Contents lists available at ScienceDirect

Computers & Graphics

journal homepage: www.elsevier.com/locate/cag

DirectVoxGO++: Grid-Based Fast Object Reconstruction using Radiance Fields

Daniel Perazzo[a,b,*], João Paulo Lima[b,c], Luiz Velho[a], Veronica Teichrieb[b]

[a] *IMPA - VISGRAF Laboratory, Rio de Janeiro, Brazil*
[b] *CIn/UFPE - Voxar Labs, Recife, Brazil*
[c] *DC/UFRPE - Visual Computing Lab, Recife, Brazil*

ARTICLE INFO

Article history:
Received April 14, 2023

Keywords: Neural Radiance Fields, Object Reconstruction, Volume Rendering, Neural Fields

ABSTRACT

In recent years, the computer vision and computer graphics communities have seen the emergence of new classes of models for representing objects based on Neural Radiance Fields (NeRFs). These techniques use ideas based on traditional methods from computer graphics, for example, radiance fields, and combine them with implicit neural representations and neural image-based rendering (IBR). However, one significant drawback of this class of techniques was that they were too slow, taking in the range of hours in high-end GPUs. Due to these limitations, new techniques have been created for the fast reconstruction of scenes, such as Direct Voxel Grid Optimization (DirectVoxGO) ~~DirectVoxGO~~[Rev2]. Alongside this limitation, initially, NeRFs had limited compositing and modeling capabilities. For example, a simple task such as separating the background from the foreground could not be modeled with NeRFs until the emergence of new techniques such as NeRF++. Our method extends DirectVoxGO to allow the handling of unbounded scenes inspired by some ideas from NeRF++, adapting it to incorporate elements from a neural hashing approach employed by other works. Our technique improved photorealism compared with DirectVoxGO and Plenoxels on a subset of the Light Field Dataset~~LF dataset~~[Rev1] on average in at least 2%, 8%, and 8% for Peak Signal-to-Noise Ratio (PSNR)~~PSNR~~[Rev2], Structural Similarity Index Measure (SSIM) ~~SSIM~~[Rev2], and Learned Perceptual Image Patch Similarity (LPIPS)~~LPIPS~~[Rev2] metrics, respectively, while also being an order of magnitude faster than NeRF++. Also, we demonstrate that, for the evaluated scenes, our technique has comparable training time and memory consumption than previous works. Code is available in `https://github.com/danperazzo/dvgoplusplus`.

1. Introduction

In the computer graphics community, new graphical representations based on neural networks have recently gained adoption. Among those, one the most popular is Neural Radiance Fields (NeRFs) [1]. These techniques expanded on previous works on radiance and the classic theory of light fields [2], and their emergence caused a significant surge in the number of works published about such themes in a relatively short period [3].

The great breakthrought in this class of techniques were their impressive photorealistic rendering and flexibility for a wide

*Corresponding author:
e-mail: `drp@cin.ufpe.br` (Daniel Perazzo)

range of applications, from reconstructing the entire San Diego city [4] to human body modeling [5]. However, the original NeRF technique is computationally expensive, taking hours to run on high-end GPUs [1].

To counter this problem, other authors developed techniques to speed up the optimization process of NeRFs [6]. Some authors started developing techniques inspired by classical computer graphics algorithms and aiming to improve the speed of the rendering process itself [7, 8]. One direction some authors have been exploring is using grid-based NeRFs [9, 10, 11]. While DirectVoxGO [11] employed a two-stage optimization scheme based on a coarse representation to obtain an estimate of the geometry of the object and a fine representation to obtain its reconstruction[Rev4], Plenoxels [9] used spherical harmonics coefficients instead of a Multilayer Perceptron (MLP)~~MLP~~[Rev2]. In contrast, Instant-NeRFs [10] used a multi-resolution neural hashing approach. These methods enable faster rendering times at a significantly higher memory footprint cost [12].

Additionally, the original NeRF only allowed the capture of one scene, not being possible to separate, for example, foreground objects from the background, which would be interesting for 3D reconstruction applications [13]. Due to this limitation ~~To counter both of these issues~~[Rev3], NeRF++ [13] employed a background compositing approach that improved results for[Rev3] 360° settings and enabled[Rev3] the ability to separate foreground from background.

In this work, we present a technique that, after receiving a set of images and their respective intrinsic and extrinsic parameters, obtains a 3D model of the object of interest and background in 360° scenes. In addition, we aim to make our method efficient in terms of memory usage and scene optimization time. Differently from the original DirectVoxGO [11], we use a formulation that enables extracting NeRFs from 360° scenes in a reasonable time. Additionally, we integrated DirectVoxGO with a state-of-the-art multi-resolution hash encoder.

The contributions of this work are:

- Define and develop an improvement of DirectVoxGO for 360° scenes and extraction of a 3D model of an object of interest (Section 4);
- Perform qualitative and quantitative evaluations of our method on widely-used datasets. In our case, we test on the Light Field (LF) [14] dataset and compare it with other works such as the original DirectVoxGO [11] and Plenoxels [9] (Section 5).

Compared with our previous work on this subject [15], of which this paper is an extension, this paper presents a more detailed expansion of the background and foundations of our technique in Section 2, with a discussion about light fields, neural networks for IBR and implicit neural representations. We also added Section 3 for reviewing the literature on NeRFs and fast training for NeRFs. Furthermore, we include a more detailed description of our work in Section 4, with explanations about the traditional NeRF pipeline, the pre-processing procedure, and the coarse and fine training of our technique. Finally, we expanded our exploration of the results in Section 5 by adding time and memory measurements for each scene.

2. Background and Foundations

2.1. Light Fields

The light field is a modeling technique representing incoming lights as a vector function, similar to how physics models electromagnetic fields. In fact, in the mid-19th century, Michael Faraday was of the first researchers to conjecture that light could be modeled as a field [16]. Later, at the beginning of the 20th century, physics works, such as the ones by Gershun [17], and Moon and Spencer [18], would mathematically model light as a field. Finally, in the late 90s, after the development of digital computers and computer graphics, both Levoy and Hanrahan [2] and Gortler et al. [19] suggested using light-field formulations and the 4D light field for image-based rendering (IBR). Investigations on capturing and manipulating the light field would lead to the development of a multi-camera array for the capture of light fields [20], and even a light-field microscope [21]. However, in this area, one watershed moment was the development of light-field cameras that enabled the easy capture of light fields [22]. These cameras use micro-lens arrays, which enable the camera

to take photos from multiple views at once and capture the ~~many~~ [Rev3] light rays that enter the camera aperture. Moreover, with the acquisition of light fields, it is possible to perform image processing (e.g., changing the focus in the photos).

2.2. Neural Networks for Image-Based Rendering

IBR is a technique for modeling and rendering that uses images instead of geometry as primitives [23]. In the last few years, many techniques tried to perform novel view synthesis of scenes given a few views using neural networks and representations such as multi-plane images (MPIs) [24, 25].

Although these techniques create impressive novel views given a few input images, they sometimes generate artifacts originated from the MPI formulation. Due to this, Mildenhall et al. [26] developed a new technique for view synthesis using ideas from light-field rendering. Even though these techniques obtain excellent results and great photorealism, they do not output geometry like 3D reconstruction techniques.

2.3. Neural Implicit Representations

Computer graphics applications often use implicit representations, such as Signed Distance Functions (SDFs)~~SDFs~~ [Rev2] [27]. Neural implicit representations use the fundamental idea that a neural network can work as the function being evaluated.

Neural implicit representations use formulations popular in computer graphics. Novel techniques, such as Occupancy Networks [28] and DeepSDFs [29], used implicit formulations, replacing the traditional functions used in other implicit representations by neural networks. Researchers even studied specific neural networks for these tasks, such as SIREN [30]. Later, new models of implicit representations were constructed with NeRFs [1]. A complete survey is presented by Schirmer et al. [31].

3. Related NeRF-based Techniques

This section reviews the literature ~~many works~~ [Rev3] on NeRFs, including the original technique [1], its ramifications and improvements, and, finally, the techniques that tried to speed up NeRFs. The techniques explained in this section will serve to build our method. As stated previously, we will build up from DirectVoxGO [11], and extend it to 360° scenes by employing an inverted sphere parametrization from NeRF++ [13] and the enconding from InstantNeRFs [10] [Rev3]

3.1. Neural Radiance Fields

Generally, many practitioners' approaches separate physically based rendering (PBR) and IBR in computer graphics. Both of these fields have been around for some time and have been used in various forms. Nowadays, we have seen staggering progress in the IBR domain, from 3D photos based on MPIs [24] to light fields [32]. These developments enabled taking 3D photos from a single shot on ~~many~~ [Rev3] consumer devices [33].

