Low latency edge rendering scheme for interactive 360 degree virtual reality gaming

Low Latency Edge Rendering Scheme for Interactive 360 Degree Virtual Reality Gaming

Marko Viitanen, Jarno Vanne, Timo D. Hämäläinen, Ari Kulmala*
Laboratory of Pervasive Computing, Tampere University of Technology, Tampere, Finland
*Datacenter Infrastructure Modules, Nokia Corp., Tampere, Finland

***Abstract*—This paper describes the core functionality and a proof-of-concept demonstration setup for remote 360 degree stereo virtual reality (VR) gaming. In this end-to-end scheme, the execution of a VR game is off-loaded from an end user device to a cloud edge server in which the executed game is rendered based on user's field of view (FoV) and control actions. Headset and controller feedback is transmitted over the network to the server from which the rendered views of the game are streamed to a user in real-time as encoded HEVC video frames. This approach saves energy and computation load of the end terminals by making use of the latest advancements in network connection speed and quality. In the showcased demonstration, a VR game is run in Unity on a laptop powered by i7 7820HK processor and GTX 1070 GPU. The 360 degree spherical view of the game is rendered and converted to a rectangular frame using equirectangular projection (ERP). The ERP video is sliced vertically and only the FoV is encoded with Kvazaar HEVC encoder in real time and sent over the network in UDP packets. Another laptop is used for playback with a HTC Vive VR headset. Our system can reach an end-to-end latency of 30 ms and bit rate of 20 Mbps for stereo 1080p30 format.**

Keywords-virtual reality (VR); edge computing; video coding; High Efficiency Video Coding (HEVC); 360 degree video

I. INTRODUCTION

Virtual reality (*VR*) gaming in the cloud has gained ground after the release of multiple VR systems with *head-mounted displays* (*HMDs*) on the consumer market, such as Oculus Rift, HTC Vive, Windows Mixed Reality, and PlayStation VR. One can identify two primary prerequisites for remote gaming:

1) A remote server is equipped with high-end *graphics processing units* (*GPUs*) for smooth game pipeline processing and *central processing units* (*CPUs*) for real-time video compression
2) Available network capacities are high and stable enough for managing the communication between the server and a client with an acceptable motion-to-photon latency

Remote gaming boosts the global growth of VR and *augmented reality* (*AR*) traffic, which is forecast to be 20-fold in five years over that of 2016 [1]. Hence, migrating to remote approaches inevitably calls for efficient video compression. The latest international video coding standard, *High Efficiency Video Coding* (*HEVC*/*H.265*) [2], reduces the bit rate by almost 40% over the preceding state-of-the-art standard AVC/H.264 [3]. Therefore, HEVC is a potential technology for remote gaming to meet the available network capacities. Thanks to the latest advancements in network technologies, top 100 countries reach average speeds of at least 10 Mbps with both mobile and broadband Internet connections [4].

For the time being, compressed video has been utilized by some commercial remote gaming solutions such as in Amazon AWS cloud servers [5] and in some open-source implementations [6], [7]. However, such services are not yet publicly available for the VR gaming. Hou et al. [8] presented a study of different methodologies for 360 degree AR/VR streaming. The compound latency of head tracking and HMD was mentioned to be less than 20 ms with the current technology, but the complete latency of video delivery was not reported. A similar approach was described in [9] for *field of view* (*FoV*) adaptive HEVC video system, but it was targeted at non-interactive video content. The latency is not reported, but in similar systems [10] it exceeds 100 ms because of the standard *Dynamic Adaptive Streaming over HTTP* (*DASH*) approach. There also exist some dedicated hardware approaches [11] but these commercial solutions fall out of the scope of this paper.

This work proposes an end-to-end system for real-time remote VR gaming with low latency. To the best of our knowledge, this is the first software-oriented solution with a working prototype in the field. The core components of the system are Unity® game engine and Kvazaar open-source HEVC encoder [12] that takes care of real-time video encoding. Recent versions of the Kvazaar software support an enhanced coding mode for 360 video, making Kvazaar an obvious choice for a low delay 360 video compression and transmission system. Bandwidth and computation resources are further saved with selective transmission of the surrounding 360 degree view.

The remainder of this paper is structured as follows. Section 2 goes through the main phases of the proposed edge rendering scheme. Section 3 introduces the respective proof-of-concept setup to demonstrate the proposed approach. Performance aspects of our proposal are considered in Section 4. Section 5 concludes the paper.

