ABSTRACT

Title of thesis: FOVEATED RENDERING TECHNIQUES IN MODERN COMPUTER GRAPHICS

Xiaoxu Meng, Master of Science, 2018

Thesis directed by: Professor Joseph F. JaJa
Electrical and Computer Engineering

Thesis co-directed by: Professor Amitabh Varshney
Computer Science

Foveated rendering coupled with eye-tracking has the potential to dramatically accelerate interactive 3D graphics with minimal loss of perceptual detail. I have developed a new foveated rendering technique: Kernel Foveated Rendering (KFR), which parameterizes foveated rendering by embedding polynomial kernel functions in log-polar mapping. This GPU-driven technique uses parameterized foveation that mimics the distribution of photoreceptors in the human retina. I present a two-pass kernel foveated rendering pipeline that maps well onto modern GPUs. In the first pass, I compute the kernel log-polar transformation and render to a reduced-resolution buffer. In the second pass, I have carried out the inverse-log-polar transformation with anti-aliasing to map the reduced-resolution rendering to the full-resolution screen. I carry out user studies to empirically identify the KFR parameters and observe a $2.8X - 3.2X$ speedup in rendering on $4K\ UHD$ ($2160p$) displays. The eye-tracking-guided kernel foveated rendering can resolve the mutually conflicting goals of interactive rendering and perceptual realism.

FOVEATED RENDERING TECHNIQUE IN MODERN GRAPHICS

by

Xiaoxu Meng

Thesis 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
Master of Science
2018

Advisory Committee:
Professor Joseph F. JaJa: *Advisor, Chair*
Professor Amitabh Varshney: *Co-advisor*
Professor Tudor Dumitras

Acknowledgments

I owe my gratitude to all the people who have made this thesis possible and because of whom my graduate experience has been one that I will cherish forever.

First and foremost I'd like to thank my advisor, Professor Joseph F. JaJa for giving me an invaluable opportunity to work on challenging and extremely interesting projects over the past three years. He has always made himself available for help and advice and there has never been an occasion when I've knocked on his door and he hasn't given me time. It has been a pleasure to work with and learn from such an extraordinary individual.

I would also like to thank my co-advisor, Dr. Amitabh Varshney. Without his extraordinary theoretical ideas and computational expertise, this thesis would have been a distant dream.

Thanks are due to Professor Tudor Dumitras for agreeing to serve on my thesis committee and for sparing his invaluable time.

My colleagues at the Graphics and Visual Informatics Laboratory (GVIL) have enriched my graduate life in many ways and deserve a special mention. Ruofei Du helped me in solving many technical questions. My interactions with Hsueh-Chien Cheng, Eric Krokos, Xuetong Sun, Tara Larrue, Shuo Li, Somay Jain, Mukul Agarwal, David Li, and Alexander Rowden have been very fruitful.

I would also like to acknowledge help and support from some of the staff members. Tom Ventsias, Barbara Brawn-Cinani, Sida Li, Eric Lee, and Jonathan Heagerty's technical help and encouragement are highly appreciated.

I owe my deepest thanks to my family – my mother and father who have always stood by me and guided me through my career, and have pulled me through against impossible odds at times. Words cannot express the gratitude I owe them. I would also like to thank Yunchuan Li, Hua Luo, and Hong Wei who are like family members to me.

I would like to acknowledge financial support from UMIACS for all the projects discussed herein.

It is impossible to remember all, and I apologize to those I've inadvertently left out.

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

1.1 Overview

Human vision spans a field of view of $135^\circ \times 160^\circ$, but the highest-resolution foveal vision covers only the central $1.5^\circ \times 2^\circ$ [3]. Patney *et al.* [5] have estimated that in modern virtual reality head-mounted displays (HMD) only 4% of the pixels are mapped onto the fovea. Therefore, foveated rendering techniques that allocate more computational resources for foveal pixels and fewer resources elsewhere can dramatically speed up rendering [13] for large displays, especially for virtual and augmented reality headsets equipped with eye trackers.

Araujo and Dias [14] use a log-polar mapping to approximate the excitation of the cortex in the human vision system. The classic log-polar transformation has been used for foveating 2D images on the GPU [15]. However, to the best of my knowledge, direct use of the log-polar mapping for 3D graphics has not yet been attempted on GPUs.

In my thesis, I present a kernel foveated rendering pipeline for modern GPUs that parameterizes foveated rendering by embedding polynomial kernel functions in the classic log-polar mapping. This allows us to easily vary the sampling density and distribution, and match them to human perception in virtual reality HMDs. In

contrast to adaptive sampling in Cartesian coordinates, which requires a complex interpolation process [10] and the classic three-pass foveated rendering pipeline [3], KFR just needs a two-pass algorithm. In the first pass, I carry out the kernel log-polar transformation and render to a reduced-resolution framebuffer using deferred shading [16,17]. In the second pass, I apply the inverse kernel log-polar transformation to the reduced-resolution framebuffer to map the final foveated rendering to the full-resolution display.

I have built several foveated renderings with varying sampling density and distribution and evaluate them via pilot and final user studies. I have found the optimal parameters with minimal perceptual errors that correspond to the distribution of photoreceptors in the retina. This algorithm is designed to achieve a high frame rate by shading fewer pixels in the peripheral vision. Finally, I have validated my approach on 3D rendering of textured meshes as well as ray-marching scenes.

The KFR pipeline is broadly applicable for eye-tracking devices, and efficiently testing, or previewing real-time rendering results with global lighting and physically based rendering.

In summary, my contributions include:

1. designing the kernel log-polar mapping algorithm to enable a parameterized trade-off of visual quality and rendering speed for foveated rendering,

2. conducting user studies to identify the kernel foveated rendering parameters governing the sampling distribution and density to maximize perceptual realism and minimize computation,

3. mapping kernel foveated rendering onto the GPU to achieve speedups of $2.8X$ for textured 3D meshes and $3.2X$ for ray-casting scenes for 3840×2160 displays with minimal perceived loss of detail.

1.2 Human Vision System

There is a large research literature that documents the falloff of accuracy in visual periphery. Most of the previous research on foveated perception has been surveyed in [18]. Recent research has also addressed the issue of perception time for depicting information in the far peripheral field [19].

A commonly used model is the *linear acuity model*, which measures the minimum angle of resolution (MAR). A linear model matches both anatomical data and is applicable for many low-level vision tasks [18]. However, the model only works for the "central" vision (with angular radius $\leq 8°$), after which MAR rises more steeply [3]. In the periphery, receptors become increasingly sparse relative to the eyes' optical system Nyquist limit.

Another model is the *log acuity model*. It has been found that the excitation of the cortex can be approximated by a log-polar mapping of the eye's retinal image [20]. The calculation of this model is cheap and fast, thus being used in many practical applications such as computer vision, robotics, and other fields.

Curcio [1, 2] proposed the mixed acuity model. As shown in Figure 1.1, the Ganglion cell (orange) density tends to match the photoreceptor density in the fovea (left), but many photoreceptors map to the same Ganglion cell away from the fovea.

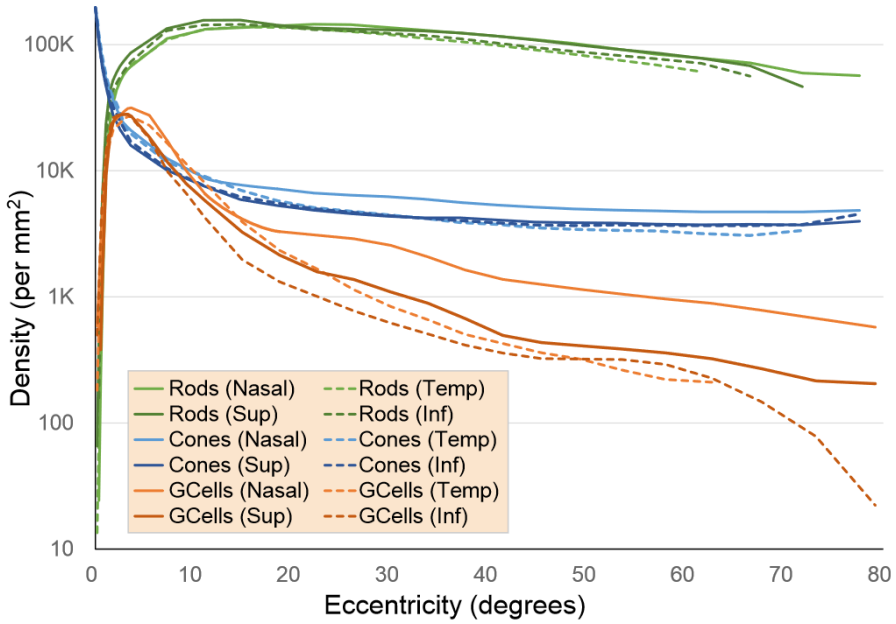


Figure 1.1: Spatial distribution of various retinal components, using data from [1] and [2]. Ganglion cell (orange) density tends to match photoreceptor density in the fovea (left), but away from the fovea many photoreceptors map to the same Ganglion cell. 'Nasal', 'Temp', 'Sup' and 'Inf' indicate the four directions (nasal, temporal, superior, inferior) away from the fovea.

'Nasal', 'Temp', 'Sup' and 'Inf' indicate the four directions (nasal, temporal, superior, inferior) away from the fovea.

Most of the foveated rendering algorithms are inspired by the acuity models mentioned above. However, it is not easy to quantify the pixel density rate by using the models mentioned above. Previous research has addressed the issue of adjusting visual acuity by conducting user studies with eye tracking technologies.

Kortum *et al.* [21] invented a foveated imaging system and validate their system using eye tracking. Minimal perceptual artifacts are reported with bandwidth reductions of up to 94.7%. Guenter *et al.* [3] conducted user study by letting the users compare the foveated image and non-foveated image. They found a more aggressive mean threshold and a more conservative threshold, which achieved speedups of 5.7 and 4.8 compared to non-foveated rendering. Patney *et al.* [6] conducted user study by fitting a psychometric function to the user performance for different shading strategies, and found a satisfactory shading rate which provides the most speedup.

1.3 Foveation Techniques

1.3.1 Foveated Images and Videos

The last few decades have seen significant advances in foveated rendering for 2D images and videos.

Burt [22] has generated foveated images with multi-resolution Gaussian pyramids. He takes advantage of a coarse-to-fine scheme to adaptively select the critical information for constructing the foveated image. Kortum and Geisler [21] have

developed one of the earliest eye-tracking-based foveated imaging systems with space-variant degradation. The structure of their image foveation system is shown in Figure 1.2. Using 256×256 8-bit gray-scale images, they have achieved bandwidth reduction of up to 94.7% with minimal perceptual artifacts. Other image foveation techniques include embedded zero-tree wavelets [23], set partitioning in hierarchical trees [24], wavelet-based image foveation [25], embedded foveation image coding [26], and gigapixel displays [27].

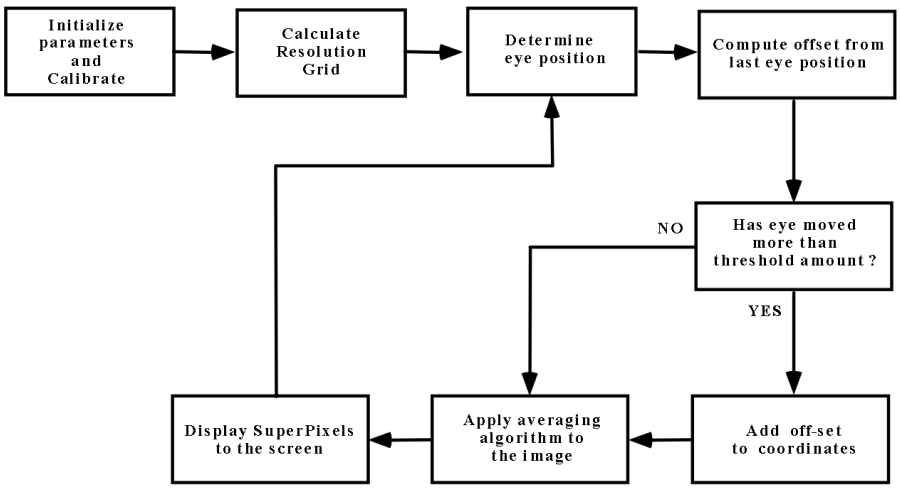


Figure 1.2: General flow diagram of the operation of the foveated imaging system.

Video foveation has also been explored [28–30]. The filter bank method is used for video preprocessing before using standard video compression algorithms (*e.g. MPEG and H.26x*) [29, 31, 32]. Foveation filtering has been implemented with the quantization processes in standard MPEG and H.26x compression [33, 34]. Video foveation coupled with eye tracking could reduce overall system latency, including network latency, processing latency, and display latency [35]. An overview of video foveation is shown in Figure 1.3.

