Instant-3D: Instant Neural Radiance Field Training Towards On-Device AR/VR 3D Reconstruction

Sixu Li*
Georgia Institute of Technology
Atlanta, GA, USA
sli941@gatech.edu

Chaojian Li*
Georgia Institute of Technology
Atlanta, GA, USA
cli851@gatech.edu

Wenbo Zhu
Georgia Institute of Technology
Atlanta, GA, USA
eiclab.gatech@gmail.com

Boyang (Tony) Yu
Georgia Institute of Technology
Atlanta, GA, USA
eiclab.gatech@gmail.com

Yang (Katie) Zhao
Georgia Institute of Technology
Atlanta, GA, USA
eiclab.gatech@gmail.com

Cheng Wan
Georgia Institute of Technology
Atlanta, GA, USA
cwan39@gatech.edu

Haoran You
Georgia Institute of Technology
Atlanta, GA, USA
hyou37@gatech.edu

Huihong Shi
Georgia Institute of Technology
Atlanta, GA, USA
eiclab.gatech@gmail.com

Yingyan (Celine) Lin
Georgia Institute of Technology
Atlanta, GA, USA
celine.lin@gatech.edu

ABSTRACT

Neural Radiance Field (NeRF) based 3D reconstruction is highly desirable for immersive Augmented and Virtual Reality (AR/VR) applications, but achieving instant (i.e., < 5 seconds) on-device NeRF training remains a challenge. In this work, we first identify the inefficiency bottleneck: the need to interpolate NeRF embeddings up to 200,000 times from a 3D embedding grid during each training iteration. To alleviate this, we propose Instant-3D, an algorithm-hardware co-design acceleration framework that achieves instant on-device NeRF training. Our algorithm decomposes the embedding grid representation in terms of color and density, enabling computational redundancy to be squeezed out by adopting different (1) grid sizes and (2) update frequencies for the color and density branches. Our hardware accelerator further reduces the dominant memory accesses for embedding grid interpolation by (1) mapping multiple nearby points' memory read requests into one during the feed-forward process, (2) merging embedding grid updates from the same sliding time window during back-propagation, and (3) fusing different computation cores to support the different grid sizes needed by the color and density branches of Instant-3D algorithm. Extensive experiments validate the effectiveness of Instant-3D, achieving a large training time reduction of 41× - 248× while maintaining the same reconstruction quality. Excitingly, Instant-3D has enabled instant 3D reconstruction for AR/VR, requiring a reconstruction time of only 1.6 seconds per scene and meeting the AR/VR power consumption constraint of 1.9 W.

*Equal contribution.

ISCA '23, June 17–21, 2023, Orlando, FL, USA.

ACM ISBN 979-8-4007-0095-8/23/06…$15.00
https://doi.org/10.1145/3579371.3589115

CCS CONCEPTS

• **Hardware → Application specific processors**; • **Computer systems organization → Neural networks**; **Embedded hardware**.

KEYWORDS

Neural Radiance Field (NeRF), Hardware Accelerator

ACM Reference Format:
Sixu Li, Chaojian Li, Wenbo Zhu, Boyang (Tony) Yu, Yang (Katie) Zhao, Cheng Wan, Haoran You, Huihong Shi, and Yingyan (Celine) Lin. 2023. Instant-3D: Instant Neural Radiance Field Training Towards On-Device AR/VR 3D Reconstruction. In *Proceedings of the 50th Annual International Symposium on Computer Architecture (ISCA '23), June 17–21, 2023, Orlando, FL, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3579371.3589115

1 INTRODUCTION

On-the-fly 3D reconstruction has become a fundamental task in numerous augmented and virtual reality (AR/VR) applications which involve fast-changing environments [11, 12, 39], e.g., virtual room planner [2], VR painting [31], metaverse 3D asset creation [34], and virtual telepresence [3]. Specifically, 3D reconstruction takes 2D images from a set of *sparsely sampled views* of a 3D scene as its inputs and then generates images of the same scene from *any desired new view*. Compared to offloading to the cloud, on-the-fly 3D reconstruction can offer a smaller communication data size and enhanced privacy protection. For example, a 20 MB reconstructed model [22] may be used instead of 120 MB jpeg images [10]. This alternative is critical for application scenarios with unstable or unavailable internet connections, such as virtual telepresence [3] under which each attendee's environment needs to be reconstructed under varying internet bandwidths at a latency of < 2 seconds [23, 25].

Among the tremendously growing efforts devoted to pushing forward the achievable quality of 3D reconstruction, neural radiance field (NeRF)-based reconstruction has stood out [22] thanks to its state-of-the-art (SOTA) performance in terms of photorealistic reconstruction quality. However, while instant (i.e., < 5 seconds [24])

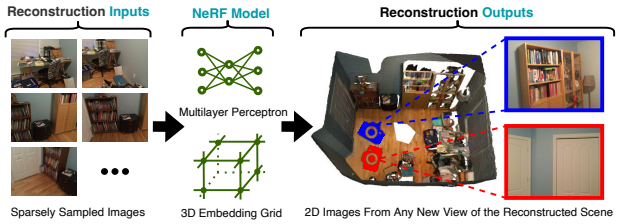


Figure 1: An illustration of NeRF-based 3D reconstruction, which takes 2D images from a set of *sparsely sampled views* of a 3D scene as its inputs and then generates images of the same scene from *any desired new view*.

NeRF-based reconstruction on AR/VR devices for new scenes is highly desirable in unleashing the big promise of photorealistic 3D reconstruction in many emerging applications, it is still not possible even using the most efficient SOTA NeRF training algorithm [24].

To close the aforementioned gap, we first set out to understand and identify the key bottleneck that has limited the achievable runtime efficiency of training NeRF-based reconstruction for new scenes. To do so, we start by conducting extensive profiling measurements of the most efficient NeRF training algorithm, called Instant-NGP [24], on multiple commercial devices with varying levels of power consumption (e.g., 10 W ~ 20 W). After that, we perform various analyses on the runtime breakdown of each step in Instant-NGP [24]'s training pipeline and locate the key bottleneck: the step of interpolating NeRF embeddings from a 3D embedding grid and its corresponding back-propagation process. The purpose of this step is to generate the embeddings of 3D points in the scene to be reconstructed, which include the corresponding points' color and density information, during the reconstruction process. This step needs to be executed more than 200,000 times during each training iteration to ensure photorealistic reconstruction quality. In particular, the heavy workload of this step dominates around 80% of the training runtime of the whole pipeline, as discussed and analyzed in Sec. 2.2.

To tackle the inefficiency bottleneck identified above, we develop an algorithm-hardware co-design acceleration framework, and make the following contributions:

- We comprehensively profile and analyze the runtime bottleneck in the pipeline of the most efficient NeRF training algorithm [24] on multiple devices, and identify the primary cause of the inefficiency: the necessity of interpolating NeRF embeddings from a 3D embedding grid > 200,000 times per training iteration.
- We develop an algorithm-hardware co-design acceleration framework called *Instant-3D* for training NeRFs, which to the best of our knowledge is **the first** that has achieved *instant* on-device NeRF-based 3D reconstruction. Specifically, Instant-3D targets resolving the aforementioned inefficiency by developing dedicated algorithm and hardware innovations that compress the storage requirement, the number of computations, and the number of accesses for the 3D embedding grid in the identified bottleneck step of embedding grid interpolation.
- On the algorithm level, leveraging our discovery that color and density have different sensitivities when it comes to compressing NeRFs, we propose to **decompose the aforementioned embedding grid** in the identified bottleneck in terms of color and density, and then adopt different (1) **grid sizes** and (2) **update frequencies** for the resulting color and density branches, allowing for orthogonally squeezing out the computational redundancy in both branches without compromising the reconstruction quality.
- On the hardware level, we propose a dedicated accelerator that leverages the properties of the aforementioned algorithm to boost hardware efficiency while further reducing the dominant memory accesses during the required embedding grid interpolation of NeRFs. The latter leverages our finding that the memory access pattern during embedding grid interpolation is predictable within a specific region of the scene to be reconstructed, as shown in Sec. 4.2. Specifically, our Instant-3D accelerator highlights (1) **a feed-forward read mapper** that maps the memory read requests of the embeddings of multiple nearby points into one read request during the feed-forward process of NeRF training, (2) **a back-propagation update merger** that merges multiple embedding grid updates from the same sliding time window into one update during the back-propagation phase of NeRF training, and (3) **a multi-core-fusion-based reconfigurable scheme** to fuse different computation cores for supporting the different grid sizes needed by the color and density branches of the Instant-3D algorithm.
- Benchmarking experiments and ablation studies on NeRF-Synthetic [22], SILVR [9], and ScanNet [10] consistently validate the effectiveness of Instant-3D, achieving a training run reduction of 41× - 248× while maintaining the same reconstruction quality as the most efficient NeRF training solution. Excitingly, Instant-3D has enavked instant on-device NeRF-based 3D reconstruction for AR/VR, requiring only 1.6 seconds per scene to reach a decent reconstruction PSNR of 25 (acceptable for image representations [20, 38]) on NeRF-Synthetic [22] dataset while meeting the AR/VR power consumption constraint of 1.9 W.

