Proceedings of the 2014 Federated Conference on
Computer Science and Information Systems pp. 687–694

DOI: 10.15439/2014F42
ACSIS, Vol. 2

High quality, low latency in-home streaming of multimedia applications for mobile devices

Daniel Pohl*, Stefan Nickels†, Ram Nalla*, Oliver Grau*
* Intel Labs
Intel Corporation
Email: {Daniel.Pohl|Ram.Nalla|Oliver.Grau}@intel.com
† Intel Visual Computing Institute
Saarland University
Saarbrücken
Email: nickels@intel-vci.uni-saarland.de

***Abstract*—Today, mobile devices like smartphones and tablets are becoming more powerful and exhibit enhanced 3D graphics performance. However, the overall computing power of these devices is still limited regarding usage scenarios like photo-realistic gaming, enabling an immersive virtual reality experience or real-time processing and visualization of big data. To overcome these limitations application streaming solutions are constantly gaining focus. The idea is to transfer the graphics output of an application running on a server or even a cluster to a mobile device, conveying the impression that the application is running locally. User inputs on the mobile client side are processed and sent back to the server. The main criteria for successful application streaming are low latency, since users want to interact with the scene in near real-time, as well as high image quality. Here, we present a novel application framework suitable for streaming applications from high-end machines to mobile devices. Using real-time ETC1 compression in combination with a distributed rendering architecture we fully leverage recent progress in wireless computer networking standards (IEEE 802.11ac) for mobile devices, achieving much higher image quality at half the latency compared to other in-home streaming solutions.**

I. Introduction

THE ADVENT of powerful handheld devices like smartphones and tablets offers the ability for users to access and consume media content almost everywhere without the need for wired connections. Video and audio streaming technologies have dramatically evolved and have become common technologies. However, in scenarios where users need to be able to interact with the displayed content and where high image quality is desired, video streaming derived technologies are typically not suitable since they introduce latency and image artifacts due to high video compression. Remote desktop applications fail when it comes to using 3D graphics applications like computer games or real-time visualization of big data in scientific HPC applications. Enabling these applications over Internet connections suffers significantly due to restrictions induced by the limits of today's Internet bandwidth and latency. Streaming those in local networks, commonly referred to as in-home streaming, still remains a very challenging task in particular when targeted at small devices like tablets or smartphones that rely on Wi-Fi connections.

In this paper, we present a novel lightweight framework for in-home streaming of interactive applications to small devices, utilizing the latest developments in wireless computer networking standards (IEEE 802.11ac [1]) for mobile devices. Further, we use a distributed rendering architecture [2] in combination with a high-quality, hardware-accelerated decompression scheme utilizing the capabilities of modern handheld devices, resulting in much higher image quality and half the latency compared to other streaming solutions.

The setup is shown in Figure 1. The main application is running on a server or even a group of servers. Via network connection, the graphical output of the server application is streamed to a client application running on a mobile device. In addition, a back channel connection is present that collects user input events on the client and sends it back to the server. The server reacts to this input and produces an updated image, which is then transferred back and displayed at the client. The Quality of Experience is determined mainly by two factors: firstly, the delay between a user input issued on the client and the server-provided graphics refresh displayed at the client should be as low as possible. Secondly, the graphics quality of the streamed application on the client side should be as high as possible, even during scenarios with high motion.

II. Related Work

In this section we give an overview of known streaming technologies and applications, which we separate into three classes.

Classical desktop sharing and terminal applications: Examples are Microsoft's Remote Desktop Connection [3] or VNC (Virtual Network Computing) [4]. These are optimized for typical 2D applications like text processing or spreadsheet

