Selective Foveated Ray Tracing for Head-Mounted Displays

Youngwook Kim* Yunmin Ko† Insung Ihm*
*Department of Computer Science and Engineering, Sogang University, Korea †Snow Corp., Korea

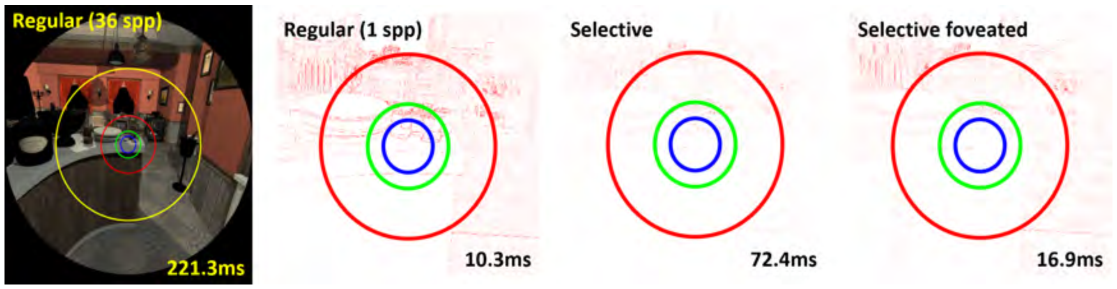


Figure 1: Comparison of rendering results inside the central pixel region (`Interior`). Sufficient pixel sampling is essential for ray tracing, especially when the rendered images are displayed on head-mounted displays. We compared the image quality of different sampling techniques within a field of view of $60.0°$ in diameter (indicated by the yellow circle) that included the foveal ($6.6°$), parafoveal ($11.0°$), perifoveal ($24.2°$), and near-peripheral ($60.0°$) visual fields. The leftmost image was rendered by a Whitted-style ray tracer that used 36 stratified samples per pixel to generate a ground truth image. We then evaluated the performance of three ray tracers that were based on the regular sampling with one sample per pixel, the selective supersampling by Jin et al. [8], and our selective foveated supersampling, respectively. The timings to render stereo images of 1280×1440 pixels per eye for an Oculus Rift S headset were measured on a PC with two NVIDIA GeForce RTX 2080 Ti GPUs. Both Jin et al.'s and our techniques subdivided each image pixel into 2×2 subpixels for adaptive supersampling. Jin et al.'s method always took four samples per subpixel selectively, achieving a pixel sampling rate as effective as up to 16 samples per pixel. On the other hand, ours varied the selective subpixel sampling rate from 9, 4, and 2, to 1 samples per subpixel in the four-component visual fields (enumerated from the innermost one) so that effective pixel sampling rates as high as 36, 16, 8, and 4 were achieved accordingly. A comparison between the three (cropped) difference images against the ground truth visually demonstrates the effectiveness of our method, where it was also confirmed by measuring the peak signal-to-noise ratio (PSNR) against the ground truth, which resulted in $(\mathbf{29.36}, \mathbf{27.54}, 29.76, 32.44)$, $(\mathbf{38.27}, \mathbf{36.84}, 39.34, 41.89)$, and $(\mathbf{45.96}, \mathbf{36.70}, 37.58, 35.75)$, respectively, in the four visual fields (dB, enumerated from the innermost one). As expected, our method could focus computational effort on the most important foveal (and parafoveal) region for minimizing visual artifacts in the neighborhood of the fixation point while achieving high rendering speed (16.9 ms). On the other hand, Jin et al.'s method offered uniform sampling quality across all visual fields, which tended to be better than ours with increasing eccentricity outside the parafovea. However, it came at the cost of increased rendering time (72.4 ms), and the visual difference between the two results outside the parafoveal visual field was hardly perceptible on the tested headset.

Abstract

Although ray tracing produces significantly more realistic images than traditional rasterization techniques, it is still considered computationally burdensome when implemented on a head-mounted display (HMD) system that demands both wide field of view and high rendering rate. A further challenge is that to present high-quality images on an HMD screen, a sufficient number of ray samples should be taken per pixel for effective antialiasing to reduce visually annoying artifacts. In this paper, we present a novel foveated real-time rendering framework that realizes classic Whitted-style ray tracing on an HMD system. In particular, our method proposes combining the selective supersampling technique by Jin et al. [8] with the foveated rendering scheme, resulting in perceptually highly efficient pixel sampling suitable for HMD ray tracing. We show that further enhanced by foveated temporal antialiasing, our ray tracer renders nontrivial 3D scenes in real time on commodity GPUs at high sampling rates as effective as up to 36 samples per pixel (spp) in the foveal area, gradually reducing to at least 1 spp in the periphery.

Index Terms: Computing methodologies—Computer graphics—Rendering—Ray tracing; Human-centered computing—Visualization—Visualization technique

*{kimyu7, ihm}@sogang.ac.kr
†yunmin1130@naver.com

1 Introduction

Achieving photorealistic rendering in real time on high-quality, head-mounted displays (HMDs) with a wide field of view is a challenging topic in the fields of virtual reality (VR) and augmented reality (AR). In order to provide vivid immersive experiences for users of VR/AR applications, a real-time renderer needs to satisfy the fundamental requirements of photorealistic real-time rendering. First, it should implement a methodology that is faithful to the theory of the rendering equation [9] (and, in fact, the volume rendering equation [10], too), from which various realistic rendering techniques have been derived in computer graphics. Second, the adopted rendering algorithm should be efficient enough to guarantee sufficiently high speeds to meet the display refresh rate of HMDs (e.g., 80 Hz or above), which is particularly important for avoiding motion sickness. Unfortunately, these two key elements of photorealistic VR/AR rendering are mutually exclusive, demanding a balance between quality and speed in developing a practical rendering algorithm.

Ray tracing is a good candidate for simulating the rendering equations, as it allows natural implementation of Monte Carlo methods that may effectively simulate the light transportation within a given scene. Unlike the rasterization-based renderers that only rely on approximate, often inaccurate, algorithms, Monte Carlo ray tracers compute various advanced rendering effects in a physically correct manner, which is an essential element of photorealistic rendering. However, due to the limited capability of current hardware technology, full Monte Carlo ray tracing is impractical for display systems such as HMDs that require high pixel densities and frame rates. Even the classic Whitted-style ray tracing [36], which is only capable of creating such effects as shadow and specular reflection/refraction, is still regarded as a highly expensive rendering mechanism for VR/AR applications. This is truer if the image pixels must be supersampled at high sampling rates to reduce visual artifacts in the rendering results. To alleviate the computational burden of the rendering task beyond available computing power, the idea of *perception-driven rendering* has frequently been explored in computer graphics. It is based on an understanding of the characteristics and limitations of the human visual system. Above all, the acuity of human vision, that is, the ability of the human eye to resolve the detail of what it sees, is spatially variable around the center of the gaze. It is at maximum level only inside a very small central foveal region, and falls off dramatically as the visual eccentricity, that is, the angular distance from the eye's center of fixation, grows. Therefore, a variety of image synthesis techniques exploiting such properties of human vision have naturally been developed targeting display systems such as HMDs or wide-screen projection systems, especially those integrated with eye-tracking capability. In this computational scheme, called the *foveated rendering* or the *gaze-contingent rendering*, the computational resources are adaptively allocated to the image pixels for a perceptually optimized rendering in such way that those in the central vision are rendered with greater detail and quality than those in the peripheral vision, which is largely imperceptible.

In this work, we present a novel framework for foveated ray tracing that implements the full Whitted-style ray tracer. Our method aims to provide spatial sampling rates on current commodity GPUs, which are as effective as up to 36 samples per pixel (spp) in the foveal region, progressively decreasing to at least 1 spp in the peripheral region. To realize such high pixel sampling in real-time ray tracing, we combine the selective supersampling technique by Jin et al. [8] with the foveated rendering scheme. By coupling these two methods, a perceptually very efficient pixel sampling suited well for high-performance stereo HMD rendering becomes possible. We also propose an antialiasing technique to reduce temporal visual artifacts so that high-quality images can be presented on HMDs in real time.

