Compression of PlenOctree Model Attributes Enabling Fast Communication and Rendering of Neural Radiance Fields

Davi R. Freitas
INRIA
Rennes, France
davi-rabbouni.de-carvalho-freitas@inria.fr

Christine Guillemot
INRIA
Rennes, France
christine.guillemot@inria.fr

Ioan Tabus
Tampere University
Tampere, Finland
ioan.tabus@tuni.fi

***Abstract*—Neural Radiance Fields (NeRF) have led the advancement of techniques for 3D scene representations for synthesizing views through the use of a multilayer perceptron. Its inability to achieve real-time performance for photorealistic image rendering, however, has enabled the advent of methods that speed-up rendering time at the cost of rendering quality or model complexity. In this work, we focus on the existing Plenoctree method, which possesses high rendering speed but unfortunately needs a large space for storing and transmitting its model. We address this weakness by proposing improvements to the different blocks of its pipeline and adding an efficient compression stage, without modifying the underlying representation, while maintaining high rendering quality and speed. Results over a set of test camera poses – using our methods which were obtained with data from the training and validation datasets – show that we can reduce about eight times the bit rates of the encoded models and still obtain a higher quality of the synthesized images when comparing them to the original PlenOctree models or, alternatively, a reduction of about 50 times while presenting minimal degradation for novel view synthesis.**

***Index Terms*—Compression, Real-Time Volume Rendering, NeRF, G-PCC, PlenOctrees, Neural Scene Representation**

I. Introduction

Modeling 3D scenes from image data to render novel photorealistic views has been a central topic in the field of computer graphics and vision. The most recent advances have come from the use of neural volumetric representations, such as the volumetric Neural Radiance Fields (NeRF) [1]. NeRF is an implicit scene representation that models a scene as a continuous neural function through the use of multi-layer perceptrons (MLP). This function maps a 5D input – 3D spatial coordinates and 2D viewing directions – into a view-dependent RGB triplet and one value corresponding to the volume density. NeRF's success has led to several extension works targeting its limitations, such as generalization [2], relighting [3], and different imaging input types [4], [5].

In this work, our interest lies in the rendering performance aspect of NeRF. Although the implicit representation of the vanilla method is able to encapsulate the information about the scenes very efficiently from a size point of view, its rendering speed is very slow. Therefore, recent works have attempted to improve the rendering time performance of NeRF [6]–[9]. Nonetheless, these rendering time speed-ups come with a tradeoff either as a decrease in the rendering quality [7], [9], an increase in storage size [6], [9] or speed-ups in rendering that are not as significant [8], [9]. Accordingly, PlenOctrees [6] is, to the best of our knowledge, the state-of-the-art (SOTA) solution in terms of combining rendering quality and speed. Thus, we propose in this paper to improve upon Plenoctree's biggest drawback for real-time applications, which is storage size, by working on the method's pipeline in its entirety.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 956770.

II. NeRF: Key Concepts

Neural Radiance Fields (NeRF) [1] are an implicit scene representation approach through a continuous function F:

$$(\mathbf{c}, \sigma) = F_\Theta(\mathbf{x}, \mathbf{d}), \tag{1}$$

where $\mathbf{x} = (x, y, z)$ are the 3D coordinates of a point and $\mathbf{d} = (\theta, \phi)$ are the viewing direction. In NeRF, F is implemented by two MLPs that predict a color $\mathbf{c} = (r, g, b)$ and a volume density parameter σ required for volumetric rendering.

NeRF uses the $\mathbf{c}, \sigma$ output parameters for a classic volume rendering [10]. A predicted pixel color $\hat{C}(\mathbf{r})$ from the original $C(\mathbf{r})$ is given by accumulating the transmittance T of a ray $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$, where $\mathbf{o}$ is the ray origin that is dependent of the input image camera position,

$$\hat{C}(\mathbf{r}) = \sum_{i=1}^{N} T_i(1 - exp(-\sigma_i\delta_i))\mathbf{c}_i, \tag{2}$$

with $T_i = exp(\sum_{j=1}^{i-1} \sigma_j\delta_j)$ and $\delta_i = t_{i+1} - t_i$ is the step-size.

Then, the MLPs are trained by minimizing the squared error with a rendering loss $\mathcal{L}_r$ between ground truth and predicted pixels from a set of training images with known camera poses,

$$\mathcal{L}_r = \sum_{r\in\mathcal{R}} \| \hat{C}(\mathbf{r}) - C(\mathbf{r}) \|_2^2, \tag{3}$$

where $\mathcal{R}$ is the batch of rays.

