Tobii White Paper 20211201-FOV (2021)

A Lightweight Foveation Codec for VR

Alexey Bezugly[a], Dennis Rådell[b], Ralf Biedert[☆c]

[a] *alex8bezugly@gmail.com*
[b] *dennis.radell@tobii.com*
[c] *ralf.biedert@tobii.com*

Abstract

We present a foveated VR video codec, consisting of an image pre-reduction (warp) that can be combined with conventional video codecs such as H.265. Initial live testing between two machines several kilometers apart gave no visible artifacts at a compression ratio of 5:1. The warp's visibility impact was subsequently evaluated on various real-world test scenes, yielding a performance envelope with respect to acceptable latencies and compression ratios. The compression ratio was approximately confirmed, with a trade-off zone ranging from 4:1 to 7:1, depending on system latency. Ultimately, we argue that this approach can reasonably be used at least at a compression ratio of 5:1, leading to *additional* total bandwidth savings of 34 % over an assumed static warp with a compression ratio of 3:1 (up to 48 % at a compression ratio of 7:1 that worked for some users); is practicably implementable on current architectures and can improve visual quality over static foveation.

1. Introduction

Low latency, broadband connections made it feasible to generate high-resolution interactive content *in the cloud*.[1,2], or generally away from the device consuming the content, allowing for higher fidelities and reduced client resource consumption.

Yet, producing content remotely means the transmission of *many* pixels at low latencies, *many* times per second. VR is particularly demanding, due to its stringent latency requirements preventing nausea, and generally high resolution requirements required for *presence* [1]. For example, while early headsets displayed about 0.6 GB s^{-1} (Oculus Rift) to 1 GB s^{-1} raw pixel-byte data, more recent ones can easily reach 2.35 GB s^{-1} (HP Reverb G2) or even 5.56 GB s^{-1} (Pimax 8K). Over remote connections these numbers would be significantly higher by factors up to 3 when accounting for the increased resolution which compensates for lens warp and reprojection margins [2].

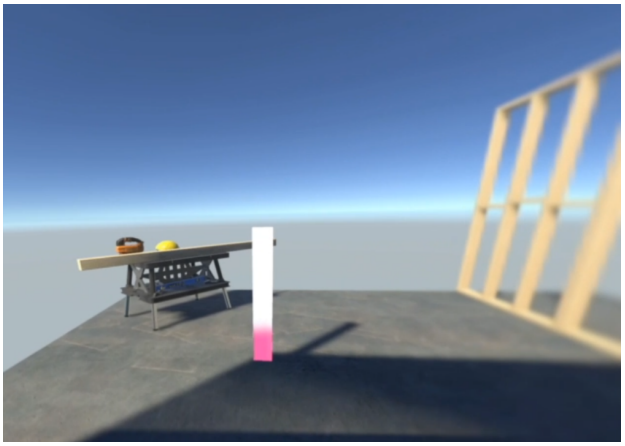

Figure 1. A screen capture of the user's view in the first run of our foveation codec over the internet. Several question arose from the prototype, some of which are covered in this report.

☆Corresponding author.
[1] https://stadia.google.com/
[2] https://www.nvidia.com/en-us/geforce-now/

Although modern compression codecs such as H.265 can lower them again to more manageable rates for consumer downlinks, bandwidth still remains an issue:

- For cloud providers, bandwidth-at-large is a cost factor. Wasting a few MB per second on a single consumer VR stream might be acceptable, wasting several 20 MB s^{-1} on 1000 concurrent users equal an excess bandwidth of 156 Gbit s^{-1}. Likewise it is a factor for metered consumer connections, such as 5G, where even half an hour of usage would accrue an additional 52 GB of traffic cost.
- Bandwidth affects latency. The less bandwidth used, the fewer packets have to be serialized, the earlier a frame can be presented, reducing nausea and improving the experience. At the data volumes involved, we would expect a 30 % data reduction to approximately produce a 30 % reduction of network latencies.
- Bandwidth affects quality. Better compression can deliver higher quality at equivalent bandwidths and latencies.

A promising alternative is to apply higher compression to parts of the image, based on lens and display properties, such as Axis-Aligned Distortion Transmission (AADT)[3] does. It compresses the image so that the center is kept at high quality, while the more blurry periphery is compressed at a higher rate, where the degraded optical quality of the lens helps hiding compression artifacts. However, as optical quality, pixel density, and field of view increase, this approach will be naturally limited in the degree it can compress without being noticeable.

An alternative approach is to exploit imperfections of the eye itself. Gaze contingent displays have been envisioned for over 50 years [3], and the use of eye tracking to save bandwidth having been around for at least 20 years, with the proposal of so-called *foveated (video) codecs*. These proposed codecs aim to compress a real time video stream watched by a single user, based on physiological properties of the eye [4]. Areas *currently looked at* are compressed at high fidelity, *areas not looked at* are compressed at low fidelity.

Such a foveation codec is split over two devices: The **client**, which includes the wearable display, eye tracking and other motion sensors and generally houses the *decoder*; and the **server** that runs and renders the application to be displayed, receives the motion and gaze signals from the client, renders a new frame, and sends the frame back to it, and generally houses the *encoder*. Encoding means converting a series of images, along with a point of regard on these images, into compressed network packages. Decoding is the reverse process of transforming these network packages back into viewable images.

Many codecs are conceivable, differing along dimensions such as:

Processing Overhead The amount of silicon and runtime required to execute encoder or decoder. This may include video processing units, shader units, memory bandwidth, implicitly, the thermal overhead of their use.

Visual Quality How closely the decoded image matches a user's perceptual limitations. Ideally, any decoded image is perceived by a viewer as identical to the original pre-encoded image. There are probably varying degrees of individual sensitivity.

Resiliency Refers to the codec's ability to embed and utilize redundancy, for the detection and mitigation of transmission errors (e.g., internet packet loss, local radio interference).

Compression Ratio Describes how many bytes have to be transmitted for every byte of raw image data.

Any codec operates within an envelope along these dimensions, with compression ratio being the prime factor traded against most others.

1.1. Our Work

With these considerations in mind, the rest of this paper is structured as follow: In Section 2 we give a brief overview of existing foveated compression schemes. In Section 3 we outline our foveation codec, consisting of a novel *warp*, i.e., a composable preprocessing step that reduces image dimensions while still maintaining certain visual properties, combined with an existing H.265 compression step, optimized for high compression levels while maintaining good visual transparency. Section 4 gives a technical overview how the codec was employed in both a live server-client remote test, as well as loopback simulation testbed for our subsequent study. In Section 5 we detail how we ran our experiments, presenting our results in Section 6 and discussing them in Section 7. Section 8 presents possible future work.