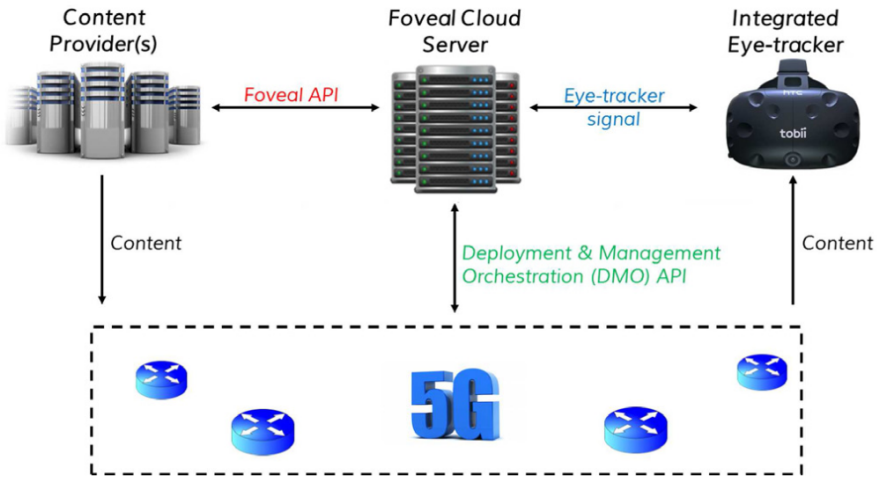


Figure 1.3: Overview of the envisioned end-to-end architecture for supporting gaze-aware streaming solutions for VR.

While previous work in foveation for images and videos provides strong foundations, most of these methods cannot be easily generalized for interactive 3D graphics rendering on modern GPUs. A notable exception is the work by Antonelli *et al.* [15], which uses log-polar mapping to speed-up 2D image rendering on modern GPUs. However, their approach does not directly work with 3D graphics primitives and does not use kernel functions.

1.3.2 Foveated 3D Graphics

Weier *et al.* [36] have reviewed several approaches for foveated rendering including mesh simplification in the areas of lower acuity [37–39]. However, these days shading has often been found to dominate the cost for rendering sophisticated scenes on modern graphics pipelines [4, 8].

Ragan-Kelley *et al.* [40] use decoupled sampling for stochastic super-sampling

of motion and defocus blur at a reduced shading cost. Guenter *et al.* [3] present a three-pass pipeline for foveated 3D rendering by using three eccentricity layers around the tracked gaze point. As shown in Figure 1.4, the innermost layer is rendered at the highest resolution (native display), while the successively outer peripheral layers are rendered with progressively lower resolution and coarser level of detail (LOD). They interpolate and blend between the layers and use frame jitter and temporal re-projection to reduce spatial and temporal artifacts. However, this approach renders the scene three times, which requires lots of rendering resources.

Vaidyanathan *et al.* [4] present a novel approach using a generalization of multi-sample anti-aliasing (MSAA). They perform foveated rendering by sampling coarse pixels (2×2 pixels and 4×4 pixels) in the peripheral regions as shown in Figure 1.5. This approach targets small-form-factor devices with high resolution, such as phones and tablets rather than HMDs. It therefore presents two challenges for HMDs: the effective pixel size in current HMDs is too large for MSAA, and gaze-dependent motions exaggerate the artifacts.

Patney *et al.* [5, 6] address temporal artifacts in foveated rendering by using pre-filters and temporal anti-aliasing. Because human eyes are sensitive to edges, they add contrast preservation for the foveated image, which greatly enhances the image quality by reducing the tunneling effect as shown in Figure 1.6. They tested the foveated rendering effect on both Desktop and VR headset.

Clarberg *et al.* [7] propose a modification to the current hardware architecture, which enables flexible control of shading rates and automatic shading reuse between triangles in tessellated primitives as shown in Figure 1.7.

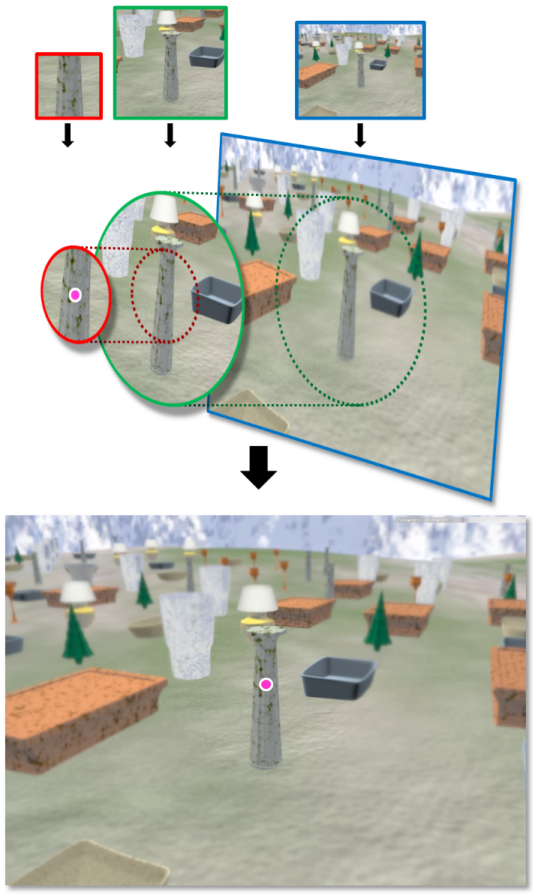


Figure 1.4: Guenter *et al.* [3] render three eccentricity layers (red border = inner layer, green = middle layer, blue = outer layer) around the tracked gaze point (pink dot), shown at their correct relative sizes in the top row. These are interpolated to native display resolution and smoothly composited to yield the final image at the bottom. Foveated rendering greatly reduces the number of pixels shaded and overall graphics computation.

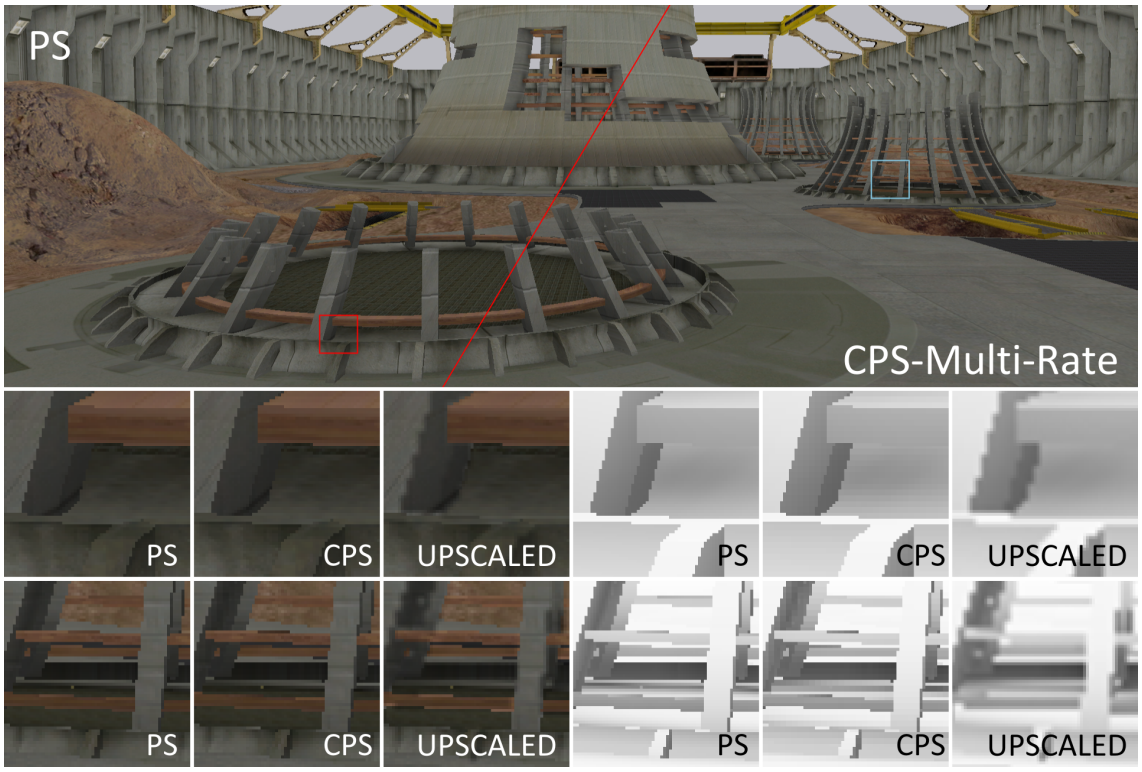


Figure 1.5: Vaidyanathan *et al.* [4] perform foveated rendering by sampling coarse pixels (2×2 pixels and 4×4 pixels) in the peripheral regions.

Figure 1.6: Patney *et al.* [5, 6] perform foveated rendering by sampling coarse pixels and address temporal artifacts in foveated rendering by using pre-filters and temporal anti-aliasing.

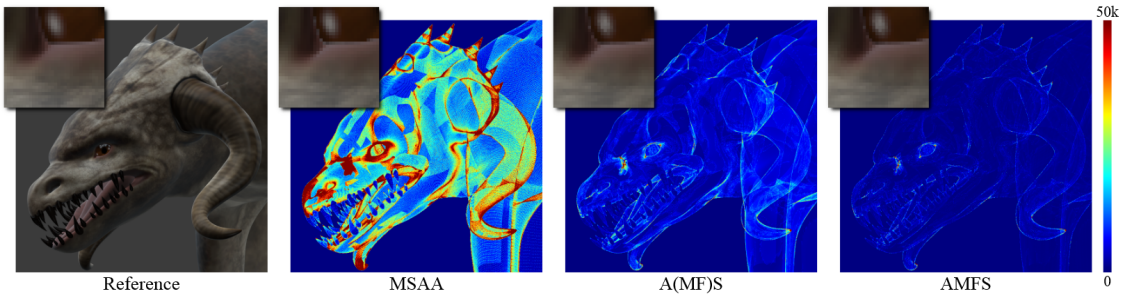


Figure 1.7: Clarberg *et al.* [7] have proposed an approach in which pixel shading is tied to the coarse input patches and reused between triangles, effectively decoupling the shading cost from the tessellation level, as shown in this example.

He *et al.* [8] introduce multi-rate GPU shading to support more shading samples near regions of specular highlights, shadows, edges, and motion blur regions, helping achieve a $3X$ to $5X$ speedup as shown in Figure 1.8. However, this implementation of multi-rate shading requires an extension of the graphics pipeline, which is not available on commodity graphics hardware.

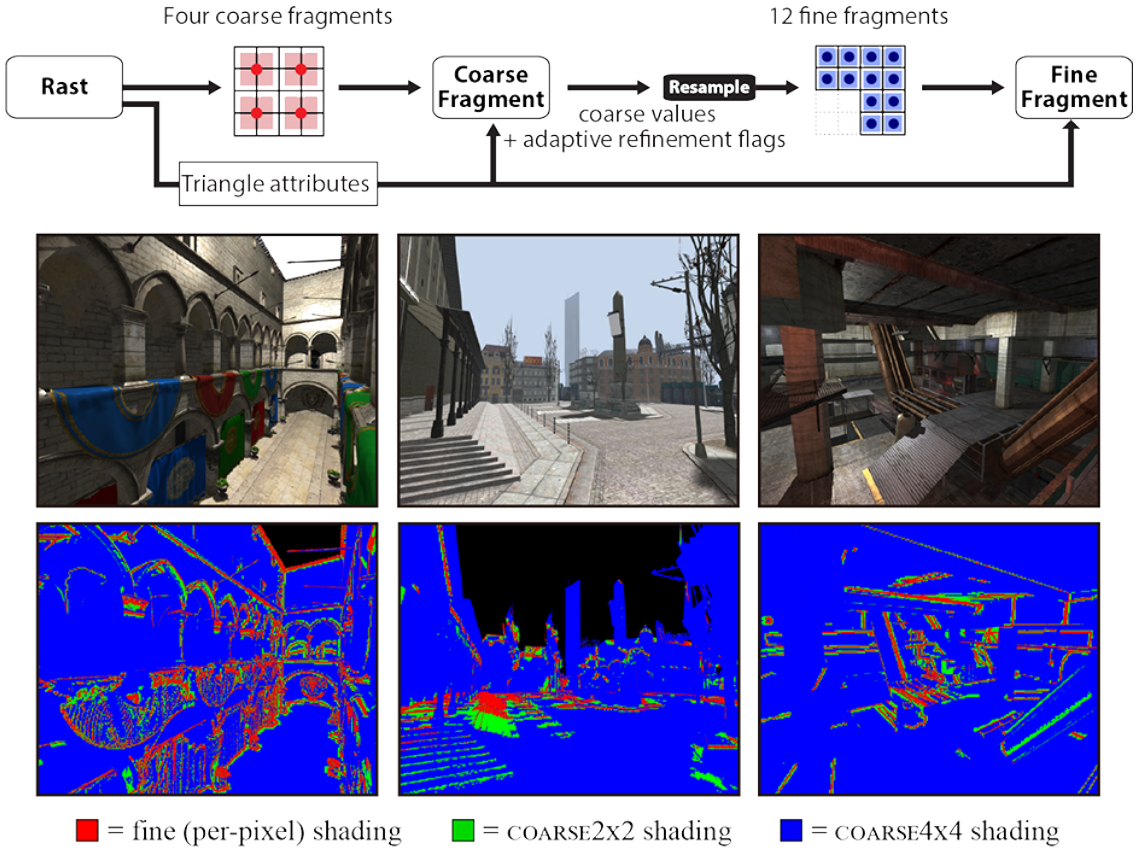


Figure 1.8: He *et al.* [8] introduce multi-rate GPU shading to support more shading samples near regions of specular highlights, shadows, edges, and motion blur regions, helping achieve a $3X$ to $5X$ speedup.

