LTS: A DASH Streaming System for Dynamic Multi-Layer 3D Gaussian Splatting Scenes

Yuan-Chun Sun[1], Yuang Shi[2], Cheng-Tse Lee[1], Mufeng Zhu[3], Wei Tsang Ooi[2], Yao Liu[3], Chun-Ying Huang[4], and Cheng-Hsin Hsu[1]
[1]National Tsing Hua University, Taiwan
[2]National University of Singapore, Singapore
[3]Rutgers University, NJ, USA
[4]National Yang Ming Chiao Tung University, Taiwan

Abstract

We present a novel DASH-based streaming system for dynamic 3D Gaussian Splatting (3DGS) scenes, addressing the challenges of streaming large amounts of 3DGS data over diverse and dynamic networks. Our **Layer, Tile, and Segment Adaptive streaming (LTS)** system combines three key features: (i) multi-layer streaming, which adapts to diverse client capabilities while balancing visual quality and bandwidth usage, (ii) tiled streaming, which reduces unnecessary data transmission by focusing on the user's viewport, and (iii) segment streaming, which divides dynamic 3DGS scenes into segments, letting clients request them dynamically to handle network fluctuations. Our experimental results demonstrate that our LTS system achieves superior performance in both live and on-demand streaming of dynamic 3DGS scenes compared to the baselines. For example, in live streaming, LTS could achieve up to 99.70% reduction in missing frames on average and deliver a maximum PSNR (Peak Signal-to-Noise Ratio) improvement of 10.08 dB. In on-demand streaming, LTS could reduce the freeze time by up to 92.01%, and increase the synthesized view quality by up to 5.14 dB in PSNR and 0.11 in SSIM (Structural Similarity Index). Our source codes are available at: https://github.com/AIINS-NTHU/LTS-DASH-Streaming-System-for-3DGS.

CCS Concepts

• **Information systems → Multimedia streaming**; • **Computing methodologies → Mixed / augmented reality**; **Image compression**; • **Networks → Data path algorithms**.

Keywords

Volumetric video, streaming, multi-layer representation, adaptive bitrate algorithm

ACM Reference Format:
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 *ACM Multimedia Systems Conference 2025 (MMSys '25), March 31-April 4, 2025, Stellenbosch, South Africa.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3712676.3714445

1 Introduction

3D Gaussian Splatting (3DGS) [19] has become one of the most popular 3D content formats in recent years, supporting novel view synthesis from multiple input views of a 3D scene. Such a capability enables 6 Degrees-of-Freedom (6-DoF) interactions, a Holy Grail of many immersive applications built around Extended Reality (XR) technologies, which seamlessly fuse the physical and virtual worlds.

Using 3DGS in immersive XR applications offers multiple advantages. First, it synthesizes visually appealing novel views, compared to 3D point clouds. Second, its fast rendering speed makes it well-suited for interactive applications consumed by Head-Mounted Displays (HMDs), compared to NeRF (Neural Radiance Fields) [25]. Third, its simple generation pipeline reduces the construction cost, compared to 3D meshes. Many of these immersive XR applications are distributed, e.g., film production [8], digital heritage [17, 35], robotic navigation [64], online education [2], tele-medicine [26], and remote collaboration [9], requiring 3DGS scenes to be exchanged among multiple parties.

Streaming dynamic 3DGS scenes, composed by multiple frames along the time domain, over the best-effort Internet for immersive XR applications is no easy task for two reasons: (i) the size of 3DGS frames that need to be generated, stored, and transmitted are large, and (ii) HMD users are vulnerable to cyber-sickness [43] that could be caused by late or incomplete 3DGS frames. More specifically, 3DGS frames are much larger than 3D point cloud counterparts because 3DGS frames consist of more attributes for the higher visual quality of synthesized novel views. For example, a single frame in Kerbl et al. [19] occupies about 270–734 MB, which is equivalent to 22.65–58.72 Gbps for streaming their dynamic versions at 10 frames per second (fps). Even with increasingly more studies on compacting and compressing 3DGS scenes [3, 5, 10, 20, 23, 28, 29, 34, 39, 40], the required bandwidth is still beyond the capacity of the network in many parts of the world.

This paper tackles the problem of streaming dynamic 3DGS scenes to heterogeneous clients over diverse and dynamic networks. To the best of our knowledge, dynamic 3DGS scenes have never been streamed with actual network protocols and standards. Instead, most existing dynamic 3DGS works propose representations of dynamic 3DGS scenes to support different scenarios, e.g.,

The work was partially supported by the NSTC of Taiwan (#112-2221-E-007-090-MY3).

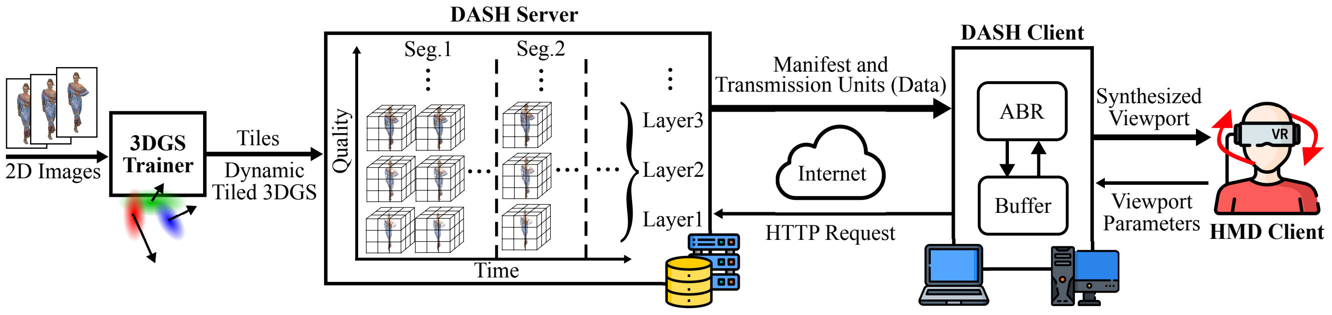


Figure 1: Overview of our LTS system for streaming dynamic 3DGS scenes.

on-the-fly 3DGS scene reconstruction [48], longer-duration 3DGS scenes [21], and resource-constrained mobile devices [54]. Only a very recent bitrate allocation work [49] mentioned RTP (Real-time Transport Protocol) and DASH (Dynamic Adaptive Streaming over HTTP) standards in their architectural design. However, they did not present a low-level design or implement an end-to-end system. In contrast, we develop, implement, and evaluate the very first dynamic 3DGS streaming system in this paper. In our work, the DASH standard is employed for (i) *live 3DGS scenes*, where late 3DGS frames are skipped as rebuffering events are prohibited, and (ii) *on-demand 3DGS scenes*, where clients pause for the packets of late 3DGS frames to arrive.

Our system relies on three principles. First, it is progressive. Different HMDs offer heterogeneous hardware resources, making one-size-fits-all 3DGS representations less suitable. Second, it is viewport-aware. It does not stream the entire 3DGS scene because each HMD client only needs a subset of Gaussian splats in its viewport at any moment. Third, it adapts to network conditions. Network bandwidth varies over time, leading to different instantaneous bitrates of dynamic 3DGS scenes. Following these principles, we propose to: (i) train each dynamic 3DGS scene into multiple *layers*, where higher layers depend on lower layers while providing incrementally better visual quality of synthesized novel views, (ii) spatially partition each dynamic 3DGS scene into multiple *tiles*, which are cuboids containing subsets of Gaussian splats, (iii) temporally divide each dynamic 3DGS scene into multiple *segments*, which are a set of consecutive 3DGS frames.

Multi-layers, tiles, and segments allow us to realize: (i) layered streaming to support heterogeneous HMD clients, (ii) viewport-dependent selection to reduce the number of requested Gaussian splats, and (iii) application-layer buffering to absorb bandwidth fluctuation. While tiling 3D information and segmenting videos can be applied easily to dynamic 3DGS frames, the current dynamic 3DGS approaches do not support layered and progressive representation. In this paper, we adapted a recent layered representation of a static 3DGS scene, LapisGS [39], to dynamic scenes and evaluated its effectiveness in streaming scenarios. Each triplet of <*layer*, *tile*, *segment*> specifies a basic *Transmission Unit (TU)* in our adaptive streaming system, named **Layer, Tile, and Segment adaptive streaming (LTS)**.

Fig. 1 presents an overview of our proposed LTS system. A 3DGS trainer produces a sequence of 3DGS frames consisting of a larger number of Gaussian splats from a set of input videos. These 3DGS frames are aggregated into multiple segments along the time domain. Each segment is represented in several layers for different quality levels. Moreover, each 3DGS scene is cut into tiles in the 3D space for view-dependent streaming. The processed content forms TU (data) files, which are stored on the DASH server. The DASH server also generates manifest files carrying metadata for the DASH client. DASH clients can be hosted on (more capable) HMDs or (nearby) computers. DASH client employs a buffer to absorb the bandwidth fluctuations for smoother playback, which is especially critical for on-demand 3DGS scenes. The crux of the DASH client is an *Adaptive BitRate (ABR)* algorithm, which selects TUs to request based on network conditions, viewport parameters, buffer dynamics, and other factors. Once the playback time arrives, the synthesized novel view for the HMD viewport is displayed.

