Performance of Augmented Reality Remote Rendering via Mobile Network

Kai Manuel Börner*, Arnulph Fuhrmann†, Michael Andreas Bösinger⋆

* ⋆ Vodafone GmbH
Technology Innovation Development
* kaimanuel.boerner@vodafone.com
⋆ michaelandreas.boesinger@vodafone.com

† TH Köln
Computer Graphics Group
arnulph.fuhrmann@th-koeln.de

Abstract: The algorithms for augmented reality (AR) applications, which are used to insert virtual objects in the real world, have been continuously improved in recent years. Software development kits (SDKs) for the mobile operating systems Android and iOS have been published which allow the usage of AR without markers. The complete rendering, as well as the tracking, are executed on the smartphone. Therefore, the graphical quality is limited. With remote rendering, large parts of the rendering can be transferred to a server. In this paper, a remote rendering system for an AR app based on Unity is presented. The system was implemented for an edge server, which is located within the network of the mobile network operator. Our evaluation shows that the system achieves better average frame times, even when the application is running for a longer time period. Furthermore, the graphical fidelity is improved compared to pure mobile rendering.

Keywords: Augmented Reality, Remote Rendering, Mobile Edge Computing, Unity

1 Introduction

Today, powerful smartphones are used for most consumer augmented reality (AR) applications, by supporting a software development kit (SDK) like *ARCore* or *ARKit*. Since the smartphone executes all the necessary components of the AR application, such as tracking and rendering, the visual quality of the virtual objects and hence the overall user experience heavily depends on the available performance of the smartphone.

In recent years, various remote rendering systems have been developed which enable cloud gaming, among other things. In such a system, the rendering is partially or completely outsourced to an external server. The result is streamed via the internet or a local network and finally displayed on the client. The advantage of this outsourcing is that the client can be much less powerful. Hence, the quality of the rendering can be increased, even on mobile devices. Remote rendering approaches can be split into two main categories [SH15]: In *model-based systems*, simplified or viewpoint dependent 3D models are transferred from the server to the client, which then performs the rendering. In basic variants of *image-based systems,* the rendering is performed on the server and a video stream is sent to the client.

As an alternative to a cloud server, a (mobile) edge server can be used. In contrast to

Veröffentlicht durch die Gesellschaft für Informatik e.V. 2020 in B. Weyers, C. Lürig, D. Zielasko (Hrsg.):
GI VR / AR Workshop 2020, 24. - 25, September 2020, Trier.

http://dx.doi.org/10.18420/vrar2020_14

cloud computing where the server is not part of the own network, in edge computing the server is closer to the client. Usually, the server is part of the provider's or the own local network. In mobile edge computing (MEC), the server is part of the Long Term Evolution (LTE) or 5G New Radio (5G NR) radio network. This architecture significantly reduces the latency and increases the bandwidth compared to the cloud servers.

In this paper, a proof of concept implementation for image-based remote rendering for AR applications is presented. The system was designed to provide high visual quality at a stable frame rate, therefore the virtual objects were rendered on an edge server and streamed as a video to the client. The client only has to handle the tracking in the real world and the user input.

2 Related Work

2.1 Remote Rendering

Early work on remote rendering was already performed over twenty years ago [Lev95, HS98]. For a complete literature review we refer the reader to the survey of Shi and Hsu [SH15], who describe two key challenges of remote rendering: The *interaction latency* and the limitations of networks such as the bandwidth. The interaction latency is the time between the user interaction and the display of the result which has been rendered (fully or partially) on the server. Over the last years, the different approaches for remote rendering were improved by reducing this latency. One main factor of latency is the network transmission and therefore the bitrate of the video. The system can reduce the latency by using an adaptive quantization and bitrate based on network-latency measurements [TMCP11].

Image-based systems, which usually have a high latency, have been extended by the transmission of a depth map. This map can be used for a warping algorithm on the client side, which adapts the viewpoint to the most recent client viewpoint. However, artifacts can occur due to previously hidden parts of the 3D scene which are disoccluded [SJNC09, CHC17]. Warping can also be used for splitting lighting computations in a remote rendering system. Costly global illumination (GI) can be calculated on a server and stored into a view specific GI-map. The client renders the scene only with ambient light and blends it with the GI-map for realistic lighting [LOJZ18].

