FovOptix: Human Vision-Compatible Video Encoding and Adaptive Streaming in VR Cloud Gaming

Ahmad Alhilal*
aalhilal@connect.ust.hk
Hong Kong University of Science and Technology
Hong Kong

Ze Wu*
zwubn@connect.ust.hk
Hong Kong University of Science and Technology
Hong Kong

Yuk Hang Tsui*
yhtsui@connect.ust.hk
Hong Kong University of Science and Technology
Hong Kong

Pan Hui†
panhui@ust.hk
Hong Kong University of Science and Technology (Guangzhou)
China

Abstract

VR cloud gaming enables users to play high-end VR games on lightweight devices by offloading rendering tasks to cloud servers. Despite video compression, high-definition video streaming requires substantial data transfer rates. Foveated rendering (FR) and video encoding (FVE) leverage the non-uniform perception of the human visual system to reduce computing and bandwidth demand. They enhance visual quality in central gaze regions and reduce it in the periphery. However, bandwidth variation may hinder the provision of smooth VR gaming experiences. We present FovOptix, a system that combines FR with adaptive FVE to deliver video stream at a lower yet adaptive bitrate while not compromising the perceived video quality. FovOptix is based on a game-agnostic open-source to ensure reproducibility and compatibility with various games. We evaluate FovOptix against benchmarks using 5G mobile network traces. FovOptix achieves a latency reduction of 3% compared to the Google standard and a significant +100% reduction compared to other solutions. Additionally, it enhances the visual quality within the player's region of interest. Consequently, FovOptix attains the highest playability and gaming scores while minimizing the severity of motion sickness. FovOptix thus offers smooth and accessible VR cloud gaming for a wider range of players.

CCS Concepts

• **Computing methodologies** → **Virtual reality**; **Perception**; • **Human-centered computing** → **User centered design**; **Activity centered design**; *Ubiquitous and mobile computing design and evaluation methods*; • **Networks** → *Network performance analysis*; • **Computer systems organization** → Real-time system architecture.

*A. Alhilal, Z. Wu and Y.H. Tsui contributed equally to this work
†Pan Hui is also affiliated with the Hong Kong University of Science and Technology, Hong Kong SAR, and the University of Helsinki, Finland.

MMSys '24, April 15–18, 2024, Bari, Italy

ACM ISBN 979-8-4007-0412-3/24/04…$15.00
https://doi.org/10.1145/3625468.3647612

Keywords

VR Cloud Gaming, Human Vision System, Foveated Video Encoding, Adaptive Video Streaming.

ACM Reference Format:
Ahmad Alhilal, Ze Wu, Yuk Hang Tsui, and Pan Hui. 2024. FovOptix: Human Vision-Compatible Video Encoding and Adaptive Streaming in VR Cloud Gaming. In *ACM Multimedia Systems Conference 2024 (MMSys '24), April 15–18, 2024, Bari, Italy*. ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3625468.3647612

1 Introduction

Virtual reality (VR) cloud gaming enables users to play VR games on lightweight devices by offloading rendering tasks to the cloud [19]. VR cloud gaming enhances the accessibility and affordability of VR gaming by eliminating the need for players to buy expensive hardware or upgrade their devices. The processing and rendering of the VR game are handled by powerful servers in the cloud, which stream the game to the user's device. The players access the VR cloud gaming service via lightweight devices (VR headsets) that have limited resources (computing and memory). However, streaming video frames from the cloud to end users necessitates stable and high network bandwidth for optimal playability. While video compression reduces the bitrate in streaming the rendered graphics to the user's device, it is insufficient to maintain acceptable latency under unstable and low-capacity networks.

Commercial cloud gaming platforms like Google Stadia and Amazon Luna rely on WebRTC [4] for media streaming. WebRTC uses Google Congestion Control (GCC) [9] for bitrate selection which favors real-time transmission to video quality. Nebula [7] adapts the video rate to cope with bandwidth variations and applies frame-level redundancy to avoid visual distortions. While WebRTC and Nebula account for the underlying network conditions, they are designed for video streaming and mobile cloud gaming.

The visual acuity of the human eye is not uniform across the visual field. The fovea, which is a few degrees in diameter, has the highest acuity, while the peripheral retina has lower acuity [27]. This non-uniformity results from the distribution of photoreceptor cells in the retina and the way visual information is processed by the brain [8]. Prior works take advantage of the visual non-uniformity through foveated rendering (FR) and foveated video encoding (FVE). FR reduces the computational workload of rendering by producing

lower visual quality for peripheral regions, while FVE reduces the bitrate of streamed video through heavier compression of peripheral areas. Illahi et al. [17], Ryoo et al. [28], and Frieß et al. [11] apply FVE through multi-resolution foveated video coding which improves the image quality in the point of fixation and lowers the encoding quality in the periphery. Illahi et al. [18], abbreviated FR-FVE, combine FR with FVE and report preliminary findings of the combination. FR-FVE mitigates the bitrate overhead as compared to uniform video encoding over a simulation of a remote rendering scenario. However, these are designed to meet the user's perception of mobile cloud gaming, traditional video-on-demand streaming, or conferencing. They do not consider the unique characteristics of VR gaming, the variation, and instability in network conditions, or the impact of frame complexity on transmission bitrate.

Based on the aforementioned research studies and associated constraints, the following challenges need to be addressed: (*i*) What video streaming strategy can be employed to ensure a smooth experience in VR cloud gaming under unstable network conditions? (*ii*) How the video compression be achieved without compromising the quality of the visual experience? (*iii*) What approaches can be implemented to ensure fair evaluation and comparison with benchmarks in VR cloud gaming environments?. To tackle these challenges, we present FovOptix that jointly accounts for the non-uniform acuity of human vision and adapts to the variations in network conditions. We integrate FR with FVE to reduce the computation and the bandwidth demand, respectively, while providing a relatively higher video quality in the central field of view (FoV) region. In particular, FovOptix renders the foveal regions with higher resolution and encodes them with higher bitrate, while it renders the peripheral regions with lower resolution (downsampling) and encodes them with lower bitrate. To ensure game independence, we utilize ALVR base code [2] to integrate a virtual head-mounted display that interoperates with SteamVR [1]. This allows players to install any VR game. We also integrate state-of-the-art solutions into VR cloud gaming by implementing them over ALVR. Our contribution is summarized as follows:

- **Develop** FovOptix, the pioneering VR game streaming system that combines foveated video encoding (FVE) with adaptive bitrate based on underlying network conditions.
- **Design** FovOptix as game-agnostic to enable playing any VR game.
- **Implement** an algorithm that associates adaptive bitrate with FVE by transforming desired bitrates into controllable quality maps corresponding to human visual perception.
- **Integrate** benchmarks into VR cloud gaming platform by implementing them within an open source VR gaming platform.
- **Evaluate** FovOptix performance against the benchmarks using objective and subjective evaluation. FovOptix presents the lowest latency and enhanced visual quality in the user's region of interest, improving the playability and game scoring.

The remainder of the paper is structured as follows. In section 2, we summarize the related work. We discuss the methodology of FovOptix in section 3, and its detailed implementation in section 4. After presenting the experiment setup in 5, we present the results in section 6. We discuss the limitations and future improvements in section 7. Finally, we conclude our work in section 8.

2 Background and Related Work

2.1 Cloud Gaming.

Cloud gaming encounters challenges related to the variation in network performance and meeting high-quality experiences (interaction and immersion). GamingAnywhere [15] is an open source for a cloud gaming system that allows game players to enjoy cloud gaming on mobile phones. However, it relies on constant bitrate for streaming. Commercial cloud gaming platforms such as Google Stadia and Amazon Luna rely on WebRTC [4] for media streaming. WebRTC uses Google Congestion Control for bitrate selection and favors real-time transmission to video quality. Nebula [7] adapts the video rate to cope with bandwidth variations and applies frame-level redundancy to avoid visual distortions. Owing to the unique characteristics of VR gaming (display for each eye and proximity to the eyes), although the existing solutions such as GamingAnywhere, Nebula, and WebRTC adapt the streaming to the underlying network conditions, they fail to meet human visual perception. Xing Liu et al. [23] illustrate a significant correlation between video quality and motion sickness severity in VR. Because of this correlation, we render and encode the central FoV region, where users focus the most, with higher quality.

2.2 Foveation-based Cloud Gaming.

Illahi et al. [17] modify GamingAnyWhere to enable gaze tracking at the client, and encode video in a foveated fashion (FVE) at the server. They control the Quantization Parameters (QPs) at the encoder API level. FVE optimizes video quality by minimizing the total QP at the gaze location and increasing it gradually away from the gaze, resulting in higher quality in the foveal FoV and lower quality in the periphery. Ryoo et al. [28] design a similar video streaming service that exploits the non-uniformity of human visual acuity. They apply multi-resolution foveated video coding and use webcam-based gaze trackers for precise gaze feedback. Frieß et al. [11] track the viewing direction of multiple users using devices equipped with reflective markers, referred to as rigid bodies. The position and orientation of each rigid body are streamed. Accordingly, the server encodes the screen-captured frames by changing the quality of the macroblocks based on their distance to the foveated regions. Although Illahi et al., Ryoo et al., and Frieß et al. utilize the non-uniformity of human vision, they are designed to operate in mobile cloud gaming, traditional video-on-demand streaming, and conferencing, respectively. Additionally, they overlook network conditions and focus solely on bandwidth preservation.