However, these techniques present limitations, such as relatively low photorealism and not much expressivity. They are also not capable of acquiring both geometry and appearance [1]. Due to this, Mildenhall et al. [1] proposed NeRF, an IBR technique that encodes the scene as a MLP and synthesizes novel views using an approach based on volumetric rendering. This approach is similar to previously discussed deep implicit methods. However, the original formulation only allowed for simple shapes.

Due to this, the authors encoded a 5D radiance field as an MLP and performed the consequent optimization. The authors base their technique on classic volumetric rendering, which is trivially differentiable, to perform rendering. Thus, the technique enables high-quality view synthesis results using images acquired from a sparse set of camera views.

However, NeRF is not without its flaws. Despite the impressive quality of the resulting views, it nonetheless has ~~some~~ [Rev3] drawbacks that ~~many~~ [Rev3] other authors decided to improve. The ~~Some of the~~ [Rev3] most relevant are the time it takes to optimize a single scene (which can be of the order of days on Nvidia RTX GPUs), the amount of memory it takes, and the speed of rendering a single scene.

Due to these limitations regarding training/rendering time and memory consumption [Rev3], ~~many~~ [Rev3] different variations and improvements on NeRF have been proposed. Here, we only highlight the techniques that seem more in line with our research, which are techniques with fast training for rendering NeRFs

photo-realistically. Other techniques, such as NeRF++ [13], extended the results for 360° scenes by employing an inverted sphere parametrization. Finally, recently researchers expanded on the quality of these results, such as MipNeRF 360 [34]Rev5.

3.2. Fast Training

One of the biggest problems of NeRFs, as mentioned previously, is their lengthy rendering and training times, which caused many techniques to aim at speeding up rendering. First, some of these adapted traditional computer graphics techniques such as PlenOctrees [7]. Other techniques explored parallelization using much smaller NeRFs instead of only one [8]. Other approaches for speeding up rendering have been proposed, such as querying the ray itself [35] to perform the analytical integration instead of computing it by sampling along the ray [36].

However, other researchers are currently discovering ways to speed up the training process. Initially, some researchers tried using techniques from meta-learning [37] to give a pre-initialized set of weights. They used fairly common meta-learning algorithms such as Reptile [38].

Another approach was using Convolutional Neural Networks (CNNs)~~CNNs~~Rev2 to extract features from the images. This way, the technique could use image features as prior for the MLP. This approach was taken in both PixelNeRF [39] and MVSNeRF [40].

Moreover, instead of using meta-learning or convolutional features to speed-up training, recent approaches aim to shift the inference function to a faster training procedure. For example, both DirectVoxGO [11] and Plenoxels [9] use a voxel-based method instead of a pure MLP that has been traditionally associated with NeRFs. Their approach assured similar results to the original NeRF technique with training times on the scale of minutes instead of hours. In another similar approach, TensoRF [12] applies tensorial decomposition to leverage a speed-up, with also a decrease in the memory footprint compared to previous methods.

Recently, neural-hashing-based techniques [10] have managed to do a speed-up that made them take seconds in settings where the original NeRF took hours. On a negative note, this new technique presents very high memory requirements. Finally, as a footnote, there has also been ~~some~~ Rev3 research in creating a specialized system-on-a-chip (SoC) core for NeRFs, such as the ICARUS architecture [41].

4. DirectVoxGO++

In this section we detail our technique along with presenting a brief overview of the traditional NeRF pipeline.

4.1. Traditional NeRF Volume Rendering Pipeline

The key idea of NeRF [1] is that it models the scene as a radiance and opacity field and renders it using volume rendering Rev3. To do this, the authors use a multi-layer perceptron ~~we define as~~ Rev3 **MLP** that receives as input the position of a point $\mathbf{p} \in \mathbb{R}^3$ and a unit view direction vector $\mathbf{d} \in \mathbb{R}^3$ and outputs both a color $\mathbf{c} = (R, G, B)$ and a density $\sigma \in \mathbb{R}$. The authors then used volume rendering to compute the color of the pixels~~image values~~ Rev3.

Formalizing, if we want to render a view, by volume rendering, for each ray $\mathbf{r}$, corresponding to a pixel in the image, we want to estimate its color $\mathbf{C}(\mathbf{r})$ Rev3. To obtain the rays, we can use the given camera parameters and estimate the origin $\mathbf{o}$ and direction $\mathbf{d}$ of each ray $\mathbf{r}(.)$. With this, any point in the ray can be parametrized by $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$ with $t \in \mathbb{R}$. To limit the number of samples in the ray, the authors of NeRF used bounds based on the points obtained with COLMAP [42]~~COLMAP~~ Rev5. Using the ray parameterization $\mathbf{r}(.)$, we ~~can~~ Rev3 sample a number N of points $\mathbf{p}_i$ from the ray and, using the classic volume rendering equation [43], we can estimate the color $\mathbf{C}(\mathbf{r})$ Rev3 for the ray $\mathbf{r}(.)$ Rev3.

For each sampled point $\mathbf{p}_i$, and using the unit viewing direction vector $\mathbf{d}$, we compute $\mathbf{c}$ and σ by

$$\sigma_i, \mathbf{c}_i = \mathbf{MLP}(\mathbf{PE}(\mathbf{p}_i), \mathbf{PE}(\mathbf{d})), \tag{1}$$

where $\mathbf{PE}(.)$ is the positional encoding operation Rev3, a technique introduced in their paper that aims to stimulate the **MLP** to learn high-frequency features [1].

Following the volume rendering approach, given a step size of the ray $\delta_i \in \mathbb{R}$, we compute an α_i value, which is the probability

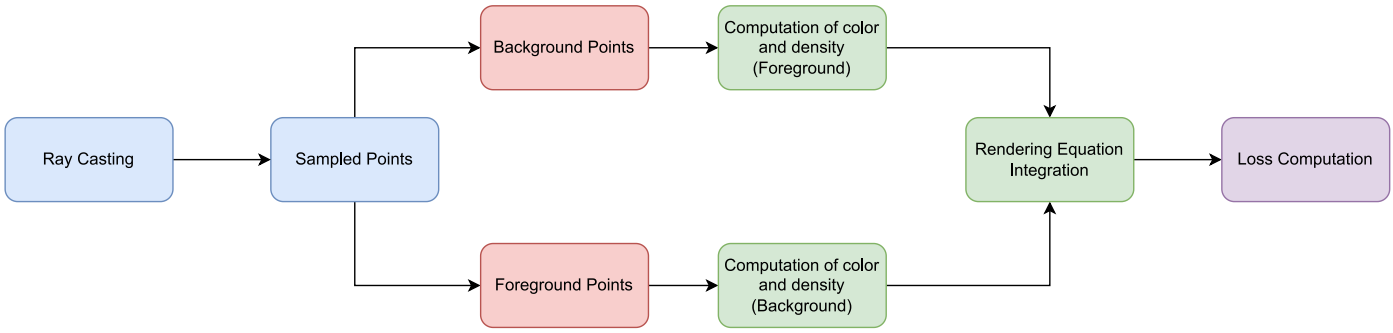


Fig. 1. Pipeline of DirectVoxGO++. In the first step, we cast rays into the scene and sample points in the background and the foreground. Next, we compute color and densities for each setting and combine both in an integration. Finally, since each step is differentiable, we use the predicted RGB values, compare them to the ground truth, and optimize the model accordingly.

that the ray will terminate at this point after traversing the δ_i step[Rev5]. This α_i is computed by:

$$\alpha_i = \mathbf{alpha}(\sigma_i, \delta_i) = 1 - \exp(\sigma_i \delta_i). \tag{2}$$

Next, we compute opacity weights T_i for each $i \in [0, N+1]$:

$$T_i = \prod_{j=1}^{j=i-1} (1 - \alpha_j). \tag{3}$$

Now, with the color value $\mathbf{c}_i$ for each of the N sampled points in the ray, we can then obtain the color of the object of interest in the foreground, [Rev2] $\mathbf{C_{fg}}(\mathbf{r})$ as follows:

$$\mathbf{C_{fg}}(\mathbf{r}) = \sum_{i=1}^{i=N} T_i \alpha_i \mathbf{c}_i. \tag{4}$$

