ABSTRACT

Title of Dissertation: Towards Immersive Streaming for Videos and Light Fields

David Li
Doctor of Philosophy, 2024

Dissertation Directed by: Professor Amitabh Varshney
Department of Computer Science

As virtual and augmented reality devices evolve with new applications, the ability to create and transmit immersive content becomes ever more critical. In particular, mobile, standalone devices have power, computing, and bandwidth limitations which require careful thought on how to deliver content to users. In this dissertation, we examine techniques to enable adaptive streaming of two types of content: 360° panoramic videos and light fields.

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. We introduce a new log-rectilinear transformation incorporating 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. Our technique results in a 31% decrease in flickering and a 10% decrease in bit rate with H.264 streaming while maintaining similar or better quality.

Neural representations have shown great promise in compactly representing radiance and light fields. However, existing neural representations are not suited for streaming as decoding can only be done at a single level of detail and requires downloading the entire neural network model. To resolve these challenges, we present a progressive multi-scale light field network that encodes light fields with multiple levels of detail across various subsets of the network weights. With our approach, light field networks can render starting with less than 7% of the model weights and progressively depict greater levels of detail as more model weights are streamed.

Existing methods for levels of detail in neural representations focus on a few discrete levels of detail. While a few discrete LODs are enough to enable progressive streaming and reduce artifacts, transitioning between LODs becomes a challenge as an instant transition can result in a popping artifact, blending requires two render passes at adjacent LODs, and dithering can briefly appear as flickering. Additionally, models with a few LODs create large model deltas and can only coarsely adapt to bandwidth and compute resources. To address these limitations, we present continuous levels of detail for light field networks to address flickering artifacts during transitions across levels of detail and enable more granular adaptation to available resources. With our approach, we reduce flickering between successive model updates by approximately $40-80\%$ and go from 4 performance levels to 385 performance levels from which the model can be executed. By rendering levels of detail at each possible network width, we additionally reduce the model size deltas from over a hundred rows and columns per layer down to a single row and column per layer, for smoother streaming potential.

Towards Immersive Streaming for Videos and Light Fields

by

David Li

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2024

Advisory Committee:

Dr. Amitabh Varshney, Chair/Advisor
Dr. Joseph F. Jaja
Dr. Furong Huang
Dr. Christopher A. Metzler
Dr. Ruofei Du

Acknowledgments

First, I would like to thank my advisor, Professor Amitabh Varshney, for which this endeavor would not have been possible. Professor Varshney joined in engaging discussions and encouragement crucial throughout every step of the journey as well as supplied the necessary resources and funding to undertake this work.

Next, I would like to thank all of my collaborators and lab mates who have supported my work through valuable suggestions that have helped shape the outcome. A special thanks is due to Ruofei Du who has provided me the initial opportunity to work in Professor Varshney's lab, invaluable guidance for my first few publications, and the opportunity to intern with him at Google.

Finally, I am grateful to my parents for their care and support throughout my upbringing which has granted me the privilege of pursuing my interests. It is only with their support that I have been able to attend the University of Maryland and study computer science for all these years.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

VR	Virtual Reality
AR	Augmented Reality
HMD	Head Mounted Display
FOV	Field of View
SAT	Summed-Area Table
CPU	Central Processing Unit
GPU	Graphics Processing Unit
CDN	Content Delivery Network
PSNR	Peak Signal-to-Noise Ratio
WS-PSNR	Weighted-to-Spherical Peak Signal-to-Noise Ratio
SSIM	Structural Similarity Index
DOF	Degree of Freedom
NeRF	Neural Radiance Field
LFN	Light Field Network
MLP	Multi-layer Perceptron
MPI	Multi-plane image
LDI	Layered Depth Image
SDF	Signed Distance Field
LOD	Level of Detail
RGB	Red, Green, Blue
RGBA	Red, Green, Blue, Alpha

Chapter 1: Introduction

Over the past several years, numerous 3D applications on desktop, virtual reality (VR), and augmented reality (AR) devices have quickly been discovered and developed. AR and VR devices with increasing resolutions and better features are being unveiled as well. In line with this, the requirements for 3D content have continued increasing with expectations of applications now having high-resolution photo-realistic graphics and visualizations. However, existing methods for representing immersive content such as 360° videos which allow for three degrees of freedom motion, or 3D content with six degrees of freedom, are not always ideal for representing high-fidelity content. In particular, as mobile VR and AR devices are being introduced, content must be represented in a way that efficiently uses available bandwidth, power, and computing resources. In this dissertation, we present our research results on foveated 360° video streaming [3], progressive multi-scale light field networks [4], and continuous levels of detail for light field networks [5].

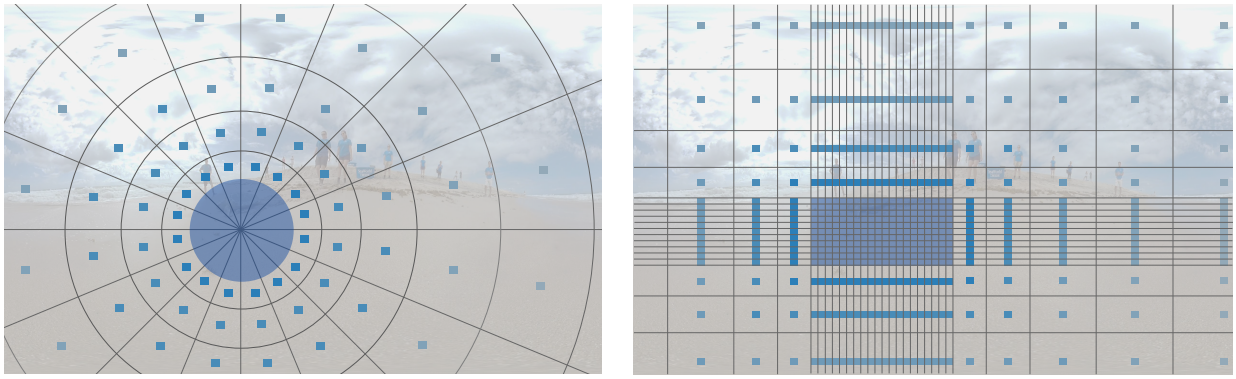
1.1 Foveated 360° Video Streaming

In the past, photorealistic content has primarily been limited to captured photographs and videos. With both of these forms of content, there is limited interaction and freedom for the viewer. The viewport is limited to what the original camera captures based on the decisions

of the photographer or filmmaker. Hence with standard photos and videos, the entire content must be transmitted to the viewer. 360° videos offer a panoramic view of the scene, giving the user three degrees of freedom to look around, either from a standard display or ideally a VR headset. However, in streaming scenarios, only a fraction of the viewport is within the user and headset's field of view. Furthermore, only a small portion of the field of view is in the foveal region where the user is directly focusing on. Hence, most of the content in a 360° video does not need to be streamed and a large portion can be streamed at a lower quality to conserve bandwidth, power, and compute. To address non-visible regions, many existing works in viewport-dependant streaming [6, 7, 8] examine how to stream only the regions of a 360° view that are in view. Typically, these employ a tiling strategy to stream regions in the viewport at a higher quality and regions near the viewport at a lower quality.

The goal of foveated streaming is to reduce the bandwidth consumption when streaming video over the internet by streaming the peripheral regions at a lower quality or sample rate. This is possible since the human visual system is less sensitive to details in the peripheral regions. Hence, a system that responds to the real-time gaze position by streaming the foveal regions only at a high quality could achieve a lower bandwidth than one that streams the entire viewport at full quality. For conventional videos with a limited field of regard, foveated streaming has been explored through tiling methods in Ryoo *et al.* [9]. These tiling methods are similar to those applied for viewport-dependant streaming for 360° panoramic videos. To achieve foveated streaming of 360° panoramic videos, a potential system would require multiple levels of detail not only in the entire 360° field of regard but also inside the viewport that the user is watching. This could potentially require dozens or hundreds of simultaneous streams, each of which would

require synchronized video decoding and would have worse compression due to being segmented into tiny regions.



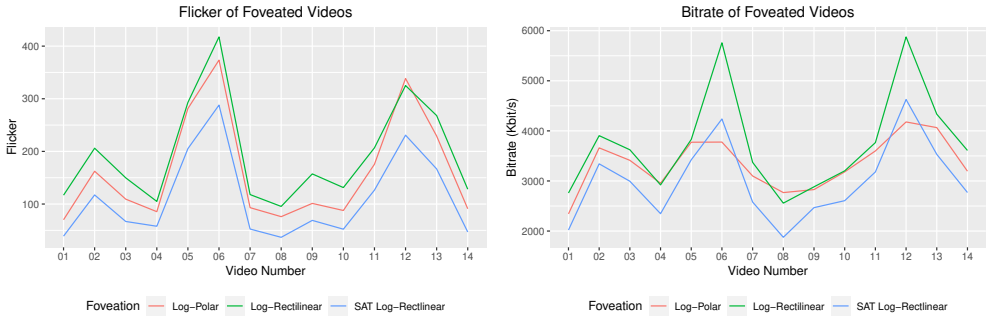
a) Log-Polar Transformation b) Log-Rectilinear Transformation

Figure 1.1: An illustration of the sampling positions of the log-polar transformation and our log-rectilinear transformation for foveation.

Instead of pursuing additional finer tiling, we explore an alternative method for foveated streaming which is based on techniques in foveated rendering. In foveated rendering, a log-polar transformation can be tuned [10] to allocate rendering computation more smoothly toward the foveal regions. However, a naive application of the log-polar transformation with single-pixel subsampling as shown in Figure 1.1 leads to foveation artifacts such as aliasing and flickering. Furthermore, the log-polar transformation inefficiently samples captured videos by oversampling the central regions. To address both issues, we propose the log-rectilinear transformation. This transformation allows us to use summed-area tables to quickly compute filtered samples of captured 360° video, stream a compact buffer over the network, and display a foveated 360° video frame.

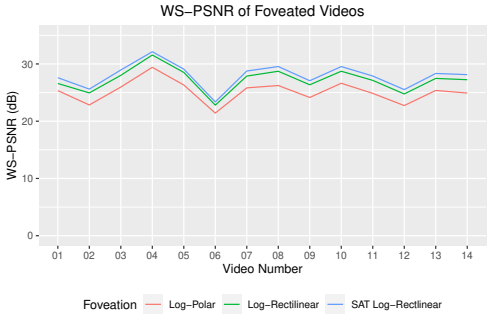
To evaluate our log-rectilinear transformation, we implement a foveated 360° streaming pipeline that decodes a 360° video in real-time on a server, compresses it into a lower resolution log-polar or log-rectilinear buffer, streams it over the internet, and displays it on a client. We

measure the resulting flicker, bandwidth usage, full-frame video quality, as well as processing overhead. In our experiments, we observe on average over 31% reduced flicker (Figure 1.2a), 10% reduced bandwidth usage (Figure 1.2b), and 3% better full-frame WS-PSNR metrics (Figure 1.2c) with only a 2 ms increase in the server compute latency compared to foveation using a log-polar transformation. Complete details of our method and experiments are described in Chapter 2.



(a) Flicker

(b) Bitrate



(c) WS-PSNR

Figure 1.2: Results of our experiments with our log-rectilinear transformation.

1.2 Progressive Multi-Scale Light Field Networks

For 3D content, traditionally meshes and textures are used to represent scenes and objects. However, creating photorealistic meshes and textures, even with captured images, is very challenging. Light fields can be used to represent photorealistic objects and scenes with view-

dependant appearance through hundreds of images. However, hundreds of images are hard to transmit, requiring lots of bandwidth to stream and computation to decode. With neural representations, images from a light field can be encoded into a relatively compact neural network, either as a neural radiance field [11] or a light field network [12].

While neural representations, and especially neural radiance fields (NeRFs), achieve high-quality photo-realistic view synthesis results, they suffer from many drawbacks compared to traditional representations which make them not yet suitable for real-time 3D graphics applications. One of the first major drawbacks is real-time rendering. NeRFs use volume rendering which requires dozens or hundreds of samples per pixel, each involving a full inference through a neural network with thousands to millions of parameters. Hence the original NeRF implementation takes roughly half a minute to render an image. Many methods have been proposed to address this including light field networks (LFNs) which use only one sample per pixel.

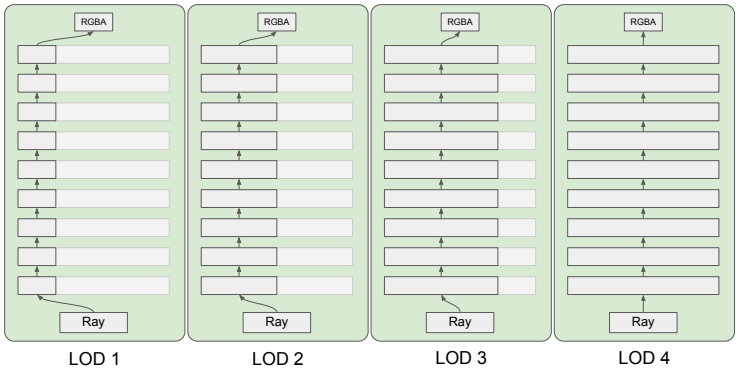


Figure 1.3: Using subsets of neurons to encode different levels of detail within a single model.

Two other challenges that make neural representations, and LFNs in particular, unsuitable for streaming are (1) the entire neural network must be downloaded before rendering can begin, and (2) rendering at a smaller scale can create aliasing and flickering artifacts. Flickering can be addressed by using light fields stored at different scales but doing so would require training,

storing, and streaming multiple light field networks. To address both of these challenges simultaneously, we have designed a novel technique to train a single light field network that can be progressively streamed and that can render a light field at different scales without significant aliasing artifacts.

With a progressive architecture, weights of the neural network can be streamed with rendering starting at a low level of detail and progressing to a higher level of detail once the full weights become available. At the lowest level of detail, we target $1/8$ scale renders using only 0.5 MB or less than 7% of the full model weights. To transition across levels of detail, dither can be used by gradually rendering pixels at the new level of detail. Furthermore, with a per-pixel level of detail, foveated rendering can be performed to further optimize the rendering performance.

Table 1.1: Average Rendering Quality Over All Datasets.

Model	LOD 1	LOD 2	LOD 3	LOD 4
Single-scale LFN	26.95	28.05	28.21	27.75
Multiple LFNs	29.13	29.88	**29.27**	27.75
Our Multi-scale LFN	**29.37**	**29.88**	29.01	**28.12**

(a) PSNR (dB) at 1/8, 1/4, 1/2, and 1/1 scale.

Model	LOD 1	LOD 2	LOD 3	LOD 4
Single-scale LFN	0.8584	0.8662	0.8527	0.8480
Multiple LFNs	0.8133	0.8572	0.8532	0.8480
Our Multi-scale LFN	**0.8834**	**0.8819**	**0.8626**	**0.8570**

(b) SSIM at 1/8, 1/4, 1/2, and 1/1 scale.

To determine the effectiveness of our method, we conduct experiments evaluating the rendering quality, training time, and rendering performance with our multi-scale light field networks. We evaluate our method using several light field scenes. For each scene, we have 240 views which are split into training, validation, and test views. From Table 3.3, we see that we observe better

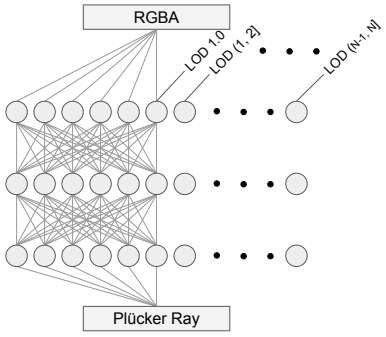
PSNR and SSIM metrics at lower resolutions due to reduced aliasing artifacts. In addition, we observe that rendering at the appropriate level of detail can improve rendering performance with $1/8$ scale rendering taking around 3.7 ms at the lowest LOD from 5.9 ms. Finally, we find that training a progressive multi-scale LFN takes only 18 hours on average compared to training four separate single-scale LFNs which takes 25 hours.

1.3 Continuous Levels of Detail for Light Field Networks

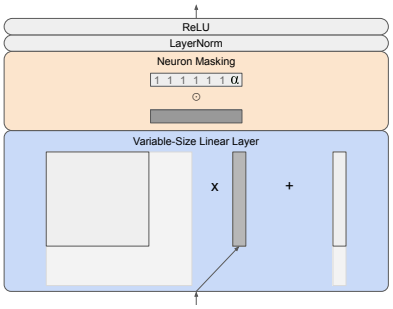
Many neural representations supporting levels of detail have emerged over the past few years. For instance, in our previous work, Progressive Multi-Scale Light Field Networks [4], we introduce light field networks that support four levels of detail which can be progressively streamed based on available network resources. Meanwhile, Progressive Implicit Networks (PINs) [13] support band-limited levels of detail for images and 3D SDFs. However, existing work generally focuses only on a few discrete levels of detail (e.g. 4-10) which have three main drawbacks for streaming scenarios. First, transitions across levels of detail can result in artifacts such as flickering or popping depending on the method. Second, streaming in additional LODs requires large model details which take longer to load and create spikes in network activity. Third, rendering between levels of detail requires rendering two separate LODs which would create a significant performance impact.

Building upon our prior work, we next introduce Continuous Levels of Detail for Light Field Networks. Continuous levels of detail address all three of these challenges simultaneously. Continuous levels of detail allow smooth transitions across LODs or model sizes without flickering or popping artifacts. Rendering in between LODs merely involves selecting an intermediate

LOD. In streaming scenarios, model deltas can be arranged such that only one additional row and column of weights are required for each layer.



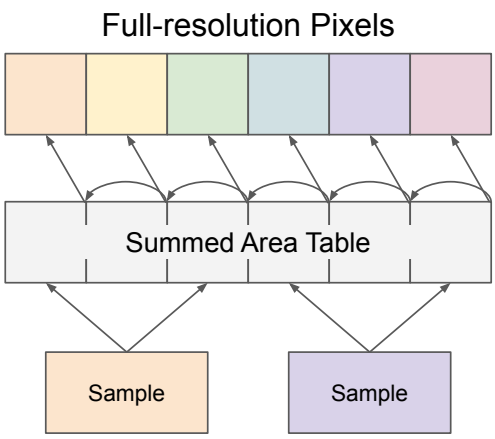
(a) Variable-size layers. By assigning a level of detail to every network width, we can achieve hundreds of performance levels.



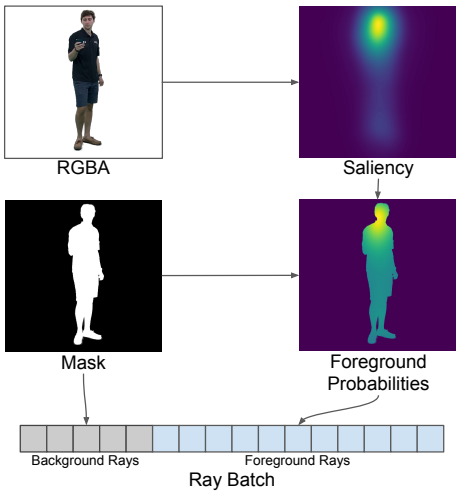
(b) Neuron masking. To render at continuous LOD ℓ, we use weights corresponding to LOD $\lceil \ell \rceil$ and apply alpha-blending to continuously fade-in features produced by the new neuron with $\alpha = \ell - \lfloor \ell \rfloor$ at each layer.

Figure 1.4: Illustrations of our method to represent continuous levels of detail.

To represent continuous levels of detail, we propose to use variable-sized layers, emitting a level of detail for every available network width as shown in Figure 1.4a. Next, we interpolate across network widths by blending in new neurons from the next network width as shown in Figure 1.4b using a continuous-valued variable α.