Swafford *et al.* [9] implement four foveated renderers as shown in Figure 1.9. The first method reduces the effective rendered pixel density of the peripheral region

while maintaining the base density of the foveal window, as shown in Figure 1.9 (a). The second varies per-pixel depth-buffer samples in the fovea and periphery for screen-space ambient occlusion. Although a very low number of per-pixel samples can cause banding, they expect these differences to go unnoticed in the periphery due to the loss of visual acuity and contrast sensitivity, as shown in Figure 1.9 (b). The third method normally casts rays to geometry and detects intersections with a given number of depth layers, represented as a series of RGBA textures mapped on the geometry, then it varies the per-pixel ray-casting steps across the field of view, as shown in Figure 1.9 (c). The final method implements a terrain renderer using GPU-level tessellation for the fovea. In order to determine the appropriate level of tessellation, we project the foveal window from screen coordinates into the scene. If a tile falls within either the foveal or peripheral field of view, the level of tessellation is set statically to the appropriate level. If the tile falls between the two regions (on the blending border) the level of tessellation is linearly interpolated between the two levels, as shown in Figure 1.9 (d).

Stengel *et al.* [10] use adaptive sampling from fovea to peripheral regions in a gaze-contingent rendering pipeline as shown in Figure 1.10. To compensate for the missing pixels caused by sparsely distributed shading samples on the periphery, they use pull-push [41] interpolation to create the full foveated image. This strategy achieves a reduction of rendertime of 25.4% (with speedup of $1.3X$) and reduction of shading time of 41% (with speedup of $1.7X$).

Sun *et al.* [11] design a real-time foveated 4D light field rendering and display system as shown in Figure 1.11. Their prototype renders only $16\% - 30\%$ of the rays

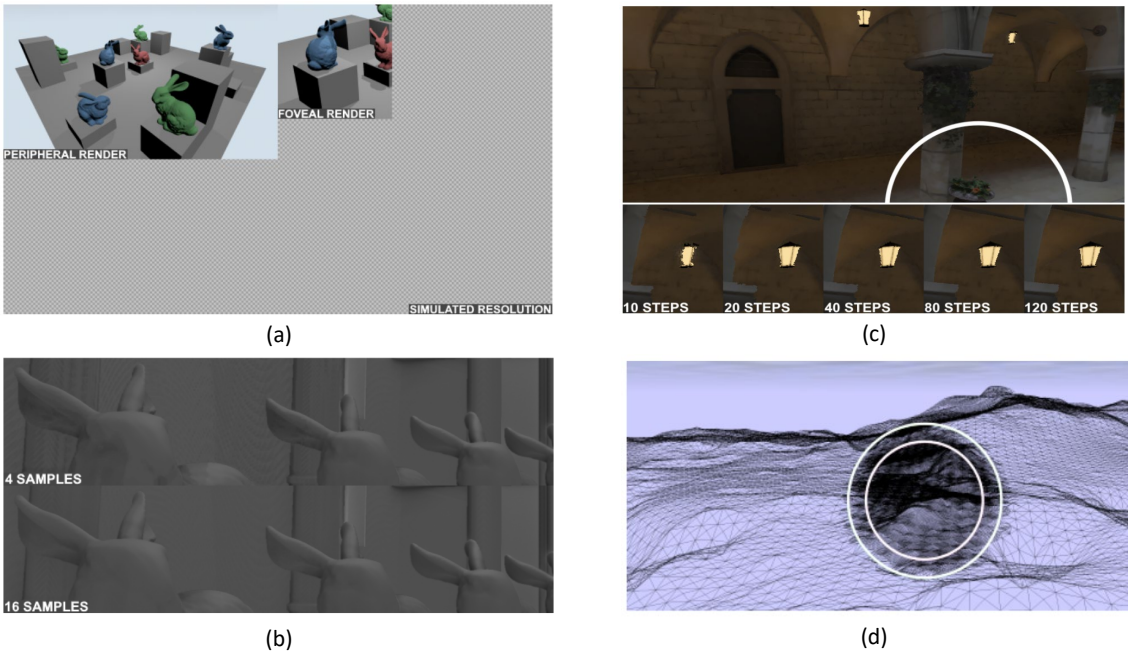


Figure 1.9: Swafford *et al.* [9] implement four foveated renderers as described in the text of the figure. (a) is the Annotated view of a foveated resolution render with moderate settings precomposition. The checkerboard area represents the proportion of pixels saved for the targeted simulated resolution; (b) is the strips from two foveated renders with the same fixation point (bottom-right) but different peripheral sampling levels. Region transition is handled smoothly, but at four samples there are noticeable artifacts in the peripheral region, such as banding; (c) Top: Sample frame from our ray-casting method with 120 per-pixel steps in the foveal region (within circle) and 10 per-pixel steps in the peripheral region (outwith circle). Bottom: Close-up ofright lamp showing artifacts across different quality levels; (d) is the Wireframe view of our foveated tessellation method. The inner circle is the foveal region, between circles is the inter-regional blending, and outside the circles is the peripheral region.

Figure 1.10: Stengel *et al.* [10] use adaptive sampling from fovea to peripheral regions in a gaze-contingent rendering pipeline and compensate for the missing pixels by pull-push interpolation.

without compromising the perceptual quality.

Recently, deferred shading has been used for antialiasing foveated rendering. Karis [42] optimizes temporal anti-aliasing for deferred shading, which uses samples over multiple frames to reduce flickering. Crassin *et al.* [43] reduce aliasing by pre-filtering sub-pixel geometric detail in the G-buffer for deferred shading. Chajdas *et al.* [44]'s subpixel anti-aliasing operates as a post-process on a rendered image with super-resolution depth and normal buffers. It targets deferred shading renderers that cannot use MSAA.

In this thesis, we present a simple two-pass foveated rendering pipeline that maps well onto modern GPUs. Kernel foveated rendering (KFR) provides gradually changing resolution and achieves $2.8X - 3.2X$ speedup with little perceptual loss.

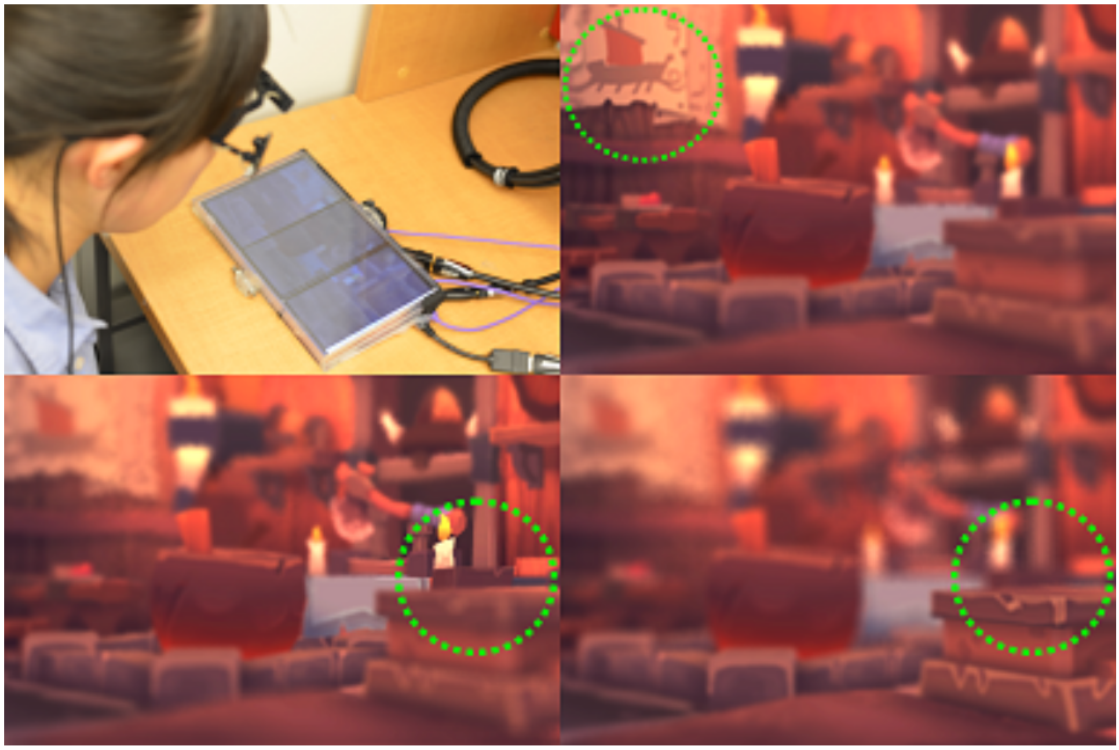


Figure 1.11: Sun *et al.* [11] design a real-time foveated 4D light field rendering and display system.

1.4 Eye Tracking

Eye tracking has been used in graphics for a number of applications, including validation of mesh saliency [45]. As the visual complexity rises in virtual worlds [46–48], biomedical datasets [49–51], and large images [52, 53], the need for eye-tracking to rapidly identify areas of interest and selectively render them at greater detail will also increase.

Eye tracking in VR is implemented by adding a series of infrared emitters and sensors embedded around the headset's lenses as shown in Figure 1.12. The headset uses the images collected by the sensors to calculate the position and direction of gaze for both eyes.

The development of eye-tracking technology makes foveated rendering possible. *FOVE* has released a foveated rendering headset with eye tracking in 2017. *HTC Vive* and *Qualcomm* are collaborating with *Tobii* in the development of an eye tracking VR headset.

Latency is one of the most important system components for eye tracking. According to Albert *et al.* [54], an extra eye tracking latency of 80 - 150 ms causes a significant reduction in acceptable amount of foveation, but a similar decrease in acceptable foveation was not found for shorter eye-tracking latencies of 20 ms 40 ms, suggesting that a total system latency of 50 ms to 70 ms could be tolerated.

Figure 1.12: *Tobii* eye-tracker in a HTC Vive VR headset.

Chapter 2: Kernel Foveated Rendering

2.1 Overview

Overall, my algorithm applies the kernel log-polar transformation for rasterization in a reduced-resolution log-polar buffer (LP-buffer), carries out shading within the LP-buffer, and then uses the inverse kernel log-polar transformation to render on the full resolution display. This is shown in Figure 1.13.

In the classic log-polar transformation [15], given a $W \times H$ pixel display screen, and an LP-buffer of $w \times h$ pixels, the screen-space pixel (x, y) in Cartesian coordinates is transformed to (u, v) in the log-polar coordinates according to Equation 2.1,

$$\begin{aligned} u &= \frac{\log\|x', y'\|_2}{L} \cdot w \\ v &= \frac{\arctan\left(\frac{y'}{x'}\right)}{2\pi} \cdot h + \mathbf{1}\left[y' < 0\right] \cdot h \end{aligned} \tag{2.1}$$

where, (x', y') represent (x, y) with respect to the center of the screen as origin, L is the log-distance from the center to the corner of the screen, and $\mathbf{1}[\cdot]$ is the indicator function,

$$x' = x - \frac{W}{2}, \quad y' = y - \frac{H}{2}, \quad L = \log\left(\left\|\frac{W}{2}, \frac{H}{2}\right\|_2\right) \tag{2.2}$$

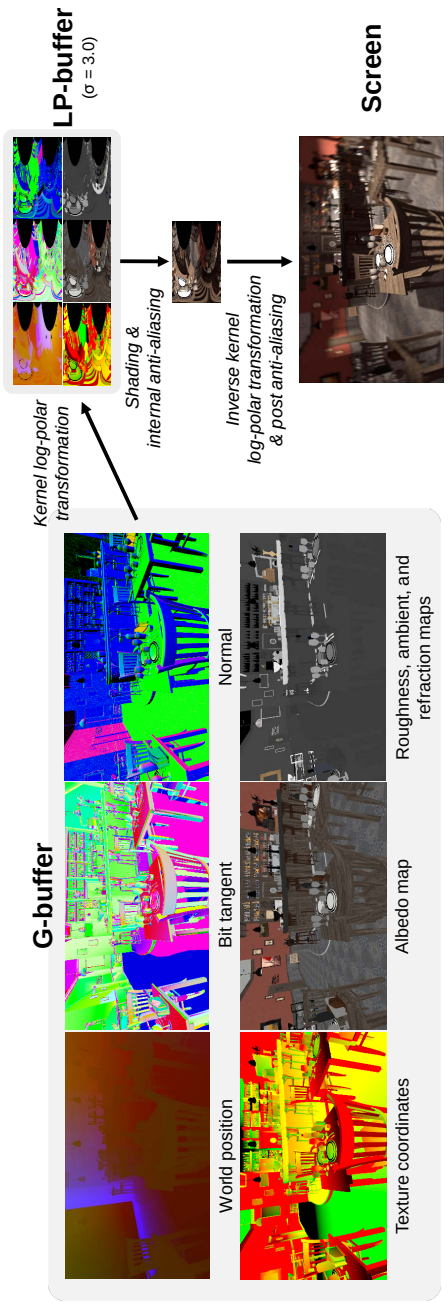


Figure 1.13: An overview of the KFR pipeline. I transform the necessary parameters and textures in the G-buffer from Cartesian coordinates to log-polar coordinates, compute lighting in the log-polar (LP) buffer and perform internal anti-aliasing. Next, I apply the inverse transformation to recover the framebuffer in Cartesian coordinates and employ post anti-aliasing to reduce the foveation artifacts.

