Pacific Graphics 2016
E. Grinspun, B. Bickel, and Y. Dobashi
(Guest Editors)

Volume 35 (2016), Number 7

Foveated Real-Time Ray Tracing for Head-Mounted Displays

Martin Weier[1,2], Thorsten Roth[1,5], Ernst Kruijff[1], André Hinkenjann[1], Arsène Pérard-Gayot[2,3], Philipp Slusallek[2,3,4], Yongmin Li[5]

[1]Bonn-Rhein-Sieg University of Applied Sciences, [2]Saarland University, [3]Intel Visual Computing Institute,
[4]German Research Center for Artificial Intelligence (DFKI), [5]Brunel University London

(a) Small foveal region with ($r_0 = 5°, r_1 = 10°, p_{min} = 0.01$)

(b) Medium foveal region with ($r_0 = 10°, r_1 = 20°, p_{min} = 0.05$)

(c) Full Renderer

Figure 1: Images generated by using our foveated renderer showing the effect of different configurations for the foveal region, including an image that was rendered by ray tracing every pixel.

Abstract

Head-mounted displays with dense pixel arrays used for virtual reality applications require high frame rates and low latency rendering. This forms a challenging use case for any rendering approach. In addition to its ability of generating realistic images, ray tracing offers a number of distinct advantages, but has been held back mainly by its performance. In this paper, we present an approach that significantly improves image generation performance of ray tracing. This is done by combining foveated rendering based on eye tracking with reprojection rendering using previous frames in order to drastically reduce the number of new image samples per frame. To reproject samples a coarse geometry is reconstructed from a G-Buffer. Possible errors introduced by this reprojection as well as parts that are critical to the perception are scheduled for resampling. Additionally, a coarse color buffer is used to provide an initial image, refined smoothly by more samples were needed. Evaluations and user tests show that our method achieves real-time frame rates, while visual differences compared to fully rendered images are hardly perceivable. As a result, we can ray trace non-trivial static scenes for the Oculus DK2 HMD at 1182×1464 *per eye within the the VSync limits without perceived visual differences.*

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Ray Tracing, Virtual Reality;

1. Introduction

A long-envisioned dream in virtual reality is the ability to present a real or imagined world in a compelling and convincing way. Advancements in display technologies and mass production have introduced a range of high-quality head-mounted displays (HMDs) with a wide field of view (FOV), gaining interest from researchers in the field of virtual and augmented reality. Thereby, pixel densities have dramatically increased over the two last decades (e.g. *Forte VFX 3D* at $263 \times 480 \times 2$ in 1997 and *StarVR* at $2560 \times 1440 \times 2$ in 2016). However, providing the highest possible visual quality at retinal resolution would require about $32k \times 24k = 768M$ pixels for the full dynamic field of view of the human eye (200° horizontally and 150° vertically) [Hun15]. Rendering at such resolutions interactively is far beyond reach of current and foreseeable hardware and software solutions. Besides rendering convincing imagery for high pixel densities, high update rates are crucial for limiting potential motion sickness. The combination of high pixel densities and update rates are a major challenge when bringing image synthesis algorithms to HMDs.

This challenge can be approached by a dynamic adaptation of rendering based on the user's gaze (foveated rendering). This technique exploits the limitations of the human eye by omitting details in the peripheral visual area that are largely imperceptible, as only our *central vision* affords high visual acuity. Thus, we can degrade

rendering quality with increasing *eccentricity* (angular distance to the user's gaze in the image). The parts of the eye responsible for *central vision* consist of the fovea (5.2° in diameter around the central optical axis), parafovea (approximately 5.2° to 9°) and perifovea (approximately 9° to 17°). Peripheral vision refers to the area outside of central vision [Wan95]. While the fovea affords high visual acuity, it rapidly drops with increasing distance to this area as the number of cones is subject to a hyperbolic falloff. In contrast, the number of rods is decreasing slower, a property that leads to a high sensitivity to spatial and temporal brightness and contrast changes in the peripheral area that needs to be regarded when developing new techniques [LK87, GP98, MD07]. While other issues like colors, patterns, shapes [LMR*12], and motion [MN84] also influence perception, we particularly looked at brightness and contrasts. Furthermore, visual attention has been shown to affect perception as it can cause visual tunneling, where details outside the point of fixation are largely ignored [Miu86].

While so far rendering for HMDs is mainly based on rasterization because of performance considerations, ray tracing has several advantages when it comes to stereo rendering, wide fields of view, correcting chromatic aberrations and low latency rendering [Hun15]. In this work, we mainly focus on its ability of distributing samples freely on the screen and the inherent possibility of creating high-quality renderings, crucial to achieve a good immersion. Yet, ray tracing has been mainly held back by its performance as it is challenging to achieve the same performance as rasterization without specific hardware acceleration. In this paper, we introduce a novel ray-tracing-based foveated rendering system capable of rendering high-quality images fast enough for modern HMDs. We reduce the sample density by adapting the ray generation to the foveal receptor density. To improve perception, there is a need for foveated rendering methods that reconstruct images at a high quality to limit the detection of visual artifacts. Using our approach, missing information from the sampling process can either be reconstructed using a *support image* that is guaranteed to sample the full scene using a lower uniform resolution or by using information from reprojected frames to improve the quality of the reconstructed final image. Note that there is no explicit handling of view-dependent effects like reflections in our system, which is generally challenging for reprojection-based methods. While the quality of such effects may suffer due to the reprojection process, the influence on the perceived quality in a system like ours still has to be analyzed. Nonetheless, our benchmarks demonstrate the high performance of our implementation when compared to standard ray tracing. We also conducted a user study using an Oculus Rift DK2 equipped with an eye tracker. This made it possible to substantiate the high visual quality provided by our method for static scenes and revealed significant effects of the foveal region's size and visual attention. The latter can greatly optimize foveated rendering performance, as fewer samples need to be generated in the periphery when users concentrate on a specific part of the scene.

In summary, we present an interactive foveated ray tracing system for HMDs that includes the following **contributions**:

- A high-performance, adaptive sampling approach for ray tracing driven by eye tracking and limitations of human perception.
- A reprojecting and merging process using a coarse approximation of the scene geometry to aid reconstruction of the final image from sparse samples.
- A user-study showing that our method only has minimal impact on the perceived quality when regarding foveal region limits. The study also reveals a great potential of deploying visual attention to further optimize foveated rendering techniques.