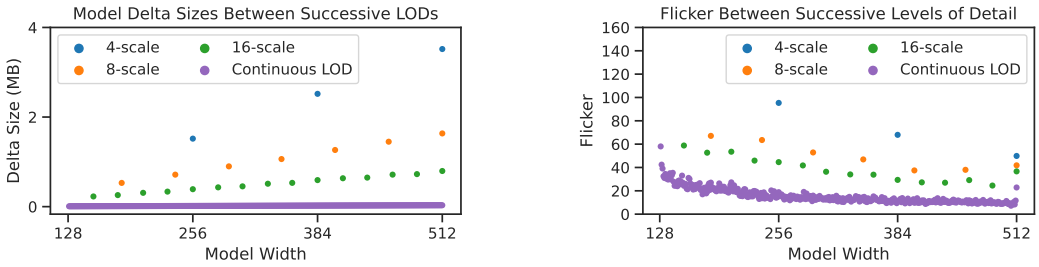
(a) Summed Area Table Sampling



(b) Saliency-based Importance Sampling

Figure 1.5: Illustrations of our method to train continuous levels of detail.

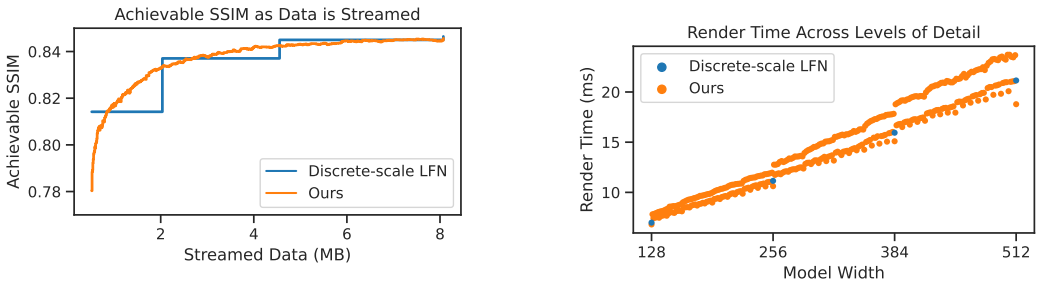
To train our network at continuous levels of detail, we use summed-area tables for arbitrary-size and arbitrary-position sampling of pixels from training images. In addition, we propose to apply saliency-based importance sampling during training to align details are lower LODs towards regions viewers are more likely to focus on.



(a) Model Delta Sizes

(b) Average Flicker

Figure 1.6: Plots showing the effects of transitioning across LODs. Transitioning with discrete LODs leads to larger network traffic spikes and more flickering.



(a) Average SSIM quality at the full resolution.

(b) Average render times at $1/4$ scale.

Figure 1.7: Plots showing our quantitative evaluation results. With continuous LODs, the LOD can be dynamically adjusted to maximize the quality based on available resources.

We evaluate our method by training continuous LOD models and discrete LOD models, comparing their quality and sizes during transitions and at fixed LODs. By assigning a usable level of detail at each model width, we reduce the maximum model delta size from 3.5 MB to 32 KB as shown in Figure 1.6a. During transitions across LODs at adjacent model sizes, we reduce the flickering from 100 down to 20. as shown in Figure 1.6b. At the same model sizes, there is

some overhead in quality compared to the discrete LOD model, requiring additional weights to achieve the quality. However, as seen in Figure 1.7a, as data is streamed in, the continuous model results in better quality for most available model sizes. Finally, we see in Figure 1.7b that render time scale mores smoothly across LODs with additional model sizes.

Chapter 2: A Log-Rectilinear Transformation for Foveated 360° Video Streaming

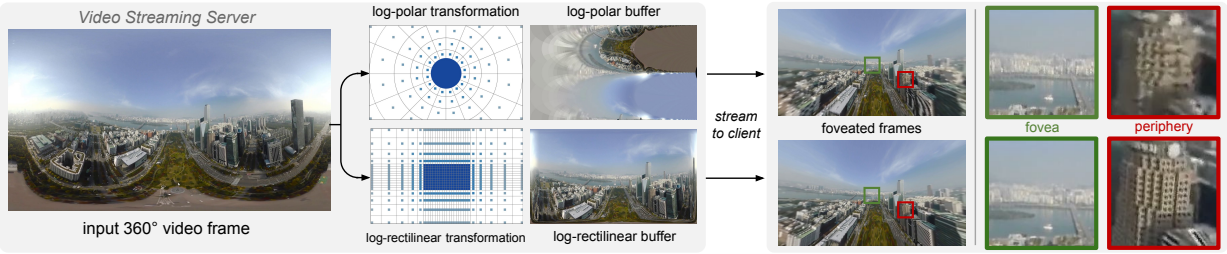


Figure 2.1: An overview of our foveated 360° video streaming pipeline. The upper and lower rows present the workflows with the prior log-polar transformation and our proposed log-rectilinear transformation, respectively. Both foveated methods convert the equirectangular video frames into down-sampled 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.

2.1 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 [7, 14]. 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 [6, 14]. Previous research in viewport-adaptive 360° video streaming [6, 8, 15, 16] 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 [9, 17, 18, 19, 20, 21] 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 [22].

In this chapter, 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

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

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

Our main contributions in this chapter 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 [1] with an ablation study demonstrating the effects of summed-area table filtering.

The remainder of this chapter 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. Supplementary material is available at `https://augmentariumlab.github.io/foveated-360-video/`.

2.2 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.

2.2.1 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 [18, 19, 20, 21] or quantization parameter adjustments [23, 24]. For 360° videos and VR applications, tiling techniques [7, 8, 16, 25, 26, 27, 28] 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* [6], 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

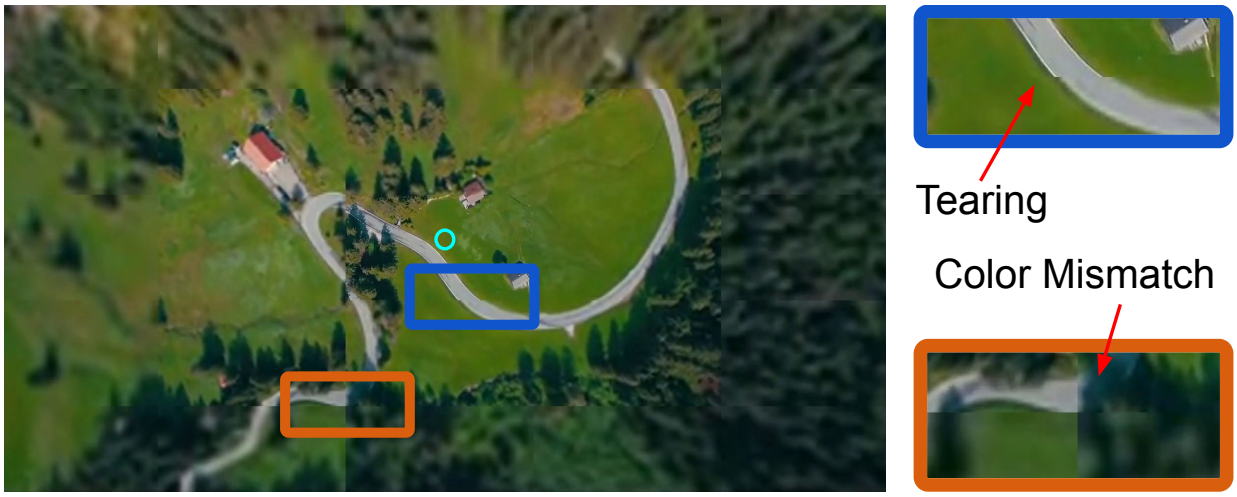
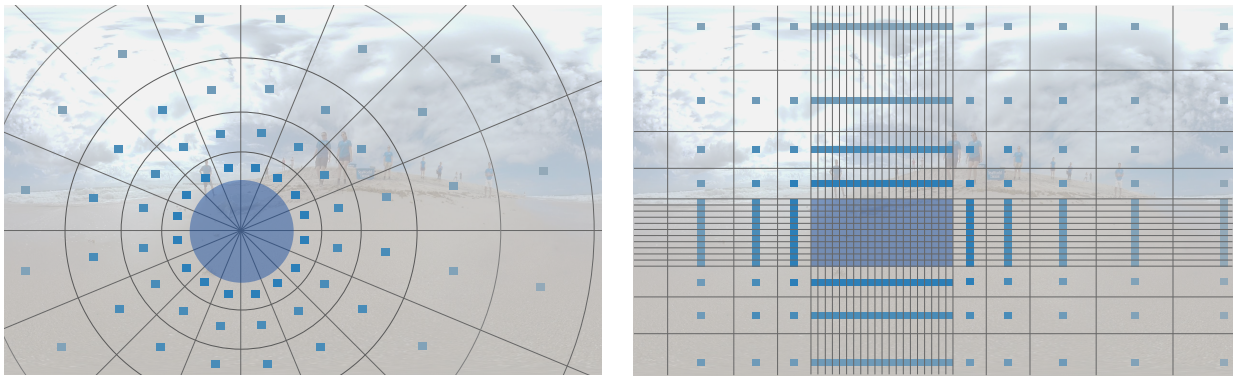


Figure 2.2: Foveated streaming of 360° videos requires too many tiles due to the large field of view and high variability of detail. Using too few tiles leads to tearing artifacts (blue box) and color mismatch artifacts (orange box) as shown above. The gaze position is marked with a cyan circle at the center of the frame.

only between visible and out-of-sight areas but also between different areas within the viewport. For instance, Ryoo *et al.* [9] 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 Figure 2.2. 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.



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

Figure 2.3: 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.

2.2.2 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.* [29] developed the first foveated rendering pipeline for 3D graphics that renders at three different resolutions based on perceptual detectability thresholds [30] 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 [31], and rendering to a log-polar buffer [10] 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.* [32] 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 [33]. Nevertheless, such methods are limited by training data and may produce flicker and ghosting artifacts for unexpected content.

2.2.3 Summed-Area Tables

Summed-area tables, also known as integral images, are a 2D extension of prefix-sum arrays. First proposed by Frank Crow [34] 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 [35, 36, 37, 38, 39]. The simplest parallel algorithm for generating a summed-area

table [39] 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 [40], 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 [35, 36] leverage memory locality, kernel synchronization, and look-back 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.* [35], 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 [41], depth of field [39, 42], glossy environmental reflections [39], and feature correspondence [43, 44, 45]. 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.

2.3 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.

2.3.1 Log-Rectilinear Transformation

The conventional geometric transformation used to emulate the resolution falloff with eccentricity for the human eye is the log-polar transformation [10, 47]. The log-polar transforma-

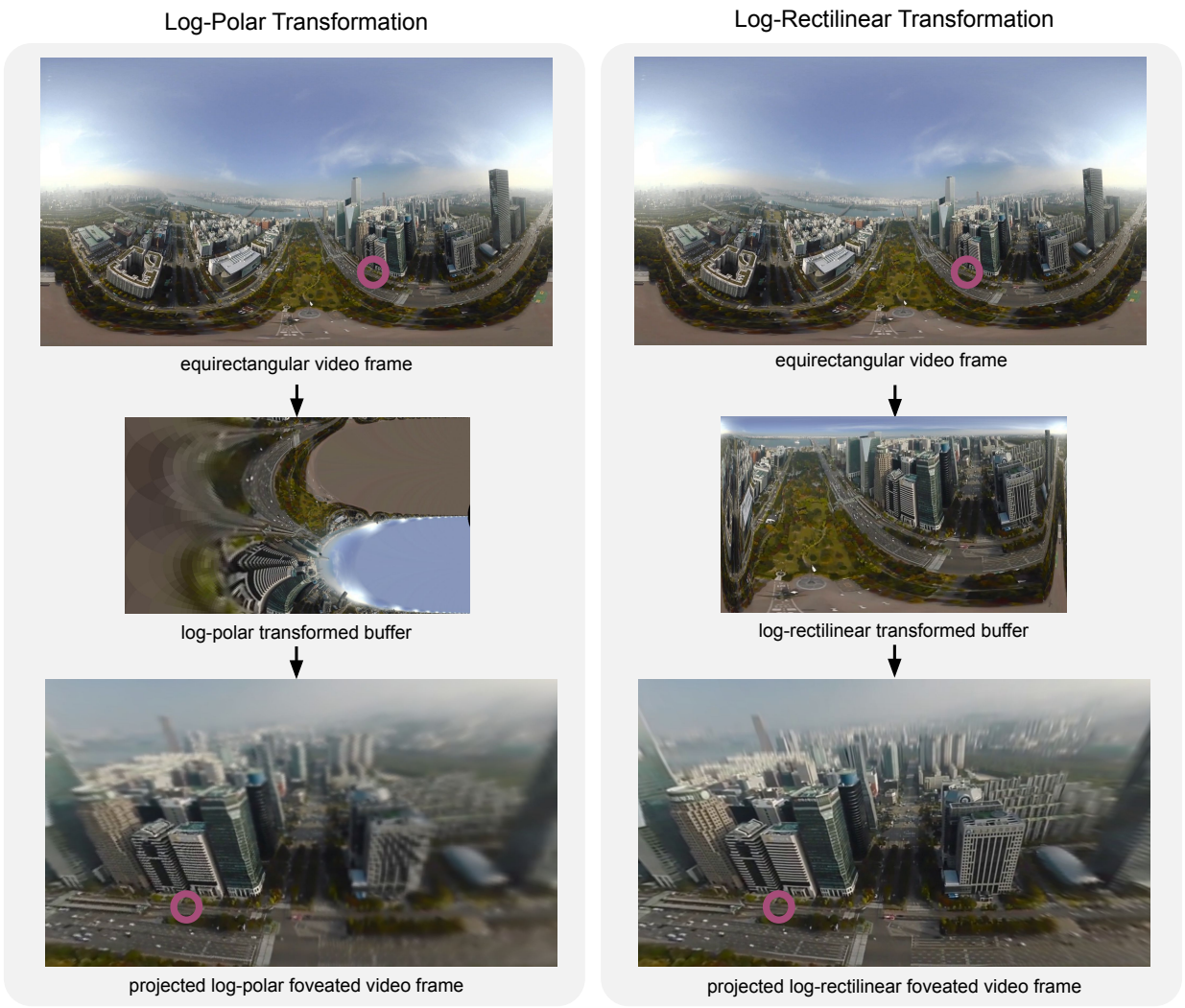


Figure 2.4: A comparison between the conventional log-polar transformation and our novel log-rectilinear transformation. The left and right columns show the same video frame being foveated using the log-polar and log-rectilinear transformations, respectively. The lowermost image in each column shows 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.

tion 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.

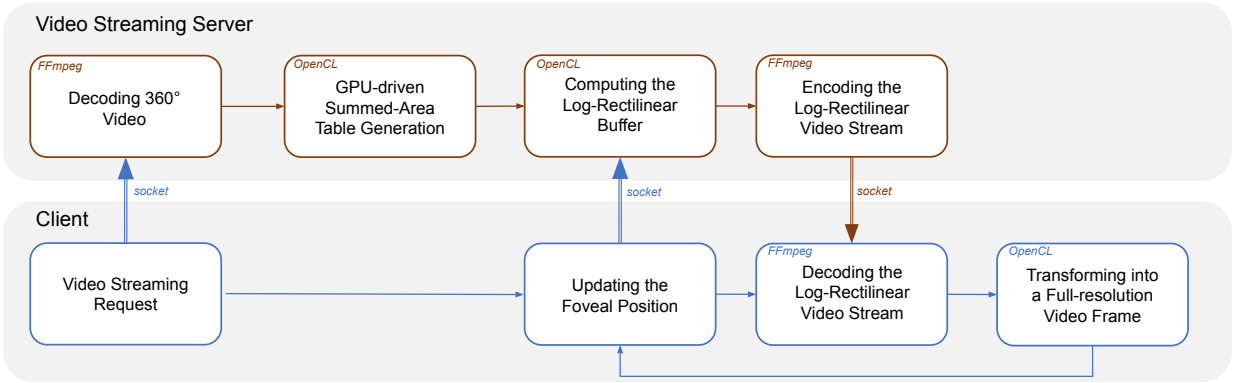


Figure 2.5: The system workflow for our integrated foveated video streaming prototype. Our server consists of four stages which include decoding the video, building a summed-area table, sampling the summed-area table into a log-rectilinear buffer, and streaming the log-rectilinear video buffers. The 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.

To address the drawbacks of the log-polar transformation, we propose a log-rectilinear transformation. Our log-rectilinear transformation, shown in Figure 2.3 and Figure 2.4, 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.* [10]. 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.* [10]) 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 section 2.4.

2.3.2 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 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.

2.4 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 Figure 2.5, consists of four processing stages for the server and two processing stages for the client.

2.4.1 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.

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

2.4.1.2 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.

2.4.1.3 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 [10]. 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.

2.4.1.4 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.

2.4.2 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.

2.4.2.1 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.

2.4.2.2 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.

2.5 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.

2.5.1 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.* [1]. 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

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

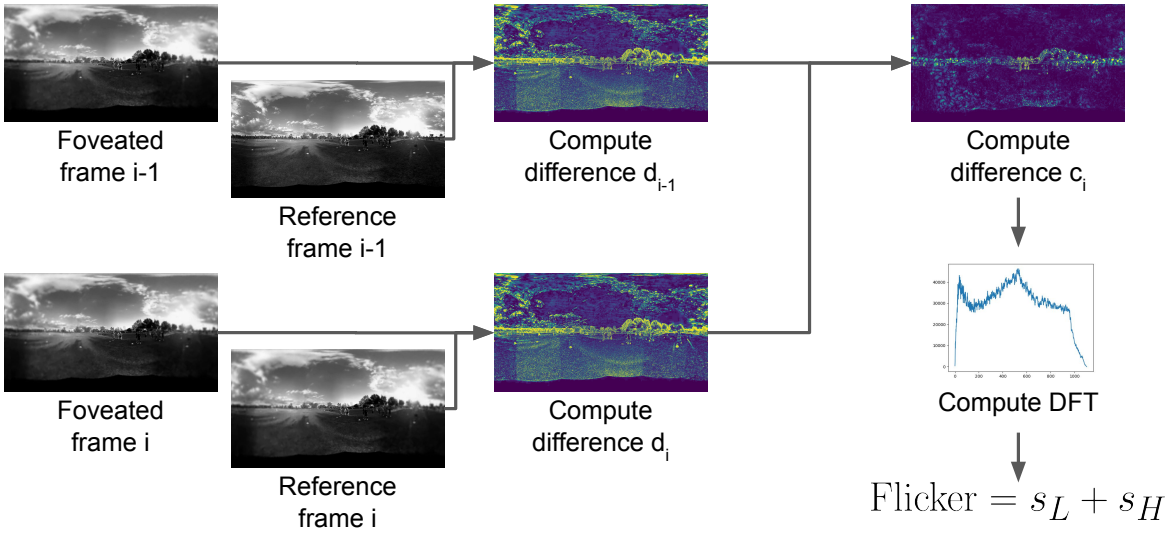


Figure 2.6: Overview of the Flicker metric by Winkler *et al.* [2] which captures the change in visual artifacts using a linear combination of Fourier coefficients.

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.* [10] during the sampling stage. No blur is applied for our log-rectilinear method.

2.5.2 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)[48] of the luminance channel (Y) because the eye is more sensitive to changes in luminance as opposed to changes in chrominance [49]. We also compute the Structural Similarity Index (SSIM) between the original videos and the foveated videos.

We adapt the metric by Winkler *et al.* [2] 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:

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

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.* [2], 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 Figure 2.6.

We show the averaged results of our quality comparison across all (participant, video) combinations with intermediate encoding of transformed buffers in Table 2.1 and without intermediate encoding in Table 2.2. Per-video averaged results with intermediate encoding are shown in Figure 2.8, Figure 2.9, Figure 2.10, Figure 2.11, and available in the Table 2.6. We also show how the WS-PSNR scales with available bandwidth in Figure 2.7 using the gaze of the first available participant (P3).

