RT-NeRF: Real-Time On-Device Neural Radiance Fields Towards Immersive AR/VR Rendering

Chaojian Li*, Sixu Li*, Yang Zhao*, Wenbo Zhu, and Yingyan Lin
{cl114,sl186,zy34,eiclab,yingyan.lin}@rice.edu
Rice University
Houston, Texas, USA

ABSTRACT

Neural Radiance Field (NeRF) based rendering has attracted growing attention thanks to its state-of-the-art (SOTA) rendering quality and wide applications in Augmented and Virtual Reality (AR/VR). However, immersive real-time (> 30 FPS) NeRF based rendering enabled interactions are still limited due to the low achievable throughput on AR/VR devices. To this end, we first profile SOTA efficient NeRF algorithms on commercial devices and identify two primary causes of the aforementioned inefficiency: (1) the uniform point sampling and (2) the dense accesses and computations of the required embeddings in NeRF. Furthermore, we propose RT-NeRF, which to the best of our knowledge is the first algorithm-hardware co-design acceleration of NeRF. Specifically, **on the algorithm level**, RT-NeRF integrates an efficient rendering pipeline for largely alleviating the inefficiency due to the commonly adopted uniform point sampling method in NeRF by directly computing the geometry of pre-existing points. Additionally, RT-NeRF leverages a coarse-grained view-dependent computing ordering scheme for eliminating the (unnecessary) processing of invisible points. **On the hardware level**, our proposed RT-NeRF accelerator (1) adopts a hybrid encoding scheme to adaptively switch between a bitmap- or coordinate-based sparsity encoding format for NeRF's sparse embeddings, aiming to maximize the storage savings and thus reduce the required DRAM accesses while supporting efficient NeRF decoding; and (2) integrates both a high-density sparse search unit and a dual-purpose bi-direction adder & search tree to coordinate the two aforementioned encoding formats. Extensive experiments on eight datasets consistently validate the effectiveness of RT-NeRF, achieving a large throughput improvement (e.g., 9.7×~3,201×) while maintaining the rendering quality as compared with SOTA efficient NeRF solutions.

ACM Reference Format:
Chaojian Li*, Sixu Li*, Yang Zhao*, Wenbo Zhu, and Yingyan Lin. 2022. RT-NeRF: Real-Time On-Device Neural Radiance Fields Towards Immersive AR/VR Rendering. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD '22), October 30-November 3, 2022, San Diego, CA, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3508352.3549380

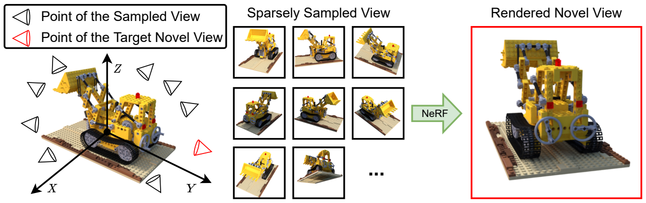


Figure 1: An illustration of novel view synthesis, which is the rendering task that NeRF [17] targets to resolve.

1 INTRODUCTION

Novel view synthesis (see Fig. 1), which renders photorealistic novel views given a set of sparsely sampled views, has become a fundamental task in various AR/VR applications [2, 6, 8, 9, 25], such as virtual meetings [15]. As such, significant efforts have been made to push forward the achievable rendering quality, among which NeRF [17] based rendering has recently attracted a much growing attention thanks to its state-of-the-art (SOTA) rendering quality. However, while immersive real-time (> 30 FPS) NeRF based rendering enabled interactions are highly desired, they are not yet possible due to the low rendering throughput that is currently achievable on AR/VR devices, e.g., < 0.04 FPS for rendering 800×800 images on a SOTA GPU such as NVIDIA V100 GPUs [17].

To close the aforementioned gap, we first perform in-depth profiling of SOTA efficient NeRF algorithms on commercial devices by characterizing the runtime of each step in the algorithm pipeline to identify the bottlenecks causing the rendering inefficiency. In particular, we locate two efficiency-bottleneck steps within the pipeline of SOTA efficient NeRF algorithms: (1) locating pre-existing points, which is to filter out the points representing an empty space and (2) computing pre-existing points' features, which is to generate the features (i.e., densities and colors) of pre-existing points based on the embeddings corresponding to a specific grid. Furthermore, we identify that these two steps' bottleneck inefficiencies are respectively due to (1) the commonly adopted uniform point sampling method despite the sparsities of the existed points and (2) the required dense accesses and computations of the embeddings corresponding to a specific grid, despite the sparsities of those embeddings.

To tackle the identified bottlenecks above, we advocate algorithm-hardware co-design to achieve real-time on-device NeRF processing towards immersive AR/VR rendering and make these contributions:

- We comprehensively profile and analyze the throughput bottlenecks in SOTA efficient NeRF-based methods on multiple commercial devices. We identify that (1) the commonly used uniform point sampling method and (2) the required dense accesses and computations for the embeddings are the primary causes of existing methods' inefficiencies.

*Equal contribution.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
ICCAD '22, October 30-November 3, 2022, San Diego, CA, USA
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-9217-4/22/10...$15.00
https://doi.org/10.1145/3508352.3549380

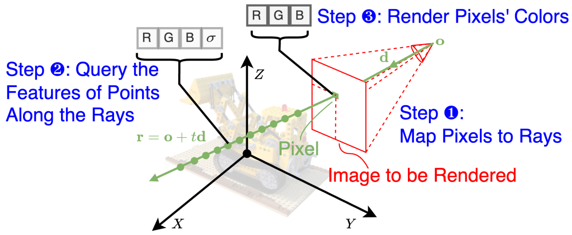

R G B
Step ❸: Render Pixels' Colors
R G B σ
Step ❷: Query the Features of Points Along the Rays
Z
o
d
Step ❶: Map Pixels to Rays
r = o + td
Pixel
Image to be Rendered
X
Y

Figure 2: NeRF [17] based rendering includes Step ❶ Map pixels to rays r = o + *t*d by marching camera rays through the scene, Step ❷ Query the features (i.e., the RGB color and the density σ) of points along the rays by inputting their locations and distance to an MLP model, and Step ❸ Render pixels' colors.

- We propose RT-NeRF, which to the best of our knowledge is the first algorithm-hardware co-design framework for accelerating NeRF. Specifically, our RT-NeRF aims to resolve the aforementioned bottleneck inefficiencies by developing tailored algorithm and hardware innovations that leverage the sparsities of both pre-existing points and specific grids' embeddings. Thus, our RT-NeRF has opened up an exciting perspective towards real-time NeRF solutions.
- On the algorithm level, by leveraging the sparsities of pre-existing points, RT-NeRF integrates an efficient rendering pipeline to alleviate the inefficiency due to commonly adopted uniform point sampling by directly computing the geometry of pre-existing points based on the corresponding non-zero cubes of the occupancy grid. Additionally, in our proposed rendering pipeline, to skip the invisible points from the pre-existing ones for further boosted efficiency, RT-NeRF leverages a coarse-grained view-dependent rendering ordering scheme to avoid processing invisible points.
- On the hardware level, our RT-NeRF accelerator adopts a hybrid encoding scheme to adaptively switch between a bitmap- or coordinate-based sparsity encoding format for NeRF embeddings with low (<80%) and high (≥80%) sparsity-ratios, respectively. Such a hybrid scheme is to maximize the storage savings and thus reduce the required DRAM accesses while supporting efficient decoding, despite the diverse sparsity ratio of NeRF embeddings (e.g., 4% ~ 92%). Furthermore, to avoid the potential computation idleness due to sparse decoding, our RT-NeRF accelerator integrates both a high-density sparse search unit and a dual-purpose bi-direction adder & search tree to coordinate the two aforementioned encoding formats for ensuring efficient sparse decoding.
- Benchmarking experiments and ablation studies on eight datasets of Synthetic-NeRF [17] consistently validate the effectiveness of RT-NeRF, e.g., achieving 9.7× ~ 3,201× throughput improvement while maintaining a similar rendering quality as compared to SOTA efficient NeRF solutions.