This paper makes the following contributions:

- We built LTS based on the DASH standard for streaming live and on-demand dynamic 3DGS scenes over diverse and dynamic networks, which is the first of its kind.
- We proposed D-LapisGS, the first multi-layer dynamic 3DGS representation for LTS, to better cope with heterogeneity and dynamics among different networks, HMDs, and users.
- We formulated and solved the bitrate adaptation problem, and we implemented it as the ABR algorithm of LTS to optimally select the transmission units to request.

Our LTS system demonstrates outstanding performance in dynamic 3DGS streaming, excelling in both live and on-demand scenarios. In live streaming, LTS reduces missing frames by 99.7% and achieves a PSNR improvement of up to 10.08 dB, compared to the baseline without tiles. For on-demand streaming, LTS minimizes freeze time by as much as 92.01%, compared to the baseline without tiling and layering, yet maintaining high visual quality of 27.61 dB in PSNR and 0.903 in SSIM.

2 Related Work and Background

2.1 DASH Streaming of Dynamic 3D Scenes

Our LTS system has three unique features: layers, tiles, and segments, which were studied for content types other than 3DGS.

Layered Streaming. Scaling 3D representation into layered structures organizes videos into multiple layers, facilitating adaptation to varying network conditions and devices [1]. A base layer provides essential quality, while enhancement layers improve resolution, frame rate, or other attributes. This approach is also applied to 3D point clouds for efficient compression. For instance, Guarda et al. [14] proposed a deep learning-based scaling method using interlaced sub-sampling for point cloud geometry coding. Similarly, Wang et al. [52] introduced a multiscale end-to-end learning framework for compressing 3D point clouds. Xie et al. [56] employed a scalable point cloud geometry compression framework that addresses the challenges of transmitting large-scale point clouds.

As for 3DGS, we are the first proposing a progressive layered approach [39], named LapisGS, for static 3DGS scenes, which encodes multiple levels of detail into a single-layered model, supporting adaptive streaming and seamless rendering. Our method demonstrates superior efficiency and flexibility, achieving substantial improvements in visual quality and model compactness. However, this method is tailored for static scenes and proves suboptimal for dynamic 3DGS streaming due to the inherent temporal relationships between successive frames caused by object movement.

Tiled Streaming. Tile-based streaming is a multimedia transmission method primarily used for interactive and immersive media streaming when a viewer can only view a subset of the content at a time. In zoomable videos [33] and 360° videos [13, 32, 36, 55, 62], for instance, the entire video frame is divided into smaller regions or *tiles*. The system dynamically selects the tiles to transmit based on the user's perspective or behavior, avoiding unnecessary data transmission. Tile-based techniques have proven highly effective for 3D content streaming, enabling efficient data management and improved streaming performance by optimizing bandwidth usage and enhancing visual quality [22]. For instance, Wang et al. [53] proposed a tile-based adaptive streaming method for point clouds to address the challenges of large data sizes and bandwidth constraints. Their approach allocates bitrates for 3D tiles based on several QoE (Quality of Experience) factors Meanwhile, Park et al. [30] proposed a system for streaming volumetric media, utilizing 3D tiles to optimize bandwidth by culling or adjusting tile details based on the user's view frustum and distance. Similarly, Yin et al. [58] utilized feature grids and residual filtering for adaptive volumetric video streaming. In the context of a large 3D scene, Forgione et al. [12] divided polygon soups representing a virtual environment into axis-aligned cells for streaming Their research indicates that the 3D tile-based method delivers high visual quality. However, 3DGS scenes are more complex than 360° videos and point clouds, which make the previous studies not directly applicable.

Segment Streaming. DASH [44] is a widely used video streaming protocol that allows the system to adaptively adjust video quality based on the user's available bandwidth. By dividing videos into small segments encoded at different bitrates, DASH ensures seamless playback by selecting the most appropriate segment quality in real-time, minimizing buffering and interruptions. DASH has been proven to significantly improve streaming quality, particularly in scenarios with fluctuating network conditions [37]. In recent years, DASH has been employed in many advanced applications, such as multi-view [45, 46, 51, 59] and 360° video streaming [16, 31, 57], where adaptive quality control is crucial for maintaining a smooth and immersive user experience. Moreover, research has extended DASH's capabilities to include 3D video streaming [18, 38, 60], demonstrating its flexibility and effectiveness in handling diverse content types, other than 3DGS.

2.2 Dynamic 3DGS Scenes

Dynamic 3DGS. 3DGS [19] renders photorealistic scenes in real-time by modeling a 3D scene as a collection of Gaussian primitives. Each Gaussian is defined by attributes such as its position x and a covariance matrix Σ, where covariance matrix $\Sigma = \boldsymbol{RSS^T R^T}$ is derived from scaling matrix $\boldsymbol{S}$ and rotation matrix $\boldsymbol{R}$.

Building on this foundation, Luiten et al. [24] proposed Dynamic 3D Gaussians, which integrates dynamic scene reconstruction and 6-DoF tracking. This method is based on an analysis-by-synthesis framework, representing scenes as a collection of 3D Gaussians with variable positions and orientations while maintaining constant attributes such as color, size, and opacity. By incorporating physical constraints, including short-term local rigidity, local rotational similarity, and long-term local isometry losses, the approach ensures the motion of Gaussian elements remains physically consistent over time. This dynamic 3DGS framework maintains temporal coherence and enables compact dynamic representation without significant quality degradation, but it still remains insufficiently optimized for streaming applications.

Dynamic 3DGS Streaming. Research on dynamic 3DGS has grown rapidly, focusing on areas such as generation, rendering, compression, augmentation, regularization, AI-generated content, simulations, dynamic scene reconstruction, and storage [6]. Streaming dynamic 3DGS content has similarly emerged as a prominent area of research, as it offers significant potential for real-time 3D applications. One notable framework [48] leveraged existing 2D video streaming architectures by transmitting 2D images from the server and reconstructing 3DGS frames on client devices. Similarly, Liu et al. [21] addressed dynamic 3DGS streaming through a sliding window approach tailored for volumetric video streaming. Their method is particularly advantageous for frequently changing dynamic scenes, enabling smoother playback. Wang et al. [54] proposed a framework that converts dynamic 3D Gaussians into 2D videos, enabling efficient transmission and rendering through hardware video codecs. The method is designed specifically for mobile devices to support high-quality volumetric video playback. However, these studies did not explicitly address how to transmit dynamic 3DGS content over the network. In contrast, our earlier work [49] proposed a Rate-Distortion (R-D)-based streaming architecture designed to optimize 3DGS streaming by estimating the R-D characteristics of 3DGS frames and encoding them with adaptive parameters. This approach enhanced streaming efficiency and demonstrates strong performance in handling complex scenes. Nevertheless, we did not consider how to dynamically adjust streaming to mitigate delays caused by encoding large 3DGS frames.

3 The Proposed LTS System

3.1 Layered Streaming

LapisGS [39] successfully balanced visual fidelity with the compactness of the model using layered structure, and Dynamic 3DGS [24] ensures temporal coherence by incorporating physical constraints. The current problem at our hand is, however, more challenging as we have to create a *multi-layered* and *frame-dependent* representation for DASH-based dynamic 3DGS streaming. To overcome these limitations, we introduce Dynamic LapisGS (D-LapisGS in short), an innovative representation specifically tailored for efficient

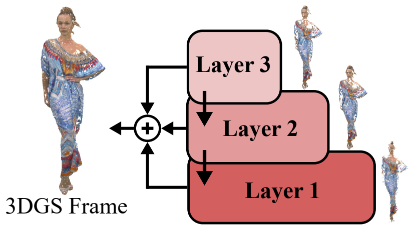


Figure 2: Layer structure of a 3DGS frame in LTS.

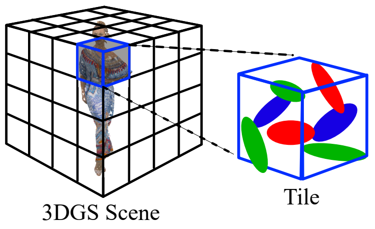


Figure 3: Spatial structure of a 3DGS frame in LTS.

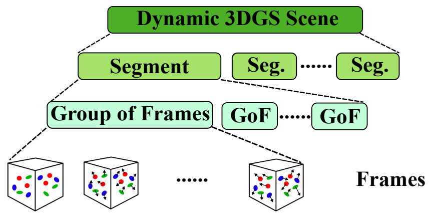


Figure 4: Temporal structure of a dynamic 3DGS scene in LTS.

