Vega: Fully Immersive Mobile Volumetric Video Streaming with 3D Gaussian Splatting

Gunjoong Kim*
Yonsei University
Seoul, Republic of Korea
gunjoong.kim@yonsei.ac.kr

Seonghoon Park*
Yonsei University
Seoul, Republic of Korea
park.s@yonsei.ac.kr

Jeho Lee
Yonsei University
Seoul, Republic of Korea
jeholee@yonsei.ac.kr

Chanyoung Jung
Yonsei University
Seoul, Republic of Korea
cy.jung@yonsei.ac.kr

Hyungchol Jun
Yonsei University
Seoul, Republic of Korea
hyungchol.jun@yonsei.ac.kr

Hojung Cha†
Yonsei University
Seoul, Republic of Korea
hjcha@yonsei.ac.kr

Abstract

For highly immersive mobile volumetric video streaming, it is essential to deliver photo-realistic full-scene content with smooth playback. Unlike traditional representations such as point clouds, 3D Gaussian Splatting (3DGS) has gained attention for its ability to represent high-quality full-scene 3D content. However, our preliminary experiments show that existing methods for 3DGS-based videos fail to achieve smooth playback on mobile devices. In this paper, we propose Vega, a 3DGS-based photo-realistic full-scene volumetric video streaming system that ensures real-time playback on mobile devices. The core idea behind Vega's real-time rendering is object-level selective computation, which allocates computational resources to visually important objects to meet strict rendering deadlines. To enable mobile streaming based on the selective computation, Vega addresses two challenges: (1) designing an encoding scheme that optimizes the data size of videos while being compatible with object-level prioritization, and (2) developing a rendering pipeline that efficiently operates on resource-constrained mobile devices. We implemented an end-to-end Vega system, consisting of a streaming server and an Android application. Experimental results on commodity smartphones show that Vega achieves 30 frames per second (FPS) for full-scene volumetric video streaming while maintaining competitive data size and visual quality compared to existing baselines.

*Co-primary authors with equal contribution.
†Corresponding author

ACM MOBICOM '25, Hong Kong, China

ACM ISBN 979-8-4007-1129-9/2025/11
https://doi.org/10.1145/3680207.3765267

CCS Concepts

• **Human-centered computing** → **Mobile computing**; **Mobile devices**; • **Computing methodologies** → **Computer vision**.

Keywords

Mobile volumetric video streaming, real-time rendering, 3D Gaussian Splatting

ACM Reference Format:
Gunjoong Kim, Seonghoon Park, Jeho Lee, Chanyoung Jung, Hyungchol Jun, and Hojung Cha. 2025. Vega: Fully Immersive Mobile Volumetric Video Streaming with 3D Gaussian Splatting. In *The 31st Annual International Conference on Mobile Computing and Networking (ACM MOBICOM '25), November 4–8, 2025, Hong Kong, China.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3680207.3765267

1 Introduction

Mobile volumetric video streaming allows users to explore 3D videos from free viewpoints on their mobile devices, providing immersive and interactive experiences. Due to its highly immersive nature, volumetric video has been gaining attention for various applications in mobile virtual reality (VR) and mixed reality (MR), including real-time remote collaboration [40], immersive education [9], and virtual entertainment [6].

For a *fully immersive* experience, mobile volumetric video streaming should deliver photo-realistic full-scene videos with smooth playback. However, conventional mobile volumetric streaming [14, 15, 24, 25, 28, 29, 44, 47], which uses fixed representations such as point clouds, has limitations in both achieving photo-realism and delivering full-scene videos. Fixed representations inherently suffer from spatial discontinuities, especially when the density is low or when objects appear close to the viewer, resulting in a degradation in photo-realism. Moreover, these solutions primarily

focus on object-based videos that include only one or a few objects without surrounding environments; as a result, they fail to handle the complexity of full-scene videos, which contain multiple objects, surrounding environments, and backgrounds [50].

Recently, 3D Gaussian Splatting (3DGS) [19], a promising novel view synthesis technique, has emerged as an attractive solution for volumetric videos due to its capability to represent photo-realistic 3D full-scene content. Compared to Neural Radiance Fields (NeRF) [31], which is another view synthesis technique, 3DGS shows considerably less rendering overhead. However, dynamic scenes represented with 3DGS contain substantial amounts of data; thus, numerous studies [27, 38, 45] have focused on optimizing the data size of dynamic 3DGS content.

Despite the superiority of 3DGS in photorealism and full-scene capability, adapting the technique for mobile volumetric video streaming—ensuring smooth playback at consistent frame rates, such as 30 frames per second (FPS)—remains a significant challenge. Our preliminary experiments show that even an approach that explicitly represents 3DGS for every frame—without applying data compression and thus incurring no decoding overhead—fails to achieve 30 FPS rendering for full-scene volumetric videos. Moreover, if additional computations are introduced for data size optimization, as in existing dynamic 3DGS methods, the decoding overhead inevitably increases, making smooth playback even more difficult to achieve. Therefore, a new approach is needed for 3DGS-based mobile volumetric video streaming—one that optimizes data size while simultaneously ensuring real-time rendering.

In this paper, we introduce Vega, a mobile streaming system for photo-realistic full-scene volumetric videos with 3D Gaussian splatting. The primary goal of Vega is to ensure smooth playback (i.e., 30 FPS) for real-time rendering while optimizing data size and video quality. As rendering all Gaussians on mobile devices cannot meet target frame rates, Vega introduces object-level selective computation, allocating computational resources primarily to Gaussians associated with visually important objects. Realizing this approach introduces two key technical challenges. First, existing data-size optimization methods for dynamic 3DGS [27, 30, 38, 43, 45] are incompatible with selective prioritization because they operate at the Gaussian level without providing object-level information. A new encoding method is needed to optimize data size while considering object-level computation. Second, designing an efficient rendering pipeline for mobile devices remains challenging. Such a pipeline must assess the visual importance of objects in real time and efficiently utilize the limited computational resources of mobile devices.

To address the challenge of encoding, Vega introduces a mobile-friendly 3DGS video encoding scheme optimized for object-level selective computation. This scheme segments a scene into multiple semantically meaningful objects and adopts an adaptive group-of-volume (GOV) structure, allowing residual frames to efficiently reduce data size by exploiting object-level data from key frames. The scheme further optimizes data size through hierarchical color encoding and dynamicity-based object filtering techniques, which improve compression for both color-related and non-color attributes. To tackle the challenge of rendering pipeline, Vega proposes a view-adaptive rendering pipeline that effectively reduces computations by selectively rendering Gaussians based on viewpoint relevance and object dynamicity. This pipeline employs object-level early culling, removing objects not visible in the current view in advance, and introduces a priority-based task scheduling method, effectively utilizing the available processors on mobile devices, such as CPU, GPU, and NPU, to further enhance rendering efficiency.

The key contributions of Vega are as follows:

- To the best of our knowledge, Vega is the first endeavor to leverage 3DGS in providing photo-realistic full-scene volumetric video streaming for mobile devices.
- Vega enables real-time rendering of 3DGS-based videos on mobile devices through object-level selective computation, featuring a mobile-friendly 3DGS video encoding scheme and a view-adaptive rendering pipeline.
- We developed an end-to-end Vega prototype with a streaming server and an Android application, demonstrating superior performance over state-of-the-art methods in real mobile device experiments.

2 Background and Motivation

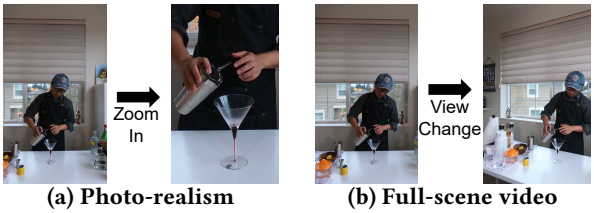
2.1 Volumetric Video Representations

Volumetric videos offer the highest degrees of freedom (DoF), providing highly immersive experiences for users. Traditional 2D videos have fixed viewpoints, while 360-degree videos support rotation of viewing directions described by yaw, pitch, and roll, providing three degrees of freedom (3DoF). In contrast, volumetric videos allow users to freely change viewing directions and positions (X, Y, and Z), thus offering six degrees of freedom (6DoF) [42]. To represent such immersive videos, various representation techniques, including point clouds, meshes, NeRF, and 3DGS, have been proposed [11].

These representation techniques can be categorized into fixed representations and learnable representations depending on whether they require training during scene reconstruction. Fixed representations, such as point clouds, meshes, and voxels, store and represent 3D space directly

(a) Visual discontinuity (b) Object-based video

Figure 1: Characteristics of conventional mobile volumetric video streaming.



(a) Photo-realism (b) Full-scene video

Figure 2: Opportunities with 3DGS-based mobile volumetric video streaming.

without additional training. In contrast, learnable representations, such as NeRF and 3DGS, require training processes to optimize structural features of the 3D data, resulting in superior visual quality compared to fixed representations. NeRF and 3DGS are methods for novel view synthesis, requiring multiple RGB cameras capturing the scene from different angles. The captured data are divided into training and test camera images to learn and represent the 3D space.

Representation techniques can also be classified as explicit or implicit. Explicit representations, such as fixed representations and 3DGS, explicitly store individual data elements, such as position, color, or other attributes, using explicitly defined values or functions. In contrast, implicit representations, such as NeRF, represent 3D spaces using neural networks as implicit functions without directly storing individual data points. Although implicit representations can greatly reduce data size, they incur considerable computational overhead due to the iterative neural network inference during rendering.