2 BACKGROUND AND MOTIVATION

2.1 Preliminaries of NeRF

NeRF for Novel View Synthesis. To render photorealistic novel views as shown in Fig. 1, NeRF [17] encodes a continuous volumetric field of points, which block and emit light rays, within the parameters of a multilayer perceptron (MLP). Fig. 2 illustrates NeRF's rendering process, which involves three steps. **Step ❶** Map pixels to rays: For each pixel to be rendered in the target novel view, a ray $\mathbf{r} = \mathbf{o} + t\mathbf{d}$ is emitted from the origin (e.g., the camera's center) of the target novel view $\mathbf{o}$ along the direction $\mathbf{d}$ to pass through this particular pixel, where t represents the distance between the sampled points along this ray and the origin is denoted as $\mathbf{o}$; **Step ❷** Query the features of points along the rays: For each point that has a distance t_k from $\mathbf{o}$, both its location $\mathbf{o} + t_k\mathbf{d}$ and direction $\mathbf{d}$ are sent as inputs to the MLP model $(\mathbf{o} + t_k\mathbf{d}, \mathbf{d}) \rightarrow (\sigma_k, \mathbf{c}_k)$, which outputs the corresponding density σ_k and an RGB color $\mathbf{c}_k$ as the extracted feature of this particular point; and **Step ❸** Render pixels' colors: Following the principles of classical volume rendering [14], the color $\mathbf{C}(\mathbf{r})$ of the pixel corresponding to the ray $\mathbf{r}$ can be computed by integrating the features of the points along the ray, which can be represented as:

$$\mathbf{C}(\mathbf{r}) = \sum_{k=1}^{N} T_k(1 - \exp(-\sigma_k(t_{k+1} - t_k)))\mathbf{c}_k,$$
$$\text{where } T_k = \exp(-\sum_{j=1}^{k} \sigma_j(t_{j+1} - t_j)), \qquad (1)$$

where N represents the number of sampled points along the ray $\mathbf{r}$ and T_k denotes the accumulated transmittance along the ray $\mathbf{r}$ to the point $\mathbf{o} + t_k\mathbf{d}$, which represents the probability of the ray traveling to the point without hitting any other points. To render an image with an resolution of $H \times W$, the above steps ❶ ~ ❸ will be repeated for $H \times W$ times, corresponding to $H \times W \times N$ number of queries to the MLP model. As such, if using (1) an MLP of 1 million FLOPs for each query to render an image of 800 × 800 resolution and (2) 192 sampled points along each ray [17] during the rendering process, the required total FLOPs would become as large as 800×800×192 ×1 million FLOPs = 117 trillion FLOPs, resulting in < 0.04 FPS on an NVIDIA V100 GPU [17].

To alleviate the prohibitive rendering FLOPs mentioned above, various techniques have been proposed to accelerate NeRF. One popular type of approaches [4, 22, 23] is to add a 3D grid, which represents the embeddings of the specific pre-set points, to be optimized together with the MLP model or even replacing the MLP model. Among them, TensoRF [4] has achieved the SOTA efficiency in terms of accuracy vs. the-number-of-parameters trade-offs, which makes it possible to be further compressed for being deployed on

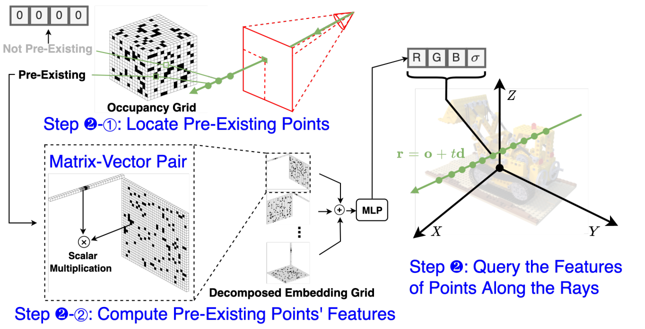

0 0 0 0
Not Pre-Existing
Pre-Existing
Occupancy Grid
Step ❷-①: Locate Pre-Existing Points
Matrix-Vector Pair
Scalar Multiplication
Decomposed Embedding Grid
MLP
R G B σ
Z
r = o + td
X
Y
Step ❷: Query the Features of Points Along the Rays
Step ❷-②: Compute Pre-Existing Points' Features

Figure 3: TensoRF [4] achieving SOTA NeRF efficiency replaces Step ❷ (i.e., query the features of points along the rays using a MLP) in NeRF [17] with both Step ❷-①, which locates pre-existing points using an occupancy grid, and Step ❷-②, which computes pre-existing points' features based on a decomposed embedding grid in terms of matrix-vector pairs.

resource-constrained AR/VR devices, e.g., the popular Oculus Quest 2 VR headset has only < 6GB RAM and < 14 watt-hour battery capacity [16]. It is worth noting that another type of approaches accelerates NeRF by caching a large amount of intermediate results, e.g., > 54 GB in FastNeRF [10], and thus can only be used in high-end GPUs with sufficient resources instead of resource-constrained AR/VR devices that our proposed RT-NeRF mainly targets.

TensoRF with SOTA Efficiency. As visualized in Fig. 3, TensoRF [4], which currently represents SOTA NeRF efficiency, replaces Step ❷ (i.e., query the features of points along the rays using a MLP) in NeRF [17] with the following two steps. **Step ❷-①** Locate pre-existing points: For each point with a distance t_k to the target novel view $\mathbf{o}$, its indices $(x_k, y_k, z_k) \in \mathbb{N}^3$ of a 3D binary occupancy grid can be computed by quantizing its coordinates $\mathbf{o} + t_k\mathbf{d} \in \mathbb{R}^3$. If the corresponding value in the occupancy grid is zero, i.e., this point is in an empty space and thus does not exist, TensoRF returns zero as the corresponding features and skips computing the following steps. **Step ❷-②** Compute pre-existing points' features: For each pre-existing point identified in the previous step, its embeddings are queried from a 3D grid of the specific pre-set points' embeddings, i.e., the embedding grid, based on the aforementioned indices (x_k, y_k, z_k). To save the cost of storing and accessing such an embedding grid, TensoRF [4] further decomposes the embedding grid into multiple matrix-vector pairs. Thus, the corresponding density σ_k can be computed by:

$$\sigma_k = \sum_{r=1}^{R} (\mathbf{v}^X_{r,x_k} \cdot \mathbf{M}^{Y,Z}_{r,(y_k,z_k)} + \mathbf{v}^Y_{r,y_k} \cdot \mathbf{M}^{X,Z}_{r,(x_k,z_k)} + \mathbf{v}^Z_{r,z_k} \cdot \mathbf{M}^{X,Y}_{r,(x_k,y_k)}), \tag{2}$$

