

Compressing Explicit Voxel Grid Representations: fast NeRFs become also small

Chenxi Lola Deng

Enzo Tartaglione

LTCI, Télécom Paris, Institut Polytechnique de Paris

Abstract

NeRFs have revolutionized the world of per-scene radiance field reconstruction because of their intrinsic compactness. One of the main limitations of NeRFs is their slow rendering speed, both at training and inference time. Recent research focuses on the optimization of an explicit voxel grid (EVG) that represents the scene, which can be paired with neural networks to learn radiance fields. This approach significantly enhances the speed both at train and inference time, but at the cost of large memory occupation. In this work we propose Re:NeRF, an approach that specifically targets EVG-NeRFs compressibility, aiming to reduce memory storage of NeRF models while maintaining comparable performance. We benchmark our approach with three different EVG-NeRF architectures on four popular benchmarks, showing Re:NeRF’s broad usability and effectiveness.

1. Introduction

The rising of Neural Radiance Fields (NeRF) techniques has heavily impacted the field of 3D scene modeling and reconstruction in recent years [25, 43, 13, 14, 11]. Efficient photo-realistic novel view generation from a fixed set of training images has been a popular area of research in computer vision with broad applications. The ability to distill the essence of the 3D object from 2D representations of it and its compactness is the main reason for making NeRF a high-impact approach in the literature.

The original NeRF [25] consists of a multi-layer perceptron, which implicitly learns the manifold representing the 3D object. Because of its great generalization for synthesizing novel viewpoints and the high compactness of the model itself, which typically consists of a few MB, NeRF has become a prevalent approach for 3D reconstruction. However, the NeRF’s MLP has to be queried million times to render a scene, leading to slow training and rendering time.

In an effort to speed up the vanilla NeRF, follow-up

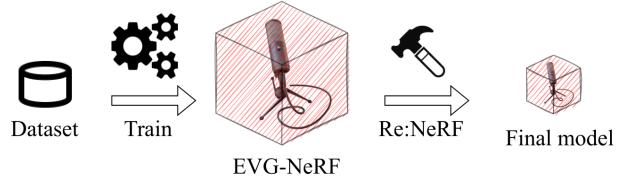


Figure 1: NeRF models with explicit voxel grid representations can be effectively compressed with Re:NeRF.

works introduced modifications to the original NeRF architecture [8, 3, 34]. One of the popular approaches, for example, encodes features of a scene in an explicit 3D voxel grid, combined with a tiny MLP. This group of methods, which utilizes an “explicit voxel grid” (EVG), is gaining more and more popularity due to the high training and rendering speed while maintaining or improving the performance of the original NeRF. Unlike traditional NeRF, EVG-NeRF models require larger memory, limiting their deployment in real-life applications, where models need to be shared through communication channels, or many of these models must be stored on memory-constrained devices.

In this work, we propose Re:NeRF, a method that reduces memory storage required by trained EVG-NeRF models. Its goal is to accurately separate the object from its background, discarding unnecessary features for rendering the scene, guided by the loss functions designed for training the specific EVG-NeRFs. Re:NeRF enables generation of highly compressed models with little or no performance loss: it is specifically designed for EVG-NeRFs as it exploits a spatial locality principle for adding-back voxels to the grid, and in such a sense its working flow resembles the one of a sculptor (Fig. 1). We observe that Re:NeRF enables high-level compression of pre-trained EVG-NeRF models, and that traditional general-purpose approaches, such as blind pruning, perform worse than Re:NeRF. We test Re:NeRF on four datasets with three recent EVG-NeRFs validating the effectiveness of the proposed approach.

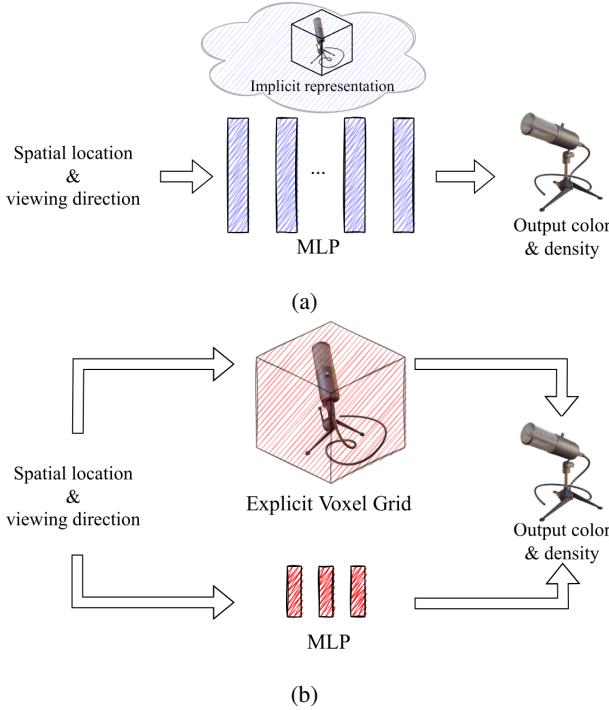


Figure 2: Visualisation of traditional NeRF approach, consisting of a multi-layer perceptron (a) and NeRF-based approaches with explicit voxel grid representation (b). The latter *can* also have a small MLP.

2. Related works

Rendering photo-realistic novel views of a 3D scene from a set of calibrated 2D images of the given scene has been a popular area of research in computer vision and computer graphics. Inspired by Mildenhall *et al*'s work in 2020, which proposed to capture the radiance and density field of a 3D scene entirely using a multi-layer perceptron (MLP) [25], a large number of follow-up studies have adopted the implicit representation of a scene. Here follows an overview of 3D representation models, neural radiance fields, and follow-up works.

3D representation for novel view synthesis. Inferring novel views of a scene given a set of images is a long-standing challenge in the field of computer graphics. Various scene representation techniques for 3D reconstruction have been studied in past decades. Light field rendering [4, 19, 31] directly synthesizes unobserved viewpoints by interpolating between sampled rays but it is slow to render and requires substantial computational resources. Meshes are another common technique that is easy to implement and allows rendering in real-time [5, 39, 41]. However, it struggles to capture fine geometry and topological information and its rendering quality is limited to mesh res-

olution. Differentiable methods have been recently proposed to perform scene reconstruction [6, 21, 33]. They use a differentiable ray-marching operation to encode and decode a latent representation of a scene and achieve excellent rendering quality.

Neural Radiance Fields. Unlike traditional explicit volumetric representation techniques, NeRF [25] stands out in recent years to be the most prevalent method for novel view rendering that infers photo-realistic views given a moderate number of input images. It encodes the entire content of the scene including view-dependent color emission and density into a single multi-layer perceptron (Fig. 2a) and achieves state-of-the-art quality. Besides, Neural Radiance Field-based approaches are proving on-the-field to have good generalization when undergoing several transformations, like changing environmental light [1, 32], image deformation [9, 27, 40] and are even usable in more challenging setups including meta learning [35], learn dynamically-changing scenes [10, 20, 23, 42] and even in generative contexts [2, 16, 29]. Compared to explicit representations, NeRF requires very little storage space, but on the contrary suffers from lengthy training time and very slow rendering speed, as the MLP is queried an extremely high number of times for rendering a single image.

NeRF with explicit voxel grids. To reduce inference and training time, explicit prior on the 3D object representation can be imposed. The most intuitive yet effective approach relies on splitting the 3D volume into small blocks, each of which is learned by a tiny NeRF model. With KiloNeRF [28], the advantage of doing this is twofold: the size of a single NeRF model is much smaller than the original one, reducing the latency time; secondly, the rendering process itself becomes parallelizable, as multiple pixels can be rendered simultaneously. The downside of this approach is that the granularity of the KiloNeRFs needs to be properly tuned, and the distillation of the single tinier NeRFs can be quite an expensive process. An interesting approach that leverages radiance fields with no explicit neural component is Plenoxels [8]. In this case, a sparse feature grid is encoded with 3D spherical harmonics (it belongs to EVG approaches without the MLP component in Fig. 2b). Hence, both the training time and the inference times are drastically improved, however, at the cost of a significant increment in-memory storage for the learned model, despite its sparse representation. Showing similar convergence time but maintaining an MLP component for complex view-dependent appearances, DVGO [34] proposes post-activation interpolation. Recently, in order to further improve the execution speed, TensoRF [3] has been proposed, which decomposes a 4D tensor into low-rank components prior to training. With the lower-quality rendering setup, the authors deliver a model of size comparable to the original NeRF, but with higher-quality rendering the memory

discrepancy with the vanilla NeRF model is still quite wide. **Compressing EVG-NeRF.** Whilst dense voxel-based representations increase rendering speed drastically, they require an order of magnitude more memory than implicit volumetric representations to achieve comparable rendering quality. Hierarchical structure representations using octrees allow the 3D scene to be encoded in a sparse manner, but the memory occupancy still remains high. Recent work addressed the problem of training a model with neural sparse voxel fields [22] progressively reducing the granularity of voxels and skipping the rendering for empty voxels. This approach, however, is designed for resource reallocation. While it improves the rendering speed, it still suffers from a long training time. To the best of our knowledge, Re:NeRF is the first approach focusing on compression specifically for EVG-NeRFs. While other works leverage the knowledge of sparsity of the 3D scene [22, 8, 34], they are focused on performance enhancement (fighting against artifacts which might appear in the empty space) and are not specific for compression. In this work, we are NeRF architecture agnostic, and our goal is to preserve the performance while reducing the model’s size.

