LiVo: Toward Bandwidth-adaptive Fully-Immersive Volumetric Video Conferencing

RAJRUP GHOSH, University of Southern California, USA
CHRISTINA SUYONG SHIN, University of Southern California, USA
LEI ZHANG, ByteDance, USA
MUYANG YE, University of Southern California, USA
TAO JIN, Carnegie Mellon University, USA
HARSHA V. MADHYASTHA, University of Southern California, USA
RAVI NETRAVALI, Princeton University, USA
ANTONIO ORTEGA, University of Southern California, USA
SANJAY RAO, Purdue University, USA
ANTHONY ROWE, Carnegie Mellon University, USA
RAMESH GOVINDAN, University of Southern California, USA

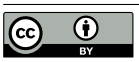
Abstract: Volumetric video allows users 6 degrees of freedom (6-DoF) in viewing continuously evolving scenes in 3D. Given broadband speeds today, volumetric video conferencing will soon be feasible. Even so, these scenes will need to be compressed, and compression will need to adapt to variations in bandwidth availability. Existing 3D compression techniques cannot adapt to bandwidth availability, are slow, and utilize bandwidth inefficiently, so they don't scale well to large scene descriptions. LiVo achieves low-latency and large-scene two-way conferencing by maximally leveraging existing 2D video infrastructure, including compression standards, rate-adaptive codecs, and real-time transport protocols. To achieve high quality, LiVo must carefully compose scenes from multiple cameras into multiple streams, encode scene geometry in a novel way, adapt to and apportion available bandwidth dynamically between streams to ensure high reconstruction quality, and cull content outside the receiver's field of view to reduce information sent into the network. These novel contributions enable LiVo to outperform the state-of-the-art by over 20% in objective quality. In a user study, LiVo achieves a mean opinion score of 4.1, while other approaches achieve significantly lower values.

CCS Concepts: • **Networks** → **Network protocol design**.

Additional Key Words and Phrases: Volumetric Video, Conferencing, Bandwidth Adaptation

ACM Reference Format:
Rajrup Ghosh, Christina Suyong Shin, Lei Zhang, Muyang Ye, Tao Jin, Harsha V. Madhyastha, Ravi Netravali, Antonio Ortega, Sanjay Rao, Anthony Rowe, and Ramesh Govindan. 2025. LiVo: Toward Bandwidth-adaptive Fully-Immersive Volumetric Video Conferencing. *Proc. ACM Netw.* 3, CoNEXT4, Article 34 (December 2025), 25 pages. https://doi.org/10.1145/3768981

This material is based upon work supported by the National Science Foundation under Grant No. 1956190.
Authors' Contact Information: Rajrup Ghosh, rajrupgh@usc.edu, University of Southern California, Los Angeles, USA; Christina Suyong Shin, cshin956@usc.edu, University of Southern California, Los Angeles, USA; Lei Zhang, geraldleizhang@gmail.com, ByteDance, Bellevue, USA; Muyang Ye, muyangye@usc.edu, University of Southern California, Los Angeles, USA; Tao Jin, taojin@andrew.cmu.edu, Carnegie Mellon University, Pittsburgh, USA; Harsha V. Madhyastha, madhyast@usc.edu, University of Southern California, Los Angeles, USA; Ravi Netravali, rnetravali@cs.princeton.edu, Princeton University, Princeton, USA; Antonio Ortega, aortega@usc.edu, University of Southern California, Los Angeles, USA; Sanjay Rao, sanjay@purdue.edu, Purdue University, West Lafayette, USA; Anthony Rowe, agr@andrew.cmu.edu, Carnegie Mellon University, Pittsburgh, USA; Ramesh Govindan, ramesh@usc.edu, University of Southern California, Los Angeles, USA.



This work is licensed under a Creative Commons Attribution 4.0 International License.
© 2025 Copyright held by the owner/author(s).
ACM 2834-5509/2025/12-ART34
https://doi.org/10.1145/3768981

Fig. 1. Left: Capture, **Right:** Two user viewpoints from an actual user trace.

1 Introduction

Volumetric Video. Volumetric video captures a three-dimensional representation of a continuously evolving scene (*e.g.,* a game, a live performance, a conference call). Each viewer of a volumetric video has 6 degrees of freedom (6-DoF) — he/she can view the video from any point in the captured space, in any direction. For example, a viewer can watch a live performance of a string quartet as though sitting next to a violinist, while at the same time, another viewer can take the cellist's perspective. Or, a technician can remotely assist in debugging and repairing an aircraft part, moving in space to inspect the part from different perspectives. Typically, viewers watch volumetric videos on a mixed-reality headset. They *interact* with the video by moving around in a physical space; the headset tracks the viewer's movement and, at any given moment, renders the video from a perspective consistent with the viewer's current position and orientation. Thus, in our example of the musical performance, viewers can move to change perspective during the performance (Fig. 1).

A volumetric video consists of a sequence of 3D frames, each representing a three-dimensional capture of a physical space. A *point cloud* [40, 55] is one representation of a frame. Each point in a point cloud corresponds to a 3D location (usually on the surface of an object) that has location coordinates (also called *geometry*) and color of the surface at that location.

In this paper, we consider volumetric videos obtained from an array of *low-cost commodity off-the-shelf* (COTS) RGB-D cameras encircling a scene. COTS RGB-D cameras, such as Azure Kinect DK [18] or Intel RealSense [13], can capture both the color (RGB) and depth (D) of points in a scene.

Truly Immersive Two-Way Conferencing. We consider the problem of streaming volumetric video between two locations (**A** and **B**) while permitting 6-DoF viewing at both ends. In contrast to prior work [51, 54, 89] on volumetric video conferencing that transmits 3D representations of a single human being (their face or torso), we seek to stream a *full-scene* from **A** that may consist of multiple participants and objects surrounding them (*e.g.,* furniture, the floor, walls, etc.). Full-scene conferencing would enable an application in which, for example, groups of actors at **A** and **B** could collaborate jointly to rehearse an upcoming theater performance. As users at **B** receive and view the video from **A**, they can move around the scene to change their perspective, permitting true immersion. Simultaneously, the movements of users at **B** are captured and transmitted to **A**, bringing a level of interactivity for human interaction comparable to physical presence.

Requirements. To realize this interactivity, a conferencing system must (a) transmit full-scene point clouds at *full frame rate* (30 frames per second) (b) with an end-to-end latency of 200-300 ms [28, 32, 52, 53]. Today's 2D conferencing systems satisfy these requirements. Full-scene point clouds can be much larger than those representing more constrained views (single human being with only their face or torso). For example, in one 3D dataset [46] we use in §4, the average point cloud frame size is about 1 MB uncompressed for a participant (single person in the scene), but is 10 MB for the full-scene (with furniture, floor, objects, *etc.*) (Table 3). As such, transmitting full-scenes will require significant network bandwidth. Today, average global broadband speeds

are nearly 100 Mbps and are expected to increase by 20% annually [16]; thus, we expect immersive two-way conferencing to be feasible in the near future, if it is not already.

Challenge: Compression. Even with increased broadband speeds, transmitting full-scenes will require compression. In our example above, each full frame must be compressed to a twentieth of its original size to fit the 100 Mbps capacity. The need for compression is unlikely to go away: even if broadband bandwidth increases, point cloud sizes might increase with better cameras.

Point cloud compression techniques such as Draco [4], V-PCC [80], and G-PCC [36] can potentially address this. Indeed, prior work has used these for on-demand [40, 55] and live [37] streaming of non-full-scene volumetric video. However, these techniques can be compute intensive, and their computational complexity grows linearly with point cloud size. For example, on a machine we use in §4, compressing a 1 MB point cloud (single person) using Draco takes 25 ms, while compressing a 10 MB frame (full-scene) takes over 300 ms. Other compression techniques are slower: 8 minutes using V-PCC for a 11 MB point cloud, and 10 seconds for G-PCC. These latencies can make it difficult to achieve the 200–300 ms end-to-end latency target discussed above. Parallelizing compression [37, 40, 45] can help, but the increase in compute cost can deter or delay adoption.

Point cloud compression techniques are also less compression-efficient than 2D video compression. This is because their *inter-frame* compression algorithms are the subject of ongoing research [42, 43, 94]. For instance, Draco's default setting can compress the 10 MB per full-frame video to about 1.78 MB per frame. In contrast, the approach we describe in this paper does not use point cloud compression and can compress to about 0.66 MB per frame on average by leveraging spatial and temporal redundancy in 2D video streams.

Challenge: Bandwidth-adaptivity. Even with increasing average bandwidths, it will be important for two-way conferencing to be *bandwidth-adaptive*. Bandwidth availability can vary spatially (*i.e.,* across different regions) and temporally (*i.e.,* during a single session). Two-way conferencing can exploit high bandwidth availability to deliver high quality. However, when bandwidth drops, it must deliver volumetric video without stalling.

Trace Names	Mean Capacity (Mbps)	MeshReduce		Livo	
		Mean TPS (Mbps)	Util. (%)	Mean TPS (Mbps)	Util. (%)
trace-1	216.90	40.19	18.53	158.75	**73.19**
trace-2	89.20	27.75	31.11	82.21	**92.16**

Table 1. LiVo achieves higher throughput, and utilizes the available capacity better than MeshReduce. The latter's indirect adaptation is conservative.

2D video conferencing systems (e.g., those that use WebRTC [23]) address this by (a) continuously estimating available bandwidth and (b) using a *rate-adaptive* codec implementation. Such a codec takes a desired bandwidth as input, and attempts to encode the frame at that target bandwidth [24] by internally controlling the *quality parameter* (QP) [71, 77], thus *directly* adapting to bandwidth changes. Today, 3D compression algorithms (*e.g.,* Draco [4], G-PCC [36]) are designed for directly compressing static point clouds or meshes, so they are **not** rate-adaptive: applications cannot specify a target output bitrate when compressing 3D frames (in theory, it should be possible to design rate-adaptive Draco codecs, but to our knowledge these don't exist yet). Instead, 3D compression algorithms can only control the *quality* of the encoder output. For example, Draco's *quality parameter* (QP) governs the degree of allowable distortion in the output; lower quality output requires lower rate for transmission. However, an application cannot know *a priori* what QP value to choose to achieve a given target bandwidth. V-PCC [80] supports *direct* rate-adaptation, since it encodes point clouds using 2D video codecs, but it takes several minutes to encode one point cloud frame.

Recent work, MeshReduce [45] is the only live volumetric video streaming system that supports bandwidth adaptivity. Since it uses Draco for compression, it profiles 3D videos offline to map bitrate to one or more parameters (like QP). During a session, given the current available bandwidth,

it instructs the compression algorithm to use parameters from the map. This approach can only *indirectly* adapt to bandwidth (as opposed to using a codec implementation that can directly encode at a given target rate). It can also be conservative: as Table 1 shows, unlike LiVo which uses direct bandwidth adaptation, MeshReduce produces encodings at significantly lower bitrates than the target bandwidth.

Contributions. To address these challenges while satisfying the performance requirements listed above, LiVo makes three distinct contributions (§3).

First, it exploits technology trends to compose streams from several RGB-D camera outputs into just two video streams: a color stream and a depth stream. This uses significantly lower encoding resources, and requires less complex receiver-side synchronization when reconstructing point clouds (§3.2). Some recent systems [39, 51, 54, 89] stream volumetric video frames as 2D color and depth streams, but they only consider constrained views (*e.g.*, the face, the torso, or single person). Full-scene conferencing requires encoding larger physical depth ranges; for this, LiVo develops a novel depth encoding and a video transmission mode that reduces distortion due to depth. This allows LiVo to use widely-deployed 2D video codecs to compress both color and depth streams.

LiVo's second contribution is a novel *bandwidth-splitting* strategy that continuously *adapts* to both bandwidth availability and scene complexity changes. Even though bandwidth-adaptivity comes for free when using 2D video, LiVo needs to carefully split available bandwidth between color and depth streams while being able to process the video at full frame rate. Humans are sensitive to distortion in depth, so LiVo must allocate more of the available bandwidth to depth than to color. A static split is sub-optimal because bandwidth needed to ensure high-quality delivery can depend on scene complexity (*e.g.*, number of participants or objects in the scene). This motivates a fast, adaptive, dynamic splitting strategy (§3.3), which no prior work has considered.

