A Log-Rectilinear Transformation for Foveated 360-degree Video Streaming

David Li *Student Member, IEEE*, Ruofei Du *Member, IEEE*,
Adharsh Babu, Camelia D. Brumar, and Amitabh Varshney *Fellow, IEEE*

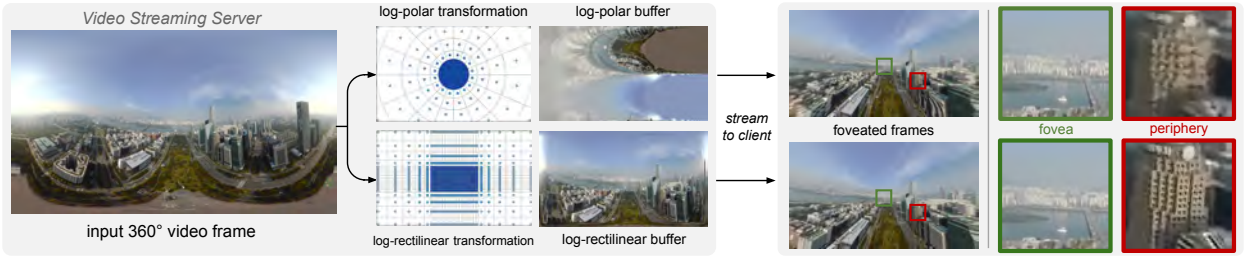


Fig. 1: An overview of our foveated 360° video streaming pipeline. The upper and lower rows present the workflows with the prior log-polar transformation and our proposed log-rectilinear transformation, respectively. Both foveated methods convert the equirectangular video frames into down-sampled buffers, which are encoded and streamed to the client. After reprojection, our log-rectilinear transformation greatly reduces the flickering and aliasing artifacts while maintaining high-quality rendering in the foveal region and reducing overall bandwidth.

Abstract— With the rapidly increasing resolutions of 360° cameras, head-mounted displays, and live-streaming services, streaming high-resolution panoramic videos over limited-bandwidth networks is becoming a critical challenge. Foveated video streaming can address this rising challenge in the context of eye-tracking-equipped virtual reality head-mounted displays. However, conventional log-polar foveated rendering suffers from a number of visual artifacts such as aliasing and flickering. In this paper, we introduce a new log-rectilinear transformation that incorporates summed-area table filtering and off-the-shelf video codecs to enable foveated streaming of 360° videos suitable for VR headsets with built-in eye-tracking. To validate our approach, we build a client-server system prototype for streaming 360° videos which leverages parallel algorithms over real-time video transcoding. We conduct quantitative experiments on an existing 360° video dataset and observe that the log-rectilinear transformation paired with summed-area table filtering heavily reduces flickering compared to log-polar subsampling while also yielding an additional 10% reduction in bandwidth usage.

Index Terms—360 video, foveation, virtual reality, live video streaming, log-rectilinear, summed-area table

1 Introduction

360° videos, also referred to as omnidirectional or panoramic videos, offer superior immersive experiences by encompassing the viewers' entire field of view and allowing them to freely look around the scene with a full 360° field of regard. However, streaming solutions for 360° videos that transmit the entire 360 video frame result in much worse perceived quality compared to streaming conventional videos [11, 35]. As most of the pixels in 360° videos are out-of-sight or in the peripheral region, streaming 360° video requires a much higher resolution and bandwidth to achieve the same perceived quality [34, 35]. Previous research in viewport-adaptive 360° video streaming [15, 30, 34, 40] has achieved significant bandwidth reductions and quality improvements by culling out-of-sight regions for streaming videos to mobile devices. However, with increasing resolutions from 360° cameras such as the 11K Insta360 Titan[1] and VR headsets such as the 3000 pixels-per-inch (PPI) Varjo VR-1[2], additional bandwidth optimizations will be needed for interactive, low-latency streaming of high-resolution 360° videos.

Several existing video processing and transmission pipelines address limitations in transmission speed by varying the resolution to match the human visual system. These foveated video techniques maintain high-fidelity video in regions the viewer is currently focusing on while reducing resolution in the peripheral areas to lower the bit rate necessary to transmit or store the video. Many existing approaches to foveated video coding and streaming [21–23, 27, 36, 37] use either multiple resolution techniques or image transformation techniques.

Multiple resolution techniques divide a video into multiple video tiles and encode each tile at several resolutions. These tiles are usually independently streamed and then combined into a single video on either the server or the client. Alternatively, image transformation techniques map the peripheral areas to a lower resolution by applying a transformation that models the spatially-varying resolution of the human visual system. While both types of techniques are effective at reducing the bit rate, they often lead to spatial and temporal artifacts due to subsampling in the peripheral region [3].

In this paper, we present a new log-rectilinear transformation that preserves full-resolution fidelity around the gaze position and a soft blur in the peripheral region. By bringing together summed-area tables, foveation, and off-the-shelf video codecs, our log-rectilinear transformation enables foveated-video streaming for eye-tracking virtual reality headsets. When incorporating our log-rectilinear transforma-

- *David Li is with University of Maryland, College Park. E-mail: dli7319@umd.edu.*
- *Ruofei Du is with Google LLC. E-mail: me@duruofei.com*
- *Adharsh Babu is with University of Maryland, College Park. E-mail: ababu@umiacs.umd.edu.*
- *Camelia D. Brumar is with University of Maryland, College Park. E-mail: cbrumar@terpmail.umd.edu.*
- *Amitabh Varshney is with University of Maryland, College Park. E-mail: varshney@umd.edu.*

Manuscript received xx xxx. 201x; accepted xx xxx. 201x. Date of Publication xx xxx. 201x; date of current version xx xxx. 201x. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org. Digital Object Identifier: xx.xxxx/TVCG.201x.xxxxxxx

[1] Insta360 Titan: `https://insta360.com`
[2] Varjo VR-1: `https://varjo.com`

tion with summed-area-table-based filtering, image artifacts from the conventional log-polar transformation are significantly reduced.

Our main contributions in this paper are:

- Introduction of a log-rectilinear transformation which leverages summed-area tables, foveation, and standard video codecs for foveated 360° video streaming in VR headsets with eye-tracking.
- Design and implementation of a foveated 360° video streaming system with full capability of video decoding, generating summed-area tables, sampling, and encoding.
- Quantitative evaluation of the log-rectilinear transformation on a public 360° video dataset [2] with an ablation study demonstrating the effects of summed-area table filtering.

The remainder of our paper is structured as follows: In Section 2, we discuss previous approaches to foveated and 360° video coding, streaming, and rendering. In Sections 3 and 4, we discuss the details of our log-rectilinear transformation and how to integrate it into a 360° video streaming pipeline. In Section 5, we quantitatively evaluate the video quality, bit rate, and performance of our log-rectilinear augmentation. Finally, in Sections 6 and 7, we discuss the limitations of our approach and potential future directions in 360° video streaming for VR headsets. Code for our pipeline is available at `https://augmentariumlab.github.io/foveated-360-video/`.

2 Related Works

Our work builds upon previous approaches to 360° video streaming. We leverage techniques from foveated rendering and summed-area tables in our video streaming pipeline.

2.1 Foveated and 360° Video Streaming

As video streaming continues to grow in popularity, various approaches have been proposed to reduce the bandwidth requirements of streaming high-resolution video. Most existing approaches can be classified as multiple resolution techniques [22,23,27,37] or quantization parameter adjustments [18, 26]. For 360° videos and VR applications, tiling techniques [11, 15, 16, 25, 31, 40, 47] stream only visible portions of the video at a high quality and stream out-of-sight portions at a significantly lower quality or even leave them out entirely.

Qian *et al.* developed *Flare* [34], a 360° video-streaming solution for mobile devices, which builds upon existing tiling approaches. The *Flare* system applies the tiling technique to viewport adaptive streaming, which aims to stream the entire viewport of a 360° video at the full-resolution on a mobile network. They develop a viewport-prediction network, a tile scheduler, and employ rate-adaptation in their system to strategically stream the entire viewport while minimizing the bandwidth overhead of streaming out-of-sight regions. While their system successfully culls out-of-sight regions for the 360° video, it can only stream videos at certain predetermined quality levels. Viewing 360° video in high-resolution VR headsets will require a finer foveated streaming approach along with out-of-sight culling to achieve the same quality over a larger viewport.