From Eq. 2, each ray $\mathbf{r}$ (or pixel) has to be sampled N times, where each one corresponds to a query to the MLP to fetch $\mathbf{c}_i$ and σ_i. Thus, an image of resolution 800×800 would require millions of passes by the MLP, making the process slow.

Therefore, proposals that seek to improve the rendering speed for real-time performance target the acceleration of this procedure. Methods like PlenOctrees [6] and SNeRG [7] achieve this by pre-baking the outputs from NeRF into explicit structures such as sparse voxels, while KiloNeRF [9] does it by breaking the MLP down into multiple smaller ones. DIVeR [8] deviates from NeRF-based approaches by changing the volume rendering integral into a deterministic integrator with the aid of a voxel grid. Overall, PlenOctrees proves to be the SOTA w.r.t. balance between rendering speed and quality, although it has high storage requirements as its biggest hindrance. We look to address this weakness in this work.

III. Proposed Method

Explicit neural representations are able to provide faster rendering speeds compared to their implicit counterparts, for the price of larger storage costs, which is why we look to reduce PlenOctree's storage footprint while keeping its speed.

The full pipeline of the work is shown in Fig. 1 and further detailed in [6], while our proposed changes are explained in the next subsections, divided by the blocks of Fig. 1. The scene is encoded into a modified NeRF to be initially trained, where we implemented changes to provide a higher baseline scene representation. Then, the neural model is densely sampled for the octree parameters extraction: the density σ, directly used in Eq. 2; and the Spherical Harmonics (SH) coefficients $\mathbf{k}$, which are evaluated according to the ray direction (θ, ϕ) to obtain the view-dependent color. After PlenOctree is built, it is then optimized by the rendering loss (Eq. 3) on the training images. Following that, we compress the fine-tuned octree by encoding both the σ and $\mathbf{k}$ as if they were attributes of a point cloud (PC), where the voxels corresponds to the octree leaves.

A. Neural Network Training

To populate the leaves of the octree, PlenOctrees use an altered NeRF, called NeRF-SH [6]. Instead of outputting a view-dependent RGB triplet, this NeRF-SH outputs a set $\mathbf{k}$ of SH of degree ℓ_{max}. SHs are a complete set of functions defined on the surface of a sphere, where the absolute value of each function $Y_\ell^m(\mathbf{d})$ corresponds to the distance of the surface from the origin in the direction $\mathbf{d}$, has degree ℓ and order m. Hence, any function that can be defined on the surface of a sphere can be expressed as a sum of the product between the $Y_\ell^m(\mathbf{d})$ and k_ℓ^m, and Eq. 1 becomes:

$$(\mathbf{k}, \sigma) = F_\Theta^{SH}(\mathbf{x}), \text{where} \quad \mathbf{k} = (k_\ell^m)_{\ell:0<\ell_{max}}^{m:-\ell\leq m\leq \ell}. \tag{4}$$

Since the outputs F^{SH} are independent of the direction, $\mathbf{d}$ is not necessary during the extraction, being required only during inference time. When rendering, the color $c(\mathbf{d}; \mathbf{k})$ in Eq. 2 can then be obtained by the sum over m and ℓ of the products $k_\ell^m Y_\ell^m(\mathbf{d})$, which in the end is normalized to the color range $(0, 1)$ by a sigmoidal function.

Our first modification to NeRF-SH stems from Mip-NeRF [11]. When generating the rays, both during the neural network (NN) stage and the octree optimization block, we add a half-pixel offset to each ray's direction, making them pass through the center of each pixel instead of the corners, as in [1].

Moreover, we also look to reduce the generalization gap between trained and novel views due to the lack of geometry awareness in NeRF's training. With that in mind, we do this by regularizing NeRF's loss function with a total variation term,

$$\mathcal{L}_{\mathcal{TV}} = \frac{\eta}{R} \sum_{i,k} \parallel \nabla\sigma(\mathbf{r}_i(t_k)) \parallel_2, \tag{5}$$

where R is the size of the rays batch $\mathcal{R}$ and η is a hyper-parameter which we empirically found as 10^{-11} for all scenes.