Finally, LiVo employs a novel, efficient *view-culling*, so a LiVo sender needs only send 3D information within the receiver's current field of view (§3.4). This increases bandwidth-efficiency by transmitting less data. Prior work [40, 55] culls 3D point clouds, but LiVo (a) culls 3D views efficiently *without reconstructing the point cloud* and (b) exploits the fact that the tighter end-to-end latency of conferencing results in a smaller prediction horizon, which permits LiVo to use cheap, accurate prediction of receiver views relative to systems for on-demand volumetric video [40, 55].

Using a complete implementation of LiVo, which we have released publicly (https://github.com/USC-NSL/LiVo), we have conducted a user study and measured objective 3D quality metrics. In these, LiVo consistently outperforms three baselines: a bandwidth-adaptive *oracle* of Draco [4], a popular point cloud compression technique; an approach that mimics a bandwidth-adaptive version of Project Starline [51]; and a complete implementation of MeshReduce [45]. Experiments also demonstrate that LiVo can achieve around 250 ms end-to-end latency at 30 frames per second (fps) with negligible stalls, comparable to the performance of existing 2D conferencing platforms [28, 53].

Statement of Ethics. We obtained Institutional Review Board (IRB) approval for collecting user interactivity traces with 3D videos, and for our user study (§4). As required by the IRB, to protect user privacy, we removed all identifiable information and our analyses use only aggregated statistics.

2 Related Work

Table 2 compares LiVo with prior work in volumetric video streaming. To our knowledge, LiVo is the only full-scene bandwidth-adaptive conferencing system that operates at 30 fps.

On-demand Streaming. A line of work has explored on-demand volumetric video streaming [26, 40, 55, 62, 91, 96], and focused on reducing data rate and improving decoding efficiency. ViVo [40] culls occluded points or those outside the predicted frustum. Groot [55] employs parallelism to improve Draco decoding efficiency and achieve 30 fps. Fumos [60] improves data rate by exploiting

inter-frame redundancy using a neural codec and supports high frame rate decoding. In contrast, LiVo avoids point cloud compression to achieve performant two-way live conferencing.

Streaming Systems	Type	Compression	Content	BW-adaptive?	FPS	Cull?
ViVo [40]	On-demand	3D	Multiple Person	Indirect	30	✓
Groot [55]	On-demand	3D	Full-scene	No	30	✓
Fumos [60]	On-demand	3D	Single Person	No	30	✓
Vues [62]	On-demand	2D	Multiple Person	Indirect	30	✓
Holoportation [74]	Conferencing	3D	Single Person	No	30	✗
LiveScan3D [50]	Live	3D	Full-scene	No	15	✗
FarFetchFusion [54]	Live	2D	Portrait	No	30	✗
Starline [51]	Conferencing	2D	Upper Torso	No	30	✗
Tele-Aloha [89]	Conferencing	2D	Upper Torso	No	30	✗
VRComm [39]	Conferencing	2D	One Person	No	30	✗
MeshReduce [45]	Live	3D	Full-scene	Indirect	15	✗
MetaStream [37]	Live	3D	One Person	No	30	✗
LiVo	Conferencing	2D	Full-scene	Direct	30	✓

Table 2. Comparison to related work, more details in §2.

Live Streaming and Conferencing. LiveScan3D [50] uses a fast binary compression method, zstd, which achieves low compression ratios. Holoportation [74] achieves conferencing by compressing a 3D representation of a single person using LZ4 binary encoding, which doesn't scale to larger scenes. Starline [51], a conferencing system, streams upper body views in 3D over fixed quality 2D video streams; §4 describes how it differs from LiVo. FarfetchFusion [54] streams only the face and is optimized for mobile devices. MeshReduce [45] supports full-scene conferencing using a mesh, but achieves a lower frame rate and lower quality (§4). MetaStream [37] uses Draco encoding to stream a single person at 30 fps by reducing the number of points in the captured point cloud. Unlike LiVo, none of the live volumetric streaming systems support full-scene 2-way live conferencing with bandwidth adaptation at 30 fps.

Other 3D Formats and Compression Standards. Besides point clouds, prior work explores textured meshes [27, 29, 45, 68, 69, 74]. Meshes are generally higher quality representations that are much more bandwidth-intensive; future work can explore mesh-based two-way bandwidth-adaptive live streaming. Besides Draco, prior work has proposed other point cloud compression techniques: G-PCC [36], V-PCC [80], and PCL [11]. Draco achieves faster compression with higher compression ratios than these, so we base our evaluations on it. Recently, Hermes [91], patchVVC [26], and DeformStream [56] have attempted to exploit inter-frame redundancy in volumetric videos, but these have high encoding latency, so can only be used for on-demand streaming.

Learned 3D Representations. Recent work uses learned 3D representations such as Neural Radiance Fields (NeRFs) [65] and Gaussian Splatting (GSplat) [48] for streaming and conferencing. Many papers [35, 57, 61, 82, 85, 86, 90, 92] have explored learned 3D representations for on-demand streaming. Tele-Aloha [89] has used GSplat for conferencing but constrains only to upper torso of a person like Starline [51]. Though these representations can generate high-quality rendering superior to point clouds or meshes, they suffer from high training and rendering overheads, rendering them currently unsuitable for live or conferencing systems.

Quality Metrics. Other work has explored different 3D objective quality metrics: point2point-PSNR [88], PCQM [64], and GraphSIM [95]. These compare point positions and attributes such as colors, normals, and curvatures in 3D. We choose PointSSIM for measuring quality since it can measure both geometry and color distortions by directly extending the popular SSIM metric to 3D. Future work can evaluate LiVo using other metrics.

3 LiVo Design

LiVo addresses the two challenges discussed above — it efficiently encodes 3D content *and* employs direct bandwidth-adaptation — while ensuring efficient, high-quality volumetric video transmission.

3.1 Overview

LiVo enables volumetric video conferencing between two sites. Each site has an *array* of off-the-shelf RGB-D cameras and a desktop-class computing device for processing, sending, or receiving

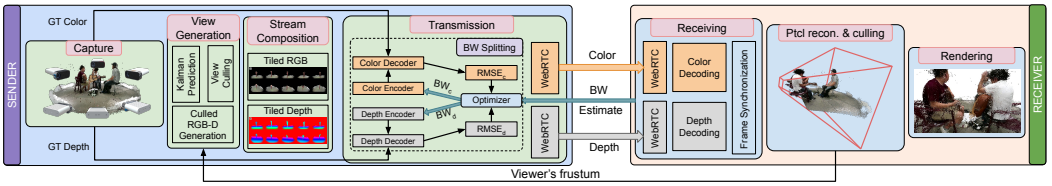


Fig. 2. LiVo Architecture. The green blocks on the left run on the sender and blue blocks on the right run on the receiver.

volumetric video from the other site, as well as a viewing device (*e.g.,* a mixed reality headset). Multi-way conferencing can be built using LiVo, but presents opportunities for optimizations (*e.g.,* across receivers from a single sender) that we leave to future work.

Fig. 2 shows the design of LiVo's sender-to-receiver pipeline; in the deployment model above, each site runs one instance of the pipeline in each direction. LiVo captures a sequence of RGB-D frames from each camera in an array encircling a scene (*e.g.,* a conference table, a small stage or performance area, or a small room). At a given instant, frames from the cameras produce a point cloud (§3.2), and a sequence of such point clouds constitutes the volumetric video.

First, LiVo transmits the volumetric video as streams of 2D video, instead of transmitting point clouds. This choice permits LiVo to re-use mature widely-deployed technology: (a) off-the-shelf highly efficient video compression standards such as H.264 and H.265; (b) mature rate-adaptive open-source codec implementations (*e.g.,* GStreamer [7] and FFmpeg [6]); and (c) real-time transport for video conferencing such as WebRTC [23] with its built-in Google congestion control [24].

Second, LiVo *culls* points that fall outside the receiver's *frustum*. The frustum represents the receiver's 3D field of view (Fig. 2). When the receiver is viewing the entire scene, her frustum includes the entire point cloud. However, if she moves closer to objects or participants in the scene, the frustum will likely include a much smaller part of the point cloud (Fig. 2).

Together, these address the two challenges described in §1. 2D codecs employ inter-frame compression, resulting in higher bandwidth-efficiency relative to point-cloud compression. Culling further reduces the bandwidth requirements. Moreover, 2D codecs can encode to a target bandwidth, so LiVo can directly adapt to bandwidth changes. Finally, there exist mature, compute-efficient implementations of 2D codecs that can, with low latency, process video streams at full frame rate.

Even though it relies maximally on 2D video technology, LiVo's design poses two significant challenges: (a) how to encode RGB-D frames in 2D videos and how to adapt these to available bandwidth; (b) how to determine receiver frustum and cull views at the sender. We describe how LiVo addresses these challenges in the remainder of this section.

3.2 2D Video Encoding

Background: Point Clouds from RGB-D Camera Array. Each camera produces RGB color and depth frames at 30 frames per second. Consider a static RGB-D camera array (Fig. 2) with N frame-synchronized cameras[1]. Then, for every inter-frame interval (30th of a second), we can generate a point cloud from N frames, one from each camera. To do this, we first need to determine the position and orientation of each RGB-D camera in a common frame of reference. There exist standard one-shot calibration techniques for this [97] that produce a transformation matrix to convert each camera's local coordinate system to a global one. Then, we can downsample[2] the color image resolution to match the depth image resolution to make them pixel-aligned [1, 10, 50]. Generating a point cloud is then simple: for each pixel of each RGB-D frame, first determine the

[1]There exist techniques to synchronize Kinect cameras [14].

[2]The alternative is to transmit color at full resolution, and use depth super-resolution at the receiver. This can incur lower quality.

pixel's position in the camera's local coordinate frame (using camera parameters such as its center and focal length [37]), and then convert it to global coordinates (using the transformation matrix) to obtain one point in a point cloud.

The Challenge. Prior work in volumetric video generates a point cloud at the sender, then applies point cloud compression (*e.g.,* Draco [4]) [37, 50]. LiVo is qualitatively different: it encodes 2D RGB-D video frames *directly* at the sender; the receiver then decodes and constructs a point cloud for viewing. An array of N cameras produces N color frames and N depth frames at 30 fps.

For these frames, LiVo must solve three distinct problems: *stream composition*, *depth encoding*, and *bandwidth splitting*. We discuss these in this and the next (§3.3) subsection.

Stream Composition. Stream composition refers to the problem of multiplexing the $2N$ images into one or more video streams. Multiplexing all images, depth and color, onto a single stream defeats inter-frame prediction because successive frames might be from different cameras or might interleave color and depth frames. At the other extreme, independently encoding each camera's output into a color stream and a depth stream increases complexity in three ways.

First, the sender needs to run $2N$ parallel encoders, N for color and N for depth streams. Hardware-accelerated codecs often limit the number of streams that can be concurrently encoded. For example, `nvenc`, NVIDIA's GPU-accelerated codec library [8], allows up to 8 parallel encoders on any desktop-class GPU [72]. In conferencing setups with more than 4 cameras, it is then infeasible to encode these streams in parallel on a GPU. They can be encoded in batched-parallel fashion; for example, 20 streams from 10 RGB-D cameras can be encoded in 3 batches (8/8/4, respectively), but this can increase latency. CPU encoding is expensive: encoding 20 parallel streams using `x265enc` [21] incurs 2-3 sec per stream on a machine we use in §4, which is infeasible for conferencing.

Second, if LiVo composes frames into more than one stream, it must determine how to split available bandwidth to each stream (§3.3). The more streams there are, the more complex the splitting algorithm. Third, the receiver must reassemble corresponding color and depth frames from different streams to reconstruct the point cloud. The reassembly logic can be complex, especially when different streams incur different delays.

LiVo's approach: Tiling. LiVo leverages technology trends to compose streams in a novel way. Off-the-shelf RGB-D cameras have lower depth resolution than color resolution. For example, the Azure Kinect DK [18] offers 640×576 *depth* image resolution, and the Intel RealSense D457 [19] supports up to 1280×720, but their color images can have up to 4K resolution. This is because these cameras usually include a time-of-flight sensor that measures, for each pixel, the depth of that pixel. The cost of these sensors has resulted in the slower growth of RGB-D camera depth resolution.

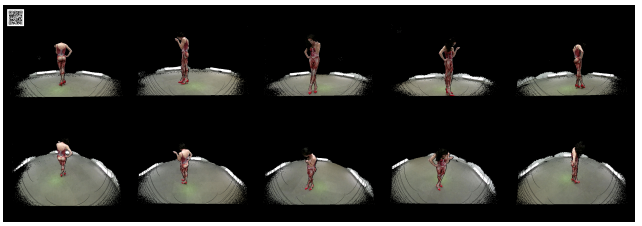


Fig. 3. Tiled color view for 10 Kinect cameras. A Similar view is generated for depth.