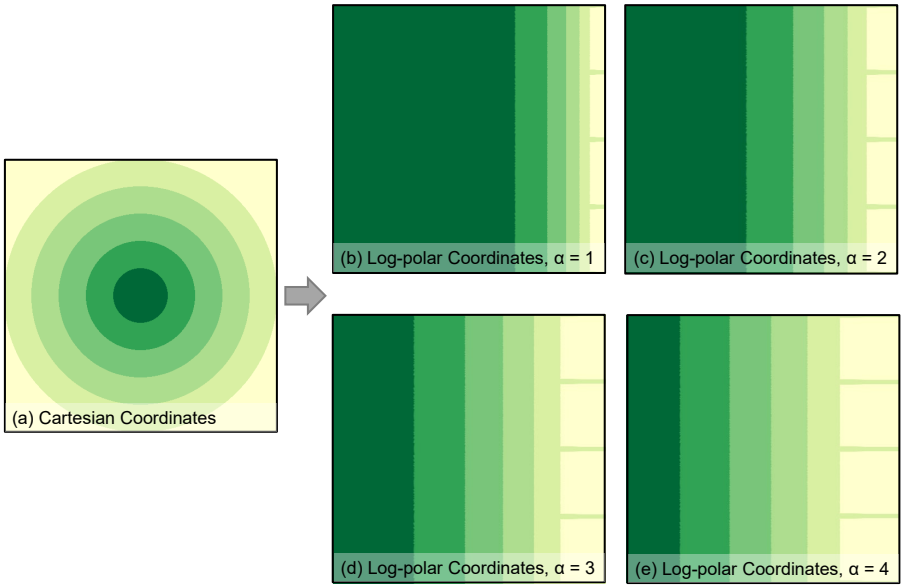


Figure 1.14: Transformation from Cartesian coordinates to log-polar coordinates with kernel function $\mathbf{K}(x) = x^{\alpha}$. (a) is the image in the Cartesian coordinates, (b)–(e) are the corresponding images in the log-polar coordinates with varying kernel parameter α. Matching colors in the log-polar and Cartesian coordinates show the same regions.

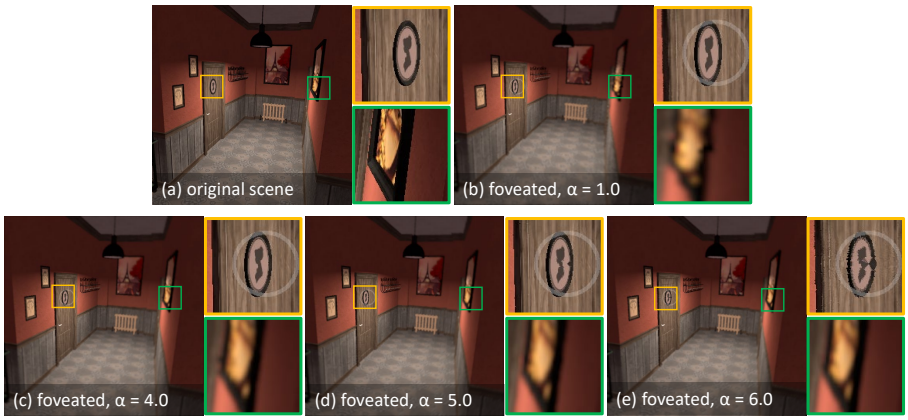


Figure 1.15: Comparison of foveated frame with different α (fovea is marked as the semi-transparent ring in the zoomed-in view): (a) original scene, (b) foveated with $\alpha = 1.0$, (c) foveated with $\alpha = 4.0$, (d) foveated with $\alpha = 5.0$, and (e) foveated with $\alpha = 6.0$. The lower zoomed-in views show that large α enhances the peripheral detail; the upper zoomed-in views show that when $\alpha \geq 5.0$, foveal quality suffers.

$$\mathbf{1}\left[y' < 0\right] = \begin{cases} 1 & , y' < 0 \\ 0 & , y' \geq 0 \end{cases} \tag{2.3}$$

Notice how the central dark green area in Figure 1.14 (a) is mapped to a relatively large region in the left part of the log-polar coordinates in Figure 1.14 (b), while the peripheral regions of Figure 1.14 (a) are mapped to a relatively small part of Figure 1.14 (b).

In the inverse log-polar transformation, a pixel with log-polar coordinates (u, v) is transformed back to (x'', y'') in Cartesian coordinates. Let

$$A = \frac{L}{w}, \quad B = \frac{2\pi}{h}, \tag{2.4}$$

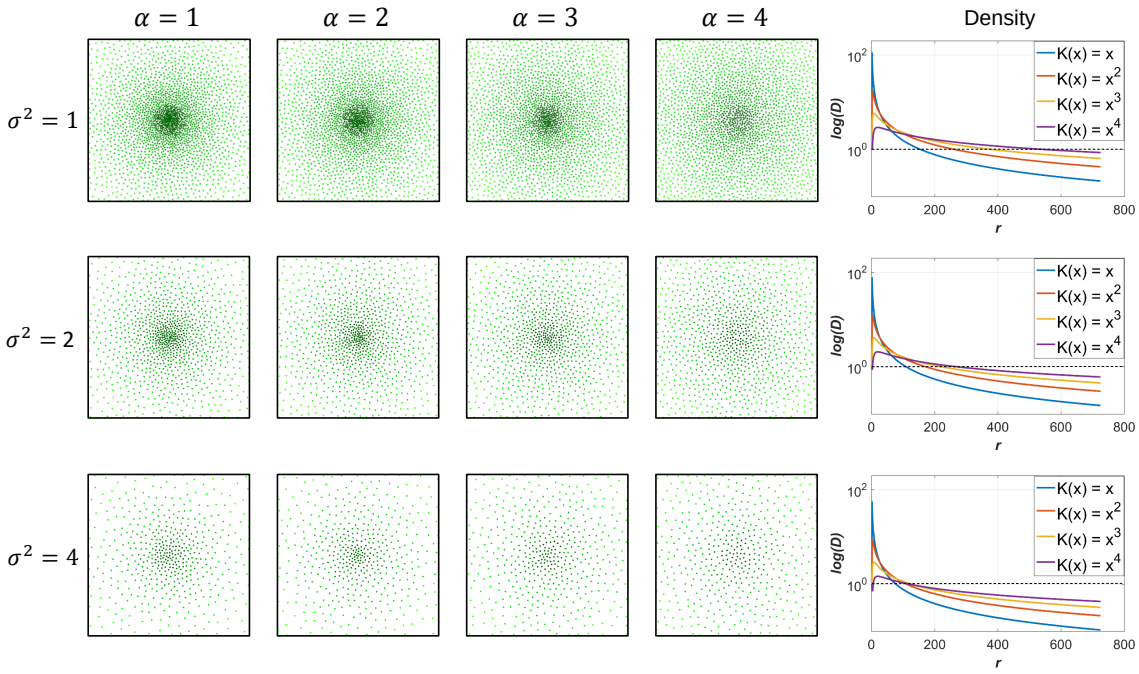


Figure 1.16: The relationship among σ^2, $\mathbf{K}(x) = x^\alpha$, and the sampling rate. The number of samples in each image is proportional to σ^2. I use a variant of the PixelPie algorithm [12] to generate the Poisson samples shown.

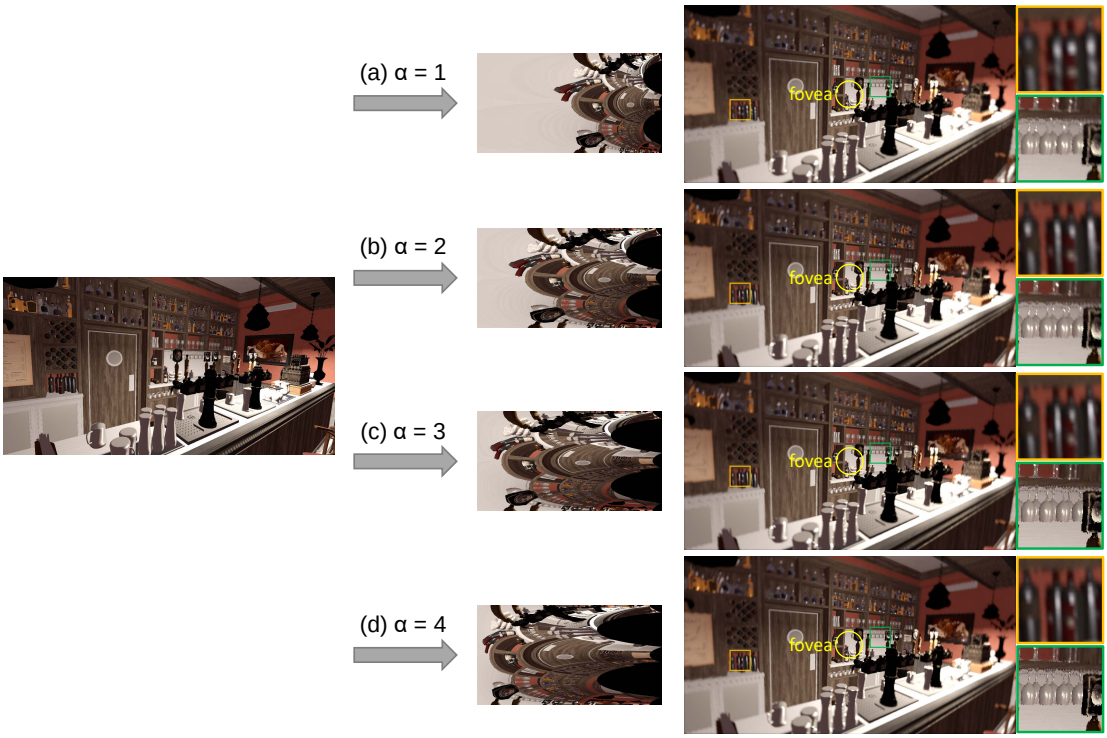


Figure 1.17: Comparison of foveated rendering with varying α for 2560×1440 resolution. From left to right: original rendering, kernel log-polar rendering, and the foveated rendering with zoomed-in view of the peripheral regions. Here $\sigma = 1.8$, (a) classic log-polar transformation, i.e. $\alpha = 1.0$, (b) kernel function with $\alpha = 2.0$, (c) kernel function with $\alpha = 3.0$, and (d) kernel function with $\alpha = 4.0$. The foveated rendering is at 67 FPS while the original is at 31 FPS.

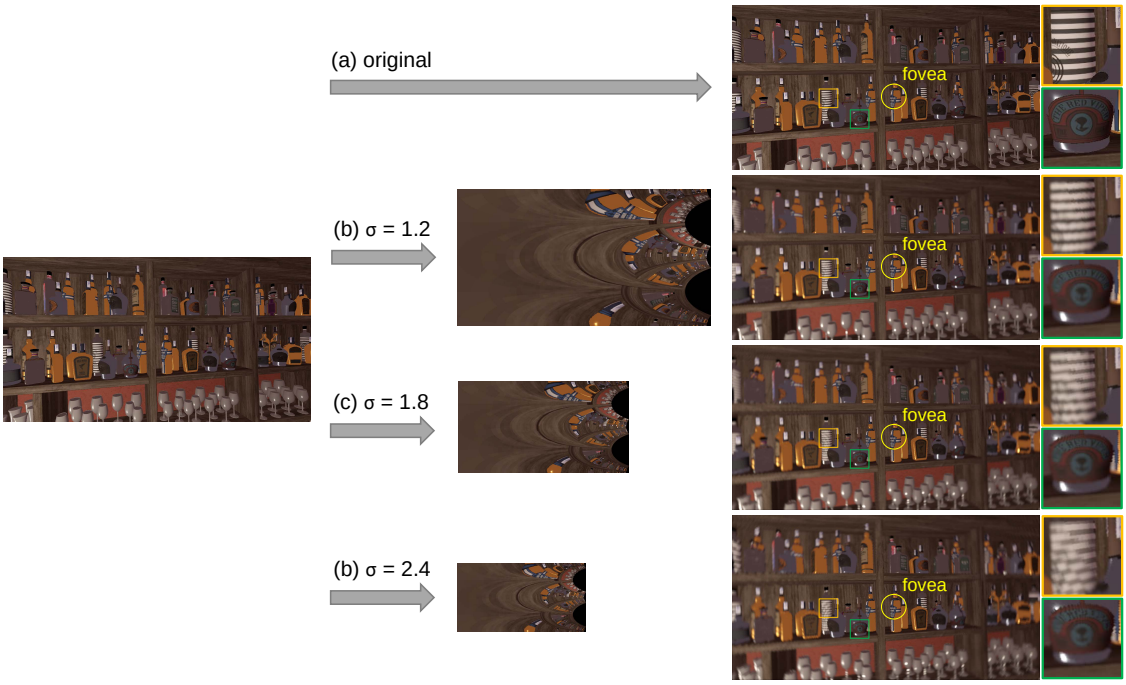


Figure 1.18: Comparison of foveated rendering with varying σ for 2560×1440 resolution. From left to right: original rendering, kernel log-polar rendering, the recovered scene in Cartesian coordinates, and a zoomed-in view of peripheral regions. Here, $\mathbf{K}(x) = x^4$, (a) full-resolution rendered at 31 FPS, (b) $\sigma = 1.2$ at 43 FPS, (c) $\sigma = 1.8$ at 67 FPS, and (d) $\sigma = 2.4$ at 83 FPS.

then the inverse transformation can be formulated as Equation 2.5,

$$\begin{aligned} x'' &= \exp(Au)\cos(Bv) \\ y'' &= \exp(Au)\sin(Bv) \end{aligned} \tag{2.5}$$

To understand how the resolution changes in the log-polar space, consider $r = \|x, y\|_2 = \exp(Au)$.