2. Related Work

In the following section, we provide an overview of the main fields of research related to our system: Foveated rendering and temporal rendering optimization techniques. Early work in foveated rendering has exploited the perceptual deficits of the human visual system without eye tracking, making use of focus assumptions [FS93] or using models of visual attention, e.g. [YPG01]. Such systems, however, have limited applicability as the user's exact focus cannot be predicted. This requires eye tracking, which has been deployed in general within the domain of 3D graphics [ODH02], as well as specifically for gaze-contingent 2D displays [RLMS03, Dc07]. An early foveated rendering system for volume rendering was developed by [LW90], using a multi-resolution technique to alter the resolution and sampling rates. Other techniques have been used for HMDs, e.g., in [WWH04], where geometric level-of-detail based on eccentricity is introduced. Another approach to foveated rendering was introduced in [GFD*12]. Their system employs raster graphics to render three layers with different sampling rates blended into a final image. Rasterization only allows for a fixed resolution, making a continuous change of the sampling rate inefficient. However, in [SGEM16] a deferred shading system is introduced that creates a stochastic sampling pattern to adaptively select the rasterized fragments for shading based on the user's gaze - shading is the most computationally demanding part of modern image synthesis algorithms. Moreover, they also point out the importance of contrast and brightness perception by specifically shading samples with a high saliency and contrasts.

However, ray-based methods are better-suited for sampling the adaptive patterns needed for foveated rendering not only limited to shading. An approach introduced in [MD07] uses a precomputed mesh specifying ray locations for sampling and increasing the sampling density near object silhouettes and regions with high contrast. However, their method cannot accelerate rendering and does not reproject samples over time to improve image quality. One approach to foveated ray tracing was presented in [FH14]. A precomputed sampling pattern together with a kNN scheme is used to reconstruct the image from sparse samples. However, the proposed system still shows artifacts, does not consider the eye's sensitivity to contrasts, and a user study is missing. In contrast, we apply a temporal caching and resampling scheme that enables us to cope with increasing resolutions and improve the quality of the reconstruction specifically for regions that expose high contrasts and silhouettes. Finally, [PBNG15] adapt rendering by exploiting lens astigmatism in HMDs, which leads to a decrease in image quality towards outer regions. Their system deploys ray tracing on the CPU combined with rasterization to reduce the sampling rate for areas outside the lens center. However, the system does not adapt to the user's gaze or the visualized content.

To speed up rendering and perform image reconstruction we ex-

ploit techniques that reuse data temporally. Early approaches in this field include the RenderCache [WDP99] and the Holodeck ray cache [WS99]. However, these techniques either require dense sampling or higher order representations like Voronoi regions or a spherical Delaunay mesh [Sim00] to reconstruct a full image, which makes them computationally costly. To overcome this, the Tapestry system [SS00], the Shading Cache [TPWG02], and the system by Jeschke et al. [JW02] accumulate samples in a 3D mesh and perform hardware-rasterization to reproject and reconstruct the scene. However, such a mesh does not eliminate all the artifacts: Geometric edges have to be reconstructed using a large number of point samples. Adaptive Frameless rendering [DWWL05] uses 2D kD-trees for sample selection and a splatting process for image reconstruction. Since the splat's size rendered to the image must be overestimated taking a neighborhood of splats into account, it elicits artifacts and is computationally demanding for high samples counts. Despite sparsely sampling an image the reuse of samples for anti-aliasing by utilizing reprojection has been proposed in [NSL*07]. Especially important is the question of how samples can be combined temporally using a running estimate. Yang et al. [YNS*09] improved the mathematical models behind such an estimate with the goal of reducing blur and temporal artifacts. A popular image space post-processing technique for HMDs hiding rendering latencies is Time Warping [MMB97]. The rendered image is shifted and distorted just before display to compensate for orientation changes. However, due to occlusion it only works for rotations and does not help with translations.

3. Foveated Real-time Ray Tracing

In this section we describe the building blocks of our foveated rendering system. An overview of the entire pipeline is presented in Figure 2. The system's core is a fast ray tracer based on NVIDIA CUDA using an SBVH acceleration structure [SFD09] (Figure 2, Block 2). It generates a sampling pattern from three parameters describing a foveal region to account for the user's perception and gaze (see Section 3.1). However, this foveated sampling process results in a sparse image. This means that with increasing eccentricities we get proportionally larger gaps between sampled pixels. Presenting such an image to the user would not meet the perceptual requirements, as the gaps would result in the sparse image's brightness being vastly different from the fully sampled image. Also, strong temporal flickering would be present due to the stochastic sampling process used to generate rays. Thus, it is necessary to provide a method that improves image quality outside the foveal region, generating a smooth image from the sparse samples. This method has to meet certain requirements: While performance has to be high enough to stay within VSync limits (13.3ms at 75Hz), image quality has to suffice human perception.

Therefore, the ray tracer is coupled to a reprojection scheme (Figure 2, Block 1), providing additional information to improve image quality. First, a coarse depth mesh is reconstructed from the scene and rendered from a new perspective used to reproject samples (see Section 3.2). Second, parts of the image that are poorly reprojected or critical to peripheral vision are detected (see Section 3.3). These are marked for resampling in an auxiliary buffer referred to as *resampling info*.

Finally, information is stored inside a cache and a final image is constructed (Figure 2, Block 3). Missing samples can either be reconstructed by the reprojected previous frame or by low-resolution color and G-buffers (*support image* and *support G-buffer*) which are updated completely per frame (see Section 3.4). The resolution of these buffers is only a fraction of the target resolution required for the HMD. The *support image* contains a regular low-resolution color image, while the *support G-buffer* contains the geometric normals and depth values that are later used to reconstruct the coarse geometry for the next frame. An optional post-processing step (Figure 2, Block 4) can further improve image quality when stochastic sampling processes are used. (see Section 3.5). Each of the pipeline's steps will be described in the following sections.

3.1. Ray Generation and Ray Tracing

In the beginning, a ray generation kernel is launched which creates rays with a density proportional to a foveal falloff function. In addition, rays are generated for all pixels that are either marked in the *resampling info* or belong to the *support image's* resolution. The generated rays are intersected with the scene geometry resulting in a list of hit points and their respective pixel indices. The kernel introduced by Aila and Laine [ALK12] is employed for ray traversal, extended to speed up the evaluation of fully transparent alpha values in the innermost loop.

After the intersections, a kernel computes shaded pixel colors for the rays' hit points. Shading currently supports Phong lighting with mipmapping, ambient occlusion, point, and area light sources. Besides computing the shaded pixel color, the irradiance from the area light sources and an ambient occlusion factor for diffuse surfaces are stored in a separate *light buffer*. This way the running estimate, used to combine samples temporally, can be adapted to different rates for color and lighting information, allowing to reduce noise introduced by sampling area light sources or ambient occlusion. Furthermore, these irradiance values can be spatially reused between neighboring pixels (see Section 3.5).