2.3 Human vision system

The human visual system (HVS) partitions the FoV into multiple regions with varying degrees of proficiency in the perception of visual information. As shown in Figure 1, it is categorized into central, paracentral, near-peripheral, mid-peripheral, and far-peripheral regions. Additionally, there is a region known as the macular, which is situated between the paracentral and near-peripheral regions. Distinguishing details is weaker in the peripheral region[21], and distinguishing shapes is weaker outside the near-peripheral region[20]. The existing literature [30] has not reached a definitive conclusion regarding the eccentricity of each region. However, some studies indicate that the macular region is located at an eccentricity of 9° [29], while the near-peripheral region spans from 9°to 30° [30]. The Oculus Quest 2 provides a FoV with an average of 94° [13],

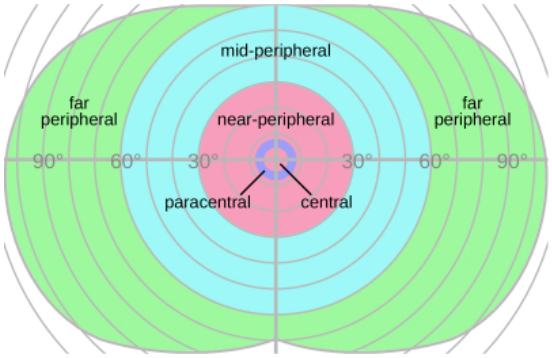


Figure 1: Angular Field of View (FoV) of the Human Eye [21]

which varies due to the user's interpupillary distance. FR assigns multiple resolutions across the video frame to leverage perceived quality variations in the FoV. FVE applies heavier compression in less important areas, such as the peripheral regions. Illahi et al. [18], abbreviated FR-FVE, report preliminary findings of the combination of FR and FVE. FR-FVE mitigates the bitrate overhead as compared to normal video encoding over a remote rendering simulation.

Our work focuses on improving the VR gaming experience by dynamically adjusting the video compression of virtual game scenes, taking into account both human visual perception and the quality of network connectivity, with a specific emphasis on addressing challenges posed by unstable network conditions. Moreover, we integrate representative state-of-the-art solutions to operate in VR Cloud gaming settings for evaluating them against FovOptix. In particular, the benchmarks include WebRTC [4], FVE [11, 17], integrated foveated rendering and FVE (FR-FVE) [18], bandwidth estimated-based video streaming (BW Probing), and ALVR [2].

3 Methodology

3.1 System Architecture

Figure 2 outlines the system architecture. The system includes the server to be installed on the cloud and the client application to be installed on the VR device. The server encompasses four primary components, virtual head-mounted device (VHMD), foveated rendering (FR), foveated video encoding (FVE), and QP Manager. The server creates a VHMD to replicate the physical HMD. The VHMD device receives and manages the physical pose and motion data from the client. The SteamVR, a game-agnostic platform, then retrieves this data from VHMD to render the game frame. The VHMD retrieves the game frame from the game engine, renders it, and then composites the left eye with the right eye frame into one frame (referred to as the game frame). The FR component re-renders the game frame to produce a multi-resolution frame (FFR frame), while the FVE component encodes the FFR frame based on the user's regions of interest (ROI) in the FoV. The encoded frame is transmitted to the client which receives the video packets, decodes them to recover the FR frame, applies reverse foveated rendering, and finally plays back the recovered frame. Simultaneously, the client monitors the player's physical pose and motion (i.e., hand controller input, head rotation, and viewing angle). The clients constantly transmit this data to the server to render and encode the next game frame.

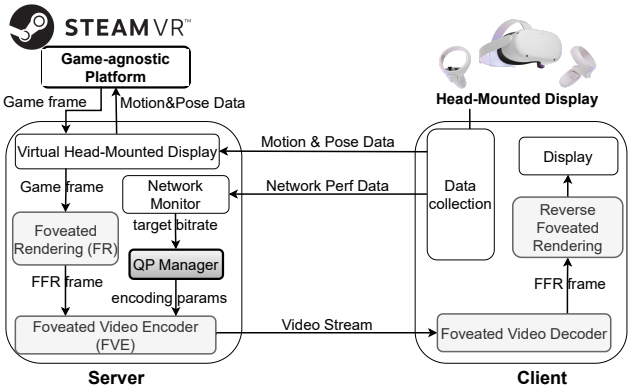


Figure 2: FovOptix's system overview.

3.2 Network Monitoring

The Network Monitor, shown in Figure 2, is responsible for maintaining the target bitrate to advise the QP manager, thus the encoding parameters.

3.2.1 ***Video Streaming Bitrate.*** The end-to-end bitrate of a server-client system is determined by the server-side bitrate and client-side bitrate. Thus, both the server and the client monitor the video streaming bitrate, denoted as $B_s(t)$ and $B_c(t)$, respectively. $B_s(t)$ is the sending bitrate, while $B_c(t)$ is the receiving bitrate. Later, we base the computation of the target bitrate on $B_c(t)$ when network congestion is detected (see Figure 3).

3.2.2 ***Frame Latency.*** This latency, also known as ***motion-to-photon (MTP) latency***, is the total input-to-display latency for each frame. It reflects the user-perceived latency that spans between the physical motion on the VR device and playing back the corresponding frame on the VR display [5]. This latency is used in our adaptive rate control model (QP Manager) to adjust the quantization parameters (QPs) according to the latency change.

3.2.3 ***Target Bitrate.*** This represents the ideal bitrate at which video frames should be encoded and sent to the client to maintain a certain level of video quality. This bitrate is dynamically adjusted during the video transmission based on the collected network parameters in response to the changing network conditions. As in WebRTC's delay-based congestion control, we compute the target bitrate based on the Arrival-time Filter, Over-use Detector, Remote Rate Controller, and Adaptive Threshold [9]. In contrast to WebRTC GCC which is based on RTP packets and RTCP feedback, we implement the delay-based controller on the server which utilizes the client's constant feedback. The feedback is transmitted as a payload on UDP packets, reporting the arrival time of frames and their motion-to-photon latency. The UDP packets offer frame-level feedback, distinguishing them from RTCP packets that provide packet-level feedback. Additionally, in contrast to the WebRTC GCC specification, which defines a group of packets as those created during a time interval of less than $5ms$, we define a ***group of packets*** as including all packets associated with a single frame. As a result, we re-define the equations in the Arrival Time Filter to estimate the queuing delay gradient $m(t_i)$.

Arrival Time Filter: This component is designed to estimate the delay gradient $m(t_i)$, the gradient of a line formed by the sequence of data points $(t_i, \Delta T_{delay}(f_i))$, where t_i is the arrival time of frame

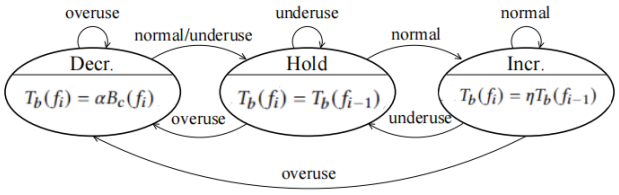


Figure 3: Modified rate controller finite state machine [9].

i and $\Delta T_{delay}(f_i)$ is the inter-delay time. $\Delta T_{delay}(f_i)$ is computed as the difference between the inter-arrival time $T_a(f_i)$ and the inter-departure time $T_d(f_i)$ of frame i as follows:

$$\begin{aligned} \Delta T_a(f_i) &= t_a(f_i) - t_a(f_{i-1}), \\ \Delta T_d(f_i) &= t_d(f_i) - t_d(f_{i-1}), \\ \Delta T_{delay}(f_i) &= \Delta T_a(f_i) - \Delta T_d(f_i). \end{aligned} \tag{1}$$

Where the inter-arrival time $\Delta T_a(f_i)$ is computed as the difference between the arrival time of the last packet of frame i, $t_a(f_i)$, and the arrival time of the last packet frame $i-1$, $t_a(f_{i-1})$. Likewise, the inter-departure time, $\Delta T_d(f_i)$, is computed as the difference between the sending time of the last packet of frame $i-1$, $t_d(f_i)$, and the sending time of the last packet of frame i, $t_d(f_{i-1})$. Note that we consider the frame a group of packets. This is because the game frame (composite of left and right eye frames) in VR cloud gaming is usually huge, even after being encoded, and thus, it is transmitted using several packets.

Like the GCC algorithm, we compute the ***adaptive threshold*** $\gamma(t_i)$ to account for the delay variations based on the network conditions.

