NeRFHub: A Context-Aware NeRF Serving Framework for Mobile Immersive Applications

Bo Chen[1], Zhisheng Yan[2], Bo Han[2], Klara Nahrstedt[1]
[1]University of Illinois at Urbana-Champaign, [2]George Mason University

ABSTRACT

Neural Radiance Fields (NeRF) are recognized for their exceptional photo-realism quality and superior modeling capabilities compared to traditional methods. NeRF empowers a novel application, termed *NeRF serving*. It delivers data from a server to a mobile client and renders 3D scenes on the client, facilitating a broad spectrum of mobile immersive applications. Towards a satisfactory user experience, we must serve NeRF with low latency while meeting constraints of high visual quality and real-time smoothness. Existing NeRF variants easily violate the constraints or cause an unnecessarily high latency when the diverse applications, mobile devices, and 3D scenes, termed the *contexts*, change in real life. In this paper, we present NeRFHub, a novel context-aware NeRF serving framework for mobile immersive applications. NeRFHub adeptly manages storage and computation costs, scales to diverse contexts, and swiftly navigates the vast design space inherent in NeRF serving. The evaluation results show that NeRFHub serves synthetic objects with 56%-66% reduced latency and realistic scenes with 26%-55% reduced latency when compared to the baseline without compromising quality or smoothness.

CCS CONCEPTS

• **Human-centered computing** → **Ubiquitous and mobile computing systems and tools**; • **Computing methodologies** → **Rendering**; **Neural networks**; • **Networks** → **Network services**.

KEYWORDS

Immersive Computing, Neural Rendering, Multi-Objective Optimization, Model Training, Photo-realism

ACM Reference Format:
Bo Chen[1], Zhisheng Yan[2], Bo Han[2], Klara Nahrstedt[1], [1]University of Illinois at Urbana-Champaign, [2]George Mason University . 2024. NeRFHub: A Context-Aware NeRF Serving Framework for Mobile Immersive Applications. In *The 22nd Annual International Conference on Mobile Systems, Applications and Services (MOBISYS '24), June 3–7, 2024, Minato-ku, Tokyo, Japan.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3643832.3661879

MOBISYS '24, June 3–7, 2024, Minato-ku, Tokyo, Japan

ACM ISBN 979-8-4007-0581-6/24/06
https://doi.org/10.1145/3643832.3661879

1 INTRODUCTION

Neural Radiance Fields (NeRF) [35] is an emerging technology for modeling 3D scenes. NeRF models a 3D scene with a multi-layer perception (MLP) neural network. It involves two major steps: training and rendering. Training turns images capturing the same scene from different view angles and locations into a. Rendering derives a 2D image based on the given viewport by evaluating the MLP. This unique 3D scene modeling technology with neural networks allows NeRF to outperform traditional methods based on mesh and point clouds, with better photo-realism [19] and modeling capability [11, 26, 27, 35, 45]. Due to the limitation of the original version of NeRF [35], numerous variants [12, 18, 25, 37, 40–42] emerges, which are specialized for the training and rendering speed, and the dynamics of scenes.

NeRF has already seen adoption in various mobile immersive applications to provide immersive traveling, gaming, medical imaging, and shopping experiences [13, 43, 48, 49, 53]. In these applications, NeRF is created offline at a cloud server, delivered over the Internet, and rendered at the mobile device. This process is referred to as *NeRF serving*. Towards better user experience, NeRF serving is assessed by metrics of quality [51, 59] (the closeness between the rendered scene and the original scene), smoothness [4] (the speed for a scene to be rendered), and latency [15] (the duration between the time when a user requests a scene and the time the first frame of the scene is rendered).

Specifically, we model NeRF serving as a problem that minimizes the size of data representation with the constraints of quality and smoothness. We term the user- or application-defined parameters in NeRF serving, e.g., application, device, scenes, and NeRF variant types and constraints values, as the *context*. Given a specific context, we define the *configuration* as the remaining configurable parameters controlling how the data is transmitted and rendered in NeRF, e.g., the training hyper-parameters and neural network specifications. The set of configurations is the design space of NeRF serving.

In realistic applications, the context varies easily in reality as the application has diverse considerations and the user has varied preferences. In contrast, the configuration for existing NeRF variants is hardly configurable as any change in what is to be transmitted and how the scene is rendered must be followed by a time-consuming training process in NeRF. To make matters worse, NeRF variants are mostly designed with no knowledge of the context. As a result, there is often a mismatch between the actual context and the NeRF variant, causing the violation of constraints or sub-optimal latency.

We present NeRFHub, a context-aware NeRF serving framework for mobile immersive applications. NeRFHub first turns an off-the-shelf NeRF variant and multi-view images through a special training procedure into a unique NeRF data representation, whose computation complexity and bitrate are flexibly configured. Based

on the context of a request for a 3D scene, NeRFHub appropriately configures this data representation to minimize latency without violating other constraints. Finally, the configured data representation is delivered to the mobile device and rendered. Realizing NeRFHub necessitates solving three challenges.

#1 Resource-intensive configuration adaptation. Optimizing NeRF serving for a specific context requires adapting the configuration of NeRF, which is resource-intensive. To adapt the computation overhead in rendering, a straightforward way is to derive data representations with varying computation complexities. However, the storage overhead would be prohibitive for a large number of computation constraints. Regarding bitrate adaptation, we identify two types of approaches for NeRF [47, 50]. One performs compression using a codebook, paired with a sophisticated MLP during rendering to attain satisfactory quality [47]. The other introduces computation-intensive operations like entropy decoding, inverse quantization, and inverse 3D DCT in rendering. These approaches introduce a significant computation overhead for rendering, making it impractical for mobile devices.

#1 Solution. Our intuition is to realize lightweight adaptation through proactive training. First, lightweight adaptation can be realized through the adaptive removal of parameters in the data representation, without performing intensive computation. For instance, the computation can be adapted by adjusting the number of hidden layer channels in the MLP, instead of storing multiple copies of data representation. A potential problem of such lightweight adaptation is that the adapted data representation is not well-trained like the original data representation, which may cause quality degradation after adaptation. Proactive training is designed to overcome this issue by tailoring offline training based on possible adaptations of the data representation. Specifically, for computation adaptation, we alternate the MLP's number of hidden layer channels in training, allowing the data representation with different configurations of the MLP to be trained consistently. With proactive training, we can adapt the configuration in a lightweight manner while avoiding quality degradation.

#2 Limited serving scalability. A straightforward way for context-aware serving is to exhaust configurations from the whole design space and locate one that minimizes the downloading size without violating constraints. However, for every context, we must repeatedly exhaust the whole design space, which makes it difficult to scale the serving to a large number of contexts.

#2 Solution. We have two insights to improve the serving scalability. First, we prove that the optimal configurations in the original design space are the same as in a reduced design space, where the MLP configuration is set to constant. Moreover, this MLP configuration can be identified by building a lookup table in constant time, which enables NeRF serving with reduced complexity. Second, we notice that the problem of NeRF serving can be decoupled into an offline multi-objective optimization (MOO) problem and an online searching problem. By solving the MOO problem offline, online searching can be performed with a much smaller design space than the original one, which scales easily to more contexts.

#3 Vast design space. Solving the offline MOO problem requires evaluating a multitude of configurations, leading to a vast design space of over a million possible configurations. Without existing analytical models that map configurations to metrics, the reliable way to find the optimal configuration is to exhaust all possible configurations, which is time-consuming.

#3 Solution. We tackle this problem with an evolutionary algorithm that efficiently explores the design space. To further accelerate it, we adopt parallelism and efficient sampling techniques. The parallelism technique leverages the finding that it is unnecessary to jointly explore all configuration knobs. Instead, we can fix the configuration knob to a few values, creating several reduced design spaces, which can be solved in parallel, accelerating profiling. The efficient sampling technique utilizes the fact that the profiling outcome is not sensitive to the number of evolutions and evaluated views in profiling. Therefore, we can make profiling efficient by reducing these numbers, with minimal impact on the profiling outcome.

We evaluate NeRFHub in a realistic setting serving synthetic 3D objects and realistic 3D scenes. Targeting the mobile scenario, we consider a baseline system implemented atop MobileNeRF [12], a state-of-the-art approach for neural rendering on mobile devices without context awareness. The results show that NeRFHub saves the latency of synthetic objects by 56%-66% and realistic scenes by 26%-55%. The quality of NeRFHub is comparable to the baseline with a difference of less than 1 dB in Peak Signal-to-Noise Ratio (PSNR). Moreover, NeRFHub's smoothness is at least real-time (30 fps) or as good as the baseline.

We summarize our contributions as follows.

(1) We design and implement NeRFHub, a context-aware NeRF serving framework.
(2) We adapt computation and bitrate in a lightweight manner by leveraging proactive training.
(3) We scale context-aware serving to a large number of contexts with a lookup table and offline MOO.
(4) We make profiling efficient via parallelism and efficient sampling techniques.
(5) We evaluate NeRFHub in a realistic setting, which shows a significant reduction in latency without compromising visual quality and rendering speed when compared to the baseline.

2 BACKGROUND AND MOTIVATION

2.1 Neural Radiance Fields (NeRF)

3D scene representation. NeRF represents a 3D scene as a 5D vector-valued function whose input is a 3D location $\vec{x} \in \mathbb{R}^3$ and a 2D viewing direction $\vec{d} \in \mathbb{R}^2$ and whose output is the RGB radiance $\vec{c} \in \mathbb{R}^3$ and opacity $\sigma \in \mathbb{R}$, as shown.

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

where F_Θ is a learned multi-layer perception (MLP) with trainable weights Θ.

Rendering. Given a ray $\vec{r}(s) = \vec{o} + s\vec{d}$ whose origin $\vec{o}$ and direction $\vec{d}$ are defined by the camera, the rendered colors of this ray $\vec{C}(\vec{r})$ on the camera is an integral of the opacity-weighted radiance over the bounds, s_n and s_f, of the volume, as described in Equation 2.

$$\vec{C}(\vec{r}) = \int_{s_n}^{s_f} T(s)\sigma(\vec{r}(s))\vec{c}(\vec{r}(s), \vec{d})ds, \tag{2}$$

where $T(s) = e^{-\int_{s_n}^{s} \sigma(\vec{r}(s))ds}$ is the accumulated opacity.

