VV-DASH: A Framework for Volumetric Video DASH Streaming

Hadi Heidarirad
Department of Computer Science
University of Calgary
Calgary, Alberta, Canada
hadi.heidarirad@ucalgary.ca

Mea Wang
Department of Computer Science
University of Calgary
Calgary, Alberta, Canada
meawang@ucalgary.ca

Abstract

With the increasing demand for immersive experiences, volumetric video has emerged as a critical technology, offering users six degrees of freedom (6DoF) to fully explore three-dimensional scenes. However, despite significant advancements, there remains a lack of a comprehensive and flexible adaptive streaming framework capable of delivering volumetric video over dynamic network conditions. To address this gap, we present VV-DASH, an end-to-end framework for adaptive volumetric video streaming over DASH (Dynamic Adaptive Streaming over HTTP). Our framework covers the entire streaming pipeline, from video source to video playback. We propose a codec-agnostic DASH Volumetric Video (DVV) segment format that consolidates compressed video content into DASH-ready segments. This segmentation improves achievable streaming throughput by 13.2%, effectively reduces bandwidth demand, and enhances the achievable streaming bitrate by up to 37.8%. In summary, VV-DASH provides a practical, high-performance framework for scalable and adaptive volumetric video streaming.

CCS Concepts

• **Information systems** → **Multimedia streaming**.

Keywords

Volumetric Video, DASH, Point Cloud, Streaming

ACM Reference Format:
Hadi Heidarirad and Mea Wang. 2025. VV-DASH: A Framework for Volumetric Video DASH Streaming. In *Proceedings of the 16th ACM Multimedia Systems Conference (MMSys '25), March 31–April 4, 2025, Stellenbosch, South Africa.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3712676.3718339

1 Introduction

While video streaming has become the primary form for multimedia entertainment (*e.g.,* YouTube [6] and Netflix [23]), demand for immersive and interactive experiences is growing alongside increasing computing and networking capacity. In volumetric video, dynamic objects and scenes are captured using multiple cameras from different angles to represent them in true 3D, offering users six degrees of freedom (6DoF) within the video scene. Unlike 2D frames in conventional videos, volumetric video frames are formed by 3D Meshes or Point Clouds, significantly increasing size and bitrate. Point clouds are more widely used and studied as they are easier to process and manipulate (*e.g.,* better suited for tiling complex scenes [39]), thus, we resort to point cloud format in this paper.

To accommodate varying network conditions and the wide range of bitrates provided by compression algorithms, Dynamic HTTP Adaptive Streaming (DASH) [33] and HTTP Live Streaming (HLS) [25] are among the widely adopted adaptive streaming protocols, utilized by major streaming service providers such as YouTube and Netflix. They are designed to stream video content segment by segment at appropriate quality levels—characterized by a bitrate ladder—subject to the network dynamics. The demand for volumetric streaming has drawn research attention to making it available over DASH for quality adaptation over general network [12, 18, 26, 40–42]. The research challenges have been primarily concerning the trade-off between fast compression/decompression and efficient bitrate for transmission. These challenges motivate the proposal of VV-DASH, an end-to-end framework for advancing the design and analysis of volumetric streaming systems.

In this paper, we propose VV-DASH[1], a highly-efficient DASH framework for volumetric video streaming. The modular design of VV-DASH spans the entire streaming pipeline, from video source to transmission and rendering. The framework is codec-agnostic, with encoder and decoder wrappers enabling integration of any codec. Along with VV-DASH, we introduce a segmentation method for volumetric video content that enables efficient streaming and decoding. Our evaluation results show that the segmentation method effectively improves the achievable streaming bitrate by 37.8% on average, and the decoding speed by up to 51%.

The rest of this paper is organized as follows. Section 2 provides an overview of the background knowledge on volumetric video streaming and related work. Section 3 introduces the codec-agnostic design of VV-DASH, followed by implementation and experiment setup in Section 4. We then present two case studies to showcase that VV-DASH can support streaming with Draco (Section 5) and V-PCC (Section 6). Finally, Section 7 presents the concluding remarks.

2 Background and Related Work

DASH is known for quality adaptation across various network conditions [32]. End-to-end DASH streaming involves the following steps on the server side: (1) compressing the original video into multiple quality levels based on a predefined bitrate ladder (henceforth, *compression*), (2) packaging the compressed videos into segments and preparing a manifest file (*i.e.,* MPD) containing metadata and segment URLs, and (3) serving these segments from an HTTP server that responds to segment requests via URLs.

[1] https://github.com/Rad6/vv-dash

MMSys '25, March 31–April 4, 2025, Stellenbosch, South Africa

ACM ISBN 979-8-4007-1467-2/25/03
https://doi.org/10.1145/3712676.3718339

Though there are server-side quality adaptation proposals [44], the most common DASH approach is client-based adaptation, allowing the use of ordinary HTTP servers ([15, 16, 19, 20] for conventional DASH and [13, 28] for volumetric DASH). Thus, a DASH streaming involves the following steps on client end: (1) Receiving the MPD file from the server and preparing the streaming accordingly. The bitrate ladder can be fetched from the MPD file. (2) Requesting the next segment at a specific quality level selected by the adaptation algorithm (according to observed/estimated bandwidth and buffer health). Quality adaptation may depend on playback buffer state [15], the estimated bandwidth [19], or a combination of factors (including, but not limited to buffer status, bandwidth estimation, codec property [16, 20]). (3) Receiving the segment and queuing it into the buffer, which will be dequeued by the decoder. (4) Decoding the segment and then rendering for playback. (5) Repeat Steps 2-4 until the end of the streaming session (either the user terminates the streaming or the end of the video has reached).

