Web-based Mixed Reality Video Fusion with Remote Rendering

1st Qiang Zhou
State Key Laboratory of Virtual Reality Technology and Systems, Beihang University
Beijing, China
ZB2006108@buaa.edu.cn

2nd Zhong Zhou
State Key Laboratory of Virtual Reality Technology and Systems, Beihang University
Beijing, China
zz@buaa.edu.cn

***Abstract*—Mixed Reality (MR) video fusion system fuses video imagery with 3D scenes. It makes the scene much more realistic and helps the users understand the video contents and temporal-spatial correlation between them, thus reducing the user's cognitive load. Nowadays, MR video fusion has been used in various applications. However, video fusion systems require powerful client machines because video streaming delivery, stitching, and rendering are computation-intensive. Moreover, huge bandwidth usage is also another critical factor that affects the scalability of video fusion systems. The framework proposed in this paper overcomes this client limitation by utilizing remote rendering. Furthermore, the framework we built is based on browsers. Therefore, the user could try the MR video fusion system with a laptop or even pad, no extra plug-ins or application programs need to be installed. Several experiments on diverse metrics demonstrate the effectiveness of the proposed framework.**

***Index Terms*—mixed reality, video fusion, WebRTC, remote rendering**

I. Introduction

Mixed Reality (MR) video fusion systems have the impressive ability to produce highly comprehensive imagery and to yield temporal-spatial consistent scenes. They thus have been extensively used in many industries such as public security and transportation. MR video fusion systems, on the other hand, face a number of challenges. To begin with, existing MR video fusion systems are built to run on the client side. Still, the core tasks of these systems, such as image encoding and rendering multiple video streams with 3D models, require a powerful server that is beyond the capability of the client's personal computer. Secondly, in some user cases, 3D models are copied and modified by malicious users when loaded by clients. Therefore, the copyright of those models could not be protected within existing MR video fusion systems. Last but not least, with the popularity of web applications, many systems have released their web application formats based on modern browsers. However, existing MR video fusion systems still lack web applications to the best of our knowledge.

We create a web-based MR video fusion framework with remote rendering to address the aforementioned issues. This framework proposes a fusion method for dynamically projecting video images into 3D models as texture. This process runs on a remote server, allowing it to render 3D models with more realistic effects by utilizing powerful servers and taking advantage of advanced graphics card features like ray tracks. By utilizing the encoder module of our framework, the rendered image will be encoded by graphics card hardware, which is more effective than soft encoding. Then the encoded videos are sent to the user's browser through Web Real-Time Communication (WebRTC) [1]–[3] protocol. It will protect the original 3D model from malicious users because the transferred data are video images displayed in the user's browser. Subsequently, the proposed framework is developed to be more user-friendly based on the B/S model, where users could access video fusion services via web browser without pre-installing client software. As shown in our qualitative experiments, users could access a smooth video with high frames per second (FPS) and don't feel lagging when taking action in the browser. It seems the system is just running locally rather than running remotely. To compare our quantitative experiments, we bring a general-purpose game engine, Unreal Engine 4 (UE4). It is found that the proposed framework exceeds UE4 for the metric like bandwidth usage, and these two systems are at the same level when comparing freeze frame count and FPS.

Our main contributions are summarized as follows:

- We propose a web-based MR video fusion framework with remote rendering. In the render server, a camera's model-view matrix and projection matrix are computed based on its position and pose and utilized to project video imagery onto scene models. Then the occlusion is detected by using the depth map of the scene. Selective projection onto camera visible models could accelerate fusion computation and add a far plane for every camera to supplement our scene structure. In the encoder component, several streaming encoding techniques are introduced to a proposed framework to fulfill different scenarios.
- Several experiments are performed on our proposed framework and another remote rendering system. Thus, the proposed framework shows its effectiveness via experiments.

The remainder of the paper is organized as follows. In

This work was supported in part by National Key R&D Program of China (Grant No. 2018YFB2100601), and in part by the National Natural Science Foundation of China (Grant No. 61872024).

Section II, we review the literature related to the proposed framework. The system architecture of our proposed system is presented in Section III. In Section IV, details of MR video fusion rendering are discussed, and we present another key component of the proposed framework, WebRTC-based video streaming with interaction, in Section V. Experimental evaluation is presented in Section VI, and the article concludes in Section VII.

II. Related Work

To achieve the above benefits of a web-based MR video fusion system, two techniques must be discussed. The first is how to fuse multiple video streaming with an augmented virtual environment (AVE). The second is how to render the mixed environment remotely and display it on a web browser.

MR video fusion technology is one of main research aspects in virtual reality (VR) to fusion models of the virtual scenes and objects into the real world. The target is to enrich the expression effects of scene models. Moezzi et al. [4] presented the concept of video and 3D scene fusion. Their system captured objects' motions using cameras with different viewpoints, reconstructed the objects utilizing 3D voxels, and finally fused the reconstructed models to the virtual environment dynamically. The concept of the AVE [5], [6] is firstly proposed by Neumann et al. to promote video augmented virtual scene technology. Based on the conception of a virtual city, they map video captured to the corresponding buildings and terrain models in real time. Several campus scenes are built, and the 3D models are dynamic according to the real-time video image. Sawhney et al. [7] presented a method that exploits real-time videos as the texture of existing 3D models. Firstly, it takes several calibrated camera videos, secondly applies texture mapping technology to render the 3D model in real time. The new method brings a uniform viewpoint for users to observe models and videos. It enhances the spatial expression of videos and extends the user's observation range. Zhou et al. [8], [9] presented a method of multiple video fusion in a 3D environment. Users initially interact with a newly designed background model named video model to register and stitch videos' background frames offline. The method then fuses the offline results to render videos in a real-time manner.