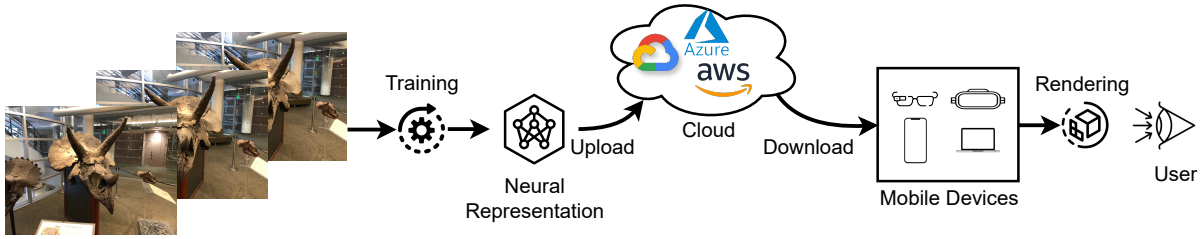


Figure 1: Pipeline of NeRF serving.

Advantages. NeRF has two main advantages.

- Photo-realism. In contrast to traditional methods [22, 30], early novel view synthesis methods [33, 34, 44] and neural 3D representation methods [20, 38, 39], NeRF attains better visual quality [19].
- Modeling capability. NeRF can model complex visual effects, e.g., transparent or translucent objects [35], varying lighting conditions [45], the human head [26], style transfer [11, 27] better than traditional methods like mesh-based or point-cloud-based approaches.

A multi-modal data representation. Due to the limitation of the original NeRF [35], NeRF has evolved into numerous variants. These NeRF variants present a multi-modal data representation mainly consisting of MLP, geometry, and feature grids.

- MLP produces the RGB radiance and opacity at a 3D location, which is shared by most NeRF variants [35, 40–42].
- Geometry is a data structure describing how the scene is occupied with voxels [25] or faces [12], which accelerates rendering [12, 25] and training [18] by skipping unoccupied area in computation.
- Feature grids are pre-computed feature vectors, which facilitate faster training [18, 37] and rendering [12, 25] and variable bitrates [47, 50] by shifting computation-intensive MLP operations from online to offline.

Applications. NeRF finds utility in various domains due to its multifold advantages.

- Travelling. NeRF has been introduced to Google Maps to accurately recreate the full context of a place like its lighting and the texture of materials [43].
- Gaming. A plugin is introduced by Luma AI to enhance the gaming experience by importing and rendering volumetric NeRFs inside Unreal Engine 5 in real time [49].
- Medical imaging. NeRF has been adopted in medical imaging to generate accurate 3D models of internal structures from 2D scans, potentially enhancing the precision of procedures and improving patient outcomes [13, 53].
- Shopping. NeRF has been integrated into Taobao, a shopping app, to provide new shopping experiences like allowing customers to place virtual goods in a house [48].

2.2 NeRF Serving

Figure 1 describes *NeRF serving*, a critical technology for the aforementioned applications, which derives NeRF data representation via model training from a set of images with different views, uploads the NeRF data representation to the cloud, delivers the NeRF data representation to mobile devices and renders the 3D scene in immersive applications.

Metrics. The metrics of quality, smoothness, and latency assess the user experience of NeRF serving.

- Quality measures the closeness between a remotely rendered scene and a baseline high-quality scene, e.g., a scene without loss in compression. The quality can be assessed by visual metrics of PSNR, SSIM [51], and LPIPS [59]. A higher quality means better preservation of the details of the scene.
- Smoothness measures the rendering speed of a particular view in a scene, which can be quantified by the number of frames being rendered per second [4]. A higher smoothness means less lagging when a user changes the view.
- Latency measures the time it takes for a scene to be completely visible to a user, similar to the latency of a HTTP request [15]. It involves network transmission of the data representation from a remote server to the local device and rendering the first frame using transmitted data. A low response time causes less waiting for the user.

Problem formulation. High quality, better smoothness, and low latency are generic goals for NeRF serving. Because of the importance of latency in multiple mainstream applications, we particularly focus on minimizing latency in this paper, with smoothness and quality constraints. Still, our framework generalizes to other utility functions as discussed in §7. Intuitively, the latency is positively correlated with the size S of the data representation. Therefore, we formulate an optimization problem, as shown.

$$\begin{aligned} \theta^* &= \underset{\theta \in \Psi}{\operatorname{argmax}}\, S(\theta) \\ \text{s.t.} \quad & Q(\theta) \geq Q_{\min} \\ & R(\theta) \geq R_{\min} \end{aligned} \tag{3}$$

Ψ is the set of all feasible configurations. $Q_{\min}, R_{\min}$ are constraints of quality and smoothness, respectively.

Significance of contexts. The *context* of NeRF serving involves the immersive application, mobile device, and 3D scene, altering the trade-offs between quality, smoothness, and size.

Firstly, immersive applications have different constraints for each metric, which alters hyper-parameters ($Q_{\min}, R_{\min}$) in Equation 3. For instance, the low quality can be tolerated in particular pixelized games, e.g., Minecraft [1], Undertale [3], and Terraria [2], but not in medical imaging where high quality is crucial for the precision of procedures. In terms of smoothness, it is highly demanded in gaming toward immersive gaming experiences [36] while it is less important in non-urgent travel planning.

Bo Chen[1], Zhisheng Yan[2], Bo Han[2], Klara Nahrstedt[1]
[1]University of Illinois at Urbana-Champaign, [2]George Mason University

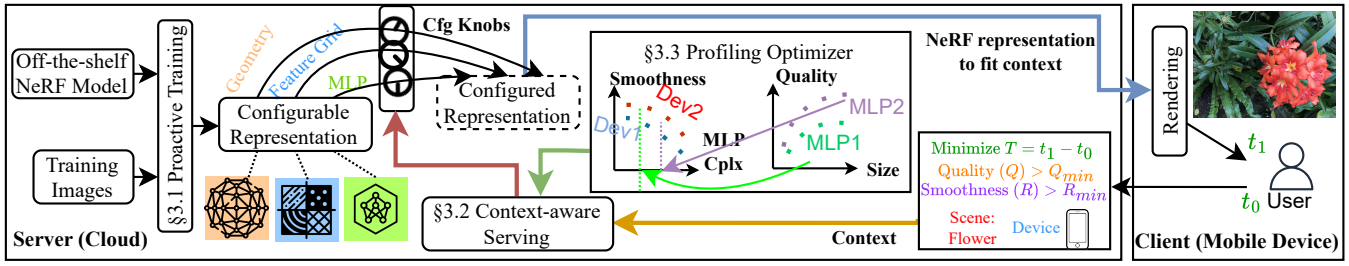
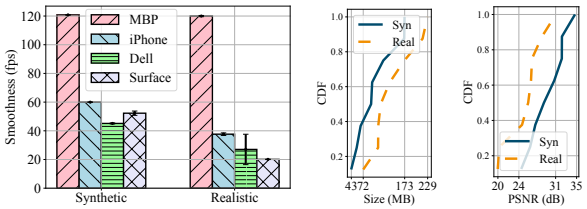


Figure 2: System Architecture.



(a) Smoothness (b) Size (c) Quality

Figure 3: The impact of mobile device and 3D scene on smoothness, size, and quality.

To study the impact of mobile device and 3D scene, we measure different metrics on MacBook Pro (MBP), iPhone 12 (iPhone), Dell Precision 5510 laptop (Dell), and Microsoft Surface (Surface) using MobileNeRF [12] with synthetic [35] and realistic [34] scene datasets. Figure 3 shows smoothness (average), size (CDF), and quality (CDF).

Secondly, mobile devices cause varying smoothness of the same data representation and the impact of mobile devices varies in different scenes. A powerful device like MBP can render roughly 3× faster than a less powerful one, Dell, on realistic scenes. While Surface has better smoothness than Dell on synthetic scenes, Dell demonstrates better smoothness on realistic scenes, which is likely a cause of different acceleration strategies for rendering with WebGL in MobileNeRF.

Thirdly, 3D scenes have diverse smoothness, sizes, and quality. As shown in Figure 3(a), the rendering speed is generally faster on synthetic scenes. In Figure 3(b), synthetic scenes are generally smaller in size than realistic scenes, and the size of scenes of the same type (realistic or synthetic) can vary by one magnitude. Figure 3(c) illustrates synthetic scenes generally have higher quality than realistic scenes, and the quality of scenes of the same type (realistic or synthetic) can vary by at most 11 dB.

The context is crucial as the solution of Equation 3 derived for one context might violate the constraints or achieve sub-optimal latency when the context varies, which occurs easily under a realistic setting.

2.3 Challenges

The varying contexts require a context-aware design to optimally solve Equation 3, involving three challenges.

(1) Resource-intensive configuration adaptation: existing solutions for computation and bitrate adaptation of NeRF require significant storage costs and extra computation overheads on the rendering device.
(2) Limited serving scalability: context-aware serving requires repeatedly searching the configuration that best fits a specific context, which scales poorly to a large number of contexts.
(3) Vast design space: the data representation for NeRF rendering has multiple configuration knobs covering the geometry and feature grids, which involve over a million possible configurations.

3 NeRFHub DESIGN

Figure 2 provides an overview of NeRFHub. NeRFHub first converts an off-the-shelf NeRF representation into one data configurable representation via a proactive training process (§3.1). Here, the data components like geometry, feature grids, and MLP can be configured to meet diverse computation demands for mobile rendering (§3.1.1) and bandwidth demands for downloading (§3.1.2). More importantly, the adaptation of configuration is lightweight, incurring minimal storage and computation overheads.