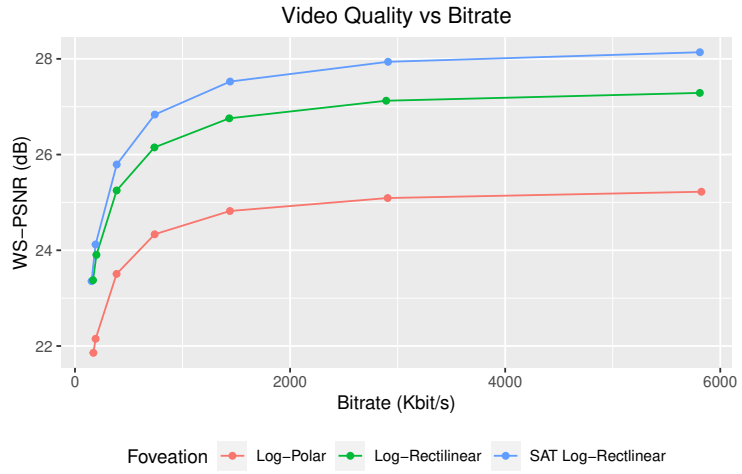


Figure 2.7: Comparison of visual quality (WS-PSNR) across bitrates ranging from 100 Kbits/s to 6 Mbits/s. Foveation with SAT Log-Rectilinear yields better quality across the entire range of bitrates.

Table 2.1: 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 2.2: 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 2.3: 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

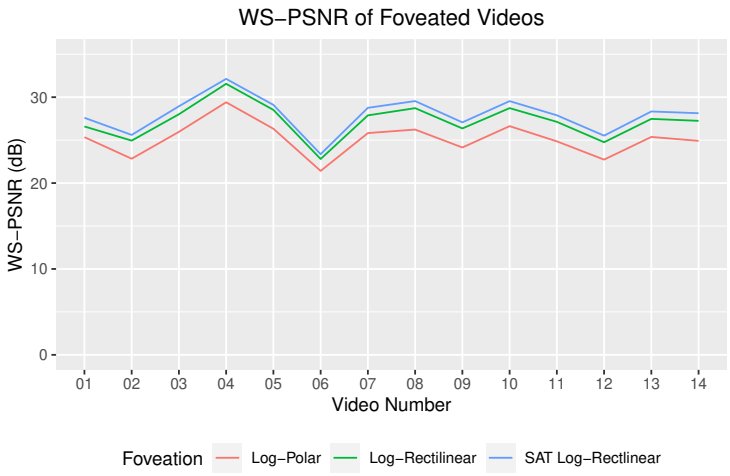


Figure 2.8: Quantitative evaluation of the average weighted spherical peak-signal-to-noise-ratio (WS-PSNR) in streaming each foveated video. Our SAT Log-Rectilinear method yields the highest WS-PSNR for all videos.

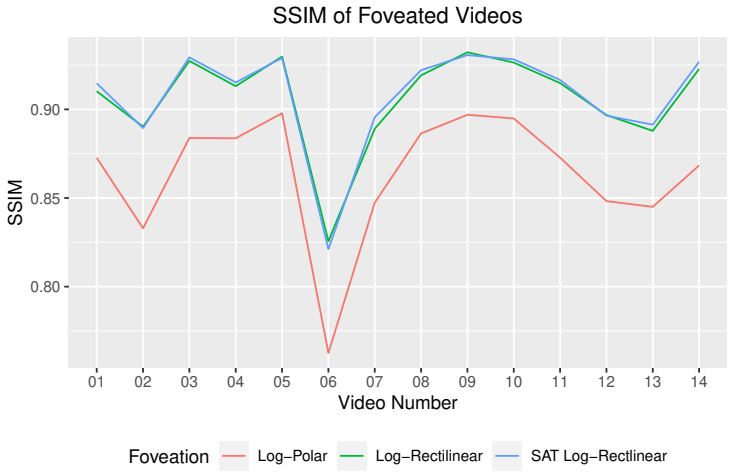


Figure 2.9: Quantitative evaluation of the average Structural similarity index (SSIM) in streaming each foveated video. Our SAT Log-Rectilinear method yields the highest SSIM for most videos.

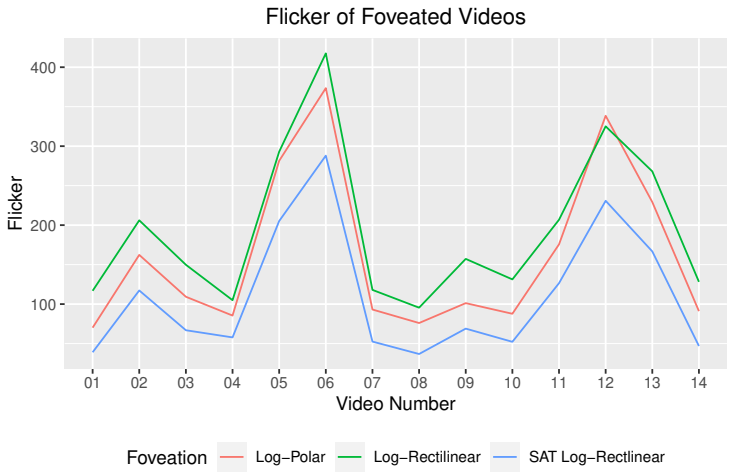


Figure 2.10: Quantitative evaluation of our flickering metrics for foveating each video in the dataset. Our SAT Log-Rectilinear method yields the lowest flickering for every video.

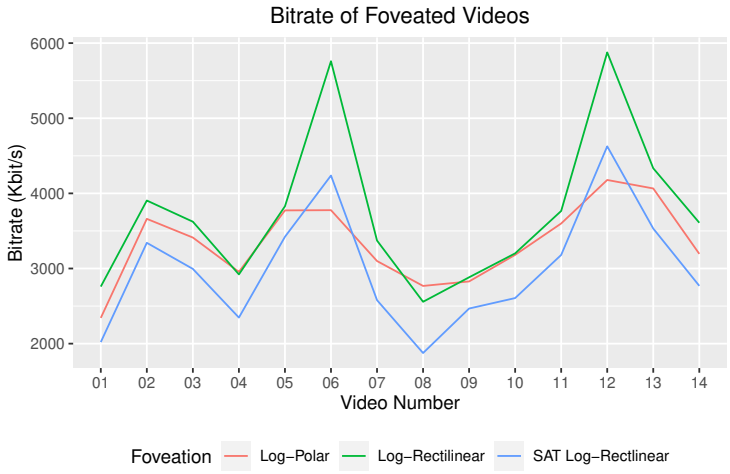


Figure 2.11: Measurements of the bitrate for streaming each video when encoding using a constant quality mode to H.264. On average, our SAT Log-Rectilinear method results in a 23% reduction in bandwidth consumption compared to Log-Polar foveation.

Table 2.4: 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	Processing	Sampling	Encoding	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

Table 2.5: 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

Table 2.6: 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.* [1]. 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

2.5.3 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 2.3.

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.

2.5.4 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 2.4, 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 2.5, 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 performance 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.

2.6 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 2.4. 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 [50, 51].

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 chapter.

[5]Google Stadia: https://stadia.google.com
[6]Project xCloud: https://xbox.com/xbox-game-streaming/project-xcloud

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 [6]. Standardizing evaluations of 360° video streaming systems is an ongoing challenge for multimedia systems researchers [52]. We leave the development of a fully-featured production-grade video pipeline and the QoE evaluation of the proposed pipeline for future work.

2.7 Conclusion

In this chapter, 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 [53, 54] would allow short-term buffering on the client. Combining our transcoding technique with previous work on pre-processed multi-resolution blocks [6] could lead to reduced server load for 360°

applications. Eye-dominance-guided foveation [55] may be adapted for streaming stereo 360° video.

With the ever-increasing resolutions of VR headsets and 360° cameras and the growing popularity of cloud-based gaming platforms as well as extended-reality applications [56, 57, 58], we believe our novel log-rectilinear transformation will make 360° VR content more immersive and accessible for everyone.

Chapter 3: Progressive Multi-scale Light Field Networks

3.1 Introduction

Volumetric images and video allow users to view 3D scenes from novel viewpoints with a full 6 degrees of freedom (6DOF). Compared to conventional 2D and 360 images and videos, volumetric content enables additional immersion with motion parallax and binocular stereo while also allowing users to freely explore the full scene.

Traditionally, scenes generated with 3D modeling have been commonly represented as meshes and materials. These classical representations are suitable for real-time rendering as well as streaming for 3D games and AR effects. However, real captured scenes are challenging to represent using the mesh and material representation, requiring complex reconstruction and texture fusion techniques [59, 60]. To produce better photo-realistic renders, existing view-synthesis techniques have attempted to adapt these representations with multi-plane images (MPI) [61], layered-depth images (LDI) [62], and multi-sphere images (MSI) [63]. These representations allow captured scenes to be accurately represented with 6DOF but only within a limited area.

Radiance fields and light fields realistically represent captured scenes from an arbitrarily large set of viewpoints. Recently, neural representations such as neural radiance fields (NeRFs) [11] and light field networks (LFNs) [12] have been found to compactly represent 3D scenes and perform view-synthesis allowing re-rendering from arbitrary viewpoints. Given a dozen or more

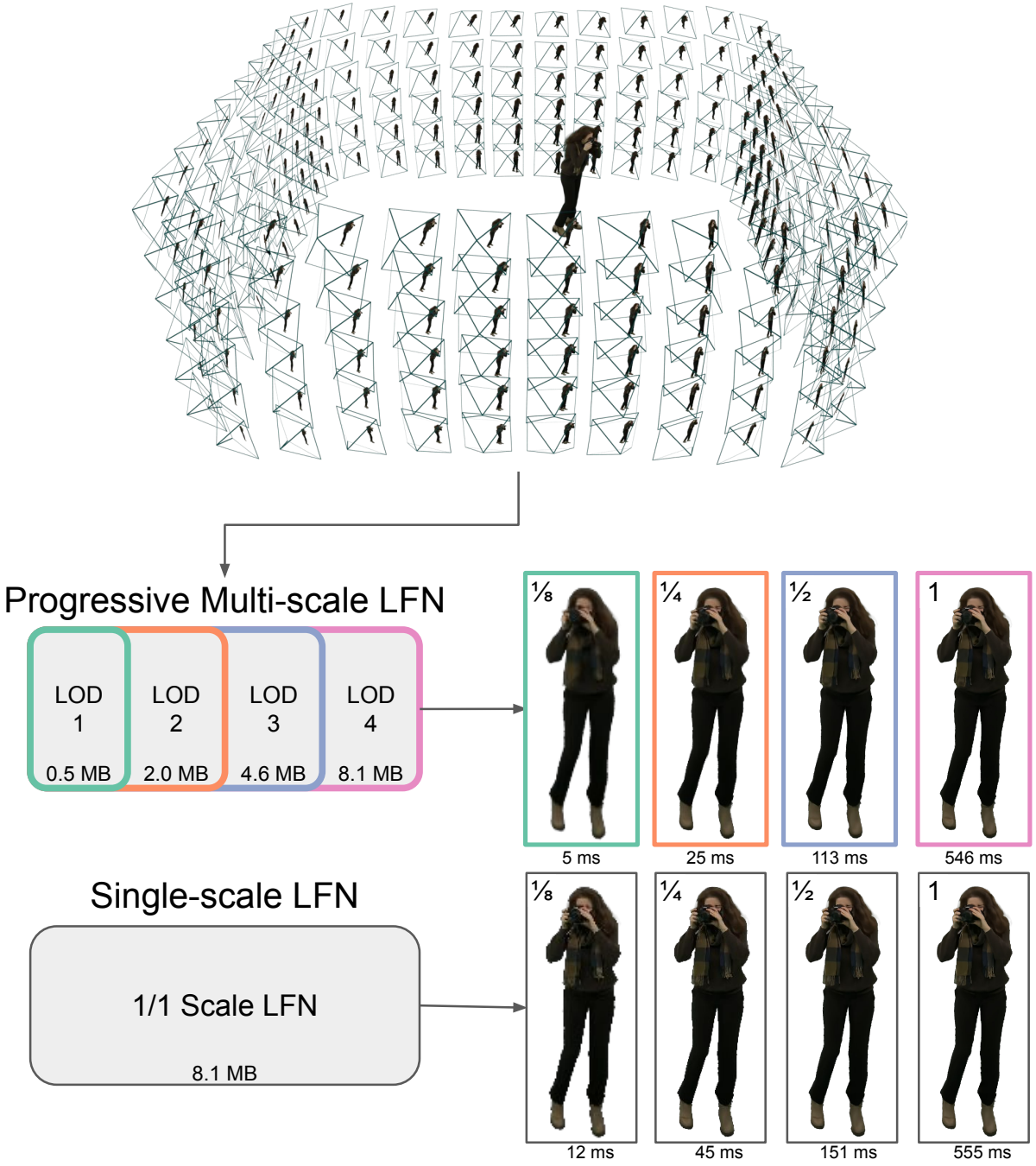


Figure 3.1: Our progressive multi-scale light field network is suited for streaming, reduces aliasing and flicking, and renders more efficiently at lower resolutions.

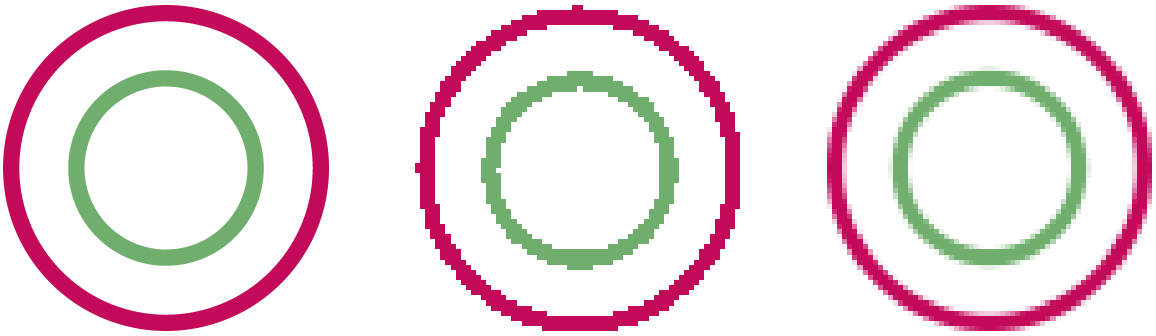
photographs capturing a scene from different viewpoints from a conventional camera, a NeRF or LFN can be optimized to accurately reproduce the original images and stored in less than 10 MB. Once trained, NeRFs will also produce images from different viewpoints with photo-realistic quality.

While NeRFs are able to encode and reproduce real-life 3D scenes from arbitrary viewpoints with photorealistic accuracy, they suffer from several drawbacks which limit their use in a full on-demand video streaming pipeline. Some notable drawbacks include slow rendering, aliasing at smaller scales, and a lack of progressive decoding. While some of these drawbacks have been independently addressed in follow-up work [64, 65], approaches to resolving them cannot always be easily combined and may also introduce additional limitations.

In this chapter, we extend light field networks with multiple levels of detail and anti-aliasing at lower scales making them more suitable for on-demand streaming. This is achieved by encoding multiple reduced-resolution representations into a single network. With multiple levels of details, light field networks can be progressively streamed and rendered based on the desired scale or resource limitations. For free-viewpoint rendering and dynamic content, anti-aliasing is essential to reducing flickering when rendering at smaller scales. In summary, the contributions of our progressive multi-scale light field network are:

1. Composing multiple levels of detail within a single network takes up less space than storing all the levels independently.

2. The progressive nature of our model requires us to only download the parameters necessary up to the desired level of detail.



High-resolution Reference | Nearest Downsampling | Area Downsampling

Figure 3.2: A high-resolution image resized using nearest and area downsampling. Rendering a low-resolution image from a high-resolution light field is similar to performing nearest downsampling, i.e. subsampling a single value. Area downsampling, subsampling with a box filter, generates an anti-aliased image but cannot be done efficiently on an LFN.

3. Our model allows rendering among multiple levels of detail simultaneously. This makes for a smooth transition between multiple levels of detail as well as spatially adaptive rendering (including foveated rendering) without requiring independent models at multiple levels of detail on the GPU.

4. Our model allows for faster inference of lower levels of detail by using fewer parameters.

5. We are able to encode light fields at multiple scales to reduce aliasing when rendering at lower resolutions.

3.2 Background and Related Works

Our work builds upon existing work in neural 3D representations, adaptive neural network inference, and levels of detail. In this section, we provide some background on neural representations and focus on prior work most related to ours. For a more comprehensive overview of neural rendering, we refer interested readers to a recent survey [66].

Table 3.1: Neural 3D Representations Comparison

Model	Real-time	Model Size	Multi-scale	Progressive
NeRF	✗	$\leq$ 10 MB	✗	✗
mip-NeRF	✗	$\leq$ 10 MB	✓	✗
Plenoctree	✓	$\geq$ 30 MB	✗	✗
LFN	✓	$\leq$ 10 MB	✗	✗
Ours	✓	$\leq$ 10 MB	✓	✓

3.2.1 Neural 3D Representations

Traditionally, 3D content has been encoded in a wide variety of formats such as meshes, point clouds, voxels, multi-plane images, and even side-by-side video. To represent the full range of visual effects from arbitrary viewpoints, researchers have considered using radiance fields and light fields which can be encoded using neural networks. With neural networks, 3D data such as signed distance fields, radiance fields, and light fields can be efficiently encoded as coordinate functions without explicit limitations on the resolution.

3.2.1.1 Neural Radiance Field (NeRF)

Early neural representations focused on signed distance functions and radiance fields. Neural Radiance Fields (NeRF) [11] use differentiable volume rendering to encode multiple images of a scene into a radiance field encoded as a neural network. With a sufficiently dense set of images, NeRF is able to synthesize new views by using the neural network to interpolate radiance values across different positions and viewing directions. Since the introduction of NeRF, followup works have added support for deformable scenes [67, 68, 69], real-time inference [70, 71, 72, 73], improved quality [74], generalization across scenes [75, 76], generative modelling [77, 78, 79],

videos [80, 81, 82], sparse views [83, 84, 85], fast training [73, 86], and even large-scale scenes [87, 88, 89].

Among NeRF research, Mip-NeRF [64, 90] and BACON [91] share a similar goal to our work in offering a multi-scale representation to reduce aliasing. Mip-NeRF approximates the radiance across conical regions along the ray, cone-tracing, using integrated positional encoding (IPE). With cone-tracing, Mip-NeRF reduces aliasing while also slightly reducing training time. BACON [91] provide a multi-scale scene representation through band-limited networks using Multiplicative Filter Networks [92] and multiple exit points. Each scale in BACON has band-limited outputs in Fourier space.

3.2.1.2 Light Field Network (LFN)

Our work builds upon Light Field Networks (LFNs) [12, 93, 94, 95]. Light field networks encode light fields [96] using a coordinate-wise representation, where for each ray $\mathbf{r}$, a neural network f is used to directly encode the color $c = f(\mathbf{r})$. To represent rays, Feng and Varshney [93] pair the traditional two-plane parameterization with Gegenbauer polynomials while Sitzmann *et al.* [12] use Plücker coordinates which combine the ray direction $\mathbf{r}_d$ and ray origin $\mathbf{r}_o$ into a 6-dimensional vector $(\mathbf{r}_d, \mathbf{r}_o \times \mathbf{r}_d)$. Light fields can also be combined with explicit representations [97]. In our work, we adopt the Plücker coordinate parameterization which can represent rays in all directions.