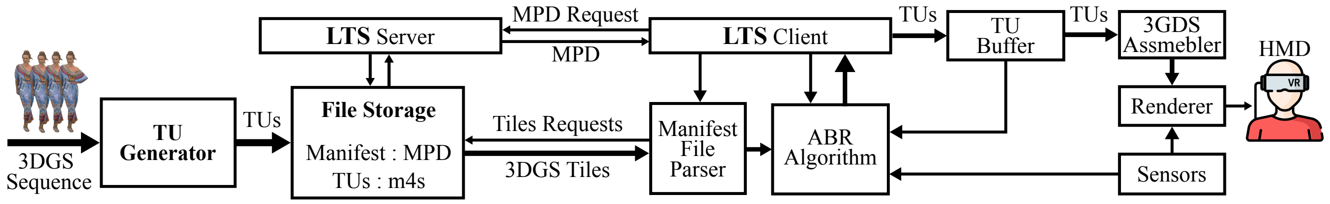


Figure 5: System architecture of our proposed LTS system.

dynamic 3DGS streaming. This approach synthesizes the fundamental concepts of Luiten et al. [24] and the LapisGS [39] to create a layered, frame-dependent architecture that better meets dynamic scene requirements.

D-LapisGS builds upon the progressive multiscale training methodology of its predecessor. The system initially trains a low-resolution base layer, followed by successive enhancement layers utilizing higher-resolution datasets, as shown in Fig. 2. These supplementary layers enhance the detail captured by previous layers while maintaining fixed parameters. The framework dynamically optimizes layer opacity to balance their relative contributions during subsequent training phases. This hierarchical structure enables the model to construct a coarse layout of the scene early on, while later layers progressively incorporate higher-frequency details, thereby enhancing convergence efficiency and minimizing redundancy across quality levels. To maintain temporal coherence, D-LapisGS treats Gaussians as particles governed by physical constraints, including local rigidity, rotational similarity, and long-term local isometry. These physical priors ensure coherent particle neighborhood movement and maintain spatial relationships over time, functioning as optimization regularizers.

3.2 Tiled Streaming

For 6-DoF immersive XR streaming, each HMD user moves freely in a 3D environment, where the user's viewport does not always cover the whole 3DGS scene. Hence, we incorporate tiled streaming that cuts the scene into multiple cuboids. As illustrated in Fig. 3, our scene is segmented into cuboids of uniform size based on the location of 3D Gaussian splats in the 3D coordinate system. More comprehensive and adatpvie tile segmentation approaches are also possible. By taking the HMD viewport parameters, such as position and orientation, as input, our system determines and only transmits the *visible* tiles to cut down the network and computing overhead. This approach reduces the latency and enhances the overall visual quality by focusing resources on the visible regions of the 3D environment.

3.3 Segmented Streaming

To adapt to diverse and dynamic network conditions over time, our system employs the segmentation approach in the DASH standard, as illustrated in Fig. 4. In particular, each dynamic 3DGS scene consists of a sequence of 3DGS frames, where G consecutive frames form a Group of Frames (GoF), encoded into one intra-coded and $G - 1$ inter-coded frames. C consecutive GoFs are then aggregated into a segment, which is streamed together. Last, M segments constitute the dynamic 3DGS scene, which contains $G \times C \times M$ 3DGS frames in total. With segments, HMD clients in our system could dynamically determine which segments to request at what time based on the changing network bandwidth and other factors. By buffering additional segments during favorable network conditions, our system can mitigate the adverse effects of poor network conditions encountered shortly thereafter, particularly when streaming on-demand 3DGS scenes.

3.4 System Architecture

Fig. 5 illustrates the overall architecture of our LTS system, which is an extension of the DASH standard designed for 2D and 360-degree videos. We first introduce the LTS server, which starts with a TU generator that turns a dynamic 3DGS scene into a set of TUs. TUs are the minimum sets of bytes from a given dynamic 3D scene that can be requested and transmitted over the network. In particular, the TU generator creates multiple layers, which are encoded by an enhanced 3D point cloud codec [49]. The encoded frames are cut into tiles spatially and then form segments temporally. The resulting TUs are saved in the storage as `m4s` files, which is a widely used container file format in DASH streaming. We note that each TU can be uniquely identified by a triplet of its layer, tile, and segment indexes. The TU generator also keeps track of the size of individual TUs and computes the expected quality of synthesized novel views, which are crucial to the client in deciding which TUs to request. Based on these results, the TU generator produces manifest files, called Media Presentation Description (MPD), consisting of codec information, playback duration, frame rate, bandwidth requirements, TU URLs, etc. These manifest files are also stored in the storage. The server implements an HTTP server that transmits the TU and manifest files to clients upon request.

Next, we present the LTS client, which is responsible for selecting the best TUs to request based on the network conditions, client capabilities, and HMD user behaviors. The client consists of an HTTP client, which retrieves manifest and TU files from the server. A manifest file parser analyzes manifest files to get metadata about the target dynamic 3DGS scene. TU files are first sent to the TU buffer, which could absorb the varying delays due to fluctuating network bandwidth. During playback, corresponding TUs are moved from the TU buffer to the 3DGS assembler, which then merges and decodes these TUs into 3DGS frames that can be rendered for HMD display based on the current HMD position and orientation (i.e., 6-DoF pose of the viewport) determined by the inertial and RF (Radio Frequency) sensors. The brain of the LTS client is its ABR algorithm, which instructs an HTTP client to request specific TUs based on: (i) the metadata from the manifest file parser, (ii) the estimated network bandwidth from the HTTP client, (iii) the buffer level from TU buffer, and (iv) the viewport position and orientation from HMD sensors. Given that the ABR algorithm is the crux of our LTS system, we detail its design in the next section.

4 Bitrate Adaptation Algorithm

4.1 Problem Statement

Our ABR algorithm consists of two steps: (i) TU selection and (ii) TU sorting. TU selection essentially solves an optimization problem to maximize the expected quality of synthesized novel views within a sliding allocation window of W segments subject to an estimated bit budget B. B is computed based on the throughput measurements of prior segments. To filter out high-frequency noise, we employ Exponentially Weighted Moving Average (EWMA), which assigns an empirically chosen weight of 20% to the most recent measurement. The optimization algorithm is solved once every allocation step, denoted as S segments, where $S \leq W$. We note that our optimization problem only considers visible TUs, as streaming other TUs is simply a waste of resources. Moreover, our optimization problem takes the already buffered TUs as inputs and only requests TUs that haven't been requested and transmitted. By repeatedly solving the optimization problem, our ABR algorithm adapts to the network, system, and user dynamics.

The second step considers the layer dependency, tile importance, and time urgency when sorting the set of selected TUs to be requested. The following sorting criteria are employed: (i) earlier segments, (ii) (within the same segment) lower layers, and (iii) (within the same layer) closer tiles are requested sooner. We request earlier segments sooner because they are closer to their playback time and are more urgent. Furthermore, we request lower layers sooner because, without layers beneath them, higher layers cannot be rendered. Last, inspired by Fang et al. [11], we compute the distance between each tile's center and the current HMD position and request closer tiles sooner. The intuition is that closer tiles tend to occupy larger fractions of the viewport, and humans are more likely to be attracted by them.

4.2 Mathematical Formulation

Next, we describe the formulation of our optimization problem. We let L, T, and W represent the numbers of layers, tiles, and segments, respectively, in the current allocation window. From the manifest files, we get to know the size of each TU, written as $R_{w,l,t}$, where $w \in \{1, 2, \ldots, W\}$, $l \in \{1, 2, \ldots, L\}$, and $t \in \{1, 2, \ldots, T\}$. The manifest files shall also provide the expected quality of synthesized novel views. Our pilot tests indicated that estimating per-tile quality at the server is challenging because many Gaussian splats span across multiple nearby tiles. Hence, we calculated the expected quality levels at the layer-segment granularity, which are computed using ground-truth images from the testing dataset. Specifically, for segment w ($w \in \{1, 2, \ldots, W\}$), the expected quality improvement of receiving layer l on top of all lower layers $l' < l$ is written as $Q_{w,l}$ ($l \in \{1, 2, \ldots, L\}$). Along the same line, we decided to perform a *visibility check* on the dynamic 3DGS scene before running the ABR algorithm. We use a boolean value $V_{w,l,t} \in \{0, 1\}$, where $w \in \{1, 2, \ldots, W\}$, $l \in \{1, 2, \ldots, L\}$, and $t \in \{1, 2, \ldots, T\}$ to capture visibility, where $V_{w,l,t} = 1$ iff this TU is visible in the HMD viewport. In addition, we need to know if a TU has already been buffered. We define $H_{w,l,t}$ in the same way as $V_{w,l,t}$, where $H_{w,l,t} = 0$ iff this TU has already been buffered. Our optimization formulation only considers downloading TUs whose $V_{w,l,t} = 1$.

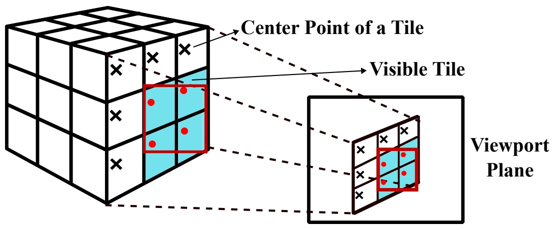


Figure 6: Illustration of visibility check.

