Multi-frame Bitrate Allocation of Dynamic 3D Gaussian Splatting Streaming Over Dynamic Networks

Yuan-Chun Sun
National Tsing Hua University
Hsin-Chu, Taiwan
yuanjiun911@gapp.nthu.edu.tw

Yuang Shi
National University of Singapore
Singapore
yuangshi@u.nus.edu

Wei Tsang Ooi
National University of Singapore
Singapore
ooiwt@comp.nus.edu.sg

Chun-Ying Huang
National Yang Ming Chiao Tung University
Hsin-Chu, Taiwan
chuang@cs.nycu.edu.tw

Cheng-Hsin Hsu
National Tsing Hua University
Hsin-Chu, Taiwan
chsu@cs.nthu.edu.tw

Abstract

Dynamic 3D Gaussian splats have emerged as an exciting new data type for modeling interactive photo-realistic 3D scenes. This work considers the problem of bitrate allocation for streaming dynamic 3D Gaussian splats under dynamic network conditions. We model four parameters that influence the rate-distortion trade-offs for different attribute categories and propose an efficient Model-driven Gradient Ascent (MGA) algorithm to search for the optimal parameters that achieve high visual quality while keeping the bitrate below a given threshold across multiple frames. In our experiments, MGA achieves up to 5.46 dB in PSNR improvement over the baseline. We further proposed an adaptive MGA that reduces close to 3x computational time with negligible visual quality loss.

CCS Concepts

• **Information systems** → **Multimedia streaming**; • **Computing methodologies** → **Point-based models**.

Keywords

System design; Computer graphics; Bitrate allocation; Adaptive streaming; 3D Gaussian Splatting

ACM Reference Format:
Yuan-Chun Sun, Yuang Shi, Wei Tsang Ooi, Chun-Ying Huang, and Cheng-Hsin Hsu. 2024. Multi-frame Bitrate Allocation of Dynamic 3D Gaussian Splatting Streaming Over Dynamic Networks. In *SIGCOMM Workshop on Emerging Multimedia Systems (EMS '24), August 4–8, 2024, Sydney, NSW, Australia.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3672196.3673394

EMS '24, August 4–8, 2024, Sydney, NSW, Australia

ACM ISBN 979-8-4007-0711-7/24/08
https://doi.org/10.1145/3672196.3673394

1 Introduction

3D Gaussian splatting (3DGS) [8] is a new learning-based visual representation that learns from 2D photographs, allowing for interactive applications with photo-realistic static [8] and dynamic 3D scenes [12, 22]. A 3DGS frame may require GBs of data without compression [8], posing challenges to streaming 3DGS for remote visualization. An alternative is to render the scene on a server and stream the rendered images to the client. Doing so, however, incurs a higher interaction latency when the viewer changes its viewpoint. Furthermore, the client lacks 3D information for further rendering decisions, such as placing objects into the scene. To ensure bandwidth-efficient streaming of 3D scenes to the client, researchers have looked into reducing the size of 3D Gaussians by constraining the number of Gaussians and their attributes during the training phase to limit the size of the resulting frame while maximizing the visual quality of the synthesized novel views [2–4, 9, 11, 17, 20]. These approaches, however, lack the flexibility to generate Gaussians at different sizes dynamically, a vital requirement for rate control when streaming 3DGS scenes over a dynamic network.

Rate control is a classic problem in media streaming. Given a rate B_T bits per second, the sender decides which bits to send to maximize the quality of the stream at the receiver. This step needs to run in a tight control loop between the sender and receiver for interactive applications. Existing methods for controlling the size of the 3DGS scenes are too expensive and slow for real-time streaming, as they require re-training to generate a new set of 3D Gaussians [1, 10]. An alternative is to pre-train multiple representations of the same 3DGS scene, each at a different bitrate, and stream the representation with the largest bitrate smaller than B_T. This approach, however, may incur high training costs, consume excessive storage space, and fail to fully utilize the given bitrate B_T.

In this work, we present the first solution that addresses rate control for 3DGS streaming by formulating it as a bit allocation problem: *Given a sequence of pre-trained 3DGS scenes, what information should we drop to keep the data rate below the given constraint B_T while maximizing the quality of the rendered scenes?* In our solution, we first identified four encoding parameters and then allocated the bitrate across 3DGS frames using our proposed multi-frame bitrate allocation algorithm–Model-driven Gradient Ascent (MGA), which

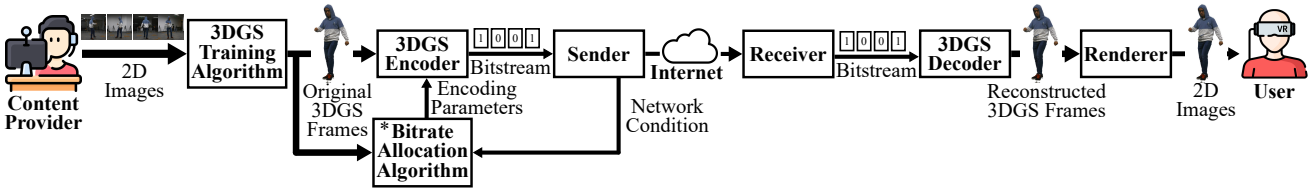


Figure 1: Architecture of a 3DGS streaming system. The bitrate allocation algorithm is our focus.

contains a fast, greedy, heuristic-based search strategy to explore the search space over an allocation window of frames to find the combination of the encoding parameters that leads to good visual quality and fits into the bandwidth constraint.