By applying the selective foveated sampling technique to full Whitted-style ray tracing, the proposed rendering scheme is capable of creating such pleasant lighting effects as reflection and refraction optically correctly with reduced visual artifacts, which is difficult to produce using traditional rasterization-based methods. In particular, by adopting the selective supersampling scheme, it is possible to emphasize the rendering features that should be rendered with more care. More importantly, our method enables control of the pixel sampling density intuitively across the visual field through a user-defined foveal function. We demonstrate the effectiveness of the proposed ray-tracing method by efficiently synthesizing stereo images for nontrivial 3D scenes on consumer-grade HMD systems.

2 Previous work

Foveated rendering represents a class of gaze-contingent rendering methods that achieve computational efficiency by exploring the essential property of the human visual system, whose acuity is maximal at the fixation and falls off rapidly with increasing eccentricity (refer to the survey article [35] for notable approaches). One of the earliest attempts at foveated rendering is found in the volume rendering algorithm of Levoy and Whitaker [15], where the sampling density in both image and volume space was determined adaptively as a function of visual acuity. The gaze-aware, level-of-detail techniques were also applied to the efficient rendering of polygonal models, e.g., [18,21,33]. For effective foveated rendering on a raster graphics system, Guenter et al. [7] rendered three layers of image with progressively larger visual fields but lower sampling rates, and blended them into a final image. A more efficient, single-pass implementation of the method was possible through coarse pixel shading by Vaidyanathan et al. [29], supporting variable shading rates in a rasterization pipeline while keeping the visibility sampling rate fixed. The perceived visual quality from these methods was then improved by Patney et al. [22] via a carefully designed temporal antialiasing technique. Applying an effective perceptually adaptive sampling pattern is important in foveated rendering. Stengel et al. [28] proposed a deferred-shading algorithm that performs costly shading only for the pixel samples chosen by a gaze-contingent, stochastic probability function and efficiently interpolates the colors for the remaining pixels. Meng et al. [16] also presented a two-pass rendering algorithm using an easily controllable sampling pattern created by combining a polynomial kernel function with the log-polar mapping.

Unlike the rasterization-based methods, an image-space rendering algorithm such as a ray-tracing algorithm enables a more straightforward way of foveated pixel sampling and shading. Weier et al. [34] cast rays through a set of pixels stochastically sampled using a piecewise-linear function approximating the hyperbolic falloff of visual acuity. The missing pixel colors in the screen were then reconstructed using extra information from a lower-resolution, support image or reprojected frames. Fujita and Harada [5] also distributed sample points on the image plane using a simple acuity function, and the ray-traced results from the k-nearest neighbor samples were blended for each image pixel. Siekawa et al. [27] used a triangular mesh generated from foveated ray samples, and reconstructed a final image by drawing the triangles. In developing a real-time foveated path tracer, Koskela et al. [13] improved the log-polar distribution to better reflect the visual acuity function. In their method, the ray sampling and denoising were performed in what is called the visual-polar space, and the resulting rendering image was mapped to a final image in Cartesian space. Recently, Peuhkurinen and Mikkonen [23] showed the possibility of foveated ray tracing for a mixed-reality headset, although only simple scenes were rendered.

While cost effective, the perception-driven, foveated rendering method inevitably introduces noticeable aliasing artifacts in the periphery due to insufficient sampling. Reducing temporal flickering is particularly necessary in the periphery because human peripheral vision is especially sensitive to the temporal alias. Smooth blending of shaded colors of neighboring pixels was basically used for filling in missing areas or resampling between different spaces. In addition, applying a low-cost temporal filter such as those developed for real-time path tracers, e.g., [12, 14, 25], was shown to be effective in reducing temporal flickering in the peripheral vision [13]. The idea of reprojection (also called image warping), whereby rendering information is recycled from previous frames (refer to such applications as in, e.g., stereoscopic rendering [26,37]), was also effectively applied in order to improve the perceived quality of rendering both within the rasterization [4,7] and ray-tracing [34] frameworks. Finally, a different approach of neural reconstruction was proposed by Kaplanyan et al. [11] to reduce visual artifacts in the periphery.

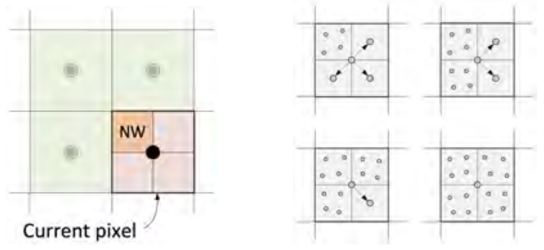
3 Selective supersampling for ray tracing

In this section, we briefly explain the selective adaptive supersampling technique that was proposed by Jin et al. [8] for efficient real-time ray tracing. The key idea of this adaptive pixel sampling method is to first gather both image- and object-space attributes of image pixels at their centers in a preliminary ray-tracing step and exploit them for determining where to take more ray samples in each pixel region based on the criteria selectively set with respect to a collection of rendering elements.

Table 1 lists the two groups of pixel attributes that were considered in their implementation. The first group consists of a single image-space attribute, which is the ray-traced, shaded color. This attribute is the most fundamental, in that it is referred in the *color disparity test* that finally decides if a subpixel area of pixel needs more ray samples. The second group of attributes are called the *geometry* *attributes* because they are collected at the surface locations in 3D space that are hit by rays. In order to balance between maximizing antialiasing effects and minimizing additional rendering costs, they collected the geometry attributes only at the first hit points of the primary rays of pixels and additionally, the surface points intersected by the secondary, reflection/refraction rays if they are generated at the first hits. Four different types of geometry attributes are gathered at each surface point as shown in Table 1, each of which is deeply related to the visual artifacts frequently generated by ray tracing.

Table 1: List of pixel attributes gathered during the presampling stage. By adjusting the threshold values, precious computing time can selectively and adaptively be distributed to desired rendering features for effective antialiasing during ray tracing.

	Sampling location	Attribute	Threshold
Image space	Pixel center	Color reference	τ_{col}
Object space	Primary ray hit point	Object ID	τ_{poid}
		Surface normal	τ_{psn}
		Shadow count	τ_{psc}
		Texture existence	τ_{pte}
	Secondary ray hit point	Object ID	τ_{soid}
		Surface normal	τ_{ssn}
		Shadow count	τ_{ssc}
		Texture existence	τ_{ste}



(a) Test against three neighboring pixels

(b) Adaptive supersampling according to the disparity test

Figure 2: Subpixel disparity test and adaptive pixel sampling. (a) For a subpixel, for instance, one marked by NW, the attributes of the pixel that it belongs to are compared with those of the three surrounding neighbor pixels. (b) Depending on the result of the test, extra rays are shot in the *problematic* subpixel area. Whereas the shaded color at the pixel center was simply transferred to the non-problematic subpixel in the original work of Jin et al. [8], our implementation used the color linearly interpolated from those of the four surrounding pixels, tending to reduce visual artifacts further.

For efficient implementation on a GPU, each image pixel was partitioned into four equal-sized subpixels, against each of which a two-step *disparity test* was carried out independently to decide if the corresponding subpixel region needs extra sampling. For a test over a given subpixel, the attributes of the pixel it belongs to were compared with those of three neighboring pixels whose locations are determined with respect to the subpixel's position (see Fig. 2(a)). Depending on the test result, the presampled pixel color was used for the current subpixel if no disparity was found, or four additional samples were taken in the problematic subpixel by firing corresponding rays otherwise (see Fig. 2(b)). After all extra ray-tracing computations were completed, the resulting shaded colors were blended with proper weights.