II. THE PROPOSED EDGE RENDERING SCHEME

Fig. 1 depicts the proposed system at high level. The selected configuration consists of a server and a single client computer.

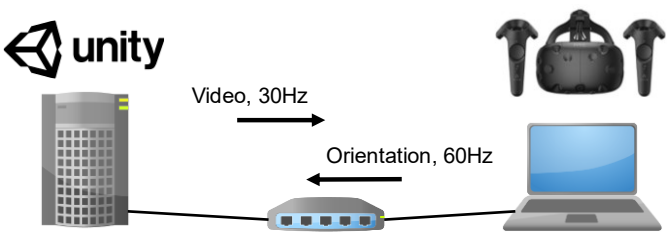

Figure 1. High-level system setup.

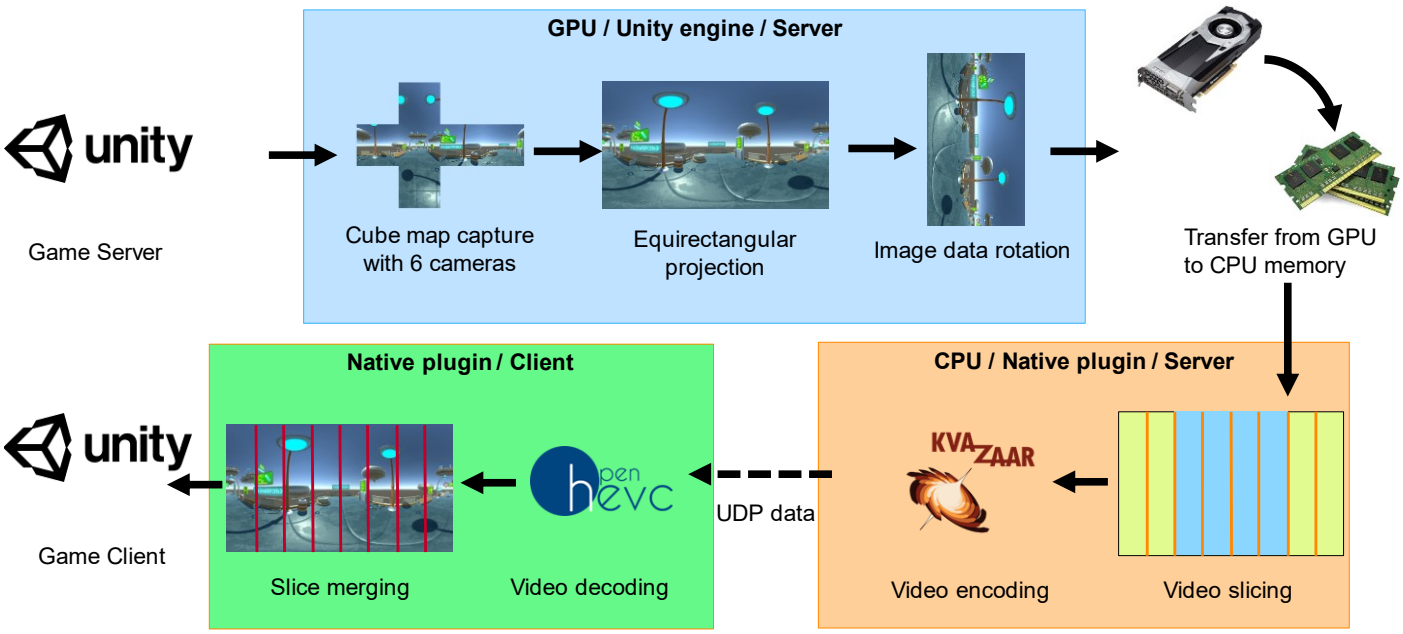

Figure 2. The processing stages of the proposed edge rendering scheme and associated platform components.

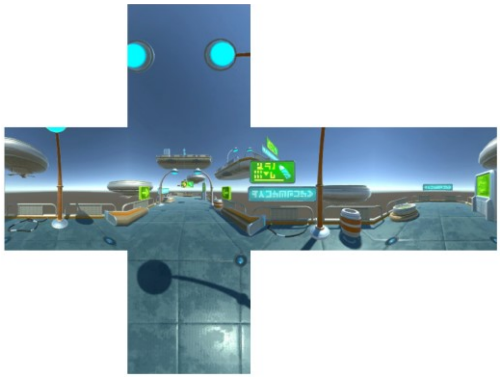

Figure 3. Cube map after rendering.