LiVo exploits this observation to multiplex the $2N$ images into *two* videos: a color video and a depth video. Specifically, it ***tiles***[3] the N depth images into a 4K frame (Fig. 3). RGB-D cameras

[3]This tiling is qualitatively different from tiling in 360° video. In LiVo, each tile represents a color or depth image from an RGB-D camera. In 360° video, tiles represent arbitrary tessellations of the 360° view and permit efficient viewport selection. It is also different from prior work such as HoloKinect [83] and VRComm [39], which tile color and depth from 1 camera into one stream and encode this stream using H.264. Moreover, their depth encoding assumes 1-1.5 meters of depth range, whereas Kinect cameras have a range of 5–6m.

generate color images at a much higher resolution (up to 4K), but it would be wasteful to transmit this, since the receiver would first downsample the color image to match the depth image resolution to construct the point cloud. Upsampling the depth to match color resolution is not followed in practice to avoid introducing distortion in geometry [1, 10, 50]. Accordingly, LiVo downsamples color images to the depth resolution, then tiles them into a 4K color frame. Using this approach, LiVo can fit 9 Intel RealSense cameras or 16 Azure Kinect DK cameras into two 4K frames (for color and depth).

While most commodity RGB-D cameras have resolution similar to Azure Kinect DK or Intel RealSense, some newer cameras have a higher depth image resolution [79]. For example, Astra 2 [73] has the highest depth image resolution of 1600×1200 among commodity RGB-D cameras. For such cameras, LiVo can tile frames into two 8K frames; these can fit up to 30 of these cameras. Today's hardware-accelerated codecs like `nvenc/nvdec` [20, 72] can support up to 8K streaming at 30 fps on desktop-class GPUs. This approach should work for high resolution LiDAR sensors [44, 75, 87] as well, but we have left it to future work since our focus is on commodity, low-cost, conferencing setups.

Given that these cameras have a modest range (5–6 m) and can only cover small spaces, we do not foresee deployments with significantly more RGB-D cameras (other work also makes similar assumptions about the number of cameras [17, 51, 54, 74]). With more cameras, we may need to multiplex images onto more streams and add more complex receiver synchronization, which we have left to future work.

Consistently tiling camera images on a 4K frame does not impact video compression efficiency. 2D video codecs predict macroblocks (8x8 or 16x16 pixel blocks) within and between frames. In successive frames, the location of the macroblocks is fixed (images from the same camera are located at the same spot in the tiled image). Inter-frame prediction is thus relatively unaffected by tiling[4] since the macroblocks preserve locality, improving compressibility.

After tiling, LiVo passes each color 4K image and depth 4K image to two separate H.265 encoders[5] (see below for depth). These can encode 4K frames at full frame rate.

Depth Encoding. Early work has designed new coding strategies for depth [33, 63, 66], or strategies that encode depth in existing codecs but specialize them for specific types of 3D displays [49, 63]. For pragmatic reasons, we focus on techniques that encode depth in existing, widely deployed codecs. These approaches have explored two different techniques to encode depth into images. One approach packs each depth pixel value into a 3-channel color pixel (RGB) before encoding using 2D video codecs such as VP9, H.264, *etc.* [76, 84]. In general, this approach can introduce significant distortions, since video compression algorithms exploit smoothness in natural images to achieve compression, but depth information can exhibit discontinuities [49, 76, 93]. These discontinuities can result in poor 3D visual quality, since the 3D views must be reconstructed from multiple decoded views, each of which can exhibit depth errors (§4.5, Fig. 17). LiVo must attempt to minimize depth distortion, especially since humans are very sensitive to point cloud depth errors [95].

Other work [51, 54] encodes 10-bit depth into the Y-channel of a YUV [12] encoding. Codecs compress the Y-channel (representing luminance) at higher bitrates to minimize loss because humans are sensitive to luminance distortions. For this line of work, this depth encoding works well because they target portrait 3D videos (of faces or upper torsos), which require a lower depth range (1–2 meters). Our RGB-D cameras output 16-bit depth values at millimeter resolution, so

[4]For the same reason, tiling does not affect reconstruction quality relative to transmitting each stream separately.

[5]Multi-view coding [25, 31] can eliminate redundancies from multiple RGB-D cameras. In our setting, we expect gains from multi-view coding to be minimal, since we envision a circular arrangement of cameras with minimal overlap.

they can capture larger scenes. Quantizing these to 10 bits can sacrifice either depth or resolution, potentially impacting perceived quality and impairing full 6-DoF capabilities.

LiVo's Depth Encoding. LiVo leverages a YUV H.265 mode which represents each channel using 16 bits [20]. LiVo stores the 16-bit depth pixel value in the Y channel and sets the U and V channels to the same fixed value. This reduces depth distortion since codecs distort the Y-channel less. This H.265 mode is supported by `nvenc` [71], an encoder available for *all* consumer and professional-grade NVIDIA GPUs; as such, it ensures wide applicability of LiVo.

However, simply storing the depth value in 16 bits on the Y channel introduces significant artifacts (Fig. A.1). Current depth cameras have a maximum depth range of 5–6 meters [13, 15, 79], so the depth values can range from $0 - 6000$ at millimeter resolution. This uses only a portion of the full 16-bit range. So, we scale the depth value to occupy the entire 16-bit range, *i.e.,* scaled depth value for 0 mm remains at 0 while it is $2^{16} - 1$ for 6000 mm[6]. This approach incurs lower depth distortion: codecs quantize depth values, and, for a given quantization step size, more unscaled depth values fall into one quantization bin than scaled depth values. To understand this, consider two nearby depth values x and $x + v$. Without scaling, if the quantization step size is larger than v, the compression algorithm will represent these two values with the same encoded number. When scaled by a factor of $k > 1$, these will be represented by distinct values, as long as the quantization step size is smaller than kv, which will ensure that the decoder can distinguish between these values.

3.3 Bandwidth Splitting

Background: WebRTC and Rate-adaptive Codecs. 2D video conferencing systems use a real-time transport protocol (*e.g.,* WebRTC [23]) with rate-based congestion control (*e.g.,* GCC [24]). The sender feeds the available bandwidth from congestion control to a rate-adaptive video encoder, which compresses the frame to fit within the target bandwidth.

The Challenge. LiVo transmits depth and color streams as two separate WebRTC streams. Suppose, at a given instant, the sender's rate control algorithm estimates an available bandwidth of B. LiVo cannot assign $B/2$ to each stream. It encodes depth (§3.2) to reduce depth distortion, but in order to get higher quality, it must *also* assign more bandwidth to the depth stream than the color stream. To understand why, Fig. 4 shows the variation in depth and color quality for different splits of a target bandwidth of 80 Mbps for one of the videos we use in §4. If each stream were to receive $B/2$ (*i.e.*, split=0.5), depth error can be significant. When the depth stream gets 90% of the available bandwidth, the error in depth and color is most balanced. This is because humans are much more sensitive to depth than to color errors [95]; allocating more bitrate to depth reduces depth distortion [98]. Moreover, distortion in depth can also affect objective measures of color quality [98].

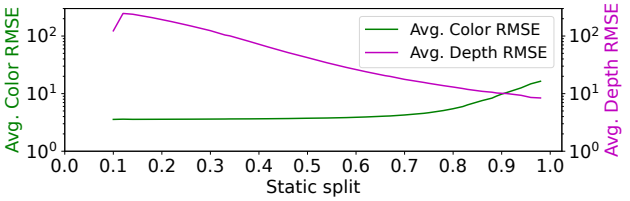


Fig. 4. Color and depth RMSE for different splits at target bandwidth of 80 mbps. Log-scale y-axis. Video sequence: *band2*

Thus, to ensure high quality, LiVo must determine the *bandwidth split s*: the fraction of available bandwidth allocated to the depth stream such that depth and color errors are the same[7]. A strawman approach might profile, offline, volumetric videos to determine an optimal *static* split at a given bandwidth. Intuitively, this split jointly minimizes distortion in depth and color. For example, Fig. 4

[6]In future, the same depth scaling mechanism can be easily applied to larger depth ranges.

[7]Other objectives are possible, such as minimizing a weighted sum of the two errors. Our choice results in high quality (§4), so we have left an exploration of other objectives to future work.

suggests a split s of 90%. This may not generalize; the optimal split can vary from frame to frame, as *scene complexity* — the number of participants, or the degree of motion — changes. It might be possible to design learned split predictors, but these might need considerable data with varying degrees of scene complexity to generalize well, and we have left this to future work.

LiVo's Approach: Adaptively Splitting Bandwidth. Instead of determining the split statically, LiVo *adaptively* and continuously determines the split to capture bandwidth and scene complexity changes. It repeatedly encodes a frame with a given split s, determines the resulting quality of the encoded frame, and then either increases or decreases s to improve depth and color quality. To do this, it must solve two problems: (a) how to estimate encoding quality and (b) how to adapt.

Estimating encoding quality. LiVo's approach encodes a frame at the sender, immediately decodes it, and then computes distortion relative to the ground truth frame (available at the sender). Today, GPU-accelerated video encoders and decoders, like NVIDIA's `nvenc/nvdec` [8], can decode multiple frames while also encoding other frames. Moreover, NVIDIA does not limit the number of parallel decoders[8] on a desktop-class GPU [72]. So, while the GPU is encoding a frame, it can simultaneously decode a previously encoded frame to determine quality. We have found that parallel decoding adds a modest 11% more GPU utilization on a desktop-class GPU like NVIDIA GeForce RTX 1080 Ti for tiled views from 10 cameras.

To estimate the quality of the decoded frames, LiVo could use a point cloud quality metric such as PointSSIM [22], but this requires the sender to reconstruct the point cloud before and after encoding. Instead, LiVo uses the root-mean-square error (RMSE) in pixel values between the original (depth or color) frame and the decoded frame[9]. This choice is far more compute-efficient. LiVo further reduces the compute overhead by computing RMSE every k frames ($k = 3$, chosen empirically in our evaluations, §4), instead of every frame, which suffices to capture scene dynamics in conferencing at 30 fps.

Adapting the split. LiVo continuously adapts the split using the following steps (Fig. 2):

- It uses two video decoders (running in parallel) which decode the compressed tiled frames to generate the distorted tiled color and depth frames.
- For each decoded color (resp. depth) frame, it calculates the color RMSE $RMSE_c$ (resp. depth RMSE $RMSE_d$) by comparing the distorted tiled frame (F') to the ground truth tiled frame (F).
- LiVo does not modify the split if $|RMSE_d - RMSE_c| \leq \epsilon$, where ϵ is a parameter to the algorithm.
- It finds the optimal split using multi-dimensional line search [41, 70]. This process additively increases or decreases s. If $RMSE_d - RMSE_c > \epsilon$, then s increases by δ (the step size). Else, s decreases by δ.

At the beginning of a session, s is set to an initial value s_i. This can be estimated empirically from video data (*e.g.,* Fig. 4) or can be set to a fixed initial value.

Multi-dimensional line search is sensitive to δ. A smaller value can result in delayed convergence when the scene complexity changes, while a larger one can oscillate around the optimal split. We have empirically chosen a step size of 0.005 to balance these two concerns. We also choose $0.5 \leq s \leq 0.9$. The lower limit ensures depth always gets more bandwidth than color. When available bandwidth is too low, $RMSE_d - RMSE_c$ may always exceed ϵ, which will drive s to 1, and this can significantly degrade color quality. To prevent this, we clamp s at 0.9.

[8]As discussed above, it only limits the number of parallel encoders.

[9]LiVo's bandwidth splitting tries to fit the best quality tiled color and depth frame within the available bandwidth instead of maximizing compression gains.

3.4 View Prediction and Culling

In addition to encoding RGB-D frames as 2D videos, LiVo *culls* points outside the receiver's view by (a) determining the receiver's *frustum* and (b) efficiently removing points outside the frustum.

Challenge: Frustum Prediction. The frustum is determined by the pose (position and orientation) of the receiver's headset, as well as the parameters of the viewing device (such as the focal length and aspect ratio). Today's headsets can provide values for these, and the receiver can transmit these to the sender. Even so, given the feedback delay between the receiver and the sender, the latter must *predict* the frustum of the receiver at the time the receiver views the frame.

On-demand 360° and volumetric video systems [34, 40, 62, 78] train viewport predictors from user data. In that setting, if many users watch the same video, learned predictors can be accurate. In a conferencing setting, it is less clear if learned predictors can be accurate, since each conference call is unique and not enough user traces may be available for the same content. We show (§4) that predictors trained on a few traces perform poorly relative to LiVo's approach.

