

The Next Generation of In-home Streaming: Light Fields, 5K, 10 GbE, and Foveated Compression

Daniel Pohl
Intel Corporation,
Saarland Informatics Campus,
Saarbruecken, Germany
daniel.pohl@intel.com

Daniel Jungmann
ArKaos S.A.
Chaussée de Waterloo 198
B-1640 Rhode-Saint-Genèse, Belgium
el.3d.source@gmail.com

Bartosz Taudul
Huuuge Games,
Mickiewicza 53,
Szczecin, Poland
wolf.pld@gmail.com

Richard Membarth
DFKI,
Saarland Informatics Campus,
Saarbruecken, Germany
richard.membarth@dfki.de

Harini Hariharan, Thorsten Herfet
Saarland University,
Saarland Informatics Campus,
Saarbruecken, Germany
{hariharan,herfet}@nt.uni-saarland.de

Oliver Grau
Intel Corporation,
Saarland Informatics Campus,
Saarbruecken, Germany
oliver.grau@intel.com

Abstract—Interacting with real-time rendered 3D content from powerful machines on smaller devices is becoming ubiquitous through commercial products that enable in-home streaming within the same local network. However, support for high resolution, low latency in-home streaming at high image quality is still a challenging problem. To enable this, we enhance an existing open source framework for in-home streaming. We add highly optimized DXT1 (DirectX Texture Compression) support for thin desktop and notebook clients. For rendered light fields, we improve the encoding algorithms for higher image quality. Within a 10 Gigabit Ethernet (10 GbE) network, we achieve streaming up to 5K resolution at 55 frames per second. Through new low-level algorithmic improvements, we increase the compression speed of ETC1 (Ericsson Texture Compression) by a factor of 5. We are the first to bring ETC2 compression to real-time speed, which increases the streamed image quality. Last, we reduce the required data rate by more than a factor of 2 through foveated compression with real-time eye tracking.

Index Terms—in-home streaming, ETC1, ETC2, DXT1, light fields.

I. INTRODUCTION

“IN-HOME STREAMING” refers to interacting with real-time content on a thin client that has been generated on a more powerful computing device. The user’s inputs are forwarded to the server, which processes these and sends back updated video to the client. “In-home” refers to a local network, either wired or wireless, but not over the Internet. Ideally, in-home streaming is transparent to the user, delivering the perception as if the interactively streamed application were running locally on the target device. To achieve this, latency between user inputs and screen updates needs to be lower than 100 ms [1] and the image quality needs to be high, free of noticeable artifacts. Comparing with the state of the art approaches, these requirements still leave room for significant improvements as we will show in this paper. Our contributions

are extending an open source in-home streaming approach [2] with the following features:

- Support for multiplexed rendered light field images
- Higher image quality through ETC2 support
- Optimizations for ETC1, ETC2 and DXT1 encoding
- Streaming up to 5K resolution using 10 GbE
- Foveated compression through real-time eye tracking

II. RELATED WORK

The idea of controlling one compute device from another has been around for a long time. Desktop-sharing apps like Microsoft Remote Desktop and VNC (Virtual Network Computing) [3] are used, but are only optimized for 2D content. Cloud gaming approaches like “PlayStation Now” focus on lower bandwidth and use H.264 [4] compression. Specific in-home streaming solutions, supporting 3D real-time rendered content provide an opportunity to deliver a high *Quality of Experience* without the latency of the Internet, and with higher available data rates in the network. We compare our approach with the most commonly known in-home streaming products, which use H.264 internally: SplashTop, NVidia Shield Android TV Box and Steam.

Current in-home streaming approaches are optimized towards sequences of regular 2D images, generated from rendering 3D real-time content. Auto-stereoscopic and light field displays are getting attention again, enabling a way of perceiving stereoscopic content without glasses [5]. To drive these displays, multiple views of the scene are rendered and multiplexed together into a 2D image. This can create high frequency content in the multiplexed image which does not correspond to high frequencies in the single views and hence can lead to artifacts when using classical image or video coding standards.