Then, NeRFHub leverages a context-aware serving module (§3.2) that configures NeRF representation according to the context sent with the user's request. Then, the configured data representation is delivered to the mobile device and rendered. It scales to multiple contexts via the reduction of the configuration dimension using a lookup table (§3.2.1) and a decoupled serving process (§3.2.2).

The profiling optimizer (§3.3) improves the offline profiling in decoupled serving with an evolutionary algorithm. It further accelerates this algorithm via parallelism (§3.3.1) and efficient sampling techniques (§3.3.2).

3.1 Proactive Training

Proactive training prepares the NeRF representation to enable lightweight adaptation strategies for computation and bitrate. For computation, we achieve storage-efficient adaptation by training multiple MLP channels (§3.1.1). Regarding bitrate, we attain computation-efficient adaptation through fine-tuning for various bit depths (§3.1.2).

3.1.1 Shrinking-Based Computation Adaptation. A straightforward idea for storage-efficient computation adaptation is to have a single data representation and adapt to different computation demands by rendering a portion of this data representation. As the MLP

mainly accounts for the computation demands, we can meet different computation demands by "shrinking" the number of hidden layer channels in the MLP. The shrinking is determined by a ratio $r \in (0, 1)$, termed the *model width*. A "shrunk" MLP has the same input and output shapes and the same number of layers as the original MLP. The difference is that a shrunk MLP with model width $r \in (0, 1)$ keeps the most important $r \times L$ channels of hidden layers, where L represents the number of channels in each hidden layer. As such, we meet various computation demands with a single data representation.

Challenge. An issue with this adaptation strategy is the shrunk MLP is not well-trained like the original MLP. Hence, the rendering quality may degrade after adaptation. One possible solution to this problem is the progressive training technique [7, 54, 55]. First, it trains one MLP until convergence. Second, it jointly trains the existing MLP and a shrunk MLP until convergence by randomly sampling one of them in each training iteration. It progressively performs the above steps to incorporate as many shrunk MLPs with smaller model widths as needed. With this approach, we can jointly train an MLP with multiple shrunk MLPs.

Aiming at flexibility in computation adaptation, it is natural to optimize MLPs in NeRF with a wide range and a dense set of model widths, e.g., $1, 2, \ldots, 100$ channels in hidden layers. Unfortunately, this choice degrades the quality of NeRF, e.g., 1-2 dB in PSNR, compared to the same MLP trained individually. The reason is that MLPs of different model widths operate at different sets of parameters when optimized individually. Although they achieve reasonable performance when optimized together, they affect each other and compromise the performance, which becomes more noticeable when we optimize a large number of MLPs simultaneously.

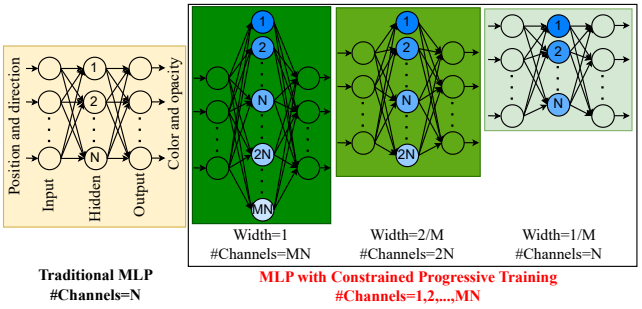


Figure 4: The traditional MLP has a fixed computation complexity. Constrained progressive training allows one MLP to adapt to different computation complexities.

Constrained progressive training. Our insight is that a wide range and a dense set of channels are not necessary in practice for two reasons. First, real-world devices' computation capacity is discrete and sparse. Second, the performance of an MLP marginally improves with the model width when its number of hidden layer channels exceeds a threshold, which is validated in Figure 23.

Therefore, we propose to perform progressive training in a constrained way which includes only a sparse set of MLPs with a limited number of channels in hidden layers. Given the channel number of MLP N in an off-the-shelf NeRF representation, we optimize MLPs shrunk to mN channels, where $m = 1, 2, \ldots, M$. M and N

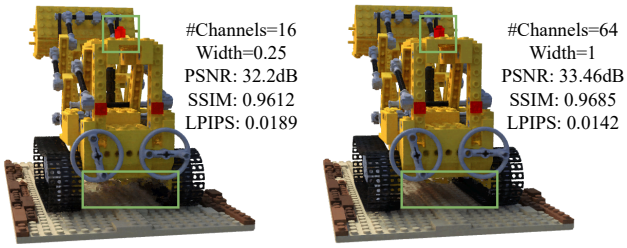


Figure 5: Constrained progressive training allows one MLP to adapt computation for different qualities.

are configurable parameters. Figure 4 compares a traditional MLP and how an MLP with constrained progressive training adapts to different computation complexities and saves storage costs. Figure 5 illustrates how constrained progressive training allows the adaptation of the MLP to achieve different visual quality with significant improvement in the green bounding box.

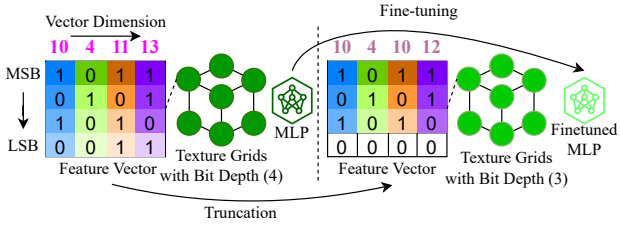


Figure 6: Truncation-based bitrate adaptation involves truncating values in the feature grids according to different bit depth and fine-tuning MLP.

3.1.2 Truncation-Based Bitrate Adaptation. Bitrate adaptation involves MLP, geometry, and feature grids. MLP is less significant for bitrate adaptation in the mobile scenario as it is typically lightweight to support a real-time frame rate [12]. For instance, the MLP is around four orders of magnitudes smaller than the total size of the data representation of MobileNeRF [12] (Figure 20(c)). Regarding the adaptation of the size of geometry, it can be achieved by popular methods like Draco [21]. Hence, we focus on feature grids.

While it is possible to apply traditional or neural compression approaches [9, 10, 46, 50, 52] to the compression of feature grids, they would incur computation-intensive operations at the rendering side. Aiming at computation-efficient bitrate adaptation, we draw insights from a previous study [12]. This study shows quantizing the floating point feature values in the feature grids to 8 bits minimally affects rendering quality. Hence, a natural thought is to adapt the bitrate via the truncation of the bit depth. Specifically, given a bit depth d, we keep the top-d most significant bits (MSB) of the binary representation of each feature value while iteratively replacing the least significant bits (LSB) with zeros, which will not be transmitted over the network to reduce bitrates, as shown in Figure 6. For example, a binary representation of 1011 becomes 1010 after truncation based on the bit depth 3. In this way, we can

Bo Chen[1], Zhisheng Yan[2], Bo Han[2], Klara Nahrstedt[1]
[1]University of Illinois at Urbana-Champaign, [2]George Mason University

effectively adapt the bitrate without causing additional computation on the rendering side while maintaining reasonable rendering quality.

Challenge. Nevertheless, truncation introduces errors in feature values, which result in visible glitches in the rendered image, as shown in Figure 7 (left). The glitch gets more noticeable with smaller bit depths, which introduces more errors. Although we can train data representations with different configurations of bit depths in truncation, there would be a significant training cost and storage cost.

Figure 7: Selected MLP fine-tuning improves quality.

Selected MLP fine-tuning. The key intuition is that the loss in visual quality caused by truncation can be effectively mitigated by fine-tuning. As illustrated in Figure 7, fine-tuning the MLP improves the quality in PSNR from 16.77 dB to 21.8 dB with noticeable visual improvement within the purple rectangle. More importantly, as mentioned earlier the MLP in NeRF for mobile rendering is typically lightweight, fine-tuning the MLP and storing it for all possible bit depths would have only a small impact on the training time and storage.

Moreover, we notice that the impact of fine-tuning diminishes (less than 0.1 dB gain in PSNR) when the bit depth gets large enough. This is reasonable as a larger bit depth tends to introduce fewer errors in truncation, which makes the glitch less visible and fine-tuning less necessary. In our approach, we selectively fine-tune for smaller bit widths between 1 and D, which further reduces the time and storage cost of fine-tuning. Here, D is a configurable parameter.

3.2 Context-Aware Serving

Context-aware serving aims to properly configure the NeRF representation depending on the user's context. It scales serving by first reducing the configuration dimension (§3.2.1) and then decoupling the serving process (§3.2.2).

3.2.1 Configuration Dimension Reduction. We reduce the configuration dimension by fixing a few configuration knobs to constant. As such, the original problem (Equation 3) can be solved more efficiently in a smaller design space, i.e., the set of all feasible configurations. To illustrate, we represent the design space of the original problem Ψ in Equation 4.

$$\Psi = \{\theta \mid \theta = (\theta_1, \theta_2, \ldots, \theta_n)\}, \tag{4}$$

where θ represents a configuration, n is the number of configuration knobs, and $\theta_i, i = 1, 2, \ldots, n$ is one of the knobs. Equation 5 illustrates the reduced design space $\Psi_{i_1,i_2,\ldots,i_k}$ after fixing configuration knobs indexed by $i_1, i_2, \ldots, i_k$ to constant.

$$\Psi_{i_1,i_2,\ldots,i_k} = \{(\theta_1, \theta_2, \ldots, \theta_n) \in \Psi \mid (\theta_{i_1}, \theta_{i_2}, \ldots, \theta_{i_k}) = \text{const}\}. \tag{5}$$

Challenge. While configuration dimension reduction is straightforward, the choice of the reduced design space is crucial and non-trivial. The reduced design space must ensure the solution derived from it is optimal, even with fewer possible configurations. In other words, the solution derived from the reduced design space, i.e., the *local solution*, must be the same as that derived from the original design space, i.e., the *global solution*. Otherwise, we might serve NeRF with the sub-optimal configuration, compromising the latency.