Finally, we also change the sparsity loss from PlenOctrees for the Cauchy loss introduced in SNeRG [7]

$$\mathcal{L}_{\text{cauchy}} = \lambda_s \sum_{i,k} \log(1 + 2\sigma(\mathbf{r}_i(t_k))^2, \tag{6}$$

where λ_s is a hyper-parameter set at 10^{-4} for all experiments (see [7]). We use $\mathcal{L}_{\text{cauchy}}$ to promote sparsity rather than the one in [6] due to how it concentrates opaque voxels around the surfaces, hence creating a more consistent geometry [7].

All in all, the total training loss $\mathcal{L}$ of our network becomes

$$\mathcal{L} = \mathcal{L}_r + \mathcal{L}_{\text{cauchy}} + \mathcal{L}_{TV}. \tag{7}$$

B. Extraction and Optimization

The usage of SHs for the color with NeRF-SH is especially important for the extraction. As the network does not require the viewing direction on the input (Eq. 4), it allows the generation of Plenoctree nodes with just the positional information. This representation gives PlenOctrees the ability to represent the view-dependent appearances of non-Lambertian surfaces.

1) Data Format: The main disadvantage of building a tabulated structure for rendering is the size: PlenOctrees can be up to two orders of magnitude larger than NeRF-like models. We detail in Table I the size distribution for a PlenOctree of the *Lego* scene from the Blender dataset [1].

TABLE I: Breakdown of components and their sizes for a PlenOctree checkpoint. Estimated sizes are given in bytes.

Name	Shape	Format	Estimated Size (B)
Child	$2,982,398 \times 8$	32-bit	91.0×2^{20}
Parent_depth	$2,982,398 \times 2$	32-bit	22.8×2^{20}
Data (σ, $\mathbf{k}$)	$2,982,398 \times 8 \times 49$	16-bit	2.18×2^{30}
Metadata	-	32/64-bit	92
Total	-	-	2.29×2^{30}

Table I shows that the σ and the SHs from the PlenOctree nodes are responsible for about 95% of the 2.3 GB required to store the PlenOctree model. From this aforementioned 95% of data, 98% stems from the 48 coefficients needed to represent the SHs of degree $\ell = 3$. Thus, the capability to represent view-dependent effects, such as specularity and non-Lambertian surfaces, proves to be the biggest bottleneck for a reduced bit rate. *Child* and *Parent_depth*, responsible for most of the remaining 5%, are two data structures for fast traversing the octree. Hence, for the fast renderer procedure to be as it is, they have to be conveyed losslessly in our compression.

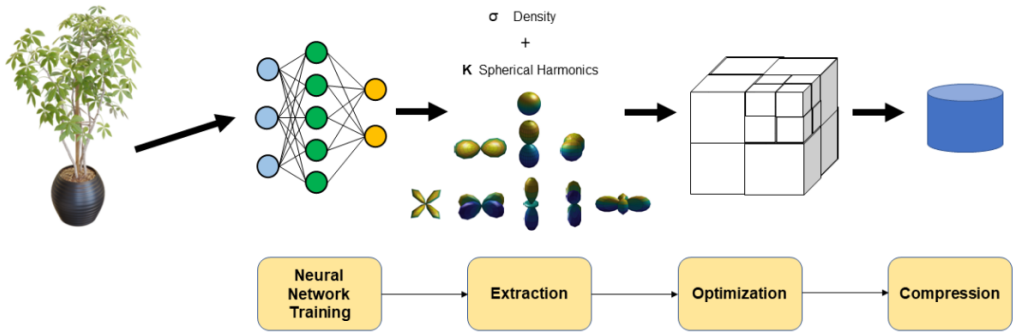


Fig. 1: Schematic for the pipeline of processing stages of the volumetric representation and rendering. A scene is encoded in a NeRF-like network through training; then the trained network is evaluated in to extract the density and SHs into a PlenOctree model; then, this octree is fine-tuned, from where it can be compressed so that it can be conveyed in a lesser bit-rate range.

2) Reducing the Number of Occupied Voxels During Optimization: The optimization in [6] uses the loss in Eq. 3. To reduce the model complexity for further bit-rate savings during compression, we introduce here an additional step of thresholding over the σ after the fine-tuning with cross-validation. This aims to reduce the number of points while minimizing the degradation due to their removal, by taking out the points that have a lesser influence for volume rendering. This can be done by estimating the opacity α for each node,

$$\alpha = 1 - exp(-\sigma\delta), \quad \delta \approx 2/2^N \tag{8}$$

where N is the corresponding octree's depth (grid size of 2^N). After that, the "distilled" PlenOctree is again fine-tuned with cross-validation, which lets the model partially compensate for the knowledge gap caused by the points removal.