2.2 Limitations of Conventional Mobile Volumetric Video Streaming

Mobile volumetric video streaming delivers volumetric videos from servers to mobile devices over mobile networks. Compared to traditional 2D or 360-degree videos, volumetric videos require significantly more data and computational resources [11, 42]. Thus, to reduce complexity, traditional mobile volumetric video streaming systems mainly focus on fixed representations, such as point clouds [14, 15, 25, 28, 29, 44, 47] or voxels [24].

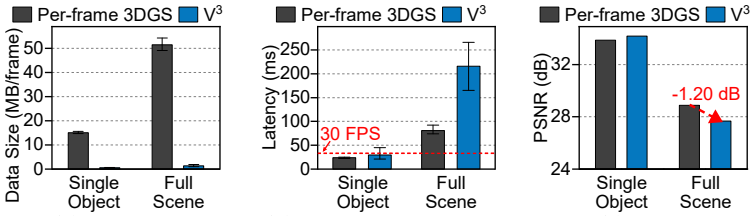
To provide the most immersive experience, volumetric video streaming should deliver photo-realistic full-scene videos, including comprehensive backgrounds and multiple objects [50]. However, traditional systems face two primary limitations, visually illustrated in Figure 1. First, fixed representations inherently represent 3D objects discretely, which limits their ability to achieve photo-realistic results. Although increasing the density of representations can reduce visual gaps, users inevitably observe discontinuities when approaching objects closely, as shown in Figure 1a. Second, traditional systems typically focus on object-based videos composed of only one or several objects, without surrounding backgrounds, as illustrated in Figure 1b. Fixed representations often struggle to naturally represent complex and continuous backgrounds, restricting their application in full-scene volumetric video streaming.

2.3 Opportunities: 3D Gaussian Splatting

Emerging in novel view synthesis research, NeRF and 3DGS are techniques designed to synthesize 3D scenes from data captured from multiple viewpoints. Unfortunately, NeRF requires extensive computation due to the iterative neural network inference, making it infeasible for mobile devices to render dynamic scenes represented by NeRF in real-time [7, 41]. Therefore, 3DGS, which has considerably lower rendering overhead than NeRF, is a promising method to deliver photo-realistic full-scene volumetric videos for mobile devices. Figure 2 highlights the photo-realism and potential for full-scene streaming with 3DGS-based videos.

Original 3DGS [19] represents a static 3D scene using multiple Gaussians, rendering new views by projecting and blending these Gaussians. Each Gaussian carries attributes such as position, rotation, scale, opacity, and color. Color is generally represented using spherical harmonic (SH) coefficients. These Gaussians are optimized through an iterative training process that minimizes the difference between rendered images and actual captured images from multiple viewpoints. Recent studies have explored using 3DGS to represent dynamic 3D scenes incorporating the temporal dimension [27, 38, 45]. These studies aim to reduce the data size and training time for dynamic 3DGS. However, when it comes to rendering, they assume powerful GPU resources typically available on desktop computers, making it impractical to directly apply these methods in mobile streaming scenarios [46].

V^3 [43] recently proposed a method for dynamic 3DGS-based video streaming on mobile devices, optimizing data with minimal decoding overhead to accommodate mobile streaming constraints. However, this approach is specifically designed for object-based videos featuring only a single human, neglecting the complexity of full-scene videos that contain multiple objects and complex backgrounds.



(a) Data size (b) Rendering latency (c) PSNR

Figure 3: Performance of per-frame 3DGS and V³.

(a) Per-frame 3DGS (b) V³

Figure 4: Qualitative results.

3 Preliminary Experiments

The primary constraint of mobile video streaming is whether real-time rendering can be achieved under the given frame rate requirements. Failing to meet the specified time deadlines results in choppy playback, severely degrading the user experience. For instance, many mobile video applications consider 30 frames per second (FPS) as the threshold for real-time rendering [18, 21, 33, 49]. Furthermore, for mobile streaming, as long as real-time playback is ensured, reducing data size while preserving video quality is also important because 3DGS inherently requires large amounts of data.

Unfortunately, as discussed in Section 2.3, while numerous efforts have been made to optimize data size, the challenge of real-time rendering for full-scene 3DGS-based videos on mobile devices has barely been addressed in prior studies. To evaluate the existing methods with low rendering overhead for 3DGS-based mobile streaming, we conducted preliminary experiments using two approaches: per-frame 3DGS and V³. Per-frame 3DGS generates a static 3DGS representation independently for each frame without applying any temporal compression techniques, resulting in no decoding or additional rendering overhead, which leads to considerable data sizes. V³ is a recent approach specifically optimized for mobile environments. Our experiments involved two distinct volumetric videos: a single-object video and a full-scene video. The single-object video, `4K_Actor2_Dancing` from the HiFi4G dataset [16], features a single human without a background. In contrast, the full-scene video, `flame_steak` from the Neu3D dataset [26], represents a comprehensive scene containing multiple objects and a detailed background.

Figure 3 compares the performance of per-frame 3DGS and V³ in terms of (a) data size, (b) rendering latency, and (c) peak signal-to-noise ratio (PSNR). The rendering latency was measured on the Galaxy S25 smartphone. As shown in Figure 3a, per-frame 3DGS generates substantially large data sizes. The result underscores the need for an effective data size optimization technique for mobile streaming, such as V³ that notably reduces data size compared to per-frame 3DGS. However, for the full-scene video where the number of Gaussians is large, the methods fail to meet the target frame rates, as shown in Figure 3b. Additionally, V³ exhibits higher rendering latency compared to per-frame 3DGS, due to the additional decoding overhead required by its data-size optimization strategy. These findings indicate that no existing method can achieve real-time rendering on mobile devices when rendering all Gaussians in full-scene videos, despite being designed for low rendering overhead.

The results also indicate that V³ experiences quality degradation for full-scene videos due to its data size reduction process. As shown in Figure 3c, while V³ achieves a 0.32 dB higher PSNR than per-frame 3DGS for the single-object video, V³ exhibits a 1.20 dB lower PSNR for the full-scene video. Figure 4 illustrates this rendering quality degradation. This degradation is attributed to the motion estimation technique introduced by V³, which relies on a lightweight neural network primarily optimized for single-human scenarios and may not generalize well to complex full-scene videos involving multiple objects with diverse motion patterns. To sum up, the results highlight the need for a sophisticated mobile streaming solution that can ensure real-time rendering for full-scene volumetric videos while effectively optimizing data size and minimizing quality degradation.

4 Vega Overview

4.1 Goals and Approach

We propose Vega, a mobile volumetric video streaming system leveraging 3D Gaussian Splatting (3DGS) to provide photo-realistic, full-scene rendering. The primary goal of Vega is to ensure smooth playback at the target frame rate (e.g., 30 FPS). Under the time constraint, Vega also aims to reduce the data size while minimizing the rendering quality degradation.

As discussed in Section 3, existing methods with minimal rendering overhead fail to meet real-time rendering requirements for full-scene videos due to the large number of Gaussians. A promising strategy to address the challenge is to differentiate visually important from less important regions and allocate computational resources accordingly. For example, ViVo [15] introduced a strategy in mobile volumetric video streaming based on point clouds, distinguishing between important and less-important regions using visibility criteria. However, full-scene videos typically contain

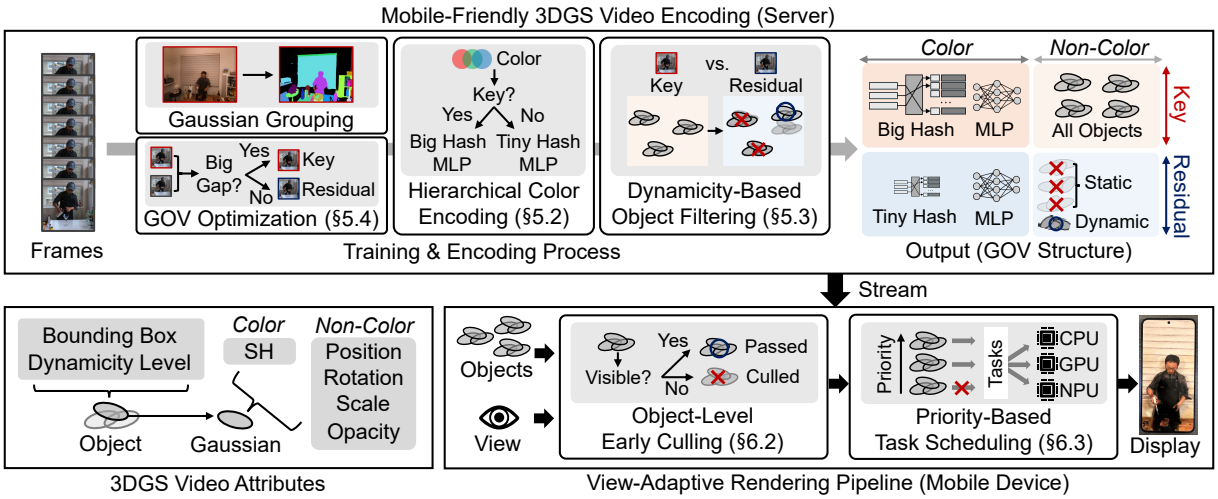


Figure 5: The overall architecture of Vega.

hundreds of thousands of Gaussians, leading to severe computational overhead when determining the importance of each Gaussian in real time.