Compared to NeRF, LFNs are faster to render as they only require a single forward pass per ray, enabling real-time rendering. However, LFNs are worse at view synthesis as they lack the self-supervision that volume rendering provides with explicit 3D coordinates. As a result, LFNs

are best trained from very dense camera arrays or from a NeRF teacher model such as in [98]. Similar to NeRFs, LFNs are encoded as multi-layer perceptrons, hence they remain compact compared to voxel and octree NeRF representations [65, 70, 73, 99] which use upwards of 50 MB. Therefore LFNs strike a balance between fast rendering times and storage compactness which could make them suitable for streaming 3D scenes over the internet.

3.2.2 Levels of Detail

Level of detail methods are commonly used in computer graphics to improve performance and quality. For 2D textures, one of the most common techniques is mipmapping [100] which both reduces aliasing and improves performance with cache coherency by encoding textures at multiple resolutions. For neural representations, techniques for multiple levels of detail have also been proposed for signed distance functions as well as neural radiance fields.

For signed distance fields, Takikawa *et al.*[101] propose storing learned features within nodes of an octree to represent a signed distance function with multiple levels of detail. Building upon octree features, Chen *et al.*[102] develop Multiresolution Deep Implicit Functions (MDIF) to represent 3D shapes which combines a global SDF representation with local residuals. For radiance fields, Yang *et al.* [103] develop Recursive-NeRF which grows a neural representation with multi-stage training which is able to reduce NeRF rendering time. BACON [91] offers levels of detail for various scene representations including 2D images and radiance fields with different scales encoding different bandwidths in Fourier space. PINs [13] uses a progressive fourier encoding to improve reconstruction for scenes with a wide frequency spectrum. For more explicit representations, Variable Bitrate Neural Fields [104] use a vector-quantized auto-decoder to com-

press feature grids into lookup indices and a learned codebook. Levels of detail are obtained by learning compressed feature grids at multiple resolutions in a sparse octree. In parallel, Streamable Neural Fields [105] propose using progressive neural networks [106] to represent spectrally, spatially, or temporally growing neural representations.

3.2.3 Adaptive Inference

Current methods use adaptive neural network inference based on available computational resources or the difficulty of each input example. Bolukbasi *et al.* [107] propose two schemes for adaptive inference. Early exiting allows intermediate layers to route outputs directly to an exit head while network selection dynamically routes features to different networks. Both methods allow easy examples to be classified by a subset of neural network layers. Yeo *et al.*[108] apply early exiting to upscale streamed video. By routing intermediate CNN features to exit layers, a single super-resolution network can be partially downloaded and applied on a client device based on available compute. Similar ideas have also been used to adapt NLP to input complexity [109, 110]. Yang *et al.* [111] develop Resolution Adaptive Network (RANet) which extracts features and performs classification of an image progressively from low-resolution to high-resolution. Doing so sequentially allows efficient early exiting as many images can be classified with only coarse features. Slimmable neural networks [112] train a model at different widths with a switchable batch normalization and observe better performance than individual models at detection and segmentation tasks.

3.3 Method

Our method consists of a multi-scale light field encoded using a progressive neural network to reduce aliasing and enable adaptive streaming and rendering.

3.3.1 Multi-scale Light Field

We propose encoding multiple representations of the light field which produce anti-aliased representations of a light field, similar to mipmaps for conventional images. For this, we generate an image pyramid for each image view with area downsampling and encode each resolution as a separate light field representation. In contrast to bilinear or nearest pixel downsampling, area downsampling creates an anti-aliased view as shown in Figure 3.2 where each pixel in the downsampled image is an average over a corresponding area in the full-resolution image.

Single-scale Light Field Network

Multi-scale Light Field Network

Figure 3.3: Rendering a single full-scale LFN at a lower (1/8) resolution results in aliasing, unlike the multi-scale LFN at the same (1/8) resolution. When changing viewpoints (3 shown above), the aliasing results in flickering.

As with mipmaps, lower-resolution representations trade-off sharpness for less aliasing. In this case, where a light field is rendered into an image, sharpness may be preferable over some aliasing. However, in the case of videos where the light field is viewed from different poses or the light field is dynamic, sampling from a high-resolution light field leads to flickering. With a multi-scale representation, rendering light fields at smaller resolutions can be done at a lower scale to reduce flickering.

3.3.2 Progressive Model

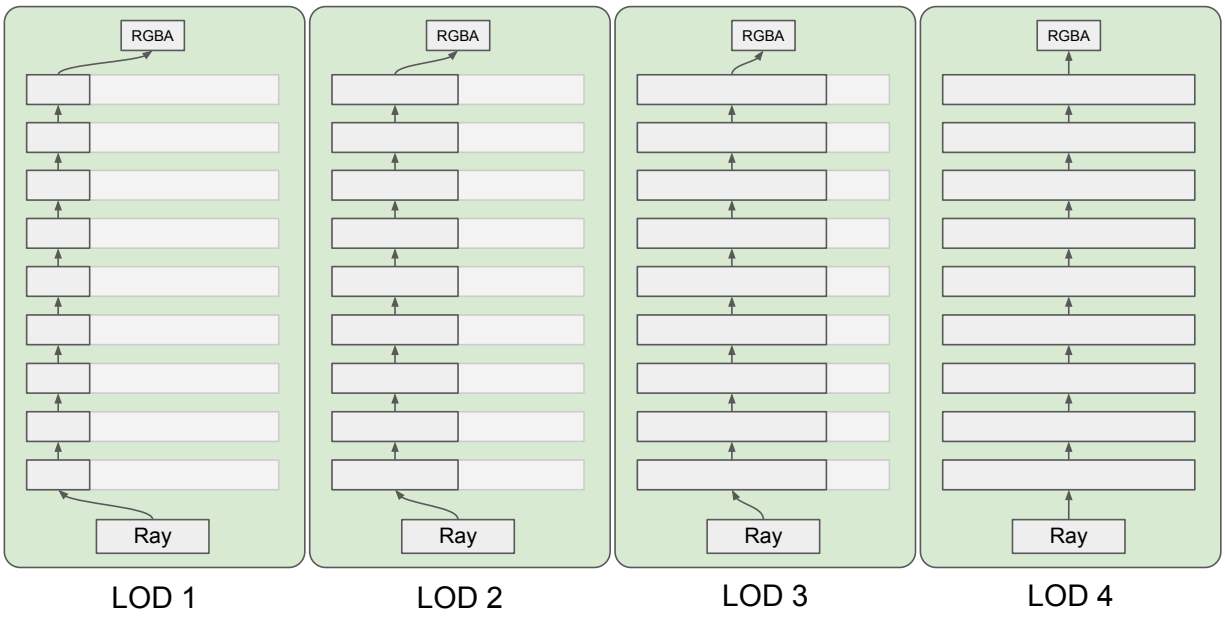


Figure 3.4: Using subsets of neurons to encode different levels of detail within a single model.

For light field streaming, we wish to have a compact representation that can simultaneously encode multiple levels of detail into a single progressive model. Specifically, we seek a model with the following properties: Given specific subsets of network weights, we should be able to render a complete image with reduced details. Progressive levels of details should share weights with lower levels to eliminate encoding redundancy.

With the above properties, our model offers several benefits over a standard full-scale network or encoding multi-scale light fields using multiple networks. First, our model composes all scales within a single network hence requiring less storage space than storing four separate models. Second, our model is progressive, thus only parameters necessary for the desired level of detail are needed in a streaming scenario. Third, our model allows rendering different levels of detail across pixels for dithered transitions and foveation. Fourth, our model allows for faster inference of lower levels of detail by utilizing fewer parameters. Model sizes and training times appear in Table 3.4.

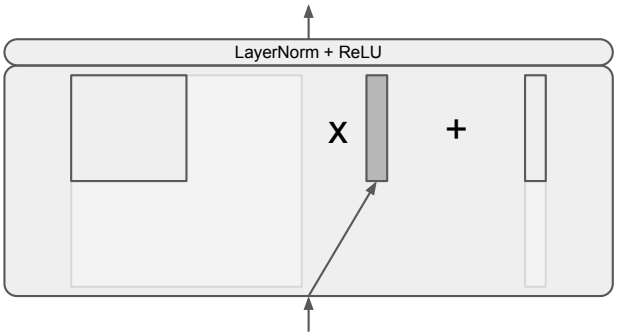


Figure 3.5: Illustration of a single layer where a subset of the weight matrix and bias vector are used, evaluating half of the neurons and reducing the operations down to approximately a quarter.

3.3.2.1 Subset of neurons

To encode multiple levels of detail within a unified representation, we propose using subsets of neurons as shown in Figure 3.4. For each level of detail, we assign a fixed number of neurons to evaluate, with increasing levels of detail using an increasing proportion of the neurons in the neural network. An illustration of the matrix subset operation applied to each layer is shown in Figure 3.5. For example, a single linear layer with 512 neurons will have a 512×512 weight matrix and a bias vector of size 512. For a low level of detail, we can assign a neuron subset size

of 25% or 128 neurons. Then each linear layer would use the top-left 128×128 submatrix for the weights and the top 128 subvector for the bias. To keep the input and output dimensions the same, the first hidden layer uses every column of the weight matrix and the final output layer uses every row of the weight matrix. Unlike using a subset of layers, as is done with early exiting [107, 108], using a subset of neurons preserves the number of sequential non-linear activations between all levels of details. Since layer widths are typically larger than model depths, using subsets of neurons also allows more granular control over the quantity and quality of levels.

3.3.2.2 Training

With multiple levels of detail at varying resolutions, a modified training scheme is required to efficiently encode all levels of detail together. On one hand, generating all levels of detail at each iteration heavily slows down training as each iteration requires an independent forward pass through the network. On the other hand, selecting a single level of detail at each iteration or applying progressive training compromises the highest level of detail whose full weights would only get updated a fraction of the time.

We adopt an intermediate approach that focuses on training the highest level of detail. During training, each batch does a full forward pass through the entire network, producing pixels at the highest level of detail along with a forward pass at a randomly selected lower level of detail. The squared L2 losses for both the full detail and reduced detail are added together for a combined backward pass. By training at the highest level of detail, we ensure that every weight gets updated at each batch and that the highest level of detail is trained to the same number of iterations as a standard model. Specifically, given a batch of rays $\{\mathbf{r}_i\}_{i=1}^{b}$, corresponding ground truth colors

$\{\{\mathbf{y}_i^c\}_{i=1}^b\}_{c=1}^4$ for four levels of details, and a random lower level of detail $k \in \{1, 2, 3\}$, our loss function is:

$$L = \frac{1}{b} \sum_{i=1}^{b} \left[\left\| f_4(\mathbf{r}_i) - \mathbf{y}_i^4 \right\|_2^2 + \left\| f_k(\mathbf{r}_i) - \mathbf{y}_i^k \right\|_2^2 \right]$$

As rays are sampled from the highest-resolution representation, each ray will not correspond to a unique pixel in the lower-resolution images in the image pyramid. Thus for lower levels of detail, corresponding colors are sampled using bilinear interpolation from the area downsampled training images.

3.3.3 Adaptive Rendering

Our proposed model allows dynamic levels of detail on a per-ray/per-pixel level or on a per-object level. As aliasing appears primarily at smaller scales, selecting the level of detail based on the distance to the viewer reduces aliasing and flickering for distant objects.

3.3.3.1 Distance-based Level of Detail

When rendering light fields from different distances, the space between rays in object space is proportional to the distance between the object and the camera. Due to this, aliasing and flickering artifacts can occur at large distances when the object appears small. By selecting the level of detail based on the distance of the object from the camera or viewer, aliasing and flickering can be reduced. With fewer pixels to render, rendering smaller objects involves fewer forward passes through the light field network. Hence, the amount of operations is typically proportional to the number of pixels. By rendering smaller representations at lower LODs, we

further improve the performance as pixels from lower LODs only perform a forward pass using a subset of weights.

Figure 3.6: With per-pixel level of detail, dithering can be applied to transition between levels of detail.

3.3.3.2 Dithered Transitions

With per-pixel LOD, transitioning between levels can be achieved with dithered rendering. By increasing the fraction of pixels rendered at the new level of detail in each frame, dithering creates the appearance of a fading transition between levels as shown in Figure 3.6.

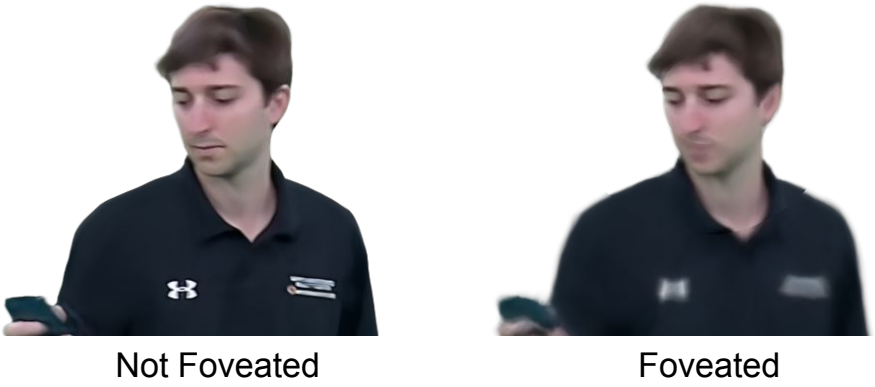
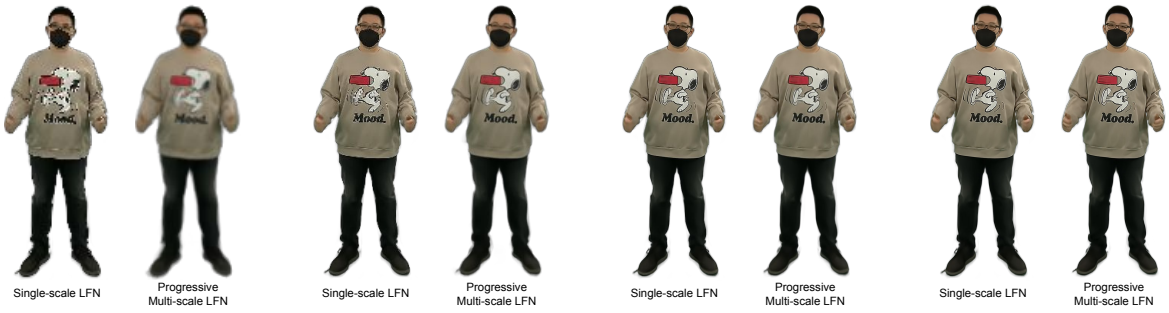


Figure 3.7: An example of foveated rendering from our progressive multi-scale LFN. Foveated rendering generates peripheral pixels at lower levels of detail to further improve performance in eye-tracked environments. In this example, foveation is centered over the left eye of the subject.

3.3.3.3 Foveated Rendering

In virtual and augmented reality applications where real-time gaze information is available through an eye-tracker, foveated rendering [3, 10, 113] can be applied to better focus rendering compute. As each pixel can be rendered at a separate level of detail, peripheral pixels can be rendered at a lower level of detail to improve the performance. Figure 3.7 shows an example of foveation from our progressive multi-scale LFN.



(a) Dataset C at 1/8 scale (b) Dataset C at 1/4 scale (c) Dataset C at 1/2 scale (d) Dataset C at full scale

Figure 3.8: Qualitative results of one of our datasets at four scales. Single-scale LFNs (left) trained at the full-resolution exhibit aliasing and flickering at lower scales. Our progressive multi-scale LFNs (right) encode all four scales into a single model and have reduced aliasing and flickering at lower scales.

(a) Dataset A (b) Dataset B (c) Dataset C (d) Dataset D (e) Dataset E

Figure 3.9: Scaled qualitative results from our progressive multi-scale LFNs at four levels of detail.

3.4 Experiments

To evaluate our progressive multi-scale LFN, we conduct experiments with four levels of detail. We first compare the quality when rendering at different scales from a standard single-scale LFN and our multi-scale LFN. We then perform an ablation comparing multiple LFNs trained at separate scales to our progressive network. Finally, we benchmark our multi-scale LFN to evaluate the speedup gained by utilizing fewer neurons when rendering at lower levels of detail. Additional details are in the supplementary material.

3.4.1 Experimental Setup

For our evaluation, we use datasets of real people consisting of 240 synchronized images from our capture studio. Images are captured at 4032×3040 resolution. Our datasets are processed with COLMAP [114, 115] to extract camera parameters and background matting [116] to focus on the foreground subject. In each dataset, images are split into groups of 216 training poses, 12 validation poses, and 12 test poses.

We quantitatively evaluate the encoding quality by computing the PSNR and SSIM metrics. Both metrics are computed on cropped images that tightly bound the subject to reduce the contribution of empty background pixels. We also measure the render time to determine the relative speedup at different levels of detail. Metrics are averaged across all images in the test split of each dataset.

When training LFNs, we use a batch size of 8192 rays with 67% of rays sampled from the foreground and 33% sampled from the background. For each epoch, we iterate through training images in batches of two images. In each image batch, we train on 50% of all foreground rays

sampling rays across the two viewpoints. For our multi-scale progressive model and single-scale model, we train using 100 epochs. For our ablation LFNs at lower resolutions, we train until the validation PSNR matches that of our progressive multi-scale LFN with a limit of 1000 epochs. Each LFN is trained using a single NVIDIA RTX 2080 TI GPU.

Table 3.2: Model Parameters at Different Levels of Detail.

Level of Detail	1	2	3	4
Model Layers	10	10	10	10
Layer Width	128	256	384	512
Parameters	136,812	533,764	1,193,860	2,116,100
Full Size (MB)	0.518	2.036	4.554	8.072
Target Scale	$1/8$	$1/4$	$1/2$	1
Train Width	504	1008	2016	4032
Train Height	380	760	1520	3040

3.4.2 Rendering Quality

We first evaluate the rendering quality by comparing a standard single-scale LFN trained at the highest resolution with our multi-scale progressive LFN. For the single-scale LFN, we train a model with 10 layers and 512 neurons per hidden layer on each of our datasets and render them at multiple scales: $1/8$, $1/4$, $1/2$, and full scale. For our multi-scale progressive LFN, we train an equivalently sized model with four LODs corresponding to each of the target scales. The lowest LOD targets the $1/8$ scale and utilizes 128 neurons from each hidden layer. Each increasing LOD uses an additional 128 neurons. Model parameters for each LOD are laid out in Table 3.2. Note that our qualitative results are cropped to the subject and hence have a smaller resolution. Qualitative results are shown in Figure 3.8 with renders from both models placed side-by-side. Quantitative results are shown in Table 3.3.

At the full scale, both single-scale and multi-scale models are trained to produce full-resolution images. As a result, the quality of the renders from both models is visually similar as shown in Figure 3.8. At $1/2$ scale, the resolution is still sufficiently high so little to no aliasing can be seen in renders from either model. At $1/4$ scale, we begin to see significant aliasing around figure outlines and shape borders in the single-scale model. This is especially evident in Figure 3.8b where the outlines in the cartoon graphic have an inconsistent thickness. Our multi-scale model has much less aliasing with the trade-off of having more blurriness. At the lowest level of detail, we see severe aliasing along the fine textures as well as along the borders of the object in the single-scale model. With a moving camera, the aliasing seen at the two lowest scales appears as flickering artifacts.

Table 3.3: Average Rendering Quality Over All Datasets.

Model	LOD 1	LOD 2	LOD 3	LOD 4
Single-scale LFN	26.95	28.05	28.21	27.75
Multiple LFNs	29.13	29.88	29.27	27.75
Multi-scale LFN	29.37	29.88	29.01	28.12

(a) PSNR (dB) at 1/8, 1/4, 1/2, and 1/1 scale.

Model	LOD 1	LOD 2	LOD 3	LOD 4
Single-scale LFN	0.8584	0.8662	0.8527	0.8480
Multiple LFNs	0.8133	0.8572	0.8532	0.8480
Multi-scale LFN	0.8834	0.8819	0.8626	0.8570

(b) SSIM at 1/8, 1/4, 1/2, and 1/1 scale.

