Demonstrating Adaptive Many-to-Many Immersive Teleconferencing for Volumetric Video

Matthias De Fré, Jeroen van der Hooft, Tim Wauters, Filip De Turck
matthias.defre@ugent.be
IDLab, Ghent University - imec
Ghent, Belgium

ABSTRACT

In today's world, the use of video conferencing applications has risen significantly. However, with the introduction of affordable head-mounted displays (HMDs), users are now seeking new immersive and engaging experiences that enhance the 2D video conferencing applications with a third dimension. Immersive video formats such as light fields and volumetric video aim to enhance the experience by allowing for six degrees-of-freedom (6DoF), resulting in users being able to look and walk around in the virtual space. We present a novel, open source, many-to-many streaming architecture using point cloud-based volumetric video. To ensure bitrates that satisfy contemporary networks, the Draco codec encodes the point clouds before they are transmitted using web real-time communication (WebRTC), all while ensuring that the end-to-end latency remains acceptable for real-time communication. A multiple description coding (MDC)-based quality adaptation approach ensures that the pipeline can support a large number of users, each with varying network conditions.

In this demo, participants will be seated around a table and will engage in a virtual conference using an HMD, with each participant being captured using a single depth camera. To showcase the quality effectiveness of the MDC-based adaptation algorithm, a dashboard is used to monitor the status of the application and control the bandwidth available to each participant. The available bandwidth and position of the user are taken into account to dynamically assign a quality level to each participant, ensuring a higher quality experience compared to having a uniform quality level for each point cloud object.

CCS CONCEPTS

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

KEYWORDS

Volumetric Video, Adaptive Streaming, WebRTC, Virtual Conferencing, Multiple Description Coding, Virtual Reality

MMSys '24, April 15–18, 2024, Bari, Italy

ACM ISBN 979-8-4007-0412-3/24/04…$15.00
https://doi.org/10.1145/3625468.3652192

ACM Reference Format:
Matthias De Fré, Jeroen van der Hooft, Tim Wauters, Filip De Turck. 2024. Demonstrating Adaptive Many-to-Many Immersive Teleconferencing for Volumetric Video. In *ACM Multimedia Systems Conference 2024 (MMSys '24), April 15–18, 2024, Bari, Italy*. ACM, Bari, Italy, 6 pages. https://doi.org/10.1145/3625468.3652192

1 INTRODUCTION

In recent years, the use of many-to-many conferencing tools, such as Microsoft Teams and Zoom, has increased significantly [12]. These platforms offer the ability to set up a virtual conference call, allowing for remote meetings. However, these tools offer a limited experience in terms of interactivity and participants often lack a personal connection with each other compared to face-to-face conversations [15]. To address these limitations, immersive six degrees-of-freedom (6DoF) video approaches have emerged as a potential solution. Unlike traditional 2D conferencing tools, 6DoF streaming allows for both rotational and positional freedom [1], enabling participants to interact with each other and allowing them to view each other from multiple angles. This newfound freedom helps to create a more personal experience between participants.

Immersive 6DoF video content is represented by one of two distinct classes. Image-based videos, such as light fields, use the viewing angle of the user to generate a 2D image by using a large number of images that were captured with a specialized camera [6]. In contrast, volumetric-based approaches use cameras to capture a 3D representation of an object. These representations, such as meshes and point clouds, are subsequently placed in a virtual environment, allowing them to be viewed from all angles and positions [2]. Compared to light field video, volumetric videos have lower bandwidth and computational requirements, which is imperative to ensure low bitrates and latency required for real-time communication [26].

Despite having significantly lower bandwidth requirements compared to image-based videos, contemporary network infrastructures still struggle to supply bitrates sufficient enough to stream volumetric videos. Therefore, it is crucial to implement additional mechanisms, such as compression or adaptation, in order to enable the use of such immersive content by a variety of users with varying network conditions and bandwidth restrictions: not all objects are always visible to each user, and sending objects that are currently outside of the field-of-view (FoV) results in a waste of bandwidth. An intelligent streaming system should be capable of utilizing the position and FoV, which are tracked by the head-mounted display (HMD) [11], to dynamically assign a quality level to each object in the scene. This involves assigning a higher quality to objects that are closer in proximity and a lower quality to those farther away [27].

