Kernel Foveated Rendering

XIAOXU MENG, University of Maryland, College Park
RUOFEI DU, University of Maryland, College Park
MATTHIAS ZWICKER, University of Maryland, College Park
AMITABH VARSHNEY, University of Maryland, College Park

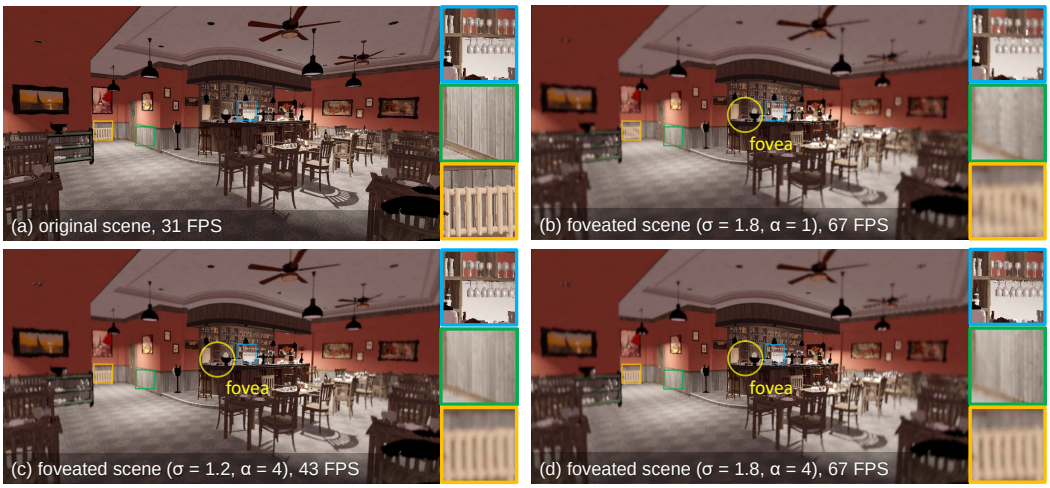


Fig. 1. Results of kernel foveated rendering (KFR): (a) original full-resolution scene rendered at 31 FPS, (b) foveated rendering with $\sigma = 1.8$, $K(x) = x$, rendered at 67 FPS, (c) foveated rendering with $\sigma = 1.2$, $K(x) = x^4$, rendered at 43 FPS, (d) foveated rendering with $\sigma = 1.8$, $K(x) = x^4$, rendered at 67 FPS.

Foveated rendering coupled with eye-tracking has the potential to dramatically accelerate interactive 3D graphics with minimal loss of perceptual detail. In this paper, we parameterize foveated rendering by embedding polynomial kernel functions in the classic log-polar mapping. Our GPU-driven technique uses closed-form, parameterized foveation that mimics the distribution of photoreceptors in the human retina. We present a simple two-pass kernel foveated rendering (KFR) pipeline that maps well onto modern GPUs. In the first

Authors' addresses: Xiaoxu Meng, xmeng525@umiacs.umd.edu, Augmentarium, Department of Computer Science and the University of Maryland Institute for Advanced Computer Studies (UMIACS), University of Maryland, College Park; Ruofei Du, ruofei@umiacs.umd.edu, Augmentarium, Department of Computer Science and the University of Maryland Institute for Advanced Computer Studies (UMIACS), University of Maryland, College Park; Matthias Zwicker, zwicker@umiacs.umd.edu, Augmentarium, Department of Computer Science and the University of Maryland Institute for Advanced Computer Studies (UMIACS), University of Maryland, College Park; Amitabh Varshney, varshney@umiacs.umd.edu, Augmentarium, Department of Computer Science and the University of Maryland Institute for Advanced Computer Studies (UMIACS), University of Maryland, College Park.

2577-6193/2018/5-ART5 $15.00
https://doi.org/10.1145/3203199

pass, we compute the kernel log-polar transformation and render to a reduced-resolution buffer. In the second pass, we carry out the inverse-log-polar transformation with anti-aliasing to map the reduced-resolution rendering to the full-resolution screen. We have carried out pilot and formal user studies to empirically identify the KFR parameters. We observe a $2.8X - 3.2X$ speedup in rendering on $4K\ UHD\ (2160p)$ displays with minimal perceptual loss of detail. The relevance of eye-tracking-guided kernel foveated rendering can only increase as the anticipated rise of display resolution makes it ever more difficult to resolve the mutually conflicting goals of interactive rendering and perceptual realism.

CCS Concepts: • **Computing methodologies** → **Perception**; *Visibility*;

Additional Key Words and Phrases: foveated rendering, perception, log-polar mapping, eye-tracking, virtual reality, head-mounted displays

ACM Reference Format:
Xiaoxu Meng, Ruofei Du, Matthias Zwicker, and Amitabh Varshney. 2018. Kernel Foveated Rendering. *Proc. ACM Comput. Graph. Interact. Tech.* 1, 1, Article 5 (May 2018), 20 pages. https://doi.org/10.1145/3203199

1 INTRODUCTION

Human vision spans a field of view of $135° \times 160°$, but the highest-resolution foveal vision covers only the central $1.5° \times 2°$ [Guenter et al. 2012]. Patney *et al.* [2016b] 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 [Levoy and Whitaker 1990] for large displays, especially for virtual and augmented reality headsets equipped with eye trackers.

Araujo and Dias [1996] 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 [Antonelli et al. 2015]. However, to the best of our knowledge, direct use of the log-polar mapping for 3D graphics has not yet been attempted on GPUs.

In this paper, we 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 [Stengel et al. 2016] and the classic three-pass foveated rendering pipeline [Guenter et al. 2012], KFR just needs a two-pass algorithm. In the first pass, we carry out the kernel log-polar transformation and render to a reduced-resolution framebuffer using deferred shading [Duluk Jr et al. 2004; Hargreaves and Harris 2004]. In the second pass, we apply the inverse kernel log-polar transformation to the reduced-resolution framebuffer to map the final foveated rendering to the full-resolution display.

We have carried out user studies to empirically establish the parameters for kernel foveated rendering that allow preservation of perceptually accurate foveal detail and lower peripheral detail. We have validated our approach on 3D rendering of textured meshes as well as ray-marching scenes.

In summary, our 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.

2 RELATED WORK

Our work builds upon a rich literature of prior art on foveated images, videos, and foveated rendering for 3D graphics.