While tile-based approaches work well for viewport adaptive 360° video streaming, they would not be practical for foveated 360° video streaming where the quality levels differ not only between visible and out-of-sight areas but also between different areas within the viewport. For instance, Ryoo *et al.* [36] design a foveated streaming system for 2D videos which requires 144 tiles with 6 resolutions for each tile for a total of 864 files. Applying the same approach for foveated 360° video streaming would require an order of magnitude more files due to the panoramic nature. Using too few tiles would lead to tearing and color mismatch artifacts as shown in Fig. 2. Splitting up videos this way creates challenges for client devices that would need to stream, decode, and render hundreds of streams simultaneously in sync while also blending them to hide edge artifacts. Furthermore, creating and storing hundreds of files per video is impractical for services such as YouTube and Facebook where videos accumulate over time as users upload more content each day.

Fig. 2: Foveated streaming of 360° videos requires too many tiles due to the large field of view and high variability of detail. Using too few tiles leads to tearing artifacts (blue box) and color mismatch artifacts (orange box) as shown above. The gaze position is marked with a cyan circle at the center of the frame.

2.2 Foveated and 360° Video Rendering

The critical need for higher performance is motivating research in foveated rendering which aims to improve graphics performance in real-time applications.

Guenther *et al.* [12] developed the first foveated rendering pipeline for 3D graphics that renders at three different resolutions based on perceptual detectability thresholds [10] and composites the result to yield the final foveated image. They conducted a user-study to determine an acceptable foveation threshold and evaluated their system by measuring the performance speedup. Their rendering technique achieves an effective speedup of $5-6\times$ on the prevalent displays and they predicted higher speedups on higher-resolution, wider field-of-view displays. Other foveated rendering techniques such as shading at multiple resolutions [32], and rendering to a log-polar buffer [28] have also been found to yield significant performance boost.

Although foveated rendering has the potential to dramatically increase performance for high-resolution VR rendering, many current approaches still yield undesirable visual artifacts. Turner *et al.* [41] present a technique called phase-aligned foveated rendering to reduce motion-induced flickering and aliasing artifacts in foveated rendering. By aligning the pixel sampling to the virtual scene rather than the rotation of the user's head, they can remove flickering caused by rotational movement with only 0.1 ms overhead rendering to a 2560×1440 VR headset. While their technique yields very impressive results, it only works for 3D graphics rendering but not for displaying 360° videos. Recent advances in neural rendering present the potential to reconstruct the foveated frame with generative adversarial neural networks [19]. Nevertheless, such methods are limited by training data and may produce flicker and ghosting artifacts for unexpected content.

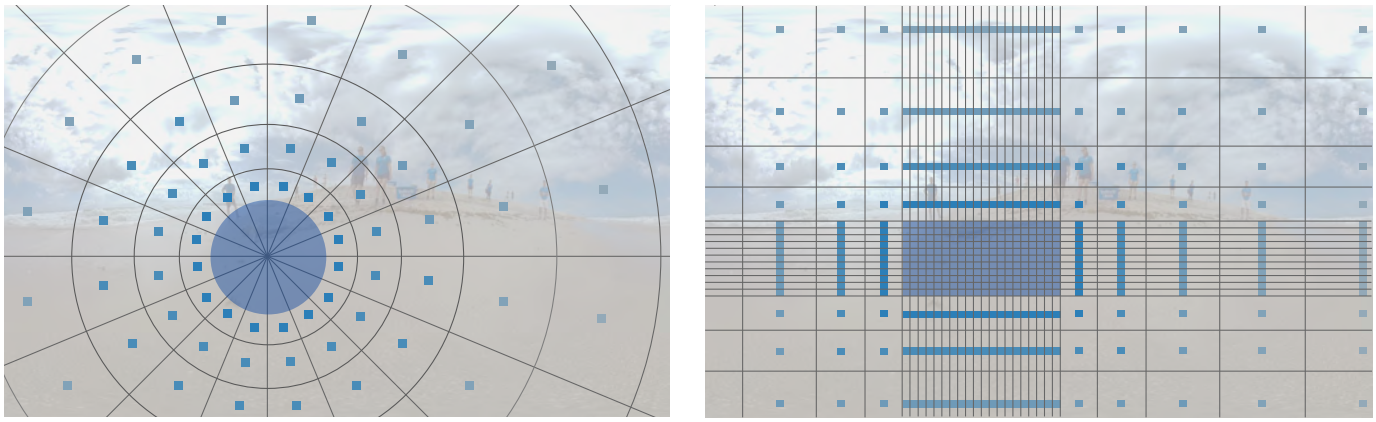
2.3 Summed-Area Tables

Summed-area tables, also known as integral images, are a 2D extension of prefix-sum arrays. First proposed by Frank Crow [6] as an alternative to mipmaps for texturing, summed-area tables have found a wide range of uses within computer graphics and computer vision. Given a 2D array $A = \{a_{ij}\}$, the summed-area table $S = \{s_{ij}\}$ for array A has elements:

$$s_{ij} = \sum_{x=0}^{i} \sum_{y=0}^{j} a_{xy}.$$

The element at position i, j of S is the sum of all the elements in the rectangle sub-array of A with diagonal corners $(0,0)$ and (i, j).

Summed-area tables can be efficiently calculated on the GPU using a variety of parallel algorithms [8, 9, 13, 20, 29]. The simplest parallel algorithm for generating a summed-area table [13] calculates prefix sums across each row in parallel on the GPU followed by each column in parallel. Since a prefix scan can be computed in $O(\log(n))$ time and $O(n)$ work using Blelloch's scan algorithm [5], a summed-area table can be computed in $O(\log(m)+\log(n))$ time and $O(mn)$ work for an $m*n$ image given $m*n$ processors or threads. Recent algorithms [8, 9] leverage memory locality, kernel synchronization, and look-back



a) Log-Polar Transformation b) Log-Rectilinear Transformation

Fig. 3: An illustration of the sampling positions of the log-polar transformation and our log-rectilinear transformation for foveation. Each square corresponds to a single pixel. Larger blocks correspond to collections of pixels. In an ideal scenario, each sampled pixel should correspond to an average of every pixel in their region. With the log-polar transformation, typically only one pixel is sampled in each region leading to foveation artifacts. With our log-rectilinear transformation, we can get the average for each region with just four samples from a summed-area table. The position of the sampling is offset by the gaze position.

techniques to compute summed-area tables with less than 10% overhead over a GPU memory copy. Using the *1R1W-SKSS-LB* algorithm by Emoto *et al.* [8], an $8K \times 8K$ summed-area table of 32-bit floats can be calculated in less than 1 ms.

In computer graphics, summed-area tables have been used for various effects such as ambient occlusion [7], depth of field [13, 24], glossy environmental reflections [13], and feature correspondence [4, 42, 43]. Summed-area tables have also been extended to cube-maps [46] for shadow mapping and translucent environment mapping. To the best of our knowledge, we are the first to use summed-area tables for streaming foveated 360° videos.

3 Method

Our log-rectilinear transformation combines resolution-reduction techniques in foveated rendering and the constant-time filtering effects of summed-area tables to enable foveated video streaming.

3.1 Log-Rectilinear Transformation

The conventional geometric transformation used to emulate the resolution falloff with eccentricity for the human eye is the log-polar transformation [28, 44]. The log-polar transformation emulates the spatially-varying resolution of the human visual system but leads to an inefficient mapping from conventional, rectangular-packed video sources. Around the gaze position, multiple pixels of the log-polar buffer could be sampled from the same pixel in the original image. In the peripheral regions, subsampling from the log-polar buffer causes flickering and aliasing artifacts.

To address the drawbacks of the log-polar transformation, we propose a log-rectilinear transformation. Our log-rectilinear transformation, shown in Fig. 3 and Fig. 4, preserves full-resolution detail near the gaze position while emulating the spatially-varying resolution of the human visual system similar to the existing log-polar transformation. To achieve this, our log-rectilinear transformation satisfies several properties. First, regions of our log-rectilinear transformation are rectangular, allowing constant-time filtering using summed-area tables. Second, our log-rectilinear transformation expresses a one-to-one mapping from the full-resolution video frame to the reduced-resolution buffer near the gaze position. This is represented as $\Delta x = \Delta u$ where Δx is the distance from the gaze position in the full-resolution $W \times H$ video frame and Δu is the distance from the center of the reduced-resolution $w \times h$ buffer. Third, our log-rectilinear transformation uses an exponential resolution decay based on the properties of the human visual system. We accomplish this by using an exponential decay $\Delta x = \exp(A \cdot K(u))$ with a kernel function $K(u) = u^4$ from Meng *et al.* [28]. Here u represents an axis on the reduced-resolution log-polar buffer and A is a variable set to represent the scaling between the full-resolution frame and the reduced-resolution log-polar buffer.

