NeRFlow: Towards Adaptive Streaming for NeRF Videos

Rui-Xiao Zhang[1] Tianchi Huang[2] Bo Chen[1] Klara Nahrstedt[1]
[1]University of Illinois Urbana-Champaign [2]Tsinghua University

Abstract

Neural Radiance Field (NeRF) has emerged as a powerful technique for 3D scene representation due to its high rendering quality. Among its applications, mobile NeRF video-on-demand (VoD) is especially promising, benefiting from both the scalability of the mobile devices and the immersive experience offered by NeRF. However, streaming NeRF videos over real-world networks presents significant challenges, particularly due to limited bandwidth and temporal dynamics. To address these challenges, we propose NeRFlow, a novel framework that enables adaptive streaming for NeRF videos through both bitrate and viewpoint adaptation. NeRFlow solves three fundamental problems: first, it employs a rendering-adaptive pruning technique to determine voxel importance, selectively reducing data size without sacrificing rendering quality. Second, it introduces a viewpoint-aware adaptation module that efficiently compensates for uncovered regions in real time by combining pre-encoded master and sub-frames. Third, it incorporates a QoE-aware bitrate ladder generation framework, leveraging a genetic algorithm to optimize the number and configuration of bitrates while accounting for bandwidth dynamics and ABR algorithms. Through extensive experiments, NeRFlow is demonstrated to effectively improve user Quality of Experience (QoE) by 31.3% to 41.2%, making it an efficient solution for NeRF video streaming.

CCS Concepts

• **Information systems → Multimedia information systems**.

Keywords

Content Delivering, Neural Radiance Field, Video Streaming

ACM Reference Format:

Rui-Xiao Zhang[1] Tianchi Huang[2] Bo Chen[1] Klara Nahrstedt[1], [1]University of Illinois Urbana-Champaign [2]Tsinghua University . 2025. NeRFlow: Towards Adaptive Streaming for NeRF Videos. In *The 23rd Annual International Conference on Mobile Systems, Applications and Services (MobiSys '25), June 23–27, 2025, Anaheim, CA, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3711875.3729160

1 Introduction

Volumetric videos are gaining popularity due to their immersive viewing experience, with the global market projected to exceed 5000 million by 2030 [45]. Among various techniques, Neural Radiance Field (NeRF) has emerged as a leading solution since its introduction in 2020. NeRF represents 3D scenes using a neural network that maps 3D coordinates to view-dependent colors and densities, enabling high-quality multi-view synthesis [16, 57]. Compared to point clouds and meshes, NeRF offers superior photorealism and memory efficiency.

NeRF has been adopted in mobile applications for travel, medical imaging, gaming, and shopping [12, 46, 55, 56]. In these applications, users download NeRF data via mobile networks (e.g., Wi-Fi or cellular) and render it on their devices for real-time 3D interaction. Recent advances have extended NeRF from static to dynamic scenes [17, 44], paving the way for NeRF video streaming.

However, streaming NeRF videos to mobile devices in real-world networks remains challenging. Our analysis shows that network conditions still face 1) limited bandwidth and 2) temporal fluctuations, which prevent smooth streaming even for state-of-the-art (SOTA) models [58] (§3.1).

To address the above challenges, we propose NeRFlow, a novel framework for NeRF video streaming that adapts to dynamic real-world network conditions. NeRFlow's core idea focuses on two types of adaptations: bitrate adaptation and viewpoint adaptation. For bitrate adaptation, NeRFlow encodes NeRF videos at multiple bitrates rather than relying on a single version, enabling dynamic selection of the most suitable bitrate based on current network conditions. For viewpoint adaptation, it streams only the parts of the data closely related to the viewer's perspective, further reducing data volume. While these two types of adaptation are common in traditional video streaming [6, 19, 38], NeRF's implicit data representation introduces unique challenges that require tailored solutions. Specifically, we realize NeRFlow by addressing the following challenges.

▹ *In-efficient NeRF video compression:* Traditional 2D videos leverage "pixels" with explicit RGB information, which allow the encoder to accurately assess pixel importance and selectively retain key details during compression (e.g., areas containing high-frequency information [39], and areas with more brightness [35]). In contrast, NeRF videos leverage "voxels" (i.e., 3D unit) but contain neural-based features. This implicit representation makes it difficult for the encoder to identify which parts of the representation are more important, resulting in uniform compression that ultimately degrades rendering quality. This implicit representation also makes it difficult to apply previous streaming optimizations designed for VR/point clouds [25, 38, 47, 66], as their downsampling decisions are based on the explicit RGB information visible to users [19, 48].
Solution: We tackle this problem with a rendering-adaptive pruning method based on two key observations. First, although NeRF uses implicit feature representation, voxel importance can be inferred from intermediate variables in the rendering process. Second, users' viewpoints are consistently concentrated in a few specific regions across different users. Therefore, by evaluating each voxel's contribution to rendering, particularly for these concentrated viewpoints, we selectively prune less important voxels instead of compressing all voxels uniformly. This approach effectively reduces data size while preserving quality.

MobiSys '25, June 23–27, 2025, Anaheim, CA, USA

ACM ISBN 979-8-4007-1453-5/2025/06
https://doi.org/10.1145/3711875.3729160

▹ *Resource-intensive NeRF encoding:* Generating more bitrates provides finer-grained bitrate selection for ABR control, it also introduces significant computational overhead. Unlike 2D videos, NeRF videos encode 3D spatial information with multi-dimensional features for each voxel, making them considerably larger than traditional 2D videos. Moreover, encoding NeRF videos requires intensive 3D space sampling, which further increases computation time. These factors make it impractical to pre-generate multiple bitrates for NeRF videos, posing a significant burden on the server's computational resources.

Solution: We solve this problem through a novel bitrate generation framework. Specifically, to consider both bandwidth and ABR algorithm when deciding the optimal NeRF encoding bitrates, we model the bitrate generation process as an optimization problem, aiming to select the minimal set of encoding bitrates while maintaining QoE. We use the genetic algorithm (GA) to efficiently solve this problem. By building up a faithful virtual player and using the feedback of the ABR algorithm for GA iteration, our method can automatically capture the environment dynamics and find the most suitable encoding bitrates.

▹ *Artifact due to data incompleteness:* When transmitted data fails to fully cover the user's viewpoint, artifacts may occur, severely degrading the viewing experience. NeRF videos are particularly vulnerable to this issue due to their complex feature space, which encodes far more information than traditional RGB channels. Unlike explicit representations such as mesh and point clouds, where missing data can often be reconstructed through interpolation [23, 33, 36], NeRF's implicit representation lacks this capability. As a result, even the loss of a small number of voxels can significantly disrupt rendering quality and negatively impact the user experience.

Solution: To address this issue, we adopt a hybrid transmission strategy using a master frame and sub-frames. The key insight is that the importance of NeRF voxels for rendering is highly concentrated, meaning only a small number of voxels require compensation through sub-frames. This allows us to manage the overall transmission overhead, even if splitting the raw feature slightly reduces coding efficiency. Specifically, by employing a voxel selection algorithm based on rendering contributions and pre-generating a lookup table offline, we can accurately and efficiently identify the required voxels in real time.

In summary, this paper makes the following contributions:

- We analyze network traces and NeRF models, revealing that current networks cannot effectively support NeRF video streaming, motivating the need for adaptive streaming techniques (§3).
- We present `NeRFlow`, a novel framework for NeRF video streaming under dynamic network conditions. By addressing key algorithmic and system challenges through three core modules, `NeRFlow` reduces data size, maintains user experience, and balances QoE and encoding overhead (§4).
- We evaluate `NeRFlow` through comprehensive experiments, showing that it improves QoE by 31.3% to 41.2% compared to the baseline (§5).

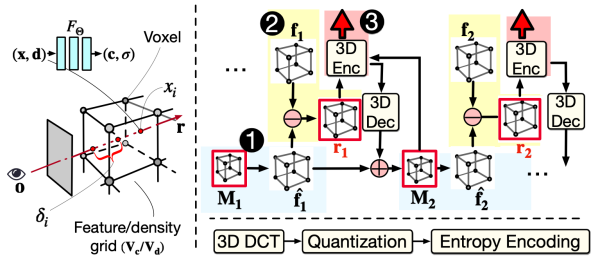


Figure 1: Left: frame-level NeRF rendering; Right: intergrating frame-level representation to NeRF video (better viewed in color).

2 Background

NeRF video represents volumetric video using implicit multi-dimensional features instead of explicit RGB values [16, 43, 57], and this allows NeRF video to model complex visual effects such as transparency and dynamic lighting. NeRF video generation involves two steps: 1) generating a NeRF for each frame as a static scene, and 2) integrating multiple frames into a dynamic video by leveraging temporal similarity.

Representing each frame with NeRF: NeRF represents each frame of a volumetric video using an implicit neural network (NN). When provided with the input coordinate $\mathbf{x} \in \mathbb{R}^3$ and the viewing direction $\mathbf{d} \in \mathbb{R}^2$, the NN model F_Θ can generate a density $\sigma(\mathbf{x})$ and color $\mathbf{c}(\mathbf{x}, \mathbf{d})$, i.e., $F_\Theta : (\mathbf{x}, \mathbf{d}) \dashrightarrow (\mathbf{c}, \sigma)$. Given a ray $\mathbf{r} = \mathbf{o} + t\mathbf{d}$ casting from the viewpoint $\mathbf{o} \in \mathbb{R}^3$, the pixel color $\hat{C}(\mathbf{r})$ can be calculated by accumulating the density and color of the sampling points (denoted as x_i) along the ray:

$$\hat{C}(\mathbf{r}) = \sum_{i=1}^{N} T_i \cdot \alpha_i \cdot \mathbf{c_i} \tag{1}$$

$$\alpha_i = 1 - exp(-\sigma_i \delta_i), \qquad T_i = \prod_{j=1}^{i-1} (1 - \alpha_j) \tag{2}$$

in which σ_i is the distance between adjacent sample points, T_i denotes the accumulated transmittance when reaching point x_i, and α_i means the probability that the ray terminates at point x_i. Instead of using a fully implicit representation, which suffers from slow training and inference due to heavy neural network querying, recent work has proposed combining a voxel-based structure to improve learning and rendering [4, 53, 60]. A voxel is a 3D unit, like a pixel in 2D space, and they are organized into grids. In traditional computer graphics, voxels store explicit data like color and density [13], but in NeRF, they store implicit features. With a density grid $\mathbf{V}_\sigma$ and a color feature grid $\mathbf{V_c}$, the density σ and color $\mathbf{c}$ will be represented as:

$$\sigma = interp(\mathbf{x}, \mathbf{V}_\sigma), \quad \mathbf{c} = F_\Theta(interp(\mathbf{x}, \mathbf{V_c}), \mathbf{d}) \tag{3}$$

where $interp(\cdot)$ means the tri-linear interpolation function on the grids. We have also illustrated the process in Figure 1 (see the left part). After getting the rendered results, we can calculate the difference between the ground truth and update F_Θ, $\mathbf{V}_\sigma$, $\mathbf{V_c}$ through backpropagation. Therefore, the data structure of NeRF representation usually contains three parts: the obtained tensor with multi-dimension features (i.e., $\mathbf{V}_\sigma$, $\mathbf{V_c}$) and an MLP (i.e., F_Θ).