Finally, we can compose it with a pre-defined background color $\mathbf{c_{bg}} = (R, G, B)$, where we multiply it by the opacity corresponding to the background T_{N+1}. Doing so enable us to obtain the final $\mathbf{C}(\mathbf{r})$:

$$\mathbf{C}(\mathbf{r}) = \mathbf{C_{fg}}(\mathbf{r}) + T_{N+1}\mathbf{c_{bg}}. \tag{5}$$

~~Although we wrote this equation in discrete form, it is only a discretized version of an integral, which makes it differentiable.~~[Rev3] Based on this approach, the authors optimized the MLP using a simple photometric loss for each ray $\mathbf{r}$ ~~$\mathbf{r}_i$~~[Rev3], comparing it with its pixel color $\mathbf{C_{pix}}(\mathbf{r})$~~the corresponding observed pixel color corresponding to that ray $\mathbf{C_{pix}}(\mathbf{r})$~~[Rev3] while iterating in the set of all rays $\mathcal{R}$:

$$\mathcal{L}_{photo} = \frac{1}{||\mathcal{R}||} \sum_{\mathbf{r} \in \mathcal{R}} ||\mathbf{C_{pix}}(\mathbf{r}) - \mathbf{C}(\mathbf{r}))||^2. \tag{6}$$

Then, we can optimize the neural network parameters using a gradient-descent-based approach.

4.2. *Overview of DirectVoxGO~~our Method~~*[Rev3]

As mentioned in the introduction, we used DirectVoxGO [11] as a foundation.~~We recomend reading their original paper for more details.~~[Rev3] Differently from Sun et al. [11], we compute background and foreground colors separately and compose both using Equation 5. To compute the background and foreground colors, we performed the classic volume rendering approach in NeRF [44]. In this section we will review the pipeline of DirectVoxGO, which will be similar to the pipeline of our technique, and in the next sections we will indicate the additions and changes we made to the original method.[Rev3]

In the DirectVoxGO pipeline~~our technique~~[Rev3], we use a voxel grid coupled with a two-stage training procedure that aims to find a coarse model for the basic geometry of the scene. This coarse model allows for a tighter bounding box and skips free space during the fine-training stage. We perform a trilinear interpolation to extract the color and density information from a point given a grid.

The coarse training step only uses coarse color and density grids to find an estimate for the object's geometry and, given the values of the voxels, uses trilinear interpolation to obtain the density and color for the points.~~The coarse training step only uses coarse color and density grids to find a coarse geometry of the scene and only a trilinear interpolation to find the color and density values~~[Rev3] This step is much faster than the next fine training step. After this step, the fine training step uses this coarse geometry to train with much higher resolution grids and an MLP on a tighter bounding box, thereby skipping free space.

~~After this step, the fine training step use this coarse geometry to train with much higher resolution grids and an MLP on a tighter bounding box, thereby skipping free space~~[Rev3] We use the photometric loss, background entropy loss, and point loss regularizer ~~introduced in DirectVoxGO [11]~~[Rev3] to perform the training. In Figure 1, we can see a pipeline for the rendering in our method. In the first stage, we sample from points in the background and foreground. These points will go through an encoding stage, where the features extracted will be used to estimate the color and density values for the points. Finally, we use the rendering equation [44] to extract the color value of a pixel.

For the optimization procedure, we compose a final loss $\mathcal{L}_{total}$ using three partial losses: a photometric loss $\mathcal{L}_{photo}$, a per-point regularization $\mathcal{L}_{point}$ and a background entropy regularization $\mathcal{L}_{bg}$.

The photometric loss $\mathcal{L}_{photo}$ is the one used by the original NeRF, as shown in Equation 6. The per-point regularization $\mathcal{L}_{point}$ can be written as:

$$\mathcal{L}_{point} = \frac{1}{|\mathcal{R}|} \sum_{\mathbf{r}\in\mathcal{R}} \sum_{\mathbf{i}=\mathbf{1}}^{\mathbf{i}=\mathbf{N}} \mathbf{T_i}\alpha_\mathbf{i}\|\mathbf{c_i} - \mathbf{C(r)}\|^2. \tag{7}$$

The background entropy regularization $\mathcal{L}_{bg}$, which aims to create a balance between the background and the foreground, is defined as:

$$\mathcal{L}_{bg} = -T_{N+1}\log(T_{N+1}) - (1 - T_{N+1})\log(1 - T_{N+1}). \tag{8}$$

The final loss $\mathcal{L}_{total}$ is defined by using the hyperparameter weights w_{photo}, w_{point} and w_{bg} for each partial loss:

$$\mathcal{L}_{total} = w_{photo} \cdot \mathcal{L}_{photo} + w_{bg} \cdot \mathcal{L}_{bg} + w_{point} \cdot \mathcal{L}_{point}. \tag{9}$$

For the optimization, we use the Adam optimizer [45]~~Adam optimizer~~[Rev2].

4.3. Preprocessing

Before we begin, we must detail our preprocessing steps before DirectVoxGO++ starts. Given a set of N_{images} RGB images I, we first pass this set to the COLMAP [42] camera calibration pipeline for obtaining the intrinsic matrix $\mathbf{K} \in \mathbb{R}^{3\times3}$ and the N_{images} extrinsic matrices. The extrinsic matrices encode the camera's position and orientation in 3D space. We can represent a extrinsic matrix $\mathbf{M} \in \mathbb{R}^{3\times4}$ as $\mathbf{M} = [\mathbf{R}|\mathbf{t}]$, where $\mathbf{R} \in \mathbb{R}^{3\times3}$ is a rotation matrix and $\mathbf{t} \in \mathbb{R}^{3\times1}$ is a translation vector.

Additionally, as in the original NeRF [1], we also extract bounds as to where the scene is localized. No other COLMAP [42] results are used in the remaining pipeline.

However, following the steps of NeRF++ [13], we need to ensure that the mean of the camera centers is at the origin of the camera coordinate system. Additionally, these camera centers are bounded by a unit sphere centered at the origin of the coordinate system. By this, given a set $\mathbf{T}$ of all vectors $\mathbf{t}$[Rev2] which represent the camera centers, we compute the mean center using the $\mathbf{t_{mean}}$~~$\mathbf{c_{mean}}$~~[Rev2] function defined by:

$$\mathbf{t_{mean}} = \frac{1}{|\mathbf{T}|} \sum_{\mathbf{t}\in\mathbf{T}} \mathbf{t}. \tag{10}$$

With this, we can get a new set of camera centers by translating all points in $\mathbf{T}$ by this mean center, ensuring that the origin of the coordinate system $\mathbf{o}$ is the new mean of the centers and getting the new set $\mathbf{T}'$ by

$$\mathbf{T}' = \{\mathbf{t} - \mathbf{t_{mean}} \mid \mathbf{t} \in \mathbf{T}\}. \tag{11}$$

We can now discover the point distance d_{max} which is farthest from the origin by computing:

$$d_{max} = \max_{\mathbf{t}\in\mathbf{T}'} \|\mathbf{t} - \mathbf{o}\|. \tag{12}$$

Now we scale these points by d_{max} to ensure that they are bounded by the unit sphere centered at the origin $\mathbf{o}$ and obtain the final set of translation vectors $\mathbf{T}''$ doing

$$\mathbf{T}'' = \{\frac{\mathbf{t}}{d_{max}} \mid \mathbf{t} \in \mathbf{T}'\}. \tag{13}$$

After performing this processing, we ensure that we can do the background sampling proposed in NeRF++.

4.4. Coarse Training

This stage was adapted from the original DirectVoxGO, and it~~This stage~~[Rev3] aims to find a coarse density grid prior. Given as input initial hyperparameter grid dimensions $(\mathbf{N_x^{(c)}}, \mathbf{N_y^{(c)}}, \mathbf{N_z^{(c)}})$, where (c) stands for coarse,[Rev2] we can then use these grids to perform optimization. For this, we optimize two grids, an

RGB grid $\mathbf{V}^{(\mathbf{c})}_{\mathbf{RGB}} \in \mathbf{V}^{\mathbf{3}\times\mathbf{N}^{(\mathbf{c})}_{\mathbf{x}}\times\mathbf{N}^{(\mathbf{c})}_{\mathbf{y}}\times\mathbf{N}^{(\mathbf{c})}_{\mathbf{z}}}$ and a density grid $\mathbf{V}^{(\mathbf{c})}_{\mathbf{density}} \in \mathbf{V}^{\mathbf{1}\times\mathbf{N}^{(\mathbf{c})}_{\mathbf{x}}\times\mathbf{N}^{(\mathbf{c})}_{\mathbf{y}}\times\mathbf{N}^{(\mathbf{c})}_{\mathbf{z}}}$, where the aim is to find a prior $\mathbf{V}^{(\mathbf{c})}_{\mathbf{density}}$ that represents the coarse geometry of the scene.