Visual acuity is subject to a hyperbolic falloff with increasing eccentricity (distance to the visual center) [GFD*12, SRJ11]. However, we chose to use a model with a linear falloff for the sampling probability, as the increased sampling rate compared to a hyperbolic falloff allows for better motion perception in the periphery. Also, a higher sampling rate reduces spatial as well as temporal aliasing artifacts in these areas that are critical due to the high flickering sensitivity at larger angular distances to the center of vision. Moreover, a linear model also matches visual acuity well for small angles [GFD*12]. We refer to this model as the *foveal function*, which is illustrated in Figure 3.

To achieve a linear behavior, ray generation is based on two user-defined *eccentricity* thresholds: An inner threshold r_0 and an outer threshold r_1, both given in degrees in the visual field. r_0 determines the size of the foveal region (i.e., the area rendered at full detail), while r_1 together with the minimum sampling probability p_{min} determines the probability falloff beyond r_0. All three values together are referred to as a *foveal region configuration* (FRC). Pixels with a larger eccentricity than r_1 are only sampled with a probability of p_{min}. Generally, pixels are only queued for sampling if $\xi_q < p(q)$

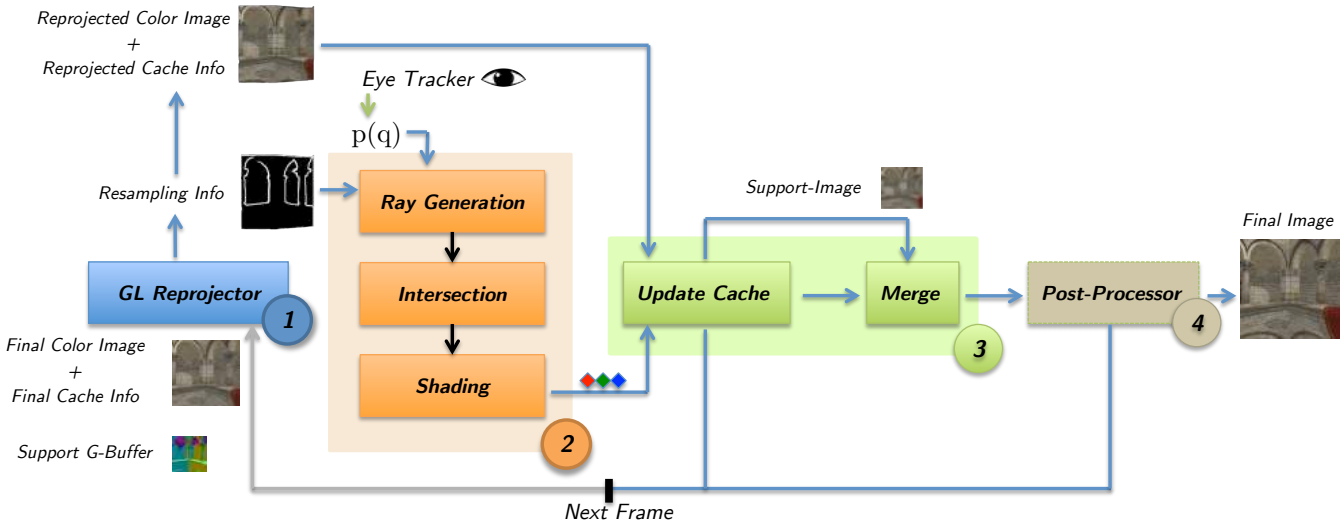

Figure 2: Building blocks of our reprojection pipeline. Old *color* and auxiliary *cache info* are reprojected using a *GL Reprojector* (Block 1). New rays are generated based on the user's gaze and possible errors introduced by the reprojection are marked in a *resampling info* buffer. The ray traced pixel values (Block 2) are blended using a temporal caching and merging scheme (Block 3). An optional Post-Processor is used to smooth artifacts arising from stochastic sampling processes like ambient occlusion (Block 4).

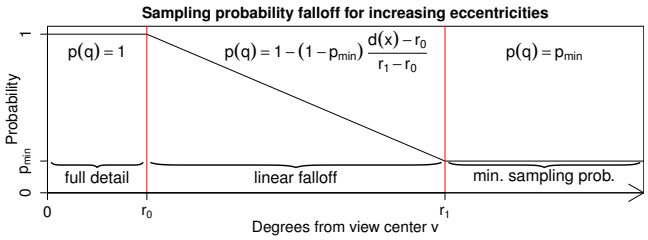

Figure 3: Probability $p(q)$ for sampling a specific pixel q based on its eccentricity and user-defined parameters r_0, r_1 and p_{min}. Despite the hyperbolic falloff of cones towards outer regions a linear falloff is employed to improve motion perception in the periphery and to reduce spatial and temporal aliasing artifacts in these areas.

with $\xi_q \in [0,1]$ being a uniformly distributed random number. If a pixel is needed for the *support image* or is marked in the *resampling info*, it is always scheduled for sampling.

3.2. Reprojection

After computing an array of newly shaded samples along with the pixel indices for the current frame, our reconstruction relies on information from the previous frame. As computing more samples is expensive, reusing information from previous frames helps to increase performance while it can also improve image quality. To create a perspective-correct reprojection of information from the previous frame the scene geometry must be taken into account. Possible approaches include the reprojection methods in [SS00,TPWG02], constructing and updating an irregular mesh with potentially as many vertices as pixels at the highest quality level. However, such approaches are costly. Ray tracing is fast (up to 200 Mrays/sec for traversal [ALK12]) and expensive methods do not pay off. Hence, we decided to use a reprojection strategy based on a coarse uniform mesh. Reprojection errors resulting from this geometry are resolved by computing additional samples for these regions instead of constructing a more precise mesh representation. The reprojection process (Figure 2, Block 1) transforms the *final color image* along with the *final cache info* to a new *reprojected color image* and *reprojected cache info* buffer. The *cache info* buffers are use to keep a state in the buffer that maintains how samples should be combined temporally. As described in Section 3.4, this buffer is a `float4` texture that is reprojected along with the color information. For each frame, a uniform mesh is generated from the *support G-Buffer's* by creating and displacing a uniform grid of vertices matching the *support G-Buffer's* resolution.