Now, $\mathrm{d}r$ represents the change in r based on u,

$$\mathrm{d}r = A \cdot \exp(Au)\,\mathrm{d}u. \tag{2.6}$$

Inversely, $\mathscr{D}$ is defined as the number of pixels in the LP-buffer that map to a single pixel on the screen,

$$\mathscr{D} = \frac{\mathrm{d}u}{\mathrm{d}r} = \frac{1}{A} \cdot \exp(-Au). \tag{2.7}$$

Equation 2.7 shows the foveation effect of pixel density decreasing from the fovea to the periphery. In this formulation, it is not easy to systematically alter the density fall-off function and evaluate the perceptual quality of foveated rendering.

I propose a kernel log-polar mapping algorithm that allows us more flexibility to better mimic the fall-off of photo-receptor density of the human visual system,

$$\mathscr{D} = \frac{\exp\left(-wC\sigma \cdot \mathbf{K^{-1}}\left(\frac{u}{w}\right)\right)}{C\sigma \cdot \mathbf{K^{-1'}}\left(\frac{u}{w}\right)}. \tag{2.8}$$

Here, the constant parameter $C = \sqrt{1 + \left(\frac{H}{W}\right)^2}$ represents the ratio between screen diagonal and screen width. $\sigma = \frac{W}{w}$ represents the ratio between the full-resolution screen width and the reduced-resolution LP-buffer width, $\sigma^2 = \frac{W^2}{w^2}$ represents the ratio between the number of pixels in the full-resolution screen and the number of pixels in the reduced-resolution LP-buffer. Larger σ^2 corresponds to more

condensed LP-buffer, which means less calculation in the rendering process. A more condensed LP-buffer also means more foveation and greater peripheral blur.

The kernel function $\mathbf{K}(x)$ can be any monotonically increasing function with $\mathbf{K}(0) = 0$ and $\mathbf{K}(1) = 1$, such as the sum of power functions,

$$\mathbf{K}(x) = \sum_{i=0}^{\infty} \beta_i x^i, \quad \text{where} \sum_{i=0}^{\infty} \beta_i = 1. \tag{2.9}$$

Such kernel functions can be used to adjust the pixel density distribution in the LP-buffer. I use $\mathbf{K}(x) = \sum_{i=0}^{\infty} \beta_i x^i$ in this thesis because the calculation of power functions is fast on modern GPUs. There may be other kernel functions worth trying, such as $\mathbf{K}(x) = sin(x \cdot \frac{\pi}{2})$ and $\mathbf{K}(x) = \frac{e^x - 1}{e - 1}$. For example, for $C = \sqrt{2}$ and $\mathbf{K}(x) = x^\alpha$, the relationship between $\mathscr{D}$ and r under varying σ^2 and α is illustrated in Figure 1.16 [1]. Kernel functions can adjust the pixel density such that the percentage of the peripheral regions in the LP-buffer increases as shown in Figure 1.14 (c), (d), and (e). This makes it possible to increase the peripheral image quality while maintaining the same frame rates. A comparison among different kernel functions is shown in Figure 1.17 with $\sigma = 1.8$ and $C = \sqrt{2}$. The use of the kernel function reduces the artifacts in the zoomed-in peripheral view, improving the peripheral image quality. Meanwhile, as shown in Figure 1.15, when $\alpha \geq 5.0$, the sampling rate of even the foveal region drops, affecting the visual quality of the fovea. A comparison among different σ is shown in Figure 1.18 with fixed $\alpha = 4.0$, $C = \sqrt{2}$.

[1] The figure is the visualization of sampling rate rather than the true sampling map.

2.2 Pass 1: Forward Kernel Log-polar Transformation

2.2.1 Kernel Log-polar Transformation

For each pixel in screen space with coordinates (x, y), foveal point $\mathbf{F}(\mathring{x}, \mathring{y})$ in Cartesian coordinates, I change Equation 2.1 to Equation 2.10,

$$\begin{aligned} u &= \mathbf{K^{-1}}\left(\frac{\log\|x', y'\|_2}{L}\right) \cdot w \\ v &= \left(\arctan\left(\frac{y'}{x'}\right) + \mathbf{1}\left[y' < 0\right] \cdot 2\pi\right) \cdot \frac{h}{2\pi} \end{aligned} \tag{2.10}$$

Here,

$$x' = x - \mathring{x}, \quad y' = y - \mathring{y}. \tag{2.11}$$

$\mathbf{K^{-1}}(\cdot)$ is the inverse of the kernel function, and L is the log of the maximum distance from fovea to one of the four corners of the screen as shown in Equation 2.12,

$$L = \log\left(\max\left(\max\left(l_1, l_2\right), \max\left(l_3, l_4\right)\right)\right). \tag{2.12}$$

Here,

$$\begin{aligned} l_1 &= \|\mathring{x}, \mathring{y}\|_2 \\ l_2 &= \|W - \mathring{x}, H - \mathring{y}\|_2 \\ l_3 &= \|\mathring{x}, H - \mathring{y}\|_2 \\ l_4 &= \|W - \mathring{x}, \mathring{y}\|_2 \end{aligned} \tag{2.13}$$

2.2.2 Lighting

In lighting calculation for traditional deferred shading, mesh positions, normals, depth and material information such as roughness, index of reflection, and normal

ALGORITHM 1: Kernel Log-polar Transformation

Input:

Fovea coordinates in screen space: $(\mathring{x}, \mathring{y})$,

pixel coordinates in screen space: (x, y).

Output:

Pixel coordinates in the log-polar space: (u, v).

acquire fovea coordinates $(\mathring{x}, \mathring{y})$

for $x \in [0, W]$ **do**

  for $y \in [0, H]$ **do**

    $x' = x - \mathring{x}$

    $y' = y - \mathring{y}$

    $u = \mathbf{K^{-1}} \left(\frac{\log \|x', y'\|}{L} \right) \cdot w$

    $v = \left(\arctan \left(\frac{y'}{x'} \right) + \mathbf{1} \left[y' < 0 \right] \cdot 2\pi \right) \cdot \frac{h}{2\pi}$

  end for

end for

maps are fetched from the G-buffer [16, 17]. Instead of obtaining information from the G-buffer with texture coordinates (x, y), in my approach, I sample from the transformed kernel log-polar texture coordinates (u, v). The reduced-resolution of the log-polar (LP) buffer helps in reducing the lighting calculation to only those pixels that matter in the final foveated rendering.

2.2.3 Internal Anti-aliasing

Due to the low-resolution of the LP-buffer, there may be artifacts in the peripheral regions after the inverse transformation. However, I can directly perform denoising in the log-polar space. To reduce artifacts in the peripheral regions, I use a Gaussian filter with a 3×3 kernel for the right part of the texture (corresponding to the peripheral regions) in the LP-buffer. Since the LP-buffer pixels correspond to the adaptive detail of foveated rendering, the Gaussian filtering in the LP-buffer gives us higher-level of anti-aliasing in the peripheral regions.

2.3 Pass II: Inverse Kernel Log-Polar Transformation

Pass II performs the inverse kernel log-polar transformation to Cartesian coordinates, applies anti-aliasing, and renders to screen.

2.3.1 Inverse Kernel Log-polar Mapping Transformation

I can recover the Cartesian coordinates (x'', y''), from the pixel coordinates (u, v) and the fovea coordinates $(\mathring{x}, \mathring{y})$ using Algorithm 2.

2.3.2 Post Anti-aliasing

One of the crucial considerations in foveated rendering is mitigating temporal artifacts due to aliasing in the peripheral, high eccentricity regions. I apply temporal anti-aliasing (TAA) [42] with Halton sampling [55] to the recovered screen-space pixels after the inverse kernel log-polar transformation. I also use Gaussian filtering with different kernel sizes η for different L (as defined in Equation 2.12) in post anti-aliasing. The kernel size η is shown in Equation 2.14, which depends on the normalized distance between the pixel coordinate and the fovea,

$$\eta = 3 + 2 \times \left\lfloor \frac{\frac{\|x', y'\|_2}{e^L} - 0.10}{0.05} \right\rfloor. \tag{2.14}$$

ALGORITHM 2: Kernel Log-polar Inverse Transformation

Input:

Fovea coordinates in screen space: $(\mathring{x}, \mathring{y})$,

pixel coordinates in the log-polar coordinates: (u, v).

Output:

Screen-space coordinates (x'', y'') for pixel coordinates (u, v).

update L with fovea coordinates $(\mathring{x}, \mathring{y})$ with Equation 2.12

let $A = \frac{L}{w}$, $B = \frac{2\pi}{h}$

for $u \in [0, w]$ **do**

  for $v \in [0, h]$ **do**

    $x'' = \exp(A \cdot \mathbf{K}(u)) \cdot \cos(Bv) + \mathring{x}$

    $y'' = \exp(A \cdot \mathbf{K}(u)) \cdot \sin(Bv) + \mathring{y}$

  end for

end for

Chapter 3: User Study

I have carried out user studies to empirically establish the most suitable foveation parameter values for σ and α that result in visually acceptable foveated rendering. To systematically investigate this, I conducted a pilot study to examine a broad range of the two parameters, σ^2 and α. I used the results and my experience with the pilot study to fine tune the protocol and ranges of σ^2 and α for the final user study.

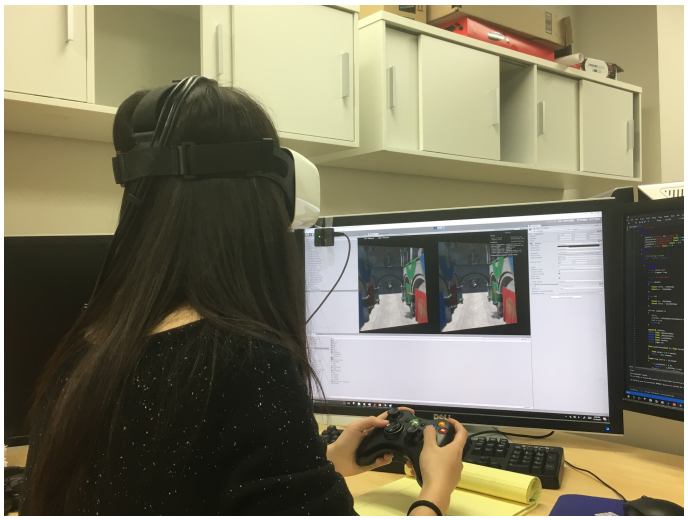
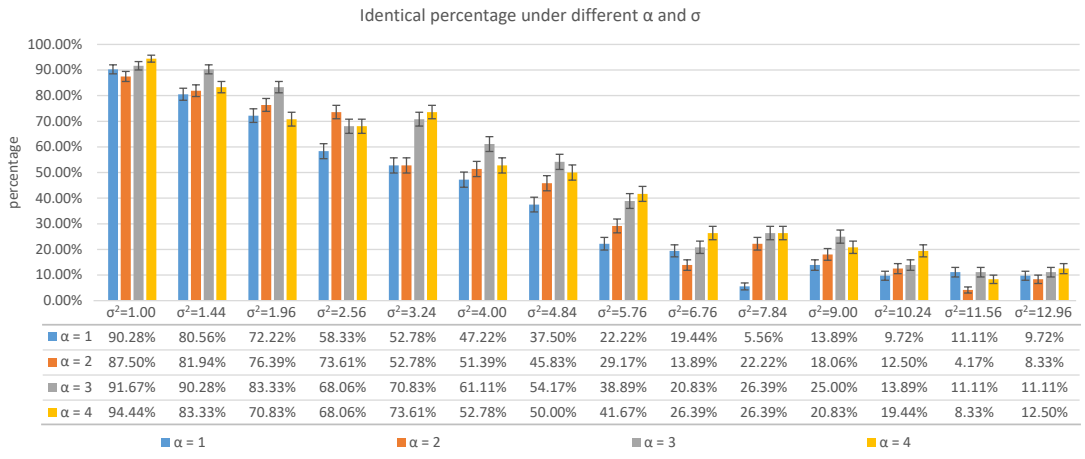


Figure 2.1: User study setup.