Challenge: Efficient Culling. Given a frustum and a point cloud, the *culled point cloud* consists of points inside the frustum. Prior work [40, 55] has used culling for on-demand volumetric video streaming. However, RGB-D cameras *do not provide point clouds*. LiVo can generate the point cloud from RGB-D frames, but this process can be slow. LiVo's frustum prediction and fast culling scheme can generate culled RGB-D frames within 30 ms (§4.4).

LiVo's Approach: Frustum Prediction. When culling a frame at time t, LiVo's sender must predict the receiver's frustum at $t + \Delta t$, where Δt is the one-way delay from sender to receiver (including both network and processing delays at the two ends). LiVo obtains Δt by halving a smoothed application-level RTT estimate.

LiVo predicts frustums by applying a Kalman Filter [47] on the 6 dimensions of receiver pose (position and orientation) based on prior work [38]. Still, prediction errors can arise from sudden user movement or errors in one-way delay estimates. To counter these, LiVo expands the predicted frustum by a guard-band (ϵ). For our datasets, an ϵ of 20 cm represents a sweet-spot in the trade-off between compression efficiency and reconstruction accuracy (§4.5). Future work can explore other techniques to ensure better tracking accuracy.

LiVo's Approach: RGB-D View Culling. This step, performed *before* stream composition and depth encoding (§3.2), is invoked every inter-frame interval, and takes as input the viewer's frustum and the N RGB-D views (instead of a point cloud, as in prior work). For each pixel in each image, it must determine if that pixel falls within the frustum or not, and replace culled pixels with a zero value (both for color and depth). LiVo culls without reconstructing the point cloud. Instead, it determines whether a pixel in an RGB-D frame is within the receiver's frustum, using the following technique. For each RGB-D camera, LiVo first transforms the frustum into the local coordinate system of the camera. Then, for each pixel, it obtains that pixel's local coordinates and determines if it lies within the frustum as follows. At its core, culling must determine if a point P lies outside the frustum. A frustum is a 3D truncated pyramid defined by six planes — near, far, top, bottom, left, and right — whose plane normals point inwards. P is outside the frustum if distance of the point from either of the six planes is positive, *i.e.,* the point lies in the direction of one of the six normals of the planes pointing outwards. Otherwise, it is inside or on the frustum.

Other Design Details (§A.1). LiVo speeds up receiver rendering by voxelizing the point cloud, pipelines computation stages to achieve 30 fps, and adjusts socket buffer settings to minimize loss.

4 Evaluation

We compare LiVo to other alternatives, using both a user study and a trace analysis on real bandwidth traces. We also quantify its compute-efficiency and quantify the impact of its design choices.

4.1 Methodology

Implementation. Our LiVo implementation (Fig. 2) contains 15K lines of C++ and a few hundred lines of Unity code. It implements pipelined capture, view generation, and tiling using Boost [2] and OpenMP [9]. It also uses: Eigen [5] for linear algebra operations for point cloud transformation; the Kalman Filter implementation in OpenCV; WebRTC[10] and `NVENC` (NVIDIA Video Codec) in GStreamer [20] for color (H.265, BGRA) and depth (H.265, Y444_16LE); and Open3D [99] for rendering on PC or on a headset using Unity. LiVo captures RGB-D frames using the Azure Kinect SDK [1], and synchronizes them as described in [14].

Evaluation testbed. Our evaluation uses two desktop-class machines, each with 32 GB RAM and 1 Gbps Ethernet. Both have modest computing resources: one is an Intel i7-8700K @ 3.70GHz, with one NVIDIA GeForce GTX 1080 Ti; the other is an Intel Core i9-10900KF @ 3.70GHz and one NVIDIA GeForce RTX 3080. We use Chrony [3] to synchronize these two machines. While we evaluate one-way streaming, LiVo supports two-way streaming if each user runs an instance of LiVo's sender as well as its receiver.

Traces and Datasets. To compare LiVo against other alternatives while subjecting all of them to the same workload, we use volumetric video *replay* instead of live transmission.

Videos	Duration (s)	Objects	Frame Size (MB)
band2; Musical performance	197	9	11.1
dance5; Dance	333	1	10.8
office1; Person working	187	7	10.6
pizza1; Food and party	47	14	13.8
toddler4; A child playing games	127	3	10.6

Table 3. Summary of 5 videos in the Panoptic Dataset. Duration is in seconds, objects include people, and frame sizes are in MB.

Volumetric videos. We use the Panoptic Dataset [46], which contains videos captured at 30 fps using 10 Kinect v2 RGB-D cameras. We select 5 videos (Table 3) of duration 1–5 minutes that capture 1–6 participants performing a range of activities providing a fully immersive live experience, *e.g.,* dance, musical band, party, and games. Their frame sizes, 10.6–13.8 MB, are much larger than the ones in prior work (Table 2).

User traces. When a user interacts with a volumetric video by moving to change perspective, the sequence of her instantaneous poses (position and rotation) constitutes a user trace. There are no publicly available user traces for the videos in Panoptic dataset, so we collected these under an IRB-approved study. In a 30–45 minute session, we asked each user to view 2–3 videos selected randomly. Users could freely move around in a sufficiently empty room while viewing the videos, and their headset recorded their instantaneous poses. We collected three user traces for each video.

Trace Names	Bandwidth (Mbps)				
	Mean	Max	Min	90^{th}	10^{th}
trace-2	89.20	106.37	36.35	98.09	80.52
trace-1	216.90	262.19	151.91	234.41	191.52

Table 4. Statistics of the bandwidth traces.

Network traces. We replay the videos on two real-world bandwidth traces representative of user experience in stationary and mobile environments. **trace-1** (Table 4) captures home Wifi throughput variation [59]. **trace-2** captures a WiFi connection while moving around in a shopping mall [58]. These traces likely exhibit variability (§A.3) comparable to broadband connections, but not their capacity. As such, by themselves, these traces cannot support volumetric video conferencing. For this reason, we scaled *trace-2* by 15× and *trace-1* by 10× to reach mean throughputs around 90 Mbps and 217 Mbps, respectively; the

[10]WebRTC is widely used for low-latency video conferencing today. Future work can explore RTP over Quic [30].

former corresponds to current broadband bandwidth, while the latter is representative of broadband bandwidth in 3–4 years [16].

Trace replay. This reads RGB-D frames from disk at 30 fps and feeds them into LiVo sender (Fig. 2). Using the frame sequence number, the receiver retrieves the corresponding user frustum from the selected user trace file, then culls and renders the point cloud. We replay the network traces using Mahimahi [67] to emulate the bandwidth conditions between sender and receiver.

Comparison Alternatives. We compare the performance of LiVo with other alternatives inspired by live volumetric video streaming systems such as Holoportation [74], Project Starline [51], LiveScan3D [50], and MetaStream [37] (Table 2). None of these schemes are bandwidth-adaptive, and all of them achieve live transmission on a network with sufficient bandwidth by sending more constrained views (*e.g.,* single person or torso, Table 2). Recall that these constrained views require an order of magnitude lower bandwidth than full-frame transmission. Most of these do not provide open-source implementations, so we resort to mimicking their essential video compression and transmission strategies, but adapt these for full-frame transmission. To this end, we design two baselines: Draco-Oracle and LiVo-NoCull. We tune essential parameters of these systems, so we can compare them with LiVo. MeshReduce [45] is the only system that is open-source, supports bandwidth adaptivity, and streams full-scene, so we compare it head-to-head.

Draco-Oracle. Volumetric video on-demand systems such as ViVo [40] and GROOT [55] and live systems such as MetaStream [37] use Draco [4], an open-source point cloud compression library. They have shown that Draco provides either better compression ratio or lower encoding latency than other point cloud compression schemes like PCL [11], G-PCC [36], and V-PCC [80]. Draco by itself is not rate-adaptive, which is why prior work that uses Draco is not bandwidth-adaptive. To understand what would happen if Draco were to develop rate-adaptivity, and yet transmit the information that LiVo does in our evaluations, we designed a Draco-Oracle: given a target bandwidth and a perfect estimate of a receiver's frustum (*perfect culling*), it picks the highest quality compression for the point cloud that fits within the target bandwidth.

To do this, we compute *offline* a table from each video frame and each user trace. This maps, for each video frame and user frustum and each Draco compression level (Draco has 10 of these) and quantization parameter (Draco supports 31 of these), the *time to compress* the perfectly-culled frame, and the *compressed size* of this frame. During playback, we use this map to find the best quantization parameter and compression level that fits the bandwidth estimate, and whose compression time is smaller than the inter-frame interval. If no such entry exists, we record a *stall* for Draco-Oracle.

At 30 fps, Draco-Oracle exhibits over 90% stalls for most of our video sequences. So, our evaluations use a lower frame rate, 15 fps, consistent with prior work [50].

LiVo-NoCull. This baseline runs LiVo without culling. In addition to quantifying the importance of culling, LiVo-NoCull is intended to mimic one small aspect of Project Starline [51], which also streams RGB-D captured from multiple depth cameras over 2D video streams using WebRTC and performs 3D reconstruction in the receiver. Starline streams 2D videos at a fixed quality, but LiVo-NoCull includes bandwidth splitting (§3.3) and bandwidth-adaptivity (in §4.5, we also quantify the impact of disabling bandwidth-adaptivity). So, this baseline seeks to understand what would happen if Starline were to be bandwidth-adaptive. Starline is an impressive system that differs in *many* other respects from LiVo, so we do not intend LiVo-NoCull to represent Starline's performance. We run LiVo-NoCull and LiVo at 30 fps on all our videos.

MeshReduce. MeshReduce [45] is a mesh-based full-scene live volumetric video streaming system, which represents any 3D surface as a collection of interlocking triangles, not a collection of points. The sender captures a RGB-D frame from off-the-shelf RGB-D cameras, reconstructs

a per-frame mesh, encodes the geometry and color separately, and transmits over 2 TCP socket connections. It compresses mesh geometry using Draco and mesh texture using H.264.

MeshReduce employs *indirect* bandwidth adaptation (§1): using a profile obtained from an offline analysis, it determines the best compression parameters for a given level of available bandwidth. We use the publicly available MeshReduce implementation, which decides these parameters based on the average bandwidth availability in a trace and by profiling videos offline.

Metrics. Aside from end-to-end latency and frame rate, we also measure objective point cloud quality. Prior work [40, 55, 62] has used 2D metrics such as SSIM and PSNR on a rendered 2D image of the point cloud. However, these metrics are inadequate for several reasons. 3D to 2D projection introduces errors. 2D pixel size influences quality, but points in a point cloud do not capture size. A slight change in point position due to depth coding error can impact scene geometry minimally, but can result in a large drop for 2D metrics. These metrics depend on the distance of the viewer from the point cloud. Finally, they are influenced by the color of background pixels.

We use PointSSIM [22] (or PSSIM), which is analogous to SSIM for 2D images, but extends it to 3D by exploring a higher-dimensional feature space that includes geometry, normals, curvatures, and color attributes. PSSIM separately captures the quality of depth and color. It ranges from 0 to 100; higher values are better, and values in the high 80s or above are generally considered good. PSSIM is not defined for meshes, so to compare MeshReduce against LiVo, we sample as many points from the rendered mesh as there are in the ground truth point cloud, then compute PointSSIM.

4.2 User Study

Setup. We conducted an IRB-approved user study to measure the perceptual quality of LiVo against other alternatives. Each participant viewed 2-3 videos from Table 3 over 45–60 minutes. For each video and a randomly selected user and network trace, the participant passively observed the video from the perspective of the user who contributed the trace (§4.1). Participants rated the videos across 4 schemes on a Likert scale of 1 (worst) to 5 (best) with respect to the ground truth and (optionally) left comments. These experiments used the replay infrastructure described in §4.1.

Perceptual Quality Results. Fig. 5 shows the distribution of perceptual quality scores for each of the 4 schemes across 20 participants. Each scheme received a total of 57 ratings across all combinations of <video, user trace, network trace>. Draco-Oracle performs poorly with a Mean Opinion Score (MOS) of 1.5 due to 3 factors: Draco is streamed at 15 fps; even at 15 fps, it experiences 36 – 98% stalls; and Draco-Oracle settles for lower quality to remain within compute and bandwidth budgets. While other work on volumetric video streaming has used Draco to achieve good quality, in our setting with large point clouds from full-scenes, Draco proves to be a poor choice as it is both compute-intensive and compression-inefficient.