Over-use Detector: This compares the delay gradient $m(t_i)$ with the adaptive threshold $\gamma(t_i)$:

$$State = \begin{cases} Overuse & if \quad m(t_i) > \gamma(t_i) \\ Normal & if \quad -\gamma(t_i) <= m(t_i) <= \gamma(t_i) \\ Underuse & if \quad m(t_i) < -\gamma(t_i) \end{cases} \tag{2}$$

Remote rate controller: Owing to the modification in computing delay gradient $m(t_i)$ in Arrival Time Filter, this controller is a modified version of GCC finite state machine (Figure 3). It adjusts a target frame-level bitrate, $T_b(f_i)$ based on the state produced by the Over-use Detector where α is 0.85 and η is 1.05. When the state is "Decr" (decrease $T_b(f_i)$), $T_b(f_i)$ is computed based on the client receive bitrate $B_c(f_{i-1})$, otherwise it is computed based on the previous frame target bitrate, $T_b(f_{i-1})$.

3.3 QP Management Algorithm

To achieve the desired quality, we manipulate the QP of video frames' macroblocks in compatibility with human visual perception. The QP Manager, shown in Figure 2, is responsible for generating encoding parameters based on the target bitrate, actual frame size, and total latency. The encoding parameter includes the QP action to increase or decrease the QP values and the Size factor to guide the Encoder in updating the QP map for the next frame. In the QP Manager, there are two modules: the Adaptive Control Module to strike a balance between the visual quality and encoded frame size, and the Latency Control Module to prioritize low latency.

3.3.1 ***Adaptive Control Module.*** This module is responsible for generating the encoding parameter when the latency is imperceptible (<100 ms) [25]. It derives the QP action based on the target

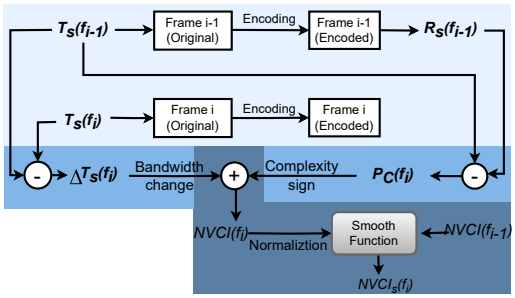


Figure 4: Computation of Network-aware Visual Complexity Index (NVCI) in Adaptive Control Model

and actual frame size of the current and previous frames and then computes a network-aware visual complexity index (NVCI) to drive the QP actions (see Figure 4).

Target encoded frame size T_s **:** The target encoded frame size of frame i, $T_s(f_i)$, is computed by dividing the target bitrate by the frame per second (FPS).

Delta Target frame size ΔT_s **:** The delta target frame size of frame i, $\Delta T_s(f_i)$, is computed as the difference between the target frame size of current frame i and previous frame $i-1$ as follows:

$$\begin{aligned} T_s(f_i) &= T_b(f_i)/FPS \\ \Delta T_s(f_i) &= T_s(f_i) - T_s(f_{i-1}) \end{aligned} \tag{3}$$

$\Delta T_s(f_i)$ accounts for the change in network throughput since $T_s(f_i)$ and $T_s(f_{i-1})$ represents the up-to-date available bandwidth. Therefore, $\Delta T_s(f_i)$ provides signs for adjusting the QP map. Positive $\Delta T_s(f_i)$ indicates an increase in the available bandwidth, while negative $\Delta T_s(f_i)$ indicates a decrease.

Real encoded frame size R_s**:** The real encoded frame size of f_i, $R_s(f_i)$, is collected after the actual encoding of frame i. This size is influenced by factors other than QP, as the complexity of the frame also exerts an impact [31]. Managing the QP map solely based on the network state might lead to under or overutilizing the bandwidth due to variations in frame complexity. Thus, we introduce a complexity penalty to reduce the severity.

Complexity Penalty P_C **:** The Complexity Penalty of frame i, $P_C(f_i)$, is computed as the difference between the target encoded frame size ($T_s(f_{i-1})$) and the real encoded frame size ($R_s(f_{i-1})$) of the previous frame as follows:

$$P_C(f_i) = T_s(f_{i-1}) - R_s(f_{i-1}) \tag{4}$$

While measuring the frame complexity is inherently intricate, we estimate the frame complexity of frame $i-1$ by calculating the difference between the target and real encoded frame sizes. This estimation helps to adjust the QP map in order to accommodate the complexity of frame $i-1$. Given that the complexity of two consecutive frames tends to be highly similar [14], we utilize the estimated complexity of frame $i-1$ to guide the modification of the QP map for frame i. Positive $P_C(f_i)$ indicates a lower complexity, while negative values indicate a higher complexity than anticipated.

The Network-aware Visual Complexity Index (NVCI) is proposed as a means to manage the QPs more considerably, taking into account the index of throughput variations (Delta Target frame size ΔT_s) and the visual complexity of the frame (Complexity Penalty P_C). ΔT_s represents the frame-level available bandwidth difference between recent consecutive frames ($i-1$, i). Therefore, a positive

Table 1: Bitrate states and corresponding QP actions according to bandwidth utilization Index

$NVCI_s$	$NVCI'_s$	State	Action
≥ 0	≥ 0	Bandwidth is underutilized with no tendency to utilize	Decrease QP with constant rate
≥ 0	< 0	Bandwidth is underutilized with tendency to utilize	no change
< 0	≥ 0	Bandwidth is overutilized but with no tendency to worsen	Increase QP with Size factor
< 0	< 0	Bandwidth is overutilized with tendency to worsen	Increase QP with Size factor

ΔT_s value indicates an increase in bandwidth, thus the need to decrease the QP value for enhancing the visual quality. Conversely, negative values indicate a decrease in the available bandwidth which demands increasing the QP value. Similarly, $P_C(f_i)$ indicates changes in the frame's complexity, necessitating a compensatory adjustment in QP value in the opposite direction. Since the change in T_s and P_C, and their corresponding QP actions exhibit consistent directional trends, we construct the NVCI by summing up $T_s(f_i)$ and $P_C(f_i)$ with appropriate weights as follows:

$$NVCI(f_i) = \alpha \Delta T_s(f_i) + \beta P_C(f_i) \tag{5}$$

$NVCI(f_i)$ serves as a sole guiding index for controlling the change of QP for frame i. Positive $NVCI$ values suggest a decrease in the QP map to enhance the visual quality while negative $NVCI$ values require higher QP to avoid large encoded frame size and thus network congestion. Taking into consideration the correlation between consecutive frames, we smooth the $NVCI(f_i)$ value using $NVCI(f_{i-1})$ of the previous frame, resulting in the smoothed value $NVCI_s(f_i)$. $NVCI_s$ plays a role in determining an initial coarse-grained tuning of the QP map. However, to effectively drive the fine-grained tuning phase, it is crucial to consider the tendency, or the direction in which the available bandwidth is moving. We introduce the $NVCI_s$ gradient ($NVCI'_s$) and the Size factor $S(f_i)$ for fine-tuning the QP map. We compute $NVCI'_s$ by finding the best-fitting line using least squares regression. This is achieved by using a moving window of the latest n frames. $S(f_i)$ is a penalty mechanism that is used in cases when the encoded frame size significantly exceeds the target frame size. It is applied when increasing the QP value to enable adaptive adjustment. $S(f_i)$ is computed based on $NVCI(f_i)$ and target encoded frame size $T_s(f_i)$ as reference to normalize it, and expressed as Equation 6:

$$S(f_i) = NVCI(f_i)/T_s(f_i) \tag{6}$$

Table 1 illustrates the states and actions corresponding to four combinations of different signs of $NVCI_s$ value and the gradient $NVCI'_s$. In short, this module aims to achieve the highest visual quality while not exceeding the available bandwidth.

3.3.2 ***Latency Control Module.*** This module is designed to monitor the user-perceived gaming latency and adjust the QP value based on this latency. Motion-to-photon (MTP) latency below 100 ms is imperceptible to humans [25] and acceptance of MTP latency differs across various game genres [10]. Therefore, the latency control module has the highest priority to be applied in the QP management algorithm. When the latency is larger than 100 ms, this module overwrites the action signal, forcing the encoder to increase QP with the Size factor. As such, this module ensures a playable and responsive gaming experience.

3.4 Foveated Video Encoding (FVE)

3.4.1 ***Encoder Setting.*** The system uses NVIDIA Encoder (NVENC), a hardware-based H.264/HEVC/AV1 video encoder for Nvidia GPU [3]. The selection of NVENC over other encoding APIs is primarily attributed to Nvidia's substantial market share in the GPU market. GPUs are critical in achieving video encoding with excellent compression ratios and minimal latency. In contrast to WebRTC and ALVR, which perform encoding based on the target bitrate as a primary parameter, we use the delta QP map function and ConstQP mode to enable the Foveated Video Encoding feature. In WebRTC and ALVR, the encoder allocates QP to each macroblock based on the complexity and target bitrate, which is favorable in controlling the encoded frame size. However, this approach fails to assign proportionate quality across FoV's regions of interest (ROI) which may negatively impact the overall visual user experience. The delta QP map function offers manipulative capabilities for managing the QP distribution via the NVENC API. To ensure proportionate quality and QP assignment, we classify the frame's macroblocks into three regions compatible with HVS (section 2.3).