	σ²=1.00	σ²=1.44	σ²=1.96	σ²=2.56	σ²=3.24	σ²=4.00	σ²=4.84	σ²=5.76	σ²=6.76	σ²=7.84	σ²=9.00	σ²=10.24	σ²=11.56	σ²=12.96
α = 1	90.28%	80.56%	72.22%	58.33%	52.78%	47.22%	37.50%	22.22%	19.44%	5.56%	13.89%	9.72%	11.11%	9.72%
α = 2	87.50%	81.94%	76.39%	73.61%	52.78%	51.39%	45.83%	29.17%	13.89%	22.22%	18.06%	12.50%	4.17%	8.33%
α = 3	91.67%	90.28%	83.33%	68.06%	70.83%	61.11%	54.17%	38.89%	20.83%	26.39%	25.00%	13.89%	11.11%	11.11%
α = 4	94.44%	83.33%	70.83%	68.06%	73.61%	52.78%	50.00%	41.67%	26.39%	26.39%	20.83%	19.44%	8.33%	12.50%

Figure 2.2: The percentage of times that the participants considered the foveated rendering and the full-resolution rendering to be the same for varying σ^2 and α in pilot user study with 24 participants.

3.1 Apparatus

My user study apparatus, shown in Figure 2.1, consists of an *Alienware* laptop with an *NVIDIA GeForce GTX 1080*, a *FOVE* head-mounted display, and an *XBOX* controller. The *FOVE* display has a 100° field of view, a resolution of 2560×1440, and a 120 Hz infrared eye-tracking system with a precision of 1° and a latency of 14 ms, the system latency meets the eye tracking delay requirement of 50 ms - 70 ms.

3.2 Pilot Study

3.2.1 Procedure

The session for each participant lasted between $35 - 50$ minutes and involved four stages: introduction, calibration, training, and testing.

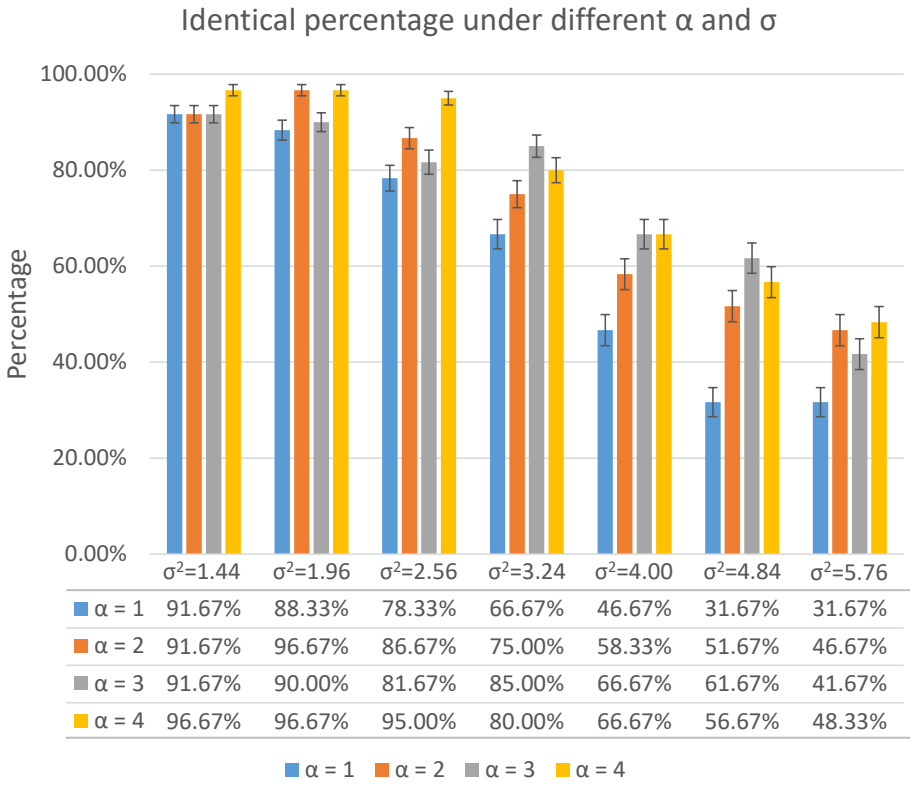


Figure 2.3: The percentage of times that participants considered the foveated rendering and the full-resolution rendering to be identical for different σ^2 and α in the final user study with 18 participants.

In the introduction stage, I showed participants the *FOVE* headset, the eye trackers, and the *XBOX* controllers and discussed how to use them. I did not provide any information about the research or the algorithm to avoid biasing the participants towards any rendering.

After the participant comfortably wore the HMD, I moved forward to the calibration stage, where I ran a one-minute eye-tracking calibration program provided by the *FOVE* software development kit.

In the training stage, I presented the participants with 20 trials with different

combinations of σ^2 and α, to ensure that they are familiar with the HMD and the controller.

Trials in the training and testing stages were identical. In each trial of the two-alternative forced choice test, I presented participants with a pair of rendered scenes, each for 2 seconds and separated by a black-screen interval of 0.75 seconds. One scene uses full-resolution rendering, and the other uses KFR with different parameters σ^2 and α. In each trial, I presented the KFR scene and the full-resolution scene in a random order. The participant indicated whether the two images look the same by pressing a button on the *XBOX* controller. I instructed the participants to maintain their gaze at the center of the screen, even though the foveated renderer can use eye-tracking to update the foveated image.

The testing stage had three sessions, each with 56 trials. The LP-buffer resolution reduction parameter σ ranges from 1.0 to 3.6 with step size 0.2 (σ^2 ranges from 1.00 to 12.96), and the kernel sampling distribution parameter α ranges from 1 to 4 with step size 1. I rendered scenes from the *Sponza* and *Amazon Lumberyard Bistro* datasets for different sessions. I allowed the participants to have some rest between different sessions.

3.2.2 Participants

In the pilot study, I recruited 24 participants via campus email lists and flyers. All participants are at least 18 years old with normal or corrected-to-normal vision (with contact lenses). The participants are collected using campus flyers and emails.

3.2.3 Results and Analysis

I define P_I as the percentage of the trials for which participants reported the two images shown in a trial to be the same. The results of P_I are shown in Figure 2.2. First, I find that P_I is inversely related to σ^2. With increase in σ^2, the LP-buffer gets smaller, thus reducing the overall sampling rate in foveated rendering. Second, I notice that with the increase of α, P_I significantly increases for σ ranging from 1.2 to 2.8 (σ^2 ranging from 1.44 to 7.84). This shows that for the same σ, the perception of the quality of foveated rendering increases by the use of α for kernel functions. For $\sigma = 1.0$ and $\sigma > 2.8$ the improvement is not significant. The reason is that for $\sigma = 1.0$, the foveated renderings for $\alpha = 1$ and $\alpha > 1$ are both clear, there is little space for improvement. Similarly, for $\sigma > 2.8$, even if the quality improves by applying kernel functions, it still looks blurry for both images. Therefore, the participants choose "different" for these comparisons. Third, some participants reported that the length of the study led to visual fatigue and that they were not sure about some of their responses.

Using the above observations, I modified the final user study to be shorter and more focused.

First, to reduce the total time that participants are in the HMD, I used the fact that most participants found foveated renderings different from the full-resolution rendering for $\sigma > 2.4$ ($\sigma^2 > 5.76$). Since the goal is to accelerate rendering while maintaining perceptually similar quality, I reduced the range of σ to be between 1.2 to 2.4 (σ^2 between 1.44 to 5.76) in the final user study.

Second, I observed that the participants quickly came up to speed within a couple of trials in the training session. I therefore reduced the number of trials in the training session from 20 to 5. This also allowed us to shorten the user study duration and maintain a high level of visual attentiveness of the participants. Third, some of the participants reported that the rendering time of 2 seconds was too short. To address this I increased the time of each rendering to 2.5 seconds in the final study. Fourth, to continually check for the visual attentiveness of the participants, I modified the final user study by randomly inserting 30% of the trials to be "validation trials" that had identical full-resolution renderings for both choices. If the participant declared these validation renderings to be different, I would ask the participant to stop, get some rest, and then continue. After making these changes, the total time participants spent in the HMD was reduced from around 25 minutes in the pilot study to around 15 minutes in the final study.

3.3 Final User Study

3.3.1 Procedure

The introduction and calibration stages are the same as the pilot user study. The training session includes five trials with different parameters. Each testing session involves 28 trials with multiple parameter combinations (parameter σ ranging from 1.2 to 2.4 with the step size 0.2 (σ^2 ranging from 1.44 to 5.76); and kernel parameter α ranging from 1 to 4 with the step size 1) as well as additional "validation trials". Order of the parameters is fully counterbalanced. The participants are asked

to rest after each session or if they do not pass a "validation trial". I also changed the rendering-display time to 2.5 seconds.

3.3.2 Participants

I recruited 18 participants via campus email lists and flyers. All participants were at least 18 years old with normal or corrected-to-normal vision (with contact lenses). The participants are collected using campus flyers and emails.

3.3.3 Results and Analysis

I report the percentage P_I and the corresponding standard error in Figure 2.3. I make the null hypothesis (H_0) that the foveated rendering results with the four kernel functions are equally effective. As shown in Table 1, with a Cochran's Q test [56, 57], I have found that there exists a significant difference across the multiple α for $\sigma = 1.6, 1.8, 2.2$ ($\sigma^2 = 2.56, 3.24, 4.84$) with $\chi^2(3) = 7.81, p < 0.05$. The results with very small $\sigma = 1.2, 1.4$ ($\sigma^2 = 1.44, 1.96$) and very large $\sigma = 2.4$ ($\sigma^2 = 5.76$) are not significantly different, which are reasonable. For small σ^2, the rendering result without kernel function is clear enough, so there is little room for improvement. For large σ^2, both the rendering results with and without kernel function are blurry for the users.

To achieve visually acceptable results for foveated rendering, I use a threshold of 80% responses considering foveated rendering to be visually indistinguishable from full-resolution rendering. To achieve the highest rendering acceleration, I look for

Table 3.1: Cochrans Q values at different σ^2.

σ^2	1.44	1.96	2.56	3.24	4.00	4.84	5.76
Cochrans Q value	1.72	5.79	8.20	8.25	7.49	14.27	5.48
p value	0.631	0.122	0.042	0.041	0.058	0.002	0.139

the highest σ that met this threshold. I therefore choose $\sigma = 1.8$ ($\sigma^2 = 3.24$) and $\alpha = 4$ as the desired parameters for the interactive rendering evaluation.

Chapter 4: Rendering Acceleration

I implemented kernel foveated rendering on *NVIDIA GeForce GTX 1080*, by using the deferred shading pipeline of the *Falcor* engine [58]. I report results of my rendering acceleration for resolutions of 1920×1080, 2560×1440, and 3840×2160. Using the results from the final user study, I selected the LP-buffer parameter $\sigma = 1.8$, and kernel parameter $\alpha = 4$ for the evaluations below.

4.0.1 3D Textured Meshes

I use the *Amazon Lumberyard Bistro* [59] scene with physically-based shading, reflection, refraction, and shadows to simulate the complex shading effects as shown in Figure 4.1. I choose *Amazon Lumberyard Bistro* because this scene has complex triangular meshes, rendering textures and compact lighting effect. The comparison of the break-down of rendering time between KFR and the ground truth of deferred shading is shown in Table 2. I observed that the frame rate increases for all resolutions as shown in Table 3, with a speedup of $2.0X - 2.8X$.

4.0.2 Ray-casting Rendering

Ray casting uses raysurface intersection tests to solve a variety of problems in computer graphics and computational geometry. It can also be used for creating scenes. Rendering of high-resolution ray cast scenes can be an extremely time-consuming process. I used the complex ray-casting scene with 16 different primitives by Íñigo Quílez to evaluate the acceleration of kernel foveated rendering. Figure 4.2 shows a comparison of the foveated scene and the ground truth. The frame rate increases for all resolutions as shown in Table 3, with a speedup of $2.9X - 3.2X$.

Figure 4.1: Comparison of (a) full-resolution rendering and (b) foveated rendering for 3D meshes involving a geometry pass with $1,020,895$ triangles as well as multiple G-buffers at 2560×1440 resolution.