To address this, Vega introduces object-level selective computation, which allocates computational resources primarily to visually important Gaussians that are grouped into objects. Although a scene consists of numerous Gaussians, those belonging to the same semantically meaningful object tend to move or rotate similarly. Leveraging this characteristic, Vega performs encoding and rendering at the object level, which is more efficient than handling Gaussians independently. In other words, Vega treats a full-scene video as a collection of multiple objects—including backgrounds—rather than as numerous independent Gaussians.

4.2 Challenges and Architecture

To provide mobile volumetric video streaming based on object-level selective computation, two key issues must be addressed. First, developing a new encoding scheme tailored for object-level prioritization is challenging. Existing dynamic 3DGS methods [27, 30, 38, 43, 45] are not suitable for object-level prioritization because they operate at the Gaussian level without providing object-level information. The scheme also needs to minimize the additional decoding and rendering overhead introduced by the data compression process. Second, designing an efficient rendering pipeline that operates based on object-level selective computation is challenging. The pipeline must dynamically determine the visual importance of objects in real time, which can vary greatly depending on factors such as object motion and user viewpoint. Moreover, efficient rendering requires maximizing the utilization of available processors on mobile devices, such as the CPU, GPU, and NPU.

Figure 5 shows the overall architecture of Vega. For encoding, Vega introduces a mobile-friendly encoding scheme, which supports object-level selective computation. For rendering, Vega proposes a view-adaptive rendering pipeline. Detailed explanations of the encoding scheme and rendering pipeline are presented in Sections 5 and 6, respectively.

5 Mobile-Friendly 3DGS Video Encoding

5.1 Overview and GOV Structure

Vega introduces a mobile-friendly 3DGS video encoding scheme designed to optimize the data size of 3DGS-based videos. Inspired by the widely used Group-of-Pictures (GOP) structure in 2D video encoding [10], the scheme introduces an analogous structure called Group-of-Volumes (GOV) for 3DGS-based videos. A GOV consists of a key frame and multiple residual frames, where the residual frames leverage information from the key frame to reduce overall data size. Additionally, to support object-level selective computation, the encoding scheme segments a 3DGS scene into multiple semantically meaningful objects using Gaussian Grouping [48] and compresses data at the object level. The compressed object-level data are transmitted to mobile devices for object-level selective computation.

To determine the data stored in key frames and residual frames that compose the GOV structure, the encoding scheme categorizes the diverse Gaussian attributes—SH coefficients, position, rotation, size, and opacity—into color-related and non-color data. The color-related attribute refers to SH coefficients. The non-color attributes, including position, rotation, size, and opacity, primarily represent the physical properties of each Gaussian. To optimize data size, this encoding scheme employs hierarchical color encoding, which significantly reduces the size of color data using hash

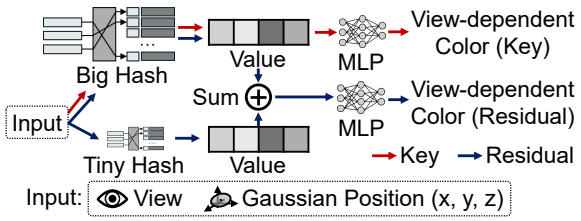


Figure 6: Hierarchical color encoding and decoding.

tables and MLP models, as color data alone constitutes more than half of the total size. Key frames utilize a large hash table, whereas residual frames minimize the hash table size to achieve data size optimization. For non-color data, the encoding scheme introduces dynamicity-based object filtering, which selectively stores information for objects in residual frames based on their dynamicity levels.

5.2 Hierarchical Color Encoding

To efficiently compress the color-related information in 3DGS-based videos, we introduce hierarchical color encoding that operates based on Instant NGP [32]. With Instant NGP, the SH coefficients, which are related to view-dependent color, are represented as relatively small hash tables and MLP models. While previous studies have employed Instant NGP to optimize the data size of 3DGS for static scenes [22] and dynamic scenes [23], these methods are not aware of the GOV structure. The hierarchical color encoding is an optimization technique specifically tailored for dynamic scenes based on the GOV structure.

Figure 6 shows the hierarchical color encoding and decoding pipeline. Hash tables and MLP models are used to capture the temporal variation of color, and the data size of hash tables is larger than that of MLP models. To reduce the overall size, the encoding scheme employs two types of hash tables: a big hash table for key frames and a tiny hash table for residual frames. The tiny hash is designed to have approximately 200 times smaller data size compared to the big hash. As shown in Figure 6, for key frames, an input consisting of the position and viewing direction is encoded into an embedding using the big hash, and then the MLP model predicts the view-dependent color. For residual frames, the tiny hash table encodes the residual color information relative to the corresponding key frame. The embedding input to the MLP model for the residual frame is computed by adding two embedding values: one from the big hash and another from the tiny hash for the residual frame.

The training process of hierarchical color encoding involves learning the hash tables and MLP models used for color prediction. During training, only the big hash is updated for key frames. For residual frames, only the tiny hash is updated while keeping the big hash fixed. By iteratively repeating these encoding and decoding processes during training, the big hash, tiny hash, and MLP models are effectively learned.

5.3 Dynamicity-Based Object Filtering

Vega employs dynamicity-based object filtering to effectively optimize the size of non-color data of 3DGS-based videos at an object level. The core problem of dynamicity-based object filtering is how to determine which objects within a frame should be treated as static and which objects should be treated as dynamic. Objects exhibiting minimal changes relative to the corresponding key frame can reuse the information from that key frame, thus reducing data size significantly.

To solve the problem, the filtering scheme introduces a method to estimate the dynamicity level of each object using the key frame and the immediately subsequent residual frame within a GOV. When training the residual frame right after a fully trained key frame, dynamic objects typically produce higher loss values and thus larger gradient magnitudes.

For a given camera view, the reconstructed 2D image I, which consists of RGB pixel values, is generated by a differentiable rendering function $\mathcal{R}$ as below:

$$I = \mathcal{R}(\Theta), \tag{1}$$

where Θ represents the set of parameters for all Gaussians in the scene.

The loss function used for training the 3DGS scene is defined as follows:

$$L(I) = \alpha \cdot \text{MSE}(I_{\text{gt}}, I) + \beta \cdot (1 - \text{SSIM}(I_{\text{gt}}, I)), \tag{2}$$

where I_{gt} denotes the ground-truth RGB image. α and β are weighting factors normally set to 0.8 and 0.2, respectively. MSE is the mean squared error, and SSIM is the structural similarity index measure.

Based on the loss function, the gradient magnitude $m(g)$ for a given Gaussian g and the gradient magnitude $M(O)$ for an object O are computed as follows:

$$m(g) = \frac{1}{K}\sum_{k=0}^{K-1}\left\|\nabla_{\theta_g} L_k\right\|, \quad M(O) = \frac{1}{n(O)}\sum_{g\in O} m(g), \tag{3}$$

where K is the number of training iterations, and $n(O)$ is the number of Gaussians in object O. The term $\nabla_{\theta_g} L_k$ represents the gradient of the total scene loss L at iteration k with respect to the parameters θ_g of a single Gaussian g. This term quantifies how a change in an individual Gaussian's parameters affects the final image loss, calculated via the chain rule. $\|\cdot\|$ indicates the L_2 norm of this gradient vector.

Finally, the following equations determine which objects are stored in residual frames:

$$\text{dyn}(O) = \frac{M(O)}{M_{\text{max}}}, \quad \text{dyn}(O) > M_{\text{threshold}}. \tag{4}$$

Here, $M_{\max}$ is the gradient magnitude of the most dynamic object in the current frame, and $\mathrm{dyn}(O)$ represents the normalized dynamicity level of object O. In practice, the normalized dynamicity is computed separately for each attribute of the object, and the final dynamicity level is obtained by averaging these values. Objects with a dynamicity level exceeding the threshold $M_{\text{threshold}}$, empirically set to 0.5, are considered dynamic, and only their attributes remain in the residual frame.

5.4 GOV Structure Optimization

To construct an optimized GOV structure, the encoding scheme must determine which frames to select as key frames and how long each group should be. The problem can be formulated similarly to the Rate-Distortion (RD) optimization used for optimizing the Group-of-Pictures (GOP) structure in 2D videos. The algorithm aims to minimize the overall cost of the video, computed as follows:

$$C(i) = R(i) + \lambda \cdot D(i), \tag{5}$$

where i denotes the frame index, $R(i)$ represents the data size of the frame, and $D(i)$ represents the distortion. λ is a parameter balancing the trade-off between rate and distortion, which we empirically set to 0.003.

For Vega, $D(i)$ is defined based on the sum of squared errors (SSE) between the original RGB image I_{gt} and the reconstructed RGB image I as follows:

$$D(i) = \sum_{p \in \Omega} \big(I_{\text{gt}}(p) - I(p)\big)^2, \tag{6}$$

where Ω denotes the set of pixels in the given image.

Key frames have relatively high R and low D, while residual frames have low R and high D. The RD optimization problem for the GOV structure can be simplified as follows:

$$\text{Minimize} \sum_{i=0}^{N-1} \Big(x(i) \cdot C_{\text{key}}(i) + \big(1 - x(i)\big) \cdot C_{\text{res}}(i) \Big), \tag{7}$$

where N is the total number of frames, $x(i)$ indicates whether the frame is selected as a key frame (1 for key frames, 0 for residual frames). C_{key} is the cost of a key frame, and C_{res} is the cost of a residual frame.

