MetaStream: Live Volumetric Content Capture, Creation, Delivery, and Rendering in Real Time

Yongjie Guan*
New Jersey Institute of Technology
yg274@njit.edu

Xueyu Hou*
New Jersey Institute of Technology
xh29@njit.edu

Nan Wu
George Mason University
nwu5@gmu.edu

Bo Han
George Mason University
bohan@gmu.edu

Tao Han
New Jersey Institute of Technology
tao.han@njit.edu

Abstract

While recent work explored streaming volumetric content on-demand, there is little effort on live volumetric video streaming that bears the potential of bringing more exciting applications than its on-demand counterpart. To fill this critical gap, in this paper, we propose MetaStream, which is, to the best of our knowledge, the first practical live volumetric content capture, creation, delivery, and rendering system for immersive applications such as virtual, augmented, and mixed reality. To address the key challenge of the stringent latency requirement for processing and streaming a huge amount of 3D data, MetaStream integrates several innovations into a holistic system, including dynamic camera calibration, edge-assisted object segmentation, cross-camera redundant point removal, and foveated volumetric content rendering. We implement a prototype of MetaStream using commodity devices and extensively evaluate its performance. Our results demonstrate that MetaStream achieves low-latency live volumetric video streaming at close to 30 frames per second on WiFi networks. Compared to state-of-the-art systems, MetaStream reduces end-to-end latency by up to 31.7% while improving visual quality by up to 12.5%.

CCS Concepts

• **Human-centered computing** → **Mobile computing**; Visualization systems and tools; • **Computing methodologies** → **Volumetric models**.

ACM MobiCom '23, October 2–6, 2023, Madrid, Spain

ACM ISBN 978-1-4503-9990-6/23/10…$15.00
https://doi.org/10.1145/3570361.3592530

Keywords

live volumetric video streaming, multi-camera calibration, point cloud synthesis, volumetric content rendering

ACM Reference Format:
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 *The 29th Annual International Conference on Mobile Computing and Networking (ACM MobiCom '23), October 2–6, 2023, Madrid, Spain.* 15 pages. https://doi.org/10.1145/3570361.3592530

1 Introduction

Volumetric videos enable six degrees of freedom (6DoF) motion for viewers by modeling (physical) objects as point clouds or 3D meshes [36, 48]. When watching a volumetric video, users can freely explore its content in 3D space by changing not only viewing directions, which is supported by 360° videos [37, 65], but also, more importantly, viewpoints (*i.e.,* translational position in 3D space), which is a unique feature of volumetric content. Thus, volumetric content can be integrated into virtual, augmented, and mixed reality (VR/AR/MR) applications to offer a truly immersive user experience [59]. As the key component of holographic communication [26] that is envisaged for 6G [70, 71], the capture, creation, delivery, and rendering of volumetric content (*i.e.,* volumetric video streaming) has registered numerous applications in healthcare, education, entertainment, *etc.*

Existing work on volumetric video streaming [33, 34, 36, 48, 60, 64, 72, 82, 83] mainly focused on video on demand (VOD) that streams *pre-recorded* content, and there is little effort on live streaming that simultaneously captures and delivers volumetric content in real-time. Different from VOD, live streaming can facilitate more exciting use cases. For example, a surgeon can operate on remote patients via their live volumetric content feed to support telesurgery [25], saving people's lives on battlefields and in underdeveloped areas.

*These authors contributed equally to this work.

Video	Cameras	Data Volume	Content Creation	Rendering Overhead	DoF
2D	Single	Low	–	Low	–
360°	Multi.	Medium	2D Stitch	Low	3DoF
Volumetric	Multi.	High	3D Synthesis	High	6DoF

Table 1: Comparison of live streaming for different types of videos, including regular 2D, 360°, and volumetric videos.

We list the key challenges of live volumetric video streaming, by comparing different types of live streaming including regular 2D, 360°, and volumetric videos in Table 1. Among them, volumetric video streaming provides the richest amount of information of the delivered content by enabling 6DoF motion and providing a truly immersive experience, at the cost of high computation and communication overhead. For example, volumetric video capture requires a multi-camera setup and generates a high volume of data to process; it demands 3D content synthesis and rendering that consumes much higher computing resources than 360° videos; and the required bandwidth for streaming volumetric content is much higher than 2D and 360° videos. Thus, it is extremely challenging to design and implement a practical live volumetric video streaming system under the constraints of off-the-shelf commodity devices and today's Internet.

In this paper, we propose MetaStream, a first-of-its-kind live volumetric content capture, creation, delivery, and rendering system for enhancing the user experience of immersive applications. It consists of three key components: cameras equipped with computation resources (referred to as smart cameras) to capture and pre-process RGB and depth frames with the goal of reducing the transmitted data, a server to optimize the computation overhead of creating high-quality 3D content and decrease the content-generation latency, and an MR client to render, in real-time, live volumetric content for seamless integration with the surrounding environment. In a nutshell, MetaStream achieves low-latency live volumetric video streaming at close to 30 frames per second (FPS) by intelligently balancing the trade-offs between computation overhead, network resource utilization, and visual quality of volumetric content. The visual result of MetaStream is shown in Fig. 1. Our design of MetaStream involves the following innovations that make it practical.

Adaptivity to Practical Scenarios via Dynamic Camera Calibration. In multi-camera volumetric video streaming systems, camera calibration is a critical component that directly affects the performance of the whole streaming session [63]. Previous work calibrated cameras in a naive way (*e.g.,* predefined markers [41, 46, 57]), and no movement of cameras would be allowed after calibration. However, practical volumetric video streaming systems often involve moving objects in a wide range, which makes fixed cameras not applicable. For example, a person walking around in a large room cannot be always captured by fixed cameras. Moreover, there is an increasing demand for deploying cameras on mobile platforms such as autonomous vehicles and drones. In such scenarios, the fixed-camera assumption no longer holds. To resolve such a challenge, MetaStream develops a lightweight online calibration method that allows the dynamic movement of cameras during streaming. By adopting ORB feature extraction [67] and tracking algorithms [58], our calibration method can accurately update the cameras' 6DoF pose during the streaming session.

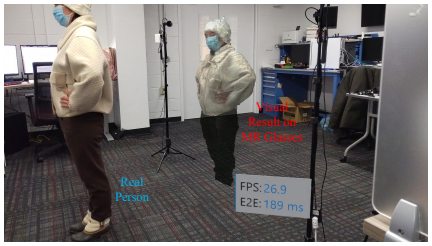


Figure 1: Visual Results of MetaStream on Microsoft HoloLens 2 [8].

Collaborative Edge Design for Low-latency Content Capture and Creation. Existing multi-camera volumetric video streaming systems suffer from large streaming data size and high computation overhead for the point cloud generation, which requires high network bandwidth and powerful machines [41, 46, 59]. However, to enable the movement of cameras, captured content should be wirelessly transmitted to the server. MetaStream presents a collaborative edge pipeline for volumetric video streaming that effectively distributes computing loads across smart cameras and the server, significantly reducing computing pressure on the server and decreasing the streaming data size. We also design selective segmentation on smart cameras to intelligently segment out target objects from complex backgrounds with low overhead. The transmission delay is largely reduced by locally segmenting target objects on smart cameras.

Efficient Point Cloud Synthesis by Removing Redundant Data. Though multiple cameras are required to capture a complete high-quality point cloud, their fields of view (FoVs) could overlap with one another, which generates a large portion of redundant points in the overlapped regions. Thus, we propose a cross-camera redundancy removal method that deletes the redundant points. In this way, we not only improve the visual quality of the complete point cloud but also reduce the transmitting data size between the server and the client.

Foveated Rendering of Point Clouds on MR Devices. Head-mounted displays (HMDs) are becoming a crucial factor in bringing users an immersive experience in AR/MR

systems. Motivated by such a trend, we optimize the rendering performance for users (*i.e.,* clients who receive and play the streamed volumetric content) with MR HMDs such as Microsoft HoloLens 2 [8]. Specifically, we propose an empirical foveated rendering method that adaptively renders volumetric content with different point densities (and thus visual qualities) based on the user's foveal area. Additionally, we utilize the user's motion as the reference to adaptively decide the normal of 2D squares for rendered 3D points. With all these proposed techniques, MetaStream achieves high-quality rendering with neglectable overhead.

We build a prototype implementation of MetaStream using commodity devices and thoroughly evaluate its performance via live controlled experiments. We summarize our evaluation results as follows.

- Comparing its performance and a state-of-the-art system LiveScan3D [46] on different edge devices and in various wireless network environments, MetaStream keeps nearly 30 FPS in all conditions while the average FPS of LiveScan3D is only 14.3 FPS (on Jetson Nano [14]).
- With the proposed modules in MetaStream, the visual quality of the delivered volumetric content that is rendered on the MR client is 6.84% to 12.5% better than LiveScan3D.
- Compare to LiveScan3D, MetaStream reduces the end-to-end latency, a key metric of live video streaming, by 31.7%.

2 Background and Motivation

2.1 Background

Volumetric Content Capture and Creation. Point cloud and 3D mesh are two popular representations of volumetric content. A point cloud is essentially a set of unstructured 3D points with color and/or intensity [31]. 3D mesh models an object using not only vertices but also edges and faces to form polygons [55, 62]. Different from 3D mesh, the point cloud is more flexible and easier to manipulate and is thus the focus of this work. Volumetric content can be captured by RGB-D cameras (*e.g.,* Intel RealSense [5] and Microsoft Kinect [7]) and various LiDAR scanners [66]. These devices acquire 3D data by leveraging the time of flight (*i.e.,* calculating depth based on the speed of light) or structured light (*i.e.,* light with a known pattern). In order to get colorful point clouds for volumetric content delivery, we need to merge RGB and depth images (from multiple cameras) to construct 3D models via proper synchronization, calibration, and filtering.