2. Related Work

Various foveation approaches have been proposed over the past two decades, although the meaning of *foveation*

[3] https://ocul.us/2LFKwCY

changed over time:

On early systems and pre-recorded video streams often a static, generalized foveation pattern was assumed[4] with use cases ranging from home videos to Unmanned Aerial Vehicles (UAVs) [5] [6]; and adaptive foveation, being *[...] means of providing dynamic quality of service (QoS); that is, dynamic control of the temporal and spatial parameters characterizing a video sequence* [7] via [8]. Such approaches were integrated with MPEG-compatible video, sampling foveal and peripheral areas differently via a rate adaptor, and the cost of maintaining conformance with existing MPEG codecs vs. deviating from them [8]. Other approaches used a multi-resolution pyramid codec, including motion estimation, with an applied blending function to compensate for visual artifacts [4]. Additionally, various wavelet compression schemes and metrics were used for foveation scalable video coding [9] [10]. Other works included *foveal visual quality metric, the foveal signal-to-noise ratio (FSNR) to determine the best compression and rate control parameters for a given target bit rate* [11], the exploitation of nonuniform spatial resolution for more efficient processing and compression [12], biologically motivated bottom-up models for attention on video compression [13], and a web-cam based *foveation server*, accounting for lag via various pre-encoded regions compressed at different levels, fused ad-hoc based on real time gaze data handling up to 660 ms [14]. For cloud-based foveated video game streaming, bandwidth savings of 50 % were reported [15].

Meng et al. employed kernel foveated rendering by rendering into log-polar coordinates, which is then transformed with its inverse function. Anti-aliasing is added before presenting on the display with *[...] 2.8X - 3.2X speedup in rendering on 4K UHD (2160p) displays with minimal perceptual loss of detail* [16].

On the VR side, latency limits and their impact on artifacts for foveated rendering have been studied, with reported acceptable upper bounds of 70 ms total system latency [17]; as well as visual angles below certain thresholds, but were highly dependent on scene and user [18]. Another approach based on HEVC with I-frame insertion, splitting the video into tiles assigned to foreground and background regions, reported 83 % reduction in bandwidth [19]. Foveated rendering and video compression has also been performed with deep learning, trained on natural videos, yielding impressive warp-like compression ratios of 10:1, however, with an immense processing overhead [20]. Lastly, as mentioned earlier, Axis-Aligned Distortion Transmission (AADT) as a compression scheme for use with Oculus Link has been utilized in real world applications.[5,6]

[4] That is, regions of interest where the user on an existing video stream *will likely look at* in a statistical sense, in contrast to the use of real-time eye tracking.

3. Foveation Codec

Transmitting video over a network requires a *codec*, i.e., an encoding and decoding scheme of pixel data to and from bytes. If a live eye tracking signal is involved in such a scheme we call it a *foveation codec* [4]. In this section we outline basic concepts and describe the foveation codec we used, from now on called *C1*.

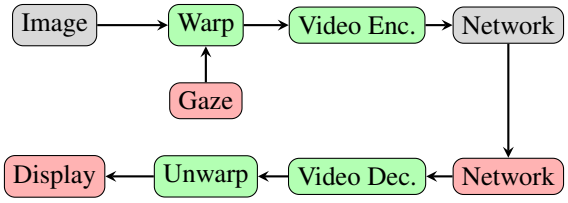

Figure 2. Simplified diagram of our architecture. On the server side an image is being created (also based on sensor data from the client, not depicted). In our codec C1 (green), the image is then warped and handed off to the GPU's video encoder. The result is serialized into network packages. On the client side (red) this process is reversed and the resulting image is being displayed.

C1 is outlined in Figure 2. It is a proof-of-concept codec for the purpose of allowing us to test the real-world feasibility of foveated image streaming. In essence, it consists of a warping-unwarping stage, and video compression-decompression stage. The warping stage receives one image per eye, with the VR headset's hidden area stenciled out, rendered from the VR runtime.

For a rendered texture I_r the warp algorithm creates a new, warped texture I_w which divides the texture into two parts. One part, called the **direct remap area**, located at the foveal region of the user, is not compressed by our codec. The part of the texture outside of the direct remap area is compressed in accordance to a **remapping function** which describes how to map pixel coordinates between textures. This remapping function was inspired by the physiological properties of the eye. The warped, compressed output is then transferred to the client which applies the inverse of remapping function to restore the image to its original size, and then presents it to the user. Figure 3 depicts an image in the original, warped, and

[5] https://ocul.us/3lsy9X4
[6] https://youtu.be/9gocUADwqo8

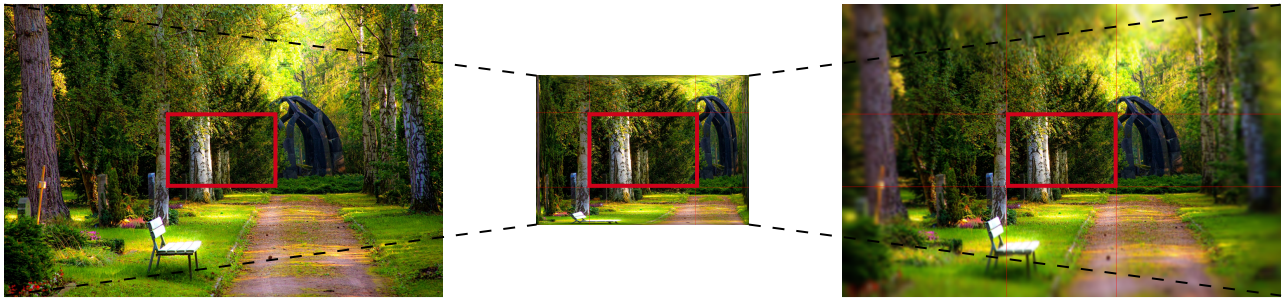

Figure 3. Warp application. The left image is the original, with the *direct remap area* marked in the center. The image is then compressed, resulting in the center image. The *direct remap area*, is left unaltered, while the peripheral area is heavily compressed. This results in a smaller texture, and effectively less data to be encoded and transported. The rightmost image shows the reconstructed result, which has visible blurring at the edges due to the compression, but the *direct remap area* is kept at original quality.

unwarped state.

3.1. Warping

For simplicity, we will describe the algorithm for a single, horizontal dimension on the original texture. In the actual algorithm the steps below are performed for both horizontal and vertical dimensions, for both left and right eye textures.