where R represents the number of matrix-vector pairs in three decomposition modes, i.e., the 3D embedding grid is decomposed into the outer product of the (1) matrices in the Y, Z plane (i.e., $\mathbf{M}^{Y,Z}$) and vectors along the X axis (i.e., $\mathbf{v}^X$), (2) matrices in the X, Z plane (i.e., $\mathbf{M}^{X,Z}$) and vectors along the Y axis (i.e., $\mathbf{v}^Y$), and (3) matrices in the X, Y plane (i.e., $\mathbf{M}^{X,Y}$) and vectors along the Z axis (i.e., $\mathbf{v}^Z$). Thus, $\mathbf{v}^X_{r,x_k}$ denotes the x_k-th element of the vector along the X axis in the r-th matrix-vector pair, the subscripts of $\mathbf{v}^Y_{r,y_k}$, $\mathbf{v}^Z_{r,z_k}$, $\mathbf{M}^{Y,Z}_{r,(y_k,z_k)}$, $\mathbf{M}^{X,Z}_{r,(x_k,z_k)}$, and $\mathbf{M}^{X,Y}_{r,(x_k,y_k)}$ can be interpreted in the same way. Meanwhile, the corresponding color $\mathbf{c}_k$ (see Eq. 1) can be accessed in the same aforementioned way from another set of matrix-vector pairs. Additionally, a small MLP takes both (1) the concatenated results of the scalar multiplication among different matrix-vector pairs and (2) the direction $\mathbf{d}$ as its inputs to generate the view-dependent color for the next steps. Note that TensoRF [4] also adopts early-ray-termination [13] to filter out invisible points from pre-existing points when computing colors. Specifically, pre-existing points with an accumulated transmittance T_k that is smaller than a pre-set threshold nearly do not contribute to the final rendered color $\mathbf{C}(\mathbf{r})$ as suggested in Eq. 1, and thus can be regarded as invisible and the corresponding computations for generating their colors can be skipped.

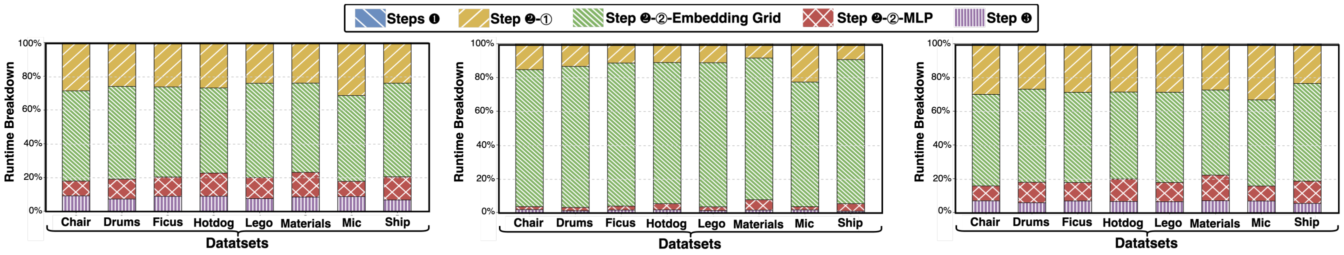
2.2 Profile SOTA Efficient NeRF Solutions

To better understand the throughput bottleneck of SOTA efficient NeRF solutions, we analyze the runtime breakdown of each step within TensoRF [4]'s rendering pipeline on three representative commercial devices, including RTX 2080Ti [18] (a GPU for cloud computing), AMD Threadripper 3970x [1] (a CPU for cloud computing), and Jetson Nano [19] (an embedded GPU for edge computing). As shown in Fig. 4, the profiling results on the eight commonly used datasets of Synthetic-NeRF [17] show that Step ❷-① (i.e., locate the pre-existing points) and Step ❷-② (i.e., compute those pre-existing points' features) dominate the overall rendering latency of TensoRF [4] on all these commercial devices that target both cloud and edge computing.

2.2.1 Analyze Step ❷-①

Identified Causes of Inefficiency. As shown in Fig. 3, to locate pre-existing points, all the candidate points are uniformly sampled along rays and then the existence of pre-existing points are identified via a query process based on the occupancy grid. From this process, we identify two sources of redundant costs: (1) the sparsity of the occupancy grid is not leveraged, and thus the number of queries to the occupancy grid is fixed as $H \times W \times N$ regardless of the values in the occupancy grid, where H and W represent the height and width of the image to be rendered, respectively, and N denotes the number of sampled points along each ray; and (2) the DRAM accesses to the occupancy grid are irregular because the emitted rays can come from any direction, and thus the order of their accesses to the occupancy grid can not be predicted in advance.

Proposed Solution. To eliminate the above redundant computations and memory accesses, we propose an efficient NeRF pipeline, which directly computes the coordinates of pre-existing points by looping over the non-zero cubes of the occupancy grid instead of all the sampled candidate points. The detailed description of our proposed pipeline is provided in Sec. 3.1.



(a) Profiling on RTX 2080Ti (b) Profiling on AMD Ryzen 3970x (c) Profiling on Jetson Nano

Figure 4: Runtime breakdown across eight datasets on three representative commercial devices, which shows that among Step ❶ (i.e., map pixels to rays), Step ❷-① (i.e., locate the pre-existing points), Step ❷-② (i.e., compute pre-existing points' features), and Step ❸ (i.e., render pixels' colors), the SOTA efficient NeRF solution [4] is bottlenecked by Step ❷-① and Step ❷-②, the latter of which includes Step ❷-②-Embedding-Grid and Step ❷-②-MLP that correspond to the operations in Eq. 2 and for the MLP inference, respectively).

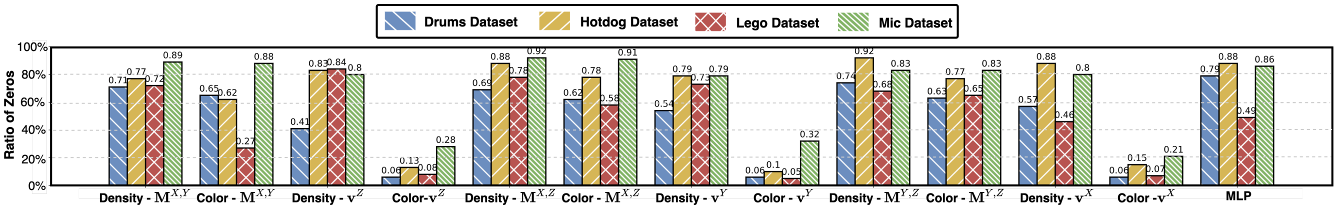


Figure 5: The sparsity of different weights in Eq. 2 on the Drums, Hotdog, Lego, and Mic datasets, where Density - $\mathrm{M}^{X,Y}$ represents the matrices in the X, Y plane for densities and the notations of the other weights can be interpreted in the same way.

2.2.2 *Analyze Step ❷-②*

Identified Cause of Inefficiency. As illustrated in Fig. 4, for Step ❷-② (i.e., compute pre-existing points' features), the required latency of querying the embeddings from the decomposed embedding grid in the format of matrix-vector pairs is much higher (e.g., 4 × ∼ 45 ×) than that of computing the view-dependent colors using the MLP model. We identify that this is because the matrix-vector pairs are treated as dense matrices and/or vectors, causing both redundant computations and DRAM accesses despite their high sparsity (e.g., up to 92% sparsity) as visualized in Fig. 5.

