Master thesis report

3D Scene Compression for Autonomous Driving using Neural Radiance Fields

M. F. G. Enting

TU Delft

Master thesis report

3D Scene Compression for Autonomous Driving using Neural Radiance Fields

Thesis report

by

M. F. G. Enting

to obtain the degree of Master of Science in Robotics
at the Delft University of Technology
to be defended publicly on Friday May 3, 2024 at 10:00 AM.

Student number:	4659147	
Project duration:	August 1, 2023 – May 3, 2024	
Thesis committee:	Dr. H. Caeser,	TU Delft, supervisor
	Dr. M. Weinmann,	TU Delft, supervisor
	Dr. P. Kellnhofer,	TU Delft

TU Delft

3D Scene Compression for Autonomous driving using Neural Radiance Fields

Marnix Enting

***Abstract*—Neural Radiance Fields (NeRFs) have showcased remarkable effectiveness in capturing complex 3D scenes and synthesizing novel viewpoints. By inherently capturing the entire scene in a compact representation, they offer a promising avenue for applications such as simulators, where efficient storage of real-world data, fast rendering and dynamic generation of new content are crucial. However, the potential for compression in NeRFs has been largely neglected in the existing literature. Moreover, the practical deployment of NeRFs in real-world scenarios, including simulators, faces significant obstacles such as constraints in training time, rendering speed, and scalability to large scenes. While recent advancements have tackled some of these hurdles individually, none have offered a comprehensive solution. In this paper, we introduce a new NeRF architecture based on a textured polygon-based method and augment this architecture by integrating encodings to expedite training. Additionally, we introduce learned pose refinement and an appearance embedding to enhance scalability to larger scenes. Through experimentation on the nuScenes dataset, we demonstrate that our method achieves competitive reconstruction performance with existing techniques while surpassing them in rendering speed. Furthermore, in terms of compression, our findings indicate that our method achieves competitive compression rates comparable to image-based compression techniques, while also enabling novel-view synthesis. This underscores its potential utility in applications like simulators.**

***Index Terms*—Neural Radiance Fields, 3D Scene Compression, Autonomous Driving, Deep Learning, Driving Simulator**

I. Introduction

For many years, simulators have been core to the development and improvement of autonomous driving models, by offering a safe and cost-effective environment for testing and training. Central to the efficacy of these simulators is the fidelity and realism of the virtual environments they represent. Despite the advancements made, a persistent challenge is the sim-to-real gap, an unavoidable disparity between simulated environments and real-world conditions. To mitigate this, in addition to simulations, models are often trained on real driving data. However, this data typically encompasses only a narrow range of driving scenarios, posing limitations. The substantial storage and bandwidth demands associated with this capturing and storing 3D scene data further underscore the need for effective compression techniques.

Capturing 3D scenes can be done explicitly or implicitly. For explicit representation, the two most common methods for autonomous vehicles include explicitly capturing the scene using a collection of points, i.e. point clouds, or through 2D images with depth. Point clouds consist of a collection of individual points in 3D space, each representing a specific location in the scene. Specifically, they represent 3D data as a set of points with (x, y, z) coordinates, which is referred to as point cloud geometry, and associated attributes, which may include colours, normals, and reflectance. Furthermore, these point clouds can include a temporal dimension. Thus, a distinction can be made between static and dynamic point clouds.

Depth images capture 3D space by representing each pixel's distance from the camera. Compared to point clouds, these images offer a dense and structured representation of the environment, where each pixel contains depth information. While depth images offer a more continuous representation of the scene, point clouds provide a sparse yet precise depiction, allowing for efficient storage and processing of 3D data.

On the other hand, implicit representations offer an alternative approach to encoding 3D scenes. Instead of explicitly storing individual points, implicit representations describe the scene as a continuous function or surface that can be evaluated at any given point. This function provides information about the scene's geometry, appearance, and other relevant properties. Implicit representations have gained attention for their ability to compactly represent complex 3D scenes while enabling efficient storage, transmission, rendering as well as their ability to generate novel viewpoints.

Neural Radiance Fields (NeRF) [27] have emerged as a prominent form of implicit representation that has garnered significant attention in recent times. NeRF captures an entire scene by utilizing a 5D radiance function, which is obtained through the over-fitting of a neural network on a single scene. This implicit representation offers great potential for compressing and rendering 3D scenes, as it encompasses the entire scene and enables the generation of novel viewpoints. In contrast to point clouds, which only capture a subset of the scene, NeRF's continuous structure allows for a more comprehensive representation. However, research into the compression capabilities of NeRF, which hold potential value for training autonomous driving models, remains scarce. In addition, the initial NeRF implementation [27] suffered from limitations such as long training times, high inference, and the inability to handle dynamic 3D scenes. Fortunately, subsequent extensions of NeRF (e.g. Instant-NGP [29], Block-NeRF [35], Dynamic-NeRF [31]) have made substantial progress in addressing these limitations.

Explicit representations, favoured for vehicle-to-vehicle communication, are not ideal in simulator applications due to their limitation of only describing a single driving scenarios without facilitating novel view synthesis. Instead, NeRFs are more suitable for this purpose due to their novel view synthesis. However, for driving simulators, it's crucial that the

method is efficient in terms of training speed, rendering time, and scalability to handle large scenes. These are capabilities that none of the existing methods have fully achieved. Furthermore, there is a notable lack of research on their compression capabilities.

In this context, we propose a novel NeRF method tailored for autonomous driving simulators, inspired by Instant-NGP, MobileNeRF, and Block-NeRF. This method allows for a fast to train, fast to render NeRF that is capable of extending to larger outdoor scenes. Moreover, we analyse the compression capabilities of our model as well as Instant-NGP, MobileNeRF and Nerfacto [36].

Contributions:

- Extension of MobileNeRF to scenes larger than 100m using components from Block-NeRF.
- Capable of rendering large scenes over 50× faster than NeRF methods with standard ray marching rendering pipelines (Instant-NGP).
- Achieve a 10× faster training time than MobileNeRF.
- Analysis of NeRFs compression capabilities, showing 3x improved compression rate to image-based compression when generating intermediate poses.

II. Related Work

A. Compression

This work lies in the field of 3D scene compression, a broad field encompassing various research areas such as point cloud compression, polygon compression, and neural radiance fields. To refine our focus, we focus on methodologies applicable to autonomous driving.

Learned image and video based compression. Image and video compression techniques have been extensively explored in the literature, with numerous standards available such as JPEG, PNG, and TIFF for images, and H.264 and HEVC for videos. The rise of deep convolutional networks has revolutionized image understanding, leading to a growing interest in utilizing such models for deep learning-based image and video compression [2], [3], [25], [38], and [40]. Many of these approaches employ (variational) auto-encoder architectures for compression tasks. These models comprise an encoder-decoder structure, where the encoder compresses data via convolutional layers into a latent vector, which is then decompressed by deconvolutional layers in the decoder. Notably, these methods are able to outperform the compression rates of the standards.

Point cloud compression. Recent advancements in learning-based image compression have influenced the development of point cloud geometry compression significantly. Deep neural networks are increasingly used for this purpose, with various approaches proposed, broadly classified into point-based, voxel-based, and octree-based methods.

Point-based methods, [20], [12] operate directly on input points, enabling better capturing of density information due to the absence of quantization. This makes them particularly suitable for compressing high detail point clouds, such as point clouds involving densely captured objects or human models, where preserving density is crucial for high-quality visual reconstruction. In line with this, D-DPCC [16] is a density-preserving Point Cloud Compression (PCC) approach, integrating density, local position, and ancestor embeddings to effectively compress the data while maintaining density fidelity.

More commonly employed are voxel-based methods [1], [26], [15]. These methods rely on the initial quantization of the input, after which the quantized points are compressed. These methods are often employed for deep entropy based models [30], which require quantization, and allow for the lossy as well as lossless compression of these quantized point clouds. One such methods developed by [21] uses a deep entropy model to losslessly encode the input point cloud.

Octree-based approaches [13] use an octree to represent and compress the quantized point cloud. The Motion Pictures Experts Group (MPEG) has introduced a standard called geometry-based PCC (G-PCC) [28]. Here they use an octree structure and various models to predict features of the finer layer. Octsqueeze [19] introduced a conditional deep entropy model to estimate octree occupancy symbols, while MuSCLE [5] enhances this approach by incorporating temporal priors from previous frames. VoxelContext [32] further extends the ancestor model by integrating spatial information through a voxel context model, which predicts octree occupancies by capturing neighbouring voxel distributions and incorporating temporal information for dynamic point clouds, ultimately achieving superior compression performance. More recently, [48] presented a state-of-the-art deep geometry compression method using autoencoders, featuring a hierarchical sparse-convolution-based framework with multi-layer residual modules and employing multiple IRN blocks akin to [12] to preserve feature information through the entropy encoding stage. Another family of compression methods designed for lidar involves range image compression, which can be beneficial as you can capitalize on lidar scanning patterns. This sparked another family of compression methods involving the compression of these images. For this, image-based compression techniques can be used. Examples include [18], [43] and [41] which apply traditional image compression techniques. RIDDLE [50] employs a deep neural network to predict pixel values in raster scanning order, drawing on contextual information from current and past scans encoded as a 4D point cloud. This approach enables compression of the deltas between predictions and original values through entropy encoding.

