DeltaStream: 2D-Inferred Delta Encoding for Live Volumetric Video Streaming

Hojeong Lee
Korea University
hojeong0507@korea.ac.kr

Yu Hong Kim
University of Colorado Boulder
yuki9447@colorado.edu

Sangwoo Ryu
POSTECH
rswoo@postech.ac.kr

James Won-Ki Hong
POSTECH
jwkhong@postech.ac.kr

Sangtae Ha
University of Colorado Boulder
sangtae.ha@colorado.edu

Seyeon Kim
Korea University
seyeon625@korea.ac.kr

Abstract

Live volumetric video streaming enables immersive user experiences but poses significant challenges due to the high bandwidth requirement that 3D representations entail. Recent research has focused on reducing volumetric video bandwidth, but it struggles to effectively address temporal redundancy under real-time constraints, limiting its applicability for live streaming scenarios. To address these challenges, we present *DeltaStream*, a novel live volumetric video streaming system that efficiently encodes 3D point clouds by leveraging 2D information. By utilizing 2D RGB and depth frames, *DeltaStream* efficiently infers inter-frame changes to reduce the streaming bandwidth of volumetric video. Furthermore, *DeltaStream* introduces an adaptive block-based approach that can reduce the client-side decoding load. Through extensive evaluations, our results demonstrate that *DeltaStream* reduces bandwidth by up to 71% with 1.63× faster decoding speed while maintaining visual quality compared to state-of-the-art systems.

CCS Concepts

• **Human-centered computing → Mixed / augmented reality**; **Virtual reality**; • **Information systems → Multimedia streaming**.

Keywords

live volumetric video streaming, point cloud, inter-frame compression

ACM Reference Format:
Hojeong Lee, Yu Hong Kim, Sangwoo Ryu, James Won-Ki Hong, Sangtae Ha, and Seyeon Kim. 2025. *DeltaStream*: 2D-Inferred Delta Encoding for Live Volumetric Video Streaming. In *The 23rd Annual International Conference on Mobile Systems, Applications and Services (MobiSys '25), June 23–27, 2025, Anaheim, CA, USA*. ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3711875.3729131

1 Introduction

Volumetric video streaming is an emerging technology that facilitates highly immersive and interactive user experiences in augmented, virtual, and mixed reality (AR/VR/MR) applications. Such experiences are made possible by representing each video frame as a 3D space or object using point clouds or meshes, providing six degrees of freedom (6-DoF) movements for users. Unlike on-demand volumetric video streaming, which delivers pre-recorded contents, live volumetric video streaming includes the entire pipeline from capture and volumetric content generation to delivery and rendering. Live streaming applications, such as remote surgery, virtual concerts, and telepresence [18, 32] require real-time performance, including maintaining high frame rates (e.g., over 30 frames per second) and minimizing end-to-end latency.

While volumetric videos provide an immersive experience, streaming the volumetric videos requires extremely high network bandwidth [48]. For example, streaming the raw point cloud frames in 8i dataset at 30 FPS demands bandwidth over 3 Gbps [21]. To address the challenges of large data sizes in volumetric video frames, video compression is a crucial aspect, which reduces the size to an acceptable bandwidth for streaming. Although several methods have been adopted for volumetric content compression [5, 45], they do not address temporal correlation between frames, which is the key factor to improving compression efficiency in 2D video compressions and decompressions (CODECs). 2D video compression algorithms, such as H.264/AVC [44, 50], achieve high compression rates by leveraging both spatial redundancy within frames and temporal redundancy between consecutive frames. They employ techniques such as motion estimation and compensation to exploit temporal correlations, significantly reducing the amount of data to be encoded.

Nevertheless, unlike in 2D, encoding methods that leverage temporal correlations between adjacent frames remain largely underexplored for volumetric videos. Focusing on point cloud as the representation of 3D video frame in this paper, we identify two key challenges in addressing temporal correlations for live volumetric video streaming.

First, the data representation of a point cloud frame is inherently irregular. In the case of 2D video, as a frame has a structured (i.e., fixed $width \times height$) representation, it is straightforward to process spatial and temporal correlation (e.g., by a subtracting frames to compute the difference). However, as a point cloud is an unstructured set of points and has a different number of points across frames, defining subtraction operations between point clouds is difficult. Therefore, it is challenging to compute and identify temporal and spatial correlations between point clouds.

This work is licensed under a Creative Commons Attribution 4.0 International License.
MobiSys '25, Anaheim, CA, USA
© 2025 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-1453-5/2025/06
https://doi.org/10.1145/3711875.3729131

Second, the computational complexity of 3D data processing, including encoding and decoding, is excessively high. The high-precision (x, y, z) position coordinates (i.e., 4 bytes for each coordinate) and (R, G, B) color attributes in point clouds, along with their irregular structure significantly increases computational complexity, necessitating efficient processing on resource-limited devices like head-mounted displays (HMDs) [1, 10].

Tackling these fundamental and practical challenges, we propose *DeltaStream*, the first live volumetric video streaming system that leverages 2D information to efficiently infer the *delta*[1] between volumetric video frames and reduce bandwidth. The key insight of *DeltaStream* is that 3D data is derived from 2D data, such as RGB and depth frames. Therefore, our approach aims to efficiently leverage 2D information to reduce the overhead associated with 3D processing and bandwidth usage, thereby enabling live volumetric video streaming. *DeltaStream* consists of the following three key components:

(1) The first component of *DeltaStream* involves transforming 2D temporal information into 3D. As previously discussed, the unstructured representation of point clouds makes it highly challenging to compute the differences between two frames efficiently.

To address this issue, we propose an inverse transformation method that maps 3D point cloud back to 2D frame to utilize 2D temporal information in the point cloud. This approach leverages pre-shared camera parameters.

(2) The second component is the set of two block-based delta encoding methods that address temporal redundancy between consecutive point cloud frames: 3D motion vectors and delta point cloud. *DeltaStream* classifies blocks in 2D frames into *matched blocks* and *mismatched blocks*. *Matched blocks* are efficiently compressed using 3D motion vectors, whereas *mismatched blocks* are handled with delta point clouds. Such behavior is analogous to using residual blocks in 2D video encoding. For 3D motion vectors, we propose a method that leverages 2D motion vectors extracted from the 2D video CODEC and combines them with depth information to generate accurate 3D motion vectors.

(3) The third component addresses client-side computation to ensure real-time performance by leveraging 3D motion vectors and delta point clouds. While these methods reduce bandwidth by utilizing temporal correlation, they can introduce additional computational costs on the client when updating the previous point cloud frame to the current frame. To mitigate this, the server dynamically adjusts delta encoding parameters based on the observed latency characteristics of 3D motion vector updates and delta point cloud decoding. This supports to meet real-time constraints for live volumetric video streaming.

We extensively evaluate *DeltaStream* on a wide range of environments with various network bandwidths, client compute power, scene contents, and multiple RGB-D cameras. Our experiment results show that *DeltaStream*, compared to the state-of-the-art system, MetaStream [24], reduces the bandwidth by 71% for static scenes and 49% for dynamic scenes with 1.15 to 1.63× faster decoding speed while maintaining visual quality. As a result, *DeltaStream* successfully enables real-time volumetric video streaming by achieving a reliable 30 FPS and is broadly applicable to any AR/VR application such as 3D video conferencing that streams 3D point cloud objects live-captured from multiple depth cameras.

[1] *delta* refers to the difference between the previous and current frames.

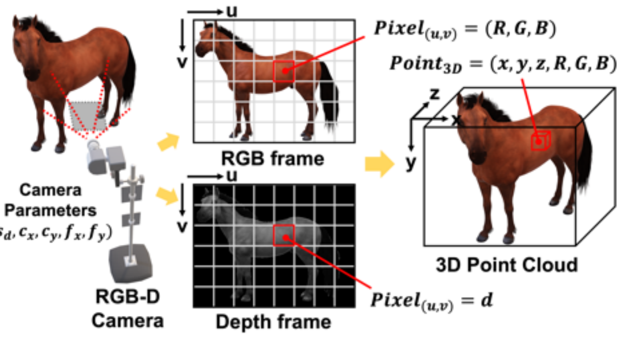

u
v
$Pixel_{(u,v)} = (R, G, B)$
$Point_{3D} = (x, y, z, R, G, B)$
z
x
y
RGB frame
Camera Parameters $(s_d, c_x, c_y, f_x, f_y)$
RGB-D Camera
Depth frame
3D Point Cloud
$Pixel_{(u,v)} = d$

Figure 1: Generation of the 3D point cloud from an RGB-D camera. A pixel in RGB and depth frame is mapped to a point in the 3D point cloud frame combined with camera parameters.

The main contributions of this paper are three-fold:

- We identify the challenges in existing live volumetric video streaming and explore the challenges in leveraging 2D information for 3D point cloud encoding.
- We propose *DeltaStream*, the first live volumetric video streaming system that leverages temporal redundancies inferred from 2D frames to efficiently encode point cloud frames, thereby reducing bandwidth usage.
- Based on extensive experiments, we verify that *DeltaStream* successfully achieves a reliable 30 FPS streaming performance, by reducing the bandwidth usage up to 71% with 1.63× faster decoding speed, compared to the state-of-the-art methods.

