Cloud Rendering-based Volumetric Video Streaming System for Mixed Reality Services

Serhan Gül
serhan.guel@hhi.fraunhofer.de
Fraunhofer HHI

Dimitri Podborski
dimitri.podborski@hhi.fraunhofer.de
Fraunhofer HHI

Jangwoo Son
jangwoo.son@hhi.fraunhofer.de
Fraunhofer HHI

Gurdeep Singh Bhullar
gurdeepsingh.bhullar@hhi.fraunhofer.de
Fraunhofer HHI

Thomas Buchholz
thomas.buchholz@telekom.de
Deutsche Telekom AG

Thomas Schierl
thomas.schierl@hhi.fraunhofer.de
Fraunhofer HHI

Cornelius Hellge
cornelius.hellge@hhi.fraunhofer.de
Fraunhofer HHI

ABSTRACT

Volumetric video is an emerging technology for immersive representation of 3D spaces that captures objects from all directions using multiple cameras and creates a dynamic 3D model of the scene. However, processing volumetric content requires high amounts of processing power and is still a very demanding task for today's mobile devices. To mitigate this, we propose a volumetric video streaming system that offloads the rendering to a powerful cloud/edge server and only sends the rendered 2D view to the client instead of the full volumetric content. We use 6DoF head movement prediction techniques, WebRTC protocol and hardware video encoding to ensure low-latency in different parts of the processing chain. We demonstrate our system using both a browser-based client and a Microsoft HoloLens client. Our application contains generic interfaces that allow for easy deployment of various augmented/mixed reality clients using the same server implementation.

KEYWORDS

volumetric video, augmented reality, mixed reality, edge cloud, cloud rendering

1 INTRODUCTION

Recent technical advances in capturing and displaying immersive media sparked a huge market interest in virtual reality (VR) and augmented reality (AR) applications. Although the initial focus was more on omnidirectional (360°) video applications, with advances in volumetric capture technology as well as availability of mobile devices that are able to register their environment and place 3D objects at fixed places, the focus has started to shift towards volumetric video applications [13].

Volumetric video captures an object or scene with multiple cameras from all directions and create a dynamic 3D model of that object [14] . Users can view such content using an AR device (e.g. an optical see-through AR display) with accurate six degrees of freedom (6DoF) positional trackers, or on a 2D screen albeit in a less immersive fashion. Volumetric video is expected to enable novel use cases in the entertainment domain (e.g. gaming, sports replay) as well as in cultural heritage, education, and commerce [3, 14].

Despite the significant increases in computing power of mobile devices, rendering rich volumetric videos on such devices is still a very demanding task. Especially, presence of multiple volumetric objects in the scene can increase the processing complexity significantly. Furthermore, efficient hardware implementations for decoding of volumetric data (e.g. point clouds or meshes) are still not available, and software decoding can be very demanding in terms of battery usage and real-time rendering requirements.

One way to reduce the processing load on the client is to send a 2D view of the volumetric object that is rendered according to the actual user position, rather then sending the entire volumetric content to the client. This is achieved by offloading the expensive rendering process to a a powerful server and transmitting the rendered views over a network to less powerful client devices. This technique is typically known as remote (or interactive) rendering [15]. Another advantage of this approach is that network bandwidth requirements are significantly reduced because only a single 2D video is transmitted instead of full 3D volumetric content. The rendering server can also be deployed within a cloud computing platform to provide flexible allocation of computational resources and scalability based on changes in processing load.

Recently, the reduced network latency enabled by the emerging 5G networks fostered a resurgence of interest in cloud rendering applications, especially in the domain of cloud gaming [9]. Nvidia has released an SDK called CloudXR [11] which aims to deliver advanced graphics performances to *thin* clients by rendering complex immersive content on Nvidia cloud servers and streaming only the result to the clients. Google Stadia [5], a cloud gaming service operated by Google, has already been launched in November 2019.

While cloud-based rendering helps to reduce the processing load on the client side, a major drawback is the added network latency which is not present in systems that perform rendering entirely on the end device. In addition to the added network latency, rendering and encoding on the server side also contribute to the increase in the end-to-end latency [6]. Degraded user experience and motion sickness are known to be common consequences of a perceivable motion-to-photon (M2P) latency [1, 2]. Therefore, it is crucial to employ latency reduction techniques in every element of the processing chain.