The optimization formulation selects the layers and segments to request, as the invisible tiles have been determined and excluded. We use a boolean variable $x_{w,l}$, where $w \in \{1, 2, \ldots, W\}$ and $l \in \{1, 2, \ldots, L\}$ to denote whether this layer-segment should be requested: $x_{w,l} = 1$ iff it is chosen. Before moving on, we define an intermediate boolean variable $Y_{w,l}$, where $w \in \{1, 2, \ldots, W\}$ and $l \in \{1, 2, \ldots, L\}$ to indicate if any tiles in this layer-segment need to be requested. Note that $Y_{w,l}$ is actually a function of our inputs (and independent to our decision variables $x_{w,l}$), which can be computed by:

$$Y_{w,l} = \left\lceil \sum_{t'=1}^{T} V_{w,l,t'} \cdot H_{w,l,t'} \Big/ T \right\rceil, \tag{1}$$

$\forall\ w \in \{1, 2, \ldots, W\}$ and $l \in \{1, 2, \ldots, L\}$. Here $Y_{w,l} = 1$ iff this layer-segment has at least a tile that hasn't been buffered.

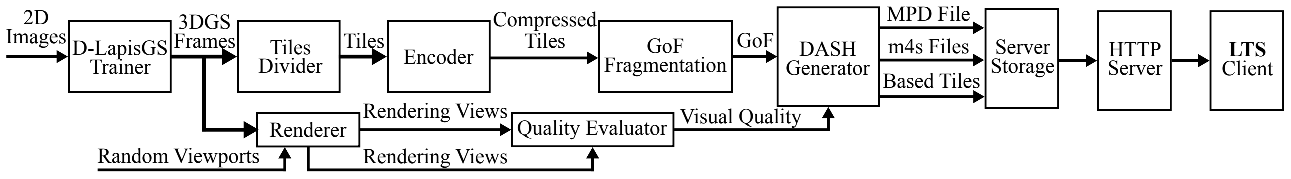


Figure 7: The LTS server implementations.

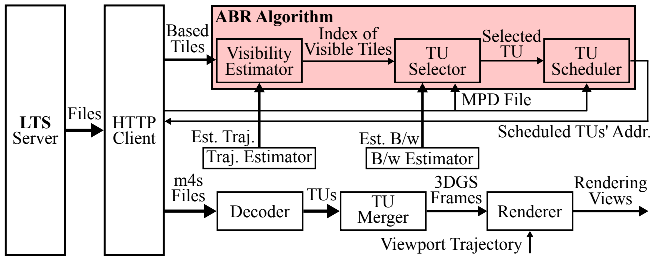


Figure 8: The LTS client implementations.

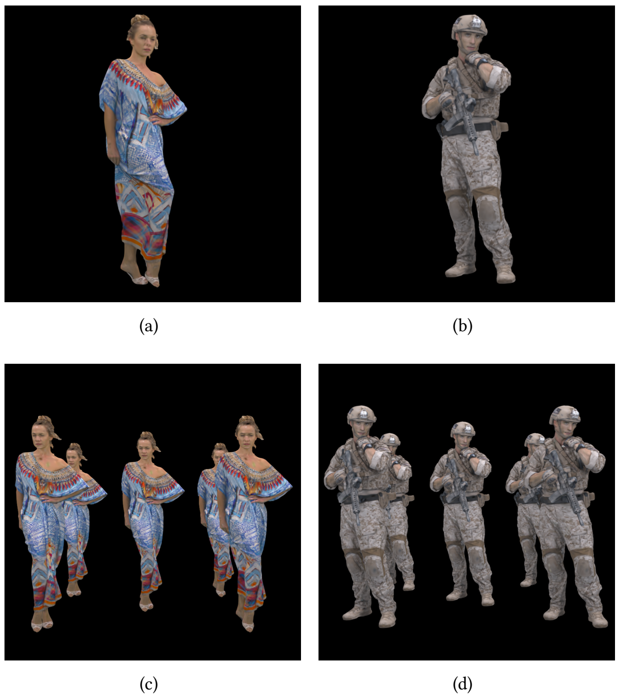


Figure 9: Sample rendered views from: (a) LongDress$_1$, (b) Soldier$_1$, (c) LongDress$_5$, and (d) Soldier$_5$

The optimization problem can be written as:

$$\mathbf{X}^* = \text{argmax}_{\mathbf{X}} \sum_{w=1}^{W} \sum_{l=1}^{L} \left[Q_{w,l}(1 - Y_{w,l}) + x_{w,l} Y_{w,l} \right] \cdot \alpha^{W-1} \quad (2a)$$

$$\text{s.t.} \sum_{w=1}^{W} \sum_{l=1}^{L} x_{w,l} \sum_{t=1}^{T} R_{w,l,t} \cdot H_{w,l,t} \leq B; \quad (2b)$$

$$x_{w,l} \leq Y_{w,l} \ \forall\ w = \{1, 2, \ldots, W\}, l = \{1, 2, \ldots, L\}; \quad (2c)$$

$$x_{w,l} \cdot Y_{w,l} \geq x_{w,l+1} \cdot Y_{w,l+1}$$
$$\forall\ w = \{1, 2, \ldots, W\}, l = \{1, 2, \ldots, L-1\}; \quad (2d)$$

$$x_{w,l} \in \{0, 1\} \ \forall\ w = \{1, 2, \ldots, W\}, l = \{1, 2, \ldots, L\}. \quad (2e)$$

Here, the objective in Eq. (2a) considers the overall quality of the synthesized novel views, including both the buffered and selected layer-segments. We note that a factor $0 \leq \alpha \leq 1$ is used to prioritize the layer-segments with closer playback time; α is a system parameter. In terms of the constraints, Eq. (2b) ensures the total size of the requested tiles does not exceed the bit budget; Eq. (2c) makes sure that we don't request already buffered layer-segments; and Eq. (2d) validates the layer dependency, i.e., we request a layer only if all its lower layers are either buffered or also requested. Lastly, Eq. (2e) defines the boolean decision variables, which are collectively written as $\mathbf{X}$.

4.3 The Proposed ABR Algorithm

We propose an ABR algorithm with two steps: visibility check and full space search.

Visibility check. Before solving the optimization problem, we first check if each tile is visible in the HMD viewport. Here, a virtual camera projects every tile in a 3D coordinate system onto a 2D viewport plane. In addition, two 3D coordinate systems, world and camera, exist. Several viewport/camera parameters are needed to transform a point in the world coordinate system to the camera coordinate system and to project a point in the camera coordinate system to a 2D viewport plane. These parameters include the rotation matrix, translation matrix, focal length, principal point, and resolution.

Fig. 6 gives a high-level idea of our visibility check. We use the tile center $(x, y, z)_W$ in the world coordinate system to represent that tile. Following Szeliski [50], we transform $(x, y, z)_W$ to $(x, y, z)_C$ in the camera coordinate system using the rotation matrix and translation matrix. After that, we project $(x, y, z)_C$ to $(\phi, \psi)_V$ on the viewport plane using the focal length, principal point, and resolution. Comparing $(\phi, \psi)_V$ with the width and height of the HMD viewport allows us to determine if the tile is visible or not. Repeating the same check with all tiles, potentially in parallel, helps us to filter out invisible tiles.

Full space search. The formulation in Eq. (2) is a Binary Integer Programming (BIP) problem, which is non-convex by nature and is difficult to solve. Fortunately, the search space of our problem is rather small, as typically: (i) a small number of layers (L) and (ii) a rather short allocation window (W) are considered. This is because too many layers incur high training overhead, while too long allocation windows increase the estimation error on the network bandwidth. Hence, we propose to solve our formulation optimally using a full space search. Throughout our experiments, we found our algorithm terminates within 40 ms, which is fairly reasonable as our problem is only solved once every few seconds at most.

5 Performance Evaluations

5.1 Implementations

We have implemented the proposed LTS system in a mixture of Python and C. Fig. 7 illustrates the server-side implementations of our LTS system. The server first turns 2D images into multi-layer 3DGS frames using the *D-LapisGS Trainer*. These frames are then divided into tiles through the *Tile Divider*. Each tile is compressed by an *Encoder*. We use an enhanced Draco [49] in its lossless mode for 3DGS compression. The compressed tiles are grouped into GoF via the *GoF Fragmentation* and subsequently passed to a *DASH Generator*. Meanwhile, the frames are also sent to *Renderer* to generate 2D views from 30 random viewpoints, which are evaluated by the *Quality Evaluator* for the estimated visual quality. The *DASH Generator* creates tiles based on the estimated qualities, TU size, and other parameters, such as codec, frame rate, and playback time. The results are stored in MPD and m4s files in the *Server Storage*, which is connected to the *HTTP Server* for data transmission.

Fig. 8 illustrates the client-side implementations of our LTS system. For brevity, we omit all the arrows representing requests. When the streaming starts, the client requests MPD file through the *HTTP Client*. After receiving MPD files, the system runs the ABR algorithm. The *Visibility Estimator* determines which tiles are visible based on the estimated trajectory provided by the *Trajectory Estimator*. If not otherwise specified, we employ user traces in our experiments, which can be considered as a very accurate estimator. Then, *TU Selector* selects the best TUs based on the available bandwidth estimated by the *Bandwidth Estimator* and MPD files. We implement the EWMA filter [4] for a throughput-based bandwidth estimator. The *TU Scheduler* determines the priority levels of individual TUs and outputs them in an ordered list. The HTTP server follows the list to request the m4s files. Once the client receives the files, the *Decoder* decompresses them into TUs and merges the TUs by the *TU Merger* to reconstruct the 3DGS frames. Finally, the *Renderer* renders the views following the client trajectory for visualization.