2 Background

2.1 Construction of a 3D Video Frame

While 2D video frames usually consist of a 3-tuple data (e.g., RGB or YUV), 3D video frames are mostly represented by two methods: point cloud and mesh. A point cloud is a set of points in a 3-dimensional space where each point is a 15 byte sized 6-tuple. In detail, a point consists of a 3D coordinate using 4 bytes per axis (i.e., x, y, z) and 1 byte each for the color channels (i.e., R, G, B). Mesh, on the other hand, connects the points with edges to form polygons, typically triangles, creating a contiguous surface representation of a 3D object. Point clouds can be generated in several ways. One method is to capture an object using multiple RGB cameras, followed by considerable post-processing of the images to combine them into a point cloud [21, 27]. Another way to generate a point cloud is to capture color and depth frames using a depth camera (RGB-D), such as Intel RealSense [6] and Orbbec Femto [12]. As illustrated in Figure 1, a point cloud can be directly created from the captured frame using the camera parameters according to Eq. (1):

$$z = \frac{d}{s_d},\ x = \frac{(u - c_x) \cdot z}{f_x},\ y = \frac{(v - c_y) \cdot z}{f_y} \tag{1}$$

where u and v are 2D pixel coordinates and d is the depth value corresponding to the pixel. Note that depth scale s_d, principal point offsets c_x and c_y, focal lengths f_x and f_y are all camera intrinsic parameters, which are fixed values. Each point is generated as long as the pixel has a valid depth value. As the latter approach does not involve excessive computation, it is appropriate for real-time streaming. When using multiple depth cameras, point clouds are generated from the color and depth captures of each camera and are aligned into a single coordinate system.

2.2 2D and 3D Video CODECs

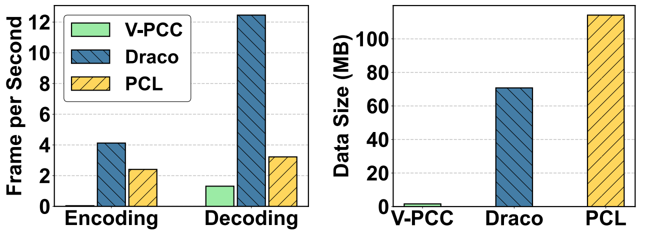
2D video CODECs have been extensively researched and developed through the leading efforts of the Moving Picture Experts Group (MPEG). This has resulted in the development of efficient video CODECs, such as advanced video coding (AVC) [44, 50] and high-efficiency video coding (HEVC) [47]. The principle behind the CODECs is to encode only the differences between frames (i.e., delta information) with spatial and temporal correlation. To reduce the temporal redundancy, the CODECs use motion vectors instead of directly calculating these differences [35, 58]. The performance of those CODECs can be further optimized through hardware acceleration [11].

While 2D video CODECs have been developed over the decades, point cloud compression techniques are still in their early stages. MPEG is actively working on video-based point cloud compression (V-PCC) [23, 46], which projects the point cloud onto multiple planes from various angles and then utilizes existing 2D video CODECs for compression, albeit at the cost of high encoding latency. Consequently, V-PCC is not well-suited for live volumetric video streaming and is primarily used for storage purposes. Other approaches are based on tree structures. Draco [5] and Point Cloud Library (PCL) [45], widely used 3D CODECs, utilize KD-tree [15] and Octree [40], respectively. Both Octree and KD-tree divide the 3D space into smaller regions recursively, mapping each point in the space with a node in the tree structure. The main difference between the two types of tree structures is that every node in an Octree has 8 child nodes, whereas each node in a KD-tree can have a varying number of child nodes. However, these state-of-the-art point cloud compression techniques overlook the temporal correlation between neighboring point cloud frames, failing to encode 3D video frames efficiently compared to 2D video CODECs.

3 Motivation

3.1 Inefficiency of Existing Point Cloud Compression Methods

Transmitting raw point cloud frames requires an extremely high bandwidth (e.g., over 3 Gbps for the 8i dataset [21] to achieve 30 FPS), making point cloud compression essential before transmission. Alongside reducing bandwidth usage, encoding and decoding latencies must be addressed to enable real-time streaming at 30 FPS. To conduct a motivational study comparing the encoding and decoding speed, as well as the data size, of three widely used point cloud compression methods, we use the 8i dataset [21] as a benchmark, which is one of the most commonly used datasets in on-demand volumetric video streaming scenarios [34, 38, 55]. As shown in Figure 2, V-PCC offers high compression efficiency by leveraging a 2D video CODEC to exploit the temporal correlation but suffers from extremely slow encoding (<1 FPS) and decoding (<2 FPS) performance. V-PCC's high latency [16, 34, 53] and excessive computational demands [30] make it unsuitable for real-time volumetric video streaming. In contrast, while Draco and PCL achieve lower compression efficiency, they offer much faster encoding and decoding speeds compared to V-PCC. Notably, Draco provides significantly faster decoding speeds and maintains stable performance even when encoding noisy point clouds [52]. Consequently, recent volumetric video streaming systems [20, 24, 25, 34, 55] are predominantly designed using Draco, the state-of-the-art point cloud compression technique.



(a) Average encoding and decoding speed. **(b) Average data size for 30 frames after compression.**

Figure 2: Point cloud compression performance on Intel Core i9-14900K CPU using high resolution *longdress* sequence from 8i dataset [21].

While Draco [5] is widely used to support real-time streaming in many systems, it has a critical inefficiency as it employs a frame-by-frame compression approach, ignoring the temporal redundancy between consecutive point cloud frames. The same limitation applies to PCL point cloud compression. To overcome these challenges, *DeltaStream* introduces a novel approach that effectively reduces temporal redundancy, significantly lowering bandwidth requirements.

3.2 High Computational Cost in Point Cloud Inter-Frame Compression

Addressing temporal redundancy between point cloud frames entails significant challenges because: **(1)** the number of points in the previous and current point cloud frames can be different, **(2)** the inherently unstructured nature of point clouds results in no explicit geometric correspondence between frames, making it challenging to identify spatial relationships across consecutive frames, and **(3)** processing point clouds is computationally intensive, as it requires additional processing steps compared to 2D video frames and takes 6-tuple data (i.e., (x, y, z, R, G, B)) as input.

Encoding and decoding bottleneck. Due to the high computational cost, achieving real-time performance remains a significant challenge. Mekuria *et al.* [41] proposed a 3D cube block (i.e., $l \times l \times l$) based inter-prediction approach, claiming nearly real-time encoding. However, the encoding takes approximately 1 second ($\approx$ 1 FPS) for each frame. Deep learning-based inter-frame compression methods struggle to achieve real-time decoding even with GPU support (e.g., decoding takes 0.714 seconds for each frame in [13])

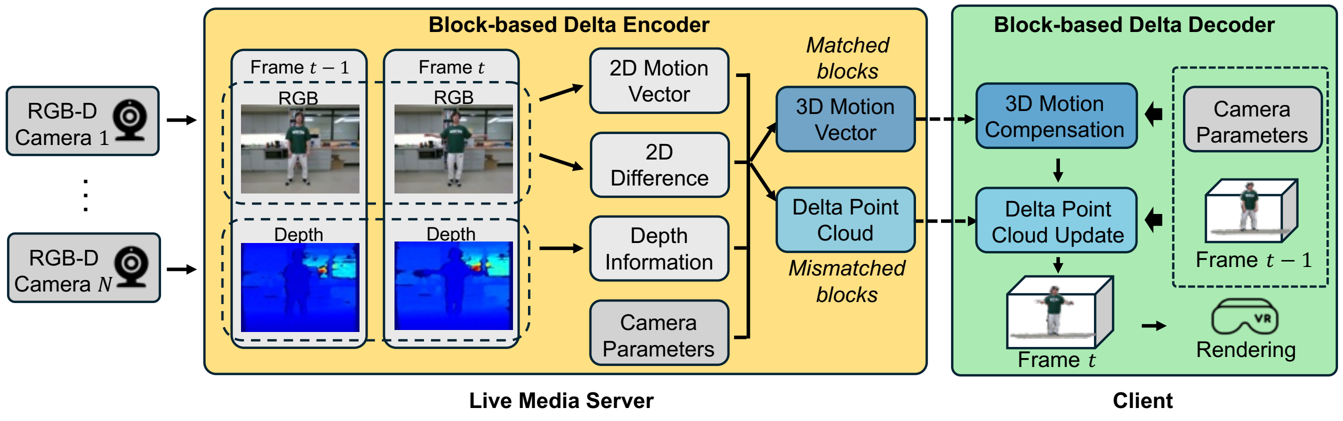


Figure 3: *DeltaStream* overview. Block-based Delta Encoder in the live media server performs delta encoding leveraging 2D information for each camera input.

and inevitably become a bottleneck for clients without powerful GPUs [37]. Hermes [49] generates a reference frame with low entropy, but its extremely slow encoder performance (< 1 FPS) [43] limits its applicability only to on-demand volumetric video streaming. Octree-based XOR operation for differential encoding [29] is designed for real-time processing but is limited to nearly static scenes as it considers only geometry and ignores colors [41, 49]. Therefore, a new point cloud compression method that can efficiently compute temporal redundancy with minimal processing for both encoding and decoding is highly needed.

3D residual processing comlexity. A naive method to use 3D motion vectors is to simply follow the same approach used for 2D motion vectors. That is, applying 3D motion vectors for motion compensation on the point cloud and updating the residual information. In the case of 2D videos, this process is straightforward because the data structure has fixed dimensions (e.g., addition of two 2D matrices with RGB channels of $width \times height$ dimensions). In contrast, a point cloud is an unstructured set of points, therefore a point matching is needed to identify which points require residual updates. This approach introduces significant computational overhead on the client side. The computation complexity is $O(n \cdot m)$, where n and m denote the number of points in the previous and current point cloud, whereas 2D residual update has much lower complexity of $O(1)$.