Proposed Solution. Looking into the aforementioned step of computing pre-existing points' features, we find that there consistently exist sparsities in the corresponding weights, and these sparsities feature imbalanced patterns and are dataset-dependent, which however have not been leveraged by SOTA efficient NeRF solutions. As shown in Fig. 5, regarding the aspect of imbalanced sparsity patterns, we can observe that the sparsity ratio of different types of weights can range from 4% to 92%; regarding the aspect of dataset-dependent sparsity, the sparsity ratio of the same type of weights can range from 46% to 88% across different datasets. To utilize the aforementioned sparsities for boosted efficiency, we propose a hybrid encoding scheme for the matrix and/or vectors that adaptively adopts a bitmap- or coordinate-based sparsity encoding format for low (<80%) and high (≥80%) sparsity-ratio scenarios, respectively, aiming to maximize the storage savings and thus reduce the required DRAM accesses. Additionally, we propose a high-density sparse search unit and a dual-purpose bi-direction adder & search tree to coordinate these two encoding formats above for ensuring efficient sparse decoding. The proposed hybrid encoding scheme, sparse search unit, and adder & search tree are introduced in Sec. 4.2.2.

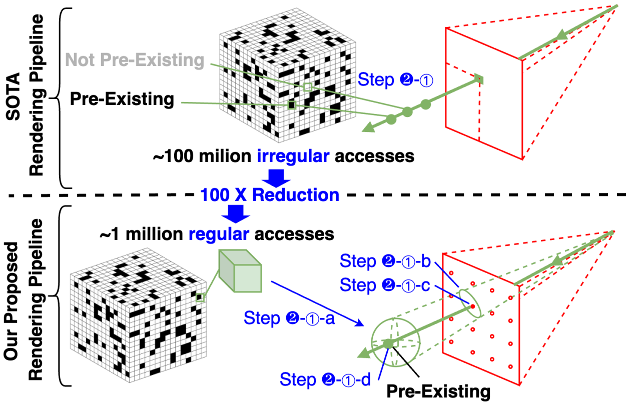


Figure 6: The proposed rendering pipeline which directly computes the geometry of pre-existing points, enabling occupancy grid accesses that are both *fewer* and *more regular* than the SOTA rendering pipeline.

3 RT-NERF: PROPOSED ALGORITHM

3.1 Efficient Rendering Pipeline

To alleviate redundant computations and memory accesses due to commonly adopted uniform point sampling in Step ❷-① (i.e., locate pre-existing points) as analyzed in Sec. 2.2.1, we propose an efficient rendering pipeline, which directly computes the geometry of pre-existing points by looping over only the non-zero cubes of the occupancy grid instead of all the sampled candidate points. As shown in Fig. 6, compared to the SOTA rendering pipeline in [4], the proposed one can reduce the number of accesses to the occupancy grid by 100 × and also makes the corresponding DRAM accesses more regular, i.e., looping over the non-zero cubes in the occupancy grid with a fixed order instead of randomly accessing the grid based on the unpredictable ray directions. Specifically, the proposed rendering pipeline consists of **Step ❷-①-a**: Approximate each non-zero cube in the occupancy grid as a ball for ease of computations in the following steps; **Step ❷-①-b**: Project the aforementioned ball to the image to be rendered as an oval; **Step ❷-①-c**: Identify the points inside the oval based on the regular arrangement of the points in the image to be rendered, i.e., one point corresponds to one pixel; **Step ❷-①-d**: Solve the geometry of the points that are both (1) inside the ball and (2) along the rays passing through the points inside the oval, using an analytic solution of line–sphere intersections [7]. Thus, in the proposed pipeline, only the pre-existing points are included in the loop instead of all the sampled candidate points, resolving the limitations of (1) ignoring the sparsity of the occupancy grid and (2) requiring irregular DRAM accesses in the SOTA rendering pipeline [4] as analyzed in Sec. 2.2.1.

3.2 View-Dependent Rendering Ordering

As illustrated in Sec. 2.1, early-ray-termination [13], which can filter out invisible points from pre-existing points, has been widely

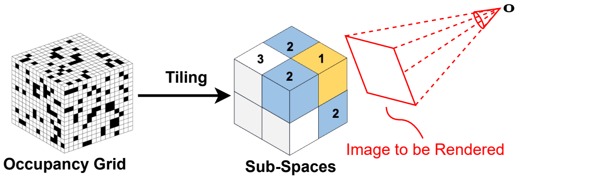


Figure 7: The tiled sub-space (marked as yellow) that is closest to the origin of the target view o will be processed first during our rendering process based on the current target view.

adopted in SOTA efficient NeRF solutions [4, 11, 23]. However, applying early-ray-termination [13] to our proposed efficient rendering pipeline in Sec. 3.1 is not straightforward. This is because a given point's visibility is dependent on the features of the points that are closer to the view origin as suggested in Eq. 1, where a lower accumulated transmittance T_k indicates lower visibility. Thus, there exists redundant computations and data accesses in Step ❷-② (i.e., compute pre-existing points' features) if those invisible but pre-existing points are accessed first while the corresponding accumulated transmittance is still unknown because of the lack of features for points that are closer to the view origin than those invisible points.

To close the gap above, we propose a coarse-grained view-dependent rendering order. Specifically, as shown in Fig. 7, the occupancy grid is first tiled into eight sub-spaces and the non-zero cubes in the sub-space that is closest to the origin of the target view will enter Step ❷-① (i.e., locate the pre-existing points) earlier. In this way, the features of points that are closer to the view origin are calculated first which can help determine the visibility of points that are more distant from the view origin and thus can prevent the processing of invisible points. Additionally, such firstly located pre-existing points will also enter Step ❷-② (i.e., compute pre-existing points' features) earlier. Therefore, only the partial sum of the final rendered color $\mathbf{C}(\mathbf{r})$ in Eq. 1 needs to be stored as the intermediate results during rendering, in contrast to the queried features of all pre-existing points in SOTA solutions. Thus, the proposed coarse-grained view-dependent rendering order not only prevents unnecessary computations and memory accesses for invisible points in the steps of locating pre-existing points and computing the features of pre-exisitng points, but also reduces the memory accesses in Step ❸ (i.e., render pixels' colors) based on Eq. 1, effectively accelerating all the steps that account for > 99 % of the rendering latency as visualized in Fig. 4.

4 RT-NERF: PROPOSED ACCELERATOR

Motivation. As shown in Fig. 8 (a) and (b), when executing our RT-NeRF algorithm (see Sec. 3) on commercial devices, Step ❷-② (i.e., compute pre-existing points' features) now becomes the only throughput bottleneck, which can cost 96% of the total rendering latency. This set of profiling results (1) verify the effectiveness of both our proposed efficient rendering pipeline and view-dependent rendering ordering method described in Sec. 3.1 and Sec. 3.2, respectively, e.g., ↓1.4× rendering latency reduction as compared to TensoRF [4]; and (2) suggest further accelerating the step of computing pre-existing points' features by leveraging the sparsity of the matrix-vector pairs as analyzed in Sec. 2.2.2. To this end, we propose a dedicated RT-NeRF accelerator to take advantage of the sparsity in the matrix-vector pairs to further accelerate the rendering process, e.g., reducing the rendering latency by ↓24× as compared to that of the RTX 2080Ti GPU when running the same algorithm, as visualized in Fig. 8 (c).