5.2 Setup

Dynamic 3DGS scenes. We employ two dynamic 3D sequences from the 8i point cloud dataset [7], *LongDress* and *Soldier*, in our experiments. To generate corresponding dynamic 3DGS scenes from them, we render each point cloud frame from multiple viewpoints into images and use them to train a 3DGS frame following the settings in Shi et al. [42]. Specifically, Open3D [63] version 0.15.1 is used for 2D rendering, where the width and height of the rendered images are both 600 pixels. For model training and testing, we generate 100 random views for training and another 200 random views for testing with the camera settings of Mildenhall et al. [25].

To vary the scene complexity level, we duplicate the considered 8i object d times and refer to them as LongDress$_d$ (or LD$_d$) and Soilder$_d$ (or SO$_d$). To get long enough dynamic 3DGS scenes, we repeat 30 3DGS frames 30 times, which leads to four 30-second scenes at 30 fps. In the rest of this paper, we present sample results from LD$_1$, SO$_1$, LD$_5$, and SO$_5$, as shown in Fig. 9. In our dynamic 3DGS scenes, each GoF consists of 30 frames, with 1 I-frame and 29 P-frames, encoded at full, half, and one-eighth resolutions for 3-, 2-, and 1-layer representations. For tiling, LD$_1$ and SO$_1$ have bounding boxes of $2.4\times2.4\times2.4$ m^3, while LD$_5$ and SO$_5$ have bounding boxes of $4.8\times4.8\times2.4$ m^3. The dimensions of the tiles in all dynamic 3DGS scenes are $0.6\times0.6\times0.6$ m^3. Table 1 gives the key statistics of these four scenes. Using these scenes, we generate four representations for comparisons: (i) our proposed D-LapisGS, (ii) LapisGS [39], (iii) D3DGS [24], and (iv) ordinary 3DGS [19]. Among them, LapisGS and D3DGS are the State-Of-The-Art (SOTA), while the resource-demanding 3DGS generates the highest-quality novel views serving as reference views for quality metrics. All training hyperparameters were maintained constant throughout all experimental stages.

HMD trajectories. As reported in previous studies [38, 41, 47, 61], the users tend to spend most of their time looking toward the frontal body of the avatar in volumetric videos, with particular focus on the avatar's face. To better evaluate the generalization ability of our system, we consider four 30-second trajectories, all following a spiral path rotating at 8 degrees per frame and pointing towards the center of the dynamic 3DGS scenes. The first two trajectories always maintain a distance of 2.5 m to the center, while the other two trajectories start with a distance of 5 m and gradually reduce it to 2.5 m. All four trajectories start at an elevation of 2 m, but two of them move up for 2 m in total, while the other two move down. We repeat all our experiments with these four diverse trajectories and report average performance among them.

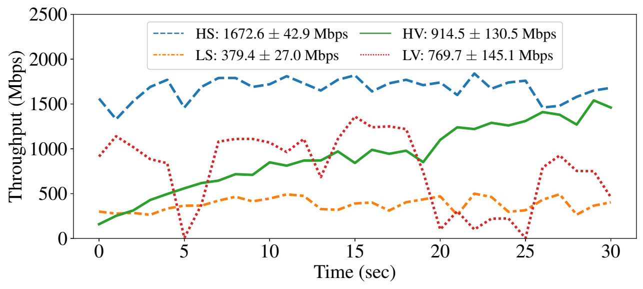


Figure 10: Throughput of four considered network traces from a 5G testbed.

Network traces. To emulate real network conditions, we employ the network traces from the Lumos5G dataset [27]. We first classify all the network traces into four categories:

- *High Stable (HS)*, whose average throughput is in the top 33% and whose throughput variance is in the bottom 10%.
- *Low Stable (LS)*, whose average throughput is in the bottom 33% and whose throughput variance is in the bottom 10%.
- *High Varying (HV)*, whose average throughput is in the top 33% and whose throughput variance is in the top 10%.
- *Low Varying (LV)*, whose average throughput is in the bottom 33% and whose throughput variance is in the top 10%.

We randomly select four network traces from each category and plot their throughput over time in Fig. 10. For evaluations, we emulate the network condition following the network traces.

Table 1: Dynamic 3DGS Scenes Used in Our Experiments (3-/2-/1-Layer)

Scene	LongDress$_1$	Soldier$_1$	LongDress$_5$	Soldier$_5$
Per-Frame No. Gau. (10^3)	255 / 137 / 66	306 / 166 / 78	1274 / 683 / 331	1528 / 830 / 392
Average Frame Size (Mb)	72.01 / 38.63 / 18.69	86.40 / 46.91 / 22.15	260.36 / 193.12 / 93.46	308.17 / 234.56 / 110.74
Bounding Box (m^3)	$2.4 \times 2.4 \times 2.4$	$2.4 \times 2.4 \times 2.4$	$4.8 \times 4.8 \times 2.4$	$4.8 \times 4.8 \times 2.4$

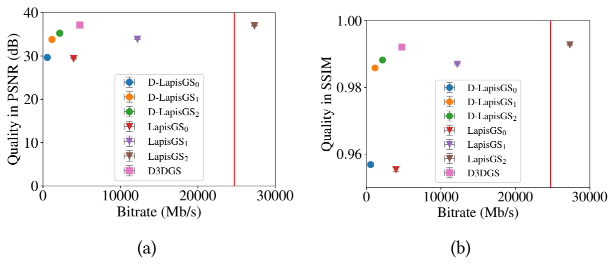


Figure 11: Comparing different representations and layers with the ordinary 3DGS: (a) quality in PSNR and (b) SSIM. The red vertical line indicates the size of the ordinary 3DGS.

Metrics. We evaluate our system using both visual quality and streaming performance metrics. PSNR and SSIM [15] are used to measure the pixel-level fidelity and the perceived visual quality of synthesized views, respectively. For streaming performance, we measure missing frames, which represent the number of frames that fail to meet their designated playback deadlines, and freeze time, which indicates the cumulative duration of rebuffering events required to accommodate delayed tile deliveries. Missing frames are particularly critical for live streaming scenarios, while freeze time is crucial for on-demand streaming applications.

Streaming scenarios. We consider two common scenarios: live and on-demand streaming. In live streaming, the server generates frames in real-time, while user focus more on timely delivery of frames. In contrast, in on-demand streaming, the server prestores all frames, the client can preload frames, and users ask for better visual quality.

ABR baselines. We are not aware of any ABR algorithm designed for DASH streaming of dynamic multi-layer 3DGS scenes. Therefore, we employ three ABR heuristics that request all TUs but with different playback criteria to serve as the baseline for our comparisons. When a baseline ABR algorithm is executed, it requests all TUs in its allocation window. Moreover, at the playback time of the current segment, the baseline algorithm checks its playback criterion to determine if the frames should be skipped (for live streaming) or the playout should freeze (for on-demand streaming). In particular, the considered baselines are: (i) *No Adaptation (NA)*, which requires all TUs of the current segment to be buffered, (ii) *No Tiling (NT)*, which requires all TUs belong to all tiles of at least one layer of the current segment to be buffered, and (iii) *No Layer (NL)*, which requires all TUs belong to the visible tiles of all layers of the current segment to be buffered.

Experiment environment. The LTS server and client are executed on workstations with AMD 32-core CPUs at 2.80 GHz and NVIDIA GTX-3090 GPUs with 24 GB VRAM. The 3DGS trainer is executed on a workstation with an NVIDIA H100 GPU. If not otherwise specified, we let $\alpha = 0.9$, $G = 30$, $C = 1$, $W = 3$, $S = 1$, $M = 30$, $L = 1$, and viewport resolution to be 1024 ×1024.

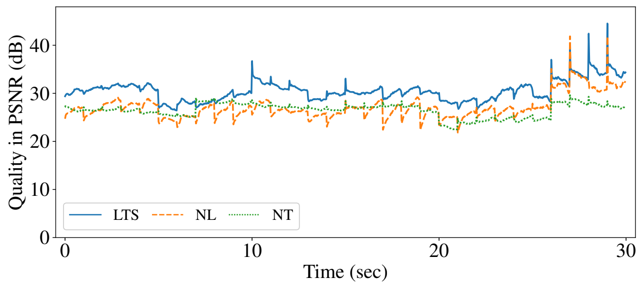


Figure 12: Sample visual quality from LongDress$_1$ under different ABR algorithms.

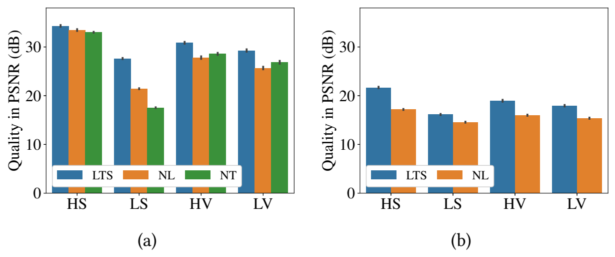


Figure 13: Visual quality with different network traces under different ABR algorithms: (a) simpler LongDress$_1$ and (b) more complex LongDress$_5$.