To adapt kernel foveated rendering expression (Equation 8 from Meng *et al.* [28]) for the log-polar formulation to our log-rectilinear transformation we make the following changes. First, we replace u with $\frac{|\Delta u|}{w/2} \in [0,1]$, the normalized distance from the center of the reduced-resolution buffer. Second, we subtract 1 from the exponential so that $|\Delta x| = 0$ when $|\Delta u| = 0$. Third, we move the variable A out of the exponential and set it to a constant λ_x. We set $\lambda_x = \frac{W}{e-1}$ so that $|\Delta x| = |W|$ when $\frac{|\Delta u|}{w/2} = 1$, allowing the entire frame to be in view when the user is looking at a corner. Our final exponential decay is $\lambda_x \left(\exp\left(\left(\frac{|\Delta u|}{w/2} \right)^4 \right) - 1 \right)$. To ensure a one-to-one mapping near the gaze position, we take the maximum between $|\Delta u|$ and our exponential decay, giving us our final equation:

$$\Delta x = \max\left(|\Delta u|,\ \lambda_x \left(\exp\left(\left(\frac{|\Delta u|}{w/2} \right)^4 \right) - 1 \right) \right) * \operatorname{sign}(\Delta u).$$

Inverting our log-rectilinear transformation is accomplished with the following equation:

$$\Delta u = \min\left(|\Delta x|, \frac{w}{2} \cdot \ln^{1/4}\left(\frac{|\Delta x|}{\lambda_x} + 1 \right) \right) \cdot \operatorname{sign}(\Delta x).$$

We detail how the log-rectilinear transformation and its inverse are applied to foveated streaming in Sect. 4.

3.2 Summed-Area Table Images

Although our log-rectilinear transformation maintains the high-quality in the fovea region using a one-to-one mapping, it alone can not address aliasing artifacts in the peripheral region. We propose sampling from a summed-area table rather than directly from the image to reduce the artifacts from foveated sampling.

Using summed-area tables allows us to quickly find the sum of any axis-aligned rectangular block of the original array A:

$$\sum_{x=i_0}^{i_1} \sum_{y=j_0}^{j_1} a_{xy} = s_{i_1 j_1} - s_{(i_0-1) j_1} - s_{i_1 (j_0-1)} + s_{(i_0-1)(j_0-1)}.$$

Dividing the sum by the size of the rectangle yields the average of all values in the block. Using the average RGB values for peripheral regions significantly improves the quality compared to sampling

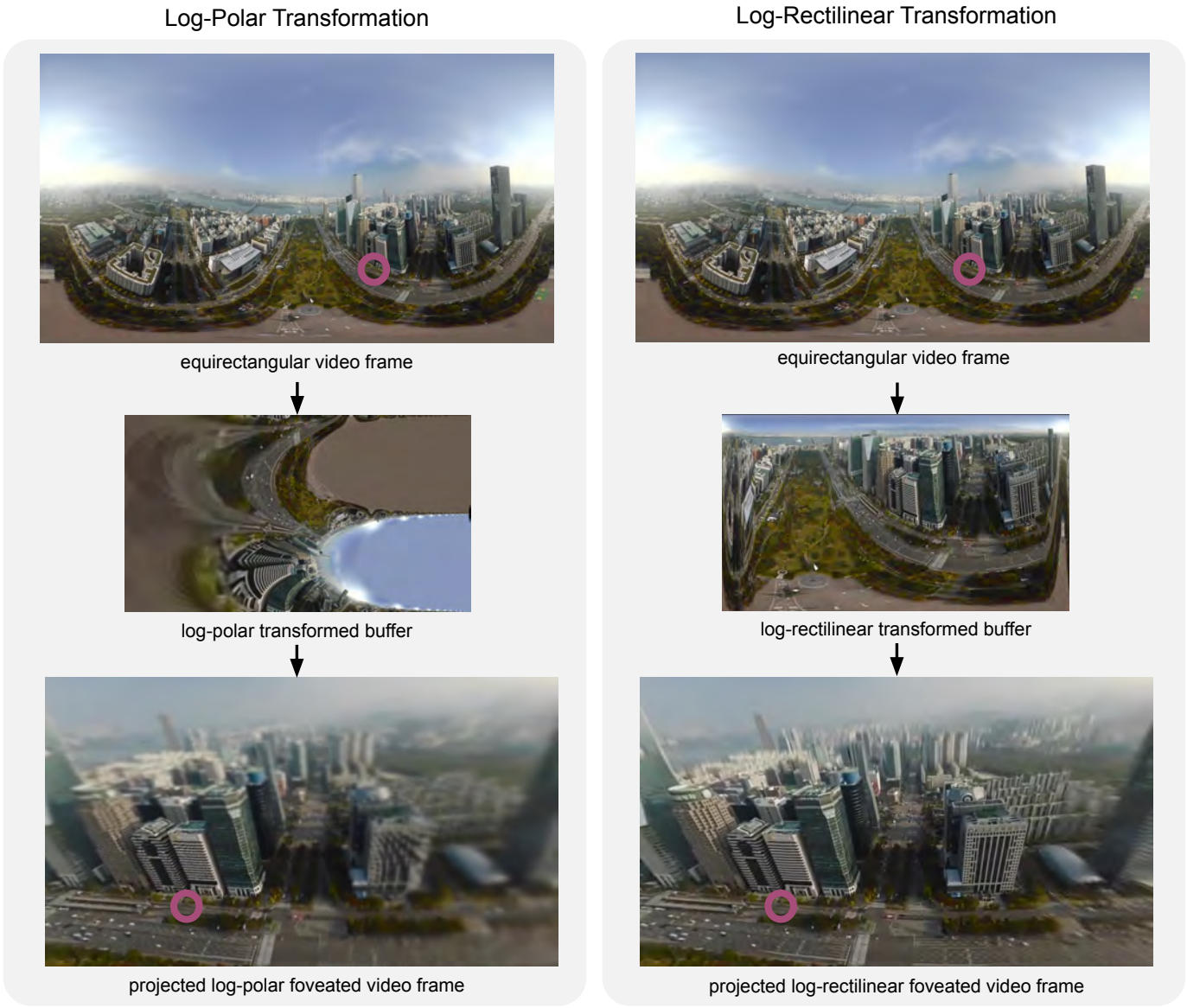


Fig. 4: A comparison between the conventional log-polar transformation and our novel log-rectilinear transformation. The left and right columns show the same video frame being foveated using the log-polar and log-rectilinear transformations, respectively. The lowermost image in each column shows the final gnomonic projected video frame with foveation. The gaze position is marked with a purple circle. For both log-polar and log-rectilinear, we use a buffer resolution of 1072×608.

the original image, eliminating aliasing artifacts as well as reducing temporal flickering. When sampling from the summed-area table, we are able to retrieve the average color values with only four memory reads per pixel. For a $m * n$ frame, traditional filtering for foveation requires $O(mn)$ work after the gaze position is available. However, on the highly-parallel GPUs one could use the summed-area tables to carry out filtering much faster. Building the summed-area table on a GPU using parallel-prefix sums takes only $O(\log(m) + \log(n))$ time over $m * n$ processors and does not require knowing the latest gaze position. As soon as the gaze position is available, sampling the summed-area table only takes $O(1)$ time, reducing the latency in the sampling stage. By reducing the overhead in sampling, we minimize the latency between receiving the latest gaze position of the viewer and sending out the next frame on the server.

4 System

We demonstrate the applicability of our transformation by designing and implementing a foveated video streaming pipeline for eye-tracking VR headsets. Our pipeline consists of a real-time video transcoding pipeline with two additional steps on the server: a summed-area-table encoding step and a log-rectilinear sampling step. Our full pipeline, shown in Fig. 5, consists of four processing stages for the server and two processing stages for the client.