Neural Radiance Field compression. Neural radiance fields, introduced by [27], have been primarily used for novel view synthesis. Although not traditionally used for compression, these are closely related. Specifically, these methods capture the entire scene using a sparse subset of explicit images. In this way, you compress the entire 3D scene into an implicit representation. This characteristic makes them very efficient in tasks that require many views, e.g. for simulators. However, the compression aspect of NeRFs has been little explored. One paper that does evaluate the compression capabilities of NeRF is Re:NeRF [10]. This approach effectively prunes model parameters without sacrificing performance by implementing a re-inclusion mechanism. This mechanism enables the reintroduction of previously pruned parameters, located

near remaining parameters, if they exhibit significant gradient loss, thereby maintaining model efficacy.

B. Neural Radiance Fields

In the original NeRF implementation by [27], a 5D neural radiance field is employed, capturing scene characteristics such as volume density $\sigma(x)$ and directional emitted radiance at any point in space. This function signifies the likelihood of a ray intersecting with a particle at position $\mathbf{x}$. Consequently, the expected colour $C(r)$ is computed by:

$$C(r) = \int_{t_n}^{t_f} T(t)\, \sigma(\, r(t)\,)\, c(\, r(t),\, d)\, dt, \tag{1}$$

$$\text{where } T(t) = exp\left(- \int_{t_n}^{t} \sigma(r(s))\, ds \right) \tag{2}$$

Here, $T(t)$ represents the ray's probability of travelling from t_n to t_f without intersecting with any other particle, while $\sigma(\, r(t)\,)$ and $c(\, r(t),\, d)$ denote the volume density and colour functions, respectively, both modelled using an MLP. Rendering an image from this neural radiance field requires estimating the integral of $C(r)$ for each pixel. This implementation faces limitations in accurately representing large or dynamic scenes and lacks suitability for real-time rendering due to slow inference. Moreover, training is computationally expensive, taking over a day for a single scene, and hindering practical application in real-world scenarios. Hence, multiple methods have been introduced to deal with the limitations.

NeRF did not scale well to larger outdoor scenes, such as ones available for autonomous driving. Multiple methods have been introduced to tackle this [33], [42], [47]. Mip-NeRF [4] extends NeRF to represent scenes across a continuous scale by rendering anti-aliased conical frustums instead of rays, effectively reducing aliasing artefacts and enhancing NeRF's ability to capture fine details. Urban-NeRF [33] extends Mip-NeRF [4] by incorporating lidar data for depth supervision, addressing exposure variation with affine mapping, and utilizing image segmentation to capture the sky. These extensions improve street view synthesis performance, but the method is limited to a single city block, unable to capture moving obstacles, and faces challenges with long training and slow inference times. Block-NeRF [35] extends NeRF to encompass multiple city blocks by partitioning cities into individual blocks, each with its neural representation. This method addresses radiance and exposure variations via appearance and exposure embeddings, respectively. The blocks are merged using a visibility model which acts independently of the density and colour MLPs. Despite improved alignment and city-wide representation generation, it struggles with transient objects and computationally expensive block merging, resulting in slow inference and suboptimal real-world performance.

Several approaches have been proposed to mitigate the training time of NeRFs [11], [22]. Instant-NGP [29] accelerates training for various Neural Graphic Primitives by employing hierarchical grids with trainable feature vectors stored in a multi-resolution hash table, facilitating efficient input encoding and interpolation. Despite its limitation to bounded scenes, it achieves accelerated training with minimal quality loss, demonstrating favourable results through experiments with specific parameter settings.

Lastly, several methods have been introduced to improve the inference time of NeRF [14], [22], [17]. Mobile NeRF [9] achieves rendering speeds 10 times faster than previous methods like sNeRG [17] by representing scenes with textured polygons and utilizing traditional rasterization pipelines. However, its reliance on texture maps poses challenges in accurately capturing fine geometric details.

III. Method

Given a set of images and their respective positions, the goal is to devise a concise representation for 3D scene compression. Moreover, we aspire to develop a method capable of adapting to large urban environments, while prioritizing fast rendering and training capabilities. In pursuit of this objective, we employ a polygonal mesh with concentric boxes and texture maps to efficiently encode both features and opacity, similar to MobileNeRF [9]. Our method incorporates two optimizable input encodings similar to Instant-NGP [29]: hash encoding for position and spherical harmonics for direction. These encodings act as feature extractors, thus reducing the complexity required of the neural networks and consequently enhancing training speed. Furthermore, to accommodate larger scenes, we subdivide the large scene into smaller blocks and make three additions to the architecture based on Block-NeRF [35]. These additions are: an appearance embedding to capture variations in weather conditions, a learned camera pose refinement for further alignment, and a decoupled visibility model to combine multiple NeRFs. An overview of the full model can be seen in Figure 1.

Similar to MobileNeRF, the method consists of three distinct training stages, where the final stage is the final mesh and texture generation stage:

- In **Continuous opacity training**, the approach is similar to classical NeRF, where the model is trained on a continuous opacity. The model is trained with a hash encoding, spherical harmonics encoding, camera pose optimization, an appearance embedding and a visibility MLP. Here, the volume is rendered by alpha-compositing the colour at the quadrature points on the mesh. These quadrature points are computed for both the optimizable polygonal mesh and the static concentric boxes around the mesh grid.
- In **Binary opacity training**, the focus shifts to binary opacity modelling. In this phase, the model is adapted to handle discrete opacity, which is required for polygon based rendering. In this stage, we utilize the identical model from stage 1, with the only modifications being the binarization of opacity and the introduction of a new binary RGB loss.
- In **Feature image generation**, the representation is finalized by extracting a sparser polygonal mesh. Opacities and features are integrated into texture maps, and the weights of the small colour and visibility MLPs are stored.

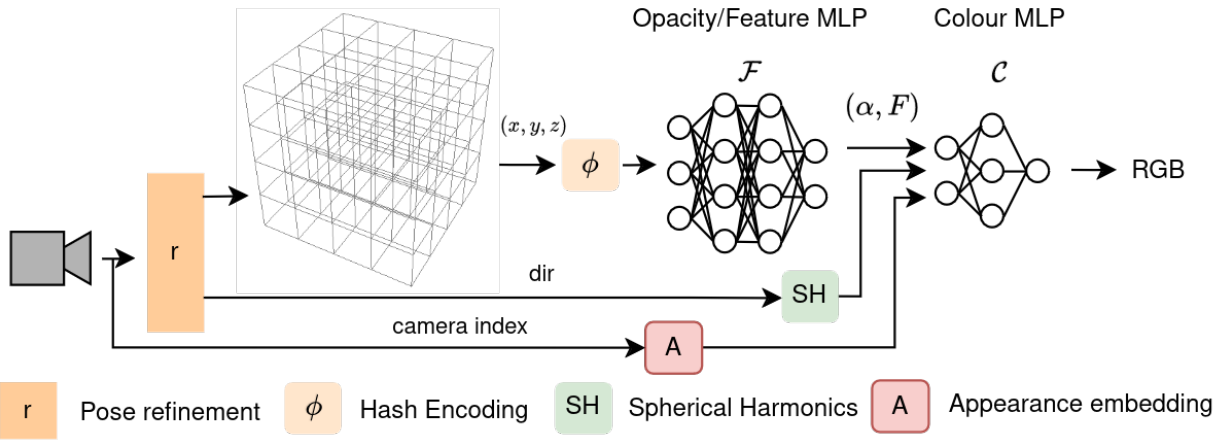


Fig. 1: Overview of the architecture. The architecture comprises a mesh initialized as a regular grid, an optimizable hash grid ϕ, and an MLP $\mathcal{F}$ responsible for representing opacity and features for each point on the mesh. Additionally, a compact colour MLP is employed to derive the final output colour. Furthermore, our model integrates pose refinement (r), spherical harmonics encoding (SH), and appearance embedding (A) to capture weather variations. Given a camera viewpoint, 1) a pose correction is applied using the learned pose refinement. 2) Rays are generated from the camera pose and intersect with the mesh grid to produce points. 3) The generated points are encoded using hash encoding. 4) The encoded points are then fed into $\mathcal{F}$ to compute opacities and features. 5) These computed values are combined with spherical harmonics encoded direction. 6) The appearance embedding outputs are then integrated. 7) Finally, the result is passed through the colour MLP to generate the final RGB values. The architecture remains consistent across both training stages 1 & 2, although in stage 2, opacities are binarized.