ViVo [13] has been one of the pioneering systems for volumetric video streaming on mobile devices. However, the system lacks support for a DASH implementation and does not support encoders other than Draco, limiting the range of encoded bitrates available to accommodate wider network conditions. Away from the V-PCC and Draco codecs, Kyungjin *et al.* targeted mobile point-cloud streaming and proposed a novel data structure called the Parallel Decodable Tree (PD-Tree) [17], which differs from traditional structures like Octree and KD-Tree. The GROOT framework, proposed based on this data structure, can achieve real-time decoding through parallel processing. These proposals are aiming to transform specific encodings of volumetric video for streaming, which are orthogonal work to VV-DASH.

Closely related to VV-DASH, we found `vvtk` [34], an open-source framework consisting of individual modules for encoding, decoding, rendering, and transmission. However, these modules are not streamlined to form an end-to-end streaming system. Moreover, `vvtk` provides only a simulation of DASH, without using an actual manifest (MPD) file, making it not suitable for conducting realistic research on volumetric DASH streaming. To the best of our knowledge, VV-DASH is the first general end-to-end framework that can work with different codecs and bitrate reduction algorithms, as well as efficient encoding and decoding processes. In particular, we propose a codec-agnostic DVV container format that wraps around volumetric video data (encoded in any format) along with information necessary for decoding and playback of DASH segments. Any existing or new video compression techniques can be incorporated into VV-DASH, as exemplified by the two case studies in the paper.

3 Design of VV-DASH

This section presents the design of VV-DASH. As shown in Figure 1, VV-DASH spans the entire streaming pipeline, from video source to rendering for playback. The video source module supports Video-on-Demand. The Encoder module prepares videos at multiple quality levels based on a bitrate ladder. The DASH Volumetric Video (DVV) Segmenter bundles encoded media content with metadata into DVV segments, akin to video segments in conventional DASH streaming. These segments are packaged by a DASH Packager into DASH segments along with an MPD file and stored on an HTTP server.

On the client side, VV-DASH is adaptable by any DASH client, allowing integration with any bitrate adaptation algorithm to select, request, and receive segments based on buffer status and network conditions. VV-DASH wraps a DASH client with a DVV parser, a decoder module, and a renderer. The DVV parser extracts embedded media and playback information from DVV segments. These are passed to a codec-specific decoder module, which decodes the segments and places them into a playback buffer for the renderer module to dequeue and render. VV-DASH also supports headless delivery of media content to external modules, enabling rendering on diverse devices (e.g., head-mounted displays) or for objective quality assessment tools.

Overall, VV-DASH is a comprehensive end-to-end framework for DASH streaming of VoD voluemtric video. Since bottlenecks are potentially presented in every part of this pipeline, VV-DASH is designed to be lightweight and resource-efficient for optimal performance. Not only is VV-DASH a complete setup that facilitates research on any part of the pipeline individually or as a whole, but it is also designed to be modular and flexible, allowing seamless integration of DASH volumetric video streaming into most existing DASH players. The rest of this section details VV-DASH's server-side (Section 3.1) and client-side (Section 3.2) designs.

3.1 VV-DASH Server-side

The Encoder module in VV-DASH adopts any specific encoder (*e.g.,* Google Draco [7] and MPEG V-PCC [9]), and encodes raw volumetric video into multiple quality levels as specified by the bitrate ladder. The Encoder module can also adopt codec-independent bitrate reduction algorithms such as down-sampling [14, 30] for simpler processing.

Diverse bitrate reduction algorithms impose different DASH implementations. For instance, V-PCC provides its own segmentation format, while Draco encodes content frame by frame, which is inefficient for DASH streaming. For a 30 fps video, each frame represents only 0.033 seconds. Frame-level streaming results in frequent requesting and fetching, leading to high control and communication overhead, as well as inefficient decoding. This would make DASH streaming impractical. VV-DASH introduces a novel DVV Segmenter that transforms any coded volumetric video into DASH-ready segments.

The DVV Segmenter can aggregate discrete frames or codec-specific segments, or wrap only one codec-specific segment, along with the necessary presentation and playback information for DASH. This would not only enable more efficient streaming of independently encoded frames but would also allow the system to support the streaming of both the independently encoded frames (as in Draco or down-sampled raw frames) and regular media segments (such as V-PCC V3C) within the same system, using a unified segment format. To the best of our knowledge, no such segment format currently exists for either raw or encoded point cloud-based media. Therefore, we propose the DVV segment format packing one or more encoded media contents (*e.g.,* frames or segments) for a specific playback duration.

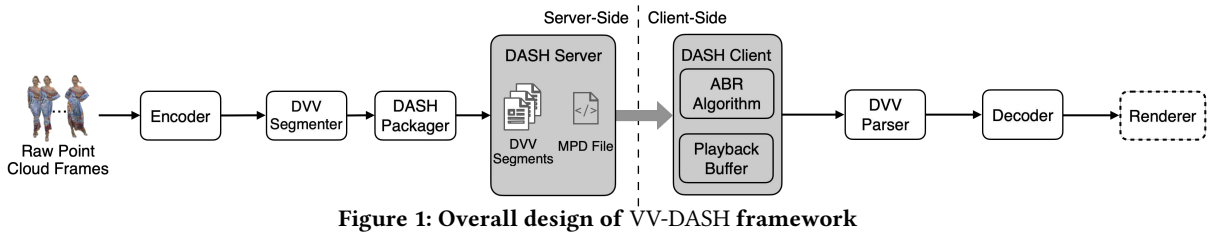

Figure 1: Overall design of VV-DASH framework

As shown in Figure 2, the DVV segment format bundles frames or codec-specific segments and includes essential information for decoding and playback. The bitstream starts with a 3-byte DVV format version and a 4-byte Header Encoding field (e.g., "JSON") to specify header serialization. A 4-byte field defines the header size, separating header and payload. A 2-byte Sequence Number field follows for error-checking, along with a 3-byte Timescale field specifying the segment's time units (e.g., 90,000 ticks per second). Before the header, a 4-byte Codec field identifies the codec (e.g., "DRCO" for Draco or "VPCC" for VPCC).