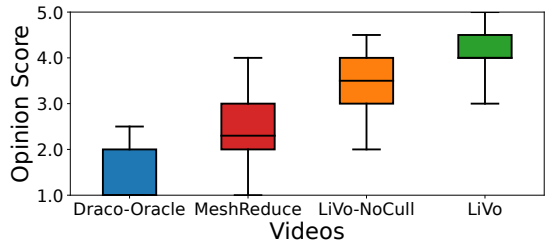


Fig. 5. Aggregated opinion scores for 4 methods.

MeshReduce has a median opinion score of around 2.3 (MOS 2.5), higher than Draco-Oracle. In theory, MeshReduce should perform worse than Draco-Oracle, since it doesn't use a bandwidth oracle. It likely performs better because it uses meshes, which have better perceptual quality than point clouds in general, and because MeshReduce incurs fewer stalls.

By comparison, LiVo-NoCull has a median opinion score of 3.5 (and MOS of 3.4), higher than Draco-Oracle and MeshReduce; its 2D video encoding and *direct* bandwidth-adaptivity lead to higher perceptual quality. LiVo has the highest MOS of 4.1 and median opinion score of 4.0, 14 – 20%

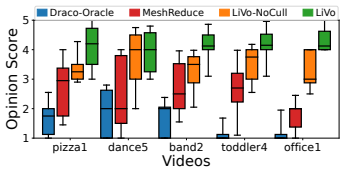


Fig. 6. Opinion scores across 5 videos.

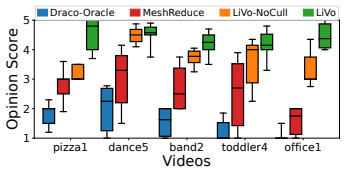


Fig. 7. Opinion scores for *trace-1*.

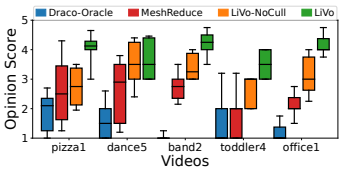


Fig. 8. Opinion scores for *trace-2*.

better than LiVo-NoCull and 64 – 74% better than MeshReduce. Culling enables LiVo to encode at higher quality and incur fewer stalls.

Across 5 videos (Fig. 6), LiVo and LiVo-NoCull perform significantly better than Draco-Oracle. LiVo also outperforms MeshReduce by 48 – 135% in MOS, and LiVo-NoCull by 10 – 33% in MOS. For the *dance5* video, the median opinion score for LiVo and LiVo-NoCull are comparable. *dance5* contains only one person and no objects, so it requires less bandwidth, and culling does not help.

Across our bandwidth traces, quality improves with increasing bandwidth. In *trace-1* (Fig. 7), LiVo's MOS is about 4.3, going up to 4.5 for the *pizza1* video. The performance improvement over LiVo-NoCull is 6.6 – 43.9% (except for *dance5* on trace-1, for the reasons mentioned above). The *direct* bandwidth adaptation in LiVo-NoCull shows 2.0 – 113.3% improvement over MeshReduce, which uses an *indirect* adaptation. When bandwidth is low (*trace-2*, Fig. 8), LiVo's MOS is 3.9; compression at lower bitrates affects both color and geometry. At lower bandwidth, multiple competing factors influence MOS — lower overall quality and higher stalls vs. quality improvement due to culling. These results are consistent with our objective evaluation (§4.3).

Qualitative feedback. Participants were invited to optionally provide free-form feedback. We received 184 responses across 20 participants. We classified these responses into three categories (quality, frame rate, and stalls). For each category, we assigned a high/medium/low rating to each response (Table 5). Descriptions such as "smooth" or "seamless" indicate low stalls and high frame rate, "more distorted" indicate low quality, and "some glitches" indicate medium stalls.

Schemes	Frame Rate (in %)			Stalls (in %)			Quality (in %)		
	L	M	H	L	M	H	L	M	H
Draco-Oracle	94.4	5.6	0.0	0.0	12.5	87.5	35.0	45.0	20.0
MeshReduce	73.3	26.7	0.0	**90.9**	9.1	0.0	61.3	34.1	4.6
LiVo-NoCull	15.0	25.0	60.0	25.0	50.0	25.0	12.5	53.1	34.4
LiVo	0.0	0.0	**100.0**	**70.8**	25.0	4.2	6.1	33.3	**60.6**

Table 5. Percentage of comments providing a Low (L), Medium (M), or High (H) rating for frame rate, stalls, and quality.

Draco-Oracle exhibits most stalls (high=87.5%); every response describes "glitches" or "blanks" for Draco-Oracle. No response indicated high stalls for LiVo; 96% of responses discussing stalls for LiVo agree that it had fewer stalls (low=71%, medium=25%). MeshReduce was rated best in terms of stalls: this is because it conservatively uses a lower frame rate. No response rated it high in terms of frame rate, compared to 100% for LiVo. MeshReduce encodes at a lower quality (only 4.6% of responses rated it high, compared to 60.6% for LiVo). Other alternatives also perform poorly in terms of quality: only 34% of responses rated LiVo-NoCull high, only 20% rated Draco-Oracle high. 95% of the participants said MeshReduce had low to medium quality, with many responses such as "triangles are disturbing", "block of black mass", and "blobs". Users rated LiVo and LiVo-NoCull superior in terms of frame rate to MeshReduce and Draco-Oracle.

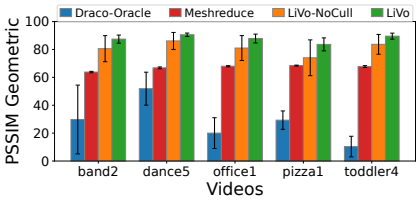


Fig. 9. PSSIM Geometry across 5 videos.

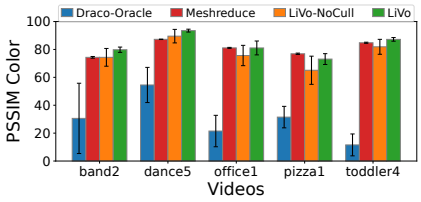


Fig. 10. PSSIM Color across 5 videos.

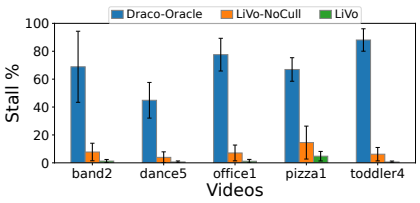


Fig. 11. Stalls in 3 methods across 5 videos.

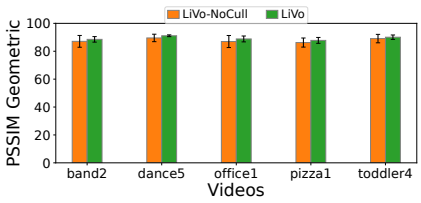


Fig. 12. Effect of culling on PSSIM Geometry, no stalls.

4.3 Objective Quality Comparisons

Setup. We use the same experimental setup as described above but now the receiver logs the received point cloud. We compare each point cloud against ground truth for the same 5 videos, 3 user traces per video and 2 network traces.

Results. Fig. 9 and Fig. 10 separately plot PSSIM for depth (geometry) and color across 4 schemes. For each video sequence, we aggregate the geometry and color PSSIM values over all user traces and network traces. We use a PSSIM of 0 for frames that experience stalls. Draco-Oracle achieves the lowest geometry and color score with mean value of 28.3 (std. 19.1) and 29.9 (std. 19.8), respectively, which correlates with the lower opinion scores in our user study.

LiVo outperforms LiVo-NoCull, achieving a mean PSSIM geometry of 87.8 (std. 3.7) compared to 81.0 LiVo-NoCull's (std. 9.5). This is consistent with our user study and demonstrates the benefits of culling. LiVo's benefits relative to LiVo-NoCull are more pronounced on videos with more subjects (Table 3); in such videos, users often focus on a few subjects at any given instant, so culling is very effective. In most instances, LiVo requires 2× lower bandwidth after encoding compared to LiVo-NoCull, which enables LiVo to transmit at higher quality for the same available bandwidth. MeshReduce has a significantly lower PSSIM geometry of 67.0 (std. 1.8) because it uses Draco, which is less bandwidth efficient, and because it adapts indirectly to bandwidth.

LiVo is slightly better than LiVo-NoCull by the PSSIM color metric (82.9, std. 7.59 vs. 80.9, std. 5.06). This is because our bandwidth splitting allocates only a small fraction of available bandwidth to color, so any bandwidth reductions from culling for color are proportionally lower. Interestingly, MeshReduce compares more favorably with LiVo for color (77.30, std. 10.6), probably because meshes are less affected by depth distortion, a conjecture future work can verify.

Fig. 11 shows the aggregate stalls across videos for 3 schemes. We omit MeshReduce since it doesn't experience stalls: it uses reliable transmissions and its parameters are designed to match average bandwidth, so instead of experiencing stalls, it exhibits varying frame rates. Draco-Oracle suffers from a high mean stall rate of 69.3%. Even on the *dance5* video, which has only 1 participant and no objects in the scene (Table 3), it has a stall rate of 37.8%. LiVo-NoCull incurs an average stall rate of 7.9% (std. 7.5%) while LiVo incurs 1.7% (std. 2.3%); culling accounts for this difference. LiVo's infrequent stalls occur when the rate-adaptive codec overshoots the bandwidth target.

Culling improves quality: LiVo improves over LiVo-NoCull by an average difference of 2% for PSSIM geometry (Fig. 12) and 1% for color (omitted for brevity), without accounting for stalls.

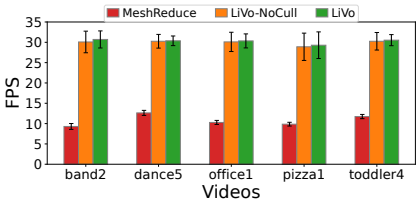


Fig. 13. FPS for *trace-1*.

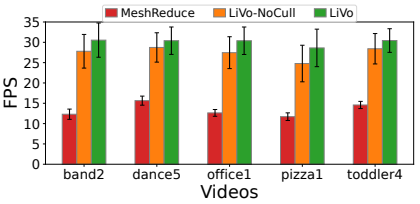


Fig. 14. FPS for *trace-2*.

Guard Band	Prediction Window Size			
	W=5	W=10	W=20	W=30
10	98.44 (0.57)	96.69 (0.54)	94.37 (0.57)	91.76 (0.57)
20	99.26 (0.62)	98.37 (0.62)	96.13 (0.62)	93.95 (0.62)
30	99.61 (0.66)	98.97 (0.66)	97.22 (0.67)	95.41 (0.67)
50	99.89 (0.73)	99.56 (0.73)	98.46 (0.73)	97.18 (0.73)

Fig. 15. Accuracy of culling using Kalman Filter by varying the guard band (in cm) for *band2*. The numbers in brackets represent fraction of points within frustum.

Method	# Hidden Units	Position (m)	Rotation (degree)
MLP	3	0.40	33.34
	32	0.09	3.69
	64	0.07	2.17
Kalman Filter	-	0.04	7.19

Fig. 16. Comparing prediction errors in Kalman Filter-based to learning-based prediction methods.

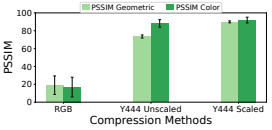


Fig. 17. LiVo with and without depth scaling, and comparison with RGB-based depth encoding [39, 76, 84].

4.4 Performance Validation

Table 6 lists the per-component latency for LiVo and LiVo-NoCull. Through our careful choice of design (§3) and implementation (§4.1), both LiVo and LiVo-NoCull achieve end-to-end latency within 300 ms, in line with production 2D video conferencing systems such as Zoom, Google Meet, *etc.* [32, 53]. Another essential requirement of volumetric video streaming systems is rendering content within the motion-to-photon (MTP) latency (<20 ms) [40, 55]. LiVo can consistently render within 6 ms, easily meeting the MTP requirement. For LiVo, the sender processing latency is around 64 ms while the receiver processing latency is 53 ms. In contrast, LiVo-NoCull incurs lower processing latency at the sender since it does not perform view culling, but requires culling at the receiver. The largest latency is incurred by WebRTC transmission (color and depth transmissions are parallel), which is about 137 ms. Much of this is attributable to the jitter buffer in WebRTC: we use 100 ms [81], consistent with current practice.