To reconstruct the scene geometry from the depth information stored in the *support G-buffer*, the vertices are adjusted to these image space depth values with a geometry shader. Afterwards, an "unprojecting" step is performed using the model, view, and projection (MVP) matrices of the previous frame. These were also used to create the *support G-buffer*. This yields the vertex positions in world space representing the surface of the visible scene geometry from the previous frame. In the next step the vertices are projected to the new frame using the current *MVP* matrices. This mesh is rasterized at the full rendering resolution, textured with the last frame's *final color image* and *final cache info*, finally yielding the reprojected version of the previous frame.

Due to the changed perspective, each pixel's footprint may cover a couple of texels of the previous frame's *final color image*. Therefore, simple bilinear filtering of the texture is not sufficient and special care has to be taken to filter this texture during rendering. As computing a mipmap hierarchy along with anisotropic filtering for the reprojected texture per frame based on the new pixel's footprint

is too expensive, another texture filtering method has to be used. We chose to randomly sample the pixel footprint multiple times using a normal distribution inside the fragment shader to compute the final reprojected color. Another option would be to use MSAA and oversample each pixel's extent. However, additional geometry samples computed by MSAA are not necessary as they do not improve the quality of the reprojection.

3.3. Handling Reprojection Errors

The uniform mesh employed for reprojection is not a perfect representation of the actual scene geometry. This may lead to perceptible errors. Possible causes for these errors include geometry newly entering the view frustum, disocclusions, and undersampling [MMB97]. If a part of the scene has not been inside the view frustum in the previous frame and sampling has not been triggered by the foveal function, missing pixels are reconstructed from the coarse *support image* (see Section 3.4). Disocclusions and undersampling can both cause strong visual artifacts to appear in the image (see Figure 1a). This is caused by incorrectly or incompletely reprojected information. The coarse sampling adds new correctly computed samples on top, which also adds noise. Therefore, we try to detect and create additional samples for those areas, consequently improving perceived image quality.

First, to detect regions that need further sampling, the scene is rendered using the coarse resolution matching the *support image* and *support G-Buffer* using the reprojection procedure described in Section 3.2. If there is a depth or luminance difference between a pixel and its direct neighborhood in the reprojected image that is larger than a user-defined threshold ε_{depth} or ε_{lum}, a pixel is marked for resampling in the *resampling info*. This process resembles methods like SMAA [JESG12].

Depending on the chosen value for ε, it may happen that geometry that does not resemble the scene is used for reprojection anyway, e.g., in case of relatively flat objects in front of a wall. If such geometry is looked at frontally in frame $t-1$, moving the camera in frame t can result in undersampling artifacts because the possibly wrongly closed geometry is reconstructed, connecting the object to the wall. These objects might expose depth and luminance distances well below the respective ε-thresholds, while the closed geometry resulting from the reprojection process is actually wrong [MMB97]. Such surfaces occur along the user's viewing direction, i.e., the angle between the surface normal and the observer is close to 90°. We detect such artifacts with an additional test looking at the surface geometry. From the previous frame's geometric normal $\vec{n}$ and camera orientation $\vec{d}$, we compute $\text{edge}_t = \max\{\vec{n}\cdot\vec{d}, 0\}$. If $\text{edge}_t < \varepsilon$, the pixel is marked for ray generation. Partial derivatives of texture coordinates would be another measure to detect regions that need sampling. They yield information about an observer's angle towards a potentially undersampled surface. However, we have not found noticeable visual enhancements by using this information as head movements are limited when wearing an HMD. In case of complex geometry, a huge part of the image may be covered by possibly undetected and thus undersampled edges (see Figure 1a). This necessitates a measure for sample quality accounting for a sample's age, as presented in the next section.

3.4. Cache Update and Merging

At this point, the current image consists of the previous frame's *reprojected color image* (see Figure 2). Newly shaded samples from the ray tracing process have to be combined with this cache image using a temporal blending method. This accumulation process should be designed in a way that reduces the weight of older samples, as simply accumulating samples with equal weights does not make sense for two reasons: First, due to the sparse sampling process, each pixel may have been sampled last at a different point in time. Second, assigning a high weight to old samples leads to visual artifacts like smearing on edges. However, at the same time just using the new sample without considering cached values can lead to disturbing temporal noise, especially because of the human eye's high peripheral flickering sensitivity. This is caused by correctly reprojected regions having temporally varying color values due to the sampling process. To overcome this, we chose to apply a smooth temporal blending process with a limit to the samples' age. This age is used to give higher weights to samples that should be combined with old samples. While such a process reduces temporal flickering, large-scale contrast for the visual periphery is preserved.

A sample's color is written to the output if it belongs to the foveal region, is part of the resampling process (marked in the *resampling info*) or is written to a part of the image that did not contain any reprojected color due to disocclusion or movement. Moreover, the *support image* and *support G-buffer* are extracted as well (see Figure 2). For all other samples, geometry is accounted for by looking at the depth difference between $c_{t-1}(q)$ and $s_t(q)$ with $c_t(q)$ referring to a cached pixel with index q, cached at frame t, while $s_t(q)$ is the newly computed sample at time t for pixel index q. If this value is above a threshold ε the reprojection contains an error, as the ray has hit a part of the scene different from the cache. Therefore, we take the newly generated sample without considering anything from the cache. If the depth difference is below the threshold, the cached color values at pixel index q can be combined with the new sample s. The new pixel value $s'_t(q)$ is now computed using a blending value $\alpha_t(q)$:

$$s'_t(q) = \alpha_t(q)\cdot s_t(q) + (1-\alpha_t(q))\cdot s_{t-1}(q) \tag{1}$$

To account for the issues mentioned above, $\alpha_t(q)$ has to be adjusted according to the number of samples accumulated at pixel position q as well as the most recent update-time of this pixel. To do this, $\alpha'_t(q)$ is computed as:

$$\alpha'_t(q) = \frac{1}{\rho_{t-1}(q)+1} \qquad \rho_t(q) = \left(\alpha_t^2 + \frac{(1-\alpha_t)^2}{\rho_{t-1}(q)}\right)^{-1},$$

where $\rho_t(q)$ represents the number of samples that have been accumulated at pixel index q at frame t [YNS*09]. To avoid infinite accumulation of samples, k is the minimum possible weight for the new sample to finally compute $\alpha_t(q) = \min\{\alpha'_t, k\}$.