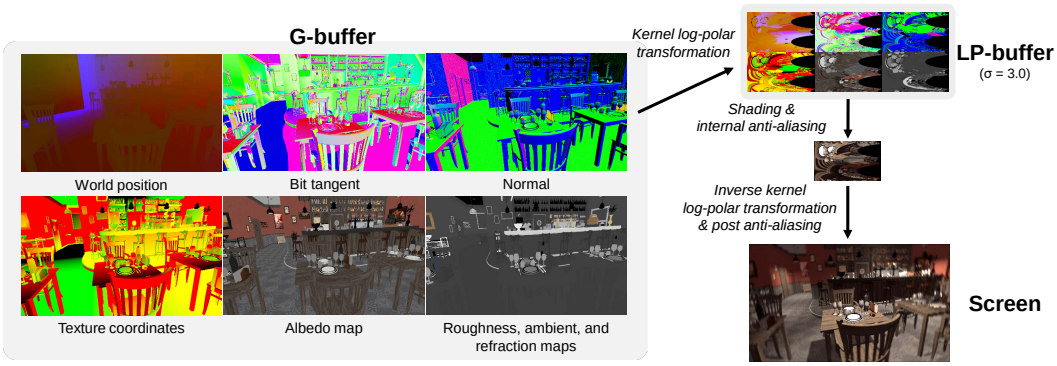


Fig. 2. An overview of our KFR pipeline. We transform the necessary parameters and textures in the G-buffer from Cartesian coordinates to log-polar coordinates, compute lighting in the log-polar (LP) buffer and perform internal anti-aliasing. Next, we apply the inverse transformation to recover the framebuffer in Cartesian coordinates and employ post anti-aliasing to reduce the foveation artifacts.

2.1 Foveated Images and Videos

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

Burt [1988] 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 [1996] have developed one of the earliest eye-tracking-based foveated imaging systems with space-variant degradation. 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 [Shapiro 1993], set partitioning in hierarchical trees [Said and Pearlman 1996], wavelet-based image foveation [Chang et al. 2000], embedded foveation image coding [Wang and Bovik 2001], and gigapixel displays [Papadopoulos and Kaufman 2013].

Video foveation has also been explored [Lee and Sanghoon 2000; Reeves and Robinson 1996; Wang and Bovik 2005]. The filter bank method is used for video preprocessing before using standard video compression algorithms (*e.g. MPEG and H.26x*) [Lee and Sanghoon 2000; Lee et al. 2001, 2002]. Foveation filtering has been implemented with the quantization processes in standard MPEG and H.26x compression [Sheikh et al. 2003, 2002].

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.* [2015], 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.

2.2 Foveated 3D Graphics

Weier *et al.* [2017] have reviewed several approaches for foveated rendering including mesh simplification in the areas of lower acuity [Hoppe 1998; Hu et al. 2010; Ohshima et al. 1996]. However, these days shading has often been found to dominate the cost for rendering sophisticated scenes on modern graphics pipelines [He et al. 2014; Vaidyanathan et al. 2014].

Ragan-Kelley *et al.* [2011] use decoupled sampling for stochastic super-sampling of motion and defocus blur at a reduced shading cost. Guenter *et al.* [2012] present a three-pass pipeline for foveated 3D rendering by using three eccentricity layers around the tracked gaze point. 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 LOD. They interpolate and blend between the layers and use frame jitter and temporal re-projection to reduce spatial and temporal artifacts.

Vaidyanathan *et al.* [2014] present a novel approach using a generalization of multi-sample antialiasing (MSAA). They perform foveated rendering by sampling coarse pixels (2×2 pixels and 4×4 pixels) in the peripheral regions. 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.* [2016a; 2016b] address temporal artifacts in foveated rendering by using pre-filters and temporal anti-aliasing. They also show that contrast preservation greatly enhances the image quality by reducing the tunneling effect. Clarberg *et al.* [2014] propose a modification to the current hardware architecture, which enables flexible control of shading rates and automatic shading reuse between triangles in tessellated primitives. He *et al.* [2014] 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. However, this implementation of multi-rate shading requires an extension of the graphics pipeline, which is not available on commodity graphics hardware. Swafford *et al.* [2016] implement four foveated renderers. The first method reduces peripheral resolution. The second varies per-pixel depth-buffer samples in the fovea and periphery for screen-space ambient occlusion. The third implements a terrain renderer using GPU-level tessellation for the fovea. The final method varies the per-pixel ray-casting steps across the field of view. Stengel *et al.* [2016] use adaptive sampling from fovea to peripheral regions in a gaze-contingent rendering pipeline. To compensate for the missing pixels caused by sparsely distributed shading samples on the periphery, they use pull-push [Gortler et al. 1996] 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.* [2017] design a real-time foveated 4D light field rendering and display system. Their prototype renders only $16\% - 30\%$ of the rays without compromising the perceptual quality.

Recently, deferred shading has been used for antialiasing foveated rendering. Karis [2014] optimizes temporal anti-aliasing for deferred shading, which uses samples over multiple frames to reduce flickering. Crassin *et al.* [2015] reduce aliasing by pre-filtering sub-pixel geometric detail in the G-buffer for deferred shading. Chajdas *et al.* [2011]'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 paper, we present a simple two-pass foveated rendering pipeline that maps well onto modern GPUs. KFR provides gradually changing resolution and achieves $2.8X - 3.2X$ speedup with little perceptual loss.

3 OUR APPROACH

Overall, our 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 2.

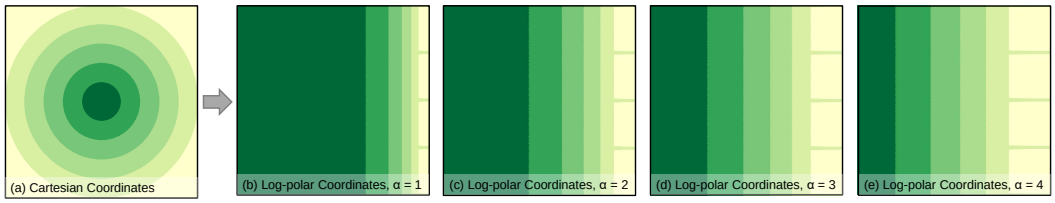


Fig. 3. Transformation from Cartesian coordinates to log-polar coordinates with kernel function $\mathbf{K}(x) = x^{\alpha}$. (a) is the image in the Cartesian coordinates, (b)–(e) are the corresponding images in the log-polar coordinates with varying kernel parameter α. Matching colors in the log-polar and Cartesian coordinates show the same regions.

Fig. 4. 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.