C. Compression

PlenOctrees provided attempts to reduce the model's size, such as increasing the σ threshold during extraction, removing the auto-bounding box scaling, and reducing the resolution of the grid [6]. Also, a method for compressing PlenOctree's colors consisted of applying a median-cut algorithm [6] to quantize these coefficients to a color map of 2^{16} colors.

In this work, we aim to compress both the σ and the $\mathbf{k}$ coefficients. Consider the coordinate indices in the grid of the n-th child node of a PlenOctree of depth M to be $\mathbf{x}_{oct}(n) = [X_n, Y_n, Z_n]$. We can view this set of nodes as a depth-M voxelized PC P of N voxels, such that the coordinates representing its geometry $\mathbf{x}_{pc}$ are given by $\mathbf{x}_{pc} = \mathbf{x}_{oct} \times 2^M - 0.5$. The attributes corresponding to each point are then $\mathbf{s}(n) = [\sigma_n, \mathbf{k}^0_{n0}, \mathbf{k}^{-1}_{n1}, \mathbf{k}^0_{n1}, \mathbf{k}^1_{n1}, \cdots, \mathbf{k}^\ell_{n\ell}]$, with $\mathbf{k}^m_{n\ell} \in \mathbb{R}^3$. In other words, P is a voxelized PC where instead of having a single attribute for the color, has multiple attributes corresponding to the parameters needed for PlenOctree's volume rendering. Thus, P indirectly carries view-dependent colors through this set of SH coefficients.

Thus, a natural way to encode P is by using SOTA codecs for PCs, such as the Geometry- and the Video-Based Point Cloud Coding standards [12] (G-PCC and V-PCC, respectively) from the Moving Picture Experts Group (MPEG). Since we aim to keep the geometry $\mathbf{x}_{pc}$ unchanged so it does not affect Plenoctree's structure, the choice of G-PCC [13] is due to its "lossless geometry, lossy attributes" mode to encode P.

Due to P's multiple attributes, we proceed as in the modified G-PCC for multiple attribute coding in [14]. While the work in [14] compressed Plenoptic Point Clouds, the manner with which the color attributes were encoded as coefficients of a Karhunen–Loève Transform (KLT) over the camera space makes it suitable for our work since our SHs in s are also coefficients of an orthogonal basis. However, although it was possible to cascade the SH decomposition with a KLT decorrelation stage, we found that SH decomposition has the channels already decorrelated enough, making KLT unnecessary.

While on one hand the $\mathbf{x}_{pc}$ is losslessly encoded by direct geometry quantization using octrees (further explained in [13]), we encode our attributes s with both predictive coding schemes based on RAHT and Prediction and Lifting transforms (*Predlift* [13]), using quantization stepsizes in the bit-rate range expected from G-PCC's common test conditions.

Each attribute is individually scaled for each attribute color channel to the range of $[0, T]$, with T varying per attribute type. For σ, T is chosen to be 2047 due to the range of α and considering Eq. 8 with an expected grid size of 512. For the SHs, T is selected as 1023 for the DC ($\ell, m = 0$) and 511 the ACs ($\ell \neq 0$), due to the coefficients' ranges. The scaling on the SHs is done over their absolute values, with their corresponding signs being sent as side information.

After the attributes are decoded and unscaled, the PC P can be reconstructed into $\hat{P}$. From the geometry $\mathbf{x}_{pc}$ of $\hat{P}$ that was encoded losslessly, we can create a reconstructed PlenOctree with the same octree structure as the original, but with the lossy reconstructed σ and $\mathbf{k}$, hence fully reconstructing the tree data structure of Table I for the fast renderer of [6].

IV. Experiments

A. Experimental Setup

1) Dataset: We use in our experiments the *NeRF-Synthetic 360* – also known as Blender – dataset from the original NeRF [1]. Here we use the set of scenes {*"chair"*, *"drums"*, *"ficus"*, *"hotdog"*, *"lego"*, *"materials"*, *"mic"*}, where each one contains 100 training, 100 validation and 200 test images of resolution 800×800 with ground truth camera poses.