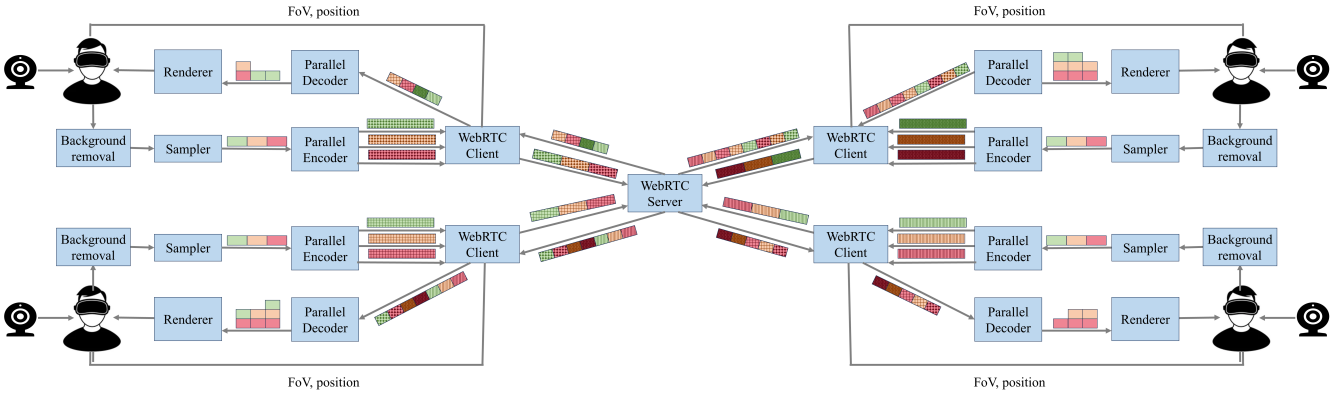


Figure 1: The system architecture for the used multiple description coding (MDC)-based many-to-many volumetric video streaming solution, which requires a fixed number of encoders per participant based on the amount of descriptions. The number of local decoders for each client scales with the number of received descriptions.

Virtual conferencing requires real-time latency, and therefore relies on a transport protocol that is capable of meeting these demands. Transmission control protocol (TCP)-based approaches, such as low-latency HLS (LL-HLS) and low-latency DASH (LL-DASH), struggle in achieving real-timeliness by having a latency between 1 and 5 seconds [29]. This is mainly due to the overhead caused by the forced reliability and flow control mechanisms of TCP, as well as the grouping of frames into a segment [5]. User datagram protocol (UDP)-based solutions do not have these restrictions and allow for the implementation of these mechanisms at the application layer, resulting in a lower latency [4]. Web real-time communication (WebRTC) is such an example, and has been used to ensure low-latency transmission for existing 2D conferencing solutions [29].

In our previous research [7], we proposed a one-to-many streaming pipeline designed for conferencing in virtual reality (VR). This proposed architecture used an MDC-based quality adaptation solution that adapts the quality of input frames based on the network condition and position of the receiving user. This pipeline served as a basis to create a demo with the following contributions:

- An extension to our one-to-many pipeline to allow for many-to-many virtual conferencing [21].
- A dashboard to view the overall state of the application and control the available bandwidth for each participant.
- A demo which showcases a many-to-many virtual conference with up to four participants, illustrating the increased quality of our MDC-based approach compared to uniformly dividing the bandwidth between all other participants.
- A Unity application which renders the received point cloud objects in a 3D environment and streams the rendered frames to a HMD.