978-83-60810-58-3/$25.00

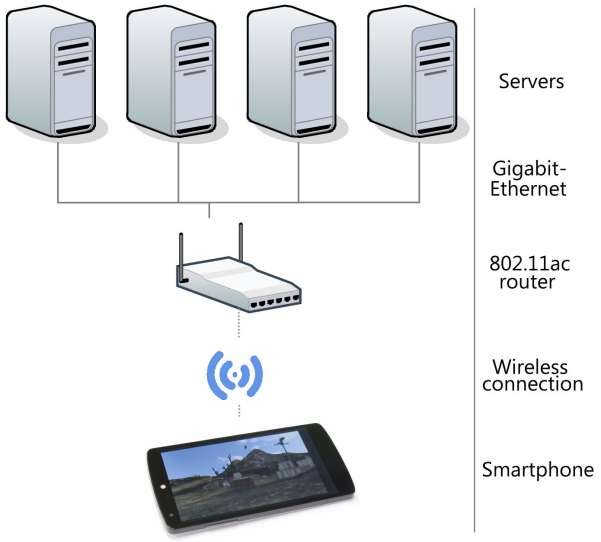

Fig. 1. *Distributed rendering, in-home streaming setup targeted at mobile devices. The four servers are rendering an image. Over Gigabit-Ethernet they transport it to a router with support for IEEE 802.11ac. The router sends the image data wirelessly to the client device (smartphone) which displays it.*

calculations. 3D support is typically very limited and, if supported, not capable to cope with the demands of real-time 3D games and visualizations.

Cloud gaming: The second class of streaming technologies has emerged from the field of computer gaming. Popular commercial solutions like Gaikai [5] or OnLive [6] aim at streaming applications, mainly games, via Internet connection from a cloud gaming server to a user's desktop. There are also open source approaches like Gaming Anywhere [7]. All of these are specifically optimized for usage with Internet connections and typically rely on the H.264/MPEG-4 AVC [8] video codec for streaming graphics. Gaikai and OnLive require a 3-5 Mbps internet connection at minimum and end-to-end latency values are at best 150 ms under optimal conditions (c.f. [7]). OnLive uses a proprietary hardware compression chip in dedicated gaming servers hosted by OnLive. Gaikai's approach has meanwhile been integrated into the PlayStation 4 console [9] after Sony aquired the company and has been renamed to "PlayStation Now" [10]. The service is limited on both the client and server side to dedicated hardware. In general, cloud gaming approaches are not optimized for in-home streaming, sacrificing image quality for lowering network traffic and reducing processing latency.

Dedicated in-home streaming: The third category are approaches designed for delivering applications to other devices in a local network using wired or wireless connections. Recently, Valve's game distribution platform Steam [11] introduced in-home streaming support in a beta version. Nvidia released a portable game console named Shield [12] in 2013, based on current mobile device hardware running an Android operating system. From a PC a game can be streamed to the console. Both approaches rely again on the H.264 codec and are exclusively capable of streaming games. In addition, Nvidia Shield is bound to Nvidia graphics cards and mobile platform architectures. A further streaming approach, which also handles in-home usage is Splashtop [13]. It can stream any application, game or the complete desktop from a single machine using H.264 compression.

We are not considering solutions like Miracast [14], Intel's WiDi [15] or Display as a Service [16] as these are aimed at replicating pixels to another display, not at streaming interactive applications. Further we exclude approaches like Games@Large [17] that stream hardware-specific 3D function calls that are usually not compatible with mobile devices.

III. System

In the following, we will propose a new framework for streaming applications from one or many high-end machines to a mobile device. Using a hardware-enabled decompression scheme in combination with a distributed rendering approach, we fully utilize the potential of recent progress in wireless computer networking standards on mobile devices. With this setup we achieve higher image quality and significantly lower latency than other established in-home streaming approaches. We first give an overview of the hardware setup used in our approach. Then we explain our decision on the compression scheme we used and after that we talk about the details of our software framework and application setup.

Our general system setup is depicted in Figure 1. The graphical output of a server application is streamed to a mobile device, in this case a smartphone, running the client mobile app. The server side consists of four machines in a distributed rendering setup. All devices operate in the same LAN. The server machines are connected by wire to a router which additionally spans a Wi-Fi cell to which the smartphone is connected.

A. Hardware Setup

Here, we describe the hardware specifics of our servers, the client and the network devices.

Our distributed rendering setup consists of four dual-socket workstations using the Intel Xeon X5690 CPUs (6 cores, 12 threads, 3.46 GHz) and the Intel 82575EB Gigabit Ethernet NIC. The client is a LG Nexus 5 smartphone which uses the Snapdragon 800 CPU (4 cores, 2.3 GHz) and the Broadcom BCM4339 802.11ac wireless chip. The devices are connected together through a Netgear R6300 WLAN Gigabit Router. The servers use wired Gigabit Ethernet to connect to the four Ethernet ports of the router. The smartphone connects wirelessly over 802.11ac (1-antenna setup).