Volumetric Video Streaming is an emerging research topic, in particular to the networking community. We can classify the state-of-the-art into two categories: *direct streaming* and *transcoded streaming*. The former fetches the encoded 3D models, either in their full form or segmented parts, before decoding and rendering them [36, 60], whereas the latter

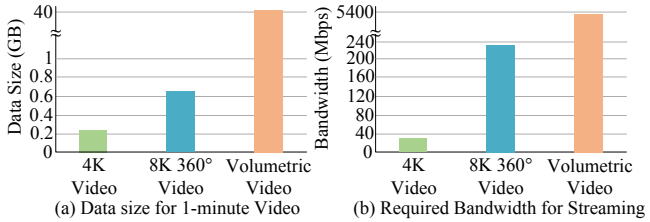


Figure 2: Comparison of 4K, 8K 360°, and volumetric videos.

performs real-time *transcoding* (*e.g.,* at the edge), which renders 3D scenes/objects into 2D images based on users' (predicted) 6DoF pose and streams transcoded 2D video to the client [34, 64]. While direct streaming offers superior scalability (*i.e.,* the server is stateless) and user interaction, its downside is the high bandwidth utilization and on-device decoding overhead caused by the delivery and processing of 3D data. Since the client receives and decodes 2D content, the network and client-side overheads for transcoded streaming are dramatically reduced, becoming independent of the complexity of 3D scenes. However, it has two limitations, heavyweight 3D-to-2D transcoding and potential distortion of displayed content due to inaccurate viewport prediction.

Existing Live Volumetric Video Streaming Systems. LiveScan3D [46] is an open-source system for live, 3D data acquisition using multiple Kinect v2 sensors. Benefiting from pre-set makers, it allows the user to place sensors in any configuration and gather data in near real-time. It has two main components: LiveScanServer and LiveScanClient. Each LiveScanClient is equipped with a desktop computer and a Kinect sensor. It can capture and stream the created point cloud to the LiveScanServer. The LiveScanServer manages all LiveScanClients simultaneously. It can merge and compress point clouds from different LiveScanClients based on the spatial position of their sensor. Holoportation [59] is another system that directly connects cameras to the server. By using powerful GPUs, it can reconstruct a 3D model of humans in real-time and transmit it to headsets for display [8]. Hu *et al.* [40] combine point cloud capturing and streaming as an entire pipeline and build a prototype, which contains a depth sensor, an edge-computing device, and a smartphone. Project Starline [47] utilizes a static setup to capture mainly the upper body of a stationary person, which requires multiple powerful GPUs and custom-built hardware (*i.e.,* not compatible with mobile headsets such as HoloLens 2) to create and display the 3D representation of captured users.

2.2 Motivation

We next demonstrate the challenges of live volumetric video streaming over wireless networks and analyze the limitations of state-of-the-art systems. MetaStream is proposed

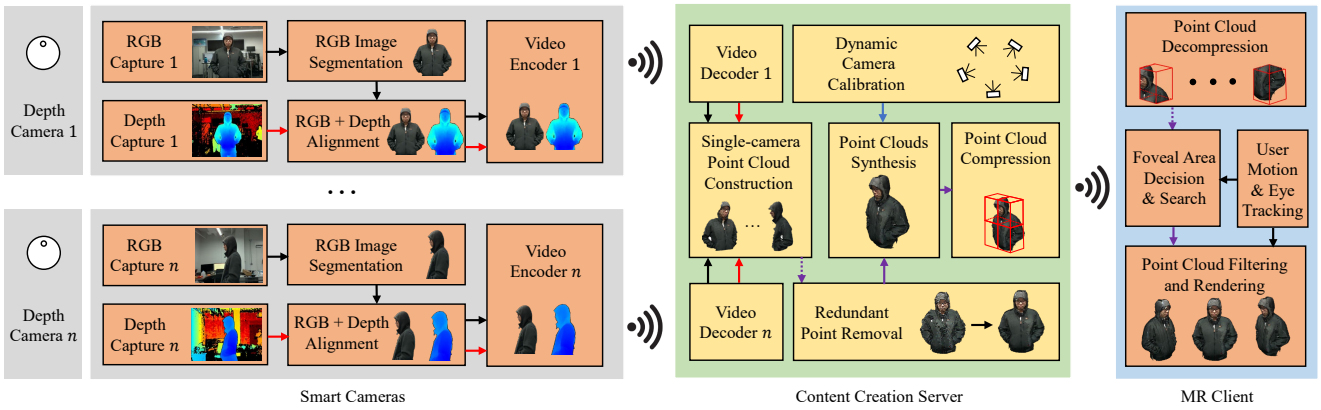


Figure 3: System architecture and workflow of MetaStream.

to resolve them and realize high-quality live streaming on commodity networks and devices.

One of the most recent video types that has been studied for live streaming is 360° video. However, 360° video is not true 3D content and supports only 3DoF motion (rotation on x, y, z axes), and thus cannot fully take advantage of the unique features of MR/AR headsets. For example, it does not allow users to observe an object in 6DoF with the headsets. As shown in Fig. 2, the key challenge of streaming point cloud-based volumetric videos is the large data size. We compare point-cloud videos with 4K videos and 8K 360° videos on the data size of a one-minute video in Fig. 2 (a) and the required bandwidth for streaming in Fig. 2 (b). A one-minute point-cloud video at 30 FPS requires over 60 times the data size compared to an 8K 360° video and 200 times compared to a 4K video, both at the same frame rate. To guarantee live video streaming at 30 FPS, the required bandwidth of a raw point-cloud video is almost 26 times over an 8K 360° video and 216 times over a 4K video. Due to such a large data size, point cloud-based live volumetric video streaming is more challenging than its 2D and 360° counterparts.

Among existing systems of live volumetric video streaming, Holoportation [59] does not optimize the transmission overhead of high-quality 3D content and requires >1Gbps network bandwidth. Hu *et al.* [40] design a system that streams from a single camera to a mobile phone, which does not consider point cloud synthesis from multiple cameras. Starline [47] focuses on capturing and streaming the front view of the upper body of a user, who sits next to the display. Thus, we could not conduct apple-to-apple comparisons with them without modifying their design. In this work, we choose LiveScan3D [46] as the baseline for comparison.

We observe three key limitations in LiveScan3D [46], which motivate the design of MetaStream. First, as directly transmitting point clouds to the content server consumes high bandwidth, it is challenging to deploy LiveScan3D on WiFi networks, especially for the multi-camera setup. Second, due to the constraint of the segmentation method in LiveScan3D [46], other objects close to the captured object may be kept in the segmentation results by mistake. Third, as there are overlapped areas of the FoVs of cameras, the merged point cloud contains redundant points, which not only increases transmission data size but also reduces the visual quality of displayed content.

3 MetaStream Overview

MetaStream is a live volumetric content capture, creation, delivery, and rendering system that realizes high-quality video streaming for immersive applications. We depict its system architecture and workflow in Fig. 3. Compared to existing systems [40, 46, 47, 59], the design of MetaStream addresses the following challenges that exist in practice.

- MetaStream significantly reduces the high transmission delay caused by the large data size. By exploiting the computing capacity of lightweight edge devices, it dispatches pre-processing loads on camera-side edge devices and filters out redundant RGB and depth data around the target object(s) before transmission. Hence, the transmission data size from cameras to the server is largely reduced, significantly decreasing the transmission delay.
- MetaStream effectively reduces pre-processing overhead on the camera side without sacrificing the overall performance. By adaptively selecting frames for segmentation based on cross-frame differences, MetaStream keeps accurate and fast deep learning-based segmentation on smart cameras with limited computing resources.
- MetaStream boosts content creation efficiency. With the RGB and depth data of only the target object(s), it takes less time on the server to create the point clouds and synthesize them into a complete one because the number of points for processing is drastically reduced.
- MetaStream improves the applicability for practical scenarios where cameras may move and follow the target object (*e.g.,* a person walking around in a room) during streaming. A

lightweight yet effective dynamic camera calibration method is proposed by extracting and tracking ORB features [44] from the environment to realize online camera-pose updates.
• MetaStream accelerates point cloud rendering on MR/AR devices whose computing resources are limited. It develops an adaptive foveated rendering technique to significantly reduce the rendering overhead with a neglectable sacrifice of user experience. The rendering overhead is further reduced by limiting the search space for the foveated area.

With all the above techniques, MetaStream realizes real-time live volumetric video streaming over a WiFi network.

4 System Design of MetaStream

4.1 Smart Cameras

As shown in Fig. 3, a smart camera consists of two parts: a depth camera (*e.g.,* Intel L515 [4]) and an edge computing device (*e.g.,* Jetson Nano [14]). The edge device keeps a persistent wireless connection with the server. Though such a setup can be easily created in environments such as offices and classrooms, the following challenges exist.
• As the target object (*e.g.,* a person) can move around in a large space, the cameras need to move to follow the object, which is not yet supported by existing work [3, 10, 46, 59].
• The camera side of a live volumetric video streaming system involves multiple tasks including RGB/depth frames capturing, object segmentation, and video encoding [46]. To make the system design practical, MetaStream supports low-cost portable edge devices on the camera-side, which has limited computing resources (*e.g.,* only 128 CUDA cores on Jetson Nano [14]) compared to desktop computers (*e.g.,* 3,584 CUDA cores on Nvidia RTX 3060).