The remainder of the paper is organized as follows. Section 2 presents the implementation details of our many-to-many system architecture. Following this, Section 3 presents our demo goal, setup, and functionalities of our dashboard used to control demo parameters. Finally, Section 4 concludes the paper with a brief overview of the discussed topics and future improvements to the pipeline.

2 SYSTEM ARCHITECTURE

For the demonstration, we employ the MDC-based pipeline as depicted in Figure 1. The MDC-based approach facilitates the creation of multiple combinable quality representations from one single captured object, allowing for a wider range of quality representations with differing bitrates. This section contains an extensive overview of each component of this pipeline and its respective implementation details.

2.1 Capturing

Each user utilizes a single D455 Intel RealSense [16] camera with a frame rate of 30 FPS and a resolution of 848x480 pixels. This camera captures both depth and color images, which are transformed into a point cloud representation using a mapping function based on an example function included in the RealSense SDK [17]. Certain performance improvements have been incorporated into this function, optimizing it in order to reduce the overall latency. By default, the camera may occasionally produce an image with numerous gaps when capturing in certain environments, thereby reducing the quality of the resulting point cloud. However, by maximizing the laser power of the camera, visual quality is greatly enhanced.

2.2 Preprocessing

The raw point cloud produced by the camera is too large (1.4 Gbit/s) to be transmitted directly over contemporary networks. Additionally, the camera also captures background details that are not needed for virtual conferencing. To reduce the required bitrate, following preprocessing steps are executed:

- A rudimentary distance filter removes points that are too far away from the camera.
- The number of points in the point cloud is limited to a maximum, preventing sudden bandwidth increases between consecutive frames.
- Three fixed percentages (15%, 25%, 60%) are used to uniformly sample the point cloud into three descriptions each containing a distinct fraction of the points corresponding with their respective percentage.

- The Draco [8] codec encodes the three resulting descriptions in parallel.

2.3 WebRTC Transport

To achieve minimal delay, WebRTC is used to facilitate real-time communication between the capturing and rendering components. WebRTC is a UDP-based solution, which achieves lower latency by eliminating the overhead introduced by TCP mechanisms such as forced reliability and acknowledgments. However, the presented pipeline requires frames to be fully received before they can be decoded. To address this issue, features such as retransmissions of lost packets are implemented in software using methods more optimal for low-latency transmission.

Before streaming the point clouds, the clients first need to establish a connection to the multipoint control unit (MCU) using a signaling algorithm. This algorithm involves exchanging session description protocol (SDP) messages that contain details on supported codecs, as well as available network routes [24]. The specifics of implementing the signaling process are not standardized and there are various methods available. In this particular pipeline, a websocket server is co-located at the WebRTC server. Clients are able to setup a websocket connection between them and this server. Subsequently, this connection is used to exchange a variety of messages between the server and the client. Initially, this connection is only used for the exchange of the SDP messages. However, once the WebRTC connection is set up, it is also used by the client to transmit his current position and FoV.

The Golang programming language [10] is used to implement both the WebRTC server and client applications. Additionally, the Pion [23] Golang package is an ideal candidate to implement the WebRTC communication due to its preexisting implementation of the WebRTC protocol suite. This package includes server-side bandwidth estimation using the Google congestion control (GCC) [14] algorithm. Pion allows for the transmission of data via both data channels and media tracks. Although both methods could be adapted for point cloud video, the use of media tracks enables access to the congestion control feedback mechanisms implemented in Pion and allows complex bandwidth estimation using GCC. As the pipeline uses a novel frame format, a new WebRTC media track type is created to accommodate the differences in transcoding and packetization compared to traditional 2D video. The implementation of this track is based on the included track types, with the major exception being a custom payloader and an RTP packet write function. The *payloader* interface is invoked by the packetizer to divide the frame into multiple packets. Furthermore, to ensure that frames remain decodeable in the event of packet loss, any lost packets are retransmitted using a negative acknowledgment (NACK)-based solution.