There are several commercial software or web services that can perform remote or cloud rendering. Nvidia has released an SDK called CloudXR, which aims to deliver advanced graphics performances to thin clients by rendering complex immersive content on Nvidia cloud servers and streaming the result to the clients. Google Stadia, a cloud gaming service operated by Google, launched in November 2019, streams games directly to users' desktop, laptop, compatible phone or tablet, or TV with Chromecast Ultra. Unreal Engine presents a new feature called pixel streaming since its version 4.21. With pixel streaming, users run Unreal Engine applications remotely on a computer that they probably never see. The Unreal Engine uses the resources available to that computer like CPU, GPU, and memory to run the game logic and render every frame. It continuously encodes this rendered output into a media stream, passing through a lightweight web service stack. Users can then view that broadcast stream in standard web browsers running on other computers and mobile devices. Zhang et al. [10] implemented an educational laboratory platform. Models in this platform are divided into two categories: background and interactive objects. On the cloud side, the background is rendered, encoded into an image with H.264, and then pushed to the client via the real-time message protocol (RTMP). Besides, lightweight rendering is used for interactive models. Finally, the rendering results are combined at the end terminal. Lightweight rendering leverages a terminal-oriented adaptive algorithm to transfer rendered models based on computing power and network latency. The authors also propose an improved 3D-warping and hole-filling algorithm that significantly improves image quality when the user's viewpoint changes. Viitanen et al. [11] presented a low latency edge rendering scheme for remote VR gaming. The proposal aims to reduce the energy and computation burden of the end user devices by performing game rendering on the server side. The rendered views are sent to the user as encoded high efficiency video coding (HEVC) video frames. The system creates 360 equirectangular projection (ERP) video from the rendered views, divides it into 128 pixel-wide image slices, and uses the user's head orientation data to limit the transmission of the image slices to that of the field of view (FoV). The selected slices are encoded in real time and in around half bit rate over the case where all slices are encoded.

As indicated above, existing VR/panorama video systems benefit greatly from remote/cloud rendering technology. However, video fusion systems usually need to fuse multiple video streaming and large-scale 3D scenes, so they are quite demanding for computing power, limiting their application scenarios. Remote/cloud rendering is just in time to eliminate this obstacle. We present a web-based MR video fusion system with remote rendering, introducing remote rendering to the video fusion system and using a web browser as the front end, which is quite user-friendly.

III. System Architecture

This section describes the key components of our web-based MR video fusion system and the data flow and interfaces between these components.

An overview of the system architecture is shown in Fig. 1. Our system is developed based on the B/S model. The server-side implementation consists of a video connector, a render server, and a generic WebRTC-based cloud rendering library consisting of a command receiver, a video encoder, and a transmitter.

The render server can read the 3D scene files from the database and multiple videos streaming from the video connector. Then render server will fuse videos into the 3D scene and then render original pixel-data images into shared memory where other components could access these images (More details will be illustrated in Section IV). When users manipulate the scene via their browsers, the command receiver component will publish a command that users trigger, such as

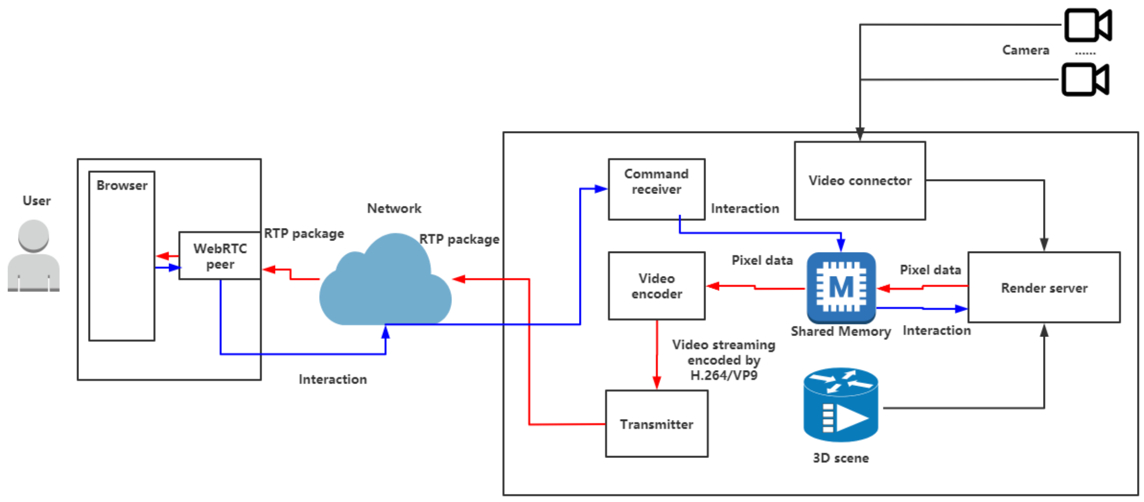

Fig. 1. System architecture.

changing viewport or zoom in/out a particular area. Because the render server subscribes to a Redis (an open source, in-memory data structure store) queue, those commands will be pushed to the render server and triggered to re-render and generate a new image to the shared memory.