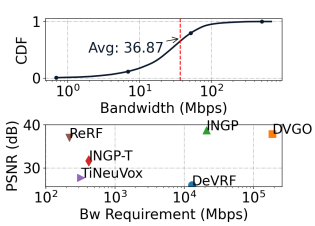


Figure 2: Network capacity v.s. NeRF models.

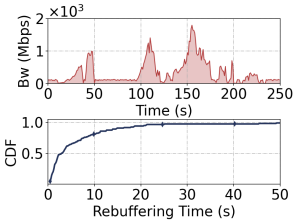


Figure 3: The impact of temporal dynamics in networks.

Integrating frames into video: After obtaining per-frame feature/density grids (for simplicity, we use $\mathbf{f_t} = \{\mathbf{V}^t_\sigma, \mathbf{V}^t_c\}$), we can then compress the data size by generating per-frame motion $\mathbf{M_t}$ and residual $\mathbf{r_t}$ through spatial-temporal coherency [28, 58]. Similar to 2D video, motion $\mathbf{M_t}$ in NeRF video represents the movement of corresponding voxels between adjacent frames, which is a grid. Notably, since NeRF operates in feature space, $\mathbf{M_t}$ is usually derived together with the raw feature (i.e., $\mathbf{f_t}$) through training process [51, 58]. As denoted in Figure 1, encoding NeRF video has three main steps: i.e., 1) predicted frame generation, 2) residual calculation, and 3) 3D data encoding. Predicted frame is generated by warping previous feature grid $\mathbf{f_{t-1}}$ into the current frame $\mathbf{f_t}$ through sampling $\mathbf{M_t}$; residual $\mathbf{r_t}$ is calculated by comparing the difference between the predicted frame and the real frame; 3D data encoding is used to convert generated residual/motion into bit streams.

We also give a detailed example in Figure 1. The first frame is encoded independently as a key frame without a motion grid. Starting from the second frame ($\mathbf{f_1}$), we predict the frame ($\hat{\mathbf{f_1}}$) using the previous frame's reconstruction and the trained motion grid (step 1). The residual $\mathbf{r_t}$ is then calculated as the difference between the actual frame ($\mathbf{f_1}$) and the predicted frame ($\hat{\mathbf{f_1}}$) (step 2). The motion grid ($\mathbf{M_t}$) and residual ($\mathbf{r_t}$) are more compact than raw frame data [58] and are encoded using a 3D encoder (DCT, quantization, entropy encoding) to produce multiple bitrates by varying quantization parameters (step 3). Larger quantization parameter leads to higher compression ratio but more quality degradation. 3D DCT is similar to traditional DCT, but it extends the transformation into three dimensions instead of two. Therefore, unlike the 2D DCT, which operates on image frames, the 3D DCT processes a volumetric block, capturing correlations across spatial and volumetric dimensions. This enables more efficient compression by exploiting redundancy within the volume, making it particularly useful for volumetric data compression. Decoding reverses these steps with entropy decoding, de-quantization, and 3D IDCT. We have also shown the components of 3D encoder in Figure 1. To clarify and distinguish with 3D encoder, we denote the encoder that includes residual calculation as **NeRF encoder**.

3 Motivation and Challenges

3.1 Streaming NeRF Videos in the Wild is Non-trivial

Although NeRF representation is promising, the following characteristics of current network conditions prevent efficient NeRF video streaming.

Limited bandwidth capacity. We first use a public dataset [63] to measure real-world bandwidth distribution. This dataset is collected by a product-level web player, which supports users watch VoD across different platforms. This player has been used by millions of users under mixed network conditions around the world in the past few years. Therefore, it can be regarded as a good sample for current network conditions. The results are shown in Figure 2 (see the top sub-figure). We can see that the average bandwidth value is about 36.87 Mbps, and for most of the traces (over 94%), the bandwidths are below 100 Mbps. Actually, this capacity cannot handle NeRF videos. For demonstration, we further investigate the data sizes and reconstruction performance of various NeRF models, including static-based models (i.e., representing each frame separately, and no integration from frames to video) like DVGO [51] and INGP [44], and dynamic-based models (i.e., leveraging temporal coherency to integrate frames into video) like DeVRF [37], INGP-T [9], TiNeuVox [14], and ReRF [58]. Specifically, we calculate the average bandwidth requirement for each model at 30 FPS. The results are shown in Figure 2 (see the bottom subfigure). We can see that the dynamic-based models can achieve smaller model sizes but may suffer from performance issues. E.g., the bandwidth requirement for INGP and DVGO is over $2{\times}10^4$ Mbps, and their PSNR is greater than 37dB, while INGP-T and TiNeuVox can compress the bandwidth to about 200-300 Mbps, but their PSNRs only achieve 27-30dB. At the same time, we can also see that the SOTA model ReRF is quite promising: it reduces the bandwidth by over 100 times (to about 200 Mbps) with only a 1dB PSNR degradation compared to the best static model (i.e., INGP). However, ReRF still remains too large to be streamed under real-world network conditions.

Significant temporal dynamics. Only considering network capacity is insufficient, as real network bandwidth is also equipped with temporal dynamics. In Figure 3 (the top sub-figure), we arbitrarily select a trace from our dataset as an example and present how bandwidth varies over time. We can see that between 125 and 175 seconds, the bandwidth peaks at nearly 1800 Mbps but drops below 20 Mbps. These fluctuations can significantly degrade performance when transmitting NeRF video at a single bitrate. For illustration, we set up an experimental scenario to simulate a user downloading a NeRF video encoded with ReRF frame-by-frame from the server. Specifically, to better indicate the influence of temporal dynamics, we select all 5G traces to make sure the average capacity is enough. We monitor the user's player buffer and record the user's rebuffering time (i.e., the time interval when the buffer in the player's device is empty until new data is downloaded). We present the rebuffering time distribution across all network traces in Figure 3 (see the bottom sub-figure). We can see that even for 5G traces, the user can still suffer rebuffering events across almost all traces. In addition, more than 20% of the traces have a rebuffering time exceeding 10 seconds. This is expected: when the network deteriorates, the user cannot download frames that match the current conditions, which causes the buffer to run empty.

The above observations highlight the limitations of current networks for NeRF video transmission and motivate the use of adaptation techniques to enhance streaming. Specifically, bitrate adaptation can be employed to select the appropriate bitrate to adapt to network conditions, while viewpoint adaptation further reduces

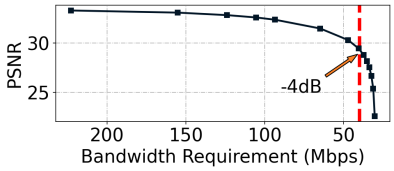


Figure 4: NeRF encoder suffers quality degradation.

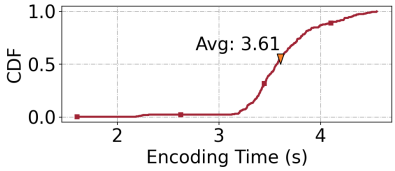


Figure 5: NeRF video encoding is resource-intensive.

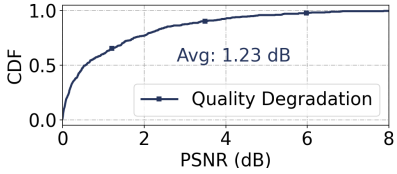


Figure 6: NeRF representation is sensitive to data incompleteness.

data volume by transmitting only the data relevant to the predicted viewpoint.

For bitrate adaptation, we will include two main processes: 1) **bitrate ladder construction** (i.e., encode raw video into multiple bitrates) and 2) **bitrate selection** (i.e., use certain ABR algorithm to select the proper bitrate based on network conditions). For viewpoint adaptation, we will also include two main processes: 1) **viewpoint prediction** (i.e., estimate the user's expected viewing region), and 2) **foveated streaming** (i.e., transmit only the voxels contributing to the predicted viewpoint.) For these processes, bitrate selection for NeRF video can directly leverage existing ABR algorithms for 2D videos, as both types can be treated as "media files"; similarly, viewpoint prediction for traditional volumetric videos like point-cloud can also be seamlessly extended to NeRF video, as users interact with the rendered images rather than the underlying data format, making viewing behavior independent of how the pre-rendered data is represented. However, bitrate ladder construction and foveated data transmission can be quite challenging as NeRF videos are represented in features and NNs.

3.2 Challenges of Bitrate Adaptation and Viewpoint Adaptation for NeRF Video

First, NeRF encoder suffers from quality degradation. As shown in Figure 1, NeRF videos store implicit features instead of explicit RGB, which prevents the encoder from identifying important voxels, forcing uniform compression and leading to a loss of quality. To demonstrate this, we examine the SOTA model ReRF. ReRF follows the NeRF encoder pipeline shown in Figure 1: i.e., first calculate motion and residual, and then compresses them using 3D encoder. We test ReRF with different QPs and present the results in terms of PSNR and bandwidth requirements. The results are shown in Figure 4. We can see that although ReRF can reduce the bitrate from a maximum of 250 Mbps to less than 25 Mbps when using smaller QPs, the corresponding loss in rendering quality is significant. For instance, when the bitrate is reduced to around 40 Mbps (which is the average bandwidth level in our dataset, see Figure 2), the quality degradation exceeds 4 dB.

