ViVo: Visibility-Aware Mobile Volumetric Video Streaming

Bo Han
AT&T Labs Research
Bedminster, NJ
bohan@research.att.com

Yu Liu
University of Minnesota, Twin Cities
Minneapolis, MN
liu00885@umn.edu

Feng Qian
University of Minnesota, Twin Cities
Minneapolis, MN
fengqian@umn.edu

ABSTRACT

In this paper, we perform a first comprehensive study of mobile volumetric video streaming. Volumetric videos are truly 3D, allowing six degrees of freedom (6DoF) movement for their viewers during playback. Such flexibility enables numerous applications in entertainment, healthcare, education, *etc.* However, volumetric video streaming is extremely bandwidth-intensive. We conduct a detailed investigation of each of the following aspects for point cloud streaming (a popular volumetric data format): encoding, decoding, segmentation, viewport movement patterns, and viewport prediction. Motivated by the observations from the above study, we propose ViVo, which is to the best of our knowledge the first practical mobile volumetric video streaming system with three *visibility-aware* optimizations. ViVo judiciously determines the video content to fetch based on how, what and where a viewer perceives for reducing bandwidth consumption of volumetric video streaming. Our evaluations over real wireless networks (including commercial 5G), mobile devices and users indicate that ViVo can save on average 40% of data usage (up to 80%) with virtually no drop in visual quality.

ACM Reference Format:
Bo Han, Yu Liu, and Feng Qian. 2020. ViVo: Visibility-Aware Mobile Volumetric Video Streaming. In *The 26th Annual International Conference on Mobile Computing and Networking (MobiCom '20), September 21–25, 2020, London, United Kingdom.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3372224.3380888

1 INTRODUCTION

Recent advances in wireless technology such as mmWave 5G have fueled a wide range of emerging applications. Among them, immersive video streaming plays an extremely important role. In this paper, we study a new type of video content called *Volumetric Video*[1]. Volumetric video streaming is one of the enabling technologies for mixed reality (MR) [4], and will become a key application of 5G [3, 15]. According to a recent market research report [22], it is expected that the volumetric video market will grow from $578 million in 2018 to $2.78 billion by 2023. Major video content providers such as Google [7] and Facebook [21] have started to investigate commercializing volumetric video streaming.

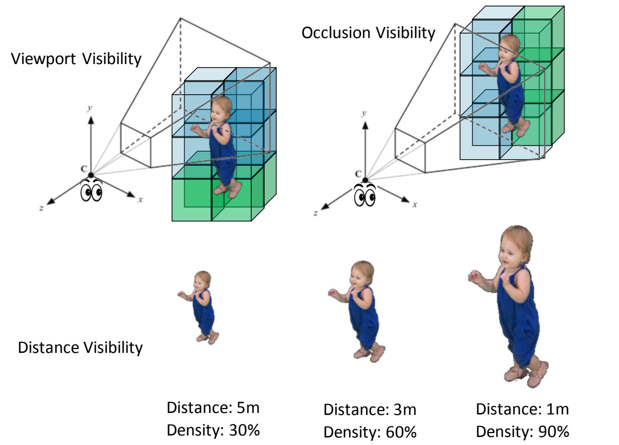

Figure 1: Illustrative examples of the three visibility-aware optimizations in ViVo. The content in green cells is not visible because it is either out of the viewport (Viewport Visibility, VV) or occluded by content in other cells (Occlusion Visibility, OV); as a result, it will be fetched at a lower point density. Distance Visibility (DV) dynamically adjusts the density of a point cloud based on its distance to the viewer.

Unlike regular videos (including 360° panoramic videos) that consist of 2D pixels, volumetric videos are 3D. Each frame of a volumetric video is comprised of 3D points or meshes. During playback, viewers can freely navigate themselves with six degrees of freedom (6DoF) movement, 3DoF of translational movement (X, Y, and Z) and another 3DoF corresponding to rotational movement (yaw, pitch, and roll). Volumetric videos enable numerous applications that cannot be supported by regular videos. For example, by shooting a high-resolution volumetric video of a family member's daily activity, one may later view it from different locations as if she/he was in the same room as the viewer [52]. In another example, a surgery can be broadcasted as a live volumetric feed, allowing medical students as viewers to get an immersive telepresence experience.

Besides the content format, volumetric videos differ from regular videos in other aspects including capturing, streaming, and analytics. In this paper, we address the problem of *streaming*. We first conduct comprehensive measurements of volumetric videos to gain key insights that facilitate our study. We then design and implement a streaming system called ViVo (Visibility aware Volumetric video streaming), which delivers high-quality volumetric content to commodity mobile devices. Among various volumetric data formats, we focus on the Point Cloud (referred to as PtCl) representation where each frame is a set of unsorted 3D points with attributes such as color and intensity. PtCl is arguably the most popular volumetric data format due to its flexibility and simplicity (§2). Nevertheless, as

[1] An introductory video made by a 3rd party can be found here: https://www.youtube.com/watch?v=feGGKasvamg

MobiCom '20, September 21–25, 2020, London, United Kingdom
© 2020 Association for Computing Machinery.
ACM ISBN 978-1-4503-7085-1/20/09…$15.00
https://doi.org/10.1145/3372224.3380888

we will discuss in §9, the high-level concepts of ViVo are applicable to other data formats of volumetric videos such as 3D mesh [49, 54].

Our study consists of the following.

Characterizing PtCl Encoding and Segmentation. Similar to regular videos, PtCl video streams need to be encoded (compressed) to reduce their bandwidth footprint. We create five representative PtCl videos using RGB-D cameras, and analyze the performance of state-of-the-art PtCl encoding/decoding schemes. When they are properly configured, typical PtCl frames can enjoy a lossless compression ratio of 4× to 6×, and commodity smartphones can achieve a decoding speed of 30+ FPS using multiple threads.

We further investigate the scheme where a PtCl stream is spatially *segmented* into smaller 3D *cells*, each being individually encoded and can be separately fetched. We carefully quantify the tradeoff between cell size and segmentation overhead. In this case, we do observe non-trivial segmentation overhead, which is higher than that of 2D panoramic video segmentation. The above findings dictate the design of encoding/segmentation schemes for ViVo.

Characterizing Viewport Movement and Designing Viewport Prediction. Through an IRB-approved user study, we collect viewport trajectories of 32 participants as they watch PtCl videos using two interaction methods: through an MR headset and on a smartphone. We find that both the video content and the interaction paradigm affect viewing behaviors such as movement trajectory, frequency, and speed. We then apply off-the-shelf, lightweight machine learning algorithms for 6DoF viewport movement prediction. On our data collected from the 32 users, it achieves a median error of 7–9cm for translational movement and 3–5° for rotational movement when the prediction window is 200ms.

Optimizing Content Fetching for Bandwidth Efficiency. Next, we propose three effective optimizations that significantly reduce bandwidth consumption for volumetric video streaming. As shown in Figure 1, their common high-level idea is to make the video player *visibility-aware*, by judiciously determining the video content to fetch based on how, what and where a viewer perceives. The three key optimizations are: *Viewport Visibility* (VV) that strategically discards cells not falling into the predicted viewport, *Occlusion Visibility* (OV) that reduces the point density of cells occluded by others, and *Distance Visibility* (DV) that adjusts the point density of a cell based on the viewpoint-to-cell distance. Note that the concept of VV is inspired by viewport-adaptive 360° video streaming [33, 56]. However, OV and DV are unique to volumetric videos, which explore the depth information in point cloud data. We jointly apply the three optimizations to maximize mobile data savings.

Developing and Evaluating the ViVo System. We integrate the above building blocks (encoding, segmentation, viewport prediction, VV, DV, and OV) into a holistic system with careful system-level engineering. All its key logic resides on a commodity Android device that wirelessly communicates with a stateless server. We thoroughly evaluate ViVo over diverse networks (WiFi, emulated LTE, and commercial 5G), users (MR headset and smartphone viewers), and content (4 different PtCl videos), using both objective metrics (structural similarity index, SSIM [64]) and subjective scores rated by real viewers. We highlight our evaluation results as follows.

- When bandwidth is sufficiently high, VV, OV, and DV can reduce the average data usage for PtCl streaming by 29.1%, 11.4%, and 7.4% (up to 84.5%, 14.7%, and 46.9%), respectively, compared to the corresponding baseline without the optimization. Meanwhile, ViVo incurs virtually no perceived quality loss (SSIM >0.99).
- Jointly applying all three optimizations yields a data saving of 41.8% and 39.5% on average (up to 70.1% and 84.5%) for two categories of PtCl videos, compared to the baseline fetching the entire PtCl. The perceived quality loss also remains negligible.
- On commercial 5G mmWave networks (∼50m line-of-sight), ViVo reduces the data usage by 36% and 39% (compared to the above baseline) when playing two PtCl videos and significantly shortens stall time, while also maintaining nearly perfect visual quality.
- When network bandwidth is constrained or fluctuating, subjective scores rated by 12 participants watching 144 pairs of videos (ViVo and baseline) indicate that ViVo outperforms (equals) the baseline for 62.5% (28.5%) of the QoE ratings.

Overall, this is to our knowledge a first comprehensive study of volumetric video streaming on mobile devices. We make the following contributions: (1) detailed investigation of PtCl encoding, decoding, segmentation, and viewport movement patterns in the mobile context; (2) three visibility-aware optimizations that effectively reduce mobile data usage and decoding overhead for volumetric video streaming; and (3) system design, implementation, and evaluation of ViVo, a full-fledged, ready-to-deploy volumetric video streaming system for commodity mobile devices. ViVo provides a platform for future research on volumetric video streaming for experimenting with more advanced algorithms such as sophisticated 6DoF viewport prediction and visibility-aware optimization schemes.

2 BACKGROUND ON VOLUMETRIC VIDEOS

Capturing. We can capture volumetric videos using RGB-D cameras (D for *depth*), *e.g.*, Microsoft Kinect [11], Intel RealSense [9], and various LIDAR scanners [57]. They are equipped with depth sensors and can acquire 3D data from different viewpoints. The data captured from multiple cameras will then be merged to form the entire scene through proper synchronization, calibration, and filtering. There are open-source software systems for realizing this pipeline. We use an enhanced version of LiveScan3D [44] to capture volumetric videos for this study. Structure-from-motion [60] is another widely used technology for 3D reconstruction, and has the potential for volumetric video capturing as well.