The video encoder is a key component within the cloud rendering library. It accesses shared memory, decoding the original pixel data generated by the render server utilizing H.264/VP9 [12], which could be customized. The encoder will utilize hardware acceleration technology if it detects supported GPU. For example, Nvidia encoder (NVENC) is a feature in Nvidia graphics cards that performs video encoding, offloading this compute-intensive task from the CPU to a dedicated part of the GPU. It was introduced with the Kepler-based GeForce 600 series in 2012. The encoded stream will be sent to the transmission component. It will be packaged to the real-time transport protocol (RTP) package and sent to WebRTC peer resident in the user's browser.

The transmitter component is responsible for establishing a connection with web peers and delivering media. Because of the network address translation (NAT) and firewalls, it may have problems to establish a peer-to-peer connection. These constraints were solved in the transmitter component utilizing the session traversal utilities for NAT (STUN) server, traversal using relays around NAT (TURN) server, and interactive connectivity establishment (ICE) protocol. Firstly, ICE tries to connect peers directly with UDP. In this process, the STUN server is helping the peer behind NAT to find its public IP address and port. When UDP does not pass through, the transmission control protocol (TCP) is utilized. The TURN server will be utilized finally if TCP still does not work. When the connection is established, the transmitter component packages media data into RTP and send them to the web peer. A Google congestion control(GCC) algorithm working with RTP/RTCP protocols is utilized in the transmitter component. It is based on the idea of using delay gradient to infer congestion.

Interaction is an important feature of the MR system. For our proposed system, user interactions from the web will be sent to the command receiver, and parameters will be transformed and transmitted to the render server. Thus, the render server acts as a thrift server and the command receiver as a thrift client in this process. More details will be discussed in Section V.

IV. MR Video Fusion Rendering

The render server is responsible for fusing videos and 3D scenes, which is the key feature of our system.

As illustrated in Fig. 2, the input of the render server are 3D models and video images, and the output of the render server is fusion result in a frame buffer. The whole process is described as follows. (1) Transform the position and posture of the camera to the position and posture of the 3D environment. (2) Calculate the model-view matrix and projection matrix based on the output of step (1). (3) Calculate the frustum structure in the 3D environment of the camera and take the far clipping plane as a far plane to supplement scene structure. (4) Select visible model collections based on the frustum to speed up the fusion process. (5) Render depth information of camera viewport using the model-view matrix and projection matrix, and then perform occlusion detection toward the model's vertex utilizing depth information. We only perform video fusion for the not occluded parts and keep the occlusion part of models their original texture. (6) Do the fragment texturing and coloring (FTC) operation in graphic cards and put frames into a frame buffer.

A. *Visible model selection*

Fusing videos requires traversing all models of the scene. However, the camera can only observe a subset of the models Therefore, choosing visible models from the view point of the camera could speed up the traversal process. We calculate

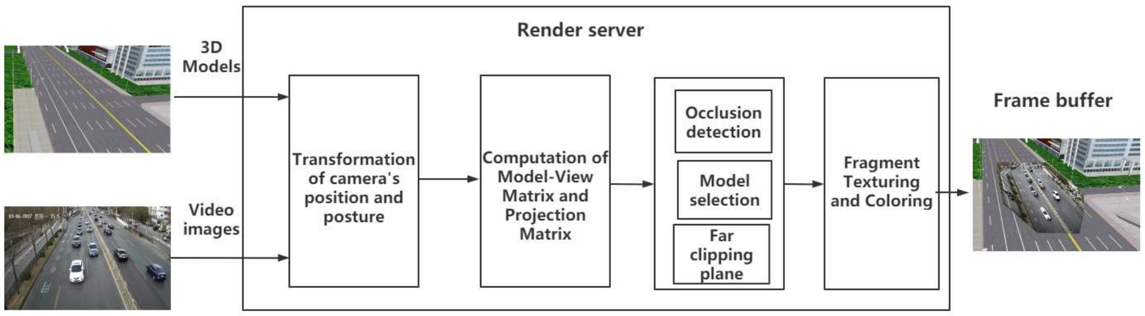
Render server
3D Models
Video images
Transformation of camera's position and posture
Computation of Model-View Matrix and Projection Matrix
Occlusion detection
Model selection
Far clipping plane
Fragment Texturing and Coloring
Frame buffer

Fig. 2. Rendering process of video fusion.

the frustum bounding box of the camera, and perform an intersection operation for the camera's bounding box with each model. Only the models whose bounding box intersects with the camera frustum will be taken into account.

The models we are going to fuse consist of construction models and a terrain model. For the construction models, we only fuse video images with constructions that intersect with the camera frustum; for the terrain, we slice it based on prior knowledge, demonstrated in Fig. 3.

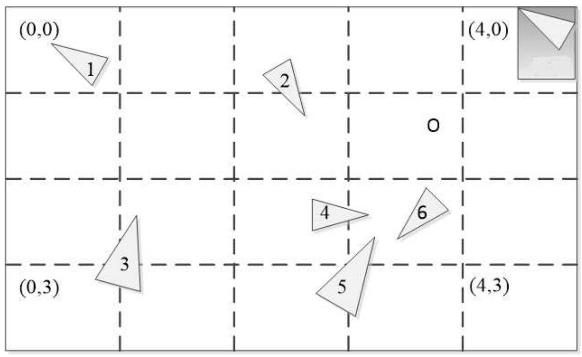

Fig. 3. Terrain slices.