We augmented efficient Draco [5] to enable real-time dynamic 3DGS scene streaming and evaluated MGA through experiments using five dynamic 3DGS scenes and four real 5G network traces. Our extensive experiment results revealed that the proposed MGA algorithm: (i) effectively allocated the bit budget among 3DGS attributes across frames at diverse and dynamic bitrates; (ii) utilized up to 95.90% of bitrate and outperformed a baseline by 1.95 dB in PSNR, 0.04 in SSIM [6], and 5.09 in VMAF [16] on average, and 5.46 dB in PSNR, 0.20 in SSIM, and 25.52 in VMAF at most; (iii) improved the bandwidth utilization by 5.08% and visual quality by 0.65 dB in PSNR, 0.02 in SSIM, and 4.66 in VMAF with larger allocation windows; and (iv) led to reasonable resource consumption, including the number of encoding operations and running time. For resource-limited environments, we proposed a lighter-weight algorithm called Adaptive MGA (MGAA), which further reduces the number of encoding operations of MGA by up to 86.46% with negligible loss in visual quality.

2 Related Work

3DGS streaming imposes staggering storage requirements, which have been coped with by (i) training storage-efficient 3DGS representations of scenes and (ii) encoding pre-trained 3DGS scenes while streaming. Most existing works [2–4, 9, 11, 17, 20] proposed learning data-efficient representations of 3DGS scenes. For example, Fan et al. [2] pruned insignificant points and applied quantization to compress the 3DGS scenes. Girish et al. [4] compressed color and transformation attributes through latent quantization. Lu et al. [11] presented a hierarchical 3D Gaussian representation to represent the scene effectively. Some other studies consider the temporal redundancy of dynamic 3DGS scenes [7, 21]. Particularly, Katsumata et al. [7] reduced memory usage and overfitting by defining positions and rotations as time functions. Pranckevicius [19] decreases the size of Gaussians by encoding pre-trained 3DGS scenes. He proposed to prune unused data elements, reduce data precision, and encode texture images without considering scene geometry. His work did not consider the allocation of bits to different Gaussian attributes.

Some studies [1, 10] investigated the rate control of 3DGS compression during 3DGS training. Specifically, Liu et al. [10] introduced a hybrid primitive structure with a rate-constrained optimization scheme to reduce redundancies. Chen et al. [1] utilized the relations between unorganized 3D Gaussians and a structured hash grid for compression. Their solution employed entropy coding and adaptive quantization for compression. While the methods above effectively reduce the size of the 3DGS representation, they are applied during the training phase of 3DGS. This approach limits their applicability in rate-control scenarios.

3 3D Gaussian Splatting Streaming

3DGS models a scene using 3D Gaussians, with each Gaussian consisting of 62 attributes [8]. We classified these attributes into four categories: (i) geometry in three coordinates capturing the scene's structural composition; (ii) Spherical Harmonic (SH) coefficients representing view-dependent colors; (iii) opacity; and (iv) transformation, including scaling and rotation matrices. This categorization facilitates systematic bitrate allocation, as each category influences the rendering quality and storage size differently.

Fig. 1 shows a 3DGS streaming system compatible with various streaming protocols like DASH and Real-time Transport Protocol (RTP). A content provider produces a series of 3DGS frames captured and trained from multiple 2D images. The 3DGS frames are encoded before transmission. We extended Draco [5], a widely used real-time, geometry-based point cloud encoder, at this step as both point clouds and 3D Gaussians represent 3D spatial information with collections of points endowed with attributes such as position, color, and opacity. Note that we experimented with G-PCC [14] and found that existing G-PCC implementation is several orders of magnitude slower. Our algorithm is encoder-agnostic, however, and could be applied to G-PCC. In this extended encoder, we utilize four encoding parameters tailored to four categories: e_G for geometry, e_S for SH coefficients, e_O for opacity, and e_T for transformation. These parameters serve as inputs of 3DGS encoders to trade off visual quality and frame size. To find the best encoding parameters that maximize the quality of novel views under bandwidth constraints, the bitrate allocation algorithm adjusts the encoding parameters among different categories of attributes and across multiple 3DGS frames, driven by the Rate-Distortion (R-D) characteristics. After encoding, the sender transmits the compressed bitstream over the Internet to the receiver. The receiver decodes the bitstream and prepares the 3DGS for rendering and display.

Our work focuses on the bitrate allocation step, where the details are discussed in the following sections.

4 Bitrate Allocation

4.1 Problem Formulation

We address the multi-frame bitrate allocation problem in a recurring window of N 3DGS frames from a dynamic scene. This problem is solved once per allocation window. Let B_T be the bitrate budget of

the current allocation window, determined by system constraints such as network bandwidth. Our goal is to optimize the overall quality of synthesized novel views of 3DGS frames in the current allocation window while keeping the bitrate of the N frames below B_T.

The bitrate allocation algorithm needs to decide on the encoding parameters $\mathbf{e}_n = \langle e_{G,n}, e_{S,n}, e_{O,n}, e_{T,n} \rangle$ for each 3DGS frame, $n = 1, 2, \ldots, N$. The range of these encoding parameters depends on the codecs, which could be bit depth, quantization parameter, etc. We note that encoding parameters $e_{G,n}$, $e_{S,n}$, $e_{O,n}$, and $e_{T,n}$ tradeoff between the quality contribution and the number of bits used to encode each of the geometry, SH, opacity, and transformation attributes of the n-th 3DGS frame, respectively. We denote the functions that map $\mathbf{e}_n$ to the visual quality and frame size as $Q_n(\mathbf{e}_n)$ and $R_n(\mathbf{e}_n)$, respectively. These two functions are generally nonlinear and depend on factors such as scene complexity and codec efficiency, and thus could vary across different frames.