A. Position and Direction Encoding

The first major additions to the MobileNeRF architecture are the hash and direction encodings. The hash encoding consists of an optimizable multi-resolution hash table, denoted as $\mathcal{G}$, similar to the approach outlined in Instant-NGP [29]. Concurrently, the direction encoding is implemented through a non-optimizable spherical harmonics encoder [8]. To integrate the hash encoding, we consolidate the Opacity and Feature MLPs from MobileNeRF into an unified MLP, denoted as $\mathcal{F}$. In this configuration, the first output corresponds to opacity, while the subsequent n outputs represent features. Similar to MobileNeRF, we set n to 8 in our experiments. By employing a single MLP, we streamline the framework, requiring only one optimizable hash grid, thereby mitigating potential memory and storage overheads. Moreover, this encoding replaces the traditional position encoder by [27]. The hash table entries are jointly optimized along with $\mathcal{F}$ and the small colour MLP $\mathcal{C}$ in both training stages.

B. Appearance Embedding

MobileNeRF was designed for small scenes with minimal variation in variations in radiance due to weather fluctuations. However, for larger outdoor scenes, these may be present. Hence, to capture these variations in radiance, we incorporate an appearance embedding similar to [24] and [35]. Utilizing a Generative Latent Optimization technique [6], we optimize appearance embedding vectors per image based on the camera. During rendering, there are two options: either employ an average appearance embedding to render the scene with a blend of lighting conditions, or select a specific camera index is used to replicate the lighting conditions experienced by that particular camera.

C. Camera pose optimization

To achieve precise reconstruction and capture fine details, it is imperative to ensure accurate camera pose alignment. This becomes particularly challenging in scenarios like autonomous driving, where camera poses can be inherently noisy. Therefore, we incorporate an extra camera pose refining step similar to [23], [34], [45] and [35]. It encompasses both rotational and translational components and undergoes joint optimization with the NeRF model in both training stages. To stabilize the training and allow the model to learn fine details, we apply a strong regularization term and maintain a comparatively lower learning rate than that of the model. These measures ensure that the pose refinement process minimally affects the learning of initial coarse features at the start of training and, instead, contributes to the refinement of fine-details in the final output.

D. Polygonal mesh

Similar to MobileNeRF, the polygonal mesh is represented as a regular grid G with dimensions P. Here the vertices are optimizable and stored relative to the voxel centres and the grid is situated within a unit cube. To capture geometry outside the unit cube, we use a series of $L+1$ concentric boxes around the regular grid. Unlike the polygonal mesh, these boxes maintain fixed positions and geometries. Further, their distance to the origin are determined by the Equation 3.

$$d_i = \left(\exp\left(\frac{w_i}{L}\right) + w - 1\right)/(2w) \tag{3}$$

Here, i ranges from 0 to L, L is set to 64, and w is chosen such that d_L equals 8, resulting in $d_i \in [0.5, 8]$.

E. Transient objects

While changes in lighting conditions are captured in the model, the model assumes that the objects remain static during training. Hence, a mask is applied to remove dynamic objects from the scene, and these regions are subsequently ignored during training. For the masks, annotated 3D cuboids around the objects are used, as these were present in the dataset. Moreover, masked regions are considered transient when the object has a velocity larger than 0.75 m/s in any frame.

F. Visibility model

For merging multiple NeRFs, a compact visibility MLP $\mathcal{V}$ is added to gauge point visibility, similar to Block-NeRF. This MLP is used to estimate whether a region is visible to a NeRF during training given some camera position and direction. It is trained independently of the $\mathcal{F}$ and $\mathcal{C}$ MLPs. Along each training ray, $\mathcal{V}$ considers the location and view direction, producing estimates for point transmittance (T_k). Transmittance serves as an indicator of point visibility from the input camera: a value of 1 denotes points in free space or on the surface of the initial intersected object, while 0 indicates points situated behind it. Points that are observed from only but not all receive a transmittance between 0 and 1. To merge multiple overlapping NeRFs, the output colour in the overlapped region is determined through a weighted sum of visibility scores from each individual NeRF. This approach diverges from the Block-NeRF method, where the entire scene is merged as a whole. We opt for solely merging the overlapped regions as it simplifies rendering, and the performance gains are minimal due to the sparsity and misalignment of the concentric boxes.

IV. Experiments

This section presents the performance evaluation results for the proposed model, the compression capabilities of the model, as well as relevant ablations. The detailed architectural and optimization specifics are provided in Appendix A.

A. Experimental setup

We conducted our evaluation using the nuScenes [7] and Nerfstudio [36] datasets. Specifically, the model was trained on scenes from the 5 US nuScenes dataset and 3 scenes from the Nerfstudio dataset. The implementation of our model was realized within the Nerfstudio framework [36], an open-source NeRF pipeline. It is worth noting that Nerfstudio prioritizes readability over speed in its design. Hence, to get a fair comparison, the model is evaluated against other methods available within the Nerfstudio environment.

Reconstruction performance. To show a fair comparison between the metrics, the quantitative analysis is performed on the full scene, thus, no block subdivisions are included. Instead, the effect of the block divisions is evaluated in the ablations, see Section IV-D.

Rendering performance. The rendering resolution matches the training images: (1080, 1920) for the small scenes from the Nerfstudio dataset and (1600, 900) for the nuScenes scenes. Further, the hardware used for rendering can be found in Appendix A. Table V.

Training Time. To ensure equitable comparison in training time, all models are assessed within the same framework, Nerfstudio, and utilizing identical hardware, leveraging 8 RTX 3070 GPUs. Evaluation spans varying numbers of training steps, with the optimal step count either determined based on model convergence or the listed number of steps from the original paper. Here, convergence is gauged by monitoring metrics such as the PSNR curve, observing the point where improvements diminish, indicating a plateau in performance. For our model, this corresponds to 60k steps for the training stage 1 and 90k steps for training stage 2.

The methods employed for comparison include Instant-NGP's [29] implementation in Nerfstudio, Nerfacto [36], MobileNeRF [9], and Block-NeRF [35]. Due to computational constraints, values for both Block-NeRF and Mobile-NeRF are sourced from literature rather than tested. The exceptions to this are Mobile-NeRF's training time and storage requirements, which were tested within the Nerfstudio pipeline. Quality assessment across all methods is conducted using PSNR, [44] and LPIPS [49]. The results are summarized in Table I. Further, a comprehensive visual comparison can be found in Appendix D.

B. Comparisons

To show the model's performance, it is tested on various metrics, including reconstruction performance, training time and rendering speed. The results of this comparison are provided in Table I.

Scaling to larger scenes. The model achieves superior performance in comparison to Instant-NGP and Nerfacto, in larger scenes as indicated in Table I, where it achieved notably higher PSNR values, though, lower SSIM and LPIPS scores. Moreover, as depicted in Figure 2 and Appendix D, the model struggled to accurately capture colours compared to the other models, resulting in a predominantly grey hue overlaying the scene. This can be explained by a too small colour MLP, which is unable to capture the variety of features present in the large scene. As can be seen in Appendix D, this is less of an issue for smaller scenes and less complex scenes. Despite producing incorrect colours in the output, our model still manages to attain a higher PSNR, suggesting that it produces less noisy images compared to Instant-NGP and Nerfacto.

While our model demonstrates improvements over Instant-NGP and Nerfacto, it lags behind Block-NeRF in terms of PSNR and LPIPS performance. However, it excels in SSIM scores, showcasing its proficiency in capturing diverse luminance conditions. It is worth noting that the reported results for Block-NeRF were obtained from its original paper, where the method was trained on the San Francisco Bay Area dataset [46], a dataset for NeRFs. This is not a fair comparison, as their dataset consists of 2,818,745 training images in a $1km^2$ block. Comparatively, nuScenes contains approximately 240 images per 100m driving segment. Hence, their model is trained on more images, resulting in better overlapping and consequently better reconstruction performance. However, the

Model	Reconstruction metrics			Rendering speed ↑ [FPS]			Training speed		Storage requirements	
	PSNR ↑	SSIM ↑	LPIPS ↓	Phone	Laptop	PC	Total training time ↓ [hours]	Time per step ↓ [ms]	Storage ↓ [MB]	BPP ↓
Large scenes (75-185m)										
Block NeRF	*23.60	*0.649	*0.417	-	-	*0.16	*9–24	N/A	N/A	N/A
Instant-NGP	17.77	**0.814**	0.454	-	-	0.16	**0.57**	**29**	**298.1**	**0.86**
Nerfacto	17.24	**0.814**	**0.382**	-	-	0.36	1.67	60	613.4	1.77
Ours	17.83	0.652	0.587	**13.82**	**24.45**	**85.38**	9.62	247	367.3	1.06
Small scenes (8-20m)										
MobileNeRF	*21.95	*0.47	*0.47	*9.24	*15.28	***192.59**	73.92	380	**135.7**	**0.29**
Instant-NGP	**22.62**	**0.892**	0.192	-	-	0.62	0.40	36	198.4	0.42
Nerfacto	20.41	0.845	**0.182**	-	-	0.95	**0.37**	**33**	176.1	0.38
Ours	21.77	0.502	0.512	**16.32**	**24.45**	144.90	7.35	189	386.5	0.82

TABLE I: Quantitative analysis, comparing our model with existing methods. Here, the range given corresponds to the size of the scene box. Entries marked with * indicate values directly sourced from the original paper, as these were unable to be run due to computational limitations. It's important to note that the hardware specifications, provided in Appendix B, and datasets for the referenced metrics differ. For Block-NeRF the dataset used is the Alamo Square Dataset [46]. MobileNeRF utilizes their proprietary real 360 unbounded dataset [9]. Note that, both MobileNeRF and Block-NeRF are represented by only one entry each, as they were exclusively trained on small and large scenes, respectively, in their original papers. Finally, 'N/A' denotes missing values from original papers, while '-' indicates the inability to render the scene on a specific device due to memory limitations.