The whole terrain is sliced into 20 pieces, and all pieces are numbered. For example, the upper left slice is numbered as (0,0), while the bottom right slice is numbered as (4,3). Six cameras are deployed in the whole terrain, and each camera can only observe a subset of the terrain pieces. The actual terrain slices guarantee that the frustum of a certain camera should cover four slices at most. During the fusion process, the proposed method will filter nearby slices based on the camera's position and then intersect those bounding boxes of slices with the camera's frustum, finally fuse the intersected models with the video images.

B. Optimization of fusion result

We optimize our fusion result from the following aspects: (1) avoiding occluded parts to be fused with video images; (2) projecting video image without a corresponding scene to the far plane; (3) tailoring the outside structure of buildings to observe indoor fusion result.

As illustrated in Fig. 4, if observed from the camera's viewport O, area ABCD of model M is occluded by model N, so a patch E inside area ABCD is invisible. However, if observed from the user's viewport, E is not occluded and visible to the user. If we don't perform occlusion detection, a video image patch will be wrongly projected at the occluded part. When the user observes point E from D, the observed wrong picture will cause cognitive confusion.

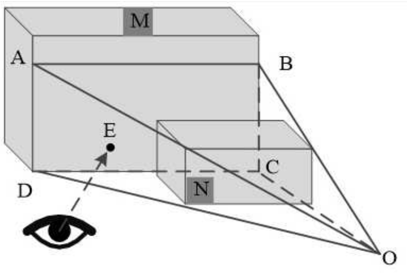

Fig. 4. Occlusion between models.

To solve this problem, detecting occluded parts based on depth values should be performed to correct fusion results. Our system utilizes the render-to-texture (RTT) technology to render and save depth information of visible scenes based on the model-view matrix and projection matrix. However, the depth value's accuracy error exists because of the scale differences for the camera's visible space and the whole scene's space. To reduce this kind of accuracy error, the depth of scene can be projected to the RGB color space, and will be transformed back to depth if a depth judgment is needed. Fig. 5 shows a depth texture picture from one of our cameras.

When performing FTC, we calculate the relative depth of fragment of the current viewport. If the relative depth of fragment is less than the corresponding depth in texture, our system treats this fragment as visible for the camera and then assigns pixels of the image to this fragment. Otherwise, the fragment will keep its original color. When the scene varies or the camera's parameters change, our system willre-render depth to get the correct depth texture.

Normally, when modeling for an outdoor scene, we don't model the sky, trees, road signs, street lamps, etc., so corresponding objects in the video are missing in the 3D scene.

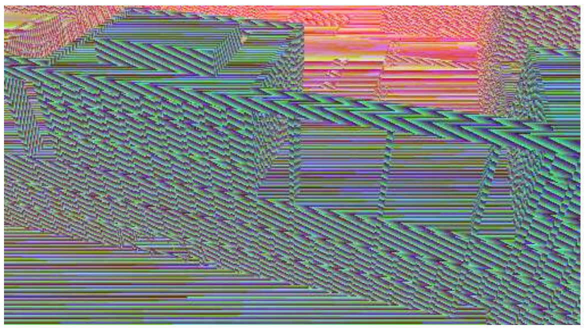

Fig. 5. Depth texture of a visible scene.

Because of this, those contents will not be projected to models during the fusion process. As a result, the user will be aware of this situation of object missing in the image. In our system, a far plane will be set to improve this missing issue to some extent. Each camera generates the stereoscopic plane based on the far clipping plane of the frustum and then adds it to the far plane node. The system processes vertexes in a far plane as follows. (1) If the relative depth of vertex is less than the corresponding depth in depth texture, it implies that vertex in video image doesn't have a corresponding model in the scene. In this case, the video image will be projected to the corresponding position in the far plane. (2) If the relative depth of vertex is more than the corresponding depth in depth texture, it implies that models in the scene occlude vertex. Users should not observe the corresponding picture, so this vertex is set to be transparent.The comparison for whether adding a par plane is shown in Fig. 6. The left part of Fig. 6 shows the fusion effect before adding the far plane and the right part shows the fusion effect after adding the far plane. The right picture is quite intact compared to the left one, and the visual effect improves to some extent.

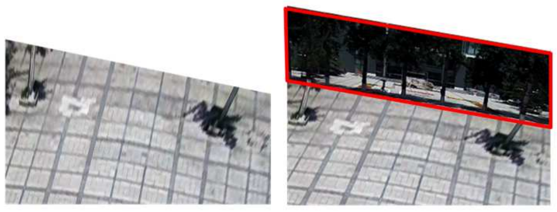

Fig. 6. Before (left) and after (right) adding a far plane.

In another scenario when users want to observe indoor fusion results, the indoor models are occluded by the outside model because the indoor models are inside buildings. To make the indoor models visible, the system should tailor part of the outside model to observe the indoor fusion results. The tailoring effect is shown in Fig. 7, where the left is the fusion result before tailoring the building's outside part, while the right is the fusion result after tailoring. Apparently, the view after the tailoring is better because it would not be occluded by the outside part so that users could observe the fusion result of the indoor scene directly.