We present an encoding algorithm optimized for multiplexed images.

Our work is an extension to the open source in-home streaming approach from Pohl et al. [2]. They introduced a client/server model that allows to interact with real-time rendered content created from one or many machines and can be remote controlled over wireless IEEE 802.11ac [6] from mobile clients like smartphones. Compared to using H.264 compression like other commercially available solutions, it relies on using ETC1 [7] (Ericsson Texture Compression), which results in a system with a motion to photons latency of 60–80 ms, compared to NVidia Shield Portable at 120–140 ms [2]. As ETC1 has a fixed 1:6 compression ratio for RGB data, it requires a high data rate inside the network. We extend the framework to significantly reduce encoding times, stream efficiently to notebooks and desktop PCs and enable even higher image quality through ETC2 [8]. Furthermore, as ETC1 and ETC2 native decoding is not supported on all desktop/notebook GPUs (see Table I), we extend the framework to use DXT1 [9] for encoding with our new highly optimized routines.

Guenter et al. [10] introduced a foveated rendering approach, generating and blending together three images of different quality in real-time depending on the inputs from a desktop eye tracker. Instead of using the eye gaze for rendering optimizations, we introduce a foveated compression method to lower the required data rate for in-home streaming. Zund et al. [11] follow a similar approach, changing the amount of pixels used in certain regions of a video for compression based on automatically extracted saliency maps [12]. Ours is based purely on the real-time eye gaze of the user.

III. SYSTEM

A. Hardware Setup

We use two hardware setups. The first uses four workstations with Dual-CPU's (Intel Xeon E5-2699 v3, 2.3 GHz, 18 physical cores) with a NVidia GeForce 970. The workstations stream to a desktop PC with an Intel Core i5-6500 (3.2 GHz, 4 cores). As GPU we evaluate both the internal Intel HD 530 graphics and a GeForce 970, connected to the Dell UP2715K 5K monitor (5120×2880 pixels). This setup is depicted in Figure 1. The second setup uses one of the workstations, streaming to the desktop PC connected to either a self-built light field display (2560×1440 pixels) or the low latency monitor Asus MG278Q at the same resolution. All machines use 10 Gigabit Ethernet over the Intel Ethernet Network Adapter X540-T1, connected to the Netgear XS708E-100NES Switch.

B. Compression Setup

The open source framework [2] that our work extends, supports ETC1 for encoding. In addition to this, we add ETC2 support for higher image quality. Prior to our work, there have been no real-time ETC2 encoders suitable for in-home streaming. Native ETC1 and ETC2 decoding in texture units works very well on most smartphones and tablets. It has been available in the form of OpenGL ES extensions and was made

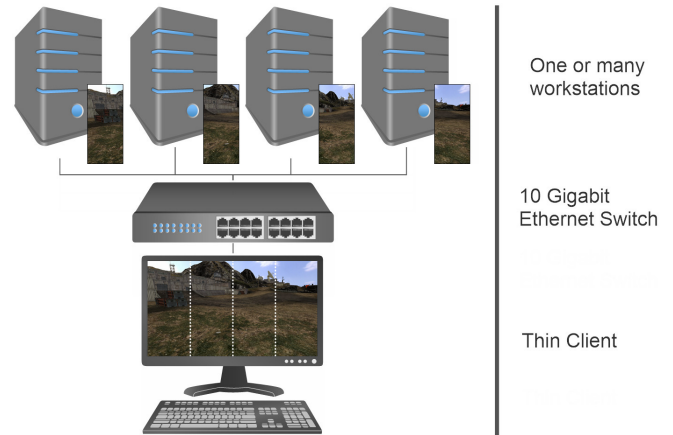


Figure 1. Rendering from multiple machines and streaming the content interactively to a thin client.

Table I
NATIVE DECODING OF COMPRESSED TEXTURES.