The key to this selective supersampling technique lies in the mechanism of the *two-step disparity test*. Before rendering starts, the user selectively and adaptively sets the threshold values in Table 1 according to the importance of respective rendering features to be reflected in the process of supersampling. During rendering, the first step determines the threshold τ_{sel} that will be actually employed in the color disparity test in the next step. For a given subpixel, feature-to-feature comparisons were first done between its pixel and three surrounding neighbors. When discordance is detected for at least one geometry attribute, τ_{sel} is set to the minimum value of the thresholds of the disagreeing attributes. Otherwise, τ_{col}, the threshold of the color reference attribute, is simply used for τ_{sel}. In the second step, the final color disparity test is carried out with respect to the color reference attribute using the selectively chosen threshold τ_{sel}, where the contrast measure, adopted by Mitchell [17], was used for this color comparison with threshold vector $(\tau_{sel} \cdot 1.36, \tau_{sel} \cdot 1.02, \tau_{sel} \cdot 2.04)$. Note that the two-step subpixel test enables the user to allocate relatively more computational effort to selectively chosen rendering features by setting their thresholds to smaller values, efficiently reducing the visual artifacts as intended to the extent that time permits.

4 Our Selective Foveated Ray Tracing Algorithm

The proposed ray-tracing scheme specially designed for high-quality HMD rendering partitions the image space into two parts, called the *central* and *peripheral pixel regions*. The central pixel region corresponds to the visual field of about 60 degrees in diameter from fixation, for which a sophisticated ray-tracing algorithm is applied to create a high-quality foveated image sampled with rate of up to 36 rays per pixel around the fixation point. On the other hand, our method guarantees the quality of the Whitted-style ray tracing taking one sample per pixel in the peripheral pixel region, where visual artifacts may occur due to insufficient pixel sampling, but they are hardly perceptible on the HMDs. Before presenting our ray-tracing scheme, we first describe how these two regions are defined.

4.1 Foveation-based image space partition

Light enters the eye through the pupil and projects onto the retina to create an image. The (anatomical) *macula* or *macula lutea* is an oval-shaped pigmented area near the center of the retina of the human eye, providing central, high-resolution color vision for humans [6]. Located in the center of the macula, the *fovea*, approximately 1.5 mm wide and corresponding to approximately 5.0° of the visual field, is filled with closely packed *cones*, which are photoreceptor cells on the retina surface linked to clear color vision under bright light. Most densely packed with the cone cells, the fovea area is responsible for the highest visual acuity in time and space. On the macula region, it is surrounded by the *parafovea* (approx. 2.5 mm/9.0°) and the *perifovea* (approx. 5.5 mm/17.0°). The retinal tissue extends beyond the macula where another type of photoreceptor, i.e., *rods*, provides peripheral vision, perceiving monochromatic shades of gray and movement in low-light conditions.

The density of cones is closely related to visual acuity, which is a measure of our ability to resolve small details. According to Curcio et al.'s study on photoreceptor cells [3], the peak foveal cone density falls off steeply on the macula with increasing visual eccentricity. While the foveal center provides the maximum resolving power for the eye, visual acuity is reduced by more than 75% at the boundary of the fovea. Special effort should thus be made to prevent aliasing in a ray-traced image in the pixel area corresponding to the foveal vision field. Peripheral vision is also a vital part of our vision, enabling us to sense objects and movement in the peripheral visual field even under dim light. However, central visual acuity is clearly more important than peripheral sensitivity when looking at a virtual world through various display devices, including head-mounted ones. Therefore, to achieve frame rates sufficient for the refresh rate of current VR displays (80 Hz or above), it is necessary to concentrate the computational effort for effective antialiasing in the central visual field while maintaining a minimum quality of rendering in the peripheral visual field.

Fig. 3 illustrates how the image space for each eye is subdivided in our work with respect to increasing eccentricity. First, the pixel regions corresponding to the fovea, parafovea, and perifovea in the macula are defined by the three eccentricity parameters: e_f, e_p, and e_m. According to the width of the three regions described above, they match 2.5°, 4.5°, and 8.5°, respectively. Note that the distribution of cone and rod cells on the retina is variable between individuals [3], and thus different sets of threshold values have been adopted in related work. For instance, in the design of foveated contact lens displays, Chen et al. [2] set e_f, e_p, and e_m to 3.3°, 5.5°, and 12.1°, respectively. Our foveated ray-tracing system also employs these conservative parameters (unless mentioned otherwise) to guarantee a minimum quality of ray tracing within the central visual field.

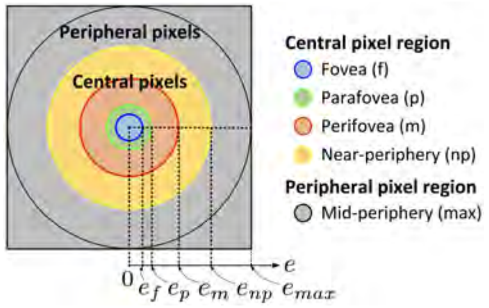


Figure 3: Partition of image space by visual eccentricity. For each eye of the HMD device, the image space spans the visual field defined by an eccentricity parameter e_{max}. In our foveated ray tracing, the *central pixel region*, on which computational effort is focused, conservatively includes the near periphery in addition to the fovea, parafovea, and perifovea of the macula. The remaining area in the image space for an eye is treated as the *peripheral pixel region*.

Second, among the remaining image pixels corresponding to the peripheral vision, those in the near periphery, determined by the eccentricity parameter e_{np}, are handled with more care for smooth transition from the central to peripheral zones. e_{np} is also controllable by a user where we find 30° (the visual field of 60° in diameter) to be sufficient for preventing perceptually annoying artifacts in near-peripheral vision. Note that for each eye, our ray-tracing system renders an image with a square field of view circumscribing the visual field determined by another eccentricity parameter e_{max}. For the Oculus Rift S headset used for experiment in this work, an OpenVR tool indicates that its field of view is 90° horizontally and 94° vertically [30]. Therefore, we set e_{max} to 45° for the full resolution 2560×1440 of the device, meaning the pixel resolution of each eye's image space for ray tracing is 1280×1440.

4.2 An overview of our algorithm

Fig. 4 presents an overview of our selective foveated ray-tracing pipeline, where stereo images are synthesized independently on a GPU or GPUs for input camera poses detected in current time frame. When the rendering starts for each eye, the image pixels are first sampled at their centers to gather the necessary pixel attributes as described in Section 3. Unlike the implementation of Jin et al. [8], we take an approach of hybrid ray tracing such as [1], combining deferred shading (Boxes A-1 and A-2) and ray tracing (Box A-3), where the OpenGL rendering pipeline speeds up the first two substeps markedly. Also, the shadow attributes are computed differently in the central and peripheral pixel regions, as will be described shortly. The obtained color reference attribute, which is the result of Whitted-style ray tracing with one ray sample per pixel, forms the rendering image for the peripheral pixel region. On the other hand, further processing continues to enhance the image quality in the central pixel region as follows. First, based upon the collected pixel attributes, the *pixel reusability test* is performed (Box B) for pixels in the central pixel region to speed up the rendering further, where the surface point visible through each pixel, that is, the first hit of each pixel in the scene, is back-projected to the image plane of the immediately preceding time frame, determining if the pixel color from the last frame may be reused for the current pixel without incurring costly ray-tracing computation. For pixels that pass the pixel reusability test (Box C), their colors are simply copied from those of the corresponding pixels in the previous frame.

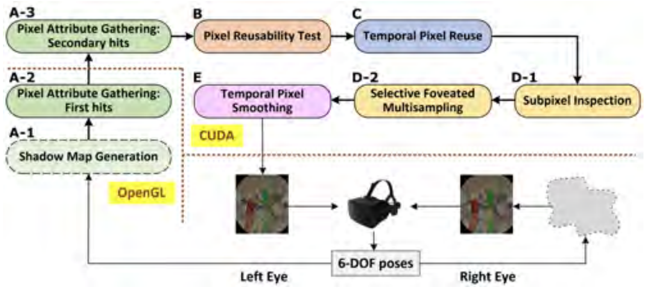


Figure 4: Our selective foveated ray tracing pipeline. The same ray tracing computation is performed independently for each eye.