Although the total training time takes on average 2 days per scene as in [1] due to the NN stage, this step can be sped up by terminating early while producing minimal rendering losses, due to its optimization step with the octree (see [6]).

2) Baselines: We use the original PlenOctrees [6] as the main baseline, since our work uses this method as a base due to its balance between rendering quality and speed. Moreover, we also use as references KiloNeRF [9] and DIVeR [8], which are not as comparably fast, and SNeRG [7].

3) Performance evaluation: The models are evaluated using the 100 training views (with training stoppage based on the 100 validation views). All PSNR values reported in the paper (Fig. 2 and 3, Tab. II-IV) are the averages of the individual image PSNRs according to the respective color spaces as it is commonly done in reference methods such as [1], [6](measuring the quality of the RGB synthesized images compared to ground truth test image) at each of the 200 test poses in each scene, further averaged over the reported scenes. Reported sizes are given in either mebibytes (MiB) or gibibytes (GiB), which correspond to 2^{20} and 2^{30} bytes, respectively.

B. Improved Neural Network Stage

We evaluate the changes to the NN step by performing an ablation study by taking the average PSNR, SSIM, and LPIPS for the testing data over the scenes *lego* and *drums*. Results are summarized in Tab. II.

TABLE II: Quantitative ablation study of our solution by the average of the "lego" and "drums" scenes.

	PSNR ↑	SSIM ↑	LPIPS ↓
NeRF-SH [6]	28.97	0.947	0.064
NeRF-SH + Recenter	29.45	0.951	0.061
Ours - (NN stage only)	29.48	0.951	0.062

The ablation results show a larger contribution of the recentering pixels to the improvement of the model, at least in terms of rendering quality – which is around 0.5 dB for these two scenes. Moreover, the next sections show how our changes affect the rest of the processing pipeline.

C. Octree model extraction with Thresholding

For our experiments with threshold and re-optimization, we choose different values of σ in the set of [0.01, 0.1, 2, 10, 25, 50, 100]. Fig. 2 shows the decrease of the rendered PSNR over the test data for the *lego* scene, due to the reduction of points. Notice how, as the threshold value increases, the greater the effect the re-optimization has. Values of σ of 2 and 10 provide a nice balance between quality loss and reduction of the number of points, which is further shown in Tab. III.

Results over the test data of all scenes from Tab.III suggest that for $\sigma = 2$, the loss of quality is minimal while eliminating $1/4$ of the points from the original model. One notices an insignificant drop of the test synthesis PSNR from the original ($\sigma = 0.01$) to the thresholded at $\sigma = 1$, although the number of points of the model is reduced by more than half on average.

TABLE III: Quality and size comparison of the octree according to different σ values for thresholding with re-optimization. Average from the test data of all scenes.

	PSNR ↑	SSIM ↑	LPIPS ↓	Points (10^7) ↓
W/O Threshold	32.33	0.970	0.039	2.10
σ = 2	32.31	0.970	0.039	1.44
σ = 10	32.11	0.968	0.043	1.02

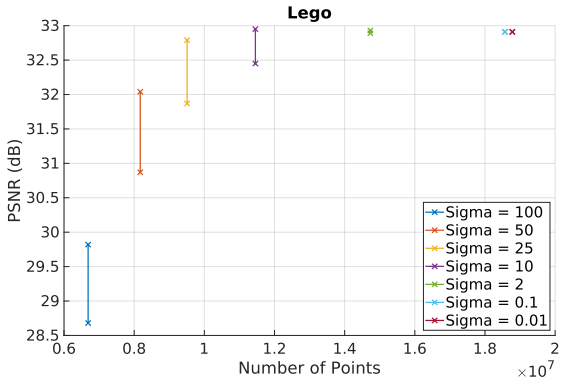


Fig. 2: Results for *lego* in terms of PSNR (in dB) vs number of points (in 10^7) for threshold + optimization stage. Each segment corresponds to a different σ value for the threshold. The bottom points are the PSNR values before the re-optimization procedure, while the points at the top are post-optimization.