One way to reduce latency is to serve the volumetric content from a geographically closer *edge* server, thereby reducing the

network latency [8]. Also, deployment of recent real-time communication protocols such as WebRTC is crucial for meeting the demands of interactive low-latency streaming applications [7]. In terms of video encoding latency, the use of fast hardware-based video encoders (e.g. Nvidia NVENC [12]) is critical for reducing video compression latency while maintaining good quality and compression efficiency. Finally, the system should predict the future position of the user using various prediction algorithms to further minimize the perceived end-to-end latency [4].

In this work, we present a system for low-latency volumetric video streaming using the cloud rendering concept that comprises a server and two different client implementations. To ensure low latency streaming, we utilize 6DoF head movement prediction techniques as well as the WebRTC protocol together with Nvidia hardware encoder (NVENC). Our system provides generic interfaces such that any client implementation can be deployed using our server implementation. For our demonstration, we have implemented a client application for Microsoft HoloLens and another browser-based client that can be run on different browsers.

2 SYSTEM ARCHITECTURE

This section describes the key components of our cloud-based volumetric video streaming system as well as the dataflow and interfaces between these components.

2.1 Server side architecture

An overview of the overall system architecture is shown in Fig. 1. Our server implementation consists of a volumetric video player and a generic cross-platform *cloud rendering library* that can be integrated into different applications.

The **volumetric video player** is implemented using Unity with several native plug-ins. The player is able to play volumetric sequences stored in a single MP4 file which consists of a video track containing the compressed texture data, and a mesh track containing the compressed mesh data. Before the start of the playback, the player registers all game objects (e.g. a volumetric object stored as an MP4 file or a virtual camera) that are to be controlled by the cloud rendering library and/or the client. After registration, the player can start playing the MP4 file by demultiplexing it and feeding the elementary streams to the corresponding video, audio and mesh decoders. Then, each mesh is synchronized with the corresponding texture and rendered to the scene. Then, the camera representing the client's viewport captures the rendered view of the volumetric object and passes the RenderTexture to the cloud rendering library for further processing. The player concurrently asks the library for the latest positions of the previously registered game objects and updates the rendered view accordingly.

The **cloud rendering library** is a cross-platform library written in C++ that can be easily integrated into different applications. In our Unity application, we integrated it as a native plug-in into the player. The library contains various modules for application control, media processing, and the communication interfaces between the server and the client. The main modules of our library are the WebSocket (WS) Server, GStreamer module, Controller, ObjectPool and Prediction Engine. Each module runs asynchronously in its own thread to achieve high performance.

The **WebSocket Server** is used for exchanging signaling data between the client and the server. Such signaling data includes Session Description Protocol (SDP), Interactive Connectivity Establishment (ICE) as well as application-specific metadata for scene description. The WS connection is also used for transmission of the control data in order to modify the position and orientation of any registered game object or camera. Our system also allows the usage of WebRTC data channels for control data exchange after a peer-to-peer connection is established. Both plain and secure WebSockets are supported which is important for running the system in real use cases.

The **Gstreamer** module contains the media processing pipeline which takes the rendered texture (at 60 fps) as input, compresses it as a video stream, and transmits it to the client using WebRTC. Specifically, the Unity RenderTexture is inserted into the pipeline using the *appsrc* element of GStreamer. Since the texture is in RGBA format, it has to be passed through a *videoconvert* element to bring it to the I420 format accepted by the encoder. We use the Nvidia encoder (NVENC) to compress the texture using H.264/AVC (high profile, level 3.1, IPPP.. GOP structure, no B frames) but it is also possible to encode in H.265/HEVC using NVENC, or replace the encoder block with a different encoder e.g a software encoder such as x264. Finally, the resulting compressed bitstream is packaged into RTP packets, encrypted, and sent to the client using the WebRTC protocol. For our volumetric sequence, the bitrate of the encoded bitstream (at a resolution of 1280×720) varies between 3-9 Mbps depending on the size of the volumetric object inside the viewport and user movement. Our system can also generate videos at 1080p and 4K resolution.

The **ObjectPool** is a logical structure that maintains the states of all registered objects (such as position, orientation and scale) enabling the client to position virtual objects correctly.

The **Controller** contains the application logic and controls the other modules depending on the application state. For example, it closes the media pipeline if a client disconnects, and re-initializes the pipeline when a new client is connected. The controller is also responsible for creating response messages for the client. For example, when the client requests the scene description, the WS Server notifies the controller that the request has arrived, and the controller creates a response message in JSON format with all available objects from the pool. The response message goes back to the WS Server which deals with the transmission it back to client.