3. Re:NeRF

In this section, we present Re:NeRF, our approach towards storage memory reduction for EVG-NeRFs. To reduce the model size, we iteratively remove parameters with the least ranked importance. Following each round of pruning, we design a strategy that adds back neighbor voxels to avoid a drop in performance.

3.1. Which parameters are important?

One of the key characteristics making EVG-NeRF an effective approach is the possibility of end-to-end training: given some target loss function \mathcal{L} evaluated on the rendered image, using back-propagation, it is possible to train all the parameters w of the model. This learning approach is common with any standard deep neural network, which allows us to build on top of the existing technique with the same set of optimizers (such as SGD and Adam). Methods based on mini-batches of samples have gained popularity, as they allow better generalization than stochastic learning while being memory and time efficient. They also benefit from libraries that exploit parallel computation on GPUs. In such a framework, a network parameter w_i is updated towards the averaged direction which minimizes the averaged loss for the mini-batch. Evidently, if the gradient’s magnitude is zero, the parameter is not updated, meaning that the local loss landscape for it is *flat*. A typical approach to reduce the number of parameters in a deep neural network is to *threshold* the parameters according to some hyper-parameters that

determine the amount to be removed [36, 7]:

$$w_i = \begin{cases} w_i & \text{if } |w_i| > \mathcal{Q}_{|w|}(\gamma) \\ 0 & \text{otherwise,} \end{cases} \quad (1)$$

where $\mathcal{Q}_{|w|}(\cdot)$ is the quantile function for the ℓ_1 norm of the parameters and $\gamma \in [0; 1]$ is the percentage of parameters to be removed. Despite its simplicity and broad application, this approach has a potential issue: parameters having very low magnitude can be important for the model. For example, a parameter can have a very low magnitude but a high gradient: hard-setting it to zero according to (1) can significantly influence the loss value/performance. Because of this, other works have suggested evaluating the importance of a parameter using the gradient of a parameter as a criterion [17, 37]. A parameter w_i can have a low gradient locally, but removing it may potentially impose a drastic change in both the loss value and its gradient. It is necessary, hence, to find a compromise between these two conditions. We can estimate the variation of the loss value using a Taylor series expansion truncated to the first order:

$$\Delta\mathcal{L}(w_i) \approx \frac{\partial\mathcal{L}}{\partial w_i} w_i, \quad (2)$$

and from (2) we can define how to remove the parameters according to

$$w_i = \begin{cases} w_i & \text{if } |\mathcal{L}(w_i)| > \mathcal{Q}_{|\Delta\mathcal{L}(w)|}(\gamma) \\ 0 & \text{otherwise.} \end{cases} \quad (3)$$

It is a known fact, however, that both gradient and weight magnitudes for the parameters change depending on the typology of layers taken into consideration [18]. Hence, in order to address a parameter-removing strategy that could be applied globally (hence, removing a given ratio of the parameters from the whole model, without imposing uniformity in this removal), the quantile function should be evaluated on the layer-normalized quantity

$$\Delta\hat{\mathcal{L}}(w_i) = \frac{\frac{\partial\mathcal{L}}{\partial w_i} w_i}{\max \left| \frac{\partial\mathcal{L}}{\partial w_j} w_j \right|}, w_j \text{ in same layer as } w_i. \quad (4)$$

Consequently, (3) becomes

$$w_i = \begin{cases} w_i & \text{if } |\mathcal{L}(w_i)| > \mathcal{Q}_{|\Delta\hat{\mathcal{L}}(w)|}(\gamma) \\ 0 & \text{otherwise.} \end{cases} \quad (5)$$

This strategy, however, evaluates the loss variation for each parameter w_i independently as in (2), which is known to be sub-optimal, as there is a dependence between parameters inside the model. How can we correct a potential “excessive” removal of parameters?

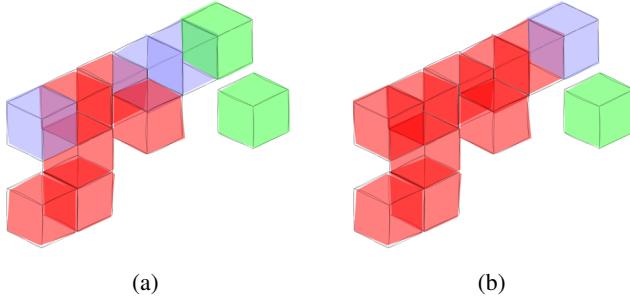


Figure 3: Effect of RE-INCLUDE before (a) and after running one iteration (b). In red: voxels already in the model; in green: non-neighbor voxels satisfying the re-inclusion rule; in blue: neighbor voxels satisfying the re-inclusion rule.

3.2. Removing only?

Removing parameters from a model is always a matter of delicacy: if the parameters are removed too fast, at some point the performance can not be recovered. On the contrary, if they are removed too slowly, the training complexity becomes large. Furthermore, the strategy to identify which parameters can be removed from the model, for a matter of efficiency, is limited to a first-order approximation in (2), making the parameter removal mechanism potentially prone to approximation errors. How can we identify the parameters, which have been removed, and should be added back in order not to degrade the performance excessively?

Let us consider the subset of parameters $\bar{\mathcal{W}}$ which have been removed. Since these parameters have been removed, according to (2), $\Delta\mathcal{L}(w_i) = 0 \forall w_i \in \bar{\mathcal{W}}$, meaning that this metric cannot be used to eventually re-include parameters in the model.

In order to determine whether the re-inclusion of a previously removed parameter will enhance the performance further (or in other words, will cause the minimization of the evaluated loss function) we can, for instance, look at the value for its gradient. If the gradient is above a given threshold, the parameter is added back. A simple threshold could be defined by the distribution of the magnitude of the gradients for the remaining parameters \mathcal{W} :

$$\left| \frac{\partial \mathcal{L}}{\partial w_i} \right| \geq \mathcal{Q}_{|\frac{\partial \mathcal{L}}{\partial w}|, w \in \mathcal{W}}(\delta) \Rightarrow w_i \in \mathcal{W}, \quad (6)$$

where $\delta \in [0; 1]$ determines the relative threshold for the re-inclusion.

Although (6) is a general rule and is potentially applicable to all the layers for EVG-NeRFs, we can leverage the voxel grid structure, imposing a prior over the 3D manifold representation for the object itself. We expect it to be *compact* and *the least sparse possible*. Towards this end,

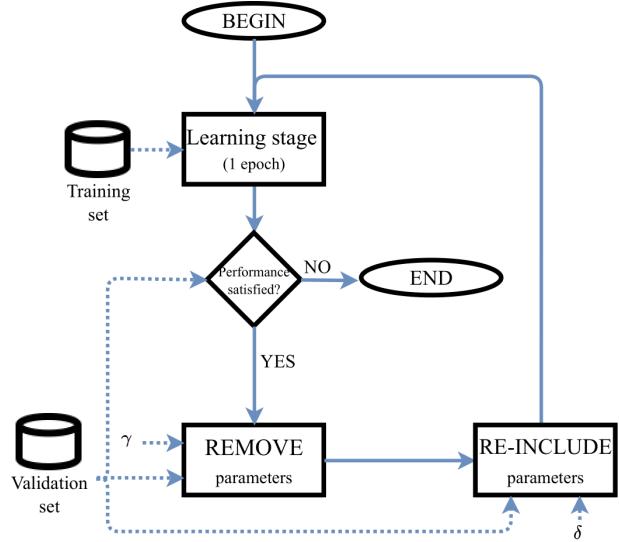


Figure 4: Overview on Re:NeRF. The dashed arrows indicate usage of some specific dataset/hyper-parameter at every stage.

we add, as an additional constraint to (6), that a parameter $w_i \in \bar{\mathcal{W}}$, in order to be re-included, it should also be connected, or should be a *neighbor* of some $w_j \in \mathcal{W}$. Hence, the re-inclusion rule becomes

$$\begin{aligned} \left| \frac{\partial \mathcal{L}}{\partial w_i} \right| &\geq \mathcal{Q}_{|\frac{\partial \mathcal{L}}{\partial w}|, w \in \mathcal{W}}(\delta) \\ &\wedge \\ \exists w_j \in \mathcal{W} | w_j &\in \Omega(w_i), \end{aligned} \quad (7)$$

where $\Omega(w_i)$ is the subset of parameters that are neighbors of w_i . Fig. 3a displays a practical case where there are some voxels not included (white space), voxels in the model (red), voxels removed which satisfy (7) (blue) and voxels which satisfy the condition on the gradient, but are not neighbors of any voxel in the model (green). After one re-inclusion iteration, the blue voxels are included, and some green voxels (the neighbors of the blue ones) become the new candidates for the re-inclusion (Fig. 3b). In order to find the whole subset of voxels to be added-back, it is necessary to iterate over the re-inclusion mechanism, until there are no voxels in blue to add. Follows an overview on Re:NeRF.