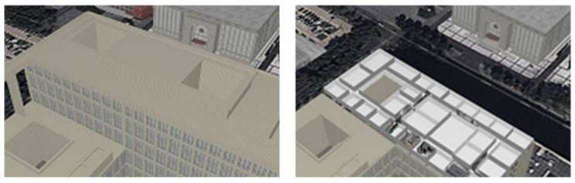

Fig. 7. Before (left) and after (right) tailoring a building's outside part.

V. WebRTC-based Video Streaming with Interaction

The output is a video image from the render server. We should find a way to transmit those images to the user's browser, and at the same time, the user should interact with 3D models. Action on the client side should be got immediate feedback. To achieve this, as illustrated in Fig. 8, we develop a WebRTC-based cloud rendering library. Images will be transformed from RGB to YUV format and encoded using specified standards. H.265 outperforms H.264 in several benchmarks, but due to lack of support by mainstream web browsers, we select H.264 as one of our options. VP9 introduced by Google has a higher compression ratio than H.264 for the same video quality, apparently lower bandwidth usage, so it's another encoding standard for our system.

Encoder component consists of a hardware encoder based on the GPU and a software encoder like x264/libvpx [13]. A user could change one of the encoders to encode the image rendered by the render server. The principle of choosing an encoder is described as follows. If using codecs that the current hardware encoder can't support (like VP9), the encoder module will utilize a software encoder (libvpx); If a server has a killer CPU (like an AMD Ryzen 9 or Intel i9), the encoder module should utilize x264 with CPU, because x264 with powerful CPU could get better image quality than hardware-accelerated encoders; Otherwise, the encoder module utilizes hardware-accelerated encoders to encode images.

In the next step, encoded images go into the transmitter component, where they are packaged into the real-time protocol (RTP) [14] message and sent to the user's browser via networks.

Streaming keyboard and mouse events to the render server makes our system interactive. Keyboard and mouse events are captured using their native JavaScript event handlers. Next, they are converted to binary commands and sent over a data channel where the events are decoded and passed to the command receiver. For example, when users press a key, the browser generates a keydown-type event, and when users release a key, a keyup event is generated. The events contain the localized key character, such as a, s, d, or f, and a JavaScript key code such as KeyA, KeyS, KeyD, or KeyF. Mouse events are captured by JavaScript codes as absolute positions or as relative motions. By calculating the browser client viewport, video scaling factor, and page offsets, our system can send the translated mouse events to the command receiver.

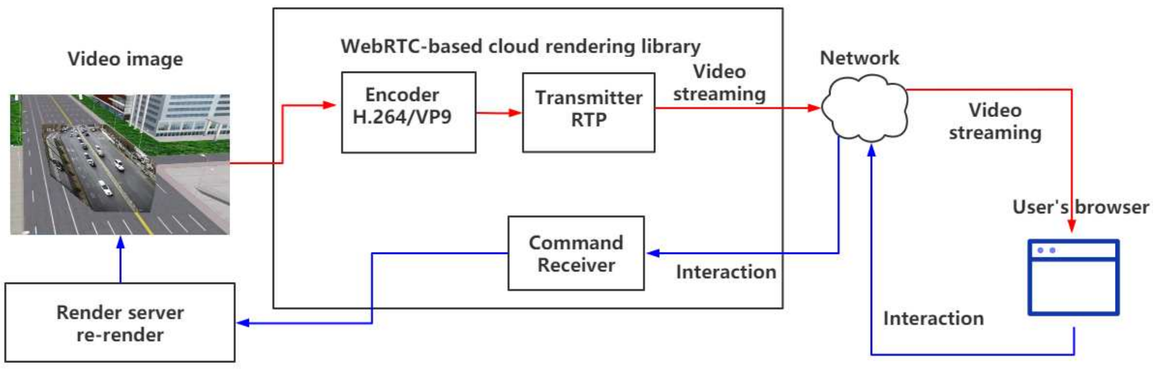

Fig. 8. The workflow of the WebRTC-based cloud rendering library.

When users perform an interaction generated by keyboard or mouse in the browser like zoom in/out, change viewport, etc., as illustrated in Fig. 9, our system captures this interaction and then delivers it to the data channel of WebRTC. The data channel is an embedded mechanism of WebRTC to exchange data between peers when data arrive the command receiver component, where it will be transformed into a thrift event.

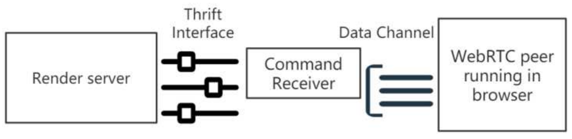

Fig. 9. The interface between components.

The render server has an interface of thrift, so the thrift event generated by the command receiver will be sent to the render server, and then a re-render will be triggered. Finally, a new image will be transmitted to the user browser.

We develop a cloud rendering library based on WebRTC. The original native WebRTC could only get video sources from a webcam or captured from the desktop. We develop a new source connector to access the pixel image data from shared memory where three-party applications can generate video frames.

In our case, the render server fuses multiple videos with 3D scene data and then renders the image to the shared memory area as a publisher. The connector fetches pixel data from this shared memory as a subscriber. This connector is designed to be general-purpose, suitable for other applications which want to transmit via WebRTC, and to be integrated seamlessly with other WebRTC implementations utilizing this connector.