In the classic log-polar transformation [Antonelli et al. 2015], 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 1,

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

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,

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

$$\mathbf{1}\left[y' < 0\right] = \begin{cases} 1 & , y' < 0 \\ 0 & , y' \geq 0 \end{cases} \tag{3}$$

Notice how the central dark green area in Figure 3 (a) is mapped to a relatively large region in the left part of the log-polar coordinates in Figure 3 (b), while the peripheral regions of Figure 3 (a) are mapped to a relatively small part of Figure 3 (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

$$A = \frac{L}{w}, \quad B = \frac{2\pi}{h}, \tag{4}$$

then the inverse transformation can be formulated as Equation 5,

$$\begin{aligned} x'' &= \exp(Au)\cos(Bv) \\ y'' &= \exp(Au)\sin(Bv) \end{aligned} \tag{5}$$

To understand how the resolution changes in the log-polar space, consider $r = \|x, y\|_2 = \exp(Au)$. Now, d$r$ represents the change in r based on u,

$$\mathrm{d}r = A \cdot \exp(Au)\,\mathrm{d}u. \tag{6}$$

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

$$\mathscr{D} = \frac{\mathrm{d}u}{\mathrm{d}r} = \frac{1}{A} \cdot \exp(-Au). \tag{7}$$

Equation 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.

We 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,

$$\mathscr{D} = \frac{\exp\left(-wC\sigma \cdot \mathbf{K^{-1}}\left(\frac{u}{w}\right)\right)}{C\sigma \cdot \mathbf{K^{-1'}}\left(\frac{u}{w}\right)}. \tag{8}$$

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,

$$\mathbf{K}(x) = \sum_{i=0}^{\infty} \beta_i x^i, \quad \text{where} \sum_{i=0}^{\infty} \beta_i = 1. \tag{9}$$

Such kernel functions can be used to adjust the pixel density distribution in the LP-buffer. We use $\mathbf{K}(x) = \sum_{i=0}^{\infty} \beta_i x^i$ in this paper 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 5 [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 3 (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 6 with $\sigma = 1.8$ and $C = \sqrt{2}$. The use of the kernel function reduces the artifacts in the zoomed-in peripheral view, improving

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

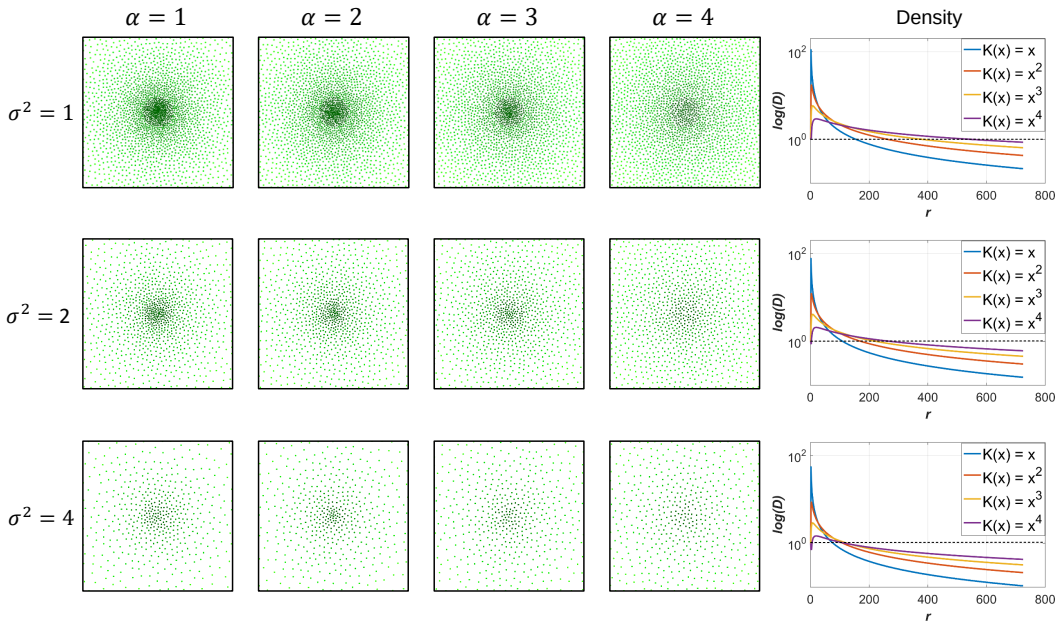


Fig. 5. The relationship among σ^2, $\mathbf{K}(x) = x^{\alpha}$, and the sampling rate. The number of samples in each image is proportional to σ^2. We use a variant of the PixelPie algorithm [Ip et al. 2013] to generate the Poisson samples shown.

the peripheral image quality. Meanwhile, as shown in Figure 4, 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 7 with fixed $\alpha = 4.0$, $C = \sqrt{2}$.

3.1 Pass I: Forward Log-Polar Transformation

The G-buffer for the deferred shading pipeline contains object and world-space coordinates, world-space normals, texture coordinates, depth, and material-related information. In Pass I, we transform the contents in the G-buffer from the Cartesian coordinates to the log-polar coordinates, compute direct and indirect lighting at each pixel, and render to the reduced-resolution log-polar (LP)-buffer.

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, we change Equation 1 to Equation 10,

$$\begin{aligned} u &= \mathbf{K}^{-1}\left(\frac{\log\|x', y'\|_2}{L}\right) \cdot w \\ v &= \left(\arctan\left(\frac{y'}{x'}\right) + \mathbf{1}\left[y' < 0\right] \cdot 2\pi\right) \cdot \frac{h}{2\pi} \end{aligned} \tag{10}$$

Here,

$$x' = x - \mathring{x}, \quad y' = y - \mathring{y}. \tag{11}$$

$\mathbf{K}^{-1}(\cdot)$ 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 12,

$$L = \log\left(\max\left(\max\left(l_1, l_2\right), \max\left(l_3, l_4\right)\right)\right). \tag{12}$$