3.4.3 Progressive Model Ablation

We next perform an ablation experiment to determine the benefits and drawbacks of our progressive representation. For a non-progressive comparison, we represent each scale of a multi-

scale light field using a separate LFN. Each LFN is trained using images downscaled to the appropriate resolution. This ablation helps determine the training overhead our progressive LFNs encounter by compressing four resolutions into a single model and how the rendering quality compares to having separate models.

Table 3.4: Multiple LFNs *vs* Progressive Multi-scale LFN.

Model	LOD 1	LOD 2	LOD 3	LOD 4	Total
Multiple LFNs	0.518	2.036	4.554	8.072	15.180
Multi-scale LFN	———	———	8.072	———	8.072

(a) Model Size (MB).

Model	LOD 1	LOD 2	LOD 3	LOD 4	Total
Multiple LFNs	3.51	6.36	4.09	11.35	25.31
Multi-scale LFN	———	———	17.78	———	17.78

(b) Average Training Time Over All Datasets (hours).

Progressive and non-progressive model sizes and training times are shown in Table 3.4. In terms of model size, using multiple LFNs adds up to 15 MB compared to 8 MB for our progressive model. This 47% savings could allow storing or streaming of more light fields where model size is a concern. For the training time, we observe that training a multi-scale network takes on average 17.78 hours. Additional offline training time for our multi-scale network compared to training a single standard LFN is expected since each training iteration involves two forward passes. Encoding a multi-scale light field using multiple independent LFNs requires training each network separately, totaling 25.31 hours. Despite the higher compression achieved by our progressive multi-scale LFN, we observe little to no training overhead. On average, training our multi-scale progressive LFN saves 30% of training time compared to naively training multiple light field networks at different resolutions.

Quantitative PSNR and SSIM quality metrics are shown in Table 3.3. We observe that utilizing multiple LFNs to encode each scale of each light field yields better PSNR results compared to rendering from a single-scale LFN. However, utilizing multiple LFNs increases the total model size. Our multi-scale LFN achieves the superior PSNR and SSIM results between these two setups without increasing the total model size. In an on-demand streaming scenario, using only a single-scale model would incur visual artifacts and unnecessary computation at lower scales. Streaming separate models resolves these issues but requires more bandwidth. Neither option is ideal for battery life in mobile devices. Our multi-scale model alleviates the aliasing and flickering artifacts without incurring the drawbacks of storing and transmitting multiple models.

Table 3.5: Training Ablation Average Rendering Quality Results.

Ablation	LOD 1	LOD 2	LOD 3	LOD 4
Progressive Training	26.30	30.33	29.08	28.10
108 Training Views	29.04	29.63	28.93	27.98
Ours	29.37	29.88	29.01	28.12

(a) PSNR (dB) at 1/8, 1/4, 1/2, and 1/1 scale.

Model	LOD 1	LOD 2	LOD 3	LOD 4
Progressive Training	0.7947	0.8719	0.8447	0.8390
108 Training Views	0.8765	0.8771	0.8604	0.8566
Ours	0.8834	0.8819	0.8626	0.8570

(b) SSIM at 1/8, 1/4, 1/2, and 1/1 scale.

3.4.4 Training Ablation

To evaluate our training strategy, we perform two ablation experiments. First, we compare our strategy to a coarse-to-fine progressive training strategy where lower levels of detail are trained and then frozen as higher levels are trained. Progressive training takes on average 24.20

hours to train. Second, we use half the number of training views to evaluate how the quality degrades with fewer training views. Both experiments use our progressive multi-scale model architecture. Our results are shown in Table 3.5 with additional details in the supplementary material.

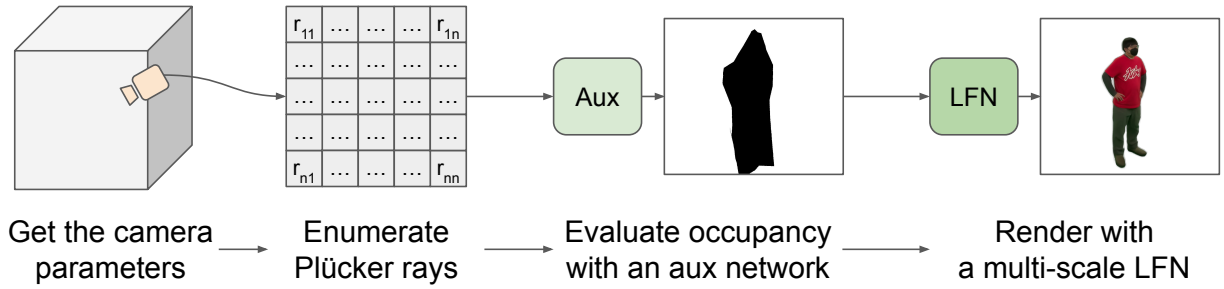


Figure 3.10: Overview of our rendering pipeline which utilizes an auxiliary network to skip evaluation of empty rays.

Table 3.6: Average Model Rendering Performance for our Multi-scale LFN (milliseconds per frame).

LOD 1	LOD 2	LOD 3	LOD 4
3.7	3.9	4.7	5.9

(a) Rendering each LOD at 1/8 scale.

LOD 1	LOD 2	LOD 3	LOD 4
4	11	58	305

(b) Rendering at 1/8, 1/4, 1/2, and 1/1 scale.

3.4.5 Level of Detail Rendering Speedup

We evaluate the rendering speed of our progressive model at different levels of detail to determine the observable speedup from rendering with the appropriate level of detail. For optimal rendering performance, we render using half-precision floating-point values and skip empty rays using an auxiliary neural network encoding only the ray occupancy as shown in Figure 3.10.

Our auxiliary network is made up of three layers with a width of 16 neurons per hidden layer. Evaluation is performed by rendering rays from all training poses with intrinsics scaled to the corresponding resolution as shown in Table 3.2. Average rendering time results are shown in Table 3.6. Rendering times for lower LODs (1/8 scale) from our multi-scale network are reduced from ~ 5.9 to ~ 3.7 msec.

3.5 Discussion

The primary contribution of this chapter is the identification of a representation more suitable for on-demand streaming. This is achieved by enhancing light field networks, which support real-time rendering, to progressively support multiple levels of detail at different resolutions. Our representation inherits some of the limitations of LFNs such as long training times and worse view-synthesis quality compared to NeRF-based models. Since our method does not support continuous LODs, we require the user to decide the LODs ahead of time. For intermediate resolutions, rendering to the nearest LOD is sufficient to avoid aliasing and flickering. While our light field datasets could also be rendered with classical techniques [96, 117, 118], doing so over the internet would not be as bandwidth-efficient. Future work may combine our multi-scale light field networks with PRIF [119] to encode color and geometry simultaneously or with VIINTER [120] for dynamic light fields.

3.6 Conclusion

In this chapter, we propose a method to encode multi-scale light field networks targeted for streaming. Multi-scale LFNs encode multiple levels of details at different scales to reduce alias-

ing and flickering at lower resolutions. Our multi-scale LFN features a progressive representation that encodes all levels of details into a single network using subsets of network weights. With our method, light field networks can be progressively downloaded and rendered based on the desired level of detail for real-time streaming of 6DOF content. We hope progressive multi-scale LFNs will help improve the real-time streaming of photorealistic characters and objects to real-time 3D desktop, AR, and VR applications [56, 58].

Chapter 4: Continuous Levels of Detail for Light Field Networks

4.1 Introduction

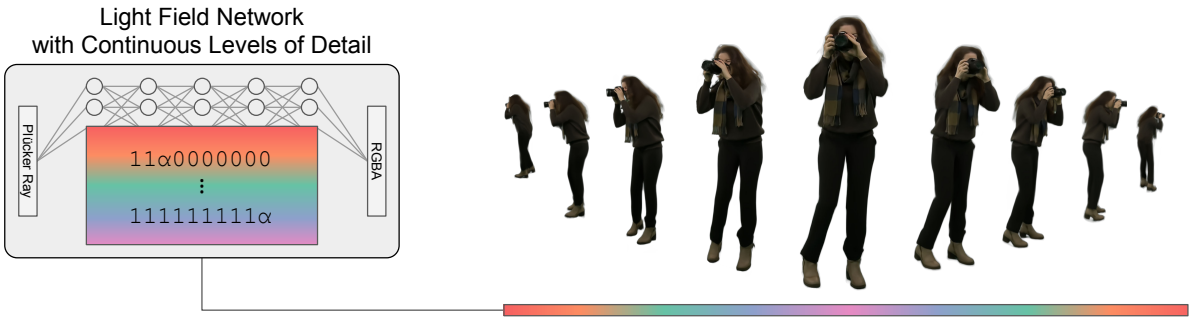


Figure 4.1: Our light field network features continuous levels of detail, enabled by training with summed-area table filtering and saliency-based importance sampling. Continuous levels of detail enable the interactive streaming of light fields with smooth transitions and finely tuned adaptivity.

In the past few years, implicit neural representations [11, 121] have become a popular technique in computer graphics and vision for representing high-dimensional data such as 3D shapes with signed distance fields and 3D scenes captured from multi-view cameras. Light Field Networks (LFN) [12] are able to represent 3D scenes with support for real-time rendering as each pixel of a rendered image only requires a single evaluation through the neural network.

In computer graphics, levels of detail (LODs) are commonly used to optimize the rendering process by reducing resource utilization for smaller distant objects in a scene. LODs prioritize resources to improve the overall rendering performance. In streaming scenarios, LODs can pri-

oritize and reduce network bandwidth usage. While LODs for implicit neural representations are beginning to be explored [4, 13, 91, 102, 105], most existing work focuses on offering a few discrete LODs which have three drawbacks for streaming scenarios. First, with only a few LODs, switching between them can result in flicker or popping effects as there are significant jumps from one LOD to the next. Second, discrete LODs require streaming in larger model deltas which take longer and create spikes in network activity. Third, transitioning across successive LODs can require rendering multiple levels and impact the rendering performance. Continuous LODs resolve these challenges by allowing smoother transitions with finer quality adaptation and size differences across LODs. Additionally, they allow rendering engines to dynamically adjust the rendering quality based on real-time bandwidth and compute resource availability without needing the viewer to select the LOD.

In this chapter, we develop a method to achieve continuous levels of detail for neural representations, focusing on light field networks. In summary, our light field networks with continuous LODs have the following benefits:

- Continuous LODs allow us to smoothly transition across LODs without popping artifacts,
- In streaming scenarios, the LOD can be increased or decreased by downloading only one additional row and column of weights per layer, allowing lower latency transitions with smoother network patterns, and
- Importance sampling allows LFNs to focus their capacity on the most salient regions of the light field, allowing details to resolve at lower LODs.

4.2 Background and Related Works

In this section, we provide some background on general neural fields and more specifically light field networks. We then overview recent methods for achieving multiple levels of detail with neural fields. For a general overview of neural rendering, we refer readers to Tewari *et al.* [66].

4.2.1 Implicit Neural Representations

Coordinate-based neural networks have been used to encode various signals such as images [122], signed distance functions [121], radiance fields [11, 64, 123], and light fields [12, 93]. These neural network-based models are often referred to as implicit neural representations or neural fields. Among these representations, neural radiance fields (NeRFs) and light field networks (LFNs) are both able to represent colored 3D scenes with view-dependant appearance effects.

Neural radiance fields (NeRFs) [11] employ differentiable volume rendering to encode a 3D scene into a multi-layer perceptron (MLP) neural network. By learning the density and color of the scene and using a positional encoding, NeRF can perform high-quality view synthesis, rendering the scene from arbitrary camera positions, while maintaining a very compact representation. However, the original NeRF implementation has many drawbacks, such as slow rendering times, which has limited its practicality. With an incredible amount of interest in neural rendering, many follow-up works have been proposed to improve NeRFs with better rendering performance [71, 73, 124], better quality [64], generalizability [125], and deformations [67, 68, 69]. Additionally, feature grid methods [73, 86] enable learning scenes in seconds and rendering in real-time. Importance sampling [126] can achieve faster learning with fewer training rays.

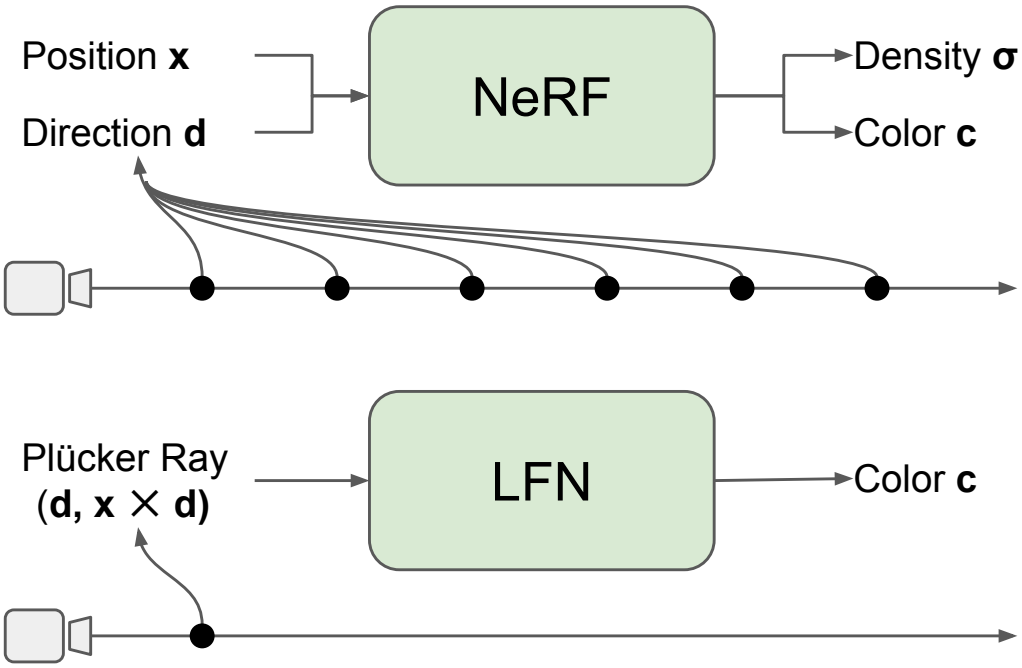


Figure 4.2: LFNs directly predict the RGB color for each ray in a single inference using Plücker coordinates, avoiding the dozens to hundreds of inferences required by NeRFs.

Light Field Networks Light Field Networks (LFNs) [12, 93, 94, 95, 127] encode light fields [96, 128] by directly learning the 4D variant of the plenoptic function for a scene. Specifically, LFNs directly predict the emitted color for a ray which eliminates the need for volume rendering, making light fields much faster to render compared to other neural fields. Earlier work in light field networks focus on forward-facing scenes using the common two-plane parameterization for light fields. SIGNET [93, 129] uses Gegenbauer polynomials to encode light field images and videos. NeuLF [94] proposes adding a depth branch to encode light fields from a sparser set of images. Plücker coordinates have been used [12, 119] to represent 360-degree light fields.

4.2.2 Levels of Detail

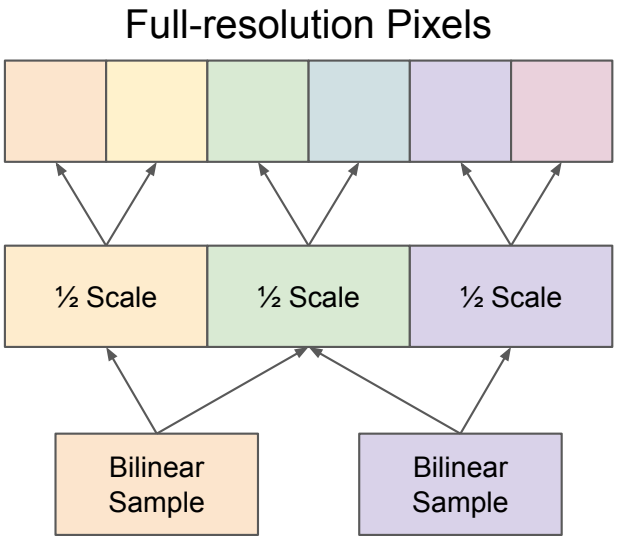
Several methods have been proposed for neural representations with multiple levels of detail. NGLOD [101] encode signed distance functions into a multi-resolution octree of feature vectors. VQAD [104] adds vector quantization with a feature codebook and presents results on NeRFs. BACON [91] encodes LODs with different Fourier spectrums for images and radiance fields. PINs [13] develop a progressive Fourier feature encoding to improve reconstruction and provide progressive LODs. MINER [130] trains neural networks to learn regions within each scale of a Laplacian pyramid representation. Streamable Neural Fields [105] propose growing neural networks to represent increasing spectral, spatial, or temporal sizes. Progressive Multi-Scale Light Field Networks [4] train a light field network to encode light fields at multiple resolutions.

To generate arbitrary intermediate LODs, existing methods blend outputs across discrete LODs. With only a few LODs, the performance does not scale smoothly since the next discrete LOD must be computed entirely. Our method offers continuous LODs with hundreds of performance levels allowing for finer adaptation to resource limitations.

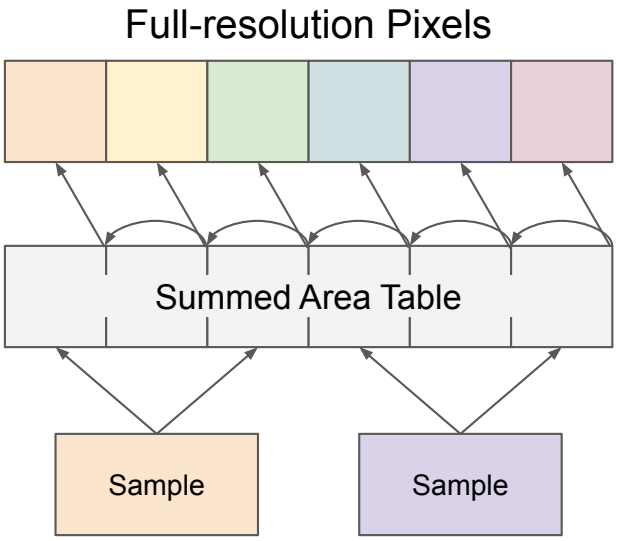
4.3 Method

Our method primarily builds upon *Light Field Networks* (LFNs) [12]. Specifically, we represent rays $\mathbf{r}$ in Plücker coordinates $(\mathbf{r}_d, \mathbf{r}_o \times \mathbf{r}_d)$ which are input to a multi-layer perceptron (MLP) neural network without any positional encoding. The MLP directly predicts RGBA color values without any volume rendering or other accumulation. Each light field network is trained to overfit a single static scene.

4.3.1 Arbitrary-scale Arbitrary-position Sampling with Summed Area Tables



(a) Discrete Scale Sampling



(b) Summed Area Table Sampling

Figure 4.3: An illustration of discrete and summed-area table sampling. (a) Sampling from a discrete resolution requires linear interpolation from a downsampled image to the target scale and position. (b) Summed area tables allow us to sample at both arbitrary scales and positions without significant additional memory or compute.