Component	LiVo-NoCull (in ms)		LiVo (in ms)	
	Mean	Std	Mean	Std
Capture Frame	32.75	6.81	29.85	5.09
View Generation	19.50	8.93	30.59	2.91
Tile Creation	7.16	5.08	5.49	0.86
Synchronization	31.74	14.24	32.33	4.60
Point cloud Reconstruction	21.80	2.47	15.83	3.64
Rendering	5.16	1.32	4.91	1.23
WebRTC Send-Recv	133.19	2.13	132.82	0.65
Total latency	**251.30**	**23.32**	**251.82**	**6.93**

Table 6. Component-wise latency.

Fig. 13 plots the rendering frame rate for LiVo-NoCull and LiVo for *trace-1*. LiVo maintains 30 fps with a lower standard deviation than LiVo-NoCull. For *trace-2* (Fig. 14), LiVo maintains close to 30 fps (std. 0.7), while LiVo-NoCull achieves 28 fps (std. 1.8). At lower bandwidth, LiVo-NoCull experiences more stalls (mean fps drops to 24 fps for *pizza1*) than LiVo because compressed non-culled frames occasionally exceed the bandwidth budget. MeshReduce achieves a mean frame rate of 12.1 fps, 2.5x lower than LiVo, despite fully utilizing all cores on the sender to encode frames. MeshReduce's frame rate for *trace-2* is slightly higher than *trace-1*, because it decimates the mesh more to fit the lower bandwidth, and the encoder needs to perform less work.

4.5 Validating Design Choices

Depth Encoding. Unlike prior work which has encoded depth in RGB images, LiVo encodes scaled depth in the 16-bit Y-channel of an H.265 mode. Fig. 17 shows that LiVo's scaled depth encoding outperforms the RGB-based encoding used in [39, 76, 84] and unscaled depth encoding.

Kalman Filter buffer size. LiVo uses a static guard-band ϵ of 20 cm around the frustum to minimize quality loss due to prediction errors. Fig. 15 shows that for guard-bands up to 30 cm,

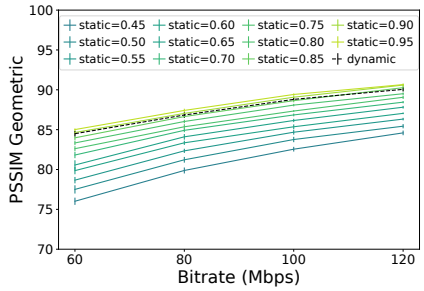


Fig. 18. PSSIM Geometric for static vs dynamic split.

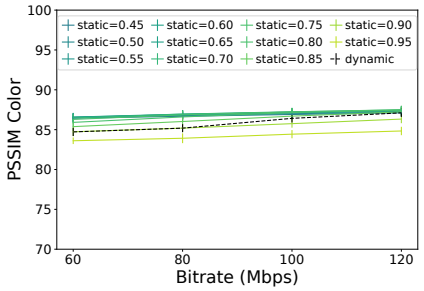


Fig. 19. PSSIM Color for static vs dynamic split.

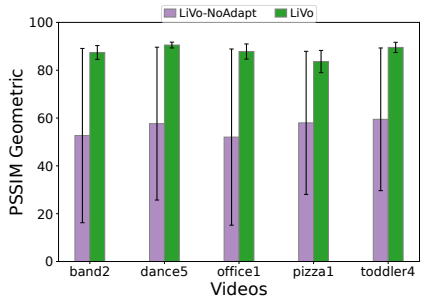


Fig. 20. PSSIM Geometric, LiVo-NoAdapt vs LiVo.

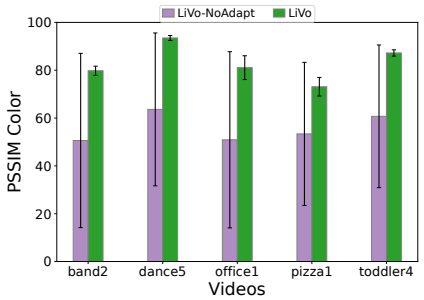


Fig. 21. PSSIM Color, LiVo-NoAdapt vs LiVo.

LiVo's accuracy and data volume are relatively insensitive to the choice of guard-band, hence a static guard-band works well. We have verified this for other videos as well.

Bandwidth Splitting. Fig. 18 compares PSSIM for different static splits with LiVo's dynamic split (§3.3) for bitrates ranging from 60-120 Mbps for *office1* (similarly, color in Fig. 19). Dynamic splitting achieves quality within 0.5 PSSIM points for geometry and 3 PSSIM points for color relative to the best static split. With higher available bandwidth, both geometry and color PSSIM values are within 0.5 of the best static split, demonstrating the effectiveness of dynamic splitting.

Bandwidth Adaptation. What if LiVo did not use bandwidth adaptation and view culling at all, but instead used fixed quality parameters? We set fixed color QP (quantization parameter) to 22 and depth QP to 14. Starline [51] uses these values. Fig. 20 and Fig. 21 show that this LiVo-NoAdapt's quality drops by $30-41\%$ for geometry and $27-37\%$ for color, with PSSIM going below 60.

Frustum Prediction. To estimate quality impairment caused by our predictive culling, we compare it against LiVo with perfect culling (using predictions obtained from the user trace). The average quality difference in PSSIM geometry and color is 1% across all 5 videos when run on *trace-1*.

Prior work has trained frustum predictors from user traces [40, 62] for on-demand volumetric videos. We evaluate (Fig. 16) whether an MLP with 3 hidden layers used in ViVo [40] could learn effectively from a small number of our traces. We find that it may be possible to match LiVo's predictor only with 64 hidden layers. With 3 layers, prediction errors are unacceptably high.

5 Conclusions

LiVo enables bandwidth-adaptive conferencing of full-scene volumetric video by maximally leveraging 2D video compression, rate-adaptive codecs, and real-time transport. Both user studies and objective quality assessments show that it outperforms baselines while maintaining its performance goals. Future work can explore better frustum prediction techniques, extend the work to support multi-way conferencing, devise additional techniques to be robust to packet losses, extend bandwidth splitting to 2D video, and adapt LiVo for mobile devices.

References

[1] [n. d.]. Azure-Kinect-Sensor-SDK. https://github.com/microsoft/Azure-Kinect-Sensor-SDK.
[2] [n. d.]. Boost C++ Libraries. https://www.boost.org/.
[3] [n. d.]. Chrony. https://chrony-project.org/.
[4] [n. d.]. Draco 3D Compression. https://github.com/google/draco.
[5] [n. d.]. Eigen. https://eigen.tuxfamily.org/.
[6] [n. d.]. FFmpeg. https://ffmpeg.org/.
[7] [n. d.]. GStreamer. https://gstreamer.freedesktop.org/.
[8] [n. d.]. NVIDIA Video Codec SDK. https://developer.nvidia.com/video-codec-sdk.
[9] [n. d.]. OpenMP. https://www.openmp.org/.
[10] [n. d.]. PanopticStudio Toolbox. https://github.com/CMU-Perceptual-Computing-Lab/panoptic-toolbox.
[11] [n. d.]. Point Cloud Library. https://pointclouds.org/.
[12] [n. d.]. YUV. https://en.wikipedia.org/wiki/YUV.
[13] 2020. Intel RealSense Depth Camera D455. https://www.intelrealsense.com/depth-camera-d455/.
[14] 2020. Synchronize multiple Azure Kinect DK devices. https://learn.microsoft.com/en-us/azure/kinect-dk/multi-camera-sync.
[15] 2021. Azure Kinect DK hardware specifications. https://learn.microsoft.com/en-us/azure/kinect-dk/hardware-specification.
[16] 2022. Cisco Annual Internet Report (2018–2023) White Paper. https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html.
[17] 2022. Depthkit Holoportation via Webcam. https://www.depthkit.tv/tutorials/depthkit-holoportation-via-webcam.
[18] 2024. Azure Kinect DK. https://azure.microsoft.com/en-us/products/kinect-dk.
[19] 2024. Intel RealSense Depth Camera D457. https://www.intel.com/content/www/us/en/products/sku/230571/intel-realsense-depth-camera-d457/specifications.html.
[20] 2025. nvh265enc. https://gstreamer.freedesktop.org/documentation/nvcodec/nvh265enc.html.
[21] 2025. x265enc. https://gstreamer.freedesktop.org/documentation/x265/index.html.
[22] Evangelos Alexiou and Touradj Ebrahimi. 2020. Towards a Point Cloud Structural Similarity Metric. In *2020 IEEE International Conference on Multimedia & Expo Workshops (ICMEW)*. 1–6. doi:10.1109/ICMEW46912.2020.9106005
[23] Niklas Blum, Serge Lachapelle, and Harald Alvestrand. 2021. WebRTC - Realtime Communication for the Open Web Platform: What was once a way to bring audio and video to the web has expanded into more use cases we could ever imagine. *Queue* 19, 1 (mar 2021), 77–93. doi:10.1145/3454122.3457587
[24] Gaetano Carlucci, Luca De Cicco, Stefan Holmer, and Saverio Mascolo. 2016. Analysis and design of the google congestion control for web real-time communication (WebRTC). In *Proceedings of the 7th International Conference on Multimedia Systems* (Klagenfurt, Austria) *(MMSys '16)*. Association for Computing Machinery, New York, NY, USA, Article 13, 12 pages. doi:10.1145/2910017.2910605
[25] Jacob Chakareski. 2013. Adaptive multiview video streaming: challenges and opportunities. *IEEE Communications Magazine* 51, 5 (2013), 94–100. doi:10.1109/MCOM.2013.6515052
[26] Ruopeng Chen, Mengbai Xiao, Dongxiao Yu, Guanghui Zhang, and Yao Liu. 2023. patchVVC: A Real-time Compression Framework for Streaming Volumetric Videos. In *Proceedings of the 14th Conference on ACM Multimedia Systems*. 119–129.
[27] Alvaro Collet, Ming Chuang, Pat Sweeney, Don Gillett, Dennis Evseev, David Calabrese, Hugues Hoppe, Adam Kirk, and Steve Sullivan. 2015. High-quality streamable free-viewpoint video. *ACM Transactions on Graphics (ToG)* 34, 4 (2015), 1–13.
[28] Sandesh Dhawaskar Sathyanarayana, Kyunghan Lee, Dirk Grunwald, and Sangtae Ha. 2023. Converge: QoE-driven Multipath Video Conferencing over WebRTC. In *Proceedings of the ACM SIGCOMM 2023 Conference* (New York, NY, USA) *(ACM SIGCOMM '23)*. Association for Computing Machinery, New York, NY, USA, 637–653. doi:10.1145/3603269.3604822
[29] Mingsong Dou, Sameh Khamis, Yury Degtyarev, Philip Davidson, Sean Ryan Fanello, Adarsh Kowdle, Sergio Orts Escolano, Christoph Rhemann, David Kim, Jonathan Taylor, et al. 2016. Fusion4d: Real-time performance capture of challenging scenes. *ACM Transactions on Graphics (ToG)* 35, 4 (2016), 1–13.
[30] Mathis Engelbart, Joerg Ott, and Spencer Dawkins. 2025. *RTP over QUIC (RoQ)*. Internet-Draft draft-ietf-avtcore-rtp-over-quic-14. Internet Engineering Task Force. https://datatracker.ietf.org/doc/draft-ietf-avtcore-rtp-over-quic/14/ Work in Progress.
[31] Dinei Florencio and Cha Zhang. 2009. Multiview video compression and streaming based on predicted viewer position. In *2009 IEEE International Conference on Acoustics, Speech and Signal Processing*. 657–660. doi:10.1109/ICASSP.2009.4959669

[32] Sadjad Fouladi, John Emmons, Emre Orbay, Catherine Wu, Riad S. Wahby, and Keith Winstein. 2018. Salsify: Low-Latency Network Video through Tighter Integration between a Video Codec and a Transport Protocol. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. USENIX Association, Renton, WA, 267–282. https://www.usenix.org/conference/nsdi18/presentation/fouladi

[33] Yongying Gao, Yuwen Wu, and Ying Chen. 2009. H.264/Advanced Video Coding (AVC) Backward-Compatible Bit-Depth Scalable Coding. *IEEE Transactions on Circuits and Systems for Video Technology* 19, 4 (2009), 500–510. doi:10.1109/TCSVT.2009.2014018

[34] Ehab Ghabashneh, Chandan Bothra, Ramesh Govindan, Antonio Ortega, and Sanjay Rao. 2023. Dragonfly: Higher Perceptual Quality For Continuous 360° Video Playback. In *Proceedings of the ACM SIGCOMM 2023 Conference* (New York, NY, USA) *(ACM SIGCOMM '23)*. Association for Computing Machinery, New York, NY, USA, 516–532. doi:10.1145/3603269.3604876

