowest errors are indicated in bold. Time here stands for tracking time per frame.

	RGB PSNR↑	Depth diff. (m/pixel)↓
Vox-Fusion [28]	19.379	0.0705
NICE-SLAM [33]	18.455	0.0514
Ours	24.411	0.0469

Table 3: comparison of map geometry accuracy on ScanNet. The values are average through 5 sequences reported in Tab. 4.

3.4 Tracking

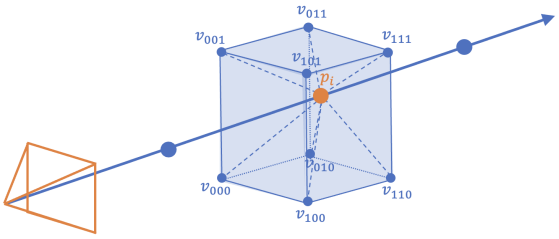

Figure 4: Illustration of trilinear interpolation on sample point p_i lying on the sample ray in grid base radiance field.

Until now, we have assumed known and fixed camera poses while treating the model $(\vec{\sigma},\vec{c})$ parameters as variables. To estimate pose on a model build apriori, we use the same volumetric equations but consider the pose as the variable and the model parameters as static. The objective is then to perform image-to-model alignment using the fixed volumetric color and density parameters from the radiance field, referencing the RGB-D input. In this context, we aim to determine the camera's pose by finding the derivative of color and depth rendering with respect to the ray $\vec{r}=\vec{o}+t_i\vec{d}$, which is determined by the camera pose. This can be done by leveraging the use of an alternative version of the chain rule:

$$\begin{aligned}\frac{\partial \hat{C}}{\partial \vec{r}} &= \sum_{i=1}^{N} \frac{\partial \hat{C}}{\partial \vec{p}_i}\frac{\partial \vec{p}_i}{\partial \vec{r}} \\ &= \sum_{i=i}^{N}\left(\frac{\partial \hat{C}}{\partial \sigma_i}\frac{\partial \sigma_i}{\partial \vec{p}_i}+\frac{\partial \hat{C}}{\partial c_i}\frac{\partial c_i}{\partial \vec{p}_i}\right)\frac{\partial \vec{p}_i}{\partial \vec{r}}\end{aligned} \tag{12}$$

$$\frac{\partial \hat{D}}{\partial \vec{r}} = \sum_{i=i}^{N}\frac{\partial \hat{D}}{\partial \sigma_i}\frac{\partial \sigma_i}{\partial \vec{p}_i}\frac{\partial \vec{p}_i}{\partial \vec{r}}. \tag{13}$$

The partial derivatives $\frac{\partial \hat{C}}{\partial \sigma_i}$, $\frac{\partial \hat{C}}{\partial c_i}$, $\frac{\partial \hat{D}}{\partial \sigma_i}$ are already given in equations 8, 9 and 10 and are shared by all volumetric radiance field methods while $\frac{\partial \sigma_i}{\partial \vec{p}_i}$, $\frac{\partial c_i}{\partial \vec{p}_i}$ and $\frac{\partial \sigma_i}{\partial \vec{p}_i}$ are model specific. For our chosen representation, this is a function based on the tri-linear interpolation functions. As they are rather messy functions, the full analytical equations are given in Appendix A.

The derivative of a sample point $\vec{p}_i$ with respect to the ray $\vec{r}$ can further be broken down as:

$$\frac{\partial \vec{p}_i}{\partial \vec{r}} = \left[\frac{\partial \vec{p}_i}{\partial \vec{o}}, \frac{\partial \vec{p}_i}{\partial \vec{d}}\right] = [1, t_i]. \tag{14}$$

This allows for direct optimization of the position and orientation of the camera parameters.

During tracking, instead of sampling all pixels in an image, M number of rays are randomly sampled from the image at every iteration to increase the computational efficiency of our algorithm.

4 Experiments

4.1 Experimental Setup

We test both mapping and tracking algorithms on synthetic and real indoor datasets of varying sizes and compare them to existing state-of-the-art methods performing similar tasks.

Datasets. The algorithms are tested on two different RGB-D datasets: 1) The Replica dataset [23] containing a total of 18 different synthetic indoor environments with highly accurate sensor data, 2) The ScanNet dataset [5], a real-world dataset also captured from indoor scenes, containing over 1000 unique sequences. As a real-world dataset, the latter inherently includes a significant amount of missing and imperfect depth measurements. To make the results tractable, we use a subset of eight sequences from the Replica dataset processed by [25] and five sequences from the ScanNet dataset. Both of these subsets have been the standard for comparison in previous works [25,28,33].