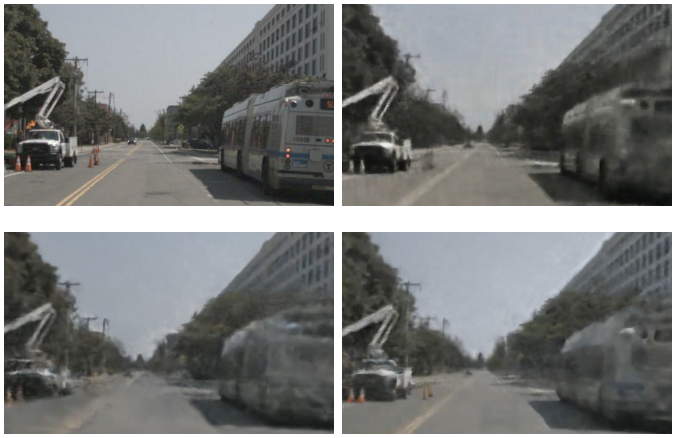


Fig. 2: Visual comparison of the ground truth (top left), our method (top right), Nerfacto (bottom left) and Instant-NGP (bottom right) for scene-0655. These are zoom-ins of the rendered (1600×800) image. As evident, our method demonstrates superior performance in capturing the static background of the scene compared to other methods. However, it encounters difficulties in accurately representing colours resulting in grey scale.

larger area also results in more weather variations, which could explain the lower SSIM scores.

Rendering. As depicted in Table I, our model demonstrates significant performance improvements over Instant-NGP, Nerfacto and Block-NeRF. On CUDA-accelerated hardware, the method is approximately 200 times faster than Nerfacto and 500 times faster than Instant-NGP. Likewise, it is 500 times faster than Block-NeRF which is rendered on a TPU v3 core. Notably, unlike MobileNeRF and our method, these approaches were unable to run on the phone and the laptop due to memory limitations. For smaller scenes, our model achieves performance levels comparable to MobileNeRF. However, it encounters a reduction in rendering performance when handling larger urban scenes, due to an increased number of polygons available in the scene.

Training time. The results in Table I reveal notable speed enhancements achieved with respect to the MobileNeRF model. Namely, the method is approximately 10 times faster to train than MobileNeRF. However, it still falls short when compared to other fast-to-train NeRF variants like Nerfacto and Instant-NGP. It is worth noting that the speed of both our method and MobileNeRF is closely linked to the quantity of rays employed. Specifically, a halving of the number of rays results in an approximately similar reduction in training time. Thus, in our case, the primary computational burden does not stem from querying the model, as is the case with Nerfacto. Instead, it primarily resides in the ray sampling and backward propagation processes due to the large number of operations required for computing the quadrature points.

C. Compression

Table I presents the total storage space and Bit Per Pixel (BPP) of the uncompressed final models. For large scenes, our method outperforms Nerfacto in compression rate, but falls short of Instant-NGP. For the small scene, our method is outperformed by both. Nonetheless, our approach maintains a consistent compression rate regardless of model complexity. Specifically, the final model consists of three components: the mesh grid, the texture images, and the colour MLP. Here, the size of the colour MLP is negligible in comparison to the size of the mesh and texture images. Both the mesh and texture images are heavily influenced by the sparsity of the scene and the grid size utilized. Thus, for the same scene and grid size, the reconstruction performance can be improved through the utilization of a bigger model without affecting the compression rate. In contrast, for Instant-NGP and Nerfacto, storage space is predominantly determined by the size of the hash grid. Consequently, employing a bigger hash grid to improve reconstruction performance, which is

done by both Instant-NGP and Nerfacto for the large scene, leads to increased storage requirements.

Finally, we analyse the effects of various model sizes on their compression performance. Specifically, we test the impact of varying grid sizes, subdividing the scene into blocks, and applying a final lossless compression to the textured mesh. The results of these tests are presented in Table II, covering assessments of both the model architecture and the resulting output mesh. Doubling the training mesh grid size results in a significant increase in storage requirements, with the model's storage needs increasing by approximately 374% and the mesh size by around 241%. Subdividing the scene into blocks leads to a 68% increase in storage requirements for non-overlapping scenes and approximately 14% for overlapping blocks.

Moreover, the initial mesh can undergo further compression using existing techniques. For example, employing lossless compression methods like ZIP yields compression rates ranging from approximately 3.8 to 4.6. By using an additional lossless compression technique, our model achieves compression rates surpassing those of Nerfacto and Instant-NGP. Though, the method is still outperformed by image-based compression to the same reconstruction methods. However, this does not take into account novel-view synthesis. Specifically, for the image-based compression technique, compressed to 20.76 PSNR, our zipped method (1 block and $P = 256$) is able to out-compete image-based compression after 170k newly generated images. Likewise, for the method with a similar LPIPS value, the method can out-compete the image-based compression after 3.3k generated images.

Model	PSNR ↑	LPIPS ↓	Storage ↓ [MB]	BPP ↓
Grid size $P = 128$				
Ours, model	17.34	0.662	257.1	0.61
Ours	17.12	0.664	144.3	0.42
Ours, zip	17.12	0.664	**45.6**	**0.13**
Ours, no overlap	**18.12**	**0.552**	263.7	0.71
Ours, no overlap, zip	**18.12**	**0.552**	81.1	0.23
Grid size $P = 256$				
Ours, model	17.86	0.587	961.7	2.83
Ours	17.68	0.589	348.5	1.00
Ours, zip	17.68	0.589	**69.1**	**0.20**
Ours, no overlap	**18.37**	**0.527**	586.1	1.67
Ours, no overlap, zip	**18.37**	**0.527**	156.9	0.45
Ours, overlap	18.01	0.547	398.2	1.14
Ours, overlap, zip	18.01	0.547	102.4	0.29
Image compression				
Uncompressed (BMP)	-	-	1049.8	3.00
JPEG2000	20	0.702	**0.1**	**1.3e-3**
JPEG2000	25	0.671	0.5	1.8e-3
JPEG2000	30	0.521	5.1	1.5e-2
JPEG2000	35	0.309	22.2	0.06
JPEG2000	**40**	**0.140**	34.7	0.10
JPEG2000	N/A	0.00	43.8	0.13

TABLE II: Comparison of varying the grid size and block size on the compression rate of the final output mesh and model. The values are for the final mesh, unless otherwise specified. For JPEG2000, N/A indicates that the model is losslessly compressed, while - indicates that it is uncompressed.

D. Ablations

In this section, we conduct ablation experiments to analyse the impact of the various model additions as well as the grid size, and the effects of stage 2 on the nuScenes dataset. Notably, we do not include an ablation study on the size of the hash grid. This ablation was omitted due to computational and time constraints. However, for an in-depth exploration of its effects, refer to the investigation provided in Instant-NGP [29]. The results of the ablations can be found in Table III and IV.

	Reconstruction metrics				
	# Blocks	Grid size	PSNR ↑	SSIM ↑	LPIPS ↓
Stage 1	1	128	19.41	0.676	0.553
	1	256	20.22	0.729	0.547
	1, no appearance	256	20.11	0.725	0.552
	1, no pose refinement	256	19.23	0.705	0.574
	2, with overlap	128	20.01	0.773	0.495
	2, with overlap	256	20.79	0.774	0.490
	2, no overlap	128	21.20	0.805	**0.422**
	2, no overlap	256	**21.51**	**0.809**	0.438
Stage 2	1	128	17.34	0.630	0.662
	1	256	17.83	0.652	0.587
	2, with overlap	256	18.01	0.675	0.547
	2, no overlap	128	18.14	0.697	0.551
	2, no overlap	256	**18.41**	**0.703**	**0.527**

TABLE III: Ablation study evaluating the impact of subdividing the scene into two blocks and employing a smaller grid. The scene blocks are tested with either a 50% or no overlap. The visibility model is utilized in computing the final colour for the overlapped area, taking into account the visibility score.

Model	Rendering speed ↑ [FPS]			Training Time	
	Phone	Laptop	Desktop	Total ↓ [hours]	Per step ↓ [ms]
Full scene, $P = 128$	**22.3**	**38.3**	**98.3**	**7.35**	**189**
Full scene, $P = 256$	13.8	24.5	85.4	9.62	247
Blocks, overlap	2.1	9.6	63.9	19.23	247
Blocks, no overlap	3.8	11.1	67.2	19.23	247

TABLE IV: Ablation on the effect of block sizes on training time and rendering speed. Here, the overlapping blocks have a 50% overlap and utilize the visibility model for the overlapped area. Note that the training time is the same for both block sizes, as we, for consistency, maintain the same number of training steps for each case. In practice, the steps can be reduced when fewer images are used.