By contrast, for each image pixel that fails the test, the selective foveated ray-tracing computation is carried out. First, the two-step disparity test is performed for each of the four subpixel areas as in [8] to find out if additional ray samples should be taken there (Box D-1). In this inspection process, however, we use a modified threshold in the color disparity test of the second substep so that the decision can be made *foveatedly*, relaxing the test condition gradually with increasing eccentricity. Next, extra rays are shot in a stratified jittered fashion in each problematic subpixel area, where a different number of ray samples per subpixel is employed in decreasing manner to each of the four-component areas of the central pixel region (Box D-2). Then, the shaded ray colors are accumulated with proper weights to their respective pixels to complete an image. In the final step, the shaded pixel colors of the image are blended with those of the preceding frames to reduce any possible temporal visual artifacts (Box E). For this temporal antialiasing, the current pixel is again back-projected to a few previous frames, and the colors of the corresponding pixels that satisfy a given condition are combined to get the final pixel color of the current frame.

4.3 Step 1: Pixel attribute gathering

In order to efficiently collect the pixel attributes at their centers, we perform a deferred-shading-assisted ray tracing in contrast to the method of Jin et al. [8], which relied on the standard recursive ray tracing. (Please see again the attributes handled in the selective ray tracing in Table 1.) The input 3D scene is first rendered using OpenGL with shadow mapping (Box A-2 in Fig. 4), storing into four texture images the intermediate pixel information that describes several rendering properties at the first hits of primary rays. In our implementation, the pixel color due to local illumination is stored in a uchar4 texture; the position (float3) and the `object ID` (float) of the visible point in a float4 texture; the `surface normal` (float3) and `shadow count` (float) in a float4 texture; and the reflection and transmission coefficients (float2), the refractive index (float), and the `texture existence` (float) in a float4 texture.

To refine the shadow from the OpenGL rendering and evaluate the shadow count in the central pixel region, shadow rays are fired toward lights from each first hit of the central pixels in a CUDA-based ray tracer (Box A-3). (The shadow count is needed for shadow antialiasing only for the pixels in the central region.) Also, if each first-hit surface of both central and peripheral pixels is either reflective or transmissive, the respective secondary ray is generated for recursive ray tracing, computing the radiance of incoming light (for both the central and peripheral pixels) as well as gathering the four geometry attributes at the secondary ray-hit points in additional two float4 textures (for the central pixels only). The results of these additional computations on CUDA are then reflected in the local colors, completing the `color reference` attributes of all pixels. Note that our hybrid ray tracing accelerates the pixel-attribute gathering process by handling the primary (for all pixels) and shadow (for peripheral pixels) rays on the efficient rasterization pipeline of OpenGL, where the shadow map generation is done in the preprocessing stage (Box A-1) only when the lighting parameters change. See Fig. 5.

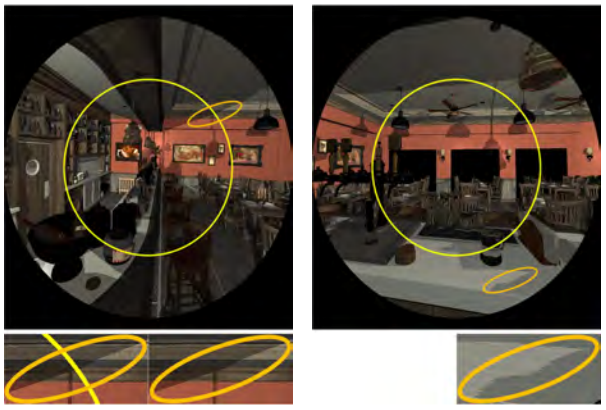

(a) Example 1 (b) Example 2

Figure 5: Hybrid generation of shadow effects (`Interior`). Two sample images of the color reference attribute collected with the eye fixated at the center of the screen are shown, where the yellow circle marks the boundary between the central and peripheral pixel regions. The shadows were generated by the CUDA ray tracer in the central region and by the OpenGL rasterization pipeline in the peripheral region. (a) Usually, the shadows in the two parts were well aligned to each other along the boundary circle, making it difficult to distinguish them from each other. (b) Aliasing in the shadows was occasionally observed in the peripheral pixel region due to the limited resolution of the shadow maps. However, we rarely recognized such aliases on the tested HMD, as they are outside the near-peripheral vision.

4.4 Step 2: Pixel reusability test

In the relatively less important peripheral pixel region, our renderer shades the image pixels using both forward rasterization, enabled with shadow mapping, plus backward reflection/refraction ray tracing with a sampling rate of one ray per pixel, which guarantees the minimum quality of foveated rendering. On the other hand, in the central pixel region, a sophisticated ray-tracing calculation is carried out, where the pixel sampling rate varies adaptively across the image area. In fact, despite our efforts for efficient ray tracing, the rendering in this inner area is inherently expensive, especially when many rays need to be generated per pixel.

Sometimes when a user wearing a headset stares at a fixed point in a 3D scene or moves his/her head slowly, the rendered colors of some image pixels are very similar to those of *matching* pixels between the current and immediately preceding frames. In order to reuse the previous pixel colors for the current frame as much as possible without incurring costly ray tracing, a CUDA kernel in the following stage (Box B in Fig. 4) builds a *pixel reusability map* that indicates which pixels of the current frame may simply take pixel colors from the previous frame. Each (i, j)-th element of the map consists of two fields: the pixel reuse flag f_{reuse} and the coordinates $p_{reuse} = (i_{reuse}, j_{reuse})$ of pixel to reuse. Here, f_{reuse} is set to zero if the ray tracing should be carried out for the current pixel. Otherwise, if the previous pixel color is reusable, it stores a positive number that shows how many times the pixel colors have been successively reused including the current pixel. In our method, we limit the number of successive reuses for a pixel to a fixed number n_{reuse}^{max} to avoid excessive approximation in rendering a pixel color in the central region (n_{reuse}^{max} is set to 5 in our current implementation). On the other hand, p_{reuse} are the coordinates of the pixel on the image plane of the immediately preceding frame, onto which the first hit $v_{cur} \in R^3$ of a pixel $p_{cur} = (i, j)$ in the current frame is back-projected. For each pixel p_{cur} in the central region, the *pixel reusability test* is carried out against the corresponding pixel p_{reuse}, which fails if at least one of the following conditions is not satisfied:

- The value of f_{reuse} of pixel p_{reuse} in the pixel reusability map of the immediately preceding frame is less than n_{reuse}^{max}.
- The primary ray for p_{cur} intersects an object in the scene.
- The primary ray for p_{reuse} intersects an object in the scene.
- The difference between the distances to v_{cur} from the respective cameras in the current and immediately preceding frames is less than a given threshold.
- The two color reference attributes of p_{cur} and p_{prev} from the respective frames pass the two-step disparity test (described in Sect. 3) against the adjusted threshold $\tau_{sel}^* = \tau_{reuse} \cdot \tau_{sel}$, where τ_{reuse} is a factor to control the degree of temporal pixel reuse.

Note that when all the conditions are met, the two first hits in the scene corresponding to p_{cur} and p_{reuse} in the two time frames are almost identical in terms of rendering, allowing pixel color reuse. If the test succeeds, f_{reuse} of p_{cur} in the current pixel reusability map is incremented by one from that of p_{reuse} in the preceding map. Otherwise, f_{reuse} of p_{cur} gets initialized to zero in the current map. In this case, the field of p_{reuse} is ignored.

4.5 Step 3-1: Temporal pixel reuse

After the pixel reusability map is generated, a separate CUDA kernel collectively copies the rendered colors of the matching pixels p_{reuse} from the immediately preceding frame to the current pixels p_{cur} with nonzero flag f_{reuse} (Box C in Fig. 4). Refer to Fig. 6 to understand how the temporal pixel reuse scheme works.