	DXT1	ETC1/2
Mobile GPU	×	✓
NVidia GPU, Maxwell GM20x	✓	×
AMD GPU, GCN 1.2	✓	×
Intel GPU, Broadwell	✓	✓

a requirement in the OpenGL ES 3.0 standard. On desktop and notebook GPUs the situation is different. Despite ETC2 support being standardized in OpenGL 4.3 (ETC2 is backwards compatible to ETC1), we found that most GPU vendors are only doing a slow software decompression in the driver, which is not satisfying for fast in-home streaming. However, on Intel HD graphics (5th generation Core Broadwell and later), we got fast native ETC1/2 displaying. To support more GPUs, we provide a highly optimized DXT1 compression routine based on FastDXT [13]. Just like ETC1 and ETC2, DXT1 uses a fixed compression ratio of 1:6 for RGB data. We highly optimized these encoders for Advanced Vector Extensions 2 (AVX2). As AVX2 expands most integer commands to 256 bits, we packed 16 colors, each 16 bit signed integer, into the 256 bit AVX2 vector. That way, we were able to process more data faster with the AVX2 instructions for multiply and mad. In addition, we exploited the symmetry of look up tables.

As in the original framework, optional lossless LZ4 compression¹ of compressed blocks can be applied. We keep this disabled except in one test case, where we mention it.

C. Light Field Setup

Our setup has a microlens sheet on top of an LG G3 mobile phone screen with 2560×1440 pixels, which allows to view a light field on a horizontal autostereoscopic display with N views (18 in our case). One lens covers N pixels and depending

¹<https://github.com/lz4>

on the direction the user looks on it, only one of these pixels will be seen per eye. The image is created in a way that N different views are rendered which are multiplexed together in one final image with the following method: the 1st pixel of the 1st view is copied into the 1st pixel of the final image. The 1st pixel of the 2nd view is copied into the 2nd pixel of the final image and so on. After N pixels, this pattern repeats with the 2nd pixel of the 1st image. If the resolution is not evenly dividable by the number of views, we fill the remaining pixels in the image with black.

When we compress to ETC1/2, a decision is made if the 4×4 pixel block should be split into two 4×2 or 2×4 sub-blocks for better encoding properties. If encoding requires no real-time, one could make a very careful analysis on which split gives the closest match to the original data. However, as we need fast performance, it could happen that after the repeating pattern of N pixels, a hard transition is in the center of the 4×4 block and the encoder decides to split this into 4×2 subblocks. Colors across that transition would get mixed together, even though in the individual views they are not related. Our new idea is to add optional flags to the function that encodes an ETC1/2 block and forcing it to a split decision in these cases. There are also light field display configurations where a lens covers pixels in both horizontal and vertical directions. In that case, our approach can also be used to force a 4×2 split, if appropriate. If in both directions a split happens within one block, we use the default algorithm.

D. Rendering Setup

For generating 3D real-time rendered content that the client can interact with, we use a self-written ray tracing platform partly accelerated by Intel's Embree [14]. As test scene we use "island" from the game Enemy Territory: Quake Wars. The rendering workload is distributed across multiple workstations and parallelized on each node. Each workstation sends its pixels back to the client after receiving input from the client. The renderer can create out of a 3D scene description both 2D images and multiplexed light field images. In the latter case, the rays are directly shot in a way that multiplexed images are created without individual views.

E. Foveated Compression

As DXT1, ETC1 and ETC2 share the same fixed compression ratio of 1:6 for RGB data, the required data rate can become a bottleneck. To facilitate data rate savings, we use a Tobii Pro X120 eye tracker to get the current eye gaze of the user and apply foveated compression depending on it. This could be combined with foveated rendering, but we prefer to stay independent of the image generation method. In more detail, we divide the image into nine rectangular regions. One region covers the foveated area, using the original resolution for encoding. The other eight regions are resized to 50% using a bilinear filter before encoding. We modify the network protocol to send information about the number of bytes that are about to be received, then information on the nine rectangular regions and then the compressed image parts. The client reacts

Table II
ENCODING TIMES FOR 2560×1440 PIXELS. LOWER IS BETTER.