[35] Sharath Girish, Tianye Li, Amrita Mazumdar, Abhinav Shrivastava, david luebke, and Shalini De Mello. 2024. QUEEN: QUantized Efficient ENcoding for Streaming Free-viewpoint Videos. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*. https://openreview.net/forum?id=7xhwE7VH4S

[36] Danillo Graziosi, Ohji Nakagami, Satoru Kuma, Alexandre Zaghetto, Teruhiko Suzuki, and Ali Tabatabai. 2020. 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 (2020), e13. doi:10.1017/ATSIP.2020.12

[37] Yongjie Guan, Xueyu Hou, Nan Wu, Bo Han, and Tao Han. 2023. MetaStream: Live Volumetric Content Capture, Creation, Delivery, and Rendering in Real Time. In *Proceedings of the 29th Annual International Conference on Mobile Computing and Networking*. Association for Computing Machinery, New York, NY, USA, Article 29, 15 pages. https://doi.org/10.1145/3570361.3592530

[38] Serhan Gül, Sebastian Bosse, Dimitri Podborski, Thomas Schierl, and Cornelius Hellge. 2020. Kalman Filter-based Head Motion Prediction for Cloud-based Mixed Reality. In *Proceedings of the 28th ACM International Conference on Multimedia* (Seattle, WA, USA) *(MM '20)*. Association for Computing Machinery, New York, NY, USA, 3632–3641. doi:10.1145/3394171.3413699

[39] Simon N. B. Gunkel, Rick Hindriks, Karim M. El Assal, Hans M. Stokking, Sylvie Dijkstra-Soudarissanane, Frank ter Haar, and Omar Niamut. 2021. VRComm: an end-to-end web system for real-time photorealistic social VR communication. In *Proceedings of the 12th ACM Multimedia Systems Conference* (Istanbul, Turkey) *(MMSys '21)*. Association for Computing Machinery, New York, NY, USA, 65–79. doi:10.1145/3458305.3459595

[40] 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. doi:10.1145/3372224.3380888

[41] Raphael Hauser. 2007. Line search methods for unconstrained optimisation. *Lecture 8, Numerical Linear Algebra and Optimisation Oxford University Computing Laboratory* (2007).

[42] Haoran Hong, Eduardo Pavez, Antonio Ortega, Ryosuke Watanabe, and Keisuke Nonaka. 2022. Fractional motion estimation for point cloud compression. In *2022 Data Compression Conference (DCC)*. 369–378.

[43] Haoran Hong, Eduardo Pavez, Antonio Ortega, Ryosuke Watanabe, and Keisuke Nonaka. 2022. Motion estimation and filtered prediction for dynamic point cloud attribute compression. In *2022 Picture Coding Symposium (PCS)*. 139–143.

[44] Intel. 2019. Intel RealSense LiDAR Camera L515. https://www.intelrealsense.com/lidar-camera-l515/.

[45] Tao Jin, Mallesham Dasa, Connor Smith, Kittipat Apicharttrisorn, Srinivasan Seshan, and Anthony Rowe. 2024. MeshReduce: Scalable and Bandwidth Efficient 3D Scene Capture. In *2024 IEEE Conference Virtual Reality and 3D User Interfaces (VR)*. 20–30. doi:10.1109/VR58804.2024.00026

[46] 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. 2017. Panoptic Studio: A Massively Multiview System for Social Interaction Capture. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2017).

[47] Rudolf E. Kalman. 1960. A New Approach to Linear Filtering and Prediction Problems. *Journal of Basic Engineering* 82, 1 (03 1960), 35–45. arXiv:https://asmedigitalcollection.asme.org/fluidsengineering/article-pdf/82/1/35/5518977/35_1.pdf doi:10.1115/1.3662552

[48] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 2023. 3D Gaussian Splatting for Real-Time Radiance Field Rendering. *ACM Transactions on Graphics* 42, 4 (July 2023). https://repo-sam.inria.fr/fungraph/3d-gaussian-splatting/

[49] Woo-Shik Kim, Antonio Ortega, PoLin Lai, and Dong Tian. 2015. Depth Map Coding Optimization Using Rendered View Distortion for 3D Video Coding. *IEEE Transactions on Image Processing* 24, 11 (2015), 3534–3545. doi:10.1109/TIP.2015.2447737

[50] Marek Kowalski, Jacek Naruniec, and Michal Daniluk. 2015. Livescan3D: A Fast and Inexpensive 3D Data Acquisition System for Multiple Kinect v2 Sensors. In *2015 International Conference on 3D Vision*. 318–325. doi:10.1109/3DV.2015.43

[51] Jason Lawrence, Danb Goldman, Supreeth Achar, Gregory Major Blascovich, Joseph G. Desloge, Tommy Fortes, Eric M. Gomez, Sascha Häberling, Hugues Hoppe, Andy Huibers, Claude Knaus, Brian Kuschak, Ricardo Martin-Brualla, Harris Nover, Andrew Ian Russell, Steven M. Seitz, and Kevin Tong. 2021. Project Starline: A High-Fidelity Telepresence System. *ACM Trans. Graph.* 40, 6, Article 242 (dec 2021), 16 pages. doi:10.1145/3478513.3480490

[52] Insoo Lee, Seyeon Kim, Sandesh Sathyanarayana, Kyungmin Bin, Song Chong, Kyunghan Lee, Dirk Grunwald, and Sangtae Ha. 2022. R-FEC: RL-based FEC Adjustment for Better QoE in WebRTC. In *Proceedings of the 30th ACM International Conference on Multimedia* (Lisboa, Portugal) *(MM '22)*. Association for Computing Machinery, New York, NY, USA, 2948–2956. doi:10.1145/3503161.3548370

[53] Insoo Lee, Jinsung Lee, Kyunghan Lee, Dirk Grunwald, and Sangtae Ha. 2021. Demystifying Commercial Video Conferencing Applications. In *Proceedings of the 29th ACM International Conference on Multimedia* (Virtual Event, China) *(MM '21)*. Association for Computing Machinery, New York, NY, USA, 3583–3591. doi:10.1145/3474085.3475523

[54] Kyungjin Lee, Juheon Yi, and Youngki Lee. 2023. FarfetchFusion: Towards Fully Mobile Live 3D Telepresence Platform. In *Proceedings of the 29th Annual International Conference on Mobile Computing and Networking*. Association for Computing Machinery, New York, NY, USA, Article 20, 15 pages. https://doi.org/10.1145/3570361.3592525

[55] 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* (London, United Kingdom) *(MobiCom '20)*. Association for Computing Machinery, New York, NY, USA, Article 57, 14 pages. doi:10.1145/3372224.3419214

[56] Boyan Li, Yongting Chen, Dayou Zhang, and Fangxin Wang. 2024. DeformStream: Deformation-based Adaptive Volumetric Video Streaming. arXiv:2409.16615 [cs.CV] https://arxiv.org/abs/2409.16615

[57] Lingzhi Li, Zhen Shen, Zhongshu Wang, Li Shen, and Ping Tan. 2022. Streaming radiance fields for 3d video synthesis. *Advances in Neural Information Processing Systems* 35 (2022), 13485–13498.

[58] Zhuqi Li, Yaxiong Xie, Ravi Netravali, and Kyle Jamieson. 2023. Dashlet: Taming Swipe Uncertainty for Robust Short Video Streaming. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. USENIX Association, Boston, MA, 1583–1599. https://www.usenix.org/conference/nsdi23/presentation/li-zhuqi

[59] Zhuqi Li, Yaxiong Xie, Longfei Shangguan, Rotman Ivan Zelaya, Jeremy Gummeson, Wenjun Hu, and Kyle Jamieson. 2019. Towards Programming the Radio Environment with Large Arrays of Inexpensive Antennas. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. USENIX Association, Boston, MA, 285–300. https://www.usenix.org/conference/nsdi19/presentation/lizhuqi

[60] Zhicheng Liang, Junhua Liu, Mallesham Dasari, and Fangxin Wang. 2024. Fumos: Neural Compression and Progressive Refinement for Continuous Point Cloud Video Streaming. *IEEE Transactions on Visualization and Computer Graphics* 30, 5 (2024), 2849–2859. doi:10.1109/TVCG.2024.3372096

[61] Bangya Liu and Suman Banerjee. 2024. SwinGS: Sliding Window Gaussian Splatting for Volumetric Video Streaming with Arbitrary Length. arXiv:2409.07759 [cs.MM] https://arxiv.org/abs/2409.07759

[62] Yu Liu, Bo Han, Feng Qian, Arvind Narayanan, and Zhi-Li Zhang. 2022. Vues: practical mobile volumetric video streaming through multiview transcoding. In *Proceedings of the 28th Annual International Conference on Mobile Computing And Networking* (Sydney, NSW, Australia) *(MobiCom '22)*. Association for Computing Machinery, New York, NY, USA, 514–527. doi:10.1145/3495243.3517027

[63] Philipp Merkle, Karsten Müller, and Thomas Wiegand. 2010. 3D video: acquisition, coding, and display. *IEEE Transactions on Consumer Electronics* 56, 2 (2010), 946–950. doi:10.1109/TCE.2010.5506024

[64] Gabriel Meynet, Yana Nehmé, Julie Digne, and Guillaume Lavoué. 2020. PCQM: A Full-Reference Quality Metric for Colored 3D Point Clouds. In *2020 Twelfth International Conference on Quality of Multimedia Experience (QoMEX)*. 1–6. doi:10.1109/QoMEX48832.2020.9123147

[65] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. 2020. NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis. In *ECCV*.

[66] Karsten Müller, Philipp Merkle, and Thomas Wiegand. 2011. 3-D Video Representation Using Depth Maps. *Proc. IEEE* 99, 4 (2011), 643–656. doi:10.1109/JPROC.2010.2091090

[67] Ravi Netravali, Anirudh Sivaraman, Somak Das, Ameesh Goyal, Keith Winstein, James Mickens, and Hari Balakrishnan. 2015. Mahimahi: Accurate Record-and-Replay for HTTP. In *2015 USENIX Annual Technical Conference (USENIX ATC 15)*. USENIX Association, Santa Clara, CA, 417–429. https://www.usenix.org/conference/atc15/technical-session/presentation/netravali

[68] Richard A. Newcombe, Dieter Fox, and Steven M. Seitz. 2015. DynamicFusion: Reconstruction and tracking of non-rigid scenes in real-time. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 343–352. doi:10.1109/CVPR.2015.7298631

[69] Richard A Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J Davison, Pushmeet Kohi, Jamie Shotton, Steve Hodges, and Andrew Fitzgibbon. 2011. Kinectfusion: Real-time dense surface mapping and tracking. In *2011 10th IEEE international symposium on mixed and augmented reality*. Ieee, 127–136.

[70] Jorge Nocedal and Stephen J. Wright (Eds.). 1999. *Line Search Methods.* Springer New York, New York, NY, 34–63. doi:10.1007/0-387-22742-3_3

[71] NVIDIA. 2023. NVENC Video Encoder API Programming Guide. https://docs.nvidia.com/video-technologies/video-codec-sdk/12.1/nvenc-video-encoder-api-prog-guide/index.html.

[72] NVIDIA. 2024. Video Encode and Decode GPU Support Matrix. https://developer.nvidia.com/video-encode-and-decode-gpu-support-matrix-new.

[73] ORBBEC. 2023. Astra 2. https://www.orbbec.com/products/structured-light-camera/astra-2/.

[74] Sergio Orts-Escolano, Christoph Rhemann, Sean Fanello, Wayne Chang, Adarsh Kowdle, Yury Degtyarev, David Kim, Philip L. Davidson, Sameh Khamis, Mingsong Dou, Vladimir Tankovich, Charles Loop, Qin Cai, Philip A. Chou, Sarah Mennicken, Julien Valentin, Vivek Pradeep, Shenlong Wang, Sing Bing Kang, Pushmeet Kohli, Yuliya Lutchyn, Cem Keskin, and Shahram Izadi. 2016. Holoportation: Virtual 3D Teleportation in Real-Time. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology* (Tokyo, Japan) *(UIST '16)*. Association for Computing Machinery, New York, NY, USA, 741–754. doi:10.1145/2984511.2984517

[75] Ouster. 2025. Overview of our OS sensors. https://ouster.com/os-overview.

[76] Fabrizio Pece, Jan Kautz, and Tim Weyrich. 2011. Adapting standard video codecs for depth streaming. In *Proceedings of the 17th Eurographics Conference on Virtual Environments & Third Joint Virtual Reality* (Nottingham, UK) *(EGVE - JVRC'11)*. Eurographics Association, Goslar, DEU, 59–66.