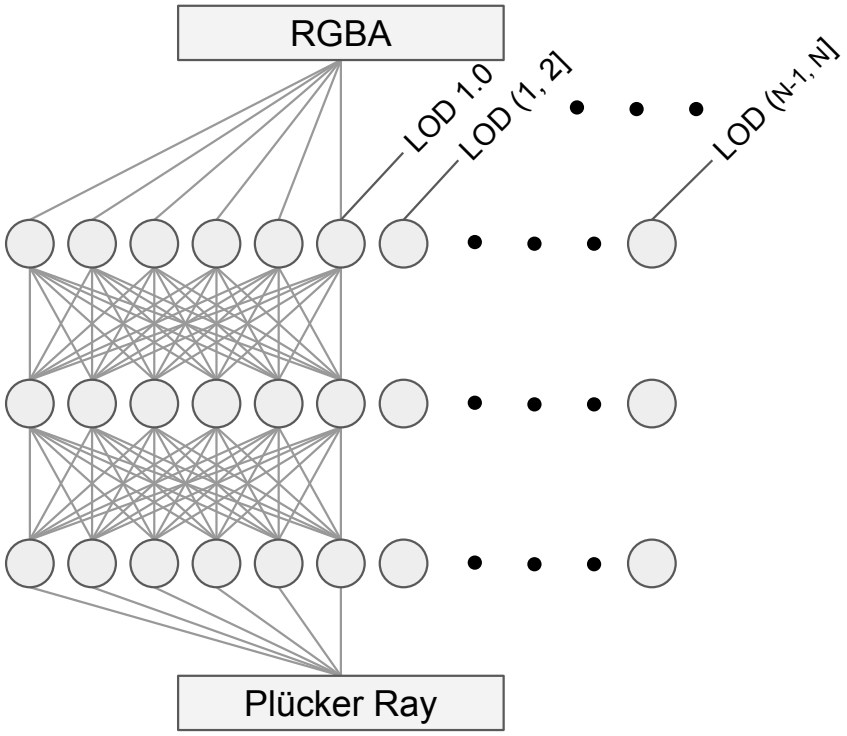
In order to reduce aliasing and flickering artifacts when rendering at smaller resolutions, e.g. when an object is far away from the user, lower levels of details need to be processed

with filtering to the appropriate resolution. In prior work, multi-scale LFNs [4] are trained on images resized to $1/2$, $1/4$, and $1/8$ scale using area downsampling. During training, rays are sampled from the full-resolution image while colors are sampled from lower-resolution images using bilinear sampling. While training on lower-resolution light fields yields multi-scale light field networks, the bilinear subsampling of the light field may not provide accurate filtered colors for intermediate positions. As shown in Figure 4.3, colors for higher-resolution rays get averaged over a larger area when performing bilinear subsampling in between low-resolution pixels.

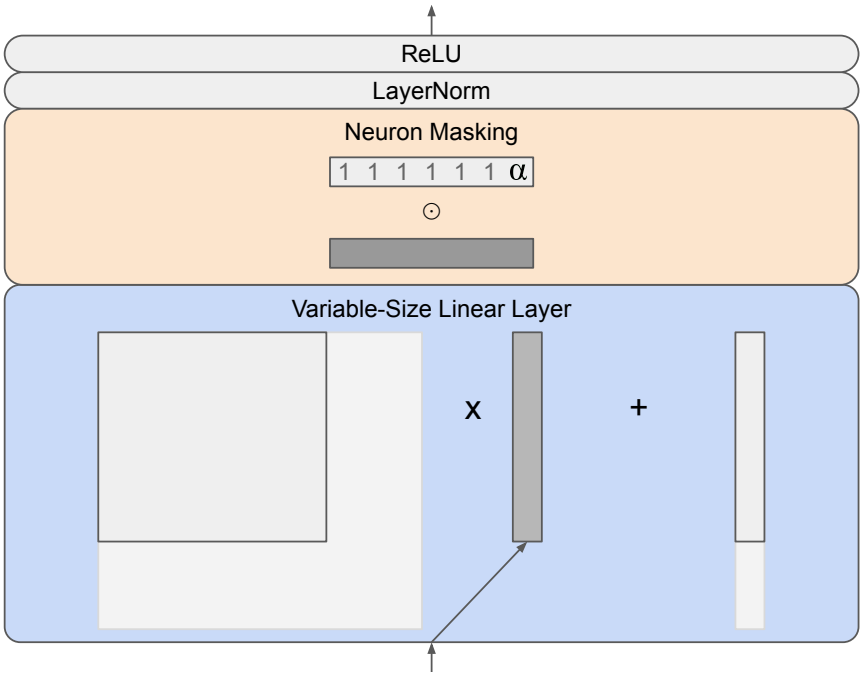
Another method for generating multi-scale light fields is to apply to filter at full resolution to get a spatially accurate anti-aliased sample for each pixel location. Naively precomputing and caching full-resolution copies of each light field image at each scale would significantly increase memory usage. Computing the average pixel color for each sampled ray at training time would require additional computation. Summed area tables [3, 34] can be used to efficiently sample pixels at arbitrary scales and positions, allowing us to sample from filtered versions of the training image without caching multiple copies. Sampling from a summed area table is a constant time operation, giving us an average over any axis-aligned rectangular region with only four samples. With additional samples, summed-area tables can also be used to apply higher-order polynomial (e.g. cubic) filters [39, 131] or Gaussian filters [132] for even better anti-aliasing, though we only use box filtering in our implementation.

4.3.2 Continuous Levels of Detail

While previous neural field methods offer static levels of detail corresponding to fixed scales [4, 102, 104] or fixed spectral frequency bands [91, 105], our goal is to generate a finer



(a) Variable-size layers. By assigning a level of detail to every network width, we can achieve hundreds of performance levels.



(b) Neuron masking. To render at continuous LOD ℓ, we use weights corresponding to LOD $\lceil \ell \rceil$ and apply alpha-blending to continuously fade-in features produced by the new neuron with $\alpha = \ell - \lfloor \ell \rfloor$ at each layer.

Figure 4.4: Illustrations of our method to achieve continuous levels of detail.

progression with continuous levels of detail. Continuous levels of detail enable smoother transitions and more precise adaptation to resolution and resource requirements.

Following existing work [4, 105, 133], we encode levels of detail using different widths of a single multi-layer perception neural network. Unlike Mip-NeRF [64, 90], this enables optimized performance with smaller neural networks at lower levels of detail. However, for continuous levels of detail, we propose two changes. First, we map the desired level of detail to every available width to extend a few levels of detail to hundreds of levels of detail as shown in Figure 4.4a. Second, we propose neuron masking which fades in new neurons to enable true continuous quality adjustments.

LOD to Scale Mapping Li *et al.* [4] train multi-scale LFNs which use width factors $1/4$, $2/4$, $3/4$, and $4/4$ (128, 256, 384, 512 widths) to encode $1/8$, $1/4$, $1/2$, and $1/1$ scale light fields respectively. To extend this to arbitrary widths, we formulate the following equations which describe the correspondence between network width w and light field scale s:

$$s = 2\hat{}(4w - 4) \tag{4.1}$$

$$w = (1/4) * (\log_2(s) + 4) \tag{4.2}$$

By using the above equations, we can assign a unique scale to each width sub-network in our multi-scale light field network. Since this is a one-to-one invertible mapping, we can also compute the ideal level of detail to use for rendering at any arbitrary resolution. In our experiments, we use a minimum width of 25% of nodes corresponding to a scale of $1/8$ to ensure a reasonable minimum quality and training image size. As an example, for a network with 512-width hidden

layers, the lowest level of detail uses only 128 neurons of each hidden layer while the highest uses 512.

Neuron Masking Since neural networks have discrete widths, it is necessary to map continuous levels of detail to discrete widths. Hence, we propose to use neuron masking to provide true continuous levels of detail with discrete-sized neural networks. As weights corresponding to each new width become available, we propose to apply alpha-blending on neurons corresponding to the width. This alpha-blending enables features from existing neurons to continuously transition, representing any intermediate level of detail between the discrete widths. Given feature $\mathbf{f}$ and fractional LOD $\alpha = l - \lfloor l \rfloor$, the new feature $\mathbf{f}'$ with neuron masking is the element-wise product:

$$\mathbf{f}' = (1, ..., 1, \alpha)^\top \odot \mathbf{f} \tag{4.3}$$

4.3.3 Saliency-based Importance Sampling

With continuous LODs representing light fields at various scales, the capacity of the LFN is constrained at lower LODs. Hence, details such as facial features may only resolve at higher levels of detail. To maximize the apparent fidelity, the capacity of the network should be distributed towards the most salient regions, *i.e.* the areas where viewers are most likely to focus. We propose to use saliency-based importance sampling which focuses training on salient regions of the light field. For all foreground pixels, we assign a base sampling weight λ_f and add a weight of $\lambda_s * s$ based on the pixel saliency s. Specifically, for a given foreground pixel x in a training

image with saliency s, we sample from the probability density:

$$p(x) = \lambda_f + \lambda_s * s \tag{4.4}$$

In our experiments, we use $(\lambda_f, \lambda_s) = (0.4, 0.6)$ which yields reasonable results. At each iteration, we sample 67% of rays in each batch from foreground pixels using the above density. The remaining 33% of rays are uniformly sampled from background pixels.

4.4 Experiments

We conduct several experiments to evaluate whether our light field networks with continuous LODs overcome the problems with discrete LODs. We also conduct quality and performance evaluations to determine the compute and bandwidth overhead associated with continuous LODs.

4.4.1 Experimental Setup

We conduct our experiments using five light field datasets. Scenes are captured using 240 cameras with 40×6 layout around the scene and a 4032×3040 resolution per camera. Each dataset includes camera parameters extracted using COLMAP [114, 115] and is processed with background matting. Of the 240 images, we use 216 for training, 12 for validation, and 12 for testing. We generate saliency maps using the mit1003 pretrained network[1] of Kroner *et al.* [134].

For our model, we use an MLP with nine hidden layers and one output layer. Each hidden layer uses LayerNorm and ReLU. We use a minimum width of 128 and a maximum width of 512 for variable-size layers. Our models are trained using a squared L2 loss for the

[1] From `https://github.com/alexanderkroner/saliency`

RGBA color with 8192 rays per batch. In all of our experiments, we train using the Adam optimizer with the learning rate set to 0.001 and exponentially decayed by $\gamma = 0.98$ after each epoch. We train for 100 epochs. Each of our models is trained using a single NVIDIA RTX 2080 Ti GPU. Our PyTorch implementation and processed datasets are available at `https://augmentariumlab.github.io/continuous-lfn/`.

4.4.2 Ablation Experiments

Our ablation experiments evaluate how each aspect of our method affects the final rendered quality. First, we replace the discrete resolution sampling in discrete-scale light field networks [4] with our summed area table sampling. Next, we add continuous LODs training which is enabled by arbitrary-scale filtering with summed-area tables. Finally, we compare the prior two setups with our full method which also includes saliency-based importance sampling.

4.4.3 Transitions across LODs

With continuous LODs, our method allows smooth transitions across LODs as additional bytes are streamed over the network or as the viewer approaches the subject. To quantitatively evaluate the smoothness of the transitions, we use the reference-based temporal flicker metric of Winkler *et al.* [2]. This flicker metric first computes the difference d between the processed images and reference images for two consecutive frames. Next, a difference image $c = d_n - d_{n-1}$ is computed across consecutive frames. The 2D discrete Fourier transform of the image c is computed and values are summed based on the radial frequency spectrum into low and high-

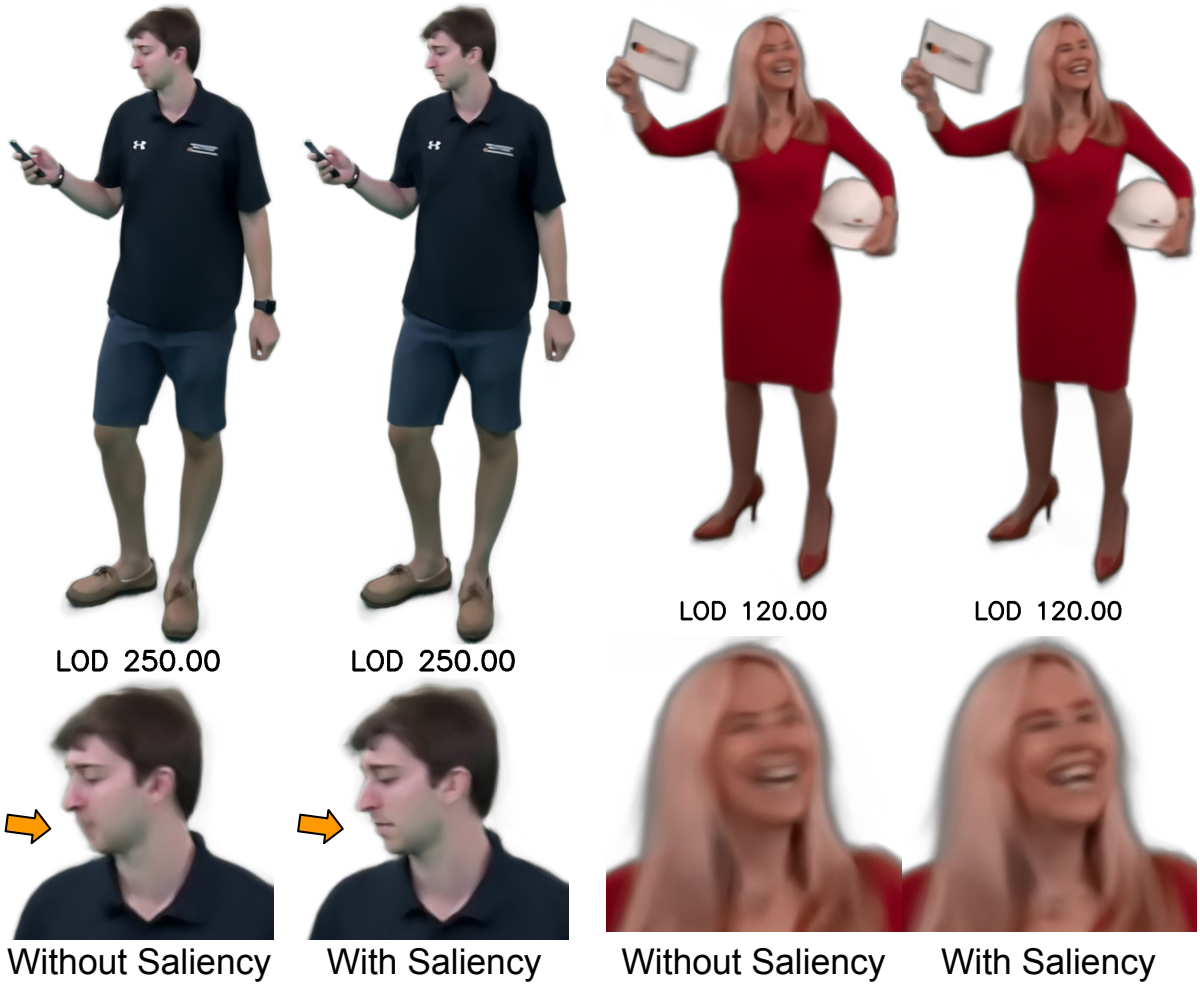


Figure 4.5: Lower LOD ablation results show the effects of our saliency-based importance sampling. With importance sampling, features such as eyes and mouths in the salient regions resolve at earlier, lower LODs.

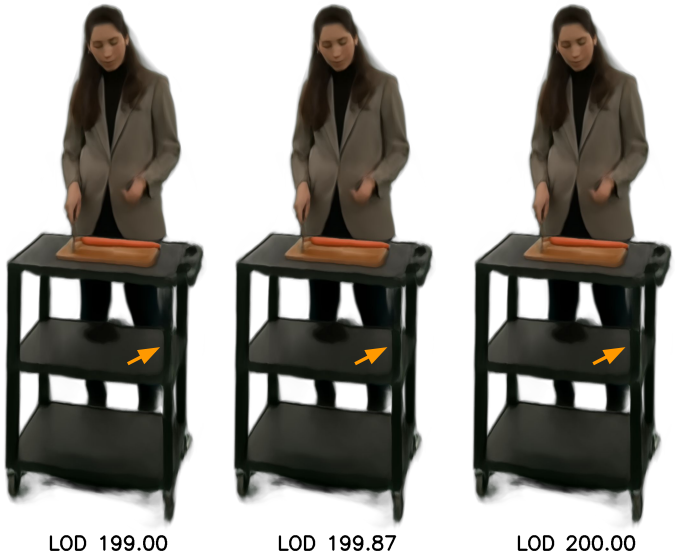
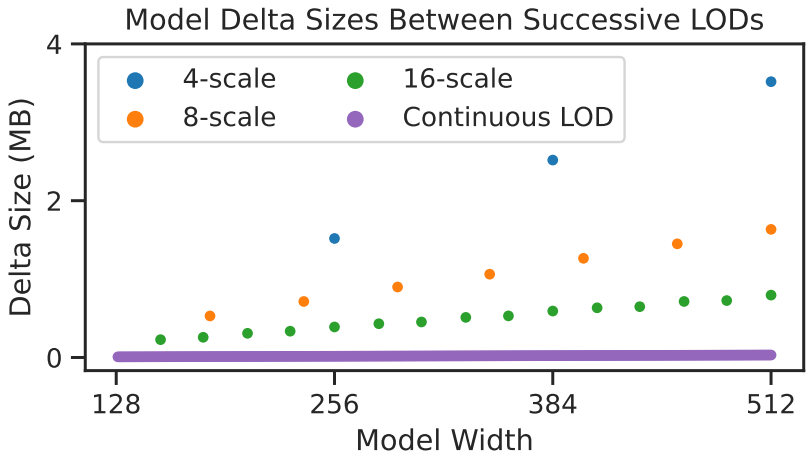
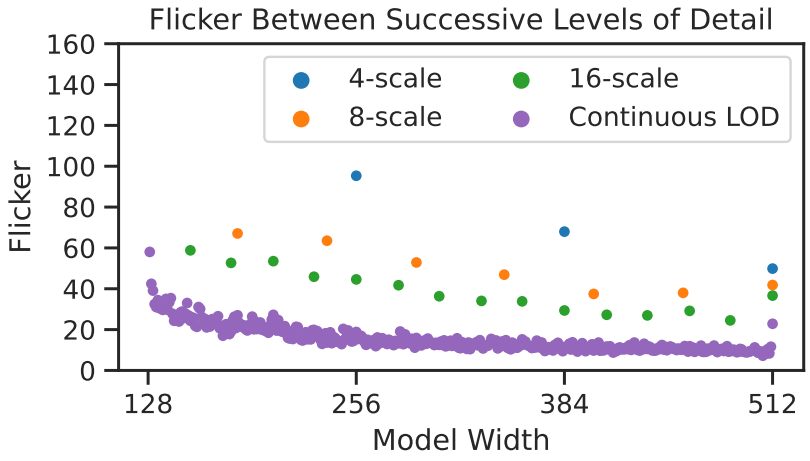


Figure 4.6: Neuron masking enables generating continuous levels of detail from discrete model widths. In the figure above, we see that one leg of the cart expands as the fractional level of detail α fades in new neurons. This effect is better seen in the accompanying supplementary video.



(a) Model Delta Sizes



(b) Average Flicker

Figure 4.7: Plots showing the effects of transitioning across LODs. Transitioning with discrete LODs leads to larger network traffic spikes and more flickering.

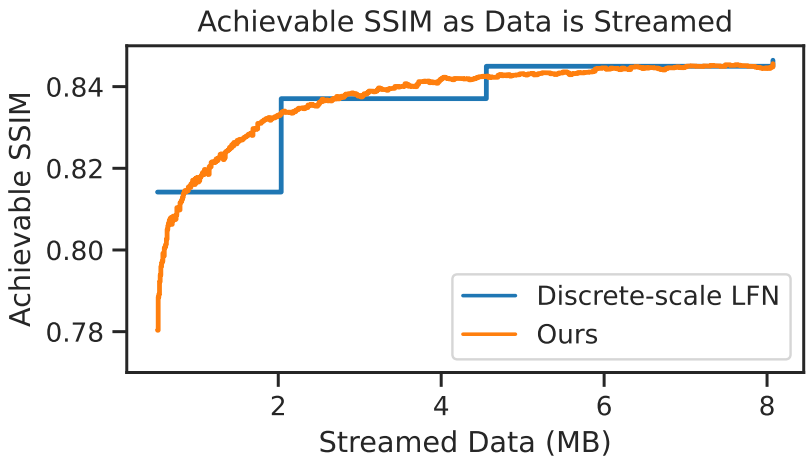
frequency sums: s_L and s_H. Finally, the flicker metric is computed by adding these together: Flicker $= s_L + s_H$.

