INTERACTIVE VOLUMETRIC VIDEO FROM THE CLOUD

S. Gül[1], D. Podborski[1], A. Hilsmann[1], W. Morgenstern[1], P. Eisert[1,3], O. Schreer[1], T. Buchholz[2], T. Schierl[1], C. Hellge[1]

[1] Fraunhofer Heinrich Hertz Institute
Berlin, Germany

[2] Deutsche Telekom AG
Bonn, Germany

[3] Humboldt University of Berlin
Institute for Computer Science, Visual Computing
Berlin, Germany

ABSTRACT

Recent advances in volumetric capture technology have started to enable the creation of high-quality 3D video content for free-viewpoint rendering on VR and AR glasses. This allows highly immersive viewing experiences, which are currently limited to experiencing pre-recorded content. However, for an immersive experience, interaction with virtual humans plays an important role. In this paper, we address interactive applications of free-viewpoint volumetric video and present a new framework for the creation of interactive volumetric video content of humans as well as real-time rendering and streaming. Re-animation and alteration of an actor's performance captured in a volumetric studio becomes possible through semantic enrichment of the captured data and new hybrid geometry- and video-based animation methods that allow a direct animation of the high-quality data itself instead of creating an animatable model that resembles the captured data. As interactive content presents new challenges to real-time rendering, we have developed a cloud-based rendering system that reduces the high processing requirements on the client side.

1 INTRODUCTION

Volumetric videos capture 3D spaces with a high degree of accuracy and enable services with six degrees of freedom (6DoF) that give the viewers the freedom to change both their position and orientation in virtual space. 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, health and e-commerce [Schre19a]. While Volumetric Video enables highly photorealistic free viewpoint video, it is usually limited to playback of recorded scenes. Alteration and animation of the content per se is not possible. If interactive and animatable content is envisioned, the classical approach is to build upon traditional hand crafted Computer Graphics models, which lack photorealism. Recent works have proposed to combine the photorealism of Volumetric Video data with the flexibility of Computer Graphics models in order to make Volumetric Video animatable or create new animations from

Volumetric Video data [Hils20]. In this approach, re-animation and alteration of an actor's performance captured in a volumetric studio becomes possible through semantic enrichment of the captured data and new hybrid geometry- and video-based animation methods that allow a direct animation of the high-quality data itself instead of creating an animatable CG model that resembles the captured data.

Despite the significant increase in computing power of mobile devices, rendering rich volumetric videos on such devices is still a very demanding task. The processing load is further increased by the presence of multiple volumetric objects in the scene and the new challenges presented by the interactivity. Particularly, interactivity requires changing the volumetric object according to the user input or position, which is especially challenging on mobile devices with low processing power. Another challenge is the lack of efficient hardware implementations for decoding of volumetric data (e.g. point clouds or meshes). Software decoding may drain the battery of mobile devices quickly and fail to meet the real-time rendering requirements.

One way to reduce the processing load on the client is to send a 2D view of the volumetric object rendered according to the viewer's position instead of sending the entire volumetric content. This technique is typically known as remote or interactive rendering [Shi15]. This is achieved by offloading the expensive rendering process to a powerful server and transmitting the rendered views over a network to a less powerful client device. Another advantage of this approach is a significant reduction of network bandwidth requirements because only a single 2D video is transmitted instead of the full 3D volumetric content. Additionally, the rendering server can be deployed within a cloud computing platform to provide flexible allocation of computational resources and scalability depending on potential changes in processing load.

Despite these advantages, one major drawback of cloud-based rendering is an increase in the end-to-end latency of the system, also known as the motion-to-photon (M2P) latency. Particularly, the added network latency and additional server-side processing (e.g. video encoding) cause an increase in M2P latency, which may significantly degrade the user experience and cause motion sickness [Alli01].