4.1 Server Pipeline

Our workflow on the server consists of four stages: video decoding, summed-area table generation, log-rectilinear buffer sampling, and log-rectilinear buffer encoding. Every frame of the video must be processed through the pipeline, but decoding and summed-area table generation are buffered on the server.

4.1.1 Stage 1: Video Decoding

Our first stage for the server is video decoding. In this stage, the full-resolution video is decoded on the server and converted from YCbCr color-space into a 24 bits-per-pixel RGB image. In our research prototype, we use FFmpeg, a multimedia library (`https://FFmpeg.org`),

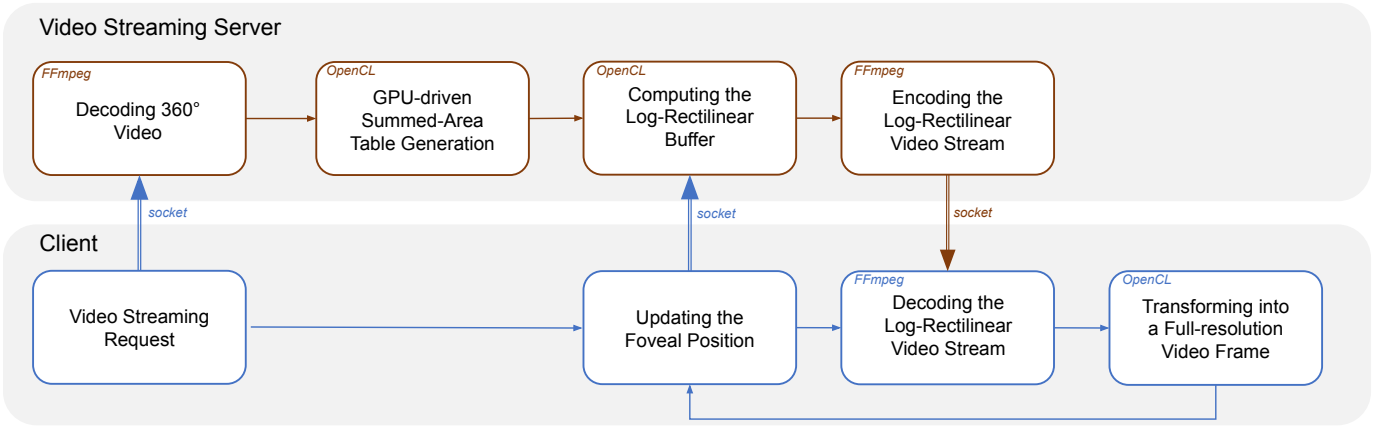


Fig. 5: The system workflow for our integrated foveated video streaming prototype. Our server consists of four stages which include decoding the video, building a summed-area table, sampling the summed-area table into a log-rectilinear buffer, and streaming the log-rectilinear video buffers. The client first sends a video streaming request to the server, then continuously updates the gaze position to the streaming server. For video processing, the client has two stages: decoding the log-rectilinear video and interpolating it into a full-resolution video frame.

for both video decoding and encoding. The decoding process can be hardware accelerated on Intel, NVIDIA, and AMD platforms using FFmpeg.

4.1.2 Stage 2: Summed-Area Table Generation

In the second stage, our server computes the summed-area table representation for the full-resolution image. We use the RGB color-space and compute the summed-area tables for each channel independently, though our method can also be applied to other color-spaces such as YCbCr or HSV. Our research prototype leverages the GPU by using the OpenCL API for parallel computation.

4.1.3 Stage 3: Log-Rectilinear Sampling

In the third stage, we compute a reduced-resolution log-rectilinear buffer. For a full-resolution 1920×1080 ($W \times H$) video, we sample to a reduced-resolution buffer of size 1072×608 ($w \times h$) so that the ratio of full-resolution to reduced-resolution (W/w) is at least 1.8, a ratio found to yield virtually indistinguishable results for users with log-polar-based foveation in VR headsets [28]. During this stage, we use the viewer's last updated gaze position when creating the reduced-resolution buffer. We use OpenCL for computing the log-rectilinear buffer.

4.1.4 Stage 4: Log-Rectilinear Encoding

The final stage of our pipeline consists of encoding the reduced-resolution log-rectilinear buffer into an H.264 video packet and muxing into a fragmented MP4 (fMP4) packet. Once again, we use FFmpeg for the encoding stage which supports hardware-accelerated encoding through NVIDIA NVENC. After encoding, the packet gets sent to the client over the network through a socket.

4.2 Client Pipeline

Our pipeline for the client augments video decoding with a post-processing step to convert the log-rectilinear transformed buffer into a standard video frame.

4.2.1 Stage 1: Video Decoding

Upon receiving the encoded packet from the server, the client decodes the packet yielding the reduced-resolution log-rectilinear buffer. Although the end-user never sees the log-rectilinear buffer, the buffer is cached on the GPU for post-processing.

4.2.2 Stage 2: Inverse Log-Rectilinear Transformation

In the second stage, the client performs a post-processing step to expand the reduced-resolution log-rectilinear buffer into a full-resolution foveated video frame. We employ bilinear interpolation on the GPU to restore the full-resolution video frame.

5 Evaluation

We evaluate the potential benefits and drawbacks of our approach by conducting a quantitative evaluation of our visual quality, bandwidth savings, streaming latency, and computational overhead. We compare our log-rectilinear transformation with summed-area table sampling against the log-polar transformation commonly used in foveated rendering. Throughout our evaluation, we also include an ablation study of the summed-area table on public datasets. We present a side-by-side visual comparison between foveation using the log-polar transformation and our approach in the supplementary video.

5.1 Datasets and Configuration

To validate our approach and ensure replicability, we quantitatively evaluate our video streaming system with the benchmark 360° video dataset of Agtzidis *et al.* [2]. The dataset consists of 14 diverse 360° videos with gaze and head paths of 13 participants using a FOVE[3] eye-tracking VR headset. We conduct quality, bandwidth, and performance overhead comparisons on all the available 172 (participant, video) pairs and report the aggregate averaged metrics. Detailed results for each video are available in the supplementary material. For visual quality, we present a side-by-side comparison with the drone video in our supplementary video and quantitatively evaluate the corresponding visual quality for log-polar foveation, log-rectilinear foveation, and log-rectilinear foveation with SAT. For a break-down analysis of performance, we measure latency and processing time also with the drone video in the dataset. Performance measurements are similar for other videos in the dataset.

We conduct all the following experiments on a server with an Intel Core i7-8700K CPU and NVIDIA RTX 2080 Ti GPU and a client with an Intel Core i5-5300H CPU and NVIDIA GTX 1050 GPU. For our benchmarks, decoding is accomplished on the CPU by x264[4] and encoding is performed on the GPU by NVIDIA NVENC using FFmpeg API. Our overall experiments and measurements are performed on the $1080p$ (1920×1080) equirectangular projected frame of each 360° video and using a 1072×608 reduced-resolution buffer. As videos in the original dataset have varying resolutions up to 3840×2160, we pre-process the videos to 1920×1080 for our experiments. For the log-polar method, we apply a 3×3 Gaussian blur to the right-half of the log-polar encoded buffer following Meng *et al.* [28] during the sampling stage. No blur is applied for our log-rectilinear method.

[3] FOVE VR Headset: `https://www.getfove.com`
[4] x264: `https://www.videolan.org/developers/x264.html`

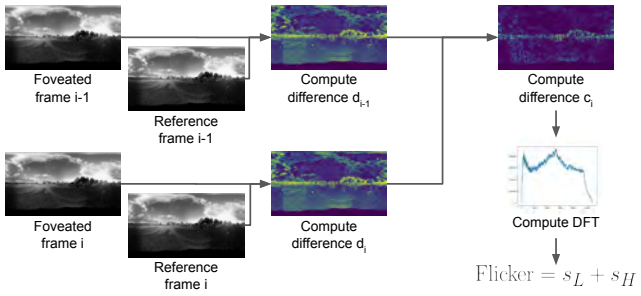


Fig. 6: Overview of the Flicker metric by Winkler *et al.* [45] which captures the change in visual artifacts using a linear combination of Fourier coefficients.