We compare against three discrete-scale baselines with 4, 8, and 16 levels of detail, with 8 and 16 LODs trained using summed-area table sampling. In our continuous LOD case, we render views at the highest LOD corresponding to each discrete width (i.e. LOD 1.0, 2.0, ..., 385.0), using the static ground truth view as the reference frames. Flicker values are computed for each LOD using the transition from the next lower LOD and then averaged across all test views. Our flicker results are shown in Figure 4.7b. With only four LODs, the discrete-scale LFN method has three transitions, each with large model deltas (up to 3.5 MB) and high flicker values. Additional levels of detail reduce the model delta sizes and the flicker values with our continuous LOD method minimizing the model delta sizes and the flicker values. With our method, the LOD can be transitioned in small (≤ 32 KB) gradual steps.

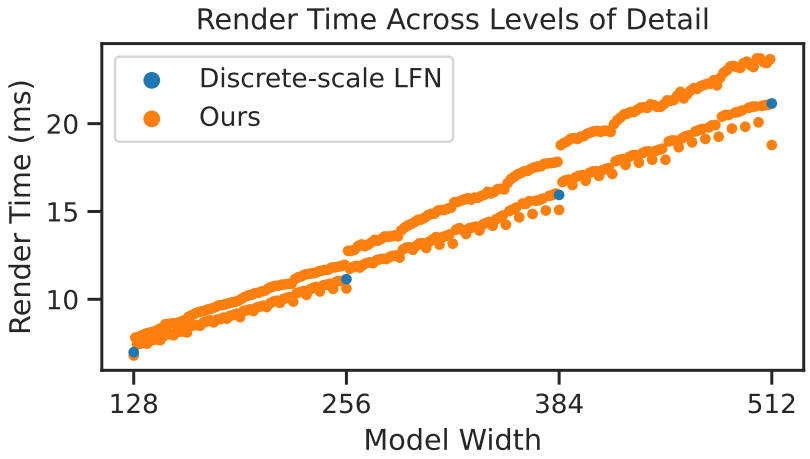
4.4.4 Rendering Quality

Quantitative PSNR and SSIM results are shown in Table 4.1. First, we see that adding summed-area table filtering to discrete-scale light field networks with four scales results in slightly improved PSNR and SSIM results while enabling arbitrary-scale sampling. Training a continuous LOD network impacts the performance at the original four LODs but allows us to have continuous LODs. Adding importance sampling allows us to focus on salient regions without significantly impacting the quantitative results.

Qualitative results of our saliency-based importance sampling ablation are shown in Figure 4.5. We see that details along faces appear at earlier LODs when using saliency for importance



(a) Average SSIM quality at the full resolution.



(b) Average render times at $1/4$ scale.

Figure 4.8: Plots showing our quantitative evaluation results. With continuous LODs, the LOD can be dynamically adjusted to maximize the quality based on available resources.

Model	1/8	1/4	1/2	1/1
Discrete-scale LFN	29.34	29.68	28.39	27.41
+ SAT filtering	29.77	29.99	28.54	27.46
+ Continuous LOD	27.87	29.77	28.38	27.38
+ Importance Sampling	28.06	29.79	28.44	27.40

(a) PSNR (dB)

Model	1/8	1/4	1/2	1/1
Discrete-scale LFN	0.8809	0.8763	0.8503	0.8465
+ SAT filtering	0.8898	0.8806	0.8513	0.8466
+ Continuous LOD	0.8370	0.8743	0.8484	0.8460
+ Importance Sampling	0.8380	0.8751	0.8487	0.8455

(b) SSIM

Table 4.1: Quantitative Training Ablation Results at 1/8, 1/4, 1/2, and 1/1 scales. Each scale is evaluated at its corresponding LOD.

sampling. All of these details resolve at the highest LODs with and without using importance sampling.

4.4.5 Rendering Performance

We evaluate the rendering performance by rendering training views across each LOD. For our rendering benchmarks, we use half-precision inference and skip empty rays with the auxiliary network which evaluates ray occupancy. Rendering performance results across the LODs are shown in Figure 4.8b. We observe that as the LOD increases according to the width of the neural network, rendering times increase as well. When rendering from a discrete-scale light field network with only four LODs, the user or application would need to select either the next higher or lower LOD, compromising on either the performance or the quality. With continuous LODs, software incorporating our light field networks would be able to gradually increase or decrease the LOD to maintain a better balance between performance and quality. In cases where

the ideal model size is not known, continuous LODs allow dynamic adjusting of the LOD to satisfy a target frame rate. In our PyTorch implementation, we observe that LODs with odd model widths have a slower render time than LODs with even model widths. LODs with model widths that are a multiple of eight perform slightly faster than other even model widths.

4.5 Discussion

By requiring light field networks to output reasonable results at each possible hidden layer width and incorporating neuron masking, we can achieve continuous of LODs. However, this applies additional constraints on the network as it needs to produce additional outputs. In our experiments, we observe slightly worse PSNR and SSIM results at the specific LODs corresponding to the $1/8$ and $1/4$ scales compared to the discrete-scale LFN which is trained with only four LODs. This is expected due to the additional constraints and less supervision at those specific LODs. The goal of our importance sampling procedure is to improve the quality of the salient regions of the light field rather than to maximize quantitative results.

Light field networks require additional cameras compared to neural radiance fields due to the lack of multi-view consistency prior provided by volume rendering. Hence, training light field networks requires additional cameras or regularization [120] compared to NeRF methods. Furthermore, light field networks do not use positional encoding [135] and represent high-frequency details as faithfully as NeRF methods. As the primary goal of our work is to enable highly granular rendering trade-offs with more levels of detail, we leave these limitations to future work.

4.6 Conclusion

In this chapter, we introduce continuous levels of details for light field networks using three techniques. First, we introduce summed area table sampling to sample colors from arbitrary scales of an image without generating multiple versions of each training image in a light field. Second, we achieve continuous LODs by combining arbitrary-width networks with neuron masking. Third, we train using saliency-based importance sampling to help details in the salient regions of the light field resolve at earlier LODs. With our method for continuous LODs, we hope to make light field networks more practical for 6DoF desktop and virtual reality applications [56, 57, 58].

Chapter 5: Conclusion

As virtual and augmented reality applications continue to develop and gradually become more widespread, the ability to stream photorealistic immersive content becomes ever more important. In this dissertation, we have explored methods for adaptive streaming of 360° videos and light fields. We first present a log-rectilinear transformation to stream foveated 360° videos. With our method, 360° videos can be streamed with foveation to significantly reduce bandwidth usage over streaming the entire 360° video at a fixed quality while also reducing the flickering artifacts of foveation from subsampling. Next, we present a progressive multi-scale light field network. With our method, light fields can be progressively streamed and rendered at multiple scales without significant aliasing and flickering artifacts. Finally, we present continuous levels of details for light field networks. To further reduce artifacts during transitions between levels of detail and enable more granular adaptivity to available resources, our method presents light field networks that can continuously interpolate between hundreds of network widths.

We envision 360° video suitable for seated VR experiences where users are immersed across their entire field of view and can rotate around the scene. Meanwhile, the full 6DoF experience of light fields allows the creation of fully interactive applications where users can move around and view different light fields across an entire 3D scene. With 3D applications becoming more interactive, photorealistic 6-DoF will become necessary for high-quality experiences.

Neural representations such as NeRFs and LFNs are currently able to synthesize photorealistic renders but still have remaining limitations compared to traditional representations such as meshes. We hope that the work presented in this dissertation provides a few steps toward making neural representations more suitable for practical streaming scenarios.

Bibliography

[1] Ioannis Agtzidis, Mikhail Startsev, and Michael 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.

[2] Stefan Winkler, Elisa Drelie Gelasca, and Touradj Ebrahimi. Toward perceptual metrics for video watermark evaluation. In Andrew G. Tescher, editor, *Applications of Digital Image Processing XXVI*, volume 5203, pages 371 – 378. International Society for Optics and Photonics, SPIE, 2003. doi: 10.1117/12.512550. URL https://doi.org/10.1117/12.512550.

[3] David Li, Ruofei Du, Adharsh Babu, Camelia D. Brumar, and Amitabh Varshney. A log-rectilinear transformation for foveated 360-degree video streaming. *IEEE Transactions on Visualization and Computer Graphics*, 27(5):2638–2647, 2021. doi: 10.1109/TVCG.2021.3067762.

[4] David Li and Amitabh Varshney. Progressive multi-scale light field networks. In *2022 International Conference on 3D Vision (3DV)*, pages 231–241, 2022. doi: 10.1109/3DV57658.2022.00035.

[5] David Li, Brandon Yushan Feng, and Amitabh Varshney. Continuous levels of detail for light field networks. In *34th British Machine Vision Conference 2023, BMVC 2023, Aberdeen, UK, November 20-24, 2023*. BMVA, 2023. URL https://papers.bmvc2023.org/0139.pdf.

[6] Feng Qian, Bo Han, Qingyang Xiao, and Vijay 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, pages 99–114, New York, NY, USA, 2018. ACM. ISBN 978-1-4503-5903-0. doi: 10.1145/3241539.3241565. URL http://doi.acm.org/10.1145/3241539.3241565.

[7] Yu Guan, Chengyuan Zheng, Xinggong Zhang, Zongming Guo, and Junchen 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, page 394–407, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450359566. doi: 10.1145/3341302.3342063. URL https://doi.org/10.1145/3341302.3342063.

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

[9] Jihoon Ryoo, Kiwon Yun, Dimitris Samaras, Samir R. Das, and Gregory 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, pages 6:1–6:11, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4297-1. doi: 10.1145/2910017.2910592.

[10] Xiaoxu Meng, Ruofei Du, Matthias Zwicker, and Amitabh Varshney. Kernel foveated rendering. *Proc. ACM Comput. Graph. Interact. Tech.*, 1(1):5:1–5:20, July 2018. ISSN 2577-6193. doi: 10.1145/3203199. URL http://doi.acm.org/10.1145/3203199.

[11] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *ECCV*, 2020.

[12] Vincent Sitzmann, Semon Rezchikov, William T. Freeman, Joshua B. Tenenbaum, and Fredo Durand. Light field networks: Neural scene representations with single-evaluation rendering. In *arXiv*, 2021.

[13] Zoe Landgraf, Alexander Sorkine Hornung, and Ricardo Silveira Cabral. PINs: Progressive Implicit Networks for Multi-Scale Neural Representations. In *International Conference on Machine Learning (ICML 2022)*, 2022.

[14] Feng Qian, Lusheng Ji, Bo Han, and Vijay Gopalakrishnan. Optimizing 360 video delivery over cellular networks. In *Proceedings of the 5th Workshop on All Things Cellular: Operations, Applications and Challenges*, ATC '16, pages 1–6, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4249-0. doi: 10.1145/2980055.2980056. URL http://doi.acm.org/10.1145/2980055.2980056.

[15] D.V. Nguyen, Huyen T.T. Tran, and Truong Cong Thang. A client-based adaptation framework for 360-degree video streaming. *Journal of Visual Communication and Image Representation*, 59:231 – 243, 2019. ISSN 1047-3203. doi: https://doi.org/10.1016/j.jvcir.2019.01.012. URL http://www.sciencedirect.com/science/article/pii/S1047320319300124.

[16] 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)*, pages 347–348, March 2017. doi: 10.1109/VR.2017.7892319.

[17] HyunWook Kim, JinWook Yang, MinSu Choi, JunSuk Lee, SangPil Yoon, YoungHwa Kim, and WooChool Park. Eye tracking based foveated rendering for 360 VR tiled video. In *Proceedings of the 9th ACM Multimedia Systems Conference*, MMSys '18, page 484–486, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450351928. doi: 10.1145/3204949.3208111. URL https://doi.org/10.1145/3204949.3208111.

[18] Oleg Komogortsev and Javed Khan. Predictive perceptual compression for real time video communication. In *Proceedings of the 12th Annual ACM International Conference on Multimedia*, MULTIMEDIA '04, pages 220–227, New York, NY, USA, 2004. ACM. ISBN 1-58113-893-8. doi: 10.1145/1027527.1027577. URL http://doi.acm.org/10.1145/1027527.1027577.

[19] Wei-Tse Lee, Hsin-I Chen, Ming-Shiuan Chen, I-Chao Shen, and Bing-Yu Chen. High-resolution 360 video foveated stitching for real-time VR. *Computer Graphics Forum*, 36(7):115–123, 2017. doi: 10.1111/cgf.13277. URL https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.13277.

[20] 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. ISSN 1077-2626. doi: 10.1109/TVCG.2018.2794119.

[21] MHD Yamen Saraiji, Kouta Minamizawa, and Susumu Tachi. Foveated streaming: Optimizing video streaming for telexistence systems using eye-gaze based foveation. Technical report, Taichi Lab, Sep 2017.

[22] Rachel Albert, Anjul Patney, David Luebke, and Joohwan Kim. Latency requirements for foveated rendering in virtual reality. *ACM Trans. Appl. Percept.*, 14(4), September 2017. ISSN 1544-3558. doi: 10.1145/3127589. URL https://doi.org/10.1145/3127589.

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

[24] 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.

[25] Shou-Cheng Yen, Ching-Ling Fan, and Cheng-Hsin 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, pages 7–12, New York, NY, USA, 2019. ACM. ISBN 978-1-4503-6300-6. doi: 10.1145/3304114.3325616. URL http://doi.acm.org/10.1145/3304114.3325616.

[26] Xing Liu, Qingyang Xiao, Vijay Gopalakrishnan, Bo Han, Feng Qian, and Matteo Varvello. 360° innovations for panoramic video streaming. In *Proceedings of the 16th ACM Workshop on Hot Topics in Networks*, HotNets-XVI, pages 50–56, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-5569-8. doi: 10.1145/3152434.3152443. URL http://doi.acm.org/10.1145/3152434.3152443.

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

[28] Jounsup Park and Klara Nahrstedt. Navigation graph for tiled media streaming. In *Proceedings of the 27th ACM International Conference on Multimedia*, MM '19, page 447–455, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450368896. doi: 10.1145/3343031.3351021. URL https://doi.org/10.1145/3343031.3351021.

[29] Brian Guenter, Mark Finch, Steven Drucker, Desney Tan, and John Snyder. Foveated 3d graphics. *ACM Trans. Graph.*, 31(6):164:1–164:10, November 2012. ISSN 0730-0301. doi: 10.1145/2366145.2366183. URL http://doi.acm.org/10.1145/2366145.2366183.

[30] Wilson S. Geisler and Jeffrey S. Perry. Real-time foveated multiresolution system for low-bandwidth video communication. In Bernice E. Rogowitz and Thrasyvoulos N. Pappas, editors, *Human Vision and Electronic Imaging III*, volume 3299, pages 294 – 305. International Society for Optics and Photonics, SPIE, 1998. doi: 10.1117/12.320120. URL https://doi.org/10.1117/12.320120.

[31] Anjul Patney, Marco Salvi, Joohwan Kim, Anton Kaplanyan, Chris Wyman, Nir Benty, David Luebke, and Aaron Lefohn. Towards foveated rendering for gaze-tracked virtual reality. *ACM Trans. Graph.*, 35(6):179:1–179:12, November 2016. ISSN 0730-0301. doi: 10.1145/2980179.2980246.

[32] 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)*, pages 1–2, March 2018. doi: 10.1109/VR.2018.8446142.

[33] Anton S Kaplanyan, Anton Sochenov, Thomas Leimkühler, Mikhail Okunev, Todd Goodall, and Gizem 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.

[34] Franklin C. Crow. Summed-area tables for texture mapping. In *Proceedings of the 11th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '84, pages 207–212. ACM, 1984. ISBN 0-89791-138-5. doi: 10.1145/800031.808600. URL http://doi.acm.org/10.1145/800031.808600.

[35] 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)*, pages 763–772, May 2018. doi: 10.1109/IPDPSW.2018.00123.

[36] 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)*, pages 11–20, Nov 2017. doi: 10.1109/CANDAR.2017.35.

[37] 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, pages 251–260, Sep. 2014. doi: 10.1109/ICPP.2014.34.

[38] Diego Nehab, André Maximo, Rodolfo S. Lima, and Hugues Hoppe. GPU-efficient recursive filtering and summed-area tables. *ACM Trans. Graph.*, 30(6):176:1–176:12, December 2011. ISSN 0730-0301. doi: 10.1145/2070781.2024210. URL http://doi.acm.org/10.1145/2070781.2024210.

[39] Justin Hensley, Thorsten Scheuermann, Greg Coombe, Montek Singh, and Anselmo 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.

[40] Guy E. Blelloch. Prefix Sums And Their Applications. In *Synthesis of Parallel Algorithms*. M. Kaufmann, 5 2004. doi: 10.1184/R1/6608579.v1. URL https://kilthub.cmu.edu/articles/Prefix_sums_and_their_applications/6608579.

[41] José Díaz, Pere-Pau Vázquez, Isabel Navazo, and Florent Duguet. Real-time ambient occlusion and halos with summed area tables. *Computers & Graphics*, 34 (4):337 – 350, 2010. ISSN 0097-8493. doi: https://doi.org/10.1016/j.cag.2010.03.005. URL http://www.sciencedirect.com/science/article/pii/S0097849310000440. Procedural Methods in Computer Graphics Illustrative Visualization.

[42] Kefei Lei and John 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, pages 119–128, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-1956-0. doi: 10.1145/2448196.2448215. URL http://doi.acm.org/10.1145/2448196.2448215.

[43] Paul Viola and Michael 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] 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.*, volume 1, pages I–I, June 2003. doi: 10.1109/CVPR.2003.1211403.

[45] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In Aleš Leonardis, Horst Bischof, and Axel Pinz, editors, *Computer Vision – ECCV 2006*, pages 404–417, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg. ISBN 978-3-540-33833-8.

[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] Richard S. Wallace, Ping-Wen Ong, Benjamin B. Bederson, and Eric L. Schwartz. Space variant image processing. *International Journal of Computer Vision*, 13(1):71–90, Sep

1994. ISSN 1573-1405. doi: 10.1007/BF01420796. URL https://doi.org/10.1007/BF01420796.

[48] 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.

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

[50] Wenxiao Zhang, Bo Han, and Pan Hui. Jaguar: Low latency mobile augmented reality with flexible tracking. In *Proceedings of the 26th ACM International Conference on Multimedia*, MM '18, page 355–363, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450356657. doi: 10.1145/3240508.3240561. URL https://doi.org/10.1145/3240508.3240561.

[51] Shu Shi, Varun Gupta, Michael Hwang, and Rittwik Jana. Mobile VR on edge cloud: A latency-driven design. In *Proceedings of the 10th ACM Multimedia Systems Conference*, MMSys '19, page 222–231, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450362979. doi: 10.1145/3304109.3306217. URL https://doi.org/10.1145/3304109.3306217.

[52] Shivang Aggarwal, Sibendu Paul, Pranab Dash, Nuka Saranya Illa, Y. Charlie Hu, Dimitrios Koutsonikolas, and Zhisheng Yan. How to evaluate mobile 360° video streaming systems? In *Proceedings of the 21st International Workshop on Mobile Computing Systems and Applications*, HotMobile '20, page 68–73, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450371162. doi: 10.1145/3376897.3377865. URL https://doi.org/10.1145/3376897.3377865.

[53] 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)*, pages 972–973, March 2019. doi: 10.1109/VR.2019.8797830.

[54] 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)*, pages 157–160, Dec 2018. doi: 10.1109/AIVR.2018.00033.

[55] X. Meng, R. Du, and A. Varshney. Eye-dominance-guided foveated rendering. *IEEE Transactions on Visualization and Computer Graphics*, 26(5):1972–1980, 2020. doi: 10.1109/TVCG.2020.2973442.