In Shading Atlas Streaming [MVD+18] server-side shading is decoupled from client-side rendering by creating the shading information in object space and efficient compression using standard Moving Picture Experts Group (MPEG) encoding. This approach reduces the interaction latency even in the presence of large end-to-end latency while maintaining a high image quality.

2.2 Mobile Edge Computing

MEC can improve the application performance and user experience in several categories, like in real-time ultra-high-resolution video playback for mobile devices with limited battery

supply and constrained computation capability [ZYM+17].

Also the usage of MEC for AR is in focus. Bohez et al. [BTV+13] implemented a middleware platform for collaborative applications, in which data together with its processing are shared between multiple user. In AR remote live support applications a supporter helps the client remotely by inserting useful annotations into the video. The tracking of these applications can be improved by outsourcing to a local edge server [SRS17].

With the release of ARCore and ARKit, markerless tracking became available on a wide field of devices. By using a cloud service, place recognition can be performed on a server [PKF18]. This conserves battery power for other tasks, but introduces latency. With MEC the latency of markerless tracking and object recognition can be kept low while offloading computation intensive tasks to the edge cloud [ZHH18].

3 System Overview

The AR remote rendering system was implemented as an image-based approach for Unity 2018 and supports simple interactions such as moving an object or pressing a button. The client handles the tracking based on AR-Foundation (1.5.0-preview.7) which is a high-level application programming interface (API) for ARCore and ARKit. This API works with the standard Unity rendering pipeline, which is often referred to as the Legacy Render Pipeline (LRP). But it also adds further features, such as the support for the Lightweight Render Pipeline (LWRP) (4.10.0-preview) which is one implementation of the scriptable rendering pipeline. The LWRP reduces the number of graphic features and visual quality for an improved performance, mainly for low performance devices such as smartphones.

For our high-quality rendering server we extended the graphic pipeline in comparison to native running Unity AR applications. The server renders the virtual objects with the LRP. Then, it encodes the current frame into a video stream which is sends to the client. Therefore, the encoding is an addition to the server's LRP pixel processing. The client decodes one frame and blends it over its captured real world camera image. The decoding takes place within the client's application stage. Furthermore, the input system must be extended. In our system, the user interface (UI) is also rendered on the server, so only the touch position needs to be transmitted. The server processes it for interactions.

Figure 1 shows the entire implemented and tested Remote Render Pipeline (RRP). It consists of two asynchronous graphic pipelines from the server and the client, which are communicating over an Internet Protocol (IP) based network which uses either wireless local area network (W-LAN) or LTE. We chose two asynchronous pipelines to avoid image loss or interruption due to lost or delayed control packets, which start the rendering of the new frame on the server. Therefore, in the event of transmission errors, the server continues to render from the last known viewpoint. The client still uses the last received image of the virtual objects from the server. With this concept, it is possible to modify and optimize both sub-pipelines independently of each other. It is also possible to run both sub-pipelines