First, for the selected dimension, the original texture is divided into 3 regions. There is a central *foveal* region determined by the gaze point, transmitted with no compression, and two peripheral regions which are to be compressed, see Figure 4.

The length of the foveal region L_{foveal} is controlled by the *direct remap* setting as D multiplied by the length of the texture L_{tex} for the selected dimension, where *direct remap* is a value between 0 and $\frac{1}{\sqrt{C}}$.

$$L_{foveal} = D \cdot L_{tex} \qquad 0 \leq D \leq \frac{1}{\sqrt{C}} \tag{1}$$

If the position of the gaze point P_{foveal} is closer to the edge of the image than half of the foveal region length L_{foveal}, then the region will be shifted and bound within the image dimensions, but its length will not be adjusted.

$$P'_{foveal} = clamp(P_{foveal}, \frac{L_{foveal}}{2}, L_{tex} - \frac{L_{foveal}}{2}) \tag{2}$$

Next, the size of the compressed texture is determined via a *compression ratio* C setting, which is any number greater than or equal to 1. The *compression ratio* defines how many times the image is compressed i.e. the ratio of the total number of pixels in the source texture and the

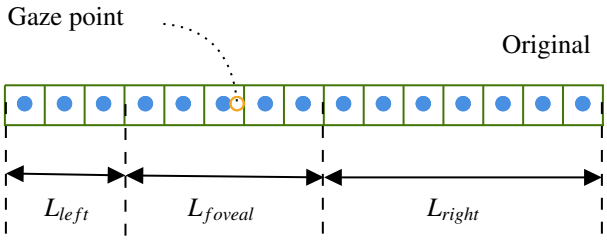

Figure 4. Warp regions on the original image, where points inside L_{left} and L_{right} are later compressed and points inside L_{foveal} are taken with no compression.

total number of pixels in the compressed texture. For the selected dimension, the length of the compressed texture will be effectively $\sqrt{C}$ times smaller than the length of the source texture.

When the eye is pointed towards the center of the VR display, each gaze vector will be perpendicular to the display plane. The number of pixels being viewed by the eye has a tangential relationship to the distance from the center of the display.[7] We define θ as the angle between the gaze vector and the vector pointed towards the current pixel. The further away from the center of the display the user is looking, the more pixels are viewed per angular change. We utilize this tangential relationship by remapping the coordinate equal to $tan\ \theta$ on the source image to the coordinate equal to $sin\ \theta$ on the compressed image.

To find the function for remapping a point, we want to find what a given point on the warped texture would equate to

[7] This mathematical relationship obviously breaks down for non-normal gaze angles. At this stage we assumed the practical error to be small enough not to cause issues. However, a more detailed analysis might be warranted, Section 8.

on the unwarped texture, refer to Figure 6.

First, we have a triangle, with the two catheti r and y, and the hypotenuse of $r + z$.

$$r^2 + y^2 = (r + z)^2 \tag{3}$$

Sine of the angle θ can be described in two different ways

$$\frac{x}{r} = \sin\theta \qquad \frac{y}{r+z} = \sin\theta \tag{4}$$

We can then solve for y. After simplifying, the resulting equation becomes

$$y = \pm\frac{r \cdot x}{\sqrt{r^2 - x^2}} \tag{5}$$

Since we are only interested in positive values, we thus define the remapping function F as:

$$F(x) = \frac{r \cdot x}{\sqrt{r^2 - x^2}} \tag{6}$$

where x is the pixel distance between the currently processed pixel on the compressed texture and the edge of the direct remap region on the compressed texture and r is the remapping radius.

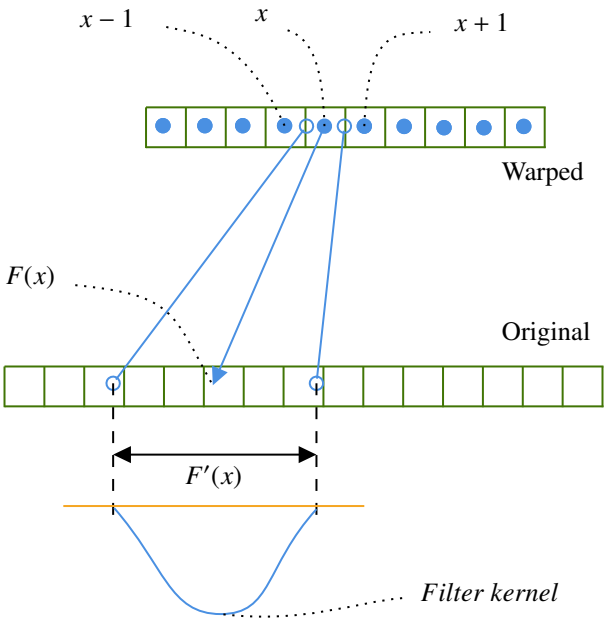

Figure 5. The warp remapping function and the Gaussian blur kernel.

The next step is to calculate the ratio between the lengths of the left and right peripheral regions in the compressed texture. As it can be seen from F, the remapping radius parameter r controls the compression ratio of F. For the given number of the source pixels, we will need less space in the compressed texture if r is low, and more space if r is high. In our case we already know the number of pixels available in the compressed texture. So the goal becomes to find a value of r such that the total number of pixels needed to compress the left and right peripheral regions sums up to the number of pixels available, assuming that we use approximately similar values of r to compress both regions. This is done by iterating though possible values of r until both peripheral regions can fit into their pre-allocated space. Visually, that means that the compression has a similar gradient towards the user's peripheral vision in all directions.

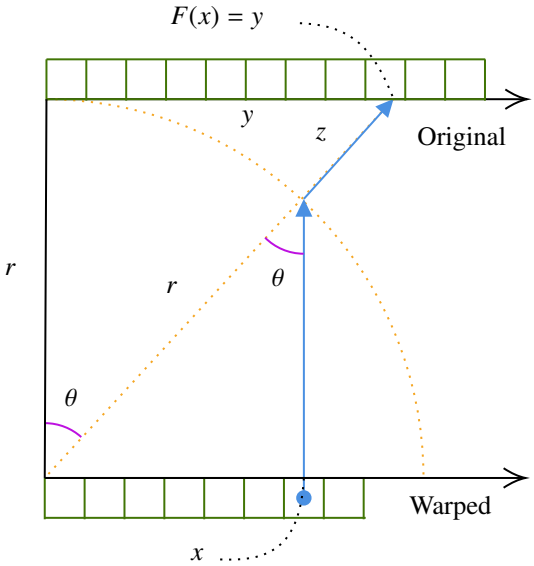

Figure 6. The remapping function used for compression. The coordinate on the warped texture x is being projected on the circle of radius r. Then it is projected to the tangent axis and results in the corresponding coordinate in the original image $F(x)$. The blue line represent the remapping logic.