The render server has a model which could encode the rendered image with different codecs and push them via lots of channels like streaming media (RTSP [15] and RTMP), shared memory, UDT (UDP-based data transfer), etc. Then, we develop a standalone video pushing service to connect those streaming data, as illustrated in Fig. 10. It is built based on a Go programming language implementation of native WebRTC and can also read video files from local disk and send them to browser peers.

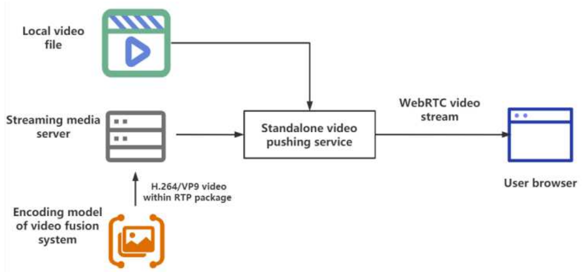
Local video file
Streaming media server
Standalone video pushing service
WebRTC video stream
User browser
H.264/VP9 video within RTP package
Encoding model of video fusion system

Fig. 10. Cloud rendering video fusion service that connects streaming data from both the streaming media server and local files.

VI. Experiments

We conduct the comparison of our proposed system and the Pixel Streaming of Unreal Engine 4 (UE4) as UE4 also provides WebRTC-based remote rendering. We deploy the pixel streaming plugins, signaling server, and web server of UE4 on one laptop, and use a web browser installed in another laptop to access pixel streaming services.

A. Experiment environment

The proposed experimental setup is depicted in Fig. 11. It comprises two gaming-grade laptops equipped with Nvidia RTX 2080 Super with Max-Q (NVENC inside) and Intel i7 2.30 GHz CPU. The laptops are connected using 1000 M wireless router for low latency communication.

The experimental scene is near the southeast door of Beihang University, Beijing, China, as shown in Fig. 12. Video streaming captured from 3 cameras is introduced and fused into a 3D scene to observe the real traffic of the road.

B. Multi-client concurrency access test

As illustrated in Fig. 11, we first use one computer as one client, then two computers, and finally three computers, to access the video fusion system. Then, the MR video fusion system is deployed on the server and fuses three video

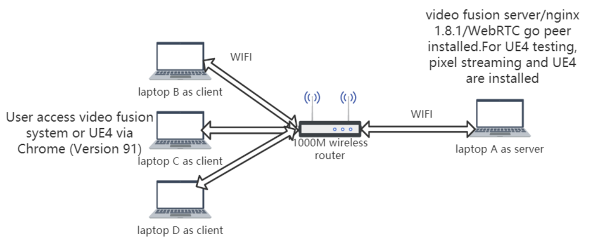
video fusion server/nginx 1.8.1/WebRTC go peer installed.For UE4 testing, pixel streaming and UE4 are installed
WIFI
laptop B as client
User access video fusion system or UE4 via Chrome (Version 91)
laptop C as client
1000M wireless router
WIFI
laptop A as server
laptop D as client

Fig. 11. Experiment setup.

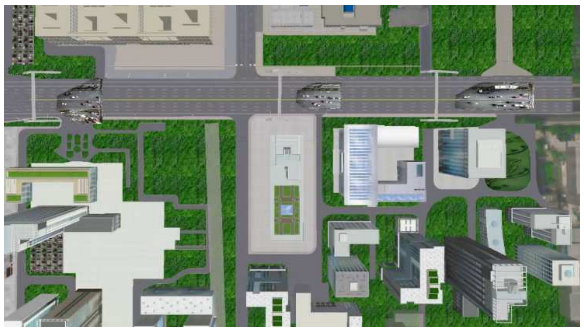

Fig. 12. Traffic video fusion in a 3D scene.

streaming in a 3D scene. Finally, we record the workload (CPU/bandwidth/GPU/memory usage) of the server.

As illustrated in Figs. 13–16, bandwidth and GPU usage increase linearly according to the number of clients. Still, CPU and memory usage do not increase linearly when the number of concurrent clients grows. So if we expect more scalability to our video fusion system, bandwidth, and GPU bottleneck should be overcome.

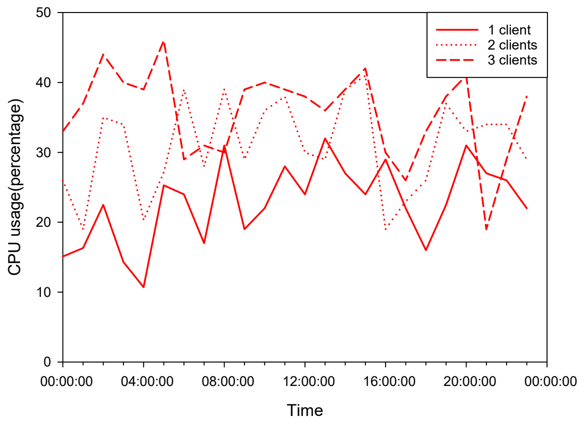

Fig. 13. CPU usage for the server when multiple clients access concurrently.

C. Contrast Experiment results

We compare two systems on several metrics like FPS, freeze frame count, and bandwidth usage.