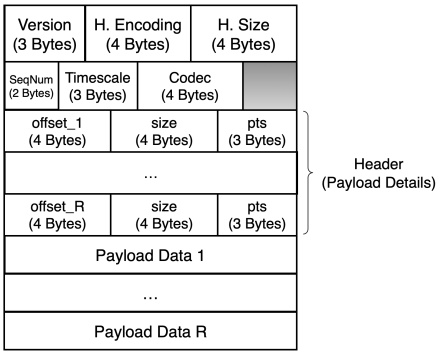

Figure 2: DVV segment format

The header contains metadata for each media payload, including a 4-byte offset for its position in the bitstream, a 4-byte size for its length, and a 3-byte PTS for its presentation timestamp (relative to the timescale). The payload section follows, containing the encoded data. This structured design ensures the DVV bitstream efficiently encapsulates both metadata and frame data, enabling precise access and synchronization during decoding and playback.

3.2 VV-DASH Client-side

The client side of VV-DASH builds on a typical DASH client, which streams DVV-formatted segments at appropriate quality levels. Upon receiving a segment, the DVV Parser identifies the codec and timescale, parses the header, and extracts encoded frames or codec-specific segments. These are passed to the Decoder module, which wraps the appropriate decoder (*e.g.,* Draco for `DRC` frames and V-PCC for `V3C` bitstream segments). The Decoder module uses parallel computing directives to accelerate decoding when possible. Since `DVV` payloads are independently decodable, VV-DASH employs multiprocessing on multi-core systems, with one decoder per payload, boosting the decoding rate. It also minimizes disk I/O by keeping the entire data flow in memory for further optimization.

Decoded frames are passed to the Renderer module for playback. The renderer retrieves frames from the buffer, loads the point cloud data and associated colors, and displays them in an `Open3D` window at the correct presentation timestamp. To the best of our knowledge, there is currently no dedicated rendering library for dynamic sequences of point clouds. Therefore, to ensure seamless playback, we update the points at their exact presentation timestamp in the same renderer window and display each frame for the appropriate duration of $1/fps$ seconds. To enhance usability, a walk-around view on 2D screens allows users to correct the 3D object in any direction using the cursor. Similar to the Decoder module, the Renderer module is designed to ensure efficient playback of video segments by utilizing asynchronous operations, multiprocessing, and concurrent programming techniques. Due to the lack of a real-time renderer for highly dense video datasets, VV-DASH also captures detailed analytical data from the streaming session (*e.g.,* stalls and quality switches from DASH) to aid research and examination regardless of the rendering being real-time. These analytical tools offer a practical playback alternative for dense volumetric video streaming until rendering techniques advance (or alternatively utilizing HMD-based native rendering)

4 Implementation and Experiment Setup

In VV-DASH, we use [2] as the DASH player, that implements a DASH protocol with bandwidth-based, buffer-based, and hybrid Adaptive Bitrate (ABR) algorithms. For our experiments, we utilized the bandwidth-based ABR.

The DASH player is wrapped with VV-DASH modules, as shown in Figure 1, and an NGINX [27] server serves as the HTTP server. To emulate bandwidth scenarios, we employ a network module using the Linux Traffic Control utility on a HAProxy [5] proxy. Positioned between the DASH server and player, the proxy throttles inbound and outbound traffic to accurately emulate desired bandwidth profiles. All components, including the server, client, and network emulator, are containerized and run on a laptop equipped with an Intel Core i9-14900HX CPU, NVIDIA GeForce RTX 4090 GPU, and 64GB of memory.

We evaluate VV-DASH using three point cloud-based volumetric video datasets: On the high quality end, we use the "Ricardo-10" video from the MVUB (Microsoft Voxelized Upper Bodies) dataset [21], featuring dense point clouds with an average of $1M$ points/frame and high spatial resolution of $1024x1024$ voxels per cube (*i.e.,* a spatial depth of 10). On the low quality end, we use the "Phil-9" video from the same dataset [21], with a sparse point cloud (average $334K$ points/frame) and lower spatial resolution ($512x512$ voxels/cube, spatial depth 9). Videos from the MVUB dataset have been used in proposals for volumetric streaming in [11, 24, 41]. In between, we use the "Longdress-10" video from the 8i Voxelized Full Bodies (8i VFB v2) dataset [4], with a relatively dense point cloud (average $834K$ points/frame) and high spatial resolution ($1024x1024$

voxels/cube, depth 10). Videos from the 8i VFB dataset are also widely used in volumetric streaming research [14, 17, 30]. All three videos have 30 fps, a common frame rate for smooth motion and detailed video playback. These datasets represent a good range of point-cloud densities and spatial depths, providing ample room for codec compression. Table 1 compares the average point cloud density and frame size (ranging from 7.43 to 21.05 *MB*). To ensure uniform test durations, each video is concatenated with itself to extend to 30 seconds.