The GCC implementation used by Pion employs transport wide congestion control (TWCC) feedback messages allowing for sender-side estimation [13], compared to receiver estimated maximum bitrate (REMB) messages that estimate the bandwidth at the receiver side [3]. By default, the client transmits these messages every 100 ms. However, the GCC specification [14] recommends to send these messages at least once per frame. With the default Pion interval of 100 ms this comes down to sending a TWCC message every three frames when using a 30 FPS stream. The interval of 100 ms does not cause significant problems when dealing with small frames that can be easily packetized into a small number of packets. However, with our encoded point cloud frames, each of which has an average size of 0.3 MB, the frame is segmented into a larger number of packets. Fortunately, Pion allows for the adjustment of the interval delay for TWCC reports. By lowering the interval time to 25 ms, the loss controller no longer reports incorrect packet loss and accurately calculates the loss of the connection.

Figure 2: Quality category is assigned based on distance and field of view [7].

Table 1: Available quality representations for each quality category.

% of original point cloud	15	25	40	60	75	85	100
Quality Category	Low	Medium		High			

2.4 Quality Adaptation

The quality adaptation uses a MDC-based approach, which combines the descriptions generated in the preprocessing step to obtain a broader spectrum of available qualities with differing bitrates. In general, n descriptions result in $2^n - 1$ possible combinations. In this demo, the pipeline uses $n = 3$, which results in seven possible representations for each user.

Using the estimated bitrate obtained from the GCC algorithm, together with the position and FoV of each user, quality adaptation takes place on the central server. The applied algorithm aims for a fair approach which searches for a balance between assigning a high quality to nearby client frames and a lower quality to those further away. The distance between the frame and the receiver's position determines the maximum assignable quality level, as illustrated in Figure 2. In the first step, the algorithm uses an iterative approach which first assigns bitrate to the furthest frames before continuing to the closest frames. Each frame is assigned at least the minimal quality, even if there is not enough available bitrate. The second step starts from the opposite direction and causes the closer frames to reallocate bitrate from the further frames until there is enough bitrate for at least their minimal quality.

The first step of the bitrate allocation adopts a fair approach by initially allocating bitrates to the frames that are farthest away but still visible. Initially, all frames are assigned a quality category based on the distance to the receiving user, as shown in Figure 2. This category determines the potential quality representations that can be assigned, as shown in Table 1, and thus how much bitrate can be assigned to each frame.

Initially, all frames within a category are assigned the same quality level, which is calculated by uniformly distributing the current bitrate, with respect to the maximum assignable quality, between all frames in that category, as seen in Algorithm 1. If there

Algorithm 1 Bitrate allocation step 1: initial bitrates allocation.

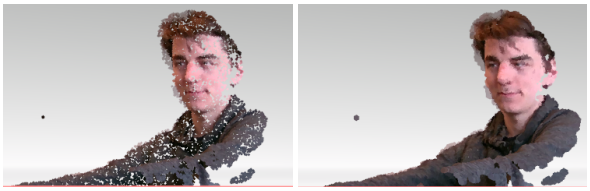
```
1:  bitrate ← getAvailableBitrate()
2:  ratesPerCategory ← Init array with zeros
3:  for c ← nCategory − 1 to 0 do
4:      catBitrate ← getFrameQuality(bitrate, c)
5:      bitrate ← bitrate − catBitrate
6:      ratesPerCategory[c] ← catBitrate
7:  end for
8:  if bitrate ≥ 0 then Concatenate client frames
9:  else Go to next step (Algorithm 2)
10: end if
```

is insufficient bandwidth available, all frames in that category are assigned the lowest quality level of that category (lines 3-7), even if this means that excessive bandwidth will be allocated. In such instances, the quality of frames must be reduced to ensure that the available bitrate remains positive, which is performed in the next step of the algorithm.