Exploring all possible cases to find the optimal solution is highly inefficient, as it would require training and evaluating numerous 3DGS frames individually. Therefore, Vega proposes a greedy approximation algorithm that minimizes the computation overhead. This algorithm relies on three assumptions. First, the very first frame of the video is always set as a key frame, and the second frame is always a residual frame. Second, within a single group, the residual frames have identical data rates R_{res} due to the operations of hierarchical color encoding and dynamicity-based object filtering. Third, within a group, the quality of residual frames farther from the key frame inherently decreases due to their increased difference from the key frame, but D_{res} might fluctuate across individual frames in practice.

The algorithm operates as follows. The cost of the first residual frame in a group is obtained directly by applying hierarchical color encoding and dynamicity-based object filtering. For subsequent residual frames, C_{key} is the cost value of the current group's initial key frame, while C_{res} is estimated using the average C_{res} of previous residual frames within a sliding window. The sliding window-based method mitigates the fluctuation of D_{res}. If the number of residual frames in the current group is smaller than the window size, the average C_{res} of all previous residual frames is used. The algorithm compares C_{key} and C_{res} for each new frame. If C_{key} is larger than C_{res}, the frame is set as a new key frame; otherwise, it is set as a residual frame. This algorithm significantly reduces the total computational overhead by estimating C_{res} with the sliding window. This approach avoids repeatedly applying hierarchical color encoding and dynamicity-based object filtering to each frame independently.

6 View-Adaptive Rendering Pipeline

6.1 Overview

On mobile devices, 3DGS-based videos are typically rendered through a sequential pipeline consisting of three main tasks: decoding, sorting, and rendering. If additional decoding steps (e.g., neural network inferences or hash table lookups) are necessary to represent temporal factors in 3DGS, they are performed first. For example, Vega's decoding process consists of hash table operations and MLP computations. The sorting task arranges the Gaussians based on their depth from the current viewpoint, which is necessary for correct alpha blending. Sorted Gaussians are then processed through the mobile graphics API [4] for final rendering. The rendering task performs frustum culling, projection, and alpha blending to generate the final rendered image.

Vega introduces a view-adaptive rendering pipeline to achieve real-time rendering on mobile devices. The main challenge of the rendering pipeline is efficiently identifying visually important objects that severely influence rendering quality under computational resource constraints. In addition, the pipeline needs to effectively assign its various tasks to the available processors on mobile devices. To address the issues, view-adaptive rendering pipeline introduces object-level early culling and priority-based task scheduling in its view-adaptive rendering pipeline.

6.2 Object-Level Early Culling

Among the tasks in Vega's rendering pipeline, hash table lookups take the longest time, followed by MLP inference,

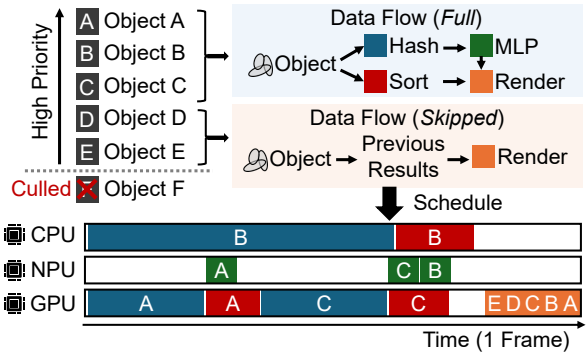


Figure 7: Priority-based task scheduling.

sorting, and rendering. Since hash table lookups are performed first in each frame, a strategy that minimizes hash computations is crucial for an efficient rendering pipeline. To address this, the view-adaptive rendering pipeline introduces object-level early culling at the beginning of each frame's processing. Although frustum culling is traditionally performed during the final rendering step using a mobile graphics API, this approach is highly inefficient because unnecessary Gaussians still go through expensive hash computations. In contrast, object-level early culling significantly improves computational efficiency by removing unnecessary Gaussians prior to hash operations.

The view frustum, which defines the visible region, consists of six planes in 3D space. By comparing the positions of Gaussians with these six plane equations, the system can determine whether each Gaussian is inside the view frustum. However, in full-scene volumetric videos containing a large number of Gaussians, performing this computation for every Gaussian in every frame is highly inefficient. To mitigate this overhead, the proposed early culling technique operates at the object level, enabling low-overhead culling per frame. Object-level early culling first reconstructs non-color attributes of objects within the current frame. Among these attributes, each object's bounding box, which is precomputed on the server, is included. By comparing this bounding box with the view frustum in 3D space, the system determines whether the object is visible or invisible in the current view.

6.3 Priority-based Task Scheduling

The view-adaptive rendering pipeline employs priority-based task scheduling to allocate more computation time to visually important objects within the current view, ensuring optimal rendering quality under real-time constraints. Figure 7 shows an example of priority-based task scheduling.

This scheduling strategy considers the characteristics of four key tasks—hash lookup, MLP inference, sorting, and final rendering—and the three processors available on mobile devices—CPU, GPU, and NPU. As shown in Figure 6, hierarchical color encoding imposes a dependency where MLP inference must always be executed after hash lookups. The rendering process, which typically runs on the GPU via a mobile graphics API, is performed only after all preprocessing tasks are completed. Since the NPU is optimized for neural networks, only MLP inference is assigned to the NPU. Hash lookups and sorting can be executed on either the CPU or GPU, depending on workload distribution.

Among the four tasks, the final rendering must be executed for all objects in every frame, while the other three tasks can reuse previous frame results. For example, in Figure 7, Objects D and E reuse previously computed data. The core challenge of priority-based task scheduling is determining which objects should not reuse previous results and instead undergo full computation in the current frame. To address this, we define a priority function for an object O as follows:

$$\text{prio}(O) = \text{dyn}(O) + \text{prox}(O) + \delta \cdot \text{age}(O). \tag{8}$$

Here, $\text{dyn}(O)$ is the normalized dynamicity level defined in Equation (4). $\text{prox}(O)$ represents how close the Gaussians forming an object are to the viewpoint and is normalized between 0 and 1. The rationale behind the equation is that highly dynamic objects require more frequent reconstruction, and objects closer to the viewpoint have a greater impact on visual quality. To prevent starvation, the priority function incorporates an aging factor $\text{age}(O)$, which represents the number of frames since the object was last fully computed, and a tunable aging coefficient δ, which we empirically set to 0.05.

Using the priority function, the problem of selecting objects for full computation in each frame is formulated as:

$$\begin{aligned} \text{Maximize} \quad & \sum_{O \in \text{frame}} \text{prio}(O) \cdot \left(y_{\text{CPU}}(O) + y_{\text{GPU}}(O)\right) \\ \text{Subject to} \quad & \sum_{O \in \text{frame}} y_{\text{CPU}}(O) \cdot T_{\text{CPU}}(O) \le T_{\text{deadline}}, \\ & \sum_{O \in \text{frame}} y_{\text{GPU}}(O) \cdot T_{\text{GPU}}(O) \le T_{\text{deadline}}. \end{aligned} \tag{9}$$

Here, $y_{\text{CPU}}(O)$ is 1 if the hash lookup and sorting operations for object O are assigned to the CPU; otherwise, $y_{\text{CPU}}(O)$ is 0. Similarly, $y_{\text{GPU}}(O)$ is 1 if these tasks are assigned to the GPU; otherwise, $y_{\text{GPU}}(O)$ is 0. The values $T_{\text{CPU}}(O)$ and $T_{\text{GPU}}(O)$ represent the estimated execution times for these operations on CPU and GPU, respectively. Since these values are proportional to the number of Gaussians in an object, they can be pre-modeled through profiling on mobile devices. The T_{deadline} represents the remaining available time after accounting for the final rendering task. Note that the MLP inference task is not considered in this problem formulation, as its execution time is shorter than hash lookups, and in practice, the NPU rarely reaches its computational limit.

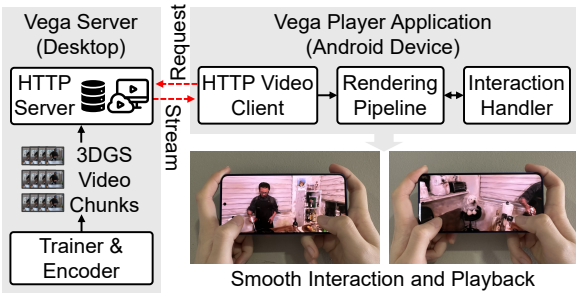
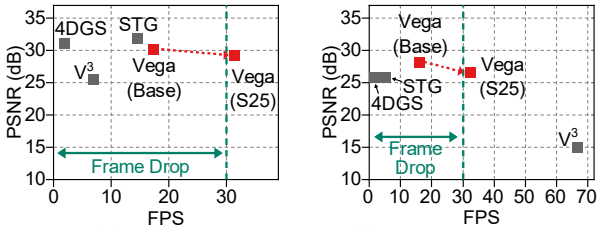


Figure 8: Vega implementation.



(a) Neu3D (b) Panoptic Sports

Figure 9: PSNR and FPS (Galaxy S25).