D. Compression

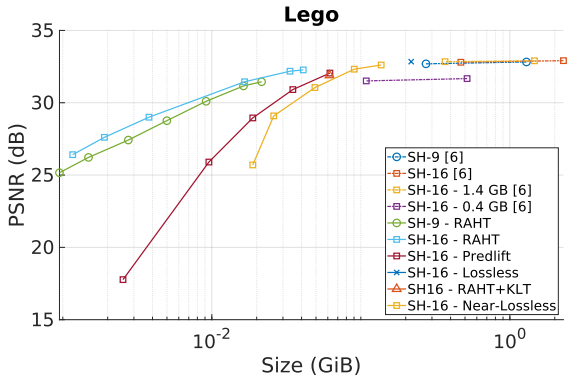


Fig. 3: RD performance results for the *lego* scene comparing PlenOctree's [6] solutions and ours, using G-PCC's attribute coders based on Predlift and RAHT.

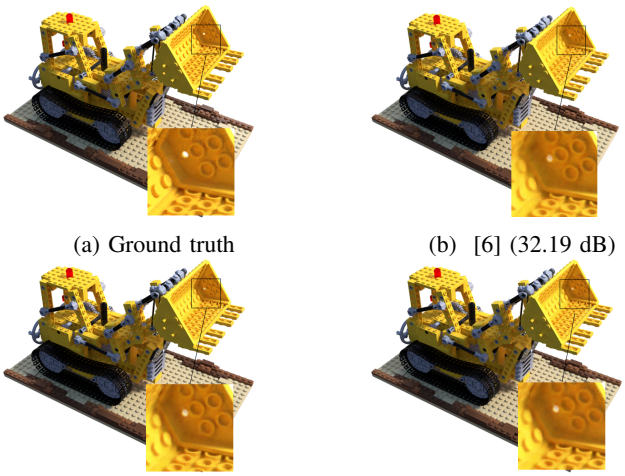
Our compression experiments with G-PCC were done with the modified version proposed in [14], based on version 11.0. Fig. 3 shows the rate-distortion (RD) performance of the baseline (original models from [6]), where the rate is shown in GiB versus the average PSNR over the synthesized views. The RD line for each model is composed of two points, where the rightmost one represents the uncompressed model and the left point is the one compressed as in [6], based on median-cut quantization. "1.4 GB" and "0.4 GB" are proposed in [6] to reduce the model size. A label of "SH-n" means that n SH coefficients are used for each color channel. Our curves for both RAHT- and Predlift-based compression use the standard quantization levels in G-PCC [13] for lossy attribute coding. All our curves encode the geometry losslessly, and these bit rate costs are taken into account. Hence, the "lossy/lossless" terminology here refers to how G-PCC encodes the σ and $\mathbf{k}$.

TABLE IV: Baseline comparisons on Synthetic scenes. "Ours - lossless" is our solution using Predlift-based compression in lossless attributes modes. "Ours - lossy" is our solution compressed in lossy mode with RAHT in its highest bit rate.

	PSNR ↑	SSIM ↑	LPIPS ↓	Size (MiB) ↓
PlenOctrees [6]	32.02	0.968	0.043	1900.50
SNeRG [7]	30.73	0.962	0.035	72.02
KiloNeRF [9]	31.25	0.964	**0.023**	159.29
DIVeR [8]	**32.40**	0.968	0.024	56.00
Ours	32.33	**0.970**	0.038	1748.3
Ours - lossless	32.29	**0.970**	0.039	233.09
Ours - lossy	32.04	0.966	0.047	**39.40**

Results show that our approach outperforms PlenOctree's. Our RAHT-based approach shows to be around three times smaller than the compressed "0.4 GB" curve of the reference while being more than 0.6 dB better in PSNR. Notice also how the RAHT-based method outperforms the Predlift one, which was expected from the performance results from [14] due to the low correlation of the neighbors' attributes. Our compression results between using 9 or 16 SHs per color channel also show the importance of the more refined view-dependence modeling capability of the SH-16 model. Finally, applying KLT over the SH-transformed coefficients ("RAHT+KLT") does not seem to improve the overall performance.

E. Novel View Synthesis



(a) Ground truth (b) [6] (32.19 dB)

(c) Ours (32.49 dB) (d) Ours - lossy (32.09 dB)

Fig. 4: Qualitative results for a test view of *lego* from (b) original Plenoctrees [6], (c) our uncompressed model, (d) our compressed model with RAHT. The hole inside the truck's bucket is better rendered in our model.

At last, we compare our uncompressed solution ("Ours"), compressed in lossless mode ("Ours - lossless"), and with RAHT-based coder at its highest rate ("Ours - lossy") against the SOTA. Results in Tab. IV are computed over the 200 testing views from the seven scenes of the Synthetic 360 [1].