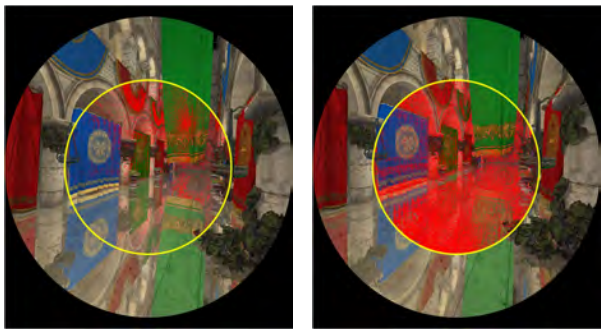
4.6 Step 3-2: Selective foveated ray tracing

Now, pixels in the central pixel region that may not reuse the rendering results from the previous frame are rendered by a selective foveated ray tracer, which performs a two-step computation (Boxes D-1 and D-2 in Fig. 4).

4.6.1 Subpixel inspection

In the first substep, the four subpixels of the pixel demanding actual rendering are examined to determine if additional sampling is necessary. As in Jin et al. [8], the same two-step disparity test is applied over each subpixel area. However, we extend the mechanism of determining the threshold value that is used in the color disparity test (τ_{sel} in the selective ray tracing), in order to take into consideration the acuity of human vision in the rendering of the central pixels. Several mathematical models, either linear or nonlinear, were proposed in the previous foveated rendering algorithms, which approximated the hyperbolic falloff of the acuity in the visual field. In our approach, such a perceptual characteristic of the eye is reflected indirectly by modifying the threshold vector in the final color comparison as $(\tau_{new} \cdot 1.36, \tau_{new} \cdot 1.02, \tau_{new} \cdot 2.04)$ with $\tau_{new} = \tau_{fov}(e) \cdot \tau_{sel}$. Here, for the pixel's eccentricity e with respect to the fixation point, $\tau_{fov}(e)$ is aimed at controlling the strictness of the color disparity test, eventually varying the pixel sampling rate adaptively over the eccentricity.

Given a set of shape parameters $(\tau_0, \tau_f, \tau_f', \tau_{np})$, the function $\tau_{fov}(e)$ is defined by a pair of C^1-continuous splines derived from



(a) Example 1 ($\tau_{reuse} = 0.01$) (b) Example 2 ($\tau_{reuse} = 0.05$)

Figure 6: Temporal pixel reuse for efficient rendering (`Crytek Sponza`). These snapshot images with similar views were captured while the user's gaze was moving slowly. Two different thresholds were applied to control the number of pixels that would be reused. By increasing the value of τ_{reuse}, the ratio of reused pixels, marked in red, increased accordingly, where 13.26% and 43.71% of the pixels in the central pixel region were reused in (a) and (b), respectively. When the rendering images generated with and without the temporal pixel reuse were quantitatively compared, the peak signal-to-noise ratio (PSNR) in dB was $(43.95, 47.46, 48.44, 54.24)$ and $(40.12, 40.34, 42.36, 45.18)$ in the four areas of the central pixel region (enumerated from the innermost one), respectively. While the PSNR values slightly dropped as more pixels were reused, the actual difference between them was very difficult to discern visually. The thresholds employed in the last two-step disparity test were $\tau_{col} = \tau_{pte} = \tau_{soid} = \tau_{ssn} = \tau_{ste} = 0.02$, $\tau_{poid} = 0.05$, and $\tau_{psn} = \tau_{psc} = \tau_{ssc} = 0.06$.

the two quadratic Bézier curves $C_1(t) = (x_1(t), y_1(t))$ and $C_2(t) = (x_2(t), y_2(t))$, satisfying the following constraints: $C_1(0) = (0, \tau_0)$, $C_1(1) = (e_f, \tau_f)$, $\frac{y_1'(0)}{x_1'(0)} = 0$, and $\frac{y_1'(1)}{x_1'(1)} = \tau_f'$ for $C_1(t)$, and $C_2(0) = (e_f, \tau_f)$, $C_2(1) = (e_{np}, \tau_{np})$, $\frac{y_2'(0)}{x_2'(0)} = \tau_f'$, and $\frac{y_2'(1)}{x_2'(1)} = 0$ for $C_2(t)$ (see Fig. 7). More specifically, τ_{fov} is numerically defined as $y_1(x_1^{-1}(e))$ for $e \in [0, e_f]$ or $y_2(x_2^{-1}(e))$ for $e \in (e_f, e_{np}]$, where the inverse functions can be evaluated by solving the respective quadratic equations.

4.6.2 Selective foveated multisampling

After the subpixel inspection is finished, extra sample rays are generated in a stratified manner for each subpixel that is deemed to be problematic. In contrast to Jin et al.'s work [8] that always took four samples per subpixel (refer to Fig. 2(b) again), our method allows a user to control the sampling density as a function of the eccentricity through the vector parameter $(n_{sd}^f, n_{sd}^p, n_{sd}^m, n_{sd}^{np})$, in which each component indicates the number of samples to be taken per a subpixel area in the respective region (see Fig. 3). Therefore, if the vector is set to $(9, 4, 2, 1)$, for instance, the pixel sampling rates become as effective as up to 36, 16, 8, and 4 spp in the fovea, parafovea, perifovea, and near periphery, respectively. When the extra sampling is finished for all problematic subpixels, the computed ray colors are then accumulated to the respective pixels with proper weights. Refer to Fig. 8 to see how effectively the image pixels in the central region are supersampled by our selective foveated ray tracer, in which the function of the temporal pixel reuse was turned off.

4.7 Step 4: Temporal pixel smoothing

Despite our efforts to achieve high effective sampling rates, temporal flickering between successive time frames is inevitable, especially in image areas with high spatiotemporal frequencies. The final step in our rendering pipeline reduces temporal aliasing artifacts by combining the rendered colors with those of preceding time frames (Box E in Fig. 4). For geometrically faithful temporal antialiasing, each pixel of a current frame is first back-projected onto the image planes of n_{tpi} immediately preceding frames, respectively (n_{tpi} is limited to 4 at most in our implementation). Every matching pixel color is then blended with the current one using the commonly used *exponential smoothing* technique [19, 24] only if the two corresponding pixels see a very close location in the 3D scene in that their respective distances to the first hits are within a given small threshold.

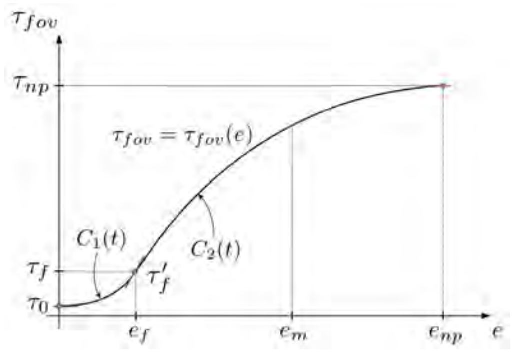
The (truncated) recursive formulation of exponential smoothing can be expressed in a closed form as $C_t^*(p) = \sum_{i=0}^{n_{tpi}} \alpha \cdot (1-\alpha)^i \cdot C(\pi_{t-i}(p)) \,/\, \sum_{i=0}^{n_{tpi}} \alpha \cdot (1-\alpha)^i$, where $C_t^*(p)$ is the color output at pixel p in time frame t, α the blending factor, and $C(\pi_{t-i}(p))$ the ray-traced color at the back-projected pixel $\pi_{t-i}(p)$ in time frame $t-i$ if the two corresponding depths are close enough or the null color otherwise. However, unlike the previous approaches that applied a fixed α across the image space, we use a blending factor that varies as a function of the visual eccentricity. Note that with a larger α, the antialiased pixel color depends more on the color of a current frame and less on those of older frames. Since the effective pixel sampling rate of our ray tracer decreases with increasing eccentricity, requiring more aggressive antialiasing to compensate for the relatively lower sampling rate, we usually set the blending factor to a monotonically decreasing function $\alpha(e)$ of eccentricity e.