It is possible to compensate for the increased M2P latency through various optimizations. One promising technique is to move the volumetric content to an edge server geographically closer to the user to reduce the network latency. Edge computing has been gaining importance with the deployment of 5G networks and is considered to be one of the key technologies for interactive use cases that will be enabled by the 5G technology [Sat17]. Secondly, usage of real-time communication protocols such as WebRTC are also considered to be vital for ultra-low latency video streaming applications [Hol15]. The processing latency at the rendering server is another significant latency component. Therefore, using fast hardware-based video encoders is critical for reducing the encoding latency. Another way is to predict the future user pose at the remote server and send the corresponding rendered view to the client. Thereby, it is possible to drastically reduce the M2P latency, if the user pose is predicted for a prediction window equal to or larger than the M2P latency of the system. However, accuracy of the prediction algorithms is critical for achieving good results and mispredictions can potentially lead to degradations of user experience.

In this context, we have developed a cloud-based rendering system for interactive streaming of volumetric videos, which includes several of the described optimizations. Our system reduces the high processing requirements on the client side and significantly decreases the network bandwidth requirements. In this paper, we present the algorithms employed for the animation of volumetric video (Section 2) and describe the components of our low-latency streaming framework (Section 3).

2 ANIMATABLE VOLUMETRIC VIDEO

This section gives an overview of our animatable volumetric video technology. Going beyond the application of free-viewpoint volumetric video, we allow re-animation and alteration of an actor's performance through (i) the enrichment of the captured data with semantics and animation properties and (ii) applying hybrid geometry- and video-based animation methods. The idea is to allow direct animation of the high-quality volumetric video data itself instead of creating an animatable model that resembles the captured data. Thereby, we bring volumetric video to life and combine interactivity with photo-realism. Details of our approach can be found in [Hils20].

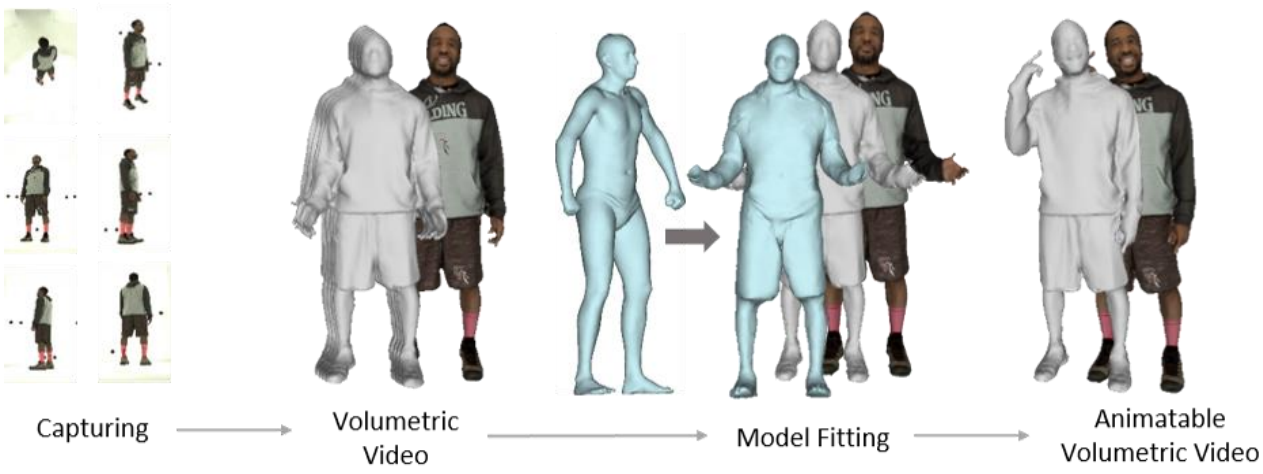

Figure 1 - Overview of our animatable volumetric video technology.