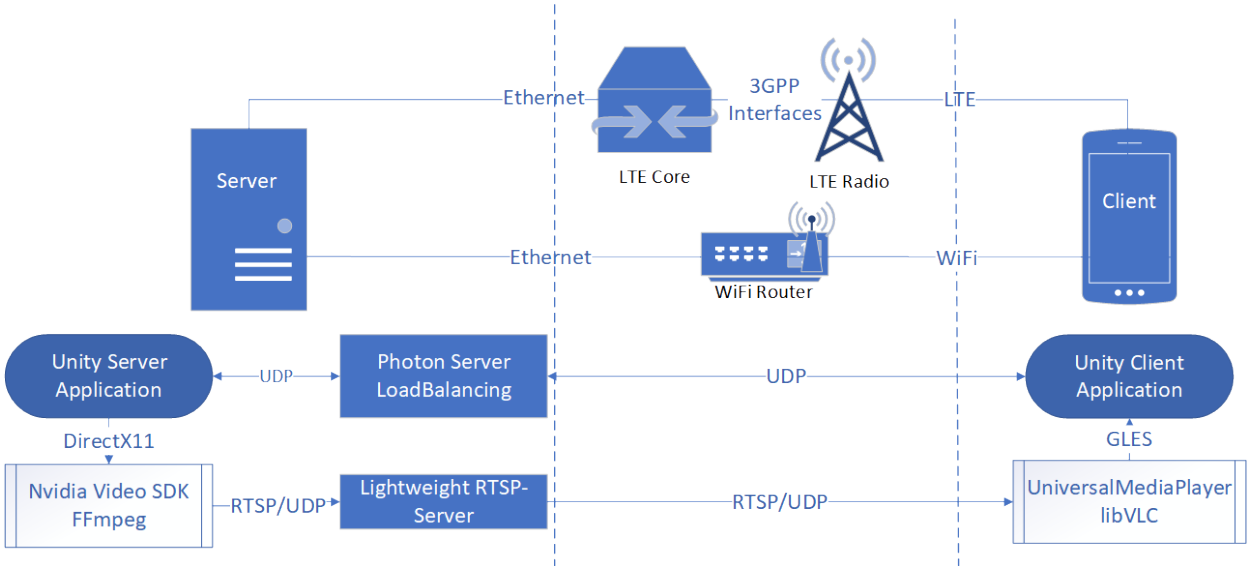

Figure 1: Overview of our remote rendering system. It uses either a W-LAN Router or a LTE Network with complete radio and core network.

with different frame rates. For example, it is possible that the server is unlimited and the client is limited to refresh with 30 frames per second (FPS). Additionally, the setting of the video frame rate can be changed separately. This can be used to optimize the latency or the stability of the video.

After the server application has completed its initialization, it waits for the client to connect. Once the client app is launched, it starts tracking and recognizing of feature points and surfaces. To synchronize the camera position and orientation, a reference point must be placed that represents the origin of the server's coordinate system. Once the connection to the server has been established , the two pipelines are linked and remote rendering is activated. The server starts the video encoding and the client starts video playing and displays the virtual objects. In addition to decoding the video, the client determines its current view position relative position and orientation to the reference point and sends it via the network to the server within the application layer. If an input was made, it is also transmitted. If the server receives new data, the transmitted view point and inputs are processed within its application layer. The rest of the server's pipeline is handled as in other applications until the encoding.

The transmission of video and audio was realized with the Real Time Streaming Protocol (RTSP) over User Datagram Protocol (UDP). Two separate streams were used, since this allows the separate analysis of video and audio latencies and a comparison of the efficiency of both implementations. The video stream uses Advanced Video Coding (AVC) and the Nvidia Video SDK[1] (9.0) allows the real-time encoding on the dedicated encoding units of the graphics processing unit (GPU). This reduces the additional encoding load on the server's GPU-shaders and the central processing unit (CPU) by allowing the encoding stage

[1] `https://developer.nvidia.com/nvidia-video-codec-sdk`

to directly access the rendered frame via the graphic API. Since the used codec does not supported the transmission of an alpha channel, a color keying must be done on the client, otherwise the camera image of the real world could not be displayed. The server fills all pixels which are not occupied by virtual objects with a defined color, which the client uses for color keying. The media framework FFmpeg[2] (4.2) was integrated into the server application for the audio coding and the multiplexing of the RTSP transmission. Opus was used as audio codec. The Universal Media Player[3] (1.7.3) Asset was integrated as the media player for the client, which enables the usage of the libVLC (3.0.9) from the VLC media player[4]. It allows access to several advanced options, like the caching. The transmission of the position and orientation of the client, as well as the input, is transmitted via the Photon Engine[5] (2.16) on UDP basis.

4 Results

The characteristics of the implemented RRP were evaluated with two different prototypes. The first prototype was used for latency measurements with a rudimentary scene and the second prototype was used for frame time measurements with a complex scene. The system was tested as an edge system with a client connected either via 5 Ghz W-LAN or via LTE on 700 MHz. The client was a Samsung Galaxy S9. The server was equipped with an Intel Core i7 6700K CPU, Nvidia Quadro RTX 4000 GPU, 32 GB RAM and Windows 10. The video encoder was set to 7 Mbps and 60 FPS. The render frame rate of the server was not limited, so that it can render more frames as necessary in order to measure its peak performance. Two resolutions 1080×2220 and 1440×2960 were used for the evaluations, further referenced as HD1080 and HD1440.