2 BACKGROUND AND MOTIVATION

2.1 Preliminaries of NeRFs

The Training Pipeline of Vanilla NeRFs. To reconstruct a specific 3D scene, NeRF takes a set of sparsely sampled views of the same scene as inputs and optimizes an underlying continuous volumetric scene function, i.e., a multilayer perceptron (MLP) model [22]. Specifically, NeRFs' training pipeline involves the following six steps as illustrated in Fig. 2. **Step ❶** randomly samples pixels as a batch: One batch of data during each training iteration consists of pixels randomly sampled from all the training images, and these sampled pixels' coordinates and their corresponding RGB colors are the inputs and the ground truth label of the whole NeRF training pipeline, respectively; **Step ❷** maps the pixels to rays: For each sampled pixel, based on its coordinate, a ray formulated as $\mathbf{r} = \mathbf{o} + t\mathbf{d}$ is emitted from the origin (i.e., the camera's center) $\mathbf{o}$ of its corresponding training view along direction $\mathbf{d}$ to pass through this particular pixel, where t represents the distance between the sampled point along this ray and the origin $\mathbf{o}$; **Step ❸** queries features of the points

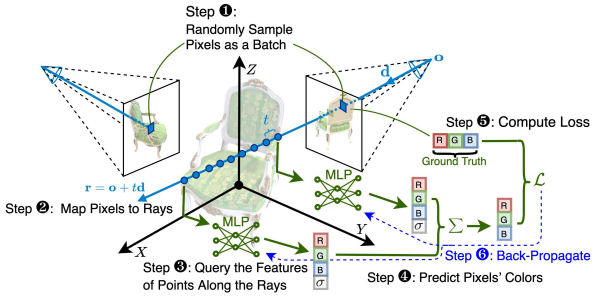


Figure 2: NeRF [22]'s training process involves a total of six steps: Step ❶ randomly samples pixels as a batch, Step ❷ maps the sampled pixels to rays $\mathbf{r} = \mathbf{o} + t\mathbf{d}$ by emitting rays to pass through the corresponding pixels, Step ❸ queries the features (i.e., the RGB color and the density σ) of points along the rays by providing their locations and directions as the inputs to an MLP model, Step ❹ predicts the pixels' colors following the principle of classical volume rendering [21], Step ❺ computes the loss as the squared error between the predicted colors and ground truth colors, and Step ❻ back-propagates through the above fully differentiable pipeline.

along the rays: For each point that has a distance t_k ($k \in [1, N]$, where N represents the total number of the sampled points along each ray) from $\mathbf{o}$, both its location $\mathbf{o} + t_k\mathbf{d}$ and direction $\mathbf{d}$ are applied to an MLP model as inputs. The MLP model then outputs the corresponding density σ_k and an RGB color $\mathbf{c}_k$ as the extracted features of this particular point, i.e., $(\mathbf{o} + t_k\mathbf{d}, \mathbf{d}) \rightarrow (\sigma_k, \mathbf{c}_k)$; **Step ❹** predicts the pixels' colors: Following the principle of classical volume rendering [21], the predicted color $\hat{\mathbf{C}}(\mathbf{r})$ of the pixel corresponding to the ray $\mathbf{r}$ can be computed by integrating the features of all the points along the ray:

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

where N is the number of sampled points along ray $\mathbf{r}$ and T_k denotes the accumulated transmittance along ray $\mathbf{r}$ to point $\mathbf{o} + t_k\mathbf{d}$, which represents the probability of the ray traveling to the point without hitting any other points; **Step ❺** computes the loss of reconstructing the scene, which is defined as the total squared error between the predicted colors $\hat{\mathbf{C}}(\mathbf{r})$ and the ground truth colors $\mathbf{C}(\mathbf{r})$:

$$\mathcal{L} = \sum_{\mathbf{r}\in\mathcal{R}} \left[\left\| \hat{\mathbf{C}}(\mathbf{r}) - \mathbf{C}(\mathbf{r}) \right\|_2^2 \right], \tag{2}$$

where $\mathcal{R}$ is the set of rays in each batch; Finally, **Step ❻** performs back-propagation: As all of the operations of the previous steps (e.g., Eq. 1 and Eq. 2) are differentiable, the weights of the MLP model in Step ❸ can be updated via gradient back-propagation based on the reconstruction loss computed in Step ❺.

After training, to generate the 2D image corresponding to any desired new view of the reconstructed scene, the only difference from the above pipeline is to replace the randomly sampled pixels in Step ❶ with all the pixels in the image of the new view and then stop at Step ❹ without further computing the reconstruction loss or back-propagation.

Vanilla NeRFs' Training Cost. To achieve photorealistic reconstruction quality, the aforementioned training process typically takes around 150,000 iterations per scene, where each iteration executes an MLP model of 1 million FLOPs with a batch size of 786,432 (192 points/pixel × 4,096 pixels). Therefore, the required total training FLOPs is as large as 353,895 trillion FLOPs, requiring > 1 day of training time on one V100 GPU [30]. Such a long training time prohibits vanilla NeRFs [22] from being used for instant on-device AR/VR scene reconstruction which is highly desirable for many emerging applications, such as photorealistic telepresence [3].

SOTA Efficient NeRF Training Technique: Instant-NGP. To alleviate the aforementioned prohibitive cost of training vanilla NeRFs, various works [1, 5, 16, 24] have been proposed to accelerate NeRFs' training process. Among them, Instant-NGP [24] achieved SOTA training speed vs. reconstruction quality trade-offs, e.g., it requires only 5 seconds per scene on a *high-end* RTX3090 GPU [28] and thus has been included in all mainstream NeRF-based 3D reconstruction infrastructures [8, 19, 36]. However, Instant-NGP still cannot fulfill the requirement of instant 3D reconstructions on *resource-constrained* AR/VR devices. In particular, to accelerate the NeRF training process, Instant-NGP [24] replaces the MLP model in Step ❸, which is used to query the features of points along the rays, of vanilla NeRFs with a 3D embedding grid; The latter is stored as a compact 1D hash table. In this way, the more costly MLP inference operations (e.g., 1 million FLOPs) in vanilla NeRFs are now converted into much less costly embeddings interpolation operations (e.g., < 0.00005 million FLOPs). Therefore, as visualized in Fig. 3, Step ❸ in Instant-NGP's training pipeline consists of the following two steps: **Step ❸-①** interpolates embeddings from the embedding grid. Specifically, for each queried point along the rays passing through the pixels of training images, the embeddings e_i of its eight nearest vertices $i \in \{000, 001, ..., 111\}$ in the 3D embedding grid will be fetched from the compact 1D hash table of size T through the hash table index h_i based on their coordinates (x_i, y_i, z_i). Finally, the result of the trilinear interpolation on these eight vertices' embeddings will be used as the embeddings of the queried point. In particular, the hash function $H(\cdot)$ that maps the grid vertices' coordinate (x_i, y_i, z_i) to the hash table index h_i is defined as:

$$h_i = H(x_i, y_i, z_i) = (\pi_1 x_i \oplus \pi_2 y_i \oplus \pi_3 z_i) \bmod T, \tag{3}$$

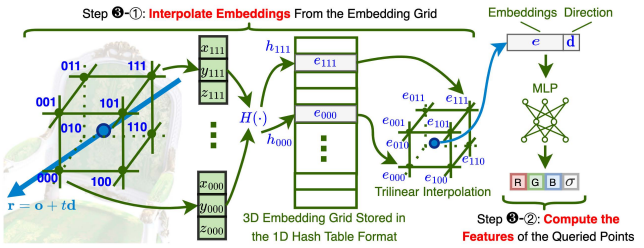


Figure 3: Instant-NGP [24] achieves SOTA training efficiency by replacing Step ❸ (i.e., querying the features of points along the rays using a large 10-layer MLP model) in vanilla NeRFs [22] with both Step ❸-① - Interpolating embeddings from the embedding grid and Step ❸-② - Computing the features of the queried points using a small MLP model.

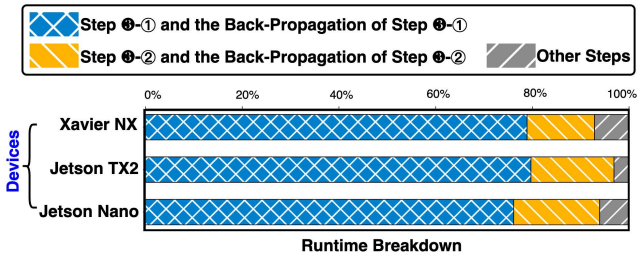


Figure 4: Training runtime breakdown averaged on the eight scenes of NeRF-Synthetic [22] on three representative commercial devices, suggesting that the most efficient NeRF training algorithm [24] is bottlenecked by Step ❸-① (i.e., interpolating embeddings from the embedding grid) and its corresponding back-propagation process on all considered scenes and devices.

where $\oplus$ denotes the bit-wise XOR operation, $\pi_1 = 1$, $\pi_2 = 2654435761$, and $\pi_3 = 805459861$, following the spatial hash function design in [37]. After the queried points, which are along the rays passing through the pixels of training images, obtain their corresponding embeddings from Step ❸-①, the embeddings will be applied as the inputs for **Step ❸-②**, which computes the features of the queried points. Specifically, for each queried point, its embeddings e and direction $\mathbf{d}$ are sent as inputs to a small MLP model to obtain the corresponding density σ and view-dependent color $\mathbf{c}$. Here the small MLP only consists of 3 layers with 64 hidden units, in contrast to the required 10 layers with each having 256 hidden units in the vanilla NeRF [22]. Both the aforementioned lower-cost steps enable Instant-NGP [24]'s SOTA training efficiency. However, even using Instant-NGP [24], it still requires minutes to days of training time to reconstruct each new scene on edge GPUs [26, 27, 29], which is far from the desired instant runtime (i.e., < 5 seconds [24]) for practical on-device 3D reconstruction.