3.3. Overview on the Re:NeRF scheme

In this section, we provide an overview of Re:NeRF, which is displayed in Fig. 4. Given a pre-trained model, we perform a one-epoch fine-tuning on the model with the same policy as in the original NeRF model, moving then to the parameters removal/re-inclusion to determine the subset \mathcal{W} of parameters belonging to the model. Every time we perform a step of parameter removal, we follow the steps as in

Algorithm 1 Re:NeRF.

```

1: procedure RE:NERF( $\mathcal{W}_{beg}$ ,  $\gamma$ ,  $\delta$ )
2:    $T_{rem} \leftarrow \mathcal{Q}_{|\Delta\hat{\mathcal{L}}(w)|, w \in \mathcal{W}_{beg}}(\gamma)$ 
3:    $\mathcal{W}, \overline{\mathcal{W}} \leftarrow \text{REMOVE}(\mathcal{W}_{beg}, T_{rem})$ 
4:    $T_{inc} \leftarrow \mathcal{Q}_{|\frac{\partial \mathcal{L}}{\partial w}|, w \in \mathcal{W}}(\delta)$ 
5:    $\mathcal{W}_{end} \leftarrow \text{RE-INCLUDE}(\mathcal{W}, \overline{\mathcal{W}}, T_{inc})$ 
6:   return  $\mathcal{W}_{end}$ 
7: end procedure
8: procedure REMOVE( $\mathcal{W}_{beg}$ ,  $T_{rem}$ )
9:    $\mathcal{W} \leftarrow \emptyset$ 
10:   $\overline{\mathcal{W}} \leftarrow \emptyset$ 
11:  for  $w_i \in \mathcal{W}_{beg}$  do
12:    if  $|\Delta\mathcal{L}(w_i)| \geq T_{rem}$  then
13:       $\mathcal{W} \leftarrow \mathcal{W} \cup \{w_i\}$ 
14:    else
15:       $\overline{\mathcal{W}} \leftarrow \overline{\mathcal{W}} \cup \{w_i\}$ 
16:    end if
17:  end for
18:  return  $\mathcal{W}, \overline{\mathcal{W}}$ 
19: end procedure
20: procedure RE-INCLUDE( $\mathcal{W}, \overline{\mathcal{W}}, T_{inc}$ )
21:    $one\_added \leftarrow True$ 
22:   while  $one\_added$  do
23:      $one\_added \leftarrow False$ 
24:     for  $w_i \in \overline{\mathcal{W}}$  do
25:       if  $|\frac{\partial \mathcal{L}}{\partial w_i}| \geq T_{inc}$  then
26:          $\Omega \leftarrow \text{NEIGHBORS}(w_i)$ 
27:         if  $\Omega \cap \mathcal{W} \neq \emptyset$  then
28:            $\mathcal{W} \leftarrow \mathcal{W} \cup \{w_i\}$ 
29:            $one\_added \leftarrow True$ 
30:         end if
31:       end if
32:     end for
33:   end while
34:   return  $\mathcal{W}$ 
35: end procedure

```

Algorithm 1. In particular, we are asked a subset of parameters to belong to the model \mathcal{W}_{beg} and two hyper-parameters $\gamma \in [0; 1]$ and $\delta \in [0; 1]$: while γ determines how many parameters are (tentatively) removed after every step, δ determines how many parameters are (eventually) added back. Hence, we distinguish two phases for Re:NeRF: one (REMOVE) splits the model parameters into those dropping below or staying above a given threshold (line 3), and the other (RE-INCLUDE) re-includes the tentatively removed parameters that have both high derivative and are neighbors of other parameters in the model. This will favor lower loss (line 5). In particular, the latter might need to be run multiple times, every time at least one parameter is re-included (line 22). This is necessary as, every time a new parameter

is added to \mathcal{W} , the neighbor test as in line 27 potentially gives a different outcome.

We iterate over this until the performance does not drop below some pre-fixed performance threshold ΔT (from the original performance): when this happens, we end our training process. In order to save the model, the state dictionary is first quantized on 8-bits with a uniform quantizer and successively compressed using LZMA.

In the next section, we are going to present the results obtained on some common benchmarks for NeRFs.

4. Results

In this section, we present the empirical results obtained on state-of-the-art datasets and three different EVG-NeRF approaches, on top of which Re:NeRF has been executed in order to reduce the storage memory. For all the experiments the models have been pre-trained using the hyper-parameter setup indicated in the respective original work. As a common stop criterion, we impose a maximum worsening in performance ΔT of 1dB on the original model's PSNR. All the other hyper-parameters have been optimized using a grid-search algorithm. Although every technique requires a specific CUDA and PyTorch version, the Re:NeRF code is compatible with pytorch 1.12 and back-compatible with PyTorch 1.6. For all the experiments an NVIDIA A40 equipped with 40 GB has been used.¹

4.1. Setup

Datasets. We have evaluated our approach on four datasets. Synthetic-NeRF [25] and Synthetic-NSVF [22] are two popular datasets, containing 8 different realistic objects each, which are synthesized from NeRF (*chair, drums, ficus, hotdog, lego, materials, mic* and *ship*) and NSVF (*bike, lifestyle, palace, robot, spaceship, steamtrain, toad* and *wineholder*), respectively. For both, the image resolution has been set up to 800×800 pixels, having 100 views for training, 100 for validation, and 100 for testing. The third dataset we have tested is Tanks&Temples [15]: our choice fell on this dataset as it is a collection of real-world images. Here we use a subset of the provided samples (namely: *ignatius, truck, barn, caterpillar* and *family*). We use here FullHD resolution, and we use also in this case 10% of the images used for validation and 10% for testing. Finally, the fourth dataset we run our experiments is LLFF-NeRF [24]. Differently from the other three datasets, this dataset contains realistic images, and non blank background. Each scene consists of 20 to 60 forward-facing images with resolution 1008×756 . In this case, we have used all the 8 available samples (*fern, flower, fortress, horns, leaves, orchids, room* and *trex*).

Architectures and compressibility configuration. We

¹The source code will be made available at the conference's dates.

Table 1: Results obtained on low compressibility regime (LOW) and high compressibility (HIGH). The first line indicates the baseline. In every dataset, the various metrics are averaged for the samples in them.