Grid size. We explore the effects of changing the grid size of the model. Namely, we explore two grid sizes: $P = 128$ and 256. The change in reconstruction performance can be seen in Table III, the changes in rendering in training time can be found in Table IV and a visualization of the improvements can be found in Appendix D. Using a finer mesh grid results in better reconstruction performance while sacrificing on the training speed. Specifically, when utilizing a larger training grid size, the reconstruction metrics are improved by approximately 0.8 dB, 8% and 1% for PSNR, SSIM and LPIPS respectively. Consequently, as is evident from Table IV, the larger grid size causes a $\sim$40% reduction in rendering speed for the phone and laptop while reducing the rendering speed of the Desktop by 13%. Moreover, the training time is increased by $\sim$30%.

Block sizes and overlap Due to computational constraints, we conduct a single subdivision of the space. For the full scene, the scene is normalized such that the absolute position

of any axis is equal to 1. For individual blocks, we divide the space into two along the axis with the greatest variance in camera position, employing both a 50% overlap similar to Block NeRF and no overlap. Following subdivision, we normalize each block into a unit cube, resulting in a slightly contracted rectangular coverage of the full scene. In Block-NeRF, it was found that maintaining the same total number of parameters over all models in the scene yielded only marginal improvements in PSNR. Conversely, when maintaining a consistent model size, they observed significant enhancements in performance. Consequently, we opt for a constant model size over a fixed total number of parameters. The models undergo evaluation using the scene as a whole, with the performance of individual blocks being averaged. Uniform test images, including some within the overlapped region, are employed for this evaluation. As illustrated in Table III and Appendix D, partitioning the scene into blocks yields performance enhancements for both overlapping and non-overlapping blocks. Notably, the initial training stage yields significant performance increases, particularly with the smaller grid size ($P = 128$). Here, non-overlapping NeRFs on average exhibit improvements of 1.79 dB in PSNR, 19% in SSIM, and 24% in LPIPS, while overlapping NeRFs experience gains of 0.60 dB, 14%, and 10%, respectively. Conversely, for the regular grid size ($P = 256$), enhancements in non-overlapping NeRFs are observed at 1.29 dB, 11%, and 20%, respectively, and in the overlapping regions, these are 0.57 dB, 6%, and 11%. However, some of these gains diminish during the binary training phase, resulting in 0.58 dB, 8%, and 10% improvements, respectively, for non-overlapping NeRFs. For overlapping NeRFs, these improvements are slightly lower, at 0.18 dB, 4%, and 7%, respectively.

Effect of stage 2. The model undergoes a dual-stage training process, with the second stage crucially involving the binarizing of the opacity. While essential for the rendering pipeline, this comes at a cost due to additional training constraints being applied to the model. This leads to notable performance decreases, as can be seen from Table III. These performance decreases are larger than in MobileNeRF, where an approximately 1 dB decrease in PSNR and $\sim 1\%$ in SSIM are achieved. In our method, these are approximately 3 dB and over 10% in SSIM and LPIPS.

Appearance embedding. As depicted in Table III, removing the appearance embedding yields only a small decline in reconstruction performance. Notably, the PSNR decreases by 0.11 dB, while the SSIM decreases by approximately 1%, and LPIPS increase by roughly 0.7%. This diminishment is relatively minor and can be attributed to the limited weather variations present in the scene. Specifically, the scene encompasses only about 20 seconds of driving data. Such a brief duration does not encompass substantial weather variations, thereby resulting in minimal performance degradations.

Camera pose refinement. The camera pose refinement shows a more significant impact on performance, as evidenced in Table III. Notably, the performance experiences a decrease of approximately 1 dB, 3%, and 5% for PSNR, SSIM, and LPIPS, respectively. This suggests that the camera positions are inherently noisy, leading to a more pronounced performance drop when this noise is not corrected for.

V. Discussion

In this section, we explore the implications of model performance, assessed across four key dimensions: rendering speed, training speed, scalability to large scenes, and compression. By delving into this analysis, our aim is to illuminate the broader significance of these performance metrics and identify potential avenues for future research and improvement.

Rendering speed On small scenes, the model is able to achieve competitive rendering performance to that of MobileNeRF [9], while significantly improving over Nerfacto [36] and Instant-NGP [29]. This was expected as no alterations were made to the rendering architecture of MobileNeRF and both Instant-NGP and Nerfacto rely on slower ray marching algorithms for rendering. For larger outdoor scenes, the rendering performance dropped. Though, the methods still significantly outperforms Nerfacto, Instant-NGP and Block-NeRF. This performance drop can be explained by the fact that large outdoor scenes use more polygons to capture the space than the small scenes. First, our model uses $P = 256$ instead of $P = 128$, as the grid size doubles the potential number of polygons increases by 8. Moreover, the small scenes only have one central object within the scene, whereas the large city scene has multiple smaller objects scattered in the scene. Thus, more polygons are used to fill in the scene. Lastly, for the scene blocks, the decrease in rendering speed is caused by the increased number of polygons, as two NeRFs are combined as well as the need to query the visibility model for the overlapped region.

Improvements in training speed The method demonstrates a significant improvement in training speed compared to MobileNeRF [9], achieving over a 10x training time improvement. However, the method still suffers from long training times compared to more recent implementations in Nerfstudio [36]. When utilizing the same 8 GPUs, our method remains at least 5x slower than Instant-NGP [29] and Nerfacto [36]. Though, this performance gap increases further when fewer GPUs are employed. For instance, with 2 GPUs, which is how our method is trained, the training speed increased by roughly 4 resulting in a total training time of 38.48 hours. Conversely, both Instant-NGP and Nerfstudio only suffer from an approximately 15% increased training time, as their ray sampling method is parallelized. While our ray sampling, that is computing the quadrature points, is computationally expensive and partially serialized per ray.

Instant-NGP achieved its performance boost by employing smaller neural networks for density and colour estimation. Here, the feature extraction is performed by an optimizable hash-grid. Similarly, our method adopts a comparable hash-map for position encoding, resulting in smaller MLPs. However, despite these optimizations, our method still contends with substantial computational overhead, primarily stemming from complex gradient computations during backpropagation due to the quadrature point computations. Unlike Instant-NGP, our approach does not mitigate this overhead, thus maintaining a high computational load.

Compression capabilities When evaluating compression rates using existing images, the model under-performs relative to image-based compression, achieving worse bits per pixel (BPP) rates for the same PSNR. Though, the comparison is not entirely fair. Specifically, the BPP of NeRF depends on the number of images used for training. The model size is constant irrespective of the images used, hence, more images will lead to higher BPP. Moreover, this comes at the expense of visual reconstruction fidelity. As, direct comparisons based solely on PSNR metrics may not provide a fair assessment of visual quality, especially when comparing against traditional compression methods like JPEG2000 [37].

For instance, when images are compressed to achieve a similar PSNR using JPEG2000, the NeRF representation yields significantly superior subjective visual reconstructions compared to traditionally compressed images. This discrepancy can be attributed to the different types of losses incurred by each compression method. Traditional image compression typically involves losses due to downscaling of the input, leading to loss of detail and fidelity. In contrast, NeRF compression introduces losses primarily through inaccurate upscaling of latent variables or variations in view directions, resulting in differences in colour and texture fidelity. This causes the subjective reconstruction quality of NeRF models to be higher, as can be seen in Figure 3. LPIPS provide a better comparison, though, the type of error still differs. This highlights the importance of considering subjective perceptual quality alongside objective metrics when assessing compression performance. Moreover, the main benefit of using NeRF lies in its novel view synthesis. Specifically, our method facilitates the generation of novel views, enabling it to surpass the capabilities of traditional image compression techniques when a sufficient number of novel views are created. For the method with similar LPIPS, our method outperforms image-based compression when 14 intermediate images are generated between the training images, which corresponds to an FPS of 28. On the PC, our method is able to achieve higher FPS, thus, beating image-based compression when generating intermediate poses. In theory, our method has the potential to generate an infinite number of views. However, in practical applications such as driving simulators, the generated views need to remain relatively close to the training dataset to produce meaningful outputs. Therefore, while the method can generate a vast number of outputs, the actual number of meaningful outputs is limited.

Ultimately, both grid size and block size have a significant effect on the model's storage demands. While adjusting the grid size offers marginal performance gains, it notably increases storage requirements. Conversely, subdividing the scene into blocks considerably enhances reconstruction metrics. Therefore, to strike a balance between achieving optimal reconstruction performance and minimizing storage needs, it proves beneficial to employ block subdivisions and opt for a smaller grid size.