To use the grids, we use a trilinear interpolation operation, which we define as **interp**(.,.), where, given as input a point $\mathbf{p}_i \in \mathbb{R}^3$ and a grid with an arbitrary number of channels C and dimensions (H, W, D), the **interp**(.,.) operation has the following mapping:

$$\mathbf{interp} : \mathbb{R}^3 \times \mathbf{V}^{\mathbf{C}\times\mathbf{H}\times\mathbf{W}\times\mathbf{D}} \longrightarrow \mathbb{R}^C. \tag{14}$$

The voxel grids are initialized with zeros and optimized during training. Following a scheme similar to the original NeRF, with known camera parameters, we can compute for each pixel the ray $\mathbf{r}(.)$ that passes through it and, with this, we sample N points $\mathbf{p}_i$ in the ray and then perform volume rendering to find the color of the ray. To perform this, we need to find the RGB color and density $\mathbf{c}_i \in \mathbb{R}^3, \sigma_i \in \mathbb{R}$ for these N points in the ray.

In the coarse training stage, we can find the RGB color $\mathbf{c}_i$ for each point $\mathbf{p}_i$ and each grid $\mathbf{V}^{\mathbf{3}\times\mathbf{N}^{(\mathbf{c})}_{\mathbf{x}}\times\mathbf{N}^{(\mathbf{c})}_{\mathbf{y}}\times\mathbf{N}^{(\mathbf{c})}_{\mathbf{z}}}$ by computing

$$\mathbf{c}_i = \mathbf{interp}(\mathbf{p}_i, \mathbf{V}^{(\mathbf{c})}_{\mathbf{RGB}}). \tag{15}$$

For the density σ_i we use a similar procedure:

$$\sigma_i = \mathbf{softplus}(\mathbf{interp}(\mathbf{p}_i, \mathbf{V}^{(\mathbf{c})}_{\mathbf{density}}) + b), \tag{16}$$

where b is a hyperparameter bias term.

To find the probability of termination at point $\mathbf{p}_i$ after traversing the δ_i step[Rev5], we just compute $\alpha_i = \mathbf{alpha}(\sigma_i)$, where **alpha**(.) is the function defined in Equation 2. The stepsize δ_i, a hyperparameter, is ommited for brevity. Also, the **softplus**(.) function is an activation function defined as: $\mathbf{softplus}(x) = \ln(1 + \exp(x))$ [46].

4.5. Encoding

Inspired by Müller et al. [10], we use a similar encoding technique as was proposed in their work. For the unitary direction vector $\mathbf{d}$, we use a spherical harmonics encoding **SE**(.) instead of the positional encoding **PE**(.) used by DirectVoxGO. The spherical harmonics encoding has a long history in the computer graphics field, and spherical harmonics coefficients are used to model lightning effects, for example [47]. Spherical harmonics can be seen as akin to Fourier transforms on spherical coordinates where, after we choose a degree, we compute the related coefficients as an encoding. In our case, we use a nested spherical harmonics setup with the degree at most 4, with the degree being a hyperparameter that can be tuned.

For the vectors that are not unitary, we use the neural hashing encoder **NHE**(.) proposed by Müller et al. [10]. Similarly to DirectVoxGO [11] and Plenoxels [9], we define a grid of vertices with values that are trilinearly interpolated. However, differently from these methods, we perform a L-levels multiresolution sampling in our encoding. In addition, we geometrically scale the resolution in which we sample the grid. The result in each level is concatenated to form the final output feature. Also, instead of each vertex storing a value, we use a hashing approach. We transform the position of each vertex along each coordinate (x, y, z) into a different number along each dimension. We then use a spatial hashing function introduced by Teschner et al. [48]:

$$\mathbf{hash}(x, y, z) = x \cdot \pi_1 \oplus y \cdot \pi_2 \oplus z \cdot \pi_3 \mod T, \tag{17}$$

where π_1, π_2, π_3 are large primes and T is the size of the hash table. The authors demonstrated the efficiency of their method, and due to this we replaced the traditional positional encoding **PE**(.) by the neural hashing encoding **NHE**(.).

Thus, in the fine-training stage, for every sampled point $\mathbf{p}_i$, considering the grid of features $\mathbf{V}^{(\mathbf{f})}_{\mathbf{features}}$, grid of densities $\mathbf{V}^{(\mathbf{f})}_{\mathbf{densities}}$ and ray directions $\mathbf{d}$, we can sample the colors using the encoders **NHE**(.) and **SE**(.) by performing

$$\mathbf{c}_i = \mathbf{MLP}(\mathbf{interp}(\mathbf{p}_i, \mathbf{V}^{(\mathbf{f})}_{\mathbf{features}}), \mathbf{NHE}(\mathbf{p_i}), \mathbf{SE}(\mathbf{d})). \tag{18}$$

For the density σ_i, we use a similar procedure, with b as a hyperparameter and using the **softplus**(.) activation function:

$$\sigma_i = \mathbf{softplus}(\mathbf{interp}(\mathbf{SE}(\mathbf{p_i}), \mathbf{V}^{(\mathbf{f})}_{\mathbf{density}}) + b), \tag{19}$$

4.6. Background Colors

As mentioned before, both the original NeRF and DirectVoxGO assumed a pre-defined background color $\mathbf{c_{bg}}$. However, this severely limited the original technique and only allowed it to be used for bounded scenes. Due to this, we aim to

extend this formulation to allow the DirectVoxGO technique to be used in unbounded scenes. With this aim, it would be ideal that the value of $\mathbf{c_{bg}}$ is different for each ray $\mathbf{r}$ and dependent on the viewing parameters. That means that we are going to estimate $\mathbf{c_{bg}}(\mathbf{r})$ for each ray $\mathbf{r}(.)$. To achieve this, we use the approach of *multi-sphere images* (used in NeRF++ [13]).

We first consider a ray $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$, where $\mathbf{o}$ is the origin of the ray and $\mathbf{d}$ is its direction. In this approach, we assume that a unit sphere centered at the origin bounds the camera centers and that the cameras are pointing towards the object of interest. First, we adopted a parameterization where, for each point (x, y, z), we have that if it is inside the unit sphere (e.g. $\|(x, y, z)\| \leq 1$) then we represent it in the usual coordinates. Else, if it is outside the sphere we represent the point as $(x', y', z', \frac{1}{r})$, where r is the distance of the point to the origin and $\|(x', y', z')\| = 1$. One form to see this parameterization is as if (x', y', z') gives us the unitary vector associated with the point. In contrast, $\frac{1}{r}$ provides us with the inverse of its distance to the center (or disparity). This representation significantly aid us during sampling. An illustration is given in Figure 2.

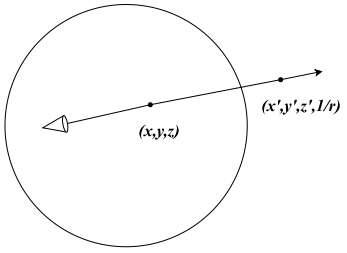


Fig. 2. Image inspired by NeRF++ [13], we can see that inside the sphere the coordinate system is unchanged while outside the sphere we normalize the coordinates (x, y, z) and add the fourth coordinate $\frac{1}{r}$ where r is the distance to the origin and B is the unit sphere

Now, to perform the sampling up until the intersection $\mathbf{a}$ of the ray and the sphere, we use the traditional sampling pipeline discussed in previous sections. We use this to compute the color of the foreground in Equation 4 just as usual. However, to sample the points $\mathbf{p^{bg}}_i$ in the background, we use the fact that since $r \in [1, \infty)$ then $\frac{1}{r} \in (0, 1]$. This allows us to use $\frac{1}{r}$ as a parameter to sample background points based on its values, as used in NeRF++ [13].