Approach	Compress	Synthetic-NeRF			Synthetic-NSVF			Tanks&Temples			LLFF-NeRF		
		PSNR [dB](\uparrow)	SSIM (\uparrow)	Size [MB](\downarrow)	PSNR [dB](\uparrow)	SSIM (\uparrow)	Size [MB](\downarrow)	PSNR [dB](\uparrow)	SSIM (\uparrow)	Size [MB](\downarrow)	PSNR [dB](\uparrow)	SSIM (\uparrow)	Size [MB](\downarrow)
NSVF [22]	-	31.74	0.953	~ 16	35.13	0.979	~ 16	28.40	0.900	~ 16	-	-	-
Instant-NGP [26]	-	33.04	0.934	28.64	36.11	0.966	46.09	28.81	0.917	46.09	20.18	0.662	46.09
DVGO [34]	-	31.92	0.957	160.09	35.42	0.979	104.12	28.26	0.909	106.48	-	-	-
	LOW	31.47	0.952	3.99	35.29	0.974	4.37	28.22	0.910	4.69	-	-	-
	HIGH	31.08	0.944	2.00	34.90	0.969	2.46	27.90	0.894	1.62	-	-	-
TensoRF [3]	-	33.14	0.963	69.26	36.52	0.982	69.05	28.56	0.920	64.04	26.73	0.839	151.79
	LOW	33.26	0.962	11.47	36.44	0.982	11.60	28.50	0.916	9.99	26.80	0.820	32.34
	HIGH	32.81	0.956	7.94	36.14	0.978	8.52	28.24	0.907	6.70	26.55	0.797	20.27
Plenoxels [8]	-	31.48	0.956	189.08	-	-	-	27.37	0.904	147.96	25.90	0.838	1484.96
	LOW	31.52	0.952	91.77	-	-	-	27.66	0.909	102.26	26.24	0.838	457.23
	HIGH	30.97	0.944	54.68	-	-	-	27.34	0.896	85.47	25.95	0.828	338.02

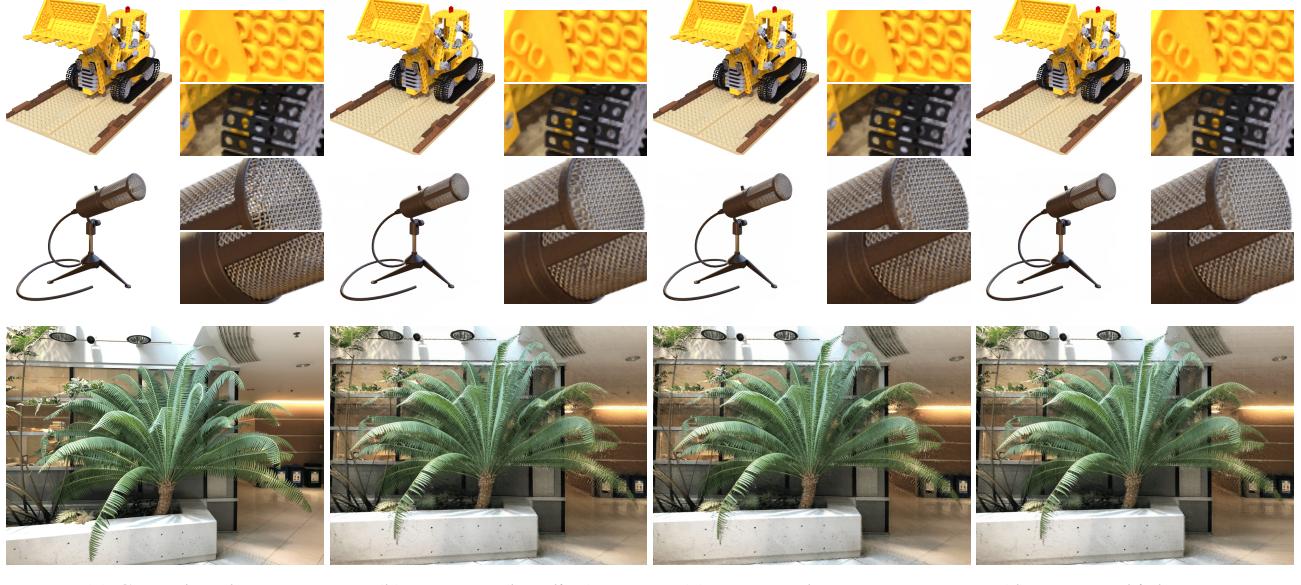


Figure 5: Qualitative results for “lego” (top), “mic” (middle) and “fern” (bottom).

have tested Re:NeRF on three very different EVG-NeRF approaches: DVGO [34], TensoRF [3] and Plenoxels [8].² DVGO models are trained using the same configuration as in the paper, in the 160^3 voxel grid size configuration. TensoRF models were obtained with their default 192-VM configuration, which factorizes tensors into 192 low-rank components and optimizes the model for 30k steps.

²Although Plenoxels is a method for learning radiance fields and does not have any “neural network”, it still leverages the same optimization tools. We include it in our experimental setup to show the even broader adaptability of Re:NeRF to any approach minimizing a differentiable loss function.

Plenoxel models have obtained training first on 128^3 grid, up-sampled to 256^3 , and finally to 512^3 . For all the architectures and datasets we have used $\gamma = 0.5$ and $\delta = 0.5$, except for Plenoxel trained on the Synthetic-NeRF and LLFF-NeRF datasets, where $\gamma = 0.66$ has been used. For a matter of comparison with other efficiencing approaches, we compare our results also with NSVF [22] and with Instant-NGP [26].

4.2. Discussion

All the results are reported in Table 1. Here the “LOW” compressibility refers to compressibility achieved with the

best PSNR evaluated on the validation set, while “HIGH” refers to the model achieved right before reaching the stop criterion (which consists of a worsening of the original performance of at most 1dB on the original PSNR). Some qualitative results are also displayed in Fig. 5.

In general, we observe that Re:NeRF effectively reduces the size of the models in all the combinations of tested datasets/EVG-NeRFs, with different impacts depending on the EVG-NeRF it is applied. In general, the approach having higher average sizes while also having slightly worse performance is Plenoxels [8], which is an EVG approach with no neural elements in it. Nevertheless, Re:NeRF is able to compress it effectively. In particular, in the low compressibility setup, the performance is improved with an overall size reduction. DVGO [34], consisting of a voxel grid and of an MLP component, is massive, achieving for example compression ratios of $80\times$ for Synthetic-NeRF and $65\times$ within the 1dB performance loss. The approach generally achieving better performance is TensoRF [3], where the low compression setup maintains almost the same performance still enabling $6\times$ compression. When compared to DVGO, TensoRF occupies less memory as it relies on factorized neural radiance fields in the 4D voxel grid, namely it is by design more efficient at training time, and of course in order to maintain such a higher performance the possible compressibility of the model is relatively limited. When compared with other approaches, we observe in general a significant improvement in performance for similar model’s size (LLFF-NeRF) or a significantly lower memory for similar performance (in the other three cases).

4.3. Ablation study

In this section, we propose the ablation study for Re:NeRF. In particular, we want to evidence the single contributions of the proposed technique, emphasizing their effect. Towards this end, we have conducted experiments on “Mic” from the Synthetic-NeRF dataset and used DVGO [34] as the EVG-NeRF approach. The summary for the ablation study is enclosed in Table 2. All the measures here proposed are averaged on 3 different runs.

Remove all the layers or a subset of them? Considering the heterogeneity of the layers in the EVG-NeRF approaches, it is not straightforward that removing parameters from all the layers is the best approach. Indeed, we observe that focusing on the layers with explicit voxel representation (indicated as “Voxel”) leads to a similar PSNR as the baseline (33.19 dB) with a very high size reduction (from 67.69MB to 7.02MB). Focusing on all the layers of the model, as it would be done in a generic model pruning scheme [12, 38, 7] leads to a very high drop in performance (26.72dB, namely -6.42dB when compared to the baseline). This shows how important it is to focus on voxels and designing specific solutions rather than relying on generic ap-

proaches.

Re-including helps. The proposed strategy needs a “balancing” for the voxel removal phase, which can be extreme. Towards this end, re-including some removed voxels slightly increases the size of the model, which however turns into performance recovery. In particular, by adding just 0.10MB we gain 0.07dB: please notice that the baseline PSNR is lower than the achieved performance with remove+re-include. This phenomenon is even more evident in the high compressibility regime, where we gain approximately 2dB with just 0.16MB added.

Effect of quantization. In traditional NeRF models quantizing is a delicate process, requiring non-uniform, custom quantization strategies [30]. In our case, however, quantizing on 8 bits maintains the performance to high PSNR values (losing 0.16 dB without additional fine-tuning) but significantly reduces the size of the compressed model (from 7.12MB to 1.48MB). This is very evident in the high compressibility result, where we move from 1.24MB to 0.34MB only.

4.4. A deeper view on Re:NeRF’s effect

As a final analysis, we wish to test what happens in the voxel grid for a baseline model and for the same with Re:NeRF applied. Fig. 6 visualizes the content of the density.grid layer for the baseline (up, in red) and for the compressed one (down, in blue). Looking at the spatial occupancy for the density grid, without Re:NeRF evidently we have a much higher than necessary voxel occupation (Fig. 6a) which is trimmed to the real object shape by Re:NeRF (Fig. 6d). Looking at the effective value of each voxel (here normalized and modeled as transparency) we can easily guess the structure of the object in the Re:NeRF case (Fig. 6e) while the density is so spread in the space for the baseline case that the object is almost impossible to distinguish (Fig. 6b). This has a clear effect on the distribution of the parameter’s value for the layer: while in the baseline case we observe very different behavior for positive and negative values, making problems like compression and quantization harder (Fig. 6c), the distribution tends to be more specular when applying Re:NeRF (Fig. 6f): this is due to both the suppression of irrelevant parameters in the model and to the exclusive re-inclusion of parameters having as neighbors others already included.