We address the above challenging issues with the following methods.
• We design a dynamic camera calibration scheme by leveraging a lightweight ORB feature extraction and tracking algorithm [58], which automatically updates the cameras' 6DoF pose with neglectable overhead during streaming.
• We decrease the transmission data size by segmenting RGB and depth frames. To reduce the computational cost of segmentation, we design selective segmentation that adaptively segments frames with low similarity to their neighboring keyframes and approximates the others with the segmentation results of keyframes. The segmented RGB and depth[1] data are encoded with 2D video encoder (*e.g.,* H.264), which further reduces transmission data size. Note that we prefer 2D video encoder rather than RGB-D data compression methods [43, 75] because their current implementations are expensive and cannot run in real-time on edge devices (*e.g.,* ~0.4s per frame on Jetson Nano [14] with 720p resolution).

[1]The gray-scale depth frames are converted to RGB frames before encoding.

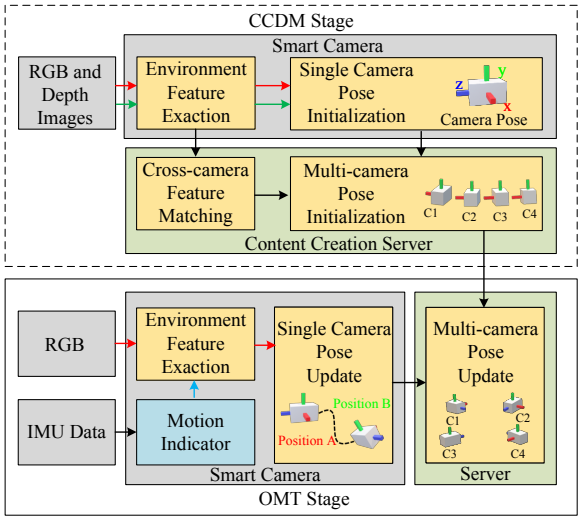


Figure 4: Dynamic Camera Calibration.

Dynamic Camera Calibration. Existing volumetric video capturing systems [3, 10, 46, 59] fix the cameras' positions, which limits their adaptivity to practical scenarios. We propose Dynamic Camera Calibration (DCC) in MetaStream to address this limitation. The DCC module dynamically obtains and manages the camera pose in real-time and calibrates cameras without any pre-defined makers [46].

As shown in Fig. 4, DCC includes two stages: cross-camera distance measurement (CCDM) and online movement tracking (OMT). The CCDM stage is activated when the system is set up. It is a one-time effort and obtains the initial relative positions across cameras. The OMT stage starts after the CCDM stage and keeps active during streaming. It continuously updates the relative displacement of each camera when it moves. By combining the relative displacement with the initial relative position, the system obtains the poses of all cameras in real time. Note that we have two assumptions about the DCC module. First, the moving speed should not exceed a certain limit to ensure the calibration performance (*e.g.,* <4.2m/s in our current design). Second, the scene should contain rich feature points. Nevertheless, such assumptions can be naturally satisfied in most real-world scenarios.
CCDM Stage. Once all smart cameras are connected to the server, the CCDM stage of DCC starts. The smart cameras are initially placed facing one direction, making their FoVs overlap with each other. Then, they begin to extract ORB features [67] from their FoVs and send them to the server. By designating the position of one camera as the reference, the server calculates the relative spatial positions of other cameras by matching their features [44]. In this way, the server obtains the initial positions of all cameras in the same coordinate system. Note that for the camera that is selected as the reference, its initial position is set to the origin of the coordinate system.

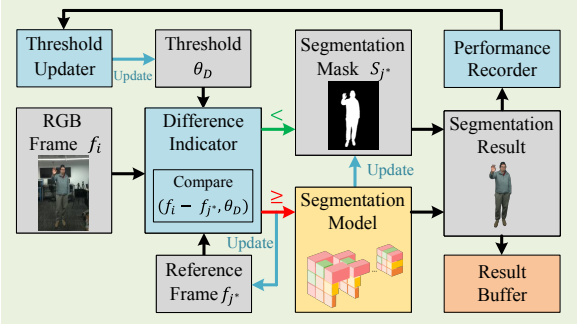


Figure 5: Selective Segmentation.

OMT Stage. Once the cameras' initial positions are calculated on the server, the cameras can be moved to other locations. To make the system more efficient, the Inertial Measurement Units (IMU) on the depth cameras [5] are used as motion monitors. The camera updates its poses only when the IMU detects movement. The updating method is based on an ORB tracking algorithm [58], which tracks features on consecutive frames and acquires the relative displacement of the camera.

Selective Segmentation. The recent success of deep learning-based schemes enables high-accuracy semantic segmentation on RGB images [73]. Its flexibility and adaptivity widely extend the applications of MetaStream in different practical scenarios. Thus, we apply it on RGB images to separate the target object from the background. While depth-based segmentation is lightweight, it is limited to handling environmental adaptation due to the lack of recognition capability [50]. For example, when a person sits on a chair, depth-based segmentation may not distinguish between the person and the chair. By exploiting the similarity between consecutive frames, we propose to *selectively* segment some frames and approximate the rest with the segmentation result on their preceding frame, as shown in Fig. 5. On the one hand, as segmentation reduces transmission delay and decreases the redundancy in point cloud synthesis on the server, it plays an important role in reducing end-to-end latency and improving the overall performance of MetaStream. On the other hand, as executing segmentation on smart cameras takes time [38, 39] (*e.g.,* 27 to 48 ms on Jetson Nano [14]), we should reduce the segmentation frequency. Thus, it is nontrivial to decide whether to segment a frame or not.

When a new frame f_i is sampled from the camera, we calculate its difference from the reference frame f_{j^*} and their difference is compared with the threshold θ_D in the Difference Indicator. The reference frame f_{j^*} is the last frame that is segmented by the segmentation model. The threshold θ_D is the upper bound of the frame difference. If the difference between f_i and f_{j^*} is less than θ_D, we directly apply the segmentation mask of f_{j^*}, S_{j^*}, to f_i (*i.e.,* the segmentation result of frame f_i is obtained by applying the mask S_{j^*} to it); otherwise, we use the segmentation model to generate the mask of f_i, S_i. In the later case, we update the reference frame f_{j^*} as f_i and update the segmentation mask S_{j^*} as S_i. Finally, the segmentation result is stored in the result buffer.

As the threshold θ_D determines whether to execute the segmentation model on frames, it is a critical parameter to balance segmentation accuracy and resource consumption. Specifically, if θ_D is higher than most frame differences, the segmentation model is seldom activated and the masks of most frames are approximated by the mask of their reference frame. Consequently, we can keep low resource consumption but may sacrifice segmentation accuracy due to mask approximation. On the contrary, if θ_D is lower than most frame differences, we can keep high segmentation accuracy but have high resource consumption as the segmentation model is executed frequently. During live volumetric video streaming, the computing capacity of smart cameras may vary as multiple tasks run simultaneously on the devices (*e.g.,* dynamic camera calibration and frame encoding). Moreover, the change of the target object's movement across frames also varies over time. Thus, we design a threshold updater to tune θ_D periodically (every T second) according to the computing capacity and the movement change rate.

In the threshold updater, we tune θ_D based on the records during the past T second. The performance recorder in Fig. 5 stores the frame differences in the past T second, denoted as $\mathcal{D}_{[t-T,t]}$. First, we calculate the average computing latency of the segmentation model during the past T second, denoted as $\bar{t}_S$, and rank the values in $\mathcal{D}_{[t-T,t]}$ from high to low, denoted as $\hat{\mathcal{D}}_{[t-T,t]}$. Note that $\bar{t}_S$ reflects the computing capacity of the device in the past T second. We then estimate the number of executions of the segmentation model that can be finished within T as $\lfloor T/\bar{t}_S \rfloor$. The actual number of executions of the segmentation model may not be the same as $\lfloor T/\bar{t}_S \rfloor$. For example, if θ_D is high and the frame difference is relatively low, the number of executions can be lower than $\lfloor T/\bar{t}_S \rfloor$, which indicates that more executions could have been done given a lower θ_D. Thus, we adjust θ_D based on $\lfloor T/\bar{t}_S \rfloor$ and $\hat{\mathcal{D}}_{[t-T,t]}$. Specifically, we find the $\lfloor T/\bar{t}_S \rfloor$-th value in $\hat{\mathcal{D}}_{[t-T,t]}$ and update θ_D to this value. The intuition is that, if θ_D were the $\lfloor T/\bar{t}_S \rfloor$-th value in $\hat{\mathcal{D}}_{[t-T,t]}$, there would be $\lfloor T/\bar{t}_S \rfloor$ frames processed by the segmentation model. In this way, we update θ_D every T second according to the device performance and frame differences in the past T second. Due to the change of reference frames with different θ_D, the number of executions of the segmentation models may not be equal to $\lfloor T/\bar{t}_S \rfloor$ when we set θ_D to the $\lfloor T/\bar{t}_S \rfloor$-th value in $\hat{\mathcal{D}}_{[t-T,t]}$. However, finding the optimal solution (the actual value that leads to $\lfloor T/\bar{t}_S \rfloor$ executions) is time-consuming. Consequently, we design the above heuristic method.

When MetaStream starts, we feed the first frame into the segmentation model and take it as the reference frame. We collect the frames sampled during segmenting the first frame and calculate their differences. Among these frames, we select the one with the highest frame difference and take it as the second reference frame, which is also fed into the segmentation model. The difference between the second and the first reference frames is the initial threshold θ_D, which is later updated periodically every T second as described above. As T is a pre-set hyperparameter, we will evaluate the system performance under different values of T in §6.2.