Before the pixels from the source texture are remapped to the compressed texture, a pre-filtering step is executed. This step applies Gaussian blur to the source texture with variable kernel size, determined by the derivative of the remapping function F', see Figure 5.

This derivative describes how many pixels of the source texture are to be compressed into a single pixel of the compressed texture. The idea behind the Gaussian blur with the given derivative as the kernel size is to aggregate weighted information about all the source pixels resulting in a given compressed pixel. Visually, this step reduces aliasing artifacts. As an optimization, Gaussian blur values are only calculated for pixels later used in the remapping step, compare Figure 7.

Figure 7. Gaussian filtering is only calculated for the pixels that will be later used in the warp remapping. Note, you might need to zoom in *a lot* for your PDF viewer to accurately render this picture.

After Gaussian blur is done, the compressed texture is constructed. For each pixel in the resulting texture, we calculate the sampling coordinates on the source texture using the remapping function $F(x)$ and sample that texture using a bilinear sampler. A bilinear filter samples a grid of 2x2 pixels and linearly interpolates the values based on the distance from the sampling point to the centers of those pixels. The pixels in the *direct remap* region of the compressed texture are remapped exactly to the centers of the corresponding pixels on the source texture, meaning that there is no interpolation involved in this region.

The compressed texture is then being transferred to the target system where the unwarp algorithm is applied to restore the texture to the original dimensions.

3.2. Unwarping

To reconstruct the compressed texture to its original dimensions, the compressed texture is sampled using the inverse of the remapping function F^{-1}.

$$F^{-1}(x) = \frac{x \cdot r}{\sqrt{x^2 + r^2}} \tag{7}$$

where x is the pixel distance between the currently processed pixel and the edge of the direct remap region on the output texture and r is the remapping radius.

For each pixel on the resulting texture, the position which should be read from the source texture is calculated and sampled using a bilinear sampler. This step is combined with the post-filtering step which is done to further reduce aliasing artifacts that arise due to the discrete nature of the pixels in the compressed texture compared to what will be shown in the uncompressed texture. Instead of sampling just a single pixel on the compressed texture, neighboring pixels are sampled. These sampled pixels are weighed to form a Gaussian filter with a kernel size determined by the derivative of the remapping function $F'(x)$. The filter produces a smoother visual gradient that is less likely to be perceived as aliasing artifacts. The effects of this filter can be seen in Figure 9.

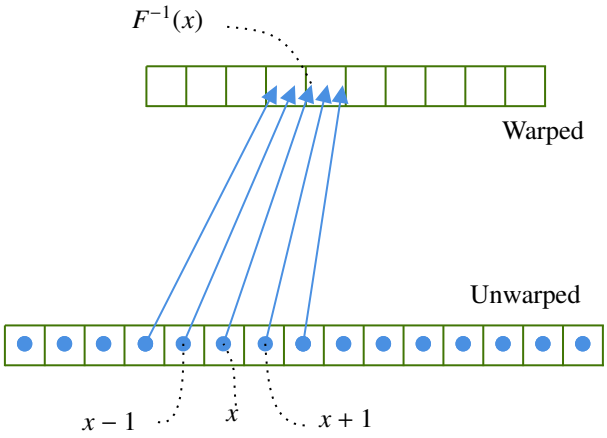

Figure 8. Unwarp remapping is an inverse of the warp remapping function.

Both warping and unwarping are executed as GPU shaders and were used for both the standalone server-client prototype, compare Section 4.1, as well as the OpenVR Hook experiment, Section 4.2.

3.3. H.265 Compression

In the server-client application, the warped output I_w was also fed into the GPU's H.265 video encoder, configured with the settings presented in Table 1. The resulting data stream was then transmitted to the client where it was decompressed, unwarped and displayed.

Figure 9. Post-filtering disabled (left side) vs. enabled (right side). Note how the image without the filter shows aliasing artifacts which are not as prominent in the filtered image.

Setting	Value
IFrame Period	1
Bitrate	50 Mbps
Format	`RGBA`
Compression	`Lossy`
Rate Control Mode	`CBR LOWDELAY HQ`

Table 1. H.265 codec settings used.

4. Experimental Architecture

4.1. Standalone Prototype

The codec from Section 3 was used in a live test. We created a client and server VR testbed, see Figure 1. At the end of regular per-eye VR rendering, but before lens distortion, the warp was applied by the server and the resulting I_w was video encoded as described. The server serialized the resulting stream of images into UDP packages to the client. The server was written as a standalone application in Unity, the client was a DirectX 11 application written in C++.

We used this testbed for multiple experiments, both over local networks and the internet. Most commonly an internet connection over a distance of about 8 km, where two PCs were connected with 250 Mbps bandwidth and approximately 2 ms ICMP latency, separated by 12 hops was used, compare Figure 10.

During these tests we made two observations. One, the foveated compression of the video stream generally *seemed to work*, in the sense that using the scene felt practically identical to using the scene locally when the sufficiently conservative parameters for warp and compression were chosen. Two, changes in the visual design of the scene seemingly had an impact on what compression parameters were acceptable.

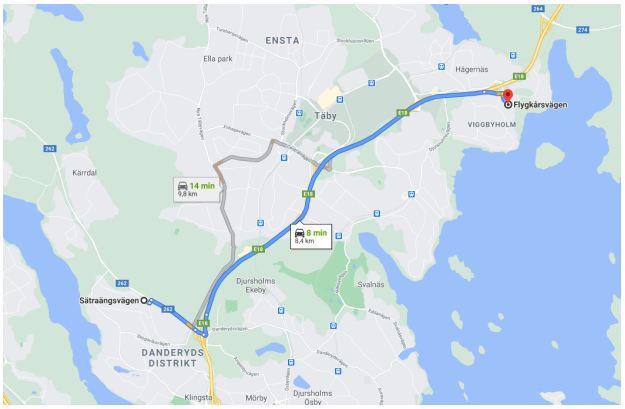

Figure 10. First internet live usage. Two machines were connected with 250 Mbps bandwidth and approximately 2 ms latency and ran the codec at varying settings.

4.2. OpenVR Hook

To better understand these mechanisms we extracted the warp and unwarp algorithms into a component we could apply to any frame buffer inside any application. We then created an application which enables intercepting eye textures submitted to OpenVR, and apply further processing such as our warping-unwarping logic. This served as the foundation for the subsequent experiments.