Table 1: Video datasets

Video	Avg # of Points	Avg Frame Size	# of Frames
Ricardo-10	1.002M	21.05 *MB*	216
Longdress-10	834K	18.9 *MB*	300
Phil-9	334K	7.43 *MB*	245

We used the Open3D library [43] as the renderer for volumetric video playback, which is a widely used and powerful tool for 3D graphics visualization. We plan to extend VV-DASH's implementation with a Unity [36, 37] module to enable natural volumetric rendering in head-mounted displays like Meta Quest [22]. With this setup, we will present two case studies demonstrating VV-DASH's capability to support two codec standards: Draco (Section 5) and V-PCC (Section 6). Using bandwidth profiles covering respective bitrate ladders, we will evaluate performance improvements from DVV segmentation. Each section will conclude with results from active DASH sessions under different bandwidth profiles.

5 Case Study #1: Streaming with Draco

We encode all three videos with five rate parameter sets using Draco (via DracoPy [29]) to define a 5-level bitrate ladder, detailed in Table 2. Parameter settings are selected carefully to ensure smooth progression in bitrate and visual quality. Compression level (CL) values span the full range (1–10) with a step size of two. Position and texture quantization parameters (PQP and TQP), which control the number of bits for position and texture quantization, are optimized by analyzing commonly used position and color values across datasets and evaluating decoded frame quality. The bitrate ladder is shown in Table 2. The "Ricardo" video exhibits the widest bitrate range when encoded with Draco, while the "Phil" video achieves high compression with a smaller bitrate range. The "Longdress" video has the highest average bitrate.

Table 2: Draco bitrate ladder

Rate	CL	PQP	TQP	Average Bitrate (Mbps)		
				Longdress	Ricardo	Phil
R1	10	7	6	309.9	141.8	143.4
R2	8	9	8	431.1	276.6	197.5
R3	6	11	10	568.8	454.1	256.4
R4	4	13	12	749.7	838.0	348.4
R5	2	15	14	982.3	1175.7	453.0

Figure 3 illustrates server-side DVV segmentation, showing an example of encapsulating the first 30 frames of a video into a DVV segment. Each frame, independently encoded as a DRC-encoded frame, is placed in an individual payload within the DVV segment. For each DRC frame, the DVV header includes a record with its starting byte offset in the payload bitstream, its size, and its presentation timestamp (PTS) relative to the timescale. The PTS starts from the `SeqNum` (0 for the first frame) and increments by 3,000 per frame, as the timescale for this 1-second segment is 90,000.

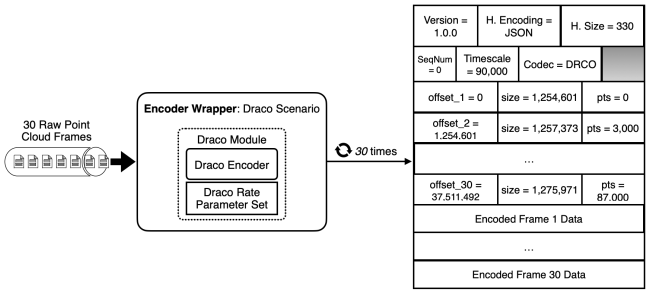

Figure 3: DVV segmentation and parallel encoding/decoding for Draco

Once a DVV segment containing Draco-encoded frames is received on the client side, frames are extracted based on the DVV header. Since each frame is independently decodable, they are decoded in parallel by separate instances of the Draco decoder. The original Draco decoder processes frames one at a time, requiring disk I/O for loading, decoding, and writing outputs, which introduces delays. To address this, we implement an in-memory Draco wrapper using DracoPy [29], allowing received and extracted frames to pass directly to decoders from memory. Decoded frames are then directly pushed into the renderer buffer, eliminating disk I/O delays entirely, as shown in Figure 3.

5.1 DVV Segmentation

The key enabling concept in VV-DASH for volumetric video streaming is DVV segmentation, which significantly enhances streaming performance. To create 1-second segments for testing videos, we package 30 frames into a DVV segment. DVV segmentation primarily improves average frame decoding time and system throughput. We conducted an experiment comparing decoding times for DVV segments and Draco frames. A DASH streaming session without bandwidth limits was run on the "Longdress" video to eliminate transmission bottlenecks and focus solely on decoding. For DVV segments, the DASH client sends all 30 frames in a segment to the Decoder module for parallel decoding. For Draco frames, a new decoding process is forked upon receiving each frame, maximizing parallelism for a fair comparison. As shown in Figure 4, DVV segmentation reduces average decoding time for 1-second of frames by up to 51%. This improvement is due to frames arriving in batches matching the frame rate (30 fps), enabling better CPU utilization. Process creation and scheduling overheads are minimized since 30 processes are launched simultaneously, rather than one every 0.033 seconds in the worst-case scenario for stall-free streaming.