Alignment and Encoding. The above segmentation is only applied to RGB frames, and we align the results to depth frames with $(\delta x, \delta y)$, which is a displacement between the FoVs of depth and RGB cameras. We apply $(\delta x, \delta y)$ to the segmented pixels' positions on the RGB frame to obtain the corresponding positions on the depth frame and remove its background area. Finally, the segmented RGB and depth frames are encoded in H.264 format. Specifically, we apply a color filter to the gray-scale depth frames and convert them to three channels (corresponding to the R, G, and B channels) before compression. In such a way, we can directly encode the converted depth frames with H.264.

4.2 Content Creation Server

To create a complete point cloud of a target object, we need to combine the outputs from multiple cameras surrounding the object, which leads to the following challenges.

• Due to discrepancies among cameras and their connections with the server, the arrivals of frames from different cameras may be asynchronous, which leads to low-quality content creation of the object.

• Creating a complete point cloud of an object involves an extremely high volume of data, which consumes a large memory footprint and requires high-performance computers for real-time content creation.

We resolve the above challenges in the following ways.

• We design a lightweight yet effective capture synchronization module to find frames across cameras that are sampled at approximately the same time.

• We present a two-step content creation workflow: single-camera point cloud construction and multi-camera point cloud synthesis. In this way, we flexibly parallel the single-camera point cloud construction of different cameras whenever their frames arrive at the server. The constructed point clouds of different cameras are synthesized based on the cameras' positions with low computational cost.

Capture Synchronization. Synchronicity is important for constructing point clouds from multiple cameras. Most existing setups (*e.g.,* Microsoft Kinect DK [7] and Holoportation [59]) use a cable to transmit synchronization signals, which limits the mobility of cameras. Instead, we utilize a header message to synchronize frames from different cameras. The header message has only 32 bytes which stores frame capturing time, frame index, frame resolution, and camera ID.

To synchronize frames across cameras, we set an upper bound (U), which is the maximum waiting time once the first frame with a new ID (*i.e.,* an ID that has not been seen by the server before) is received by the server. After frames of a new ID start to arrive, the server collects all frames with this ID that are received within U. If the frames of this ID from all cameras are received before U, the server finishes the collection of frames and sends them to the next module. Note that when a frame with this ID arrives after U, it is dropped after decoding. Thus, it does not affect the decoding of subsequent frames because all received frames are decoded regardless of whether they have missed the deadline.

During the streaming process, we adaptively adjust U. For every Δf frame IDs, we tune U by adding a value ΔU. For each frame ID i, we first compute the difference between the arrival time of the first and last received frames ΔT_i and then compute the difference between U and ΔT_i. We then define ΔU as the median of these differences of the Δf frame IDs. That is, $\Delta U = median\{\Delta T_i - U\}_{i\in[f, f+\Delta f-1]}$. N is the number of cameras and f is the first frame ID in the past Δf frames. In general, when the network delays between some cameras and the server are higher than others, the frames of the same ID arrive at the server asynchronously and the last frame of the ID may arrive at the server after U (*i.e.,* a positive ΔU); when the network delays between cameras and the server are similar, the frames of the same index number arrive at the server synchronously and the last frame of the ID may arrive at the server within U (*i.e.,* a negative ΔU).

As the network conditions are temporally correlated [54], ΔU of the past Δf frames reflects the network conditions between cameras and the server during the past Δf frames, and we can utilize it to guide the collection of future frames. Thus, in the past Δf frames, if the frames from cameras arrive at the server in a wide range, then ΔU is a positive large value, and we tune U higher accordingly for the next Δf frames; if the frames from cameras arrive at the server in a narrow range (synchronously), then ΔU is a negative value, and we tune U lower accordingly for the next Δf frames.

Single-Camera Point Cloud Construction. Given the configuration of cameras, we obtain the horizontal FoV (h_{fov}), the vertical FoV (v_{fov}), and the width (d_{width}) and height (d_{height}) of the original depth frame. Given a point (x, y) on the depth frame with depth value z, we calculate its coordinate values as: $(dx \cdot \tan(h_{fov}/2) \cdot z, dy \cdot \tan(v_{fov}/2) \cdot z, z)$, where $dx = 2 \cdot (x - d_{width}/2)/d_{width}$ and $dy = 2 \cdot (y - d_{height}/2)/d_{height}$ [32]. As we have aligned the RGB and

depth frames on the camera, we can directly obtain the corresponding RGB values of each point on the depth frame. In this way, we can create point clouds from cameras by combining their RGB and depth frames.

Redundant Point Removal. With multiple cameras surrounding an object, there can be overlapping areas sampled by more than one camera. Removing redundant points in the overlapping area improves processing and transmission efficiency. In our coordinate system, the x-axis denotes leftward and rightward directions, the y-axis indicates upward and downward directions, and the z-axis signifies forward and backward directions. Based on the segmentation results and the point cloud construction, we obtain the left-most and right-most points of an object, (x_l, y_l, z_l) and (x_r, y_r, z_r). From their coordinate values, we can approximately determine the center point of the object as $(\frac{x_l+x_r}{2}, \frac{y_l+y_r}{2}, \frac{z_l+z_r}{2})$. We repeatedly calculate the approximated center point of the object on each camera. Then, we average over these points to have a more accurate estimation of the center point of the object, which is denoted as (x_c, y_c, z_c). Consequently, the center axis of the object is (x_c, z_c). Additionally, we can identify the range of the object on the y-axis, denoted as y_{top} and y_{bottom}, according to the segmentation results.

We find the overlapping area between two adjacent cameras as follows. First, we determine the right-most vertical boundary of the object in the FoV of the camera on the left and the left-most vertical boundary of the object in the FoV of the camera on the right (*i.e.,* $x = x_r$ and $x = x_l$); then, for the points that lie between the two boundaries on the object, they are captured by both cameras. We observe that the point clouds generated by one camera may exhibit a spatial offset relative to the other camera in the overlapping area caused by the inherent errors in the point cloud construction, which include measurement errors in the depth data and precision errors during calibration. Thus, we randomly sample some points in the overlapping area (*i.e.,* $x_r \le x \le x_l$) and calculate their distances to the center axis on the same level y, $y_{bottom} \le y \le y_{top}$. Once we find that the points of one camera have a smaller distance to the center axis, we delete all points of that camera in the overlapped area. We repeatedly loop over all overlapping areas between any two adjacent cameras to remove all redundant points.

Point Clouds Synthesis and Compression. We synthesize the point clouds of different cameras into a complete point cloud of an object by translating the point clouds of other cameras to the coordinate system of the reference camera (selected by the dynamic camera calibration module). For example, for a given point (x, y, z) in the point cloud from camera i, the new position will be $(x - x_i, y - y_i, z - z_i)$, where (x_i, y_i, z_i) is the coordinate of camera i in the coordinate system of the reference camera. Further, we filter out

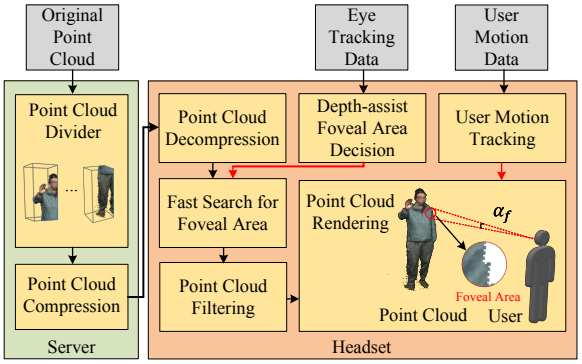


Figure 6: Volumetric Content Rendering Workflow.

and smooth the synthesized point cloud with a kNN-based algorithm [49], which reduces error points (caused by cameras' sensing noises) and further improves visual quality [18].

4.3 Mixed Reality Client

We tackle the challenge of limited computing resources on MR headsets and propose to leverage foveated rendering in MetaStream to preserve a good user experience with low overhead. We show the rendering workflow in Fig. 6.

Depth-assisted Determination of Foveal Size. Foveated rendering can reduce the computation overhead of headsets [61]. It synthesizes content with progressively less detail outside the eye fixation region by tracking the user's eye movement. Compared to traditional applications of foveated rendering in 2D images/videos [61], foveated rendering on volumetric content presents unique challenges. For example, as the user moves during rendering, the radius of the foveal area on volumetric content keeps changing due to the distance variation between the user and the displayed volumetric content.

We propose a depth-assisted foveated rendering method to resolve the above challenges. Specifically, we determine the foveal area of the user via the visual focal point and the central foveal angle α_f (*e.g.,* $\alpha_f = 7.5°$ [61]). We draw a virtual circular cone along with the user's viewing direction to the rendered content and take the center point of the intersection area with the point cloud as the visual focal point of the user, which is denoted as (x_f, y_f, z_f). Given the position of the center of the user's eyes (x_e, y_e, z_e), we obtain the distance between it to the visual focal point on the rendering content, $d_{fe} = \sqrt{(x_f - x_e)^2 + (y_f - y_e)^2 + (z_f - z_e)^2}$. Further, the foveal area is a circle with the central point at (x_f, y_f, z_f) and its radius is $r_f = d_{fe} \cdot \tan(\frac{\alpha_f}{2})$.

Fast Search for Foveal Area on Volumetric Content. After getting the size of the foveal area, MetaStream needs to calculate which points of the volumetric content are in the foveal area. However, it may generate nontrivial overhead on the headset to search for points that locate inside the user's foveal area among the whole point cloud. As the user's

ρ	0.5	0.6	0.7	0.8	0.9
Average Score	1.24	2.61	3.37	3.66	3.84

Table 2: Visual Quality for Different ρ.

foveal area may change fast [61], such overhead lowers user experience. Thus, we propose to reduce the overhead by partitioning the point cloud into several small subsets on the server (before transmitting to the MR client).