The second step of the algorithm, shown in Algorithm 2, addresses the situation where the current frame quality configuration exceeds the available bandwidth. Unlike the previous step, this step adopts a greedy approach, allowing higher quality frames to reallocate bitrates from lower quality levels in an attempt to regain sufficient bitrate for their own quality level (first iteration of inner loop lines 2-11). The reduction in quality within a category is performed on a frame-by-frame basis, starting from the frames that are the furthest from the center, and continues until either enough bitrate is regained or no more frames remain in the category. To prevent a single category from monopolizing all the available bandwidth, only the category directly beneath the current category can be downgraded in each iteration. This ensures that frames which are close, but not in the center of the FoV, still receive some bitrate. Following this, if there is still insufficient bitrate, frames within the category itself will be downgraded (second iteration of inner loop lines 2-11). This iterative downgrading is repeated until either enough bitrate is regained or there are no more frames in the current category. This entire process is repeated for each of the categories, with frames in the lowest category being discarded rather than downgraded. Finally, any remaining bandwidth is used to improve the quality of objects that have been assigned the lowest quality.

Algorithm 2 Bitrate allocation step 2: Quality downgrading.

```
1:  for c ← 0 to nCategory − 1 do
2:      for q ← c + 1 to c do
3:          while bitrate < 0 and framesInCategory > 0 do
4:              furthest ← getFurthestFrameCategory(q)
5:              gainedBitrate ← lowerFrameQuality(furthest)
6:              bitrate ← bitrate + gainedBitrate
7:              if Quality reduced to lowest level for category then
8:                  downgradeClientQualityLevel(furthest)
9:              end if
10:         end while
11:     end for
12: end for
```



(a) 40% sampled point cloud (b) Upscaled point cloud

Figure 3: Increasing the size of the rendered points by a factor of 1.6 enables a simple and fast upscaling method [7].

2.5 Decoding and Rendering

Unity, a game engine commonly used to develop video games [25], is used to design the application responsible for rendering the point clouds. Unity's extensive support for virtual reality makes it an ideal candidate to use for developing a virtual conferencing application. The OpenXR [20] plugin included within Unity enables the use of several HMDs. For this demo, the Meta Quest 2 [18] headset will be used in conjunction with the Meta Quest Link application, allowing us to build a PC VR application and stream the rendered frames to the Meta Quest headset. It is worth noting that streaming the content rendered in Unity to the headset introduces additional latency. However, when employing a USB 3.0 connection, this latency is negligible (1 ms) compared to USB 2.0 (30 ms) or wireless streaming (50 ms). The encoded point clouds are received in the Unity application via a UDP socket proxy which communicates with the Pion WebRTC client.

Unlike the encoding process, the number of received descriptions varies based on factors such as the available bandwidth and the distance to the other participants. For this reason, a thread pool is used with a number of concurrent workers. Whenever a worker becomes available, a decoding job is dequeued. Once decoding of a single object is complete, all corresponding descriptions are merged together into a single point cloud object. The decoding operation utilizes the official Draco Unity plugin [9].

Visualizing the point cloud in the application is accomplished by using the Pcx [22] package. Ordinarily, this package is employed to import point clouds during the development of the application rather than during runtime. However, by leveraging the shader included in the package and applying it to a mesh component, Unity renders the vertices of the mesh as individual point primitives. As a result, the vertices can be set at runtime in order to achieve dynamic point cloud rendering.

By utilizing point primitives instead of triangles, we have the ability to increase the size of the points in order to increase the density of the point cloud, and subsequently the overall quality. A sampled version of the point cloud can be used to a visually similar result whilst maintaining lower bandwidths. Figure 3 shows an example of this simple upscaling effect when applied to a 40% sampled point cloud. However, this method of upscaling only works when utilizing an OpenGL [19] or Vulkan [28]-based graphics pipeline, as only these pipelines support manipulation of the size of the point primitive.