Second, constructing bitrate ladders for NeRF videos is resource-intensive. From a transmission perspective, generating as many bitrates as possible is ideal, since it enables the ABR algorithm to do fine-grained control in response to network dynamics. However, this approach is impractical because NeRF encoding is far less efficient than traditional 2D video encoding, and generating as many bitrates can introduce significant server cost. There are two main reasons: First, the data size is enormous. NeRF videos store 3D spatial content, making them much larger than traditional 2D videos. Additionally, implicit feature channels are significantly higher than explicit RGB channels (e.g., ReRF has 13 channels for each voxel.). Second, generating a predicted frame based on the motion grid involves 3D space sampling, which is quite time-consuming [58]. For demonstration, we test the encoding time per NeRF frame and present the results in Figure 5. As we can see, encoding a single frame takes over 3 seconds on average, which is more than 100 times slower than 2D video (as comparison, H.265 achieves 33 ms encoding per frame even for 4K videos). It is also notable that as frames are interdependent (i.e., encoding the next frame requires completing the encoding/decoding of the previous frame see Figure 1), parallel processing is difficult to apply.

Third, NeRF videos are more sensitive to data incompleteness. When performing viewpoint-based adaptation, all data within the user's viewpoint should be available; otherwise, artifacts may appear, and negatively impacts the user experience. Unfortunately, NeRF videos are particularly sensitive to this issue due to their implicit feature representation. For illustration, we analyze how quality degrades when we randomly drop only one small 8x8x8 cube from each frame of a NeRF video (each frame contains 270*270*280 voxels). We test across all video frames provided by [58], and the results are shown in Figure 6 (bottom sub-figure). We can see that the average rendering quality drops by more than 1.2 dB, with over 20% of frames experiencing a reduction of more than 2 dB. This sensitivity primarily stems from the rich information contained in NeRF's voxels. On the one hand, the feature dimensions in NeRF are significantly higher compared to traditional RGB representations; on the other hand, the rendering process involves interpolating voxel data, meaning a single voxel can influence the rendering results of multiple pixels. Moreover, unlike explicit representations such as 360-degree videos or point clouds, NeRF's implicit features lack spatial continuity, making it impossible to compensate for missing data through interpolation or estimation based on nearby voxels.

In summary, to apply bitrate and viewpoint adaptation, we still need to 1) increase the compression quality for the NeRF encoder, 2) decrease the server cost by carefully determining how many bitrates we will generate, and 3) ensure all voxels within the viewpoint should be transmitted.

4 System Overview

Figure 7 illustrates the transmission structure of our system, comprising offline and online procedures. During the offline phase, the server pre-processes raw NeRF videos to generate all necessary data (i.e., bitrate ladder construction). In the online phase, the client selects the appropriate bitrate (bitrate selection), and the server

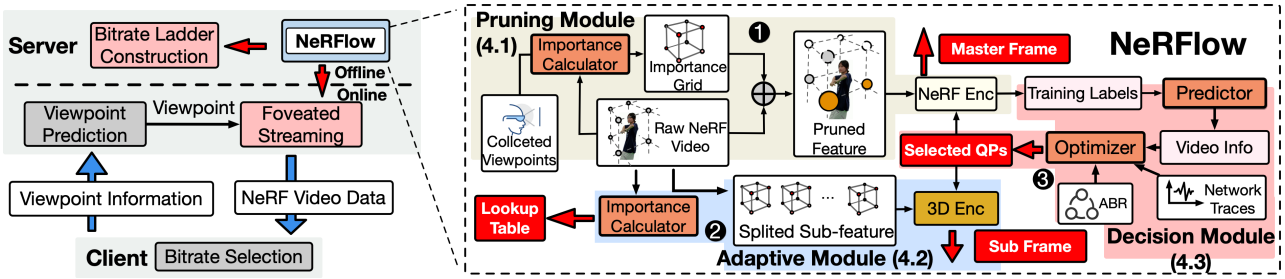


Figure 7: System overview of NeRFlow, which consists of a pruning module, a decision module, and an adaptive module.

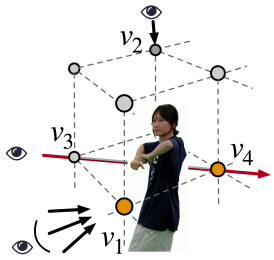
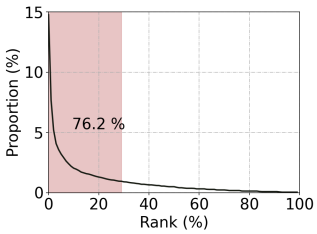
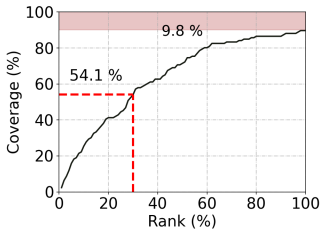


Figure 8: Ray-level and sample-level importance.



(a) Single user viewpoint distribution



(b) Viewpoint overlap across users

Figure 9: Viewpoints are concentrated in certain regions.

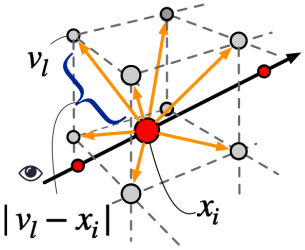


Figure 10: Assigning importance to voxels.

determines which specific data to transmit based on the predicted viewpoint (foveated streaming). NeRFlow is central to this system, managing bitrate ladder construction and foveated streaming. It achieves high compression efficiency, viewpoint adaptability, and low server overhead through three core modules: *Pruning Module*, *Adaptive Module*, and *Decision Module*.

The pruning module calculates voxel importance via an *Importance Calculator* to generate an *importance grid*, which is used to prune less important voxels and produce a compact *Pruned Feature*. The adaptive module splits raw features into sub-features for data-level flexibility and builds a lookup table for viewpoint-based online adaptation. The decision module reduces encoding overhead by selecting a limited set of QPs instead of using all possible ones. The pruned features and sub-features are then encoded as master frames and sub-frames for transmission.

4.1 Rendering-adaptive Pruning

Inspired by recent efforts in NeRF compression [9, 34], our goal for pruning module is to compresses NeRF video size while preserving visual quality. Its approach relies on two key observations. First, while voxel importance cannot be directly assessed due to NeRF's implicit features, it can be inferred from its contributions to rendering effects. This importance is evaluated at two levels. At the ray level, voxels that contribute to more rays are deemed more important, as the rendered result depends on the ray $\mathbf{r}$, according to Eq.(1). At the sample level, even along the same ray, voxel contributions vary. Based on Eq.(1) and Eq.(2), the contribution of each sampled point x_i is determined by $T_i \cdot \alpha_i$, meaning samples with higher $T_i \cdot \alpha_i$ play a more critical role in the final rendering.

To illustrate the two levels of importance, we provide an example in Figure 8. At the ray level, voxel v_1 is more important than v_2 since it is utilized by three rays, whereas v_2 is used by only one. At the sample level, for a ray crossing voxels v_3 and v_4 (shown as a red line), if $T_4 \cdot \alpha_4$ is larger than $T_3 \cdot \alpha_3$, then v_4 is considered more important than v_3. These two levels allow us to quantify each voxel's contribution to rendering effects.

Our measurements also demonstrate the potential for further compression by leveraging heterogeneous importance, which leads to our second observation: the majority of users' movements are concentrated in certain regions. Specifically, we use an IRB-approved dataset released by [62]. This dataset records gaze data from 10 users watching volumetric videos with a HoloLens. Each trace records the location (i.e., x, y, and z) and orientation (i.e., yaw and pitch) for over 2000 frames. We first partition the space into non-overlapping regions by discretizing $(x, y, z, yaw, pitch)$. Then we randomly select one user and sort the regions in descending order based on the number of this user's viewpoints that are located within them. The results are shown in Figure 9(a). We can see that the user's viewpoints are highly skewed: over 76% gaze trajectories are located in the top-30% regions. At the same time, we also investigate the region preference across users. Specifically, we split user traces evenly into training and testing sets and examine the overlap of regions as we vary the selection from the top 1% to the top 100%. The results are shown in Figure 9(b). We can find that these regions are quite consistent across users. E.g., for top-30% regions, there are over 54% regions are identical in training set and testing set. This is reasonable, as video content inherently provides heterogeneous information and naturally varies in its attractiveness to users (e.g.,

in a dancing scene, users are more likely to focus on the front view rather than the top).

In fact, these two observations allow us to compress NeRF videos more effectively: on the one hand, different viewpoints inherently hold varying significance to users, and the first level of importance naturally captures this user preference; on the other hand, unlike previous methods that uniformly compress voxel features, our approach uses the second level of importance to apply varying compression levels to different voxels, resulting in better overall rendering quality.

Then we give the details of how we calculate the importance. Formally, given a sample x_i, we can compute its importance score based on Eq.(1) and Eq.(2):

$$I^{x_i} = T_i \cdot \alpha_i \tag{4}$$

Then, this importance score will be assigned to its neighboring voxels by calculating the distance between the sample and the voxels (on normalized grid interval) in coordinate space, and the larger the distance the smaller the assigned value:

$$I_l^{x_i} = (1 - |v_l - x_i|) \cdot I^{x_i} \tag{5}$$

in which v_l are the voxels falling in the neighborhood of x_i and $|v_l - x_i| \leq 1$ (as shown in Figure 10). Through the aforementioned equations, we obtain the sample-level importance, and we then only need to accumulate all sampled rays (i.e., ray-level importance) to get the final importance:

$$I_l = \sum_{x_i \in \mathbb{X}} I_l^{x_i} \tag{6}$$

where $\mathbb{X}$ denotes the samples of all rays. Through the above process, we will generate an importance grid, and the value of each voxel records the actual contribution to the rendering results. After obtaining the importance scores of all voxels, we can then sort them in an ascending order. Specifically, we can define the cumulative function as:

$$F(\theta) = \frac{\sum I_l \cdot \mathbf{1}}{\sum I_l} \tag{7}$$

in which $\mathbf{1}$ is the indicator and $\mathbf{1} = 1$ only when $I_l < \theta$. Here θ is a hyper-parameter determined as needed. This function defines the cumulative distribution of voxel importance scores, and we can use this function to analyze different videos. Take the video provided in [58] as the example, we set $\theta = 0.99$ (i.e., we want to find the voxels contributing top 99% importance) and analyze the number of such important voxels for all frames. The results are shown in Figure 11. We can see that the distribution of voxel importance is highly skewed: i.e., we find that the top 99% contribution is made by only 0.55% voxels on average, and this highly skewed phenomenon is consistent across all frames based on the CDF. To demonstrate that our pruning method has minimal impact on quality, we choose one direction and prune out the less important voxels with $\theta = 0.99$. We compare the rendered images with and without pruning. The results are shown in Figure 12. We can see that our method well preserves visual quality (the size is reduced by more than 50%.). Notably, NeRFlow's compression efficiency stems from both ray-level and sample-level optimizations. This ensures that while increased viewpoint diversity may reduce ray-level benefits (e.g., in larger scenes with more objects), the advantages at the sample level remain (as demonstrated in §5.3). Additionally, collecting

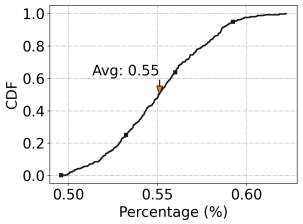


Figure 11: The highly skewed distribution of voxel importance.