Extension to large scenes. The method is able to achieve superior reconstruction performance in comparison to Instant-NGP and Nerfacto. Namely, our method was able to outcompete these methods in PSNR. However, it fell short in both SSIM and LPIPS. SSIM in particular focuses more on structural similarity, measuring the similarity of local patterns of pixel intensities. Likewise, it takes into account luminance, contrast and structure. LPIPS on the other hand, focuses on perceptual quality by measuring patch similarity between images. This allows it to test both coarse and fine details. Since our method achieves lower on both of these metrics, it indicates that it is not able to capture fine detail or colour, which can be seen from the visual comparisons in Appendix D. The reason for this can be attributed to the multiple factors;

(a) Ground Truth (b) Ours, 18.45 PSNR 0.527 LPIPS

(c) 19.22 PSNR 0.705 LPIPS (d) 24.77 PSNR 0.672 LPIPS

(e) 29.07 PSNR 0.523 LPIPS (f) 34.09 PSNR 0.309

Fig. 3: A visual comparison between the JPEG2000 compressed images compressed to various PSNR (c-f) and our method (b). Here our method is able to achieve a PSNR similar to (c), though, with arguably better visual reconstruction fidelity. Specifically, our method is closer to image (e) in terms of subjective visual reconstruction performance, which has a similar LPIPS value.

First, the scene's sparse object distribution creates vast empty areas, resulting in most of the geometry being captured by the background boxes. These boxes are coarser than the mesh grid, thus capturing only low-detail features. Second, the scene's considerable size results in large polygons within the mesh grid, posing several challenges. Much of the grid in the Z direction remains unused, resulting in inefficient memory allocation and yielding undesired coarseness. Furthermore, objects are captured by few polygons, hindering both training and rendering. For example, for scene-0103, which has a scene box of 105m^2, each grid cell measures approximately 0.41m for the large grid and 0.82m for the small grid. During training, points are sampled based on ray intersections with the deformable grid. When the grid size exceeds the object, only coarse features can be extracted, as slight shifts in the

grid would cause significant point displacements. Similarly, during rendering, texture sizes often exceed object dimensions, leading to artefacts, particularly along texture boundaries when viewed from different perspectives, such as the side. As can be seen in the visual comparison of Appendix D. Similar issues arise with rendering road textures.

Furthermore, concerning the subdivision of the scene, partitioning the scene resulted in significant improvements across all reconstruction metrics. Particularly noteworthy were the substantial gains observed with the smaller grid size and in scenarios with no overlap. This improvement can be attributed to the reduction in scene size. As the scene dimensions decrease, the relative size of objects within the scene increases. This is especially pertinent for the small grid size, where a grid cell size of 0.82m surpasses the dimensions of many objects, leading to texture coarseness. Additionally, by reducing the scene size, a greater portion of the space within the scene box is utilized to capture the scene. Scenes often consist of elongated driving segments, where one axis is significantly longer than the others. Consequently, the driving segment occupies a large portion of the scene, surpassing the width of the street, thereby resulting in substantial empty space within the grid cell. In addition, there is little necessity to overlap the scenes. In Block-NeRF, the rationale behind overlapping was to mitigate artefacts stemming from transitions between NeRFs. However, these artefacts are relatively minor. Instead, the primary visual discrepancies arise from variations in lighting across the scene. Though, this effect can be mitigated by implementing a final appearance matching of the appearance embeddings of the blocks. Lastly, the grey scale is caused by a too small colour MLP. When the scene is smaller, or less complex, the model is able to capture variations in colour. However, for more complex scenes with a lot of fine detail, the model defaults to greyscale to minimize the colour loss. An example of this is visible in Appendix D, where a complex scene solely utilizes grey scale while the city scene with more coarse features is able to capture some variations in colour.

Masking dynamic objects The proposed method handles transient objects by removing them using 3D cuboid masks. These masks can leave behind shadows or cause visual distortions. This is especially apparent at the edge of the scene where views are sparse, as is illustrated in figure 4. The masked region is very noisy, resulting in 'cloudy' artefacts in the final render. Moreover, shadows can be visible in the render, which would require additional masks to remove. Likewise, as the region was covered for most of the observations, the model was never able to learn a proper representation. This becomes a big limitation for crowded scenes with many dynamic objects or scenes such as this one where a large portion of the image is obstructed by a dynamic object. Moreover, as 3D cuboids are used, the masks can be larger than needed as illustrated in Appendix 2, where large areas of the scene are removed through the transient object masks. Thus, applying segmentation masks may improve performance by reducing the number of pixels that are removed.

Memory overhead Another limitation of the model lies in its memory requirements for both training and feature image creation. Particularly during training, the model in total demands approximately 16GB of memory, a demand that, while manageable for high-end GPUs, often exceeds the capacity of many GPUs available on the market. This limitation becomes even more pronounced during feature image creation, where depending on the complexity of the scene, memory needs can increase to over 70GB. This severely limits the hardware that can be used to train the model. As a result, addressing these memory constraints is imperative for broader accessibility and adoption of the model across various hardware configurations.

(a) Ground truth (b) Reconstruction image

Fig. 4: Comparison of the ground truth and reconstructed image. The scene consists in large of a car following scenario, where the image is taken at the boundary of the scene.

Future work In this implementation, rays are generated as single lines traced from the camera centre of projection through each pixel. However, this simplistic approach overlooks the relative footprint of the corresponding image pixel and the length of the interval $[t_{i-1}, t_i]$, leading to aliasing artefacts, especially in large scenes where the pixel footprint is more prominent. An intriguing avenue for future research involves incorporating Mip NeRF [4] frustums, where frustums are employed to better model the pixel footprint in rays.

Moreover, the model was trained on the nuScenes dataset. While this is useful due to the large amount of training data available, the dataset is not tailored for NeRFs. Specifically, the dataset has a sparse distribution of views relative to the scene size, resulting in insufficient overlapping rays. This makes it difficult to capture fine details, and it leads to cloudy areas in regions where transient objects are masked out, posing challenges for accurate reconstruction.

Lastly, analysing the performance of reconstructed images in tasks such as segmentation or object detection presents an exciting avenue for future investigation. This holds particular relevance for applications like driving simulators. Currently, our method is unable to capture dynamic objects, limiting the use case for simulators. Hence, capturing the dynamic objects using an additional NeRF as is done in NeuRAD [39] would be a promising avenue for future work. Similarly, the impact of generating novel views away from the training image poses can be investigated. Success in this area could significantly expand the range of meaningful novel views that can be generated, thereby enhancing the method's utility in simulators.

VI. Conclusion

In this paper, we introduce a new NeRF method that combines MobileNeRF with elements from Instant-NGP and

Block-NeRF. We include additional hash and spherical harmonics encodings from Instant-NGP to speed up training, an appearance embedding, a camera pose refinement, and a visibility model from Block-NeRF to enable block subdivision and capture larger scenes. Our method achieves faster training convergence than MobileNeRF while maintaining high rendering speeds, outperforming Instant-NGP, Block-NeRF, and Nerfacto significantly. It is also capable of handling larger scenes. However, it still falls short compared to Block-NeRF in reconstruction performance.

Furthermore, we have assessed our method's compression capabilities compared to image compression, an area that has received little attention in existing literature. Our findings reveal that our method falls short in compression compared to image compression when no new images are generated. However, the storage requirements of our method remain unaffected by the number of training images, potentially surpassing image compression with a sufficient number of images. Furthermore, it compensates with novel-view synthesis, enabling it to generate any number of novel views. Specifically, our method can out-compete image-based compression when rendering new images at the achieved rendering frames per second (FPS).

In summary, in this work we introduce a method that renders quickly and is scalable to large scenes, offering promising applications for simulators. Our approach provides insights into enhancing computational efficiency and scalability, extending the usefulness of NeRF techniques to scenarios where these factors are critical. However, there is still room for improvement in real-world applications, such as capturing dynamic objects akin to NeuRAD [39]. Additionally, our analysis of the model's compression capabilities sheds light on a previously overlooked aspect of NeRF performance. Further research is needed to evaluate its effectiveness across various tasks like object detection or semantic segmentation.

References

[1] Evangelos Alexiou, Kuan Tung, and Touradj Ebrahimi. "Towards neural network approaches for point cloud compression". In: *Applications of digital image processing XLIII*. Vol. 11510. SPIE. 2020, pp. 18–37.

[2] Johannes Ballé, Valero Laparra, and Eero P Simoncelli. "End-to-end optimized image compression". In: *arXiv preprint arXiv:1611.01704* (2016).

[3] Johannes Ballé et al. "Variational image compression with a scale hyperprior". In: *arXiv preprint arXiv:1802.01436* (2018).

[4] Jonathan T Barron et al. "Mip-nerf: A multiscale representation for anti-aliasing neural radiance fields". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 5855–5864.

[5] Sourav Biswas et al. "Muscle: Multi sweep compression of lidar using deep entropy models". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 22170–22181.

[6] Piotr Bojanowski et al. "Optimizing the latent space of generative networks". In: *arXiv preprint arXiv:1707.05776* (2017).

[7] Holger Caesar et al. "nuscenes: A multimodal dataset for autonomous driving". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 11621–11631.

[8] Anpei Chen et al. "Tensorf: Tensorial radiance fields". In: *European Conference on Computer Vision*. Springer. 2022, pp. 333–350.

[9] Zhiqin Chen et al. "Mobilenerf: Exploiting the polygon rasterization pipeline for efficient neural field rendering on mobile architectures". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2023, pp. 16569–16578.