5. Conclusion & future works

In this work we have presented Re:NeRF, an approach to compress NeRF models that utilizes explicit voxel grid representations. This approach removes parameters from the model, while at the same time ensures not to have a large drop in performance. This is achieved by a re-inclusion mechanism, which allows previously removed parameters

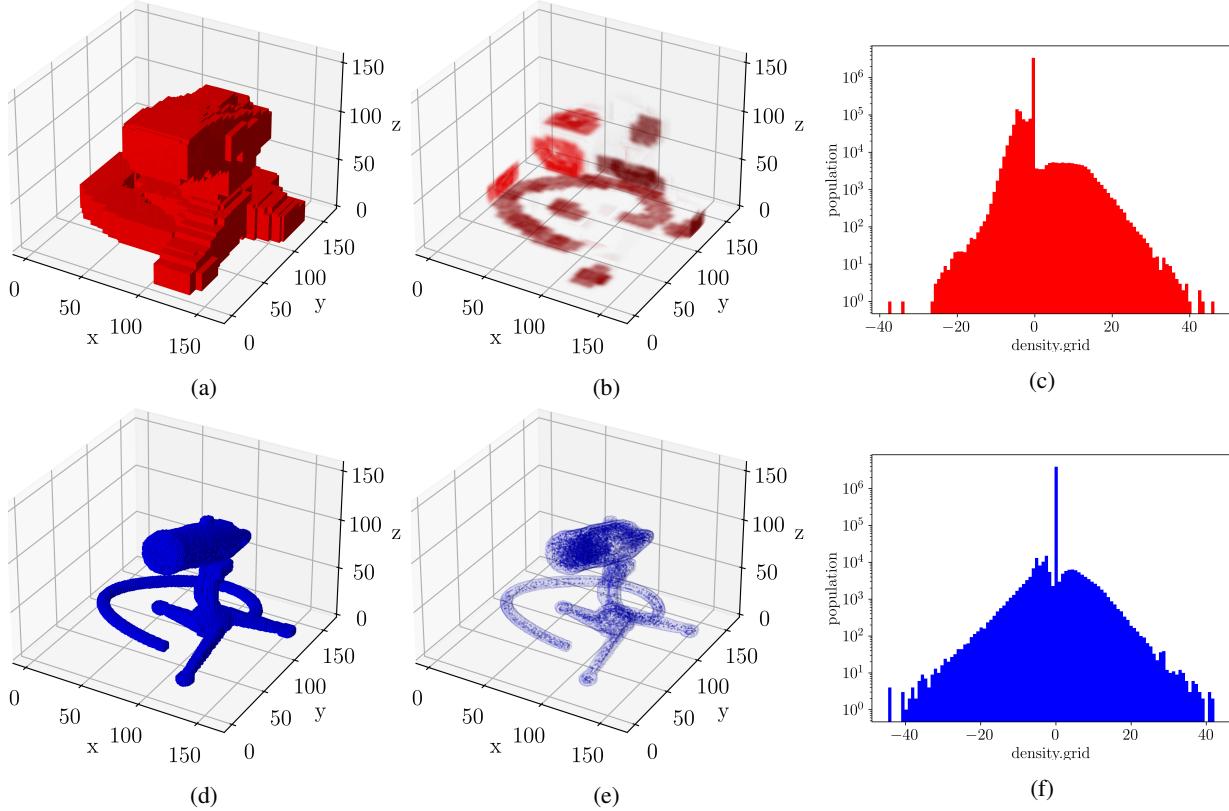


Figure 6: Visualization of the `density.grid` layer for DVGO [34] trained on “Mic” (Synthetic-NeRF). Up: baseline model; down: Re:NeRF applied. Here are visualized the non-empty voxels (a, d), their effective value (b, e), and the distribution of their values, in log scale (c, f). For visualization, the values in (b) have been amplified by a factor $10\times$.

Table 2: Ablation study conducted on “Mic” from Synthetic-NeRF. The approach used here is DVGO [34]. The first line is the reference baseline.

Layers	Remove	Re-include	Quantization	LOW compressibility		HIGH compressibility	
				PSNR[dB]	Size[MB]	PSNR[dB]	Size[MB]
\times	\times	\times	\times	33.15	67.69	-	-
All	✓	\times	\times	26.72	6.88	25.09	0.87
Voxels	✓	\times	\times	33.19	7.02	27.67	1.08
Voxels	✓	✓	\times	33.26	7.12	29.54	1.24
Voxels	✓	✓	✓	33.10	1.48	29.41	0.34

that are neighbors of the remaining parameters to be re-included if they show high gradient loss. Re:NeRF is easily deployable for any model, having different architecture, training strategy, or objective function. For this reason, we have tested its effectiveness on three very different approaches: DVGO [34], where a part of the model learns the density and the other maps complex voxel dependencies with an MLP, TensoRF [3] which learns a 4D grid and performs low-rank decomposition on the radiance fields, and

Plenoxels [8] which optimized the voxel grid directly with no MLP supporting the learning. These approaches have been tested on four popular datasets, two synthetic and two from real images.

In all the cases, Re:NeRF is able to compress the approaches with compression rates scaling up to $80\times$. Reducing the storage memory required by these models, designed mainly to improve training and inference time but sacrificing storage memory when compared to the original

NeRF [25], further emphasizes EVG-NeRF’s benefits and pushes towards their large-scale deployability in memory-constrained or bandwidth-limited applications. Interestingly, in a low compressibility setup, the performance is essentially unharmed, while the model is effectively compressed. This opens the road towards the model’s budget re-allocation, like efficient ensembling, towards further performance enhancement with specific memory constraints.

References

- [1] Mark Boss, Raphael Braun, Varun Jampani, Jonathan T Barron, Ce Liu, and Hendrik Lensch. Nerd: Neural reflectance decomposition from image collections. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 12684–12694, 2021.
- [2] Eric R Chan, Marco Monteiro, Petr Kellnhofer, Jiajun Wu, and Gordon Wetzstein. pi-gan: Periodic implicit generative adversarial networks for 3d-aware image synthesis. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 5799–5809, 2021.
- [3] Anpei Chen, Zexiang Xu, Andreas Geiger, Jingyi Yu, and Hao Su. Tensorf: Tensorial radiance fields. *ECCV*, 2022.
- [4] Abe Davis, Marc Levoy, and Fredo Durand. Unstructured light fields. In *Computer Graphics Forum*, volume 31, pages 305–314. Wiley Online Library, 2012.
- [5] Paul E Debevec, Camillo J Taylor, and Jitendra Malik. Modeling and rendering architecture from photographs: A hybrid geometry-and image-based approach. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 11–20, 1996.
- [6] John Flynn, Michael Broxton, Paul Debevec, Matthew DuVall, Graham Fyffe, Ryan Overbeck, Noah Snavely, and Richard Tucker. Deepview: View synthesis with learned gradient descent. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2367–2376, 2019.
- [7] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *ICLR*, 2019.
- [8] Sara Fridovich-Keil, Alex Yu, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. Plenoxels: Radiance fields without neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5501–5510, 2022.
- [9] Guy Gafni, Justus Thies, Michael Zollhofer, and Matthias Nießner. Dynamic neural radiance fields for monocular 4d facial avatar reconstruction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8649–8658, 2021.
- [10] Chen Gao, Ayush Saraf, Johannes Kopf, and Jia-Bin Huang. Dynamic view synthesis from dynamic monocular video. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5712–5721, 2021.
- [11] Pengsheng Guo, Miguel Angel Bautista, Alex Colburn, Liang Yang, Daniel Ulbricht, Joshua M Susskind, and Qi Shan. Fast and explicit neural view synthesis. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 3791–3800, 2022.
- [12] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems*, 28, 2015.
- [13] Yi-Hua Huang, Yue He, Yu-Jie Yuan, Yu-Kun Lai, and Lin Gao. Stylizednerf: consistent 3d scene stylization as stylized nerf via 2d-3d mutual learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 18342–18352, 2022.
- [14] Berk Kaya, Suryansh Kumar, Francesco Sarno, Vittorio Ferrari, and Luc Van Gool. Neural radiance fields approach to deep multi-view photometric stereo. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 1965–1977, 2022.
- [15] Arno Knapitsch, Jaesik Park, Qian-Yi Zhou, and Vladlen Koltun. Tanks and temples: Benchmarking large-scale scene reconstruction. *ACM Transactions on Graphics (ToG)*, 36(4):1–13, 2017.
- [16] Adam R Kosiorek, Heiko Strathmann, Daniel Zoran, Pol Moreno, Rosalia Schneider, Sona Mokrá, and Danilo Jimenez Rezende. Nerf-vae: A geometry aware 3d scene generative model. In *International Conference on Machine Learning*, pages 5742–5752. PMLR, 2021.
- [17] Yann LeCun, John Denker, and Sara Solla. Optimal brain damage. *Advances in neural information processing systems*, 2, 1989.
- [18] Namhoon Lee, Thalaiyasingam Ajanthan, and Philip HS Torr. Snip: Single-shot network pruning based on connection sensitivity. *ICLR*, 2019.
- [19] Anat Levin and Fredo Durand. Linear view synthesis using a dimensionality gap light field prior. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1831–1838. IEEE, 2010.
- [20] Zhengqi Li, Simon Niklaus, Noah Snavely, and Oliver Wang. Neural scene flow fields for space-time view synthesis of dynamic scenes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6498–6508, 2021.
- [21] Zhengqi Li, Wenqi Xian, Abe Davis, and Noah Snavely. Crowdsampling the plenoptic function. In *European Conference on Computer Vision*, pages 178–196. Springer, 2020.
- [22] Lingjie Liu, Jiatao Gu, Kyaw Zaw Lin, Tat-Seng Chua, and Christian Theobalt. Neural sparse voxel fields. *Advances in Neural Information Processing Systems*, 33:15651–15663, 2020.
- [23] Ricardo Martin-Brualla, Noha Radwan, Mehdi SM Sajjadi, Jonathan T Barron, Alexey Dosovitskiy, and Daniel Duckworth. Nerf in the wild: Neural radiance fields for unconstrained photo collections. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7210–7219, 2021.
- [24] Ben Mildenhall, Pratul P Srinivasan, Rodrigo Ortiz-Cayon, Nima Khademi Kalantari, Ravi Ramamoorthi, Ren Ng, and Abhishek Kar. Local light field fusion: Practical view syn-