In contrast to [YNS*09], we do not sample each pixel in every frame. Therefore, it is best to adapt k dynamically based on a sample's age. If a pixel has been sampled a couple of frames ago, it has undergone the potentially imprecise reprojection process multiple times, especially since the camera is constantly moving when head tracking is active. If the time span between the previous update and the current time is high, it is better to account for the current sam-

ple with a higher weight. Therefore, instead of a fixed k we use the following exponential function:

$$k_t(\Delta t) = \min\left\{\exp\left(x_0 + \frac{\Delta t - 1}{t_{max} - 1}(x_1 - x_0)\right), k_{max}\right\},$$

with $x_0 = \ln k_{min}$ and $x_1 = \ln k_{max}$. It can be parameterized based on a fixed interval $[k_{min}, k_{max}]$ and the maximum time span t_{max} we allow for accumulating samples. $\Delta t = t - t_{touched}$ is the difference of the current frame index and the frame index a value has last been touched and updated in the cache. t_{max} is the user specified maximum number of frames between two samples. Computing k this way poses a trade-off: t_{max} should be chosen according to the refresh rate and in a way that resolves possible artifacts as early as possible by giving the new sample a higher weight. At the same time, weighting older samples relatively high guarantees a smooth temporal transition and reduces flickering. An additional *cache information* buffer is used to keep track of $\alpha_t(q)$, $\rho_t(q)$ and $t_{touched}$, stored per pixel along with the *color image*. However, to have these estimates available in the next frame it is necessary to reproject this buffer to the new perspective the same way as it is done for the color values described in Section 3.2. Eventually, another kernel is launched to *merge* the images with the *support image* (see Figure 2, Block 3). Since both the reprojection process and foveated sampling can fail for parts that have not been in the view frustum for frame $t-1$, the *support image* is used to fill in all missing information.

3.5. Post-Processing

When rendering scenes with stochastic sampling processes (e.g., for area lights or ambient occlusion), the resampling process presented above leads to a discrepancy in convergence for reconstructed and resampled image regions, as the latter do not consider any cache information. This leads to a visual difference between noise in these areas, appearing as high-frequency temporal flickering - almost exclusively caused by the stochastic processes. An optional post-processing filter (see Figure 2, Block 4) can be applied to resampled regions marked in the *resampling info* to reduce noise. For each pixel q in such a region, the nearest reconstructed (i.e., non-resampled) neighbor along the horizontal and vertical axis on the image plane is searched. The distance to this neighbor is then used to create a search window which is randomly sampled n times. This process then selects the closest reconstructed pixel r found during the sampling step and applies the information stored in the *light buffer* for r to the noisy pixel q.

4. Evaluation

Below, we describe the performance of our system and analyze the perceived visual quality of our methods.

4.1. Benchmarks

The hardware configuration for the performance benchmarks consisted of an Intel Core i7-3820 CPU, 64GiB of RAM and a NVIDIA GeForce Titan X driving a Oculus Rift DK2. Using the Oculus SDK, we determined the FOV for a single eye and computed the projection matrix. Rendering was done at a resolution of 1182×1464 per eye. Table 1 lists the benchmark results of flythroughs with 1000 frames each. We decided to use the parameters $(r_0 = 10^\circ, r_1 = 20^\circ, p_{min} = 0.05)$ to configure the user's foveal region. As shown in Section 4.2, users were mostly unable to detect any visual differences to the full renderer for this FRC. For the benchmark process, the foveal region was statically positioned at the image center. A resolution of 256×318 was selected for the *support image* and *support G-buffer*. This was chosen empirically as it provided a good trade-off between speed and quality for the used HMD and scenes. The four following test scenes were selected: *Sibenik*, *Sponza*, *Rungholt*, *Urban Sprawl* (see Figure 5). Each of the scenes was rendered with one point light source, an area light source with 8 samples per pixel (spp), and ambient occlusion using 16 spp.

Table 1 shows that the speed-up of our foveated ray tracer compared to a full ray tracer scales well with increasing ray workloads, as it reduces the number of rays. Hence, the smallest speed-up of 1.46 is achieved for rendering the scene *Urban Sprawl* with a single point light, while the maximum speed-up of 4.18 is achieved for *Sponza* with ambient occlusion (see Table 1). It also shows the time required for reprojection, cache update, merging, and the post-processing step. The latter are nearly constant for all scenarios, as they are mainly tied to the rendering and *support buffer* resolutions. The influence of the FRC on the rendering performance measured in FPS for *Sponza* is illustrated in Figure 5.

Even though rasterization is inherently different from ray tracing, we provide a few numbers for comparison to state-of-the-art approaches employing this method. In [GFD*12], rasterizing the image in three layers with different resolutions yields a speedup of 6.2 with only 7% of the pixels being rendered as the images are strongly undersampled. To still achieve an acceptable visual quality, this method needs to rely on specific anti-aliasing methods, limiting its applicability [SGEM16]. Moreover, numbers are only reported for a single scene. By using NVIDIAs Multi-Resolution Shading [Ree15] a speedup of 1.3 to 2 is reported depending on the configuration. Stengel et al. [SGEM16] report a speedup of 1.34 on average, with the number of shaded pixels being decreased by 65% for a resolution of 1280×1440 pixels and 83% for twice as many pixels. Our method has shown a reduction of sampled pixels by 79% on average for all benchmark scenes, with an average speedup of 2.55. The ray-based approach [FH14] report similar frame rates compared to our approach even though they use different and more GPUs rendering at a lower resolution. The performance of our method could be further improved by generating rays that directly match the image distortion of the HMD, making it possible to cope with even higher resolutions and refresh rates.

4.2. User Study

The user study addressed the perceived visual quality of our method. It was driven by the following research questions:

- RQ1: Can subjects differentiate between scenes with varying graphical contexts, rendered with and without our foveated rendering method?
- RQ2: Do modifications of the foveal region parameters in the ray generation have an effect on the perceived visual quality?