4.1 Video and Audio Latency

For the latency measurements, a rudimentary Unity scene was used. It contains only a cube, which can be moved, an audio source and an UI with a timer. The timer displays the seconds and milliseconds of the server's system time at the beginning of the current frame rendering. The video and audio latencies were verified locally on the server as well as via W-LAN and LTE for the complete system. The local latency measurements were made with the VLC media player, as it is based on the libVLC library as the media player of the client application. Additionally, the latency over W-LAN and LTE was also measured with the VLC media player on a laptop, in order to check, if the VLC Unity integration produces separate latency. During the latency tests, the client was limited to 60 FPS. Due to the usage of two RTSP streams, the video and audio latencies could be measured separately.

[2] `https://ffmpeg.org/`
[3] `https://assetstore.unity.com/packages/tools/video/ump-pro-android-ios-83283`
[4] `https://www.videolan.org/vlc/`
[5] `https://www.photonengine.com/`

For the video latency measurements, the timer which is part of the UI of the transmitted frame and a camera were used. The camera was placed so that the timer was visible on the server display as well as on the client display. The refresh rate of the server display and the camera exposure time were set to 144 Hz and 1/500 s. Ten pictures, containing both screens, were taken randomly. As the timer is part of the video transmission to the client, the difference of the timer state between the output of the server application and the media player represents the processing time of the video encoding, transmission, decoding and displaying. The audio latency was evaluated by using a test impulse which was played back by the server application and the media player. During the playback a recording was made by using a microphone and the audio program Audacity. The test impulse was triggered randomly 10 times. The time difference between the original impulse and the player impulse represents the time required for encoding, transmission and decoding. The means and standard deviations of the 10 video and audio latency measurements are summarized in Table 1.

The latency of the laptop is very similar to the local video latency. The video latency of about 140 ms seems to be caused by the caching of the VLC player and could be reduced by optimizations. The lower average latency of the laptop compared to that of the server is probably due to the fact that the server has to perform simultaneously encoding and decoding. But, this requires further investigation. The video latencies of the client application are about 200 ms larger than those of the VLC player. Since both are based on libVLC, we speculate that the current integration in Unity seems to cause those additional latencies.

The audio latency of the VLC player is much higher than the video latency. But, the client application achieves better audio latencies than the VLC media player, which are even lower than the video latencies. The audio integration in Unity seems to be more efficient. The increased audio latency of over 4 seconds for the laptop via LTE is conspicuous. In this scenario, the used modem seems to give a low priority to forwarding of audio data.

Configuration	Connection	HD1440	HD1080	Audio
Server	Local	145.00 / 6.6	143.00 / 6.63	461.3 / 0.46
Laptop	W-LAN	148.1 / 20.02	132.3 / 11.33	449.1 / 0.7
Laptop	LTE	140.8 / 10.51	136.2 / 7.00	4222.3 / 48.25
Client	W-LAN	343.3 / 13.85	342.2 / 11.22	221.22 / 1.54
Client	LTE	345.8 / 10.29	342.6 / 10.79	246.3 / 14.39

Table 1: Mean and standard deviation of measured latencies in milliseconds.

4.2 Performance Comparison

For the performance comparison we used the Unity tutorial project *Spotlight Tunnel* [6] and integrated it into our RRP system. We tested the scene in two scales (1:100) and (1:10) each

[6] `https://blogs.unity3d.com/2018/03/09/spotlight-team-best-practices-making-believable-visuals-in-unity/`