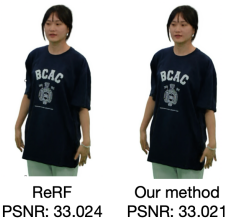


Figure 12: The rendering results before/after pruning.

viewpoints from users incurs minimal overhead, as each data entry only involves frame index and viewpoint. Privacy concerns can also be mitigated through anonymization and de-identification, similar to how browsing traces are handled in recommendation systems [22].

4.2 Viewpoint-aware adaptation

Rendering-adaptive pruning can cover most user viewing traces. However, due to individual user variability, some user viewpoints still fall outside the calculated important regions. As shown in Figure 9(b), even with all regions from the training set, 9.8% of regions in the testing set remain uncovered. Given NeRF's high sensitivity to data incompleteness (Figure 6), it is crucial to bridge this gap with a compensatory approach.

To address this issue, we propose an *Adaptive Module*. The core idea is to split the raw feature into multiple small, non-overlapping sub-features in addition to generating the pruned feature through the importance calculator. The pruned feature, referred to as the *master frame*, serves as the primary data for user viewpoints, while the sub-features, encoded as *sub-frames*, act as backups to compensate for regions not included in the master frame (as illustrated in Figure 13). This dual-frame approach ensures robust coverage of user viewpoints while maintaining efficient data transmission.

Offline data preparation. Master frames are generated by encoding pruned features through NeRF encoder (i.e., Figure 1), and we need to determine how to generate these sub-frames. There will be two straightforward ideas. The first one is to directly use and transmit raw feature without encoding. However, this is impractical as uncompressed features are extremely large (each frame is 2.2 GB). Even if the sub-frame contains only a small portion of the data, its size would still exceed that of the master frame, negating the benefits of our pruning efforts described in §4.1. The second idea is to leverage NeRF encoder to compress sub-frames. However, this approach introduces significant pre-encoding time (see Figure 5), contradicting our goal of reducing the encoding overhead for server. To address this problem, we decide to only use 3D encoder: i.e., we remove deform/residual calculation. This approach has two benefits: on the one hand, the per-frame compression time is significantly reduced since no feature space sampling is required. More importantly, it supports parallel processing as each sub-feature is compressed independently (i.e., we do not need to encode/decode to calculate residual). To demonstrate these benefits, Figure 14(a) compares the encoding time of 3D encoder with NeRF encoder. We

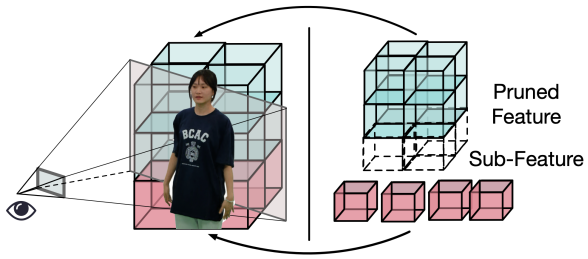
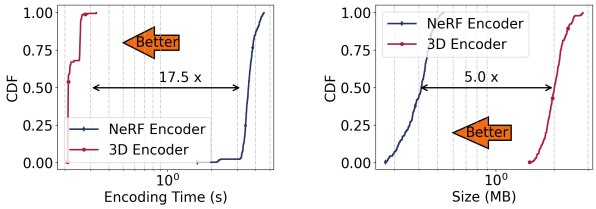


Figure 13: Using sub-features to compensate for uncovered part of the viewpoint.



(a) Encoding time (b) Frame size

Figure 14: Comparing 3D encoder and NeRF encoder in terms of time and size.

can see that 3D encoder achieves a 17.5x speedup in per-frame encoding time. Notably, a key drawback of removing residual/motion calculation is the reduced compression efficiency, as it fails to exploit temporal redundancy between frames: Figure 14(b) compares the CDF of the frame sizes for 3D encoder and NeRF encoder. We can see that the frame size will be enlarged about 5x. However, this inefficiency of encoding sub-frames only has minimal impact on the transmitted data volume: on one hand, the master frame (i.e., pruned feature) already integrates most of the viewpoints when calculating the voxel importance; on the other hand, the distribution of voxel importance is highly skewed (i.e. Figure 11), therefore there will only be a small number of voxels need to be transmitted. We have also demonstrate this in our evaluation section. It is notable that the sub-feature should not be too small, as it will make 3D DCT less efficient to characterize data redundancy. Based on our experiments, we set the sub-feature size to 8×8×8 voxels.

Online transmission control. After generating and encoding sub-frames offline, we determine which to transmit based on users' viewpoints (i.e., online transmission control). Traditional volumetric video methods (e.g., point clouds) often use frustum-based approaches [19, 38], which transmit sub-frames containing voxels within the viewpoint frustum. However, this approach is too coarse-grained for NeRF videos, as it selects too many voxels, resulting in excessive sub-frames and high transmission overhead. To address this, we leverage the importance calculator from 4.1. Using Eq.(7), we identify the top 99% of voxel indices relevant to the current viewpoint and select the corresponding sub-frames. This method significantly reduces the number of selected voxels without compromising rendering quality. As shown in Figure 15,

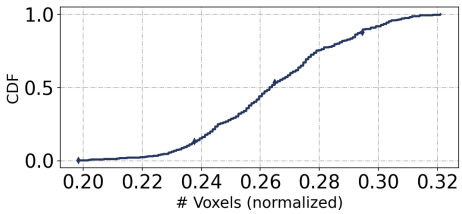


Figure 15: The voxel number compared with frustum method.

our approach selects 70% fewer voxels on average compared to the frustum-based method.

Notably, real-time computation for Eq.(7) is impractical (e.g., in 30 FPS). Therefore, we propose to offline generate a lookup table: i.e., we discretize the overall space into regions based on (x, y, z, yaw, pitch) and pre-calculate the top-99% voxel indices for each region using Eq.(7). Then the lookup table uses regions as keys and stores the corresponding top-99% voxel indices as values. We acknowledge that computing the lookup table may introduce additional overhead; however, this can be mitigated by using a coarser space division interval.

Since there have been plenty of methods focused on viewport prediction, we do not dive deep on this problem and directly use the method from [38], which uses several linear regression models to independently predict each dimension of viewpoint. our online adaptation logic is outlined as: we will get the required voxels based on predicted users' viewpoint through lookup table; then we will calculate the voxels not covered by the master frame, and transmit these voxels through sub-frames.

4.3 QoE-aware Bitrate Generation

Problem Formulation: Considering the encoding overhead of NeRF videos (see Figure 5), we aim to select the minimum number of QPs without compromising the users' experience. Mathematically, suppose the number of total QP settings is E (in our settings, QP ranges from 1 to 100, i.e., $E = 100$), then we can formulate the problem as

$$\min \sum_{qp=1}^{E} I_{qp} \tag{8}$$

$$\text{s.t.} \quad I_{qp} = \{0, 1\} \quad (qp = 1, \cdots, E). \tag{9a}$$

$$\sum_{n \in \mathbb{N}} QoE(\bigcup I_{qp}, n) \geq C \tag{9b}$$

where I_{qp} is a binary value indicating whether we choose QP=qp; $\mathbb{N}$ denotes networking environment; $QoE(\cdot)$ represents the user's quality of experience. Modeling QoE for NeRF video is out of this paper's scope, so we use the QoE definition from 2D video as an example in this paper (notably, our method can be adapted to other QoE definitions): i.e., suppose the NeRF video contains L frames, then the QoE can be represented as:

$$QoE = \zeta \sum_{i=1}^{L} q_i - \xi \sum_{i=1}^{L} R_i - \omega \sum_{i=1}^{L-1} |q_{i+1} - q_i| \tag{10}$$

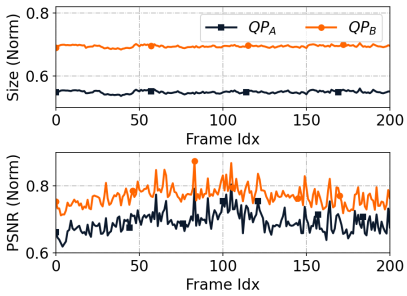


Figure 16: Variation of size and PSNR across frames.

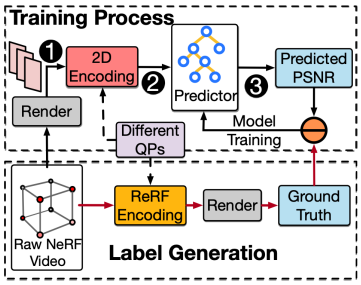


Figure 17: Process of how to get the PSNR predictor.

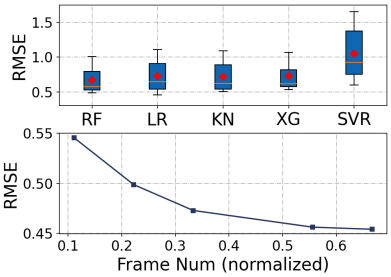


Figure 18: Prediction results for different models.

i.e., the weighted sum of quality q_i (here we use PSNR), rebuffering time R_i, and smoothness when downloading all frames. Notably, R_i relates to the bandwidth (denoted as C_i), current buffer length (denoted as B_i), and also the frame size (we denote it as s_i) [50, 64].