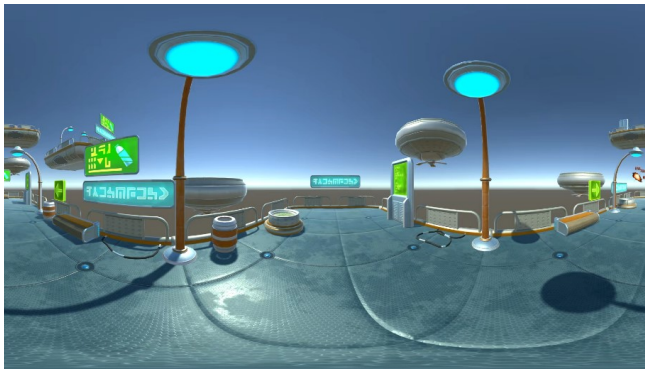

Figure 4. Final ERP format image.

The whole functionality is realized on standard GPUs and CPUs without any dedicated hardware acceleration.

The proposed scheme and the related platform components are detailed in Fig. 2 with separation of GPU and CPU processing. GPU, in addition to processing the game graphics, is used for accelerating the 360 video capture by allowing the pixel data to stay in the GPU memory after rendering. This way, the costly GPU to CPU memory transfer is avoided.

A. VR game

The proposed system supports VR games built with Unity® game engine. In principle, the system is compatible with any Unity game, but direct interaction with objects may be limited because the client sees only the video stream.

Our demonstration is run with a shooter type game where the user can control a turret with the HTC Vive controller. Game counts points for each hit and scores the player for a round of game. Player rig is built into a single Unity module called Prefab, where a C# script interfaces with the native plugin that handles the game communication.

B. Game rendering

A custom camera rig is set up inside the Unity C# script with six cameras per eye. Each camera renders to a Unity RenderTexture of 960×960 pixels. Fig. 3 shows the output after rendering six camera views to a cube map composed of top, bottom, and four sides.

C. Equirectangular projection (ERP)

The rendered cube map textures are passed to an *equirectangular projection* (*ERP*) that is implemented with a compute shader inside the Unity®. The ERP is commonly applied with 360 video since it has an advantage of storing a spherical input frame in a single rectangular image.

In this work, the ERP shader produces 1920×1080 pixel 32-bit RGBA output. Supersample antialiasing is used to reduce edge pixelation and smoothing the output. This means that when the ERP pixels are looped through, each kernel pass calculates the values for four adjacent pixel positions and outputs the average. The output ERP image is shown in Fig. 4.

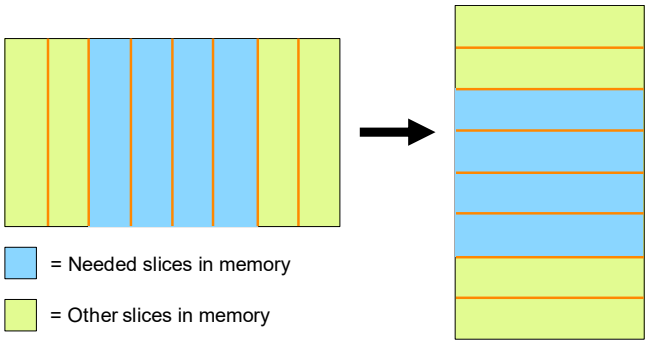

Figure 5. Image rotation from row to column major order.

Figure 6. Projected 360 video on the client side.

D. *Video rotation*

Since only the FoV is of interest at a time, retrieving the whole ERP image from the GPU is not needed. ERP shader buffer, however, requires that the loaded data is in a continuous memory. Rotating the ERP image from row to column major order, as shown in Fig. 5, converts the image data to this kind of a continuous format. The rotation is conducted by permutating the pixels inside the ERP shader to different positions without any performance loss. Two separate memory areas are needed from the buffer at maximum because the view wraps around at the ERP seam with some FoVs.

E. *GPU to CPU memory transfer*

Unity C# scripting interface is running in a single thread, making it difficult to read data from the GPU without blocking the execution. New beta versions of Unity include support for asynchronous GPU reading, but the implementation is not yet available for OpenGL on Linux. A workaround is to use native plugin interface and native OpenGL to read from the compute shader buffer outside of the Unity script loop. Multiple threads are started in the native plugin when initializing it. With a signal from the scripting interface, the threads start waiting for the compute shader to complete and read back the compute shader buffer from the GPU memory.