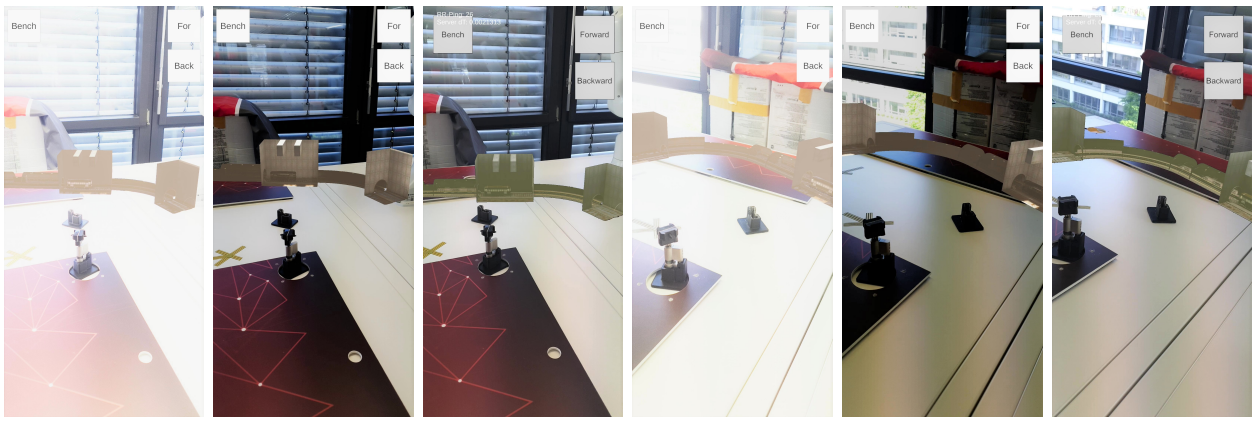

Figure 2: Comparison of two views of the AR object with different rendering pipelines (twice from left to right: LWRP, LRP, RRP). The real world view of the LWRP is overexposed due to the usage of linear color space for the Spotlight Tunnel and the used version of AR-Foundation, which can not handle a linear color space on the camera image.

with the "Ultra" graphic settings. The smaller scale served as an use case for an *AR object* application, where the majority of the pixels are not occupied by the virtual objects. We expected, that in this case the video coding can compress the empty areas more efficiently. The larger scale resulted in a *full screen* rendering of the Spotlight Tunnel. This use case was additionally chosen in order to be able to analyze whether the bit rate of the video will be lower with AR and therefore AR has an advantage over classic virtual reality (VR) applications for remote rendering. Figure 2 shows the AR test scene from two different view points with the three tested pipelines, Figure 3 shows the full screen test scene.

The frame times were recorded during several one minute long test runs. Every run was performed using a Franka Emika robot arm that moved the Galaxy S9 through the scene. Five runs were completed in succession to identify the impact of possible temperature-related throttling. We performed the five test runs for each of our rendering pipelines (LRP, LWRP and RRP) and for each of the two resolutions. The RRP was performed twice, i.e. for W-LAN and LTE.

We limited the frame rate of the client and the native applications to 30 FPS by syncing to the camera. For the RRP client we recorded two different frame times, the *screen refresh time* and the *effective frame time*. The screen refresh time is usually determined by the frame rate limit, but can be larger in cases of high workload on the client. The effective frame time is composed of the screen refresh time and whether a video frame is available and could be decoded in time. For example, if the screen refreshes after 33.3 ms and the frame is available, then this is the effective frame time. If one frame can not be decoded, but the next one, then the effective frame time increases to the sum of the last refresh and the current refresh time. If several frames are not available in time, the effective frame time is the sum of the display refresh times since the last available frame. In these cases, the client continues to display the last decoded frame for this duration. If the RTSP stream is interrupted it may occur in extreme cases that for a period of more than one second no new

Figure 3: Comparison of two views of the full screen test with different rendering pipelines (twice from left to right: LWRP, LRP, RRP). The LWRP shows less details and uses only a simply lighting model. The RRP shows more details than the native rendering pipelines.

frame is displayed. As soon as the stream is received again, playback will resume.

4.2.1 AR object

Figure 4 shows the frame times of the LRP and LWRP and the effective frame time of the RRP for the first AR test run in both resolutions. The LWRP is able to render more frames than the LRP, but it reduces the visual quality, which is clearly noticeable in Figure 2. The RRP shows better performance than the native pipelines in HD1440. The loss of more than one frame can be observed in Figure 4 at the peaks. This behavior requires further investigation and optimization. Although the RRP shows slightly less frames in HD1080 due to transmission errors, the graphic quality is better. The server is able to encode and render the scene in both resolutions with a much higher frame rate than the client (cf. Table 2).