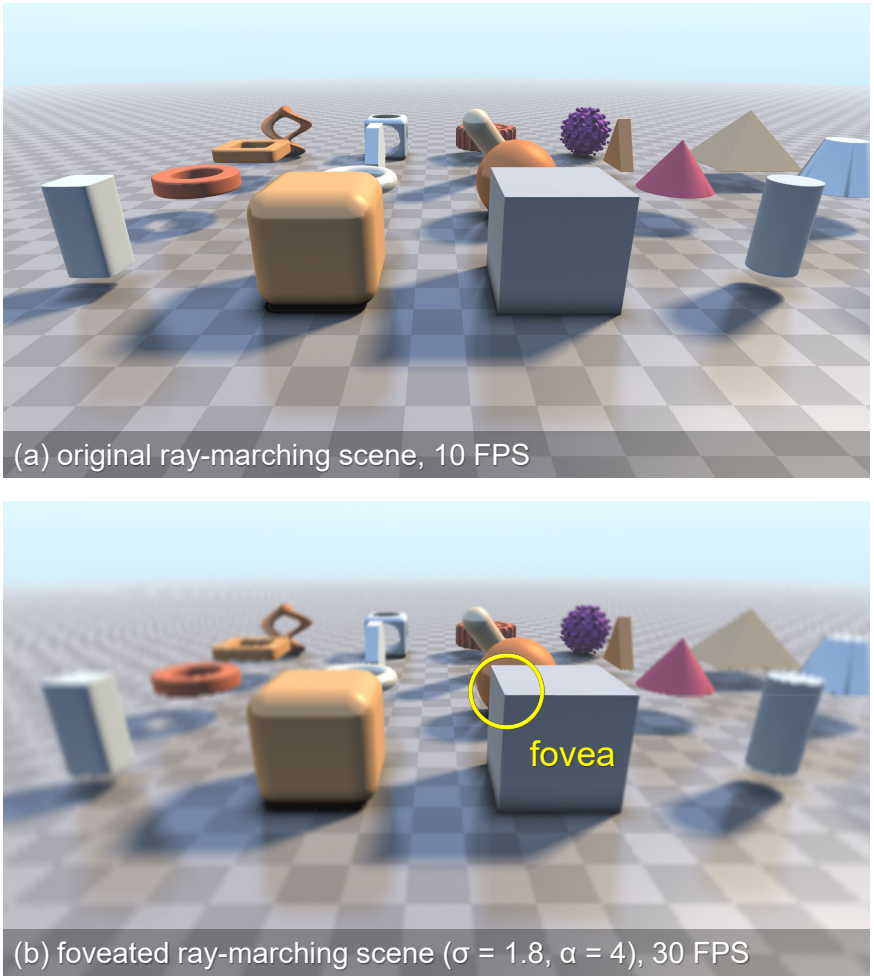


Figure 4.2: Comparison of (a) full-resolution rendering and (b) foveated ray-marching scene with 16 samples per pixel rendered at 2560×1440.

Table 4.1: Timing comparison between the ground truth and KFR for one frame. The resolution is 1920×1080.

Procedure	Timing (ms)	
	Ground Truth	KFR
Depth Pass	0.327	0.309
Shadow Pass	3.744	4.503
Defer Pass	2.985	3.034
SkyBox	0.039	0.039
Shading / Pass1	22.043	6.674
Pass2	N/A	0.090
Total	29.138	14.649
Total GPU Time	31.892	17.052

Table 4.2: Frame rate and speedup comparison for kernel foveated rendering at different resolutions with $\sigma = 1.8$, $\alpha = 4.0$.

Scene	3D Textured Meshes			Ray Casting		
Resolution	Ground Truth	Foveated	Speedup	Ground Truth	Foveated	Speedup
1920×1080	55 FPS	110 FPS	2.0X	20 FPS	57 FPS	2.9X
2560×1440	31 FPS	67 FPS	2.2X	10 FPS	30 FPS	3.0X
3840×2160	8 FPS	23 FPS	2.8X	5 FPS	16 FPS	3.2X

Chapter 5: Discussion and Limitations

5.1 Discussion

Here I compare the Kernel Foveated Rendering (KFR) pipeline with selected prior art, including: *Foveated 3D Graphics* (F3D) [3], *Multi-rate Shading* (MRS) [8], *Coarse Pixel Shading* (CPS) [4], and *Adaptive Image-Space sampling* (AIS) [10].

As mentioned by [10], both MRS and CPS pipelines require adaptive shading features which are not yet commonly available on commodity GPUs and so they rely on software simulator implementations. In contrast, F3D, AIS, and the KFR pipelines can be easily mapped onto the current generation of GPUs.

The F3D pipeline has achieved impressive speedups of $10X-15X$ in the informal user study, and a factor of $4.8X - 5.7X$ in the formal user study. Nevertheless, the F3D approach uses three discrete layers, while the KFR parameterizes the distribution of samples continuously in the log-polar domain. F3D relies on specifically designed anti-aliasing algorithms including jitter sampling and temporal reprojection, thus limiting F3D to simpler material models and less complex geometry [10]. In contrast, KFR could easily be coupled with the state-of-the-art screen-space anti-aliasing techniques, such as TAA [42] and recent G-buffer anti-aliasing strategies [43].

Both the AIS and KFR pipelines mimic the continuously changing distribution

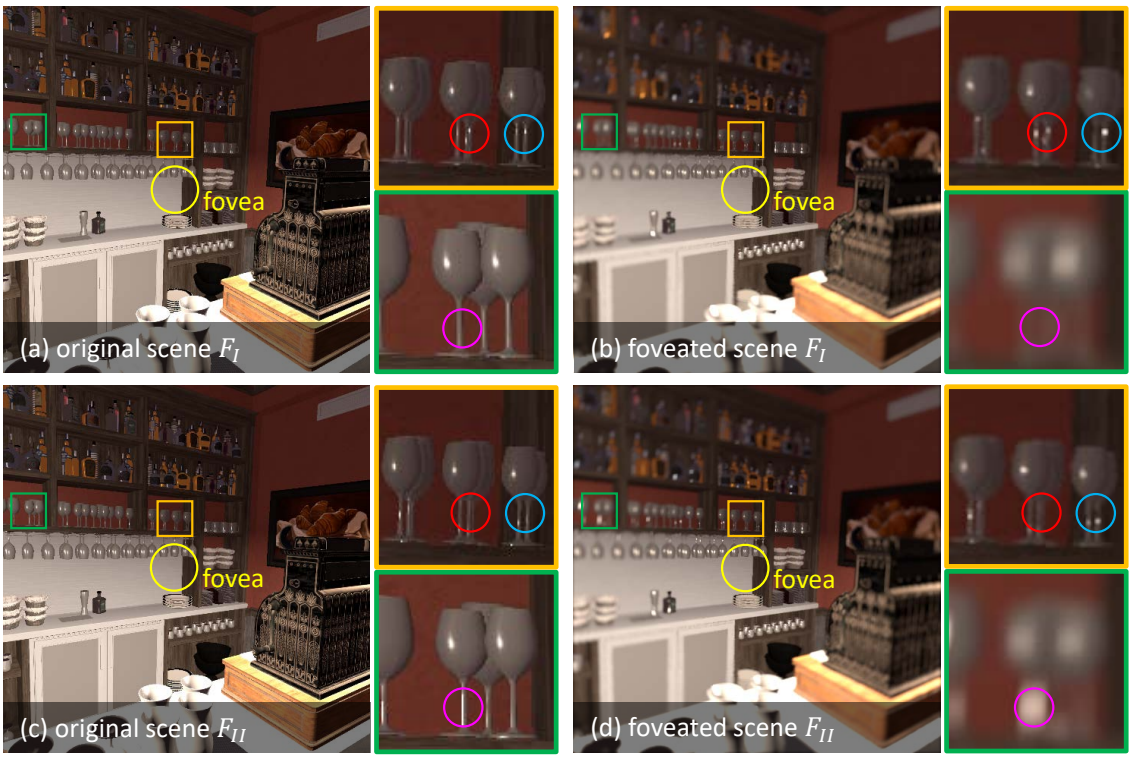


Figure 4.3: Temporal flickering issue. The original scene and the foveated scene of two consecutive frames (F_I and F_{II}). In F_I, the specular reflection in the original scene as shown in the red and blue circles in the zoomed-in view of (a) are amplified in the foveated scene as shown in the zoomed-in view of (b). In the next frame F_{II}, the specular reflection in the original scene as shown in the pink circle in the zoomed-in view of (c) is amplified in the foveated scene as shown in the zoomed-in view of (d).

of photo-receptors in the retina. Nonetheless, there are three significant differences: complexity and evaluation of the perceptual model, interpolation, and speedup. First, AIS uses four parameters from [60] to approximately model the linear degradation behavior of acuity with 30° eccentricity. However, these parameters have not yet been evaluated on how they affect foveation and perception in HMDs or beyond 30° eccentricity. In contrast, KFR uses only two parameters: the reduced-resolution LP-buffer parameter σ and the kernel parameter α in conjunction with a simple coordinate transformation. KFR has established desirable values of α and σ through user studies in head-mounted displays. Second, AIS relies on the pull-push interpolation [41] to fill the pixels that are missed due to variable sampling of silhouette features and object saliency. In comparison, KFR uses the built-in GPU-driven mipmap interpolation which reduces the additional interpolation cost. However, it is worth investigating how incorporating object saliency [45,61] could further improve the KFR pipeline. It is a challenge to compare multiple foveated approaches given the varying hardware and perceptual quality of the results. One possibility is to compare the speedups as percentages of rendering time reduction with certain reduction sampling rate. By rendering with 59% of the total amount of shaded pixels, AIS reports an overall rendering time reduction of 25.4%, while KFR achieves 29.9% reduction on average. Like AIS, KFR speedup also depends on the amount of time spent in shading; the greater the shader computations, the higher will be the KFR speedup.

5.2 Limitations

Even though I have devised an efficient and effective foveated rendering pipeline, my system is not without some limitations.

5.2.1 Foveation Parameters

As discussed in Hsu *et al.* [62], the perceived quality of foveated rendering systems is highly dependent on the user and the scene. As the initial step towards kernel foveated rendering for 3D graphics, the user study in this thesis is only designed for selected static scenes. The foveation parameters may vary in dynamic scenes. Exploring the relationship between user demographic (*e.g.*, pupil size, contrast sensitivity, vision condition, and diopter) and display-dependent parameters of KFR is a potential future direction.

5.2.2 Temporal Flickering

In the post-processing stage, I have applied TAA to tackle the temporal flickering problem. However, in fly-through of the scene with glossy objects, I notice that view-dependent specular reflection changes before and after applying KFR. As shown in Figure 4.3, foveation amplifies the specular reflection regions, and makes the specular highlights flicker more.

5.2.3 Other Mapping Algorithms and Kernel Functions

KFR makes intuitive sense as the log-polar mapping has an initial resolution proportional to e^{-r}, and the kernel functions can fine tune this mapping. My choice of kernel functions is not unique; other mapping algorithms with different kernel functions could provide a better mapping to the human vision system.

Chapter 6: Conclusion and Future Work

In this thesis, I have presented the kernel log-polar mapping model, user studies for finding the best parameters, as well as a GPU-based implementation and quantitative evaluation of the kernel foveated rendering pipeline. My research in this thesis has been published in a peer-reviewed journal [63] and I presented it at *ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* held during May 15 - 18, 2018 in Montreal, Canada.

First, I systematically vary the sampling rate and sampling distribution to better mimic the distribution of photo-receptors in the human vision system by embedding the polynomial kernel functions in the classic log-polar mapping. Second, I have carried out user studies, including a pilot study and a final one, in virtual reality HMD with integrated eye tracking to determine the best parameters for KFR. Finally, I evaluate the KFR pipeline through a quantitative evaluation with physically-based deferred-rendering scenes and ray-marching scenes. I have observed that the algorithm achieves a speedup of $3.2X$ for the procedural rendering scenes with ray-marched primitives and $2.8X$ for the physically-based deferred-rendering scenes.

With high frame rates, the KFR pipeline allows rendering more complex shaders

(*e.g.,* real-time global illumination and physically-based rendering [64]) in real time, thus bringing higher power efficiency and better user experience for 3D games and other interactive visual computing applications. The work presented in this thesis is the first step towards kernel foveated rendering.

In future, I would like to apply the algorithm for interactively testing high-fidelity path-tracing algorithms with the state-of-art noise reduction algorithms [43, 65, 66]. Considering that current consumer HMDs have lens distortions [67], I also plan to explore warping from LP-buffer to "lens undistorted" image directly in order to provide substantial performance improvements.

Bibliography

[1] Christine A Curcio, Kenneth R Sloan, Robert E Kalina, and Anita E Hendrickson. Human photoreceptor topography. *Journal of comparative neurology*, 292(4):497–523, 1990.

[2] Christine A Curcio and Kimberly A Allen. Topography of ganglion cells in human retina. *Journal of comparative Neurology*, 300(1):5–25, 1990.

[3] 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.

[4] K. Vaidyanathan, M. Salvi, R. Toth, T. Foley, T. Akenine-Möller, J. Nilsson, J. Munkberg, J. Hasselgren, M. Sugihara, P. Clarberg, T. Janczak, and A. Lefohn. Coarse pixel shading. In *Proceedings of High Performance Graphics*, HPG '14, pages 9–18, Aire-la-Ville, Switzerland, Switzerland, 2014. Eurographics Association.

[5] 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.

[6] Anjul Patney, Joohwan Kim, Marco Salvi, Anton Kaplanyan, Chris Wyman, Nir Benty, Aaron Lefohn, and David Luebke. Perceptually-based foveated virtual reality. In *ACM SIGGRAPH 2016 Emerging Technologies*, SIGGRAPH '16, pages 17:1–17:2, New York, NY, USA, 2016. ACM.