With the notations developed above, we write our problem into a constrained optimization problem:

$$\mathbf{e}^* = \langle \mathbf{e}_1^*, \mathbf{e}_2^*, \ldots, \mathbf{e}_N^* \rangle = \underset{\langle \mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_N \rangle}{\operatorname{argmax}} \sum_{n=1}^{N} Q_n(\mathbf{e}_n) \quad \text{subject to:} \sum_{n=1}^{N} R_n(\mathbf{e}_n) \leq B_T. \tag{1}$$

Here, the objective function is the average visual quality across all 3DGS frames. Other objective functions, such as max-min fairness, may also be incorporated for different optimization criteria based on usage scenarios.

4.2 Proposed Algorithm

Solving the multi-frame 3DGS bitrate allocation problem in Eq. (1) is no easy task, mainly because of the non-linear mappings $Q_n(\mathbf{e}_n)$ and $R_n(\mathbf{e}_n)$. These two mappings could be derived by encoding 3DGS frame n with a probing $\mathbf{e}_n$ in a trial-and-error manner, searching for the best encoding parameters. Such brute-force attempts, however, lead to high computation overhead. Hence, we propose an iterative search algorithm that systematically selects the probing encoding parameters for a balanced: (i) accuracy of $Q_n(\mathbf{e}_n)$ and $R_n(\mathbf{e}_n)$ and (ii) encoding time complexity due to probing attempts.

Design rationales. Our proposed algorithm is called *Model-driven Gradient Ascent* (MGA), as it employs ML-based models for $Q_n(\mathbf{e}_n)$ and $R_n(\mathbf{e}_n)$ to effectively refine the encoding parameters based on the gradient of visual quality over frame size (i.e., $\nabla = \Delta Q / \Delta R$) among all possible changes on the encoding parameters. The MGA algorithm consists of two phases: (i) exponential search for quickly zooming into a region close to $\mathbf{e}^*$ with an acceptable number of probing attempts and (ii) linear search for further approaching $\mathbf{e}^*$ using ML-based models to avoid excessive probing attempts. We introduce these two phases in the following.

Exponential search. There are two system parameters in this phase: (i) initial encoding settings $\dot{\mathbf{e}} = \langle \dot{\mathbf{e}}_1, \dot{\mathbf{e}}_2, \ldots \dot{\mathbf{e}}_N \rangle$ for individual frames and (ii) a scaling factor $\lambda > 1$, $\lambda \in \mathbb{R}^+$ shared by all frames. Our exponential search consists of two steps. In the first step, we perform per-window scaling by scaling the current encoding setting $\mathbf{e}$ up or down, depending on whether the consumed bitrate $\sum_{n=1}^{N} R_n(\mathbf{e}_n)$ exceeds B_T. More specifically, we set $\mathbf{e} = \dot{\mathbf{e}}$ and then update $\mathbf{e}$ with:

$$\begin{cases} \lfloor \lambda \mathbf{e} \rfloor = \langle \lfloor \lambda \mathbf{e}_1 \rfloor, \lfloor \lambda \mathbf{e}_2 \rfloor, \ldots, \lfloor \lambda \mathbf{e}_N \rfloor \rangle, & \text{if } \sum_{n=1}^{N} R_n(\mathbf{e}_n) < B_T; \\ \lfloor \mathbf{e}/\lambda \rfloor = \langle \lfloor \mathbf{e}_1/\lambda \rfloor, \lfloor \mathbf{e}_2/\lambda \rfloor, \ldots, \lfloor \mathbf{e}_N/\lambda \rfloor \rangle, & \text{otherwise.} \end{cases}$$

The per-window scaling stops when $\sum_{n=1}^{N} R_n(\mathbf{e}_n) \leq B_T$ and $\sum_{n=1}^{N} R_n(\lfloor \lambda \mathbf{e}_n \rfloor) > B_T$. We then move into the second step for per-frame scaling. In this step, we scale across all frames in the allocation window by identifying the most promising $\tilde{n} \in \{1, 2, \ldots, N\}$ and category $\tilde{C} \in \{\mathrm{G}, \mathrm{S}, \mathrm{O}, \mathrm{T}\}$ once in every iteration. In each iteration, we go through each frame n ($n = 1, 2, \ldots, N$), probing each of the four encoding parameters $\hat{\mathbf{e}}_{\tilde{C},n}$, by updating $\mathbf{e}_{\tilde{C},n}$ with $\lfloor \lambda \mathbf{e}_{\tilde{C},n} \rfloor$, Next, we compute:

$$\nabla_{C,n} = \frac{\Delta Q_{C,n}}{\Delta R_{C,n}} = \frac{Q_n(\hat{\mathbf{e}}_C) - Q_n(\mathbf{e})}{R_n(\hat{\mathbf{e}}_C) - R_n(\mathbf{e})}, \text{where } C \in \{\mathrm{G}, \mathrm{S}, \mathrm{O}, \mathrm{T}\}.$$

We then let $\langle \tilde{n}, \tilde{C} \rangle = \operatorname{argmax}_{n \in \{1,2,\ldots,N\}, C \in \{\mathrm{G},\mathrm{S},\mathrm{O},\mathrm{T}\}} \nabla_{C,n}$. After updating $\mathbf{e}_{\tilde{n}}$ with $\mathbf{e}_{\tilde{C},\tilde{n}} = \lfloor \lambda \mathbf{e}_{\tilde{C},\tilde{n}} \rfloor$, we move to the next iteration. The iterative process stops when: (i) none of the frames in $\{1, 2, \ldots, N\}$ and categories in $\{\mathrm{G}, \mathrm{S}, \mathrm{O}, \mathrm{T}\}$ lead to quality improvement or (ii) further scaling $\tilde{C}$ of $\tilde{n}$ results in excessive bitrate $> B_T$.