B. Compression Setup

In this section we explain why we have chosen the Ericsson Texture Compression format (ETC1) [18] over other commonly used methods.

First we have a look at how displaying of streamed content is usually handled on the client side. Using the popular video library FFmpeg [19] and the H.264 codec an arriving stream at the client needs to be decoded. Using a CPU-based pipeline the decoding result is an image in the YUV420 [20] color space. As this format is usually not natively supported for displaying, the data is converted into the RGB or RGBA format. From there, the uncompressed image data will be uploaded to the graphics chip to be displayed.

If a hardware H.264 decoder is available then the arriving stream needs to be converted into packets, suited for that hardware unit and uploaded to it. The decoding process is started over a proprietary API and usually acts as a black box. Some decoders only handle parts of the decompression procedure; others do the full work and offer an option for either directly displaying the content or sending it back into CPU memory. Hardware H.264 decoders are usually optimized to enable good video playback, but not specifically for low-latency.

Next we have a look at our approach on displaying streamed content. An important feature of modern mobile device GPUs (supporting OpenGL ES 1.0 or higher) is that they have native support for displaying ETC1 textures. Therefore once an ETC1 compressed image arrives at the client we can directly upload it to the graphics chip where decoding to RGB values happens. Given the fixed compression ratio of ETC1 of 1:6 for RGB data the required transfer to the graphics chip is lower compared to uploading uncompressed RGB or RGBA data as described in the CPU-based pipeline for H.264.

ETC1 does an image by image (intra-frame) compression instead of using information across multiple frames (inter-frame). Therefore even if there is a lot of motion between frames a robust image quality is guaranteed. The video codec MJPEG [21] also has this characteristic, but as it lacks hardware decompression support on mobile devices it is not suited as it still requires non-accelerated decompression and the more bandwidth-intensive upload of uncompressed RGB/RGBA pixels to the GPU. Nevertheless, when comparing the image quality of an intra-frame with an inter-frame approach (like H.264) at the same bit rate, the latter will usually be of higher quality. However, codecs with inter-frame compression usually have higher latency.

C. Software Setup

Here, we describe the software setup and the communication between the client and server to enable streamed, distributed rendering.

The Microsoft Windows 7, 64-bit servers are running our custom written HPC ray tracing software, partly accelerated by Intel Embree [22] and multi-threaded through Intel Cilk Plus [23]. The ray tracer can be given the task to only render certain regions of the complete image. The ray tracer hands over the image section to the streaming module of our framework. This module can compress image data into the ETC1 format by using the etcpak library [24] which we multi-threaded using Intel Cilk Plus. Compressed data can be sent to the client using TCP/IP, supported by libSDL_net 2.0 [25]. Further, the server listens on a socket for updates that the client sends.