4.2.2 Full screen

The results of the full screen tests reveal slight differences to the AR results. In contrast to AR, the LRP has a worse performance as the RRP even for HD1080. The server frame times are shown in Table 2. Additionally, the RRP was also tested in one run with HD1440 over W-LAN with a client limited to 60 FPS instead of 30. This allows to determine the potential of the RRP at a video frame rate of 60 fps. The effective frame time of the RRP was (M=20.64, SD=12.45), so the client RRP reaches 50 FPS. In this case, 10 FPS and thus every sixth video frame were lost during transmission. Reducing the high loss rate, is a topic of future work. Furthermore, it can be concluded that a higher video than display rate stabilizes the presentation.

4.2.3 Data usage and temperature related throttling

During the performance tests, the video bitrate was recorded. Table 3 shows the minimum and maximum data rates of both applications for the two used resolutions. The maximum

Figure 4: The frame times of the AR scene in HD1080 (top) and HD1440 (bottom). The RRP has several peaks compared to LRP and LWRP, which are related to transmission errors. Even with this peaks, the RRP shows a higher performance in HD1440.

	HD1080		HD1440	
Application	W-LAN	LTE	W-LAN	LTE
AR object	M=2.98, SD=2.28	M=2.88, SD=2.19	M=5.61, SD=4.93	M=5.51, SD=4.60
Full screen	M=5.50, SD=4.97	M=5.30, SD=4.80	M=10.31, SD=7.11	M=10.07, SD=6.95

Table 2: Mean and standard deviation of the frame time (rendering + encoding) of the remote rendering server in milliseconds during the first test run. The large standard deviation is due to the blocking of the render thread by the encoding, whereby not all rendered frames were encoded.

datarate from the AR representation is lower than the minimum rate from the full screen application. This behaviour is related to the smaller number of used pixels. The unused (black) pixels can be compressed more effectively, because within this area are no major changes between the images. Since the maximum value is reached in full screen applications, the encoder is exhausted. Hence, a higher bit rate setting would be required for these applications in order to avoid coding artifacts and dropouts. The AR version has more than 1 Mbps free and can therefore be transferred without coding artifacts.

	HD1080		HD1440	
Application	Min.	Max	Min.	Max
AR object	5,373	5,627	5,418	5,637
Full screen	6,799	6,941	6,674	6,896

Table 3: Minimum and maximum video bitrate in kbps during the performance tests. The AR-representations can be compressed more efficiently by the encoder and needs less bitrate. The configured bitrate for the encoder was 7 Mbps.

Native running AR applications have special demands on the hardware, the end devices heat up due to this load. Modern devices therefore throttle their performance under certain circumstances. This effect occurred in the applications during the five test runs. Table 4 shows the performance in percent of the fifth run compared to the first run. The LRP and LWRP lost up to 36 % , in contrast to the RRP with 13 % . Even though the RRP displayed on HD1080 less frames than the LWRP in the first run, and also less than the LRP in AR, there is less throttling on the end device. This results in a better long-use performance.

	HD1080			HD1440		
Application	LRP	LWRP	RRP	LRP	LWRP	RRP
AR object	66.7 %	78.5 %	100.6 %	66.3 %	67.1 %	93.0 %
Full screen	70.5 %	78.8 %	95.9 %	72.7 %	64.7 %	87.6 %

Table 4: Average FPS of run 5 compared to run 1 in percent. The RRP frame rate depends on the current network situation as well, hence a value larger 100 % can occur. The RRP values are from the LTE runs.

5 Conclusions

In this paper, the implementation and evaluation of a Unity AR remote rendering system was presented. The system can work as a mobile edge computing system. The lightweight client is built on top of AR-Foundation from Unity for benefiting from its marker-less tracking feature. The rendering server is using the Nvidia Video SDK for high-quality low-latency video encoding. Additionally, it was investigated how to optimize the connection between the rendering pipeline of the server and the video encoding. The proof of concept implementation

demonstrates that there is only a slight difference in the end-to-end latency of the edge setup over LTE and the edge computing over W-LAN. However, further work is necessary in which the approaches of the presented system will be further optimized and investigated. The similar video latencies between W-LAN and LTE show that further optimization of the ultra-reliable and low-latency communication (URLLC) capability of the integrated media player is necessary, where improvements can also be applied to other low latency video applications.