2.1 Making Volumetric Video Data Animatable

The creation of high-quality animatable volumetric video starts with the creation of high-quality free viewpoint video by capturing an actor's performance in a volumetric studio and computing a temporal sequence of 3D meshes. For the capturing and data processing steps, either high-quality professional capturing with sophisticated setups, e.g. [Schre19a, Coll15, Vol, Mic] as well as lighter and cheaper systems [Rob16, All19] can be applied, depending on the use case requirements. The reconstructed temporally inconsistent meshes are converted into spatio-temporally coherent mesh (sub-) sequences using template-based approaches in order to facilitate texturing and compression. In order to allow for topological changes during the sequence, we use a key-frame-based method to decompose the captured sequence into subsequences and register a number of key-frames to the captured data [Morg19]. The final volumetric video data can be inserted as volumetric video assets in virtual environments and viewed from arbitrary directions. In order to make the volumetric data animatable, we fit a parametric rigged human model [Angu05, Fech19a] to the captured data [Fech19b]. Thereby, we enrich the captured data with semantic pose and animation data taken from the parametric model, used to drive the animation of the captured volumetric

video data itself.

2.2 Hybrid Animation of Volumetric Video Data

The captured content contains real deformations and poses and we want to exploit this data for the generation of new performances as much as possible. For this purpose, we propose a hybrid example-based animation approach that exploits the captured data as much as possible and only minimally animates the captured data in order to fit the desired poses and actions. We treat the temporally consistent subsequences of the volumetric video data as essential basis sequences, containing motion and appearance examples. Given a desired target pose sequence, we retrieve close subsequences or frames based on the semantic enrichment of the volumetric video data. The retrieved subsequences or frames can then be concatenated and interpolated in order to form new animations, similar to surface motion graphs [Huang15]. The generated sequences are restricted to poses and movements already present in the captured data and might not perfectly fit the desired poses. Hence, after we have created a synthetic sequence from the original data that resembles the target sequence as closely as possible, we now animate and kinematically adapt the recomposed frames in order to fit the desired poses. The kinematic animation of the individual frames is facilitated through the body model fitted to each frame. For each mesh vertex, the location relative to the closest triangle of the template model is calculated, virtually glueing the mesh vertex to the template triangle with constant distance and orientation. This parameterization between each mesh frame and the model allows a direct animation of the volumetric video frame.

One application is interaction with virtual humans in AR or VR where the virtual human follows the user with his head when the user moves in the virtual scene in order to enhance the feeling of a true conversation with the virtual character (see Fig. 2).

Figure 2 - Animated virtual human moving his head to follow the user.

3 CLOUD-BASED VOLUMETRIC VIDEO STREAMING

This section presents the system architecture of our cloud-based volumetric video streaming system and describes its different components. A simplified version of this architecture is shown in Fig. 3.