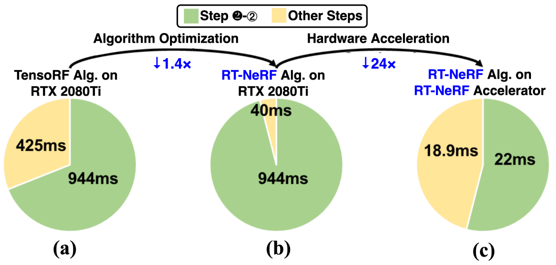


Figure 8: The resulting changes in the runtime breakdown (averaged over eight datasets of Synthetic-NeRF [17]) of Step ❷-② (i.e., compute pre-existing points' features) and other steps, after applying our proposed algorithm optimization and hardware acceleration introduced in Sec. 3 and Sec. 4, respectively.

In this section, we first analyze the design challenges in leveraging the sparsity of the aforementioned matrix-vector pairs in Sec. 4.1, and then present our proposed RT-NeRF accelerator in Sec. 4.2.

4.1 Unique Challenges

Benefiting from the inherent sparsity in the matrix-vector pairs of decomposed embedding grids (see Fig.3), the corresponding computations and memory accesses can be skipped to minimize the rendering latency. However, the sparsity ratios of different matrices and vectors feature *imbalance* patterns and are *dataset-dependent*, e.g., ranging from 4% to 92% among different matrices and vectors even within one dataset as analyzed in Sec. 2.2.2. Such imbalanced and dataset-dependent sparsity makes it difficulty, i.e., **the first challenge**, to efficiently **encode** the sparse matrices and vectors for minimizing both the required storage size and thus DRAM accesses. Furthermore, since decoding the sparse metadata (i.e., information about the indices of non-zero elements) can require several processing cycles on the metadata and thus additional latency, computation idleness may be introduced due to the necessity of waiting for this decoding processing [5]. Therefore, **the second challenge** for the accelerator design is to ensure efficient **decoding** of the sparse metadata to prevent potentially introduced computation idleness.

To tackle the above two challenges, we propose a dedicated accelerator which (1) adopts a hybrid encoding scheme to adaptively switch between a bitmap- and coordinate-based encoding format for NeRF's matrices and vectors of low sparsity-ratio (<80%) and high (>80%) sparsity-ratio, respectively, which is to minimize their memory storage size and thus the required DRAM accesses [5] while supporting efficient decoding; and (2) integrates two efficient decoding&search modules including both a high-density sparse search unit and a dual-purpose bi-direction adder & search tree to implement efficient sparse decoding for both of the two sparse encoding formats above and thus prevent computation idleness commonly observed due to sparse decoding.

4.2 The Proposed Accelerator

In this section, we first provide an overview of the accelerator in Sec. 4.2.1, and then present both the high-density sparse search unit and dual-purpose bi-direction adder & search tree unit, which is to coordinate with the adopted hybrid encoding scheme (see Sec. 4.2.2) to take advantage of the sparsity in NeRF's matrices and vectors mentioned above (see Fig. 5) and address the unique challenges discussed in Sec. 4.1.

4.2.1 Architecture Overview

Fig. 9 (a) shows the overall architecture of our RT-NeRF accelerator, consisting of a memory controller for handling data communication with the off-chip DRAM (Fig. 9 (a) top-right), *N* Serial Processing Units (SPUs) (Fig. 9 (a) top-left), and *N* Parallel Processing Units

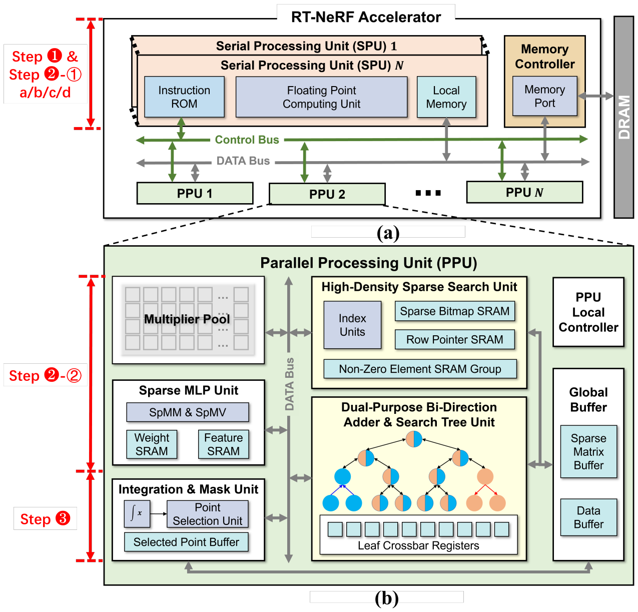


Figure 9: Overall architecture of the proposed accelerator, illustrating the block diagram of both the (a) overall micro-architecture and (b) Parallel Processing Unit.

(PPUs) (Fig. 9 (a) bottom), which are demonstrated in Fig. 9 (b). Specifically, the SPUs are dedicated to Step ❶ and Step ❷-①-a/b/c/d; while Step ❷-② and Step ❸ is processed on the PPUs to leverage the sparsity of the matrix-vector pairs analyzed in Sec. 2.2.2. **The SPU** adopts a RISC-V based processor design, the core of which is a shared floating point computing unit (FPU) for efficiently processing floating point computations, e.g., the analytic solution of line–sphere intersections [7] for solving the geometry of pre-existing points in Step ❷-①-d described in Sec. 3.1. The reason for adopting a shared FPU other than assigning separate FPUs for each step is that a shared FPU is already sufficient thanks to our proposed RT-NeRF algorithm, in which the operations to be deployed on the FPU account for only $< 5\%$ of the total rendering computations.

The PPUs (see Fig. 9 (b)) are to accelerate the remaining $\geq 95\%$ of computations in both Step ❷-② and Step ❸. Specifically, a PPU includes (1) a multiplier pool (top-left in Fig. 9 (b)) for the multiplications in computing pre-existing points' features (i.e., densities and colors), (2) a sparse MLP unit (middle-left in Fig. 9 (b)) for processing the MLP model, (3) an integration & mask unit (bottom-left in Fig. 9 (b)) for computing the final rendered color by integrating the features of the points along the same ray following Eq. 1, (4) a dual-purpose bi-direction adder & search tree unit (bottom-middle in Fig. 9 (b)) for handling the accumulations of computing pre-existing points' features and efficient metadata decoding of matrices/vectors with a high sparsity ($\geq 80\%$), (5) a high-density sparse search unit (top-middle in Fig. 9 (b)) for efficient metadata decoding of matrices/vectors with a low sparsity ($< 80\%$), (6) memories for storing the encoding metadata, elements of the matrix-vector pairs, and the intermediate results, and (7) a local controller.

4.2.2 Hybrid Sparse Encoding Scheme and Efficient Sparse Decoding&Search Modules

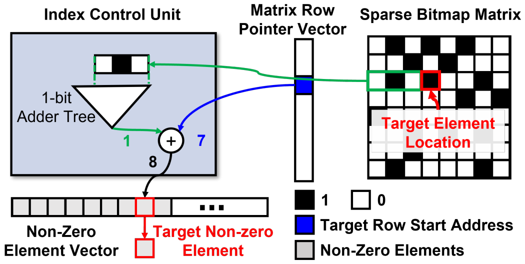


Figure 10: The proposed bitmap-based sparse encoding format and the high-density sparse search unit.