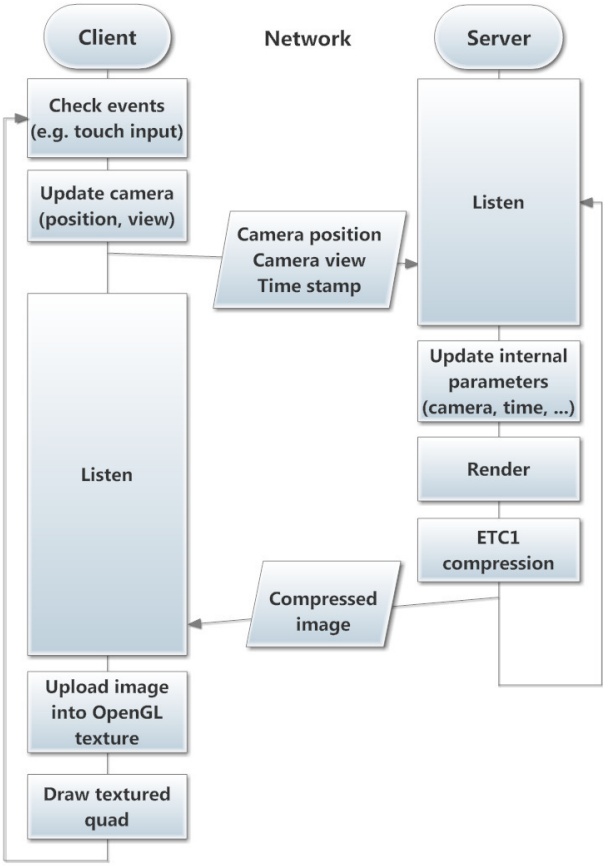

Fig. 2. *Tasks of the client and server architecture.*

The client runs Android 4.4.2 and executes an app that we wrote using libSDL 2.0 [25], libSDL_net 2.0 and OpenGL ES 2.0. All relevant logic has been implemented using the Android Native SDK (NDK r9b).

In the initialization phase the client informs the servers about the rendering resolution and which parts the render server should handle, parameters for loading content, and initial camera settings for rendering. Then the client will receive an image (or part of it) from the server. After the initialization steps the following procedure as described in Figure 2 will be processed every frame. The client checks for user input (like touch) and interprets this into changes in the camera setup (this step can also be done on the server side to stay application-independent). Those new settings, an unique time stamp and other application relevant data (total of 192 bytes) are then sent to the server. The server receives this

over the network and updates its internal states. An image (or part of it) is rendered, then compressed to ETC1 and sent to the client. There, the compressed image data is uploaded as an OpenGL ES texture. Next, a quad is drawn on the screen using that texture at the representing areas that have been assigned earlier to the rendering server.

Our used rendering algorithm, ray tracing, is known as an "embarrassingly parallel" problem [26] with very high scalability across the number of cores, CPUs and servers, because rendering the image can be split into smaller, independent tasks without extra effort. Therefore for the multi-server setup we naïvely split the image into four parts (one for each server), dividing the horizontal resolution by 4 and keeping the full vertical resolution. In order to achieve good scaling for a high number of servers we recommend instead using smaller tiles and to smartly schedule them over e.g. task stealing [27]. In addition to Figure 2 the client will now send one packet (192 bytes) to each server regarding the updates. The client will await the first part of the image from the first server, then upload it as OpenGL ES texture, then receive the second part of the image and so on.

As exemplary rendering content the "island" map from the game *Enemy Territory: Quake Wars*[1] is used.

It is our goal to make a very smooth experience by fully utilizing the display refresh rate of the smartphone (60 Hz). Given the properties of our hardware setup we choose to render at 1280×720 pixels instead of the full physical display resolution of the Nexus 5 (1920×1080 pixels) as it was in the current set-up not possible to transit higher resolutions with 60 Hz due to limitations of the data rate of a 1-antenna 802.11ac setup in current smartphones.

IV. Evaluation

Here, we evaluate our and other in-home streaming approaches in terms of data rate, latency, image quality and power consumption. We compare our implementation with Nvidia Shield[2] and Splashtop[3]. Steam's in-home streaming is currently still in beta and we encountered a high amount of frame drops during our tests, so we will not further compare it here.

A. Data Rate

We discuss the available data rate and the implications of it. The wireless networking standard 802.11ac allows a maximum data rate of 433 Mbit/s (for a 1-antenna setup). Hardware tests show an effective throughput of 310 Mbit/s (38.75 MB/s) for our router [28]. As our rendering resolution is 1280×720 pixels and has 8 bit per color channel this makes about 2.64 MB per image for uncompressed RGB data. Using ETC1 with the fixed compression ratio of 1:6 leads to 0.44 MB per image. Assuming the effective throughput this results in a maximum of about 88 frames per second (fps).

[1] id Software and Splash Damage

[2] Android 4.3, System Update 68

[3] Splashtop Personal 2.4.5.8 and Splashtop Streamer 2.5.0.1 on the Nexus 5

B. Latency

We have a detailed look at the latency of our and other streaming approaches.