3.4.2 ***Frame Division into Regions.*** The frame is divided into three distinct sections, including the center, middle, and outer regions, which correspond to the macular, near-peripheral, and other peripheral regions. To convert the FoV regions to areas on the frame, we transform the eccentricity to the number of pixels by the ratio of the eccentricity to the FOV of the VR device (94°). The boundary eccentricity values of the macular and near-peripheral regions, corresponding to the FoV angles 9°and 30°, respectively, are transformed into two circular boundaries. These boundaries have radii of $\frac{9}{94}w$ and $\frac{30}{94}w$, where w is the frame width. These boundaries are utilized to divide the overall area into three distinct regions. QP_{cen}, QP_{mid}, and QP_{out} are assigned to the macroblocks in central, middle, and outer regions, respectively. For every update, only one of the QPs or none is changed.

3.4.3 ***Policy of QP Value Distribution.*** We introduce a policy for updating and distributing the QP values across the three frame regions. We set a different priority for each region and the maximum level gap allowed between two adjacent regions. This is intended to enhance the quality of the central region while maintaining a smooth visual transition between neighboring regions. The comprehensive execution is presented in section 4.2.4.

3.4.4 ***Re-Encoding Module.*** Since QP maps can only control the visual quality, the encoded frame size may rapidly increase due to the complexity of the game frame [24]. The initial implementation of NVENC applies a multipass approach, whereby the first pass is dedicated to complexity estimation and bit distribution, while the second is to perform the encoding process based on the adjusted parameters. Since the constQP mode does not support the multipass approach, we introduce a simple ReEncode procedure. This not only avoids transmitting large encoded frames, thus causing network congestion but also reduces the size of the successive frame, thus ensuring a low latency transmission. Although the ReEncode process may decrease visual quality across multiple frames. Nevertheless, it is important to note that the visual quality can rapidly improve under favorable network conditions.

4 Implementation

The system encompasses server and client modules (see Figure 2). Most of the components operate on the server for lower computational load on the client. To achieve a game-agnostic rendering and streaming, we implement the system on the ALVR base code[2].

4.1 Client

The client is an Android application that can be installed on the head-mounted device (HMD). The client is responsible for data collection, video decoding, and frame reverse rendering and playback.

4.1.1 ***Data Collection Module.*** Besides the motion data, we implement this module to monitor and compute network condition parameters. These parameters are transmitted to the server using a UDP socket to be used in the adaptive control module.

Video Receiving Bitrate $B_c(t)$ is estimated by accumulating the size of the packets that are received during the time unit (500 ms), $B_c(t) = \frac{\sum_{i=0}^{n} P_s(p_i)}{0.5}$, where p_0 to p_n denote the video packets received by the client over a one-time unit. ***Frame Latency*** reflects the user-perceived latency that spans between the time of collecting the physical motion information, $t_t(f_i)$, on the VR device and the time of playing back the corresponding frame on the VR display, $t_p(f_i)$. The corresponding times are recorded for each frame. Thus, the frame latency $T_t(f_i)$ is equal to $t_p(f_i) - t_t(f_i)$. ***Inter-Arrival Time*** $\Delta T_a(f_i)$ is computed as the time difference between two consecutive frames' times $t_a(f_i) - t_a(f_{i-1})$. The frame time is recorded as the time at which the last packet of the frame is received.

4.1.2 ***Video Decoder & Reverse Foveated Rendering.*** We use the ALVR code base without modification to decode the received encoded frames and execute reverse rendering. For the foveated video decoder, the ALVR client utilizes the Android internal decoder with the corresponding encoding configuration received from the server. The reverse-foveated rendering is implemented using the OpenGL shader library that optimizes the computation.

4.2 Server

The server is a Windows application that is installed on the gaming PC. The server includes network monitoring, adaptive control, foveated rendering, and foveated video encoding modules.

4.2.1 ***Network Monitoring Unit.*** This is implemented by maintaining a frame-level statistics manager on the server side. For each frame, the data collected on the server and the feedback received from the client are maintained by this module.

Video Streaming Bitrate $B_s(t)$ is the sending bitrate that is estimated by accumulating the size of the encoded frames, $R_s(f_i)$, over a pre-defined time unit (500 ms). $B_s(t) = \frac{\sum_{i=0}^{n} R_s(f_i)}{0.5}$, where f_0 to f_n denote the frames sent over a one-time unit (500 ms). ***Inter-Departure Time*** $\Delta T_d(f_i)$ is computed as the time difference between two consecutive frames' times $t_d(f_i) - t_d(f_{i-1})$. The frame time is recorded as the time at which its last packet is transmitted.

The Inter-Departure Time and Inter-Arrival Time (on the client) help to implement the modified version of the Arrival Time Filer (see Section 3.2.3). We implement other components of frame-level delay-based congestion control based on GCC standard [4] and the modification presented in Figure 3. Accordingly, the network monitoring module produces the information: $\Delta T_a(f_i)$, $\Delta T_d(f_i)$, $B_c(f_i)$, $T_b(f_{i-1})$. Any feedback received triggers updating this information. The target bitrate $T_b(f_i)$ is then passed to the QP Manager.

4.2.2 ***QP Manager*** calculates the QP Action and S_f based on the above-computed information, following Algorithm 1. It computes the target frame size $T_s(f_{i-1})$ based on $T_b(f_{i-1})$. The real encoded frame size $R_s(f_{i-1})$ is recorded after encoding. Then, the NVCI of the next frame (frame i) is computed. However, the coefficients (α, β) of Equation 5 have to be computed and tuned. Our experimental investigation reveals that the change in visual complexity demands a larger adjustment in QP change, compared to the change in available bandwidth. Therefore, higher weight must be assigned to $P_C(f_{i-1})$ as the value of β. Accordingly, α is set to 1 while β is set to 2. This results in $NVCI(f_i)$ value that is then smoothed using $NVCI(f_{i-1})$ of the previous frame, resulting in $NVCI_s(f_i)$ expressed as follows:

$$\begin{aligned} NVCI(f_i) &= \Delta T_s(f_i) + 2P_C(f_{i-1}) \\ &= (T_s(f_i) - T_s(f_{i-1})) + 2(T_s(f_{i-1}) - R_s(f_{i-1})) \\ &= T_s(f_i) + T_s(f_{i-1}) - 2R_s(f_{i-1}) \\ NVCI_s(f_i) &= 0.9NVCI(f_i) + 0.1NVCI(f_{i-1}) \end{aligned} \tag{7}$$

The NVCI gradient, $NVCI'_s$, is computed using least squares regression latest 20 data points (times t_s and $NVCI_s$) of the latest 20 frames. The signs of the gradient and the value of $NVCI_s$ determine the QP actions (see Table 1). Notably, when $NVCI_s$ is negative, "Increase QP with size factor" is applied regardless of whether $NVCI'_s$ is negative or positive. Afterward, QP action, Size factor $S(f_i)$, and target encoded frame size $T_s(f_i)$ are passed to the encoder as the encoding parameters. However, when the latency is larger than 100 ms, the Latency Control module overwrites the action signal, forcing the encoder to increase QP according to the Size factor.

Algorithm 1 QP Management Pseudocode

Input: $T_b(f_{i-1})$, $R_s(f_{i-1})$, $T_b(f_i)$, FPS
Output: QPAction, S_f
$T_s(f_{i-1}) = T_b(f_{i-1})$/FPS ▷ Eq: 3
$T_s(f_i) = T_b(f_i)$/FPS
$\Delta T_s(f_i) = T_s(f_i) - T_s(f_{i-1})$
$P_C(f_i) = T_s(f_{i-1}) - R_s(f_{i-1})$ ▷ Eq: 4
$NVCI(f_i) = T_s(f_i) + T_s(f_{i-1}) - 2R_s(f_{i-1})$ ▷ Eq: 7
$NVCI_s(f_i) = 0.9NVCI(f_i) + 0.1NVCI(f_{i-1})$
$NVCI'_s(f_i) \leftarrow$ least-square regression with 20 data points
QPAction $\leftarrow NVCI(f_i), NVCI'_s(f_i)$ ▷ Table 1
$S_f = NVCI(f_i)/T_s(f_i)$ ▷ Eq: 6

4.2.3 ***Foveated Rendering.*** As Figure 2 illustrates, the server obtains the frame from the gaming engine through VHMD to apply fixed-foveated rendering (FFR). FFR uses a gradient-based mask to reduce the level of detail or resolution towards the periphery of the frame while maintaining higher resolution in the central region. This resource redistribution prioritizes the central region, reducing the frame resolution and transmission data. The FFR uses OpenGL Shading Language and leverages the computational capabilities of the GPU to reduce processing time. The FFR process is crucial to accelerate frame processing as the game frame contains both the left-eye and right-eye images. This might cause an increase in frame width, which may eventually exceed the maximum limit of the encoding dimension and thus lead to a much higher frame size.