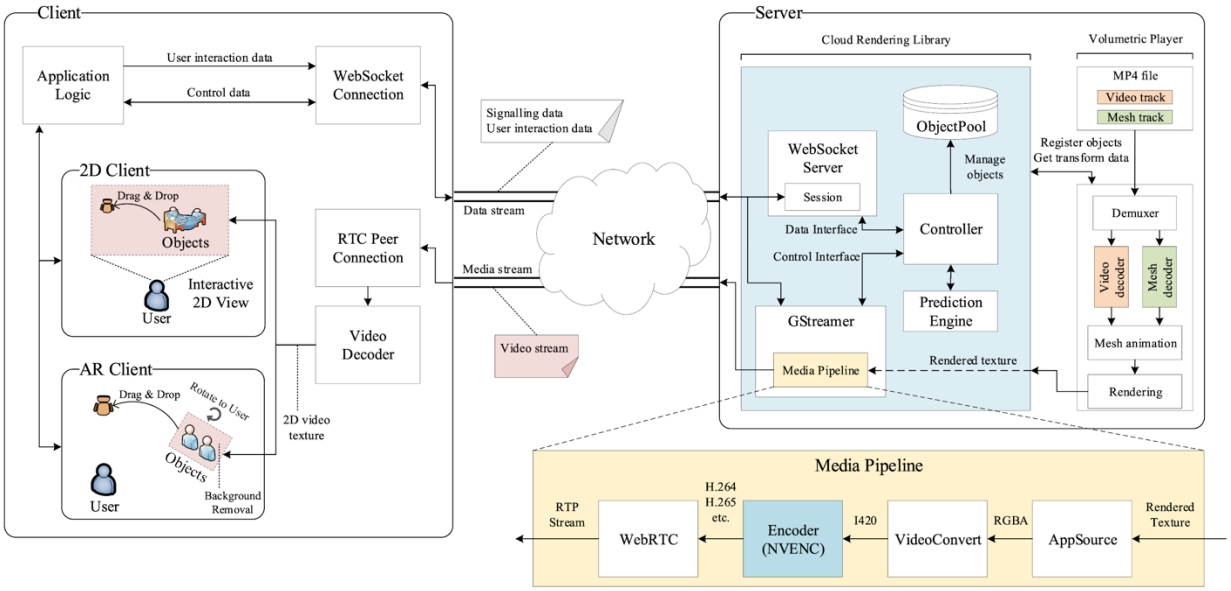

Figure 3 - Overview of cloud-based volumetric streaming framework.

3.1 Volumetric asset storage format

Different components of our volumetric assets are stored in a single MP4 file to facilitate integration as a dedicated plugin into the game engines such as Unity. Particularly, we encode the texture atlas using video compression (H.264/AVC), and compress the mesh data using Google Draco [Goo] which implements the well-established mesh compression algorithm Edgebreaker [Ross99]. The compressed mesh and texture data are multiplexed into different tracks of an MP4 file ready for storage and transmission [Schre19b].

3.2 Server architecture

The server-side implementation is composed of two main parts: a volumetric video player and a cross-platform cloud rendering library that can be integrated into different applications. We describe each block further in detail.

The **volumetric video player** is implemented in Unity using 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, all required game objects are registered by the player. These can include e.g. a volumetric object stored as an MP4 file or a virtual camera. The registered game objects can then be controlled by the cloud rendering library and/or the client application. 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. The volumetric mesh is altered based on user input, e.g. for the eye contact application described above, the head is turned based on the user position as described in Section 2.2. Then, each mesh is synchronized with the corresponding texture and rendered to the scene. Subsequently, 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. While rendering the scene, the player

concurrently asks the library for the latest positions of the previously registered game objects and updates the rendered view accordingly.

We created a cross-platform **cloud rendering library** written in C++ that can be integrated into a variety of applications. The library is implemented as a native Unity plugin for the present application and contains various processing and communications blocks and interfaces. In the following, we describe the main modules implemented in the library, each running on a different thread to achieve high performance.

Signaling and control data between the server and client is exchanged via a **WebSocket Serve**r. Signaling data includes Session Description Protocol (SDP) and Interactive Connectivity Establishment (ICE) messages necessary for establishing the WebRTC connection. The WebSocket connection is also used to transmit the scene description metadata as well as control data to modify the position of a registered game object or camera.

Media processing is performed using the **GStreamer** media framework [Gst]. GStreamer is a very flexible framework that allows chaining various media elements together to create complex media *pipelines*. Our Gstreamer pipeline takes the rendered texture from Unity as input, compresses it as a video stream, and transmits the video stream to the client over a WebRTC connection. Since encoder latency is a significant contributor to the overall M2P latency, we evaluated the encoding performances of different video encoders for a careful selection.
We experimented with different encoders such as the software-based encoders x264, x265, and Intel SVT-HEVC as well as the GPU-based encoder from Nvidia (NVENC) [Nve]. Consequently, we decided to use **NVENC** in our media pipeline due to its significantly higher encoding speed at a comparable picture quality with respect to other tested encoders. We use NVENC to compress the texture using **H.264/AVC** (high profile, level 3.1, IPPP.. GOP structure, no B frames). However, our system does not depend on a specific codec and allows using different H.264/AVC, H.265/HEVC, or in the future VVC encoders. After compression, the resulting compressed bitstream is packaged into RTP packets and sent to the client using WebRTC. WebRTC was chosen as the delivery method since it allows us to achieve an ultra-low latency while using the P2P connection between the client and server. In addition, WebRTC is already widely adopted by different web browsers allowing our system to support several different platforms. For our volumetric sequence “Josh”, the bitrate of the encoded bitstream (at a resolution of 1280x720) 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. Further details on our media pipeline and an evaluation of the encoder performance can be found in [Gül20a].