Next, we compare the streaming experience using 1-second DVV segments versus individual frames (Draco default). With DVV segments, the DASH client requests a new segment at most every 1 second, provided segment transmission completes within this time

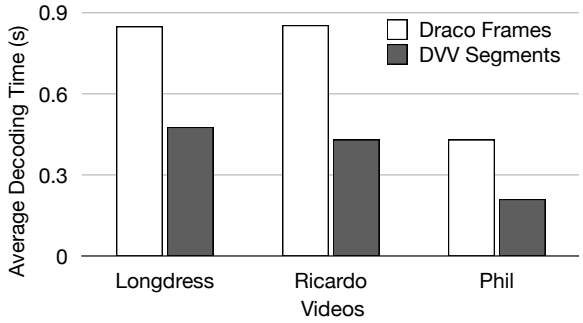

Figure 4: Impact of DVV segmentation on decoding time

for smooth streaming. Without segmentation, the player must request 30 frames consecutively within 1 second (*i.e.,* one request every 0.033 seconds at worst) to maintain the same quality level. To assess the impact of DVV segmentation, we created bandwidth profiles enabling each video to stream at specific quality levels. For example, streaming the "Longdress" video at quality level *R2* (as per Table 2) requires bandwidth between 431.1 *Mbps* and 568.8 *Mbps*. We selected 475 *Mbps*, ensuring the quality adaptation algorithm consistently picks *R2* for DVV segments. Similar bandwidth profiles were created for levels *R2*–*R5*, detailed in Table 3. We measured the average bitrate achieved during streaming for all three videos under these bandwidth profiles. Figure 5 shows that DVV segments consistently achieve the target quality level, whereas Draco frames mix the target and lower quality levels. Bitrates improved by 36.2%, 44.9%, and 32.4% for the "Longdress," "Ricardo," and "Phil" videos, respectively, leading to an overall throughput gain under identical network conditions.

Table 3: Bandwidth profiles for evaluating impact of DVV segmentation

Profiles	Bandwidth Setting (Mbps)		
	Longdress	Ricardo	Phil
L2	475	300	215
L3	610	500	270
L4	850	950	380
L5	1050	1280	490

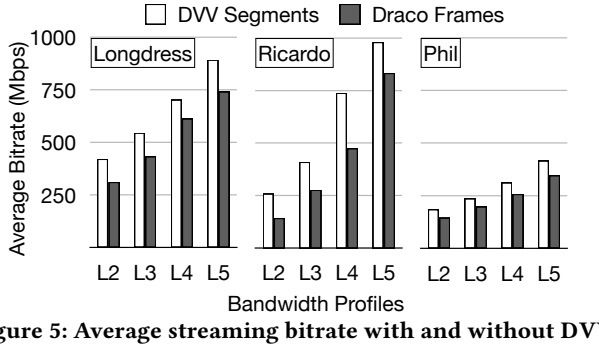

Figure 5: Average streaming bitrate with and without DVV segmentation

The bitrate gain from DVV segmentation is due to reduced protocol overhead. We analyzed the streaming timeline and plotted segment-by-segment and frame-by-frame transmissions for the "Longdress" video under a fixed bandwidth of 900 *Mbps* (maintaining quality level R4). As shown in Figure 6, the average turnaround time for requesting and downloading the first DVV segment is 0.91 seconds, while individually requesting and downloading 30 Draco frames from the same segment takes over 1 second. This pattern is consistent across other DVV segments and Draco frames of this video, explaining the 13.2% throughput gain. The improvement not only allows the player to switch to a higher quality level if close to its boundary—thereby improving the Quality-of-Experience under the same network conditions, but also helps prevent stalls or shorten stall duration under poor network conditions.

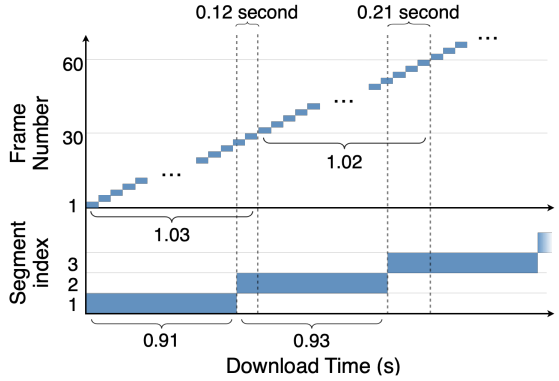

Figure 6: Segment-by-segment download time versus frame-by-frame download timeline from streaming "Longdress" video with 900*Mbps* fixed bandwidth

5.2 DASH with Draco

To demonstrate VV-DASH's adaptive bitrate streaming capability, we conducted experiments on the Draco-encoded "Longdress" video under two bandwidth profiles (as similar to [1]). The first profile, shown in Figure 7 (top), was designed for smooth, stall-free streaming with quality switches across all five quality levels. Bandwidth levels were adjusted every 5 seconds to enable DASH streaming at the marked quality levels. As shown in Figure 7 (top), the DASH session successfully adapted to the available bandwidth and selected appropriate quality levels.

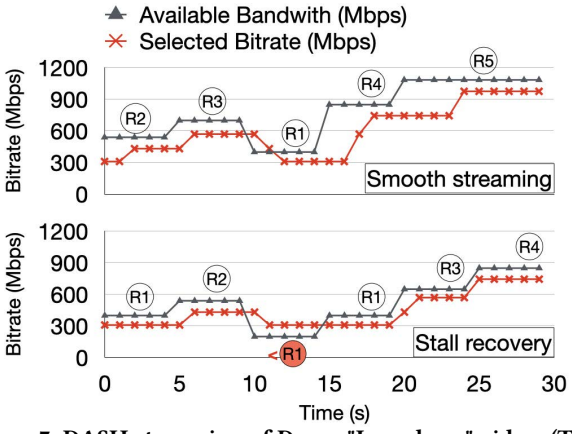

Figure 7: DASH streaming of Draco "Longdress" video: (Top) Smooth playback under sufficient bandwidth supply, (Bottom) Stall recovery after a bandwidth drop at 10 second.