Comparison algorithms. To the best of our knowledge, no other algorithms are currently performing asynchronous mapping and tracking in a radiance field. Therefore, to draw comparisons with existing approaches, we select NICE-SLAM [33], and Vox-fusion [28] as competing methods, as these are the state-of-the-art algorithms for radiance field-based simultaneous localization and mapping. To make the comparison fair, we modify these algorithms first to optimize the map using ground truth poses with a subset of the sequence images and then perform tracking based on that model.

Metric. For the offline mapping task, we report on two primary metrics to illustrate the geometrical accuracy of the constructed map: the average L1 depth loss in metric units and PSNR score of the

Table 4: Trajectory estimation results on the ScanNet dataset. The lowest errors are indicated in bold. Time here stands for tracking time per frame.

Methods	Metric	0000	0106	0169	0181	0207	Avg.	time(avg.)(ms)
Vox-Fusion* [28]	**ATE**[m]↓	0.0274	0.2424	0.0315	0.0924	0.0323	0.0852	1021.19
	RPE$_t$[m]↓	0.0222	0.0763	0.0284	0.0443	0.0303	0.0403	
	RPE$_a$[°]↓	0.6157	2.7116	0.8805	1.8274	1.0733	1.4217	
NICE-SLAM [33]	**ATE**[m]↓	0.0405	0.1188	0.1952	-	0.0491	0.1009*	360.70
	RPE$_t$[m]↓	0.0360	0.0666	0.0643	-	0.0457	0.0531*	
	RPE$_a$[°]↓	1.0862	2.3725	1.4703	-	1.4871	1.6040*	
Ours	**ATE**[m]↓	**0.0246**	**0.0387**	**0.0165**	**0.0373**	**0.0259**	**0.0286**	350.84
	RPE$_t$[m]↓	**0.0177**	**0.0209**	**0.0160**	**0.0264**	**0.0230**	**0.0208**	
	RPE$_a$[°]↓	**0.5030**	**0.6216**	**0.4484**	**0.7872**	**0.7231**	**0.6166**	

RGB values of the model. These metrics are calculated based on randomly sampled pixels and images from the sequence.

To measure tracking accuracy, we use three key metrics stemming from the root mean square error (RMS): Absolute Trajectory Error (ATE) [24] and the Relative Pose Error for both translation (RPE$_t$) and rotation (RPE$_r$). ATE reflects the global accuracy of the trajectory as it responds delicately on prior drifts. In our image-to-map alignment with a known map, we give more highlights on RPE$_t$ and RPE$_r$, which offer local accuracy over fixed distances or duration. We measure RPE in 1-meter intervals. These tracking metrics were implemented using the "evo" Python package for odometry and SLAM evaluation [11].

Implementation details and parameter selection. All experiments were conducted on a machine equipped with an Intel Core i9-11900KF CPU and an NVIDIA RTX 3090 graphics card, with custom CUDA code for efficient runtimes. For the voxel grid of our method, the experimental setup involved a dynamic grid size with a peak voxel resolution of 512^3, irrespective of the size of the scene. While all images are used for tracking, only every 10th image is used for mapping. Any pixels with invalid depth values are disregarded in both mapping and tracking.

For mapping, we allot our algorithm roughly 6 seconds per frame for mapping, while the competing methods are given up to 18 seconds per frame. The parameters of the competing methods are adjusted to take full advantage of this extra time.

For tracking, the number of sample rays and the number of iterations varied between each method and each dataset. We adhere to the default configurations for these parameters provided by NICE-SLAM and Vox-fusion for each dataset. Meanwhile, we adjusted the parameters for our model to match the per-frame tracking time of the two comparison models.

4.2 Mapping Results

To first demonstrate the point highlighted in Sec. 2, we test the difference in mapping performance between the original Plenoxel and our RGB-D modified algorithm on the Office-1 scene of the Replica dataset. The results are shown in Fig. 2 and Tab. 1. Despite the PSNR for RGB rendering actually being higher for the RGB-only reconstruction, the average L1 distance per pixel is 70cm, compared to the RGB-D L1 distance of just 9mm. Confirming that although RGB alone might look convincing for use in novel view synthesis, it is insufficient to learn the underlying geometrical information, especially in the areas containing less gradients.

The results of computing the average PSNR and L1 difference for five sequences of Scannet are presented in Tab. 3. Even with significantly less training time than the competing methods, our model achieves a significantly better RGB reconstruction loss (PSNR) and produces the best 3D reconstruction by exhibiting the smallest L1 depth loss, keeping within a range of 5cm. The qualitative difference in RGB estimation can be observed in Fig. 3, where our method shows a visually more correct image compared to the over-smooth baseline comparisons. This builds a strong foundation for achieving better tracking accuracy.