4 *DeltaStream* Design

4.1 *DeltaStream* Overview

Figure 3 illustrates the system overview of *DeltaStream*. A live media server is fed RGB-D video frames from multiple RGB-D cameras. Using preset camera parameters, the block-based Delta Encoder in the server calculates the temporal difference between frame $t-1$ and frame t, where each frame includes two types of 2D inputs, RGB and depth frames. Using block-based calculations, the encoder classifies each block as either a *matched block* or a *mismatched block*. For *matched blocks*, the Delta Encoder computes 3D motion vectors, while for *mismatched blocks*, it generates delta point cloud. Then, the 3D motion vectors and delta point cloud are encoded and transmitted to the client along with their respective block coordinates.

On the client side, the block-based Delta Decoder utilizes preset camera parameters to decode the point cloud frame t based on the frame $t-1$. First, points corresponding to *matched blocks* are identified in point cloud frame $t-1$, and motion compensation is performed on those points. Next, points corresponding to *mismatched blocks* are identified and updated using the delta point cloud. Finally, the resulting delta-decoded point cloud frame t is transformed into the world coordinate system before being rendered on the client display, such as an HMD. Since the system assumes stationary cameras, the coordinate transformation from each camera to the world coordinate can be performed straightforwardly using methods like checkerboard calibration [26].

The goal of *DeltaStream* system design is to address the following practical challenges:

- *DeltaStream* overcomes the main challenge of volumetric video streaming, extremely high bandwidth usage between the server and client, by significantly reducing the bandwidth through the inference of temporal redundancy between point cloud frames in 2D space.
- Achieving high compression efficiency and real-time inter-frame compression processing at the same time is challenging due to the unstructured data representation of point cloud frames. *DeltaStream* addresses this challenge with a computationally efficient real-time delta encoding and decoding system, which backtracks from 3D to 2D data to establish a mapping between 2D and 3D blocks.
- *DeltaStream* enhances compression efficiency but maintains visual quality through a block-based approach that effectively filters information corresponding to frame-to-frame differences.
- *DeltaStream* adjusts the computational burden on the client based on the latency feedback from the client observations to adaptively support a variety of user devices with different computational abilities.

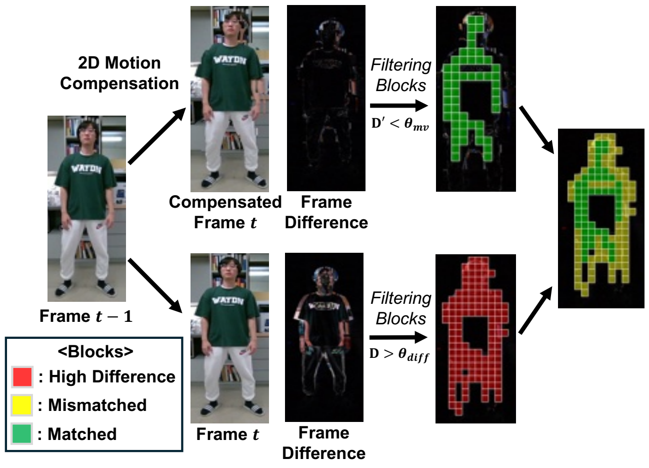


Figure 4: Process of classifying 2D blocks into *matched blocks* and *mismatched blocks* based on two predefined thresholds θ_{diff} and θ_{mv} in Delta Encoder.

4.2 Block-based Delta Encoder

Calculating and updating motion vectors for every pixel in a 2D video frame is highly inefficient. Consequently, existing 2D CODECs process video frames on a 2D block-by-block[2] basis, with these matrix computations optimized through hardware acceleration [11]. To leverage this efficiency, the Delta Encoder employs 2D block-based encoding to calculate 2D motion vectors, 2D RGB frame differences, and depth information at the block level. Based on this, the Delta Encoder classifies *high difference blocks* into *matched* and *mismatched blocks*, with each block type being encoded differently for computational efficiency in compressing temporal redundancy.

***Matched* and *mismatched* blocks.** Figure 4 describes how Delta Encoder processes the blocks through several steps and finally classifies them into *matched* and *mismatched* blocks using two thresholds, θ_{diff} and θ_{mv}[3].

i) High difference blocks are those requiring updates due to the large temporal differences observed between two consecutive frames. To compute these blocks, Delta Encoder first assigns 16×16 block indices for all blocks in the 2D frame as set $B = \{b_0, b_1, \ldots, b_n\}$, where n is the total number of blocks determined by the width and height of the 2D frame and the block size. For each block i in frame t, Delta Encoder calculates the temporal difference between frame t and frame $t-1$ as $D_i = \sum_{(x,y)\in b_i} \|I_t(x,y) - I_{t-1}(x,y)\|_1$, where (x,y) represents the pixel coordinates within the block and $I(x,y)$ is the RGB vector at those coordinates. Based on the temporal difference D_i, high difference blocks are filtered with a predefined threshold θ_{diff} into set $B_{high} = \{b_i \in B \mid D_i > \theta_{diff}\}$. Among *high difference blocks*, *matched* and *mismatched* blocks are identified using a threshold θ_{mv} after motion compensation.

ii) Matched blocks refer to blocks that can be efficiently encoded through 3D motion vectors. To identify these blocks, Delta Encoder calculates the temporal difference D'_i between RGB frame t and 2D motion compensated RGB frame from $t-1$ in the same way as D_i is computed. Using a pre-defined threshold θ_{mv}, the *matched blocks* are determined as $B_{match} = \{b_i \in B \mid D'_i < \theta_{mv}\}$. The same approach could be used to leverage the depth information as well. However, based on our observations, it is more efficient to use motion vectors from the RGB frame alone for the following reasons. First, the depth information from RGB-D cameras is often noisy, making it difficult to rely on the temporal difference in z-axis data between consecutive frames. Second, since the Delta Encoder employs 16×16 blocks, the z-values within each block tend to be consistent, reducing the need for additional depth-based calculation.

iii) Mismatched blocks are blocks that fail to reduce the block difference after 2D motion compensation among the high difference blocks, B_{high}, and denoted as $B_{mismatch} = B_{high} - B_{match}$. Since these blocks cannot be efficiently encoded using motion vectors despite large temporal differences, Delta Encoder processes the temporal difference D_i as a *delta point cloud*.

3D motion vector. *DeltaStream* introduces a method that transforms 2D motion vectors combined with depth information into 3D motion vectors for *matched blocks* as described in Figure 5. Suppose the 2D frame and 2D motion vectors have (u,v)-coordinates, while the 3D counterparts have (x,y,z)-coordinates. To transform 2D motion vectors, $\Delta(u,v)$, into 3D motion vectors, $\Delta(x,y,z)$, the Delta Encoder first extracts 2D motion vectors from each block in B_{match} by using a 2D video CODEC. Based on the 2D motion vector, it identifies how much a 16×16 source block in the previous frame needs to be shifted along the (u,v) axes to estimate the motion and reduce the difference with the destination block in the current frame. Subsequently, the points in the 2D source and destination blocks are transformed into $(x_{\text{src}}, y_{\text{src}}, z_{\text{src}})$ and $(x_{\text{dst}}, y_{\text{dst}}, z_{\text{dst}})$, respectively by following Eq. (1). Finally, the differences between these transformed points are used as the 3D motion vector, $\Delta(x,y,z)$. Then, the live media server sends the coordinate of the source block corresponding to *matched block* and calculated 3D motion vectors (i.e., pairs of (u_{src}, v_{src}) and $\Delta(x,y,z)$ are transmitted).

Delta point cloud. Using 3D motion vectors can improve compression efficiency. But, it also imposes a significant computational

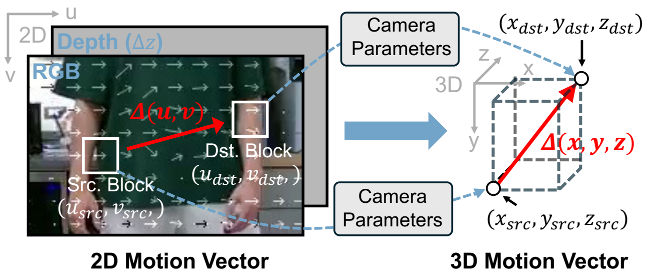


Figure 5: Transformation of 3D motion vectors from 2D motion vectors along with source/destination blocks and camera parameters.

[2]A 2D block refers to the process of dividing a $width \times height$ 2D frame into smaller fixed-size regions of pixels (i.e., 16x16). Similarly, a 3D block usually refers to a cube with dimensions of $l \times l \times l$, used to partition a point cloud in 3D space into smaller subspaces.

[3]In our system, we empirically set $\theta_{diff} = 10000$ and $\theta_{mv} = 5000$. The decision on these threshold values will be explained in Section 6.7.

burden on the client during decoding. Therefore, the Delta Encoder compresses the temporal difference using point cloud only for blocks that cannot be efficiently encoded with 3D motion vectors (i.e., *mismatched blocks*). At first, Delta Encoder transforms the pixels in *mismatched blocks* in frame t to point cloud which we call *delta point cloud* and encodes it using Draco [5]. Delta Encoder uses Draco when compressing delta point clouds, as Draco exploits spatial redundancy within a point cloud frame. When encoding *mismatched blocks*, Delta Encoder does not encode the frame difference but instead encodes the point cloud in the frame t directly. This approach is adopted to enhance decoding efficiency by leveraging vector operations, which will be addressed in detail in Section 4.3. Then, the server transmits the encoded delta point cloud with the corresponding block indices (i.e., 2D-coordinates of $B_{mismatch}$ and Draco-encoded delta point cloud are transmitted). As a result, Delta Encoder leverages both temporal and spatial correlations for more efficient encoding.