Linear search. At the end of the exponential search, we have the best-known encoding parameter $\bar{\mathbf{e}}$, so that $\sum_{n=1}^{N} R_n(\bar{\mathbf{e}}_n) \leq B_T$. If the resulting bitrate is exactly B_T, we are done. Otherwise, we perform an additional linear search from $\mathbf{e} = \bar{\mathbf{e}}$ trying to approach $\mathbf{e}^*$ by incrementing the most promising category $\tilde{C}$ of each frame by a step size $\eta \in \mathbb{N}$, which is another system parameter. While linear search examines finer-grained encoding parameters, probing so many encoding parameters for $Q_n(\mathbf{e}_n)$ and $R_n(\mathbf{e}_n)$ is not computationally feasible. Hence, for each frame n ($n = 1, 2, \ldots, N$), we consider 2^4 probing encoding parameters $\langle \hat{e}_{G,n}, \hat{e}_{S,n}, \hat{e}_{O,n}, \hat{e}_{T,n} \rangle$, where $\hat{e}_{G,n} \in \{e_{G,n}, \lfloor \lambda e_{G,n} \rfloor\}$, $\hat{e}_{S,n} \in \{e_{S,n}, \lfloor \lambda e_{S,n} \rfloor\}$, $\hat{e}_{O,n} \in \{e_{O,n}, \lfloor \lambda e_{O,n} \rfloor\}$, and $\hat{e}_{T,n} \in \{e_{T,n}, \lfloor \lambda e_{T,n} \rfloor\}$. Given $Q_n(\hat{\mathbf{e}}_n)$ and $R_n(\hat{\mathbf{e}}_n)$ from these 16 probing samples, we build ML-based models $\tilde{Q}_n(\mathbf{e}_n)$ and $\tilde{R}_n(\mathbf{e}_n)$ to avoid high probing overhead.

The actual iterations of linear search are almost identical to the per-frame scaling presented above, with two distinctions. First, we compute gradients $\nabla_{C,n}$ using $\tilde{Q}_n(\mathbf{e}_n)$ and $\tilde{R}_n(\mathbf{e}_n)$ and then select $\tilde{n}$ and $\tilde{C}$ with the largest gradient. Second, at the end of each iteration, we update $\mathbf{e}_{\tilde{n}}$ with $e_{\tilde{C},\tilde{n}} = e_{\tilde{C},\tilde{n}} + \eta$. Other aspects of the linear search, such as the stopping criteria, are the same as the per-frame scaling in the exponential search. Upon meeting the stopping criteria, the MGA algorithm returns the most recent $\mathbf{e}$ as $\mathbf{e}^*$.

5 Experiments

5.1 Setup

Implementations. We implemented our algorithm with Draco [5], controlling the quantization of 3DGS attributes by $\mathbf{e}$. Draco then uses a K-D tree for geometry and predictive coding for other attributes. It employs entropy coders with static and dynamic codebooks afterward.

We have implemented our MGA algorithm in Python. For comparison, we have also implemented two anchor algorithms: (i) *Static*, which let $\mathbf{e}_{C,n} = 8$ for all $C \in \{\mathrm{G}, \mathrm{S}, \mathrm{O}, \mathrm{T}\}$ and $n = 1, 2, \ldots, N$, which is the default settings of Draco, and (ii) *Uniform (Uni)*,

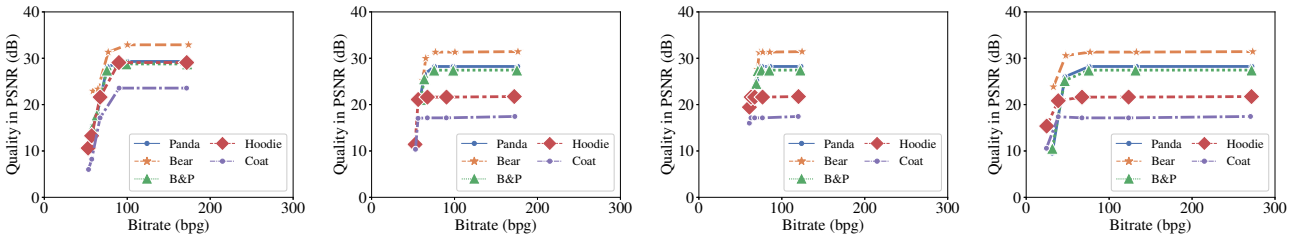


Figure 2: R-D curves for the sample frame of different 3DGS scenes by individually varying: (a) e_G, (b) e_S, (c) e_O, and (d) e_T.

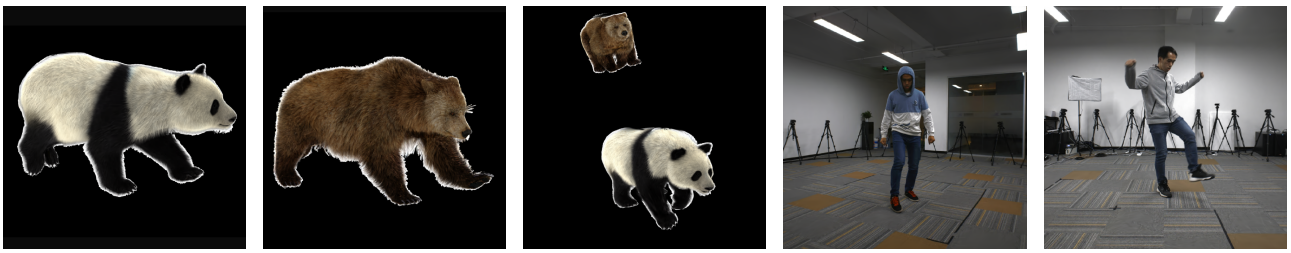


Figure 3: Sample novel views from: (a) Panda, (b) Bear, (c) P&B, (d) Hoodie, and (e) Coat.