The total latency from an user input to an update on the screen (motion-to-photons time) can have various causes of lag in an interactive streaming setup. The user input takes time to get recognized by the operating system of the client. Next, the client application needs to react on it. However, especially in a single-threaded application, the program might be busy doing other tasks like receiving image data. Afterwards, it takes time to transfer that input (or its interpretation) to the rendering server over network. There, the server can process the new data and start calculating the new image. Then compression takes place and the data is sent to the client, which might be delayed if the client is still busy drawing the previous frame. Once the image data is received, it will be uploaded to the GPU for displaying. Fixed refresh rates through VSync might add another delay before the frame can be shown. Some displays have input lag, which describes the time difference between sending the signal to the screen and seeing the actual content there. Additional delay happens when the client or server use multiple buffers for graphics. 3D application use double buffering, sometimes even triple buffering, to smooth the average frame rate. In our setup we are keeping the number of buffers as low as possible.

For the following measurements of our implementation we took the setup with four servers. As the distributed rendering approach does not work with the solutions we are comparing to, we modified the setup to use only one server and a very simple scene that achieves the same frame rate on a single machine as our four servers in the more complex rendering scenario. That way we have a fair comparison of the latency across the approaches. To get accurate motion-to-photons time we captured videos of user input and waiting for the update on the screen. Those videos are sampled at 480 frames per second using the Casio Exilim EX-ZR100 camera. In a video editing tool we analyzed the sequence of frames to calculate the total latency. Using our approach led to a motion-to-photons latency of 60 to 80 ms. On Nvidia Shield, which uses H.264 video streaming, we measured 120 to 140 ms. The Splashtop streaming solution, also relying on H.264, shows 330 to 360 ms of lag.

C. Image Quality

In this section we compare the image quality of our method with Splashtop, Nvidia Shield and with creating our own H.264 stream with different bit rate settings.

To analyze the difference in image quality we chose one image of a sequence in which a lot of camera movement is happening as shown in Figure 3. We quantify the image quality using the metrics of Peak Signal-to-Noise Ratio (PSNR) [29]

Fig. 3. *Left: Previous frame. Right: Frame for analysis with marked red area.*

and Structural Similarity (SSIM) [30] index, which takes human visual perception into account. For Nvidia Shield and Splashtop we were not able to test the distributed rendering setup, so we precalculated the ray traced frames offline and played them back from a single machine at the same speed that they would have been generated using four servers. That way a fair comparison of the image quality happens across all approaches. In Table I one can see that our approach has better image quality compared to Nvidia Shield, Splashtop and H.264 encoding at 5 Mbit/s. As expected, higher bit rate inter-frame encoding offers even higher image quality: going to 50 MBit/s using H.264 succeeds the quality delivered by ETC1. Using an even higher bit rate than 50 MBit/s during H.264 encoding does practically result in the same image quality for our setup.

D. Performance

Here, we report the rendering performance of our approach, the effective throughput and which tasks consume how much time.

We are able to achieve 60 frames per second at 1280×720 pixels. Given the ETC1 compression ratio of 1:6, this corresponds to an effective throughput of 210.93 Mbit/s (26.36 MB/s). The relevant components on the server side are rendering of the image region ($\sim$7 ms), network transfer ($\sim$3 ms) and ETC1 compression ($\sim$2 ms). On the client side network transfer ($\sim$12 ms) and OpenGL ES commands including swapping the display buffer and waiting on VSync ($\sim$4 ms) are the most time consuming tasks.

E. Battery Drain

Here we compare the battery drain of our approach, Splashtop and a locally rendered 3D application on the same smartphone and Nvidia Shield.

For the Nexus 5 we use the "CurrentWidget" app [31]. In our approach we observed an average battery usage of 850 mA, 605 mA for Splashtop and 874 mA for the locally rendered 3D game "Dead Trigger 2" [32] . The higher battery usage of our approach compared to Splashtop can be explained by the fact that we are handling much more data. The Nvidia Shield console uses different hardware; therefore the architectural difference has impact on the result and cannot be compared directly. Nevertheless, we report the number for completeness. The "CurrentWidget" app does not work on Nvidia Shield, so we measured the drop in percentage of the available battery power for an hour and by knowing the total battery capacity we got a value of 880 mA.

V. Enhanced Applications

In this section we have a look on various application scenarios that are enhanced by our high-quality, low latency in-home streaming solution. These can vary widely as the content of the displayed image is independent of the internals of the used compression, transportation and displaying method.