The left picture of Fig. 17 shows the FPS of UE4, and the right one is that of our proposed system. Most of this time,

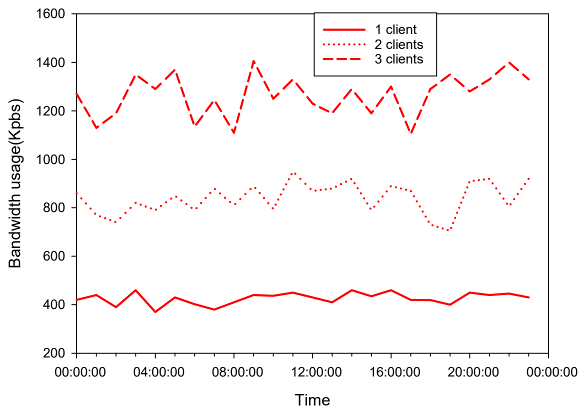

Fig. 14. Bandwidth usage for the server when multiple clients access concurrently.

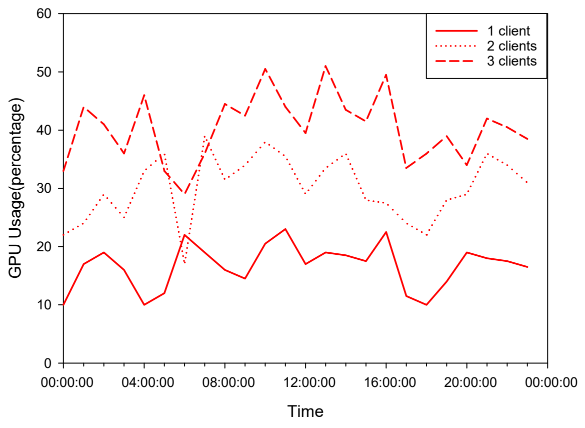

Fig. 15. GPU usage for the server when multiple clients access concurrently.

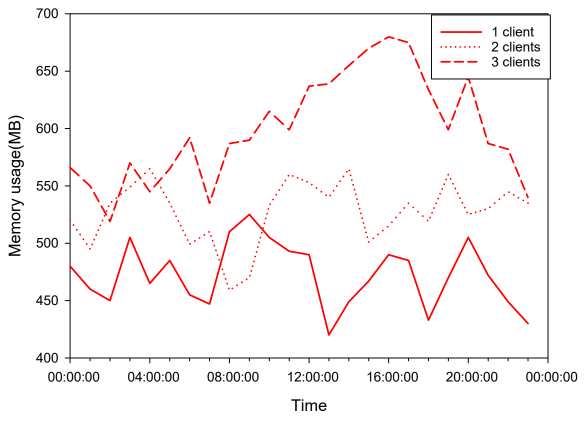

Fig. 16. Memory usage for the server when multiple clients access concurrently.

the two systems could keep around 60 FPS and only have one or two sudden drops during the one-hour testing.

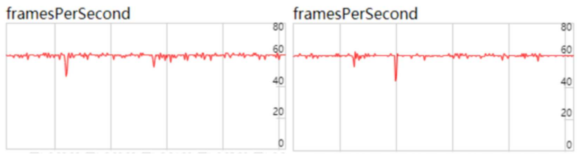

Fig. 17. FPS comparison of UE4 (left) and our proposed system (right).

We perform bandwidth usage testing between UE4 and our proposed system for 24 hours. The results are shown in Fig. 18.

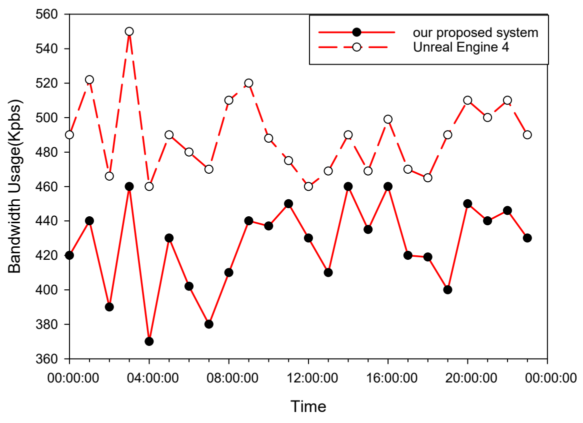

Fig. 18. Bandwidth usage.

Our proposed system could reduce roughly 15% bandwidth usage compared to UE4. This improvement is likely because VP9 utilized in our system has a better compression ratio than H.264 that UE4 uses. Next, we compare freeze frame count of the two systems. This metric means to count the total number of video freezes experienced by the receiver. It is a freeze if frame duration, which is a time interval between two consecutively rendered frames, equals or exceeds Max(3 * avg_frame_duration_ms, avg_frame_duration_ms + 150), where avg_frame_duration_ms is the linear average of durations of the last 30 rendered frames. As illustrated in Fig. 19, the freeze frame count is almost the same for the two systems during the 8-hour testing.

VII. Conclusion and Future Work

In this paper, we present a web-based video fusion framework with remote rendering. Our framework consists of a render server that could project video images as texture to 3D models, a low latency streaming WebRTC protocol with interaction, a low latency encoder, a command receiver, and a transmitter. This framework could overcome the limitation of PC and mobile device, so that a video fusion system could be applied on a web browser of a low-profile PC or mobile device. Based on the developed video fusion framework, our future work include: (1) We will develop an effective 6-DoF prediction technique to reduce latency. (2) The remote rendering method proposed in this paper is universal, not only applied to the MR video fusion system, but can also be adapt to real-time Augmented Reality (AR) if the network condition is fine so that the interaction latency is tolerable. Therefore, we will attempt an AR scenario utilizing the proposed method. (3) We will customize the data channel in WebRTC by utilizing the quick UDP Internet connection (QUIC) protocol [16] to improve network performance.