F. *Video slicing*

Before encoding, the ERP video is sliced into 128 pixel wide slices. All slices are independently compressed and processed, so it is possible to discard part of the slices based on the client's FoV. With 1080p output, seven out of the 15 image slices are sent to the client, i.e., around 170 degree view. HTC Vive with horizontal FoV of around 110 degrees has enough slices for fast head movement without the view limitation being visible. Fig. 6 visualizes the rendering range with the limited FoV.

G. *Video encoding*

The selected video slices are compressed with Kvazaar encoder software to HEVC format. Each slice is encoded by a separate Kvazaar instance.

The poles of the ERP image have redundant pixels, which cause unnecessary overhead in compression. Therefore, Kvazaar includes an option for ERP *Adaptive QP* (*AQP*) coding that allows an adaptive quality for the image. The ERP-AQP mode is based on the metric called *weighted-to-spherically-uniform peak-signal-to-noise ratio* (*WS-PSNR*) [13] according to which less weight is given to the poles of the sphere, since they are spatially compressed into a smaller area in the sphere surface. Besides improving coding efficiency, AQP mode also speeds up the coding process by reducing the number of coding coefficients. To prevent error propagation with lost frames, all-intra HEVC encoding is used.

Image rotation from row to column major order (Fig. 5) also improves coding efficiency. The rotated slices are 128 pixels high so they allow two *Wavefront Parallel Processing* (*WPP*) [14] threads to operate simultaneously when the size of *Coding Tree Units* (*CTUs*) is 64×64 pixels. This decreases compression delay when enough computation threads are available.

H. *Video transmission*

Basic *User Datagram Protocol* (*UDP*) packets are used for transmitting the video stream over the network to the clients. Each packet contains a header with video slice index, timestamp, and length followed by the HEVC video frame. With the proposed image slicing, each compressed image slice can fit into a single UDP packet with payload under 65,507 bytes. In case of a packet loss, only a single slice of an image is lost whereas others can be decoded.

I. *Video decoding and playback*

In the client end, an empty Unity scene with SteamVR support is running with a script-generated sphere per eye. The texture is allocated in Unity and passed to the native plugin. A client listens to a UDP socket and when data is received, headers are parsed and the video stream is decompressed using OpenHEVC decoder [15].

Output of the decoding is copied to the allocated texture in the correct position, depending on the slice. Looking around in the scene is near instant, but the video surface of the sphere is updated only at 30 Hz. This does not cause additional motion sickness because of the low motion-to-photon latency.

J. *Client feedback*

Feedback from the headset and controllers are sent to the server at 60 Hz. The feedback data includes orientation, position, rotation, and button states. With server sending video frames at 30 Hz, the feedback data will always be from maximum of 17 ms before the video frame. Even faster update rates would be allowed since the HTC Vive gives position and rotation information at 100 Hz.

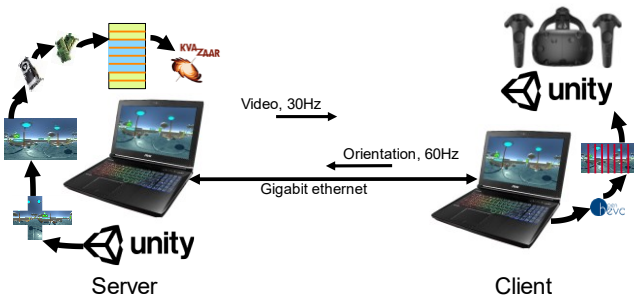

Figure 7. Demonstration setup.

III. Demonstration Setup

The proposed demonstration setup is depicted in Fig. 7. It is composed of two gaming grade laptops, both equipped with Nvidia GTX 1070 GPU and Intel i7 7820HK CPU. The laptops are connected using Gigabit Ethernet for low latency communication.

A. Server and client

The other one of the laptops acts as the server for running the game software and rendering the 360 view from the scene at 30 Hz (six cameras per eye). The resulting images are sent to the ERP shader. The ERP shader buffer is shared with the native plugin, which handles the data reading from the GPU buffer to the CPU memory. In CPU, the ERP image is sliced to 128 pixel wide segments, the slices of FoV are compressed with Kvazaar, and the compressed bit streams are sent to the Ethernet.

The other laptop acts as the end terminal equipped with a HTC Vive VR system. This laptop receives data from the server, parses the headers, and decodes the data using OpenHEVC decoder. The raw YUV data is copied to a texture, which the Unity renders to a sphere surface and displays using HTC Vive. Feedback from the headset and the controllers is read and sent to the server at 60 Hz.