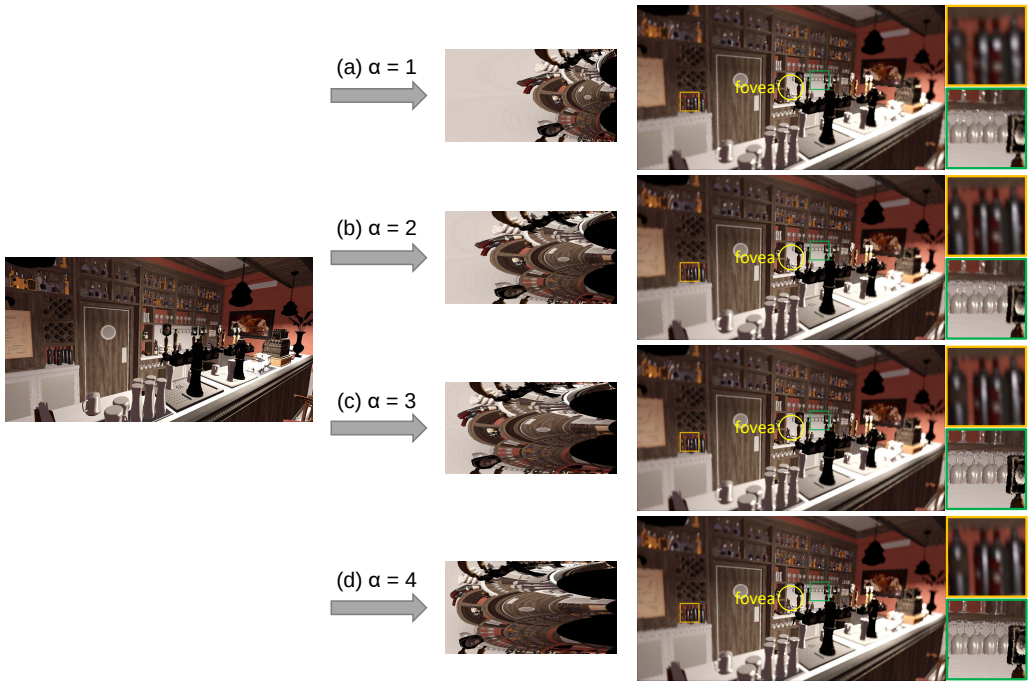


Fig. 6. Comparison of foveated rendering with varying α for 2560×1440 resolution. From left to right: original rendering, kernel log-polar rendering, and the foveated rendering with zoomed-in view of the peripheral regions. Here $\sigma = 1.8$, (a) classic log-polar transformation, i.e. $\alpha = 1.0$, (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.

Here,

$$\begin{aligned} l_1 &= \|\mathring{x}, \mathring{y}\|_2 \\ l_2 &= \|W - \mathring{x}, H - \mathring{y}\|_2 \\ l_3 &= \|\mathring{x}, H - \mathring{y}\|_2 \\ l_4 &= \|W - \mathring{x}, \mathring{y}\|_2 \end{aligned} \tag{13}$$

Lighting. In lighting calculation for traditional deferred shading, mesh positions, normals, depth and material information such as roughness, index of reflection, and normal maps are fetched from the G-buffer [Duluk Jr et al. 2004; Hargreaves and Harris 2004]. Instead of obtaining information from the G-buffer with texture coordinates (x, y), in our approach, we 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.

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, we can directly perform denoising in the log-polar space. To reduce artifacts in the peripheral regions, we use a Gaussian filter with a 3×3 kernel for the right part of the texture (corresponding to the peripheral regions) in the

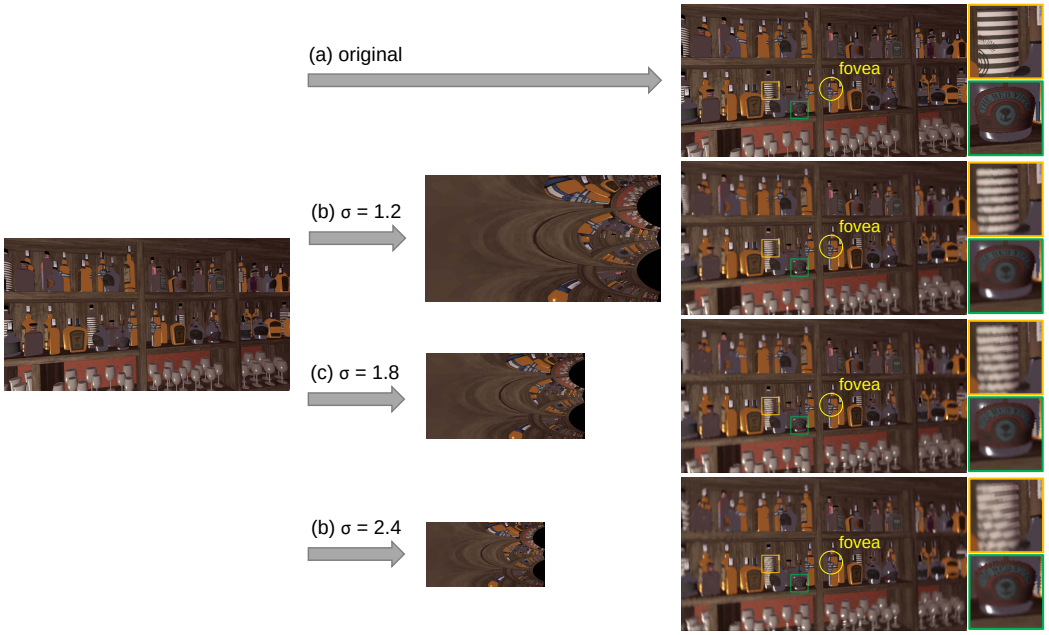


Fig. 7. Comparison of foveated rendering with varying σ for 2560×1440 resolution. From left to right: original rendering, kernel log-polar rendering, the recovered scene in Cartesian coordinates, and a zoomed-in view of peripheral regions. Here, $\mathbf{K}(x) = x^4$, (a) full-resolution rendered at 31 FPS, (b) $\sigma = 1.2$ at 43 FPS, (c) $\sigma = 1.8$ at 67 FPS, and (d) $\sigma = 2.4$ at 83 FPS.

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

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.

3.2 Pass II: Inverse Log-Polar Transformation

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

Inverse Kernel Log-polar Mapping Transformation. We can recover the Cartesian coordinates (x'', y''), from the pixel coordinates (u, v) and the fovea coordinates $(\mathring{x}, \mathring{y})$ using Algorithm 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. We apply temporal anti-aliasing (TAA) [Karis 2014] with Halton sampling [Pengo et al. 2009] to the recovered screen-space pixels after the inverse kernel log-polar transformation. We also use Gaussian filtering with different kernel sizes η for different L (as defined in Equation 12) in post anti-aliasing. The kernel size η is shown in Equation 14, which depends on the normalized distance between the pixel coordinate and the fovea,

$$\eta = 3 + 2 \times \left\lfloor \frac{\frac{\|x', y'\|_2}{e^L} - 0.10}{0.05} \right\rfloor. \tag{14}$$

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 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

4 USER STUDIES

We 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, we conducted a pilot study to examine a broad range of the two parameters, σ^2 and α. We used the results and our experience with the pilot study to fine tune the protocol and ranges of σ^2 and α for the final user study.

4.1 Apparatus

Our user study apparatus, shown in Figure 8, consists of an *Alienware* laptop with an NVIDIA 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.