To do so, we compute $\mathbf{a}$, the intersection of the ray $\mathbf{r}(.)$ and the unitary sphere, and $\mathbf{b}$, the midpoint of the chord aligning with the ray. Since both of these points are in the ray, then we have $\mathbf{a} = \mathbf{o} + t_a\mathbf{d}$ and $\mathbf{b} = \mathbf{o} + t_b\mathbf{d}$. We also define $\mathbf{c_{sph}}$ as the origin of the coordinate system and the center of the unit sphere.

However, we have that $\|\mathbf{a} - \mathbf{c_{sph}}\| = \|\mathbf{o} + t_a\mathbf{d} - \mathbf{c_{sph}}\| = 1$ and $\mathbf{d}^T(\mathbf{b} - \mathbf{c_{sph}}) = \mathbf{d}^T(\mathbf{o} + t_b\mathbf{d} - \mathbf{c_{sph}}) = 0$. We can solve both of these equations to obtain $\mathbf{a}$ and $\mathbf{b}$. Next, to obtain a new point $\mathbf{p^{bg}}_i$ in the ray $\mathbf{r}(.)$ given $\frac{1}{r}$, we rotate $\mathbf{a}$ around the axis $(\mathbf{b} - \mathbf{c_{sph}}) \times \mathbf{d}$ by an angle $\omega = \arcsin\|\mathbf{b} - \mathbf{c_{sph}}\| - \arcsin\|\mathbf{b} - \mathbf{c_{sph}}\| \cdot \frac{1}{r}$. This parameterization allows us to sample along with the background points.

Finally, as a final step in our parameterization, we convert it to a 3-coordinate system by the transformation $T(x, y, z, \frac{1}{r}) = (\frac{x}{r}, \frac{y}{r}, \frac{z}{r})$. We transform our originally unbounded set of points into a bounded sphere of unitary radius, which allows us to perform linear interpolation and use the previously discussed neural hashing encoding.

Since we now have a manner to sample N^{bg} points $\mathbf{p^{bg}}_i$, we can now compute $\mathbf{c_{bg}}(\mathbf{r})$. We use a procedure similar to the main pipeline during the fine training stage for the foreground, where we optimize a network $\mathbf{MLP^{bg}}$, a neural-hash encoder $\mathbf{NHE^{bg}}$ and a voxel density grid $\mathbf{V^{(bg)}_{density}} \in \mathbf{V^{1 \times N_x^{(bg)} \times N_y^{(bg)} \times N_z^{(bg)}}}$, with the dimensions of the grid being hyperparameters. Unlike the grid used in the main pipeline, this grid is static, although we acknowledge this could be a direction for further research.

Similar to the foreground case, given a point $\mathbf{p^{bg}}_i$, we compute $\mathbf{c}_i$ by:

$$\mathbf{c}_i = \mathbf{MLP^{bg}}(\mathbf{NHE^{bg}}(\mathbf{p_i^{bg}}), \mathbf{SE}(\mathbf{d})). \tag{20}$$

For the density σ_i, we use a similar procedure:

$$\sigma_i = \mathbf{softplus}(\mathbf{interp}(\mathbf{p^{bg}}_i, G \cdot \mathbf{V^{(bg)}_{density}}) + b), \tag{21}$$

where b is a bias hyperparameter and G is a gain hyperparameter. To finally compute $\mathbf{c_{bg}}(\mathbf{r})$, we use the same traditional volume rendering pipeline we have been operating in the previous sections.

5. Results

We implemented our technique using PyTorch [49] ~~Pytorch~~[Rev2] and with DirectVoxGO [11] as a starting point. We used the original code for DirectVoxGO and Plenoxels [9] to make the evaluation fairer between the techniques. We tested the techniques in a ~~Samsung Odyssey~~ [Rev3] laptop with a CPU i7-7700HQ @ 2.80 GHz with 16 Gb RAM and a GPU NVIDIA GTX 1060 with 6 Gb. Due to the low memory of our set-up, we had to modify the original parameters of the Plenoxels and DirectVoxGO technique. Still, we tried to tune the parameters to evaluate the methods fairly.

We tested with four scenes from the LF Dataset [14], with a small set of the available images, using the data shared by the authors of NeRF++ [13]. The four scenes are all 360° rotations around a single small object. To extract the poses, we used the COLMAP technique. The same dataset was used to evaluate DirectVoxGO++, DirectVoxGO and Plenoxels~~the three techniques~~[Rev5].

As for the parameters, in our technique, we adopted a bias density term of $b = 0.163$. Regarding the grid size, we used $303,000$ voxels in the coarse stage and 125^3 voxels in the fine stage. For sampling, we used a step size of $\delta_i = 0.5$, and a number of samples $N = N_{bg} = 220$, both in the background and foreground in the fine stage.

We used the following evaluation metrics:

- **PSNR**: Peak signal-to-noise ratio, a standard metric for signal processing and compression, which measures the amount of noise available in a signal compared with the original signal.~~We can use it for many types of signals.~~[Rev3] A higher PSNR, up to infinity, indicates a less noisy image;
- **SSIM**: Structural Similarity Index [50], which was developed for images and uses perceptual cues, trying to create a metric that attempts to measure the similarity of the structure of the picture. A higher SSIM indicates, up to one, a more similar image;
- **LPIPS**: Learned Perceptual Image Patch Similarity [51], which uses features extracted from deep neural networks to create its similarity metric. In our comparisons, we use a VGG backbone. A small LPIPS, with a minimum of zero, indicates a perceptually similar image in this metric.

We chose these metrics because of their use in similar works [13, 9, 11]. We also reported memory and training time metrics.

We also report the values for the metrics from NeRF++ [13], extracted from its paper. We only display these values for comparisons as a gold standard since NeRF++ has a long training time (around 24 hours per scene, using 3 NVIDIA RTX 2080Ti GPUs [13]), making it infeasible in various~~many~~[Rev3] applications. Due to these reasons, we omit it in the memory and time comparisons.[Rev4] All images have a resolution of 320×180. In the qualitative comparison figures, PNXS stands for Plenoxels, and DVGO stands for DirectVoxGO.

5.1. Quantitative Comparison

Table 1 reports the mean values for each of the metrics in the four LF Dataset scenes. Our method achieved better results than both Plenoxels and the standard DirectVoxGO. As we discuss in more detail in Section 5.2~~the qualitative results section~~[Rev3], we observed that the region inside the object of interest has better quality than the region outside. Our technique also managed to achieve better object segmentation than Plenoxels.

To investigate this hypothesis experimentally, we used a segmentation mask created from the foreground reconstruction obtained by our technique. This approach allowed us to directly compare the techniques regarding the reconstruction of the object of interest, as shown in Table 2 with the average value in the four LF Dataset scenes for each of the metrics.

These tests showed that our technique learned and modeled the object very well in low-memory settings compared to state-of-the-art methods. We obtained better results and a much higher difference between our results and previous techniques.

As for memory usage and execution time for training on each scene, the original DirectVoxGO runs in around 20 minutes, being faster since it does not need to compute the background and foreground as our technique and Plenoxels need to perform. We~~Since both are comparable techniques in~~

Technique	PSNR(↑)	SSIM(↑)	LPIPS(↓)
DirectVoxGO [11]	20.461	0.658	0.366
Plenoxels [9]	21.912	0.746	0.292
DirectVoxGO++ (ours)	**22.436**	**0.804**	**0.266**
NeRF++ [13]	24.820	0.885	0.221

Table 1. Comparison with previous methods on LF Dataset, we highlight the best result in each metric.

Technique	PSNR(↑)	SSIM(↑)	LPIPS(↓)
DirectVoxGO [11]	27.002	0.904	0.102
Plenoxels [9]	28.919	0.932	0.074
DirectVoxGO++ (ours)	**33.953**	**0.980**	**0.018**

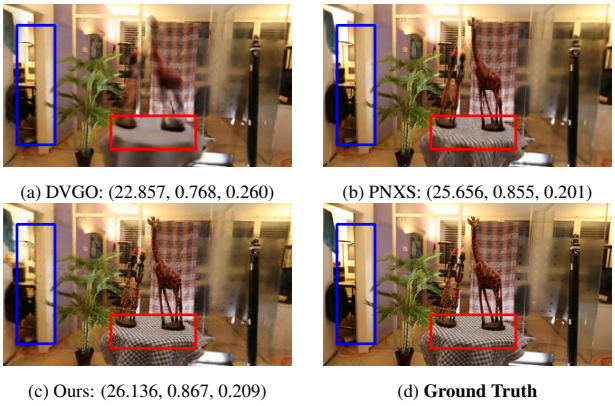
Table 2. Comparison with previous methods on LF Dataset with our masks applied to the objects, we highlight the best result in each metric.