Keyframe. A keyframe refers to a frame that transmits the entire point cloud of the object. It is similar to an I-frame in 2D video CODECs. The keyframe is compressed using Draco [5] point cloud compression without delta encoding, same as intra-coding for 2D CODECs. In addition to serving as a reference frame, the keyframe also helps alleviate cumulative errors that may arise from continuous delta encoding. In *DeltaStream*, we set the keyframe rate as every 5 frames.

4.3 Block-based Delta Decoder

After receiving the encoded data from the server, the Delta Decoder performs the decoding process sequentially as shown in Figure 6.

Block matching. To decode block-based delta encoding on the client, a computationally efficient method is required to link the 3D positions of the point cloud with the 2D-based information from the delta encoding. Therefore, we propose an inverse transformation technique that traces 3D point positions back to 2D blocks and refer to this process as block matching, which is a different concept from the classification of *matched* and *mismatched blocks* based on the threshold defined in Section 4.2. This approach alleviates the high computational cost in the 3D space and is performed through the following process. From the inverse process of Eq. (1), the transformation from point cloud 3D (x, y, z) positions into (u, v) 2D frame domain (i.e, pixel location) is derived as Eq. (2). The resulting (u, v) coordinates are rounded to integers to align with pixel indices. The transformation is computationally efficient because the operation can be performed on entire vectors with (X, Y, Z) at once, without the need to repeatedly compute for each individual point. As a result, 2D image coordinates (u, v) are mapped to either the source block location for 3D motion compensation or the 2D block indices for delta point cloud update.

$$u = \frac{x}{z} \cdot f_x + c_x,\ v = \frac{y}{z} \cdot f_y + c_y \tag{2}$$

3D motion compensation. Block-based Delta Decoder on the client, it first processes 3D motion compensation for *matched* blocks in B_{match}, followed by the delta point cloud update. From the server, the client receives 3D motion vectors and corresponding source block locations. With this information, the client identifies the points in the point cloud in the source block through an inverse

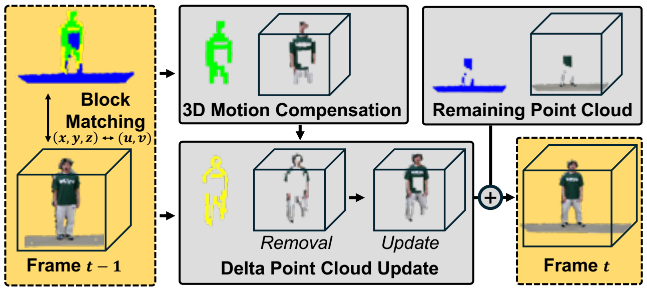


Figure 6: Decoding process of Delta Decoder, performed sequentially: (1) block matching, (2) 3D motion compensation, and (3) delta point cloud update.

transformation process. Specifically, given (u_{src}, v_{src}) as the top-left coordinate of the 2D source block, it filters the points for $u_{src} \le u < u_{src} + 16$ and $v_{src} \le v < v_{src} + 16$. Finally, the client shifts the points in the source block $(x_{src}, y_{src}, z_{src})$ by adding the corresponding 3D motion vector $\Delta(x, y, z)$ to each point.

Delta point cloud update. After 3D motion compensation process, the client decodes and update delta point cloud along with corresponding block indices of $B_{mismatch}$. The only difference is that the delta point cloud is paired with the block index, while the 3D motion vectors are paired with the source block's location (u_{src}, v_{src}). The received *mismatched* blocks in $B_{mismatch}$ are the blocks which should be removed in the previous point cloud and replaced with the delta point cloud. To remove the blocks, block indices are computed with $b_i = \lfloor \frac{v}{16} \rfloor \cdot \lfloor \frac{w}{16} \rfloor + \lfloor \frac{u}{16} \rfloor$, where w denotes the 2D frame width.

Vector-wise computation. Figure 7 shows the log-scale difference between point-based computation and vector-wise computation, assuming that each of the two point clouds has 100K points. Block-wise or vector-wise computations are very efficiently processed compared to point-based computations by two reasons: 1) Point-based computation needs to match the point in two point clouds for the further process, therefore has $O(n^2)$ complexity, and 2) the block-wise computation can be processed using vector-wise operation that can be additionally accelerated by parallel processing.

4.4 Computation-Adaptive Online Control

DeltaStream addresses temporal redundancy through delta encoding using 3D motion vectors and a delta point cloud. However, this approach is not without cost. While it effectively reduces bandwidth, it affects the computational burden on the client side during decoding. Therefore, control is necessary on the server considering the level of decoding complexity to meet the real-time decoding and rendering on the client. We show that the control could be adaptive according to the client's computational power.

We first conduct a correlation analysis on latency and the factors affecting latency. We use *longdress, soldier, loot,* and *redandblack* sequences from 8i dataset [21] and point clouds generated from RGB-D frames captured with Intel RealSense cameras [6]. Figure 8 (a) shows a linear correlation between the number of points in point cloud and Draco [5] decoding latency, which is consistent with findings from previous studies [20]. We also observe a linear

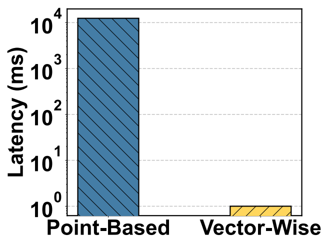
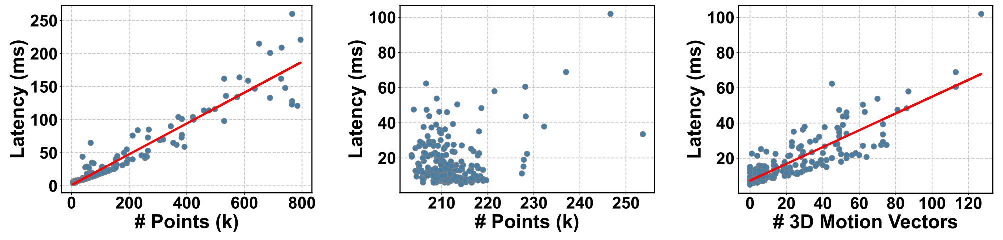


Figure 7: Computation complexity of point-based approach and vector-wise operation.



(a) # of points and point cloud decoding latency. **(b) # of points and 3D motion compensation latency.** **(c) # of 3D motion vectors and its processing latency.**

Figure 8: Correlation among latency, the number of points, and 3D motion vectors. Tested on Intel Core i5-1135G7 @ 2.40GHz.

correlation between the number of 3D motion vectors and update latency, as shown in Figure 8 (c). However, it is hard to find the relationship between the number of points and 3D motion vector update processing latency, as shown in Figure 8 (b). By leveraging the two clear correlations, the server adjusts the the number of 3D motion vectors used in delta encoding to support the client's real-time processing.

The primary control knob is the limitation on the number of maximum 3D motion vectors that can be used in delta encoding. Intuitively, increasing the number of 3D motion vectors reduces the number of points transmitted in the delta point cloud, but proportionally increases the computation burden on the client side at the same time. As shown in Figure 8 (a) and 8 (c), the slope of the linear regression function for the relationship between the number of 3D motion vectors and the update latency (≈ 0.5) is much steeper than that of the function relating the number of points to decoding latency (≈ 0.00025). Each 3D motion vector processes a 16×16 block in delta encoding, meaning that using k motion vectors can approximately reduce the number of points by $16 \times 16 \times k$ from the delta point cloud. Since changes in the number of 3D motion vectors have a dominant impact on latency (i.e., latency change according to one 3D motion vector change is $0.5 \gg 0.00025 \times 16 \times 16$), the number of 3D motion vectors is controlled to target FPS (i.e., 30) rendering on the client side. The client periodically feedback its rendering FPS to the server. If the rendering FPS is below target FPS (e.g. 28 FPS), the server update the number of 3D motion vectors $N_{mv} \leftarrow \gamma * N_{mv}$, where $\gamma = 0.8$ is a decaying parameter. The decaying parameter γ was empirically set as 0.8 in *DeltaStream*. Note that the decoding complexity can be adjusted according to the client's computational capability during the initial phase of streaming. In particular, a lower γ value results in more coarse-grained control, as motion vectors are adjusted more aggressively.

5 Implementation

Hardware. Figure 9 shows our testbed setup for live volumetric video streaming. In the experiments, our system uses four depth cameras capturing an object in the center from front, back, right, and left. The camera models include two Intel RealSense D435i, Intel RealSense D455, and Intel RealSense D415 [6]. These cameras are connected to the server through wired connections and capture RGB and depth frames at 30 FPS to generate point clouds. The

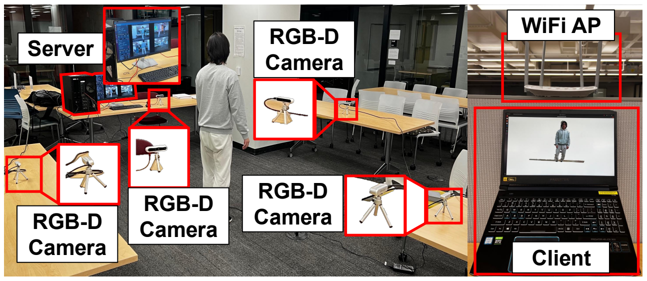
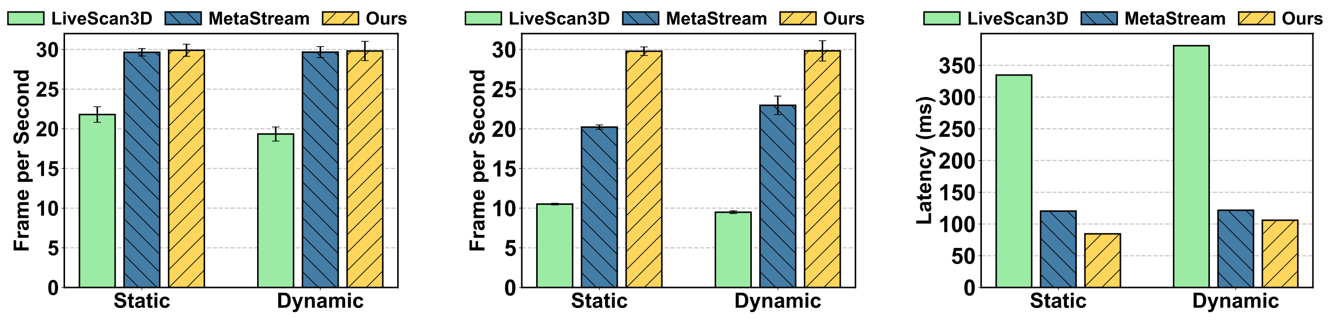