4.2 Pilot Study

Procedure. The session for each participant lasted between $35 - 50$ minutes and involved four stages: introduction, calibration, training, and testing. In the introduction stage, we showed participants the *FOVE* headset, the eye trackers, and the *XBOX* controllers and discussed how to use them. We did not provide any information about our research or the algorithm to avoid biasing the participants towards any rendering. After the participant comfortably wore the HMD, we moved forward to the calibration stage, where we ran a one-minute eye-tracking calibration program provided by the *FOVE* software development kit. In the training stage, we 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.

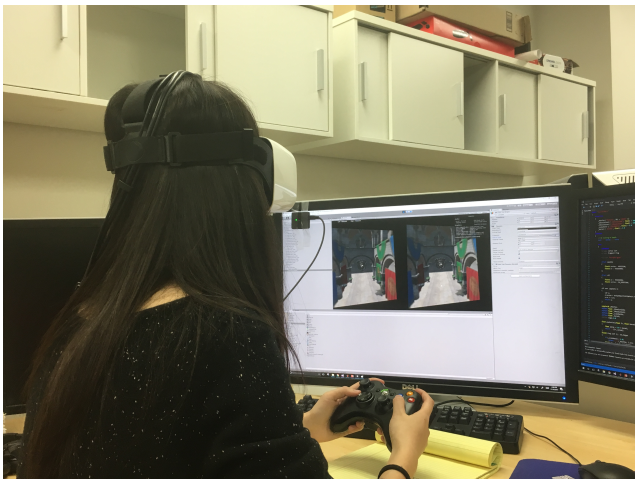
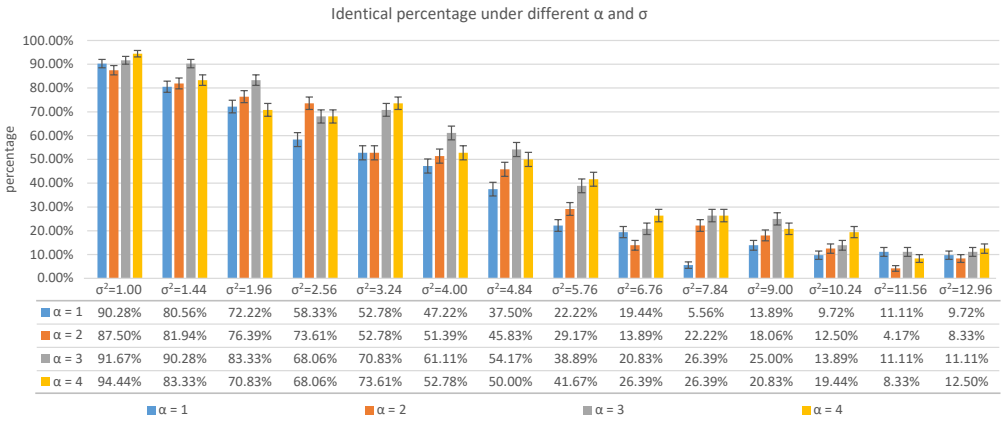


Fig. 8. User study setup.



	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%

Fig. 9. 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.

Trials in the training and testing stages were identical. In each trial of our two-alternative forced choice test, we 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, we 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. We instructed the participants to maintain their gaze at the center of the screen, even though our 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. We rendered scenes from the *Sponza* and *Amazon Lumberyard Bistro* datasets for different sessions. We allowed the participants to have some rest between different sessions.

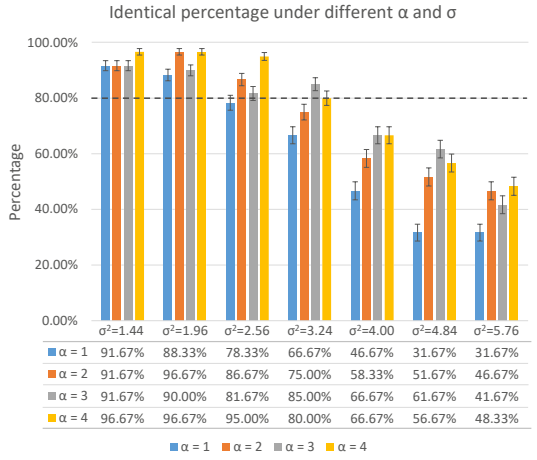
Participants. In the pilot study, we recruited 24 participants via campus email lists and flyers. All participants are at least 18 years old with normal or corrected-to-normal vision (with contact lenses).

Results and Analysis. We 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 9. First, we 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, we 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. 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, we modified the final user study to be shorter and more focused. First, to reduce the total time that participants are in the HMD, we used the fact that most participants found foveated renderings different from the full-resolution rendering for $\sigma > 2.4$ ($\sigma^2 > 5.76$). Since our goal is to accelerate rendering while maintaining perceptually similar quality, we 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, we observed that the participants quickly came up to speed within a couple of trials in the training session. We 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 we 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, we 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, we 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.

4.3 Final User Study

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.



	σ²=1.44	σ²=1.96	σ²=2.56	σ²=3.24	σ²=4.00	σ²=4.84	σ²=5.76
α = 1	91.67%	88.33%	78.33%	66.67%	46.67%	31.67%	31.67%
α = 2	91.67%	96.67%	86.67%	75.00%	58.33%	51.67%	46.67%
α = 3	91.67%	90.00%	81.67%	85.00%	66.67%	61.67%	41.67%
α = 4	96.67%	96.67%	95.00%	80.00%	66.67%	56.67%	48.33%

Fig. 10. 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.

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

Participants. We 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).

Results and Analysis. We report the percentage P_I and the corresponding standard error in Figure 10. We 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 [Patil 1975; Vinnikov and Allison 2014], we 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.

Table 1. Cochran's Q values at different σ^2.

σ^2	1.44	1.96	2.56	3.24	4.00	4.84	5.76
Cochran's 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

To achieve visually acceptable results for foveated rendering, we use a threshold of 80% responses considering foveated rendering to be visually indistinguishable from full-resolution rendering. To achieve the highest rendering acceleration, we look for the highest σ that met this threshold. We therefore choose $\sigma = 1.8$ ($\sigma^2 = 3.24$) and $\alpha = 4$ as our desired parameters for the interactive rendering evaluation.

Table 2. 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 3. 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

5 RENDERING ACCELERATION

We implemented kernel foveated rendering on NVIDIA GeForce GTX 1080, by using the deferred shading pipeline of the *Falcor* engine [Benty et al. 2017]. We report results of our rendering acceleration for resolutions of 1920×1080, 2560×1440, and 3840×2160. Using the results from our final user study, we selected the LP-buffer parameter $\sigma = 1.8$, and kernel parameter $\alpha = 4$ for the evaluations below.