Data Format. 3D mesh and point cloud (PtCl) are two popular representations of volumetric videos. 3D mesh models the structural build of an object using a collection of vertices, edges, and faces (polygons). There is a plethora of research on 3D mesh in the computer graphics community [49, 54]. A PtCl is essentially a set of 3D points with attributes such as color or intensity. Compared to 3D mesh, PtCl is a more flexible and simpler representation, because it involves only unstructured points, and does not need to maintain the topological consistency (a requirement for 3D mesh [43]). Hence, we focus on PtCl-based volumetric videos in this paper.

Compression of 3D PtCls has been investigated in the literature. Most existing schemes leverage either octree-based compression [31, 36, 39, 51, 59] or k-d tree based compression [25, 27, 37, 46] for PtCls. We refer interested readers to Maglo *et al.* [49] and Peng *et al.* [54] for solutions of 3D mesh compression.

Vid.	# Pts/Frm	Frms	# People	SP (m^3)	Bitrate
P1	85.9K±3.5K	3622	1 (close)	2×2×2	302Mbps
P2	160K±9.2K	3490	2 (close)	2×2×2	563Mbps
P3	228K±3.6K	1941	3 (close)	2×2×2	802Mbps
M2	145K±17K	1847	2 (sep.)	5×2×5	510Mbps
M4	241K±3.3K	2612	4 (sep.)	7×2×7	847Mbps

Table 1: Volumetric video dataset for this study (SP: space).

3 MOTIVATION AND OVERVIEW

Streaming volumetric videos is resource-demanding. Raw (uncompressed) PtCl data is often prohibitively large for streaming over wireless networks. The representation of a 3D point typically takes 15 bytes: 4 bytes for each of the (X, Y, Z) position dimensions, and 1 byte for each of the (R, G, B) color dimensions. Thus, for instance, streaming a volumetric video with 200K points per frame at 30 FPS requires 15×200K×30×8 = 720Mbps of bandwidth. Compression (encoding) is thus essential for PtCl streaming (§4).

In many scenarios, merely compressing the PtCl stream still cannot provide a satisfactory QoE, because of either high decoding overhead or poor network conditions that occur in today's wireless networks – even in 5G due to its vulnerability to blockage and attenuation. Regarding the decoding overhead, we will quantitatively show that the client-side decoding may become the performance bottleneck even on high-end devices with multi-core processors, due to the inherent complexity of PtCl compression (§8.8).

Given the above observations, we explore in this study how to further reduce bandwidth consumption and client-side processing overhead for streaming PtCl volumetric videos to mobile devices. Our proposed system, referred to as ViVo, consists of three key optimizations for achieving this goal: VV, OV, and DV (§6).

We face several challenges when designing ViVo.

- Optimizations such as VV and OV require selectively fetching a portion of (encoded) PtCl at a specified point density level. This can be achieved by segmenting the entire PtCl into small blocks or cells, but the segmentation itself may incur high overhead (§4).
- ViVo requires robust viewport prediction (VP) to facilitate content prefetching and reduce the motion-to-photon delay [6]. In §5.2, we characterize real users' viewport movement patterns when watching volumetric videos, and propose our 6DoF VP techniques.
- Since all ViVo's logic resides on mobile devices, they need to be sufficiently lightweight, otherwise optimizations themselves may become the performance bottleneck. We make several algorithmic and system-level design decisions to address this challenge (§6).

4 PTCL ENCODING & SEGMENTATION

This section explores PtCl compression (encoding) and segmentation, which are two important components of ViVo.

Volumetric Video Dataset. We use Microsoft Kinect [11] to capture more than 10 PtCl streams for this study. We show the setup of one of our capturing systems with three Kinects in Figure 2. We then construct a volumetric video dataset with five videos: **P1**, **P2**, **P3**, **M2**, and **M4**. Among them, P1 shows a person giving a presentation. P2 and P3 consist of 2 and 3 people talking with each other. P1, P2, and P3 are created with a single captured PtCl per frame. M2 depicts a cosplay show performed by two artists who are situated at the following coordinates in meters (0, 0, 0) and (3, 0, 3). M4 consists of 4 artists performing singing. The four performers are at (0, 0, 0), (0, 0, 6), (3, 0, 3) and (-3, 0, 3). Due to their large scenes, M2 and M4 are produced by shooting each person individually, and then merging the PtCls into a whole scene. Table 1 summarizes the five videos in terms of their number of points per frame, number of frames, number of people in the scene, the displayed physical dimensions, and the raw video bitrate before compression.

	Draco (k-d tree)			PCL (Octree)			LEPCC		
Video	CR	E	D	CR	E	D	CR	E	D
P1	**6.0**	25	76	3.8	53	40	4.0	20	166
P2	**5.4**	14	46	3.8	26	21	3.7	11	90
P3	**5.6**	9.7	32	3.9	20	16	3.7	7.5	64
M2	**4.5**	14	49	3.9	31	23	3.3	11	99
M4	**4.8**	8.3	29	3.6	17	13	3.2	7.1	60

Table 2: Point cloud compression/decompression performance of open-source libraries for five volumetric videos.

Comparing PtCl Compression Schemes. We evaluate the performance of three open-source PtCl compression solutions: Draco from Google [5], Point Cloud Library (PCL) [16], and LEPCC (Limited Error Point Cloud Compression) from `Esri` [12]. The three solutions use different algorithms. Draco employs k-d tree [25, 27, 37, 46] based compression. LEPCC extends LERC (Limited Error Raster Compression) [13] for 2D images to 3D point clouds. PCL's compression algorithm is based on Octree [31, 36, 39, 51, 59]. To ensure fair comparisons, we configure lossless compressions (both the position and color dimensions, to the precision of the raw data) for all three algorithms, except for LEPCC's color compression.

Table 2 compares the compression ratio, compression and decompression latency of the above three solutions. The "CR" column represents the compression ratio, defined as the ratio between the raw bytes and the compressed bytes. CR differs across different algorithms. Compared to PCL's Octree and LEPCC, Draco's k-d tree yields the highest lossless CR likely due to its quantization feature [37], which can flexibly keep the precision of decompressed data only up to that of the original data. The impact of video content on CR is small, as the five videos belong to the same category (performance of people, a major content type for volumetric videos [1, 8, 10]). The "E" and "D" columns in Table 2 measure the average per-frame encoding and decoding latency (in FPS), respectively. It is tested on a server with Intel Xeon CPU E3-1270 v5 @ 3.60GHz. By comparing Table 2 with Table 1, we note that the latency is well correlated with the number of points per frame.

Based on the results in Table 2, the highest bitrate among these videos is still around 180 Mbps even after compression. Such a high encoded bitrate stems inherently from the 3D nature of volumetric videos; it also serves as a key motivation of ViVo's visibility-aware optimizations that significantly reduce bandwidth utilization for volumetric video streaming. We believe volumetric video streaming will be an important application for 5G, under which such required bandwidth is practical and can be achieved (§8).

Decoding on Mobile Devices. In ViVo, the encoded PtCl stream needs to be efficiently decoded on mobile devices. We next study the PtCl decoding performance of Draco on three mainstream smartphones: Samsung Galaxy S8 (SGS8, released in 2017), Huawei Mate 20 (2018), and Samsung Galaxy S10 (2019), with different CPU

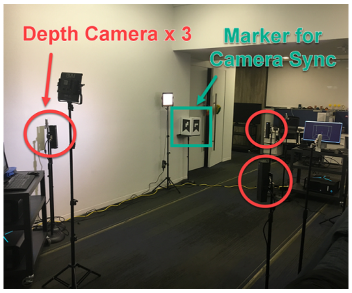

Figure 2: Setup of one of the volumetric video capturing systems.

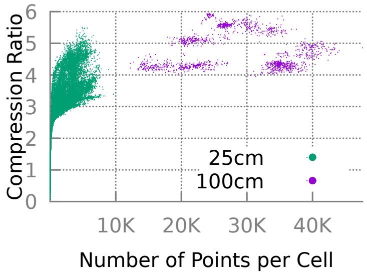

Figure 3: Compression ratio of cells (video P3, 25cm vs. 100cm).

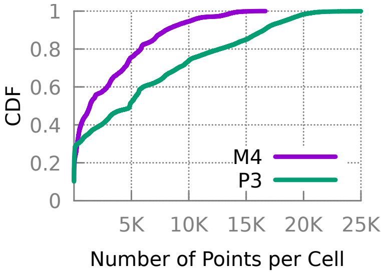

Figure 4: Cell size (# of points) distribution for two videos P3 and M4.

Phone	SGS8 (2017)	Mate20 (2018)	SGS10 (2019)
CPU Conf.	4×2.45 GHz 4×1.9 GHz	2×2.6 GHz 2×1.92 GHz 4×1.8 GHz	1×2.84 GHz 3×2.42 GHz 4×1.8 GHz
1 Thread	10.3±0.3	13.5±1.1	18.4±0.0
2 Threads	16.9±1.4	23.9±0.7	27.1±1.7
3 Threads	26.2±0.7	28.5±1.8	36.6±0.5
4 Threads	32.9±0.4	30.2±1.3	46.0±0.1
6 Threads	37.0±1.0	31.4±1.9	50.7±1.8
8 Threads	40.0±0.6	30.4±1.4	58.1±3.2

Table 3: Decoding performance of Draco (video P3).

	P1	P2	P3	M2	M4
100cm	1.151	1.194	1.198	1.266	1.362
50cm	1.297	1.343	1.362	1.375	1.501
25cm	1.514	1.554	1.582	1.527	1.666

Table 4: Segmentation overhead for different videos. The baseline is the corresponding video without segmentation.

configurations. Table 3 shows their decoding performance (in FPS) for video P3 over 1941 frames, using different numbers of threads.

Using multi-threading, all three mobile devices can achieve higher than 30 FPS decoding performance that is considered to be perceptually smooth. For SGS8/10, increasing the number of threads (CPU cores) helps improve the decoding performance. The CPUs of all three devices use ARM's big.LITTLE heterogeneous architecture [2]. For Mate20, the performance remains stable for more than 3 threads, because the combination of the four big CPU cores of Mate20 is weaker than those of SGS8/10.

Impact of Segmentation on Compression. A salient feature of ViVo is viewport-adaptive streaming of PtCl, which (ideally) delivers only the user's perceived portion. This approach brings significant reduction of bandwidth utilization and client-side decoding overhead in particular for videos consisting of objects that are spatially separated (*e.g.,* M2 and M4), because a viewer oftentimes cannot see all the objects at once. One of its key prerequisites is to spatially segment the original PtCl into smaller "sub" point clouds which we call *cells*. The cells are independently downloadable and decodable, making it feasible for the client to fetch and display portions of the original PtCl at the cell-level granularity. However, segmentation incurs overhead: independently encoding each cell eliminates the opportunity for inter-cell compression, making the overall video size (across all cells) larger than that of the unsegmented video.