2.2 Profiling Analysis of the SOTA Efficient NeRF Training Process

To close the aforementioned gap between the desired instant on-device 3D reconstruction runtime and the achievable training runtime of the most efficient NeRF training algorithm [24], we conduct extensive profiling measurements of Instant-NGP [24] on commercial devices with varying levels of power consumption, including Jetson Nano [29], which typically consumes power consumption of 10 W, Jetson TX2 [26], which typically consumes power consumption of 15 W, and Xavier NX [27], which typically consumes 20 W. As shown in Fig. 4, the runtime breakdown on the eight commonly used scenes of NeRF-Synthetic [22] consistently indicates that Step ❸-① (i.e., interpolating embeddings from the embedding grid) and its corresponding back-propagation process dominate the overall training runtime of Instant-NGP [24] on all these devices. Based on the above observations, we develop dedicated algorithm and hardware innovations to compress *(1) the storage size of, (2) the number of computations of, and (3) the number of accesses to* the 3D embedding grid in the identified bottleneck of Step ❸-① (i.e., interpolating embeddings from the embedding grid). We present our algorithm design in Sec. 3 and hardware design in Sec. 4, respectively.

3 INSTANT-3D: PROPOSED ALGORITHM

In this section, we present our proposed Instant-3D algorithm. First, we hypothesize that the color and density features have different sensitivities when it comes to compressing NeRFs, and thus can evolve at a different pace during NeRF training. Our hypothesis has been empirically validated based on the consistent qualitative observations as well as the quantified analyses across different datasets, as discussed in Sec. 3.1. Second, to alleviate the training runtime bottleneck during embedding grid interpolation in Instant-NGP, we leverage the above verified sensitivity difference by decoupling the embedding grid into color and density parts. Specifically, this opens up opportunities to adopt (1) different grid sizes (see Sec. 3.2) and (2) different update frequencies (see Sec. 3.3) for the aforementioned color and density branches of the 3D embedding grid. In this way, the computational redundancy can be squeezed out orthogonally in both branches adaptively to boost the overall training efficiency without compromising the reconstruction quality.

3.1 Different Paces of Color and Density During Training

To validate our hypothesis that the color and density features have different sensitivities to the reconstruction quality of NeRFs, and thus can evolve at a different pace during NeRF training, we have conducted extensive visualization experiments and analysis. Both consistently validate our hypothesis. Here we illustrate and discuss one set of experiment results. Fig. 5(a) visualizes the reconstructed

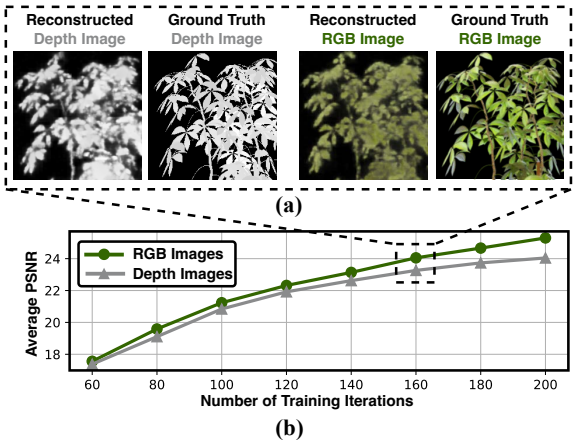


Figure 5: (a) Color and density feature visualization during training: Colors are learned faster than the densities under the same number of training iterations. Here we can see that the color features are of higher quality than those of the density under the same number of training iterations (i.e., at the 160th iteration) on the Ficus scene [22], where the ground truth color and density features are shown as a reference. (b) Quantified PSNR of the color and density features during the whole training trajectory: The PSNR of the color features is consistently higher than that of the density features during the whole training process. Here the plot shows the average RGB/depth images PSNR on the eight scenes [22] vs. the number of training iterations.

RGB image and depth image of the same scene under the same number of training iterations (here a total of 160). It is worth noting that the depth image is not generated during training and is merely used to test the learned density quality here. We can see that the color information has been better learned than that of the density information under the same training iterations, as the reconstructed RGB color of the ficus is closer to the ground truth one while the fine-grained geometry of the ficus, which is decided by the learned density feature, is still not as clear as compared to its ground truth counterpart. Meanwhile, to further quantify the difference regarding the color and density features, Fig. 5(b) shows the Peak Signal-to-Noise Ratio [13] (PSNR) (a higher value corresponds to a better-reconstructed quality) of both the reconstructed color and density features during the whole training trajectory. The averaged PSNR vs. the number of training epochs on eight different datasets shows that the color feature is learned at a faster pace than that of density during NeRF training. For example, it takes a total of 160 iterations vs. 200 iterations for the color and density features to evolve to a quality of 24 dB PSNR. We conjecture that the aforementioned observations are caused by the different optimization for the color and density, i.e., the training loss (Eq. 2) is based on the predicted color rather than the predicted density, indicating that optimizing the color is easier and thus the color features are less sensitive to model compression. Motivated by our above discovery regarding the different sensitivities and pace of the color and density features during NeRF training, we propose to decompose the embedding grid into a color one and a density one. In this way, we can explore the possibility of adopting different grid sizes and different update frequencies for the decomposed branches, as discussed and analyzed in the following two subsections.

3.2 Different Grid Sizes for the Color and Density Branches

Observations. Leveraging the discovery above and our proposed decomposition of the embedding grid, we propose to adopt different grid sizes for the resulting color and density branches of the decomposed embedding grids, aiming to reduce the training time without hurting the achieved reconstruction quality.

Tab. 1 shows one set of our validation experiments in terms of the achieved reconstruction PSNR vs. the measured training time, where we vary the density grid size S_D and color grid size S_C from $S_D : S_C = 0.25 : 1$ to $S_D : S_C = 1 : 0.25$. Here we make two observations. (1) Adopting **different grid sizes for the color and density grids** leads to a better PSNR vs. training runtime trade-off, as compared to adopting the same grid size for both grids as in Instant-NGP [24]. In particular, doing so reduces the training runtime by 12.5% while maintaining the same PSNR as compared to Instant-NGP [24]. (2) **Color features are less sensitive than density features** when their grid size is reduced. Particularly, when the color grid size or density grid size is reduced to 0.25× of the vanilla one in Instant-NGP [24], the achieved reconstruction PSNR is 26.0 dB vs. 25.4 dB, respectively, indicating the lower sensitivity of the former to the grid size (i.e., a higher level of **spatial redundancy**).

Table 1: The achieved reconstruction quality (PSNR) vs. the required training time on the eight scenes of NeRF-Synthetic [22], when adopting different grid sizes for the density grid S_D and the color grid S_C. Here the training runtime is measured on an edge GPU [27].

$S_D : S_C$	Average Training Runtime (s)	Average Test PSNR
1:1 [24]	72	26.0
0.25:1	65 (↓ 9.7%)	25.4
1:0.25	**63 (↓ 12.5%)**	**26.0**

Proposed Technique. Built upon our discovery above and consistent observation, we propose to use a smaller grid size for the color grid than that for the density grid, i.e., we ensure $S_D > S_C$ in our Instant-3D algorithm as visualized in Fig. 6.

3.3 Different Update Frequencies for the Color and Density Branches

Observations. With similar motivation, we propose to adopt different update frequencies for the resulting color and density branches of the decomposed embedding grids to reduce the training time without hurting the reconstruction quality.

Tab. 2 shows one set of our validation experiments in terms of the achieved reconstruction PSNR vs. the measured training time, where we vary the update frequencies for the density grid F_D and color grid F_C from $F_D : F_C = 0.5 : 1$ to $F_D : F_C = 1 : 0.5$. Similarly, we can make two observations. (1) Adopting **different update frequencies for the color and density** grids leads to a better PSNR vs. training runtime trade-off, as compared to adopting the same update frequency for both as in Instant-NGP [24]. Specifically, doing so reduces the training runtime by 9.7% while keeping a similar PSNR as compared to Instant-NGP [24]. (2) **Color features are less sensitive than density features** when the update frequency is reduced. In particular, when the update frequency for the color or density grid is reduced to 0.5× of the vanilla one in Instant-NGP [24], the achieved reconstruction PSNR is 25.9 dB vs. 24.3 dB, respectively, indicating the lower sensitivity of the former to the update frequency (i.e., a higher level of **temporal redundancy**).

Table 2: The achieved reconstruction quality (PSNR) vs. the required training time on the eight scenes of NeRF-Synthetic [22], when adopting different update frequencies for the density grid F_D and the color grid F_C. Here the training runtime is measured on an edge GPU [27].

$F_D : F_C$	Average Training Runtime (s)	Average Test PSNR
1:1 [24]	72	26.0
0.5:1	67 (↓ 6.9%)	24.3
1:0.5	**65 (↓ 9.7%)**	**25.9**

Proposed Technique. Similarly, we propose to use a lower update frequency for the color grid than that for the density grid, i.e., $F_D > F_C$ in our Instant-3D algorithm (see Fig. 6).