Table 1: Dynamic 3DGS Scenes in Our Experiments

Scene	Synthetic			Natural	
	Panda	Bear	P&B	Hoodie	Coat
Per-Frame No. Gau. (10^6)	0.165	0.222	0.356	0.532	0.851
Average Frame Size (MB)	39.00	52.27	84.21	131.85	211.06

which first assigns B_T/N bit budget to each 3DGS frame n, assume $e_{G,n} = e_{S,n} = e_{O,n} = e_{T,n}$, and exhaustively tests all possible $\mathbf{e}_n$ values for the feasible encoding parameters with the highest visual quality.

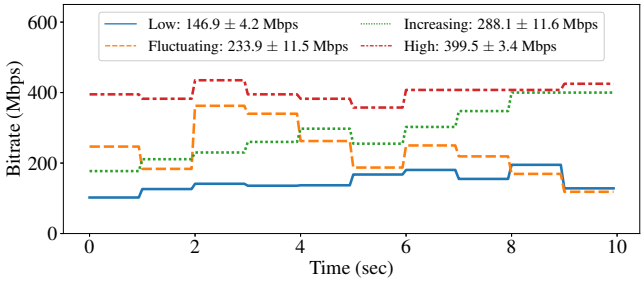


Figure 4: Per-frame bitrate of different network traces.

Datasets. Table 1 lists the dynamic 3DGS scenes used in our experiments, including three synthetic [13] and two natural scenes [23] with diverse characteristics. In the synthetic scenes, Panda and Bear, a panda and a bear run horizontally within an $R \times R \times R$ box. The scenes were merged into a Panda&Bear (P&B) scene, where the animals run across the scene alternatively. In both natural scenes, the avatars wave their arms and kick at the center without transitional movement. For the scene, we trained ten 3DGS frames using the vanilla 3DGS algorithm[1] [8] with SH degree 0, as we found that higher SH degrees did not significantly improve view quality. Fig. 3 presents sample novel views of the dataset. We streamed and looped the 3DGS frames in our 10-sec experiments at 15 fps. To evaluate visual quality, each scene is rendered at 720p from 24 virtual cameras uniformly placed around the main subject. Note that these 3DGS scenes exhibit diverse characteristics in terms of rate-distortion trade-offs as shown in Fig. 2. To generate the R-D curves in this figure, we set the default value of e to 8, and individually adjust e_G, e_S, e_O, and e_T in the range of {2, 4, 8, 16, 32} for sample frame of different scenes. The quality in PSNR and the bitrate in bits per Gaussian (bpg) for each encoding parameter were then calculated.

For available bitrates over time, we adopt real Lumos5G [15] network traces captured from a commercial 5G mmWave cell. We chose four 10-sec network traces, named *High* (at 399.5 Mbps on average with a 95% confidence interval of ±3.4 Mbps), *Low* (146.9 ± 4.2 Mbps), *Increasing* (288.1 ± 11.6 Mbps), and *Fluctuating* (233.9 ± 11.5 Mbps)[2], as we shown in Fig 4. *High* and *Low* are two steady network traces in good and bad network conditions, respectively. *Increasing* comes with increasing bitrate at a smooth pace. *Fluctuating* has highly varying bitrates over time.

[1] Our method can be applied to enhanced 3DGS algorithms, which is one of our future tasks.

[2] Because the dataset logs the total bandwidth of multiple mobile users, we divide the total bitrate by four.

To model $\tilde{Q}_n(\mathbf{e}_n)$ and $\tilde{R}_n(\mathbf{e}_n)$, we employed PSNR as a sample quality metric and experimented with various machine learning models from Scikit-Learn [18] with default parameters. We chose Random Forest for $\tilde{Q}_n(\mathbf{e}_n)$; and Linear Regression for $\tilde{R}_n(\mathbf{e}_n)$, as they achieve the best accuracy. Our models achieve: (i) RMSE between 1.380 and 2.341, (ii) MAE between 0.768 and 2.028, and (iii) R^2 between 0.891 and 0.932 across all considered 3DGS scenes by uniformly splatting 100 encoding parameters in each frame.

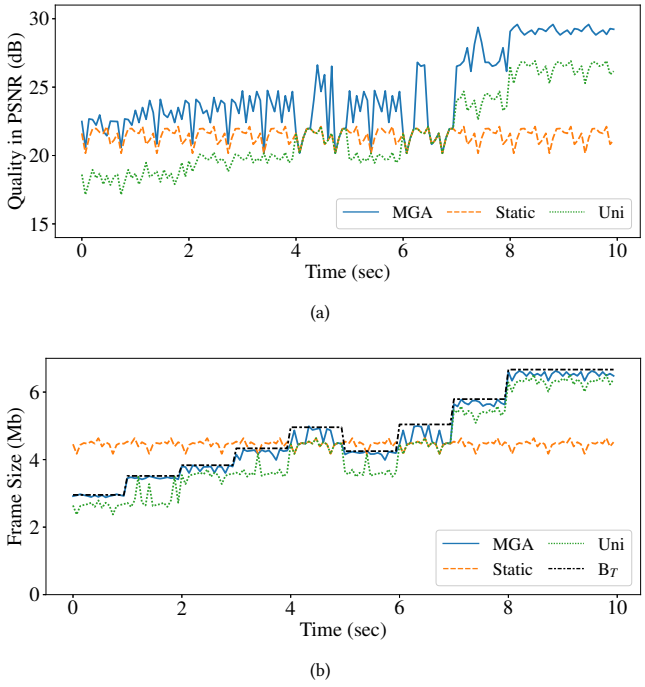


Figure 5: Per-frame performance results: (a) quality in PSNR and (b) frame size. Sample results from Hoodie under trace Increasing are shown.