We next quantify this overhead by segmenting our five videos into cells of three sizes: 25×25×25 cm^3, 50×50×50 cm^3, and 100×100×100 cm^3. We encode each cell using the same compression scheme (lossless Draco). Table 4 shows the segmentation overhead, defined as the ratio between the sum of all cells' encoded bytes over the encoded bytes of the original video. The overhead is non-trivial, ranging from 1.15 to 1.67 for the five videos. The smaller the cell size is, the higher this overhead becomes. The reason is that cells with fewer points have lower CRs and thus higher segmentation overhead. This is illustrated in Figure 3, which shows a scattered plot between CR and the number of points per cell, for P3 with two cell sizes (25cm and 100cm). We also find that for the same cell size, the five videos exhibit different segmentation overhead. It is again attributed to their points-per-cell distribution, which is shown in Figure 4 for videos P3 and M4 using 50cm cells. Compared to P3, M4 has statistically fewer points per cell. This translates to a lower CR on average and thus a higher segmentation overhead for M4.

We notice that for PtCl volumetric videos, the segmentation overhead is typically higher than that of 2D panoramic video segmentation (as reported by Flare [56]). This is because PtCl encoding, which is still under active development by MPEG [61], is less mature than that of traditional video encoding. Note that the above observations are also valid for PCL and LEPCC.

Implications on System Design. The above measurement results offer several implications on the design of ViVo. First, PtCl streams need to be properly compressed to reduce bandwidth footprint. ViVo employs Draco that can effectively reduce bandwidth usage by a factor of 4+. Second, off-the-shelf smartphones are capable of decoding PtCl streams with a decent number of points (>200K per frame) at 30 FPS. ViVo employs multiple CPU cores for decoding and leaves the rendering to GPU. Third, cell segmentation incurs non-trivial overheads. Therefore, ViVo needs to make judicious decisions about whether to employ segmentation and how to set the cell size.

5 VIEWPORT MOVEMENT & PREDICTION

We now study viewport prediction, which is another essential part of viewport-adaptive video streaming systems.

5.1 Understanding Viewport Movement for Volumetric Videos from Real Users

Collecting Viewport Trajectory from Real Users. To understand how real users change their viewport when watching volumetric

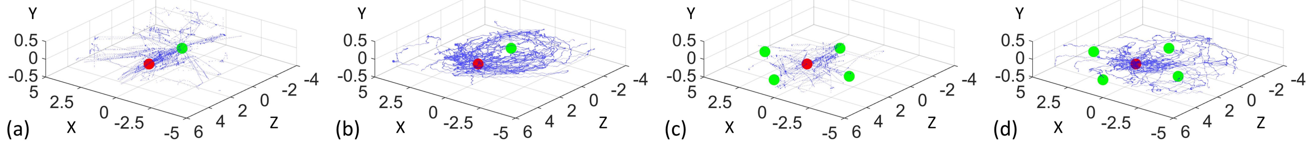

Figure 5: Translational viewport trajectories (left to right: P2 of Group P, P2 of Group H, M4 of Group P, and M4 of Group H).

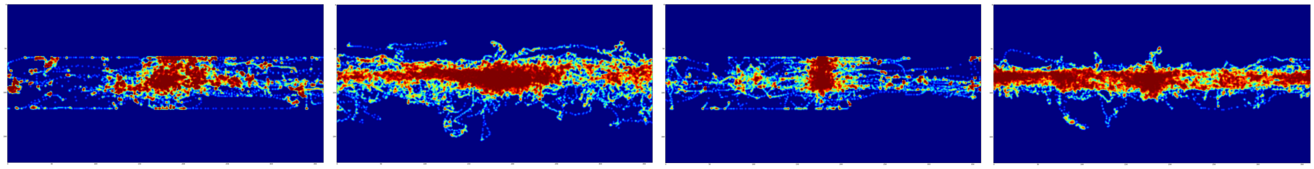

Figure 6: Heatmaps for yaw and pitch (left to right: P2 of Group P, P2 of Group H, M4 of Group P, and M4 of Group H).

videos, we conduct an IRB-approved user trial. We recruit 32 users from a large university and a large corporation. The participants are diverse in terms of their gender (16 females), age (from 20 to 55), and education levels (from freshman to Ph.D.). We divide the 32 participants into two groups **P** (Smartphone) and **H** (MR Headset). Each group consists of 8 females and 8 males; the users' other profiles (age, education, *etc.*) are also largely balanced between the two groups. We ask the participants in each group to watch 4 out of the 5 videos (P1 is used for the tutorial purpose) listed in Table 1.

The difference between Groups P and H is the equipment that the participants use. Group P uses an SGS8 smartphone. Its users can rotate their viewport by moving the motion-sensor-equipped phone, or swiping on the screen using one finger. Users can make translational movements using two fingers: pinch or spread to move backward or forward (Z dimension), and perform two-finger swipe to move leftward or rightward (X dimension), and upward or downward (Y dimension). To ensure good usability, the above interaction paradigm has been evaluated on real users through a separate study.

Group H employs a state-of-the-art MR headset (Magic Leap One [14]). Using the headset, users can make natural rotational movement (turning head or body) and translational movement (walking or crouching) that will be picked up by high-precision on-board motion sensors. Correspondingly, we have developed two applications, one for smartphones and another for headsets, which play videos in a random order while recording the viewport trajectory at 30Hz. Before starting data collection, we give a tutorial using video P1 to the users, in order to help them get familiar with our players.

Characterizing Phone Based vs. MR Headset Based Interactions. Leveraging this unique dataset, we characterize phone-based and headset-based viewing behavior of volumetric videos. The subplots (a) and (b) in Figure 5 plot all 16 users' translational movement for P2 using phones and headsets, respectively. The red dot is the initial position (0, 0, 3) of the viewer and the green dot is the center of video content (*e.g.,* the two side-by-side people in P2).

We make three observations. First, almost all movements appear on the same plane of $Y = 0$, indicating that viewers seldom move vertically, since, for example, crouching down and jumping up may be inconvenient for most users. The median and 90th percentile of vertical position Y during a video playback are -0.007m and 0.370m for users in Group P, respectively (-0.009m and 0.055m for users in Group H). Second, subplots (a) and (b) show differences between Groups P and H: the former's translational movement appears to be straight as users typically maintain the direction during an on-screen swipe. In contrast, Group H's translational trajectories are more smooth, as users make natural body movement when wearing an MR headset. Subplots (c) and (d) show similar trends for M4. Third, comparing subplots (a) and (c) as well as (b) and (d) indicates different moving patterns. When viewing a single object or co-located objects (P2), the viewer usually moves around the object as illustrated by the "circling" pattern in subplot (b). In contrast, when observing multiple spatially separated objects such as those in M4, the viewer's positions usually concentrate at the center among these objects, in order to watch different objects by only rotating her viewport. This will bring more data savings for VV (§8.3) as in this viewing paradigm, it is more likely that the viewer consumes a small portion of the whole scene.

We find that users in Group H have a statistically higher movement speed than those in Group P. Figure 7 and Figure 8 plot the CDF of translational speed (in m/s) and rotational speed (in degree/s), in each dimension across all users[2]. For all dimensions' movement speed (X, Y, Z, yaw, and pitch), Group P's 75th percentile is lower than that of Group H. This can be explained by the convenience brought by the MR headset that allows users to move swiftly. Meanwhile, for translational movement, users in Group P exhibit a longer tail than those in Group H, because some users in Group P perform quick on-screen swipes.

Besides movement speed, another interesting finding relates to movement frequency: users in Group H move more frequently than those in Group P. This is demonstrated in Figure 9 that plots the percentage of idle period, when movement occurs in none of the translational dimensions (X, Y, Z), across all videos watched by each group's users. The median idle percentages are 20% and 63% for Groups H and P, respectively. Such a disparity leads to the fact that users in Group H tend to explore more areas than those in Group P. Figure 5 illustrates this for X, Y and Z dimensions, as the Group-H users' trajectories appear to be more "dense" (subplot (a) vs. (b) and (c) vs. (d)). The heatmaps in Figure 6 show a similar trend for the yaw and pitch dimensions, where Group H's heatmaps cover more viewing directions than those of Group P.

[2] In this work, we consider only yaw and pitch for rotational movement, because viewers rarely change the roll (similar to the viewing behavior of 360° videos [24, 56]).

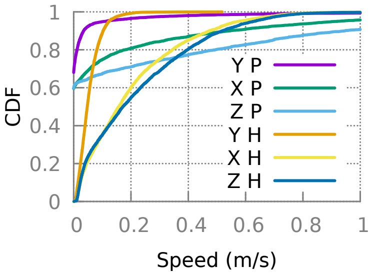

Figure 7: CDF of translational speed.

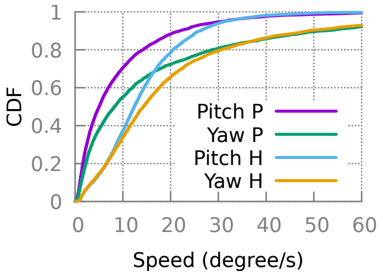

Figure 8: CDF of rotational speed.

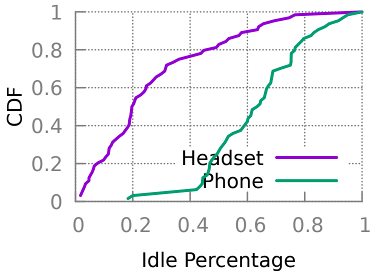

Figure 9: CDF of translational idle %.

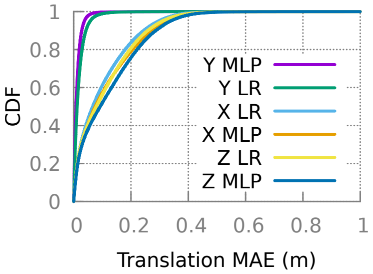

Figure 10: LR vs. MLP: translation.

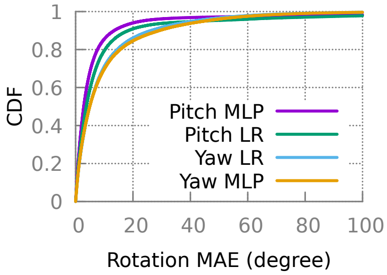

Figure 11: LR vs. MLP: rotation.

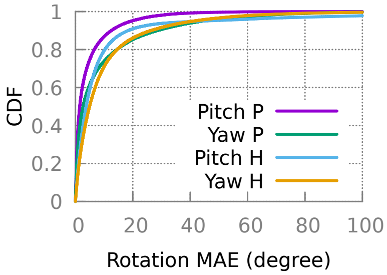

Figure 12: Group H vs. Group P: rotation.