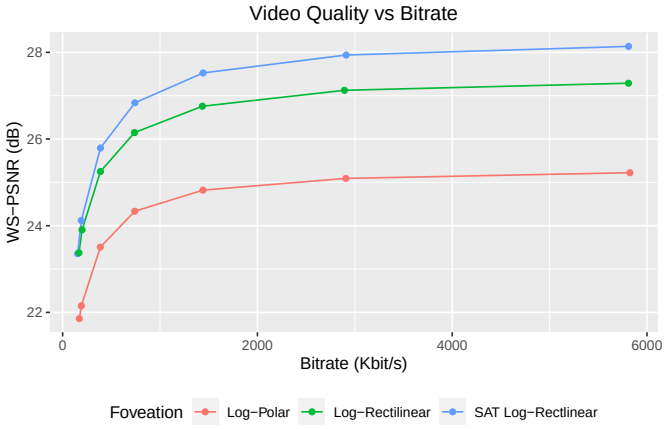


Fig. 7: Comparison of visual quality (WS-PSNR) across bitrates ranging from 100 Kbits/s to 6 Mbits/s. Foveation with SAT Log-Rectilinear yields better quality across the entire range of bitrates.

5.2 Quality

To evaluate the visual quality of our approach, we compare the differences between our foveated video and the original video when encoding the reduced-resolution representations in a constant quality mode. We compute the weighted spherical peak signal-to-noise ratio (WS-PSNR) [39] of the luminance channel (Y) because the eye is more sensitive to changes in luminance as opposed to changes in chrominance [17]. We also compute the Structural Similarity Index (SSIM) between the original videos and the foveated videos.

We adapt the metric by Winkler *et al.* [45] to measure flickering. We first compute the difference in luminance between foveated frames $\{x_i\}$ and reference frames $\{y_i\}$ to yield deltas $\{d_i = x_i - y_i\}$. Then we compute the difference between consecutive deltas $\{c_i = d_i - d_{i-1}\}$. For each difference c_i, we compute the discrete Fourier transform which gives a vector of Fourier coefficients $\mathbf{r}_i$. Next, we compute a sum s_L over low frequencies and a sum s_H over high frequencies to get a per-frame flicker $s_L + s_H$. Finally, we average the flickering across all consecutive frames in the video to get a per-video flicker value. As each video in the dataset is of a single scene, we evenly weigh every frame when computing the total per-video Flicker metric.

Our final Flicker metric is as follows:

$$
\begin{aligned}
d_i &= x_i - y_i \\
\mathbf{r}(i) &= \mathrm{DFT}(d_i - d_{i-1}) \\
s_L(i) &= \frac{1}{f_M - f_L} \sum_{k=f_L}^{f_M} r_k(i), \\
s_H(i) &= \frac{1}{f_H - f_M} \sum_{k=f_M}^{f_H} r_k(i), \\
\mathrm{Flicker} &= \frac{1}{N-1} \sum_{i=2}^{N} (s_L(i) + s_H(i))
\end{aligned}
$$

where N is the total number of frames, $\{x_i\}_{i=1}^N$ are frames of the foveated video, $\{y_i\}_{i=1}^N$ are frames of the reference video, DFT represents computing Fourier coefficients, and f_L, f_M, f_H are predefined frequency limits. As in Winkler *et al.* [45], we use frequency limits of $f_L = 1\%$, $f_M = 16\%$, and $f_H = 80\%$ relative to the maximum frequency. An illustration of this Flicker metric is shown in Fig. 6.

We show the averaged results of our quality comparison across all (participant, video) combinations with intermediate encoding of transformed buffers in Table 1 and without intermediate encoding in Table 2. Per-video averaged results with intermediate encoding are shown in Fig. 8, Fig. 9, Fig. 10, Fig. 11, and available in the Table 6. We also show how the WS-PSNR scales with available bandwidth in Fig. 7 using the gaze of the first available participant (P3).

5.3 Bandwidth

We measure the average bitrate of the H.264 stream produced by FFmpeg to determine the compression ratio achieved by our foveation technique. All pipelines encode their representations into the main profile of H.264 with the constant quality parameter (`cq`) set to 25. Bitrates and the corresponding per-frame packet sizes for each pipeline are shown in Table 3.

In our implementation, the server reads and decodes a single full-resolution 1080p H.264 MP4 file. No lower-resolution copies of the video are stored on disk or used during the decoding process. Our server calculates the summed-area table in real-time and buffers it in GPU memory. For a 1080p frame, this requires only 24 MB of GPU memory when storing the summed-area table using 32-bit integers.

Table 1: Quality comparison of various foveation methods based on the aggregate average values of WS-PSNR, SSIM, and Flicker metrics. Our SAT Log-Rectilinear method yields significantly less flickering with comparable visual fidelity.

Sampling Method	WS-PSNR (db) ↑	SSIM ↑	Flicker ↓
Log-Polar	25.18	0.864	160.8
Log-Rectilinear	27.23	0.906	192.8
SAT Log-Rectilinear	28.00	0.908	110.0

Table 2: Quality comparison of various foveation methods without encoding intermediate log-polar and log-rectilinear buffers to H.264.

Sampling Method	WS-PSNR (db) ↑	SSIM ↑	Flicker ↓
Log-Polar	25.47	0.877	161.4
Log-Rectilinear	27.48	0.918	196.4
SAT Log-Rectilinear	28.50	0.921	98.4

Table 3: Bandwidth evaluation of different video streaming methods based on the average packet size and bit rate. Our SAT log-rectilinear method yields significant bandwidth savings compared to other methods when encoding with H.264 in constant quality mode.

Sampling Method	Average Packet Size ↓	Bit rate ↓
Full Resolution	24.50 KB	5.88 Mbps
Log-Polar	13.91 KB	3.34 Mbps
Log-Rectilinear	15.50 KB	3.72 Mbps
SAT Log-Rectilinear	12.44 KB	2.99 Mbps

Table 4: Performance comparison of video streaming based on various foveation methods. Our Summed-Area Table (SAT) Log-Rectilinear pipeline requires an additional 1 to 2 ms compared to other pipelines which sample directly from the raw video frames.

Sampling Method	Decoding (ms)	Processing (ms)	Sampling (ms)	Encoding (ms)	Total (ms)
Log-Polar	6.14	1.91	0.55	2.86	11.46
Log-Rectilinear	6.13	1.91	0.53	2.85	11.43
SAT Log-Rectilinear	6.14	3.00	0.46	2.84	12.44

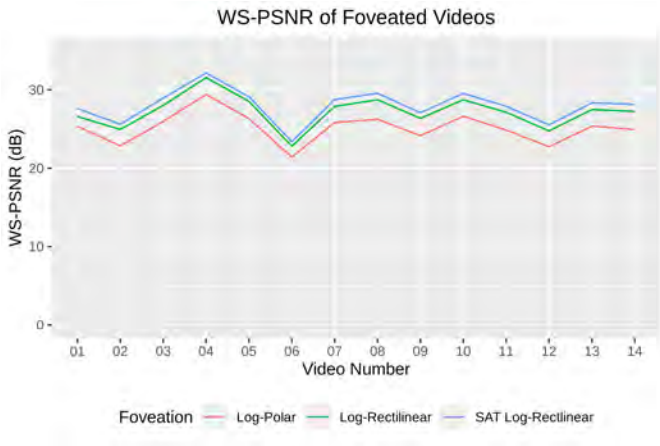


Fig. 8: Quantitative evaluation of the average weighted spherical peak-signal-to-noise-ratio (WS-PSNR) in streaming each foveated video. Our SAT Log-Rectilinear method yields the highest WS-PSNR for all videos.

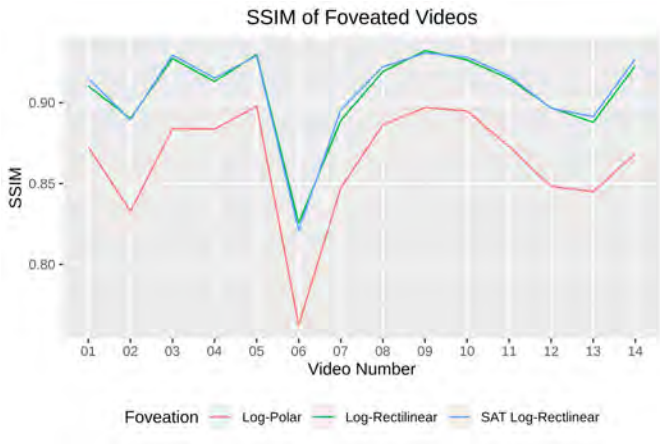


Fig. 9: Quantitative evaluation of the average Structural similarity index (SSIM) in streaming each foveated video. Our SAT Log-Rectilinear method yields the highest SSIM for most videos.