3D Textured Meshes. We use the *Amazon Lumberyard Bistro* [Lumberyard 2017] scene with physically-based shading, reflection, refraction, and shadows to simulate the complex shading effects as shown in Figure 11. The comparison of the break-down of rendering time between KFR and the ground truth of deferred shading is shown in Table 2. We observed that the frame rate increases for all resolutions as shown in Table 3, with a speedup of $2.0X - 2.8X$.

Ray-casting Rendering. Rendering of high-resolution ray cast scenes can be an extremely time-consuming process. We used the complex ray-casting scene with 16 different primitives by Íñigo Quílez to evaluate the acceleration of kernel foveated rendering. Figure 12 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 $2.9X - 3.2X$.

6 DISCUSSION

Here we compare our Kernel Foveated Rendering (KFR) pipeline with selected prior art, including: *Foveated 3D Graphics* (F3D) [Guenter et al. 2012], *Multi-rate Shading* (MRS) [He et al. 2014], *Coarse*

Fig. 11. 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.

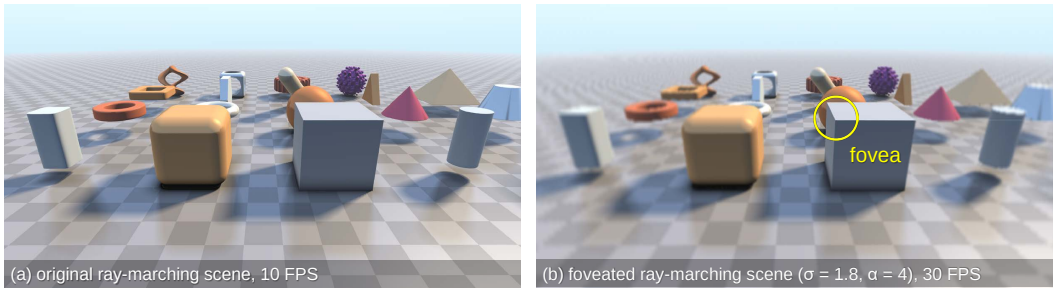


Fig. 12. Comparison of (a) full-resolution rendering and (b) foveated ray-marching scene with 16 samples per pixel rendered at 2560×1440.

Pixel Shading (CPS) [Vaidyanathan et al. 2014], and *Adaptive Image-Space sampling* (AIS) [Stengel et al. 2016].

As mentioned by [Stengel et al. 2016], 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 our 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 our 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 [Stengel et al. 2016]. In contrast, KFR could easily be coupled with the state-of-the-art screen-space anti-aliasing techniques, such as TAA [Karis 2014] and recent G-buffer anti-aliasing strategies [Crassin et al. 2015].

Both the AIS and KFR pipelines mimic the continuously changing distribution 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 [Weymouth 1958] 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 [Gortler et al. 1996] 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 [Kim et al. 2010; Lee et al. 2005] 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.

7 LIMITATIONS

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

Foveation Parameters. As discussed in Hsu *et al.* [2017], 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 paper 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.

Temporal Flickering. In the post-processing stage, we have applied TAA to tackle the temporal flickering problem. However, in fly-through of the scene with glossy objects, we notice that view-dependent specular reflection changes before and after applying KFR. As shown in Figure 13, foveation amplifies the specular reflection regions, and makes the specular highlights flicker more.

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. Our choice of kernel functions is not unique; other mapping algorithms with different kernel functions could provide a better mapping to the human vision system.

8 CONCLUSION AND FUTURE WORK

In this paper, we 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.

First, we 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, we 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, we evaluate our KFR pipeline through a quantitative evaluation with physically-based deferred-rendering scenes and ray-marching scenes. We have observed that our 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, our KFR pipeline allows rendering more complex shaders (*e.g.*, real-time global illumination and physically-based rendering [Pharr and Humphreys 2010]) in real time, thus bringing higher power efficiency and better user experience for 3D games and other interactive

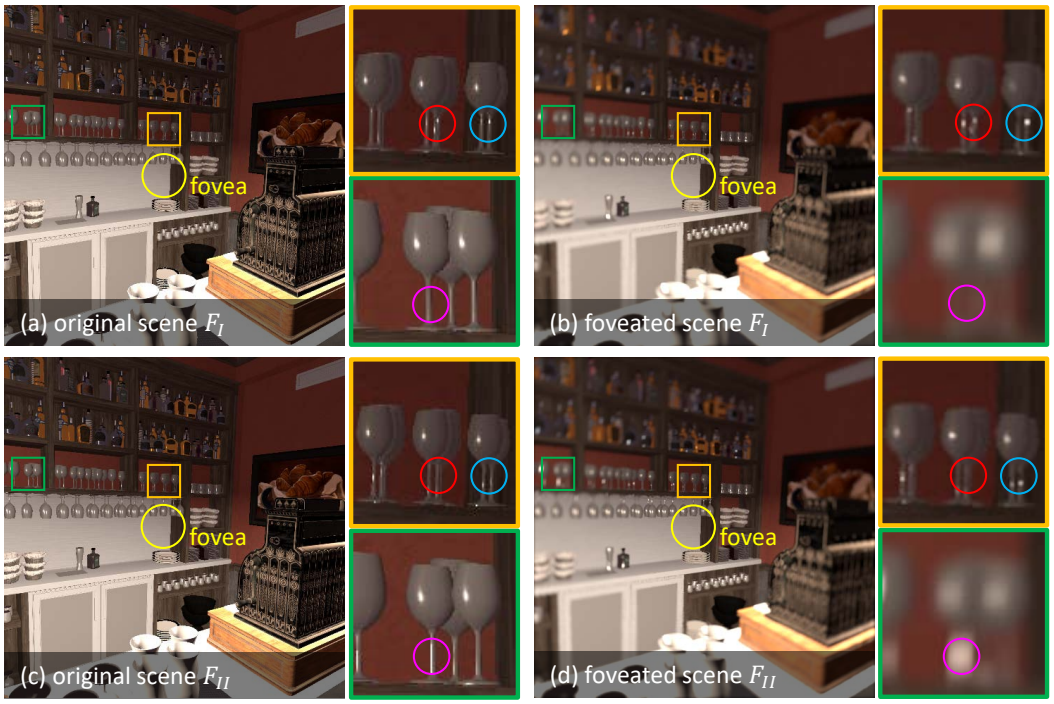


Fig. 13. Temporal flickering issue. The original scene and the foveated scene of two consecutive frames (F_I and F_{II}). In F_I, the specular reflection in the original scene as shown in the red and blue circles in the zoomed-in view of (a) are amplified in the foveated scene as shown in the zoomed-in 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).

visual computing applications. The work presented in this paper is the first step towards kernel foveated rendering.