Designing an algorithm for this problem is non-trivial due to the following two aspects: on the one hand, solving the above problem requires knowing the PSNR and the size of the NeRF video at different QPs in advance (i.e., q_i and s_i). However, these values are typically available only after encoding, which contradicts our goal of minimizing pre-encoding overhead. On the other hand, even if we know these values, solving the optimization problem remains challenging as QoE depends on various factors (e.g., ABR algorithm and network distribution), which is hard to predict. Moreover, the solution space is huge (i.e., there will be $E!$ possible QP combinations), which makes the problem even more complex. To address the above concerns, we propose a predictor to characterize video features and an optimizer to solve the optimization problem. Specifically, the predictor provides necessary information to the optimizer.

Predictor: Formally, we want to predict the PSNR q^i_{qp} and data size (denoted as s^i_{qp}) of each frame i for all possible QPs (i.e., $qp \in E$). Fortunately, predicting sizes is relatively manageable. In Figure 16 (see the top sub-figure), we present how data sizes vary along the frames for different QPs, and the values are normalized by the raw video size. We can see that the normalized data size is quite stable across frames. E.g., with QP_A, the encoded frame size consistently accounts for about 69% of the raw video size. This actually indicates that for each QP, we only need to calculate the frame sizes of the first several frames, and then directly apply these normalized data sizes to the remaining frames.

Despite the stability in data size, PSNR does not exhibit the same consistency. For illustration, Figure 16 also shows the normalized PSNR values of two QPs across all frames (see the bottom sub-figure). As denoted, the values vary significantly. E.g., for QP_B, the normalized PSNR fluctuates over 20% (i.e., from 0.6 to 0.8). The results indicate that we cannot simply encode the first few frames and reuse their normalized PSNRs for the remaining frames. This phenomenon is explainable: the video inherently has temporal dynamics, which can cause variations in the NeRF's quality across different frames (E.g., highly dynamic scenes may lead to lower NeRF video quality).

To solve this problem, we use three steps to predict PSNRs. We first render multiple 2D images for each frame based on the viewpoints and the raw NeRF video (we denote the images rendered at frame i as P^i_{raw}), for each of which we can calculate its PSNR (denoted as q^i_{raw}). Second, we will use 2D encoder (in our experiment, we use MPEG [30]) to encode P^i_{raw} into P^i_{qp} with different QPs, and we then calculate the PSNR between P^i_{qp} and P^i_{raw}, which we denote as $q^{i,\text{img}}_{qp}$. Finally, we would like to generate a prediction model Φ to predict q^i_{QP} based on q^i_{raw} and $q^{i,\text{img}}_{qp}$, i.e.,

$$q^i_{qp} = \Phi(q^i_{\text{raw}}, \bigcup q^{i,\text{img}}_{qp}), \text{ for } qp \in E \tag{11}$$

The rationales behind this design are two folds: on the one hand, to capture the temporal dynamics within the video, we use rendering results of the raw NeRF (i.e., the use of q^i_{raw}); On the other hand, since the quantization process for NeRF video extends directly from 2D videos (as shown in Figure 1), we can use the quantization results on rendered images to infer the corresponding quantization outcomes for the NeRF (i.e., the use of $q^{i,\text{img}}_{qp}$). We present the process of how we train the prediction model in Figure 17 (denoted as *Training Process*). Notably, we still need to quantize a few frames of the NeRF video (e.g., the first several frames) to get the labels for the prediction model (denoted as *Label Generation* in Figure 17). However, this represents only a small fraction of the total video length, and the storage and time overhead of encoding these 2D images are negligible.

We have tested different prediction models, including *Linear Regression (LR)*, *Random Forest (RF)*, *K Nearest Neighbor (KNN)*, *XGBoost (XB)*, and *Support Vector Regression (SVR)*. We use the first 10% frames as a training set, and the rest 90% frames as a testing set, and the results are shown in Figure 18 (the top sub-figure). We can see that all prediction models demonstrate good accuracy, with RF performing the best across all results. Therefore, we finally select RF as our prediction model. In addition, we also test RF's accuracy with different training set sizes (i.e., the number of frames), and the results are also shown in Figure 18 (the bottom sub-figure). We can see that as more frames are utilized, the prediction error gradually decreases and stabilizes. Considering both accuracy and encoding overhead, we finally use 10% of the frames to train the model.

Optimizer: Since this problem 1) lacks an explicit expression and 2) has a huge solution space, finding the closed-form solution

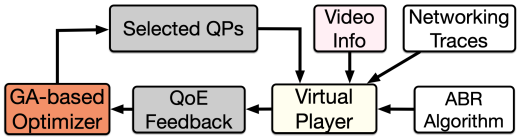


Figure 19: The workflow of GA-based optimizer.

is infeasible. Therefore, we will use the following two techniques to solve the problem. First, we convert the hard constraint in Eq.(9) to a soft constraint and adjust the objective function as a dual form [1]:

$$\max \sum_{n\in\mathbb{N}} QoE(\bigcup I_e, n) - \lambda \sum_{e=1}^{E} I_e \tag{12}$$

where λ is the penalty factor that balances the trade-off between QoE and transcoding overhead (i.e., higher λ means we care more about encoding overhead). Second, we consider adopting the Genetic Algorithm (GA) [29] to approximate the solution. The basic idea of GA is to use *natural evolution process* to model the solution search process. GA is suitable in this problem due to the following properties: First, it can well avoid local optima: it iteratively improves a set of candidate solutions through *selection* (i.e., choosing whether to retain specific solutions), *crossover* (i.e., combining multiple different solutions), and *mutation* (i.e., exploring new solutions), to converge to the optimal solution gradually. Second, GA does not rely on specific problem properties (such as continuity or differentiability), making it well-suited for our problem, where QoE lacks an explicit expression and the solution space is discrete. The iteration logic of the GA-based optimizer has been presented in Figure 19. Specifically, inspired by recent advances in ABR simulation work [3, 49], we implement a virtual player to simulate the playback process of the NeRF video and use it to assist GA in finding the optimal solution. The virtual player's input consists of three parts: the QPs selected by GA (along with the corresponding PSNR predicted by our aforementioned predictor, denoted as video info in Figure 19 and Figure 7), networking traces, and the ABR algorithm. The output is the user's QoE (as defined in Eq.(10)), which serves as feedback to iterate GA. The virtual player works as follows. Given timestamp t_k, we first calculate the download time δ_k for chunk k as $\delta_k = s_k/C_k$ (C_k is the bandwidth recorded in networking traces). We then update the current buffer size B_{k+1} as $B_{k+1} = \max[(B_k - \delta_k), 0] + L$, in which L is the chunk length (e.g., 1 second in our experiment). The rebuffering event occurs when there is no data in player's buffer. Therefore the rebuffering time can be calculated as $R_k = \max[(B_k - \delta_k), 0]$.

5 Evaluation

5.1 Methodology

Dataset. For NeRF videos, we use the Kpop dataset provided by [58]. Specifically, we retrain the model and generate the corresponding NeRF representation following [51, 58]. We use two public datasets for network traces: Solis [40] and Puffer [63]. For viewpoints, we use the dataset provided in [62] which records movement from ten users watching the volumetric video.

Implementation We split the user viewpoint dataset into 70% for training and 30% for testing. For rendering-adaptive pruning (§4.1), we generate the importance grid and master frame using all viewing traces from the training set. The voxel pruning criterion is set to $\theta = 0.99$, with further analysis in §5.3. For sub-frame generation, the raw NeRF feature grid is divided into $8 \times 8 \times 8$ non-overlapping cubes, each treated as a sub-frame. Viewpoint prediction is achieved using linear regression models. For QoE-aware bitrate generation, we set $\zeta = 1$, $\xi = 4.3$, and $\omega = 1$ in Eq.(10) as previous work [20, 42]. We use HYB [2] as the ABR algorithm to generate optimal QPs and evaluate other ABR algorithms in §5.3. We set $\lambda = 0.02$ in Eq.(12) and discuss its impact in §5.3.

5.2 Results and Discussion

Compression efficiency: We start with analyzing the efficiency of our rendering-adaptive pruning method through an evaluation of the master frame. We investigate the bandwidth requirement and PSNR under different QPs for NeRFlow and ReRF [58]. The results are shown in Figure 20, and we can obtain the following observations. First, compared with ReRF, NeRFlow achieves a much smaller bandwidth requirement with negligible quality loss. For example, we can see that in the highest quality (i.e., QP=100), NeRFlow reduces about 20% data size with only 0.4 dB PSNR reduction. Specifically, we can see for bandwidth around 40 Mbps (i.e., the average level in our measurement. See Figure 2), the PSNR only drops 1 dB compared with the highest quality NeRF video (recall that ReRF drops over 4 dB. See Figure 4). This is rational: on the one hand, NeRFlow prunes out voxels, therefore reducing data size; on the other hand, the pruned voxels are carefully selected by well leveraging their heterogeneous contribution made to the rendering effects, therefore having less impact on rendering quality. Second, NeRFlow's benefits are consistent across all QPs, and even obtain higher benefits for low QPs. E.g., we can see that the size reduction for QP=90 is about 62%, while for QP=20, the size is reduced over 77%. For better demonstration, we also present the size reduction of each QP in Figure 20 (see the bottom sub-figure). This is because the efficiency of entropy encoding is influenced by the distribution of data after quantization (i.e., the more scattered the distribution, the lower the encoding efficiency). In the case of NeRFlow, pruning out voxels helps concentrate the data distribution, especially at lower quantization levels, further improving encoding efficiency and reducing data size.