High-quality Gaming As there are already products evolving for in-home streaming for games, like Nvidia Shield and Steam's in-home streaming, this could potentially be an area of growth. The benefits are e.g. to play on the couch instead of sitting in front of a monitor or to play in another room where an older device is located that would not be able to render the game in the desired high quality. In fast-paced action games like first person shooters it is important to be able to react as fast as possible, therefore our reduced latency setup enriches the gaming experience. The commercially available solutions for in-home streaming of games are typically limited to using the rendering power of only one machine. Through the distributed rendering approach we potentially enable closer to photo-realism games by combining the rendering power of multiple machines.

Virtual Reality for Smartphones For virtual reality there are projects like FOV2GO [33] and Durovis Dive [34] (see Figure 4) that developed cases for smartphones with wide-angle lenses attached to it. Once this is strapped on the head of a user, mobile virtual reality can be experienced. For a good Quality of Experience high-quality stereo images need to be rendered that have pre-warped optical distortion compensation to cancel out spatial and chromatic distortions of the lenses. While this works good on desktop PCs [35], the performance

	Original	H.264 211 Mbit/s	H.264 50 Mbit/s	ETC1	H.264 5 Mbit/s	Splashtop	Nvidia Shield
PSNR	-	47.0	46.9	37.3	32.9	31.0	29.8
SSIM	1.0	0.997	0.997	0.978	0.877	0.861	0.779

TABLE I

PSNR and SSIM values for different codecs and platforms, exemplified by respective image sections as marked in Figure 3. Higher values are better. While with a high bit rate the image quality of H.264 surpasses ETC1, the later approach has significantly lower latency, which we show in section IV-B.

and quality that smartphones can achieve today is not very compelling for virtual reality. To achieve higher image quality, these applications have to switch from a local to a server-based rendering approach. As latency is an even more important issue in virtual reality, our latency-optimized approach is in particular suitable for this scenario. Solutions with a latency of 120 to 140 ms (Nvidia Shield) would lead to much more motion sickness compared to a latency of 60 to 80 ms. Nevertheless, optimizing virtual reality streaming applications for even lower latency might become more relevant in the future.

Fig. 4. *A mobile virtual reality platform that can be strapped on the head. In front of the case a smartphone is plugged in. Lenses bring the image into focus for the viewer. To compensate for optical distortions a high-quality, pre-warped stereoscopic image is used and streamed with low latency using our framework.*

Wearables: One of the emerging trends in the space of wearables are smartwatches. E.g. the Neptune Pine [36] is a fully independent Android device, equipped with its own CPU and GPU, 802.11n Wi-Fi and a 320x240 resolution display. Size, battery life and cost are limiting factors, so these devices are usually equipped with less capable processing units compared to other mobile devices. There is a chance that ETC1 streaming could unlock the full power of smartwatches - independent of their weak internal components. Further one could consider having the more powerful smartphone or tablet acting as a rendering server to feed the low-resolution smartwatch.

HPC and Big Data: A scenario where our streaming solution is also well suited for is real-time visualization of data-intensive computations like in the big data and HPC domain. Here, specialized applications either run analyses on huge amounts of data or computationally intensive calculations, typically relying on powerful back ends with a high amount of system memory. Typical application domains are health sciences, simulations in engineering, geographic information systems or marketing and business research. Being able to control, monitor and visualize these computations running on big server farms from small handheld devices is a very convenient benefit. Our solution, in comparison to other in-home streaming approaches, enhances graphics streaming for HPC applications as it supports a distributed scheme for rendering natively at high-quality and low latency. A testbed where we are currently integrating our streaming solution into is the molecular modeling and visualization toolkit BALL/BALLView [37], [38], see Figure 5. In BALLView,

running computationally demanding molecular dynamics simulations in combination with real-time ray tracing visualization on complex molecular data sets [39] requires a powerful compute server. Operating these experiments from a handheld device like a tablet is considered highly preferable.