Figure 9: The testbed setup for live volumetric video streaming consists of 4 Intel RealSense RGB-D cameras connected to the server machine, with the server and client communicating via WiFi.

processing server is equipped with an Intel Core i9-14900K CPU and 64GB of memory. We implemented the remote client on a laptop equipped with Intel Core i7-9750H @ 2.60GHz CPU, which has slightly lower computational capabilities compared to Apple Vision Pro M2 chip [1].

Software. The server operates on a docker container using the Ubuntu base image and the client runs on Ubuntu Desktop. RGB and depth frames are captured through the Intel RealSense software development kit (SDK) [7], and the captured data is processed into point clouds using the Open3D library. The Open3D library is also employed for visualizing and manipulating point clouds. Delta point cloud compression is performed with Draco [5]. OpenCV and Open3D are used for handling 2D and 3D object manipulations, such as calculating difference between 2D frames. Eigen [3] is used to perform matrix operations and transformations on point cloud data. This includes computing block indices and determining points to remove based on motion vectors. To extract motion vectors, FFmpeg [4] is used to create a MPEG-4 coded video of the current and the previous 2D frame. Then, the *av_frame_get_side_data* function in the libavutil library [8] is used to extract motion vectors from the MPEG-4 video. For asynchronous network communication between the server and the client, the Boost Asio library is used whilst serialization is done through the cereal library [2]. The implementation employs multi-threading to concurrently process heavy tasks such as Draco encoding/decoding and motion vector extraction. Finally, Open3D is used to render the transmitted point cloud on the client.



(a) Frame per second without bandwidth limit.

(b) Frame per second with 100 Mbps bandwidth.

(c) End-to-end latency.

Figure 10: Comparisons of end-to-end performance results with two baselines (LiveScan3D [31] and MetaStream [24]) and our *DeltaStream* for static and dynamic scene contents targeting 30 FPS.

The implementation comprises approximately 6k lines of code written in C++, including modules for camera calibration, depth data processing, 3D visualization, and communication protocols. The source code of *DeltaStream* is available online[4].

6 Performance Evaluation

6.1 Evaluation Setup

Experiment environment. For performance evaluation, we set up a network connecting the server and client through a commercial WiFi (802.11ac, 5 GHz) and used Linux *tc* [9] for bandwidth-constrained experiments on the server. Four Intel RealSense cameras were assumed to have fixed positions during the experiments and preset camera parameters were shared among the server and client prior to streaming.

Evaulation metrics. The evaluation metrics include frames per second (FPS), latency (ms), structural similarity index measure (SSIM) for visual quality assessment, and network bandwidth usage.

Video contents. We evaluated each metric in two live-captured scenes, static and dynamic, rather than using the 8i dataset [21], which is designed for on-demand volumetric video streaming. There is no standard open-source dataset captured with multiple (e.g., four) depth cameras that also provides fully disclosed intrinsic and extrinsic camera parameters. To address this, we created our own dataset using four Intel RealSense depth cameras [6], enabling synchronized live capture under realistic streaming conditions. The static scene resembles a telepresence scenario where a person is seated on a chair, making small movements with their arms or facial expressions. In the dynamic scene, a person moves freely, including actions such as sitting down, standing up, and moving body parts. The static scene consists of around 400K points and the dynamic scene consists of around 500K points.

Baselines. We compare the system performance of *DeltaStream* with that of recent studies. 1) *LiveScan3D* [31] is a state-of-the-art system prior to MetaStream [24], which presents live volumetric video capturing using multiple RGB-D cameras. For a fair comparison, we extended its functionality by compressing point clouds with Draco [5] before streaming. 2) *MetaStream* [24], a state-of-the-art solution, is a bandwidth-efficient live point cloud video streaming system leveraging image segmentation processing on smart cameras. The performance of our baseline implementations closely matches the results reported in their papers. Note that we do not evaluate the bandwidth between the cameras and the server, as it is assumed to require only a few Mbps.

Other live volumetric video systems, such as Holoportation [42], and Project Starline [32], are excluded because they are limited to scenarios where the server and the client are connected via Gbps scale high bandwidth in wired connections and equipped with multiple powerful GPUs. FarfetchFusion [33] is excluded because it primarily focuses on face reconstruction. MagicStream [17] is not considered as a baseline since it does not utilize point cloud-based streaming and relies heavily on multiple deep learning models for components such as data reconstruction and neural rendering, but none of the model parameters or codes are open-sourced.

6.2 End-to-end Performance of *DeltaStream*

This section compares the overall performance in live volumetric video streaming for each system, including frames per second (FPS), end-to-end latency (ms), and visual quality (SSIM).

Frames per second (FPS). In Figure 10 (a), both *DeltaStream* and MetaStream achieve a stable 30 FPS, except for LiveScan3D. *DeltaStream* and MetaStream achieve stable 30 FPS by streaming only the region of interest point cloud object through segmentation. However, LiveScan3D lacks such optimizations, leading to lower FPS as compared to the other two systems. When the bandwidth is limited to 100 Mbps in Figure 10 (b), MetaStream and LiveScan3D drop to 20–22 and 9-10 FPS. As both systems are inefficient at addressing temporal redundancy, the 100 Mbps bandwidth becomes a bottleneck in the streaming pipeline. However, *DeltaStream* maintains at 30 FPS even in the bandwidth-constraint case.

End-to-end latency. In a live video streaming system, end-to-end latency is crucial to ensure the motion-to-photon latency between the server and the client and interaction delay between users [14]. To evaluate this, we measure the end-to-end latency that consists of encoding latency on the server, transmission delay, decoding, and rendering latency on the client. In Figure 10 (c), *DeltaStream*

[4]https://github.com/delta-stream/DeltaStream

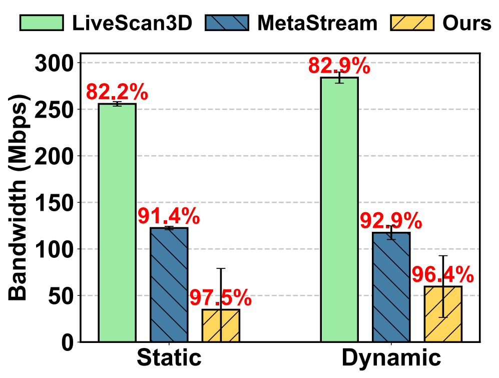


Figure 11: Average bandwidth usage of LiveScan3D [31], MetaStream [24], and *DeltaStream* for static and dynamic scenes. The number above the bar represents the bandwidth saving (%) compared to raw data transmission for each system.

	MetaStream	DeltaStream
Static	0.9050	0.8993
Dynamic	0.9080	0.9031

Table 1: Average SSIM values for MetaStream and *DeltaStream* on static and dynamic scenes.

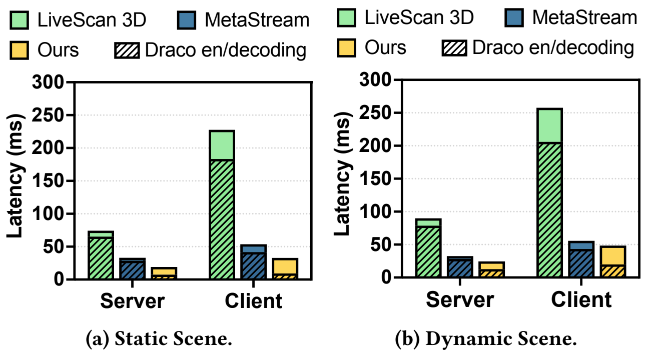


Figure 12: Latency breakdown of LiveScan3D [31], MetaStream [24] and *DeltaStream* for static and dynamic scene. Hatched region shows the latency spent by Draco encoding/decoding.

demonstrates lower latency compared to the other two baseline systems. As *DeltaStream* efficiently addresses temporal redundancy, it reduces the number of transmitted points and decreases the processing time taken by Draco [5] encoding and decoding. On the other hand, other baselines have to process more points in point clouds, resulting in higher latencies. A more detailed analysis on processing latency and overhead will be provided in Section 6.4.

Visual quality. We evaluate SSIM values on rendered scenes, similar to other systems [17, 24]. SSIM is a perceptual metric that quantifies quality degradation caused by data compression and processing. Note that *DeltaStream* transmits encoded data independently of viewport changes, ensuring that both user experience and visual quality remain unaffected even during extreme viewport movements. Compared to the state-of-the-art MetaStream, which has SSIM values of 0.9050 and 0.9080, *DeltaStream* achieves comparable SSIM values with 0.8993 and 0.9031 for static and dynamic scenes respectively, while performing delta encoding. As the rendered scene differs from the other two systems because of the region of interest, the SSIM metric is not directly compared with LiveScan3D.