- thesis with prescriptive sampling guidelines. *ACM Transactions on Graphics (TOG)*, 38(4):1–14, 2019.
- [25] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *European conference on computer vision*, pages 405–421. Springer, 2020.
 - [26] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Trans. Graph.*, 41(4):102:1–102:15, July 2022.
 - [27] Atsuhiro Noguchi, Xiao Sun, Stephen Lin, and Tatsuya Harada. Neural articulated radiance field. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5762–5772, 2021.
 - [28] Christian Reiser, Songyou Peng, Yiyi Liao, and Andreas Geiger. Kilonerf: Speeding up neural radiance fields with thousands of tiny mlps. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 14335–14345, 2021.
 - [29] Katja Schwarz, Yiyi Liao, Michael Niemeyer, and Andreas Geiger. Graf: Generative radiance fields for 3d-aware image synthesis. *Advances in Neural Information Processing Systems*, 33:20154–20166, 2020.
 - [30] Jinglei Shi and Christine Guillemot. Distilled low rank neural radiance field with quantization for light field compression. *arXiv preprint arXiv:2208.00164*, 2022.
 - [31] Lixin Shi, Haitham Hassanieh, Abe Davis, Dina Katabi, and Fredo Durand. Light field reconstruction using sparsity in the continuous fourier domain. *ACM Transactions on Graphics (TOG)*, 34(1):1–13, 2014.
 - [32] Pratul P Srinivasan, Boyang Deng, Xiuming Zhang, Matthew Tancik, Ben Mildenhall, and Jonathan T Barron. Nerv: Neural reflectance and visibility fields for relighting and view synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7495–7504, 2021.
 - [33] Pratul P Srinivasan, Richard Tucker, Jonathan T Barron, Ravi Ramamoorthi, Ren Ng, and Noah Snavely. Pushing the boundaries of view extrapolation with multiplane images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 175–184, 2019.
 - [34] Cheng Sun, Min Sun, and Hwann-Tzong Chen. Direct voxel grid optimization: Super-fast convergence for radiance fields reconstruction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5459–5469, 2022.
 - [35] Matthew Tancik, Ben Mildenhall, Terrance Wang, Divi Schmidt, Pratul P Srinivasan, Jonathan T Barron, and Ren Ng. Learned initializations for optimizing coordinate-based neural representations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2846–2855, 2021.
 - [36] Enzo Tartaglione, Andrea Bragagnolo, Attilio Fiandratti, and Marco Grangetto. Loss-based sensitivity regularization: towards deep sparse neural networks. *Neural Networks*, 146:230–237, 2022.
 - [37] Enzo Tartaglione, Andrea Bragagnolo, Francesco Odierna, Attilio Fiandratti, and Marco Grangetto. Serene: Sensitivity-based regularization of neurons for structured sparsity in neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 2021.
 - [38] Enzo Tartaglione, Skjalg Lepsøy, Attilio Fiandratti, and Gianluca Francini. Learning sparse neural networks via sensitivity-driven regularization. *Advances in neural information processing systems*, 31, 2018.
 - [39] Justus Thies, Michael Zollhöfer, and Matthias Nießner. Deferred neural rendering: Image synthesis using neural textures. *ACM Transactions on Graphics (TOG)*, 38(4):1–12, 2019.
 - [40] Edgar Tretschk, Ayush Tewari, Vladislav Golyanik, Michael Zollhöfer, Christoph Lassner, and Christian Theobalt. Non-rigid neural radiance fields: Reconstruction and novel view synthesis of a dynamic scene from monocular video. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 12959–12970, 2021.
 - [41] Michael Waechter, Nils Moehrle, and Michael Goesele. Let there be color! large-scale texturing of 3d reconstructions. In *European conference on computer vision*, pages 836–850. Springer, 2014.
 - [42] Wenqi Xian, Jia-Bin Huang, Johannes Kopf, and Changil Kim. Space-time neural irradiance fields for free-viewpoint video. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9421–9431, 2021.
 - [43] Alex Yu, Vickie Ye, Matthew Tancik, and Angjoo Kanazawa. pixelnerf: Neural radiance fields from one or few images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4578–4587, 2021.

A. Detailed results

Here follow the detailed tables for Synthetic-NeRF (Table 3), Synthetic-NSVF (Table 4) and Tanks & Temples (Table 6). For these, we also include, as output quality metric, the LPIPS score evaluated on the VGG backbone. Besides, we also provide evaluations on the BlendedMVS dataset for DVGO (Table 5), whose configuration follows.

A.1. Configuration for BlendedMVS

BlendedMVS, although being a synthetic dataset, differently from Synthetic-NeRF and Synthetic-NSVF, has more realistic ambient lighting, which is taken from real image blending. In this case, following the same approach as [34], we have used a subset of 4 objects (namely: *jade*, *fountain*, *character* and *statue*). We have here used as image resolution 768×576 pixels; 10% of the images are used for validation and 10% for testing. For Re:NeRF, we have used $\gamma = 0.5$ and $\delta = 0.5$, with $\Delta T = 1\text{dB}$.

Table 3: Results for Synthetic-NeRF.