4.3 Tracking Results

Tab. 2 shows the estimated pose tracking accuracy for the entire sequence on the Replica dataset. NICE-SLAM estimates the pose using efficient ray sampling with relatively few rays. This leads to a much faster processing speed than Vox-fusion but at a significant reduction in overall accuracy. When we matched our method to NICE-SLAM's tracking speed, we outperformed it across almost all metrics and sequences. Notably, NICE-SLAM showed instability in the rotational aspect of relative pose accuracy, relying on the neural network on a relatively large voxel size that causes difficulty in accurately estimating minute differences in rotation. NICE-SLAM marginally outperformed our method in the Office-0 sequence in terms of ATE but is beaten on the relative pose metrics. On the other hand, Vox-Fusion showed remarkable tracking performance, taking advantage of its considerably longer tracking time. It only trailed slightly behind our proposed method in most of the sequences. However, like NICE-SLAM, it also tended to have lower accuracy for the orientation of the relative pose error. The significant tracking times reinforce the statement that Vox-fusion is not meant for online use.

Tab. 4 displays the comparison of tracking performance on the five ScanNet sequences. Despite the highly local optimization approach of our method (by only optimizing the voxels the rays pass through), which we initially presumed might lead to holes in the final model since ScanNet has some larger sections of invalid depth measurements, our method successfully fills all these gaps. Moreover, our method outperforms the comparison algorithms on all metrics over all sequences while requiring significantly less computation time. When calibrated to match NICE-SLAM's tracking time, which is approximately a third of Vox-fusion's, our model exhibited superior performance across all metrics for all sequences.

For a quantitative analysis of our model's speed vs tracking accuracy trade-off, please see Appendix B.

5 Conclusion

We have presented the analytical augmentation of the Plenoxel algorithm to allow for RGB-D mapping and tracking based on the radiance field equations. We have also argued for and shown both qualitatively and quantitatively the need for RGB-D sensors to accurately and reliably reconstruct outward-facing scenes when modeling using methods from the current paradigm learning-based radiance field algorithms. Our method achieves superior results with much less time on both the mapping and tracking tasks compared to state-of-the-art radiance field-based SLAM methods, modified to perform offline mapping based on ground truth pose data.

REFERENCES

[1] J. T. Barron, B. Mildenhall, M. Tancik, P. Hedman, R. Martin-Brualla, and P. P. Srinivasan. Mip-nerf: A multiscale representation for anti-aliasing neural radiance fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 5855–5864, 2021.

[2] J. T. Barron, B. Mildenhall, D. Verbin, P. P. Srinivasan, and P. Hedman. Mip-nerf 360: Unbounded anti-aliased neural radiance fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 5470–5479, 2022.

[3] M. Bloesch, J. Czarnowski, R. Clark, S. Leutenegger, and A. J. Davison. Codeslam—learning a compact, optimisable representation for dense visual slam. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2560–2568, 2018.

[4] C.-M. Chung, Y.-C. Tseng, Y.-C. Hsu, X.-Q. Shi, Y.-H. Hua, J.-F. Yeh, W.-C. Chen, Y.-T. Chen, and W. H. Hsu. Orbeez-slam: A real-time monocular visual slam with orb features and nerf-realized mapping. *arXiv preprint arXiv:2209.13274*, 2022.

[5] A. Dai, A. X. Chang, M. Savva, M. Halber, T. Funkhouser, and M. Nießner. Scannet: Richly-annotated 3d reconstructions of indoor scenes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 5828–5839, 2017.

[6] A. Eftekhar, A. Sax, J. Malik, and A. Zamir. Omnidata: A scalable pipeline for making multi-task mid-level vision datasets from 3d scans. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 10786–10796, 2021.

[7] J. Engel, V. Koltun, and D. Cremers. Direct sparse odometry. *IEEE transactions on pattern analysis and machine intelligence*, 40(3):611–625, 2017.

[8] C. Forster, Z. Zhang, M. Gassner, M. Werlberger, and D. Scaramuzza. Svo: Semidirect visual odometry for monocular and multicamera systems. *IEEE Transactions on Robotics*, 33(2):249–265, 2016.

[9] S. Fridovich-Keil, A. Yu, M. Tancik, Q. Chen, B. Recht, and A. Kanazawa. Plenoxels: Radiance fields without neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 5501–5510, 2022.

