DirectVoxGO++: Fast Neural Radiance Fields for Object Reconstruction

Daniel Perazzo  
Voxar Labs, CIn  
UFPE  
Recife, Pernambuco  
drp@cin.ufpe.br

João Paulo Lima  
Visual Computing Lab, DC, UFRPE  
Recife, Pernambuco  
joao.mlima@ufrpe.br, jpsml@cin.ufpe.br

Luiz Velho  
Visgraf, IMPA  
IMPA  
Rio de Janeiro, Rio de Janeiro  
lvelho@impa.br

Veronica Teichrieb  
Voxar Labs, CIn  
UFPE  
Recife, Pernambuco  
vt@cin.ufpe.br

Abstract—In recent years, new reconstruction techniques based on Neural Radiance Fields (NeRFs) have created new forms to model objects instead of the traditional mesh and point cloud-based representations, allowing for more photorealistic rendering. However, these techniques were too slow to be practical in real-world settings, taking in the range of hours in high-end GPUs. Due to these limitations, new techniques have been created for fast reconstruction of scenes, such as DirectVoxGO. Alongside this limitation, one issue with NeRFs is that they were initially unable to separate the foreground from the background and had problems with 360° scenes 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 LF dataset on average in at least 2%, 8%, and 8% for PSNR, SSIM, and LPIPS metrics, respectively, while also being an order of magnitude faster than NeRF++. Code will be available in https://github.com/danperazzo/dvogoplusplus.

I. INTRODUCTION

In recent years, Neural Radiance Field (NeRF) techniques experienced a significant surge in the number of works published in a relatively short period [3]. These methods, based on the classic theory of light fields [6], allow the capture and rendering of scenes in an impressive photorealistic manner and a significant number of applications, ranging from reconstructing the entire San Diego scene [14] to human body modeling [11]. However, the original NeRF technique is computationally expensive, taking hours to run on high-end GPUs [7] and has a large memory footprint when learning a single scene.

To counter this problem, other authors developed techniques to speed up the optimization process of NeRFs [18]. Some authors started developing techniques inspired by classical computer graphics algorithms and aiming to improve the speed of the rendering process itself [21][12]. However, soon authors attempted new methods aiming at speeding up the training process itself, from using visual features as priors [11][20] to using meta-learning techniques to achieve the best weight initialization [16].

Recently, one direction some authors have been exploring is using grid-based NeRFs [22][8][13]. While DirectVoxGO [13] employed a two-stage optimization scheme, Plenoxels [22] used spherical harmonics coefficients instead of an MLP. In contrast, Instant-NeRFs [9] used a multi-resolution neural hashing approach. These methods enable faster rendering times at a significantly higher memory footprint cost [2].

Additionally, the original NeRF only allowed the capture and representation of an entire scene, not being possible to separate, for example, foreground objects from the background, which would be interesting for 3D reconstruction applications [24]. Although some techniques consider compositing, they generally need some prior information, such as background masks [9]. To counter both of these issues, NeRF++ [24] employed a background compositing approach which improved 360° settings and the ability to separate foreground from background. A similar approach was adopted in Plenoxels [22], along with the previously mentioned grid-based system.

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 [13], 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 III).
- Perform qualitative and quantitative evaluations of our method on widely-used datasets. In our case, we test on the Light Field (LF) [23] dataset and compare it with other works such as the original DirectVoxGO [13] and Plenoxels [22] (Section III).

II. DIRECTVOXGO++

A. Overview

As mentioned in the introduction, we used DirectVoxGO as a foundation. For more details, we recommend reading the original paper by Sun et al. [13]. Differently from Sun et al. [13], we compute background and foreground colors separately and compose both using:
\[
C(r) = C_{\text{pre}}(r) + T_{N+1}c_{\text{bg}}(r).
\] (1)

Where \( C(r) \) is the color of the ray, \( T_{N+1} \) is the associated transmittance of the last \( N \)-sampled point, \( C_{\text{pre}}(r) \) is the color of the foreground and \( c_{\text{bg}}(r) \) is the color of the background. To compute the background and foreground colors, we performed the classic volume rendering approach in NeRF [5].

In our technique, 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 a coarse geometry of the scene and only a trilinear interpolation to find the color and density values. After this step, the fine training step will use this coarse geometry to train with much higher resolution grids and an MLP on a tighter bounding box, thereby skipping free space. We use the photometric loss, background entropy loss, and point loss regularizer introduced in DirectVoxGO [13] 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 background and foreground points. These points will go through an encoding stage, and we will extract color and density values. Finally, we will use the rendering equation [5] to extract the color value of a pixel.