4 INSTANT-3D: PROPOSED ACCELERATOR

In this section, we first profile Instant-NGP [24] when applying our proposed algorithm (see Sec. 4.1), showing that the achieved training time is still not satisfactory for instant on-device NeRF training. After that, we further analyze the memory access patterns of the dominant embedding grid interpolation during both the feed-forward and back-propagation processes when training Instant-NGP [24] using our algorithm in Sec. 4.2. Finally, we present

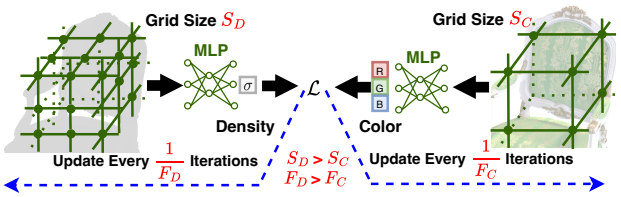


Figure 6: Overview of the proposed Instant-3D algorithm pipeline with the decomposed color and density branches.

our proposed Instant-3D accelerator in Sec. 4.3, where the featured components include (1) a feed-forward read mapper to make good use of the on-chip multi-bank SRAM arrays (see Sec. 4.4), (2) a back-propagation update merger to minimize the number of required SRAM writes (see Sec. 4.5), and (3) a multi-core-fusion-based reconfigurable scheme to support the different grid sizes needed by our Instant-3D algorithm (see Sec. 4.6).

4.1 Motivation: Profiling Instant-NGP with Our Algorithm

Fig. 7 depicts the profiling results of our proposed algorithm, described in Sec. 3, which accelerates the most efficient NeRF training algorithm Instant-NGP [24] by 17.0% on average. However, the required training time per scene to achieve the satisfactory average PSNR of > 25 dB [20, 38] is still around 60 seconds when being executed on edge GPU Xavier NX [27], as shown in Tab. 1 and Tab. 2. Thus, the achievable training runtime is still far from the desired instant (i.e., < 5 seconds [24]) on-device 3D reconstruction. From Fig. 7, we observe that Step ❸-① (interpolating embeddings from the embedding grid) and its corresponding back-propagation process still dominate the training runtime, accounting for around 80% of the total training runtime, motivating us to develop a dedicated accelerator to further boost the training efficiency for achieving instant on-device NeRF-based reconstruction.

4.2 Analyzing the Memory Access Patterns During Training

Memory Access Patterns During Feed-Forward. Motivated by the observation that memory capacity is the bottleneck of runtime in [24], we further analyze the memory access patterns during the embedding grid interpolation that dominates the total training runtime. As illustrated in Sec. 2.1, to obtain the embeddings of each queried point along the camera rays passing through the pixels of the training images, the embeddings of the eight nearest surrounding vertices of the queried point are read from the 3D embedding grid. Multiple 3D points can share the same cube of the 3D embedding grid; thus, their gradients will back-propagate to the same embedding of the cube, indicating the opportunity to merge those memory accesses to the same or similar addresses. Specifically, we inspect all accessed addresses of the embedding grid during the feed-forward process by clustering the eight surrounding vertices into four groups, each of which contains two vertices with the same y-axis and z-axis, as illustrated in Fig. 8. Through extensive measurements on eight scenes [22], we find that (1) the inter-group distances can be very large (the average distance is as high as 60,000); and (2) about 90% of intra-group distances are < 5, as shown in Fig. 9. Both phenomena are consistently observed across different training iterations.

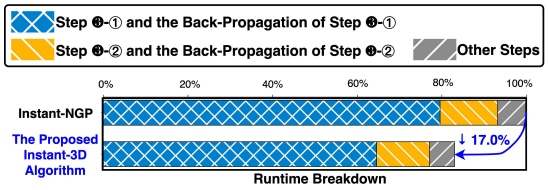


Figure 7: Although the proposed Instant-3D algorithm can further accelerate the most efficient NeRF training algorithm [24] by 17.0% on average, the runtime breakdown averaged on the eight scenes of NeRF-Synthetic [22] on edge GPU Xavier NX [27] suggests that Step ❸-① (i.e., interpolating embeddings from the embedding grid) still dominates the training runtime of the proposed Instant-3D algorithm.

We conjecture that the above two memory access patterns are caused by the **remoteness** and **locality** of memory accessing, respectively. Specifically, after passing the coordinates of the surrounding vertices as the inputs to the hash function in Eq. 3, the output addresses of the embedding grid can be classified into the following two cases. Case 1: If the differences between the coordinates of the surrounding vertices happen on the y-axis or z-axis, then such differences will be amplified by $\pi_2 = 2654435761$ or $\pi_3 = 805459861$, respectively, according to Eq. 3. Such **remoteness** results in quite different addresses among the different groups as observed above (e.g., 60,000 on average, as shown in Fig. 8). Case 2: If the aforementioned differences of coordinates happen on the x-axis, then such differences will not be amplified because of $\pi_1 = 1$ in Eq. 3. Such **locality** results in similar addresses in the same group as observed above (e.g., 90% of the address distances are < 5, as shown in Fig. 9).

Motivated by (1) the fact that the 1D hash table, which represents the 3D embedding grid, is stored in multi-bank SRAM arrays due to the limited SRAM cell's size and (2) the aforementioned unique memory access patterns of the embedding grid during the feed-forward process, we propose a feed-forward read mapper to map the memory read requests of multiple nearby points' embeddings without bank access collisions into one read request, aiming to improve the resource utilization of the multi-bank SRAM arrays. We introduce the proposed detailed design of the feed-forward read mapper in Sec. 4.4.

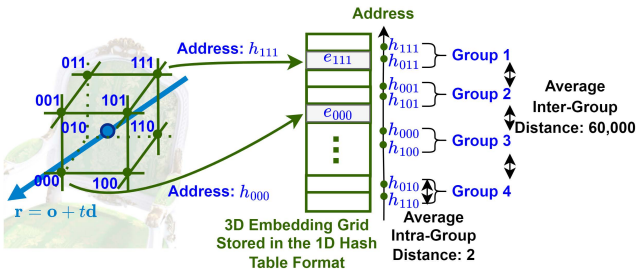


Figure 8: The addresses for the embeddings of the eight neighboring vertices during embedding grid interpolation can be clustered into four groups, which is consistently observed on the eight scenes of NeRF-Synthetic [22].

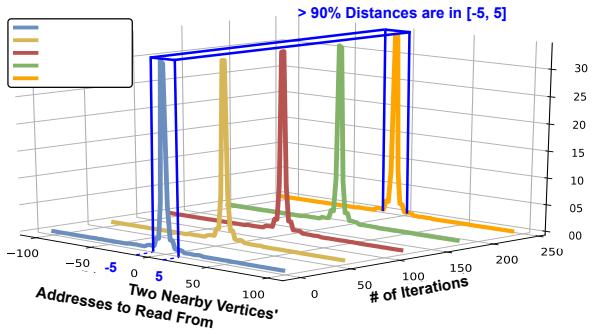


Figure 9: More than 90% of distances between the accessed addresses of neighboring vertices are between [-5, 5] when observing the memory access patterns during the training of the eight scenes in NeRF-Synthetic [22].

Memory Access Patterns During Back-Propagation. As indicated by Eq. 3, when the size of the 1D hash table that stores the 3D embedding grid is larger than the number of vertices in the grid, multiple vertices can share the same stored embeddings in the 1D hash table. This indicates an opportunity to reduce the memory accesses by merging multiple accesses of such shared embeddings into only one. To verify whether such shared embeddings are common in the bottleneck step of embedding grid interpolation, we analyze the number of unique accessed addresses within a small sliding time window (e.g., within 1000 continuous accesses) in Fig. 10. We can observe that (1) the number of unique accessed addresses varies along the training process and features predictable access patterns during the feed-forward and back-propagation processes; (2) for the access patterns during the feed-forward process, the aforementioned cases of shared embeddings do not exist, i.e., all of the 1000 continuous accesses in the sliding window are unique; (3) for the access patterns during the back-propagation process, there exists the cases of shared embeddings among more than five accesses from different timesteps, where ~200 unique accesses are observed among the 1000 continuous accesses.

Based on such consistently observed patterns of shared embeddings during the back-propagation process, we propose a back-propagation update merger to merge the embedding grid updates

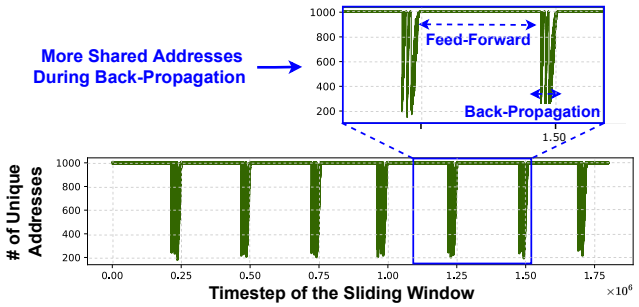


Figure 10: The number of unique accessed addresses within a sliding window of 1000 continuous accesses indicates that there are fewer unique accessed addresses during the back-propagation process, enabling the opportunity to merge the accesses to those addresses of the shared embeddings.

that share the same embedding addresses but in different timesteps into one update operation. We present the detailed design of the proposed back-propagation update merger in Sec. 4.5.