[10] Chenxi Lola Deng and Enzo Tartaglione. "Compressing explicit voxel grid representations: fast nerfs become also small". In: *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*. 2023, pp. 1236–1245.

[11] Kangle Deng et al. "Depth-supervised nerf: Fewer views and faster training for free". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 12882–12891.

[12] Tingyu Fan et al. "D-dpcc: Deep dynamic point cloud compression via 3d motion prediction". In: *arXiv preprint arXiv:2205.01135* (2022).

[13] Tingyu Fan et al. "Multiscale Latent-Guided Entropy Model for LiDAR Point Cloud Compression". In: *IEEE Transactions on Circuits and Systems for Video Technology* (2023).

[14] Stephan J Garbin et al. "Fastnerf: High-fidelity neural rendering at 200fps". In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2021, pp. 14346–14355.

[15] André FR Guarda, Nuno MM Rodrigues, and Fernando Pereira. "Point cloud coding: Adopting a deep learning-based approach". In: *2019 Picture Coding Symposium (PCS)*. IEEE. 2019, pp. 1–5.

[16] Yun He et al. "Density-preserving Deep Point Cloud Compression". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 2333–2342.

[17] Peter Hedman et al. "Baking neural radiance fields for real-time view synthesis". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 5875–5884.

[18] Hamidreza Houshiar and Andreas Nüchter. "3D point cloud compression using conventional image compression for efficient data transmission". In: *2015 XXV international conference on information, communication and automation technologies (ICAT)*. IEEE. 2015, pp. 1–8.

[19] Lila Huang et al. "Octsqueeze: Octree-structured entropy model for lidar compression". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 1313–1323.

[20] Tianxin Huang and Yong Liu. "3d point cloud geometry compression on deep learning". In: *Proceedings of the 27th ACM international conference on multimedia*. 2019, pp. 890–898.

[21] Dat Thanh Nguyen Andre Kaup. "Lossless Point Cloud Geometry and Attribute Compression Using a Learned Conditional Probability Model". In: *arXiv preprint arXiv:2303.06519* (2023).

[22] Ruilong Li et al. "Nerfacc: Efficient sampling accelerates nerfs". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2023, pp. 18537–18546.

[23] Chen-Hsuan Lin et al. "Barf: Bundle-adjusting neural radiance fields". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 5741–5751.

[24] Ricardo Martin-Brualla et al. "Nerf in the wild: Neural radiance fields for unconstrained photo collections". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 7210–7219.

[25] Fabian Mentzer et al. "Practical full resolution learned lossless image compression". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019, pp. 10629–10638.

[26] Simone Milani. "A syndrome-based autoencoder for point cloud geometry compression". In: *2020 IEEE International Conference on Image Processing (ICIP)*. IEEE. 2020, pp. 2686–2690.

[27] Ben Mildenhall et al. "NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis". In: *ECCV*. 2020.

[28] MPEG. *G-PCC Codec Description*. Tech. rep. Moving Picture Experts Group, Sept. 2021.

[29] Thomas Müller et al. "Instant neural graphics primitives with a multiresolution hash encoding". In: *ACM Transactions on Graphics (ToG)* 41.4 (2022), pp. 1–15.

[30] Dat Thanh Nguyen et al. "Lossless coding of point cloud geometry using a deep generative model". In: *IEEE Transactions on Circuits and Systems for Video Technology* 31.12 (2021), pp. 4617–4629.

[31] Albert Pumarola et al. "D-nerf: Neural radiance fields for dynamic scenes". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 10318–10327.

[32] Zizheng Que, Guo Lu, and Dong Xu. "Voxelcontextnet: An octree based framework for point cloud compression". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 6042–6051.

[33] Konstantinos Rematas et al. "Urban radiance fields". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 12932–12942.

[34] Shih-Yang Su et al. "A-nerf: Articulated neural radiance fields for learning human shape, appearance, and pose". In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 12278–12291.

[35] Matthew Tancik et al. "Block-nerf: Scalable large scene neural view synthesis". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 8248–8258.

[36] Matthew Tancik et al. "Nerfstudio: A modular framework for neural radiance field development". In: *ACM SIGGRAPH 2023 Conference Proceedings*. 2023, pp. 1–12.

[37] David S Taubman and Michael W Marcellin. "JPEG2000: Standard for interactive imaging". In: *Proceedings of the IEEE* 90.8 (2002), pp. 1336–1357.

[38] George Toderici et al. "Full resolution image compression with recurrent neural networks". In: *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*. 2017, pp. 5306–5314.

[39] Adam Tonderski et al. "NeuRAD: Neural rendering for autonomous driving". In: *arXiv preprint arXiv:2311.15260* (2023).

[40] James Townsend et al. "Hilloc: Lossless image compression with hierarchical latent variable models". In: *arXiv preprint arXiv:1912.09953* (2019).

[41] Peter Van Beek. "Image-based compression of lidar sensor data". In: *Electronic Imaging* 31 (2019), pp. 1–7.

[42] Peng Wang et al. "F2-NeRF: Fast Neural Radiance Field Training with Free Camera Trajectories". In: *CVPR* (2023).

[43] Sukai Wang and Ming Liu. "Point cloud compression with range image-based entropy model for autonomous driving". In: *European Conference on Computer Vision*. Springer. 2022, pp. 323–340.

[44] Zhou Wang et al. "Image quality assessment: from error visibility to structural similarity". In: *IEEE transactions on image processing* 13.4 (2004), pp. 600–612.

[45] Zirui Wang et al. "NeRF–: Neural radiance fields without known camera parameters". In: *arXiv preprint arXiv:2102.07064* (2021).

[46] Waymo. *San Francisco Mission Bay dataset*. 2023. URL: %5Curl%7Bhttps://waymo.com/research/block-nerf/%7D.

[47] Yuanbo Xiangli et al. "Bungeenerf: Progressive neural radiance field for extreme multi-scale scene rendering". In: *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXXII*. Springer. 2022, pp. 106–122.

[48] Jiawen Yu et al. "Point Cloud Geometry Compression Based on Multi-Layer Residual Structure". In: *Entropy* 24.11 (2022), p. 1677.

[49] Richard Zhang et al. "The unreasonable effectiveness of deep features as a perceptual metric". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 586–595.

[50] Xuanyu Zhou et al. "Riddle: Lidar data compression with range image deep delta encoding". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 17212–17221.

Appendix A
Architecture and Optimization details

Our model comprises an optimizable hash grid $\mathcal{H}$ and three networks: the feature and opacity network $\mathcal{F}$, the small colour network $\mathcal{C}$, and the visibility network $\mathcal{B}$. Inspired by the findings in Instant-NGP, $\mathcal{H}$ and $\mathcal{F}$ employ a similar architecture, featuring 3 hidden layers with a size of 64 for $\mathcal{F}$. For $\mathcal{H}$, we set the maximum resolution N_{max} to 2048, the hash map size to 2^{19}, and the features per level F to 2. Additionally, we use 4 levels for the spherical harmonics embedding.

For $\mathcal{C}$, we adopt MobileNeRF's implementation, utilizing a compact network composed of 2 hidden layers with a width of 16. Meanwhile, $\mathcal{B}$ consists of 3 layers with a width of 16. Our model also incorporates an appearance embedding of dimension 4 and a pose refinement acting on both position and orientation.

In terms of optimization, we anneal the learning rate from 10^{-2} for both the point and acceleration grids to 10^{-5} using an exponential decay scheduler over 200k iterations. Similarly, for pose refinement, we adjust the learning rate from 6×10^{-4} to 6×10^{-6} over the same 200k steps. Further, we apply a regularization term of 10^{-5} to the point grid. The batch size begins at 2048 total rays, preserving $2 \times P$ points per ray post-sampling. This gradually increases to 4096 at 20k steps and finally to 8192 at 40k steps. At the 40k-step mark, the points retained post-sampling are reduced to P. To enable the model to learn initial geometry more effectively, we defer the initialization of the acceleration grid until after 10k steps. Finally, for feature image generation, we employ a quadrature size of $K = 9$.

Appendix B
Hardware specifications

The hardware specifications used for rendering can be found in Table V. For Block-NeRF, rendering evaluations were conducted on a TPU v3 core, which is roughly equivalent to 5 v100s, and the model is trained utilizing 32 TPU v3 cores. Regarding MobileNeRF, rendering hardware configurations include a Pixel 3 smartphone running Android 12 with an integrated GPU of 9W, a Chromebook with Chrome OS featuring an integrated GPU of 15W, and a desktop PC running Ubuntu 16.04 equipped with an NVIDIA RTX 2080 Ti GPU boasting a power rating of 250W. V.

Type	Device	OS	GPU	Power
Phone	Samsung A53	Android 12	Integrated GPU	7.55W
Laptop	Laptop	Windows 11	Integrated GPU	25W
Desktop	Desktop	Ubuntu 22.04	RTX 3070	220W

TABLE V: The hardware specifications of the devices used. Here, the listed power indicates the maximum GPU capability for a dedicated GPU and the combined CPU and GPU power for integrated GPUs. Here the maximum power setting listed by the manufacturer.