Smoothness lookup table. We find two properties of the model width that allow us to determine a reduced design space. Firstly, the model width is negatively correlated with the smoothness (Figure 21). Secondly, a wider MLP improves quality while minimally increasing the total size of the NeRF data representation, which is dominated by the size of geometry and feature grids, as discussed in §3.1.2. Put differently, a more complex MLP tends to have a better trade-off between quality and size, as illustrated in Figure 22(b).

Based on these observations, we can prove the *optimality of the MLP-based reduced design space*: if the model width of the MLP is set to the largest value without violating the smoothness constraint, i.e. the *critical model width*, the local solution derived with the critical model width, i.e., the *critical local solution*, is the same as the global solution.

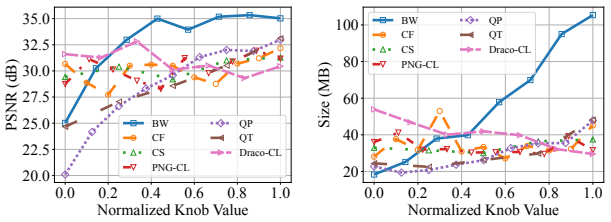
Proof. Assume there exists a configuration in the global solution with a model width unequal to the critical model width. If the global solution has a larger model width, given the first observation, it would violate the smoothness constraint. If the global solution has a smaller model width, based on the second observation, it will have a worse quality-size trade-off than that of the local solution, making it sub-optimal. Therefore, the model widths of configurations in the global solution must equal the critical model width. By the definition of the reduced design space, we can prove the critical local solution is the same as the global solution. □

According to the above proof, we can create the reduced design space with the model width. In particular, we devise a smoothness lookup table that maps the model width of the MLP to the smoothness. Then, the configuration dimension can be reduced by fixing the value of the model width that maps to the smoothness closest to the constraint. If there are multiple model widths mapped to the same smoothness, we only keep the largest one for the best quality-size trade-off. For context-awareness in serving, given a smoothness constraint, we configure the model width that maps to a smoothness closest to the constraint. The lookup table is created per device and per scene type. However, its creation is lightweight, taking less than one minute for each device.

Table 1: The configurable knobs and their ranges.

Approach	Knobs with Ranges
PNG	BD (1-8), CF (0-10), CS (0-4), PNG-CL (0-9)
Draco	QP (6-14), QT (10-14), Draco-CL (0-6)

3.2.2 Decoupled Serving. After reducing the configuration space, this part focuses on balancing the quality and size with the remaining configurations, involving feature grids and geometry. The feature grids are compressed by a widely used lossless image compression algorithm, Portable Network Graphics (PNG) [5]. PNG is



(a) Quality vs. Knobs (b) Size vs. Knobs

Figure 8: Modeling the correlation between multiple knobs and metrics is complex. The size and quality tend to increase with BW, QP, and QT but have no clear correlation with CF, CS, PNG-CL, and Draco-CL.

configurable by the bit depth (BD), compression filter (CF), compression strategy (CS), and compression level (PNG-CL). The geometry is compressed by an open-source library for 3D data compression, Draco [21], which can be adapted via the quantization bits for the position attribute (QP), the quantization bits for the texture coordinate attribute (QT), and the compression level (Draco-CL).

The name and range of configurable knobs are summarized in Table 1, where the ranges are chosen to ensure the change of a knob value does affect quality or size and does not produce significant artifacts. For the upper limits of QP and QT, we set them to the same value 14, a value beyond which does not affect quality or size. To prevent perceivable artifacts, we configure the lower limits of QP and QT to 6 and 10, respectively. For the upper limit of Draco-CL, we set it to 6, a value beyond which has no impact on quality or size. Other configurations in Draco like the quantization bits for the normal vector attribute (QN) and any generic attribute (QG) are ignored as they have no impact on the compression of geometry.

Challenge. The mapping from values of configurable knobs to quality and size is complex to model, as shown in Figure 8. While we can derive the optimal configuration for Equation 3 by exhausting all possible combinations of configuration knobs in Table 1, these configuration knobs result in a vast design space of $1.386e^6$ possible configurations. Such an exhaustive approach fails to scale to a large number of contexts a server may encounter online.

Quality-size profile. Our insight is to decouple the problem in Equation 3 into an offline MOO problem and an online searching problem. Instead of exhausting the original design space, solving the offline MOO problem allows us to avoid a large number of sub-optimal configurations in the original design space. With a reduced design space, online searching scales more easily to a large number of contexts.

Specifically, we formulate the MOO problem in Equation 6.

$$\underset{\theta\in\Psi}{\text{argmax}}\, \boldsymbol{f}(\theta) = (f_1(\theta), f_2(\theta)) = (Q(\theta), -S(\theta)), \tag{6}$$

where Ψ represents the set of feasible configurations as described in Table 1. The MOO problem jointly optimizes metrics of quality (Q) and size (S) over all feasible configurations. The solution to the MOO problem is a set of *Pareto optimal* configurations, which is also termed the *quality-size profile*. Here, Pareto optimal means the configuration is not outperformed by another configuration in one of quality and size metrics without compromising the other metric.

To formally define the quality-size profile, we introduce the *dominance* relation $\prec$ that compares the goodness of different configurations as shown.

$$\theta \prec \theta' = \begin{cases} f_i(\theta) \le f_i(\theta') & \forall i = 1,2 \\ f_i(\theta) < f_i(\theta') & \exists i = 1,2 \end{cases} \tag{7}$$

Based on the dominance relation, the quality-size profile Ω can be formally expressed as shown.

$$\Omega = \{\theta | \neg\exists\theta' s.t. \theta \prec \theta', \theta' \in \Psi\}. \tag{8}$$

Figure 9 illustrates the Pareto optimal configurations (red stars) in contrast to sub-optimal ones (black circles). The optimal configuration for different contexts (blue, orange, and green lines) can be efficiently searched by skipping sub-optimal configurations.

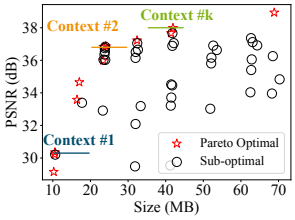


Figure 9: MOO reduces configurations by finding the Pareto optimal configurations offline.

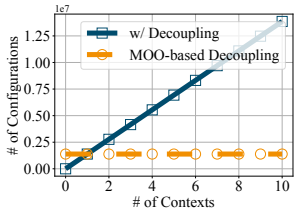


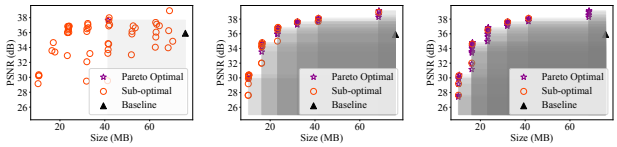
Figure 10: The decoupled approach scales better than repeatedly exploring configurations.

Figure 10 compares the number of configurations to be explored with and without decoupled serving. Decoupled serving explores $1.386e^6$ total configurations offline, which result into 50 Pareto optimal configurations (detailed in §3.3). Therefore, when the number of contexts is zero, decoupled serving explores more configurations than serving without decoupling, which involves no offline cost. However, when the number of contexts increases, the number of configurations that serving without decoupling requires drastically increases. In contrast, decoupled serving only needs to explore 50 configurations for each additional context, which scales better than serving without decoupling.

3.3 Profiling Optimizer

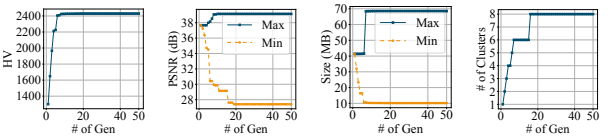
The profiling optimizer aims to make offline profiling in §3.2.2 efficient. As the relation between the design space and the metrics is complex, we leverage evolutionary algorithms that make profiling efficient with fewer evaluations of configurations. In particular, we utilize the Non-dominated Sorting Genetic Algorithm II (NSGA-II) [14], one of the most adopted evolutionary algorithms. NSGA-II starts by generating an initial population of configurations, representing a potential solution to the MOO problem. Then, these configurations are evaluated to derive objectives. Based on the evaluation, these configurations will be sorted, selected, mutated, and recombined to form the next generation of configurations, which balances exploration (the diversity of all objective space) and exploitation (the refinement of the existing Pareto front). As NSGA-II iteratively evolves more generations, the new generation better approximates the Pareto front. To further accelerate NSGA-II, we adopt two approaches: parallelism (§3.3.1) and efficient sampling (§3.3.2).

Bo Chen[1], Zhisheng Yan[2], Bo Han[2], Klara Nahrstedt[1]
[1]University of Illinois at Urbana-Champaign, [2]George Mason University



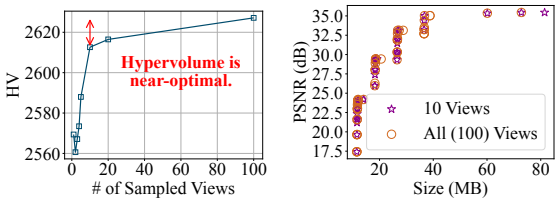
(a) N=1 (b) N=10 (c) N=20

Figure 11: More generations improve profiling.



(a) Hypervolume (b) Quality (c) Size (d) # of Clusters

Figure 12: Impact of generations on different metrics.



(a) Hypervolume vs #view. (b) Profiles with different #view.

Figure 13: No need to sample a large number of views.

3.3.1 Parallelism. Offline profiling involves evaluating a design space consisting of all possible combinations of configuration knobs. For each scene, an intuitive way is to have a single thread of NSGA-II to explore all configuration knobs of its representation jointly.

Challenge. However, even with NSGA-II, it is time-consuming to explore this design space. The most computation-intensive part is the evaluation of configurations. While the size resulting from each configuration can be measured easily, the quality must be evaluated by rendering the scene multiple times on the mobile device and comparing it with the ground truth, which is slow. To make matters worse, the profiling duration increases linearly with more possible configurations of MLPs.

MLP-based parallelism. The analysis in §3.2.1 shows that the profile can be generated from a single reduced design space based on MLP. Therefore, it is not necessary to explore all configuration knobs jointly. Instead, we can parallelize the profiling with multiple threads, each of which has a unique configuration of the MLP, which greatly speeds up profiling and scales to multiple MLP configurations.