In this subsection, we first (1) introduce the bitmap-based encoding format for the matrices/vectors with a low sparsity ($< 80\%$), which account for 68% of the overall matrices/vectors across eight datasets of Synthetic-NeRF [17], and the high-density sparse search unit for the corresponding efficient decoding; and then (2) demonstrate the coordinate-based encoding format for the matrices/vectors with a high sparsity ($\geq 80\%$), which account for 32% of the overall matrices/vectors across eight datasets of Synthetic-NeRF [17], and the dual-purpose bi-direction adder & search tree unit for efficiently handling the corresponding decoding.

Bitmap-Based Sparse Encoding Format and High-Density Sparse Search Unit. The bitmap encoding [5], which represents the sparsity of each element in the sparse matrices/vectors as 1-bit binary metadata (i.e., 0 for zero elements and 1 for non-zero elements), is a commonly used sparse encoding method for matrices/vectors with a low sparsity. However, directly using bitmap encoding for sparse matrices/vectors can result in varying decoding latencies, which depends on the location of the elements to be decoded, and thus introduce potential computation idleness when a decoding process with a large latency occurs. To tackle this issue of varying decoding latencies, we propose a bitmap-based sparsity encoding format for the matrices/vectors of decomposed embedding grids. As shown in Fig. 10, the proposed bitmap-based sparsity encoding format contains a newly-introduced matrix row pointer vector (middle in Fig. 10), a sparse bitmap matrix (right in Fig. 10), and an vector of non-zero elements (bottom-left in Fig. 10). The newly-introduced matrix row pointer vector stores the addresses of the first non-zero element of each row in the sparse matrix or the start addresses of each row. Thanks to the proposed bitmap-based encoding format, the maximum search latency for any element location in the sparse matrix is fixed, e.g., three cycles in our specific design, eliminating the potential computation idleness for the pipelined decoding processes of multiple elements.

Here is the decoding process of the proposed high-density sparse search unit: Given a target element's location (x, y), Cycle 1: the row x of the sparse bitmap matrix is fetched and the 1-bit encoding metadata of (x, y) is checked. If the 1-bit encoding metadata is zero, the search result is zero; otherwise, the search unit executes the following steps of Cycle 2 and Cycle 3; Cycle 2: the 1-bit encoding metadata of locations $[0, ..., y-1]$ in the pre-fetched row in Cycle 1 is summed up through a binary adder tree and then added with the start address of this row to obtain the address of the target non-zero element in the non-zero element array; Cycle 3: the target non-zero element is fetched via the address calculated in Cycle 2.

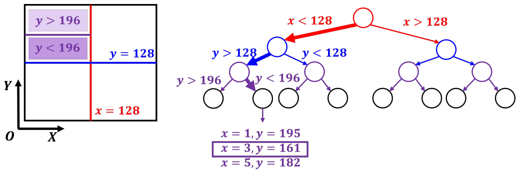


Figure 11: An example of decoding the element at location (x=3, y=161) in the adopted coordinate-based decoding & searching using a binary search tree.

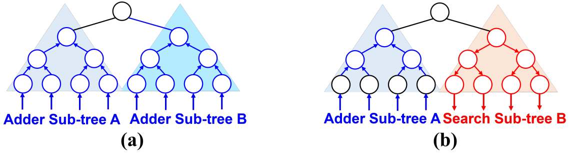


Figure 12: Different reconfigurable modes of the dual-purpose bi-direction adder & search tree unit.

Coordinate-Based Sparse Encoding Format and Dual-Purpose Bi-Direction Adder & Search Tree. Inspired by [5], we adopt a coordinate-based (i.e., COO) sparse encoding format for matrices/vectors with a high sparsity (≥ 80%). In addition, we adopt a binary search tree based decoding method as illustrated in Fig. 11 to better match the corresponding coordinate-based encoding format. Specifically, each node of the search tree will handle one part of the decoding process. For example, the lines in bold in Fig. 11 indicate a search path for $x = 3, y = 161$, where each node is responsible for one comparison (e.g., $x < 128$, $y > 128$, $y < 196$). In the leaf node, there is a one-to-one match crossbar that directly fetches the value of $x = 3, y = 161$.

The binary search tree can suffer from under-utilized in scenarios when most of the processed matrices' sparsity ratio is relatively low (< 80%) and bitmap-based encoding is used. For example, such scenarios are frequently observed since 32%/68% of the overall matrices and vectors are of high/low sparsity and utilize the coordinate-/bitmap-based encoding formats, respectively. To ensure a high hardware utilization, we combine the binary search tree for sparse decoding and the adder tree for computing the features (e.g., densities) of the pre-existing points (as demonstrated in Eq. 2), and propose a dual-purpose bi-direction adder & search tree unit. Specifically, as illustrated in Fig. 12, the proposed dual-purpose bi-direction adder & search tree unit is designed to be reconfigured between (1) an adder tree with multiple adder sub-trees (Fig. 12 (a)) for the low sparsity scenario (< 80%) and (2) a mixed tree with adder sub-trees and search sub-trees (Fig. 12 (b)) for the high sparsity scenario (≥ 80%). Fig. 13 shows the corresponding hardware implementation:

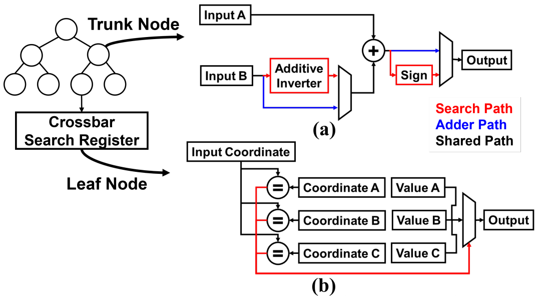


Figure 13: The dual-purpose bi-direction adder & search tree. Each trunk node can be reconfigured between an adder or a comparator by simply turning on/off an additive inverter before the adder or turning off/on a sign detector after the adder, respectively (Fig. 13 (a)). Meanwhile, each leaf node in Fig. 13 (b) is equipped with a crossbar-based search register to decode the target element when being configured as a search tree.

5 EVALUATION

5.1 Experiment Setup

Datasets & Baselines. To evaluate the performance of the proposed RT-NeRF, we conduct experiments on the eight datasets of Synthetic-NeRF [17]. For the evaluation baseline hardware devices, we adopt two categories of devices: edge and cloud devices, and consider a total of five baseline devices, including commercial GPUs and CPUs as well as a dedicated NeRF accelerator. Specifically, for edge devices, we choose NVIDIA Jetson Nano [19] (commonly-used edge GPU) and ICARUS [21] (a dedicated ASIC accelerator for NeRF); For cloud devices, we select NVIDIA V100 [20] (commonly-used cloud GPU with large GPU memory), NVIDIA RTX 2080Ti [18] (commonly-used cloud GPU), and AMD Threadripper 3970X [1] (common-used cloud CPU, 1 core is used in [4]). Table. 1 summarizes the considered devices' specifications.

Note that ICARUS [21] is a concurrent work of RT-NeRF, and proposes an architecture for the vanilla MLP-dominated NeRF-based rendering process. In contrast, our proposed RT-NeRF is built on top of SOTA efficient NeRF algorithms and dedicated to accelerate their unique bottleneck operations other than merely the MLP inference (see Sec. 2). Thus, RT-NeRF achieves 1393× speedup and 5.1× energy efficiency over ICARUS [21] as shown in Sec. 5.3.

RT-NeRF Hardware Implementation. We implement the PPU in the proposed RT-NeRF accelerator with Verilog, and then synthesize, place & route the design based on a commercial 28nm CMOS technology using Cadence Genus & Innovus [3]. Additionally, we

Table 1: A summary of the considered devices' specifications.