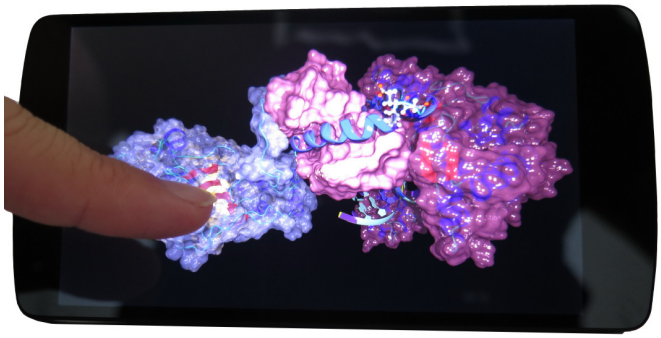

Fig. 5. *Molecular visualization from BALLView displayed on a mobile device.*

VI. Conclusion and Outlook

We conclude what we have demonstrated in this paper and give an outlook on future work.

We have shown a new approach to in-home streaming that fully leverages the latest development in wireless network standards and utilizes a hardware-accelerated intra-frame decompression scheme supported by modern mobile devices. The approach is well suited for streaming of interactive real-time applications and offers significantly higher image quality and half the latency in comparison to other recent solutions targeting this space.

Further optimizations like hardware encoders for ETC, switching to ETC2 [40] compression and using a 2×2 antenna network connection setup, as supported by IEEE 802.11ac, will lead to even higher image quality, faster performance and will enable 1080p at 60 fps.

Acknowledgements

Thanks to Timo Bolkart (MMCI, Saarland University) for his continued support during the writing of this paper. Thanks to Sven Woop for his support with Intel Embree. We thank Bradley Jackson for his feedback. Furthermore, the authors acknowledge financial support through the Intel Visual Computing Institute (IVCI) of Saarland University.