5.2 Viewport Prediction (VP)

Compared to 360° videos, volumetric videos make VP more challenging due to the additional translational dimensions (X, Y, Z) that also need to be accurately predicted. Our high-level design principles of ViVo's VP approach consist of two aspects. First, the method should be lightweight for mobile devices. Second, since volumetric videos involve as many as 6DoF, having a single model that jointly considers all dimensions may be too complex due to the large prediction space; we therefore predict each dimension separately, and derive the predicted viewport by combining each dimension's prediction result. We later experimentally demonstrate that such an approximation, when complemented with other optimizations, can yield a satisfactory QoE (§8).

We use two lightweight off-the-shelf machine learning models to predict each dimension of X, Y, Z, yaw, and pitch: linear regression (LR) and multilayer perceptron (MLP). Suppose the current time is T in a playback. Both methods employ a history window of h ms, which covers the viewports of frames played from $T - h$ to T, to train an LR or MLP model and predict the viewport position at $T + p$ where p is the prediction window. The above training and inference are thus performed online on a per-frame basis. We configure the MLP with a single hidden layer with 3 neurons, and employ hyperbolic tangent [40] as the activation function. The MLP employs L-BFGS (Limited-memory Broyden-Fletcher-Goldfarb-Shanno) [47] for optimization. We use scikit-learn [17] for training and prediction. Note that we choose LR and MLP due to their low computational overhead that enables frequent online training and inference on mobile devices. We discuss the feasibility of employing more sophisticated deep learning models for 6DoF VP in §9.

We utilize our viewport trajectory traces collected from the user study to understand the prediction accuracy. We consider several prediction windows from 50ms to 500ms, and a wide range of history windows from 33ms to 660ms at a step of 33ms (one frame's duration). We find that LR achieves the best prediction accuracy when h is roughly $p/2$, while MLP's accuracy is maximized when h is 660ms for X, Y and Z and 66 ms for yaw and pitch, for all p.

Figure 10 plots the mean absolute errors (MAEs) of translational movement prediction for all users in Group H, with the prediction window set to 200ms – a practical upper bound of the looking-ahead delay that ViVo needs to deal with in today's WiFi and cellular networks. Decreasing the prediction window further improves the accuracy (figures not shown). Each sample in the CDF corresponds to a single prediction instance. As shown, the Y (vertical) movement exhibits the lowest MAE as users seldom move vertically (§5.1); the X and Z predictions yield similar accuracy, with the median being around 0.07m–0.09m. Figure 11 plots the MAE of rotational movement prediction for users in Group H. The pitch and yaw show similar predictability, with the median MAE being around 3°–5°.

There is no qualitative difference between the accuracy of LR and MLP. We thus adopt LR for ViVo given its more lightweight nature. Recall that we perform online learning where in each prediction step (every 33ms, for an FPS of 30), a new model is trained and applied for prediction. On SGS8, this can be completed within 1ms for all five dimensions. We will discuss system-level optimizations that help tolerate the prediction inaccuracy in §6.

Figure 12 compares the rotational prediction accuracy between Groups P and H, using LR and p=200ms. We observe that users in Group P exhibit slightly better predictability than those in Group H, because the former move less frequently and more predictably than the latter as quantified in §5.1. We make similar observations for translational movement prediction. The above results indicate that the same prediction approach can work reasonably well for both the phone-based and headset-based interaction paradigms.

6 VISIBILITY-AWARE OPTIMIZATIONS

In this section, we present the design of ViVo's three visibility-aware optimizations: Viewport Visibility (VV), Occlusion Visibility (OV) and Distance Visibility (DV).

Viewport Visibility (VV). From a given viewpoint (X, Y, Z) and for a specific viewing direction (yaw, pitch, roll), the viewport of a 3D scene is typically determined by the so-called view frustum (a truncation with two parallel planes of the pyramid of vision) in 3D graphics. When viewers watch a volumetric video, only the content inside this view frustum will be rendered on the display. The view frustum depends on four parameters: the field of view (FoV) angle in Y dimension (FoV-Y), the aspect ratio (*i.e.,* the ratio between FoV's width and height) that indirectly specifies the FoV in X dimension, and the distances from the viewer to the near and far clipping planes. A typical value of FoV-Y is 45° (as adopted by ViVo), and the aspect ratio is usually the ratio between the width and height of the display.

The basic idea of VV is to fetch the volumetric content that overlaps with the predicted viewport. Specifically, given the predicted viewport, ViVo computes its corresponding frustum F through 3D geometry. It then determines the cells that are inside or intersect with F using view frustum culling [19, 20], as the to-be-fetched cells.

In practice, the predicted viewpoint and viewing direction may be inaccurate. Recall from §5.2 that in order to keep it lightweight, ViVo predicts the viewport by combining all dimensions' prediction results. Therefore, small prediction errors in individual dimensions will accumulate into a large deviation from the ground truth. Empirically, using our collected viewport trajectory data, we find that even if we double FoV-Y from 45° to 90°, it still cannot tolerate all VP errors, due to the inherent difficulty of 6DoF VP.

To address the above challenge, ViVo conservatively employs a large *virtual* viewport V_L to accommodate inaccurate VP. It has the same viewpoint and viewing direction as the predicted viewport. We configure its FoV-Y, a critical parameter, to F_L=120°. Based on our viewport trajectory data, a 120° FoV-Y can cover almost all (>99%) content in the actual viewport. To reduce data usage, instead of using the same point density level (PDL) for all cells overlapping with V_L, we gradually reduce the PDL from the center to the periphery of V_L. Note that PDL is a key factor that affects the QoE of a PtCl. In this work, we use 5 density levels that randomly sample 20%, 40%, 60%, 80%, and 100% of the points for each cell.

We set up several smaller virtual viewports $V_1, ..., V_{L-1}$ whose FoV-Ys are $F_1 < ... < F_{L-1} < F_L$. For each cell c, let the initial PDL passed to the VV algorithm be $D(c)$. We identify a virtual viewport V_k such that c overlaps with V_k but not $V_1, ..., V_{k-1}$, and set the PDL of c to $\max\{D(c) - k + 1, 0\}$. If such a V_k does not exist, c will not be fetched. Hence, the further a cell is away from the predicted viewport, the lower its PDL will be. In ViVo, we empirically choose $L = 3$ and V_1, V_2 and V_3 to be 60°, 90°, and 120°, by investigating our viewport trajectory dataset. As a result, even when the prediction is not accurate, users usually can still see the video with a lower quality instead of a blank screen (or experiencing video stalls). In the extreme case where part of the predicted viewport is out of the 120° FoV-Y, the corresponding cells will be missing, but that happens very rarely as will be demonstrated by our performance evaluation in §8.

Occlusion Visibility (OV). Ideally we want to discard the video content that falls into the viewport's frustum but is occluded by others. Such a PtCl visibility problem has been investigated by the computer graphics community [41, 42, 50]. One well-known algorithm is the Hidden Point Removal (HPR) operator [42]. However, HPR together with many other solutions along the line are largely designed for static PtCls, and they are unacceptably slow for processing PtCl frames in real time. We implement HPR and evaluate it on a server machine with Intel Xeon CPU E3-1270 v5 @ 3.60GHz. It takes HPR more than 10 seconds to process a PtCl with 225K points due to heavy-weight geometric computation (*e.g.,* constructing a convex hull), making it infeasible for ViVo. In addition, HPR requires the position of each point, and thus can only run on the server side.

We propose an efficient algorithm that determines the occluded portion at the cell level given the predicted viewport. The algorithm enumerates all cells overlapping with the predicted viewport. For each cell c, it applies robust heuristics to calculate its occlusion level denoted as $O(c)$. The larger $O(c)$ is, the more likely that c will be occluded by others. Note that since the algorithm does not know the coordinates of individual points within a cell, it is inherently impossible to derive the precise occlusion relationship. Instead, our OV algorithm only computes a heuristic-driven likelihood of occlusion, which, as we will evaluate later, can already bring non-trivial data savings while maintaining almost lossless visual quality.

Now let us consider how to calculate $O(c)$ for a given cell c. We first draw a ray from the predicted viewpoint to the center of c. Intuitively, all cells that occlude c must (1) intersect with the ray, and (2) be closer to the viewpoint than c. We then use the above two criteria to look for such cells. For performance consideration, instead of searching for all cells, we only test c's surrounding cells whose distance to c is up to L. We use the Chebyshev distance defined as $\mathbf{L}_\infty(c^1, c^2) = \max(|c_x^1 - c_x^2|, |c_y^1 - c_y^2|, |c_z^1 - c_z^2|)$. For example, when L=1 (as adopted by ViVo), we consider only c's 26 surrounding neighbors. We employ Ray-Box Intersection algorithm [65] to perform fast intersection tests. Through the above process, we calculate the total number of surrounding cells that meet both criteria, denoted as $S(c)$. In addition, among all such cells, let the cell with the largest number of points be c'. We also calculate the ratio of points between c' and c, denoted as $R(c)$. For example, if c' and c have 150 and 100 points respectively, then $R(c)$=1.5. Note that the client knows the number of points that each cell contains, through a manifest file delivered from the server before video playback (§7).

$S(c)$ and $R(c)$ are good indicators for potential occlusion. $O(c)$ is positively correlated with $S(c)$ (the number of cells that may occlude c) and $R(c)$ (the highest point density ratio among all cells that may occlude c). We thus set up an empirical mapping from $(S(c), R(c))$ to $O(c)$ as:

$$O(c) = \begin{cases} 0 & R(c) < \alpha_0 \beta^{S(c)-1} \\ 1 & \alpha_0 \beta^{S(c)-1} \leq R(c) < \alpha_1 \beta^{S(c)-1} \\ 2 & \alpha_1 \beta^{S(c)-1} \leq R(c) < \alpha_2 \beta^{S(c)-1} \\ 3 & \alpha_2 \beta^{S(c)-1} \leq R(c) \end{cases}$$

where parameters $\{\alpha_0, \alpha_1, \alpha_2, \beta\}$ are set to {0.6, 1.0, 3.0, 0.8}, based on our examination of a large number of viewports from our dataset (§5.1). Decreasing (increasing) these values makes OV more conservative (aggressive). The last step is to use $O(c)$ to adjust the PDL of c. Let c's initial PDL passed to OV be $D(c)$. We reduce it to $\max\{D(c) - O(c), 0\}$ to accommodate the occlusion level.