DXT1 (ours)	0.5 ms
ETC1 (ours)	1.0 ms
ETC2 (ours)	1.3 ms
DXT1 (FastDXT)	1.6 ms
ETC1 (Pohl et al. [2])	5.3 ms
Intel Media SDK, Software H.264	20 ms
Intel Media SDK, Software MVC H.264	60 ms
FFmpeg, H.264	60 ms
Intel Media SDK, QuickSync MVC H.264	120 ms
Intel Media SDK, QuickSync H.265	600 ms

accordingly, uploading the nine parts into individual textures. The client combines them together and rescales them through OpenGL, if required.

IV. EVALUATION

A. Performance

We achieve 35–55 frames per second, depending on the rendering complexity of the view, using four workstations streaming interactively to a desktop PC at 5K (5120×2880 pixels) resolution with DXT1 compression. Considering a 20 ms frame, 45% of the time is spent for rendering, 5% copying internal buffers, 5% compressing to DXT1, 7% for sending data over TCP. The remaining 38% is spent on waiting for command updates from the client. Rendering one frame ahead using double buffering could fill that gap, but would increase latency. On the client side, 32% of the time is required for receiving TCP data, 12% for texture upload to the GPU and drawing. 56% for waiting on image data from the servers. Again, double buffering would help, but add latency. If the GPU has hardware support in the texture units for the compressed format, decoding does not consume any extra time when blitting data onto the screen which makes this ideal for low latency.



For driving the light field display with streamed content from one workstation, we achieve 50–70 frames per second.

In Table II, we compare the average encoding times for the workload of images with 2560×1440 pixels. For DXT1, ETC1 and ETC2 it is performance-agnostic if we compress individual views or multiplexed light field images. We use the H.264 profiles for the highest quality as we assume the availability of high data rate in the in-home setup.

B. 10 Gigabit Ethernet

Testing various network adapter driver options, we get additional speed ups in the 10 GbE environment. The default maximum transmission unit (MTU) is set to 1500 bytes for Ethernet, which leads to much packet overhead when sending big amounts of data. With the "Jumbo Frame" feature, we can increase the MTU size to 9014 bytes, increasing the frame rate in the 5K setup by five percent.

Table III
IMAGE QUALITY COMPARISON FOR 2D IMAGE AND LIGHT FIELD IMAGE. PSNR/SSIM: HIGHER IS BETTER.

	Original	H.265 Intel	H.264 Intel [†]	ETC2	ETC1	DXT1	Steam	Splashtop	NVidia Shield
2D image									
MBit/s	4752	248	290	792	792	792	55	4	12
PSNR	—	38.6	38.6	37.4	37.1	36.9	34.9	28.4	26.4
SSIM	1.0	0.984	0.978	0.982	0.980	0.972	0.936	0.744	0.644
									
Reconstructed view of light field image									
MBit/s	4752	183	566	792	792	792	55	4	12
PSNR	—	39.8	39.7	36.8	36.4	35.9	20.4	24.7	23.8
SSIM	1.0	0.997	0.996	0.988	0.986	0.975	0.725	0.782	0.733
									

[†] Using the Multi View Coding (MVC) profile for light field images.

C. Image Quality

We compare the image quality of a compressed 2D image and a reconstructed view of a compressed light field image through PSNR [15] and SSIM [16] in Table III. To highlight the difference, we show contrast-enhanced close up on the images. Wherever possible, we set the quality / bit rate to the highest modes for encoding. For Steam, we had to switch back to the “balanced” settings to avoid frame drops. As the metrics show, using high bit rate for H.264 [4] and H.265 [17] achieves in most cases higher image quality than ETC2, but only at higher encoding times (see Table II).