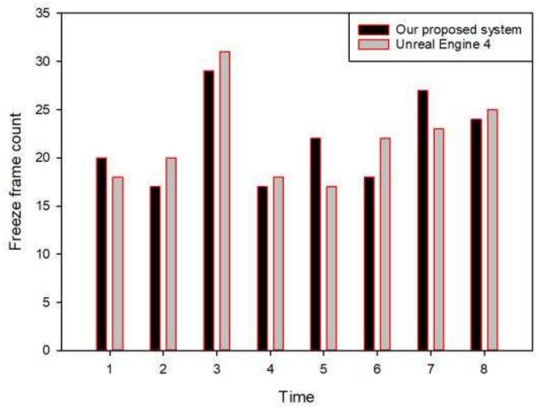

Fig. 19. Freeze frame count.

References

[1] C. Vogt, M. J. Werner, and T. C. Schmidt, "Leveraging webrtc for p2p content distribution in web browsers," Proceedings of 2013 21st IEEE International Conference on Network Protocols (ICNP), pp. 1–2, 2013.
[2] F. Rhinow, P. P. Veloso, C. Puyelo, S. Barrett, and E. O. Nuallain, "P2p live video streaming in webrtc," Proceedings of 2014 World Congress on Computer Applications and Information Systems (WCCAIS), pp. 1–6, 2014.
[3] N. Tindall and A. Harwood, "Peer-to-peer between browsers: cyclon protocol over webrtc," Proceedings of 2015 IEEE International Conference on Peer-to-Peer Computing (P2P), pp. 1–5, 2015.
[4] S. Moezzi, A. Katkere, D. Y. Kuramura, and R. Jain, "Reality modeling and visualization from multiple video sequences," Comput. Graph. Appl., vol. 16, pp. 58–63, 1996.
[5] U. Neumann, S. You, J. Hu, and B. Jiang, "Augmented virtual environments (AVE): dynamic fusion of imagery and 3D models," Proceedings of the IEEE Virtual Reality 2003, pp. 61–70, 2003.
[6] L. Wang, S. You, and U. Neumann, "Single view camera calibration for augmented virtual environments," Proceedings of 2017 IEEE Virtual Reality Conference, pp. 255–258, 2007.
[7] H. S. Sawhney, A. Arpa, R. Kumar, and S. Samarasekera, "Video flashlights: real time rendering of multiple videos for immersive model visualization," Proceedings of the 13th Eurographics Workshop on Rendering, pp. 157–168, 2002.
[8] Y. Zhou, M. Cao, J. You, M. Meng, Y. Wang, and Z. Zhou, "MR video fusion: interactive 3D modeling and stitching on wide-baseline videos," Proceedings of the 24th ACM Symposium on Virtual Reality Software and Technology, pp. 1–11, 2018.
[9] Y. Zhou, P. Liu, J. You, and Z. Zhou, "Streaming location-based panorama videos into augmented virtual environment," Proceedings of 2014 International Conference on Virtual Reality and Visualization (ICVRV), 2014.
[10] H. Zhang, J. Zhang, X. Yin, K. Zhou, and Z. Pan, "Cloud-to-end rendering and storage management for virtual reality in experimental education," Virtual Reality Intell. Hardware, vol. 2, pp. 368–380, 2020.
[11] M. Viitanen, J. Vanne, T. D. Hamalainen, and A. Kulmala, "Latency edge rendering scheme for interactive 360 degree virtual reality gaming," Proceedings of 2018 IEEE 38th International Conference on Distributed Computing Systems, 2018.

[12] D. Grois, D. Marpe, A. Mulayoff, B. Itzhaky, and O. Hadar, “Performance comparison of H.265/MPEG-HEVC, VP9, and H. 264/MPEG-AVC encoders,” Proceedings of 2013 Picture Coding Symposium (PCS), 2013.

[13] L. Guo, J. De Cock, and A. Aaron, “Compression performance comparison of x264, x265, libvpx and aomenc for on-demand adaptive streaming applications,” Proceedings of 2018 Picture Coding Symposium (PCS), 2018.

[14] S. Biaz, R. O. Chapman, and J. P. Williams, “RTP and TCP based MIDI over IP protocols,” Proceedings of the 43rd Annual Southeast Regional Conference, vol. 2, pp. 112–117, 2005.

[15] Real Time Streaming Protocol (RTSP), https://datatracker.ietf.org/doc/html/rfc2326, last accessed 01.07.2021.

[16] A. Langley, A. Riddoch, A. Wilk, A. Vicente, C. Krasic, D. Zhang, F. Yang, F. Kouranov, I. Swett, J. Iyengar, J. Bailey, J. Dorfman, J. Roskind, J. Kulik, P. Westin, R. Tenneti, R. Shade, R. Hamilton, V. Vasiliev, W.-T. Chang, and Z. Shi, “The QUIC transport protocol: design and internet-scale deployment,” Proceedings of the Conference of the ACM Special Interest Group on Data Communication, pp. 183–196, 2017.