User QoE: We then use the three users' traces from testing dataset to evaluate NeRFlow in an end-to-end way. We first present the bitrate ladders generated after applying NeRFlow to the Solis and Puffer datasets. Specifically, NeRFlow generates the ladders with five bitrates (QP=40, 64, 79, 83, 88) for Solis, and seven for Puffer (QP=17, 37, 59, 68, 79, 84, 88). This difference is reasonable given that the Puffer dataset contains more varied network conditions, necessitating a broader bitrate range. We then evaluate NeRFlow by comparing its QoE with that of ReRF. For fair comparison, we apply the same QPs as NeRFlow for ReRF, and encode it into multiple bitrates (denoted as ReRF-ABR). It is also notable that ReRF-ABR will not have foveated streaming logic as it encodes all features in a uniform way. The results are depicted in the top sub-figures of Figure 21 and Figure 22. We can see that NeRFlow significantly outperforms ReRF-ABR in terms of QoE, with consistent improvements across datasets. For example, in the Solis

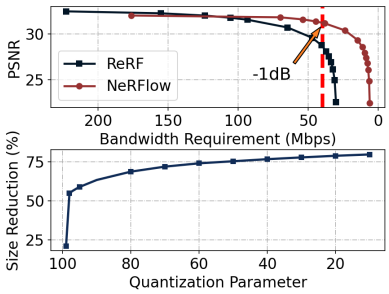


Figure 20: Evaluation for data size and quality

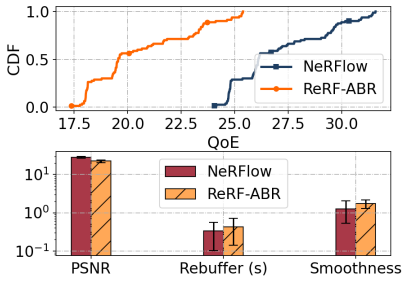


Figure 21: QoE evaluation and breakdowns on Solis dataset.

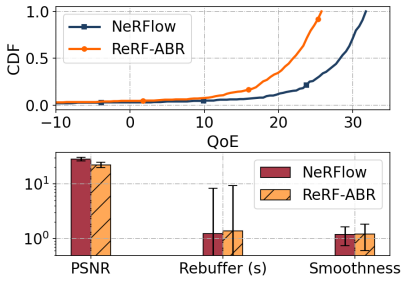
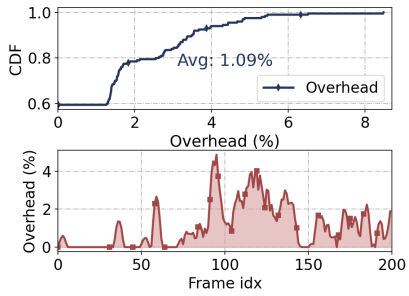
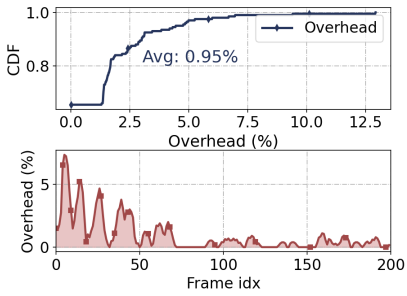


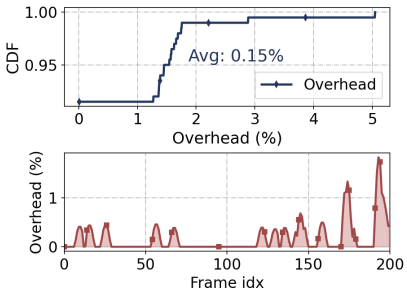
Figure 22: QoE evaluation and breakdowns on Puffer dataset



(a) Overhead for user A



(b) Overhead for user B



(c) Overhead for user C

Figure 23: The bandwidth overhead required to transmit sub-frames for three users. We can see that the overhead is manageable.

dataset, NeRFlow increases the average QoE by 31% (i.e., 27.0 v.s. 20.6), while in the Puffer dataset, it achieves an improvement of 41% (i.e., 23.2 v.s. 16.4). This result is expected as NeRFlow provides better compression efficiency while maintaing quality.

We further break down the QoE to better understand NeRFlow's performance gains, and the results are also illustrated in Figure 21 and Figure 22 (bottom sub-figure). We can see that a significant portion of NeRFlow's superior performance is due to its ability to maintain quality. For instance, NeRFlow improves average PSNR by 6.1 dB (28.4 v.s. 22.3 dB) in the Solis dataset and by 6.4 dB (28.3 v.s. 21.9) in the Puffer dataset. Additionally, NeRFlow also performs better in rebuffering (e.g., NeRFlow reduces over 21.1% and 10.5%).

As sub-frames are used to compensate for regions not covered by master frames, we analyze the bandwidth required to transmit these sub-frames. Specifically, for each frame, we calculate the proportion of the overall bandwidth occupied by the sub-frames. We then compute the CDFs separately for three users in our test set and present the results in Figure 23 (top sub-figure). We observe that sub-frames introduce minimal overhead. For instance, the average overhead for all three users is below 1.1%, with the third user's overhead as low as 0.15%. These results align with our measurements shown in Figure 9(a) and Figure 9(b), which indicate that user viewpoints are highly concentrated.

At the same time, we analyze how the bandwidth overhead varies with time for three tested users, and the results are also shown in Figure 23 (bottom sub-figure). We can obtain the following observations. First, for the same user, the sub-frame overhead fluctuates at different period of time. This is reasonable as user viewpoints shift dynamically over time. Second, for different users, the time periods with the highest sub-frame overhead also differ from each other. E.g., for the first user, the peak overhead is located between 60-150th frames, while for the second user, it comes from 0-80th frames). This highlights the fact that there exists heterogeneity in user viewing preferences, and it further suggests that relying solely on master frames is insufficient.

5.3 NeRFlow Deep Dive

The factors influencing compression efficiency. As pointed in §4.1, the size reduction of master frame is influenced by the viewpoint distribution. E.g., in some complex scenes with more objects, the viewers may be interested in multiple points, which can introduce more diverse viewpoints. In these cases, the compression benefits from ray-level will be degraded since each voxel is equally important at the ray level. To illustrate, we create synthetic viewing traces with the video at the center and evenly distributed viewpoints (fixed height). We gradually expand the view range from 10% to 100% and analyze the resulting size reduction, as shown in Figure 24 (top sub-figure). We can see that as the view range increases, the size reduction decreases. However, even at a 100% view range, NeRFlow still achieves a 15% size reduction, since it can still perform voxel

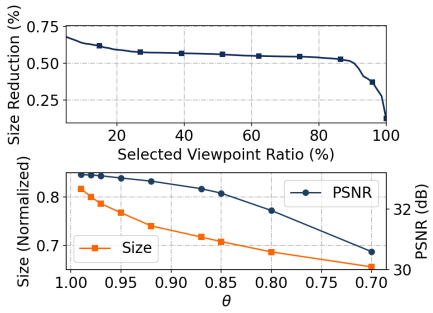


Figure 24: The influence of selected viewpoints and importance threshold θ.

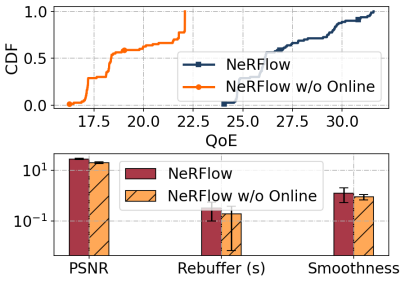


Figure 25: QoE evaluation and breakdowns on Solis dataset.

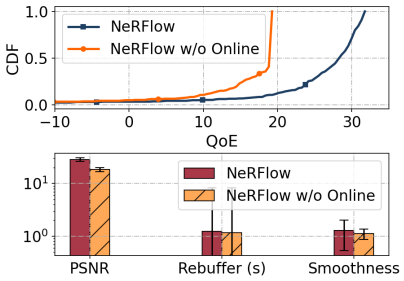


Figure 26: QoE evaluation and breakdowns on Puffer dataset.

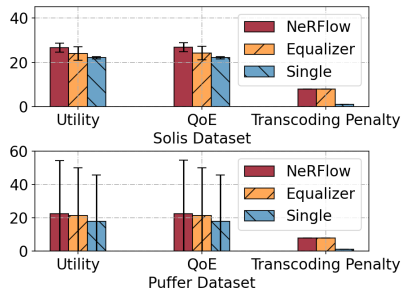


Figure 27: The necessity of GA-based optimizer.

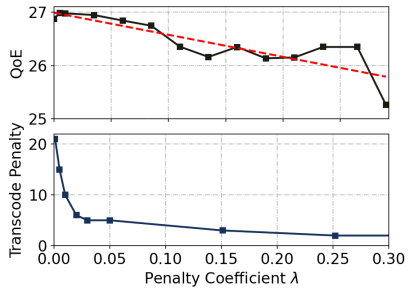


Figure 28: The influence of λ.

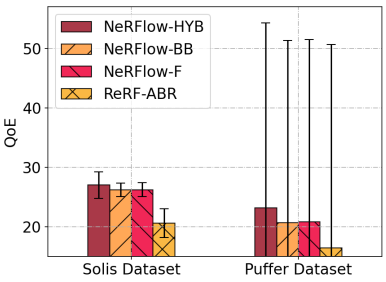


Figure 29: NeRFlow with different ABR algorithms.

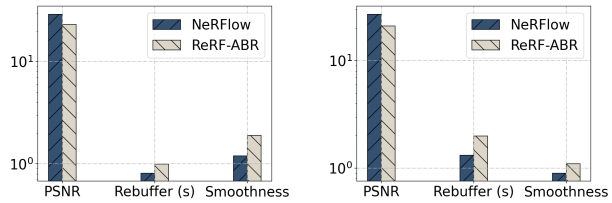
pruning based on sample-level importance. Actually, as long as the user's viewpoint isn't uniformly distributed (which is quite common in real-world), our method remains effective.

Second, the pruning threshold θ affects compression efficiency. We examine how frame size and quality change with θ, as shown in Figure 24 (bottom). Lower θ (fewer voxels selected) yields greater size reduction but larger quality loss. We also observe diminishing returns in size reduction as θ decreases, consistent with Figure 11, since voxel importance becomes more uniform at lower θ, making effective pruning more difficult.

The necessity of online adaptation. We conduct an ablation study to evaluate the effectiveness of the adaptation module by comparing NeRFlow with and without it. As shown in Figures 25 and 26, the absence of the adaptation module leads to significant QoE degradation (39.8% for Solis and 74.1% for Puffer). This drop is primarily due to reduced quality, with PSNR decreasing by 8 dB in Solis, as relying solely on the master frame fails to cover all viewpoints. We also observe a slight increase in rebuffering time due to the additional overhead from transmitting sub-frames (see Figure 23). Notably, NeRFlow without the adaptation module performs worse than ReRF-ABR (Figure 21), as ReRF's uniform voxel compression avoids data incompleteness. These results highlight the necessity of the adaptation module.