5 Results

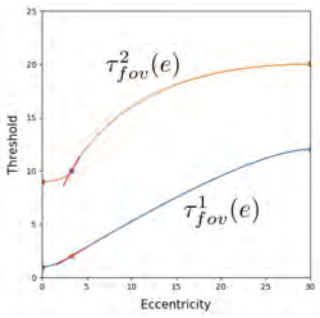
To evaluate the effectiveness of our rendering method, we first generated ground-truth images using a Whitted-style ray tracer that applied 36 stratified samples per pixel. Three ray tracers that adopted a simple regular sampling, the selective supersampling [8], and our selective foveated supersampling, respectively, were then tested. Four nontrivial 3D scenes `Crytek Sponza (CS)`, `Interior (IN)`, `Rungholt (RU)`, and `Power Plant (PP)`, respectively made of 279K, 1,021K, 6,704K, and 12,749K triangles, were used in the test, in which the kd-tree structure [32] was applied to accelerate the ray-tracing computation. All timings were measured on a PC with dual Nvidia GeForce RTX 2080 Ti GPUs, and included the total time taken for both rendering stereo images of 1280×1440 pixels per eye and displaying them on an Oculus Rift S headset.

Fig. 1 and Fig. 9 compare the rendering results from the three ray tracers against the ground truth with maximum ray bounce depth of 3, where the different pixel sampling strategies were applied inside the central pixel region while they all used one sample per pixel for ray tracing the peripheral pixel region. Then, Table 2 shows the statistics that reveal the quantitative behavior of the tested ray tracers. Clearly, our selective foveated ray tracer was highly efficient in that it processed far fewer rays than the selective-only ray tracer, leading to significant accelerations in the rendering computation. Nevertheless, the PSNR values in the table strongly indicate that our method provided rendering of a much better quality within the foveal field of view, which is an essential requirement for effective foveated rendering. Although the image quality of the selective foveated sampling was often rather lower than that of the selective-only ray tracer in the outer, perifoveal and near-peripheral visual fields, the visual differences were largely imperceptible on the tested HMD.

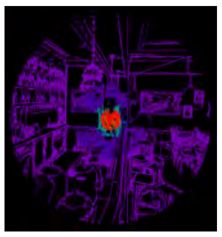
In the experiment reported in Table 2, the feature of temporal pixel reuse was turned off for fair comparisons with other methods. With this feature on, the rendering time clearly improved at only little loss of rendering quality. However, the degree of improvement varied depending on the situation. For instance, the `Crytek Sponza` scene was rendered for a camera view while the user's gaze was moving slowly, the rendering time reduced from 27.6 ms to 21.6 ms, in which the number of the total rays traced decreased from 8,502,017 to 8,434,997. On the other hand, the computation time for the `Interior` scene only decreased from 19.1 ms to 18.3 ms, even though far fewer rays were traced (5,158,702 versus 3,712,484). As noted in Fig. 6, the images produced with and without the temporal pixel reuse were visually indistinguishable, with the differences



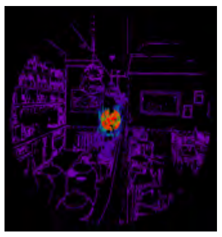
(a) Definition of $\tau_{fov}(e)$



(b) Two example functions

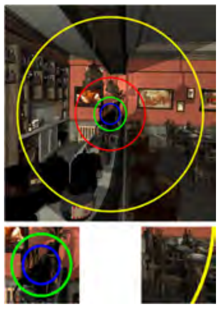


(c) Result of $\tau^1_{fov}(e)$

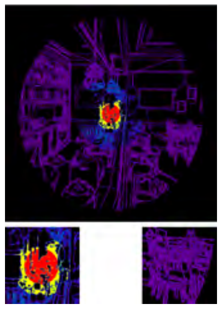


(d) Result of $\tau^2_{fov}(e)$

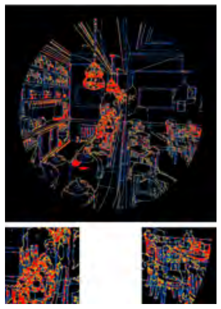
Figure 7: Reflection of visual acuity in the central pixel region (`Interior`). (a) By using two quadratic Bézier curves $C_1(t)$ and $C_2(t)$ defined by the shape parameters $(\tau_0, \tau_f, \tau'_f, \tau_{np})$, a user can control the shape of the function $\tau_{fov} = \tau_{fov}(e)$ with intuition and flexibility. Note that the smaller the function value, the higher the pixel sampling rate during ray tracing. When the two functions $\tau^1_{fov}(e)$ and $\tau^2_{fov}(e)$ in (b), respectively defined by $(1, 2, 0.5, 12)$ and $(9, 10, 1.5, 20)$, were applied for the foveated ray tracing, the supersampling pattern varied as expected, as shown in (c) and (d). Here, we used the control vector of $(n^f_{sd}, n^p_{sd}, n^m_{sd}, n^{np}_{sd}) = (9, 4, 2, 1)$, where the actual number of extra samples taken per pixel is shown in a rainbow scheme from 36 (red) through purple (1) to black (0).



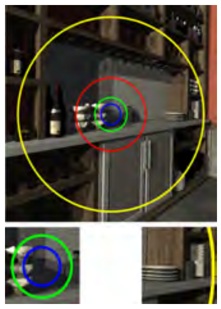
(a) Example 1



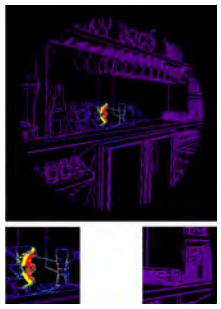
(b) Ours



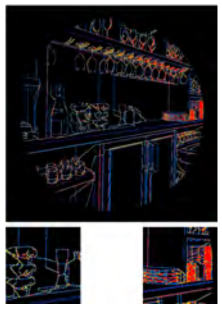
(c) Selective [8]



(d) Example 2



(e) Ours



(f) Selective [8]

Figure 8: Efficient distribution of ray samples (`Interior`). In these examples, we employed another control vector of $(n^f_{sd}, n^p_{sd}, n^m_{sd}, n^{np}_{sd}) = (4, 3, 2, 1)$, in which the number of extra samples taken per pixel is color-coded, decreasing from 16 (red) through purple (1) to black (0). To render the first image (a), our ray tracer sampled most fovea area at the maximum possible density because the innermost region had very detailed rendering features (b). Then the sampling rate progressively decreased with increasing eccentricity. On the other hand, the selective ray-tracing scheme [8], which always took four ray samples per subpixel without taking into account the eccentricity, often spent large computational effort in the outer region, which is less important perceptually (c). As also demonstrated in the second example (d), (e), and (f), our method automatically decided where to consume the precious ray resources both foveatedly and selectively, therefore sampling more only in the pixels with details even in the foveal region. Note that the selective ray tracing needed to process 6,335,413 and 4,487,153 extra rays in total (3.182 and 1.753 spp on average) in the central pixel region to render the two example images, respectively. By contrast, our method required only 2,865,734 and 2,270,141 extra rays (1.086 and 0.617 spp on average), respectively, which was not only markedly faster but also produced perceptually more effective images.

always exceeding 40 dB in PSNR in the central pixel region.

Finally, note that we had to carefully choose the blending factor α during the last stage of temporal pixel smoothing. It was important to preserve the high-quality rendering in the foveal area with a rather large α and to perform more aggressive smoothing in the peripheral area with a smaller α to reduce temporal noises due to relatively lower sampling rates. In particular, we found experimentally that the monotonically decreasing piecewise-linear function $\alpha(e)$, defined by $\alpha(0) = 0.9$, $\alpha(e_f) = 0.8$, $\alpha(e_p) = 0.7$, $\alpha(e_m) = 0.5$, $\alpha(e_{np}) = 0.2$, and $\alpha(e_{max}) = 0.1$, is an effective choice (see Table 3).