The scriptable rendering pipeline introduced in Unity 2018 provides additional options for optimization. In addition, this opens the necessary access to integrate further remote rendering methods. Nevertheless, certain accesses to subsystems, such as the input system, are severely restricted or not possible. For future works, a further opening of Unity or cooperation with the developers would therefore be necessary. With the introduction of 5G NR and later 5G Core (5GC), new features are available such as network slicing, which guarantees a certain performance level to a specific user. Further investigations with 5G NR and slicing should also be carried out for remote rendering systems, since the special network requirements of real-time applications supposed to be covered with them.

References

[BTV+13] Steven Bohez, Joeri De Turck, Tim Verbelen, Pieter Simoens, and Bart Dhoedt. Mobile, collaborative augmented reality using cloudlets. In *2013 International Conference on MOBILe Wireless MiddleWARE, Operating Systems, and Applications*. European Alliance for Innovation, nov 2013.

[CHC17] Yu-Jung Chen, Chung-Yao Hung, and Shao-Yi Chien. Distributed rendering: Interaction delay reduction in remote rendering with client-end gpu-accelerated scene warping technique. In *2017 IEEE International Conference on Multimedia Expo Workshops (ICMEW)*, pages 67–72, July 2017.

[HS98] Gerd Hesina and Dieter Schmalstieg. A network architecture for remote rendering. In *Proceedings of the 2nd International Workshop on Distributed Interactive Simulation and Real-Time Applications*, pages 88–91, 1998.

[Lev95] Marc Levoy. Polygon-assisted JPEG and MPEG compression of synthetic images. In *SIGGRAPH '95*, page 21–28, New York, NY, USA, 1995. ACM.

[LOJZ18] Chang Liu, Wei Tsang Ooi, Jinyuan Jia, and Lei Zhao. Cloud baking: Collaborative scene illumination for dynamic Web3D scenes. *ACM Trans. Multimedia Comput. Commun. Appl.*, 14(3s):59:1–59:20, June 2018.

[MVD+18] Joerg H. Mueller, Philip Voglreiter, Mark Dokter, Thomas Neff, Mina Makar, Markus Steinberger, and Dieter Schmalstieg. Shading atlas streaming. In *SIG-*

GRAPH Asia 2018 Technical Papers, SIGGRAPH Asia '18, pages 199:1–199:16, New York, NY, USA, 2018. ACM.

[PKF18] Javier R. Puigvert, Till Krempel, and Arnulph Fuhrmann. Localization service using sparse visual information based on recent augmented reality platforms. In *2018 IEEE International Symposium on Mixed and Augmented Reality Adjunct (ISMAR-Adjunct)*, pages 415–416, 2018.

[SH15] Shu Shi and Cheng-Hsin Hsu. A survey of interactive remote rendering systems. *ACM Computing Surveys*, 47(4):1–29, may 2015.

[SJNC09] Shu Shi, Won J. Jeon, Klara Nahrstedt, and Roy H. Campbell. Real-time remote rendering of 3D video for mobile devices. In *Proceedings of the 17th ACM International Conference on Multimedia*, MM '09, pages 391–400, New York, NY, USA, 2009. ACM.

[SRS17] Michael Schneider, Jason Rambach, and Didier Stricker. Augmented reality based on edge computing using the example of remote live support. In *2017 IEEE International Conference on Industrial Technology (ICIT)*. IEEE, mar 2017.

[TMCP11] Nicolas Tizon, Christina Moreno, Mihai Cernea, and Marius Preda. MPEG-4-based adaptive remote rendering for video games. In *Proceedings of the 16th International Conference on 3D Web Technology*, Web3D '11, pages 45–50, New York, NY, USA, 2011. ACM.

[ZHH18] Wenxiao Zhang, Bo Han, and Pan Hui. Jaguar: Low latency mobile augmented reality with flexible tracking. In *Proceedings of the 26th ACM International Conference on Multimedia*, MM '18, pages 355–363, New York, NY, USA, 2018. ACM.

[ZYM+17] Quanxin Zhao, Tong You, Xiaohui Ma, Yuming Mao, Supeng Leng, Ning Yang, and Zhiwei Zhao. Mobile edge decoding for saving energy and improving experience. In *2017 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*. IEEE, jun 2017.