Synthetic-NeRF											
	Architecture	Pruning	Chair	Drums	Ficus	Hotdog	Lego	Materials	Mic	Ship	Avg
PSNR(dB) (\uparrow)	DVGO [34]	-	34.11	25.48	32.59	36.77	34.69	29.52	33.16	29.04	31.92
		LOW	33.75	25.34	32.36	36.00	34.30	29.26	33.16	28.69	31.61
		HIGH	33.45	24.98	32.11	35.44	33.90	28.38	32.77	28.60	31.20
	Plenoxels [8]	-	33.98	25.35	31.83	36.43	34.10	29.14	33.26	27.78	31.48
		LOW	34.35	25.09	31.69	36.33	34.40	28.73	33.92	27.71	31.52
	TensoRF [3]	HIGH	33.65	24.96	31.21	35.44	33.91	28.15	33.14	27.31	30.97
Size(MB) (\downarrow)		-	35.76	26.01	33.99	37.41	36.46	30.12	34.61	30.77	33.14
DVGO [34]	LOW	36.00	26.01	34.12	37.58	36.73	30.01	34.67	30.92	33.26	
	HIGH	35.66	25.59	33.57	37.35	36.38	29.64	34.00	30.31	32.81	
Plenoxels [8]	-	103.994	92.06	108.71	130.01	124.201	171.36	49.41	100.98	110.09	
	LOW	4.44	2.62	2.67	5.19	5.38	7.47	1.50	5.43	4.33	
TensoRF [3]	HIGH	2.53	1.21	1.82	2.41	2.85	3.06	0.89	2.85	2.20	
	-	187.04	160.83	108.39	290.64	291.65	196.15	80.83	197.08	189.08	
SSIM(\uparrow)	DVGO [34]	LOW	85.59	56.34	38.51	179.26	183.48	91.25	37.69	62.07	91.77
		HIGH	47.26	42.34	28.8	96.81	97.78	66.4	21.07	36.94	54.68
	Plenoxels [8]	-	65.46	65.62	67.87	77.61	65.75	80.06	64.45	67.12	69.24
		LOW	12.78	8.47	13.30	14.77	12.82	15.65	7.67	8.77	11.78
	TensoRF [3]	HIGH	8.51	6.11	8.88	9.47	8.41	10.31	5.58	6.26	7.94
		-	0.976	0.930	0.977	0.986	0.976	0.950	0.983	0.878	0.957
LPIPS _{VGG} (\downarrow)	DVGO [34]	LOW	0.974	0.924	0.975	0.973	0.973	0.943	0.982	0.871	0.952
		HIGH	0.971	0.916	0.962	0.969	0.967	0.915	0.981	0.867	0.943
	Plenoxels [8]	-	0.977	0.933	0.976	0.98	0.975	0.949	0.985	0.869	0.956
		LOW	0.978	0.922	0.97	0.979	0.976	0.939	0.985	0.867	0.952
	TensoRF [3]	HIGH	0.971	0.915	0.97	0.972	0.972	0.923	0.977	0.854	0.944
		-	0.985	0.937	0.982	0.982	0.983	0.952	0.988	0.895	0.963
LPIPS _{VGG} (\downarrow)	TensoRF [3]	LOW	0.985	0.931	0.982	0.982	0.983	0.947	0.987	0.894	0.962
		HIGH	0.983	0.917	0.979	0.980	0.981	0.939	0.984	0.882	0.956
	DVGO [34]	-	0.027	0.079	0.025	0.034	0.027	0.059	0.018	0.161	0.054
		LOW	0.035	0.090	0.033	0.060	0.032	0.070	0.022	0.167	0.064
	Plenoxels [8]	HIGH	0.038	0.103	0.037	0.067	0.039	0.102	0.026	0.171	0.073
		-	0.031	0.067	0.026	0.037	0.028	0.057	0.015	0.178	0.055
LPIPS _{VGG} (\downarrow)	TensoRF [3]	LOW	0.026	0.081	0.038	0.043	0.027	0.074	0.019	0.18	0.061
		HIGH	0.033	0.088	0.044	0.062	0.033	0.090	0.031	0.193	0.072
	DVGO [34]	-	0.022	0.073	0.022	0.032	0.018	0.058	0.015	0.138	0.047
		LOW	0.022	0.103	0.026	0.035	0.018	0.067	0.022	0.138	0.054
	Plenoxels [8]	HIGH	0.028	0.157	0.042	0.045	0.022	0.081	0.042	0.159	0.072

Table 4: Results for Synthetic-NSVF.

Synthetic-NSVF											
	Architecture	Pruning	Bike	Lifestyle	Palace	Robot	Spaceship	Steamtrain	Toad	Wineholder	Avg
PSNR(dB)(\uparrow)	DVGO [34]	-	38.13	33.64	34.32	36.23	37.56	36.47	33.02	30.21	34.95
		LOW	38.16	33.68	34.47	36.29	37.26	36.10	32.98	30.11	34.88
		HIGH	37.97	33.16	33.88	36.00	36.82	35.79	32.39	29.75	34.47
	TensoRF [3]	-	39.23	34.51	37.56	38.26	38.6	37.87	31.32	34.85	36.53
		LOW	39.38	34.68	37.92	38.72	38.58	38.06	34.85	31.77	36.75
		HIGH	38.90	34.33	37.53	38.40	38.10	37.40	33.20	31.23	36.14
Size(MB)(\downarrow)	DVGO [34]	-	104.10	97.12	105.00	97.17	128.31	144.64	128.30	97.71	112.79
		LOW	3.55	3.53	4.84	3.76	4.95	5.41	5.67	3.30	4.38
		HIGH	2.52	2.38	2.63	2.55	2.78	2.77	1.87	1.65	2.39
	TensoRF [3]	-	70.92	65.46	64.94	68.63	68.01	80.02	68.71	65.74	69.05
		LOW	13.74	12.84	12.67	13.15	13.38	15.39	14.24	12.77	13.52
		HIGH	9.11	8.47	8.36	8.80	8.83	7.12	9.12	8.35	8.52
SSIM(\uparrow)	DVGO [34]	-	0.991	0.964	0.992	0.992	0.987	0.989	0.965	0.949	0.979
		LOW	0.991	0.963	0.961	0.991	0.985	0.986	0.966	0.950	0.974
		HIGH	0.990	0.958	0.953	0.991	0.982	0.982	0.957	0.942	0.969
	TensoRF [3]	-	0.993	0.968	0.979	0.994	0.989	0.991	0.961	0.978	0.982
		LOW	0.993	0.968	0.980	0.995	0.988	0.991	0.978	0.963	0.982
		HIGH	0.992	0.963	0.978	0.994	0.985	0.988	0.966	0.957	0.978
LPIPS _{VGG} (\downarrow)	DVGO [34]	-	0.011	0.055	0.045	0.013	0.020	0.019	0.047	0.059	0.034
		LOW	0.015	0.056	0.043	0.013	0.024	0.027	0.045	0.055	0.035
		HIGH	0.015	0.063	0.050	0.013	0.028	0.036	0.054	0.067	0.041
	TensoRF [3]	-	0.003	0.021	0.011	0.003	0.009	0.006	0.024	0.016	0.012
		LOW	0.011	0.049	0.020	0.010	0.022	0.017	0.035	0.054	0.027
		HIGH	0.016	0.061	0.022	0.011	0.027	0.029	0.059	0.077	0.038

Table 5: Results for BlendedMVS

BlendedMVS							
	Architecture	Pruning	Character	Fountain	Jade	Statue	Avg
PSNR(dB)(\uparrow)	DVGO [34]	-	30.26	28.27	27.75	26.41	28.17
		LOW	30.05	28.21	27.48	26.08	27.86
		HIGH	29.78	27.90	27.08	25.97	27.68
Size(MB)(\downarrow)	DVGO [34]	-	131.73	72.33	158.03	97.08	114.79
		LOW	5.43	3.54	5.24	2.81	4.26
		HIGH	2.92	1.70	1.85	1.85	2.08
SSIM(\uparrow)	DVGO [34]	-	0.963	0.923	0.916	0.887	0.922
		LOW	0.960	0.921	0.909	0.876	0.917
		HIGH	0.957	0.910	0.887	0.867	0.908
LPIPS _{VGG} (\downarrow)	DVGO [34]	-	0.046	0.116	0.106	0.137	0.101
		LOW	0.048	0.114	0.107	0.142	0.103
		HIGH	0.052	0.126	0.129	0.151	0.115

Table 6: Results for Tanks & Temples.

Tanks & Temples								
	Architecture	Pruning	Barn	Caterpillar	Family	Ignatius	Truck	Avg
PSNR(dB)(↑)	DVGO [34]	-	26.84	25.70	33.68	28.00	27.09	28.26
		LOW	26.76	25.67	33.60	28.06	27.04	28.23
		HIGH	26.32	25.22	33.36	27.86	26.78	27.91
	Plenoxels [8]	-	25.95	24.63	32.25	27.49	26.52	27.37
		LOW	26.58	24.78	32.86	27.22	26.87	27.66
		HIGH	26.31	24.40	32.29	27.00	26.69	27.34
Size(MB)(↓)	TensoRF	-	28.34	27.14	27.22	26.19	33.92	28.56
		LOW	27.28	26.09	33.75	28.06	27.32	28.50
		HIGH	26.99	25.77	33.36	27.86	27.20	28.40
	DVGO [34]	-	128.21	109.94	92.72	95.10	106.43	106.48
		LOW	5.52	5.23	3.85	3.51	5.35	4.69
		HIGH	1.75	1.89	2.37	1.08	1.87	1.79
SSIM(↑)	Plenoxels [8]	-	282.85	133.43	103.64	115.81	104.08	147.96
		LOW	213.45	87.08	73.16	93.94	43.65	102.26
		HIGH	181.09	66.67	60.53	84.06	35.01	85.47
	TensoRF	-	73.95	64.56	60.06	61.25	65.36	65.04
		LOW	9.39	8.16	7.40	12.13	12.87	9.99
		HIGH	6.62	5.69	5.17	7.66	8.34	6.70
LPIPS _{VGG} (↓)	DVGO [34]	-	0.836	0.904	0.961	0.941	0.905	0.909
		LOW	0.838	0.904	0.962	0.941	0.904	0.910
		HIGH	0.826	0.859	0.958	0.931	0.895	0.894
	Plenoxels [8]	-	0.828	0.899	0.954	0.942	0.899	0.904
		LOW	0.856	0.894	0.959	0.935	0.902	0.909
		HIGH	0.844	0.871	0.95	0.923	0.892	0.896
	TensoRF	-	0.948	0.914	0.864	0.912	0.965	0.920
		LOW	0.862	0.901	0.961	0.941	0.913	0.916
		HIGH	0.852	0.888	0.956	0.934	0.907	0.907
LPIPS _{Net} (↓)	DVGO [34]	-	0.297	0.171	0.071	0.089	0.162	0.158
		LOW	0.291	0.172	0.079	0.091	0.161	0.159
		HIGH	0.312	0.194	0.073	0.107	0.174	0.172
	Plenoxels [8]	-	0.306	0.169	0.081	0.102	0.167	0.165
		LOW	0.263	0.174	0.071	0.112	0.155	0.155
		HIGH	0.285	0.200	0.081	0.126	0.165	0.171
	TensoRF	-	0.078	0.145	0.252	0.159	0.064	0.140
		LOW	0.258	0.187	0.067	0.087	0.149	0.149
		HIGH	0.277	0.211	0.077	0.096	0.169	0.166