B. Visitor experience

Users will be able to play a simple VR game using the HTC Vive headset and the controllers for interaction. Head movement and drawing of the scene is carried out locally, making the latency of the game almost invisible to the user. However, by allowing objects spawning with a click of the button, the latency is visible to the user by observing the spawn time. Fig. 6 illustrates the user experience in this demonstrator.

IV. System Performance

Instant feedback on head or controller movements is a crucial factor for a low motion-to-photon latency, i.e., a pleasant VR experience. Since the local scene rendering with projection spheres takes place on the client side, the instant feedback is possible even with some latency on the video transmission.

Our experiments show that an end-to-end latency of under 30 ms is achieved by rendering stereo 1080p30 video on the high-end server. The i7 processor applied in the demonstration is fast enough to encode seven video slices at 30 fps without any hardware acceleration. Sending only the slices of the FoV reduces bit rate by around 50%. Depending on the quality settings and the game content, the output bit rate to the client is around 10 Mbps for mono and 20 Mbps for stereo rendering.

V. Conclusions

This paper presented a low latency edge rendering scheme for remote VR gaming. The proposal aims at reducing energy and computation burden of the end user devices by performing game rendering on the server side from which the rendered views are sent to a user as encoded HEVC video frames. The system creates 360 ERP video from the rendered views, divides it into 128 pixel-wide image slices, and makes use of the user's head orientation data to limit the transmission of the image slices to that of FoV. The selected slices are encoded in real time and in around half bit rate over the case where all slices are encoded. The obtained performance results indicate that the 30 ms latency is low enough for enjoyable experience when playing a remotely rendered VR game.

Acknowledgment

This work was supported in part by Nokia and the Academy of Finland (decision no. 301820).

References

[1] Cisco, Cisco Visual Networking Index: Forecast and Methodology, 2016-2021, Jun. 2017.

[2] *High Efficiency Video Coding*, document ITU-T Rec. H.265 and ISO/IEC 23008-2 (HEVC), ITU-T and ISO/IEC, Apr. 2013.

[3] *Advanced Video Coding for Generic Audiovisual Services*, document ITU-T Rec. H.264 and ISO/IEC 14496-10 (AVC), ITU-T and ISO/IEC, Mar. 2009.

[4] *Speedtest Global Index* [Online]. Available: http://www.speedtest.net/global-index

[5] *Parsec* [Online]. Available: https://parsecgaming.com

[6] J. Beyer and R. Varbelow, "Stream-A-Game: an open-source mobile cloud gaming platform," *in Proc. Int. Workshop Network and Systems Support for Games*, Zagreb, Croatia, Dec. 2015.

[7] C. Y. Huang, C. H. Hsu, Y. C. Chang, and K. T. Chen, "GamingAnywhere: an open cloud gaming system," *in Proc. ACM Multimedia Syst. Conf.*, Oslo, Norway, Feb. 2013.

[8] X. Hou, Y. Lu, and S. Dey, "Wireless VR/AR with edge/cloud computing," *in Proc. IEEE Int. Conf. Computer Commun. and Networks*, Vancouver, Canada, Jul. 2017.

[9] R. Skupin, Y. Sanchez, C. Hellge, and T. Schierl, "Tile based HEVC video for head mounted displays," *in Proc. IEEE Int. Symp. Multimedia*, San Jose, California, USA, Dec. 2016.

[10] N. Bouzakaria, C. Concolato, and J. Le Feuvre, "Overhead and performance of low latency live streaming using MPEG-DASH," *in Proc. IEEE Int. Conf. Information, Intelligence, Systems and Applications*, Chania, Greece, Jul. 2014.

[11] *S. Solotko,* "The first billion users: Powering virtual reality with advanced video encoding & rendering," *whitepaper by TIRIAS Research*, Jan. 2018.

[12] *Kvazaar HEVC encoder* [Online]. Available: https://github.com/ultravideo/kvazaar

[13] Y. Sun, A. Lu, and L. Yu, "AHG8: WS-PSNR for 360 video objective quality evaluation," *document JVET-D0040*, Chengdu, China, Oct. 2016.

[14] F. Henry and S. Pateux, "Wavefront parallel processing," *Document JCTVC-E196*, Geneva, Switzerland, Mar. 2011.

[15] OpenHEVC [Online]. Available: https://github.com/OpenHEVC/openHEVC