4.3 Proposed Instant-3D Accelerator

Overview of Our Proposed Instant-3D Accelerator. Aiming to design a complete acceleration system for NeRF-based 3D reconstruction, our system includes Dynamic Random Access Memory (DRAM), a host System on Chip (SoC), and our proposed accelerator that consists of three major components: the I/O interface, the MLP units, and the grid cores, as shown in Fig. 11. Specifically, during the feed-forward process of each training iteration, the host SoC first performs Step ❶ (i.e., randomly sampling pixels as a batch) and Step ❷ (i.e., mapping pixels to rays) in the NeRF training pipeline demonstrated in Sec. 2.1. After that, in Step ❸ (i.e., querying the features of points along the rays), the coordinates of those queried points along the rays that pass through the pixels of the training images are applied to our proposed Instant-3D accelerator. Inside our Instant-3D accelerator, Step ❸-① (i.e., interpolating embeddings from the embedding grid) and Step ❸-② (i.e., computing the features of the queried points) are accelerated by the grid cores and MLP units, respectively. Finally, the features (i.e., color and density) of the queried points outputted by our Instant-3D accelerator are fed back to the DRAM, and then the remaining Step ❹ (i.e., predicting pixels' color) and Step ❺ (i.e., computing the reconstruction loss) are performed by the host SoC. The corresponding back-propagation process follows the same workload assignment as the aforementioned feed-forward process, e.g., the back-propagation of Step ❸-① and Step ❸-②, which are identified as the bottleneck step of NeRF training in Sec. 2.2, are performed on the proposed Instant-3D accelerator by the grid cores and MLP units, respectively. We provide detailed descriptions of our grid core and MLP unit designs as follows.

Grid Core Design. To perform the embedding grid interpolation and its corresponding back-propagation process of Step ❸-①, our Instant-3D accelerator consists of four grid cores, four of our proposed Back-Propagation Update Merger (BUM) units (i.e., 1 BUM unit per grid core), and seven of our proposed Feed-Forward Read Mapper (FRM) units (i.e., 4 FRM unit inside grid cores and 3 FRM unit among grid cores). In particular, to perform Step ❸ (i.e., interpolating embeddings from the embedding grid) with the grid cores in our Instant-3D accelerator, the components in the grid cores adopt the following order to execute the feed-forward process of this step: (1) The 3D Coordinate Buffer SRAM first caches all the 3D coordinates of the queried points along the rays that pass through the pixels of the current batch's training images; (2) The coordinates of each queried points' nearest eight vertices in the grid are calculated by *the Interpolation Coord. Pre Compute Unit*; (3) The aforementioned eight vertices' coordinates are applied to *the Hash Function Compute Unit* to execute the hash function (Eq. 3) and output the corresponding eight addresses of the embedding grid stored in a 1D hash table format; (4) The addresses of the embedding grid to be accessed are fed into *the Interpolation Address Multi-Output Double Buffer*; (5) The FRM units (see Sec. 4.4) first map multiple embedding grid read requests into fewer ones without causing memory bank access collisions and then send those

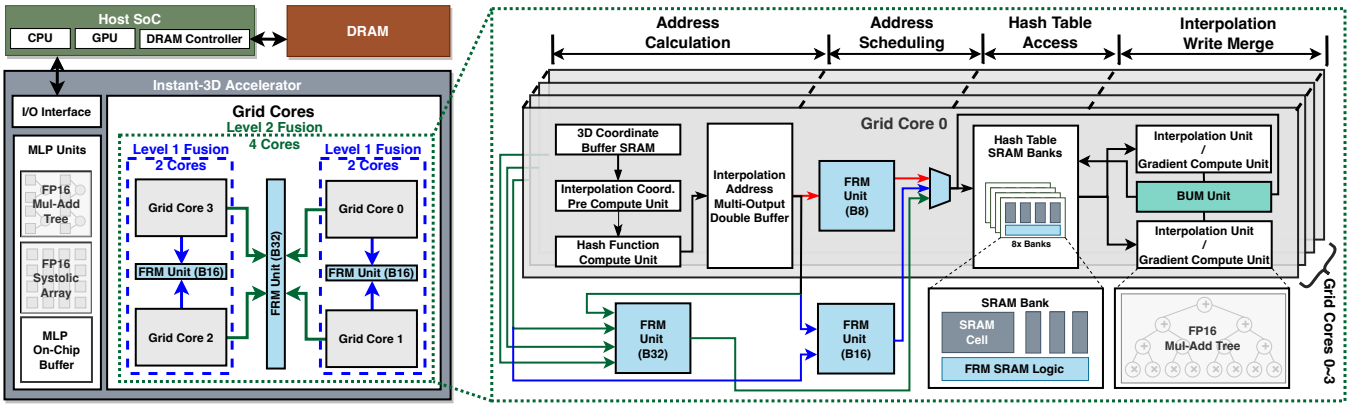


Figure 11: Overall architecture of our proposed Instant-3D accelerator. There are 3 different modes corresponding to different hash table sizes: (1) For a hash table size of 256 KB, Level 0 standalone mode (marked as red) is activated. In this mode, the four grid cores run independently, and the SRAM access of each core is managed by its internal FRM unit (B8); (2) For a hash table size of 512 KB, Level 1 Fusion mode (marked as blue) is activated, during which two grid cores are fused by scheduling the SRAM access of the two cores in the shared FRM Unit (B16); (3) For a hash table size of 1MB, Level 2 Fusion mode (marked as green) is activated, during which all the 4 grid cores are fused together, and the SRAM access is managed by one FRM Unit (B32). Here the "B8/B16/B32" stands for 8/16/32 SRAM banks.

requests to the Hash Table SRAM Banks to fetch the corresponding embeddings; (6) Finally, the fetched embeddings are applied into *the Interpolation Unit* to obtain the trilinear interpolated values for generating the embeddings of the queried points. During the back-propagation of Step ❸, all the aforementioned components, except the Interpolation Unit, perform the same workload as the aforementioned feed-forward process. Meanwhile, the Interpolation Unit is reconfigured into the Gradient Computation Unit for the calculation of the gradients of each accessed embeddings in the current batch. Moreover, to reduce memory accesses during the aforementioned action of writing back the updated embeddings, we propose to include the BUM unit (see Sec. 4.5) in each grid core to merge the updates of the same address into one update operation.

MLP Unit Design. To perform the feed-forward and back-propagation process of MLP in Step ❸-②, we adopt two types of computing unit for them: (1) a systolic array MLP Unit and (2) a multiplier-adder-tree MLP Unit, dedicated to matrix multiplications with (1) a relatively large output channel (e.g., > 3) and (2) a relatively small output channel (e.g., ≤ 3), respectively. Such a design with two unit types is inspired by the observations in [14, 33], showing that the multiplier-adder-tree can achieve a higher hardware utilization than the systolic array under the cases with relatively small output channels (e.g., ≤ 3).

4.4 Feed-Forward Read Mapper to Better Use SRAM Arrays

As mentioned in Sec. 4.2, to read all the eight embeddings of each queried point's nearest vertices in the embedding grid, we divide the whole 1D hash table that stores the 3D embedding grid into eight banks equally and consider 2 cells per bank. Thus, the four clustered groups of eight embeddings are observed to be distributed in four or two memory banks, which causes low utilization of the multi-bank SRAM arrays (i.e., $4/8 = 50\%$ or $2/8 = 25\%$ SRAM bank utilization, assuming 8 banks in total for this case). To improve the memory utilization, we propose the FRM unit to better utilize SRAM bandwidth during the feed-forward process. Specifically, as shown in Fig. 12(a), multiple SRAM read requests from different clock cycles can be mapped into one clock cycle when there is no bank access collision. To achieve the goal of improving memory utilization, as shown in Fig. 12(b), the FRM unit is designed to first fetch a batch of addresses from the address buffer, then detect the bank access collisions of those addresses. After that, the FRM unit maps the memory read requests without collisions into one and finally sends those mapped requests to the SRAM banks.

4.5 Back-Propagation Update Merger Minimizing SRAM Writes

During the back-propagation process of each training iteration, there can be multiple (e.g., more than five) update operations to

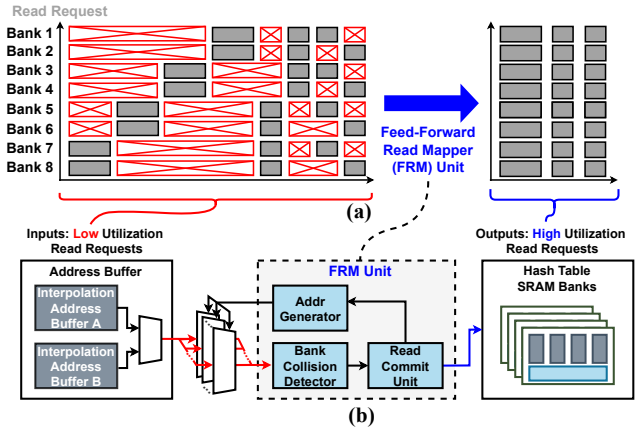


Figure 12: The proposed FRM unit maps low utilization read requests to high utilization ones, as visualized in its (a) dataflow and implemented by its (b) hardware schematic.

the same embeddings of the embedding grid, because of the cases where multiple vertices in the grid share the same embeddings stored in the 1D hash table, as analyzed in Sec. 4.2. Inspired by the design strategy of trading higher-cost memory storage/access for lower-cost computation in [6, 40], we propose a BUM unit to trade higher-cost memory write accesses for the lower-cost operations of merging the updates by accumulating the update values first and then writing back to the embedding grid.

Specifically, as shown in Fig 13(a), with the proposed BUM unit, if the current input address matches any cached ones in the BUM buffer (i.e., the case of t_0 in Fig 13(a)), the corresponding update operations will be merged into one by accumulating the values to be updated; If the current input address does not match any cached ones (i.e., the case of t_1 in Fig 13(a)), then the input address will be inserted into the BUM buffer. Meanwhile, if any address in the BUM buffer reaches the tail of the BUM buffer, this address will be popped out, and the corresponding accumulated values to be updated will be one write request to the SRAM.