Table 7: Results for forward-facing scenes from NeRF

Forward-facing-NeRF											
	Architecture	Pruning	Fern	Flower	Fortress	Horns	Leaves	Orchids	Room	Trex	Avg
PSNR(dB)(↑)	Plenoxel [8]	-	24.57	27.64	30.17	27.10	21.56	20.54	29.20	26.41	25.90
		LOW	25.45	27.82	30.57	27.51	21.38	20.37	30.33	26.50	26.24
		HIGH	25.23	27.65	30.18	26.98	21.12	20.32	29.76	26.27	25.94
	TensoRF [3]	-	25.27	28.60	31.36	28.14	21.30	19.87	32.35	26.97	26.73
		LOW	24.50	28.64	31.30	28.87	21.22	19.31	33.33	27.26	26.80
		HIGH	24.40	28.29	31.09	28.49	20.78	19.08	33.10	27.16	26.55
Size(MB)(↓)	Plenoxel [8]	-	1658.40	1471.78	1407.66	1726.17	1851.71	720.01	1421.66	1622.31	1484.96
		LOW	407.02	726.00	383.61	432.01	438.90	221.51	495.84	552.94	457.23
		HIGH	305.97	523.17	291.59	321.62	327.51	163.36	366.04	404.93	338.02
	TensoRF [3]	-	148.49	152.44	149.99	152.51	151.72	159.80	151.14	148.20	151.79
		LOW	19.26	30.72	19.69	31.29	19.66	21.96	86.23	29.93	32.34
		HIGH	12.97	19.44	13.36	20.17	13.38	15.29	48.47	19.09	20.27
SSIM(↑)	Plenoxel [8]	-	0.830	0.863	0.884	0.857	0.763	0.681	0.937	0.890	0.838
		LOW	0.831	0.862	0.880	0.859	0.758	0.684	0.938	0.895	0.838
		HIGH	0.821	0.858	0.873	0.840	0.734	0.681	0.927	0.889	0.828
	TensoRF [3]	-	0.814	0.871	0.897	0.877	0.752	0.649	0.952	0.900	0.839
		LOW	0.764	0.864	0.891	0.892	0.725	0.570	0.955	0.900	0.820
		HIGH	0.744	0.842	0.880	0.873	0.677	0.520	0.950	0.889	0.797
LPIPS _{VGG} (↓)	Plenoxel [8]	-	0.225	0.177	0.180	0.230	0.194	0.271	0.194	0.237	0.213
		LOW	0.225	0.177	0.183	0.228	0.197	0.264	0.199	0.234	0.213
		HIGH	0.241	0.178	0.188	0.254	0.226	0.265	0.234	0.250	0.229
	TensoRF [3]	-	0.237	0.169	0.148	0.196	0.217	0.278	0.167	0.221	0.204
		LOW	0.299	0.158	0.144	0.158	0.299	0.383	0.149	0.203	0.224
		HIGH	0.337	0.200	0.172	0.193	0.364	0.449	0.168	0.227	0.264

Table 8: Examples generated from the Synthetic-NeRF dataset with TensoRF.

Ground Truth	Baseline	LOW compression	HIGH compression
			
			
			
			

Table 9: Examples generated from the Synthetic-NSVF dataset with TensoRF.

Ground Truth	Baseline	LOW compression	HIGH compression
			
			
			
			
			

Table 10: Examples generated from the Tanks&Temples dataset with TensoRF.

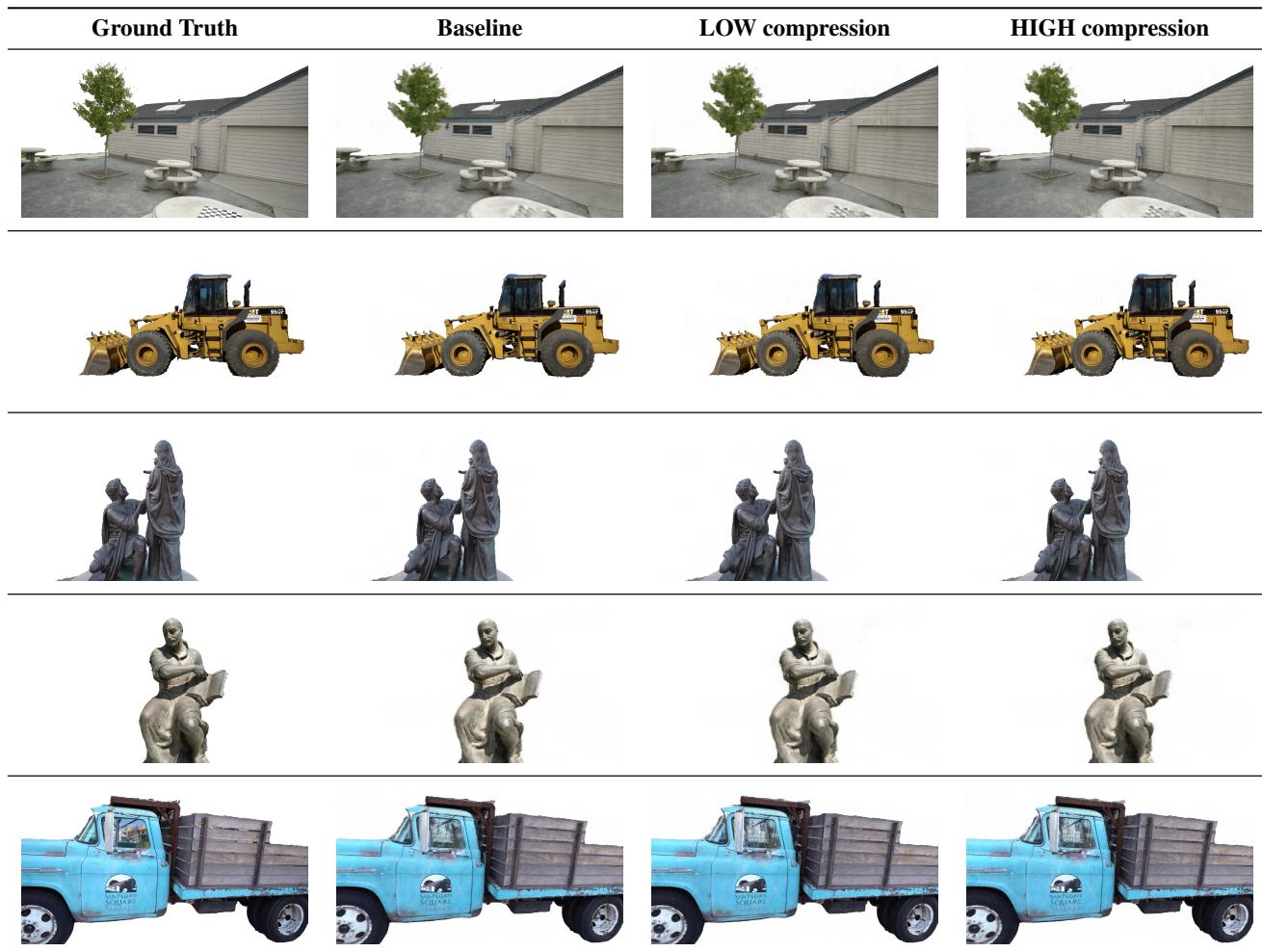


Table 11: Examples generated from the LLFF dataset with TensoRF.