Given the limited computational resources of mobile devices, priority-based task scheduling employs a greedy approach to solve the problem efficiently. First, among the objects identified as visible during object-level early culling, the scheduler sorts them in descending order of prio(O). Then, starting from the highest-priority object, the algorithm checks whether its $T_{\text{CPU}}(O)$ and $T_{\text{GPU}}(O)$ fit within the remaining computational budget. If both CPU and GPU have available capacity, the object is assigned to the processor where the remaining time ratio (i.e., available budget divided by execution time) is smaller. If neither CPU nor GPU can accommodate the object, the object is skipped for that frame. Through this approach, priority-based task scheduling efficiently assigns high-priority objects to the CPU, GPU, and NPU within the given time constraints.

7 Implementation

The Vega implementation consists of the Vega server running on a Linux desktop PC and the Vega player application designed for Android devices. Figure 8 shows the end-to-end video streaming with the Vega implementation. In the server, the video encoder is implemented by extending the D-3DGS [30] and using PyTorch [34]. Residual frames are optimized using the motion modeling methodology of D-3DGS based on the scene generated from the key frame. Key frames undergo 15,000 training iterations with 5,000 densifications, while residual frames are trained with 2,000 iterations. Hierarchical color encoding is implemented using Tiny Cuda NN [3]. The video is structured as a sequence of frames composed of multiple objects, and it is streamed alongside a metadata file containing object-level information such as dynamicity values and bounding box coordinates. In our implementation, the big hash table has a size of 24 MB, the tiny hash table is 0.125 MB, and the MLP parameters occupy 0.03 MB. The server stores the video in GOV units and transmits video chunks to the client via HTTP.

The video player for Android smartphones is written in Java and C++. The player requests the video chunks from the server via HTTP, decodes them upon reception, and plays the video. The hash module reimplements the multi-resolution hash encoding technique from Tiny Cuda NN [3] to use the feature on Android devices. The GPU implementation of this module utilizes OpenGL ES [4] compute shaders. The MLP module is implemented using TensorFlow Lite [2], and MLP inference is delegated to the NPU via QNN [1]. For the sorting module, the CPU version uses quick sort, while the GPU version employs parallel radix sort with OpenGL ES compute shaders. The rendering module reimplements the CUDA-based rasterizer for 3DGS using OpenGL ES. Vega requires approximately 300 Mbps of network bandwidth for video streaming. We confirmed that the bandwidth requirement is fully supported in 5G network, which typically provides 500 Mbps to over 1 Gbps of downlink bandwidth [13, 20].

8 Evaluation

We evaluated Vega from three perspectives: (1) experiments comparing its end-to-end performance with existing methods, (2) an ablation study, and (3) a detailed analysis of Vega's policies.

8.1 End-to-End Performance

To evaluate the end-to-end performance of Vega, we conducted benchmark tests on real mobile devices: Samsung Galaxy S24 and S25 smartphones. The datasets used for the experiment are Neu3D [26] and Panoptic Sports [17]. The scenes in Panoptic Sports are more dynamic than those in Neu3D. The baselines include V^3, a mobile-centric method, and two server-level dynamic 3DGS methods: 4DGS [45] and STG [27]. Methods with extremely large data sizes, such as Per-frame 3DGS [19] and D-3DGS [30], were excluded as they are impractical for mobile streaming. For V^3 and Vega, FPS was measured using an actual Android application. For 4DGS and STG, which do not have real Android implementations, FPS was evaluated using Qualcomm AI Hub [5], which is known for precisely profiling neural networks on a testbed of actual mobile devices. The time for sorting and rendering was measured using our Android applications. For Vega, two cases were evaluated: rendering all Gaussians without considering FPS constraint (Base) and applying policies to ensure 30 FPS on each mobile device (S24 and S25).

Table 1: Quantitative results.

Model	Neu3D Compression Rate (%)	Neu3D PSNR↑ (dB)	Neu3D SSIM↑	Neu3D LPIPS↓	Neu3D FPS (S24)	Neu3D FPS (S25)	Panoptic Sports Compression Rate (%)	Panoptic Sports PSNR↑ (dB)	Panoptic Sports SSIM↑	Panoptic Sports LPIPS↓	Panoptic Sports FPS (S24)	Panoptic Sports FPS (S25)
4DGS	99.9%	31.22	0.936	0.148	1.4	1.9	99.1%	25.94	0.899	0.195	1.1	1.5
STG	99.5%	31.97	0.948	0.140	9.7	14.5	99.8%	25.84	0.910	0.143	3.6	5.1
V^3	98.7%	25.56	0.818	0.322	6.6	6.8	96.3%	15.02	0.449	0.606	59.1*	66.6*
Vega (Base)	98.9%	30.19	0.930	0.155	10.1	17.3	98.9%	28.26	0.907	0.154	8.5	16.0
Vega (S24)	–	28.11	0.913	0.159	**31.7**	–	–	26.18	0.877	0.193	**33.3**	–
Vega (S25)	–	29.19	0.923	0.156	–	**31.4**	–	26.64	0.881	0.185	–	**32.3**

* Although these cases exceed 30 FPS, this is due to an abnormally low number of Gaussians, resulting in severe quality degradation.

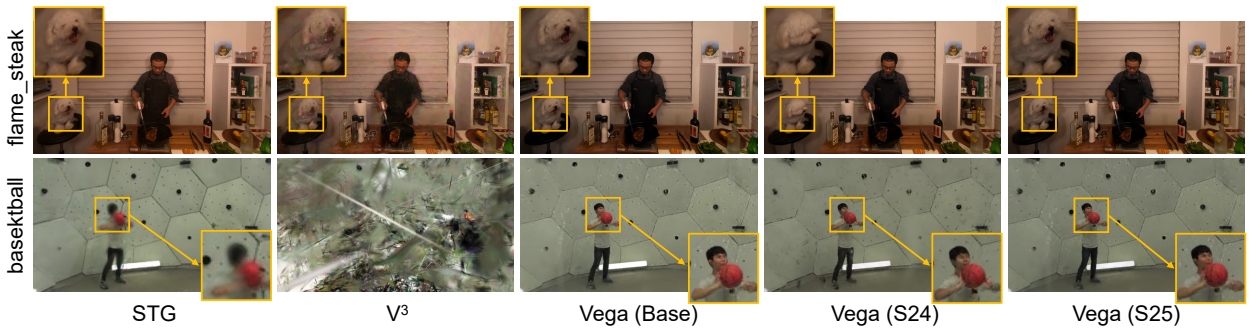


Figure 10: Qualitative analysis of solutions.
(`flame_steak` and `basketball` belong to the Neu3D and Panoptic Sports datasets, respectively)

Figure 9 shows the PSNR and FPS of each method on the Galaxy S25, and Table 1 presents the overall results. In the table, PSNR, SSIM, and the VGG-based LPIPS [53] were used as visual quality metrics. As shown in the table, all the methods used in the experiment achieved a compression rate of at least 98.7% when compressing per-frame 3DGS, effectively addressing the excessive data size issue of 3DGS-based videos. Meanwhile, in terms of FPS, Vega showed real-time rendering capability across all datasets. Although V^3 also achieved real-time rendering only on the Panoptic Sports dataset, it exhibited an extremely low PSNR of 15.02 dB. This suggests that the high FPS result was caused by a training issue, where an insufficient number of Gaussians were generated for the video. 4DGS and STG, on the other hand, suffer from high rendering latency, leading to severe frame drops and choppy playback. This issue would become even more pronounced when the user dynamically changes viewpoints, inevitably resulting in a degraded visual experience.

Figure 10 shows the qualitative results for STG, V^3, Vega (Base), Vega (S24), and Vega (S25). As shown in the figure, V^3 failed to represent the `basketball` scene in the Panoptic Sports dataset, highlighting its limitations. When comparing Vega (S24) and Vega (S25) to STG, the perceptual quality degradation in Vega was not significant. In the `flame_steak` scene on S24, the puppy looked different from other results because the region was reconstructed using previous frame results to maintain 30 FPS. On S25, however, the quality of this region was almost fully maintained. In the `basketball` scene, although STG achieved slightly better numerical metrics, Vega (S24) and Vega (S25) render the human figure more naturally than STG. These visual results suggest that Vega's strategy of focusing on dynamic objects is effective.

To sum up, Vega is the only solution capable of providing real-time rendering for full-scene 3DGS-based mobile volumetric video streaming while maintaining low data size and high visual quality. Vega achieves a high compression rate for 3DGS, similar to other methods. Moreover, Vega meets the given frame rate while minimizing perceptual quality degradation.

8.2 Ablation Study

Effect of mobile-friendly 3DGS video encoding. We conducted an ablation study to evaluate the effect of each technique presented in Section 5. This experiment includes four methods. The first method explicitly transmits SH coefficients without using a hash table (Base). The second method applies a big hash uniformly to all frames (Base+Big). The third method applies a tiny hash to residual frames (Base+Big+Tiny). The final method incorporates dynamicity-based object filtering (Vega). All methods train key frames for

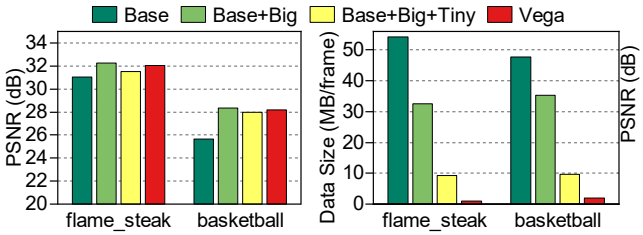


Figure 11: Ablation study on mobile-friendly 3DGS video encoding.