Specifically, we denote the boundary of the point cloud ($\mathcal{S}$) on the x-y plane (the plane that is perpendicular to the ground) as $\{X_{min}, X_{max}, Y_{min}, Y_{max}\}$. We divide the point cloud into $K \times K$ subsets and each subset $\mathcal{S}_{ij}$ $(1 \le i, j \le K)$ contains the points inside the boundary of $\{(i-1)\cdot\Delta X, i\cdot\Delta X, (j-1)\cdot\Delta Y, j\cdot\Delta Y\}$, where $\Delta X = \frac{X_{max}-X_{min}}{K-1}$ and $\Delta Y = \frac{Y_{max}-Y_{min}}{K-1}$. In this way, given each subset's boundary and the foveal area, we can determine the subset that contains the foveal area and search inside only that subset on the client. Note that we can extend the boundary of each subset (making neighboring subsets overlap with each other) to simplify the case in which the foveal area includes points from multiple neighboring subsets. When sending them to the headset, we compress these subsets individually and mark each subset with its boundary.

Point Cloud Filtering and Rendering. For points that are inside the foveal area, we render them at the original density. For other points, we apply a filter to reduce the rendering density by a ratio ρ. That is, each point is rendered with a probability of ρ. We conduct an IRB-approved user study to find the minimal acceptable ρ. We develop an application to render and display point clouds with different ρ (*i.e.,* 0.5, 0.6, 0.7, 0.8, and 0.9) on HoloLens 2. We invite 16 participants, aged from 23 to 54, and 12 of them have 20/20 corrected vision. We ask each participant to wear HoloLens 2 and watch the point clouds with different ρ (we keep them agnostic to ρ and randomly order the five point clouds for a fair study). We ask the participants to provide their mean opinion scores (MOS) for the visual quality of the point cloud from 1 to 5 (1: bad, 2: poor, 3: fair, 4: good, 5: excellent). The results are shown in Table 2. The selection of ρ is to balance the trade-off between visual quality and rendering overhead. Specifically, with a large ρ, there will be a large number of points to render, which can preserve high visual quality but lead to a long rendering delay, and vice versa. We observe a significant increase in MOS (0.76) between $\rho = 0.6$ and $\rho = 0.7$ and a small increase between $\rho = 0.7$ and $\rho = 0.8$. Consequently, we set $\rho = 0.7$.

Besides foveated rendering, we apply motion-based rendering. Specifically, we design a motion tracking module to track the user's orientation in real-time, as shown in Fig. 6. We adopt the particle system of Unity engine [16] for rendering on HoloLens 2. In the rendering process, we convert the (r, g, b) values of each point into 2D textures and set each

Multi-camera Setup Single Camera

Figure 7: Testbed of MetaStream.

basic particle (point) as a 2D square to reduce rendering overhead. Based on the tracked user's motion, we adjust the normal of the 2D square to align with the user's viewing direction (*i.e.,* making the 2D square perpendicular to the user's view). In this way, the rendering of the point cloud offers good visual quality. The rendering overhead is ~25.4ms for a point cloud with 330K points based on our evaluation.

5 Implementation

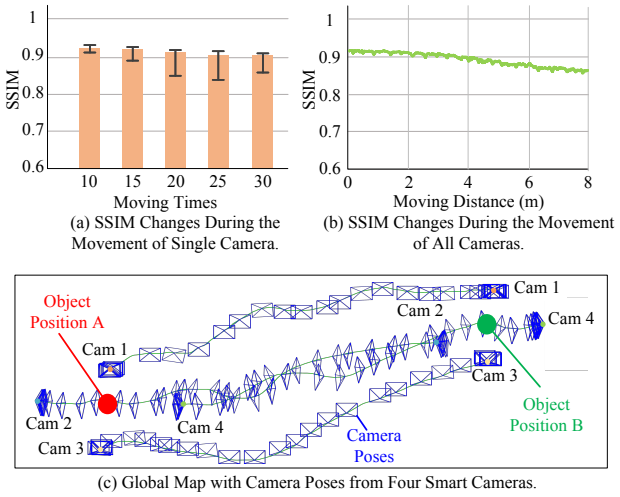
Hardware: The setup of our testbed is shown in Fig. 7. Each smart camera is an Intel RealSense L515 [4] mounted on Nvidia Jetson Nano [14]. The content creation server is equipped with AMD 5950X CPU and Nvidia RTX 2080S GPU. The MR client is Microsoft HoloLens 2 [8].

Software: The smart camera and content creation server are developed on Linux, and the MR client is developed on Universal Windows Platform (UWP) [9]. On smart cameras, we implement the dynamic camera calibration module based on ORB feature extraction and tracking [58], OpenCV [23], and RealSense SDK [5]; we implement the segmentation and streaming with RealSense SDK [5], OpenCV [23], Nvidia Docker [12], TensorFlow [20], and x264 library [19]; the segmentation model is Segnet [21], compressed by Jetson-inference [13]. The segmented results are encoded with H.264 [74] and transmitted to the content server. On the content creation server, we utilize x264 [19], Open3D [84], Point Cloud Library (PCL) [68], and Draco [1] to implement content creation and compression. On the MR client, we utilize Unity [16] and Mixed Reality Toolkit (MRTK) [11] to implement foveated content rendering. After decompressing the received data, we filter and load the point clouds into the VFX Graph [17] plugin of Unity based on the foveal data collected by eye tracking sensor [2] of HoloLens 2.

In total, our implementation consists of 4,500+ lines of code (LoC): 1,300+ LoC in C++ for the smart camera, 1,500+ LoC in C++ for the content creation server, and 1,700+ LoC in C# for the MR client.

6 Performance Evaluation

In this section, we evaluate the performance of MetaStream with live experiments and compare it with the state-of-the-art, LiveScan3D[46]. We use the FPS, structural similarity



(a) SSIM Changes During the Movement of Single Camera.
(b) SSIM Changes During the Movement of All Cameras.
(c) Global Map with Camera Poses from Four Smart Cameras.

Figure 8: Evaluation of Dynamic Camera Calibration.

index measure (SSIM) [18], network throughput, and end-to-end latency as the metrics for our evaluation. SSIM is a weighted combination of luminance, contrast, and structure comparison measurements between two images to measure their similarity. Its values range between 0 to 1, and 1 means the target image is the same as the reference image. We calculate the SSIM[2] between the screenshots on HoloLens 2 [8] and the original images from smart cameras. For the end-to-end latency, we measure the difference between the time when a frame is captured by cameras and the time when HoloLens 2 [8] renders the point cloud constructed by the frame. We average the end-to-end latency of all frames during a streaming session.

6.1 Dynamic Camera Calibration

We first evaluate the performance of the dynamic camera calibration method with a setup of four smart cameras. We move one of the four cameras by several times (10, 15, 20, 25, and 30) and return it to its original position; we then compare the SSIM of the same captured object before and after the camera is moved. As shown in Fig. 8 (a), the SSIM drops by only 0.02 with the increase in moving times of the camera, which indicates the robustness of our method to the movement of a single camera. In Fig. 8 (b), we show the change of SSIM along with the moving distance of the target object (a person) from position A to position B in Fig. 8 (c) following an unplanned route. The four cameras follow the object, and their routes are shown in Fig. 8 (c). With the movement of cameras, the SSIM of the captured object decreases by 0.081 when the moving distance is 8m. The decrease is caused by the accumulated tracking error, which is unavoidable in unclosed-loop routes [58]. We will study how to further improve the calibration performance when the cameras move in future work.

[2]The SSIM is measured on the whole point cloud without point filtering (§4.3) rather than within the foveal area unless specifically mentioned.

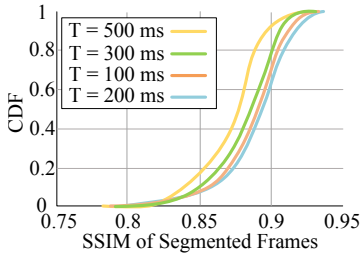


Figure 9: Evaluation of Selective Segmentation The CDF of SSIM with Different T.

6.2 Selective Segmentation

We then evaluate the performance of the selective segmentation method on a one-minute volumetric video. The video covers the static scenario and the slow and fast movements of the target person. We evaluate the performance under different threshold-update frequencies, T=100, 200, 300, and 500ms. In Fig. 9, we show the cumulative distribution function (CDF) of SSIM with different T. As discussed in §4.1, MetaStream updates the frame-difference threshold based on the records in the past T second. Consequently, with a large T, it may fail to adaptively adjust the threshold based on the change in the target object's movement. As shown in Fig. 9, the SSIM of 92.2% frames is under 0.9 when T = 500ms.

Specifically, there are two reasons for the low SSIM with large T. First, when the target object switches from fast movement to slow movement, MetaStream with large T still keeps a high threshold, and few frames are fed into the segmentation model, which reduces segmentation accuracy. Second, when the target object switches from static/slow movement to fast movement, MetaStream with large T still keeps a low threshold, and almost all frames are fed into the segmentation model, which leads to severe segmentation delay. For live volumetric video streaming, we have the maximum waiting time on the content creation server as described in §4.2. Thus, the segmentation delay further causes frame drops on the server, and the dropped frames can cause degradation in the quality of the synthesized point cloud. Nevertheless, we observe similar patterns for T = 100ms and T = 200ms. It indicates that the value of T does not have to be extremely small, which means we do not necessarily update the threshold with a high frequency.

6.3 Comparison with LiveScan3D

In the original design of LiveScan3D [46], each camera is connected to a desktop computer, which is connected to a server via a cable. In our evaluation, we implement LiveScan3D on the same hardware as MetaStream for a fair comparison.