Overall, leveraging the predicted viewpoint and the cells' point densities, OV applies our domain knowledge and heuristics to determine the occlusion level and use it to reduce the point density

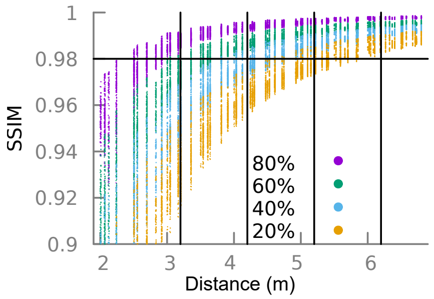

Figure 13: SSIM of PtCls viewed at various distances.

for each cell. Compared to HPR, the algorithm executes at sub-millisecond level on mobile devices. It may benefit other algorithms such as lighting estimation [30] and light field depth estimation [63] where occlusion detection serves as a key building block.

Distance Visibility (DV). A unique property of volumetric videos is that a user can view the content at different distances. When the viewpoint is far away, the perceived PtCl will become small and neighboring 3D points may be projected to the same displayed 2D pixel. In this case, reducing the PDL brings no or little degradation of the perceived quality. This forms the central idea of DV, which selects each cell's PDL according to the viewpoint-to-cell distance.

The challenge here is how to establish the mappings between the distance and the PDL. We address it through a data-driven approach, which consists of an offline and an online phase. At the offline phase, we sample many PtCl frames and "view" (*i.e.,* render) them at different distances, angles, and PDLs, thus creating snapshots of different viewports. We then assess the perceptual quality for these 2D snapshots using objective metrics such as SSIM. In this way, we obtain a large number of mappings from geometric properties (distance, angle, and PDL) to visual qualities. We next utilize such mappings to build a model that dictates the PDL selection without affecting the QoE. In ViVo, we build a model for each video, but depending on the content, it may be feasible to construct a model for a category of videos or a part of a video. In the online phase, ViVo utilizes the model and the geometric properties of the predicted viewpoint to calculate in real time the proper PDL for each cell.

We next illustrate the DV algorithm by walking through an example for video P2. In the offline phase, we select 10 frames (PtCls) from P2. For each PtCl, we take 6,000 snapshots by varying the translational positions – 20 values of X, 20 values of Z, 3 values of Y (given that viewers seldom move vertically, see §5.1), and 5 PDLs (sampling 20% to 100% of points in each cell). To reduce the feature space and thus the model building time, we fix the viewing direction to be always toward the center of the PtCl. The rationale is our empirical observation that rotational movements have a much smaller impact on the perceptual quality than translational movements, which directly changes the viewpoint-to-PtCl distance.

For each of the 48,000 snapshots rendered from PtCls without the highest PDL (100%), we calculate their SSIM indexes, using the corresponding snapshots with the highest PDL as the ground truth. We then visualize our model in Figure 13, which plots the relationship between the viewpoint-to-PtCl distance and the SSIM for all 48,000 snapshots. As shown, given a PDL, the SSIM statistically improves as the distance increases. According to a previous study [26], an SSIM higher than 0.98 means that the rendered content is visually lossless. To determine the PDL at a given distance d, we thus apply an SSIM threshold of 0.98, and select the *lowest* PDL l such that all snapshots with d and l bear SSIM indexes of at least 0.98.

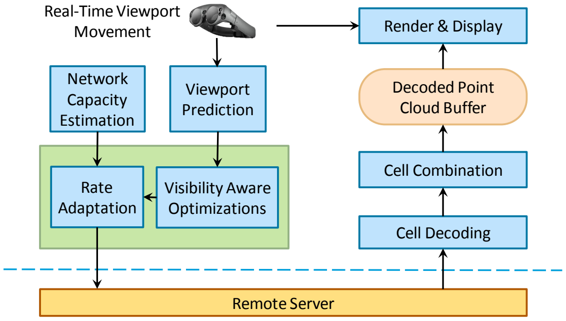
Real-Time Viewport Movement
Render & Display
Network Capacity Estimation
Viewport Prediction
Decoded Point Cloud Buffer
Rate Adaptation
Visibility Aware Optimizations
Cell Combination
Cell Decoding
Remote Server

Figure 14: The system architecture of ViVo.

Figure 13 indicates that given a PDL, its lowest SSIM increases almost monotonically as the distance increases. This allows us to conveniently simplify the model into discrete mappings from distance ranges to PDLs. In the example shown in Figure 13, the mappings are [0, 3.2)→100%, [3.2, 4.2)→80%, [4.2, 5.2)→60%, [5.2, 6.2)→40%, and [6.2, ∞)→20%. They correspond to the vertical lines in Figure 13. The above model construction procedure is a one-time effort per video, and can be fully automated. We apply it to other videos in our dataset (*e.g.,* the model for P3 is almost the same as that for P2). Then in the online phase, for each cell c, ViVo will use the (predicted) viewpoint-to-cell distance to lookup a PDL, and use that level for c if it is lower than the initial level passed to the DV algorithm. The size of the lookup table is determined by the number of PDLs, which, similar to the encoding bitrate ladder for adaptive bitrate (ABR) streaming, is usually small (*e.g.,* $<$10) in practice.

Combining VV, OV, and DV. The three optimizations focus on different aspects of volumetric data fetching and rendering and can be jointly leveraged in a natural manner. Specifically, ViVo first applies DV to obtain an initial point density level for each cell, followed by VV and OV, which may further reduce the cells' density levels depending on the viewport and the occlusion levels, respectively.

7 SYSTEM INTEGRATION

We integrate the components described in §4, §5 and §6 into a holistic system and show its architecture in Figure 14. All ViVo's key logic resides on a client device that wirelessly communicates with a stateless server, not requiring support from a cloud or edge server for offloading/acceleration. The segmentation, encoding, and DV model training are performed offline (not shown in the figure). At runtime, before playing a video, the player first fetches a small manifest file from the server. It contains each cell's position, size, the number of points, and the DV model. During the playback, ViVo keeps collecting the viewport trajectory and performing VP. The VP results are utilized by DV, VV, and OV. Subsequently, ViVo executes a lightweight throughput-based rate adaptation algorithm [38] to assess if the PDLs chosen by the optimizations can be satisfied by the estimated network capacity. If not, the rate adaptation module will globally reduce the PDLs. After rate adaptation, ViVo issues the requests for the cells to the server.

With the help of VP, ViVo (pre)fetches the cells in the predicted viewport and decodes them with multiple decoders, before merging and caching them in the Decoded Point Cloud Buffer (shown in Figure 14). At the playback time (*i.e.,* when the ground-truth viewport is determined), ViVo can thus immediately render the decoded point cloud data in the buffer. Thus, the motion-to-photon latency is essentially the rendering latency, which is measured to be low (*e.g.,* <6ms for a frame with around 100K points on SGS8).

The above processing is executed on a per-frame basis. Benefiting from our various performance optimizations, it can be performed at a line rate of 30+ FPS. Meanwhile, ViVo properly decodes the received cells and renders them at 30 FPS based on the user's viewport. It maintains a decoded cell buffer of 5 frames. The shallow buffer facilitates a short VP window and therefore good VP accuracy. We emphasize that ViVo determines the viewport visibility, occlusion visibility, and distance visibility all on the client side. This helps address the scalability issue faced by a server-side approach.

Implementation. We implement the ViVo PtCl video player on Android devices and its video server on Linux. For performance consideration, we use Android NDK (C++) for visibility determination, scheduling, cell fetching, and PtCl decoding. We cross-compile the Draco library for decoding PtCls on Android devices and utilize GPU-accelerated OpenGL ES for rendering PtCls. Unless otherwise mentioned, PtCl decoding is performed over four threads. Our implementation does not require root access. We have tested ViVo on multiple mainstream mobile devices, including SGS8, SGS9, SGS10, and Mate20, all loaded with Android OS 7.0+. We implement the ViVo video server in C/C++ on Linux (tested on Ubuntu 18.04, CentOS 7 and RHEL 7.6 distributions). The client-server communication is realized by a custom protocol over TCP, but it can also be implemented using standard application layer protocols such as HTTP(S). In total, our ViVo implementation consists of 8,000+ lines of code (LoC): 2,000+ LoC in Java and 4,000+ LoC in C++ for the video player, and 2,000+ LoC in C++ for the server.

8 PERFORMANCE EVALUATION

8.1 Experimental Setup

Our client devices are off-the-shelf SGS8 (Android 7.0, Snapdragon 835 system-on-chip, 4GB RAM) and SGS10 (Android 9.0, Snapdragon 855 system-on-chip, 8GB RAM). Unless otherwise mentioned, we use SGS8 in our experiments. We set up a commodity server with Intel Xeon CPU E5-2680 v4 @ 2.40GHz and Ubuntu 18.04 as the video server. For controlled experiments, we connect the mobile device and the video server using a commodity 802.11ac AP at 5GHz. The end-to-end peak downlink throughput is 320+ Mbps, and the client-server PING latency is <10ms. In addition to testing on bare WiFi network, we assess ViVo's performance over fluctuating network conditions in a reproducible fashion, by using the `tc` tool to replay 10 network bandwidth traces collected at multiple locations from a large commercial LTE network in the U.S. The average bandwidth of these traces ranges from 73 Mbps to 174 Mbps, and the standard deviation ranges from 13 Mbps to 28 Mbps. During the replay, we also use `tc-netem` to increase the end-to-end latency to 60ms, a typical RTT for today's LTE networks. For real-world experiments, we use a commercial 5G mmWave network in the U.S. (launched in 2019). Regarding the PtCl content, we use videos P2, P3, M2, and M4 listed in Table 1 (P1 is used for the tutorial purpose) and replay the 32 users' viewport traces collected in §5.1.

	VV	OV	DV
P2	0.9977±0.0068	0.9967±0.0057	0.9968±0.0063
P3	0.9977±0.0044	0.9964±0.0046	0.9963±0.0061
All	0.9977±0.0060	0.9966±0.0054	0.9966±0.0062

Table 5: SSIM for individual optimizations.

8.2 Effectiveness of Individual Optimizations

We first assess the effectiveness of individual optimizations: VV, OV, and DV. We replay all 16 Group-H users' viewport traces of P2 and P3, and measure two metrics: traffic volume and visual quality. They incur the key tradeoff that ViVo needs to balance. For VV, the baseline fetches the whole PtCl without segmentation; ViVo uses $100\times100\times100$ cm^3 cells, and dynamically adjusts the viewport according to the VV algorithm as described in §6. For the baseline of OV and DV, it uses $100\times100\times100$ cm^3 cells and a very large viewport of 120° (FoV-Y, see §6); only cells overlapping with the viewport will be fetched. For a fair comparison, ViVo uses the same configuration as above when evaluating OV or DV. The rationale of having such a baseline is to separate the impact of OV and DV themselves from that of segmentation. We conduct our experiments on unthrottled 802.11ac WiFi networks and configure the baseline scheme to fetch the content at the highest PDL. For ViVo, the initial PDLs for all cells are set to the highest; the three optimizations may selectively reduce some of them based on the optimization logic.