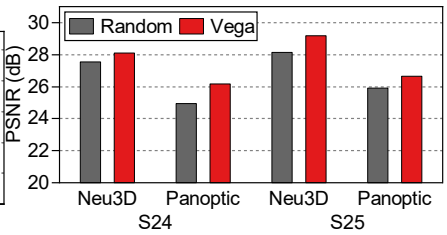


Figure 12: Random policy vs. priority-based task scheduling.

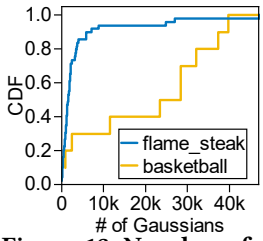


Figure 13: Number of Gaussians per object.

15,000 iterations and residual frames for 2,000 iterations. This experiment used the `flame_steak` sequence from the Neu3D dataset and the `basketball` sequence from the Panoptic Sports dataset.

Figure 11 presents the PSNR and data size per frame for each method. Interestingly, introducing the big hash reduces the data size by 39.87% and 26.03% for each video while improving PSNR by 1.24 dB and 2.70 dB, respectively. This PSNR improvement occurs because, in the Base method, SH coefficients must be directly learned instead of using hash, but the large number of SH coefficients makes training difficult. The hash table reduces the number of parameters that need to be trained, enabling more effective and efficient video learning. When the tiny hash is introduced, the data size decreases by 71.45% and 72.7% compared to the Base+Big method. Despite this reduction, the PSNR remains 0.49 dB and 2.34 dB higher than that of the Base method. Finally, applying dynamicity-based object filtering results in a 0.51 dB and 0.21 dB PSNR improvement compared to the Base+Big+Tiny method while reducing the data size by 88.86% and 80.18%, respectively. This result suggests that focusing on dynamic objects is more effective than processing all objects. As a result, Vega's encoding scheme achieves 1.00 dB and 2.55 dB higher PSNR than the Base while achieving 98.1% and 96.0% compression rates, demonstrating the superiority of this scheme.

Effect of view-adaptive rendering pipeline. We conducted an experiment to evaluate the effectiveness of Vega's view-adaptive rendering pipeline presented in Section 6. This experiment compares two methods for selecting objects to meet the 30 FPS constraint: random selection (Random) and priority-based selection (Vega). Figure 12 shows the PSNR of both approaches under the 30 FPS condition. On the Galaxy S24, Vega achieves 0.58 dB and 1.23 dB higher PSNR than Random for the Neu3D and Panoptic Sports datasets, respectively. On the Galaxy S25, Vega outperforms Random by 1.04 dB and 0.75 dB on the Neu3D and Panoptic Sports datasets, respectively. These results suggest that Vega's priority-based object selection is effective in maintaining higher visual quality while meeting real-time rendering constraints.

8.3 Detailed Analysis

We conducted a detailed analysis to justify the design choices of Vega's policies. In this analysis, we used the `flame_steak` sequence from the Neu3D dataset and the `basketball` sequence from the Panoptic Sports dataset. `flame_steak` contains 49 objects, while `basketball` contains 10 objects.

Number of Gaussians per object. We analyzed the number of Gaussians per object in both videos. Figure 13 illustrates the distribution of Gaussians across objects in each video. The average number of Gaussians per object in the `flame_steak` and `basketball` sequences is 10,130 and 25,210, respectively. This observation suggests that, compared to Gaussian-level policies, an object-level policy can significantly reduce computational cost. For example, in operations such as culling, performing computations at the Gaussian level requires processing over 200K Gaussians, whereas an object-level approach reduces the number of comparisons to just a few dozen objects, leading to substantial efficiency gains.

Data size of color and non-color data. We compared the sizes of color and non-color data in 3DGS-based videos. Figure 14 presents the sizes of color and non-color data in per-frame 3DGS and Vega. As shown in the figure, color accounts for 81.36% of the total size in per-frame 3DGS. This result indicates that Vega's encoding scheme, which separately compresses color and non-color data, is effective. Specifically, hierarchical color encoding reduces the size of color data by 99.0% and 99.4%, respectively, while dynamicity-based object filtering reduces the size of non-color data by 94.2% and 94.75%.

Distribution of dynamicity levels. We analyzed the dynamicity levels of various objects in each video. Figure 15 illustrates the distribution of dynamicity levels in the two videos. The results show that when setting the threshold value to 0.5, only about 20% of objects are classified as dynamic. These findings suggest that Vega's dynamicity-based object filtering can optimize data size by utilizing key frames for approximately 80% of static objects, instead of storing redundant information. Furthermore, the results highlight the importance of leveraging existing results for static objects in the view-adaptive rendering pipeline to maximize efficiency.

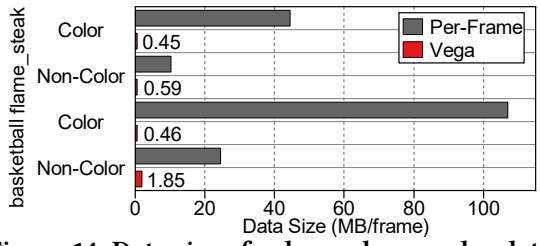


Figure 14: Data size of color and non-color data.

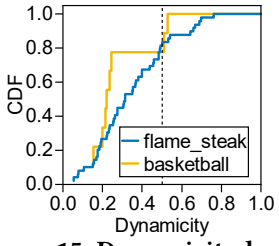


Figure 15: Dynamicity level.

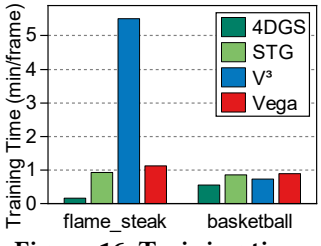


Figure 16: Training time.

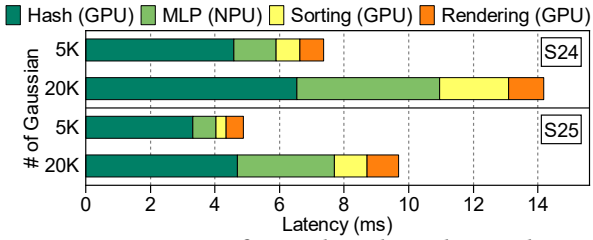


Figure 17: Latency for each task in the rendering pipeline for a single object.

Training time. We evaluated the training time of Vega, measured on a server with an NVIDIA GeForce RTX 3090 GPU, with methods used in Section 8.1. Figure 16 shows the training time of each method. In the `flame_steak` scene, Vega's training time is 1.1 minutes per frame, which is similar to STG's 0.9 minutes per frame and significantly lower than V^3's 5.5 minutes per frame. In the `basketball` scene, Vega's training time is 0.9 minutes per frame, comparable to V^3's 0.7 minutes per frame and STG's 0.9 minutes per frame. These results confirm that Vega maintains training time on par with existing methods while avoiding substantial overhead.

Rendering pipeline breakdown. We conducted an experiment to measure the latency of the four tasks in the rendering pipeline—hash, MLP, sorting, and rendering—on the Galaxy S24 and Galaxy S25. For this experiment, we selected objects with 5K and 20K Gaussians. MLP latency was measured on the NPU, while the latency of the other tasks was measured on the GPU. Figure 17 shows the results, indicating that hash table lookups take the longest time, followed by MLP inference, sorting, and rendering. These findings suggest that performing the culling process before hash lookups can improve overall performance. In addition, the results show that the latency of each task is proportional to the number of Gaussians, suggesting that the number of Gaussians can be used to estimate task latency in priority-based task scheduling.

Performance on heterogeneous processors. We conducted a study to assess how Vega's priority-based task scheduling performs under various hardware configurations: CPU, CPU+GPU, and CPU+GPU+NPU. Note that the CPU+GPU+NPU setup is the target configuration for Vega. In all configurations, the final rendering task is handled by the GPU, while the different configurations determine which processors are available for the remaining tasks. Table 2 shows the results. Note that Vega sustained a frame rate of over 30 FPS across all configurations. The results show that performance improves as more specialized processors are utilized. For instance, on the S25 with the `flame_steak` video, the CPU+GPU configuration improves the PSNR from 27.04 dB to 29.41 dB compared to the CPU-only configuration. Further enabling the NPU results in the highest PSNR of 30.16 dB. This suggests that Vega effectively leverages available processors to maximize rendering quality within the given time budget. The results also highlight Vega's robust performance on diverse hardware. Vega maintains the 30 FPS target on devices even without an NPU, accepting a slight quality trade-off to meet the real-time rendering goal.

Table 2: PSNR for various processor configurations.

Device	Configuration	PSNR (dB) `flame_steak`	PSNR (dB) `basketball`
S24	CPU	26.98	18.71
	CPU+GPU	29.30	26.34
	CPU+GPU+NPU	29.66	26.69
S25	CPU	27.04	18.81
	CPU+GPU	29.41	26.35
	CPU+GPU+NPU	30.16	26.97

9 Related Work

Mobile volumetric video streaming. Volumetric videos provide immersive experiences, but their large data size and high rendering overhead make delivering on mobile devices challenging. To address this, numerous studies have focused on optimizing data size and computational overhead for mobile volumetric video streaming. Some approaches propose view-adaptive streaming [14, 15, 25, 44, 47], while others leverage edge devices to assist with computation [28, 29, 35]. Another strategy involves performing 3D super-resolution on the client side to reduce bandwidth usage [51, 52]. However, this approach imposes a substantial computational burden on mobile devices. Meanwhile, MagicStream [8] and Fang et al. [12] adopt mesh-based volumetric content and employ neural rendering techniques to reconstruct smooth surfaces, but these methods primarily focus on accurately rendering a single human. Furthermore, recent studies [36, 50]

Table 3: Comparison between volumetric video studies.