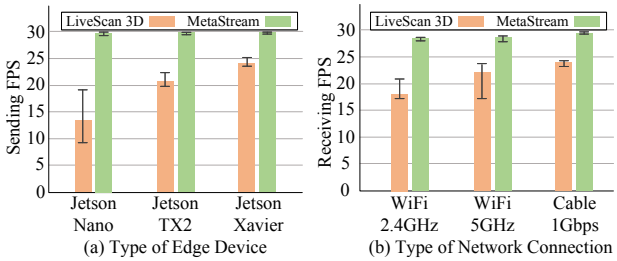


Figure 10: Comparison of sending (a) and receiving (b) FPS of LiveScan3D and MetaStream with different hardware and network configurations.

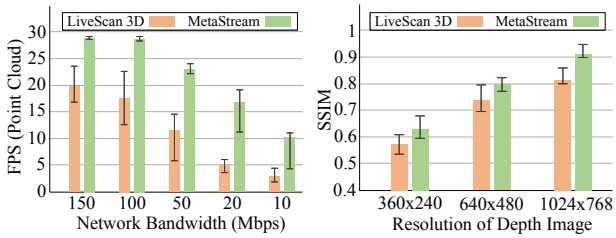


Figure 11: Comparison of FPS and SSIM between LiveScan3D and MetaStream.

6.3.1 Capturing Performance with Different Hardware Configurations. We compare the sending FPS on the camera side with different types of edge devices: Jetson Nano [14], TX2 [15], and Xavier [6], and the receiving FPS on the server under different types of networks: 2.4GHz WiFi, 5GHz WiFi, and 1Gbps cable. As shown in Fig. 10 (a), MetaStream outperforms LiveScan3D for the sending FPS by 20.8% to 1.13×. As MetaStream offloads point cloud synthesis to the server, its requirement on the computing capability of edge devices on the camera side is largely lowered. Thus, we hardly observe any difference in sending FPS among the three types of edge devices. In other words, MetaStream is relatively robust to edge devices' computing capability on the camera side compared to LiveScan3D. Similarly, MetaStream outperforms LiveScan3D by 16.1% to 64.7% when we vary the type of network between smart cameras and the server, as shown in Fig. 10 (b). Due to the large transmission data size in LiveScan3D, the receiving FPS significantly decreases when the network bandwidth is low (*e.g.,* 2.4GHz WiFi).

6.3.2 FPS under Different Network Bandwidth. In Fig. 11 (a), we compare the end-to-end FPS of MetaStream with LiveScan3D under different network bandwidths (150, 100, 50, 20, and 10Mbps). Note that the end-to-end FPS refers to the frame rate observed on the MR client. We vary the network bandwidth on both the network connection from the smart cameras to the server and that from the server to the MR client. Overall, MetaStream outperforms LiveScan3D by 23.2% to 3.32×. Several factors in the design of MetaStream contribute to such FPS improvement: (1) the selective segmentation and H.264 encoding on smart cameras, (2) the distribution of computation across smart cameras and the server, and (3) the redundant point removal on the server.

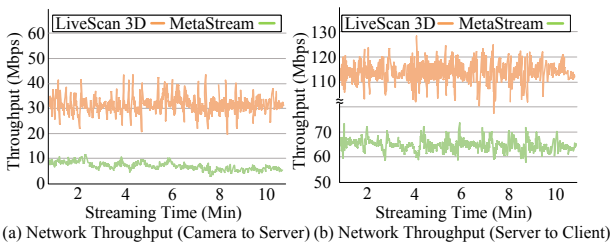


Figure 12: Comparison of Network Throughput between LiveScan3D and MetaStream.

6.3.3 SSIM vs. Resolution of Depth Image In Fig. 11 (b), we compare the SSIM of MetaStream and LiveScan3D with different frame resolutions of depth images. Benefiting from the visual quality improvement by removing redundant points and motion-based rendering (§4.3), MetaStream outperforms LiveScan3D by 6.84% to 12.5%.

6.3.4 Network Throughput. We measure the throughput of LiveScan3D and MetaStream by continuously streaming live videos for 10 min. As shown in Fig. 12, the network throughput of LiveScan3D is much higher than MetaStream, for both the link from a randomly selected smart camera to the server and that from the server to the MR client. As MetaStream sends 2D videos encoded by H.264, the throughput from the smart camera to the server is significantly low, around only 12Mbps with an average FPS of 28.7, as shown in Fig. 12 (a). In contrast, LiveScan3D generates around 38.1Mbps throughput with an average FPS of 23.7. Though MetaStream transmits point clouds from the server to the MR client as LiveScan3D does, the average throughput of MetaStream is < 58.1% of that of LiveScan3D. There are two reasons for this large throughput improvement. First, MetaStream reduces transmitted data by the DNN-based segmentation model, which effectively segments the target person from the background. In contrast, due to the limitation of depth-based segmentation in LiveScan3D, the segmented frames still keep some background pixels with the target person. Second, the redundant points removal in MetaStream reduces the number of points in delivered volumetric content without affecting visual quality.

6.3.5 End-to-end Latency. We compare the end-to-end latency of live volumetric video streaming with LiveScan3D and MetaStream, by breaking it down to each component. Specifically, we divide the streaming procedure into four parts. The first component includes the process from the camera sampling to H.264 encoding on smart cameras. The second one includes the process of receiving encoded RGB and depth data to point cloud compression on the server. The third one includes the process from receiving the compressed point cloud to displaying it to the user on the MR client. The

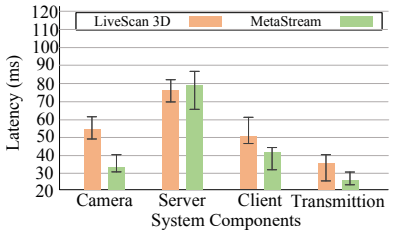


Figure 13: Breakdown of End-to-end Live Streaming Latency into Four Components.

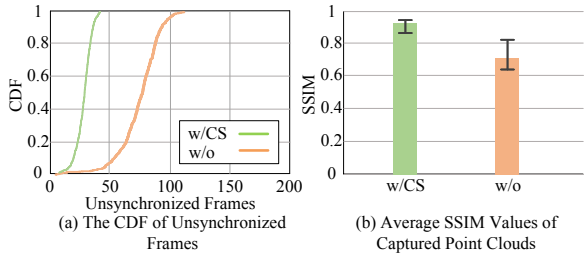


Figure 14: Evaluation of Capture Synchronization (CS).

fourth one includes the two transmission processes (*i.e.,* from smart cameras to the server and from the server to the MR client). Note that MetaStream utilizes Jetson Nano as the edge device for smart cameras. We employ Jetson Xavier [6] as the edge device for LiveScan3D, which has a much higher computing capability than Jetson Nano.

As shown in Fig. 13, the latency of each component in LiveScan3D is higher than that in MetaStream, except for the computing latency on the server. Overall, the end-to-end of MetaStream is 31.7% lower than LiveScan3D. The computing latency on the cameras in MetaStream is only 53.5% of that in LiveScan3D. Though point cloud synthesis is offloaded to the server in MetaStream, the computing latency on the server in MetaStream is only slightly higher (around 3%) than that in LiveScan3D because of the parallel pipeline design and redundant point removal in MetaStream. In addition, we observe the video start-up time is 182 to 193ms in MetaStream and 239 to 247ms in LiveScan3D.

6.4 Ablation Study

We evaluate the effect of capture synchronization for 2,000 frames and show the results in Fig. 14. Without synchronization, the server reads the streaming buffer without checking frame IDs. As a result, frames from different cameras may be synthesized to one point cloud even if they are not sampled at the same time. As shown in Fig. 14 (a), the number of unsynchronized frames without capture synchronization is over twice that with capture synchronization due to network transmission delay. Moreover, with capture synchronization, the SSIM is 28.5% higher than that without capture synchronization as shown in Fig. 14 (b), which indicates a higher-quality content creation.

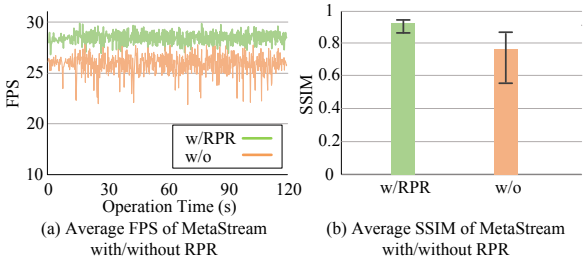


Figure 15: Evaluation of Redundant Point Removal (RPR).

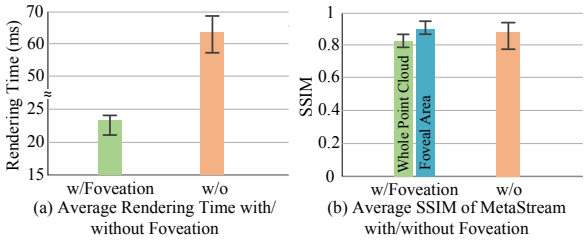


Figure 16: Evaluation of Foveated Content Rendering.

We evaluate the effect of redundant point removal in a challenging environment, a WiFi network with 50Mbps bandwidth. We show the FPS on the MR client in Fig. 15 (a). As the transmission data size from the server to the headset is reduced by the redundant point removal, we observe an FPS increase of 2 to 8. Moreover, the redundant point removal improves the visual quality of volumetric content. As shown in Fig. 15 (b), the volumetric content with redundant point removal reaches an SSIM that is on average 16.4% higher than that of without redundant point removal.