4.2.4 ***Foveated video Encoding.*** The different FoV regions defined in section 3.4.2 filling with the updated $QP_{cen}, QP_{mid}, QP_{out}$ to form the QP map for encoding. We implement the QP map update policy and re-encoding module to adjust the encoding parameters based on the network situation and frame complexity.

Algorithm 2 QP Map Update

Input: QPAction, Size factor (S_f), $QP_{cen}, QP_{mid}, QP_{out}$
Output: QPMap
$Lv_{cen}, Lv_{mid}, Lv_{out} \leftarrow$ level of $QP_{cen}, QP_{mid}, QP_{out}$
if QPAction = decrease **then**
  if $Lv_{mid} > Lv_{cen} + 1$ **then**
    $QP_{mid} \leftarrow QP_{mid} - 0.1$
  else if $Lv_{out} > Lv_{mid} + 1$ **then**
    $QP_{out} \leftarrow QP_{out} - 0.1$
  else if $QP_{cen} \geq 1$ **then**
    $QP_{cen} \leftarrow QP_{cen} - 0.1$
  else if $QP_{mid} \geq 1$ **then**
    $QP_{mid} \leftarrow QP_{mid} - 0.1$
  else if $QP_{out} \geq 1$ **then**
    $QP_{out} \leftarrow QP_{out} - 0.1$
  end if
else if QPAction = increase **then**
  if $Lv_{mid} < Lv_{out} - 1$ **then**
    $QP_B \leftarrow QP_B + 0.1 + \frac{1}{5}S_f$
  else if $Lv_{cen} < Lv_{mid} - 1$ **then**
    $QP_A \leftarrow QP_A + 0.1 + \frac{1}{5}S_f$
  else if $QP_C \leq 51$ **then**
    $QP_C \leftarrow QP_C + 0.1 + \frac{1}{5}S_f$
  else if $QP_B \leq 51$ **then**
    $QP_B \leftarrow QP_B + 0.1 + \frac{1}{5}S_f$
  else if $QP_A \leq 51$ **then**
    $QP_A \leftarrow QP_A + 0.1 + \frac{1}{5}S_f$
  end if
end if
QPMap $\leftarrow QP_{cen}, QP_{mid}, QP_{out}$: pre-divided regions
return QPMap

QP Map update. To implement the policy in section 3.4.3, we categorize the QP values, ranging from 1 to 51, into six levels: 1-6, 7-13, 14-20, 21-28, 29-38, and 39-51 levels, with the increasing priority as outer, middle, and center and the decreasing priority in reverse order. The QP adjustment follows the priority policy when the outer region is one level higher or equal to the inner region. When the level difference between neighboring areas exceeds 1, the QP update is applied to the region with the lowest level gap to maintain a balanced progression. The QP update process is detailed as a pseudo-code in Algorithm 2.

Re-Encoding Module. Frame re-encoding during streaming presents a challenge due to the reliance on predicted frames (P-frames) that depend on the successful encoding or decoding of previous frames. To mitigate computational load and reduce dependencies on previous frame content, this module opts to re-encode the frame as an I-frame instead. If the size of the encoded frame is double or more than the size of the target frame, we re-encode the frame by increasing the QP_{cen} by 1, QP_{mid}, QP_{out} by 2. Following the update of the QP value, the QP map is regenerated accordingly. The next frame is then formed by transmitting the identical frame to the encoder along with the new QP map and an I-frame request.

5 Experiment Setup

In this section, we introduce the cloud gaming emulator and the network traces utilized, outline the data collection process for assessing visual quality, and provide an overview of the baselines.

5.1 Measurement Setup

We first design a physical testbed to create a realistic emulation of the gaming over 5G mobile networks. We configure a Linux PC with WiFi and Ethernet interfaces (Ubuntu 22.04.3 LTS, Intel(R) Core(TM) i9-13900K @ 5.80 GHZ) to function as a router between wired and wireless networks. This router relays traffic data received from an Oculus Quest 2 (Android 12, Octa-core Kryo 585, Adreno 650) over a wireless network to a Windows PC on an Ethernet network. Likewise, the PC router relays the video stream (received from the Windows PC over Ethernet) to the Oculus Quest 2 over the WiFi network. The Windows PC (Windows 11 22H2, Intel(R) Core(TM) i9-12900H @ 2.50 GHZ, RTX 3080 Ti Laptop GPU) acts as a cloud server in the emulated environment. We develop a bash script that operates on the PC router to emulate mobile network connectivity for the Oculus. The script utilizes Linux tc [16], and incorporates real-world 5G network traces (section 5.2) to regulate the bandwidth on the outgoing WiFi interface.

We retrieve the game frame and rendered frame and store them in byte format, while we obtain and store the encoded frame in H264 format. The resolution of the game frame is 3712×2016, whereas the resolution of the rendered frame is 2048×960. The capturing frequency is 1000 frames to avoid affecting performance since saving the frames uses up huge hardware resources. The encoded frame is decoded and reverse-foveated rendered with the rendered frame to reach the same resolution as the game frame. The three frames are compared by cropping them in three different ways: under the entire frame, the central fovea (30°), and the central fovea (9°).

Table 2: Baselines considered in the system evaluation

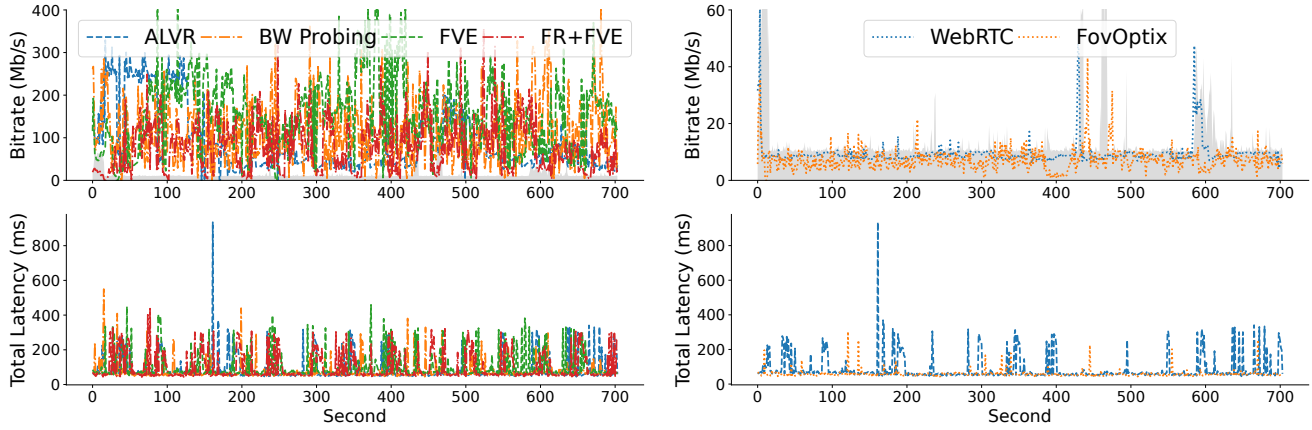
Protocol	Rate Control	Description
ALVR [2]	Latency-based	Adapt bitrate according to latency
WebRTC [4]	Google Congestion Control (GCC)	Adapt bitrate according to bandwidth utilization, loss and latency
BW Probing	Throughput-based	Adapt bitrate using packet probing-based bandwidth estimation
FVE [11, 17]	HVS-based	Uneven resolution across FoV regions
FR-FVE [18]	HVS-based	Uneven resolution and quality across FoV regions for rendering and encoding
FovOptix	HVS and Enhanced GCC-based	HVS-compatibility and bitrate adaptability to bandwidth usage and latency

5.2 Mobile Network Traces

To emulate the real-world network between the player's VR headset and the cloud (section 5.1), we leverage a corpus of ***5G mobile network traces*** collected from a major Irish mobile operator. The traces are generated across two application patterns (video streaming and file download). Specifically, they are generated during continuous large file downloads, streaming of video content from Netflix service provider, and streaming of video content from Amazon Prime service provider [26]. VR gamers typically play VR games at home or in stationary conditions. Therefore, we choose the traces of static patterns (out of driving and static) to reflect the typical game-playing conditions. Because some baselines collapse when the available bandwidth is less than 10 Mb/s, we filter the traces to keep only the records with a throughput of greater than 10 Mb/s and eliminate those falling below this threshold.