[56] Ruofei Du, David Li, and Amitabh Varshney. Geollery: A mixed reality social media platform. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, CHI '19, page 1–13, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450359702. doi: 10.1145/3290605.3300915. URL https://doi.org/10.1145/3290605.3300915.

[57] Ruofei Du, David Li, and Amitabh Varshney. Project geollery.com: Reconstructing a live mirrored world with geotagged social media. In *The 24th International Conference on 3D Web Technology*, Web3D '19, page 1–9, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450367981. doi: 10.1145/3329714.3338126. URL https://doi.org/10.1145/3329714.3338126.

[58] David Li, Eric Lee, Elijah Schwelling, Mason G. Quick, Patrick Meyers, Ruofei Du, and Amitabh Varshney. Meteovis: Visualizing meteorological events in virtual reality. In *Extended Abstracts of the 2020 CHI Conference on Human Factors in Computing Systems*, CHI EA '20, page 1–9, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450368193. doi: 10.1145/3334480.3382921. URL https://doi.org/10.1145/3334480.3382921.

[59] Mingsong Dou, Sameh Khamis, Yury Degtyarev, Philip Davidson, Sean Ryan Fanello, Adarsh Kowdle, Sergio Orts Escolano, Christoph Rhemann, David Kim, Jonathan Taylor, Pushmeet Kohli, Vladimir Tankovich, and Shahram Izadi. Fusion4D: Real-Time Performance Capture of Challenging Scenes. *ACM Trans. Graph.*, 35(4), jul 2016. ISSN 0730-0301. doi: 10.1145/2897824.2925969. URL https://doi.org/10.1145/2897824.2925969.

[60] Ruofei Du, Ming Chuang, Wayne Chang, Hugues Hoppe, and Amitabh Varshney. Montage4D: Interactive seamless fusion of multiview video textures. In *Proceedings of ACM Interactive 3D Graphics (I3D) 2018*, 2018.

[61] Tinghui Zhou, Richard Tucker, John Flynn, Graham Fyffe, and Noah Snavely. Stereo magnification: Learning view synthesis using multiplane images. In *SIGGRAPH*, 2018.

[62] Jonathan Shade, Steven Gortler, Li-wei He, and Richard Szeliski. Layered depth images. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '98, page 231–242, New York, NY, USA, 1998. Association for Computing Machinery. ISBN 0897919998. doi: 10.1145/280814.280882. URL https://doi.org/10.1145/280814.280882.

[63] Benjamin Attal, Selena Ling, Aaron Gokaslan, Christian Richardt, and James Tompkin. MatryODShka: Real-time 6DoF video view synthesis using multi-sphere images. In *European Conference on Computer Vision (ECCV)*, August 2020. URL https://visual.cs.brown.edu/matryodshka.

[64] Jonathan T. Barron, Ben Mildenhall, Matthew Tancik, Peter Hedman, Ricardo Martin-Brualla, and Pratul P. Srinivasan. Mip-nerf: A multiscale representation for anti-aliasing neural radiance fields, 2021.

[65] Alex Yu, Ruilong Li, Matthew Tancik, Hao Li, Ren Ng, and Angjoo Kanazawa. PlenOctrees for real-time rendering of neural radiance fields. In *ICCV*, 2021.

[66] Ayush Tewari, Justus Thies, Ben Mildenhall, Pratul Srinivasan, Edgar Tretschk, Yifan Wang, Christoph Lassner, Vincent Sitzmann, Ricardo Martin-Brualla, Stephen Lombardi,

Tomas Simon, Christian Theobalt, Matthias Niessner, Jonathan T. Barron, Gordon Wetzstein, Michael Zollhoefer, and Vladislav Golyanik. Advances in neural rendering, 2021.

[67] Albert Pumarola, Enric Corona, Gerard Pons-Moll, and Francesc Moreno-Noguer. D-NeRF: Neural Radiance Fields for Dynamic Scenes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020.

[68] Keunhong Park, Utkarsh Sinha, Jonathan T. Barron, Sofien Bouaziz, Dan B Goldman, Steven M. Seitz, and Ricardo Martin-Brualla. Nerfies: Deformable neural radiance fields. *ICCV*, 2021.

[69] Keunhong Park, Utkarsh Sinha, Peter Hedman, Jonathan T. Barron, Sofien Bouaziz, Dan B Goldman, Ricardo Martin-Brualla, and Steven M. Seitz. Hypernerf: A higher-dimensional representation for topologically varying neural radiance fields. *ACM Trans. Graph.*, 40(6), dec 2021.

[70] Stephan J Garbin, Marek Kowalski, Matthew Johnson, Jamie Shotton, and Julien Valentin. Fastnerf: High-fidelity neural rendering at 200fps. *arXiv preprint arXiv:2103.10380*, 2021.

[71] Christian Reiser, Songyou Peng, Yiyi Liao, and Andreas Geiger. KiloNeRF: Speeding up neural radiance fields with thousands of tiny mlps. *arXiv preprint arXiv:2103.13744*, 2021.

[72] Thomas Neff, Pascal Stadlbauer, Mathias Parger, Andreas Kurz, Joerg H. Mueller, Chakravarty R. Alla Chaitanya, Anton S. Kaplanyan, and Markus Steinberger. DONeRF: Towards Real-Time Rendering of Compact Neural Radiance Fields using Depth Oracle Networks. *Computer Graphics Forum*, 40(4), 2021. ISSN 1467-8659. doi: 10.1111/cgf.14340. URL `https://doi.org/10.1111/cgf.14340`.

[73] Alex Yu, Sara Fridovich-Keil, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. Plenoxels: Radiance fields without neural networks, 2021.

[74] Jianxiong Shen, Adria Ruiz, Antonio Agudo, and Francesc Moreno-Noguer. Stochastic neural radiance fields: Quantifying uncertainty in implicit 3d representations. In *2021 International Conference on 3D Vision (3DV)*, pages 972–981, 2021. doi: 10.1109/3DV53792.2021.00105.

[75] Alex Yu, Vickie Ye, Matthew Tancik, and Angjoo Kanazawa. pixelNeRF: Neural radiance fields from one or few images, 2020.

[76] Daniel Rebain, Mark Matthews, Kwang Moo Yi, Dmitry Lagun, and Andrea Tagliasacchi. Lolnerf: Learn from one look, 2021.

[77] Katja Schwarz, Yiyi Liao, Michael Niemeyer, and Andreas Geiger. Graf: Generative radiance fields for 3d-aware image synthesis. *arXiv preprint arXiv:2007.02442*, 2020.

[78] Michael Niemeyer and Andreas Geiger. Campari: Camera-aware decomposed generative neural radiance fields. In *2021 International Conference on 3D Vision (3DV)*, pages 951–961, 2021. doi: 10.1109/3DV53792.2021.00103.

[79] Christopher Xie, Keunhong Park, Ricardo Martin-Brualla, and Matthew Brown. Fig-nerf: Figure-ground neural radiance fields for 3d object category modelling. In *2021 International Conference on 3D Vision (3DV)*, pages 962–971, 2021. doi: 10.1109/3DV53792.2021.00104.

[80] Zhengqi Li, Simon Niklaus, Noah Snavely, and Oliver Wang. Neural scene flow fields for space-time view synthesis of dynamic scenes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6498–6508, 2021.

[81] Wenqi Xian, Jia-Bin Huang, Johannes Kopf, and Changil Kim. Space-time neural irradiance fields for free-viewpoint video. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9421–9431, 2021.

[82] Liao Wang, Jiakai Zhang, Xinhang Liu, Fuqiang Zhao, Yanshun Zhang, Yingliang Zhang, Minye Wu, Lan Xu, and Jingyi Yu. Fourier plenoctrees for dynamic radiance field rendering in real-time, 2022.

[83] Julian Chibane, Aayush Bansal, Verica Lazova, and Gerard Pons-Moll. Stereo radiance fields (srf): Learning view synthesis for sparse views of novel scenes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7911–7920, June 2021.

[84] Michael Niemeyer, Jonathan T. Barron, Ben Mildenhall, Mehdi S. M. Sajjadi, Andreas Geiger, and Noha Radwan. Regnerf: Regularizing neural radiance fields for view synthesis from sparse inputs, 2021.

[85] Ajay Jain, Matthew Tancik, and Pieter Abbeel. Putting nerf on a diet: Semantically consistent few-shot view synthesis. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 5885–5894, October 2021.

[86] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *arXiv:2201.05989*, January 2022.

[87] Konstantinos Rematas, Andrew Liu, Pratul P. Srinivasan, Jonathan T. Barron, Andrea Tagliasacchi, Thomas Funkhouser, and Vittorio Ferrari. Urban radiance fields, 2021.

[88] Yuanbo Xiangli, Linning Xu, Xingang Pan, Nanxuan Zhao, Anyi Rao, Christian Theobalt, Bo Dai, and Dahua Lin. Bungeenerf: Progressive neural radiance field for extreme multi-scale scene rendering. In *The European Conference on Computer Vision (ECCV)*, 2022.

[89] Haithem Turki, Deva Ramanan, and Mahadev Satyanarayanan. Mega-nerf: Scalable construction of large-scale nerfs for virtual fly-throughs, 2021.

[90] Jonathan T. Barron, Ben Mildenhall, Dor Verbin, Pratul P. Srinivasan, and Peter Hedman. Mip-NeRF 360: Unbounded anti-aliased neural radiance fields. *CVPR*, 2022.

[91] David B. Lindell, Dave Van Veen, Jeong Joon Park, and Gordon Wetzstein. BACON: Band-limited coordinate networks for multiscale scene representation. *arXiv preprint arXiv:2112.04645*, 2021.

[92] Rizal Fathony, Anit Kumar Sahu, Devin Willmott, and J. Zico Kolter. Multiplicative filter networks. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL https://openreview.net/forum?id=OmtmcPkkhT.

[93] Brandon Yushan Feng and Amitabh Varshney. SIGNET: Efficient neural representation for light fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 14224–14233, October 2021.

[94] Zhong Li, Liangchen Song, Celong Liu, Junsong Yuan, and Yi Xu. NeuLF: Efficient Novel View Synthesis with Neural 4D Light Field. In Abhijeet Ghosh and Li-Yi Wei, editors, *Eurographics Symposium on Rendering*. The Eurographics Association, 2022. ISBN 978-3-03868-187-8. doi: 10.2312/sr.20221156.

[95] Paramanand Chandramouli, Hendrik Sommerhoff, and Andreas Kolb. Light field implicit representation for flexible resolution reconstruction, 2021. URL https://arxiv.org/abs/2112.00185.

[96] Marc Levoy and Pat Hanrahan. Light field rendering. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '96, page 31–42, New York, NY, USA, 1996. Association for Computing Machinery. ISBN 0897917464. doi: 10.1145/237170.237199. URL https://doi.org/10.1145/237170.237199.

[97] Julian Ost, Issam Laradji, Alejandro Newell, Yuval Bahat, and Felix Heide. Neural point light fields. *CoRR*, abs/2112.01473, 2021. URL https://arxiv.org/abs/2112.01473.

[98] Benjamin Attal, Jia-Bin Huang, Michael Zollhöfer, Johannes Kopf, and Changil Kim. Learning neural light fields with ray-space embedding networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[99] Peter Hedman, Pratul P. Srinivasan, Ben Mildenhall, Jonathan T. Barron, and Paul Debevec. Baking neural radiance fields for real-time view synthesis. *ICCV*, 2021.

[100] Lance Williams. Pyramidal parametrics. *SIGGRAPH Comput. Graph.*, 17(3):1–11, jul 1983. ISSN 0097-8930. doi: 10.1145/964967.801126. URL https://doi.org/10.1145/964967.801126.

[101] Towaki Takikawa, Joey Litalien, Kangxue Yin, Karsten Kreis, Charles Loop, Derek Nowrouzezahrai, Alec Jacobson, Morgan McGuire, and Sanja Fidler. Neural geometric level of detail: Real-time rendering with implicit 3d shapes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11358–11367, June 2021.

[102] Zhang Chen, Yinda Zhang, Kyle Genova, Sean Fanello, Sofien Bouaziz, Christian Häne, Ruofei Du, Cem Keskin, Thomas Funkhouser, and Danhang Tang. Multiresolution deep implicit functions for 3d shape representation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 13087–13096, October 2021.

[103] Guo-Wei Yang, Wen-Yang Zhou, Hao-Yang Peng, Dun Liang, Tai-Jiang Mu, and Shi-Min Hu. Recursive-nerf: An efficient and dynamically growing nerf, 2021.

[104] Towaki Takikawa, Alex Evans, Jonathan Tremblay, Thomas Müller, Morgan McGuire, Alec Jacobson, and Sanja Fidler. Variable bitrate neual fields. In *SIGGRAPH 2022*. ACM, 2022.

[105] Junwoo Cho, Seungtae Nam, Daniel Rho, Jong Hwan Ko, and Eunbyung Park. Streamable neural fields. In *ECCV*, 2022.

[106] Andrei A. Rusu, Neil C. Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks, 2016. URL https://arxiv.org/abs/1606.04671.

[107] Tolga Bolukbasi, Joseph Wang, Ofer Dekel, and Venkatesh Saligrama. Adaptive neural networks for efficient inference. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 527–536. PMLR, 06–11 Aug 2017. URL https://proceedings.mlr.press/v70/bolukbasi17a.html.

[108] Hyunho Yeo, Youngmok Jung, Jaehong Kim, Jinwoo Shin, and Dongsu Han. Neural adaptive content-aware internet video delivery. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pages 645–661, Carlsbad, CA, October 2018. USENIX Association. ISBN 978-1-939133-08-3. URL https://www.usenix.org/conference/osdi18/presentation/yeo.

[109] Wangchunshu Zhou, Canwen Xu, Tao Ge, Julian McAuley, Ke Xu, and Furu Wei. Bert loses patience: Fast and robust inference with early exit. In *Advances in Neural Information Processing Systems*, volume 33, pages 18330–18341. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/d4dd111a4fd973394238aca5c05bebe3-Paper.pdf.

[110] Weijie Liu, Peng Zhou, Zhe Zhao, Zhiruo Wang, Haotang Deng, and Qi Ju. FastBERT: a self-distilling bert with adaptive inference time. In *Proceedings of ACL 2020*, 2020.

[111] Le Yang, Yizeng Han, Xi Chen, Shiji Song, Jifeng Dai, and Gao Huang. Resolution adaptive networks for efficient inference. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[112] Jiahui Yu, Linjie Yang, Ning Xu, Jianchao Yang, and Thomas Huang. Slimmable neural networks. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=H1gMCsAqY7.

[113] Xiaoxu Meng, Ruofei Du, Joseph JaJa, and Amitabh Varshney. 3D-Kernel Foveated Rendering for Light Fields. *IEEE Transactions on Visualization and Computer Graphics*, 27 (8):3350–3360, Mar. 2020. doi: 10.1109/TVCG.2020.2975801.

[114] Johannes Lutz Schönberger and Jan-Michael Frahm. Structure-from-motion revisited. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[115] Johannes Lutz Schönberger, Enliang Zheng, Marc Pollefeys, and Jan-Michael Frahm. Pixelwise view selection for unstructured multi-view stereo. In *European Conference on Computer Vision (ECCV)*, 2016.

[116] Shanchuan Lin, Andrey Ryabtsev, Soumyadip Sengupta, Brian Curless, Steve Seitz, and Ira Kemelmacher-Shlizerman. Real-time high-resolution background matting. *arXiv*, pages arXiv–2012, 2020.

[117] Zhaolin Xiao, Qing Wang, Guoqing Zhou, and Jingyi Yu. Aliasing detection and reduction in plenoptic imaging. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014.

[118] Ingmar Peter and Wolfgang Straßer. The wavelet stream: Interactive multi resolution light field rendering. In Steven J. Gortler and Karol Myszkowski, editors, *Rendering Techniques 2001*, pages 127–138, Vienna, 2001. Springer Vienna. ISBN 978-3-7091-6242-2.

[119] Brandon Feng, Yinda Zhang, Danhang Tang, Ruofei Du, and Amitabh Varshney. PRIF: Primary Ray-Based Implicit Function. In *European Conference on Computer Vision*, ECCV. Springer, 2022.

[120] Brandon Feng, Susmija Jabbireddy, and Amitabh Varshney. Viinter: View interpolation with implicit neural representations of images. In *SIGGRAPH ASIA*, 2022.

[121] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[122] Vincent Sitzmann, Julien N.P. Martel, Alexander W. Bergman, David B. Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. In *Proc. NeurIPS*, 2020.

[123] Jonathan T. Barron, Ben Mildenhall, Dor Verbin, Pratul P. Srinivasan, and Peter Hedman. Zip-NeRF: Anti-aliased grid-based neural radiance fields. *ICCV*, 2023.

[124] Nianchen Deng, Zhenyi He, Jiannan Ye, Budmonde Duinkharjav, Praneeth Chakravarthula, Xubo Yang, and Qi Sun. Fov-nerf: Foveated neural radiance fields for virtual reality. *IEEE Transactions on Visualization and Computer Graphics*, pages 1–11, 2022. doi: 10.1109/TVCG.2022.3203102.

[125] Jiatao Gu, Lingjie Liu, Peng Wang, and Christian Theobalt. Stylenerf: A style-based 3d aware generator for high-resolution image synthesis. In *International Conference on Learning Representations*, 2022.

[126] Wenyuan Zhang, Ruofan Xing, Yunfan Zeng, Yu-Shen Liu, Kanle Shi, and Zhizhong Han. Fast learning radiance fields by shooting much fewer rays. *arXiv preprint arXiv:2208.06821*, 2022.

[127] Junli Cao, Huan Wang, Pavlo Chemerys, Vladislav Shakhrai, Ju Hu, Yun Fu, Denys Makoviichuk, Sergey Tulyakov, and Jian Ren. Real-time neural light field on mobile devices, 2022. URL https://arxiv.org/abs/2212.08057.

[128] Steven J. Gortler, Radek Grzeszczuk, Richard Szeliski, and Michael F. Cohen. The lumigraph. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '96, page 43–54, New York, NY, USA, 1996. Association for Computing Machinery. ISBN 0897917464. doi: 10.1145/237170.237200. URL https://doi.org/10.1145/237170.237200.

[129] Brandon Yushan Feng and Amitabh Varshney. Neural subspaces for light fields. *IEEE Transactions on Visualization and Computer Graphics*, pages 1–11, 2022. doi: 10.1109/TVCG.2022.3224674.

[130] Vishwanath Saragadam, Jasper Tan, Guha Balakrishnan, Richard G. Baraniuk, and Ashok Veeraraghavan. MINER: multiscale implicit neural representations. *CoRR*, abs/2202.03532, 2022. URL https://arxiv.org/abs/2202.03532.

[131] Paul S. Heckbert. Filtering by repeated integration. In *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '86, page 315–321, New York, NY, USA, 1986. Association for Computing Machinery. ISBN 0897911962. doi: 10.1145/15922.15921. URL https://doi.org/10.1145/15922.15921.

[132] Peter Kovesi. Fast almost-gaussian filtering. In *2010 International Conference on Digital Image Computing: Techniques and Applications*, pages 121–125, 2010. doi: 10.1109/DICTA.2010.30.

[133] Jiahui Yu and Thomas S Huang. Universally slimmable networks and improved training techniques. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 1803–1811, 2019.

[134] Alexander Kroner, Mario Senden, Kurt Driessens, and Rainer Goebel. Contextual encoder-decoder network for visual saliency prediction. *Neural Networks*, 129:261–270, 2020. ISSN 0893-6080. doi: https://doi.org/10.1016/j.neunet.2020.05.004. URL http://www.sciencedirect.com/science/article/pii/S0893608020301660.

[135] Matthew Tancik, Pratul Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan Barron, and Ren Ng. Fourier features let networks learn high frequency functions in low dimensional domains. In

H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 7537–7547. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/55053683268957697aa39fba6f231c68-Paper.pdf.