	Edge Devices			Cloud Devices			
Device (Algorithm)	**NVIDIA Jetson Nano [19] (TensorRF [4])**	**ICARUS [21] (Original NeRF [17])**	**RT-NeRF-Edge (RT-NeRF)**	**AMD Threadripper 3970x [1] (TensorRF [4])**	**NVIDIA RTX 2080Ti [18] (TensorRF [4])**	**NVIDIA V100 [20] (FastNeRF [10])**	**RT-NeRF-Cloud (RT-NeRF)**
Computing Units	1 (SM) 128 (CUDA)	N/A N/A	1 (Core) 1 (SPU) + 1 (PPU)	32 (Core) 64 (Thread)	68 (SM) 4352 (CUDA)	80 (SM) 5120 (CUDA)	30 (Core) 30 (SPU) + 30 (PPU)
SRAM	2 MB	0.96 MB	3.5 MB	146 MB	29.5 MB	36 MB	105 MB
Area (mm^2)	118	N/A	18.85	296	754	815	565
Frequency	0.9 GHz	0.3GHz	1 GHz	3.7 GHz	1.35 GHz	1.5 GHz	1 GHz
DRAM Bandwidth	LPDDR4-1600 25.6 GB/s	N/A N/A	LPDDR4-1600 17 GB/s	DDR4-3200 95.4 GB/s	GDDR6 616 GB/s	HBM2 900 GB/s	HBM2 510 GB/s
Technology	20nm	40nm	28nm	7nm	12nm	12nm	28nm
Typical Power	10 W	0.3 W	8 W	270 W	250 W	300 W	240 W
Typical FPS	0.01	0.03	**45**	0.5	0.8	200	**1300**

Table 2: Benchmark our proposed RT-NeRF with SOTA efficient NeRF algorithms in terms of the PSNR [12] (higher value represents better rendering quality).

Methods	Avg.	Chair	Drums	Ficus	Hotdog	Lego	Materials	Mic	Ship
NeRF [17]	31.01	33.00	25.01	30.13	36.18	32.54	29.62	32.91	28.65
ICARUS [21]	30.21	33.14	30.38	28.57	-	29.48	-	-	29.48
FastNeRF [10]	29.90	32.32	23.74	27.79	34.72	32.27	28.88	31.76	27.68
TensoRF [4]	32.00	34.68	25.37	32.30	36.30	35.42	29.30	33.21	29.46
RT-NeRF (Ours)	31.79	34.52	25.05	32.11	36.02	35.21	29.10	33.01	29.27

develop a cycle-accurate simulator to simulate the performance of our PPU, for which the unit computation cost is derived from the post-layout simulation. After that, we verified the simulator against the Verilog implementation to ensure its correctness. The SPU in our RT-NeRF accelerator is simulated using a modified simulator based on SonicBOOM [24] which is an open source out-of-order RISCV core. For a fair comparison with both the edge and cloud baseline devices, we configure two RT-NeRF hardware settings accordingly: one corresponds to the edge device (denoted as RT-NeRF-Edge) and the other corresponds to the cloud device (denoted as RT-NeRF-Cloud). Specifically, for RT-NeRF-edge, we set the corresponding hardware configuration such that RT-NeRF-edge can achieve > 30FPS throughput requirement for all eight datasets of Synthetic-NeRF [17], which results in an average of 45FPS over the eight datasets. For RT-NeRF-Cloud, we configure the hardware resources to match the power of a RTX 2080Ti [18] GPU. The total area of our RT-NeRF-Edge and RT-NeRF-Cloud accelerators are 18.85mm^2 and 565mm^2, respectively. The detailed hardware configurations are summarized in Tab. 1.

5.2 Algorithm Evaluation

To evaluate the effectiveness of our RT-NeRF algorithm in Sec. 3, in addition to the changes of the runtime breakdown before and after applying our proposed algorithm on 2080 Ti [18] in Fig. 8, we benchmark it with SOTA efficient NeRF algorithms in terms of achieved PSNR [12] (a higher value indicates better rendering quality) on eight datasets of Synthetic-NeRF [17] in Tab. 2. We can see that (1) our RT-NeRF algorithm surpasses all the SOTA efficient NeRF algorithms except TensoRF [4] in terms of the rendering quality (i.e., ↑0.78 ~ ↑1.89 PSNR, averaged over the eight datasets), while achieving a higher energy efficiency (i.e., ↑5.1× ~ ↑4002×, averaged over the eight datasets) as suggested in Fig. 14; (2) Compared with TensoRF [4], the average PSNR achieved by our proposed algorithm is only ↓0.21 than TensoRF [4], which is caused by approximating the cubes as a ball in Step ❷-①-a, however, our RT-NeRF algorithm can reduce the rendering latency by ↓1.4× over TensoRF [4] on commercial devices as suggested in Fig. 8.

5.3 Hardware Evaluation

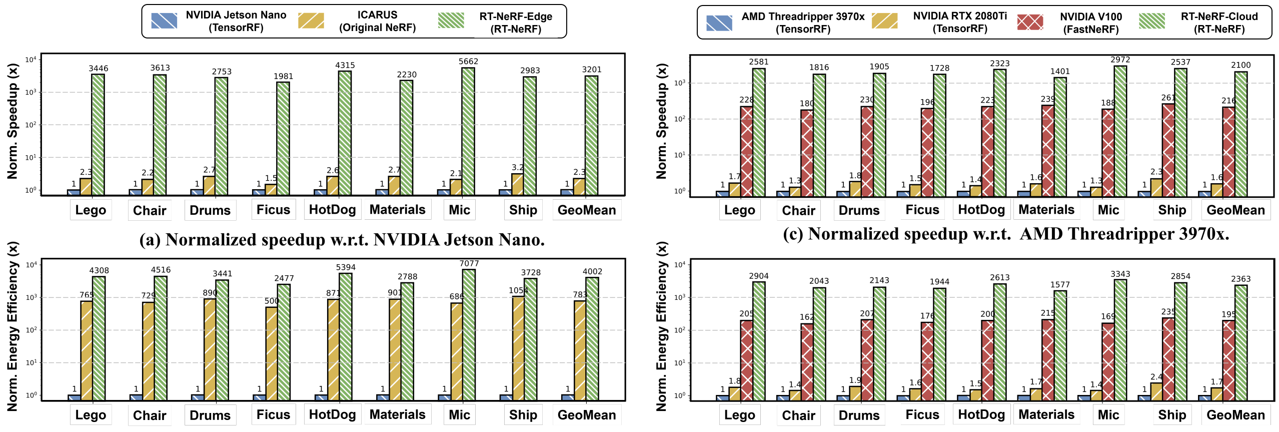
Fig. 14 presents the efficiency improvements achieved by the proposed RT-NeRF in comparison with the five baselines on the eight datasets of Synthetic-NeRF [17]. Compared with the edge devices, the proposed RT-NeRF on average offers 3201×, 1391× speedup and 4002×, 5.1× energy efficiency over NVIDIA Jetson Nano [19] and ICARUS [21] (Fig. 14 (a) and (b)), respectively. Compared with cloud devices, the proposed RT-NeRF on average offers 2100×, 1312×, and 9.7× speedup and 2363×, 1390×, and 12.1× energy efficiency over AMD Threadripper 3970X [1], NVIDIA RTX 2080Ti [18], and NVIDIA V100 [20] (Fig. 14 (c) and (d)), respectively. It is worth noting that when running the same proposed RT-NeRF algorithm on both NVIDIA RTX 2080Ti [18] and our proposed accelerator, as visualized in Fig. 8 (b) and (c), respectively, our RT-NeRF-Edge accelerator can further decrease 97% of the latency in Step ❷-② and achieves 24× speed up over NVIDIA RTX 2080Ti [18], validating the effectiveness of our proposed accelerator.