The **Controller** implements the application logic and manages the other modules depending on the application state. Particularly, it retrieves the object states from the **ObjectPool** (a logical structure that maintains the IDs and positions of all registered objects) and sends them to the client as a JSON file in order to inform the client about the object states. Depending on the predictions from the Prediction Engine, the Controller updates the positions of the virtual objects such that the rendering engine creates a scene corresponding to the predicted positions. The Controller is also responsible for initializing/closing the media pipeline when a new client joins/disconnects.

The **Prediction Engine** attempts to predict the head movement of the user in 6DoF space for a given prediction interval. By predicting the user's future pose and rendering the matching frames in advance, we aim to reduce the effective M2P-latency and thus compensate for the added network latency due to cloud-based rendering. Currently, we use an autoregression model for prediction; however, the module allows deployment of different kinds of algorithms that can bring higher prediction accuracy and/or run faster, e.g. efficient recursive algorithms like the Kalman filter. A latency analysis of our system and details of our initial prediction technique are described in [Gül20a].

3.3 Client architecture

The client-side architecture is depicted on the left side of Fig. 3. It consists of various connection interfaces, a video decoder, our application logic, and a client application.

Before the streaming session starts, the client establishes a WebSocket 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 which can later be updated by the client. After receiving the scene description, the client replicates the scene and initiates a peer-to-peer (P2P) WebRTC connection to the server. The server and client carry out the WebRTC negotiation process by sending SDP and ICE data over the established WebSocket connection. Finally, the P2P connection is established, and the client starts receiving a video stream corresponding to the current view of the volumetric video. At the same time, the client can use the WebSocket connection (or optionally the RTCPeerConnection) for sending control data to the server and modify the properties of the scene. For example, the client may change its position in the 6DoF space, or it may apply transformations (e.g. rotate, scale) to any volumetric object in the scene.

We have implemented both a web player in JavaScript and a native application for the HoloLens, the untethered AR headset from Microsoft. While our web application targets VR, our HoloLens application is implemented for AR use cases. In the HoloLens application, we perform further processing to remove the background of the video texture before rendering the texture onto the AR display. In general, the client-side architecture remains the same for both VR and AR use cases, and the most complex client-side module is the video decoder, which is implemented for H.264/AVC in hardware in most devices. Thus, the complexity of our system is concentrated largely in our cloud-based rendering server. A demonstration of our system with the web browser and HoloLens client implementations is presented in [Gül20b].

3.4 Motion-to-photon latency measurement

Characterizing the M2P latency is important for assessing the success of our optimizations towards creating a low-latency streaming framework. Therefore, we developed a framework to measure the latency of our system.

Using our cloud rendering library described in the previous section, we implemented a server-side console application which sends predefined textures (known by the client) depending on the received control data from the client. These textures consist of simple vertical bars with different colors. For example, if the client instructs the server application to move the main camera to the position P1, the server pushes the texture F1 into the media pipeline. Similarly, another camera position P2 results in the texture F2.

On the client side, we implemented a web-based application that connects to the server application and renders the received video stream to a canvas. Since the client knows exactly how those textures look like, it can evaluate the incoming video stream and determine when the requested texture was rendered on the screen. As soon as the client application sends P1 to the server, it starts the timer and checks the canvas for F1 at every web browser window *repaint* event, which matches the refresh rate of the display according to the W3C recommendation [Rob15]. As soon as the texture F1 is detected, the client stops the timer and computes the M2P latency.