In the proposed BUM unit shown in Fig 13(b), for each input address and its corresponding gradient, the gradient is multiplied by the pre-set learning rate, and the address is first fed into the One-to-All-Match module to verify whether it matches the addresses in different entries of the BUM buffer; After that, the address is sent to the matched entry to perform the accumulation of the values to be updated. However, if there is no matched entry for the input address, an empty entry is used to store the input address and its corresponding values to be updated (i.e., the gradient multiplied by the learning rate). Because the size of the BUM buffer is fixed, a controller and a counter to count the timesteps to the last update values accumulation for each entry are included in the BUM unit to read out the accumulated values to be updated in the entry and write them to the SRAM when the counter exceeds a pre-set threshold. Thus, the proposed BUM unit with the aforementioned design can merge the multiple memory write accesses to the same address into a single one by accumulating the update values in the BUM buffer.

4.6 Reconfigurable Scheme Supporting Different Grid Sizes

To leverage the properties of our proposed Instant-3D algorithm, i.e., adopting different grid sizes and update frequencies for the decomposed color and density grids (see Sec. 3), it is desirable for our Instant-3D accelerator to be scalable to different grid sizes and update frequencies. Meanwhile, it is worth noting that our Instant-3D accelerator is naturally scalable to different update frequencies by skipping one back-propagation process every $(\frac{1}{1-F})$ iteration, where F can denote the update frequencies of the color or density grid. Thus, the key is to support different grid sizes.

To tackle the aforementioned challenge, we propose a multi-core-fusion-based reconfigurable scheme. As shown in Fig. 14, such a scheme is implemented by including a 16-bank FRM unit for each pair of grid cores to enable a Level 1 Fusion Mode for the pair (i.e., utilizing two grid cores in total to support a grid size of 512KB) and a 32-bank FRM unit between two pairs of the grid cores to make up a Level 2 Fusion Mode for the two pairs (i.e., utilizing four grid cores in total to support a grid size of 1MB).

5 EVALUATION

In this section, we first introduce the detailed settings for evaluating our proposed Instant-3D framework in Sec. 5.1, and then benchmark

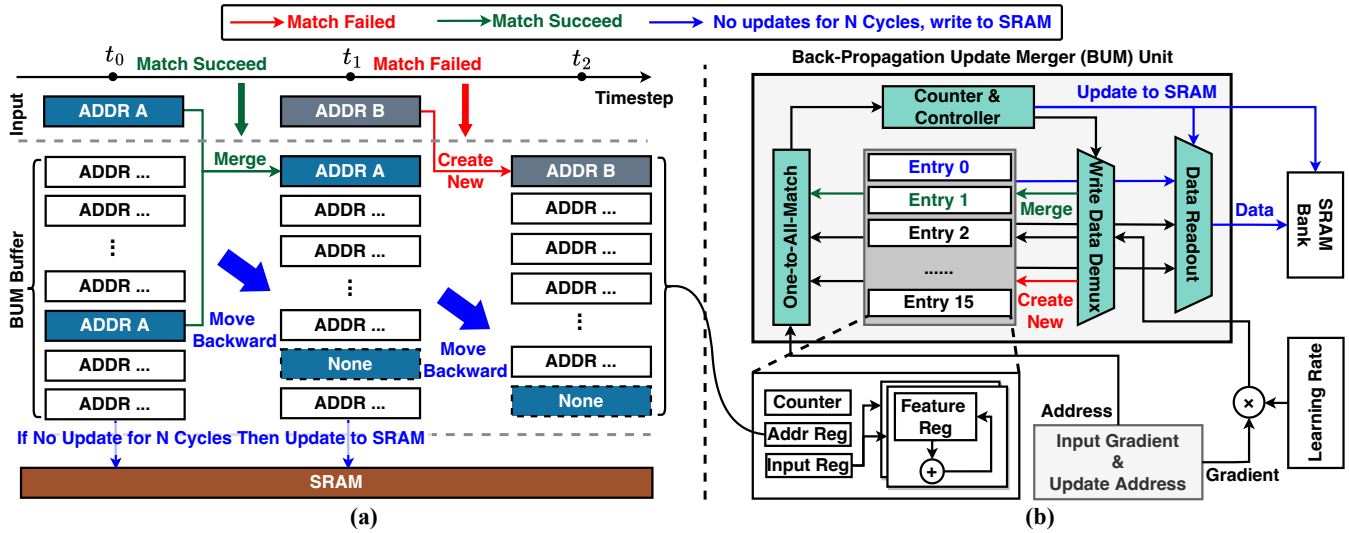


Figure 13: The proposed BUM unit merges the update operations to the same embeddings but at different timesteps into one update operation, as visualized in its (a) dataflow and implemented by its (b) hardware schematic. Case 1 (marked as red): A match fails between the input address and any cached address in the BUM buffer (e.g., Address B at timestep t_1), then BUM creates a new entry to store the input address and its corresponding values to be updated; Case 2 (marked as green): A successful match (e.g., Address A at timestep t_0), then the corresponding update operations of the matched pair are merged into one. If any address in the BUM buffer has not been matched for N cycles, its corresponding values to be updated will be written back to SRAM (marked as blue).

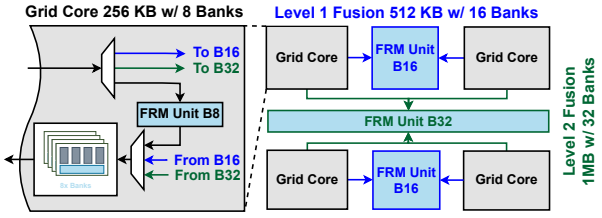


Figure 14: A reconfigurable scheme for supporting different grid sizes in our Instant-3D algorithm in Sec. 3, where B8/B16/B32 stands for 8/16/32 SRAM banks. Because different numbers of SRAM banks require different input sizes for enabling full utilization (e.g., 8/16/32 banks need 8/16/32 addresses), we design the FRM unit with different bank sizes for achieving full utilization of the SRAM bandwidth.

our proposed Instant-3D algorithm and accelerator in Sec. 5.2 and Sec. 5.3, respectively.

5.1 Evaluation Settings

Datasets & Baselines. *Datasets:* To evaluate the achieved reconstruction quality and training efficiency of our proposed Instant-3D, we conduct experiments on the commonly-used NeRF-Synthetic [22], the large-scale SILVR [9], and the real-world-captured ScanNet [10] datasets. The reconstruction quality is measured by the PSNR of the corresponding test set. *Baselines:* We consider three commercial hardware devices as our baselines, including a Jetson Nano [29] with a typical power consumption of 10 W, a Jetson TX2 [26] with a typical power consumption of 15 W, and a Xavier NX [27] with a typical power consumption of 20 W. In our experiments, the energy of the aforementioned baseline devices is measured using embedded power-rail monitors following [17]. We emphasize that to the best of our knowledge, our proposed Instant-3D is the first to develop accelerators for NeRF-based 3D reconstruction training, and thus there are no dedicated NeRF training accelerator baselines for comparison. The hardware specifications of all baselines and our Instant-3D are summarized in Tab. 3. Note that we do not benchmark with RT-NeRF [15] and ICRUAS [33], which are the prior works on NeRF acceleration, as they can only perform NeRF inference instead of NeRF training, failing to support the desired instant on-device 3D reconstruction.

Instant-3D Algorithm Implementation. The implementation of the proposed Instant-3D algorithm is based on the open-sourced Instant-NGP [24]'s official CUDA implementation. We follow the default algorithm settings in Instant-NGP [24] excepting for the density grid's size and update frequency, i.e., S_D and F_D, and the color grid's size and update frequency, i.e., S_C and F_C, which are set as $S_D : S_C = 1 : 0.25$ and $F_D : F_C = 1 : 0.5$. Such a configuration is selected as the one that compresses the training cost most but also maintains the same reconstruction quality with Instant-NGP [24] by performing a grid search from $1 : 0.125$, $1 : 0.25$, $1 : 0.5$, and $1 : 0.75$. Thus, to implement $S_D : S_C = 1 : 0.25$, the 1D hash tables store the density grid and color grid that have 2^{16} and 2^{18} entries, respectively, and both have 2 features per entry, following Instant-NGP [24]. Additionally, to implement $F_D : F_C = 1 : 0.5$, the density grid is updated by the back-propagation of the reconstruction loss every iteration, and the color grid is updated every two iterations.

Table 3: A summary of the considered devices' specifications.