As described in Section III-C, it can happen in ETC1/2 encoding, that a wrong decision on splitting into sub-blocks is made during hard transitions of the light field image. This is exposed in the color bleeding in Figure 2, where we also show the impact on the recovered individual view and how our forced split fixes this.

D. Latency

We measure the motion to photons latency with a high speed camera. The starting time is from the video camera frame in which we first touch the mouse on the client for moving. The ending time is when we see pixel changes, sent from the server, displayed on the screen of the client. The results in Table IV. We explain the difference to Pohl et al. [2] from our wired vs. their wireless network.

E. Data Rate and Foveated Compression

While the latency of 40–60 ms of our pipeline is much better compared to the approaches using H.264, the required data rate is high. For our in-home streaming at 5120×2880 pixels at 55 frames per second, it is 3.1 GBit/s. For 2560×1440 pixels, 0.8 GBit/s. However, in an in-home network, there is usually a lot of data rate available. Wired approaches like

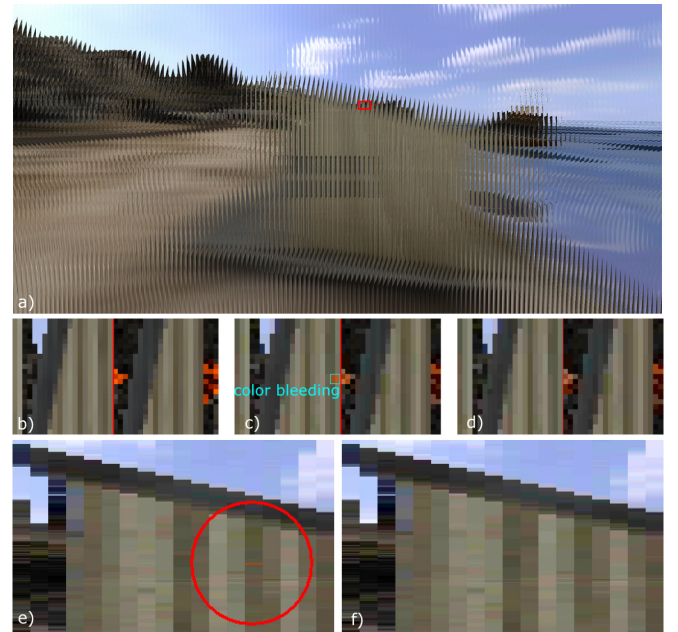


Figure 2. a) multiplexed light field image; b) uncompressed close-up across transitions (vertical red line added to mark the transition); c) compressed with original ETC1 encoder, orange color is leaking across the transition; d) compressed with our modified ETC1 encoder; e) impact on the reconstructed, stretched individual view: orange leak inside the house; f) leak fixed through forced split.

10 GbE and Gigabit Ethernet deliver up to 10 GBit/s and 1 GBit/s respectively. On the wireless side, data rates up to 3.5 GBit/s can be achieved for IEEE 802.11ac, 4x4 MIMO Wave 2. Looking ahead, 802.11ax will deliver 10–14 GBit/s and 802.11ay up to 100 GBit/s [18]. Nevertheless, we can reduce the required data rate using foveated compression. Depending on the size of the monitor, the distance from it or if an HMD

Table IV
LATENCY COMPARISON. LOWER IS BETTER.

Our approach (DXT1/ETC1/ETC2)	40–60 ms
Pohl et al. [2] (ETC1)	60–80 ms
NVidia Shield Android TV (H.264/H.265)	100–120 ms
Steam in-home streaming (H.264)	140–150 ms
Splashtop (H.264)	450–550 ms



Figure 3. Foveated compression. Top: the center area around the eye gaze is compressed in original resolution, while the eight other areas have been resized by 2x in each dimension before compression. Bottom: close-up on the green marked rectangle, showing on the left the high resolution image while on the right the area with reduced pixel size is shown, upscaled with bilinear filtering on the GPU.

is used instead of a regular 2D screen, the parameters for this approach can be varied. For our desktop setup, we use 40% of the horizontal resolution for the size of the squared, foveated area (see Figure 3). Then, the required data rate for either DXT1, ETC1 or ETC2 of one image at 2560×1440 pixels is reduced by a factor of 2.2 from 1.76 MB to 0.81 MB.