3 DEMONSTRATION

The demonstration will consist of a many-to-many virtual conferencing scenario, assuming a fixed camera position per participant

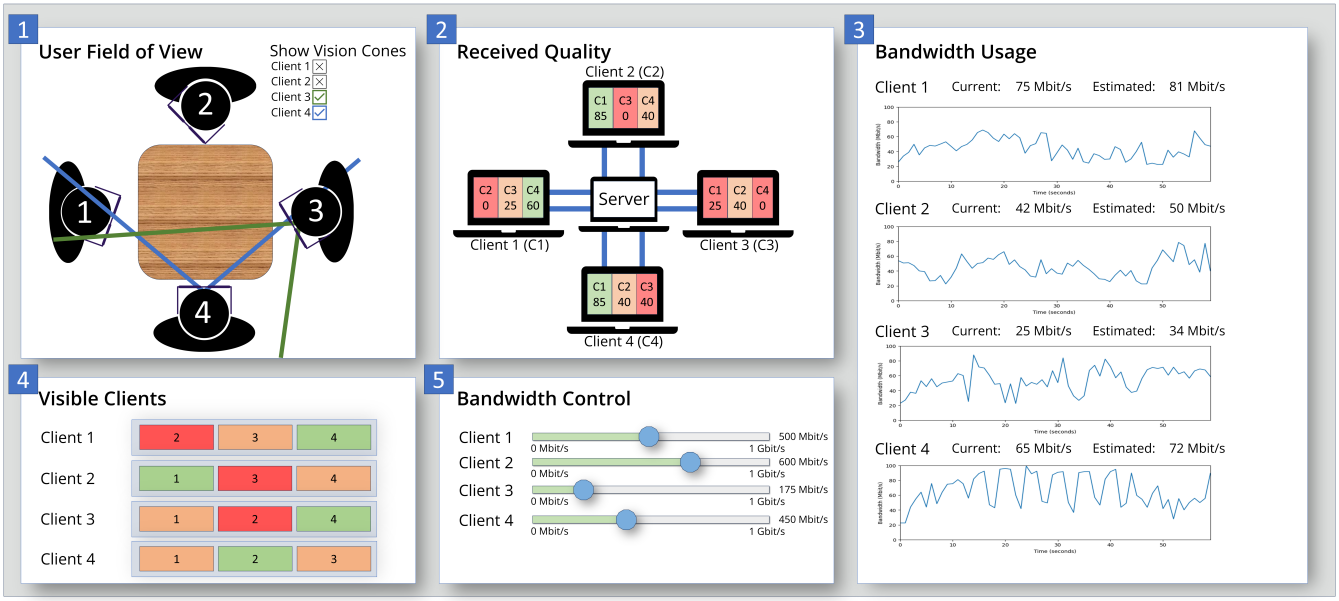


Figure 4: The dashboard that will be used during the demo. This dashboard allows us to view the current quality, latency and bandwidth usage for each user, and to change parameters such as the available bandwidth and network latency.

with the ability to look around the virtual environment. We aim to illustrate the positive impact on the video quality of the proposed MDC-based adaptation approach, which assigns bandwidth based on visibility and distance to the user, compared to the uniformly dividing the bandwidth. The demo consists of the following three phases:

(1) Sufficient bandwidth to stream every point cloud object at high quality.
(2) Lower bandwidth and assign quality uniformly to each object.
(3) Lower bandwidth and assign quality using the proposed MDC-based approach, ensuring that the currently viewed objects are assigned a greater quality

3.1 Setup

The demonstration will take place with a maximum of four participants seated around a table. Each participant will be captured using an Intel RealSense D455 camera. A Meta Quest 2 HMD allows the participants to see each other in a virtual space rendered by the Unity application. Both this application and the WebRTC client will run on a Windows 10 laptop with the following specifications: *CPU: i7 11th Generation, GPU: RTX 3070 RAM: 32GB.*

The central server application will be executed on a separate laptop with identical specifications, running the Ubuntu 20.04 operating system. This setup facilitates the use of traffic control (TC) in order to vary the bandwidth available to each participant. Using TC together with internet protocol (IP)-based filters, a separate bandwidth can be assigned to each participant by using the IP address of each user. This laptop will be physically connected to a network switch, allowing each client to connect to the server via the Ethernet interfaces of the switch.