It should be noted that these hooked experiments were conducted only locally, with warp enabled, but networking and video compression disabled. [8] However, the ability to introduce artificial latency by showing older frames was added, which allowed us to simulate varying rendering, video compression and network transport situations in terms of their latency.

Overall, this approach allowed us to experiment with a variety of scenes and simulate various warp and latency scenarios, compare Figure 11.

5. Study Design

With the algorithm described in Section 3 and the setup described in Section 4.2, a pilot and subsequent study

[8] We disabled video compression because we already operated in the *high quality* compression regimen, and the video compression sometimes had unpredictable timing behavior which could impact the measured latency. Also, we were primarily interested in how much we could save *in addition* when composing with existing codecs. Network transmission was avoided since at this stage we were not interested in the codec's resiliency.

(a) (b) (c) (d)

Figure 11. Frames captured during 4 different pilot trials. Each participant used *VR Chat*, but was free to select any environment within it. This resulted in high inter-, but relatively low intra-trial variability of visuals. During the study most users converged on scenes with brightness, contrast and details similar to (c).

were performed.

We recruited in-house participants and asked them to join a *VR study*. We only considered users that had a VR device readily available at home, and, due to COVID-19, the experiment was conducted remotely via a live Slack or Teams session. They were then informed to share their screen, launch and log into VR Chat[9], enable *Display VR View* in *SteamVR* and select a chat room of their choice.

Users were allowed a few minutes of acclimatization time within the scene, until the actual experiment started: For 30 (pilot) or 34 (study) rounds we asked them to close their eyes while both compression and latency were changed via double-blind stratification along the axes *warp compression* and *latency*. We wanted the participants to see each combination twice, in order to verify that users gave the same response both times. Visiting every combination twice would have led to a long experiment duration, so we decided to instead only show half the combinations per users. These combinations were distributed in a checkerboard pattern. The two reference points, 1:1 compression and 0 frames of latency as well as 10:1 compression with 5 frames of latency, were visited 4 times each. Each combination was shown for about 15 s, where the user was allowed to interact with the scene. After the 15 s were over, the user was asked to rate the visual experience. Users were then asked to close their eyes in order to not be inclined to compare the current visual experience to the previous one.

Rating happened on an ordinal scale from (0) **no unusual artifacts were seen**, (1) **unusual artifacts were seen but they were not annoying or distracting**, to (2) **artifacts were seen and they were annoying or distracting**. In addition, they were given the opportunity to mention and describe any other effects present for a trial. Immediately before and after each experiment a 5 point gaze accuracy test was performed.

Meanwhile several data were recorded including the used latency and compression settings, user ratings, VR performance metrics per frame, VR headset used, which VRChat environment they used, and frame captures. Frame captures of four participants can be seen in Figure 11.

The pilot and the actual study differed in the range of combinations tested, and the scenes recommended to users. These adjustments were made because during pilot we realized that users had a tendency to pick dark, low-contrast scenes, which made the detection of artifacts harder, with some users reporting not having seen any artifacts during their trial. As a consequence we nudged users towards brighter scenes. We also included more aggressive compression settings and added the control point at a compression ratio of 10:1 with latency 5, while removing some points at low compression and latency to balance for experiment duration.

Latency was added artificially by delaying frames from being rendered and ranged from 0 to 5 frames. A latency of 0 frames means the system presents the frame as soon as it has been rendered. A latency of 5 frames means that when a new frame has been rendered, it is being held back from being displayed for 5 frames. For the VR headsets used in this study, 5 frames of latency equates to roughly 55.5 ms. This is comparable to latencies measured in NVIDIA CloudXR streaming solution [21].

[9] https://vrchat.com

6. Results

The study was conducted with a total of 7 participants[10] joining the pilot and 10 joining the study, with some overlap. The majority of results were obtained from machines with the *latest*[11] software version available. Participants of the study used one of the devices listed in Table 2.

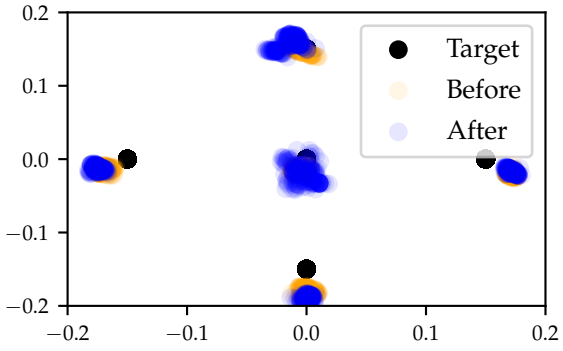

Figure 12. Sample of a user's pre- and post-test performed to ensure eye tracking data validity during experiment, depicting X and Y vector components.

#	Name	Resolution	Display	Tracker
7	Tobii DevKit	2160*x*1200	90 Hz	120 Hz
3	HTC VPE	2880*x*1600	90 Hz	120 Hz

Table 2. Devices used for the experiment.

6.1. Pilot

Out of the 7 sessions recorded 6 were deemed sufficiently high quality to be admitted for evaluation. Specifically, post-calibration matching pre-calibration data, compare Figure 12. One user's post-calibration data was lost due to operator error but was, in absence of any other problem indicators, included since the pre-calibration data was acceptable; another user with a highly inaccurate calibration was excluded.

Qualitative feedback collected throughout the experiment produced several interesting points:

- On high compression ratios and latencies a blurring of the periphery was described, along with *tunnel vision* effects.
- Likewise, after eyes opened on higher settings, sometimes a brief moment of blur was described.
- Aliasing was reported on all settings, in all scenes.
- Sometimes flickering in both periphery and fovea were described, also mentioned as *vibrations*, differing from aliasing and described as being either low frequency contrast changes; or high frequency *precision noise*.
- A *highlighting* on text was mentioned, when saccades occurred from anywhere onto text, resulting in perceived *brightness changes*.

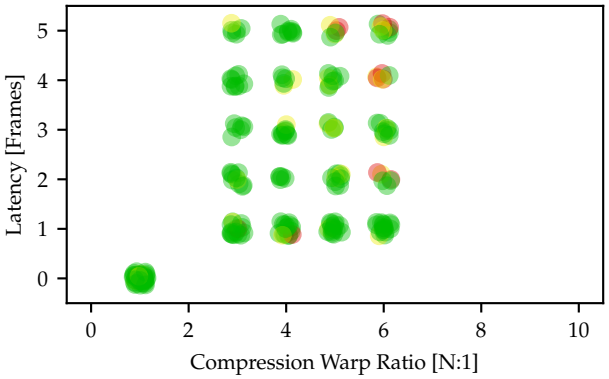

Figure 13. Aggregated user feedback over all settings and users for the pilot. Green dots indicate *No artifacts seen*, yellow dots *some artifacts seen but not annoying*, while red dots *annoying artifacts seen*. Jitter was added for better readability.