V. CONCLUSION

With our novel approaches, we bring in-home streaming to the next generation. We support 5K resolution, improve encoding algorithms for better image quality with light field rendered content and significantly improve the encoding times for ETC1. Furthermore, we design the first real-time ETC2 compression routine, enabling increased streamed image quality over ETC1. A larger variety of thin clients is supported with the option to use DXT1, for which we increase encoding performance by more than 2x. With eye tracking, we show that we can lower the required data rate by more than 2x. Our contributions are put back into the open source community under <https://github.com/ihsf>.

REFERENCES

[1] M. Claypool and K. Claypool, “Latency and player actions in online games”, *Comm. of the ACM*, vol. 49, no. 11, pp. 40–45, 2006.

[2] D. Pohl, B. Taudul, R. Membarth, S. Nickels, and O. Grau, “Advanced in-home streaming to mobile devices and wearables”, *IJCSA*, vol. 12, no. 2, 2015.

[3] 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.

[4] T. Wiegand, G. J. Sullivan, G. Bjøntegaard, and A. Luthra, “Overview of the H. 264/AVC video coding standard”, *Circuits and Systems for Video Technology*, vol. 13, no. 7, pp. 560–576, 2003.

[5] A. Travis, “Autostereoscopic displays”, in *Handbook of Visual Display Technology*. Springer, 2012, pp. 1861–1873.

[6] Wireless LAN Working Group, *IEEE Standard 802.11ac-2013 (Amendment to IEEE Std 802.11-2012)*, Dec. 2013.

[7] J. Ström and T. Akenine-Möller, “Ipackman: High-quality, low-complexity texture compression for mobile phones”, 2005, pp. 63–70. DOI: 10.1145/1071866.1071877.

[8] J. Ström and M. Pettersson, “ETC2: Texture compression using invalid combinations”, in *Graphics Hardware*, 2007, pp. 49–54.

[9] P. Brown, I. Stewart, N. Haemel, A. Pooley, A. Rasmus, and M. Shah, *OpenGL S3TC extension spec*, 2000.

[10] B. Guenter, M. Finch, S. Drucker, D. Tan, and J. Snyder, “Foveated 3D graphics”, *ACM Transactions on Graphics*, vol. 31, no. 6, p. 164, 2012.

[11] F. Zund, Y. Pritch, A. Sorkine-Hornung, S. Mangold, and T. Gross, “Content-aware compression using saliency-driven image retargeting”, in *ICIP*, 2013, pp. 1845–1849.

[12] C. Koch and S. Ullman, “Shifts in selective visual attention: Towards the underlying neural circuitry”, in *Matters of Intelligence*, Springer, 1987, pp. 115–141.

[13] L. Renambot, B. Jeong, and J. Leigh, “Real-time compression for high-resolution content”, *Proceedings of the Access Grid Retreat*, vol. 7, 2007.

[14] I. Wald, S. Woop, C. Benthin, G. S. Johnson, and M. Ernst, “Embree: A kernel framework for efficient cpu ray tracing”, *ACM Transactions on Graphics*, vol. 33, no. 4, 143:1–143:8, 2014.

[15] Y. Wang, J. Ostermann, and Y. Zhang, *Video Processing and Communications*. Prentice Hall, 2002.

[16] 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.

[17] G. J. Sullivan, J.-R. Ohm, W.-J. Han, and T. Wiegand, “Overview of the high efficiency video coding (HEVC) standard”, *Circuits and Systems for Video Technology*, vol. 22, no. 12, pp. 1649–1668, 2012.

[18] C. Taylor, “802.11ac Wave 2 with MU-MIMO: The next mainstream Wi-Fi standard”, 2015.