6.3 Bandwidth Savings

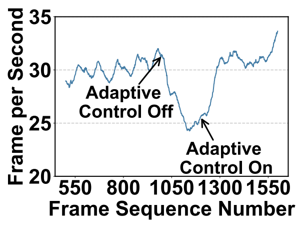
Since *DeltaStream* utilizes temporal redundancy to encode consecutive scenes efficiently, we compare the compression performance of each system by measuring the average bandwidth usage during streaming. For static and dynamic scene contents, Figure 11 compares average bandwidth usage. As a result of the high compression efficiency of Draco [5] point cloud encoding, all three systems achieve over 80% bandwidth savings compared to streaming raw data. The results show that *DeltaStream* effectively reduces the network bandwidth usage through delta encoding. On average, *DeltaStream* reduces bandwidth usage by 85% compared to LiveScan3D and by 71% compared to MetaStream for static scenes. For dynamic scenes, it achieves bandwidth reductions of 79% and 49% compared to LiveScan3D and MetaStream, respectively.

Static scene. The overall bandwidth throughput is lower than that of the dynamic scene because the static scene involves a person sitting with limited movements compared to the dynamic scene, therefore it allows greater utilization of temporal correlation from the previous frame. In extreme cases, such as when there is almost no difference between the previous and current frames, almost no data needs to be transmitted in *DeltaStream* as it efficiently handles temporal redundancy. Since MetaStream does not process temporal redundancy, *DeltaStream* saves 71% of bandwidth usage.

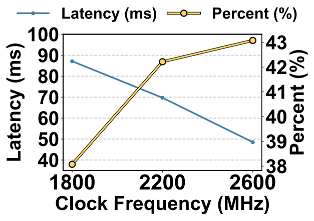
Dynamic scene. Similar to the static scene, *DeltaStream* effectively reduces bandwidth usage compared to the other two baselines. On average, *DeltaStream* reduces bandwidth by 79% compared to LiveScan3D and by 49% compared to MetaStream. The bandwidth variation over time in Figure 11 is greater compared to the static scene. In the dynamic scene, the object movement is more active, which contributes to the temporal redundancy reduction. Therefore, the bandwidth reduction rate relative to baselines is lower than in the static scene. However, *DeltaStream* still reduces a substantial portion of the bandwidth compared to the baselines.

6.4 Processing Latency and Overhead Analysis

Figure 12 illustrates the latency components for each system. On the server and client sides, the latency is divided into Draco processing time and the remaining processing time, and the time spent by Draco is hatched on the graph. In all cases, *DeltaStream* demonstrates the lowest latency on both the server and client sides. This is because the efficient processing of temporal redundancy reduces unnecessary point transmissions between point cloud frames, thereby decreasing the time required for Draco encoding and decoding,

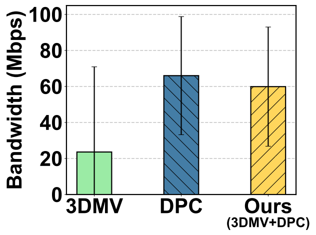


(a) FPS variation over time.

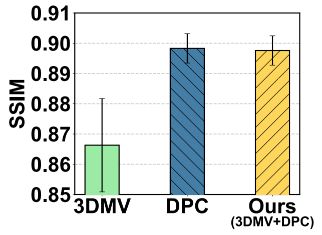


(b) Client processing latency.

Figure 13: Effectiveness of computation-adaptive online control in *DeltaStream*. (a) shows client rendering FPS variation over time on 1800 MHz clock frequency. (b) shows the overall client processing latency and the percentage of time spent only on processing 3D motion vectors.



(a) Average bandwidth usage.



(b) Average SSIM values.

Figure 14: Ablation study results for each method.

which contributes the largest portion of the overall processing latency. As shown in the relationship in Figure 8 (a), the number of points and the time taken for Draco processing are linearly related. Therefore, reducing the number of points compressed by Draco directly results in a reduced latency compared to other systems.

While reducing the temporal redundancy in the point cloud decreases the latency of Draco encoding on the server, additional processing overheads, such as extracting 2D motion vectors from the 2D video CODEC and generating 3D motion vectors, are introduced. Consequently, the latency for the additional processes in *DeltaStream* is higher in comparison to other baselines. However, the overall server-side latency remains lower than that of other two systems.

Similarly, on the client side, the latency for Draco decoding is reduced for the same reason, but the additional processing required for dealing with 3D motion vector updates results in higher latency compared to MetaStream. On the other hand, the higher client-side latency of LiveScan3D is caused by the lack of segmentation optimization for objects. Therefore, there are redundant points outside the object compared to *DeltaStream* and MetaStream. As a result, LiveScan3D incurs greater processing delays in point cloud rendering and pre-render processing steps compared to both MetaStream and *DeltaStream*.

6.5 Client Computation Adaptability

To account for the varying computational capabilities of client devices, such as device differences or runtime fluctuations in computational resources due to thermal throttling, *DeltaStream* introduces client computation-adaptive control to ensure reliable decoding speed as discussed in Section 4.4. Therefore, *DeltaStream* adjusts the number of 3D motion vectors N_{mv} during delta encoding to maintain the client's 30 FPS processing.

In Figure 13 (a), we temporarily removed the limitation on the number of 3D motion vectors (N_{mv}) during the middle of streaming (i.e., at frame sequence 1000) to artificially cause additional computational burden on the client. This causes the FPS to drop to around 24 FPS because decoding N_{mv} 3D motion vectors exceeds the real-time computational ability of the client. When the adaptive control was reactivated at frame sequence 1200, the server adjusted N_{mv} to a level manageable delta encoding. After a few frame sequences, the FPS recovered and stabilized, maintaining a rendering rate of 30 FPS.

We also conducted experiments under three different clock frequencies of the client CPU. Two clock frequencies were lower than the maximum clock frequency (2.6 GHz) of the client CPU, while one utilized the maximum clock frequency. Figure 13 (b) illustrates decoding latency and the proportion of motion compensation time within the total decoding time for each clock frequency on the client. As the clock frequency decreases, the server progressively reduces the number of 3D motion vectors to alleviate the client's computational burden. As a result, Figure 13 (b) shows that the proportion of 3D motion vector processing time relative to the overall latency decreases as the clock frequency decreases, which supports the proper operation of the adaptive control.

6.6 Ablation Study

We compare *DeltaStream* with the following two encoding methods to analyze the impact of each component of the Delta Encoder on compression efficiency and visual quality: (1) *3DMV* and (2) *DPC*. These two methods are not independent baselines. Instead, they represent integral components of *DeltaStream*. Our system (*3DMV+DPC*) balances the tradeoff between *3DMV* and *DPC* to minimize bandwidth and computation while preserving visual quality. The difference between *DeltaStream* and those methods is in the way of processing high difference blocks B_{high}. While *3DMV* utilizes 3D motion vectors for all B_{high} regardless of the accuracy of motion compensation (i.e., $\theta_{mv} = \infty$), *DPC* encodes all B_{high} as delta point cloud (i.e., $\theta_{mv} = 0$).

Figure 14 shows the evaluation results of *DeltaStream* and other two methods. The *3DMV* method is the most bandwidth-efficient among the three methods. As the more 3D motion vectors are used in *3DMV*, the fewer points need to be transmitted, therefore reducing the bandwidth usage. However, *3DMV* has the lowest visual quality, as it includes inaccurate motion vectors during encoding. On the other hand, the *DPC* replaces all B_{high} with a delta point cloud, resulting in the highest visual quality but also the highest bandwidth usage. In summary, the optimal operational point for maximizing bandwidth savings is achieved when *DeltaStream* fully utilizes 3D motion vectors. By leveraging the respective advantages of both *3DMV* and *DPC* methods, *DeltaStream* (*3DMV+DPC*) effectively reduces bandwidth usage while maintaining visual quality.

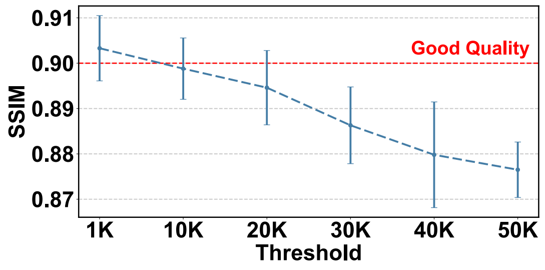


Figure 15: SSIM value changes according to θ_{diff}.

6.7 Complementary Analysis

3D motion vectors. We use the Chamfer Distance [51] to verify the accuracy of the calculated 3D motion vectors. The metric measures the average closest-point distance between the 3D motion-compensated source point cloud and the destination point cloud to verify the geometric alignment of point clouds. The Chamfer Distance between the ground truth (i.e., point cloud frames without encoding) and 3D motion compensated point cloud is 0.0043 on average, which is less than 5 mm distance, thereby demonstrating the high accuracy of the 3D motion vectors. Note that the slight misalignment arises from the block-based approach of *DeltaStream*, a common issue also observed in 2D video CODECs.

Predefined thresholds. The values of the thresholds primarily used in the delta encoding in *DeltaStream* are empirically determined through experiments. We first set a threshold θ_{diff}, which is responsible for filtering high difference blocks B_{high} between the previous frame and the current frame, with a focus on visual quality. As shown in Figure 15, visual quality degrades when θ_{diff} increases, because blocks with significant differences are excluded from delta encoding and remain unchanged from the previous frame, even when they have considerable differences. θ_{diff} targets to maintain SSIM above 0.9, which is considered to represent a good visual quality [19]. Therefore, we set $\theta_{diff} = 10000$, the closest value makes an SSIM of 0.9. On the other hand, we tested various values of θ_{mv} after fixing θ_{diff}, but observed mere changes in SSIM values. Therefore, we arbitrarily set $\theta_{mv} = 5000$, a tighter threshold value compared to θ_{diff}, to guarantee using only $B_{matched}$ blocks with minimal differences after motion compensation.