6 Concluding Remarks

In this paper, we presented a novel, real-time ray-tracing scheme specially designed for high-fidelity rendering on HMDs. Our method achieved a sufficiently high perceptual rendering quality in real time on commodity GPUs by cleverly deciding where to shoot more rays into the scene through the proposed selective foveated ray-sampling technique. As a result, our full Whitted-style foveated ray tracer that enables the creation of optically correct shadow, reflection and refraction provided high sampling rates as effective as 36 spp around the fixation point, decreasing gradually to 1 spp in the outermost peripheral vision.

The described foveated ray tracer is now limited to static geometry mainly because its CUDA-based ray-tracing part adopts the kd-tree [32] as an acceleration structure. For dynamic lighting that requires rebuilding of the OpenGL shadow maps, shadow in the peripheral pixel region could be generated by the CUDA-based ray tracer, although it would lower the frame rate slightly. We are now extending our renderer to support dynamic geometry using the ray-tracing hardware (RT Cores) in the latest Nvidia GPUs [20]. Based on the bounding volume hierarchy, which is better suited to dynamic change of geometry [31], the dedicated hardware will make available more CUDA processing power for extra rendering computations.

Acknowledgments

The test scenes are courtesy of Frank Meinl (`Crytek Sponza`), Amazon Lumberyard (`Interior`), kescha (`Rungholt`), and the UNC GAMMA group (`Power Plant`). This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. NRF-2020R1A2C2011709).

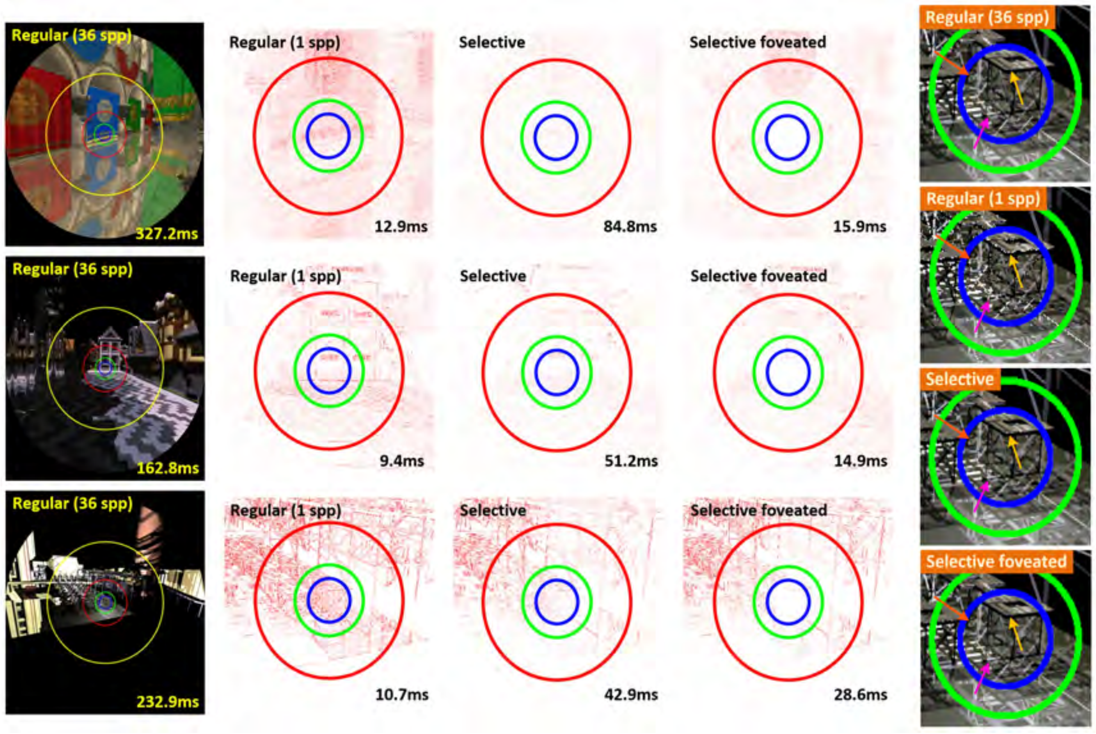


Figure 9: Comparison of rendering results inside the central pixel region (Continued from Fig. 1). More comparisons for the `Crytek Sponza`, `Rungholt`, and `Power Plant` scenes are given (from top to down in the left). In addition, the four magnified images from `Power Plant` (in the right column) compare the three pixel sampling methods to the ground-truth ray tracer that applied 36 rays per pixel for rendering. While the selective-only sampling technique provided effective sampling rates as high as up to 16 spp, subtle aliases were found occasionally in the foveal visual field (indicated by arrows), which was frequently bothersome on an HMD. By contrast, our method produced a ray-tracing result in the foveal area that compared very favorably with that of the ground-truth renderer. In summary, due to serious visual artifacts, ray tracing that traces one ray per pixel was clearly unsuitable as a rendering method for HMDs. On the other hand, the presented selective foveated ray-tracing method was highly competitive in terms of both perceptual quality and rendering speed. (Please refer to Fig. 1 and Table 2 again.)

Table 2: Quantitative comparison of four ray-tracing methods. The three ray tracers are analyzed with respect to the ground-truth renderer, where the camera views used for the four test scenes are shown in Fig. 1 and Fig. 9. Again, "Time" shows the total time for generating stereo images on the Oculus Rift S headset. "Rays" indicates the total number of rays (in thousands) that were traced while rendering the stereo images. For an explanation on "PSNR," please refer to Fig. 1.

		Regular_36	Regular_1	Selective	Sel_For
CS	Time (ms)	327.2	12.9	84.8	15.9
	Rays (K)	426,806	11,856	133,762	7,400
	PSNR (dB)	–	**24.0**/25.6	**36.4**/37.3	**47.5**/36.2
			25.4/28.6	36.9/38.8	32.3/33.2
IN	Time (ms)	221.3	10.3	72.4	16.9
	Rays (K)	288,873	8,024	104,196	4,199
	PSNR (dB)	–	**29.4**/27.5	**38.3**/36.8	**46.0**/36.7
			29.8/32.4	39.3/41.9	37.6/35.8
RU	Time (ms)	162.8	9.4	51.2	14.9
	Rays (K)	329,086	9,142	88,287	7,570
	PSNR (dB)	–	**23.8**/25.1	**31.3**/34.4	**41.8**/34.8
			27.8/30.6	35.3/37.9	34.4/33.8
PP	Time (ms)	232.9	10.7	42.9	28.6
	Rays (K)	230,250	6,395	27,799	8,928
	PSNR (dB)	–	**18.8**/20.2	**28.6**/28.1	**35.2**/29.3
			20.7/20.4	27.3/25.9	28.3/23.9

Table 3: Foveated selection of blending factor α for temporal pixel smoothing (PSNR in dB). Temporal antialiasing is essential for HMD rendering even if a user's gaze remains still because of the temporal noises in received HMD pose signals. The ray-traced images obtained with and without temporal antialiasing for a view of the `Crytek Sponza` scene in a given time frame are compared with the ground truths produced by tracing 256 rays per pixel. "Off" indicates the case when the temporal pixel smoothing step was turned off in our rendering pipeline. When a fixed α was used ("0.1" to "0.9"), the effectiveness of the antialiasing effort varied in different areas. Note that a small α was needed to sufficiently reduce temporal noises in the peripheral areas but tended to excessively blur the high quality of rendering in the foveal area. Meanwhile, our foveated choice of α ("Ours") automatically adapted the temporal antialiasing process across the image area to preserve the quality of rendering. Through temporal pixel smoothing, the spatial quality of the image actually improved slightly and the temporal flickering was reduced markedly, as expected.