5.3 Baselines

As illustrated in Table 2, our baselines include Real-time communication for the web (WebRTC) [4], Air Light VR (ALVR) [2], Bandwidth (BW) Probing method, Foveated Video Encoding (FVE) [11, 17], Integrated Foveated Rendering and Video Encoding (FR-FVE) [18]. We use ALVR open source as our first baseline. We utilize the code base of ALVR to implement and integrate the rest of the baselines. We integrate the Google Congestion Control (GCC) mechanism



(a) **non-bandwidth adaptive protocols** *(b)* **bandwidth adaptive protocols**

Figure 5: Adaptability to controlled network bandwidth using real-world 5G network traces.

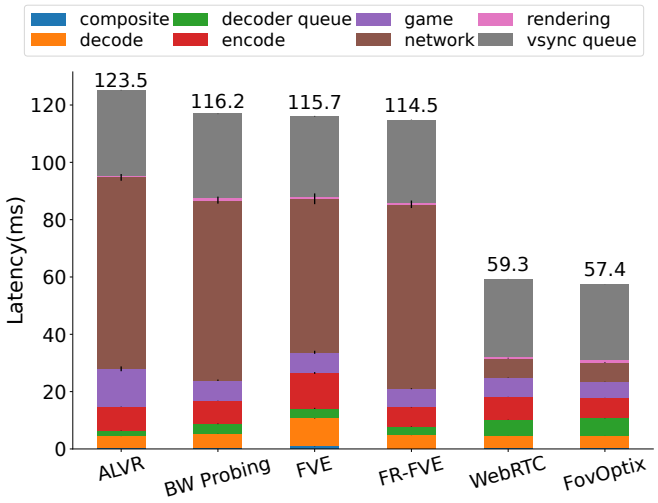


Figure 6: Motion-to-photon latency components when gaming over 5G network traces. FovOptix exhibits the lowest end-to-end latency, followed by WebRTC.

into ALVR open source to implement real-time communication for VR cloud gaming (WebRTC). We also implement a packet probing-based mechanism to estimate the network throughput (BW Probing). This method is used in HTTP-based video streaming such as FESTIVE [22]. Since FVE and FR-FVE are implemented for human vision-based rendering and streaming in mobile cloud gaming, we implement them over ALVR to operate in VR cloud gaming.

6 Evaluation

In this section, we employ a comprehensive evaluation approach for FovOptix. We assess its adaptability, latency, and visual quality objectively. Additionally, we conduct user experiments where participants play a VR game and rate their experience subjectively.

6.1 Adaptability to available bandwidth.

To assess adaptability, we utilize our testbed (section 5.1), which dynamically adjusts bandwidth to mimic 5G mobile network traces with a static mobility pattern (section 5.2). We visualize the network throughput of each solution in Figure 5. The network traces show lower throughput, mainly ranging from 10 Mb/s to 20 Mb/s, which is below the bandwidth requirements of certain protocols. As a result, non-adaptive protocols (ALVR, BW Probing, FVE, and FR-FVE) fail to function properly under these conditions (Figure 5a). They frequently exceed the bandwidth capacity, causing network congestion, frame drops, and high and variable motion-to-photon latency. In contrast, WebRTC and FovOptix adapt to the available bandwidth, resulting in low latency (Figure 5b). Notably, FovOptix demonstrates superior adaptability, leading to more stable latency.

6.2 Motion-to-photon latency.

Figure 6 illustrates the latency components: vsync queuing, game, rendering, encoding, networking, decode queuing, decode, and composite latency. FovOptix outperforms all the baselines including WebRTC (mean 59.3 ms), exhibiting the lowest end-to-end latency (mean 57.4 ms). Notably, FovOptix's adaptive QP manager accelerates the encoding (mean 7.2 ms), compared to WebRTC (mean 8.2 ms), reducing its average total latency by 1.9 ms. We attribute this reduction in FovOptix encoding latency to the usage of QP as input instead of a bitrate. WebRTC's encoder computes the QP for every macro-block and performs a multi-pass function of NvEncoder to calculate the complexity of each frame. Accordingly, it reassigns the QP for every macro-block to ensure better control of the sending bitrate. Unsurprisingly, the protocols classified as non-bandwidth adaptive encounter high network latency (brown bars). These baselines send at bitrates higher than the available bandwidth, causing frequent congestion and thus high networking latency.

6.3 Visual Quality.

Figure 7 illustrates the visual quality (VQ) of the rendered and encoded frames. We measure the VQ objectively using SSIM and PSNR by using the game frame as the reference image, against which the rendered and encoded frames are compared to assess their similarity and quality. The three sub-figures present the VQ of the rendered frames (gray markers) and compressed frames (colorful markers) for five baselines (WebRTC, ALVR, BW Probing, FR-FVE, and FovOptix). The left-hand side presents the SSIM (y-axis) and

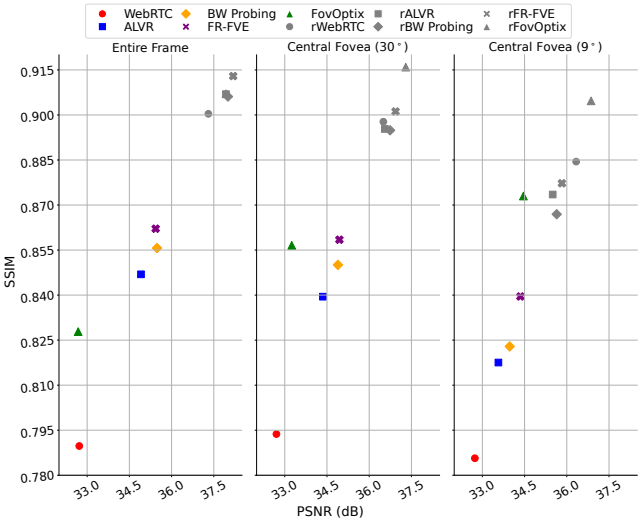


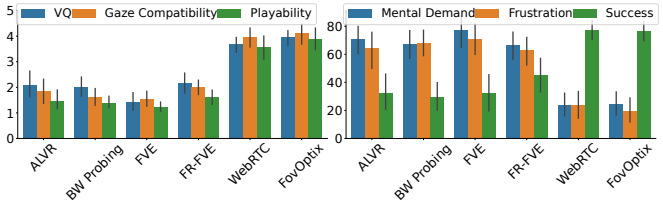
Figure 7: Average SSIM and PSNR of rendered and encoded frames and central FoV regions over emulated 5G mobile network. *r* prefix and gray markers reflect the visual quality (VQ) of the protocols after rendering, whereas colorful markers indicate their VQ after encoding. FVE doesn't involve FR and exhibits low PSNR≈30.1 and SSIM≈0.7, thus not depicted.

PSNR (x-axis) of FovOptix over the entire frame, compared to the other four baselines. The middle and right-hand sides present the SSIM and PSNR across frame areas that match the central fovea of 30°and 9°, respectively. Over the entire frame, FovOptix achieves a higher SSIM of an average of 0.83 and a similar PSNR of an average of 0.325 dB, compared to WebRTC, which has an SSIM of 0.788 and a PSNR of 0.325 dB, respectively. FVE exhibits the lowest SSIM and PSNR of an average of 0.7 and 32.1, respectively. The other baselines achieve higher visual quality compared to FovOptix and WebRTC. Notably, FovOptix demonstrates improved SSIM of an average of 0.877 and PSNR of an average of 34.5 dB in central frame tiles that correspond to the central foveal region, for a 30°central fovea. When the central fovea is 9°, FovOptix outperforms all the baselines with an average SSIM of 0.877 and PSNR of 34.5 dB.

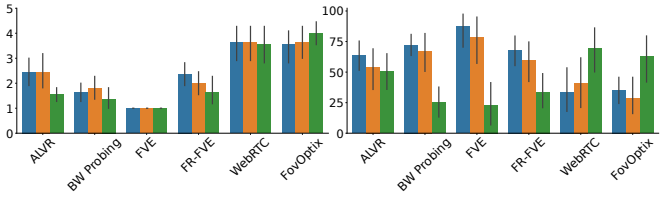
6.4 User Study

Participants and Apparatus: We recruited 33 participants, aged 22 to 42. The majority are university students and staff. They play VR games a few times a week (4), a few times a month (7), a few times a year (14), while 8 never played. Most of them have some familiarity with virtual reality. The participants played the Lab game in a VR gaming setting using the system prototype (section 3.1) in two groups. The first group (18 participants) played a first-person shooting (FPS) game called Longbow, while the second group (15 participants) played a third-person shooting (TPS) game called Xortex. Each participant played the game over the transmission techniques defined in section 5.3 over our cloud gaming emulation.

Protocol: Each participant started with 5 minutes of free play of the Lab game executed on-device. This phase allows the participants to get familiar with the game and establish a reference of playability and visual quality. The participants then played a VR game for



(a) **Perception of FPS game (Longbow). FovOptix presents highest playability, VQ, and gaze compatibility, lowest frustration and comparable perceived success, and mental demand to WebRTC.**



(b) **Perception of TPS game (Xortex). FovOptix presents the highest playability and comparable gaze compatibility, and slightly lower VQ to WebRTC, with lowest frustration, second lowest mental demand, and second highest perceived success.**

Figure 8: Users' perception of the gaming experience under typical measures (left) and task load (right) over emulated 5G network.