Figures 15, 16, and 17 plot the encoded bitrate for VV, OV, and DV. Each sample corresponds to one video's playback using the viewport trajectory of one user (among the 16 users). The "Base" and "Opt" bars show the results for unoptimized playbacks (still encoded) and optimized playbacks. We show the results of two videos (P2 and P3) separately. VV achieves the highest gain, with a median data usage reduction of 29.1% (up to 84.5%). Figure 15 indicates that in some cases, ViVo may fetch more data than the baseline. This is caused by the segmentation overhead: when, for example, the viewport contains the entire PtCl, VV pays the penalty of segmentation. This is not a hard limitation of ViVo, and can be mitigated by dynamically switching among different cell sizes based on the predicted viewport. Figure 16 shows that for OV, the saving is relatively smaller, around 10% on average. This is attributed to ViVo's conservative OV design to maintain good visual quality. Figure 17 indicates that the average data saving of DV is comparable to that of OV. However, in some cases where the viewer is far away from the PtCl, the saving can reach 46.9%.

Table 5 shows the average visual quality of viewport content using SSIM [64], yielded by the three optimizations. Recall that an SSIM higher than 0.98 means that the rendered content is visually lossless [26]. When calculating the SSIM of viewport content for each frame, we generate the ground truth by rendering the PtCl from the original volumetric video at the highest PDL based on the user's actual viewport. The visual quality is averaged over 10K randomly samples per optimization. For all optimizations, their SSIM indexes are higher than 0.99, indicating almost perfect visual quality.

8.3 Effectiveness of Combined Optimizations

We now evaluate the benefits of combining multiple optimizations (VV, OV, and DV). Similar to §8.2, we replay all 16 Group-H users' viewport trajectories over unthrottled 802.11ac WiFi, at the highest

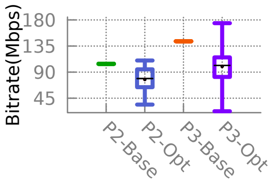

Figure 15: VV only: P2 & P3.

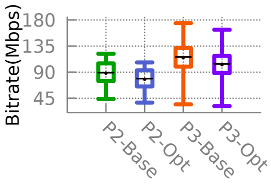

Figure 16: OV only: P2 & P3.

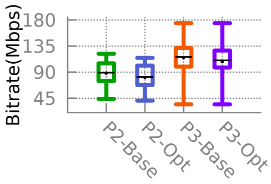

Figure 17: DV only: P2 & P3.

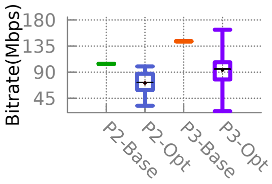

Figure 18: VV+OV: P2 & P3.

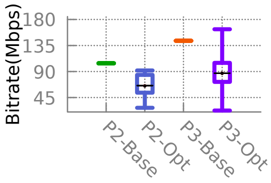

Figure 19: All Three: P2 & P3.

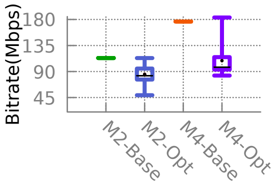

Figure 20: VV only: M2 & M4.

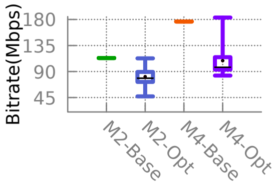

Figure 21: VV+OV: M2 & M4.

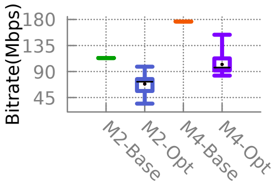

Figure 22: All Three: M2 & M4.

Videos	VV	VV+OV	VV+OV+DV
P2 & P3	0.9977±0.0060	0.9963±0.0064	0.9951±0.0066
M2 & M4	0.9973±0.0056	0.9967±0.0063	0.9946±0.0081

Table 6: SSIM for VV only, VV+OV, and VV+OV+DV.

(initial) PDL. Here we consider three schemes: (1) VV only, (2) VV + OV, and (3) VV + OV + DV. The baseline is selected as fetching the whole PtCl without segmentation, also at the highest PDL.

Figures 18 and 19 show the results of VV+OV and VV+OV+DV for P2 and P3. Compared to VV only (Figure 15), VV+OV reduces the traffic volume by 4.95% (5.64% for P2 and 4.26% for P3), and further invoking DV brings an additional reduction of 5.15% (5.71% for P2 and 4.59% for P3). The results indicate the effectiveness of the synergy among the three optimizations. Meanwhile, when jointly applied with VV, OV and DV bring lower gains than when they are applied individually, because the set of cells that different optimizations deal with may overlap. For example, both VV and DV may reduce the density of the same cell but for different reasons – not appearing in the predicted viewport or far away from the viewer.

Figures 20, 21, and 22 present the results of VV, VV+OV, and VV+OV+DV, respectively, for M2 and M4. Compared to VV only, VV+OV shrinks the median data usage by 1.80%, and using all three optimizations brings that down by an additional 7.28%. VV is more effective on M4 than M2, because the performers in M4 are more scattered, making it less likely for the viewer to see the entire scene or most of it (§5.1). Overall, the above results indicate that jointly applying VV, OV, and DV yields a data saving of 41.8% and 39.5% on average (up to 70.1% and 84.5%) for M2/M4 and P2/P3 (Group H users), respectively, compared to the baseline.

Table 6 shows the average SSIM of VV, VV+OV, and VV+OV+DV, using the same ground truth as Table 5. We make two observations. First, the visual quality is consistently high (>0.99), indicating when jointly applied the optimizations incur almost no quality degradation. Second, SSIM does drop slightly when more optimizations are used. Overall, the results suggest that ViVo trades a negligible amount of visual quality loss for a considerable reduction of data usage.

8.4 Comparing Interaction Methods

Figure 23 compares the encoded bitrate between users in Groups H and P for videos P2 and P3, with all three optimizations enabled. The setup is the same as the "VV+OV+DV" configuration in §8.3. For P2 and P3, ViVo appears to be less effective on Group P (average saving 9.98%) than on Group H (average saving 40.0%). This is because viewport movement for users in Group H is more flexible and convenient than users in Group P. As a result, we find that users in Group H are more likely to look around "randomly", leading to viewports that contain only a small (or even zero) portion of the PtCl. This creates more opportunities for ViVo's optimizations for P2 and P3. However, such a scenario is less likely to occur in M2 and M4 where the performers are separated out. As shown in Figure 24, both groups exhibit similar reductions of encoded bitrate for M2 and M4, despite the different interaction methods between the two groups (§5.1). We also calculate the perceived visual quality and observe consistently high SSIM values (>0.99) when all optimizations are applied to users in Group P or H.

8.5 Performance on Commercial 5G Networks

We conduct experiments over a commercial mmWave 5G network to evaluate the performance of ViVo. We compare VV+OV+DV and the baseline described in §8.3 (using videos P3 and M2, unthrottled 5G connectivity, the highest initial PDL, one Group H user's viewport trace with an average viewport prediction accuracy). Our SGS10 phone has a line-of-sight (LoS) distance of ~50m to the 5G base station under a clear sky. We repeat each experiment 3 times. The results are qualitatively consistent with our findings in §8.3. For P3 (M2), on average, ViVo reduces data usage by 36% (39%). Meanwhile, ViVo exhibits fewer stalls than the baseline: 1.0s/min vs. 1.2s/min on average for P3, and 0.6s/min vs. 1.6s/min for M2. All playbacks' SSIM values are consistently high (>0.99). The above results validate that ViVo can indeed bring considerable data savings while maintaining good visual quality when operating on commercial 5G networks.

8.6 ViVo Performance under Limited/ Fluctuating Bandwidth

We now assess ViVo's performance when network throughput is limited and/or fluctuating. In this scenario, video quality may degrade as dictated by the rate adaptation module, and stalls may occur. Since there is little existing work on objective QoE metrics

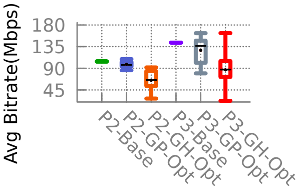

Figure 23: All Three: GH vs GP (P2 & P3).

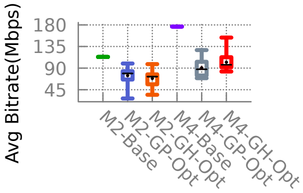

Figure 24: All Three: GH vs GP (M2 & M4).

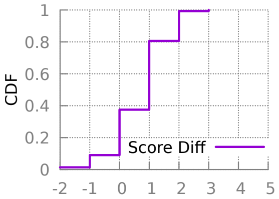

Figure 25: QoE: MOS.

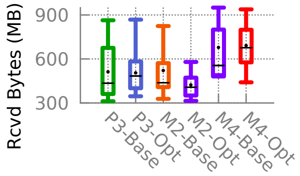

Figure 26: QoE: Bytes.

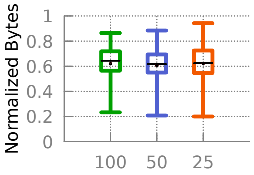

Figure 27: Different segmentations for M2.

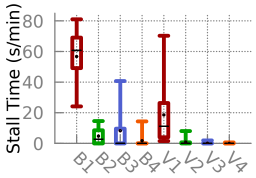

Figure 28: Stall time with different number of decoders.

for such complex scenarios in the context of volumetric videos, we resort to subjective metrics. We conduct another IRB-approved user study with 12 diverse participants. Note that this user study is different from the user trial described in §5.1 and the users here are a subset of those from the previous trial. We ask each participant to watch 12 playback *groups*. We define a group as (V, B, U). $V \in \{\text{P3, M2, M4}\}$ is the video (we exclude P1, the tutorial video, and P2, which is a bit too long); B is the network bandwidth trace randomly selected from the 10 traces described in §8.1; U is the viewport trace randomly selected from the 32 users' trajectories for V (described in §5.1).

Each group consists of a *pair* of playbacks of V, both conducted by replaying B (to emulate LTE bandwidth dynamics) and U (to reproduce a real viewer's viewport trajectory). The only difference is the streaming algorithm: one playback employs ViVo and another uses the baseline. For ViVo, we enable all three optimizations (VV, OV, and DV); the baseline fetches the entire PtCl without applying segmentation. Both schemes use the same throughput-based rate adaptation algorithm described in §7. We randomly order the two playbacks within each group, and thus a participant does not know which one is streamed by ViVo.