Region	Off	0.1	0.3	0.5	0.7	0.9	Ours
Fovea	40.53	37.82	38.83	40.20	**41.33**	41.09	41.33
Parafovea	36.74	36.11	36.60	37.11	**37.33**	36.96	37.25
Perifovea	34.11	33.59	33.82	34.04	**34.14**	34.04	34.09
Near-peri.	31.73	32.57	**32.69**	**32.69**	32.46	31.97	32.74
Mid-peri.	32.07	**33.63**	33.58	33.34	32.91	32.34	33.63

References

[1] C. Barré-Brisebois, H. Halén, G. Wihlidal, A. Lauritzen, J. Bekkers, T. Stachowiak, and J. Andersson. Hybrid rendering for real-time ray tracing. In E. Haines and T. Akenine-Möller, eds., *Ray Tracing Gems*, chap. 25, pp. 437–473. Apress, 2019.

[2] J. Chen, L. Mi, C. P. Chen, H. Liu, J. Jiang, and W. Zhang. Design of foveated contact lens display for augmented reality. *Opt Express*, 27(26):38204–38219, 2019.

[3] C. A. Curcio, K. R. Sloan, R. E. Kalina, and A. E. Hendrickson. Human photoreceptor topography. *Journal of Comparative Neurology*, 292(4):497–523, 1990.

[4] L. Franke, L. Fink, J. Martschinke, K. Selgrad, and M. Stamminger. Time-warped foveated rendering for virtual reality headsets. *Computer Graphics Forum*, 40(1):110–123, 2021.

[5] M. Fujita and T. Harada. Foveated real-time ray tracing for virtual reality headset. Technical report, Light Transport Entertainment Research, 2014.

[6] E. B. Goldstein, ed. *Encyclopedia of Perception*. SAGE Publications, Inc., 2009.

[7] B. Guenter, M. Finch, S. Drucker, D. Tan, and J. Snyder. Foveated 3D graphics. *ACM Transactions on Graphics*, 31(6):164:1–164:10, 2012.

[8] B. Jin, I. Ihm, B. Chang, C. Park, W. Lee, and S. Jung. Selective and adaptive supersampling for real-time ray tracing. In *Proceedings of High Performance Graphics*, pp. 117–125, 2009.

[9] J. T. Kajiya. The rendering equation. *ACM SIGGRAPH Computer Graphics*, 20(4):143–150, 1986.

[10] J. T. Kajiya and B. P. Von Herzen. Ray tracing volume densities. *ACM SIGGRAPH Computer Graphics*, 18(3):165–174, 1984.

[11] A. S. Kaplanyan, A. Sochenov, T. Leimkühler, M. Okunev, T. Goodall, and G. Rufo. DeepFovea: Neural reconstruction for foveated rendering and video compression using learned statistics of natural videos. *ACM Transactions on Graphics*, 38(6):212:1–212:13, 2019.

[12] M. Koskela, K. Immonen, M. Mäkitalo, A. Foi, T. Viitanen, P. Jääskeläinen, H. Kultala, and J. Takala. Blockwise multi-order feature regression for real-time path-tracing reconstruction. *ACM Transactions on Graphics*, 38(5):138:1–138:14, 2019.

[13] M. Koskela, A. Lotvonen, M. Mäkitalo, P. Kivi, T. Viitanen, and P. Jääskeläinen. Foveated real-time path tracing in visual-polar space. In *Proceedings of Eurographics Symposium on Rendering*, 2019.

[14] A. Kuznetsov, N. K. Kalantari, and R. Ramamoorthi. Deep adaptive sampling for low sample count rendering. *Computer Graphics Forum*, 37(4):35–44, 2018.

[15] M. Levoy and R. Whitaker. Gaze-directed volume rendering. In *Proceedings of the Symposium on Interactive 3D Graphics*, pp. 217–223, 1990.

[16] X. Meng, R. Du, M. Zwicker, and A. Varshney. Kernel foveated rendering. *Proceedings of the ACM on Computer Graphics and Interactive Techniques*, 1(1):5:1–5:20, 2018.

[17] D. Mitchell. Generating antialiased images at low sampling densities. In *Proceedings of SIGGRAPH 1987*, pp. 65–72, 1987.

[18] H. A. Murphy, A. T. Duchowski, and R. A. Tyrrell. Hybrid image/model-based gaze-contingent rendering. *ACM Transactions on Applied Perception*, 5(4):1–21, 2009.

[19] D. Nehab, P. V. Sander, J. Lawrence, N. Tatarchuk, and J. R. Isidoro. Accelerating real-time shading with reverse reprojection caching. In *Proceedings of the 22nd ACM SIGGRAPH/EUROGRAPHICS Symposium on Graphics Hardware*, pp. 25–35, 2007.

[20] NVIDIA. NVIDIA Ampere GA102 GPU Architecture: Second-Generation RTX. Whitepaper, 2021.

[21] T. Ohshima, H. Yamamoto, and H. Tamura. Gaze-directed adaptive rendering for interacting with virtual space. In *Proceedings of the Virtual Reality Annual International Symposium*, pp. 103–110, 1996.

[22] A. Patney, M. Salvi, J. Kim, A. Kaplanyan, C. Wyman, N. Benty, D. Luebke, and A. Lefohn. Towards foveated rendering for gaze-tracked virtual reality. *ACM Transactions on Graphics*, 35(6):179:1–179:12, 2016.

[23] A. Peuhkurinen and T. Mikkonen. Real-time human eye resolution ray tracing in mixed reality. In *Proceedings of VISIGRAPP (Poster)*, pp. 169–176. 2021.

[24] D. Scherzer, S. Jeschke, and M. Wimmer. Pixel-correct shadow maps with temporal reprojection and shadow test confidence. In *Proceedings of the 18th Eurographics Conference on Rendering Techniques*, pp. 45–50, 2007.

[25] C. Schied, C. Peters, and C. Dachsbacher. Gradient estimation for real-time adaptive temporal filtering. *Proceedings of the ACM on Computer Graphics and Interactive Techniques*, 1(2):24:1–24:16, 2018.

[26] A. Schollmeyer, S. Schneegans, S. Beck, A. Steed, and B. Froehlich. Efficient hybrid image warping for high frame-rate stereoscopic rendering. *IEEE Transactions on Visualization and Computer Graphics*, 23(4):1332–1341, 2017.

[27] A. Siekawa, M. Chwesiuk, R. Mantiuk, and R. Piórkowski. Foveated ray tracing for VR headsets. In *MMM 2019: MultiMedia Modeling*, pp. 106–117. Springer, 2019.

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

[29] 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*, pp. 9–18. 2014.

[30] Valve Software. IVRSystem::GetProjectionRaw. `https://github.com/ValveSoftware/openvr/wiki/IVRSystem::GetProjectionRaw/`, 2017. [Online; accessed 1-April-2021].

[31] I. Wald. On fast construction of SAH-based bounding volume hierarchies. In *Proceedings of the IEEE Symposium on Interactive Ray Tracing*, pp. 33–40, 2007.

[32] I. Wald and V. Havran. On building fast Kd-trees for ray tracing, and on doing that in O(Nlog N). In *Proceedings of the IEEE Symposium on Interactive Ray Tracing*, pp. 61–69, 2006.

[33] B. Watson, N. Walker, and L. F. Hodges. Supra-threshold control of peripheral LOD. *ACM Transactions on Graphics*, 23(3):750–759, 2004.

[34] M. Weier, T. Roth, E. Kruijff, A. Hinkenjann, A. Pérard-Gayot, P. Slusallek, and Y. Li. Foveated real-time ray tracing for head-mounted displays. *Computer Graphics Forum*, 35(7):289–298, 2016.

[35] 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. *Computer Graphics Forum*, 36(2):611–643, 2017.

[36] T. Whitted. An improved model for shaded display. *Communications of the ACM*, 23(6):343–349, 1980.

[37] N. Wißmann, M. Mišiak, A. Fuhrmann, and M. E. Latoschik. Accelerated stereo rendering with hybrid reprojection-based rasterization and adaptive ray-tracing. In *Proceedings of the IEEE Conference on Virtual Reality and 3D User Interfaces*, pp. 828–835, 2020.