(a) Sponza (b) Rungholt (c) Tunnel_Geom

(d) Sibenik (e) Urban Sprawl (f) Tunnel_Maps

Figure 4: Scenes used for benchmarks and user studies of our implementations

	Scene Type	# of rays		Re-project	Ray Tracing		Cache Update	Merge	Post	Total Time		Speed-Up Factor
		full	ours		full	ours				full	ours	
Sibenik	point light	3.46M	0.81M	1.41	10.74	3.85	0.64	0.84	0.00	10.74	6.74	1.59
75K Triangles	area light (8spp)	15.57M	3.64M	1.51	47.96	14.80	0.67	0.85	0.33	47.96	18.36	2.61
Figure. 5a	ao (16spp)	29.42M	6.88M	1.45	94.64	27.96	0.64	0.83	0.32	94.64	31.39	3.02
Sponza	point light	3.46M	0.64M	1.41	13.93	4.43	0.58	0.84	0.00	13.93	7.26	1.92
154K Triangles	area light (8spp)	15.54M	2.89M	1.44	81.61	20.07	0.58	0.83	0.24	81.61	23.17	3.52
Figure 5b	ao (16spp)	29.36M	5.45M	1.43	179.01	39.79	0.58	0.83	0.24	179.01	42.87	4.18
Rungholt	point light	2.90M	0.58M	1.38	9.51	3.54	0.59	0.83	0.00	9.51	6.34	1.50
6704K Triangles	area light (8spp)	11.10M	2.25M	1.38	34.20	9.98	0.59	0.83	0.19	34.20	12.97	2.64
Figure 5c	ao (16spp)	20.47M	4.17M	1.39	168.03	51.15	0.59	0.83	0.19	168.03	54.14	3.10
Urban Sprawl	point light	3.06M	0.64M	1.37	8.97	3.33	0.60	0.83	0.00	8.97	6.12	1.46
773K Triangles	area light (8spp)	12.34M	2.60M	1.36	29.67	8.68	0.60	0.83	0.28	29.67	11.75	2.52
Figure 4e	ao (16spp)	22.95M	4.85M	1.39	110.11	30.02	0.60	0.83	0.29	110.11	33.13	3.32

Table 1: Times in ms for each stage of the pipeline in comparison to a full renderer showing the speed-up of our approach. Times and speed-ups are computed for a medium sized foveal region with ($r_0 = 10^\circ$, $r_1 = 20^\circ$, $p_{min} = 0.05$) for a single eye with a resolution of 1182×1464 and no oversampling on an NVIDIA GeForce Titan X. For the chosen foveal region, users were mostly unable to detect any visual difference to full rendering in the user study.

- RQ3: Does the fixation type have an effect on the perceived visual quality?

4.2.1. Experimental Procedure and Design

The experiment was conducted as a within-subject study [FH03], employing a $4 \times 4 \times 3$ full factorial design. Each participant completed 96 trials in randomized order, consisting of a full factorial combination of four scenes {*Sponza*, *Tunnel_geom*, *Tunnel_maps*, *Rungholt*} (see Figure 5), four FRCs {small, medium, large, full} (described below) and three fixation types {fixed, moving, free}. All conditions were presented twice. Full ray tracing was included as the FRC *full*, representing our control group. Each trial consisted of an 8-second-flight with one factor combination.

RQ1: We varied the test scenes to study the effect of graphical contexts on the noticed visual differences (artifacts). The selection of scenes was driven by perceptual differences of the peripheral visual field as opposed to central vision, including colors, patterns, shapes [LMR*12] and contrasts in near peripheral field [LK87]. *Sponza* represents the most real-world-like scene: While some discontinuities (and thus hard edges) are usually visible, the scene also contains some smoother curves resembling real objects. *Tunnel_geom* contains a tunnel consisting of noisy, displaced geometry. Depending on the point of view, this scene can contain both hard edges and smooth, continuous surfaces. *Tunnel_maps* is a tunnel textured with a checkerboard map and a noise texture. *Rungholt* is a scene generated from a Minecraft map with many visible depth discontinuities, which can be challenging for our reprojection method.

RQ2: The FRCs were given by the following eccentricity thresholds and minimum sampling probabilities: *small* $(r_0 = 5^\circ, r_1 = 10^\circ, p_{min} = 0.01)$, *medium* $(10^\circ, 20^\circ, 0.05)$, *large* $(15^\circ, 30^\circ, 0.1)$ and *full* $(\infty, \infty, 1)$. FRCs were determined by using the angular

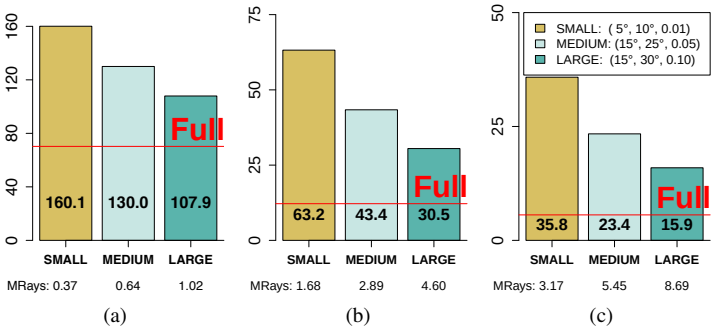

Figure 5: Influence of configuration of the foveal regions on FPS for the scene Sponza, rendered with a point light source (a), an area light source with 8spp (b) and with ambient occlusion with 16spp (c). MRays denotes the mean number of rays per frame. The scene was rendered at a resolution of 1182×1464 using a NVIDIA GeForce Titan X.

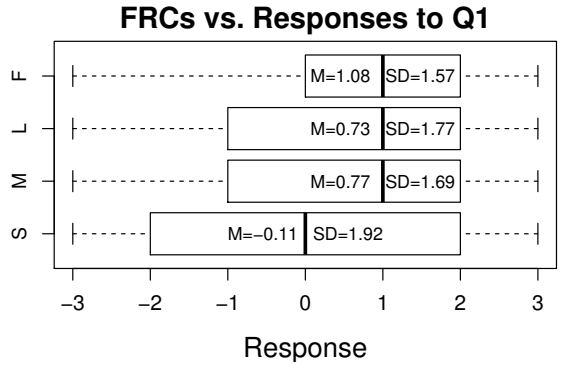

Figure 6: Likert-scale ratings of perceived visual artifacts for **S**mall, **M**edium, **L**arge, and **F**ull foveal regions. While *small* caused neutral ratings on average, ratings for *medium* and *large* were not significantly different from *full* rendering (cf. Section 4.2.2, RQ1).

size of the fovea for r_0 with a steep falloff for the smallest setting and increasing the foveal region and minimum sampling probability while reducing steepness for the other settings. The smallest FRC was expected to yield visible artifacts for most participants, as the foveal region used for rendering matched the *fovea*. The *medium* and *large* FRC extended the foveal region to include the *parafovea* and *perifovea*, respectively.

RQ3: Finally, while eye tracking determines a user's focus point in the scene (defining the foveal region), fixation may affect visual attention, potentially leading to visual tunneling effects [Miu86]. We varied fixation types to trigger different levels of visual attention. The *fixed focus* mode contained a static fixation cross at the image center to be focused for the entire trial. For the *moving target* mode, a set of paths across the image plane was generated. A green sphere linearly moved along these paths, fitted to the according scene depth to support the user's ability to focus. Paths varied in all trials except repetitions to avoid learning effects. The foveal region for *fixed focus* and *moving target* was centered around the fixation target. The moving target fixation mode was expected to cause higher visual tunneling as the user had to concentrate on following the target. It was also selected to avoid the negative influence of the eye tracker's inaccuracies (relatively low refresh rate, inaccurate tracking towards outer display regions). Trials with *free focus fixation* enabled the user to look around freely with the foveal region following the user's gaze.