The **Prediction Engine** implements a 6DoF user movement prediction algorithm to predict the future head position of the user based on her past movement patterns. Based on the client interaction and the predictions output by the Prediction Engine, the Controller updates the positions of the registered objects accordingly such that the rendering engine renders a scene matching to the predicted user position that will be attained after a predefined prediction interval. The prediction interval is set to be equal to the estimated M2P latency of the system. Currently, an autoregressive model (as described in [6]) is implemented but the module allows integration of different kind of prediction techniques e.g. Kalman filtering [4].

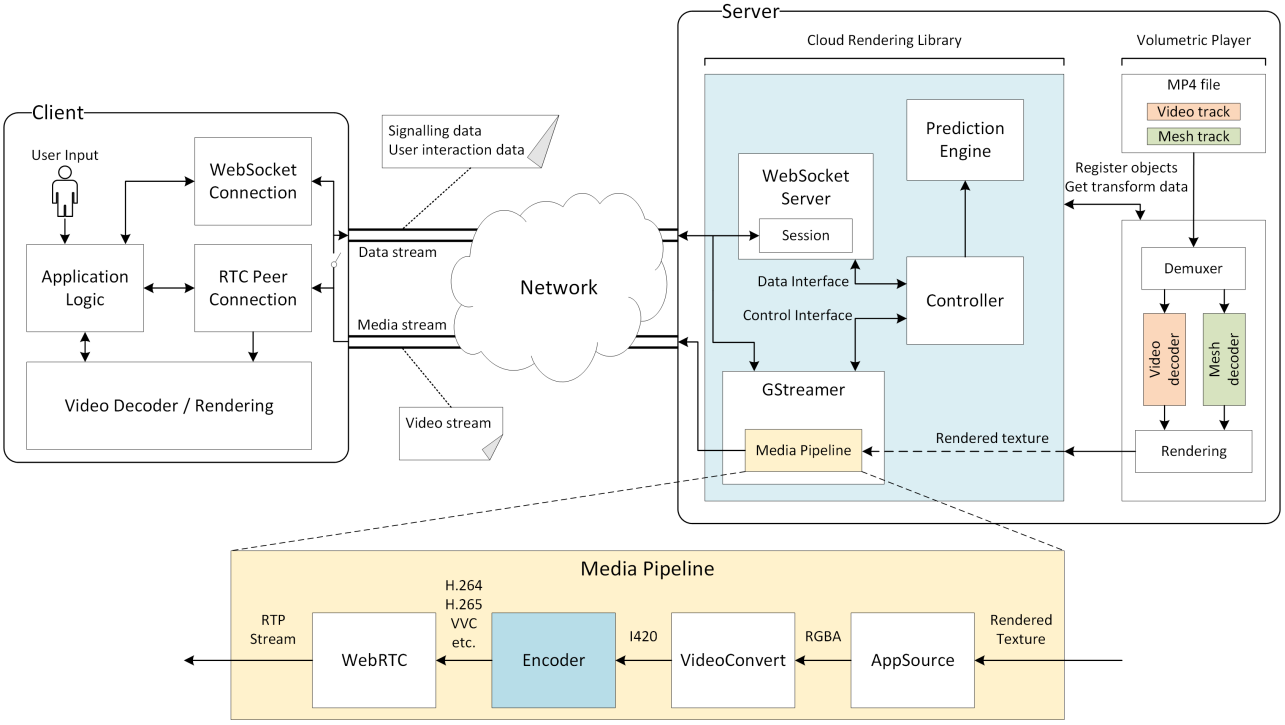

Figure 1: Overview of the system components and interfaces.

2.2 Client side architecture

Our client-side architecture is depicted on the left side of the Fig. 1 and essentially consists of the following modules: WebSocket Connection, WebRTC Connection, Video Decoder, Application Logic, and the client application: a browser client or a native client for Microsoft HoloLens.