[10] I. J. Goodfellow, M. Mirza, D. Xiao, A. Courville, and Y. Bengio. An empirical investigation of catastrophic forgetting in gradient-based neural networks. *arXiv preprint arXiv:1312.6211*, 2013.

[11] M. Grupp. evo: Python package for the evaluation of odometry and slam. `https://github.com/MichaelGrupp/evo`, 2017.

[12] S. Izadi, D. Kim, O. Hilliges, D. Molyneaux, R. Newcombe, P. Kohli, J. Shotton, S. Hodges, D. Freeman, A. Davison, et al. Kinectfusion: real-time 3d reconstruction and interaction using a moving depth camera. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*, pp. 559–568, 2011.

[13] J. T. Kajiya and B. P. Von Herzen. Ray tracing volume densities. *ACM SIGGRAPH computer graphics*, 18(3):165–174, 1984.

[14] K. N. Kutulakos and S. M. Seitz. A theory of shape by space carving. *International journal of computer vision*, 38:199–218, 2000.

[15] C.-H. Lin, W.-C. Ma, A. Torralba, and S. Lucey. Barf: Bundle-adjusting neural radiance fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 5741–5751, 2021.

[16] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. *Communications of the ACM*, 65(1):99–106, 2021.

[17] T. Müller, A. Evans, C. Schied, and A. Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Transactions on Graphics (ToG)*, 41(4):1–15, 2022.

[18] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos. Orb-slam: a versatile and accurate monocular slam system. *IEEE transactions on robotics*, 31(5):1147–1163, 2015.

[19] R. A. Newcombe, S. J. Lovegrove, and A. J. Davison. Dtam: Dense tracking and mapping in real-time. In *2011 international conference on computer vision*, pp. 2320–2327. IEEE, 2011.

[20] C. Reiser, S. Peng, Y. Liao, and A. Geiger. Kilonerf: Speeding up neural radiance fields with thousands of tiny mlps. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 14335–14345, 2021.

[21] A. Rosinol, J. J. Leonard, and L. Carlone. Nerf-slam: Real-time dense monocular slam with neural radiance fields. *arXiv preprint arXiv:2210.13641*, 2022.

[22] T. Schops, T. Sattler, and M. Pollefeys. Bad slam: Bundle adjusted direct rgb-d slam. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 134–144, 2019.

[23] J. Straub, T. Whelan, L. Ma, Y. Chen, E. Wijmans, S. Green, J. J. Engel, R. Mur-Artal, C. Ren, S. Verma, et al. The replica dataset: A digital replica of indoor spaces. *arXiv preprint arXiv:1906.05797*, 2019.

[24] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers. A benchmark for the evaluation of rgb-d slam systems. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pp. 573–580. IEEE, 2012.

[25] E. Sucar, S. Liu, J. Ortiz, and A. J. Davison. imap: Implicit mapping and positioning in real-time. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 6229–6238, 2021.

[26] Z. Teed and J. Deng. Droid-slam: Deep visual slam for monocular, stereo, and rgb-d cameras. *Advances in neural information processing systems*, 34:16558–16569, 2021.

[27] Z. Wang, S. Wu, W. Xie, M. Chen, and V. A. Prisacariu. Nerf–: Neural radiance fields without known camera parameters. *arXiv preprint arXiv:2102.07064*, 2021.

[28] X. Yang, H. Li, H. Zhai, Y. Ming, Y. Liu, and G. Zhang. Vox-fusion: Dense tracking and mapping with voxel-based neural implicit representation. In *2022 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pp. 499–507. IEEE, 2022.

[29] L. Yen-Chen, P. Florence, J. T. Barron, A. Rodriguez, P. Isola, and T.-Y. Lin. inerf: Inverting neural radiance fields for pose estimation. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1323–1330. IEEE, 2021.

[30] A. Yu, R. Li, M. Tancik, H. Li, R. Ng, and A. Kanazawa. Plenoctrees for real-time rendering of neural radiance fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 5752–5761, 2021.

[31] W. Zhang, R. Xing, Y. Zeng, Y.-S. Liu, K. Shi, and Z. Han. Fast learning radiance fields by shooting much fewer rays. *arXiv preprint arXiv:2208.06821*, 2022.

[32] Z. Zhu, S. Peng, V. Larsson, Z. Cui, M. R. Oswald, A. Geiger, and M. Pollefeys. Nicer-slam: Neural implicit scene encoding for rgb slam. *arXiv preprint arXiv:2302.03594*, 2023.