5.3 Results

Different representations. Fig. 11 shows the visual quality and overall streaming bitrate among different representations from LongDress$_1$. Compared with LapisGS, our proposed D-LapisGS can save 92.1% of bitrate with the highest layer and 70.8% with the lowest layer. Besides, the overall visual quality of our representation achieves 37.0 dB in PSNR and 0.991 in SSIM, which are fairly high. Compared with D3DGS, our proposed D-LapisGS representation consumes 54.60% less bitrate. This can be attributed to our improved training process: we consider not only the previous frames but also the lower layers, allowing us to get more compact 3DGS frames. In addition, our representation can provide different levels of detail

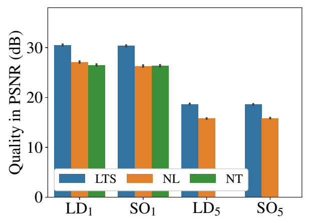


Figure 14: Visual quality from ABR algorithms in different scenes in PSNR.

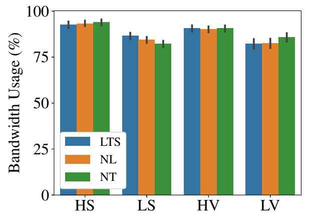


Figure 15: Bandwidth usage from individual ABR algorithms with different network traces.

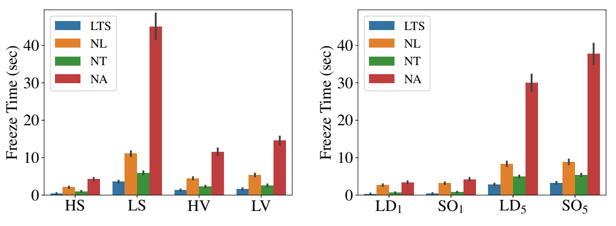
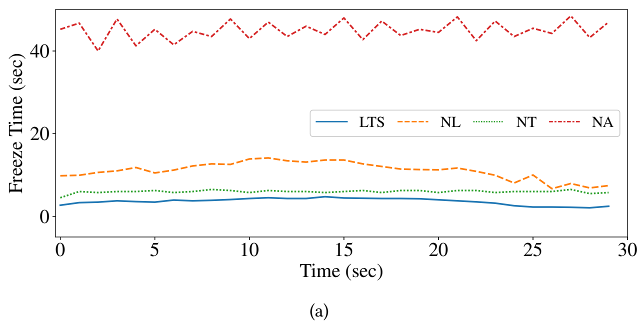


Figure 17: Average freeze time from ABR algorithms under: (a) different network traces and (b) different scenes.



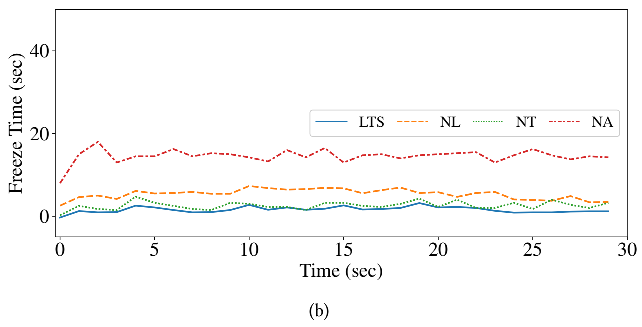


Figure 16: Freeze time under sample network traces: (a) HS and (b) LS.

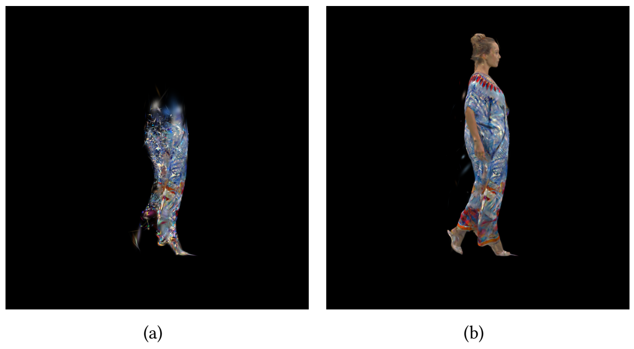


Figure 18: Sample rendered views by using: (a) NL and (b) LTS in on-demand streaming.

for more flexible 3DGS streaming to heterogeneous HMD clients over diverse networks.

Live streaming. According to our results, using NA for live streaming is not practical. Its bitrate can easily exceed the available bandwidth even under the best network conditions, i.e., none of the 3DGS frames can be rendered. Hence, we do not report the results from NA in live streaming. Besides, our results also show that NT does not work in more complex scenes, such as LongDress$_5$ and Soldier$_5$. In particular, even with the lowest layer alone, its bitrate often exceeds the available bandwidth, leading to too many missing frames. Hence, we do not report the missing frame results from NT either. In terms of missing frames from LTS and NL, we find that, on average, they only suffer from 1.88 missing frames from 900-frame traces, which is 99.7% less than NT. Adding to that, all missing frames with LTS happened in the LV trace. We note that in this trace, some extreme network conditions cause the bandwidth to drop to zero, creating an impossible mission for any ABR algorithms.

Fig. 12 shows the overall visual quality from LongDress$_1$ with different ABR algorithms. The results show that LTS achieves higher visual quality in PSNR and SSIM across all 3DGS frames. Besides, in Fig. 13, we show the visual quality of PSNR in a simple scene, LongDress$_1$, and a complex scene, LongDress$_5$, using different ABR algorithms under different network traces. The results reveal that LTS outperforms NL and NT by at most 6.20 dB and 10.08 dB in PSNR in the simple scene. It also outperforms NL by up to 4.52 dB in PSNR in the complex scene. Fig. 14 compares the overall visual quality among different scenes. We observe that LTS achieves 24.54 dB in PSNR and 0.851 in SSIM, beating: (i) NL by 4.06 dB in PSNR and 0.83 in SSIM, and (ii) NT by 3.418 dB in PSNR and 0.04 in SSIM. Fig. 15 presents the overall bandwidth usage in all network conditions. We find that in most network conditions, LTS utilizes the available bandwidth more efficiently, which can be attributed to its capability of intelligently selecting the tiles and layers.

On-demand streaming. Fig. 16 reports the per-frame average freeze time due to buffering. We observe that LTS always provides the shortest freeze time. We also give the average freeze time under different network conditions in Fig. 17(a). LTS has a 1.73-second freeze time on average, and a 3.60-second freeze time in the worst network condition. Besides, LTS reduces the freeze time by at most 70.56%, 57.57%, and 92.01% compared to NL, NT, and NA, respectively. While we could not give the figure on the average quality due

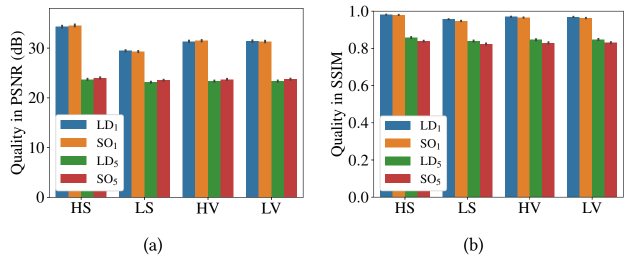


Figure 19: Average visual quality by using LTS in: (a) PSNR and (b) SSIM in on-demand streaming.

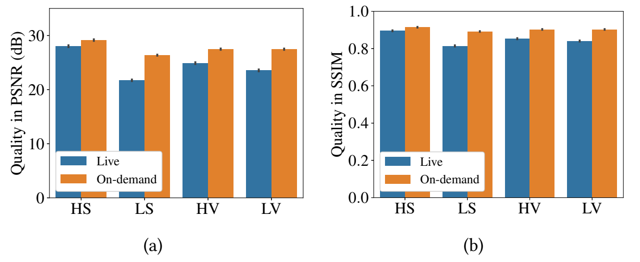


Figure 20: Average visual quality by using LTS for live and on-demand streaming under different network traces in: (a) PSNR and (b) SSIM.

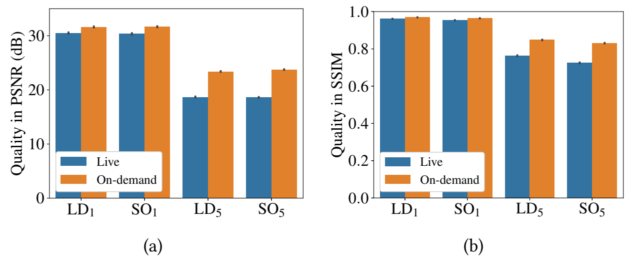


Figure 21: Average visual quality by using LTS for live and on-demand streaming under different 3DGS scenes in: (a) PSNR and (b) SSIM.

to space limitation, we note that LTS delivers an average quality of 27.61 dB in PSNR and 0.903 in SSIM, which are fairly high. Fig. 18 gives sample rendered views from LTS and NL. Fig. 18(a) reveals that with NL, only part of the 3DGS scene is streamed in high quality, which Fig. 18(b) shows that LTS is able to render a complete frame in the acceptable visual quality. The above results show that LTS provides a smooth dynamic 3DGS streaming experience for users with good visual quality, while other ABR algorithms result in long freeze time and inferior visual quality, which may drive the HMD users away from the services.