References

[1] "IEEE Standard 802.11ac-2013 (Amendment to IEEE Std 802.11-2012)," 12 2013.
[2] A. Chalmers and E. Reinhard, "Parallel and distributed photo-realistic rendering," in *Philosophy of Mind: Classical and Contemporary Readings. Oxford and*. University Press, 1998, pp. 608–633.
[3] Microsoft Corporation, "Remote Desktop Connection," http://windows.microsoft.com/en-us/windows/connect-using-remote-desktop-connection.
[4] T. Richardson, Q. Stafford-Fraser, K. Wood, and A. Hopper, "Virtual network computing," *IEEE Internet Computing*, vol. 2, no. 1, pp. 33–38, 1998. doi: 10.1109/4236.656066
[5] Gaikai Inc., "Gaikai," http://www.gaikai.com.
[6] OnLive Inc., "Onlive," http://www.onlive.com.
[7] C.-Y. Huang, C.-H. Hsu, Y.-C. Chang, and K.-T. Chen, "GamingAnywhere: An open cloud gaming system," 2013. doi: 10.1145/2483977.2483981 pp. 36–47.
[8] T. Wiegand, G. Sullivan, G. Bjøntegaard, and A. Luthra, "Overview of the H.264/AVC video coding standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7, pp. 560–576, 2003. doi: 10.1109/TCSVT.2003.815165
[9] Sony, "PlayStation 4," http://us.playstation.com.
[10] ——, "PlayStation Now," http://us.playstation.com/playstationnow.
[11] Valve Corporation, "Steam," http://store.steampowered.com.
[12] Nvidia Corporation, "Nvidia Shield," http://shield.nvidia.com.
[13] Splashtop Inc., "Splashtop personal," http://www.splashtop.com.
[14] Wi-Fi Alliance, "Miracast," http://www.wi-fi.org/wi-fi-certified-miracast\%E2\%84\%A2.
[15] Intel Corporation, "Intel wireless display," https://www-ssl.intel.com/content/www/us/en/architecture-and-technology/intel-wireless-display.html.
[16] A. Löffler, L. Pica, H. Hoffmann, and P. Slusallek, "Networked displays for VR applications: Display as a Service (DaaS)," in *Virtual Environments 2012: Proceedings of Joint Virtual Reality Conference of ICAT, EuroVR and EGVE (JVRC)*, 10 2012. doi: 10.2312/EGVE/JVRC12/037-044
[17] I. Nave, H. David, A. Shani, Y. Tzruya, A. Laikari, P. Eisen, and P. Fechteler, "Games@Large graphics streaming architecture," 2008. doi: 10.1109/ISCE.2008.4559473
[18] J. Ström and T. Akenine-Möller, "ipackman: High-quality, low-complexity texture compression for mobile phones," vol. 2005, 2005. doi: 10.1145/1071866.1071877 pp. 63–70.
[19] FFmpeg project, "FFmpeg," http://www.ffmpeg.org.
[20] "YUV420," http://www.fourcc.org/yuv.php\%23IYUV.
[21] D. Vo and T. Nguyen, "Quality enhancement for motion JPEG using temporal redundancies," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 18, no. 5, pp. 609–619, 2008. doi: 10.1109/TCSVT.2008.918807
[22] Woop, S. and Benthin, C. and Wald, I., "Embree ray tracing kernels for the Intel Xeon and Intel Xeon Phi architectures," http://embree.github.io/data/embree-siggraph-2013-final.pdf.
[23] Madhusood, A., "Best practices for using Intel Cilk Plus," Intel Corporation, White Paper, 7 2013, http://software.intel.com/sites/default/files/article/402486/intel-cilk-plus-white-paper.pdf.
[24] Taudul, B., "etcpak 0.2.1: The fastest ETC compressor on the planet," https://bitbucket.org/wolfpld/etcpak.
[25] "Simple DirectMedia Layer," http://libsdl.org.
[26] G. Fox, R. Williams, and G. Messina, *Parallel Computing works!*, 1st ed. Morgan Kaufmann, 1994.
[27] J. Singh, A. Gupta, and M. Levoy, "Parallel visualization algorithms: performance and architectural implications," *Computer*, vol. 27, no. 7, pp. 45–55, 1994. doi: 10.1109/2.299410
[28] E. Ahlers, "Rasante Datenjongleure," *c't Magazin*, vol. 1, pp. 80–89, 2014.
[29] Y. Wang, J. Ostermann, and Y. Zhang, *Video Processing and Communications*. Prentice Hall, 2002, p. 29.
[30] Z. Wang, L. Lu, and A. Bovik, "Video quality assessment based on structural distortion measurement," *Signal Processing: Image Communication*, vol. 19, no. 2, pp. 121–132, 2004. doi: 10.1016/S0923-5965(03)00076-6
[31] RmDroider, "CurrentWidget: Battery Monitor," http://code.google.com/p/currentwidget/.
[32] MADFINGER Games, "Dead Trigger 2," https://play.google.com/store/apps/details?id=com.madfingergames.deadtrigger2.
[33] "FOV2GO," http://projects.ict.usc.edu/mxr/diy/fov2go/.
[34] Shoogee GmbH & Co KG, "Durovis Dive," http://www.durovis.com/index.html.
[35] D. Pohl, G. Johnson, and T. Bolkart, "Improved Pre-Warping for Wide Angle, Head Mounted Displays," 2013. doi: 10.1145/2503713.2503752 pp. 259–262.
[36] Neptune Computer Inc., "Neptune Pine," http://www.neptunepine.com.

[37] A. Hildebrandt, A. Dehof, A. Rurainski, A. Bertsch, M. Schumann, N. Toussaint, A. Moll, D. Stockel, S. Nickels, S. Mueller, and O. Lenhof, H.-P.and Kohlbacher, "BALL - Biochemical Algorithms Library 1.3," *BMC Bioinformatics*, vol. 11, no. 1, p. 531, 2010. doi: 10.1186/1471-2105-11-531

[38] A. Moll, A. Hildebrandt, H.-P. Lenhof, and K. O., "BALLView: a tool for research and education in molecular modeling." *Bioinformatics*, vol. 22, no. 3, pp. 365–366, 2006. doi: 10.1093/bioinformatics/bti818

[39] L. Marsalek, A. Dehof, I. Georgiev, H.-P. Lenhof, P. Slusallek, and A. Hildebrandt, "Real-time ray tracing of complex molecular scenes," in *Information Visualisation (IV), 2010 14th International Conference*. IEEE, 2010. doi: 10.1109/IV.2010.43 pp. 239–245.

[40] Ericsson Labs, "Ericsson texture compression tool etcpack v2.60: ETC2," https://labs.ericsson.com/research-topics/media-coding.