~~terms of idea, we~~[Rev3] provide a time comparison between DirectVoxGO++, DirectVoxGO and Plenoxels in Table 3. As shown in Table 3. for all test scenes, DirectVoxGO++ presented slightly higher training times than Plenoxels~~The table shows that, in most scenes, Plenoxels and DirectVoxGO++ exhibit similar execution times~~[Rev4]. We also compared the memory usage in Table 4. As expected, DirectVoxGO is faster than both. However, it also has the worst results and cannot separate background and foreground. The techniques tested used around 5 GB of GPU memory during execution. We calibrated the parameters to ensure a fair evaluation for each technique.

5.2. *Qualitative Comparison*

In this section, we show a qualitative evaluation of our results. For example, in Figure 3, we can see the results obtained for one test image from the Africa scene. As can be seen, DirectVoxGO did not manage to reconstruct the giraffe very well, with the resulting image presenting blurry features. Compared with Plenoxels, DirectVoxGO++ had a worse background, as can be seen in the blue rectangle in Figure 3,~~worse background~~[Rev3] but a significantly higher quality foreground, with a special note to the checkered stamp on the table. We observe that the Plenoxels technique was better than our technique in the LPIPS metric, most probably due to the background. However, our technique outperformed Plenoxels in the PSNR and SSIM metrics.

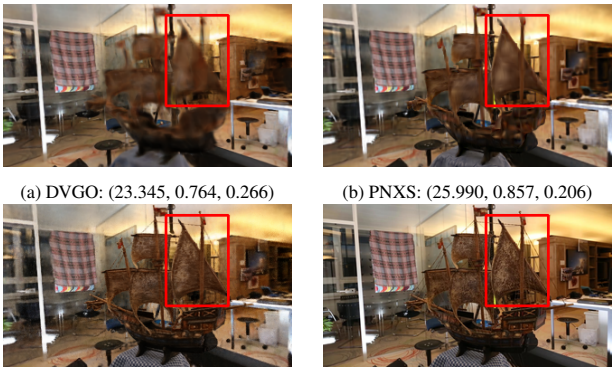
From what we observed, one of the most challenging scenes was the Ship one, as shown in Figure 4. We conjecture that this may be due to this scene's thin structures, such as the mast on the ship. As happened in the other scenes, our technique was the one that was more able to represent high-frequency details in the foreground, as can be observed by the flag on the ship.



(a) DVGO: (22.857, 0.768, 0.260) (b) PNXS: (25.656, 0.855, 0.201)

(c) Ours: (26.136, 0.867, 0.209) (d) **Ground Truth**

Fig. 3. Africa scene, with each result of the techniques and the Ground-truth. They are with their respective metrics, where: (PSNR↑, SSIM↑, LPIPS↓). We used 56 images during training and 8 images during testing. For comparison, the gold standard, NeRF++, has the following values in this scene: (27.410, 0.923, 0.163).



(a) DVGO: (23.345, 0.764, 0.266) (b) PNXS: (25.990, 0.857, 0.206)

(c) Ours: (25.576, 0.872, 0.200) (d) **Ground Truth**

Fig. 4. Ship scene, with each result of the techniques along with the Ground-truth. They are with their respective metrics, where: (PSNR↑, SSIM↑, LPIPS↓). We used 95 images during training and 14 images during testing. For comparison, the gold standard, NeRF++, has the following values in this scene: (25.350, 0.867, 0.241).

The worst result we obtained from the four scenes analyzed was the Torch scene, illustrated in Figure 5. This scene is especially challenging for the evaluated techniques due to moving persons in the middle of the photos. This factor is not accounted for in the original NeRF, albeit solved in other works [52]. The fact that it is our worst result may be due to this blurrier background. However, as in previous settings, our work managed to infer higher-frequency details in the object of interest.

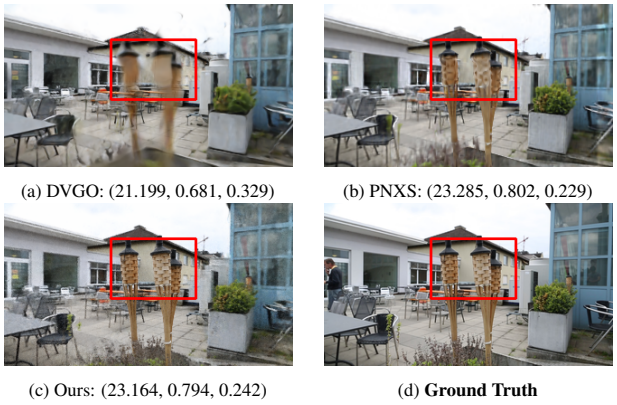
However, compared with the other techniques, our best result is with the basket scene, illustrated in Figure 6. The other

Scene	DirectVoxGO++ (ours)	Plenoxels [9]	DirectVoxGO [11]
Africa	31 Min.	29 Min.	21 Min.
Ship	31 Min.	29 Min.	21 Min.
Torch	31 Min.	28 Min.	22 Min.
Basket	32 Min.	28 Min.	20 Min.

Table 3. Training time~~Time~~[Rev4] comparison between Plenoxels and DirectVoxGO++ in each of the different scenes.

Scene	DirectVoxGO++ (ours)	Plenoxels [9]	DirectVoxGO [11]
Africa	5954 MiB.	5518 MiB.	5658 MiB.
Ship	5142 MiB.	5520 MiB.	4400 MiB.
Torch	5588 MiB.	5520 MiB.	5710 MiB.
Basket	5704 MiB.	5512 MiB.	3872 MiB.

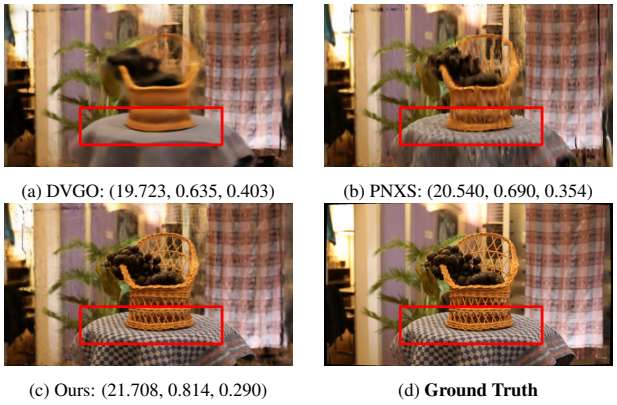
Table 4. Memory comparison between Plenoxels and DirectVoxGO++ in each of the different scenes.



(a) DVGO: (21.199, 0.681, 0.329) (b) PNXS: (23.285, 0.802, 0.229)

(c) Ours: (23.164, 0.794, 0.242) (d) **Ground Truth**

Fig. 5. Torch scene, with each result of the techniques along with the Ground-truth. They are with their respective metrics, where: (PSNR↑, SSIM↑, LPIPS↓). We used 53 images during training and 8 images during testing. For comparison, the gold standard, NeRF++, has the following values in this scene: (24.680, 0.867, 0.226).



(a) DVGO: (19.723, 0.635, 0.403) (b) PNXS: (20.540, 0.690, 0.354)

(c) Ours: (21.708, 0.814, 0.290) (d) **Ground Truth**

Fig. 6. Basket scene, with each result of the techniques along with the Ground-truth. They are with their respective metrics, where: (PSNR↑, SSIM↑, LPIPS↓). We used 74 images during training and 11 images during testing. For comparison, the gold standard, NeRF++, has the following values in this scene: (21.840, 0.884, 0.254).