7 Discussion

Compression loss. While *DeltaStream* achieves significant bandwidth savings through delta encoding, the lossy nature of compression introduces visual artifacts that can impact the quality of the reconstructed point cloud. *DeltaStream* utilizes 3D motion compensation to reduce the differences between consecutive frames, thereby minimizing the data that needs to be transmitted. However, inaccuracies in 3D motion vectors may result in slight misalignments. Compression loss also arises from the threshold used to classify *matched* and *mismatched* blocks. Smaller thresholds improve quality but increase bandwidth, while larger thresholds reduce bandwidth at the cost of quality. Adaptive thresholds and real-time perceptual metrics could further balance bandwidth efficiency and visual fidelity.

Potential performance enhancements. While we implemented *DeltaStream* with a focus on CPU processing, there is a room for performance improvement through hardware acceleration like as 2D video codecs [11]. Since our approach is block-based, it is well-suited for parallel processing on GPUs, which could further enhance the performance. *DeltaStream* efficiently addresses temporal redundancy in the current point cloud frame based on the previous frame. However, utilizing concepts such as bi-directional prediction frames or variable block sizes [44, 50] could lead to potential performance improvements in terms of bandwidth efficiency. As these encoding approaches increase decoding complexity, this must be carefully considered.

User experience and system assumptions. Since *DeltaStream* deals with 3D point cloud transmission, user experience remains unaffected even with extreme viewport movements. Viewport-adaptive solutions such as ViVo [25] can be incorporated into *DeltaStream* to further reduce the network bandwidth. Many practical immersive applications such as remote surgery, virtual concerts, and telepresence typically assume fixed camera positions. Even when the camera position changes, these updates can be transmitted in real-time using only a few bytes.

8 Related Work

Reducing temporal and spatial redundancy among adjacent frames in video streaming for AR/VR/MR applications is a crucial issue in terms of saving network bandwidth. For 360° video streaming applications, recent studies [36, 57] have explored reusing information from previous frames to enhance compression and computational efficiency. However, volumetric video streaming presents significantly greater challenges due to its inherent support for 6 degrees of freedom (6-DoF), where users can move freely in space rather than only rotating their heads. In the following, we review existing systems for volumetric video streaming, as summarized in Table 2.

Live volumetric video streaming. The most representative application of live volumetric video streaming is telepresence, which enables real-time, immersive communication between users in different locations. Prominent examples include Holoportation [42] and Project Starline [32], both of which are high-quality, real-time telepresence systems. However, these systems require extremely high bandwidth on the Gbps scale and rely on multiple powerful GPUs for processing. Additionally, Project Starline [32] is restricted to use inside a booth, limiting user mobility and flexibility. FarfetchFusion [33] introduces a mobile telepresence platform with a focus on 3D face reconstruction, allowing for greater user mobility. MagicStream [17] transmits only semantic information instead of full 3D representations. While this approach significantly reduces bandwidth usage, it requires extensive user profiling to train multiple machine learning models and is not a generalizable solution for other types of objects. LiveScan3D [31] suggests a live volumetric capturing pipeline using RGB-D cameras, offering a simpler, hardware-centric approach. MetaStream [24], regarded as the state-of-the-art system for live volumetric video streaming, supports multi-camera capture to real-time streaming. It reduces bandwidth by performing image segmentation and offloading computation to smart cameras, thereby improving processing efficiency.

Table 2: Comparison of volumetric video streaming systems. *DeltaStream* uniquely supports live 6-DoF point cloud video with both spatial and temporal encoding.

System	Video Application	DoF	Data Transmission	Spatial Encoding	Temporal Encoding
DeltaVR [36]	360°	3	2D	✓	✓
Deja View [57]	360°	3	2D	✓	✓
Dragonfly [22]	360°	3	2D	✓	✓
Vues [38]	On-demand volumetric	6	2D	✓	✓
ViVo [25]	On-demand volumetric	6	Point cloud	✓	✗
GROOT [34]	On-demand volumetric	6	Point cloud	✓	✗
YuZu [55]	On-demand volumetric	6	Point cloud	✓	✗
FarfetchFusion [33]	Live volumetric	6	2D	✓	✓
MetaStream [24]	Live volumetric	6	Point cloud	✓	✗
DeltaStream	Live volumetric	6	Point cloud	✓	✓

Despite the diversity of approaches, none of these methods effectively utilize the temporal correlation among consecutive point cloud frames, which presents an opportunity for further bandwidth and computational efficiency improvements.

On-demand volumetric video streaming. On-demand volumetric video streaming focuses on delivering pre-recorded content from the server. Unlike live volumetric video streaming, on-demand streaming does not require real-time encoding, allowing researchers to focus primarily on bandwidth-saving techniques. Various approaches have been proposed to achieve this goal, leveraging advanced compression, prediction, and data reduction methods. Vues [38] achieves bandwidth reduction by utilizing highly efficient video CODECs [28] and incorporates multiple machine learning models for viewport prediction on edge servers, thereby enhancing the quality of experience. GROOT [34] introduces an innovative Octree-based parallel decodable tree structure, enabling real-time decoding on mobile GPUs. ViVo [25] employs a visibility-aware sampling technique that considers viewport, distance, and occlusion factors to reduce bandwidth usage. YuZu [55] and VoluSR [54] utilize 3D super-resolution to lower the point density of transmitted point clouds, thereby significantly reducing bandwidth consumption. M5 [56] and MuV2 [39] are specifically designed to support multiple users under limited bandwidth conditions, ensuring stable streaming for multiple concurrent users. While many studies on on-demand volumetric video streaming have focused on improving compression rates, the computational overhead required for these techniques often makes them unsuitable for live streaming scenarios.

9 Conclusion

In this paper, we presented *DeltaStream*, a novel live volumetric video streaming system that leverages 2D information to efficiently encode and transmit point cloud data. Additionally, the computation-adaptive online control mechanism dynamically adjusts encoding strategies based on client-side computational constraints to ensure real-time streaming at 30 FPS. Comprehensive evaluations demonstrate that *DeltaStream* significantly reduces bandwidth usage while achieving low end-to-end latency and stable frame rates. Compared to state-of-the-art systems like MetaStream and LiveScan3D, *DeltaStream* reduces bandwidth usage by up to 71% for static scenes and 49% for dynamic scenes. Furthermore, it achieves 1.15-1.63× faster decoding speeds while maintaining comparable visual quality. We believe that *DeltaStream* provides a practical and effective solution for real-time volumetric video streaming in applications such as telepresence, virtual concerts, and augmented reality.

Acknowledgements

This work was supported by the National Science Foundation under Grant No. 1908910, by the Institute of Information & Communications Technology Planning & Evaluation(IITP)-ICT Creative Consilience Program grant funded by the Korea government(MSIT)(IITP-2025-RS-2020-II201819), and by the Institute of Information & Communications Technology Planning & Evaluation(IITP)-ITRC (Information Technology Research Center) grant funded by the Korea government (MSIT)(IITP-2025-RS-2024-00437866). This work was conducted while Mr. Hojeong Lee was visiting the University of Colorado Boulder. Sangtae Ha and Seyeon Kim are the corresponding authors of this work.

References

[1] Apple vision pro. https://www.apple.com/apple-vision-pro/.
[2] cereal docs - main. https://uscilab.github.io/cereal/.
[3] Eigen. https://eigen.tuxfamily.org/.
[4] Ffmpeg. https://www.ffmpeg.org/.
[5] Google draco. https://google.github.io/draco/.
[6] Intel realsense. https://www.intelrealsense.com/.
[7] Intel realsense sdk 2.0. https://www.intelrealsense.com/sdk-2/.
[8] Libavutil documentation. https://www.ffmpeg.org/libavutil.html.
[9] Linux tc man page. https://linux.die.net/man/8/tc.
[10] Meta quest. https://www.meta.com/quest.
[11] Nvidia video codec sdk. https://developer.nvidia.com/video-codec-sdk.
[12] Orbbec femto. https://www.orbbec.com/.
[13] Anique Akhtar, Zhu Li, and Geert Van der Auwera. Inter-frame compression for dynamic point cloud geometry coding. *IEEE Transactions on Image Processing*, 2024.
[14] Mario Baldi and Yoram Ofek. End-to-end delay analysis of videoconferencing over packet-switched networks. *IEEE/ACM Transactions On Networking*, 8(4):479–492, 2000.
[15] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.
[16] Ruopeng Chen, Mengbai Xiao, Dongxiao Yu, Guanghui Zhang, and Yao Liu. patchvvc: A real-time compression framework for streaming volumetric videos. In *Proceedings of the 14th Conference on ACM Multimedia Systems*, pages 119–129, 2023.

[17] Ruizhi Cheng, Nan Wu, Vu Le, Eugene Chai, Matteo Varvello, and Bo Han. Magicstream: Bandwidth-conserving immersive telepresence via semantic communication. In *Proceedings of the 22nd ACM Conference on Embedded Networked Sensor Systems*, pages 365–379, 2024.

[18] Paul J Choi, Rod J Oskouian, and R Shane Tubbs. Telesurgery: past, present, and future. *Cureus*, 10(5), 2018.