### B. Encoding

Inspired by Muller et al. [8], we use a similar encoding technique as was proposed in their work. For the unit direction vector \( d \), we use a spherical harmonics encoding \( \text{SE}(.) \) instead of the positional encoding \( \text{PE}(.) \) [15] 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 [4]. After choosing a degree, we compute the related coefficients as an encoding [4]. In our case, we use a nested spherical harmonics setup with the degree at most 4, a hyperparameter that can be tuned.

For the vectors that are not unitary, we use the neural hashing encoder \( \text{NHE}(.) \) proposed by Muller et al. [8]. Similarly to DirectVoxGO [13] and Plenoxels [22], we define a grid of vertices with values that will be trilinearly interpolated. However, differently from these methods, we perform a \( L \)-levels multi-resolution sampling in our encoding. In addition, we geometrically scale the resolution in which we will be sampling 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. [17]. The authors demonstrated the efficiency of their method, and due to this, we replaced the traditional positional encoding \( \text{PE}(.) \) with the neural hashing encoding \( \text{NHE}(.) \).

Thus, in the fine-training stage, for every sampled point \( p_i \), considering the grid of features \( V_{\text{features}}^{(f)} \), grid of densities \( V_{\text{densities}}^{(f)} \) and ray directions \( d \), we can sample the colors using the encoders \( \text{NHE}(.) \) and \( \text{SE}(.) \) by performing

\[
c_i = \text{MLP}(\text{interp}(p_i, V_{\text{features}}^{(f)}), \text{NHE}(p_i), \text{SE}(d)).
\] (2)

For the density \( \sigma_i \), we use a similar procedure, with \( b \) as a hyperparameter and using the \text{softplus}(.) activation function:

\[
\sigma_i = \text{softplus}(\text{interp}(\text{SE}(p_i), V_{\text{density}}^{(f)} + b)),
\] (3)

### C. Background Compositing

Both the original NeRF and DirectVoxGO assumed a pre-defined background color \( c_{\text{bg}} \). However, this severely limited the original technique and only allowed it to be used for bounded scenes. We aim to extend this formulation to allow the DirectVoxGO technique to be used in unbounded scenes. Due to this, it would be ideal that the value of \( c_{\text{bg}} \) is different for each ray \( r \) and dependent on the viewing parameters. That means that we are going to estimate \( c_{\text{bg}}(r) \) for each ray \( r(\cdot) \). To achieve this, we use the approach of multi-sphere images (inspired by NeRF++ [24]).

We first consider a ray \( r(t) = o + td \), where \( o \) is the origin of the ray and \( d \) is its direction. In this approach, we assume that a unit sphere centered at the origin bounds the camera centers and that the cameras point 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 will represent it in the

![Diagram](image-url)
usual coordinates. Else, if it is outside the sphere we will 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 if \((x', y', z')\) gives us the unit 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 will significantly aid us during sampling.

Now, we use the traditional sampling pipeline to compute the foreground until the intersection \(a\) of the ray and the sphere. However, to sample the points \(p_i^{bg}\) 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++ [24].

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

However, we have that \(||a - c_{sph}|| = ||o + t_o d - c_{sph}|| = 1\) and \(d^T(b - c_{sph}) = d^T(o + t_o d - c_{sph}) = 0\). Next, to obtain a new point \(p_i^{bg}\) in the ray \(r(.)\) given \(\frac{1}{r}\), we rotate \(a\) around the axis \((b - c_{sph}) \times d\) by an angle \(\omega = \arcsin(||b - c_{sph}|| - ||a - c_{sph}||) / \frac{1}{r}\). This parameterization allows us to sample along with the background points.

Finally, as a final step in our parameterization, and differently from NeRF++, 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 unit radius, which allows us to perform linear interpolation and use the previously discussed neural hashing encoding. We provide an illustration in Figure 2.

For the density \(\sigma_i\), we use a similar procedure:

\[
\sigma_i = \text{softplus}(\text{interp}(p_i^{bg}, G \cdot V_{\text{density}}^{bg}) + b),
\]

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

### III. RESULTS

We compare our technique with DirectVoxGO [13] and Plenoxels [22]. We chose to use Plenoxels as a comparison because, aside from our technique, it was the only published NeRF-based technique that enabled the 3D reconstruction of the desired object in a reasonable timeframe and managed to separate the foreground from the background. We also perform an ablation study to evaluate the impact of each of our novel components individually.