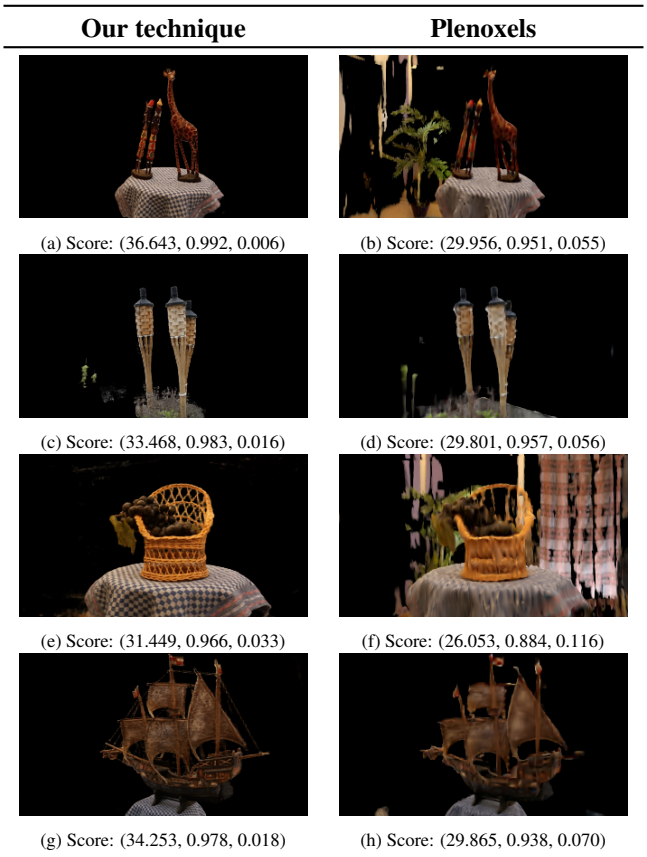
methods did not manage to capture the fine detail of the holes in the basket. This scene is a failure case for the other techniques, with DirectVoxGO presenting a blurry foreground object and Plenoxels resulting in a chopped basket.

As shown in Figure 7, the Plenoxels technique did not manage to focus its segmentation on the object. In fact, in many scenes, it seems that the Plenoxels foreground grid contains some of the content from the background. This fact may account for Plenoxels' lack of details in the foreground since a single grid or MLP does not have enough capacity to model the foreground and the background, causing the foreground areas to suffer compared with our work.

A similar effect happened in the original DirectVoxGO, and an akin issue was shown and addressed in NeRF++ [13]. Since we successfully segment the foreground from the background, our foreground model can better model it with more high-frequency details.

5.3. Ablation Study

This section evaluates the impact of both of our proposals individually. In Table 5, we can observe that the modifications proposed in this work, individually, managed to improve upon the original DirectVoxGO. For a qualitative evaluation, we report the results of one of the scenes in Figure 8. Specifically, we observed qualitatively that the background coloring managed to make the object stay sharp. Meanwhile, the neural hash encoder aided in letting our technique infer more detail from the model. Combining both techniques, we obtained the best results.



(a) Score: (36.643, 0.992, 0.006) (b) Score: (29.956, 0.951, 0.055)
(c) Score: (33.468, 0.983, 0.016) (d) Score: (29.801, 0.957, 0.056)
(e) Score: (31.449, 0.966, 0.033) (f) Score: (26.053, 0.884, 0.116)
(g) Score: (34.253, 0.978, 0.018) (h) Score: (29.865, 0.938, 0.070)

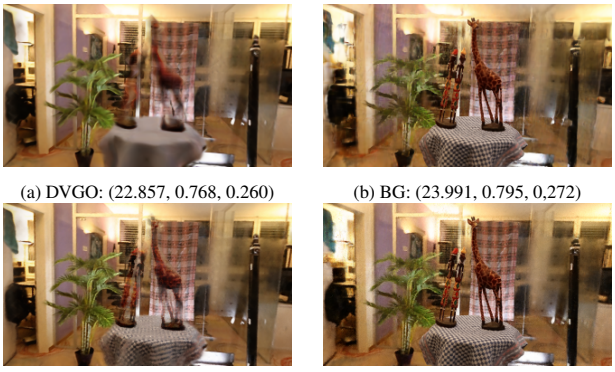
Fig. 7. The foreground captured by our DirectVoxGO++ technique compared with the foreground captured by Plenoxels. Bellow, we report the score obtained by applying the masks obtained by our technique, as reported in Table 2. We show the scores, where: (PSNR↑, SSIM↑, LPIPS↓).

Technique	PSNR(↑)	SSIM(↑)	LPIPS(↓)
DirectVoxGO	20.461	0.658	0.366
DirectVoxGO+(BG)	21.118	0.727	0.319
DirectVoxGO+(NHE)	21.015	0.722	0.312
DirectVoxGO++ (ours)	**22.436**	**0.804**	**0.266**

Table 5. Ablation study on LF Dataset.

6. Conclusion

Our results showed that by combining DirectVoxGO [11] with background compositing [13] and neural hash encoding [10] we could achieve better results and a more accurate object reconstruction. Differently from the original DirectVoxGO, we enabled object reconstruction in unbounded scenes without using any foreground segmentation.[Rev2] We also demonstrated that it exhibited similar execution time and memory consumption during training compared to the other evaluated techniques, Plenoxels and DirectVoxGO. Although our technique achieved good results in the evaluated dataset, we also mapped ~~some~~[Rev3]



(a) DVGO: (22.857, 0.768, 0.260) (b) BG: (23.991, 0.795, 0,272)
(c) NHE: (23.499, 0.813, 0.210) (d) **Ours**: (26.136, 0.867, 0.209)

Fig. 8. Africa scene, with each result of the ablation study, where DVGO stands for DirectVoxGO, BG corresponds to only adding the background coloring, and NHE corresponds to only adding the neural hash encoder. They are with their respective metrics, where (PSNR↑, SSIM↑, LPIPS↓). We used 56 images during training and 8 images during testing.

limitations, such as the artifacts in the background for some scenes and the lack of support for dynamic scenes, that could be expanded in future works[Rev3].

Also, as we have shown with our problems with the Torch scene, a possible direction would be to integrate deformable NeRF [52] ideas into our pipeline and extend it to videos. One field which is getting attention from the community is also editing of NeRFs [53] and integrating it into other graphics systems [54][Rev2]

Acknowledgment

The authors are supported by grants from CNPq (process 422728/2021-7) and FAPERJ.

References

[1] Mildenhall, B, Srinivasan, PP, Tancik, M, Barron, JT, Ramamoorthi, R, Ng, R. Nerf: Representing scenes as neural radiance fields for view synthesis. In: European conference on computer vision. 2020, p. 405–421.
[2] Levoy, M, Hanrahan, P. Light field rendering. In: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques. 1996, p. 31–42.
[3] Dellaert, F, Yen-Chen, L. Neural volume rendering: Nerf and beyond. arXiv preprint arXiv:210105204 2020;.
[4] Tancik, M, Casser, V, Yan, X, Pradhan, S, Mildenhall, B, Srinivasan, PP, et al. Block-nerf: Scalable large scene neural view synthesis. Conference on Computer Vision and Pattern Recognition (CVPR) 2022;.
[5] Peng, S, Dong, J, Wang, Q, Zhang, S, Shuai, Q, Bao, H, et al. Animatable neural radiance fields for human body modeling. International Conference on Computer Vision (ICCV) 2021;.
[6] Tewari, A, Thies, J, Mildenhall, B, Srinivasan, P, Tretschk, E, Yifan, W, et al. Advances in neural rendering. In: Computer Graphics Forum; vol. 41. Wiley Online Library; 2022, p. 703–735.
[7] Yu, A, Li, R, Tancik, M, Li, H, Ng, R, Kanazawa, A. Plenoctrees for real-time rendering of neural radiance fields. International Conference on Computer Vision (ICCV) 2021;.

[8] Reiser, C, Peng, S, Liao, Y, Geiger, A. Kilonerf: Speeding up neural radiance fields with thousands of tiny mlps. International Conference on Computer Vision (ICCV) 2021;.

[9] Yu, A, Fridovich-Keil, S, Tancik, M, Chen, Q, Recht, B, Kanazawa, A. Plenoxels: Radiance fields without neural networks. Conference on Computer Vision and Pattern Recognition (CVPR) 2022;.

[10] Müller, T, Evans, A, Schied, C, Keller, A. Instant neural graphics primitives with a multiresolution hash encoding. ACM Trans Graph 2022;41(4):102:1–102:15.

[11] Sun, C, Sun, M, Chen, HT. Direct voxel grid optimization: Super-fast convergence for radiance fields reconstruction 2022;.