As shown in Fig. 16 (a), foveated rendering reduces the rendering time on average by 63.1% on the MR device. While the SSIM of the whole point cloud with foveated rendering is 7.2% lower than the regular rendering, the SSIM inside the foveal area is higher, as shown in Fig. 16 (b). It benefits from both foveated rendering and motion-based rendering optimization. Note that we set the default rendering density ratio $\rho = 0.7$ based on Table 2, which achieves an SSIM of 0.87 as shown in Fig. 16 (b).

7 Discussion

Limitations. As the first practical live volumetric video streaming system, MetaStream has a few limitations of its current design. For example, we have not yet optimized its performance for multi-user scenarios (*i.e.,* there is only one user receiving video content in MetaStream). Recently, Zhang *et al.* [83] propose a research agenda for enabling multi-user volumetric video streaming over mmWave networks through a cross-layer design for VoD applications. Extending MetaStream to multi-user scenarios is challenging due to the stringent latency requirement of live video streaming and the diverse network conditions of different

users. Another demanding future work is to further improve the accuracy of dynamic camera calibration.

Camera-server Split. Overall, MetaStream benefits from the camera-server split setup from the following aspects. First, directly transmitting RGB+depth frames to the server consumes more network bandwidth and leads to longer delay, compared to streaming only segmented parts. Second, we target a scalable multi-camera system where the number of cameras may be large to further improve the visual quality of generated volumetric content. If we deploy the segmentation modules on the edge server, the latency of inference increases with the number of cameras (*i.e.,* batch size). In comparison, the latency of executing the segmentation module on each smart camera is relatively stable.

Enabling Interactive Applications. We plan to extend MetaStream to support interactive applications with live volumetric content delivery [26, 59]. Instead of having viewers passively receive video content from the broadcaster, we can set up MetaStream at multiple locations so that geo-distributed users can interact with each other via the captured volumetric video of their activities, fully exploiting the unique feature of volumetric content. Existing work such as Holoportation [59] may not be suitable for mobile scenarios due to its high bandwidth requirement (*e.g.,* >1Gbps).

Neural Adaptive Content Delivery. We can further improve the performance of MetaStream by reducing its mobile data usage via neural adaptive streaming [78, 79]. Recently, Zhang *et al.* [82] propose to lower bandwidth consumption of volumetric video streaming by delivering low-density point clouds that will be upsampled to high quality through 3D super resolution (SR) [51]. While it has been demonstrated to be feasible to conduct 3D SR on machines equipped with powerful GPUs, applying the idea to mobile devices remains a challenging research problem [77], especially for live volumetric content delivery.

8 Related Work

Volumetric Video Streaming. The research on volumetric video streaming is still relatively nascent, and thus there exist only a few studies on this topic [33, 34, 36, 48, 60, 64, 72, 82, 83]. ViVo [36] proposes several visibility-aware optimizations such as occlusion and distance visibility to boost the performance of volumetric video streaming. GROOT [48] introduces parallel decoding of highly-quality point-cloud data for delivering volumetric content. The above schemes directly stream volumetric content. Other approaches benefit from the remote rendering of volumetric content (*i.e.,* transcoding into 2D content) [33, 34, 64]. Given the above work, content delivery is not the focus of this paper. We can integrate their main ideas into MetaStream, as the design of MetaStream is extensible and orthogonal to them.

Live Video Streaming. There is a plethora of research on live streaming for not only regular videos [22, 45] but also 360° videos [24, 53, 80]. Skynet [81] utilizes existing P2P technologies and integrates them with minimal changes to the existing CDN infrastructure to ensure that the system scales with the number of users. CNLive [52] and Akamai live streaming [69] focus on user activities and network traffic. Twitch.tv is a live streaming service exclusively for gaming broadcast [35]. Cicco *et al.* [27] present a quality adaptation controller for an adaptive live video streaming system designed by using feedback control theory. Different from the above work, we investigate live volumetric content capture, creation, delivery, and rendering and propose MetaStream, a practical system to improve content delivery performance for emerging immersive applications.

3D Object Model Construction. Image-based 3D object model construction has been extensively studied in computer vision and graphics communities [28–30, 42, 56, 76]. For example, KinectFusion [42] is the first system that fuses point clouds into meshes using a single depth sensor. Fusion4D [28] is a real-time multi-view nonrigid reconstruction system for high-quality live performance capture. Montage4D [29] is an interactive and real-time solution to blend multiple video textures onto dynamic meshes with nearly indiscernible view transitions. Different from the mentioned work, MetaStream not only focuses on creating 3D content in real-time but also proposes a practical system that can transmit and render 3D content with low latency.

9 Conclusion

In this paper, we propose a live volumetric content capture, creation, delivery, and rendering system, MetaStream. We design MetaStream on low-cost commercial platforms and achieve close to 30 FPS on WiFi networks. We specifically address the challenges of calibration of multiple cameras, volumetric video capture on resource-constrained smart cameras, point cloud synthesis for multi-camera setup, and reduction of transmission delay over WiFi networks. With dynamic camera calibration, selective segmentation, efficient point cloud synthesis, and foveated rendering of point clouds on MR devices, MetaStream reduces end-to-end latency by up to 31.7% while improving visual quality by up to 12.5% compared to state-of-the-art systems.

Acknowledgments

We thank the anonymous shepherd and reviewers for their insightful comments and the voluntary users who participated in our user study. This work was partially supported by the U.S. NSF under Grants 2147821, 2147623, 2147624, and 2212296 and a Google Research Scholar Award.

References

[1] Draco 3D Graphics Compression. https://google.github.io/draco/.

[2] Eye Tracking on HoloLens 2. https://docs.microsoft.com/en-us/windows/mixed-reality/design/eye-tracking.

[3] HOLOSYS. https://www.4dviews.com/volumetric-systems.

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

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

[6] Jetson AGX Xavier. https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-agx-xavier/.

[7] Microsoft Azure Kinect DK. https://developer.microsoft.com/en-us/windows/kinect.

[8] Microsoft HoloLens 2. https://www.microsoft.com/en-us/hololens.

[9] Microsoft Universal Windows Platform. https://docs.microsoft.com/en-us/windows/uwp/get-started/universal-application-platform-guide.

[10] Mixed Reality Capture Studios. https://www.microsoft.com/en-us/mixed-reality/capture-studios.

[11] Mixed Reality Toolkit 2. https://docs.microsoft.com/en-us/windows/mixed-reality/mrtk-unity/mrtk2/?view=mrtkunity-2022-05.

[12] Nvidia Docker. In *https://docs.nvidia.com/datacenter/cloud-native/container-toolkit/user-guide.html*.

[13] Nvidia Jetson-inference. https://www.microsoft.com/en-us/mixed-reality/capture-studios.

[14] Nvidia Jetson Nano Developer Kit. https://developer.nvidia.com/embedded/jetson-nano-developer-kit.

[15] Nvidia Jetson TX2 Module. https://developer.nvidia.com/embedded/jetson-tx2.

[16] Unity Real-Time Development Platform. In *https://unity.com/*.

[17] Visual Effect Graph. https://unity.com/visual-effect-graph.

[18] What are SSIM and PSNR. https://github.com/deterenkelt/Nadeshiko/wiki/Docs-on-encoding.-SSIM-and-PSNR.

[19] X264 library. https://www.videolan.org/developers/x264.html.

[20] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv*, 2016.

[21] V. Badrinarayanan, A. Kendall, and R. Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.

[22] G. Baig, J. He, M. A. Qureshi, L. Qiu, G. Chen, P. Chen, and Y. Hu. Jigsaw: Robust Live 4K Video Streaming. In *Proceedings of ACM MobiCom*, 2019.

[23] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal: Software Tools for the Professional Programmer*, 25(11):120–123, 2000.

[24] B. Chen, Z. Yan, H. Jin, and K. Nahrstedt. Event-driven Stitching for Tile-based Live 360 Video Streaming. In *Proceedings of ACM MMSys*, 2019.

[25] P. J. Choi, R. J. Oskouian, and R. S. Tubbs. Telesurgery: past, present, and future. *Cureus*, 10(5), 2018.

[26] A. Clemm, M. T. Vega, H. K. Ravuri, T. Wauters, and F. De Turck. Toward truly immersive holographic-type communication: Challenges and solutions. *IEEE Communications Magazine*, 58(1):93–99, 2020.

[27] L. De Cicco, S. Mascolo, and V. Palmisano. Feedback control for adaptive live video streaming. In *Proceedings of ACM MMSys*, 2011.

[28] M. Dou, S. Khamis, Y. Degtyarev, P. Davidson, S. R. Fanello, A. Kowdle, S. O. Escolano, C. Rhemann, D. Kim, J. Taylor, et al. Fusion4D: Real-time Performance Capture of Challenging Scenes. *ACM Transactions on Graphics (ToG)*, 35(4):1–13, 2016.

[29] R. Du, M. Chuang, W. Chang, H. Hoppe, and A. Varshney. Montage4D: Interactive Seamless Fusion of Multiview Video Textures. In *Proceedings of ACM Interactive 3D Graphics (I3D)*, 2018.

[30] A. Geiger, J. Ziegler, and C. Stiller. StereoScan: Dense 3D Reconstruction in Real-time. In *2011 IEEE Intelligent Vehicles Symposium (IV)*, 2011.

[31] T. Golla and R. Klein. Real-time point cloud compression. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2015.

[32] Y. Guan, X. Hou, N. Wu, B. Han, and T. Han. DeepMix: mobility-aware, lightweight, and hybrid 3D object detection for headsets. In *Proceedings of ACM MobiSys*, 2022.