Settings. All experiments are run with AMD 32-core CPUs at 2.80 GHz and an NVIDIA GTX-3090 GPU with 24 GB VRAM. We vary the allocation window $N = \{1, 5, 15, 30\}$, and let $N = 1$ by default. We set $\lambda = 2$ and $\delta = 1$. For each 3DGS frame, we measure: (i) visual quality in PSNR, SSIM, and VMAF and take the lossless scenes as the reference; (ii) resulting bitrate $\sum_{n=1}^{N} R_n(\mathbf{e}_n)$; and (iii) computational overhead in the number of probing attempts and running time of bitrate allocation methods. We report the average results with 95% confidential intervals if applicable.

5.2 Results

Necessity of adaptive bitrate allocation algorithms. Fig. 5 presents sample frame-by-frame performance results of streaming Hoodie under trace *Increasing*, where the bitrate B_T increases over time. Results from other dynamic 3DGS senses are similar. We note two observations. First, at high bitrates (7–10 s, an average of 6.39 Mb), Fig. 5(a) shows that our proposed MGA achieves superior expected quality of 38.53 dB in PSNR, 0.90 in SSIM, and 53.69 in VMAF[3]. This contrasts with Static, which only achieves 21.43 dB in PSNR, 0.71 in SSIM, and 0.58 in VMAF. Additionally, as shown in Fig. 5(b), at high bitrates (7–10 s), MGA utilizes 97.91% of the bitrate budget, well above Static's 70.15%, indicating more efficient use of network resources by MGA. Conversely, at low bitrates (0–3 s, averaging 3.43 Mb), Static not only delivers lower quality of 21.43 dB in PSNR but also exceeds its bitrate allocation, using up to 156.94% of the allocated budget. This inefficiency underlines Static's inability to adjust to varying network conditions effectively. Therefore, Static is excluded from further analysis, as shown in Fig. 5.

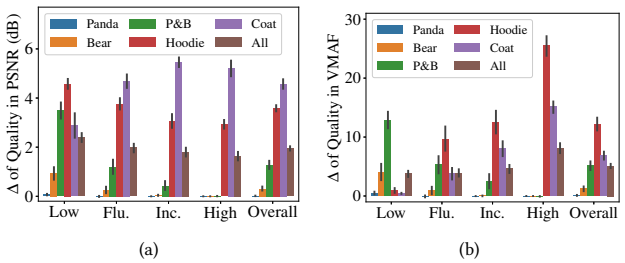


Figure 6: Overall visual quality improvement of MGA over Uni under different network traces in: (a) PSNR and (b) VMAF. SSIM is skipped for the sake of space.

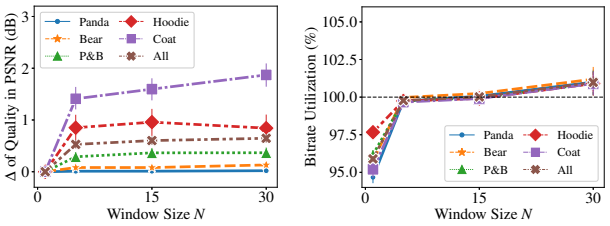
Necessity of non-uniform bitrate allocation algorithms crosses categories. Fig. 5 also reveals that, compared to Uni, MGA improves the visual quality by up to 6.75 dB in PSNR, 0.19 in SSIM, and 57.03 in VMAF. Moreover, MGA utilizes at most 8.11% more bitrates. To gain further insights into the performance of MGA over Uni, we plot the overall quality improvement of all dynamic 3DGS scenes (sorted by the number of Gaussians) under different network traces in Fig. 6. We observe that: (i) As scene complexity increases, the improvement in visual quality also rises. Considering all scenes and network traces (the right-most bar), MGA outperforms Uni by 1.95 dB in PSNR, 0.04 in SSIM, and 5.09 in VMAF on average, with peak improvements reaching 5.46 dB in PSNR, 0.2 in SSIM, and 25.52 in VMAF, which is made possible because of 5.61% higher bitrate utilization (not shown in the figure). (ii) At high bitrates, MGA shows minimal gains for simpler scenes like Panda, Bear, and P&B, delivering high visual quality at 29.82, 32.77, and 29.98 dB in PSNR, respectively, as sufficient bitrates are available. Conversely, for complex scenes such as Coat at low bitrates, MGA shows negligible improvement, as it already utilizes much of the bitrate (90.35%). Overall, Fig. 6 illustrates MGA's effectiveness in strategically allocating bitrates to critical visual attributes.

Impact of N. We increase the allocation window size N from 1 (default, used above) to 30. Fig. 7 reports the overall quality gains and overhead increases compared to $N = 1$. We have three observations. First, Fig. 7(a) depicts that the visual quality generally increases with larger N. The overall visual quality increases are 0.65 dB in PSNR, 0.02 in SSIM, and 4.66 in VMAF. The gaps enlarge with more complex scenes, e.g., Coat leads to increases of 1.8 dB in PSNR, 0.06 in SSIM, and 7.79 in VMAF. These enhancements stem

[3]Due to space limit, some figures on SSIM and VMAF are omitted.