[33] Z. Zhu, S. Peng, V. Larsson, W. Xu, H. Bao, Z. Cui, M. R. Oswald, and M. Pollefeys. Nice-slam: Neural implicit scalable encoding for slam. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 12786–12796, 2022.

6 APPENDIX

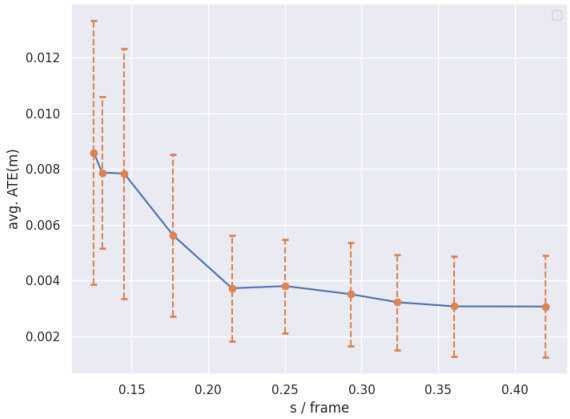

Figure 5: Average ATE error on ScanNet subsets regarding to the tracking speed. Standard deviation of error along 5 sequences are displayed in dotted line.

6.1 Appendix A

We explain the derivative of a tri-linear interpolated grid function with respect to a sample location as is present in equations 8 and 9 from the paper Let $p_i = (x_i, y_i, z_i)$ be the sample location and let the function $f(p)$ represent the tri-linearly interpolated grid function (Either $\vec{c}$ or $\vec{\sigma}$ in our case) where $[v_{000}, ..., v_{111}]$ are the eight closest vertices of p_i. Further let (x_0, y_0, z_0) represent the lattice points below, and (x_1, y_1, z_1) represent the lattice points above the location (x_i, y_i, z_i). The tri-linear interpolation can then be described by the equation:

$$
\begin{aligned}
&f(p_i) = f(x,y,z) = v_i \\
&\approx a_0 + a_1 x_i + a_2 y_i + a_3 z_i + a_4 x y_i + a_5 x_i z_i + a_6 y_i z_i + a_7 x_i y_i z_i
\end{aligned}
$$

where

$$
\begin{bmatrix}
1 & x_0 & y_0 & z_0 & x_0y_0 & x_0z_0 & y_0z_0 & x_0y_0z_0 \\
1 & x_1 & y_0 & z_0 & x_1y_0 & x_1z_0 & y_0z_0 & x_1y_0z_0 \\
1 & x_0 & y_1 & z_0 & x_0y_1 & x_0z_0 & y_1z_0 & x_0y_1z_0 \\
1 & x_1 & y_1 & z_0 & x_1y_1 & x_1z_0 & y_1z_0 & x_1y_1z_0 \\
1 & x_0 & y_0 & z_1 & x_0y_0 & x_0z_1 & y_0z_1 & x_0y_0z_1 \\
1 & x_1 & y_0 & z_1 & x_1y_0 & x_1z_1 & y_1z_1 & x_1y_0z_1 \\
1 & x_0 & y_1 & z_1 & x_0y_1 & x_0z_1 & y_1z_1 & x_0y_1z_1 \\
1 & x_1 & y_1 & z_1 & x_1y_1 & x_1z_1 & y_1z_1 & x_1y_1z_1
\end{bmatrix}
\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \end{bmatrix}
=
\begin{bmatrix} v_{000} \\ v_{001} \\ v_{010} \\ v_{011} \\ v_{100} \\ v_{101} \\ v_{110} \\ v_{111} \end{bmatrix}
\tag{15}
$$

As all voxels are locally independent we can treat the lower lattice points (x_0, y_0, z_0) as $(0,0,0)$ greatly simplifying the equations.

Then if the partial derivatives of these equations are computed with respect to $p_i = (x_i, y_i, z_i)$ we get:

$$
\begin{aligned}
\frac{\partial v_i}{\partial x_i} &= a_1 + a_4 y_i + a_5 z_i + a_7 y_i z_i \\
\frac{\partial v_i}{\partial y_i} &= a_2 + a_4 x_i + a_6 z_i + a_7 x_i z_i \\
\frac{\partial v_i}{\partial z_i} &= a_3 + a_5 x_i + a_6 y_i + a_7 x_i y_i
\end{aligned}
\tag{16}
$$

6.2 Appendix B

Fig. 5 displays the speed-accuracy trade-off curves obtained by testing different settings across the eight sequences from the Replica dataset. It indicates that even if we reduce the allotted tracking time of our method to just 75ms per frame, 50% of the time used by Nice-SLAM, it results in better tracking accuracy.