3.3.2 Efficient Sampling. Following common practice, we set the population size of NSGA-II to 50. Figure 11 illustrates how the number of generations $N = 1, 10, 20$ improves profiling using the MobileNeRF model [12] using the "chair" scene from the realistic scene dataset [35]. Every point represents the size (x-axis) and the quality (y-axis) of one evaluated configuration. The quality is measured by PSNR across all training views between the evaluated configuration and the best-quality configuration. The best-quality configuration adopts default lossless PNG and Draco compression with the largest model width. For reference purposes, we use the profile derived after a large number of generations, 50 in our case, as the groundtruth profile (GP). Then, we can identify configurations explored in different generations that are Pareto-optimal (denoted by stars), i.e., existing on GP, or sub-optimal (denoted by circles), i.e., not in GP. For comparison, the baseline (the default implementation of MobileNeRF) is illustrated by a triangle.

Challenge. While more evolutions improve profiling by finding more Pareto-optimal configurations on the GP, it linearly increases the profiling time. In addition, to comprehensively assess the visual quality, it is natural to consider as many views as possible. However, this would create a substantial computation cost in evaluation and prolong profiling.

Early stopping and view selection. We adopt early stopping and view sampling to reduce the profiling duration.

The intuition for early stopping is that the profiling stops improving, i.e., converges, after a certain number of generations. We use several performance indicators to illustrate the convergence from different perspectives as follows.

- Hypervolume calculates the area, which is dominated by the explored Pareto-optimal configurations with respect to a reference point. In our case, we set the x-axis and y-axis of the reference point to the size of baseline and zero. The grey area in Figure 11 illustrates the hypervolume.
- Max/min visual quality measures the maximal and minimal visual quality in explored configurations.
- Max/min size measures the maximal and minimal downloading size in explored configurations.
- Cluster number measures the number of clusters in the profile, selected based on the observation that the profile tends to group in several clusters (Figure 11). It is derived using the Density-Based Spatial Clustering of Applications with Noise algorithm [16] by setting the maximum distance between two neighborhood samples to 1 and the minimum number of samples per cluster to 1.

Figure 12 exemplifies how the above performance indicators stabilize when the number of generations increases, with the same experimental setup as Figure 11. Empirically, we find 20 to be a sufficient number of generations for profiling to converge, which prevents prolonged profiling duration on generations that minimally affect the profile.

The key observation for view sampling is that the number of views does not affect the profiling result significantly as long as it is above a threshold. Figure 13(a) shows the hypervolume of profiles evaluated using all test views of the "chair" scene but derived from different numbers of training views, with other setups the same as Figure 11. We empirically find that 10 views are sufficient to get approximately the same profile as using all views for different scenes. Figure 13(b) demonstrates 10 views and 100 views produce similar profiles for the "chair". Therefore, we can sample a small number of views, e.g., 10, in profiling without compromising the performance.

4 IMPLEMENTATION

Model and training. Aiming at rendering NeRF on mobile devices and demonstrating NeRFHub's capability to handle a complex data representation, we adopt MobileNeRF [12] as the basic model, which runs on most mobile devices in real-time consisting of geometry, feature grids, and MLP. The model is implemented and trained based on Jax [6]. The original training procedure of MobileNeRF consisted of continuous training (using continuous opacity), binarized training (using quantized opacity), and post-processing (removing triangles with zero opacity) [12]. We apply shrinking-based computation adaptation (§3.1.1) with $M = 4$ and $N = 16$ to the continuous training stage, continuing the last phase of progressive training that randomly samples model width in binarized training, and apply truncation-based bitrate adaptation (§3.1.2) with $D = 4$ in the post-processing stage. Here, the values of M and N are empirically chosen to ensure (1) sufficient performance differences between different MLP configurations and (2) a slight impact of a progressive training procedure on each configuration, compared to a separate training procedure. The value of D is set to 4 as the quality drop is negligible when the bit depth is greater than 4.

Serving. The server is implemented with Flask on a Linux desktop featuring dual NVIDIA Geforce RTX 3090 Ti GPUs and an AMD Ryzen 9 CPU running at 4.95GHz. We implement the neural renderer in an HTML file using Javascript and WebGL with the THREE.js library. The renderer is fetched by the client from the server and runs on a Chrome browser.

Profiling. The client runs on the same desktop as the server. To assess the visual quality on the Chrome browser using a server without a display, we utilize the web driver from the Selenium library in Python, which can simulate the Chrome browser with the headless mode in the background.

5 EVALUATION

Dataset. We evaluate two datasets: the 8 synthetic 360-degree scenes from NeRF [35], and the 8 forward-facing realistic scenes from LLFF [34]. The rendering resolutions are 800×800 for synthetic and 1008×756 for realistic.

Hardware and networking. We consider Microsoft Surface, Dell Precision 5510 laptop, iPhone 12, and MacBook Pro for the client hardware in our experiments. The server is wired to the campus network via a 1Gb cable with 1000 Mbps NIC. The client is connected to on-campus WiFi with 550 Mbps download speed, measured by GoogleFiber [17].

Baseline. We compare our system to a baseline system that trains NeRF according to the original training procedure of MobileNeRF [12] and fetches the default NeRF representations with lossless PNG compression (for feature grids) and lossless DRACO compression (for geometry).

Metrics. Performance metrics are measured as follows. For clarity, we use the up arrow ↑ to denote the metric whose higher value means better and the down arrow ↓ otherwise.

- The latency measures the time duration between the moment a user requests a scene from the web page and the moment the scene is fully rendered, which is averaged over five requests per scene.
- The quality of a scene is measured between a view and the same view sampled from a scene fetched using the highest-quality setting: lossless PNG and Draco compression with the most complex MLP averaged over all test views specified by NeRF [35] and LLFF [34] datasets.
- The smoothness is quantified by the number of times `WebGLRenderer()` from THREE.js can be called in a second, which is averaged over five seconds after rendered.

Constraints. We configure the quality constraint via a quality tolerance value, the acceptable amount of quality drop compared to the baseline, and the smoothness constraint via a configured smoothness value, a targeted smoothness to attain. The quality constraint and smoothness constraints are defaulted to zero and 30 fps in §5.1, but adapted in §5.2.

5.1 Overall Performance

Latency. Figure 14 reports the latency of NeRFHub, compared to the default configuration, is effectively reduced on different mobile devices by 56%-66% on synthetic scenes and 26%-55% on realistic scenes. We observe a more significant reduction in 1) synthetic scenes than realistic scenes and 2) more powerful devices, e.g., MacBook Pro, than less powerful ones, e.g., Surface. The reason is that synthetic scenes involve less data to render a scene than realistic scenes (Figure 3(b)) and a more powerful device can perform ray-marching computation at a faster rate (Figure 21). Therefore, we can render using a more complex MLP with synthetic scenes or more powerful devices without violating smoothness constraints, which has better trade-offs of quality and size (Figure 22).

Size and quality. Figure 16 shows the size and quality (measured by PSNR, SSIM, and LPIPS) of NeRFHub on different mobile devices and the baseline. The size is reduced by 70%-75% on synthetic scenes and 11%-57% on realistic scenes, where more reduction is shown on more powerful mobile devices. The quality in PSNR is similar between ours and baseline with less than a 1dB difference. Although NeRFHub does not explicitly use the SSIM and LPIPS metrics, the quality degradation is less than 0.02 in SSIM and 0.01 in LPIPS.

Smoothness. Figure 15 shows the rendering speed of NeRFHub on different mobile devices. The rendering speed of NeRFHub is nearly real-time (higher than 30 fps) or as good as the baseline. This result demonstrates that our design effectively adapts the rendering speed to a value close to the constraint (30 fps). Such a capability allows NeRFHub to gain better trade-offs of quality and size without violating the smoothness constraint. Note that some less powerful devices like the Surface do not support a rendering speed of 30 fps even with the most lightweight configuration of MLP. In this case, NeRFHub maintains the same rendering speed as the baseline.

5.2 Varying Experiment Environments

Vary smoothness requirements. We evaluate how NeRFHub adapts to different configured smoothness of 15, 30, 45, and 60 fps. For illustration purposes, we fix the mobile device to iPhone 12 as it covers a wide range of frame rates from 10 fps to 60 fps within our design space. However, the observed trend in Figure 17 should apply to all mobile devices. Figure 17 shows the increase in configured smoothness increases the latency (Figure 17(a)), actual

Bo Chen[1], Zhisheng Yan[2], Bo Han[2], Klara Nahrstedt[1]
[1]University of Illinois at Urbana-Champaign, [2]George Mason University

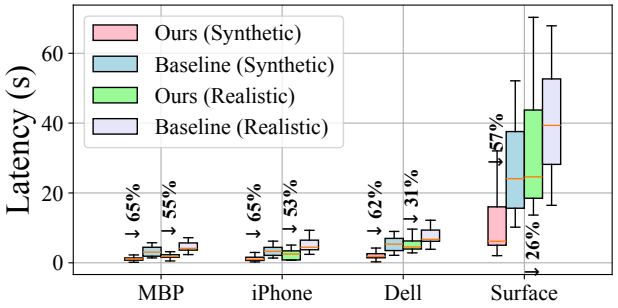
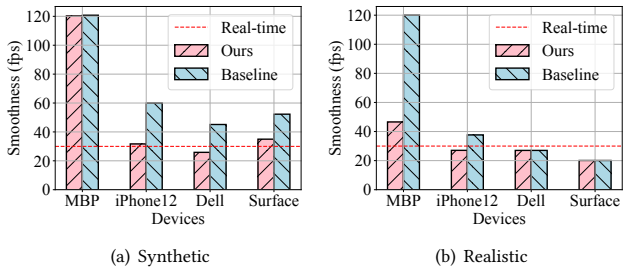
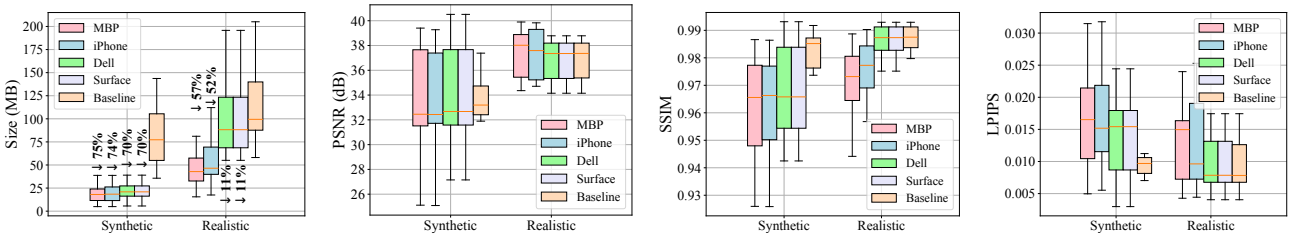