The setup used for the user study differed from the benchmark configuration. It comprised an Oculus Rift DK2 (SDK 0.8) on a Windows 10 system including an Intel Xeon E5-2609 (2.4GHz), and 64GiB of RAM. The DK2's native refresh rate of 75Hz was used as the baseline for our user study. As both the foveated rendering and the OpenGL-based reprojection process had to be parallelized in order to achieve this frame rate, it was necessary to deploy two Quadro K6000 cards. These were required because the unavailability of a Linux driver for the utilized eye tracker tied us to Windows, which does not allow for multi-GPU rendering on NVIDIA's consumer cards. The Oculus was equipped with an SMI binocular eye tracker running at 60Hz (asynchronous). The rendering resolution was equal to the benchmarks with one image being rendered for each eye. With some minor optimizations, we achieved frame rates of at least 75Hz for all scenes, including the final image warping to display them in the Oculus. However, all sequences used for full ray tracing had to be pre-recorded (excluding any optimization like reprojection), loaded at runtime, augmented with the specific trial fixations and displayed at 75Hz.

After signing informed consent and receiving instructions, participants were seated and equipped with the HMD. Prior to the main experiment, six test trials of an alternative flight through *Sponza* were shown, including the smallest FRC, full rendering and all fixation types. After each main trial the participant rated if the shown sequence was free of visual artifacts (Q1) as well as the level of confidence for the given answer (Q2), both on a 7-point Likert scale from *strongly disagree* (-3) to *strongly agree* (3).

4.2.2. Results and Discussion

15 subjects (10 male/5 female, all with academic background) aged between 26 and 51 ($M = 33$, $SD = 7.24$), with normal or corrected-to-normal vision participated in the user study. A multifactor analysis of variance (ANOVA) [FH03] was performed on the data (1440 trials). We analyzed significant interactions and the observed main effects with post-hoc t-tests using Holm's method for p-value adjustment. Confidence values (Q2) were mostly high ($M = 1.62$, $SD = 1.14$), with very small differences. Consequently, we do not consider confidence in our analysis any further.

RQ1: Differentiation between foveated and non-foveated rendering. Mostly, users cannot reliably differentiate between full and foveated rendering. This is the case for foveal regions not smaller than roughly 10°, and scenes without too much high-frequency geometry. Figure 7 shows responses for varying FRCs, grouped by scenes. However, statistical data reveals that differentiation depends significantly on all the test variables: FRC size, fixation mode and the displayed scene. While FRC shows a significant main effect ($F \approx 30.54$, $p \approx 0$), there has also been strong inter-

action between FRC and SCENE ($F \approx 3.09$, $p < 0.005$). Hence, we performed t-tests, which showed that significant differences between the *medium*, *large*, and *full* FRC were only present in the scene *Tunnel_Maps*. All other scenes only showed significant differences when the *small* FRC was involved. We mainly attribute this to the regular rather extreme high-contrast checkerboard pattern shown in *Tunnel_Maps*.

RQ2: The effect of foveal region scales. As noted (RQ1), if the foveal region is not too small, users will hardly notice visual artifacts for most scenes.Figure 6 shows the responses for varying FRCs, including the mean values and standard deviations. The small FRC scored significantly lower, while medium and large FRCs were almost identical regarding perceived visual artifacts. The difference to full rendering was limited to a larger standard deviation. As Figure 7 illustrates, this can again be mainly attributed to the artifacts visible in *Tunnel_Maps*.

RQ3: The effect of fixation types. Fixation types, associated with different levels of visual attention, had a significant main effect ($F = 3.46$, $p = 0.03$) on the perceived visual quality. While *free* and *fixed* modes showed nearly identical responses (($M = 0.43, SD = 1.89$) and ($M = 0.43, SD = 1.81$)), the *moving target* was rated significantly better ($M = 0.99, SD = 1.63$). Thus, fewer visual artifacts were noticed with presumed higher visual attention of the moving target, as users were likely less aware of details outside the focus area [Miu86]. This is highly interesting as it could further reduce the sampling rate outside the foveal region - targeted in future work. Furthermore, the foveal region matched the gaze when the target was perfectly followed. We assume visual tunneling did not affect the other fixation modes, which made it easier to spot artifacts in the peripheral.

As the eye tracking system suffers from inaccuracies in outer image regions, we filtered the logged data used for analyzing the tracking information to only include the region utilized by SMI's calibration method. This region extends to maximum eccentricities of approximately 10.3° left/right and 11.68° up/down. The numbers were taken from the SMI SDK's 9-point calibration method and converted to angles. We analyzed angular differences between the fixation point and the tracked gaze. Participants stayed closer to the fixation point for the fixed mode ($M = 0.31^\circ$, $SD = 0.4^\circ$) than for the moving target ($M = 1.9^\circ$, $SD = 1.52^\circ$). Keeping in mind that *Tunnel_Maps* had the greatest amount of visible artifacts for the participants, it is important to mention that the median angular differences for *Sponza*, *Tunnel_Geom* and *Rungholt* were between 1.25° and 1.58°, while for *Tunnel_Maps* a median difference of 2.24° was found. As this larger distance to the foveal region's center indicates that the gaze was closer to sparsely sampled regions, it is one explanation for the relatively low Likert ratings for this scene. There were no significant differences between angular differences for varying FRCs. The median values over all scenes for the four FRCs were all within $[1.62^\circ, 1.7^\circ]$ for the moving target and $[0.22^\circ, 0.25^\circ]$ for the fixed mode.

5. Conclusion

In this paper, we presented a novel approach for foveated rendering using adaptive ray tracing and reprojection from previous frames.