3.2 Dashboard

During the demonstration, a dashboard, shown in Figure 4, is used to provide an overview of various metrics and alter the demo parameters, such as available bandwidth. This dashboard contains the following panels:

(1) Illustrates the FoV of each client.
(2) Indicates, with a color and a numerical value, which quality level is received for each client.
(3) Shows the used bandwidth and estimated bandwidth, as well as a graph illustrating the bandwidth trace for each user.
(4) Shows which clients are visible for each participant, as well as having the ability to change the position of each participant in the virtual environment.
(5) Is used to change the bandwidth available to a client.

This dashboard is used to showcase the overall quality improvement (panel 4) that occurs when using the proposed MDC-based approach compared to uniformly distributing the quality between the point cloud objects. Furthermore, it shows that the MDC-based approach uses available bandwidth more optimally (panel 3).

4 CONCLUSION

In this paper we propose a demonstration of a many-to-many volumetric streaming architecture, with the aim of illustrating the possibility of multi-user conferencing in VR. The demo showcases the impact of bandwidth allocation based on the position and FoV of the user. This MDC-based adaptation algorithm has the goal of dynamically assigning a quality level to each of the point cloud objects representing the other participants, while maintaining a fair quality balance between the nearby objects and those objects that are further away.

In future research, we will be containerising the encoding and decoding modules and deploy these containers in a cluster-based

setup. In addition to allowing for more active users, this cluster also enables the implementation of other modules such as viewport prediction and server-side rendering. Having an accurate viewport prediction facilitates the use of tiling, which would allow the MDC-based adaptation to assign a separate quality to parts of the object, rather than the complete object. This provides flexibility to assign a higher quality level to specific regions of the content that are more important than others, such as the head and the hands of the participants.

ACKNOWLEDGMENTS

This work has been funded by the European Union (SPIRIT project, Grant Agreement 101070672, https://www.spirit-project.eu/).

REFERENCES

[1] Sun Joo Ahn, Laura Levy, Allison Eden, Andrea Stevenson Won, Blair MacIntyre, and Kyle Johnsen. 2021. Ieeevr2020: exploring the first steps toward standalone virtual conferences. *Frontiers in Virtual Reality*, 2, 648575.

[2] Dimitrios S Alexiadis, Dimitrios Zarpalas, and Petros Daras. 2012. Real-time, full 3-d reconstruction of moving foreground objects from multiple consumer depth cameras. *IEEE Transactions on Multimedia*, 15, 2, 339–358.

[3] Harald T. Alvestrand. 2013. RTCP message for Receiver Estimated Maximum Bitrate. Internet-Draft draft-alvestrand-rmcat-remb-03. Work in Progress. Internet Engineering Task Force, (Oct. 2013). 8 pp. https://datatracker.ietf.org/doc/draft-alvestrand-rmcat-remb/03/.

[4] Muhammad Ajmal Azad, Rashid Mahmood, and Tahir Mehmood. 2009. A comparative analysis of dccp variants (ccid2, ccid3), tcp and udp for mpeg4 video applications. In *2009 International Conference on Information and Communication Technologies*. IEEE, 40–45.

[5] Nassima Bouzakaria, Cyril Concolato, and Jean Le Feuvre. 2014. Overhead and performance of low latency live streaming using mpeg-dash. In *IISA 2014, The 5th International Conference on Information, Intelligence, Systems and Applications*. IEEE, 92–97.

[6] Michael Broxton et al. 2020. Immersive light field video with a layered mesh representation. *ACM Transactions on Graphics (TOG)*, 39, 4, 86–1.

[7] Matthias De Fré, Jeroen van der Hooft, Tim Wauters, and Filip De Turck. 2024. Scalable mdc-based volumetric video delivery for real-time one-to-many webrtc conferencing.