Figure 14: NeRFHub **saves the latency by 56%-66% on synthetic scenes and 26%-55% on realistic scenes.**



(a) Synthetic (b) Realistic

Figure 15: NeRFHub**'s rendering speed is nearly real-time or as good as the default configuration on different mobile devices.**



(a) Size (↓) (b) Quality (PSNR↑) (c) Quality (SSIM↑) (d) Quality (LPIPS↓)

Figure 16: NeRFHub **reduces the size on synthetic (70%-75%) and realistic (11%-57%) scenes with similar quality.**

smoothness (Figure 17(b)), and size (Figure 17(d)) while minimally affecting quality (Figure 17(c)). The intuition for increased latency is that a higher smoothness demands a less computation-intensive MLP, which must be paired with a larger size to render the scene at the same quality.

Vary visual quality requirements. We show NeRFHub adapts to different quality requirements by changing the quality tolerance from 0 to 2 dB. Figure 18 shows a higher quality tolerance reduces the latency (Figure 18(a)), the visual quality (Figure 18(b)), and the downloading size (Figure 18(c)). The intuition behind this is that the reduced quality requirement allows smaller data representation, which reduces the downloading size and the latency. It is worth noting that the quality tolerance has no impact on the rendering speed.

Vary network conditions. We vary the network conditions between a high-speed WiFi network: on-campus WiFi with 550 Mbps download speed and a normal-speed WiFi network: off-campus WiFi with 220Mbps download speed. The speed is measured by GoogleFiber [17]. Figure 19 shows 65%-71% latency reduction for the synthetic scene and 55% latency reduction for the realistic scene. A more noticeable reduction is found in the normal-speed WiFi network, as the latency is more dominated by the network transmission latency, which benefits more from a smaller size. We do not report results on slower network speeds, e.g., 4G and 3G LTE, because the latency, usually tens of seconds or minutes, is unrealistic even though the reduced latency would be more prominent.

5.3 Analysis

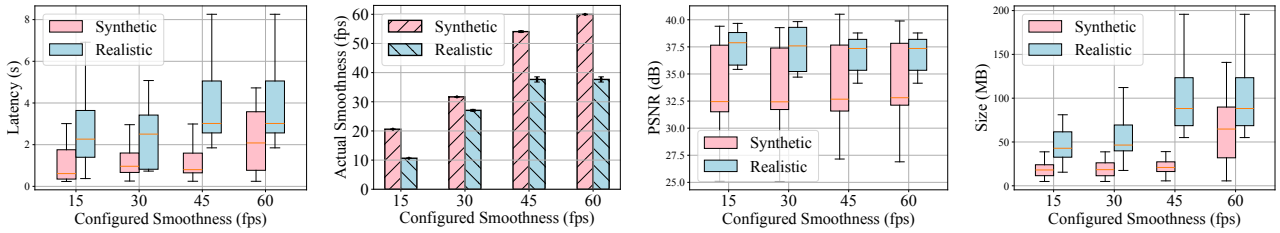
Computation and storage cost. Figure 20(a) compares the training duration of MobileNeRF (Default) and NeRFHub (Ours), averaged over all scenes. NeRFHub spends three times the duration in continuous training (CT) than MobileNeRF due to constrained progressive training (§3.1.1), almost the same duration in binarized training (BT), and twice the duration in post-processing (PP) due to selected MLP fine-tuning (§3.1.2). Overall, NeRFHub requires almost twice the training duration of MobileNeRF.

Figure 20(b) shows the profiling duration for different hidden layer channels of MLP, averaged over all scenes. The profiling duration is between 7 to 11 hours per MLP per scene. The profiling of a larger model width is more time-consuming because the rendering with them during profiling involves more computation.

Figure 20(c) contrasts the storage size of different data representation components and in total for MobileNeRF (Default) and NeRFHub (Ours). There is a substantial increase in the size of MLPs due to the storing of MLPs for selected MLP fine-tuning (§3.1.2). However, as the size of MLP is negligible compared to other components, the impact on the total size is insignificant. The sizes of the feature grids and geometry, in particular, are slightly increased due to the constrained progressive training procedure (§3.1.1).

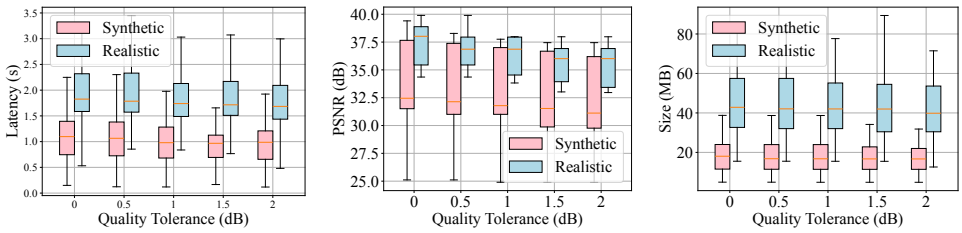
Overall, NeRFHub incurs an additional time and storage cost than a default NeRF implementation like MobileNeRF. However, the cost can be amortized by NeRFHub's capability to serve a wide range of mobile devices and diverse rendering speed and visual quality requirements.

Profiles. Figure 21 presents the smoothness on different devices, scenes, and numbers of channels, which is saved in the LUT for



(a) Latency (↓) (b) Smoothness (↑) (c) Quality (PSNR↑) (d) Size (↓)

Figure 17: NeRFHub **is adaptable to different configured smoothness.**



(a) Latency (↓) (b) Quality (PSNR↑) (c) Size (↓)

Figure 18: NeRFHub **is adaptable to different different quality tolerance values.**

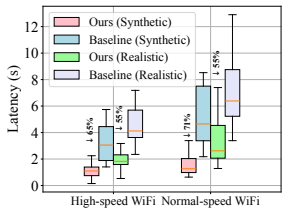
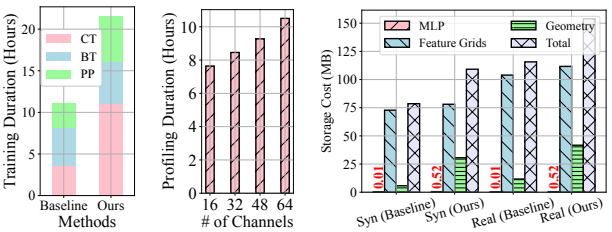


Figure 19: NeRFHub**'s reduction improves with a slower network speed.**



(a) Training (b) Profiling (c) Storage

Figure 20: The cost for context-aware NeRF serving.

smoothness adaptation (§3.2.1). Figure 22(a) shows the quality-size profile for a single model width (0.5) where the oval characterizes the distribution of the profiles and the point represents the baseline without any adaptation, which is used for quality adaptation (§3.2.2). Figure 22(b) zooms in the profile of "leaves", which describes the profile for MLPs with hidden layer channels of 16, 32, 48, and 64.

Benefits of constrained progressive training. Figure 23 demonstrates the increased quality in PSNR of NeRFHub compared to that of the baseline. We notice that the loss of visual quality is negligible with the same hidden layer channels (16). There is also a substantial increase in the quality as the model width increases, which shows that constrained progressive training (§3.1.1) effectively adapts the quality.

Benefits of selected MLP fine-tuning (SMF). Figure 24 illustrates the visual quality with and without SMF at different bit depths. SMF significantly improves the visual quality at most by 5 dB. The benefits diminish as the bit depth increases, validating the design of SMF (§3.1.2).

6 RELATED WORKS

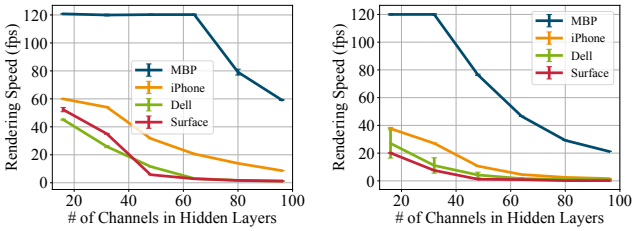
Volumetric video streaming. Volumetric video streaming delivers point cloud data over the Internet, which is relevant to our work by delivering immersive data. Numerous efforts are proposed that enhance volumetric video streaming via visibility-aware optimization [24], super resolution [56], multiview transcoding [32], and point cloud compression optimization [29]. Our work focuses on NeRF, a novel immersive data representation with many advantages (§2.1). Still, we believe the existing system optimizations on volumetric data would be complementary to ours.

Resource allocation. Resource allocation for images and video analytics has been well-studied [8, 9, 23, 28, 31, 57, 58]. Typically, existing works first generate a profile between the resource, e.g., GPU time, CPU time, or a utility function, e.g., F1 score or accuracy. This process, referred to as profiling, explores the combinations of various control knobs, e.g., resolution, sampling rate, and analytical model, and find Pareto optimal configurations. Profiling be accelerated with various techniques including parallelism [57], pruning [8], heuristics [58], and Bayesian optimization [9, 23, 28]. These techniques are considered complementary to our profiling optimizer (§3.3). Based on the profile, proper configurations are chosen to maximize the utility with constraints. In contrast, we focus on the resource allocation of a novel problem, which involves diverse knobs (MLP, feature grids, and geometry) and targets (latency, smoothness, and quality).