[77] PixelTools. 2022. Rate Control and H.264. https://www.pixeltools.com/rate_control_paper.html.

[78] Feng Qian, Bo Han, Qingyang Xiao, and Vijay Gopalakrishnan. 2018. Flare: Practical Viewport-Adaptive 360-Degree Video Streaming for Mobile Devices. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking* (New Delhi, India) *(MobiCom '18)*. Association for Computing Machinery, New York, NY, USA, 99–114. doi:10.1145/3241539.3241565

[79] ROS-Industrial. 2025. A Comprehensive List of 3D Sensors Commonly Leveraged in ROS Development. https://rosindustrial.org/3d-camera-survey.

[80] Sebastian Schwarz, Marius Preda, Vittorio Baroncini, Madhukar Budagavi, Pablo Cesar, Philip A. Chou, Robert A. Cohen, Maja Krivokuća, Sébastien Lasserre, Zhu Li, Joan Llach, Khaled Mammou, Rufael Mekuria, Ohji Nakagami, Ernestasia Siahaan, Ali Tabatabai, Alexis M. Tourapis, and Vladyslav Zakharchenko. 2019. Emerging MPEG Standards for Point Cloud Compression. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 9, 1 (2019), 133–148. doi:10.1109/JETCAS.2018.2885981

[81] Taveesh Sharma, Tarun Mangla, Arpit Gupta, Junchen Jiang, and Nick Feamster. 2023. Estimating WebRTC Video QoE Metrics Without Using Application Headers. In *Proceedings of the 2023 ACM on Internet Measurement Conference* (Montreal QC, Canada) *(IMC '23)*. Association for Computing Machinery, New York, NY, USA, 485–500. doi:10.1145/3618257.3624828

[82] Jianxin Shi, Miao Zhang, Linfeng Shen, Jiangchuan Liu, Yuan Zhang, Lingjun Pu, and Jingdong Xu. 2024. Towards Full-scene Volumetric Video Streaming via Spatially Layered Representation and NeRF Generation. In *Proceedings of the 34th Edition of the Workshop on Network and Operating System Support for Digital Audio and Video* (Bari, Italy) *(NOSSDAV '24)*. Association for Computing Machinery, New York, NY, USA, 22–28. doi:10.1145/3651863.3651879

[83] Stephen Siemonsma and Tyler Bell. 2022. HoloKinect: Holographic 3D Video Conferencing. *Sensors* 22, 21 (2022). doi:10.3390/s22218118

[84] Tetsuri Sonoda and Anders Grunnet-Jepsen. 2025. Depth image compression by colorization for Intel RealSense Depth Cameras. https://dev.intelrealsense.com/docs/depth-image-compression-by-colorization-for-intel-realsense-depth-cameras.

[85] Jiakai Sun, Han Jiao, Guangyuan Li, Zhanjie Zhang, Lei Zhao, and Wei Xing. 2024. 3DGStream: On-the-Fly Training of 3D Gaussians for Efficient Streaming of Photo-Realistic Free-Viewpoint Videos. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 20675–20685.

[86] Yuan-Chun Sun, Yuang Shi, Wei Tsang Ooi, Chun-Ying Huang, and Cheng-Hsin Hsu. 2024. Multi-frame Bitrate Allocation of Dynamic 3D Gaussian Splatting Streaming Over Dynamic Networks. In *Proceedings of the 2024 SIGCOMM Workshop on Emerging Multimedia Systems* (Sydney, NSW, Australia) *(EMS '24)*. Association for Computing Machinery, New York, NY, USA, 1–7. doi:10.1145/3672196.3673394

[87] Mapix technologies. 2025. Velodyne 3D LiDAR sensors. https://www.mapix.com/lidar-scanner-sensors/velodyne/.

[88] Dong Tian, Hideaki Ochimizu, Chen Feng, Robert Cohen, and Anthony Vetro. 2017. Geometric distortion metrics for point cloud compression. In *2017 IEEE International Conference on Image Processing (ICIP)*. 3460–3464. doi:10.1109/ICIP.2017.8296925

[89] Hanzhang Tu, Ruizhi Shao, Xue Dong, Shunyuan Zheng, Hao Zhang, Lili Chen, Meili Wang, Wenyu Li, Siyan Ma, Shengping Zhang, Boyao Zhou, and Yebin Liu. 2024. Tele-Aloha: A Telepresence System with Low-budget and High-authenticity Using Sparse RGB Cameras. In *ACM SIGGRAPH 2024 Conference Papers* (Denver, CO, USA) *(SIGGRAPH '24)*. Association for Computing Machinery, New York, NY, USA, Article 116, 12 pages. doi:10.1145/3641519.3657491

[90] Shengze Wang, Ziheng Wang, Ryan Schmelzle, Liujie Zheng, YoungJoong Kwon, Roni Sengupta, and Henry Fuchs. 2024. Learning View Synthesis for Desktop Telepresence with Few RGBD Cameras. *IEEE Transactions on Visualization and Computer Graphics* (2024), 1–13. doi:10.1109/TVCG.2024.3411626

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

[92] Zhe Wang and Yifei Zhu. 2024. Towards Real-Time Neural Volumetric Rendering on Mobile Devices: A Measurement Study. In *Proceedings of the 2024 SIGCOMM Workshop on Emerging Multimedia Systems* (Sydney, NSW, Australia) *(EMS '24)*. Association for Computing Machinery, New York, NY, USA, 8–13. doi:10.1145/3672196.3673399

[93] Andrew D. Wilson. 2017. Fast Lossless Depth Image Compression. In *Proceedings of the 2017 ACM International Conference on Interactive Surfaces and Spaces* (Brighton, United Kingdom) *(ISS '17)*. Association for Computing Machinery, New York, NY, USA, 100–105. doi:10.1145/3132272.3134144

[94] Mengyu Yang, Zhenxiao Luo, Miao Hu, Min Chen, and Di Wu. 2023. A Comparative Measurement Study of Point Cloud-Based Volumetric Video Codecs. *IEEE Transactions on Broadcasting* 69, 3 (2023), 715–726. doi:10.1109/TBC.2023.3243407

[95] Qi Yang, Zhan Ma, Yiling Xu, Zhu Li, and Jun Sun. 2022. Inferring Point Cloud Quality via Graph Similarity. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 44, 6 (2022), 3015–3029. doi:10.1109/TPAMI.2020.3047083

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

[97] Zhengyou Zhang. 2000. A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22, 11 (2000), 1330–1334. doi:10.1109/34.888718

[98] Junquan Zhong, Haodan Zhang, Quanlu Jia, Jiangkai Wu, Peiheng Wang, Haoyang Wang, Liming Liu, Xinggong Zhang, and Zongming Guo. 2024. Low-bitrate Volumetric Video Streaming with Depth Image. In *Proceedings of the 2024 SIGCOMM Workshop on Emerging Multimedia Systems* (Sydney, NSW, Australia) *(EMS '24)*. Association for Computing Machinery, New York, NY, USA, 39–44. doi:10.1145/3672196.3673397

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

A Appendix

A.1 Other Details

Our LiVo implementation combines techniques from various sources to achieve high-quality, low-latency volumetric video streaming. We describe these briefly.

Stream Synchronization. WebRTC does not permit embedding frame numbers in video streams. Following prior work [32], the LiVo sender embeds a (pre-generated) QR code in each 4K depth and color tiled frame that encodes the frame sequence number (Fig. 3). The receiver decodes the QR code to obtain frame sequence numbers. If both depth and color frames have not been decoded by the time necessary to render the point cloud, LiVo simply skips the frame.

Receiver-side rendering. The receiver must reconstruct point clouds from received 4K frames. To do this, it needs the parameters and positions of the RGB-D cameras; these are exchanged once during connection setup or whenever they change. The receiver uses these to transform the position of each pixel from each 4K frame into the global coordinate frame. Together with the corresponding color attributes, these constitute the reconstructed point cloud at the receiver. This point cloud may be more dense than necessary and may have more points because LiVo transmits a guard-band around the frustum (§3.4). To speed up rendering, LiVo first voxelizes the point cloud, then further culls the point cloud to the actual (current) frustum, following prior work [40, 55].

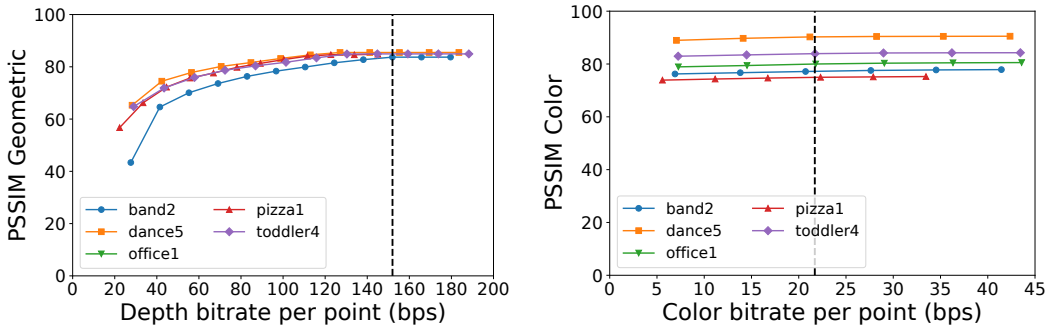
Pipelining and parallelism. To ensure frame-rate processing, LiVo consists of several stages that run in parallel, and each stage incurs a delay per frame of less than one inter-frame interval. The sender has four stages: capture, view generation, tiling, and transmission. The receiver also has three stages: receiving and synchronization, point cloud reconstruction, and rendering. Thus, the total end-to-end *processing* latency is within 180 ms. Each stage has a dedicated thread and is connected to the next stage via a small inter-stage buffer (implemented using a thread-safe queue). We also employ parallelism within some stages when possible: *e.g.*, view generation and point cloud generation.

Fig. A.1. Block artifacts arising from unscaled depth encoding when we use naive H.265 YUV 16-bit variant.

Minimizing the impact of packet loss. We enable several WebRTC features, including negative acknowledgments, Picture Loss Indication (PLI), and Full Intraframe Request (FIR). Because 4K videos are large, the default Linux UDP socket buffer (213 KB) proved insufficient, so we increased it.

A.2 Depth vs Color Distortion

Fig. A.2 studies the variability in PSSIM quality metric in depth (similarly for color) when we fix color (similarly for depth) bitrate and vary depth bitrate. Fig. A.2a shows how PSSIM Geometry metric varies when we increase depth bitrate (normalized as depth bitrate per point); Fig. A.2b similarly reports PSSIM color. We observe that depth quality improves significantly with increased bitrate before it flattens, while the variation in color quality is minimal. Also, the bitrate that needs to be allocated for depth is almost 7× higher before it saturates (around the vertical dotted line). This confirms that depth is much more sensitive to bitrate than color and requires careful bitrate allocation.



(A) Depth distortion vs depth bitrate.

(B) Color distortion vs color bitrate.

FIG. A.2. PSSIM distortion across videos when bitrate is varied

A.3 Variability in Bandwidth Traces

We show the variability in the two traces we use - *trace-1* and *trace-2* in Fig. A.3.

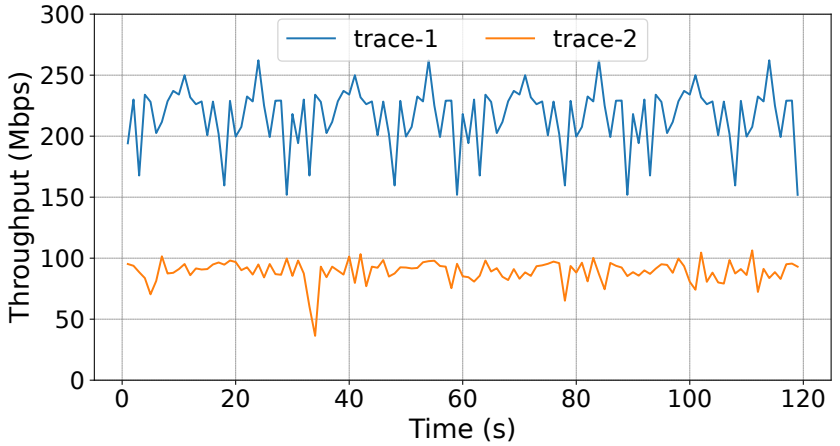


FIG. A.3. Variability in the 2 network traces we consider.

Received June 2025; accepted September 2025