The second profile, shown in Figure 7(bottom), included a stall period. Similar to the first profile, bandwidth levels were adjusted every 5 seconds, but at the 11th second, the bandwidth dropped below level R1. This drop lasted 5 seconds, exceeding the buffer

duration (4 seconds) and causing the buffer to deplete, resulting in playback stalls. As shown in Figure 7(bottom), the DASH session streamed at R1 for 9 consecutive segments following the bandwidth drop, with 1.32 seconds of stall observed during playback.

6 Case Study #2: Streaming with V-PCC

TMC2 is a reference implementation of the MPEG V-PCC codec standard [10], providing both encoder and decoder functionality. The encoder processes raw point cloud frames with a set of configurations, performing intra- and inter-frame 2D encoding, and outputs an encoded V3C bitstream. Key rate parameters include Geometry Quantization (GeoQP) and Attribute Quantization (AttQP), which control compression levels for geometric and attribute data, respectively—higher QP values yield greater compression and lower bitrates. Occupancy Precision (OP) defines the granularity of the occupancy map, with higher precision ensuring more accurate point capture within the voxel grid. For DASH streaming, we define a 5-level bitrate ladder. Rate configurations for V-PCC are based on recommended sample configurations from the TMC2 implementation [10], using the reference HM implementation [31, 35] as the 2D video encoder. As the original TMC2 decoder is slow (*e.g.,* taking over 40 seconds to decode a 1-second "Longdress" segment), we use the faster TMC2-RS [3], which is up to 15 times faster and has been adopted in frameworks like [34]. VV-DASH employs TMC2-RS as the V-PCC decoder. The GeoQP, AttQP, and OP settings for the bitrate ladder are detailed in Table 4.

Table 4: V-PCC bitrate ladder

Rate	GeoQP	AttQP	OP	Average Bitrate (Mbps)		
				Longdress	Ricardo	Phil
R1	32	42	4	4.35	1.84	1.84
R2	28	37	4	7.49	2.48	2.72
R3	24	32	4	13.1	3.68	4.48
R4	20	27	4	24.5	5.84	8.32
R5	16	22	2	44.2	10.4	15.3

Figure 8 illustrates the integration of the V-PCC encoder into VV-DASH and a sample crafted DVV segment. The DVV Segmenter can pack any number of V3C bitstreams (Figure 8 shows one example). In this case, since the only existing V3C bitstream bundles all 30 frames into a segment, the entire timescale is assigned to the segment's PTS, which is 90,000, equivalent to 1 second in length. Utilizing the fast TMC2-RS V-PCC implementation and configuring the DVV segments with 2 frames bundled in each V3C bitstream (adding up to 30 frames in total, which are packed into a payload to form a 30-frame, 1-second segment) with negligible compression ratio loss enables us to conduct a DASH streaming session for the "Phil" video. This segmentation preserves the compression ratio while enabling quick decoding of individual V3C bitstreams, making practical streaming possible.

6.1 DASH with V-PCC

Given the real-time decoding rate achieved for the "Phil" video, we are able to showcase DASH streaming with V-PCC. Similar to the Draco case, we created two bandwidth profiles: one for smooth streaming and one for stall recovery. The bandwidth profile along with the target quality levels is illustrated in Figure 9. Bandwidth

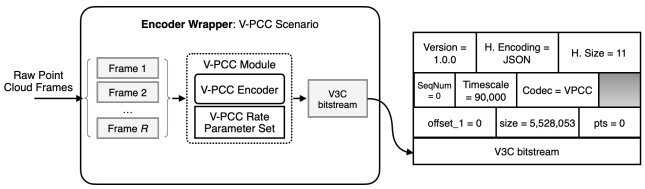

Figure 8: Parallel encoding/decoding and DVV segmentation for V-PCC

levels were adjusted every 5 seconds to enable DASH streaming at the quality levels shown in Figure 9. The DASH session with V-PCC, under the first profile, successfully adapted to the varying bandwidth conditions and selected the appropriate quality level accordingly. With the second profile, the DASH session streamed at the R1 level for four consecutive segments after the bandwidth drop, with stalls during playback.

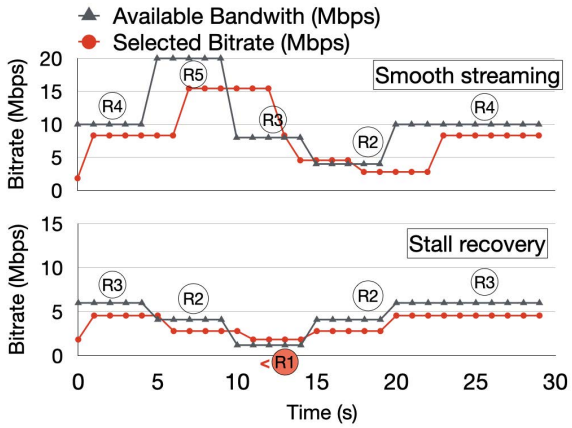

Figure 9: DASH streaming of V-PCC "Phil" video: (Top) Smooth playback under sufficient bandwidth supply, (Bottom) Stall recovery after a bandwidth drop at 10 second.

7 Conclusion

In this study, we proposed VV-DASH, a high-performance end-to-end framework for volumetric video streaming over DASH. VV-DASH improves performance through DVV segmentation, and efficient and parallel computing techniques. We presented two case studies: streaming with Draco and V-PCC, as well as demonstrating support for any bitrate reduction module and uniform packing of encoded/compressed video content. Consistent results in both studies confirm VV-DASH's effectiveness with DASH. We analyzed the impact of DVV segmentation on streaming throughput and decoding rate. Combining DVV segmentation with DASH results in up to 51% faster decoding rate Draco-encoded videos, and a 13.2% system throughput gain under the identical network conditions (*i.e.,* reducing bandwidth demands for smooth playback). In summary, VV-DASH enables practical adaptive volumetric video streaming and lays a foundation for and facilitates future research. While VV-DASH is a complete setup, future work includes containerizing DVV components into an ISOBMFF-compliant segment [8, 38] to facilitate broader modular adoption.