7 DISCUSSION

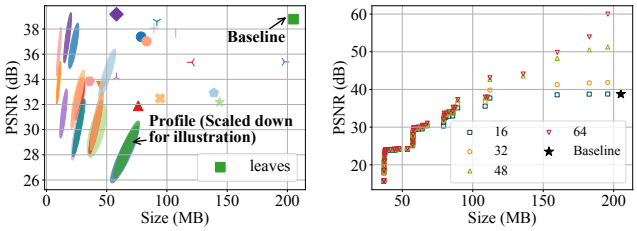
Cost for determining hyper-parameters. We heuristically determine the hyper-parameters M, N, and D in NeRFHub. The values of M and N are decided after heuristically evaluating different combinations of M and N, e.g., a grid search, each of which takes

Bo Chen[1], Zhisheng Yan[2], Bo Han[2], Klara Nahrstedt[1]
[1]University of Illinois at Urbana-Champaign, [2]George Mason University



(a) Synthetic Scene (b) Realistic Scene

Figure 21: How smoothness varies based on context.



(a) All (b) Leaves

Figure 22: Quality-size profile.

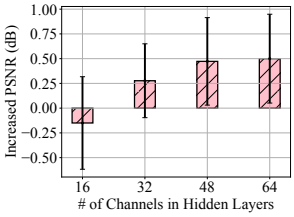


Figure 23: NeRFHub minimally affects the quality with the same hidden layer channels (16) and improves the quality with more channels.

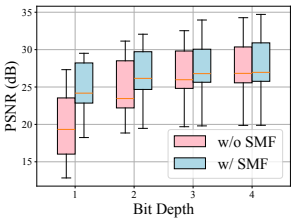


Figure 24: NeRFHub's quality gain is as high as 5 dB in medium with the bit depth of one and decreases with a higher bit depth.

roughly 20 hours. While this process may seem time-consuming, the values of M and N are shared among similar types of scenes with the same NeRF variant. The choice of D is determined via bit truncation at different depths and rendering at various angles, which is lightweight by taking less than one minute.

Generalizability to other NeRF variants. In this work, we focus on a specific variant of NeRF optimized for mobile rendering, i.e., MobileNeRF, to demonstrate the effectiveness of our framework. Still, we aim NeRFHub as a general-purpose framework that generally applies to NeRF variants that (1) employ a neural network for rendering and (2) have a multi-modal data representation. While the values of hyper-parameters, e.g., M, N, and D, are specific to the NeRF variant, e.g., MobileNeRF [12], which might not directly apply to other variants, the intuitions for choosing them are generally applicable.

Limitations of NeRF and the positioning of NeRFHub**.** NeRF at its current stage has several inherent limitations such as the long training duration, the large size of data representation, and the significant computation overhead for rendering. By applying NeRFHub to one of the NeRF variants, we are likely to experience its inherent limitations. However, eliminating the inherent limitations of NeRF is not our focus. Instead, NeRFHub aims to serve NeRF to clients with the optimal trade-offs of various metrics of interest and reliably meet the constraints set by clients.

Storage cost. Aiming at a low-latency experience, the data representations in NeRFHub with different configurations are generated before serving. Although this strategy potentially leads to a large storage space, the profile can be pruned to mitigate the storage cost. As shown in Figure 22, a certain amount of configurations have similar performance in visual quality and size. These configurations can be combined to reduce configurations on the profile, reducing the storage cost. However, this is beyond our paper.

Utility function. While we focus on minimizing latency in this paper, the optimization target in Equation 3 can generalize to other utility functions, e.g., quality and smoothness, or a weighted sum of them. NeRFHub can generalize to these utility functions easily with the decoupled approach like §3.2.2 that builds a profile offline and performs lightweight optimization online.

Quality metrics. NeRFHub focuses on the PSNR metric for quality. While this achieves acceptable performance on SSIM and LPIPS, we believe explicating adopting SSIM and LPIPS in profiling would further advance the benefits of NeRFHub regarding these metrics.

8 CONCLUSION

We design NeRFHub, a context-aware NeRF serving framework for mobile immersive applications. NeRFHub achieves context-awareness with a proactive training module that flexibly reconfigures computation and bitrate for NeRF, a context-aware serving module that configures NeRF representation based on the context, and a profiling optimizer that makes profiling for NeRF serving efficient. The evaluation shows NeRFHub significantly reduces latency of the baseline without compromising quality and smoothness, which demonstrates the effectiveness of a context-aware design.

ACKNOWLEDGMENTS

We thank our shepherd and the anonymous reviewers for their valuable feedback, which greatly improves this paper. This work was supported by NSF under the award numbers NSF OAC-1835834, NSF IIS-2140645, NSF CNS-2106592, NSF CCF-2217144, NSF CNS-1900875, NSF IIS-2140620, NSF OAC-2144764, NSF CNS-2212296, and NSF CNS-2235049.

REFERENCES

[1] 2011. Minecraft. Video Game. https://www.minecraft.net/en-us/store/minecraft-java-bedrock-edition-pc PC.
[2] 2011. Terraria. Video Game. https://terraria.org/ PC.
[3] 2015. Undertale. Video Game. https://undertale.com/ PC.
[4] Linde Virtual Academy. 2023. Is Your Frame Rate Affecting Your VR Experience? https://vr.linde.com/2022/10/06/is-your-frame-rate-affecting-your-vr-experience/
[5] Mark Adler, Thomas Boutell, John Bowler, Christian Brunschen, Adam M. Costello, Lee Daniel Crocker, Andreas Dilger, Oliver Fromme, Jean-loup Gailly, Chris Herborth, Alex Jakulin, Neal Kettler, Tom Lane, Alexander Lehmann, Chris Lilley, Dave Martindale, Owen Mortensen, Stuart Parmenter, Keith S. Pickens, Robert P. Poole, Glenn Randers-Pehrson, Greg Roelofs, Willem van Schaik, Guy Schalnat, Paul Schmidt, Andrew Smith, Michael Stokes, Simon Thompson,

Vladimir Vukicevic, Tim Wegner, and Jeremy Wohl. 2023. Portable Network Graphics (PNG) Specification (Third Edition). https://www.w3.org/TR/PNG/. Accessed: October 26, 2023.

[6] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. 2018. *JAX: composable transformations of Python+NumPy programs*. http://github.com/google/jax

[7] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. 2019. Once-for-all: Train one network and specialize it for efficient deployment. *arXiv preprint arXiv:1908.09791* (2019). https://doi.org/10.48550/arXiv.1908.09791

[8] Bo Chen, Zhisheng Yan, Haiming Jin, and Klara Nahrstedt. 2019. Event-driven stitching for tile-based live 360 video streaming. In *Proceedings of the 10th ACM Multimedia Systems Conference*. 1–12. https://doi.org/10.1145/3304109.3306234

[9] Bo Chen, Zhisheng Yan, and Klara Nahrstedt. 2022. Context-aware image compression optimization for visual analytics offloading. In *Proceedings of the 13th ACM Multimedia Systems Conference*. 27–38. https://doi.org/10.1145/3524273.3528178

[10] Bo Chen, Zhisheng Yan, Yinjie Zhang, Zhe Yang, and Klara Nahrstedt. 2024. LiFteR: Unleash Learned Codecs in Video Streaming with Loose Frame Referencing. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. USENIX Association, Santa Clara, CA, 533–548. https://www.usenix.org/conference/nsdi24/presentation/chen-bo

[11] Yaosen Chen, Qi Yuan, Zhiqiang Li, Yuegen Liu, Wei Wang, Chaoping Xie, Xuming Wen, and Qien Yu. 2022. Upst-nerf: Universal photorealistic style transfer of neural radiance fields for 3d scene. *arXiv preprint arXiv:2208.07059* (2022). https://doi.org/10.48550/arXiv.2208.07059

[12] Zhiqin Chen, Thomas Funkhouser, Peter Hedman, and Andrea Tagliasacchi. 2023. Mobilenerf: Exploiting the polygon rasterization pipeline for efficient neural field rendering on mobile architectures. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 16569–16578. https://arxiv.org/abs/2208.00277

[13] Abril Corona-Figueroa, Jonathan Frawley, Sam Bond-Taylor, Sarath Bethapudi, Hubert PH Shum, and Chris G Willcocks. 2022. Mednerf: Medical neural radiance fields for reconstructing 3d-aware ct-projections from a single x-ray. In *2022 44th Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC)*. IEEE, 3843–3848. https://arxiv.org/abs/2202.01020

[14] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE transactions on evolutionary computation* 6, 2 (2002), 182–197. https://doi.org/10.1109/4235.996017

[15] MDN Web Docs. 2023. Understanding latency. https://developer.mozilla.org/en-US/docs/Web/Performance/Understanding_latency

[16] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. 1996. A density-based algorithm for discovering clusters in large spatial databases with noise. In *kdd*, Vol. 96. 226–231. https://dl.acm.org/doi/10.5555/3001460.3001507

[17] Google Fiber. 2023. Google Fiber Speed Test. https://fiber.google.com/speedtest/. Accessed: October 26, 2023.

[18] Sara Fridovich-Keil, Alex Yu, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. 2022. Plenoxels: Radiance fields without neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 5501–5510. https://arxiv.org/abs/2112.05131

[19] Kyle Gao, Yina Gao, Hongjie He, Dening Lu, Linlin Xu, and Jonathan Li. 2022. Nerf: Neural radiance field in 3d vision, a comprehensive review. *arXiv preprint arXiv:2210.00379* (2022). https://arxiv.org/abs/2210.00379

[20] Kyle Genova, Forrester Cole, Avneesh Sud, Aaron Sarna, and Thomas Funkhouser. 2020. Local deep implicit functions for 3d shape. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 4857–4866. https://arxiv.org/abs/1912.06126

[21] Google. 2023. Draco Compression Library. https://google.github.io/draco/. Accessed: October 26, 2023.