In future, we would like to apply our algorithm for interactively testing high-fidelity path-tracing algorithms with the state-of-art noise reduction algorithms [Crassin et al. 2015; Moon et al. 2017; Selgrad et al. 2015]. Considering that current consumer HMDs have lens distortions [Pohl et al. 2016], we also plan to explore warping from LP-buffer to "lens undistorted" image directly in order to provide substantial performance improvements.

ACKNOWLEDGEMENTS

We would also like to thank the anonymous reviewers for the insightful comments that greatly helped improve this manuscript.

This work has been supported in part by the NSF Grants 14-29404, 15-64212, and the State of Maryland's MPower initiative. Any opinions, findings, conclusions, or recommendations expressed in this article are those of the authors and do not necessarily reflect the views of the research sponsors.

REFERENCES

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

2015), 533–550. https://doi.org/10.1007/s11554-012-0281-6

H. Araujo and J. M. Dias. 1996. An introduction to the log-polar mapping [image sampling]. In *Proceedings II Workshop on Cybernetic Vision*. 139–144. https://doi.org/10.1109/CYBVIS.1996.629454

Nir Benty, Kai-Hwa Yao, Tim Foley, Anton S. Kaplanyan, Conor Lavelle, Chris Wyman, and Ashwin Vijay. 2017. The Falcor rendering framework. https://github.com/NVIDIAGameWorks/Falcor

Peter J Burt. 1988. Smart sensing within a pyramid vision machine. *Proc. IEEE* 76, 8 (1988), 1006–1015. https://doi.org/10.1109/5.5971

Matthäus G. Chajdas, Morgan McGuire, and David Luebke. 2011. Subpixel reconstruction antialiasing for deferred shading. In *Symposium on Interactive 3D Graphics and Games (I3D '11)*. ACM, New York, NY, USA, 15–22 PAGE@7. https://doi.org/10.1145/1944745.1944748

Ee-Chien Chang, Stéphane Mallat, and Chee Yap. 2000. Wavelet foveation. *Applied and Computational Harmonic Analysis* 9, 3 (2000), 312–335. https://doi.org/10.1006/acha.2000.0324

Petrik Clarberg, Robert Toth, Jon Hasselgren, Jim Nilsson, and Tomas Akenine-Möller. 2014. AMFS: adaptive multi-frequency shading for future graphics processors. *ACM Trans. Graph.* 33, 4, Article 141 (July 2014), 12 pages. https://doi.org/10.1145/2601097.2601214

Cyril Crassin, Morgan McGuire, Kayvon Fatahalian, and Aaron Lefohn. 2015. Aggregate G-buffer anti-aliasing. In *Proceedings of the 19th Symposium on Interactive 3D Graphics and Games (I3D '15)*. ACM, New York, NY, USA, 109–119. https://doi.org/10.1145/2699276.2699285

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. 2004. Deferred shading graphics pipeline processor having advanced features. US Patent 6,717,576.

Steven J. Gortler, Radek Grzeszczuk, Richard Szeliski, and Michael F. Cohen. 1996. The lumigraph. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '96)*. ACM, New York, NY, USA, 43–54. https://doi.org/10.1145/237170.237200

Brian Guenter, Mark Finch, Steven Drucker, Desney Tan, and John Snyder. 2012. Foveated 3D graphics. *ACM Trans. Graph.* 31, 6, Article 164 (Nov. 2012), 10 pages. https://doi.org/10.1145/2366145.2366183

Shawn Hargreaves and Mark Harris. 2004. Deferred shading. In *Game Developers Conference*, Vol. 2. 31.

Yong He, Yan Gu, and Kayvon Fatahalian. 2014. Extending the graphics pipeline with adaptive, multi-rate shading. *ACM Trans. Graph.* 33, 4, Article 142 (July 2014), 12 pages. https://doi.org/10.1145/2601097.2601105

Hugues Hoppe. 1998. Smooth view-dependent level-of-detail control and its application to terrain rendering. In *Proceedings of the Conference on Visualization '98 (VIS '98)*. IEEE Computer Society Press, Los Alamitos, CA, USA, 35–42. http://dl.acm.org/citation.cfm?id=288216.288221

Chih-Fan Hsu, Anthony Chen, Cheng-Hsin Hsu, Chun-Ying Huang, Chin-Laung Lei, and Kuan-Ta Chen. 2017. 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)*. ACM, New York, NY, USA, 55–63. https://doi.org/10.1145/3123266.3123434

L. Hu, P. V. Sander, and H. Hoppe. 2010. Parallel view-dependent level-of-detail control. *IEEE Transactions on Visualization and Computer Graphics* 16, 5 (Sept 2010), 718–728. https://doi.org/10.1109/TVCG.2009.101

Cheuk Yiu Ip, M. Adil Yalçin, David Luebke, and Amitabh Varshney. 2013. PixelPie: maximal Poisson-disk sampling with rasterization. In *Proceedings of the 5th High-Performance Graphics Conference (HPG '13)*. ACM, New York, NY, USA, 17–26. https://doi.org/10.1145/2492045.2492047

B Karis. 2014. High-quality temporal supersampling. *Advances in Real-Time Rendering in Games, SIGGRAPH Courses* 1 (2014), 1–55. https://doi.org/10.1145/2504435.2504447

Youngmin Kim, Amitabh Varshney, David Jacobs, and Francois Guimbretere. 2010. Mesh Saliency and Human Eye Fixations. *ACM Transactions on Applied Perception* 7, 2 (2010), 1 – 13.

Philip Kortum and Wilson S Geisler. 1996. Implementation of a foveated image coding system for image bandwidth reduction. In *Electronic Imaging: Science & Technology*. International Society for Optics and Photonics, 350–360. https://doi.org/10.1117/12.238732

Lee and Sanghoon. 2000. *Foveated video compression and visual communications over wireless and wireline networks*. Ph.D. Dissertation. Dept. of ECE, University of Texas at Austin.