Appendix C
Visualization of masks

Fig. 5: Visual example of images with their respective masks for scenes: 0061, 0103, 0655, 0796 and 0916. The images include at least one example from all six views (these are: front, front left, front right, back, back left and back right). Here the back camera uses a different FOV than the other cameras and includes a mask for the car bumper. Noteworthy is the bottom right image, where only the walking pedestrian is masked.

Appendix D
Visual comparisons

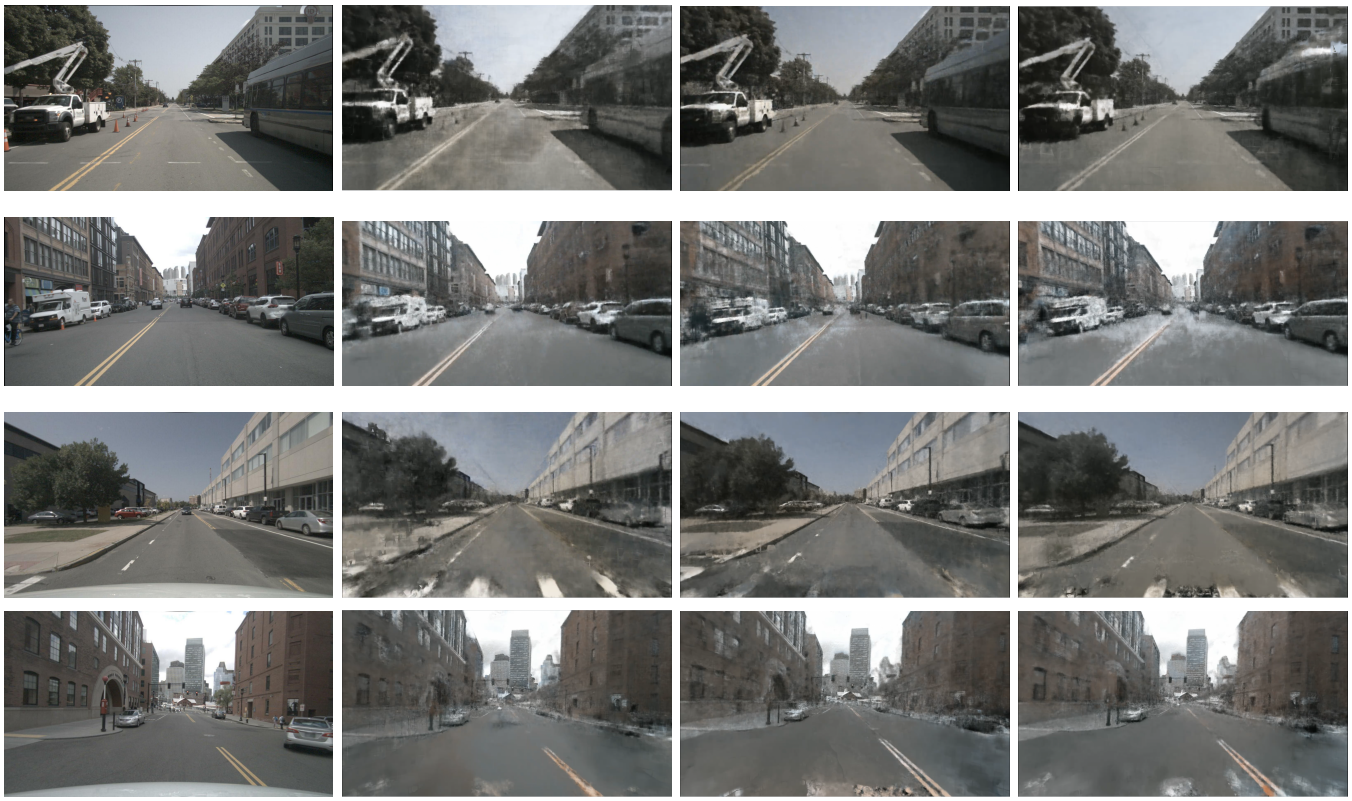
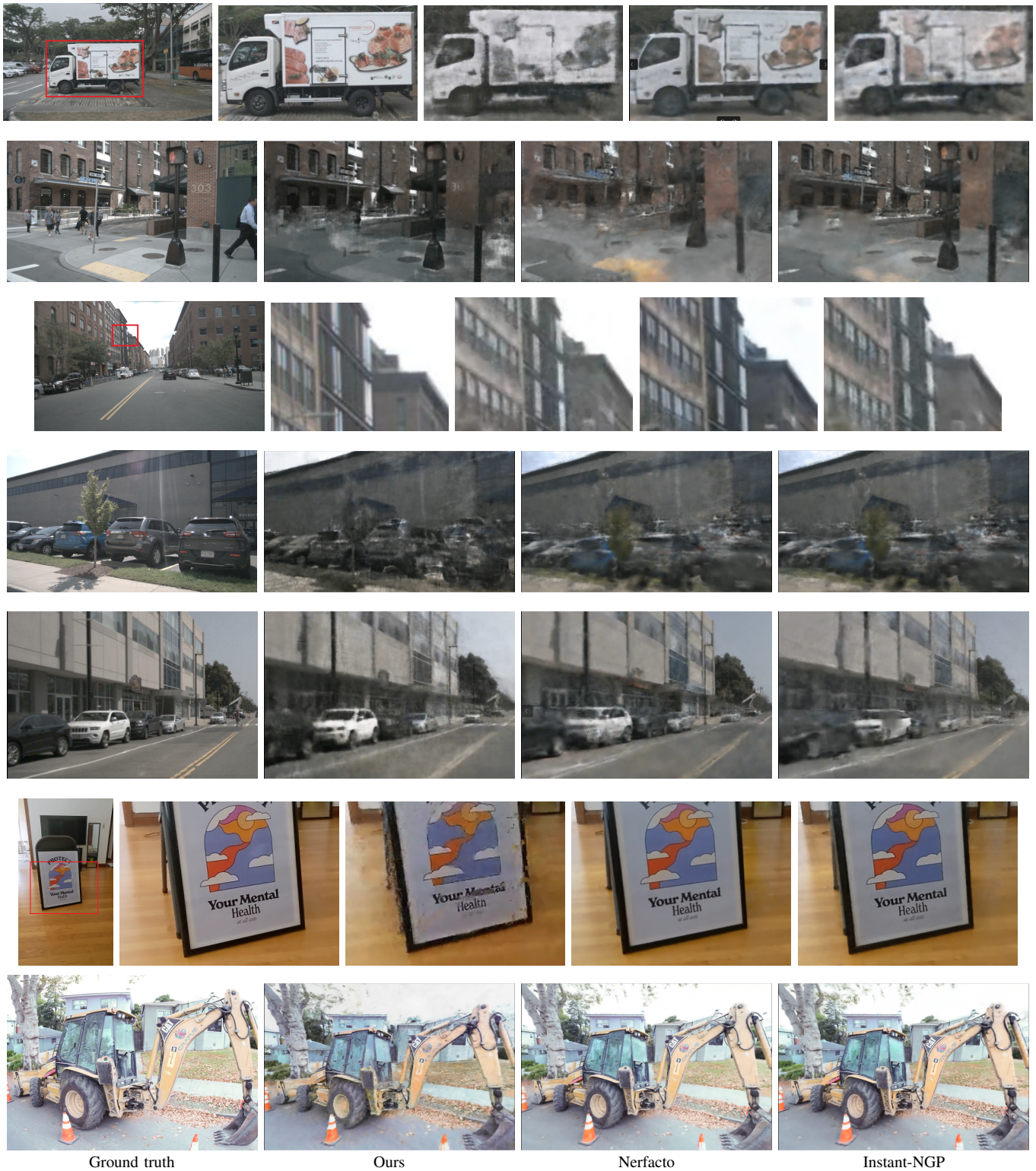


Fig. 6: Comparison of *block subdivision* results of the front and back camera views for scenes 0103 and 0655. The images are rendered at the initial unoptimized camera poses, causing a slight deviation in viewing angles between ground truth and reconstructions. As depicted in the images, the subdivided scenes are better able to capture intricate details, albeit facing pronounced visual distortions at the masked locations. These distortions arise from the limited overlaps and viewpoints within the masked area.



Ground truth Ours Nerfacto Instant-NGP

Fig. 7: Visual comparison between the ground truth, our full scene model, Nerfacto and Instant-NGP on scenes 0103, 0655, and 0916 from nuScenes and two small scenes from Nerfstudio. Here, the bottom two images correspond to the small scene. The comparison encompasses various cameras, featuring both full-rendered (1600x900) images and zoom-ins. In numerous instances, our method excels in capturing the coarse outline of objects or buildings, but faces challenges with colour fidelity and fine details. Additionally, our method exhibits reduced visual distortions in masked areas, such as the image involving pedestrians. Lastly, all renders struggle reconstructions of objects that are sparsely viewed. This is the case for the car park, that is the second image from the bottom, this is the largest scene of 185m and has sparse views on this particular driving segment.