[8] [SW] Google, Draco 2023. URL: https://google.github.io/draco/.

[9] [SW] atteneder, Draco Unity Plugin 2023. URL: https://github.com/atteneder/DracoUnity.

[10] [SW], Golang 2023. URL: https://go.dev/.

[11] Michael J Gourlay and Robert T Held. 2017. Head-mounted-display tracking for augmented and virtual reality. *Information Display*, 33, 1, 6–10.

[12] Janine Hacker, Jan vom Brocke, Joshua Handali, Markus Otto, and Johannes Schneider. 2020. Virtually in this together–how web-conferencing systems enabled a new virtual togetherness during the covid-19 crisis. *European Journal of Information Systems*, 29, 5, 563–584.

[13] Stefan Holmer, Magnus Flodman, and Erik Sprang. 2015. RTP Extensions for Transport-wide Congestion Control. Internet-Draft draft-holmer-rmcat-transport-wide-cc-extensions-01. Work in Progress. Internet Engineering Task Force, (Oct. 2015). 11 pp. https://datatracker.ietf.org/doc/draft-holmer-rmcat-transport-wide-cc-extensions/01/.

[14] Stefan Holmer, Henrik Lundin, Gaetano Carlucci, Luca De Cicco, and Saverio Mascolo. 2016. A Google Congestion Control Algorithm for Real-Time Communication. Internet-Draft draft-ietf-rmcat-gcc-02. Work in Progress. Internet Engineering Task Force, (July 2016). 19 pp. https://datatracker.ietf.org/doc/draft-ietf-rmcat-gcc/02/.

[15] Emily J Hurst. 2020. Web conferencing and collaboration tools and trends. *Journal of Hospital Librarianship*, 20, 3, 266–279.

[16] [SW], Intel Realsense 2023. URL: https://www.intelrealsense.com/.

[17] [SW] intel, Intel Realsense cepth/color frame to point cloud 2023. URL: https://github.com/IntelRealSense/librealsense/blob/master/wrappers/pcl/pcl-color/rs-pcl-color.cpp.

[18] [SW] Meta, Meta Quest 2 Virtual Reality Headset 2023. URL: https://www.meta.com/quest/products/quest-2.

[19] [SW], OpenGL 2023. URL: https://www.opengl.org/.

[20] [SW], OpenXR 2023. URL: https://www.khronos.org/api/index_2017/openxr.

[21] [SW], Point Cloud Streaming Application 2023. URL: https://github.com/MatthiasDeFre/webrtc-pc-streaming.

[22] [SW] keijiro, https://www.meta.com/be/en/quest/products/quest-2/ 2023. URL: https://github.com/keijiro/Pcx.

[23] [SW], Pion WebRTC 2023. URL: https://github.com/pion/webrtc.

[24] Branislav Sredojev, Dragan Samardzija, and Dragan Posarac. 2015. Webrtc technology overview and signaling solution design and implementation. In *2015 38th international convention on information and communication technology, electronics and microelectronics (MIPRO)*. IEEE, 1006–1009.

[25] [SW], Unity 2023. URL: https://unity.com.

[26] Jeroen van der Hooft, Hadi Amirpour, Maria Torres Vega, Yago Sanchez, Raimund Schatz, Thomas Schierl, and Christian Timmerer. 2023. A tutorial on immersive video delivery: from omnidirectional video to holography. *IEEE Communications Surveys & Tutorials*.

[27] Jeroen van der Hooft, Tim Wauters, Filip De Turck, Christian Timmerer, and Hermann Hellwagner. 2019. Towards 6DoF HTTP adaptive streaming through point cloud compression. In *Proceedings of the 27th ACM International Conference on Multimedia*, 2405–2413.

[28] [SW], Vulkan 2023. URL: https://www.vulkan.org/.

[29] Wowza. 2021. 2021 Video Streaming Latency Report. Tech. rep. Wowza.