#### A. Evaluation Setup

To better evaluate the setup of our evaluation, here we provide a more detailed explanation. We implemented our technique using PyTorch and with DirectVoxGO [13] as a basis. We used the original code for DirectVoxGO and Plenoxels to make the evaluation fairer between the techniques. We tested the techniques in a Samsung Odyssey 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 setup, we had to adapt the parameters of the Plenoxels and DirectVoxGO techniques. We tested with four scenes from the LF Dataset [23], with a small set of the available images, using the data shared by the authors of NeRF++ [24]. The four scenes are all 360° rotations around a single small object.

As for the metrics, we followed the literature [24, 22, 13] and used the following ones: PSNR, SSIM [19] and LPIPS-VGG [25]. We also report the values for the metrics from NeRF++ [24], extracted from its paper. We only display these values for comparisons as a gold standard since NeRF++ has a long training time (around 9-12 hours per scene), making it infeasible in many applications. All images have a resolution of 320 x 180. In the qualitative comparison figures, PNXS stands for Plenoxels, and DVG stands for DirectVoxGO.

Since the results are deterministic, we only needed to run our experiments once.

#### B. Quantitative Comparison

Table I 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 will discuss in more detail in the qualitative results section, 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 I 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.

<table>
<thead>
<tr>
<th>Technique</th>
<th>PSNR↑</th>
<th>SSIM↑</th>
<th>LPIPS↓</th>
</tr>
</thead>
<tbody>
<tr>
<td>DirectVoxGO</td>
<td>20.461</td>
<td>0.658</td>
<td>0.366</td>
</tr>
<tr>
<td>Plenoxels</td>
<td>21.912</td>
<td>0.746</td>
<td>0.292</td>
</tr>
<tr>
<td>DirectVoxGO++ (ours)</td>
<td>22.436</td>
<td>0.804</td>
<td>0.266</td>
</tr>
<tr>
<td>NeRF++</td>
<td>24.820</td>
<td>0.885</td>
<td>0.221</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Technique</th>
<th>PSNR↑</th>
<th>SSIM↑</th>
<th>LPIPS↓</th>
</tr>
</thead>
<tbody>
<tr>
<td>DirectVoxGO</td>
<td>27.302</td>
<td>0.904</td>
<td>0.102</td>
</tr>
<tr>
<td>Plenoxels</td>
<td>28.919</td>
<td>0.932</td>
<td>0.074</td>
</tr>
<tr>
<td>DirectVoxGO++ (ours)</td>
<td>33.953</td>
<td>0.980</td>
<td>0.018</td>
</tr>
</tbody>
</table>

As for memory usage and execution time for training on each scene, the original DirectVoxGO runs in around 15 minutes, while our technique and Plenoxels run in 25-28 minutes. The techniques tested used around 5 GB of GPU memory.

C. 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 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.

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 [10]. 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 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++ [24]. Since we successfully segment the foreground from the background, our foreground model can better model it with more high-frequency details.

D. Ablation Study

This section evaluates the impact of both of our proposals individually. In Table III, 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.

### IV. Conclusion

Our results showed that by combining DirectVoxGO [13] with background compositing [24] and neural hash encoding [8] we could achieve better results and a more accurate object reconstruction. Although our technique achieved good results in the evaluated dataset, we also mapped some limitations that suggest room for improvement. As Muller et al. [8] showed, using CUDA kernels and implementation in a lower-level language such as C++ as opposed to Python can significantly improve the performance time. Also, as we have shown with our problems with the Torch scene, a possible direction would be to integrate deformable NeRF [10] ideas into our pipeline and extend it to videos.

<table>
<thead>
<tr>
<th>Technique</th>
<th>PSNR↑</th>
<th>SSIM↑</th>
<th>LPIPS↓</th>
</tr>
</thead>
<tbody>
<tr>
<td>DirectVoxGO</td>
<td>20.461</td>
<td>0.658</td>
<td>0.366</td>
</tr>
<tr>
<td>DirectVoxGO+(BG)</td>
<td>21.118</td>
<td>0.727</td>
<td>0.319</td>
</tr>
<tr>
<td>DirectVoxGO+(NHE)</td>
<td>21.015</td>
<td>0.722</td>
<td>0.312</td>
</tr>
<tr>
<td>DirectVoxGO++ (ours)</td>
<td>22.436</td>
<td>0.804</td>
<td>0.266</td>
</tr>
</tbody>
</table>
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.

REFERENCES