Table 2: Average Visual Quality and Overhead Across Four Network Traces

Algorithm	Panda	Bear	P&B	Hoodie	Coat
Quality in PSNR (dB) / SSIM / VMAF					
Exhaustive	29.78/0.97/41.69	32.51/0.98/45.76	28.73/0.95/37.91	24.77/0.81/26.08	18.54/0.65/8.48
MGA	29.78/0.97/41.71	32.47/0.98/46.26	28.76/0.95/37.84	24.64/0.81/23.10	17.90/0.62/8.06
MGAA	29.76/0.97/41.49	32.39/0.98/45.89	28.63/0.95/36.90	24.45/0.80/23.02	17.84/0.62/8.00
Overhead in Per-frame Probing Attempts (times) / Running Time (ms)					
Exhaustive	625.00/5.08	625.00/4.99	625.00/5.00	625.00/4.96	625.00/4.93
MGA	19.79/92.75	21.68/97.54	24.83/104.21	25.30/100.28	26.56/95.60
MGAA	12.14/20.46	12.49/14.14	14.55/14.11	16.10/24.54	17.68/26.46



(a) (b)

Figure 7: Performance difference when increasing N of MGA: (a) visual quality and (b) bitrate utilization. Overall results from all traces are reported.

from the additional flexibility in bitrate allocation across frames as N increases. Second, Fig. 7(b) indicates a slight bandwidth over-utilization by 0.98% due to inaccuracies in the models $\tilde{Q}_n(\mathbf{e}_n)$ and $\tilde{R}_n(\mathbf{e}_n)$ during the linear search phase. Last, a notable 65.89% increase in running time occurs with larger window sizes, suggesting a trade-off between improved quality and higher time and buffering delays. In summary, Fig. 7 exhibits the potential (and cost) of using larger N with MGA.

MGA approaches optimal visual quality with fewer probing attempts. We check the optimality of MGA by comparing it against *Exhaustive* search, which opts for $5^4 = 625$ probing encoding parameters following the same factor λ as MGA. We note that Exhaustive is a good approximation to $\mathbf{e}^*$, which cannot be efficiently found due to the non-linear nature of $Q_n(\mathbf{e}_n)$ and $R_n(\mathbf{e}_n)$. Table 2 gives the average visual quality and overhead from Exhaustive and MGA across all network traces with $N = 1$. We observe that: (i) MGA results in very close visual quality to that of Exhaustive, e.g., PSNR > 96.45% has been achieved; (ii) MGA could lead to visual quality higher than Exhaustive, which can be attributed to fine-grained model-driven linear search; and (iii) MGA significantly reduces the probing attempts from 625 to as small as 19.68 times, which is merely 3.17%. To summarize, our proposed MGA achieves close-to-optimal visual quality with negligible probing attempts.

Adaptive MGA: MGAA. An insight from MGA is that the linear search phase may not lead to quality improvement if: (i) the gradient at $\bar{\mathbf{e}}_n$ is small or (ii) the residual bitrate budget beyond $B_T - R_n(\bar{\mathbf{e}}_n)$ is small. Driven by this insight, we modify MGA so that it performs linear search only if:

$$\frac{Q_n(\lfloor\lambda\bar{\mathbf{e}}_n\rfloor) - Q_n(\bar{\mathbf{e}}_n)}{R_n(\lfloor\lambda\bar{\mathbf{e}}_n\rfloor) - R_n(\bar{\mathbf{e}}_n)} \times \left(B_T - \sum_{n=1}^{N} R_n(\bar{\mathbf{e}}_n)\right) \geq \theta, \tag{2}$$

where θ is a threshold for potential quality improvement. We set $\theta = 1$ dB in the following experiments. While in theory, we could have checked the gradients (the first term in Eq. (2)) in each of the four categories, our empirical analysis indicates that only doing this on the geometry leads to time savings. This analysis is consistent with the intuition that the geometry category impacts rendering quality most. Hence, our implementation only checks geometry attributes. We denote this adaptive MGA as MGAA.

Table 2 also compares the quality and overhead of MGAA against MGA. It shows that MGAA: (i) reduces the running time by 72.32%–86.46%, (ii) cuts the probing attempts by 33.43%–42.39%, and (iii) yet achieves comparable visual quality, e.g., 99.23%–99.93% of PSNR across the five considered dynamic 3DGS scenes. Because MGAA successfully mitigates the overhead of MGA without sacrificing the visual quality, we recommend using MGA when computational resources are abundant and MGAA otherwise.

6 Conclusion

We proposed two algorithms, MGA and MGAA, to allocate bitrate across multiple 3DGS scenes for streaming over dynamic networks, which has never been done before. We proposed two algorithms, MGA and MGAA, for environments with sufficient and limited computational resources. Our extensive experiments with our augmented Draco coder, five dynamic 3D scenes, and four 5G network traces showed the effectiveness, efficiency, and practicality of our MGA and MGAA algorithms. This work can be extended in several directions. First, MGA can be applied to other encoding algorithms besides Draco to demonstrate its generality and efficacy. Second, the bitrate allocation problem may adopt different objective functions, e.g., to reduce the visual quality variance among adjacent 3DGS frames for a better user experience. Besides bit allocation, we are also addressing additional challenges of an end-to-end 3DGS streaming system, such as packet loss and error concealment. Third, we are integrating MGA and MGAA into an end-to-end 3DGS streaming system, where additional challenges, such as network latency, packet loss, and error concealment, must be addressed.

We note that this work does not raise any ethical issues as: (i) all the network traces are used at the aggregated level and (ii) no human subjects were involved.