An overview of quantitative pilot results are depicted in Figure 13. There was a general, and somewhat unexpected, tendency towards *no artifacts* answers. Since, as mentioned previously, we could not decide whether this positivity bias was caused by a lack of attention or a limited scope of used settings further control points were added, as described in Section 5, leading to the follow-up study.

6.2. Follow-Up Study

All of the 10 sessions we recorded were admitted into the main evaluation. Qualitative feedback was similar to the pilot, although most notably all participants now clearly voiced the presence of artifacts at high compression and high latencies. Also, the extreme compression and latency combinations seemingly helped participants to identify the type of artifacts to look for, and ignore others (such as aliasing) that were present regardless. The aggregated quantitative user feedback is depicted in Figure 14.

The time to execute the warp and unwarp on the GPU was recorded as a part of the experiment. This can be seen

[10] Including the authors of this study; which we deemed appropriate given the experiment's randomized, double-blind nature.

[11] SteamVR version 1.15.12, VR Chat 2020.4.2, build 1016, Windows 10. However, individual machines might have deviated from this.

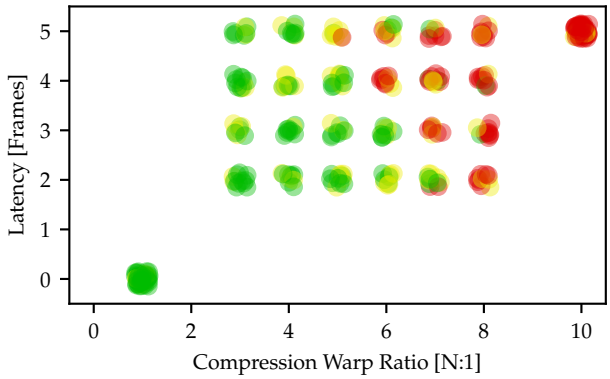

Figure 14. Aggregated feedback for the main study with same color keying as Figure 13. Notice the extended compression range and addition of a second reference point.

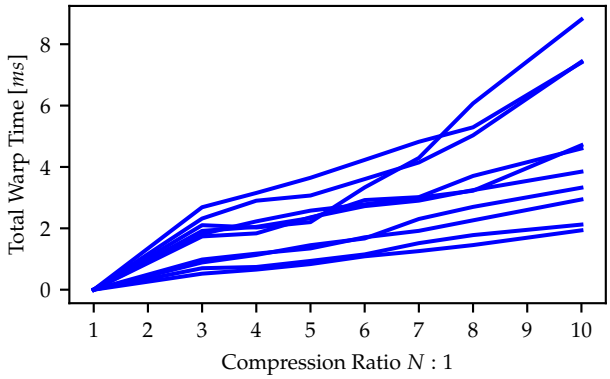

Figure 15. Average of $t_{warp} + t_{unwarp}$, each line representing one user. The graph reflects the total added *roundtrip* time for a frame at a given compression ratio. About half of that time is spent on the server, the other half on the client.

in Figure 15, which shows how long it took to execute for every user on different presets. Depending on the hardware on the computer running the experiment, as well as the resolution of the headset, there is some difference in the time to execute for different users. As can be observed from the unwarp algorithm description, the execution time for the unwarp step scales linearly with the number of pixels. The warp algorithm includes a Gaussian filter with a variable kernel size which does not scale linearly with the number of pixels and can be computationally heavy at high compression ratios.

The number of frames which were not completed within the 11.1 ms (the refresh rate of the VR display used), so called ”dropped frames” were also gathered in the study. A dropped frame means that the previous frame will be displayed again, only that it is reprojected to the user’s current head pose. Figure 16 shows that at higher

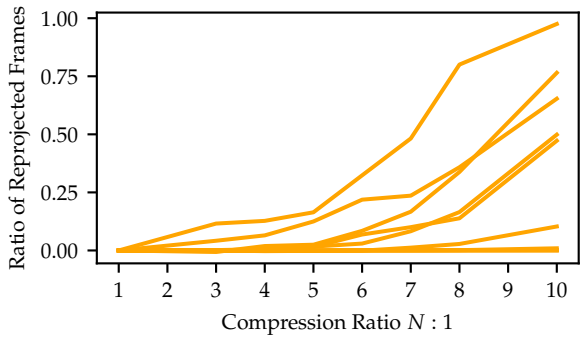

Figure 16. The ratio of reprojected frames per compression level, each line representing one user.

compression levels, and particularly for some users, the ratio of dropped frames to presented frames was more than half. This means that the latency measurements seen in Figure 14 could potentially be showing results which represent a higher latency than what was intended. For example, if a participant running at a preset with a configured 5 frames of latency were to drop a frame, the previous frame would be shown. Effectively, the user would then have a total latency of 6 frames.

7. Discussion

We presented a warp, combinable with existing video encoders such as H.265, and tested them both live and via the study. Depending on the level of compression applied, the warp and unwarp together took anywhere between 1 ms up to 9 ms. Given acceptable compression ratios between 5:1 and 7:1, practical numbers will likely be in the 2 ms total range, with GPUs rather matching the high-end variants such as RTX 2080 Ti utilized in our study, in particular in cloud scenarios [21]. As can be seen in Figure 16, the fact that many users saw a high ratio of dropped frames, especially at higher compression levels could indicate that the results are skewed negatively, since a dropped frame means that one additional frame of latency is added.