Since we use the second smallest instance type of Amazon EC2 (t2.micro), we set the size of each video frame to 512x512 pixels. Once the connection is established, the user can start the session by defining the number of independent measurements. In our setup, we run the server application on an Amazon EC2 instance in Frankfurt, and the client application in a web browser in Berlin. The web browser is connected to the Internet over WiFi.

We encode the stream using x264 configured with ultrafast preset and zerolatency tuning with an encoding speed of 80 fps. In order to get statistically accurate results, we set the client to perform 100 latency measurements and calculated the average, minimum and maximum M2P latency. Fig. 4 shows the measured values over 100 cycles. According to our results, the M2P latency of our system varies between 41 ms and 63 ms. The measured average latency is 58 ms.

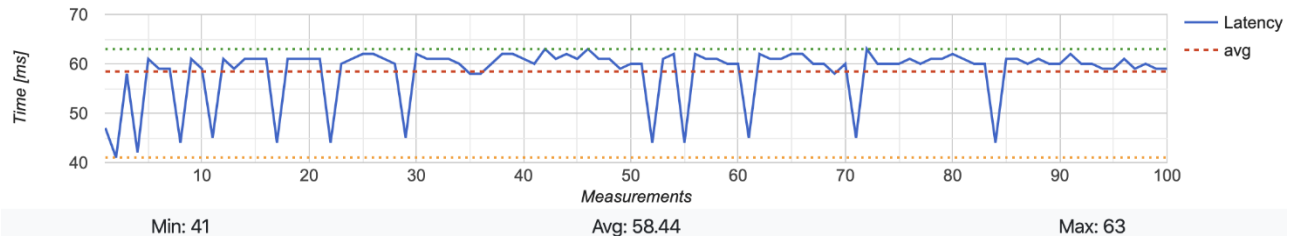

Figure 4 - Results of our motion-to-photon latency measurements.

3.5 Deployment

We deployed our server-side application on two different servers: a 5G edge server provided by Deutsche Telekom and an Amazon EC2 instance located in Frankfurt. For the Amazon EC2 instance in Frankfurt, we were able to measure an average M2P latency of 58 ms using our web browser client connected to WiFi in Berlin.

4 SUMMARY

We presented a new framework for the creation, animation, rendering and streaming of animatable volumetric content. Re-animation and alteration of an actor's performance captured in a volumetric studio becomes possible through semantic enrichment of the captured data and new hybrid geometry- and video-based animation methods that allow a direct animation of the high-quality data.

Furthermore, we presented a cloud-based streaming framework that offloads the volumetric video processing to a powerful cloud/edge server and thus alleviates the challenges brought

by interactive volumetric video in terms of real-time processing. Our framework includes several optimizations for low-latency streaming and is able to offer a smooth interactive volumetric video experience.

REFERENCES

[All19] T. Alldieck, G. Pons-Moll, C. Theobalt, and M. Magnor: Tex2shape: Detailed full human body geometry from a single image," in Proc. Int. Conf. on Computer Vision (ICCV), 2019.

[Alli01] 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.

[Angu05] Anguelov, D., Srinivasan, P., Koller, D., Thrun, S., Rodgers, J., Davis, J.: SCAPE: Shape Completion and Animation of People. Proc. Computer Graphics (SIGGRAPH), 2005.

[Coll15] A. Collet, M. Chuang, P. Sweeney, D. Gillett, D. Evseev, D. Calabrese, H. Hoppe, A. Kirk, and S. Sullivan, "High-quality streamable free-viewpoint video," ACM Trans. on Graphics, vol. 34, no. 4, p. 69, 2015.

[Goo] Google Draco: a library for compressing and decompressing 3D geometric meshes and point clouds. Online, accessed April 27, 2020: https://github.com/google/draco.