For each group, we ask the participants to compare the QoE of both playbacks by providing mean opinion scores (MOS), ranging from 1 to 5 (1=bad, 2=poor, 3=fair, 4=good, 5=excellent). Each of those 12 participants watches 12 groups (we assign 4 groups to each of the 3 videos). Therefore we collect 144 pairs of scores in total. Figure 25 plots the distribution of the score differences between ViVo and the baseline, where a positive (negative) value indicates that ViVo outperforms (underperforms) the baseline, across all 144 groups. The results indicate that for 62.5%, 9.0%, and 28.5% of the groups, ViVo yields a better, worse, and equal QoE compared to the baseline, respectively. Meanwhile, Figure 26 shows the data usage of ViVo across different sets of groups. When the bandwidth is limited and/or fluctuating, on average, ViVo fetches a similar amount of data compared to the baseline, but offers considerably better QoE as judged by the viewers. This is because ViVo's visibility-aware optimizations eliminate or reduce the point density for cells that are outside the viewport, occluded by others, or far-away from the viewer. ViVo uses the saved bandwidth to fetch cells that are visually more important at a higher PDL than the baseline.

8.7 Impact of Cell Size

So far all our experiments employ 100×100×100 cm^3 cells. We consider two additional segmentation schemes: 25×25×25 cm^3 and 50×50×50 cm^3. We use the "VV+OV+DV" setup in §8.3 to conduct the experiment, using the viewport trajectory traces of all 32 users (Groups H and P), unthrottled WiFi, and video M2 at the highest (initial) PDL. Figure 27 plots the impact of different cell sizes on data usage. A more fine-grained segmentation scheme does not reduce data usage: a small cell size allows the player to fetch fewer points, but such gains are canceled out by the incurred segmentation overhead as quantified in Table 4.

Figure 27 suggests that a small cell size increases the variation of data usage. To see the reason, consider two extreme cases. First, when the viewport contains the entire PtCl, having a small cell size incurs more data usage due to the segmentation overhead. Second, when the viewport only covers a tiny portion of the PtCl, having a small cell size may consume less data because of the fine-grained segmentation. We find the above observation applies to other videos and PDLs in general. Also, when the PDL is fixed, changing the cell size does not affect SSIM.

8.8 Impact of Multi-threaded Decoding

We now quantify how multi-threading (Table 3) affects ViVo's performance. We stream the four videos at their highest (initial) PDL over unthrottled WiFi, and measure the stall time (second per minute). Since the wireless network is not the bottleneck, a stall can be caused by only slow decoding. As shown in Figure 28, we evaluate eight configurations: B_k (1≤ k ≤4) is the baseline where the entire PtCl is streamed and decoded using k threads; V_k (1≤ k ≤4) represents ViVo with all three optimizations enabled and k threads for decoding the 100×100×100 cm^3 cells. We repeat each B_k for five times; for each V_k, we randomly pick five users (2 from Group P and 3 from Group H) and replay their viewport traces on SGS10.

We make two observations from Figure 28. First, using more threads helps reduce ViVo's stall time (*i.e.,* the decoding latency) – echoing our findings in Table 3. Second, ViVo significantly outperforms the baseline in terms of stall time: when using one thread, the 90-percentile stall time of the baseline and ViVo are 71.2 and 52.4 s/min respectively (7.95 and 0.02 s/min, respectively when using four threads). This is attributed to ViVo's optimizations that make the mobile client fetch and decode less content.

8.9 Energy and Resource Utilization

We measure the resource utilization by playing videos M2 (sparsest) and M4 (densest) using the "VV+OV+DV" setup in §8.3: unthrottled WiFi, the highest (initial) PDL, four decoding threads, and

SGS10. Starting with a fully charged phone, we repeatedly play M2 (M4) for 30 minutes and measure the resource utilization using the Snapdragon Profiler from Qualcomm [18]. The highest memory utilization during the playback is 358 MB (383 MB); the CPU utilization is less than 28% (38%), depending on how much content to decode; the GPU utilization is always less than 10% due to the small rendering overhead for PtCls. The highest temperature of the GPU and the overall device are 46°C and 50°C (53°C and 60°C), respectively. After the 30-minute playback, the battery life drops 5% (6%) from 100% for M2 (M4). Overall, we believe the above resource and energy consumption are acceptable. For all metrics above, the results of ViVo are better (lower) than those of the baseline scheme (fetching the entire PtCl). For example, when streaming M4 using the baseline, the CPU utilization is ∼60%.

9 DISCUSSION

Inter-Frame Compression. The current design of ViVo compresses each volumetric frame individually without considering the compressibility across frames. We plan to explore efficient solutions for inter-frame compression to further improve ViVo's bandwidth utilization. For instance, Kammerl *et al.* [39] extend the octree data structure to perform real time inter-frame compression for PtCl streams. However, inter-frame compression may increase decoding overhead on the client side, in particular for mobile devices with weak computational power. Another challenge is that inter-frame compression requires dividing a video into group-of-frames, which are typically the smallest fetching units. As a result, even if a predicted viewport overlaps with a single frame of a certain cell, we need to download the cell's entire group that the frame belongs to in order to properly decode it. This may reduce bandwidth saving or even increase data usage compared to the current design of ViVo. We will carefully study the above tradeoff in our future work.

3D Mesh Volumetric Videos. We can potentially apply the high-level ideas of ViVo (*i.e.,* VV, OV, and DV) to 3D mesh based volumetric video streaming where each frame consists of meshes (polygons) instead of points. In the computer graphics community, there have been several efforts relevant to streaming polygon models [32] such as view frustum culling [58], collision detection between volumetric meshes [62], and view-dependent simplification for rendering triangle meshes [29]. Nevertheless, a key challenge here is that the computation complexity of processing 3D mesh is typically much higher than that of PtCl. As a result, it might be difficult, if not impossible, to directly apply the above work to a mobile volumetric video streaming system. More research is therefore needed to develop lightweight and efficient schemes for streaming 3D mesh based volumetric videos.

Deep Learning for Viewport Prediction. We plan to investigate whether ViVo can benefit from deep learning (DL) such as long short-term memory (LSTM) [34] for more accurate viewport prediction, which will in turn improve the efficiency of the three visibility-aware optimizations. Indeed, the inherent complexity of 6DoF movement makes DL a promising approach for capturing the hidden and sophisticated patterns of viewport movement. However, DL typically requires more compute resources than traditional machine learning such as those currently used by ViVo. As a result, we may need to execute the inference on the server side or on an edge proxy.

10 RELATED WORK

Volumetric Video Streaming. Volumetric videos represent an emerging form of multimedia content, and few studies have investigated them. Park *et al.* [53] propose a greedy algorithm for volumetric media streaming. DASH-PC [35] proposes a manifest file format following the DASH standard for volumetric video streaming. AVR [57] streams and analyzes PtCl data between vehicles to enhance autonomous driving. Nebula [55] is a recent proposal that leverages edge cloud to transcode volumetric videos into 2D regular videos. To the best of our knowledge, ViVo is the first practical visibility-aware volumetric video streaming system for mobile devices.

QoE Metrics remain an open problem for volumetric video streaming, although they have been thoroughly studied for regular videos. Existing work in this category focuses on the quality of static 3D models. For instance, Alexiou and Ebrahimi [23] introduce an objective metric to measure the quality degradation of distorted PtCls. Dumic *et al.* [28] offer a nice survey on both subjective and objective quality assessments for 3D video sequences. In this paper, we evaluate the performance of ViVo using both objective metrics such as SSIM [64] and the subjective mean opinion score (MOS).

360° Video and VR Streaming. 360° panoramic video streaming has been a hot research topic recently. Researchers have developed several viewport-adaptive systems such as Rubiks [33] and Flare [56]. There also exist several VR streaming systems such as Furion [45] and Liu *et al.* [48]. ViVo instead applies viewport adaptation to 3D volumetric content, and proposes volumetric-specific optimizations such as OV and DV. It is important to note that volumetric videos and 360° panoramic videos are different types of video content. Due to its 3D nature, volumetric video streaming faces unique challenges compared to 360° videos as detailed earlier in the paper. Therefore, designing a volumetric video streaming system such as ViVo requires far more efforts than simply extending or modifying existing 360° video streaming systems.

11 CONCLUDING REMARKS

Through ViVo, we demonstrate the feasibility of streaming volumetric videos to commodity mobile devices without additional infrastructural support such as cloud offloading/acceleration. We also showcase how visibility awareness helps make volumetric video streaming bandwidth-efficient. ViVo reveals the synergy among mobile computing, video streaming, and computer graphics for facilitating the delivery of emerging volumetric content. We hope our study will offer insights for follow-up research that further improves visual quality, resource efficiency, and usability of volumetric video streaming, as well as motivate novel mobile applications that make use of volumetric content. We plan to release our volumetric videos and viewport trajectory datasets to the community.

12 ACKNOWLEDGEMENTS

We appreciate the anonymous reviewers and our shepherd for their valuable comments. We thank Jarrell Pair for capturing the volumetric videos and Cheuk Yiu Ip for insightful discussions. We thank the voluntary users who participated in our user study, as well as Arvind Narayanan for helping with the 5G experiments. Feng Qian's research was supported in part by NSF Award #1903880 and #1915122.

REFERENCES

[1] 8i introduces fully volumetric 3D video. https://www.youtube.com/watch?v=aO3TAke7_MI.

[2] ARM big.LITTLE. https://en.wikipedia.org/wiki/ARM_big.LITTLE.

[3] AT&T Continues to Lead in Bringing 5G Experiences to Life. https://about.att.com/newsroom/2018/5g_demo_2018.html.

[4] Create holograms from real life. https://www.microsoft.com/en-us/mixed-reality/capture-studios.

[5] Draco 3D Data Compression. https://google.github.io/draco/.

[6] Google VR – Fundamental Concepts. https://developers.google.com/vr/discover/fundamentals.

[7] Google's 'Welcome to Light Fields' VR App Reveals the Power of Volumetric Capture. https://www.roadtovr.com/googles-welcome-to-lightfields-vr-app-reveals-the-power-of-volumetric-capture/.

[8] How Microsoft records Holographic video content for the HoloLens. https://www.youtube.com/watch?v=kZ-XZIV-o8s.

[9] Intel RealSense Technology. https://www.intel.com/content/www/us/en/architecture-and-technology/realsense-overview.html.

[10] Intel Studios Debut Volumetric Video Demo. https://www.youtube.com/watch?v=9qd276AJg-o.