According to latency tests run on a CloudXR system over a local WiFi connection, latencies of 81.6 ms were reported as total system latency, with a standard deviation of 3.3 ms [21]. Using this as a baseline for real-world latencies would allow for compression ratios of up to 5:1 in our scenarios. Over a wired connection the compression ratio could possibly be higher due to the lower latency of

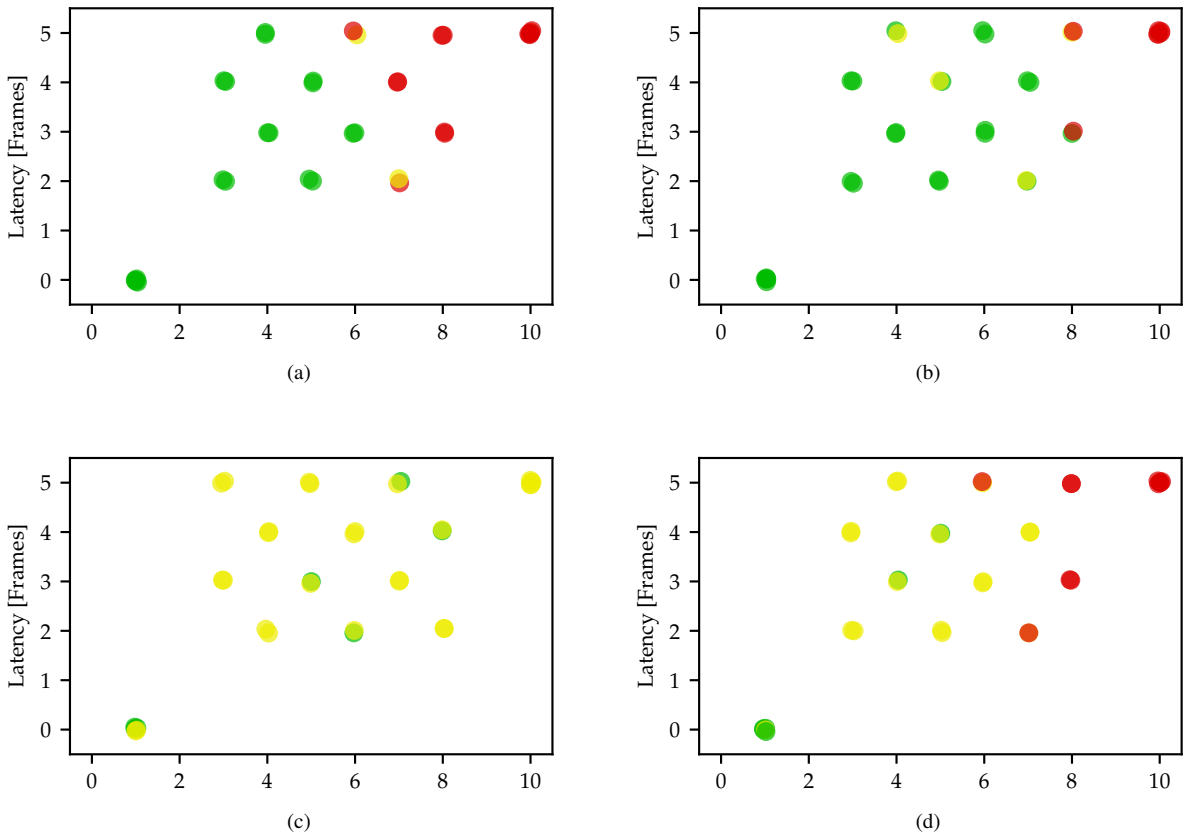

Figure 17. Individual answers for 4 different users. A range of behavior can be observed, most users (e.g., a, b, d) could clearly separate problematic settings from unproblematic settings, while other users (c) were seemingly unable to identify any issues. Also observe the difference of scale used between user (a), primarily defaulting to not seeing issues, vs. (d), primarily seeing some. Average frame drop ratios were (a) 0.22, (b) 0.11, (c) 0.19, (d) 0.00.

the connection, as well as higher bandwidth available.

In the qualitative feedback participants described a few different types of artifacts. Some mentioned that text seemed to become highlighted when performing saccades. This is likely caused by the blurring effects which happen in the periphery when warping. The blurring creates contrast changes which may become perceivable at high latencies. Other participants described ”vibrations” in both the fovea and periphery, which could be due to gaze precision errors due to insufficient signal stabilization. Many participants described aliasing issues visible in the scene. It remains somewhat unclear to what degree a particular aliasing instance may have been caused or amplified by the warp, or was caused by the user looking at a particularly difficult part of a scene by the time the comment was made. Likewise, the latency setting played a role around blinks and the opening of eyes, but it remained somewhat unclear if that was caused by biological factors, or due to the intrinsic latency during gaze measurement.

We believe it is equally likely that we underestimate real-world warp perception as that we overestimate it. While the overall exposure time in a scene was relatively short, users were actively looking for artifacts and we tested on a wide variety of settings, testing multiple settings in quick succession. User perception might also vary with continued usage, in particular at *contested* settings such as a compression ratio of 7:1 at latency 1 as well as a compression ratio of 6:1 at latency 4.

Using *dynamic*, eye tracking based foveated compression may produce higher compression and result in a noticeable reduction in bitrate needed to deliver same visual quality compared to Axis-Aligned Distortion Transmission (AADT) and other static foveation techniques.
To calculate bandwidth savings made possible by reducing the encoding resolution, we need to find a relationship between bitrate and resolution of a video stream given a specific target video quality. Netflix utilizes a metric

called *Video Multi-Method Assessment Fusion (VMAF)*, a perceptual metric for measuring video quality [22]. We used this VMAF to build a dependency graph showing how much bandwidth is needed for a given video quality (VMAF value) at different resolutions when using NVIDIA's H.265 encoder, see Figure 18; a similar relationship was discussed in [23]. Assuming the video quality at the foveal area will remain unchanged when doing foveated warp, we can use this graph to calculate possible bitrate savings for different compression ratios.

Networking and encoding/decoding latencies of 43.3 ms have been measured over WiFi, which translates into 4 frames of latency in our study. Looking at the results of the study at 4 frames of latency, see Figure 14, users either did not notice artifacts, or found them acceptable, at a compression ratio of 5:1. In a network with lower jitter, such as a cabled connection between the server and client, compression ratios of up to 7:1 could possibly be used as they were seen as acceptable by most of the participants. Compared to existing static warp solutions with an assumed 3:1 compression ratio combined with a lossless H.265 video compression on a Vive Pro Eye, our warp could lead to an *additional* total bandwidth saving of 34 % at 5:1 compression. At 7:1 compression, which some but not all users found acceptable, it can save up to 48 %. These results were calculated at a VMAF value of 95.9 which corresponds to 300 Mbps bitrate for Vive Pro Eye recommended render target resolution of 4936x2740. As pixel resolution and lens quality will increase over time, the benefit of eye-tracked approaches over static solutions will increase in turn.