[22] Steven J Gortler, Radek Grzeszczuk, Richard Szeliski, and Michael F Cohen. 2023. The lumigraph. In *Seminal Graphics Papers: Pushing the Boundaries, Volume 2*. 453–464. https://doi.org/10.1145/237170.237200

[23] Hongpeng Guo, Shuochao Yao, Zhe Yang, Qian Zhou, and Klara Nahrstedt. 2021. CrossRoI: Cross-camera region of interest optimization for efficient real time video analytics at scale. In *Proceedings of the 12th ACM Multimedia Systems Conference*. 186–199. https://arxiv.org/abs/2105.06524

[24] Bo Han, Yu Liu, and Feng Qian. 2020. ViVo: Visibility-aware mobile volumetric video streaming. In *Proceedings of the 26th annual international conference on mobile computing and networking*. 1–13. https://doi.org/10.1145/3372224.3380888

[25] 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. https://arxiv.org/abs/2103.14645

[26] Yang Hong, Bo Peng, Haiyao Xiao, Ligang Liu, and Juyong Zhang. 2022. HeadNeRF: A Real-Time NeRF-Based Parametric Head Model. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 20374–20384. https://arxiv.org/abs/2112.05637

[27] Yi-Hua Huang, Yue He, Yu-Jie Yuan, Yu-Kun Lai, and Lin Gao. 2022. Stylizednerf: consistent 3d scene stylization as stylized nerf via 2d-3d mutual learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 18342–18352. https://arxiv.org/abs/2205.12183

[28] Junchen Jiang, Ganesh Ananthanarayanan, Peter Bodik, Siddhartha Sen, and Ion Stoica. 2018. Chameleon: scalable adaptation of video analytics. In *Proceedings of the 2018 conference of the ACM special interest group on data communication*. 253–266. https://doi.org/10.1145/3230543.3230574

[29] Kyungjin Lee, Juheon Yi, Youngki Lee, Sunghyun Choi, and Young Min Kim. 2020. GROOT: a real-time streaming system of high-fidelity volumetric videos. In *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*. 1–14. https://doi.org/10.1145/3372224.3419214

[30] Marc Levoy and Pat Hanrahan. 2023. Light field rendering. In *Seminal Graphics Papers: Pushing the Boundaries, Volume 2*. 441–452. https://doi.org/10.1145/237170.237199

[31] Yuanqi Li, Arthi Padmanabhan, Pengzhan Zhao, Yufei Wang, Guoqing Harry Xu, and Ravi Netravali. 2020. Reducto: On-camera filtering for resource-efficient real-time video analytics. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*. 359–376. https://doi.org/10.1145/3387514.3405874

[32] Yu Liu, Bo Han, Feng Qian, Arvind Narayanan, and Zhi-Li Zhang. 2022. Vues: practical mobile volumetric video streaming through multiview transcoding. In *Proceedings of the 28th Annual International Conference on Mobile Computing And Networking*. 514–527. https://doi.org/10.1145/3495243.3517027

[33] Stephen Lombardi, Tomas Simon, Jason Saragih, Gabriel Schwartz, Andreas Lehrmann, and Yaser Sheikh. 2019. Neural volumes: Learning dynamic renderable volumes from images. *arXiv preprint arXiv:1906.07751* (2019). https://doi.org/10.1145/3306346.3323020

[34] Ben Mildenhall, Pratul P Srinivasan, Rodrigo Ortiz-Cayon, Nima Khademi Kalantari, Ravi Ramamoorthi, Ren Ng, and Abhishek Kar. 2019. Local light field fusion: Practical view synthesis with prescriptive sampling guidelines. *ACM Transactions on Graphics (TOG)* 38, 4 (2019), 1–14. https://arxiv.org/abs/1905.00889

[35] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. 2021. Nerf: Representing scenes as neural radiance fields for view synthesis. *Commun. ACM* 65, 1 (2021), 99–106. https://doi.org/10.1145/3503250

[36] Geoffrey Morrison. 2021. *Why FPS, frames per second, and frame rate matter for Xbox, PlayStation, movies and TV.* https://www.cnet.com/tech/home-entertainment/why-fps-frames-per-second-and-frame-rate-matter-for-xbox-playstation-movies-and-tv/

[37] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. 2022. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Transactions on Graphics (ToG)* 41, 4 (2022), 1–15. https://arxiv.org/abs/2201.05989

[38] Michael Niemeyer, Lars Mescheder, Michael Oechsle, and Andreas Geiger. 2020. Differentiable volumetric rendering: Learning implicit 3d representations without 3d supervision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 3504–3515. https://arxiv.org/abs/1912.07372

[39] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. 2019. Deepsdf: Learning continuous signed distance functions for shape representation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 165–174. https://arxiv.org/abs/1901.05103

[40] Sida Peng, Yunzhi Yan, Qing Shuai, Hujun Bao, and Xiaowei Zhou. 2023. Representing Volumetric Videos as Dynamic MLP Maps. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 4252–4262. https://arxiv.org/abs/2304.06717

[41] Albert Pumarola, Enric Corona, Gerard Pons-Moll, and Francesc Moreno-Noguer. 2021. D-nerf: Neural radiance fields for dynamic scenes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 10318–10327. https://arxiv.org/abs/2011.13961

[42] Christian Reiser, Songyou Peng, Yiyi Liao, and Andreas Geiger. 2021. Kilonerf: Speeding up neural radiance fields with thousands of tiny mlps. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 14335–14345. https://doi.org/abs/2103.13744

[43] MICHAEL RUBLOFF. 2023. *Google announces new Google Maps experience featuring Neural Radiance Fields (NeRFs).* https://neuralradiancefields.io/google-announces-new-google-maps-experience-featuring-neural-radiance-fields/

[44] Vincent Sitzmann, Michael Zollhöfer, and Gordon Wetzstein. 2019. Scene representation networks: Continuous 3d-structure-aware neural scene representations. *Advances in Neural Information Processing Systems* 32 (2019). https://arxiv.org/abs/1906.01618

[45] Pratul P Srinivasan, Boyang Deng, Xiuming Zhang, Matthew Tancik, Ben Mildenhall, and Jonathan T Barron. 2021. Nerv: Neural reflectance and visibility fields for relighting and view synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 7495–7504. https://arxiv.org/abs/2012.03927

[46] Gary J Sullivan, Jens-Rainer Ohm, Woo-Jin Han, and Thomas Wiegand. 2012. Overview of the high efficiency video coding (HEVC) standard. *IEEE Transactions on circuits and systems for video technology* 22, 12 (2012), 1649–1668.

[47] Towaki Takikawa, Alex Evans, Jonathan Tremblay, Thomas Müller, Morgan McGuire, Alec Jacobson, and Sanja Fidler. 2022. Variable bitrate neural fields. In *ACM SIGGRAPH 2022 Conference Proceedings*. 1–9. https://arxiv.org/abs/2206.07707

[48] TaoXi Technology. 2021. 3D Modeling Techniques for Products Based on Neural Rendering. Available at: https://www.alibabacloud.com/blog/3d-modeling-techniques-for-products-based-on-neural-rendering_598327. Accessed on 2023.

[49] Jim Thacker. 2023. Use NeRFs in Unreal Engine with Luma AI's new plugin. Available at: https://www.cgchannel.com/2023/04/use-nerfs-in-unreal-engine-with-luma-ais-new-plugin/. Accessed on 2023.

[50] Liao Wang, Qiang Hu, Qihan He, Ziyu Wang, Jingyi Yu, Tinne Tuytelaars, Lan Xu, and Minye Wu. 2023. Neural Residual Radiance Fields for Streamably Free-Viewpoint Videos. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 76–87. https://arxiv.org/abs/2304.04452

[51] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. 2004. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing* 13, 4 (2004), 600–612. https://doi.org/10.1109/TIP.2003.819861

[52] Thomas Wiegand, Gary J Sullivan, Gisle Bjontegaard, and Ajay Luthra. 2003. Overview of the H. 264/AVC video coding standard. *IEEE Transactions on circuits and systems for video technology* 13, 7 (2003), 560–576.

[53] Jelmer M Wolterink, Jesse C Zwienenberg, and Christoph Brune. 2022. Implicit neural representations for deformable image registration. In *International Conference on Medical Imaging with Deep Learning*. PMLR, 1349–1359. https://proceedings.mlr.press/v172/wolterink22a.html

[54] Jiahui Yu and Thomas S Huang. 2019. Universally slimmable networks and improved training techniques. In *Proceedings of the IEEE/CVF international conference on computer vision*. 1803–1811. https://arxiv.org/abs/1903.05134

[55] Jiahui Yu, Linjie Yang, Ning Xu, Jianchao Yang, and Thomas Huang. 2018. Slimmable neural networks. *arXiv preprint arXiv:1812.08928* (2018). https://arxiv.org/abs/1812.08928

[56] Anlan Zhang, Chendong Wang, Bo Han, and Feng Qian. 2022. YuZu: Neural-Enhanced Volumetric Video Streaming. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. USENIX Association, Renton, WA, 137–154. https://www.usenix.org/conference/nsdi22/presentation/zhang-anlan

[57] Ben Zhang, Xin Jin, Sylvia Ratnasamy, John Wawrzynek, and Edward A Lee. 2018. Awstream: Adaptive wide-area streaming analytics. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*. 236–252. https://doi.org/10.1145/3230543.3230554

[58] Haoyu Zhang, Ganesh Ananthanarayanan, Peter Bodik, Matthai Philipose, Paramvir Bahl, and Michael J. Freedman. 2017. Live Video Analytics at Scale with Approximation and Delay-Tolerance. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. USENIX Association, Boston, MA, 377–392. https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/zhang

[59] Richard Zhang, Phillip Isola, Alexei A Efros, and Eli Shechtman. 2018. The Unreasonable Effectiveness of Deep Features as a Perceptual Metric. *arXiv preprint arXiv:1801.03924* (2018). https://arxiv.org/abs/1801.03924