Device	Jetson Nano [29]	Jetson TX2 [26]	Xavier NX [27]	Instant-3D
Technology	20 nm	16 nm	12 nm	28 nm
SRAM	2.5 MB	5 MB	11 MB	1.5 MB
Area	118 mm^2	N/A	350 mm^2	6.8 mm^2
Frequency	0.9 GHz	1.4 GHz	1.1 GHz	0.8 GHz
DRAM Bandwidth	LPDDR4-1600 25.6 GB/s	LPDDR4-1866 59.7 GB/s	LPDDR4-1866 59.7 GB/s	LPDDR4-1866 59.7 GB/s
Typical Power	10 W	15 W	20 W	1.9 W

Instant-3D Accelerator Implementation. To evaluate the energy and the area of the proposed Instant-3D accelerator, we implement our accelerator in RTL, synthesize the RTL design using Synopsys Design Compiler [35], and then place & route the design using Cadence Innovus [4], based on a commercial 28nm CMOS technology. In all our experiments, we set the reordering pipeline depth of our proposed FRM and BUM units to be 16, based on empirical observations and find it to be generally applicable to all datasets in our experiments; and each SRAM array connected to FRM and BUM units can handle eight unique memory accesses. Specifically, we use 16-bit half-precision floating-point arithmetic for all algorithm-related computations to ensure minimal rendering quality degradation due to quantization. In addition, we develop a cycle-accurate simulator to estimate the training efficiency of our proposed Instant-3D accelerator on different datasets with the assumption of a 59.7 GB/s DRAM bandwidth, which is the same as the typical DRAM bandwidth in LPDDR4-1866 used in the baseline hardware devices, Jetson TX2 [26] and Xavier NX [27]. The proposed Instant-3D accelerator consumes an area of 6.8 mm^2 and an average power consumption of 1.9 W, depicted in Fig. 15

5.2 Instant-3D Algorithm's Performance

Benchmark with the SOTA Efficient NeRF Training Algorithm. To evaluate the effectiveness of our Instant-3D algorithm

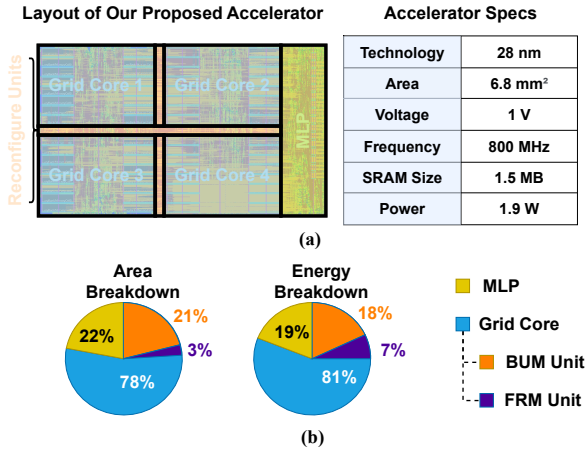
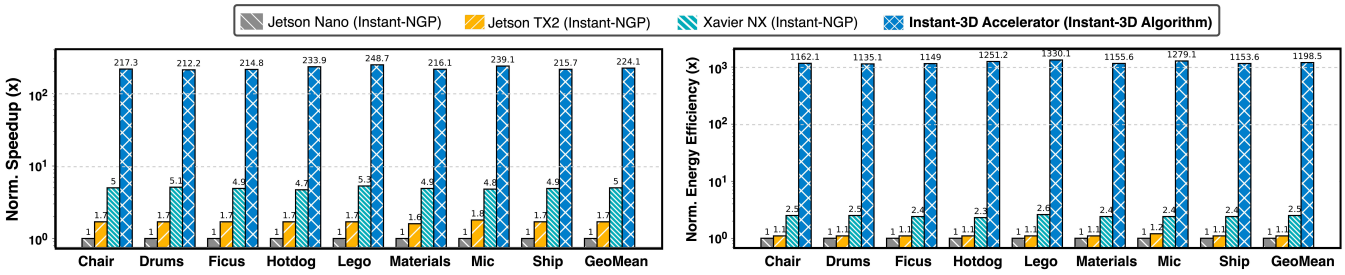


Figure 15: (a) The layout and performance specifications and (b) the layout, energy, and area breakdown of our Instant-3D accelerator.



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

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

Figure 16: The normalized speedup and energy efficiency achieved by our proposed Instant-3D and three baseline devices on the eight scenes of NeRF-Synthetic [22]. The legends follow the "device (algorithm)" format.

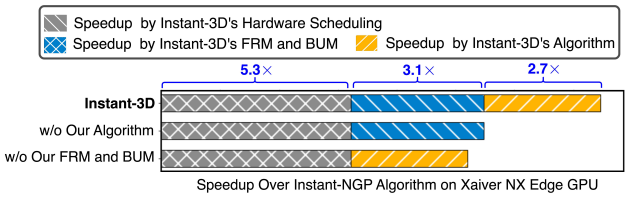


Figure 17: The speedup (in logarithmic scale) over Instant-NGP [24] algorithm on Xaiver NX Edge GPU [27] achieved by different techniques of our proposed Instant-3D on NeRF-Synthetic dataset [22].

in Sec. 3, we benchmark it with the most efficient NeRF training algorithm, Instant-NGP [24], in terms of the achieved reconstruction PSNR and training runtime on edge GPU, Xavier NX [27]. As shown in Tab. 4, we can observe that our proposed Instant-3D algorithm surpasses the most efficient NeRF training algorithm [24] in terms of the reconstruction quality vs. training runtime trade-offs, e.g., 60 seconds vs. 72 seconds to achieve the same quality (26.0 PNSR averaged on the 8 scenes of NeRF-Synthetic [22]).

5.3 Instant-3D Accelerator's Performance

Benchmark with the SOTA Efficient NeRF Training Devices. We summarize the NeRF training efficiency improvements achieved by our proposed Instant-3D accelerator in Fig. 16. Specifically, as compared to the three baselines on NeRF-Sythetic [22], the proposed Instant-3D accelerator offers on average 224×/132×/45× speedups and 1198×/1089×/479× more energy efficiency over Jetson Nano [29]/Jetson TX2 [26]/Xavier NX [27], respectively. Specifically, our Instant-3D's 45× speedup over Xaiver NX [27] results from (1) 2.7× speedup by the Instant-3D algorithm, (2) 3.1× speedup by our FRM and BUM units, which are inspired by the observed memory access patterns in Sec. 4.2, and (3) 5.3× speedup by better hardware scheduling, i.e., our multi-core-fusion-based-reconfigurable-scheme, as shown in Fig. 17.

Table 4: Benchmark our proposed Instant-3D algorithm with the most efficient NeRF training algorithm [24], in terms of the PSNR and training runtime on edge GPU Xavier NX [27].

Methods	Avg. Train. Runtime			PSNR		
	NeRF-Synthetic [22]	SILVR [9]	ScanNet [10]	NeRF-Synthetic [22]	SILVR [9]	ScanNet [10]
Instant-NGP [24]	72 sec.	135 sec.	84 sec.	26.0	25.0	24.9
Instant-3D	**60 sec.**	**111 sec.**	**72 sec.**	**26.0**	**25.1**	**25.1**

Ablation Study on the Effectiveness of the Proposed FRM and BUM Units. As illustrated in the Sec. 4.4 and Sec. 4.5, we propose an FRM unit to make good use of the on-chip multi-bank SRAM arrays and a BUM unit to minimize the number required SRAM writes. To verify the effectiveness of the two proposed units, we summarize the runtime of Instant-3D w/o the FRM unit or BUM unit in Fig. 18. In particular, the proposed FRM unit can trim down the runtime by 31.1% on average. Additionally, with the proposed BUM unit on top of the FRM unit, the runtime reduction ratio can be further enlarged to 68.6%, which indicates that both the proposed FRM unit and BUM unit are necessary for our Instant-3D accelerator to achieve the desired instant on-device NeRF-based reconstruction. It is worth noting that, to the best of our knowledge, controlling the precise/fine-grained memory accesses required by the FRM and BUM units is not currently supported by CUDA's APIs on the benchmark devices summarized in Tab. 3.

Ablation Study on Necessity of Co-Design. To the best of our knowledge, our proposed Instant-3D, as an algorithm-hardware co-design acceleration framework, is the first that has achieved instant on-device NeRF-based 3D reconstruction. To verify the necessity of such a co-design strategy, we summarize the runtime of our Instant-3D w/o the proposed algorithm techniques or hardware techniques in Tab. 5. Specifically, with our proposed Instant-3D algorithm, we can trim down the runtime by 83.0% as compared to the most efficient NeRF training algorithm [24] on the same edge GPU Xavier NX [27]. Moreover, with both our proposed Instant-3D algorithm and accelerator, the runtime can be reduced to 2.2% of the most efficient NeRF training solution (Instant-NGP [24] on a

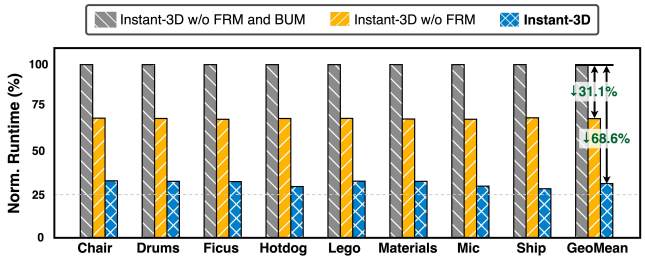


Figure 18: The normalized runtime achieved by our proposed Instant-3D accelerator w/o the proposed FRM unit, depicted in Sec. 4.4, or w/o BUM unit, depicted in Sec. 4.5, on the eight scenes of NeRF-Synthetic [22].

Table 5: The normalized runtime achieved by our Instant-3D framework w/o the proposed algorithm techniques or hardware techniques on different datasets.

NeRF Training Solution (Algorithm @ Hardware)	Normalized Runtime (%) on NeRF-Synthetic [22]	SILVR [9]	ScanNet [10]
Instant-NGP [24] @ Xavier NX [27]	100	100	100
Instant-3D Algorithm @ Xavier NX [27]	83.3	82.2	85.7
Instant-3D Algorithm @ Instant-3D Accelerator	2.3	3.4	3.2

Xavier NX [27]), which indicates the necessity of the co-design strategy of our Instant-3D framework.

6 RELATED WORKS

To the best of our knowledge, both of the only two existing works on designing dedicated accelerators for NeRF [15, 33] can only perform NeRF inference, and thus cannot be adopted to achieve the goal of instant on-device 3D reconstruction. When compared with the SOTA NeRF inference accelerator [15], our Instant-3D can achieve real-time (> 30 FPS) rendering speed-up while only consuming 19.5% of energy per frame and 36% of the chip area. Moreover, Instant-3D achieves a 1,8002× speedup over an MLP-based NeRF inference accelerator [33]. It is worth noting that the prior works on MLP or Convolutional Neural Network (CNN) training acceleration [7, 18, 32] do not support the dominant operations of interpolating embeddings from the embedding grid, as analyzed in Sec. 2.2. Thus, they are not applicable to accelerate NeRF training.