Table 3: Average motion sickness (MSS) and game score

FPS Game (Longbow)						
Sol	ALVR	BW Prob	FVE	FR-FVE	WebRTC	FovOptix
score	174	172	85	295	781	**873**
MSS	1.8	1.65	2.3	1.77	1.3	**1.05**
TPS Game (Xortex)						
score	433	580	84	276	1664	**1742**
MSS	2.18	2.18	3	2	1.27	**1.18**

five minutes for each protocol. The participants were given a brief intermediary break to avoid cumulative effects. We restarted the network emulation script (section 5.1) for each protocol to maintain consistent network conditions across the protocols. Besides, we used the lucky draw method to randomize the protocol sequence and prevent learning and order effects. Each protocol was assigned a unique number, written on a piece of paper, and placed in a pot. Participants randomly draw a paper from the pot, determining the running order. After each run, the participants filled out a short questionnaire on a 5-point Likert scale on the perceived visual quality *(1:bad to 5:excellent)*, suitability to visual experience *(1:not well to 5:Extremely well)*, playability *(1:laggy to 5:extremely responsive)*. They also fill out a NASA TLX [12] survey on the perceived mental demand, frustration, and success on a [0-100] scale, and report game score. Besides, they reported their susceptibility to motion sickness by rating the severity on a [1-7] scale *(1:no symptoms to 7:experiencing vomiting)*. We did not disclose the streaming methods used to avoid affecting the participants' ratings.

Results: Figure 8 presents the results with error bars as 95% confidence intervals. Figure 8a depicts the users' perception of the FPS game. FovOptix exhibits superior performance in terms of playability (mean 3.9), gaze compatibility (mean 4.1), and VQ (mean 3.9).

This is followed by WebRTC, which achieves average scores of 3.6, 3.9, and 3.6 for the same measures. Additionally, FovOptix exhibits the lowest frustration level with an average score of 19, while WebRTC has an average score of 22. In terms of mental demand and perceived success, FovOptix performs comparably to WebRTC.

Figure 8b depicts the users' perception of the TPS game. FovOptix presents the highest playability (mean 4), while it exhibits similar gaze compatibility (mean 3.6) and slightly lower VQ (mean 3.5), compared to WebRTC of an average of 3.5, 3.6, and 3.6, respectively. In terms of task load, In comparison to WebRTC with mean scores of 41, 34, and 69, FovOptix demonstrates lower frustration (mean 28), slightly higher mental effort (mean 35), and lower perceived success (mean 63). Other baselines (BW Probing, FVE, FR-FVE, ALVR) perform poorly across all the performance metrics in both game genres. Table 3 presents the average rating of game score and motion sickness severity of both game genres. The participants obtained the highest score using FovOptix, followed by WebRTC. This conforms to the perceived playability, where FovOptix achieves the highest rating. Besides, FovOptix exhibits the lowest motion sickness severity, implying minimal or no symptoms. Except for WebRTC, the remaining baselines elicit the feeling of discomfort or dizziness. One-way ANOVA tests show a statistically significant difference ($p < 0.002$ for all measures) between the transmission protocols.

6.5 Findings and Discussion

We conducted our experiments over an emulation framework that incorporates network variability via the use of 5G mobile network traces. This helps to understand the accessibility and performance of the state-of-the-art solutions.

FovOptix excels in achieving the lowest motion-to-photon latency and the highest quality in regions of interest (ROI) while not compromising the overall visual quality. This is attributed to its high adaptability to network throughput and compatibility with the human vision system. The user study confirms that only FovOptix and WebRTC can provide a smooth experience without mental effort or annoyance. Objectively, FovOptix demonstrates significant improvement in visual quality, especially in regions of interest, along with a slight reduction in latency compared to WebRTC. Additionally, the participants felt the least motion sickness with FovOptix and somewhat greater with WebRTC. Nevertheless, FovOptix presents a substantial collective improvement in all performance metrics, as evidenced by increased perceived playability and game scores compared to WebRTC. The outstanding performance of FovOptix under variable network conditions, coupled with its capability to install and run any VR game, demonstrates its potential to provide accessible VR gaming service across diverse network conditions. With high quality in ROI, low latency, and the likelihood of having little or no motion sickness symptoms, FovOptix also enhances the immersion and interaction and ensures seamless VR gaming experiences.

In the TPS game, the participants rated FovOptix's VQ and gaze compatibility to be comparable to or slightly inferior to that of WebRTC. We attribute this to the frequent diversion of players' attention from the central FoV to find new targets and evade incoming shots, which leads to a perception of lower VQ in the peripheral regions where their gaze is focused. This is evident by FovOptix's enhanced performance in the FPS game, where the dominant gaze focus is on the central FoV. The participants thus rated FovOptix's VQ and gaze compatibility superior to WebRTC.

7 Limitation and Future Improvement

FovOptix exhibits low latency and high visual quality due to its adaptability to network throughput. However, it is important to acknowledge the existence of several limitations. In this section, we elucidate these limitations and future directions to tackle them.

FovOptix employs fixed FR and fixed FVE to enhance the quality in the central FoV regions, preventing dynamic allocation of higher quality to the precise gaze position. While players may shift their focus to near-peripheral or peripheral areas, we will expand the data collection module to include the acquisition of gaze position in real time. We will use built-in gaze trackers like that in Quest Pro. The gaze data will be transmitted constantly to the server to render and compress the game frames accordingly. In FovOptix, the gaze position is fixed in the center of FoV, corresponding to coordinates $(\frac{w}{2}, \frac{h}{2})$ in the game frame. Therefore, the integration of real-time gaze position (x,y) will be instantaneous. The eye tracker latency is reported to be in the range of $6 - 10$ ms [6]. Consequently, the average total latency is estimated to be approximately $64 - 68$ ms, which is deemed tolerable according to Albert et al. [6].

To offer game-agnostic support, we utilized the ALVR code base, which has a limitation in detecting VR clients on a public network. Consequently, FovOptix was tested and evaluated using an emulation of a real-life mobile network, specifically based on 5G mobile network traces. We plan to extend the VHMD component to recognize clients over public networks. We also plan to evaluate the performance across various game genres like shooters, real-time strategy, and role-playing. This assessment would help to assess the system's ability to sustain video quality and dynamically allocate bitrate based on the visual complexity and user attention patterns unique to each gaming genre, providing insights into its effectiveness and suitability in diverse gaming environments.

8 Conclusion

This paper presented FovOptix, a system that combines foveated rendering with adaptive foveated video compression. FovOptix adapts the visual quality in central and peripheral regions to human vision and network conditions. Using emulation of a 5G mobile network, we evaluate FovOptix against the state-of-the-art solutions. Our findings revealed that FovOptix achieves the lowest latency and enhanced visual quality in the user's region of interest. Our user study confirmed these findings, with FovOptix presenting the highest playability and gaming score, and lowest motion sickness level. FovOptix thus facilitates smooth and accessible VR cloud gaming that accommodates players with low-capacity and unstable networks. Our proposed improvements would significantly enhance the immersion and accessibility of the VR gaming experience. This includes integrating real-time gaze tracking for dynamic quality allocation, extending the server for accessibility over public networks, and evaluation across various game genres.

9 Acknowledgements

This research was supported in part by a grant from the Guangzhou Municipal Nansha District Science and Technology Bureau under Contract No.2022ZD01 and the MetaHKUST project from the Hong Kong University of Science and Technology (Guangzhou).