[11] Kinect for Windows. https://developer.microsoft.com/en-us/windows/kinect.

[12] Limited Error Point Cloud Compression. https://github.com/Esri/lepcc.

[13] Limited Error Raster Compression. https://github.com/Esri/lerc.

[14] Magic Leap One. https://www.magicleap.com/magic-leap-one.

[15] NFL, Verizon Team Up On 5G Development. https://www.tvtechnology.com/news/nfl-verizon-team-up-on-5g-development.

[16] Point Cloud Library (PCL). http://pointclouds.org/.

[17] scikit-learn (Machine Learning in Python). https://scikit-learn.org/.

[18] Snapdragon Profiler: A product of Qualcomm Technologies, Inc. https://developer.qualcomm.com/software/snapdragon-profiler.

[19] View Frustum Culling. http://www.lighthouse3d.com/tutorials/view-frustum-culling/.

[20] Viewing Frustum Culling. https://en.wikipedia.org/wiki/Hidden-surface_determination#Viewing-frustum_culling.

[21] Volumetric video is so much more than VR. https://www.immersiveshooter.com/2019/01/10/volumetric-video-means-so-much-more-than-vr/.

[22] Volumetric Video Market by Volumetric Capture & Content Creation (Hardware (Camera & Processing Unit), Software, and Services), Application (Sports & Entertainment, Medical, Signage, Education & Training), and Geography - Global Forecast to 2023. https://www.marketsandmarkets.com/Market-Reports/volumetric-video-market-259585041.html.

[23] E. Alexiou and T. Ebrahimi. Point Cloud Quality Assessment Metric Based on Angular Similarity. In *Proceedings of IEEE ICME*, 2018.

[24] Y. Bao, H. Wu, T. Zhang, A. A. Ramli, and X. Liu. Shooting a moving target: Motion-prediction-based transmission for 360-degree videos. In *Proceedings of Big Data 2016*, 2016.

[25] D. Chen, Y.-J. Chiang, and N. Memon. Lossless Compression of Point-Based 3D Models. In *Proceedings of the 13th Pacific Conference on Computer Graphics and Applications*, 2005.

[26] E. Cuervo, A. Wolman, L. P. Cox, K. Lebeck, A. Razeen, S. Saroiu, and M. Musuvathi. Kahawai: High-Quality Mobile Gaming Using GPU Offload. In *Proceedings of ACM MobiSys*, 2015.

[27] O. Devillers and P.-M. Gandoin. Geometric Compression for Interactive Transmission. In *Proceedings of IEEE Visualization*, 2000.

[28] E. Dumic, C. R. Duarte, and L. A. da Silva Cruz. Subjective Evaluation and Objective Measures for Point Clouds – State of the Art. In *Proceedings of International Colloquium on Smart Grid Metrology*, 2018.

[29] J. El-Sana, E. Azanli, and A. Varshney. Skip Strips: Maintaining Triangle Strips for View-Dependent Rendering. In *Proceedings of IEEE Visualization*, 1999.

[30] M. Garon, K. Sunkavalli, S. Hadap, N. Carr, and J.-F. Lalonde. Fast Spatially-Varying Indoor Lighting Estimation. In *Proceedings of IEEE CVPR*, 2019.

[31] T. Golla and R. Klein. Real-time Point Cloud Compression. In *Proceedings of International Conference on Intelligent Robots and Systems*, 2015.

[32] A. Guéziec, G. Taubin, B. Horn, and F. Lazarus. A Framework for Streaming Geometry in VRML. *IEEE Computer Graphics and Applications*, 19(2):68–78, 1999.

[33] J. He, M. A. Qureshi, L. Qiu, J. Li, F. Li, and L. Han. Rubiks: Practical 360-Degree Streaming for Smartphones. In *Proceedings of ACM MobiSys*, 2018.

[34] S. Hochreiter and J. Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 1997.

[35] M. Hosseini and C. Timmerer. Dynamic Adaptive Point Cloud Streaming. In *Proceedings of ACM Packet Video*, 2018.

[36] Y. Huang, J. Peng, C.-C. J. Kuo, and M. Gopi. A Generic Scheme for Progressive Point Cloud Coding. *IEEE Trans. on Vis. and Computer Graphics*, 14(2):440–453, 2008.

[37] E. Hubo, T. Mertens, T. Haber, and P. Bekaert. The Quantized kd-Tree: Efficient Ray Tracing of Compressed Point Clouds. In *Proceedings of IEEE Symposium on Interactive Ray Tracing*, 2006.

[38] J. Jiang, V. Sekar, and H. Zhang. Improving Fairness, Efficiency, and Stability in HTTP-based Adaptive Video Streaming with FESTIVE. In *Proceedings of ACM CoNEXT*, 2012.

[39] J. Kammerl, N. Blodow, R. B. Rusu, S. Gedikli, M. Beetz, and E. Steinbach. Real-time Compression of Point Cloud Streams. In *Proceedings of International Conference on Robotics and Automation*, 2012.

[40] B. Karlik and A. Vehbi. Performance Analysis of Various Activation Functions in Generalized MLP Architectures of Neural Networks. *International Journal of Artificial Intelligence and Expert Systems*, 1(4):111–122, 2010.

[41] S. Katz and A. Tal. On the Visibility of Point Clouds. In *Proceedings of IEEE ICCV*, 2015.

[42] S. Katz, A. Tal, and R. Basri. Direct Visibility of Point Sets. In *Proceedings of ACM SIGGRAPH*, 2007.

[43] L. Kobbelt and M. Botsch. A Survey of Point-Based Techniques in Computer Graphics. *Computers & Graphics*, 28(6):801–814, 2004.

[44] M. Kowalski, J. Naruniec, and M. Daniluk. LiveScan3D: A Fast and Inexpensive 3D Data Acquisition System for Multiple Kinect v2 Sensors. In *Proceedings of International Confeerence on 3D Vision*, 2015.

[45] Z. Lai, Y. C. Hu, Y. Cui, L. Sun, and N. Dai. Furion: Engineering High-Quality Immersive Virtual Reality on Today's Mobile Devices. In *Proceedings of ACM MobiCom*, 2017.

[46] J.-M. Lien, G. Kurillo, and R. Bajcsy. Multi-camera tele-immersion system with real-time model driven data compression. *The Visual Computer*, 26(3):3–15, 2010.

[47] D. C. Liu and J. Nocedal. On the limited memory BFGS method for large scale optimization. *Mathematical Programming*, 45(1–3):503–528, 1989.

[48] L. Liu, R. Zhong, W. Zhang, Y. Liu, J. Zhang, L. Zhang, and M. Gruteser. Cutting the Cord: Designing a High-quality Untethered VR System with Low Latency Remote Rendering. In *Proceedings of ACM MobiSys*, 2018.

[49] A. Maglo, G. Lavoué, F. Dupont, and C. Hudelot. 3D Mesh Compression: Survey, Comparisons, and Emerging Trends. *ACM Computing Surveys*, 47(3), 2015.

[50] R. Mehra, P. Tripathi, A. Sheffer, and N. J. Mitra. Visibility of Noisy Point Cloud Data. *Computers and Graphics*, 34(3):219–230, 2010.

[51] R. Mekuria, K. Blom, and P. Cesar. Design, Implementation and Evaluation of a Point Cloud Codec for Tele-Immersive Video. *IEEE Trans. on Circuits and Systems for Video Technology*, 27(4):828–842, 2017.

[52] S. Orts-Escolano, C. Rhemann, S. Fanello, W. Chang, A. Kowdle, Y. Degtyarev, D. Kim, P. Davidson, S. Khamis, M. Dou, V. Tankovich, C. Loop, Q. Cai, P. Chou, S. Mennicken, J. Valentin, V. Pradeep, S. Wang, S. B. Kang, P. Kohli, Y. Lutchyn, C. Keskin, and S. Izadi. Holoportation: Virtual 3D Teleportation in Real-time. In *Proceedings of ACM UIST*, 2016.

[53] J. Park, P. A. Chou, and J.-N. Hwang. Volumetric Media Streaming for Augmented Reality. In *Proceedings of IEEE GLOBECOM*, 2018.

[54] J. Peng, C.-S. Kim, and C.-C. J. Kuo. Technologies for 3D mesh compression: A survey. *Journal of Visual Communication and Image Representation*, 16(6):688–733, 2005.

[55] F. Qian, B. Han, J. Pair, and V. Gopalakrishnan. Toward Practical Volumetric Video Streaming On Commodity Smartphones. In *Proceedings of ACM HotMobile*, 2019.

[56] F. Qian, B. Han, Q. Xiao, and V. Gopalakrishnan. Flare: Practical Viewport-Adaptive 360-Degree Video Streaming for Mobile Devices. In *Proceedings of ACM MobiCom*, 2018.

[57] H. Qiu, F. Ahmad, F. Bai, M. Gruteser, and R. Govindan. Augmented Vehicular Reality. In *Proceedings of ACM MobiSys*, 2018.

[58] S. Rusinkiewicz and M. Levoy. QSplat: A Multiresolution Point Rendering System for Large Meshes. In *Proceedings of ACM SIGGRAPH*, 2000.

[59] R. Schnabel and R. Klein. Octree-based Point-Cloud Compression. In *Proceedings of the 3rd Eurographics / IEEE VGTC conference on Point-Based Graphics*, 2006.

[60] J. L. Schonberger and J.-M. Frahm. Structure-From-Motion Revisited. In *Proceedings of IEEE CVPR*, 2016.

[61] S. Schwarz, M. Preda, V. Baroncini, M. Budagavi, P. Cesar, P. A. Chou, R. A. Cohen, M. Krivokuća, S. Lasserre, Z. Li, J. Llach, K. Mammou, R. Mekuria, O. Nakagami, E. Siahaan, A. Tabatabai, A. M. Tourapis, and V. Zakharchenko. Emerging MPEG Standards for Point Cloud Compression. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 9(1):133–148, 2019.

[62] M. Tang, D. Manocha, S.-E. Yoon, P. Du, J.-P. Heo, and R.-F. Tong. VolCCD: Fast Continuous Collision Culling between Deforming Volume Meshes. *ACM Transactions on Graphics*, 30(5):111:1–111:15, 2011.

[63] T.-C. Wang, A. A. Efros, and R. Ramamoorthi. Occlusion-aware Depth Estimation Using Light-field Cameras. In *Proceedings of IEEE ICCV*, 2015.

[64] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. Image quality assessment: From error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, 2004.

[65] A. Woo. Fast Ray-Box Intersection. In *Graphics Gems*, pages 395–396. Academic Press, 1990.