[Gst] GStreamer: open source multimedia framework. Online, accessed April 27, 2020: https://gstreamer.freedesktop.org/.

[Gül20a] Gül, S., Podborski, D., Buchholz, T., Schierl, T. and Hellge, C., "Low-latency Cloud-based Volumetric Video Streaming Using Head Motion Prediction", Proceedings of the 30th ACM Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV). 2020.

[Gül20b] Gül, S., Podborski, D., Son, J., Bhullar, G.S., Buchholz, T., Schierl, T. and Hellge, C., "Cloud Rendering-based Volumetric Video Streaming System for Mixed Reality Services", Proceedings of the 11th ACM Multimedia Systems Conference (MMSys). 2020.

[Hol15] C. Holmberg, S. Hakansson, and G. Eriksson. 2015. Web real-time communication use cases and requirements. RFC7478.

[Hils20] A. Hilsmann, P. Fechteler, W. Morgenstern, W. Paier, I. Feldmann, O. Schreer, P. Eisert, "Going beyond Free Viewpoint: Creating Animatable Volumetric Video of Human Performances", *IET Computer Vision* , 2020

[Huan15] Huang, P., Tejera, M., Collomosse, J., Hilton, A.: Hybrid Skeletal-Surface Motion Graphs for Character Animation from 4D Performance Capture. ACM Trans. Graph. 34, 2, Article 17, 2015.

[Fech19a] Fechteler, P., Hilsmann, A., Eisert, P.: Example-based Body Model Optimization and Skinning. In: Proc. Eurographics 2016. Lisbon, Portugal, 2016.

[Fech19b] Fechteler, P., Hilsmann, A., Eisert, P.: Markerless Multiview Motion Capture with 3D Shape Model Adaptation. Computer Graphics Forum 38(6), 91–109, 2019.

[Mic] Microsoft, http://www.microsoft.com/en-us/mixed-reality/capture-studios

[Morg19] Morgenstern, W., Hilsmann, A., Eisert, P.: Progressive non-rigid registration of temporal mesh sequences. In: Proc. Europ. Conf.on Visual Media Production (CVMP). London, UK, 2019.

[Nve] NVIDIA Video Codec SDK. Online, accessed April 27, 2020: https://developer.nvidia.com/nvidia-video-codec-sdk

[Rob15] James Robinson and Cameron McCormack. 2015. Timing control for script-based animations. W3C Working Draft. Online, accessed April 27, 2020: https://www.w3.org/TR/2015/NOTE-animation-timing-20150922/

[Rob16] N. Robertini, D. Casas, H. Rhodin, H.-P. Seidel, and C. Theobalt: Model-based outdoor performance capture, in Proc. Int. Conf. on 3D Vision (3DV), 2016.

[Ross99] Rossignac, J. (1999). Edgebreaker: Connectivity compression for triangle meshes. *IEEE transactions on visualization and computer graphics*, *5*(1), 47-61.

[Sat17] Satyanarayanan, Mahadev. "The emergence of edge computing." *Computer* 50.1 (2017): 30-39.

[Schre19a] Schreer, O., Feldmann, I., Renault, S., Zepp, M., Eisert, P., Kauff, P.: Capture and 3D Video Processing of Volumetric Video, Proc. IEEE International Conference on Image Processing (ICIP), Taipei, Taiwan, Sep. 2019.

[Schre19b] Schreer, O., Feldmann, I., Kauff, P., Eisert, P., Tatzelt, D., Hellge, C., Müller, K., Ebner, T. and Bliedung, S., 2019, September. Lessons learnt during one year of commercial volumetric video production. In Proceedings of IBC conference, Amsterdam, Netherlands, September.

[Shi15] Shu Shi and Cheng-Hsin Hsu. 2015. A survey of interactive remote rendering systems. Comput. Surveys 47, 4 (May 2015), 1–29.

[Vol] Volucap GmbH, http://www.volucap.de