[33] S. Gül, D. Podborski, T. Buchholz, T. Schierl, and C. Hellge. Low-latency cloud-based volumetric video streaming using head motion prediction. In *Proceedings of ACM Workshop on Network and Operating Systems Support for Digital Audio and Video*, 2020.

[34] S. Gül, D. Podborski, J. Son, G. S. Bhullar, T. Buchholz, T. Schierl, and C. Hellge. Cloud Rendering-based Volumetric Video Streaming System for Mixed Reality Services. In *Proceedings of ACM MMSys*, 2020.

[35] W. A. Hamilton, O. Garretson, and A. Kerne. Streaming on Twitch: fostering participatory communities of play within live mixed media. In *Proceedings of ACM CHI*, 2014.

[36] B. Han, Y. Liu, and F. Qian. ViVo: Visibility-Aware Mobile Volumetric Video Streaming. In *Proceedings of ACM MobiCom*, 2020.

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

[38] X. Hou, Y. Guan, and T. Han. NeuLens: spatial-based dynamic acceleration of convolutional neural networks on edge. In *Proceedings of ACM MobiCom*, 2022.

[39] X. Hou, Y. Guan, T. Han, and N. Zhang. Distredge: Speeding up convolutional neural network inference on distributed edge devices. In *Proceedings of IPDPS*, 2022.

[40] J. Hu, A. Shaikh, A. Bahremand, and R. LiKamWa. Characterizing real-time dense point cloud capture and streaming on mobile devices. In *Proceedings of ACM Workshop on Hot Topics in Video Analytics and Intelligent Edges*, 2021.

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

[42] S. Izadi, D. Kim, O. Hilliges, D. Molyneaux, R. Newcombe, P. Kohli, J. Shotton, S. Hodges, D. Freeman, A. Davison, et al. KinectFusion: Real-time 3D Reconstruction and Interaction Using a Moving Depth Camera. In *Proceedings of UIST*, 2011.

[43] H. Jun and J. Bailenson. Temporal RVL: a depth stream compression method. In *Proceedings of IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW)*, 2020.

[44] E. Karami, S. Prasad, and M. Shehata. Image matching using SIFT, SURF, BRIEF and ORB: performance comparison for distorted images. *arXiv*, 2017.

[45] J. Kim, Y. Jung, H. Yeo, J. Ye, and D. Han. Neural-Enhanced Live Streaming: Improving Live Video Ingest via Online Learning. In *Proceedings of ACM SIGCOMM*, 2020.

[46] 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 Conference on 3D Vision*, 2015.

[47] J. Lawrence, D. B. Goldman, S. Achar, G. M. Blascovich, J. G. Desloge, T. Fortes, E. M. Gomez, S. Häberling, H. Hoppe, A. Huibers, C. Knaus, B. Kuschak, R. Martin-Brualla, H. Nover, A. I. Russell, S. M. Seitz, and K. Tong. Project Starline: A high-fidelity telepresence system. *ACM Transactions on Graphics*, 40, 2021.

[48] K. Lee, J. Yi, Y. Lee, S. Choi, and Y. M. Kim. GROOT: A Real-Time Streaming System of High-Fidelity Volumetric Videos. In *Proceedings of ACM MobiCom*, 2020.

[49] J. Li, B. M. Chen, and G. H. Lee. So-net: Self-organizing network for point cloud analysis. In *Proceedings of IEEE CVPR*, 2018.

[50] L. Li. Time-of-flight camera–an introduction. *Technical white paper*, 2014.

[51] R. Li, X. Li, C.-W. Fu, D. Cohen-Or, and P.-A. Heng. PU-GAN: A Point Cloud Upsampling Adversarial Network. In *Proceedings of ICCV*, 2019.

[52] Y. Li, Y. Zhang, and R. Yuan. Measurement and Analysis of a Large Scale Commercial Mobile Internet TV System. In *Proceedings of ACM SIGCOMM*, 2011.

[53] X. Liu, B. Han, F. Qian, and M. Varvello. LIME: Understanding Commercial 360° Live Video Streaming Services. In *Proceedings of ACM MMSys*, 2019.

[54] N. M. Luscombe, M. Madan Babu, H. Yu, M. Snyder, S. A. Teichmann, and M. Gerstein. Genomic analysis of regulatory network dynamics reveals large topological changes. *Nature*, 431(7006):308–312, 2004.

[55] A. Maglo, G. Lavoué, F. Dupont, and C. Hudelot. 3D mesh compression: Survey, comparisons, and emerging trends. *ACM Computing Surveys (CSUR)*, 47(3):1–41, 2015.

[56] E. Mouragnon, M. Lhuillier, M. Dhome, F. Dekeyser, and P. Sayd. Real Time Localization and 3D Reconstruction. In *Proceedings of IEEE CVPR*, 2006.

[57] M. Munaro, F. Basso, and E. Menegatti. OpenPTrack: Open source multi-camera calibration and people tracking for RGB-D camera networks. *Robotics and Autonomous Systems*, 75:525–538, 2016.

[58] R. Mur-Artal and J. D. Tardós. ORB-SLAM2: An open-source slam system for monocular, stereo, and RGB-D cameras. *IEEE Transactions on Robotics*, 33(5):1255–1262, 2017.

[59] S. Orts-Escolano, C. Rhemann, S. Fanello, W. Chang, A. Kowdle, Y. Degtyarev, D. Kim, P. L. Davidson, S. Khamis, M. Dou, et al. Holoportation: Virtual 3D Teleportation in Real-time. In *Proceedings of UIST*, 2016.

[60] J. Park, P. A. Chou, and J.-N. Hwang. Rate-Utility Optimized Streaming of Volumetric Media for Augmented Reality. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 9(1):149–162, 2019.

[61] A. Patney, M. Salvi, J. Kim, A. Kaplanyan, C. Wyman, N. Benty, D. Luebke, and A. Lefohn. Towards foveated rendering for gaze-tracked virtual reality. *ACM Transactions on Graphics*, 2016.

[62] 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.

[63] F. Porikli and A. Divakaran. Multi-camera Calibration, Object Tracking and Query Generation. In *Proceedings of International Conference on Multimedia and Expo*, 2003.

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

[65] 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.

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

[67] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. ORB: An efficient alternative to SIFT or SURF. In *Proceedings of ICCV*, 2011.

[68] R. B. Rusu and S. Cousins. 3d is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation*, 2011.

[69] K. Sripanidkulchai, B. Maggs, and H. Zhang. An analysis of live streaming workloads on the internet. In *Proceedings of ACM SIGCOMM*, 2004.

[70] E. C. Strinati, S. Barbarossa, J. L. Gonzalez-Jimenez, D. Ktenas, N. Cassiau, L. Maret, and C. Dehos. 6G: The next frontier: From holographic messaging to artificial intelligence using subterahertz and visible light communication. *IEEE Vehicular Technology Magazine*, 14(3):42–50, 2019.

[71] F. Tariq, M. R. Khandaker, K.-K. Wong, M. A. Imran, M. Bennis, and M. Debbah. A speculative study on 6G. *IEEE Wireless Communications*, 27(4):118–125, 2020.

[72] J. van der Hooft, T. Wauters, F. D. Turck, C. Timmerer, and H. Hellwagner. Towards 6DoF HTTP Adaptive Streaming Through Point Cloud Compression. In *Proceedings of ACM MM*, 2019.

[73] P. Wang, P. Chen, Y. Yuan, D. Liu, Z. Huang, X. Hou, and G. Cottrell. Understanding convolution for semantic segmentation. In *Proceedings of IEEE Winter Conference on Applications of Computer Vision (WACV)*, 2018.

[74] T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A. Luthra. Overview of the H. 264/AVC video coding standard. *IEEE Transactions on circuits and systems for video technology*, 13(7):560–576, 2003.

[75] A. D. Wilson. Fast Lossless Depth Image Compression. In *Proceedings of ACM International Conference on Interactive Surfaces and Spaces*, 2017.

[76] M. Ye and R. Yang. Real-time simultaneous pose and shape estimation for articulated objects using a single depth camera. In *Proceedings of IEEE CVPR*, 2014.

[77] H. Yeo, C. J. Chong, Y. Jung, J. Ye, and D. Han. NEMO: Enabling Neural-enhanced Video Streaming on Commodity Mobile Devices. In *Proceedings of ACM MobiCom*, 2020.

[78] H. Yeo, S. Do, and D. Han. How will Deep Learning Change Internet Video Delivery? In *Proceedings of ACM HotNets*, 2017.

[79] H. Yeo, Y. Jung, J. Kim, J. Shin, and D. Han. Neural Adaptive Content-aware Internet Video Delivery. In *Proceedings of USENIX OSDI*, 2018.

[80] J. Yi, M. R. Islam, S. Aggarwal, D. Koutsonikolas, Y. C. Hu, and Z. Yan. An Analysis of Delay in Live 360° Video Streaming Systems. In *Proceedings of ACM MM*, 2020.

[81] H. Yin, X. Liu, T. Zhan, V. Sekar, F. Qiu, C. Lin, H. Zhang, and B. Li. Design and deployment of a hybrid CDN-P2P system for live video streaming: experiences with LiveSky. In *Proceedings of ACM MM*, 2009.

[82] A. Zhang, C. Wang, B. Han, and F. Qian. Efficient Volumetric Video Streaming Through Super Resolution. In *Proceedings of ACM HotMobile*, 2021.

[83] D. Zhang, B. Han, P. Pathak, , and H. Wang. Innovating Multi-user Volumetric Video Streaming through Cross-layer Design. In *Proceedings of ACM HotNets*, 2021.

[84] Q.-Y. Zhou, J. Park, and V. Koltun. Open3D: A modern library for 3D data processing. *arXiv*, 2018.