Next, we compare the average freeze time among different scenes in Fig. 17(b). We notice that, generally, all the ABR algorithms need longer freeze time to render more complex scenes. In addition, with tiling, LTS reduces the freeze time by up to 76.50%, and with layering, decreases the freeze time by up to 85.61%. Combining tiling and layering in our proposed LTS system cuts the freeze time by up to 91.31%.

Fig. 19 reports the average visual quality from LTS under different network conditions. We notice LTS performs better with simple scenes and under higher and more stable bandwidth (HS trace). Meanwhile, we also observe that LTS achieves shorter freeze time in those conditions (figures not shown due to space limitations). Take LongDress$_5$ as an example; streaming it causes 8X freeze time than streaming LongDress$_1$. We also observe insignificant visual quality differences in bandwidth conditions other than HS, although the freeze time do increase along with the decrease of average bandwidth.

Comparison between live and on-demand streaming. Figs. 20 and 21 compare the performance of LTS between live and on-demand streaming scenarios. The figures show that LTS achieves better visual quality in on-demand streaming, especially when the network bandwidth is low, or the scene is complex, with gaps of at most 5.14 dB in PSNR and 0.11 in SSIM. However, using on-demand streaming also leads to longer freeze time: at most, 246 seconds of freeze time is observed for 30-second traces.

6 Conclusion

In this paper, we introduce LTS, a DASH-based streaming system designed for dynamic 3DGS scenes. By integrating layering, tiling, and segmenting features, LTS achieves promising performance in both live and on-demand streaming. While streaming dynamic 3DGS scenes has never been done in the literature, by comparing to the current practice, our results show that LTS reduces missing frames during live streaming by up to 99.7%, while improving the PSNR by up to 4.06 dB and the SSIM by up to 0.83. In on-demand streaming, LTS reduces the freeze time by 92.01% while increasing the PSNR by 5.14 dB and the SSIM by 0.11. Adding to that, LTS effectively lowers bitrate requirements by up to 92.1% for the highest quality layer, demonstrating its superior efficiency in compressing dynamic 3DGS scenes. These evaluation results prove LTS's ability to stream adaptive, high-quality, and resource-efficient dynamic 3DGS scenes, making it a promising solution for immersive XR applications across diverse network conditions, dynamic systems, and heterogeneous HMD users.

This paper cannot cover all aspects related to streaming dynamic 3DGS scenes due to the space limit, and thus can be extended in several directions. For example, we primarily assess visual quality using objective metrics, while evaluating QoE of real users' trajectories through user studies is equally critical. Moreover, additional measurable QoE factors, such as motion-to-photon latency, which is the elapsed time between head movement and corresponding scene changes in the HMD, should be considered and analyzed for QoE modeling (especially, cybersickness) to further optimize LTS. Last, like other immersive streaming system LTS comes with quite a few system parameters related to tiles, layers, and segments. These parameters can be used to tradeoff system performance and resource consumption, and should be systematically and even dynamically adjusted.

References

[1] Edward H Adelson. 1995. Layered representations for vision and video. In *Proc. IEEE Workshop on Representation of Visual Scenes (In Conjunction with ICCV'95).* 3–9.

[2] Ahmed Alnagrat, Rizalafande Che Ismail, Syed Zulkarnain Syed Idrus, and Rawad Mansour Abdulhafith Alfaqi. 2022. A review of extended reality (XR) technologies in the future of human education: Current trend and future opportunity. *Journal of Human Centered Technology* 1, 2 (August 2022), 81–96.

[3] Milena T. Bagdasarian, Paul Knoll, Yi-Hsin Li, Florian Barthel, Anna Hilsmann, Peter Eisert, and Wieland Morgenstern. 2024. 3DGS.Zip: A Survey on 3D Gaussian Splatting Compression Methods. *arXiv preprint arXiv:2407.09510* (June 2024).

[4] Lars Burgstahler and Martin Neubauer. 2002. New modifications of the exponential moving average algorithm for bandwidth estimation. In *Proc. of the 15th ITC Specialist Seminar.* C327.

[5] Yuanhao Cai, Yixun Liang, Jiahao Wang, Angtian Wang, Yulun Zhang, Xiaokang Yang, Zongwei Zhou, and Alan Yuille. 2025. Radiative Gaussian Splatting for Efficient X-Ray Novel View Synthesis. In *European Conference on Computer Vision.* 283–299.

[6] Guikun Chen and Wenguan Wang. 2024. A Survey on 3D Gaussian Splatting. *arXiv preprint arXiv:2401.03890* (January 2024).

[7] Eugene d'Eon, Bob Harrison, Taos Myers, and Philip A. Chou. 2017. 8i Voxelized Full Bodies, version 2 – A Voxelized Point Cloud Dataset. https://plenodb.jpeg.org/pc/8ilabs

[8] Epic Games, Inc. 2016. *RealityCapture in Ghost in the Shell.* https://www.capturingreality.com/RealityCapture-In-Ghost-In-The-Shell.

[9] FE Fadzli, MS Kamson, AW Ismail, and MYF Aladin. 2020. 3D telepresence for remote collaboration in extended reality (XR) application. In *IOP Conference Series: Materials Science and Engineering.* 012005.

[10] Zhiwen Fan, Kevin Wang, Kairun Wen, Zehao Zhu, Dejia Xu, and Zhangyang Wang. 2024. In *Advances in Neural Information Processing Systems.* 140138–140158.

[11] Jia-Wei Fang, Kuan-Yu Lee, Teemu Kämäräinen, Matti Siekkinen, and Cheng-Hsin Hsu. 2023. Will Dynamic Foveation Boost Cloud VR Gaming Experience?. In *Proc. of the 33rd Workshop on Network and Operating System Support for Digital Audio and Video.* 29–35.

[12] Thomas Forgione, Axel Carlier, Géraldine Morin, Wei Tsang Ooi, Vincent Charvillat, and Praveen Kumar Yadav. 2018. DASH for 3D Networked Virtual Environment. In *Proc. of the 26th ACM International Conference on Multimedia.* 1910–1918.

[13] Yu Guan, Chengyuan Zheng, Xinggong Zhang, Zongming Guo, and Junchen Jiang. 2019. Pano: Optimizing 360° Video Streaming with a Better Understanding of Quality Perception. In *Proc. of the ACM Special Interest Group on Data Communication.* 394–407.

[14] André FR Guarda, Nuno MM Rodrigues, and Fernando Pereira. 2020. Deep learning-based point cloud geometry coding with resolution scalability. In *2020 IEEE 22nd International Workshop on Multimedia Signal Processing (MMSP).* 1–6.

[15] Alain Horé and Djemel Ziou. 2010. Image Quality Metrics: PSNR vs. SSIM. In *Proc. of IEEE International Conference on Pattern Recognition.* 2366–2369.

[16] Mohammad Hosseini and Viswanathan Swaminathan. 2016. Adaptive 360 VR video streaming based on MPEG-DASH SRD. In *2016 IEEE International Symposium on Multimedia (ISM).* 407–408.

[17] Chiara Innocente, Luca Ulrich, Sandro Moos, and Enrico Vezzetti. 2023. A framework study on the use of immersive XR technologies in the cultural heritage domain. *Journal of Cultural Heritage* 62 (July-August 2023), 268–283.

[18] Jack Jansen, Shishir Subramanyam, Romain Bouqueau, Gianluca Cernigliaro, Marc Martos Cabré, Fernando Pérez, and Pablo Cesar. 2020. A pipeline for multiparty volumetric video conferencing: transmission of point clouds over low latency DASH. In *Proc. of the 11th ACM Multimedia Systems Conference.* 341–344.

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

[20] Joo Chan Lee, Daniel Rho, Xiangyu Sun, Jong Hwan Ko, and Eunbyung Park. 2024. Compact 3D Gaussian Representation for Radiance Field. In *Proc. of the IEEE/CVF Conference on Computer Vision and Pattern Recognition.* 21719–21728.

[21] Bangya Liu and Suman Banerjee. 2024. SwinGS: Sliding Window Gaussian Splatting for Volumetric Video Streaming with Arbitrary Length. *arXiv preprint arXiv:2409.07759* (September 2024).

[22] Zhi Liu, Qiyue Li, Xianfu Chen, Celimuge Wu, Susumu Ishihara, Jie Li, and Yusheng Ji. 2021. Point cloud video streaming: Challenges and solutions. *IEEE Network* 35, 5 (September 2021), 202–209.

[23] 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 *Proc. of the IEEE/CVF Conference on Computer Vision and Pattern Recognition.* 20654–20664.

[24] Jonathon Luiten, Georgios Kopanas, Bastian Leibe, and Deva Ramanan. 2024. Dynamic 3D Gaussians: Tracking by Persistent Dynamic View Synthesis. In *International Conference on 3D Vision (3DV).* 800–809.