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

Sanghoon Lee, M. S. Pattichis, and A. C. Bovik. 2001. Foveated video compression with optimal rate control. *IEEE Transactions on Image Processing* 10, 7 (Jul 2001), 977–992. https://doi.org/10.1109/83.931092

Sanghoon Lee, M. S. Pattichis, and A. C. Bovik. 2002. Foveated video quality assessment. *IEEE Transactions on Multimedia* 4, 1 (Mar 2002), 129–132. https://doi.org/10.1109/6046.985561

Marc Levoy and Ross Whitaker. 1990. Gaze-directed volume rendering. In *Proceedings of the 1990 Symposium on Interactive 3D Graphics (I3D '90)*. ACM, New York, NY, USA, 217–223. https://doi.org/10.1145/91385.91449

Amazon Lumberyard. 2017. Amazon Lumberyard Bistro, Open Research Content Archive (ORCA). http://developer.nvidia.com/orca/amazon-lumberyard-bistro

Bochang Moon, Jose A. Iglesias-Guitian, Steven McDonagh, and Kenny Mitchell. 2017. Noise reduction on G-buffers for Monte Carlo filtering. *Computer Graphics Forum* 36, 8 (2017), 600–612. https://doi.org/10.1111/cgf.13155

Toshikazu Ohshima, Hiroyuki Yamamoto, and Hideyuki Tamura. 1996. Gaze-directed adaptive rendering for interacting with virtual space. In *Proceedings of the 1996 Virtual Reality Annual International Symposium (VRAIS 96) (VRAIS '96)*. IEEE Computer Society, Washington, DC, USA, 103–110, 267. https://doi.org/10.1109/VRAIS.1996.490517

C. Papadopoulos and A. E. Kaufman. 2013. Acuity-driven gigapixel visualization. *IEEE Transactions on Visualization and Computer Graphics* 19, 12 (Dec 2013), 2886–2895. https://doi.org/10.1109/TVCG.2013.127

Kashinath D Patil. 1975. Cochran's Q test: Exact distribution. *J. Amer. Statist. Assoc.* 70, 349 (1975), 186–189.

Anjul Patney, Joohwan Kim, Marco Salvi, Anton Kaplanyan, Chris Wyman, Nir Benty, Aaron Lefohn, and David Luebke. 2016a. Perceptually-based foveated virtual reality. In *ACM SIGGRAPH 2016 Emerging Technologies (SIGGRAPH '16)*. ACM, New York, NY, USA, Article 17, 2 pages. https://doi.org/10.1145/2929464.2929472

Anjul Patney, Marco Salvi, Joohwan Kim, Anton Kaplanyan, Chris Wyman, Nir Benty, David Luebke, and Aaron Lefohn. 2016b. Towards foveated rendering for gaze-tracked virtual reality. *ACM Trans. Graph.* 35, 6, Article 179 (Nov. 2016), 12 pages. https://doi.org/10.1145/2980179.2980246

T Pengo, A Muñoz-Barrutia, and C Ortiz-de solórzano. 2009. Halton sampling for autofocus. *Journal of Microscopy* 235, 1 (2009), 50–58. https://doi.org/10.1111/j.1365-2818.2009.03180.x

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

D. Pohl, X. Zhang, and A. Bulling. 2016. Combining eye tracking with optimizations for lens astigmatism in modern wide-angle HMDs. In *2016 IEEE Virtual Reality (VR)*. 269–270. https://doi.org/10.1109/VR.2016.7504757

Jonathan Ragan-Kelley, Jaakko Lehtinen, Jiawen Chen, Michael Doggett, and Frédo Durand. 2011. Decoupled sampling for graphics pipelines. *ACM Trans. Graph.* 30, 3, Article 17 (May 2011), 17 pages. https://doi.org/10.1145/1966394.1966396

T. H. Reeves and J. A. Robinson. 1996. Adaptive foveation of MPEG video. In *Proceedings of the Fourth ACM International Conference on Multimedia (MULTIMEDIA '96)*. ACM, New York, NY, USA, 231–241. https://doi.org/10.1145/244130.244218

A. Said and W. A. Pearlman. 1996. 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 (Jun 1996), 243–250. https://doi.org/10.1109/76.499834

Kai Selgrad, Christian Reintges, Dominik Penk, Pascal Wagner, and Marc Stamminger. 2015. Real-time depth of field using multi-layer filtering. In *Proceedings of the 19th Symposium on Interactive 3D Graphics and Games (i3D '15)*. ACM, New York, NY, USA, 121–127. https://doi.org/10.1145/2699276.2699288

Jerome M Shapiro. 1993. Embedded image coding using zerotrees of wavelet coefficients. *IEEE Transactions on Signal Processing* 41, 12 (1993), 3445–3462. https://doi.org/10.1109/78.258085

Hamid R. Sheikh, Brian L. Evans, and Alan C. Bovik. 2003. Real-time foveation techniques for low bit rate video coding. *Real-Time Imaging* 9, 1 (Feb. 2003), 27–40. https://doi.org/10.1016/S1077-2014(02)00116-X

H. R. Sheikh, S. Liu, Z. Wang, and A. C. Bovik. 2002. Foveated multipoint videoconferencing at low bit rates. In *2002 IEEE International Conference on Acoustics, Speech, and Signal Processing*, Vol. 2. II–2069–II–2072. https://doi.org/10.1109/ICASSP.2002.5745041

Michael Stengel, Steve Grogorick, Martin Eisemann, and Marcus Magnor. 2016. Adaptive image-space sampling for gaze-contingent real-time rendering. *Computer Graphics Forum* 35, 4 (2016), 129–139. https://doi.org/10.1111/cgf.12956

Qi Sun, Fu-Chung Huang, Joohwan Kim, Li-Yi Wei, David Luebke, and Arie Kaufman. 2017. Perceptually-guided foveation for light field displays. *ACM Trans. Graph.* 36, 6, Article 192 (Nov. 2017), 13 pages. https://doi.org/10.1145/3130800.3130807

Nicholas T. Swafford, José A. Iglesias-Guitian, Charalampos Koniaris, Bochang Moon, Darren Cosker, and Kenny Mitchell. 2016. User, metric, and computational evaluation of foveated rendering methods. In *Proceedings of the ACM Symposium on Applied Perception (SAP '16)*. ACM, New York, NY, USA, 7–14. https://doi.org/10.1145/2931002.2931011

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. 2014. Coarse pixel shading. In *Proceedings of High Performance Graphics (HPG '14)*. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 9–18. http://dl.acm.org/citation.cfm?id=2980009.2980011

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

Zhou Wang and Alan C Bovik. 2001. Embedded foveation image coding. *IEEE Transactions on Image Processing* 10, 10 (2001), 1397–1410. https://doi.org/10.1109/83.951527

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

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. 2017. Perception-driven accelerated rendering. *Comput. Graph. Forum* 36, 2 (May 2017), 611–643. https://doi.org/10.1111/cgf.13150

Frank W Weymouth. 1958. Visual sensory units and the minimal angle of resolution. *American Journal of Ophthalmology* 46, 1 (1958), 102–113. https://doi.org/10.1016/0002-9394(58)90042-4