[19] Eduardo Cuervo, Alec Wolman, Landon P Cox, Kiron Lebeck, Ali Razeen, Stefan Saroiu, and Madanlal Musuvathi. Kahawai: High-quality mobile gaming using gpu offload. In *Proceedings of the 13th annual international conference on mobile systems, applications, and services*, pages 121–135, 2015.

[20] Matthias De Fré, Jeroen van der Hooft, Tim Wauters, and Filip De Turck. Scalable mdc-based volumetric video delivery for real-time one-to-many webrtc conferencing. In *Proceedings of the 15th ACM Multimedia Systems Conference*, pages 121–131, 2024.

[21] Eugene d'Eon, Bob Harrison, Taos Myers, and Philip A Chou. 8i voxelized full bodies-a voxelized point cloud dataset. *ISO/IEC JTC1/SC29 Joint WG11/WG1 (MPEG/JPEG) input document WG11M40059/WG1M74006*, 7(8):11, 2017.

[22] Ehab Ghabashneh, Chandan Bothra, Ramesh Govindan, Antonio Ortega, and Sanjay Rao. Dragonfly: Higher perceptual quality for continuous 360 video playback. In *Proceedings of the ACM SIGCOMM 2023 Conference*, pages 516–532, 2023.

[23] Danillo Graziosi, Ohji Nakagami, Satoru Kuma, Alexandre Zaghetto, Teruhiko Suzuki, and Ali Tabatabai. An overview of ongoing point cloud compression standardization activities: Video-based (v-pcc) and geometry-based (g-pcc). *APSIPA Transactions on Signal and Information Processing*, 9:e13, 2020.

[24] Yongjie Guan, Xueyu Hou, Nan Wu, Bo Han, and Tao Han. Metastream: Live volumetric content capture, creation, delivery, and rendering in real time. In *Proceedings of the 29th Annual International Conference on Mobile Computing and Networking*, pages 1–15, 2023.

[25] Bo Han, Yu Liu, and Feng Qian. Vivo: Visibility-aware mobile volumetric video streaming. In *Proceedings of the 26th annual international conference on mobile computing and networking*, pages 1–13, 2020.

[26] Lin Huang, Feipeng Da, and Shaoyan Gai. Research on multi-camera calibration and point cloud correction method based on three-dimensional calibration object. *Optics and Lasers in Engineering*, 115:32–41, 2019.

[27] Hanbyul Joo, Tomas Simon, Xulong Li, Hao Liu, Lei Tan, Lin Gui, Sean Banerjee, Timothy Scott Godisart, Bart Nabbe, Iain Matthews, Takeo Kanade, Shohei Nobuhara, and Yaser Sheikh. Panoptic studio: A massively multiview system for social interaction capture. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.

[28] Hari Kalva. The h. 264 video coding standard. *IEEE multimedia*, 13(4):86–90, 2006.

[29] Julius Kammerl, Nico Blodow, Radu Bogdan Rusu, Suat Gedikli, Michael Beetz, and Eckehard Steinbach. Real-time compression of point cloud streams. In *2012 IEEE international conference on robotics and automation*, pages 778–785. IEEE, 2012.

[30] Yura Kim and Yong-Hwan Kim. Real-time video-based point cloud encoding system on a distributed platform. In *2023 Fourteenth International Conference on Ubiquitous and Future Networks (ICUFN)*, pages 593–595. IEEE, 2023.

[31] Marek Kowalski, Jacek Naruniec, and Michal Daniluk. Livescan3d: A fast and inexpensive 3d data acquisition system for multiple kinect v2 sensors. In *2015 international conference on 3D vision*, pages 318–325. IEEE, 2015.

[32] Jason Lawrence, Ryan Overbeck, Todd Prives, Tommy Fortes, Nikki Roth, and Brett Newman. Project starline: A high-fidelity telepresence system. In *ACM SIGGRAPH 2024 Emerging Technologies*, pages 1–2. 2024.

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

[34] Kyungjin Lee, Juheon Yi, Youngki Lee, Sunghyun Choi, and Young Min Kim. Groot: a real-time streaming system of high-fidelity volumetric videos. In *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*, pages 1–14, 2020.

[35] Reoxiang Li, Bing Zeng, and Ming L Liou. A new three-step search algorithm for block motion estimation. *IEEE transactions on circuits and systems for video technology*, 4(4):438–442, 1994.

[36] Yong Li and Wei Gao. Deltavr: Achieving high-performance mobile vr dynamics through pixel reuse. In *Proceedings of the 18th International Conference on Information Processing in Sensor Networks*, pages 13–24, 2019.

[37] Zhicheng Liang, Junhua Liu, Mallesham Dasari, and Fangxin Wang. Fumos: Neural compression and progressive refinement for continuous point cloud video streaming. *IEEE Transactions on Visualization and Computer Graphics*, 2024.

[38] Yu Liu, Bo Han, Feng Qian, Arvind Narayanan, and Zhi-Li Zhang. Vues: Practical mobile volumetric video streaming through multiview transcoding. In *Proceedings of the 28th Annual International Conference on Mobile Computing And Networking*, pages 514–527, 2022.

[39] Yu Liu, Puqi Zhou, Zejun Zhang, Anlan Zhang, Bo Han, Zhenhua Li, and Feng Qian. Muv2: Scaling up multi-user mobile volumetric video streaming via content hybridization and sharing. In *Proceedings of the 30th Annual International Conference on Mobile Computing and Networking*, pages 327–341, 2024.

[40] Donald Meagher. Geometric modeling using octree encoding. *Computer graphics and image processing*, 19(2):129–147, 1982.

[41] Rufael Mekuria, Kees Blom, and Pablo Cesar. Design, implementation, and evaluation of a point cloud codec for tele-immersive video. *IEEE Transactions on Circuits and Systems for Video Technology*, 27(4):828–842, 2016.

[42] Sergio Orts-Escolano, Christoph Rhemann, Sean Fanello, Wayne Chang, Adarsh Kowdle, Yury Degtyarev, David Kim, Philip L Davidson, Sameh Khamis, Mingsong Dou, et al. Holoportation: Virtual 3d teleportation in real-time. In *Proceedings of the 29th annual symposium on user interface software and technology*, pages 741–754, 2016.

[43] Evaristo Ramalho, Eduardo Peixoto, and Edil Medeiros. Silhouette 4d with context selection: Lossless geometry compression of dynamic point clouds. *IEEE Signal Processing Letters*, 28:1660–1664, 2021.

[44] Iain E Richardson. *The H. 264 advanced video compression standard*. John Wiley & Sons, 2011.

[45] Radu Bogdan Rusu and Steve Cousins. 3D is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 9-13 2011. IEEE.

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

[47] Gary J Sullivan, Jens-Rainer Ohm, Woo-Jin Han, and Thomas Wiegand. Overview of the high efficiency video coding (hevc) standard. *IEEE Transactions on circuits and systems for video technology*, 22(12):1649–1668, 2012.

[48] Irene Viola and Pablo Cesar. Volumetric video streaming: Current approaches and implementations. *Immersive Video Technologies*, pages 425–443, 2023.

[49] Yizong Wang, Dong Zhao, Huanhuan Zhang, Chenghao Huang, Teng Gao, Zixuan Guo, Liming Pang, and Huadong Ma. Hermes: Leveraging implicit inter-frame correlation for bandwidth-efficient mobile volumetric video streaming. In *Proceedings of the 31st ACM International Conference on Multimedia*, pages 9185–9193, 2023.

[50] Thomas Wiegand, Gary J Sullivan, Gisle Bjontegaard, and Ajay Luthra. Overview of the h. 264/avc video coding standard. *IEEE Transactions on circuits and systems for video technology*, 13(7):560–576, 2003.

[51] Tong Wu, Liang Pan, Junzhe Zhang, Tai Wang, Ziwei Liu, and Dahua Lin. Density-aware chamfer distance as a comprehensive metric for point cloud completion. *arXiv preprint arXiv:2111.12702*, 2021.

[52] Mengyu Yang, Zhenxiao Luo, Miao Hu, Min Chen, and Di Wu. A comparative measurement study of point cloud-based volumetric video codecs. *IEEE Transactions on Broadcasting*, 69(3):715–726, 2023.

[53] Dongxiao Yu, Ruopeng Chen, Xin Li, Mengbai Xiao, Guanghui Zhang, and Yao Liu. A gpu-enabled real-time framework for compressing and rendering volumetric videos. *IEEE Transactions on Computers*, 2023.

[54] Anlan Zhang, Chendong Wang, Bo Han, and Feng Qian. Efficient volumetric video streaming through super resolution. In *Proceedings of the 22nd International Workshop on Mobile Computing Systems and Applications*, pages 106–111, 2021.

[55] Anlan Zhang, Chendong Wang, Bo Han, and Feng Qian. {YuZu}:{Neural-Enhanced} volumetric video streaming. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 137–154, 2022.

[56] Ding Zhang, Puqi Zhou, Bo Han, and Parth Pathak. M5: Facilitating multi-user volumetric content delivery with multi-lobe multicast over mmwave. In *Proceedings of the 20th ACM Conference on Embedded Networked Sensor Systems*, pages 31–46, 2022.

[57] Shulin Zhao, Haibo Zhang, Sandeepa Bhuyan, Cyan Subhra Mishra, Ziyu Ying, Mahmut T Kandemir, Anand Sivasubramaniam, and Chita R Das. Déja view: Spatio-temporal compute reuse for 'energy-efficient 360 vr video streaming. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pages 241–253. IEEE, 2020.

[58] Shan Zhu and Kai-Kuang Ma. A new diamond search algorithm for fast block-matching motion estimation. *IEEE transactions on Image Processing*, 9(2):287–290, 2000.