The effectiveness of GA-based optimizer. We evaluate the effectiveness of NeRFlow's QoE-aware bitrate generation by comparing it with two baselines that use the same pruning and adaptive modules but differ in QP selection. The first baseline (*Single*) uses a single QP, chosen offline to maximize QoE on our dataset. The second baseline (*Equalizer*) uses the same number of QPs as NeRFlow but selects them based on evenly dividing the network bandwidth distribution and picking QPs closest to the average bandwidth of each segment. We assess performance using utility (Eq.(12)), QoE, and transcoding penalty. As shown in Figure 27, NeRFlow outperforms both baselines across all datasets, highlighting the importance of its GA-based optimizer in capturing network dynamics. We also investigate the influence of the penalty coefficient λ in Eq.(12), and the results are shown in Figure 28. We can see that with λ increasing, NeRFlow will select less number of QPs (see the top sub-figure); at the same time, the achieved QoE will also gradually decrease (see the bottom sub-figure). This is expected, as more QPs provide more candidates, potentially allowing the ABR algorithm to make more fine-grained decisions in response to network dynamics.

NeRFlow with different ABR algorithms. To demonstrate that our framework is generalizable, we also integrate NeRFlow with different ABR algorithms and compare with ReRF-ABR. Specifically, we choose a rate-based algorithm named Festive [26] and a buffer-based algorithm named BBA [21] (it is notable that for both ABR algorithms, we will re-run the pipeline to get the best QPs). For ease of comparison, ReRF-ABR is implemented with HYB. The results are shown in Figure 29. We can see that regardless of whether BB or Festive is used, NeRFlow achieves significantly higher QoE compared to ReRF-ABR. Notably, this performance advantage is



(a) Cellular (ave: 61.2 Mbps, std: 35.1 Mbps) (b) WI-FI (ave: 18.0 Mbps, std: 6.4 Mbps)

Figure 30: Real-world network performance

consistent across both the Solis and Puffer datasets. Additionally, we can also observe that NeRFlow integrated with HYB outperforms both Festive and BB.

5.4 Real-world Experiments.

To further evaluate NeRFlow, we also conduct real-world experiments using the test pipeline for traditional VoDs (we manully add extra data to simulate the overhead of viewpoint mismatch). We use iPerf [24] to collect network traces, based on which we determine QPs and generate multiple 2D videos to act as NeRF videos. We then deploy an HTTP server, and modify *dash.js* [15] to support HYB and download the videos. We test NeRFlow under cellular and Wi-Fi conditions. In our real-world test, cellular network has higher average bandwidth than WI-FI (61.2 v.s. 18.0 Mbps), but also higher variation (35.1 v.s. 6.4 Mbps). For ease of illustration, we also encode ReRF into multiple bitrates and implement it with *dash.js*. The results are shown in Figure 30. We can see that NeRFlow performs well under both network conditions (especially well controls rebuffering events). Also, the comparison results with ReRF-ABR are consistent with our trace-driven experiment: i.e., the benefits mainly come from higher PSNR, which is explainable as NeRFlow can better compress video data through our pruning technique. It is also notable that even in cellular networks, which exhibit greater variability compared to Wi-Fi, NeRFlow still achieves better PSNR and shorter rebuffering time.

6 Related Work

Our work is closely related to two areas: volumetric video streaming and NeRF-based volumetric representation.

Volumetric video streaming: Existing research mainly focuses on point-cloud or mesh-based approaches. Some work aims at compressing spatial information, such as [18, 32], which optimizes spatial partitioning for point clouds to reduce data redundancy. Other approaches focus on color space information, e.g., representing volumetric data using multiple 2D views and applying 2D encoders for compression, like multi-angle projection [25, 38, 47, 66] and converting depth maps to grayscale images [31]. Additionally, many studies utilize user viewing information for dynamic downsampling, such as ViVo [19] and QV4 [48]. While NeRFlow shares a similar high-level idea of achieving efficient transmission through compression, the implicit nature of NeRF requires new designs for both compression and transmission.

Novel volumetric representation: Traditional volumetric videos often rely on explicit representations like point clouds and meshes [5, 41], but these suffer from limited rendering quality. Recent advances have shifted focus to neural representations such as NeRF, with research targeting improvements in rendering speed and quality [11, 54, 60], as well as model compression through pruning [10], tensor decomposition [7], and encoding [44, 65]. However, these methods generally overlook transmission adaptability and can complement our work. Some studies do consider NeRF transmission [8], but they focus on static models and require retraining, limiting scalability to video scenarios. Gaussian Splatting has recently emerged for its fast, high-quality rendering [27, 61], but its point-based nature leads to significantly larger data sizes, e.g., 500 Mbps for 30 FPS [52], making it less transmission-friendly. While 2D video codecs have been applied to compress GS features [59], such methods still require retraining and fail to adapt to dynamic network conditions. In contrast, NeRFlow offers a scalable, adaptable design that can also be extended to GS.

7 Conclusion

We have introduced NeRFlow, a novel framework designed to enable adaptive streaming for NeRF videos under real-world network conditions. By addressing key challenges of limited bandwidth and temporal dynamics, NeRFlow optimizes the streaming process through rendering-adaptive pruning, viewpoint-aware adaptation, and QoE-aware bitrate ladder generation. Our extensive experiments demonstrate that NeRFlow significantly improves user Quality of Experience (QoE) by 31.3% to 41.2%, making it an efficient framework for NeRF video streaming.

ACKNOWLEDGMENTS

We thank our shepherd and reviewers for their valuable feedback, which has greatly improved the quality of this paper. This work was supported by NSF CNS 21-06592, NSF CNS 19-00875, NSF CCF 22-17144.

References

[1] Peter Adby. 2013. *Introduction to optimization methods.* Springer Science & Business Media.
[2] Zahaib Akhtar, Yun Seong Nam, Ramesh Govindan, Sanjay Rao, Jessica Chen, Ethan Katz-Bassett, Bruno Ribeiro, Jibin Zhan, and Hui Zhang. 2018. Oboe: Auto-tuning video ABR algorithms to network conditions. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication.* 44–58.
[3] Abdullah Alomar, Pouya Hamadanian, Arash Nasr-Esfahany, Anish Agarwal, Mohammad Alizadeh, and Devavrat Shah. 2023. {CausalSim}: A Causal Framework for Unbiased {Trace-Driven} Simulation. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23).* 1115–1147.
[4] Jonathan T Barron, Ben Mildenhall, Dor Verbin, Pratul P Srinivasan, and Peter Hedman. 2023. Zip-nerf: Anti-aliased grid-based neural radiance fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision.* 19697–19705.
[5] Saifullahi Aminu Bello, Shangshu Yu, Cheng Wang, Jibril Muhmmad Adam, and Jonathan Li. 2020. Deep learning on 3D point clouds. *Remote Sensing* 12, 11 (2020), 1729.
[6] Abdelhak Bentaleb, Bayan Taani, Ali C Begen, Christian Timmerer, and Roger Zimmermann. 2018. A survey on bitrate adaptation schemes for streaming media over HTTP. *IEEE Communications Surveys & Tutorials* 21, 1 (2018), 562–585.
[7] Anpei Chen, Zexiang Xu, Andreas Geiger, Jingyi Yu, and Hao Su. 2022. Tensorf: Tensorial radiance fields. In *European conference on computer vision.* Springer, 333–350.
[8] Bo Chen, Zhisheng Yan, Bo Han, and Klara Nahrstedt. 2024. NeRFHub: A Context-Aware NeRF Serving Framework for Mobile Immersive Applications. In *Proceedings of the 22nd Annual International Conference on Mobile Systems, Applications*

and Services. 85–98.

[9] Yihang Chen, Qianyi Wu, Mehrtash Harandi, and Jianfei Cai. 2024. How Far Can We Compress Instant-NGP-Based NeRF?. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 20321–20330.

[10] 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.

[11] Shin-Fang Chng, Sameera Ramasinghe, Jamie Sherrah, and Simon Lucey. 2022. Gaussian activated neural radiance fields for high fidelity reconstruction and pose estimation. In *European Conference on Computer Vision*. Springer, 264–280.

[12] 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.

[13] Cyril Crassin, Fabrice Neyret, Sylvain Lefebvre, and Elmar Eisemann. 2009. Gigavoxels: Ray-guided streaming for efficient and detailed voxel rendering. In *Proceedings of the 2009 symposium on Interactive 3D graphics and games*. 15–22.

[14] Jiemin Fang, Taoran Yi, Xinggang Wang, Lingxi Xie, Xiaopeng Zhang, Wenyu Liu, Matthias Nießner, and Qi Tian. 2022. Fast dynamic radiance fields with time-aware neural voxels. In *SIGGRAPH Asia 2022 Conference Papers*. 1–9.

[15] Dash Industry Forum. 2024. dash.js: Reference Client for the MPEG DASH Standard. https://github.com/Dash-Industry-Forum/dash.js. Version 4.0.0, available at https://github.com/Dash-Industry-Forum/dash.js.

[16] 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).

[17] Stephan J Garbin, Marek Kowalski, Matthew Johnson, Jamie Shotton, and Julien Valentin. 2021. Fastnerf: High-fidelity neural rendering at 200fps. In *Proceedings of the IEEE/CVF international conference on computer vision*. 14346–14355.

[18] Diogo C Garcia and Ricardo L de Queiroz. 2018. Intra-frame context-based octree coding for point-cloud geometry. In *2018 25th IEEE International Conference on Image Processing (ICIP)*. IEEE, 1807–1811.

[19] 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.

[20] Tianchi Huang, Xin Yao, Chenglei Wu, Rui-Xiao Zhang, Zhengyuan Pang, and Lifeng Sun. 2019. Tiyuntsong: A self-play reinforcement learning approach for ABR video streaming. In *2019 IEEE International Conference on Multimedia and Expo (ICME)*. IEEE, 1678–1683.

[21] Te-Yuan Huang, Ramesh Johari, Nick McKeown, Matthew Trunnell, and Mark Watson. 2014. A buffer-based approach to rate adaptation: Evidence from a large video streaming service. In *Proceedings of the 2014 ACM conference on SIGCOMM*. 187–198.