References

[1] Navid Akbari, Reza Hedayati Majdabadi, Akram Ansari, Mea Wang, and Diwakar Krishnamurthy. 2023. iStream: A Flexible Container-Based Testbed for Multimedia Streaming. In *2023 IEEE 6th International Conference on Multimedia Information Processing and Retrieval (MIPR)*. 1–6. https://doi.org/10.1109/MIPR59079.2023.00038

[2] Akram Ansari and Mea Wang. 2023. iStream Player: A Versatile Video Player Framework. In *Proceedings of the 33rd Workshop on Network and Operating System Support for Digital Audio and Video* (Vancouver, BC, Canada) *(NOSSDAV '23)*. Association for Computing Machinery, New York, NY, USA, 65–71. https://doi.org/10.1145/3592473.3592569

[3] B. Clement. 2022. Fast VPCC Point Cloud Decoder. https://github.com/benclmnt/tmc2-rs?tab=readme-ov-file

[4] Eugene d'Eon, Bob Harrison, Taos Myers, and Philip A Chou. 2017. 8i Voxelized Full Bodies-A Voxelized Point Cloud Dataset. *ISO/IEC JTC1/SC29 Joint WG11/WG1 (MPEG/JPEG) input document WG11M40059/WG1M74006* 7, 8 (2017), 11.

[5] HAProxy Enterprise. 2024. HAProxy. https://www.haproxy.com

[6] Google. 2024. https://youtube.com.

[7] Google. 2024. Draco: A Library for Compressing and Decompressing 3D Geometric Meshes and Point Clouds. https://github.com/google/draco

[8] MPEG Group. 2008. *Information Technology – Coding of Audio- Visual Objects – Part 12: ISO base media file format.* standard. ISO/IEC.

[9] MPEG Group. 2023. *Information Technology — Coded Representation of Immersive Media — Part 5: Visual Volumetric Video-based Coding (V3C) and Video-based Point Cloud Compression (V-PCC).* standard. ISO/IEC.

[10] MPEG Group. 2024. TMC2: Video Codec Based Point Cloud Compression (V-PCC) Test Model. https://github.com/MPEGGroup/mpeg-pcc-tmc2

[11] Shuai Gu, Junhui Hou, Huanqiang Zeng, Hui Yuan, and Kai-Kuang Ma. 2020. 3D Point Cloud Attribute Compression Using Geometry-Guided Sparse Representation. *IEEE Transactions on Image Processing* 29 (2020), 796–808. https://doi.org/10.1109/TIP.2019.2936738

[12] Srinivas Gudumasu, Gireg Maury, Ariel Glasroth, and Ahmed Hamza. 2023. Adaptive Streaming of Visual Volumetric Video-based Coding Media. In *Proceedings of the 15th International Workshop on Immersive Mixed and Virtual Environment Systems*. ACM, Vancouver, Canada, 30–33.

[13] Bo Han, Yu Liu, and Feng Qian. 2020. ViVo: Visibility-aware Mobile Volumetric Video Streaming. In *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking* (London, United Kingdom) *(MobiCom '20)*. Association for Computing Machinery, New York, NY, USA, Article 11, 13 pages. https://doi.org/10.1145/3372224.3380888

[14] Mohammad Hosseini and Christian Timmerer. 2018. Dynamic Adaptive Point Cloud Streaming. In *Proceedings of the 23rd Packet Video Workshop*. IEEE, San Jose, CA, USA, 25–30.

[15] Te-Yuan Huang, Ramesh Johari, Nick McKeown, Matthew Trunnell, and Mark Watson. 2014. A Buffer-based Approach to Rate Adaptation: Evidence from a Large Video Streaming Service. In *Proceedings of the 2014 ACM conference on SIGCOMM*. ACM, Chicago, IL, USA, 187–198.

[16] Jonathan Kua, Grenville Armitage, and Philip Branch. 2017. A Survey of Rate Adaptation Techniques for Dynamic Adaptive Streaming over HTTP. *IEEE Communications Surveys & Tutorials* 19, 3 (2017), 1842–1866.

[17] Kyungjin Lee, Juheon Yi, Youngki Lee, Sunghyun Choi, and Young Min Kim. 2020. GROOT: A Real-time Streaming System of High-fidelity Volumetric Videos. In *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*. MobiCom, Virtual, 1–14.

[18] Jie Li, Cong Zhang, Zhi Liu, Richang Hong, and Han Hu. 2022. Optimal Volumetric Video Streaming With Hybrid Saliency based Tiling. *IEEE Transactions on Multimedia* 25 (2022), 2939–2953.

[19] Chenghao Liu, Imed Bouazizi, and Moncef Gabbouj. 2011. Rate Adaptation for Adaptive HTTP Streaming. In *Proceedings of the second annual ACM conference on Multimedia systems*. ACM, San Jose, CA, USA, 169–174.

[20] Chenghao Liu, Imed Bouazizi, Miska M Hannuksela, and Moncef Gabbouj. 2012. Rate Adaptation for Dynamic Adaptive Streaming Over HTTP in Content Distribution Network. *Signal Processing: Image Communication* 27, 4 (2012), 288–311.