Table 5: Break-down analysis of the client latency. Our profiling results show that foveation only adds 7 milliseconds to the overall latency when responding to user interactions.

Stage	Non-foveated Client Runtime (ms)	Foveated Client Runtime (ms)
Server Response	82 ms	76 ms
Decoding	27 ms	27 ms
Unwarping	0 ms	13 ms
Total	109 ms	116 ms

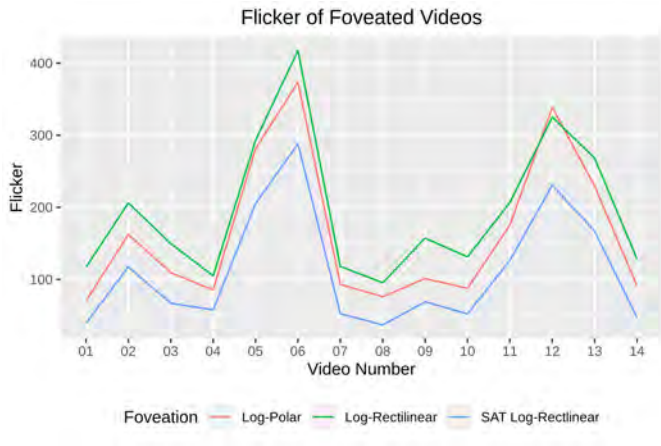


Fig. 10: Quantitative evaluation of our flickering metrics for foveating each video in the dataset. Our SAT Log-Rectilinear method yields the lowest flickering for every video.

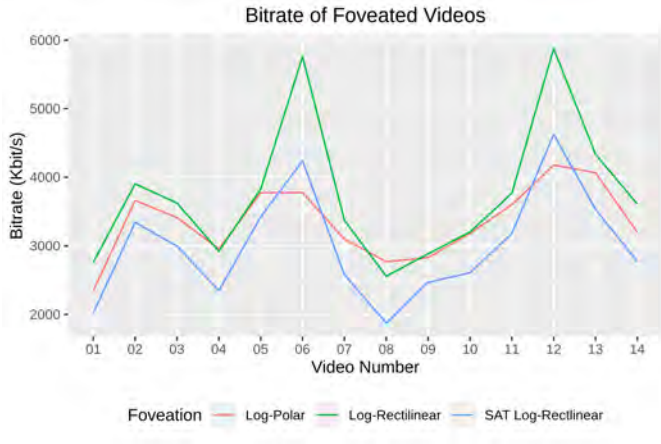


Fig. 11: Measurements of the bitrate for streaming each video when encoding using a constant quality mode to H.264. On average, our SAT Log-Rectilinear method results in a 23% reduction in bandwidth consumption compared to Log-Polar foveation.

5.4 Computational Overhead

To evaluate the streaming overhead of our log-rectilinear foveation, we compare the overall response time of a non-foveated client and a foveated video client to user input. As shown in Table 4, our log-rectilinear transformation yields similar runtime for sampling and encoding foveated video, adding only an additional 2 ms overhead when building the summed-area table. According to Table 5, our foveated pipeline adds only about 7 ms of overall latency responding to user input compared with a non-foveated pipeline. Foveation reduces the total server response time from 82 ms to 76 ms (7.3%) due to the reduced resolution the server needs to encode and the smaller packet sizes.

We implement three foveation pipelines mainly in C++ and measure the average time at each stage of the pipelines to quantify the perfor-

Table 6: Complete Results of our Visual Quality and Bandwidth Evaluation: We include full evaluation results of WS-PSNR, SSIM and bitrate for streaming each of the 14 videos in the benchmark dataset by Agtzidis *et al.* [2]. Each metric is averaged over the available participants for the corresponding video.

Video	# Participants	Foveation	WS-PSNR (db)	SSIM	Bitrate (Mb/s)	Packet Size	Flicker
01_park	13	Log-Polar	25.34	0.873	2.34	9.76	70.12
		Log-Rectilinear	26.59	0.910	2.76	11.50	116.81
		SAT Log-Rectlinear	27.60	0.915	2.02	8.42	38.97
02_festival	13	Log-Polar	22.84	0.833	3.66	15.25	162.32
		Log-Rectilinear	24.94	0.890	3.90	16.27	206.09
		SAT Log-Rectlinear	25.61	0.889	3.34	13.92	117.27
03_drone	13	Log-Polar	25.96	0.884	3.41	14.22	109.29
		Log-Rectilinear	28.01	0.927	3.62	15.09	149.81
		SAT Log-Rectlinear	28.94	0.929	2.99	12.48	66.88
04_turtle_rescue	13	Log-Polar	29.41	0.884	2.96	12.31	85.47
		Log-Rectilinear	31.56	0.913	2.92	12.17	104.94
		SAT Log-Rectlinear	32.13	0.915	2.35	9.78	57.81
05_cycling	13	Log-Polar	26.32	0.898	3.77	15.72	281.49
		Log-Rectilinear	28.52	0.930	3.83	15.95	293.00
		SAT Log-Rectlinear	29.10	0.929	3.42	14.26	205.21
06_forest	12	Log-Polar	21.41	0.763	3.78	15.74	373.50
		Log-Rectilinear	22.80	0.826	5.76	24.00	417.51
		SAT Log-Rectlinear	23.36	0.821	4.24	17.66	288.02
07_football	12	Log-Polar	25.82	0.847	3.10	12.91	93.11
		Log-Rectilinear	27.88	0.889	3.37	14.05	117.98
		SAT Log-Rectlinear	28.76	0.896	2.58	10.74	52.52
08_courtyard	12	Log-Polar	26.23	0.886	2.77	11.53	76.02
		Log-Rectilinear	28.72	0.919	2.56	10.65	95.46
		SAT Log-Rectlinear	29.55	0.922	1.88	7.81	36.80
09_expo	12	Log-Polar	24.14	0.897	2.83	11.78	101.21
		Log-Rectilinear	26.36	0.932	2.88	12.01	157.25
		SAT Log-Rectlinear	27.07	0.931	2.47	10.28	68.93
10_eiffel_tower	12	Log-Polar	26.63	0.895	3.18	13.26	87.74
		Log-Rectilinear	28.73	0.926	3.20	13.34	131.27
		SAT Log-Rectlinear	29.54	0.928	2.61	10.86	52.23
11_chicago	12	Log-Polar	24.86	0.873	3.60	15.00	175.60
		Log-Rectilinear	27.13	0.915	3.77	15.69	206.80
		SAT Log-Rectlinear	27.89	0.917	3.18	13.25	126.44
12_driving	11	Log-Polar	22.73	0.848	4.18	17.41	338.50
		Log-Rectilinear	24.76	0.897	5.88	24.48	325.10
		SAT Log-Rectlinear	25.52	0.896	4.63	19.28	230.83
13_drone_low	12	Log-Polar	25.37	0.845	4.07	16.94	229.20
		Log-Rectilinear	27.47	0.888	4.33	18.06	268.07
		SAT Log-Rectlinear	28.34	0.891	3.53	14.72	166.71
14_cats	12	Log-Polar	24.91	0.868	3.19	13.31	91.06
		Log-Rectilinear	27.24	0.923	3.61	15.04	128.21
		SAT Log-Rectlinear	28.13	0.927	2.77	11.54	46.90

mance impact of our sampling method compared with other foveation sampling methods. During streaming, we observe that GPU utilization on the server is around 11%. Thus, the server supports streaming to multiple connections at 1080p on a single GPU. However, our current implementation is limited to 1080p due to bottlenecks in CPU video decoding.

6 Discussion

Our experiments show that sampling with our log-rectilinear transformation using a summed-area table yields reduced flickering and reduced bit rate compared with using a log-polar approach sampling from the video frames directly. For most videos with still or slow camera movements, our log-rectilinear transformation paired with a summed-area table dramatically reduces flicker. In videos with abnormally large camera movements, such as the cycling video and the driving video, our flickering metric yields very high values for all three foveation approaches due to the large luminance changes between frames.