[22] Weiming Huang, Baisong Liu, and Hao Tang. 2019. Privacy protection for recommendation system: a survey. In *Journal of Physics: Conference Series*, Vol. 1325. IOP Publishing, 012087.

[23] Zhenyang Hui, Youjian Hu, Yao Ziggah Yevenyo, and Xianyu Yu. 2016. An improved morphological algorithm for filtering airborne LiDAR point cloud based on multi-level kriging interpolation. *Remote Sensing* 8, 1 (2016), 35.

[24] The iPerf Developers. 2024. iPerf3: A TCP, UDP, and SCTP Network Bandwidth Measurement Tool. https://iperf.fr. Version 3.17.1, available at https://iperf.fr.

[25] Euee S Jang, Marius Preda, Khaled Mammou, Alexis M Tourapis, Jungsun Kim, Danillo B Graziosi, Sungryeul Rhyu, and Madhukar Budagavi. 2019. Video-based point-cloud-compression standard in MPEG: From evidence collection to committee draft [standards in a nutshell]. *IEEE Signal Processing Magazine* 36, 3 (2019), 118–123.

[26] Junchen Jiang, Vyas Sekar, and Hui Zhang. 2012. Improving fairness, efficiency, and stability in http-based adaptive video streaming with festive. In *Proceedings of the 8th international conference on Emerging networking experiments and technologies*. 97–108.

[27] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 2023. 3D Gaussian Splatting for Real-Time Radiance Field Rendering. *ACM Trans. Graph.* 42, 4 (2023), 139–1.

[28] Georgios Kouros, Minye Wu, Sushruth Nagesh, Shubham Shrivastava, Punarjay Chakravarty, and Tinne Tuytelaars. 2023. Ref-DVGO: Reflection-Aware Direct Voxel Grid Optimization for an Improved Quality-Efficiency Trade-Off in Reflective Scene Reconstructio. *arXiv preprint arXiv:2308.08530* (2023).

[29] Annu Lambora, Kunal Gupta, and Kriti Chopra. 2019. Genetic algorithm-A literature review. In *2019 international conference on machine learning, big data, cloud and parallel computing (COMITCon)*. IEEE, 380–384.

[30] Didier Le Gall. 1991. MPEG: A video compression standard for multimedia applications. *Commun. ACM* 34, 4 (1991), 46–58.

[31] Kyungjin Lee, Juheon Yi, and Youngki Lee. 2023. Farfetchfusion: Towards fully mobile live 3d telepresence platform. In *Proceedings of the 29th Annual International Conference on Mobile Computing and Networking*. 1–15.

[32] 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.

[33] David Levin. 2004. Mesh-independent surface interpolation. In *Geometric modeling for scientific visualization*. Springer, 37–49.

[34] Lingzhi Li, Zhen Shen, Zhongshu Wang, Li Shen, and Liefeng Bo. 2023. Compressing volumetric radiance fields to 1 mb. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 4222–4231.

[35] Shaochen Li, Hongbin Guo, Wangqian Sun, and Xiaofei Sun. 2022. A low-illuminance image enhancement method in YUV color space. In *2022 14th International Conference on Measuring Technology and Mechatronics Automation (ICMTMA)*. IEEE, 286–291.

[36] Haojie Liu, Kang Liao, Chunyu Lin, Yao Zhao, and Meiqin Liu. 2020. Plin: A network for pseudo-lidar point cloud interpolation. *Sensors* 20, 6 (2020), 1573.

[37] Jia-Wei Liu, Yan-Pei Cao, Weijia Mao, Wenqiao Zhang, David Junhao Zhang, Jussi Keppo, Ying Shan, Xiaohu Qie, and Mike Zheng Shou. 2022. Devrf: Fast deformable voxel radiance fields for dynamic scenes. *Advances in Neural Information Processing Systems* 35 (2022), 36762–36775.

[38] 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.

[39] Yuchen Luo, Yong Zhang, Junchi Yan, and Wei Liu. 2021. Generalizing face forgery detection with high-frequency features. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 16317–16326.

[40] Gerui Lv, Qinghua Wu, Qingyue Tan, Weiran Wang, Zhenyu Li, and Gaogang Xie. 2023. Accurate Throughput Prediction for Improving QoE in Mobile Adaptive Streaming. *IEEE Transactions on Mobile Computing* (2023).

[41] Adrien Maglo, Guillaume Lavoué, Florent Dupont, and Céline Hudelot. 2015. 3d mesh compression: Survey, comparisons, and emerging trends. *ACM Computing Surveys (CSUR)* 47, 3 (2015), 1–41.

[42] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. 2017. Neural adaptive video streaming with pensieve. In *Proceedings of the conference of the ACM special interest group on data communication*. 197–210.

[43] 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.

[44] 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.

[45] Grand View Research. 2023. *Volumetric Video Market Size, Share Trends Analysis Report By Volumetric Capture (Hardware, Software, Services), By Delivery Platforms, By Application, By Region, And Segment Forecasts, 2023 - 2030*. Technical Report.

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

[47] Sebastian Schwarz, Marius Preda, Vittorio Baroncini, Madhukar Budagavi, Pablo Cesar, Philip A Chou, Robert A Cohen, Maja Krivokuća, Sébastien Lasserre, Zhu Li, et al. 2018. Emerging MPEG standards for point cloud compression. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 9, 1 (2018), 133–148.

[48] Yuang Shi, Bennett Clement, and Wei Tsang Ooi. 2024. QV4: QoE-based Viewpoint-Aware V-PCC-encoded Volumetric Video Streaming. In *Proceedings of the 15th ACM Multimedia Systems Conference*. 144–154.

[49] Kevin Spiteri, Ramesh Sitaraman, and Daniel Sparacio. 2019. From theory to practice: Improving bitrate adaptation in the DASH reference player. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)* 15, 2s (2019), 1–29.

[50] Kevin Spiteri, Rahul Urgaonkar, and Ramesh K Sitaraman. 2020. BOLA: Near-optimal bitrate adaptation for online videos. *IEEE/ACM transactions on networking* 28, 4 (2020), 1698–1711.

[51] Cheng Sun, Min Sun, and Hwann-Tzong Chen. 2022. Direct voxel grid optimization: Super-fast convergence for radiance fields reconstruction. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 5459–5469.

[52] Jiakai Sun, Han Jiao, Guangyuan Li, Zhanjie Zhang, Lei Zhao, and Wei Xing. 2024. 3dgstream: On-the-fly training of 3d gaussians for efficient streaming of photo-realistic free-viewpoint videos. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 20675–20685.

[53] Jiakai Sun, Zhanjie Zhang, Jiafu Chen, Guangyuan Li, Boyan Ji, Lei Zhao, Wei Xing, and Huaizhong Lin. 2023. Vgos: Voxel grid optimization for view synthesis from sparse inputs. *arXiv preprint arXiv:2304.13386* (2023).

[54] Shilei Sun, Ming Liu, Zhongyi Fan, Qingliang Jiao, Yuxue Liu, Liquan Dong, and Lingqin Kong. 2024. Efficient ray sampling for radiance fields reconstruction. *Computers & Graphics* 118 (2024), 48–59.

[55] TaoXi Technology. 2021. *3D Modeling Techniques for Products Based on Neural Rendering*. https://www.alibabacloud.com/blog/3d-modelingtechniques-for-products-based-on-neural-rendering_598327

[56] Jim Thacker. 2023. *Use NeRFs in Unreal Engine with Luma AI's new plugin.* https://www.cgchannel.com/2023/04/use-nerfs-in-unreal-enginewith-luma-ais-new-plugin/

[57] Chen Wang, Xian Wu, Yuan-Chen Guo, Song-Hai Zhang, Yu-Wing Tai, and Shi-Min Hu. 2022. Nerf-sr: High quality neural radiance fields using supersampling. In *Proceedings of the 30th ACM International Conference on Multimedia*. 6445–6454.

[58] 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*. 76–87.

[59] Penghao Wang, Zhirui Zhang, Liao Wang, Kaixin Yao, Siyuan Xie, Jingyi Yu, Minye Wu, and Lan Xu. 2024. V^ 3: Viewing Volumetric Videos on Mobiles via Streamable 2D Dynamic Gaussians. *ACM Transactions on Graphics (TOG)* 43, 6 (2024), 1–13.

[60] Sen Wang, Wei Zhang, Stefano Gasperini, Shun-Cheng Wu, and Nassir Navab. 2023. VoxNeRF: Bridging voxel representation and neural radiance fields for enhanced indoor view synthesis. *arXiv preprint arXiv:2311.05289* (2023).

[61] Guanjun Wu, Taoran Yi, Jiemin Fang, Lingxi Xie, Xiaopeng Zhang, Wei Wei, Wenyu Liu, Qi Tian, and Xinggang Wang. 2024. 4d gaussian splatting for real-time dynamic scene rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 20310–20320.

[62] Nan Wu, Kaiyan Liu, Ruizhi Cheng, Bo Han, and Puqi Zhou. 2024. Theia: Gaze-driven and Perception-aware Volumetric Content Delivery for Mixed Reality Headsets. In *Proceedings of the 22nd Annual International Conference on Mobile Systems, Applications and Services*. 70–84.

[63] Francis Y Yan, Hudson Ayers, Chenzhi Zhu, Sadjad Fouladi, James Hong, Keyi Zhang, Philip Levis, and Keith Winstein. 2020. Learning in situ: a randomized experiment in video streaming. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. 495–511.

[64] Xiaoqi Yin, Abhishek Jindal, Vyas Sekar, and Bruno Sinopoli. 2015. A control-theoretic approach for dynamic adaptive video streaming over HTTP. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*. 325–338.

[65] Alex Yu, Ruilong Li, Matthew Tancik, Hao Li, Ren Ng, and Angjoo Kanazawa. 2021. Plenoctrees for real-time rendering of neural radiance fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 5752–5761.

[66] Junzhe Zhang, Tong Chen, Dandan Ding, and Zhan Ma. 2023. G-PCC++: Enhanced Geometry-based Point Cloud Compression. In *Proceedings of the 31st ACM International Conference on Multimedia*. 1352–1363.