[21] Charles Loop, Qin Cai, Sergio Orts Escolano, and Philip A. Chou. 2016. Microsoft Voxelized Upper Bodies - A Voxelized Point Cloud Dataset. http://plenodb.jpeg.org/pc/microsoft/

[22] "Meta". 2024. Meta Quest 3D. https://www.meta.com/ca/quest/quest-3s/.

[23] Netflix. 2024. https://netflix.com

[24] Dat Thanh Nguyen, Maurice Quach, Giuseppe Valenzise, and Pierre Duhamel. 2021. Lossless Coding of Point Cloud Geometry Using a Deep Generative Model. *IEEE Transactions on Circuits and Systems for Video Technology* 31, 12 (2021), 4617–4629. https://doi.org/10.1109/TCSVT.2021.3100279

[25] Roger Pantos and William May. 2017. HTTP Live Streaming.

[26] Feng Qian, Bo Han, Jarrell Pair, and Vijay Gopalakrishnan. 2019. Toward Practical Volumetric Video Streaming on Commodity Smartphones. In *Proceedings of the 20th International Workshop on Mobile Computing Systems and Applications*. ACM, CA, USA, 135–140.

[27] Will Reese. 2008. Nginx: The High-Performance Web Server and Reverse Proxy. *Linux Journal* 2008, 173 (2008), 2.

[28] Michael Rudolph and Amr Rizk. 2022. View-Adaptive Streaming of Point Cloud Scenes through Combined Decomposition and Video-based Coding. In *Proceedings of the 1st International Workshop on Advances in Point Cloud Compression, Processing and Analysis* (Lisboa, Portugal) *(APCCPA '22)*. Association for Computing Machinery, New York, NY, USA, 41–49. https://doi.org/10.1145/3552457.3555732

[29] seung lab. 2024. Python Wrapper for Google's Draco Mesh Compression Library. https://github.com/seung-lab/DracoPy

[30] Yuang Shi, Pranav Venkatram, Yifan Ding, and Wei Tsang Ooi. 2023. Enabling Low Bit-rate MPEG V-PCC-encoded Volumetric Video Streaming With 3D Subsampling. In *Proceedings of the 14th Conference on ACM Multimedia Systems*. ACM, Toronto, Canada, 108–118.

[31] HEVC Test Model (HM) Reference Software. 2013. The H.265 Reference Software HM. https://github.com/listenlink/HM

[32] Thomas Stockhammer. 2011. Dynamic Adaptive Streaming over HTTP– Standards and Design Principles. In *Proceedings of the second annual ACM conference on Multimedia systems*. ACM, San Jose, CA, USA, 133–144.

[33] T. Stockhammer. 2011. Dynamic Adaptive Streaming over HTTP: Standards and Design Principles. In *in Proc. of the 3rd ACM Multimedia Systems Conference (MMSys)*. ACM, New York, NY, USA, 133–144.

[34] NUS Volumetric Video Streams. 2022. A Toolkit for Volumetric Video Research. https://github.com/nus-vv-streams/vvtk/tree/main

[35] G. J. Sullivan, J. R. Ohm, W. J. Han, and T. Wiegand. 2012. Overview of the Hight Efficiency Video Coding (HEVC) Standard. *IEEE Transactions on Circuit and Systems for Video Technology* 22, 12 (Sept 2012), 1649–1668.

[36] Keijiro Takahashi. 2017. Pcx: Point Cloud Importer & Renderer for Unity. https://github.com/keijiro/Pcx

[37] Unity Technologies. 2024. Unity - Real-time Development Platform. https://unity.com. Accessed: 2024-11-23.

[38] Christian Timmerer and Christopher Müller. 2010. HTTP Streaming of MPEG Media. *Streaming Day* 10 (2010), 1–1.

[39] Jeroen van der Hooft, Hadi Amirpour, Maria Torres Vega, Yago Sanchez, Raimund Schatz, Thomas Schierl, and Christian Timmerer. 2023. A Tutorial on Immersive Video Delivery: From Omnidirectional Video to Holography. *IEEE Communications Surveys & Tutorials* 25, 2 (2023), 1336–1375. https://doi.org/10.1109/COMST.2023.3263252

[40] Yizong Wang, Dong Zhao, Huanhuan Zhang, Teng Gao, Zixuan Guo, Chenghao Huang, and Huadong Ma. 2024. Bandwidth-Efficient Mobile Volumetric Video Streaming by Exploiting Inter-Frame Correlation. *IEEE Transactions on Mobile Computing* 1, 1 (2024), 1–15.

[41] Yizong Wang, Dong Zhao, Huanhuan Zhang, Chenghao Huang, Teng Gao, Zixuan Guo, Liming Pang, and Huadong Ma. 2023. Hermes: Leveraging Implicit Interframe Correlation for Bandwidth-Efficient Mobile Volumetric Video Streaming. In *Proceedings of the 31st ACM International Conference on Multimedia*. ACM, Ottawa, Canada, 9185–9193.

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

[43] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. 2018. Open3D: A Modern Library for 3D Data Processing. *arXiv preprint arXiv:1801.09847* 1, 1 (2018), 1–10.

[44] Junni Zou, Chenglin Li, Chengming Liu, Qin Yang, Hongkai Xiong, and Eckehard Steinbach. 2019. Probabilistic Tile Visibility-based Server-side Rate Adaptation for Adaptive 360-degree Video Streaming. *IEEE Journal of Selected Topics in Signal Processing* 14, 1 (2019), 161–176.