Before the streaming session starts, the client establishes a WS connection to the server and asks the server to send a description of the rendered scene. The server responds with a list of objects and parameters. After receiving the scene description, the client replicates the scene and initiates a peer-to-peer WebRTC connection (RTCPeerConnection) with the server. The server and the client begin the WebRTC negotiation process while sending SDP and ICE data over the previously established WS connection. Then, the peer-to-peer connection is established and the client receives a video stream over RTCPeerConnection that contains the rendered view of the 3D scene corresponding to the (predicted) user pose. The client can use the WS connection and send control data back to the server to modify the rendered view. For example, the client may signal the changes in user's pose, or it may rotate, move or scale any volumetric object in the scene based on the user interaction.

We implemented both, a web browser player and a native application for the Microsoft HoloLens. While our browser application targets use cases in which the volumetric content is viewed on a 2D screen such as a tablet or computer display, our HoloLens client targets AR/MR applications that potentially offer better immersion and more natural interactivity.

Browser client. Our browser client is implemented in JavaScript and it uses the *three.js* [16] library that allows to interact with the volumetric object using a mouse, keyboard or touchscreen. Specifically, the user can move the camera around, drag the object to change its position and use range sliders to change the orientation or scaling of the object. The client application has been tested on different web browsers such as Chrome, Firefox, and Safari.

Hololens client. Our HoloLens application is built as Universal Windows Platform (UWP) application with Unity. We build separate applications both for x86 and ARM architectures since HoloLens 1 executes applications on an x86 processor whereas HoloLens 2 uses an ARM processor. On the server side, the renderer designates the user as a camera in Unity 3D space and moves the camera according to the user's position. Due to the properties of the optical see-through display used in HoloLens, black pixels are seen fully transparent while white pixels are seen as increasingly opaque [10]. This display property can be exploited to remove the background of the volumetric object inside the video stream such that the volumetric object is perceived to be overlaid onto the real world. To achieve this, the background pixels (already rendered with a solid color on the server) are set to black in HoloLens using a shader.

The HoloLens app renders the 2D scene onto a plane orthogonal to the user's point of view in the world space (see Fig. 2). When the user changes her position and a new view is rendered on the server, this plane is always rotated towards the user. Thereby, the user perceives the different 2D views rendered onto the orthogonal plane as though a 3D object were present in the scene.

3 DEMONSTRATION

We will demonstrate our cloud-based volumetric video streaming system on the web browser client as well as the HoloLens client. In reality, we are able to deploy our server application at a 5G edge server as well as on a AWS instance. However, due to possible limitations in Internet connectivity, we will use a router to connect both clients over WiFi to a LAN.

Users will be able to interact with the volumetric content using both the browser and HoloLens clients. In both cases, users may scale, rotate, and move object. While using the browser client, the users will also be able to measure the M2P latency using our custom-built latency measurement tool [6]. This tool sends predefined textures (known by the client) depending on the control data received from the client. As soon as the client detects the corresponding texture based on its previous input to the server, it stops the timer and computes the M2P latency. Running the server on an AWS instance in Frankfurt and with the WiFi connected client in Berlin we are measuring an average network latency of 13.3 ms and a M2P latency around 60 ms.

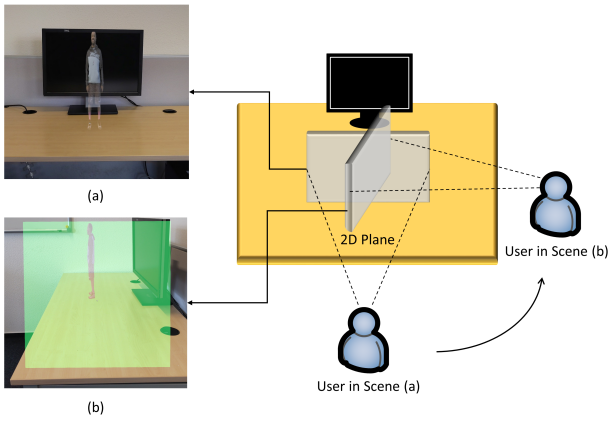

Figure 2: Captured scene from our HoloLens client; (a) Final scene after background removal (b) Scene before background removal.

Fig. 2 shows the implementation of our HoloLens client where the transmitted 2D scene is rendered on the orthogonal plane as shown in the figure. After receiving and decoding the 2D video, the renderer detects a certain range of RGB values in the fragment shader and adjusts the plane background in Fig. 2(b) to the transparent background as in Fig. 2(a). The user can then move the object to any desired position using spatial gestures designed for scaling and dragging 3D objects in HoloLens.

As supplementary material, we provide a video describing our system architecture and showing the functionality of our web browser and HoloLens clients.