[12] Chen, A, Xu, Z, Geiger, A, Yu, J, Su, H. Tensorf: Tensorial radiance fields. arXiv preprint arXiv:220309517 2022;.

[13] Zhang, K, Riegler, G, Snavely, N, Koltun, V. Nerf++: Analyzing and improving neural radiance fields. arXiv preprint arXiv:201007492 2020;.

[14] Yücer, K, Sorkine-Hornung, A, Wang, O, Sorkine-Hornung, O. Efficient 3d object segmentation from densely sampled light fields with applications to 3d reconstruction. ACM Transactions on Graphics (TOG) 2016;35(3):1–15.

[15] Perazzo, D, Lima, JP, Velho, L, Teichrieb, V. Directvoxgo++: Fast neural radiance fields for object reconstruction. In: 2022 35th SIBGRAPI Conference on Graphics, Patterns and Images (SIBGRAPI); vol. 1. 2022, p. 156–161. doi:10.1109/SIBGRAPI55357.2022.9991779.

[16] Faraday, M. Liv. thoughts on ray-vibrations. The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science 1846;28(188):345–350.

[17] Gershun, A. The light field. Journal of Mathematics and Physics 1939;18(1-4):51–151.

[18] Moon, P, Spencer, DE. The photic field. Cambridge 1981;.

[19] Gortler, SJ, Grzeszczuk, R, Szeliski, R, Cohen, MF. The lumigraph. In: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques. 1996, p. 43–54.

[20] Vaish, V, Wilburn, B, Joshi, N, Levoy, M. Using plane+ parallax for calibrating dense camera arrays. In: Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.; vol. 1. IEEE; 2004, p. I–I.

[21] Levoy, M, Ng, R, Adams, A, Footer, M, Horowitz, M. Light field microscopy. In: ACM SIGGRAPH 2006 Papers. 2006, p. 924–934.

[22] Ng, R, Levoy, M, Brédif, M, Duval, G, Horowitz, M, Hanrahan, P. Light field photography with a hand-held plenoptic camera. Ph.D. thesis; Stanford University; 2005.

[23] Shum, HY, Chan, SC, Kang, SB. Image-based rendering. Springer Science & Business Media; 2008.

[24] Zhou, T, Tucker, R, Flynn, J, Fyffe, G, Snavely, N. Stereo magnification: Learning view synthesis using multiplane images. arXiv preprint arXiv:180509817 2018;.

[25] Srinivasan, PP, Tucker, R, Barron, JT, Ramamoorthi, R, Ng, R, Snavely, N. Pushing the boundaries of view extrapolation with multiplane images. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2019, p. 175–184.

[26] Mildenhall, B, Srinivasan, PP, Ortiz-Cayon, R, Kalantari, NK, Ramamoorthi, R, Ng, R, et al. Local light field fusion: Practical view synthesis with prescriptive sampling guidelines. ACM Transactions on Graphics (TOG) 2019;.

[27] Osher, S, Fedkiw, R. Signed distance functions. In: Level set methods and dynamic implicit surfaces. Springer; 2003, p. 17–22.

[28] Mescheder, L, Oechsle, M, Niemeyer, M, Nowozin, S, Geiger, A. Occupancy networks: Learning 3d reconstruction in function space. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2019, p. 4460–4470.

[29] Park, JJ, Florence, P, Straub, J, Newcombe, R, Lovegrove, S. Deepsdf: Learning continuous signed distance functions for shape representation. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2019, p. 165–174.

[30] Sitzmann, V, Martel, JN, Bergman, AW, Lindell, DB, Wetzstein, G. Implicit neural representations with periodic activation functions. In: Proc. NeurIPS. 2020,.

[31] Schirmer, L, Schardong, G, da Silva, V, Lopes, H, Novello, T, Yukimura, D, et al. Neural networks for implicit representations of 3d scenes. In: 2021 34th SIBGRAPI Conference on Graphics, Patterns and Images (SIBGRAPI). IEEE; 2021, p. 17–24.

[32] Kalantari, NK, Wang, TC, Ramamoorthi, R. Learning-based view synthesis for light field cameras. ACM Transactions on Graphics (TOG) 2016;35(6):1–10.

[33] Kopf, J, Matzen, K, Alsisan, S, Quigley, O, Ge, F, Chong, Y, et al. One shot 3d photography. ACM Transactions on Graphics (TOG) 2020;39(4):76–1.

[34] Barron, JT, Mildenhall, B, Verbin, D, Srinivasan, PP, Hedman, P. Mip-nerf 360: Unbounded anti-aliased neural radiance fields. Mip-NeRF 360: Unbounded Anti-Aliased Neural Radiance Fields 2022;.

[35] Sitzmann, V, Rezchikov, S, Freeman, WT, Tenenbaum, JB, Durand, F. Light field networks: Neural scene representations with single-evaluation rendering. In: Proc. NeurIPS. 2021,.

[36] Lindell, DB, Martel, JN, Wetzstein, G. Autoint: Automatic integration for fast neural volume rendering. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2021, p. 14556–14565.

[37] Tancik, M, Mildenhall, B, Wang, T, Schmidt, D, Srinivasan, PP, Barron, JT, et al. Learned initializations for optimizing coordinate-based neural representations. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2021, p. 2846–2855.

[38] Nichol, A, Achiam, J, Schulman, J. On first-order meta-learning algorithms. arXiv preprint arXiv:180302999 2018;.

[39] Yu, A, Ye, V, Tancik, M, Kanazawa, A. pixelnerf: Neural radiance fields from one or few images. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2021, p. 4578–4587.

[40] Chen, A, Xu, Z, Zhao, F, Zhang, X, Xiang, F, Yu, J, et al. Mvsnerf: Fast generalizable radiance field reconstruction from multi-view stereo. International Conference on Computer Vision (ICCV) 2021;.

[41] Rao, C, Yu, H, Wan, H, Zhou, J, Zheng, Y, Ma, Y, et al. Icarus: A lightweight neural plenoptic rendering architecture. arXiv preprint arXiv:220301414 2022;.

[42] Schönberger, JL, Frahm, JM. Structure-from-motion revisited. In: Conference on Computer Vision and Pattern Recognition (CVPR). 2016,.

[43] Drebin, RA, Carpenter, L, Hanrahan, P. Volume rendering. ACM Siggraph Computer Graphics 1988;22(4):65–74.

[44] Kajiya, JT, Von Herzen, BP. Ray tracing volume densities. ACM SIGGRAPH computer graphics 1984;18(3):165–174.

[45] Kingma, DP, Ba, J. Adam: A method for stochastic optimization. arXiv preprint arXiv:14126980 2014;.

[46] Dugas, C, Bengio, Y, Bélisle, F, Nadeau, C, Garcia, R. Incorporating second-order functional knowledge for better option pricing. Advances in neural information processing systems 2000;13.

[47] Green, R. Spherical harmonic lighting: The gritty details. In: Archives of the game developers conference. 2003, p. 4.

[48] Teschner, M, Heidelberger, B, Müller, M, Pomerantes, D, Gross, MH. Optimized spatial hashing for collision detection of deformable objects. In: Vmv; vol. 3. 2003, p. 47–54.

[49] Paszke, A, Gross, S, Massa, F, Lerer, A, Bradbury, J, Chanan, G, et al. Pytorch: An imperative style, high-performance deep learning library. Advances in neural information processing systems 2019;32.

[50] Wang, Z, Bovik, AC, Sheikh, HR, Simoncelli, EP. Image quality assessment: from error visibility to structural similarity. IEEE transactions on image processing 2004;13(4):600–612.

[51] Zhang, R, Isola, P, Efros, AA, Shechtman, E, Wang, O. The unreasonable effectiveness of deep features as a perceptual metric. In: Proceedings of the IEEE conference on computer vision and pattern recognition. 2018, p. 586–595.

[52] Park, K, Sinha, U, Barron, JT, Bouaziz, S, Goldman, DB, Seitz, SM, et al. Deformable neural radiance fields. International Conference on Computer Vision (ICCV) 2021;.

[53] Yuan, YJ, Sun, YT, Lai, YK, Ma, Y, Jia, R, Gao, L. Nerf-editing: geometry editing of neural radiance fields. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2022, p. 18353–18364.

[54] Tancik, M, Weber, E, Ng, E, Li, R, Yi, B, Kerr, J, et al. Nerfstudio: A modular framework for neural radiance field development. arXiv preprint arXiv:230204264 2023;.