7 CONCLUSION

We propose Instant-3D, which to the best of our knowledge is **the first** that has achieved instant on-device NeRF-based 3D reconstruction. Instant-3D algorithm decomposes the bottleneck embedding grid in terms of color and density to orthogonally squeeze out the redundancy in both branches; Instant-3D accelerator integrates an FRM unit to make good use of the on-chip multi-bank SRAM arrays, a BUM unit to minimize the number of required SRAM writes, and a reconfigurable scheme to support our instant-3D algorithm. We believe this work can open up an exciting perspective toward instant on-device 3D reconstruction for AR/VR.

ACKNOWLEDGMENTS

This work was supported by the NSF Computing and Communication Foundations (CCF) program (Award ID: 2211815) and CoCoSys, one of the seven centers in JUMP 2.0, a Semiconductor Research Corporation (SRC) program sponsored by DARPA.

REFERENCES

[1] Alex Yu and Sara Fridovich-Keil, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. 2021. Plenoxels: Radiance Fields without Neural Networks. arXiv:2112.05131 [cs.CV]
[2] Arkio ehf. 2022. Design, mix and share realities. https://www.arkio.is/, accessed 2022-11-01.
[3] Meta Quest Blog. 2020. Tackling telepresence: 'spatial' delivers collaborative computing on Oculus quest. https://www.meta.com/nl-nl/blog/quest/tackling-telepresence-spatial-delivers-collaborative-computing-on-oculus-quest.
[4] Cadence. 2022. Innovus Implementation System - Cadence. https://www.cadence.com/en_US/home/tools/digital-design-and-signoff/soc-implementation-and-floorplanning/innovus-implementation-system.html, accessed 2022-05-20.
[5] Anpei Chen, Zexiang Xu, Andreas Geiger, Jingyi Yu, and Hao Su. 2022. TensoRF: Tensorial Radiance Fields. *arXiv preprint arXiv:2203.09517* (2022).
[6] Yu-Hsin Chen, Joel Emer, and Vivienne Sze. 2016. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. *ACM SIGARCH Computer Architecture News* 44, 3 (2016), 367–379.
[7] Seungkyu Choi, Jaehyeong Sim, Myeonggu Kang, Yeongjae Choi, Hyeonuk Kim, and Lee-Sup Kim. 2020. An energy-efficient deep convolutional neural network training accelerator for in situ personalization on smart devices. *IEEE Journal of Solid-State Circuits* 55, 10 (2020), 2691–2702.
[8] XRNeRF Contributors. 2022. OpenXRLab Neural Radiance Field Toolbox and Benchmark. https://github.com/openxrlab/xrnerf.
[9] Martijn Courteaux, Julie Artois, Stijn De Pauw, Peter Lambert, and Glenn Van Wallendael. 2022. SILVR: A Synthetic Immersive Large-Volume Plenoptic Dataset. *arXiv preprint arXiv:2204.09523* (2022).
[10] Angela Dai, Angel X. Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. 2017. ScanNet: Richly-annotated 3D Reconstructions of Indoor Scenes. In *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE.*
[11] 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.
[12] 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, 139–157.
[13] Alain Hore and Djemel Ziou. 2010. Image quality metrics: PSNR vs. SSIM. In *2010 20th international conference on pattern recognition.* IEEE, 2366–2369.
[14] Mingqiang Huang, Yucen Liu, Changhai Man, Kai Li, Quan Cheng, Wei Mao, and Hao Yu. 2022. A High Performance Multi-Bit-Width Booth Vector Systolic Accelerator for NAS Optimized Deep Learning Neural Networks. *IEEE Transactions on Circuits and Systems I: Regular Papers* 69, 9 (2022), 3619–3631. https://doi.org/10.1109/TCSI.2022.3178474
[15] 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 *2022 IEEE/ACM International Conference on Computer-Aided Design.*
[16] Chaojian Li, Bichen Wu, Albert Pumarola, Peizhao Zhang, Yingyan Lin, and Peter Vajda. 2023. INGeo: Accelerating Instant Neural Scene Reconstruction with Noisy Geometry Priors. In *Computer Vision–ECCV 2022 Workshops: Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part III.* 686–694.
[17] Chaojian Li, Zhongzhi Yu, Yonggan Fu, Yongan Zhang, Yang Zhao, Haoran You, Qixuan Yu, Yue Wang, and Yingyan Lin. 2021. Hw-nas-bench: Hardware-aware neural architecture search benchmark. *arXiv preprint arXiv:2103.10584* (2021).
[18] Jiajun Li, Guihai Yan, Wenyan Lu, Shuhao Jiang, Shijun Gong, Jingya Wu, Junchao Yan, and Xiaowei Li. 2019. TNPU: An efficient accelerator architecture for training convolutional neural networks. In *Proceedings of the 24th Asia and South Pacific Design Automation Conference.* 450–455.
[19] Ruilong Li, Matthew Tancik, and Angjoo Kanazawa. 2022. NerfAcc: A General NeRF Acceleration Toolbox. *arXiv preprint arXiv:2210.04847* (2022).
[20] Xiangjun Li and Jianfei Cai. 2007. Robust transmission of JPEG2000 encoded images over packet loss channels. In *2007 IEEE International Conference on Multimedia and Expo.* IEEE, 947–950.
[21] Nelson Max. 1995. Optical models for direct volume rendering. *IEEE Transactions on Visualization and Computer Graphics* 1, 2 (1995), 99–108.
[22] 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.
[23] Robert B Miller. 1968. Response time in man-computer conversational transactions. In *Proceedings of the December 9-11, 1968, fall joint computer conference, part I.* 267–277.
[24] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. 2022. Instant Neural Graphics Primitives with a Multiresolution Hash Encoding. *arXiv preprint arXiv:2201.05989* (2022).
[25] Fiona Fui-Hoon Nah. 2004. A study on tolerable waiting time: how long are web users willing to wait? *Behaviour & Information Technology* 23, 3 (2004), 153–163.
[26] NVIDIA Inc. 2021. NVIDIA Jetson TX2. https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-tx2/, accessed 2020-09-01.
[27] NVIDIA Inc. 2022. Jeston Xavier NX Series Modules. https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-xavier-nx/, accessed 2022-06-01.
[28] NVIDIA LLC. 2021. GEFORCE RTX 3090 FAMILY. https://www.nvidia.com/en-us/geforce/graphics-cards/30-series/rtx-3090-3090ti/, accessed 2022-06-01.
[29] NVIDIA LLC. 2021. Jetson Nano Developer Kit. https://developer.nvidia.com/embedded/jetson-nano-developer-kit, accessed 2020-09-01.
[30] NVIDIA LLC. 2021. NVIDIA V100 TENSOR CORE GPU. https://www.nvidia.com/en-us/data-center/v100/, accessed 2020-09-01.
[31] Oisoi Studio. 2022. Painting VR on Quest 2. https://www.oculus.com/experiences/quest/3106117596158066/, accessed 2022-11-01.
[32] Ximing Qiao, Xiong Cao, Huanrui Yang, Linghao Song, and Hai Li. 2018. AtomLayer: A universal ReRAM-based CNN accelerator with atomic layer computation. In *Proceedings of the 55th Annual Design Automation Conference.* 1–6.

[33] Chaolin Rao, Huangjie Yu, Haochuan Wan, Jindong Zhou, Yueyang Zheng, Minye Wu, Yu Ma, Anpei Chen, Binzhe Yuan, Pingqiang Zhou, Xin Lou, and Jingyi Yu. 2022. ICARUS: A Specialized Architecture for Neural Radiance Fields Rendering. *ACM Trans. Graph.* 41, 6, Article 234 (nov 2022), 14 pages. https://doi.org/10.1145/3550454.3555505

[34] Sketchfab. 2022. Metaverse 3D models. https://sketchfab.com/tags/metaverse, accessed 2022-11-01.

[35] Synopsys. 2022. Design Compiler - Synopsys. https://www.synopsys.com/implementation-and-signoff/rtl-synthesis-test/dc-ultra.html, accessed 2022-05-20.

[36] Matthew Tancik, Ethan Weber, Evonne Ng, Ruilong Li, Brent Yi, Justin Kerr, Terrance Wang, Alexander Kristoffersen, Jake Austin, Kamyar Salahi, Abhik Ahuja, David McAllister, and Angjoo Kanazawa. 2023. Nerfstudio: A Modular Framework for Neural Radiance Field Development. *arXiv preprint arXiv:2302.04264* (2023).

[37] Matthias Teschner, Bruno Heidelberger, Matthias Müller, Danat Pomerantes, and Markus H Gross. 2003. Optimized spatial hashing for collision detection of deformable objects.. In *Vmv*, Vol. 3. 47–54.

[38] Nikolaos Thomos, Nikolaos V Boulgouris, and Michael G Strintzis. 2005. Optimized transmission of JPEG2000 streams over wireless channels. *IEEE Transactions on image processing* 15, 1 (2005), 54–67.

[39] 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.

[40] Yang Zhao, Xiaohan Chen, Yue Wang, Chaojian Li, Haoran You, Yonggan Fu, Yuan Xie, Zhangyang Wang, and Yingyan Lin. 2020. Smartexchange: Trading higher-cost memory storage/access for lower-cost computation. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 954–967.