[7] Petrik Clarberg, Robert Toth, Jon Hasselgren, Jim Nilsson, and Tomas Akenine-Möller. Amfs: adaptive multi-frequency shading for future graphics processors. *ACM Trans. Graph.*, 33(4):141:1–141:12, July 2014.

[8] Yong He, Yan Gu, and Kayvon Fatahalian. Extending the graphics pipeline with adaptive, multi-rate shading. *ACM Trans. Graph.*, 33(4):142:1–142:12, July 2014.

[9] Nicholas T. Swafford, José A. Iglesias-Guitian, Charalampos Koniaris, Bochang Moon, Darren Cosker, and Kenny Mitchell. User, metric, and computational evaluation of foveated rendering methods. In *Proceedings of the ACM Symposium on Applied Perception*, SAP '16, pages 7–14, New York, NY, USA, 2016. ACM.

[10] Michael Stengel, Steve Grogorick, Martin Eisemann, and Marcus Magnor. Adaptive image-space sampling for gaze-contingent real-time rendering. *Computer Graphics Forum*, 35(4):129–139, 2016.

[11] Qi Sun, Fu-Chung Huang, Joohwan Kim, Li-Yi Wei, David Luebke, and Arie Kaufman. Perceptually-guided foveation for light field displays. *ACM Trans. Graph.*, 36(6):192:1–192:13, November 2017.

[12] Cheuk Yiu Ip, M. Adil Yalçin, David Luebke, and Amitabh Varshney. Pixelpie: maximal poisson-disk sampling with rasterization. In *Proceedings of the 5th High-Performance Graphics Conference*, HPG '13, pages 17–26, New York, NY, USA, 2013. ACM.

[13] Marc Levoy and Ross Whitaker. Gaze-directed volume rendering. In *Proceedings of the 1990 Symposium on Interactive 3D Graphics*, I3D '90, pages 217–223, New York, NY, USA, 1990. ACM.

[14] H. Araujo and J. M. Dias. An introduction to the log-polar mapping [image sampling]. In *Proceedings II Workshop on Cybernetic Vision*, pages 139–144, Dec 1996.

[15] Marco Antonelli, Francisco D. Igual, Francisco Ramos, and V. Javier Traver. Speeding up the log-polar transform with inexpensive parallel hardware: graphics units and multi-core architectures. *J. Real-Time Image Process.*, 10(3):533–550, September 2015.

[16] Shawn Hargreaves and Mark Harris. Deferred shading. In *Game Developers Conference*, volume 2, page 31, 2004.

[17] Jerome F Duluk Jr, Richard E Hessel, Vaughn T Arnold, Jack Benkual, Joseph P Bratt, George Cuan, Stephen L Dodgen, Emerson S Fang, Zhaoyu Gong, Thomas Y Ho, et al. Deferred shading graphics pipeline processor having advanced features, April 6 2004. US Patent 6,717,576.

[18] Hans Strasburger, Ingo Rentschler, and Martin Jüttner. Peripheral vision and pattern recognition: a review. *Journal of vision*, 11(5):13–13, 2011.

[19] Xuetong Sun and Amitabh Varshney. Investigating Perception Time in the Far Peripheral Vision for Virtual and Augmented Reality. In *ACM Symposium on Applied Perception (SAP)*, Perception. ACM, Aug 2018.

[20] E. L. Schwartz. Anatomical and physiological correlates of visual computation from striate to infero-temporal cortex. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-14(2):257–271, March 1984.

[21] Philip Kortum and Wilson S Geisler. Implementation of a foveated image coding system for image bandwidth reduction. In *Electronic Imaging: Science & Technology*, pages 350–360. International Society for Optics and Photonics, 1996.

[22] Peter J Burt. Smart sensing within a pyramid vision machine. *Proceedings of the IEEE*, 76(8):1006–1015, 1988.

[23] Jerome M Shapiro. Embedded image coding using zerotrees of wavelet coefficients. *IEEE Transactions on Signal Processing*, 41(12):3445–3462, 1993.

[24] A. Said and W. A. Pearlman. A new, fast, and efficient image codec based on set partitioning in hierarchical trees. *IEEE Transactions on Circuits and Systems for Video Technology*, 6(3):243–250, Jun 1996.

[25] Ee-Chien Chang, Stéphane Mallat, and Chee Yap. Wavelet foveation. *Applied and Computational Harmonic Analysis*, 9(3):312–335, 2000.

[26] Zhou Wang and Alan C Bovik. Embedded foveation image coding. *IEEE Transactions on Image Processing*, 10(10):1397–1410, 2001.

[27] C. Papadopoulos and A. E. Kaufman. Acuity-driven gigapixel visualization. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2886–2895, Dec 2013.

[28] T. H. Reeves and J. A. Robinson. Adaptive foveation of mpeg video. In *Proceedings of the Fourth ACM International Conference on Multimedia*, MULTIMEDIA '96, pages 231–241, New York, NY, USA, 1996. ACM.

[29] Lee and Sanghoon. *Foveated video compression and visual communications over wireless and wireline networks*. PhD thesis, Dept. of ECE, University of Texas at Austin, 2000.

[30] Zhou Wang and Alan C Bovik. Foveated image and video coding. In *Digitial Video Image Quality and Perceptual Coding*, pages 1–28. 2005.

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

[32] Sanghoon Lee, M. S. Pattichis, and A. C. Bovik. Foveated video quality assessment. *IEEE Transactions on Multimedia*, 4(1):129–132, Mar 2002.

[33] H. R. Sheikh, S. Liu, Z. Wang, and A. C. Bovik. Foveated multipoint videoconferencing at low bit rates. In *2002 IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 2, pages II–2069–II–2072, May 2002.

[34] Hamid R. Sheikh, Brian L. Evans, and Alan C. Bovik. Real-time foveation techniques for low bit rate video coding. *Real-Time Imaging*, 9(1):27–40, February 2003.

[35] P. Lungaro, R. Sjberg, 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.

[36] M. Weier, M. Stengel, T. Roth, P. Didyk, E. Eisemann, M. Eisemann, S. Grogorick, A. Hinkenjann, E. Kruijff, M. Magnor, K. Myszkowski, and P. Slusallek. Perception-driven accelerated rendering. *Comput. Graph. Forum*, 36(2):611–643, May 2017.

[37] Toshikazu Ohshima, Hiroyuki Yamamoto, and Hideyuki Tamura. Gaze-directed adaptive rendering for interacting with virtual space. In *Proceedings of the 1996 Virtual Reality Annual International Symposium (VRAIS 96)*, VRAIS '96, pages 103–110, 267, Washington, DC, USA, 1996. IEEE Computer Society.

[38] Hugues Hoppe. Smooth view-dependent level-of-detail control and its application to terrain rendering. In *Proceedings of the Conference on Visualization '98*, VIS '98, pages 35–42, Los Alamitos, CA, USA, 1998. IEEE Computer Society Press.

[39] L. Hu, P. V. Sander, and H. Hoppe. Parallel view-dependent level-of-detail control. *IEEE Transactions on Visualization and Computer Graphics*, 16(5):718–728, Sept 2010.

[40] Jonathan Ragan-Kelley, Jaakko Lehtinen, Jiawen Chen, Michael Doggett, and Frédo Durand. Decoupled sampling for graphics pipelines. *ACM Trans. Graph.*, 30(3):17:1–17:17, May 2011.

[41] 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, pages 43–54, New York, NY, USA, 1996. ACM.

[42] B Karis. High-quality temporal supersampling. *Advances in Real-Time Rendering in Games, SIGGRAPH Courses*, 1:1–55, 2014.

[43] Cyril Crassin, Morgan McGuire, Kayvon Fatahalian, and Aaron Lefohn. Aggregate g-buffer anti-aliasing. In *Proceedings of the 19th Symposium on Interactive 3D Graphics and Games*, I3D '15, pages 109–119, New York, NY, USA, 2015. ACM.

[44] Matthäus G. Chajdas, Morgan McGuire, and David Luebke. Subpixel reconstruction antialiasing for deferred shading. In *Symposium on Interactive 3D Graphics and Games*, I3D '11, pages 15–22 PAGE@7, New York, NY, USA, 2011. ACM.

[45] Youngmin Kim, Amitabh Varshney, David Jacobs, and Francois Guimbretere. Mesh saliency and human eye fixations. *ACM Transactions on Applied Perception*, 7(2):1 – 13, 2010.

[46] Ruofei Du and Amitabh Varshney. Social Street View: Blending Immersive Street Views with Geo-Tagged Social Media. In *Proceedings of the 21st International Conference on Web3D Technology*, Web3D, pages 77–85. ACM, July 2016.

[47] Ruofei Du, Sujal Bista, and Amitabh Varshney. Video Fields: Fusing Multiple Surveillance Videos into a Dynamic Virtual Environment. In *Proceedings of the 21st International Conference on Web3D Technology*, Web3D, pages 165–172. ACM, July 2016.

[48] Ruofei Du, Ming Chuang, Wayne Chang, Hugues Hoppe, and Amitabh Varshney. Montage4D: Interactive Seamless Fusion of Multiview Video Textures . In *Proceedings of ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (I3D)*, I3D, pages 124–133. ACM, May 2018.

[49] Sujal Bista, Jiachen Zhuo, Rao P. Gullapalli, and Amitabh Varshney. Visualization of Brain Microstructure through Spherical Harmonics Illumination of High Fidelity Spatio-Angular Fields. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):2516–2525, Dec 2014.

[50] Cheuk Yiu Ip, Amitabh Varshney, and Joseph JaJa. Hierarchical Exploration of Volumes Using Multilevel Segmentation of the Intensity-Gradient Histograms. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2355–2363, 2012.

[51] Hsueh-Chien Cheng, Antonio Cardone, Somay Jain, Eric Krokos, Kedar Narayan, Sriram Subramaniam, and Amitabh Varshney. Deep-learning-assisted Volume Visualization. *IEEE Transactions on Visualization and Computer Graphics*, PP(99):1–14, January 2018.

[52] Hsueh-Chien Cheng, Antonio Cardone, and Amitabh Varshney. Interactive Exploration of Microstructural Features in Gigapixel Microscopy Images. In *Proceedings of 2017 IEEE International Conference on Image Processing*, ICIP. IEEE, September 2017.

[53] Cheuk Yiu Ip and Amitabh Varshney. Saliency-Assisted Navigation of Very Large Landscape Images. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):1737 – 1746, 2011.

[54] Rachel Albert, Anjul Patney, David Luebke, and Joohwan Kim. Latency requirements for foveated rendering in virtual reality. *ACM Trans. Appl. Percept.*, 14(4):25:1–25:13, September 2017.

[55] T Pengo, A Muñoz-Barrutia, and C Ortiz-de solórzano. Halton sampling for autofocus. *Journal of Microscopy*, 235(1):50–58, 2009.

[56] Kashinath D Patil. Cochran's q test: Exact distribution. *Journal of the American Statistical Association*, 70(349):186–189, 1975.

[57] Margarita Vinnikov and Robert S Allison. Gaze-contingent depth of field in realistic scenes: The user experience. In *Proceedings of the Symposium on Eye Tracking Research and Applications*, pages 119–126. ACM, 2014.

[58] Nir Benty, Kai-Hwa Yao, Tim Foley, Anton S. Kaplanyan, Conor Lavelle, Chris Wyman, and Ashwin Vijay. The Falcor rendering framework, 07 2017.

[59] Amazon Lumberyard. Amazon lumberyard bistro, open research content archive (orca), July 2017.

[60] Frank W Weymouth. Visual sensory units and the minimal angle of resolution. *American Journal of Ophthalmology*, 46(1):102–113, 1958.

[61] Chang Ha Lee, Amitabh Varshney, and David Jacobs. Mesh saliency. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2005)*, 24(3):659 – 666, August 2005.

[62] Chih-Fan Hsu, Anthony Chen, Cheng-Hsin Hsu, Chun-Ying Huang, Chin-Laung Lei, and Kuan-Ta Chen. Is foveated rendering perceivable in virtual reality?: exploring the efficiency and consistency of quality assessment methods. In *Proceedings of the 2017 ACM on Multimedia Conference*, MM '17, pages 55–63, New York, NY, USA, 2017. ACM.

[63] 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.

[64] Matt Pharr and Greg Humphreys. *Physically Based rendering, second edition: from theory to implementation*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2nd edition, 2010.

[65] Kai Selgrad, Christian Reintges, Dominik Penk, Pascal Wagner, and Marc Stamminger. Real-time depth of field using multi-layer filtering. In *Proceedings of the 19th Symposium on Interactive 3D Graphics and Games*, i3D '15, pages 121–127, New York, NY, USA, 2015. ACM.

[66] Bochang Moon, Jose A. Iglesias-Guitian, Steven McDonagh, and Kenny Mitchell. Noise reduction on G-buffers for Monte Carlo filtering. *Computer Graphics Forum*, 36(8):600–612, 2017.

[67] D. Pohl, X. Zhang, and A. Bulling. Combining eye tracking with optimizations for lens astigmatism in modern wide-angle hmds. In *2016 IEEE Virtual Reality (VR)*, pages 269–270, March 2016.