	Photo realistic	Full scene	Mobile rendering
Conventional mobile volumetric video streaming (ViVo [15], Vues [28], etc.)			✓
Dynamic 3DGS methods (4DGS [45], STG [27], etc.), LTS [39]	✓	✓	
V^3 [43]	✓		✓
Vega	✓	✓	✓

have been conducted to provide photo-realistic full-scene volumetric video streaming via NeRF, but they require desktop-level computational power for real-time rendering.

Video streaming with 3DGS. With the emergence of 3DGS [19] in novel view synthesis, various efforts have been made to extend the technique for dynamic scene representation [27, 30, 38, 43, 45]. However, due to their large data sizes, long training times, and high rendering overhead, it is challenging to use these techniques for mobile video streaming. To address these issues, different approaches have been explored. LTS [39], for example, introduces an adaptive video streaming technique using layered 3DGS [37, 41], but its framework is not designed for real-time rendering on mobile devices. Another approach, V^3 [43], focuses on enabling real-time mobile rendering. This work presented an approach that encodes and decodes Gaussian attributes using the H.264 codec to compress data size effectively and introduced a method using a small neural network that estimates human movements to shorten training time. However, this approach targets object-based videos featuring a single human, making it difficult to apply to full-scene videos containing multiple objects and complex backgrounds.

Comparison between solutions. Table 3 compares existing volumetric video studies with Vega. Conventional mobile volumetric video streaming based on fixed representations [14, 15, 24, 25, 28, 29, 44, 47] can run on mobile devices; however, as discussed in Section 2.2, it is not suitable for photo-realistic full-scene content. Server-level dynamic 3DGS methods [27, 30, 38, 43, 45] and LTS [39] can deliver photo-realistic full-scene videos, but they fail to meet the required frame rates on mobile devices due to high rendering latency. V^3 [43] is a 3DGS-based technique designed for mobile devices, but it focuses on single-object videos. Vega is the only technique capable of delivering photo-realistic full-scene videos using 3DGS while ensuring smooth playback at the given frame rate on mobile devices.

10 Discussion

Impact of grouping accuracy. Vega's object-level policies require a scene to be segmented into semantic units, for which we employ the state-of-the-art Gaussian Grouping method [48]. While its accuracy is generally high, Vega's rendering quality can be influenced by segmentation errors. For instance, over-segmentation may increase computational overhead by creating more units to manage, and under-segmentation can lead to inaccurate dynamicity calculations due to averaging. Vega's policies ensure that the target frame rate is met even with such inaccuracies, with the only trade-off being a slight degradation in overall rendering quality.

Robustness to dynamic lighting. A potential concern regarding the gradient-based dynamicity discussed in Section 5.3 is its robustness under dynamic lighting conditions, where changes in illumination could be mistaken for physical motion. However, the dynamicity metric is calculated based on the gradient of non-color attributes (e.g., position, rotation, scale), and lighting variations mainly affect the color-related SH coefficients. Thus, Vega's motion estimation is not affected by changes in lighting conditions.

Potential for adaptive streaming. While Vega streams seamlessly in high-bandwidth 5G networks, as discussed in Section 7, a promising direction for future work is to support adaptive streaming under constrained or fluctuating network conditions. This can be achieved by controlling the rate-distortion trade-off using Vega's existing parameters, such as the dynamicity threshold (Section 5.3) and RD optimization settings (Section 5.4). Furthermore, integrating techniques like layered 3DGS [37, 39, 41] could enable more sophisticated adaptation for dynamic network environments.

11 Conclusion

To the best of our knowledge, Vega is the first attempt to provide fully immersive mobile volumetric video streaming with 3DGS. In particular, Vega introduces object-level selective computation to ensure the real-time rendering for 3DGS-based videos on mobile devices. To realize the selective computation, Vega employs mobile-friendly 3DGS video encoding to optimize data size and adopts a view-adaptive rendering pipeline to enable computationally efficient rendering on mobile devices.

Acknowledgments

This work was partly supported by Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. RS-2018-II180532, 50%) and the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. RS-2024-00344323, 50%)

References

[1] 2025. AI Engine Direct SDK. https://docs.qualcomm.com/bundle/publicresource/topics/80-63442-50/introduction.html

[2] 2025. LiteRT overview. https://ai.google.dev/edge/litert

[3] 2025. NVlabs/tiny-cuda-nn. https://github.com/NVlabs/tiny-cuda-nn

[4] 2025. OpenGL. https://www.opengl.org/

[5] 2025. Qualcomm AI Hub. https://aihub.qualcomm.com/

[6] António Baía Reis and Mark Ashmore. 2022. From video streaming to virtual reality worlds: an academic, reflective, and creative study on live theatre and performance in the metaverse. *International Journal of Performance Arts and Digital Media* 18, 1 (1 2022), 7–28. doi:10.1080/14794713.2021.2024398

[7] 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 (MobiSys '24)*. ACM, New York, NY, USA, 85–98. doi:10.1145/3643832.3661879

[8] Ruizhi Cheng, Nan Wu, Vu Le, Eugene Chai, Matteo Varvello, and Bo Han. 2024. MagicStream: Bandwidth-conserving Immersive Telepresence via Semantic Communication. In *Proceedings of the 22nd ACM Conference on Embedded Networked Sensor Systems (SenSys '24)*. ACM, New York, NY, USA, 365–379. doi:10.1145/3666025.3699344

[9] Andreas Dengel and Jutta Mägdefrau. 2018. Immersive Learning Explored: Subjective and Objective Factors Influencing Learning Outcomes in Immersive Educational Virtual Environments. In *2018 IEEE International Conference on Teaching, Assessment, and Learning for Engineering (TALE)*. 608–615. doi:10.1109/TALE.2018.8615281

[10] J.-R. Ding and J.-F. Yang. 2008. Adaptive group-of-pictures and scene change detection methods based on existing H.264 advanced video coding information. *IET Image Processing* 2, 2 (2008), 85–94. doi:10.1049/iet-ipr:20070014

[11] Erfan Entezami and Hui Guan. 2024. AI-Driven Innovations in Volumetric Video Streaming: A Review. https://arxiv.org/abs/2412.12208

[12] Tiancheng Fang, Chaoyue Niu, Yujie Sun, Chengfei Lv, Xiaotang Jiang, Ben Xue, Fan Wu, and Guihai Chen. 2024. An End-to-End, Low-Cost, and High-Fidelity 3D Video Pipeline for Mobile Devices. In *Proceedings of the 30th Annual International Conference on Mobile Computing and Networking (MobiCom '24)*. ACM, New York, NY, USA, 1162–1176. doi:10.1145/3636534.3690685

[13] Moinak Ghoshal, Pranab Dash, Zhaoning Kong, Qiang Xu, Y Charlie Hu, Dimitrios Koutsonikolas, and Yuanjie Li. 2022. Can 5G mmWave Support Multi-user AR?. In *Passive and Active Measurement*, Oliver Hohlfeld, Giovane Moura, and Cristel Pelsser (Eds.). Springer International Publishing, Cham, 180–196.

[14] Yongjie Guan, Xueyu Hou, Nan Wu, Bo Han, and Tao Han. 2023. MetaStream: Live Volumetric Content Capture, Creation, Delivery, and Rendering in Real Time. In *Proceedings of the 29th Annual International Conference on Mobile Computing and Networking (MobiCom '23)*. ACM, New York, NY, USA, 1–15. doi:10.1145/3570361.3592530

[15] 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 (MobiCom '20)*. ACM, New York, NY, USA, 1–13. doi:10.1145/3372224.3380888

[16] Yuheng Jiang, Zhehao Shen, Penghao Wang, Zhuo Su, Yu Hong, Yingliang Zhang, Jingyi Yu, and Lan Xu. 2024. HiFi4G: High-Fidelity Human Performance Rendering via Compact Gaussian Splatting. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 19734–19745. doi:10.1109/CVPR52733.2024.01866

[17] Hanbyul Joo, Tomas Simon, Xulong Li, Hao Liu, Lei Tan, Lin Gui, Sean Banerjee, Timothy Godisart, Bart Nabbe, Iain Matthews, Takeo Kanade, Shohei Nobuhara, and Yaser Sheikh. 2019. Panoptic Studio: A Massively Multiview System for Social Interaction Capture. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 41, 1 (2019), 190–204. doi:10.1109/TPAMI.2017.2782743

[18] Chanyoung Jung, Jeho Lee, Gunjoong Kim, Jiwon Kim, Seonghoon Park, and Hojung Cha. 2025. ARIA: Optimizing Vision Foundation Model Inference on Heterogeneous Mobile Processors for Augmented Reality. In *Proceedings of the 23rd Annual International Conference on Mobile Systems, Applications and Services (MobiSys '25)*. ACM, New York, NY, USA. doi:10.1145/3711875.3729161

[19] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkuehler, and George Drettakis. 2023. 3D Gaussian Splatting for Real-Time Radiance Field Rendering. *ACM Trans. Graph.* 42, 4 (2023), 1–14. doi:10.1145/3592433