References

[1] 2018. SteamVR. https://store.steampowered.com/app/250820/SteamVR/

[2] 2023. Air Light VR (ALVR). https://github.com/alvr-org/ALVR

[3] 2023. NVENC Video Encoder API Programming Guide. https://docs.nvidia.com/video-technologies/video-codec-sdk/12.1/nvenc-video-encoder-api-prog-guide/index.html

[4] 2023. Real-time Communication for the Web (WebRTC). https://webrtc.org/

[5] Mehmet N. Akcay. 2021. Improving Server and Client-Side Algorithms for Adaptive Streaming of Non-Immersive and Immersive Media. In *Proceedings of the 12th ACM Multimedia Systems Conference* (Istanbul, Turkey) *(MMSys '21)*. Association for Computing Machinery, New York, NY, USA, 383–387. https://doi.org/10.1145/3458305.3478461

[6] Rachel Albert, Anjul Patney, David Luebke, and Joohwan Kim. 2017. Latency Requirements for Foveated Rendering in Virtual Reality. *ACM Trans. Appl. Percept.* 14, 4, Article 25 (sep 2017), 13 pages. https://doi.org/10.1145/3127589

[7] Ahmad Alhilal, Tristan Braud, Bo Han, and Pan Hui. 2022. Nebula: Reliable Low-Latency Video Transmission for Mobile Cloud Gaming. In *Proceedings of the ACM Web Conference 2022* (Virtual Event, Lyon, France) *(WWW '22)*. Association for Computing Machinery, New York, NY, USA, 3407–3417. https://doi.org/10.1145/3485447.3512276

[8] D.A. Atchison. 2023. *Optics of the Human Eye*. CRC Press. https://books.google.com.hk/books?id=5-WtEAAAQBAJ

[9] Gaetano Carlucci, Luca De Cicco, Stefan Holmer, and Saverio Mascolo. 2016. Analysis and Design of the Google Congestion Control for Web Real-Time Communication (WebRTC). In *Proceedings of the 7th International Conference on Multimedia Systems* (Klagenfurt, Austria) *(MMSys '16)*. Association for Computing Machinery, New York, NY, USA, Article 13, 12 pages. https://doi.org/10.1145/2910017.2910605

[10] Matthias Dick, Oliver Wellnitz, and Lars Wolf. 2005. Analysis of Factors Affecting Players' Performance and Perception in Multiplayer Games. In *Proceedings of 4th ACM SIGCOMM Workshop on Network and System Support for Games* (Hawthorne, NY) *(NetGames '05)*. Association for Computing Machinery, New York, NY, USA, 1–7. https://doi.org/10.1145/1103599.1103624

[11] Florian Frieß, Matthias Braun, Valentin Bruder, Steffen Frey, Guido Reina, and Thomas Ertl. 2021. Foveated Encoding for Large High-Resolution Displays. *IEEE Transactions on Visualization and Computer Graphics* 27, 2 (2021), 1850–1859. https://doi.org/10.1109/TVCG.2020.3030445

[12] Sandra G Hart. 2006. NASA-task load index (NASA-TLX); 20 years later. In *Proceedings of the human factors and ergonomics society annual meeting*, Vol. 50. Sage publications Sage CA: Los Angeles, CA, 904–908.

[13] Jan Horský. 2022. Crowdsourcing VR headset data VR headset database. https://www.infinite.cz/projects/HMD-tester-virtual-reality-headset-database-utility

[14] Sudeng Hu, Hanli Wang, and Sam Kwong. 2012. Adaptive Quantization-Parameter Clip Scheme for Smooth Quality in H.264/AVC. *IEEE Transactions on Image Processing* 21 (04 2012), 1911–1919. https://doi.org/10.1109/TIP.2011.2176347

[15] Chun-Ying Huang, Cheng-Hsin Hsu, Yu-Chun Chang, and Kuan-Ta Chen. 2013. GamingAnywhere: An Open Cloud Gaming System. In *Proceedings of the 4th ACM Multimedia Systems Conference* (Oslo, Norway) *(MMSys '13)*. Association for Computing Machinery, New York, NY, USA, 36–47. https://doi.org/10.1145/2483977.2483981

[16] Bert Hubert. [n.d.]. Linux TC Man Page. https://linux.die.net/man/8/tc

[17] Gazi Karam Illahi, Thomas Van Gemert, Matti Siekkinen, Enrico Masala, Antti Oulasvirta, and Antti Ylä-Jääski. 2020. Cloud Gaming with Foveated Video Encoding. *ACM Trans. Multimedia Comput. Commun. Appl.* 16, 1, Article 7 (feb 2020), 24 pages. https://doi.org/10.1145/3369110

[18] Gazi Karam Illahi, Matti Siekkinen, Teemu Kämäräinen, and Antti Ylä-Jääski. 2020. On the Interplay of Foveated Rendering and Video Encoding. In *Proceedings of the 26th ACM Symposium on Virtual Reality Software and Technology* (Virtual Event, Canada) *(VRST '20)*. Association for Computing Machinery, New York, NY, USA, Article 66, 3 pages. https://doi.org/10.1145/3385956.3422126

[19] Gazi Karam Illahi, Matti Siekkinen, Teemu Kämäräinen, and Antti Ylä-Jääski. 2021. Foveated Streaming of Real-Time Graphics. In *Proceedings of the 12th ACM Multimedia Systems Conference* (Istanbul, Turkey) *(MMSys '21)*. Association for Computing Machinery, New York, NY, USA, 214–226. https://doi.org/10.1145/3458305.3463383

[20] Yoshio Ishiguro and Jun Rekimoto. 2011. Peripheral vision annotation: noninterference information presentation method for mobile augmented reality. In *Proceedings of the 2nd Augmented Human International Conference*. 1–5.

[21] Nuwan Janaka, Chloe Haigh, Hyeongcheol Kim, Shan Zhang, and Shengdong Zhao. 2022. Paracentral and Near-Peripheral Visualizations: Towards Attention-Maintaining Secondary Information Presentation on OHMDs during in-Person Social Interactions. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems* (New Orleans, LA, USA) *(CHI '22)*. Association for Computing Machinery, New York, NY, USA, Article 551, 14 pages. https://doi.org/10.1145/3491102.3502127

[22] Junchen Jiang, Vyas Sekar, and Hui Zhang. 2012. Improving Fairness, Efficiency, and Stability in HTTP-Based Adaptive Video Streaming with FESTIVE. In *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies* (Nice, France) *(CoNEXT '12)*. Association for Computing Machinery, New York, NY, USA, 97–108. https://doi.org/10.1145/2413176.2413189

[23] Xing Liu, Bo Han, Feng Qian, and Matteo Varvello. 2019. LIME: Understanding Commercial 360° Live Video Streaming Services. In *Proceedings of the 10th ACM Multimedia Systems Conference* (Amherst, Massachusetts) *(MMSys '19)*. Association for Computing Machinery, New York, NY, USA, 154–164. https://doi.org/10.1145/3304109.3306220

[24] Vignesh V Menon, Christian Feldmann, Hadi Amirpour, Mohammad Ghanbari, and Christian Timmerer. 2022. VCA: Video Complexity Analyzer. In *Proceedings of the 13th ACM Multimedia Systems Conference* (Athlone, Ireland) *(MMSys '22)*. Association for Computing Machinery, New York, NY, USA, 259–264. https://doi.org/10.1145/3524273.3532896

[25] Lothar Pantel and Lars C. Wolf. 2002. On the Impact of Delay on Real-Time Multiplayer Games. In *Proceedings of the 12th International Workshop on Network and Operating Systems Support for Digital Audio and Video* (Miami, Florida, USA) *(NOSSDAV '02)*. Association for Computing Machinery, New York, NY, USA, 23–29. https://doi.org/10.1145/507670.507674

[26] Darijo Raca, Dylan Leahy, Cormac J. Sreenan, and Jason J. Quinlan. 2020. Beyond Throughput, the next Generation: A 5G Dataset with Channel and Context Metrics. In *Proceedings of the 11th ACM Multimedia Systems Conference* (Istanbul, Turkey) *(MMSys '20)*. Association for Computing Machinery, New York, NY, USA, 303–308. https://doi.org/10.1145/3339825.3394938

[27] Jihoon Ryoo, Kiwon Yun, Dimitris Samaras, Samir R. Das, and Gregory Zelinsky. 2016. Design and Evaluation of a Foveated Video Streaming Service for Commodity Client Devices. In *Proceedings of the 7th International Conference on Multimedia Systems* (Klagenfurt, Austria) *(MMSys '16)*. Association for Computing Machinery, New York, NY, USA, Article 6, 11 pages. https://doi.org/10.1145/2910017.2910592

[28] Jihoon Ryoo, Kiwon Yun, Dimitris Samaras, Samir R. Das, and Gregory Zelinsky. 2016. Design and Evaluation of a Foveated Video Streaming Service for Commodity Client Devices. In *Proceedings of the 7th International Conference on Multimedia Systems* (Klagenfurt, Austria) *(MMSys '16)*. Association for Computing Machinery, New York, NY, USA, Article 6, 11 pages. https://doi.org/10.1145/2910017.2910592

[29] Andrea Scupola, Alessandra Mastrocola, Paola Sasso, Romina Fasciani, Lucrezia Montrone, Benedetto Falsini, and Edoardo Abed. 2013. Assessment of Retinal Function Before and After Idiopathic Macular Hole Surgery. *American Journal of Ophthalmology* 156, 1 (2013), 132–139.e1. https://doi.org/10.1016/j.ajo.2013.02.007

[30] Hans Strasburger, Ingo Rentschler, and Martin Jüttner. 2011. Peripheral vision and pattern recognition: A review. *Journal of Vision* 11, 5 (Dec. 2011), 13–13. https://doi.org/10.1167/11.5.13 _eprint: https://arvojournals.org/arvo/content_public/journal/jov/933487/jov-11-5-13.pdf.

[31] M. Tun, K.K. Loo, and J. Cosmas. 2008. Rate control algorithm based on quality factor optimization for Dirac video codec. *Signal Processing: Image Communication* 23, 9 (2008), 649–664. https://doi.org/10.1016/j.image.2008.07.003