[25] 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 (December 2021), 99–106.

[26] Tadatsugu Morimoto, Takaomi Kobayashi, Hirohito Hirata, Koji Otani, Maki Sugimoto, Masatsugu Tsukamoto, Tomohito Yoshihara, Masaya Ueno, and Masaaki Mawatari. 2022. XR (extended reality: virtual reality, augmented reality, mixed reality) technology in spine medicine: status quo and quo vadis. *Journal of Clinical Medicine* 11, 2 (January 2022), 470.

[27] 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 *Proc. of the ACM Internet Measurement Conference.* 176–193.

[28] KL Navaneet, Kossar Pourahmadi Meibodi, Soroush Abbasi Koohpayegani, and Hamed Pirsiavash. 2023. Compact3D: Compressing gaussian splat radiance field models with vector quantization. *arXiv preprint arXiv:2311.18159* (2023).

[29] Simon Niedermayr, Josef Stumpfegger, and Rüdiger Westermann. 2024. Compressed 3D Gaussian Splatting for Accelerated Novel View Synthesis. In *Proc. of the IEEE/CVF Conference on Computer Vision and Pattern Recognition.* 10349–10358.

[30] Jounsup Park, Philip A Chou, and Jenq-Neng Hwang. 2019. Rate-utility optimized streaming of volumetric media for augmented reality. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 9, 1 (February 2019), 149–162.

[31] Dimitri Podborski, Emmanuel Thomas, Miska M Hannuksela, Sejin Oh, Thomas Stockhammer, and Stefan Pham. 2018. 360-degree video streaming with MPEG-DASH. *SMPTE Motion Imaging Journal* 127, 7 (2018), 20–27.

[32] Feng Qian, Bo Han, Qingyang Xiao, and Vijay Gopalakrishnan. 2018. Flare: Practical Viewport-Adaptive 360-Degree Video Streaming for Mobile Devices. In *Proc. of the 24th Annual International Conference on Mobile Computing and Networking.* 99–114.

[33] Ngo Quang Minh Khiem, Guntur Ravindra, Axel Carlier, and Wei Tsang Ooi. 2010. Supporting zoomable video streams with dynamic region-of-interest cropping. In *Proc. of the First Annual ACM SIGMM Conference on Multimedia Systems.* 259–270.

[34] 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* (Octorber 2024).

[35] Bruno Rodríguez-García, Henar Guillen-Sanz, David Checa, and Andres Bustillo. 2024. A systematic review of virtual 3D reconstructions of Cultural Heritage in immersive Virtual Reality. *Multimedia Tools and Applications* (April 2024), 1–51.

[36] Yago Sanchez de la Fuente, Robert Skupin, and Thomas Schierl. 2015. Compressed Domain Video Processing for Tile Based Panoramic Streaming Using SHVC. In *Proc. of the 3rd International Workshop on Immersive Media Experiences.* 13–18.

[37] Michael Seufert, Sebastian Egger, Martin Slanina, Thomas Zinner, Tobias Hoßfeld, and Phuoc Tran-Gia. 2014. A survey on quality of experience of HTTP adaptive streaming. *IEEE Communications Surveys & Tutorials* 17, 1 (September 2014), 469–492.

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

[39] Yuang Shi, Simone Gasparini, Géraldine Morin, and Wei Tsang Ooi. 2024. LapisGS: Layered Progressive 3D Gaussian Splatting for Adaptive Streaming. *arXiv preprint arXiv:2408.14823* (August 2024).

[40] Yuang Shi, Simone Gasparini, Géraldine Morin, Chenggang Yang, and Wei Tsang Ooi. 2025. Sketch and Patch: Efficient 3D Gaussian Representation for Man-Made Scenes. *arXiv preprint arXiv:2501.13045* (January 2025).

[41] Yuang Shi and Wei Tsang Ooi. 2024. Perceptual Impact of Facial Quality in MPEG V-PCC-encoded Volumetric Videos. In *Proc. of the 16th International Workshop on Immersive Mixed and Virtual Environment Systems.* 71–74.

[42] Yuang Shi, Ruoyu Zhao, Simone Gasparini, Géraldine Morin, and Wei Tsang Ooi. 2024. Volumetric Video Compression Through Neural-based Representation. In *Proc. of the 16th International Workshop on Immersive Mixed and Virtual Environment Systems.* 85–91.

[43] Ashutosh Singla, Steve Göring, Dominik Keller, Rakesh Rao Ramachandra Rao, Stephan Fremerey, and Alexander Raake. 2021. Assessment of the Simulator Sickness Questionnaire for Omnidirectional Videos. In *2021 IEEE Virtual Reality and 3D User Interfaces (VR).* 198–206.

[44] Thomas Stockhammer. 2011. Dynamic adaptive streaming over HTTP –: standards and design principles. In *Proc. of the Second Annual ACM conference on Multimedia Systems.* 133–144.

[45] Tianyu Su, Abbas Javadtalab, Abdulsalam Yassine, and Shervin Shirmohammadi. 2014. A DASH-based 3D multi-view video rate control system. In *2014 8th International Conference on Signal Processing and Communication Systems (ICSPCS).* 1–6.

[46] Tianyu Su, Ashkan Sobhani, Abdulsalam Yassine, Shervin Shirmohammadi, and Abbas Javadtalab. 2016. A DASH-based HEVC multi-view video streaming system. *Journal of Real-Time Image Processing* 12, 2 (May 2016), 329–342.

[47] Shishir Subramanyam, Irene Viola, Alan Hanjalic, and Pablo Cesar. 2020. User centered adaptive streaming of dynamic point clouds with low complexity tiling. In *Proc. of the 28th ACM international conference on multimedia.* 3669–3677.

[48] 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 *Proc. of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 20675–20685.

[49] 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 *Proc. of the 2024 SIGCOMM Workshop on Emerging Multimedia Systems*. 1–7.

[50] Richard Szeliski. 2022. 3D to 2D projections. In *Computer Vision: Algorithms and Applications*. Springer Nature, Chapter 2.1.4, 41–51.

[51] Dion Tanjung, Jong-Deok Kim, Dong-Hyun Kim, Jewon Lee, Soonchoul Kim, and Joon-Young Jung. 2023. Qoe optimization in dash-based multiview video streaming. *IEEE Access* 11 (August 2023), 83603–83614.

[52] Jianqiang Wang, Dandan Ding, Zhu Li, and Zhan Ma. 2021. Multiscale point cloud geometry compression. In *2021 Data Compression Conference (DCC)*. 73–82.

[53] Lisha Wang, Chenglin Li, Wenrui Dai, Junni Zou, and Hongkai Xiong. 2021. QoE-driven and tile-based adaptive streaming for point clouds. In *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 1930–1934.

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

[55] Mengbai Xiao, Chao Zhou, Yao Liu, and Songqing Chen. 2017. OpTile: Toward Optimal Tiling in 360-degree Video Streaming. In *Proc. of the 25th ACM International Conference on Multimedia*. 708–716.

[56] Liang Xie, Wei Gao, Huiming Zheng, and Ge Li. 2024. SPCGC: Scalable Point Cloud Geometry Compression for Machine Vision. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*. 17272–17278.

[57] Shou-Cheng Yen, Ching-Ling Fan, and Cheng-Hsin Hsu. 2019. Streaming 360° videos to head-mounted virtual reality using DASH over QUIC transport protocol. In *Proc. of the 24th ACM Workshop on Packet Video*. 7–12.

[58] 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 *Proc. of the 32nd ACM International Conference on Multimedia*. 11089–11098.

[59] Dooyeol Yun and Kwangsue Chung. 2017. DASH-based multi-view video streaming system. *IEEE Transactions on Circuits and Systems for Video Technology* 28, 8 (April 2017), 1974–1980.

[60] Markos Zampoglou, Kostas Kapetanakis, Andreas Stamoulias, Athanasios G Malamos, and Spyros Panagiotakis. 2018. Adaptive streaming of complex Web 3D scenes based on the MPEG-DASH standard. *Multimedia Tools and Applications* 77, 1 (December 2018), 125–148.

[61] Emin Zerman, Radhika Kulkarni, and Aljosa Smolic. 2021. User behaviour analysis of volumetric video in augmented reality. In *2021 13th International Conference on Quality of Multimedia Experience (QoMEX)*. 129–132.

[62] Ali Zeynali, Mohammad H. Hajiesmaili, and Ramesh K. Sitaraman. 2024. BOLA360: Near-optimal View and Bitrate Adaptation for 360-Degree Video Streaming. In *Proc. of the 15th ACM Multimedia Systems Conference*. 12–22.

[63] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. 2018. Open3D: A Modern Library for 3D Data Processing. *arXiv preprint arXiv:1801.09847* (January 2018).

[64] Xin Zhou, Zhepei Wang, Hongkai Ye, Chao Xu, and Fei Gao. 2020. Ego-planner: An esdf-free gradient-based local planner for quadrotors. *IEEE Robotics and Automation Letters* 6, 2 (December 2020), 478–485.