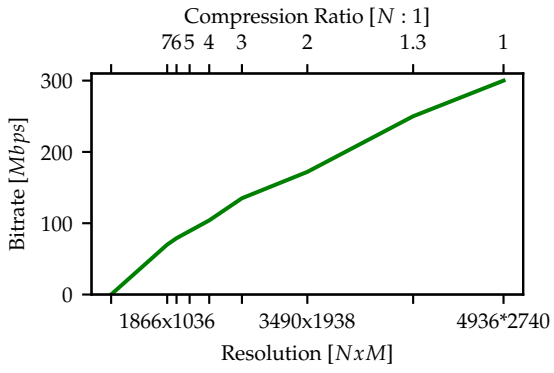

Figure 18. Bitrate needed for a lossy H.265 stream at 90 FPS at different resolutions (i.e compression ratios). The values were calculated for a VMAF value equal 95.9 which corresponds to 300 Mbps bitrate for a resolution of 4936x2740. This resolution is the recommended stereo render target size for Vive Pro Eye headset (the full display resolution plus reprojection margins and lens distortion compensation).

8. Future work

Several avenues are worth exploring further:

In order to minimize gaze latency, we did not filter gaze data, which can lead to artifacts due to precision errors in the signal. The relationship between these dimensions would be worthwhile to explore in real world applications, in particular when eye tracking frequency is higher than the display refresh rate.

Also, our warp was tested in conjunction with the currently industry standard H.265 codec, which seemed to work reasonably well for our purposes. However, we believe better compression ratios, reduced artifacts and an overall better experience can be achieved by better adapting both warp and algorithm for each other; similar to the early works [7] [8]. Likewise, it could be worthwhile repeating the experiments with a larger sample size to produce quantitative statements with low enough significance (e.g., non-inferiority tests along both dimensions) where the boundary of *no perceptual differences* lies.

The remapping function described in this method was selected due to the tangential placement of the displays in relation to the eyes. The function may also be a good fit to the distributions of rods and cones in the eye, but how well the function matches this distribution has to be further examined.

It should also be noted that the network encoding and video compression were not optimized for resiliency. In other words, one dropped package can lead to a bad frame. Any realistic use case will likely need to provide more redundancy for user comfort.

Lastly, although there were relatively clear boundaries along which artifacts were reported, some users were seemingly much more resilient to visual disturbances at higher compression levels. These differences might be explained by attention-based or biological factors, and it would be interesting to explore whether such differences could be determined a-priori to provide better automatic quality adjustments.

9. Acknowledgements

We would like to thank Ritchie Brannan for his shader work and Doug Eggert, Johan Hellqvist and Tim Biedert for their feedback.

References

[1] G. Riva, Virtual environments in clinical psychology, Psychotherapy: Theory, Research, Practice, Training 40 (2003) 68–76. doi:10.1037/0033-3204.40.1-2.68.

[2] A. Vlachos, Advanced VR rendering, Valve Advanced VR Rendering GDC 2015 Presentation.

[3] I. E. Sutherland, The ultimate display, Multimedia: From Wagner to virtual reality Proc. IFIP Congr (1965) 506–508.

[4] W. S. Geisler, J. S. Perry, A real-time foveated multiresolution system for low-bandwidth video communication, in: Proc. SPIE, Vol. 3299, 1998, pp. 294–305.

[5] A. Floren, A. Bovik, Foveated Image and Video Processing and Search, Vol. 4, 2014, pp. 349–401. doi:10.1016/B978-0-12-396501-1.00014-5.

[6] Z. Wang, A. Bovik, Foveated Image and Video Coding, 2006, pp. 431–458. doi:10.1201/9781420027822-14.

[7] R. T. Apteker, J. A. Fisher, V. Kisimov, H. Neishlos, Distributed multimedia: user perception and dynamic qos, in: High-Speed Networking and Multimedia Computing, Vol. 2188, International Society for Optics and Photonics, 1994, pp. 226–234.

[8] T. Reeves, J. A. Robinson, Adaptive foveation of mpeg video, in: Proceedings of the fourth ACM international conference on Multimedia, 1997, pp. 231–241.

[9] Z. Wang, A. C. Bovik, Embedded foveation image coding, IEEE Transactions on image processing 10 (10) (2001) 1397–1410.

[10] Z. Wang, L. Lu, A. Bovik, Foveation scalable video coding with automatic fixation selection 12 (2003) 243–254. doi:10.1109/TIP.2003.809015.

[11] S. Lee, M. S. Pattichis, A. C. Bovik, Foveated video compression with optimal rate control, IEEE Transactions on Image Processing 10 (7) (2001) 977–992.

[12] S. Lee, A. C. Bovik, Fast algorithms for foveated video processing, IEEE Transactions on Circuits and Systems for Video Technology 13 (2) (2003) 149–162.

[13] L. Itti, Automatic foveation for video compression using a neurobiological model of visual attention, IEEE transactions on image processing 13 (10) (2004) 1304–1318.

[14] J. Ryoo, K. Yun, D. Samaras, S. R. Das, G. Zelinsky, Design and evaluation of a foveated video streaming service for commodity client devices, in: Proceedings of the 7th International Conference on Multimedia Systems, 2016, pp. 1–11.

[15] G. Illahi, M. Siekkinen, E. Masala, Foveated video streaming for cloud gaming. arXiv:1706.04804.

[16] X. Meng, R. Du, M. Zwicker, A. Varshney, Kernel foveated rendering, Proceedings of the ACM on Computer Graphics and Interactive Techniques (I3D) 1 (5) (2018) 1–20. doi:10.1145/3203199.

[17] R. Albert, A. Patney, D. Luebke, J. Kim, Latency requirements for foveated rendering in virtual reality, ACM Transactions on Applied Perception (TAP) 14 (4) (2017) 1–13.

[18] C.-F. Hsu, A. Chen, C.-H. Hsu, C.-Y. Huang, C.-L. Lei, K.-T. Chen, Is foveated rendering perceivable in virtual reality? exploring the efficiency and consistency of quality assessment methods, in: Proceedings of the 25th ACM International Conference on Multimedia, MM '17, Association for Computing Machinery, New York, NY, USA, 2017, p. 55–63. doi:10.1145/3123266.3123434.

[19] P. Lungaro, R. Sjöberg, A. J. F. Valero, A. Mittal, K. Tollmar, Gaze-aware streaming solutions for the next generation of mobile vr experiences 24 (4) 1535–1544.

[20] A. S. Kaplanyan, A. Sochenov, T. Leimkühler, M. Okunev, T. Goodall, G. Rufo, Deepfovea: Neural reconstruction for foveated rendering and video compression using learned statistics of natural videos, ACM Transactions on Graphics (TOG) 38 (6) (2019) 1–13.

[21] David Weinstein, CloudXR, NVIDIA CloudXR GTC China 2019 Presentation.

[22] Z. Li, A. Aaron, I. Katsavounidis, A. Moorthy, M. Manohara, Toward a practical perceptual video quality metric.

[23] Alex Zambelli, H.265/hevc ratification and 4k video streaming.