6 CONCLUSION

We present RT-NeRF, the first algorithm-hardware co-design acceleration of NeRF. On the algorithm level, RT-NeRF integrates an efficient rendering pipeline for leveraging the sparsity of pre-existing points and a coarse-grained view-dependent rendering ordering to avoid processing invisible points. On the hardware level, RT-NeRF adopts a hybrid encoding scheme and integrates both a dual-purpose bi-direction adder&search tree unit and a high-density sparse search unit for ensuring efficient sparse decoding. We believe our work can open up an exciting perspective towards real-time NeRF solutions.

ACKNOWLEDGMENTS

The work is supported by the NSF CCRI program (Award number: 2016727) and the NSF RTML program (Award number: 1937592).



(a) Normalized speedup w.r.t. NVIDIA Jetson Nano.

(b) Normalized energy efficiency w.r.t NVIDIA Jetson Nano.

(c) Normalized speedup w.r.t. AMD Threadripper 3970x.

(d) Normalized energy efficiency w.r.t. AMD Threadripper 3970x.

Figure 14: The normalized speedup and energy efficiency achieved by our proposed RT-NeRF and five baseline devices on the eight datasets of Synthetic-NeRF [17]. All the legends follow the "device (algorithm)" format.

REFERENCES

[1] Advanced Micro Devices, Inc. 2021. 3rd Gen AMD Ryzen™ Threadripper™ 3970X | Desktop Processor | AMD. https://www.amd.com/en/products/cpu/amd-ryzen-threadripper-3970x, accessed 2020-09-01.

[2] Yulong Bian, Chenglei Yang, Fengqiang Gao, Huiyu Li, Shisheng Zhou, Hanchao Li, Xiaowen Sun, and Xiangxu Meng. 2016. A framework for physiological indicators of flow in VR games: construction and preliminary evaluation. *Personal and Ubiquitous Computing* 20, 5 (2016), 821–832.

[3] Cadence. 2022. Cadence Genus&Innovus. https://www.cadence.com/en_US/home/tools/digital-design-and-signoff/synthesis/genus-synthesis-solution.html, accessed 2022-05-20.

[4] Anpei Chen, Zexiang Xu, Andreas Geiger, Jingyi Yu, and Hao Su. 2022. TensoRF: Tensorial Radiance Fields. *arXiv preprint arXiv:2203.09517* (2022).

[5] Shail Dave, Riyadh Baghdadi, Tony Nowatzki, Sasikanth Avancha, Aviral Shrivastava, and Baoxin Li. 2021. Hardware acceleration of sparse and irregular tensor computations of ML models: A survey and insights. *Proc. IEEE* 109, 10 (2021), 1706–1752.

[6] Nashwan Dawood, R Marasini, and John Dean. 2009. 19 VR–Roadmap: A vision for 2030 in the built environment. *Virtual Futures for Design, Construction and Procurement* (2009), 261.

[7] David Eberly. 2006. *3D game engine design: a practical approach to real-time computer graphics*. CRC Press.

[8] Mana Farshid, Jeannette Paschen, Theresa Eriksson, and Jan Kietzmann. 2018. Go boldly!: Explore augmented reality (AR), virtual reality (VR), and mixed reality (MR) for business. *Business Horizons* 61, 5 (2018), 657–663.

[9] Francesco Fassi, Alessandro Mandelli, Simone Teruggi, Fabrizio Rechichi, Fausta Fiorillo, and Cristiana Achille. 2016. VR for cultural heritage. In *International conference on augmented reality, virtual reality and computer graphics*. Springer.

[10] Stephan J Garbin, Marek Kowalski, Matthew Johnson, Jamie Shotton, and Julien Valentin. 2021. Fastnerf: High-fidelity neural rendering at 200fps. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 14346–14355.

[11] Peter Hedman, Pratul P Srinivasan, Ben Mildenhall, Jonathan T Barron, and Paul Debevec. 2021. Baking neural radiance fields for real-time view synthesis. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 5875–5884.

[12] Alain Hore and Djemel Ziou. 2010. Image quality metrics: PSNR vs. SSIM. In *2010 20th international conference on pattern recognition*. IEEE, 2366–2369.

[13] Jens Krüger and Rüdiger Westermann. 2003. Acceleration techniques for GPU-based volume rendering. In *Visualization Conference, IEEE*. IEEE Computer Society, 38–38.

[14] Nelson Max. 1995. Optical models for direct volume rendering. *IEEE Transactions on Visualization and Computer Graphics* 1, 2 (1995), 99–108.

[15] Inc. Meta Platforms. 2021. Introducing Horizon Workrooms: Remote Collaboration Reimagined. https://about.fb.com/news/2021/08/introducing-horizon-workrooms-remote-collaboration-reimagined/, accessed 2021-08-01.

[16] Inc. Meta Platforms. 2021. Oculus Quest 2. https://www.oculus.com/experiences/quest/, accessed 2021-08-01.

[17] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. 2020. Nerf: Representing scenes as neural radiance fields for view synthesis. In *European conference on computer vision*. Springer, 405–421.

[18] NVIDIA LLC. 2021. GeForce RTX 2080 TI Graphics Card | NVIDIA. https://www.nvidia.com/en-me/geforce/graphics-cards/rtx-2080-ti/, accessed 2020-09-01.

[19] NVIDIA LLC. 2021. Jetson Nano Developer Kit. https://developer.nvidia.com/embedded/jetson-nano-developer-kit, accessed 2020-09-01.

[20] NVIDIA LLC. 2021. NVIDIA V100 TENSOR CORE GPU. https://www.nvidia.com/en-us/data-center/v100/, accessed 2020-09-01.

[21] Chaolin Rao, Huangjie Yu, Haochuan Wan, Jindong Zhou, Yueyang Zheng, Yu Ma, Anpei Chen, Minye Wu, Binzhe Yuan, Pingqiang Zhou, et al. 2022. ICARUS: A Lightweight Neural Plenoptic Rendering Architecture. *arXiv preprint arXiv:2203.01414* (2022).

[22] Cheng Sun, Min Sun, and Hwann-Tzong Chen. 2021. Direct Voxel Grid Optimization: Super-fast Convergence for Radiance Fields Reconstruction. *arXiv preprint arXiv:2111.11215* (2021).

[23] Alex Yu, Ruilong Li, Matthew Tancik, Hao Li, Ren Ng, and Angjoo Kanazawa. 2021. PlenOctrees for Real-time Rendering of Neural Radiance Fields. In *ICCV*.

[24] Jerry Zhao, Ben Korpan, Abraham Gonzalez, and Krste Asanovic. 2020. SonicBOOM: The 3rd Generation Berkeley Out-of-Order Machine. (May 2020).

[25] Shulin Zhao, Haibo Zhang, Sandeepa Bhuyan, Cyan Subhra Mishra, Ziyu Ying, Mahmut T Kandemir, Anand Sivasubramaniam, and Chita R Das. 2020. Déja view: Spatio-temporal compute reuse for 'energy-efficient 360 vr video streaming. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 241–253.