Results support that our uncompressed method improves upon the original PlenOctrees while reducing the average model size. Our lossy compression method provides a size reduction of almost 50 times at the cost of slight performance degradation compared to our uncompressed solution, but still at a similar quality to PlenOcrees, and outperforming SNeRG and KiloNeRF. While our model is outperformed by DIVeR in terms of quality, ours is about 30% smaller. Moreover, PlenOctree's high rendering speed results in our model also being faster by 1.5+ times, as in [8]. Our lossless approach reduces the model's average size by around 8 times. Note that the small degradation in the quality is due to the scaling step required for it to be applied to G-PCC. In Fig. 4, we also show a qualitative comparison of our uncompressed and compressed with G-PCC methods against Plenoctrees. Notice how the hole inside the truck's bucket is more prominent in our uncompressed model in 4-c) than in 4-b).

V. Conclusion

We proposed a method to address PlenOctree's size disadvantages while keeping its rendering quality and speed, by targeting changes in the different stages of its pipeline, specifically the NN training, optimization and compression. Results show that we achieve a size reduction of almost 50 times while producing reduced degradation in terms of rendering quality for novel view synthesis. The metrics suggest that our model is competitive with the SOTA.

References

[1] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, "Nerf: Representing scenes as neural radiance fields for view synthesis," in *Computer Vision – ECCV 2020*, A. Vedaldi, H. Bischof, T. Brox, and J. Frahm, Eds., Cham, 2020, pp. 405–421, Springer International Publishing.

[2] A. Chen, Z. Xu, F. Zhao, X. Zhang, F. Xiang, J. Yu, and H. Su, "Mvsnerf: Fast generalizable radiance field reconstruction from multi-view stereo," in *ICCV*, 2021, pp. 14124–14133.

[3] D. Verbin, P. Hedman, B. Mildenhall, T. Zickler, J.T. Barron, and P.P. Srinivasan, "Ref-NeRF: Structured view-dependent appearance for neural radiance fields," *CVPR*, 2022.

[4] K. Gu, T. Maugey, S. Knorr, and C. Guillemot, "Omni-nerf: neural radiance field from 360° image captures," in *ICME 2022*, Taipei, Taiwan, July 2022, pp. 1–6, IEEE.

[5] T. Otonari, S. Ikehata, and K. Aizawa, "Non-uniform sampling strategies for nerf on 360° images," in *33rd BMVC 2022, London, UK*. 2022, BMVA Press.

[6] A. Yu, R. Li, M. Tancik, H. Li, R. Ng, and A. Kanazawa, "PlenOctrees for real-time rendering of neural radiance fields," in *ICCV*, 2021.

[7] P. Hedman, P.P. Srinivasan, B. Mildenhall, J.T. Barron, and P. Debevec, "Baking neural radiance fields for real-time view synthesis," *ICCV*, 2021.

[8] L. Wu, J.Y. Lee, A. Bhattad, Y. Wang, and D. Forsyth, "DIVeR: Real-time and accurate neural radiance fields with deterministic integration for volume rendering," in *CVPR*, 2022, pp. 16200–16209.

[9] C. Reiser, S. Peng, Y. Liao, and A. Geiger, "Kilonerf: Speeding up neural radiance fields with thousands of tiny mlps," in *CVPR*, 2021, pp. 14335–14345.

[10] J.T. Kajiya and B.P. Von Herzen, "Ray tracing volume densities," *ACM SIGGRAPH computer graphics*, vol. 18, no. 3, pp. 165–174, 1984.

[11] J.T. Barron, B. Mildenhall, M. Tancik, P. Hedman, R. Martin-Brualla, and P.P. Srinivasan, "Mip-nerf: A multiscale representation for anti-aliasing neural radiance fields," *ICCV*, 2021.

[12] D. Graziosi, O. Nakagami, S. Kuma, A. Zaghetto, T. Suzuki, and A. Tabatabai, "An overview of ongoing point cloud compression standardization activities: video-based (V-PCC) and geometry-based (G-PCC)," *APSIPA*, vol. 9, 04 2020.

[13] "G-PCC Codec Description v10," ISO/IEC JTC1/SC29/WG11 MPEG, document N19331, Jun. 2020.

[14] D.R. Freitas, G.L. Sandri, and R.L. De Queiroz, "Geometry-based compression of plenoptic point clouds," in *2022 IEEE 24th MMSP*, 2022, pp. 1–5.