In our performance comparison, we see that our pipeline consumes only 1 to 2 milliseconds of additional processing time compared with other foveation approaches. Although our summed-area approach reads four pixels from the summed-area table buffer during the sampling stage, sampling still takes less than 0.6 ms in our experiments as shown in Table 4. This can be explained by caching between threads within the same work-group. While each thread reads four pixels from the full-resolution summed-area table buffer, the same values are read by neighboring threads allowing the GPU to efficiently cache values from the summed-area table. To produce a $w \times h$ size log-rectilinear buffer, only $(w+1) \times (h+1)$ values are read in total from the summed-area table. For all sampling methods, the server processing time is within $10-15$ ms, similar to other cloud-driven AR and VR systems [38, 48].

Comparing the encoded packet sizes between all the approaches, we see that all foveation methods yield significantly reduced packet sizes compared with encoding at the full resolution, often with over 50% bandwidth savings. With significantly reduced bit rates, we envision that our technique may be useful in Content Delivery Network (CDN) and edge computing devices to reduce the bandwidth needed for Internet streaming of high-resolution 360° videos for eye-tracking VR headsets.

Limitations

Despite yielding better metrics compared with other approaches, our technique requires real-time server-side video processing for each viewing session. With this limitation in mind, we expect our technique to be well suited for high-performance but low-bandwidth situations. For instance, popular movies and videos can be cached in a nearby CDN edge server and foveated for mobile VR video streaming to a wireless HMD. Recently, Google Stadia[5] and Microsoft Project xCloud[6] empower users to play video games on the cloud at a resolution up to 4K at 60 FPS. As more virtual reality games debut on the market, we expect foveated video streaming will become necessary for virtual reality games to be played on these cloud services over existing and upcoming network infrastructures.

While we have designed a full 360° video streaming pipeline to demonstrate the applicability of our transformation, this implementation is not the primary contribution of this paper. Our pipeline lacks features such as gaze prediction, rate adaptation, frame buffering, and other system-level optimizations that would be required for a fully-featured streaming service. As such, our evaluation focuses on comparing our log-rectilinear transformation to the conventional log-polar transformation for video streaming rather than an extensive Quality of Experience (QoE) evaluation against existing systems such as Flare [34]. Standardizing evaluations of 360° video streaming systems is an ongoing challenge for multimedia systems researchers [1]. We leave the development of a fully-featured production-grade video pipeline and the QoE evaluation of the proposed pipeline for future work.

7 Conclusion

In this paper, we have presented a log-rectilinear transformation that combines foveation, summed-area tables, and off-the-shelf video encoding to enable 360° foveated video streaming for eye-tracking VR headsets. To evaluate our novel transformation, we have designed and implemented a 360° foveated video streaming pipeline for the log-polar and our log-rectilinear transformation. Our evaluation measuring the quality, performance, and storage aspects shows that our log-rectilinear transformation reduces flickering when paired with summed-area table filtering.

In the future, we plan to extend our pipeline to further reduce artifacts and improve quality for foveated video streaming. Incorporating gaze prediction and viewport prediction [14, 33] would allow short-term buffering on the client. Combining our transcoding technique with previous work on pre-processed multi-resolution blocks [34] could lead to reduced server load for 360° applications.

With the ever-increasing resolutions of VR headsets and 360° cameras and the growing popularity of cloud-based gaming platforms, we believe our novel log-rectilinear transformation will make 360° VR content more immersive and accessible for everyone.

References

[1] S. Aggarwal, S. Paul, P. Dash, N. S. Illa, Y. C. Hu, D. Koutsonikolas, and Z. Yan. How to evaluate mobile 360° video streaming systems? In *Proceedings of the 21st International Workshop on Mobile Computing Systems and Applications*, HotMobile '20, p. 68–73. Association for Computing Machinery, New York, NY, USA, 2020. doi: 10.1145/3376897.3377865

[2] I. Agtzidis, M. Startsev, and M. Dorr. 360-degree video gaze behaviour: A ground-truth data set and a classification algorithm for eye movements. In *Proceedings of the 27th ACM International Conference on Multimedia (MM '19)*. ACM, 2019. doi: 10.1145/3343031.3350947

[3] R. Albert, A. Patney, D. Luebke, and J. Kim. Latency requirements for foveated rendering in virtual reality. *ACM Trans. Appl. Percept.*, 14(4), Sept. 2017. doi: 10.1145/3127589

[4] H. Bay, T. Tuytelaars, and L. Van Gool. Surf: Speeded up robust features. In A. Leonardis, H. Bischof, and A. Pinz, eds., *Computer Vision – ECCV 2006*, pp. 404–417. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[5] G. E. Blelloch. Prefix Sums And Their Applications. In *Synthesis of Parallel Algorithms*. M. Kaufmann, 5 2004. doi: 10.1184/R1/6608579.v1

[6] F. C. Crow. Summed-area tables for texture mapping. In *Proceedings of the 11th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '84, pp. 207–212. ACM, 1984. doi: 10.1145/800031.808600

[7] J. Díaz, P.-P. Vázquez, I. Navazo, and F. Duguet. Real-time ambient occlusion and halos with summed area tables. *Computers & Graphics*, 34(4):337 – 350, 2010. Procedural Methods in Computer Graphics Illustrative Visualization. doi: 10.1016/j.cag.2010.03.005

[8] Y. Emoto, S. Funasaka, H. Tokura, T. Honda, K. Nakano, and Y. Ito. An optimal parallel algorithm for computing the summed area table on the GPU. In *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pp. 763–772, May 2018. doi: 10.1109/IPDPSW.2018.00123

[9] S. Funasaka, K. Nakano, and Y. Ito. Single kernel soft synchronization technique for task arrays on CUDA-enabled GPUs, with applications. In *2017 Fifth International Symposium on Computing and Networking (CANDAR)*, pp. 11–20, Nov 2017. doi: 10.1109/CANDAR.2017.35

[10] W. S. Geisler and J. S. Perry. Real-time foveated multiresolution system for low-bandwidth video communication. In B. E. Rogowitz and T. N. Pappas, eds., *Human Vision and Electronic Imaging III*, vol. 3299, pp. 294 – 305. International Society for Optics and Photonics, SPIE, 1998. doi: 10.1117/12.320120

[11] Y. Guan, C. Zheng, X. Zhang, Z. Guo, and J. Jiang. Pano: Optimizing 360° video streaming with a better understanding of quality perception. In *Proceedings of the ACM Special Interest Group on Data Communication*, SIGCOMM '19, p. 394–407. Association for Computing Machinery, New York, NY, USA, 2019. doi: 10.1145/3341302.3342063

[12] B. Guenter, M. Finch, S. Drucker, D. Tan, and J. Snyder. Foveated 3d graphics. *ACM Trans. Graph.*, 31(6):164:1–164:10, Nov. 2012. doi: 10.1145/2366145.2366183

[13] J. Hensley, T. Scheuermann, G. Coombe, M. Singh, and A. Lastra. Fast summed-area table generation and its applications. *Computer Graphics Forum*, 24(3):547–555, 2005. doi: 10.1111/j.1467-8659.2005.00880.x

[14] J. Heyse, M. T. Vega, F. de Backere, and F. de Turck. Contextual bandit learning-based viewport prediction for 360 video. In *2019 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, pp. 972–973, March 2019. doi: 10.1109/VR.2019.8797830

[15] M. Hosseini. View-aware tile-based adaptations in 360 virtual reality video streaming. In *2017 IEEE Virtual Reality (VR)*, pp. 423–424, March 2017. doi: 10.1109/VR.2017.7892357

[16] M. Hosseini and V. Swaminathan. Adaptive 360 VR video streaming: Divide and conquer. In *2016 IEEE International Symposium on Multimedia (ISM)*, pp. 107–110, Dec 2016. doi: 10.1109/ISM.2016.0028

[17] D. H. Hubel. *Eye, brain, and vision.* Scientific American Library/Scientific American Books, 1995.