[20] Z Jonny Kong, Qiang Xu, Jiayi Meng, and Y Charlie Hu. 2023. AccuMO: Accuracy-Centric Multitask Offloading in Edge-Assisted Mobile Augmented Reality. In *Proceedings of the 29th Annual International Conference on Mobile Computing and Networking (MobiCom '23)*. Association for Computing Machinery, New York, NY, USA, 1–16. doi:10.1145/3570361.3592531

[21] Jeho Lee, Chanyoung Jung, Jiwon Kim, and Hojung Cha. 2024. Panopticus: Omnidirectional 3D Object Detection on Resource-constrained Edge Devices. In *Proceedings of the 30th Annual International Conference on Mobile Computing and Networking (MobiCom '24)*. ACM, New York, NY, USA, 1207–1221. doi:10.1145/3636534.3690688

[22] Joo Chan Lee, Daniel Rho, Xiangyu Sun, Jong Hwan Ko, and Eunbyung Park. 2024. Compact 3D Gaussian Representation for Radiance Field. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 21719–21728. doi:10.1109/CVPR52733.2024.02052

[23] Joo Chan Lee, Daniel Rho, Xiangyu Sun, Jong Hwan Ko, and Eunbyung Park. 2024. Compact 3D Gaussian Splatting for Static and Dynamic Radiance Fields. https://arxiv.org/abs/2408.03822

[24] 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 (MobiCom '23)*. ACM, New York, NY, USA, 1–15. doi:10.1145/3570361.3592525

[25] 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 (MobiCom '20)*. ACM, New York, NY, USA, 1–14. doi:10.1145/3372224.3419214

[26] Tianye Li, Mira Slavcheva, Michael Zollhoefer, Simon Green, Christoph Lassner, Changil Kim, Tanner Schmidt, Steven Lovegrove, Michael Goesele, Richard Newcombe, and Zhaoyang Lv. 2022. Neural 3D Video Synthesis from Multi-view Video. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 5511–5521. doi:10.1109/CVPR52688.2022.00544

[27] Zhan Li, Zhang Chen, Zhong Li, and Yi Xu. 2024. Spacetime Gaussian Feature Splatting for Real-Time Dynamic View Synthesis. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 8508–8520. doi:10.1109/CVPR52733.2024.00813

[28] 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 (MobiCom '22)*. ACM, New York, NY, USA, 514–527. doi:10.1145/3495243.3517027

[29] Yu Liu, Puqi Zhou, Zejun Zhang, Anlan Zhang, Bo Han, Zhenhua Li, and Feng Qian. 2024. MuV2: Scaling up Multi-user Mobile Volumetric Video Streaming via Content Hybridization and Sharing. In *Proceedings of the 30th Annual International Conference on Mobile Computing and Networking (MobiCom '24)*. ACM, New York, NY, USA, 327–341. doi:10.1145/3636534.3649364

[30] Jonathon Luiten, Georgios Kopanas, Bastian Leibe, and Deva Ramanan. 2024. Dynamic 3D Gaussians: Tracking by Persistent Dynamic View

Synthesis. In *2024 International Conference on 3D Vision (3DV)*. 800–809. doi:10.1109/3DV62453.2024.00044

[31] 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. doi:10.1145/3503250

[32] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. 2022. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Trans. Graph.* 41, 4 (2022). doi:10.1145/3528223.3530127

[33] Seonghoon Park, Yeonwoo Cho, Hyungchol Jun, Jeho Lee, and Hojung Cha. 2023. OmniLive: Super-Resolution Enhanced 360° Video Live Streaming for Mobile Devices. In *Proceedings of the 21st Annual International Conference on Mobile Systems, Applications and Services (MobiSys '23)*. ACM, New York, NY, USA, 261–274. doi:10.1145/3581791.3596851

[34] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury Google, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf Xamla, Edward Yang, Zach Devito, Martin Raison Nabla, Alykhan Tejani, Sasank Chilamkurthy, Qure Ai, Benoit Steiner, Lu Fang Facebook, Junjie Bai Facebook, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems (NeurIPS '19)*. Curran Associates Inc., Red Hook, NY, USA, 8026–8037. doi:10.5555/3454287.3455008

[35] Feng Qian, Bo Han, Jarrell Pair, and Vijay Gopalakrishnan. 2019. Toward Practical Volumetric Video Streaming on Commodity Smartphones. In *Proceedings of the 20th International Workshop on Mobile Computing Systems and Applications (HotMobile '19)*. ACM, New York, NY, USA, 135–140. doi:10.1145/3301293.3302358

[36] Jianxin Shi, Miao Zhang, Linfeng Shen, Jiangchuan Liu, Yuan Zhang, Lingjun Pu, and Jingdong Xu. 2025. Implicit Representation-based Volumetric Video Streaming for Photorealistic Full-scene Experience. *ACM Trans. Multimedia Comput. Commun. Appl.* (2025). doi:10.1145/3728472

[37] Yuang Shi, Géraldine Morin, Simone Gasparini, and Wei Tsang Ooi. 2025. LapisGS: Layered Progressive 3D Gaussian Splatting for Adaptive Streaming. https://arxiv.org/abs/2408.14823

[38] 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 *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 20675–20685. doi:10.1109/CVPR52733.2024.01954

[39] Yuan-Chun Sun, Yuang Shi, Cheng-Tse Lee, Mufeng Zhu, Wei Tsang Ooi, Yao Liu, Chun-Ying Huang, and Cheng-Hsin Hsu. 2025. LTS: A DASH Streaming System for Dynamic Multi-Layer 3D Gaussian Splatting Scenes. In *Proceedings of the 16th ACM Multimedia Systems Conference (MMSys '25)*. ACM, New York, NY, USA, 136–147. doi:10.1145/3712676.3714445

[40] Theophilus Teo, Louise Lawrence, Gun A Lee, Mark Billinghurst, and Matt Adcock. 2019. Mixed Reality Remote Collaboration Combining 360 Video and 3D Reconstruction. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems (CHI '19)*. ACM, New York, NY, USA, 1–14. doi:10.1145/3290605.3300431

[41] Yi-Zhen Tsai, Xuechen Zhang, Zheng Li, and Jiasi Chen. 2025. L3GS: Layered 3D Gaussian Splats for Efficient 3D Scene Delivery. https://arxiv.org/abs/2504.05517

[42] Irene Viola and Pablo Cesar. 2023. Chapter 15 - Volumetric video streaming: Current approaches and implementations. In *Immersive Video Technologies*, Giuseppe Valenzise, Martin Alain, Emin Zerman, and Cagri Ozcinar (Eds.). Academic Press, 425–443. doi:10.1016/B978-0-32-391755-1.00021-3

[43] 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 Trans. Graph.* 43, 6 (2024), 1–13. doi:10.1145/3687935

[44] Yizong Wang, Dong Zhao, Huanhuan Zhang, Teng Gao, Zixuan Guo, Chenghao Huang, and Huadong Ma. 2024. Bandwidth-Efficient Mobile Volumetric Video Streaming by Exploiting Inter-Frame Correlation. *IEEE Transactions on Mobile Computing* 23, 10 (2024), 9410–9423. doi:10.1109/TMC.2024.3367750

[45] 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 *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 20310–20320. doi:10.1109/CVPR52733.2024.01920

[46] Nan Wu, Weikai Lin, Ruizhi Cheng, Bo Chen, Yuhao Zhu, Klara Nahrstedt, and Bo Han. 2025. Advancing Immersive Content Delivery with Dynamic 3D Gaussian Splatting. In *Proceedings of the 26th International Workshop on Mobile Computing Systems and Applications (HotMobile '25)*. ACM, New York, NY, USA, 109–114. doi:10.1145/3708468.3711886

[47] 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 (MobiSys '24)*. ACM, New York, NY, USA, 70–84. doi:10.1145/3643832.3661858

[48] Mingqiao Ye, Martin Danelljan, Fisher Yu, and Lei Ke. 2025. Gaussian Grouping: Segment and Edit Anything in 3D Scenes. In *Computer Vision - ECCV 2024*, Aleš Leonardis, Elisa Ricci, Stefan Roth, Olga Russakovsky, Torsten Sattler, and Gül Varol (Eds.). Springer Nature Switzerland, Cham, 162–179.

[49] Hyunho Yeo, Chan Ju Chong, Youngmok Jung, Juncheol Ye, and Dongsu Han. 2020. NEMO: Enabling Neural-enhanced Video Streaming on Commodity Mobile Devices. In *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking (MobiCom '20)*. ACM, New York, NY, USA, 1–14. doi:10.1145/3372224.3419185

[50] Daheng Yin, Jianxin Shi, Miao Zhang, Zhaowu Huang, Jiangchuan Liu, and Fang Dong. 2024. FSVFG: Towards Immersive Full-Scene Volumetric Video Streaming with Adaptive Feature Grid. In *Proceedings of the 32nd ACM International Conference on Multimedia (MM '24)*. ACM, New York, NY, USA, 11089–11098. doi:10.1145/3664647.3680908

[51] Anlan Zhang, Chendong Wang, Bo Han, and Feng Qian. 2021. Efficient Volumetric Video Streaming Through Super Resolution. In *Proceedings of the 22nd International Workshop on Mobile Computing Systems and Applications (HotMobile '21)*. ACM, New York, NY, USA, 106–111. doi:10.1145/3446382.3448663

[52] 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

[53] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. 2018. The Unreasonable Effectiveness of Deep Features as a Perceptual Metric. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 586–595. doi:10.1109/CVPR.2018.00068