References

[1] Yihang Chen, Qianyi Wu, Jianfei Cai, Mehrtash Harandi, and Weiyao Lin. 2024. HAC: Hash-grid Assisted Context for 3D Gaussian Splatting Compression. *arXiv preprint arXiv:2403.14530* (2024).

[2] Zhiwen Fan, Kevin Wang, Kairun Wen, Zehao Zhu, Dejia Xu, and Zhangyang Wang. 2024. LightGaussian: Unbounded 3D Gaussian Compression with 15x Reduction and 200+ FPS. *arXiv preprint arXiv:2311.17245* (2024).

[3] Guangchi Fang and Bing Wang. 2024. Mini-Splatting: Representing Scenes with a Constrained Number of Gaussians. *arXiv preprint arXiv:2403.14166* (2024).

[4] Sharath Girish, Kamal Gupta, and Abhinav Shrivastava. 2024. EAGLES: Efficient Accelerated 3D Gaussians with Lightweight EncodingS. *arXiv preprint arXiv:2312.04564* (2024).

[5] Google. 2017. *Draco 3D Data Compression.* https://github.com/google/draco.

[6] Alain Horé and Djemel Ziou. 2010. Image Quality Metrics: PSNR vs. SSIM. In *2010 20th International Conference on Pattern Recognition (ICPR).* Istanbul, Turkey, 2366–2369.

[7] Kai Katsumata, Duc Minh Vo, and Hideki Nakayama. 2023. An Efficient 3D Gaussian Representation for Monocular/Multi-view Dynamic Scenes. *arXiv preprint arXiv:2311.12897* (2023).

[8] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 2023. 3D Gaussian Splatting for Real-time Radiance Field Rendering. *ACM Transactions on Graphics* 42, 4 (2023), 1–14.

[9] Wenkai Liu, Tao Guan, Bin Zhu, Lili Ju, Zikai Song, Dan Li, Yuesong Wang, and Wei Yang. 2024. EfficientGS: Streamlining Gaussian Splatting for Large-Scale High-Resolution Scene Representation. *arXiv preprint arXiv:2404.12777* (2024).

[10] Xiangrui Liu, Xinju Wu, Pingping Zhang, Shiqi Wang, Zhu Li, and Sam Kwong. 2024. CompGS: Efficient 3D Scene Representation via Compressed Gaussian Splatting. *arXiv preprint arXiv:2404.09458* (2024).

[11] Tao Lu, Mulin Yu, Linning Xu, Yuanbo Xiangli, Limin Wang, Dahua Lin, and Bo Dai. 2024. Scaffold-GS: Structured 3D Gaussians for View-Adaptive Rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR).* Seattle, WA, USA, 20654–20664.

[12] Jonathon Luiten, Georgios Kopanas, Bastian Leibe, and Deva Ramanan. 2023. Dynamic 3D Gaussians: Tracking by Persistent Dynamic View Synthesis. *arXiv preprint arXiv:2310.08528* (2023).

[13] Haimin Luo, Teng Xu, Yuheng Jiang, Chenglin Zhou, Qiwei Qiu, Yingliang Zhang, Wei Yang, Lan Xu, and Jingyi Yu. 2022. Artemis: Articulated Neural Pets with Appearance and Motion Synthesis. *ACM Transactions on Graphics* 41, 4 (July 2022), 1–19.

[14] MPEG. 2021. *G-PCC codec description v12.* Technical Report. ISO/IEC JTC 1/SC 29/WG 7 MPEG Output Document.

[15] Arvind Narayanan, Eman Ramadan, Rishabh Mehta, Xinyue Hu, Qingxu Liu, Rostand A. K. Fezeu, Udhaya Kumar Dayalan, Saurabh Verma, Peiqi Ji, Tao Li, Feng Qian, and Zhi-Li Zhang. 2020. Lumos5G: Mapping and Predicting Commercial mmWave 5G Throughput. In *Proceedings of the ACM Internet Measurement Conference (IMC).* Virtual Event, USA, 176–193.

[16] Netflix. 2021. *VMAF - Video Multi-Method Assessment Fusion.* https://github.com/Netflix/vmaf.

[17] Simon Niedermayr, Josef Stumpfegger, and Rüdiger Westermann. 2024. Compressed 3D Gaussian Splatting for Accelerated Novel View Synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR).* Seattle, WA, USA, 10349–10358.

[18] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (Nov. 2011), 2825–2830.

[19] Aras Pranckevicius. 2023. *Making Gaussian Splats Smaller.* https://aras-p.info/blog/2023/09/13/Making-Gaussian-Splats-smaller/.

[20] Kerui Ren, Lihan Jiang, Tao Lu, Mulin Yu, Linning Xu, Zhangkai Ni, and Bo Dai. 2024. Octree-GS: Towards Consistent Real-time Rendering with LOD-Structured 3D Gaussians. *arXiv preprint arXiv:2403.17898* (2024).

[21] 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 (CVPR).* Seattle, WA, USA, 20675–20685.

[22] Guanjun Wu, Taoran Yi, Jiemin Fang, Lingxi Xie, Xiaopeng Zhang, Wei Wei, Wenyu Liu, Qi Tian, and Xinggang Wang. 2023. 4D Gaussian Splatting for Real-time Dynamic Scene Rendering. *arXiv preprint arXiv:2310.08528* (2023).

[23] Zerong Zheng, Han Huang, Tao Yu, Hongwen Zhang, Yandong Guo, and Yebin Liu. 2022. Structured Local Radiance Fields for Human Avatar Modeling. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR).* New Orleans, LA, USA, 15893–15903.