[18] G. Illahi, M. Siekkinen, and E. Masala. Foveated video streaming for cloud gaming. In *2017 IEEE 19th International Workshop on Multimedia Signal Processing (MMSP)*, pp. 1–6, Oct 2017. doi: 10.1109/MMSP.2017.8122235

[19] A. S. Kaplanyan, A. Sochenov, T. Leimkühler, M. Okunev, T. Goodall, and G. Rufo. Deepfovea: Neural Reconstruction for Foveated Rendering and Video Compression Using Learned Statistics of Natural Videos. *ACM Transactions on Graphics (TOG)*, 38(6):212, 2019. doi: 10.1145/3306307.3328186

[20] A. Kasagi, K. Nakano, and Y. Ito. Parallel algorithms for the summed area table on the asynchronous hierarchical memory machine, with GPU implementations. In *2014 43rd International Conference on Parallel Processing*, pp. 251–260, Sep. 2014. doi: 10.1109/ICPP.2014.34

[21] H. Kim, J. Yang, M. Choi, J. Lee, S. Yoon, Y. Kim, and W. Park. Eye tracking based foveated rendering for 360 VR tiled video. In *Proceedings of the 9th ACM Multimedia Systems Conference*, MMSys '18, p. 484–486. Association for Computing Machinery, New York, NY, USA, 2018. doi: 10.1145/3204949.3208111

[22] O. Komogortsev and J. Khan. Predictive perceptual compression for real time video communication. In *Proceedings of the 12th Annual ACM International Conference on Multimedia*, MULTIMEDIA '04, pp. 220–227. ACM, New York, NY, USA, 2004. doi: 10.1145/1027527.1027577

[23] W.-T. Lee, H.-I. Chen, M.-S. Chen, I.-C. Shen, and B.-Y. Chen. High-resolution 360 video foveated stitching for real-time VR. *Computer Graphics Forum*, 36(7):115–123, 2017. doi: 10.1111/cgf.13277

[24] K. Lei and J. F. Hughes. Approximate depth of field effects using few samples per pixel. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, I3D '13, pp. 119–128. ACM, New

[5] Google Stadia: https://stadia.google.com

[6] Project xCloud: https://xbox.com/xbox-game-streaming/project-xcloud

York, NY, USA, 2013. doi: 10.1145/2448196.2448215

[25] X. Liu, Q. Xiao, V. Gopalakrishnan, B. Han, F. Qian, and M. Varvello. 360° innovations for panoramic video streaming. In *Proceedings of the 16th ACM Workshop on Hot Topics in Networks*, HotNets-XVI, pp. 50–56. ACM, New York, NY, USA, 2017. doi: 10.1145/3152434.3152443

[26] Y. Liu, Z. G. Li, and Y. C. Soh. Region-of-interest based resource allocation for conversational video communication of h.264/avc. *IEEE Transactions on Circuits and Systems for Video Technology*, 18(1):134–139, Jan 2008. doi: 10.1109/TCSVT.2007.913754

[27] P. Lungaro, R. Sjöberg, A. J. F. Valero, A. Mittal, and K. Tollmar. Gaze-aware streaming solutions for the next generation of mobile VR experiences. *IEEE Transactions on Visualization and Computer Graphics*, 24(4):1535–1544, April 2018. doi: 10.1109/TVCG.2018.2794119

[28] X. Meng, R. Du, M. Zwicker, and A. Varshney. Kernel foveated rendering. *Proc. ACM Comput. Graph. Interact. Tech.*, 1(1):5:1–5:20, July 2018. doi: 10.1145/3203199

[29] D. Nehab, A. Maximo, R. S. Lima, and H. Hoppe. GPU-efficient recursive filtering and summed-area tables. *ACM Trans. Graph.*, 30(6):176:1–176:12, Dec. 2011. doi: 10.1145/2070781.2024210

[30] D. Nguyen, H. T. Tran, and T. C. Thang. A client-based adaptation framework for 360-degree video streaming. *Journal of Visual Communication and Image Representation*, 59:231 – 243, 2019. doi: 10.1016/j.jvcir.2019.01.012

[31] J. Park and K. Nahrstedt. Navigation graph for tiled media streaming. In *Proceedings of the 27th ACM International Conference on Multimedia*, MM '19, p. 447–455. Association for Computing Machinery, New York, NY, USA, 2019. doi: 10.1145/3343031.3351021

[32] A. Patney, M. Salvi, J. Kim, A. Kaplanyan, C. Wyman, N. Benty, D. Luebke, and A. Lefohn. Towards foveated rendering for gaze-tracked virtual reality. *ACM Trans. Graph.*, 35(6):179:1–179:12, Nov. 2016. doi: 10.1145/2980179.2980246

[33] S. Petrangeli, G. Simon, and V. Swaminathan. Trajectory-based viewport prediction for 360-degree virtual reality videos. In *2018 IEEE International Conference on Artificial Intelligence and Virtual Reality (AIVR)*, pp. 157–160, Dec 2018. doi: 10.1109/AIVR.2018.00033

[34] F. Qian, B. Han, Q. Xiao, and V. Gopalakrishnan. Flare: Practical viewport-adaptive 360-degree video streaming for mobile devices. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*, MobiCom '18, pp. 99–114. ACM, New York, NY, USA, 2018. doi: 10.1145/3241539.3241565

[35] F. Qian, L. Ji, B. Han, and V. Gopalakrishnan. Optimizing 360 video delivery over cellular networks. In *Proceedings of the 5th Workshop on All Things Cellular: Operations, Applications and Challenges*, ATC '16, pp. 1–6. ACM, New York, NY, USA, 2016. doi: 10.1145/2980055.2980056

[36] J. Ryoo, K. Yun, D. Samaras, S. R. Das, and 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*, MMSys '16, pp. 6:1–6:11. ACM, New York, NY, USA, 2016. doi: 10.1145/2910017.2910592

[37] M. Y. Saraiji, K. Minamizawa, and S. Tachi. Foveated streaming: Optimizing video streaming for telexistence systems using eye-gaze based foveation. Technical report, Taichi Lab, Sep 2017.

[38] S. Shi, V. Gupta, M. Hwang, and R. Jana. Mobile VR on edge cloud: A latency-driven design. In *Proceedings of the 10th ACM Multimedia Systems Conference*, MMSys '19, p. 222–231. Association for Computing Machinery, New York, NY, USA, 2019. doi: 10.1145/3304109.3306217

[39] Y. Sun, A. Lu, and L. Yu. Weighted-to-spherically-uniform quality evaluation for omnidirectional video. *IEEE Signal Processing Letters*, 24(9):1408–1412, 2017.

[40] A. Taghavi Nasrabadi, A. Mahzari, J. D. Beshay, and R. Prakash. Adaptive 360-degree video streaming using layered video coding. In *2017 IEEE Virtual Reality (VR)*, pp. 347–348, March 2017. doi: 10.1109/VR.2017.7892319

[41] E. Turner, H. Jiang, D. Saint-Macary, and B. Bastani. Phase-aligned foveated rendering for virtual reality headsets. In *2018 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, pp. 1–2, March 2018. doi: 10.1109/VR.2018.8446142

[42] O. Veksler. Fast variable window for stereo correspondence using integral images. In *2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2003. Proceedings.*, vol. 1, pp. I–I, June 2003. doi: 10.1109/CVPR.2003.1211403

[43] P. Viola and M. Jones. Rapid Object Detection using a Boosted Cascade of Simple Features. *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 1(511-518):3, 2001. doi: 10.1109/CVPR.2001.990517

[44] R. S. Wallace, P.-W. Ong, B. B. Bederson, and E. L. Schwartz. Space variant image processing. *International Journal of Computer Vision*, 13(1):71–90, Sep 1994. doi: 10.1007/BF01420796

[45] S. Winkler, E. D. Gelasca, and T. Ebrahimi. Toward perceptual metrics for video watermark evaluation. In A. G. Tescher, ed., *Applications of Digital Image Processing XXVI*, vol. 5203, pp. 371 – 378. International Society for Optics and Photonics, SPIE, 2003. doi: 10.1117/12.512550

[46] Y. Xiao, T. Ho, and C. Leung. Summed area tables for cube maps. *IEEE Transactions on Visualization and Computer Graphics*, 24(10):2773–2786, Oct 2018. doi: 10.1109/TVCG.2017.2761869

[47] S.-C. Yen, C.-L. Fan, and C.-H. Hsu. Streaming 360° videos to head-mounted virtual reality using dash over quic transport protocol. In *Proceedings of the 24th ACM Workshop on Packet Video*, PV '19, pp. 7–12. ACM, New York, NY, USA, 2019. doi: 10.1145/3304114.3325616

[48] W. Zhang, B. Han, and P. Hui. Jaguar: Low latency mobile augmented reality with flexible tracking. In *Proceedings of the 26th ACM International Conference on Multimedia*, MM '18, p. 355–363. Association for Computing Machinery, New York, NY, USA, 2018. doi: 10.1145/3240508.3240561