4 CONCLUSION

In this paper, we presented a low-latency cloud rendering-based volumetric streaming system. Our system utilizes a powerful server for rendering of volumetric videos and decreases processing requirements and battery usage in end devices. Also, by avoiding the streaming of full 3D data, required bandwidth is significantly reduced. To reduce the increased motion-to-photon latency due to the added network latency, we use 6DoF movement prediction techniques, low latency streaming protocols (WebRTC) and low latency hardware video encoders (NVENC). Based on the developed volumetric streaming system, our future work will include investigation of the effect of latency on the user perception in AR environments as well as development of more effective 6DoF prediction techniques.

REFERENCES

[1] Bernard D. Adelstein, Thomas G. Lee, and Stephen R. Ellis. 2003. Head Tracking Latency in Virtual Environments: Psychophysics and a Model. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting* 47, 20 (Oct. 2003), 2083–2087. https://doi.org/10.1177/154193120304702001

[2] R.S. Allison, L.R. Harris, M. Jenkin, U. Jasiobedzka, and J.E. Zacher. 2001. Tolerance of temporal delay in virtual environments. In *Proceedings IEEE Virtual Reality 2001*. IEEE Comput. Soc, 247–254. https://doi.org/10.1109/vr.2001.913793

[3] John G Apostolopoulos, Philip A Chou, Bruce Culbertson, Ton Kalker, Mitchell D Trott, and Susie Wee. 2012. The road to immersive communication. *Proc. IEEE* 100, 4 (2012), 974–990.

[4] Ronald Tadao Azuma. 1995. *Predictive tracking for augmented reality.* Ph.D. Dissertation. University of North Carolina at Chapel Hill.

[5] Google. 2019. *Google Stadia.* Google Inc. https://stadia.google.com/

[6] Serhan Gül, Dimitri Podborski, Thomas Buchholz, Thomas Schierl, and Cornelius Hellge. 2020. Low Latency Volumetric Video Edge Cloud Streaming. arXiv:2001.06466v1

[7] Christer Holmberg, Stefan Hakansson, and G Eriksson. 2015. Web real-time communication use cases and requirements. *RFC 7478* (2015).

[8] Yun Chao Hu, Milan Patel, Dario Sabella, Nurit Sprecher, and Valerie Young. 2015. Mobile edge computingâĂŤA key technology towards 5G. *ETSI white paper* 11, 11 (2015), 1–16.

[9] Maria A Lema, Andres Laya, Toktam Mahmoodi, Maria Cuevas, Joachim Sachs, Jan Markendahl, and Mischa Dohler. 2017. Business case and technology analysis for 5G low latency applications. *IEEE Access* 5 (2017), 5917–5935.

[10] Microsoft. 2020. *Mixed Reality Rendering.* Microsoft Corporation. Retrieved February 25, 2020 from https://docs.microsoft.com/en-us/windows/mixed-reality/rendering

[11] Nvidia. 2019. *NVIDIA CloudXR Delivers Low-Latency AR/VR Streaming Over 5G Networks to Any Device.* Nvidia Corporation. https://blogs.nvidia.com/blog/2019/10/22/nvidia-cloudxr

[12] Nvidia. 2020. *Nvidia Video Codec SDK.* Nvidia Corporation. Retrieved February 25, 2020 from https://developer.nvidia.com/nvidia-video-codec-sdk

[13] O Schreer, I Feldmann, P Kauff, P Eisert, D Tatzelt, C Hellge, K Müller, T Ebner, and S Bliedung. 2019. Lessons learnt during one year of commercial volumetric video production. In *2019 IBC conference.* IBC, IBC.

[14] Oliver Schreer, Ingo Feldmann, Sylvain Renault, Marcus Zepp, Markus Worchel, Peter Eisert, and Peter Kauff. 2019. Capture and 3D Video Processing of Volumetric Video. In *2019 IEEE International Conference on Image Processing (ICIP).* IEEE, IEEE, 4310–4314.

[15] Shu Shi and Cheng-Hsin Hsu. 2015. A Survey of Interactive Remote Rendering Systems. *Comput. Surveys* 47, 4 (May 2015), 1–29. https://doi.org/10.1145/2719921

[16] three.js 2019. *Three.js, JavaScript 3D Library.* Retrieved February 25, 2020 from https://threejs.org/ Version r101.