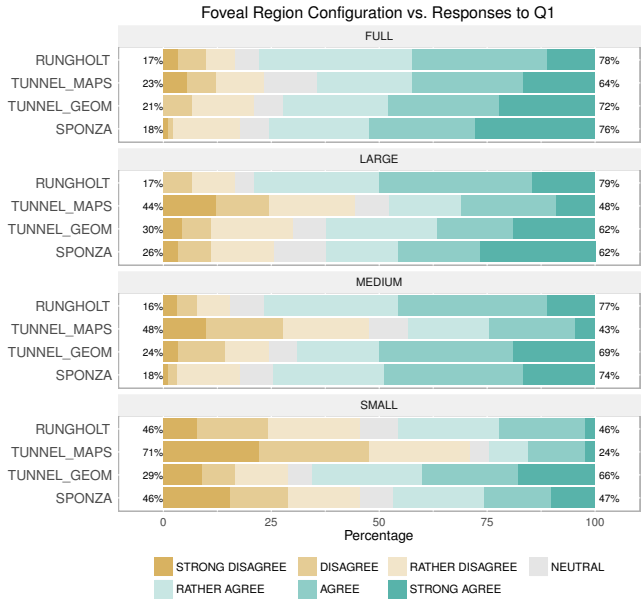

Figure 7: Likert-scale ratings for Q1 (*Has the shown sequence been free of visual artifacts?*) for all scenes grouped by foveal region configurations. The percentages on the left and right represent the fraction of all participants that had a tendency towards *disagree* and *agree*, respectively. As clearly visible, the smallest foveal region configuration revealed a significant amount of artifacts, while the larger foveal regions were close to the full renderer in this regard.

Our method is well-suited for wide-FOV HMDs equipped with eye tracking. Sparsely sampled image data is reprojected to new views using a depth mesh generated from a low-resolution G-Buffer. The influence of errors arising from reprojection or regions critical for perception is lowered by an update strategy that allows for resampling critical image regions incorporating the samples' quality. Our method enables the visualization of static scenes with millions of triangles within the Oculus Rift DK2 at a refresh rate of 75Hz. The benchmarks have shown significant performance gains, while the user study revealed the perceived visual quality for even moderately sized FRCs is almost on-par with full rendering. Anti-aliasing is crucial to rendering for HMDs: Pixels are distributed over a large FOV, making single pixels visible. Therefore, jagged edges and undersampling may easily become visible. Our strategy is to shoot more rays by jittering the ray positions over the pixels' extent. To improve the reprojected image's quality and handle errors while accounting for the user's gaze, each pixel that is either part of the foveal region or marked in the *resampling info* is sampled by shooting a ray. The results are combined using the running estimate. We are curious to evaluate how our system performs with moving objects, highly glossy materials, and dynamic light sources, also in the context of stochastic global illumination methods. In general, strongly view-dependent effects do not perform well with reprojection techniques. However, ray tracing gives us the advantage of being able to (re-)sample individual pixels efficiently. This could be used to include view-dependent effects in the resampling process. It has to be kept in mind that in extreme cases this could lead to a fully sampled image, bringing performance below the necessary refresh

rates. Moreover, we plan to dynamically alter the foveal falloff and minimum sampling probability in the peripheral region, making our system adaptive to the complexity of the visualized content. Finally, an interesting field for further research is frameless rendering: The reprojection and rendering components could run in separate threads, asynchronously generating/merging new samples and resolving reprojection errors. To conclude, we are confident that our developed approach is an important step in the process of making realistic real-time ray tracing suitable for head-mounted devices.

Acknowledgments

We would like to thank NVIDIA for providing us with two Quadro K6000 graphics cards as well as Intel Visual Computing Institute, the European Union (EU) for the co-funding as part of the Dreamspace and the FIWARE projects and the German Federal Ministry for Economic Affairs and Energy (BMWi) for funding the MATEDIS ZIM-project (grant no KF2644109)

References

[ALK12] AILA T., LAINE S., KARRAS T.: *Understanding the Efficiency of Ray Traversal on GPUs - Kepler and Fermi Addendum*. Technical Report NVR-2012-02, NVIDIA, 2012. HPG2012 poster. 3, 4

[Dc07] DUCHOWSKI A. T., ÇÖLTEKIN A.: Foveated Gaze-contingent Displays for Peripheral LOD Management, 3D Visualization, and Stereo Imaging. *ACM Trans. Multimedia Comput. Commun. Appl. 3*, 4 (Dec. 2007), 6:1–6:18. 2

[DWWL05] DAYAL A., WOOLLEY C., WATSON B., LUEBKE D.: Adaptive Frameless Rendering. In *SIGGRAPH 2005 Courses* (2005), ACM. 3

[FH03] FIELD A., HOLE G.: *How to Design and Report Experiments*. Sage publications Limited, 2003. 7, 8

[FH14] FUJITA M., HARADA T.: Foveated Real-Time Ray Tracing for Virtual Reality Headset, May 2014. Poster Siggraph Asia. 2, 6

[FS93] FUNKHOUSER T. A., SÉQUIN C. H.: Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments. In *Computer graphics and interactive techniques* (1993), ACM, pp. 247–254. 2

[GFD*12] GUENTER B., FINCH M., DRUCKER S., TAN D., SNYDER J.: Foveated 3D Graphics. *ACM Transactions on Graphics 31*, 6 (2012), 164. 2, 3, 6

[GP98] GEISLER W. S., PERRY J. S.: A real-time foveated multiresolution system for low-bandwidth video communication. In *in Proc. SPIE* (1998), pp. 294–305. 2

[Hun15] HUNT W.: *Virtual Reality: The Next Great Graphics Revolution*. Keynote Talk HPG 2015, Oculus Research, Aug 2015. 1, 2

[JESG12] JIMENEZ J., ECHEVARRIA J. I., SOUSA T., GUTIERREZ D.: SMAA: Enhanced subpixel morphological antialiasing. In *Computer Graphics Forum* (2012), vol. 31, pp. 355–364. 5

[JW02] JESCHKE S., WIMMER M.: Textured Depth Meshes for Real-time Rendering of Arbitrary Scenes. In *13th Eurographics Workshop on Rendering* (2002), Eurographics Association, pp. 181–190. 3

[LK87] LEGGE G. E., KERSTEN D.: Contrast discrimination in peripheral vision. *J. Opt. Soc. Am. A 4*, 8 (Aug 1987), 1594–1598. 2, 7

[LMR*12] LOU C. I., MIGOTINA D., RODRIGUES J. P., SEMEDO J., WAN F., MAK P. U., MAK P. I., VAI M. I., MELICIO F., PEREIRA J. G., ROSA A.: Object Recognition Test in Peripheral Vision: A Study on the Influence of Object Color, Pattern and Shape. In *International Conference on Brain Informatics* (2012), pp. 18–26. 2, 7

[LW90] LEVOY M., WHITAKER R.: Gaze-directed Volume Rendering. In *Proceedings of the 1990 Symposium on Interactive 3D Graphics* (1990), I3D '90, ACM, pp. 217–223. 2

[MD07] MURPHY H., DUCHOWSKI A. T.: Hybrid Image-/Model-based Gaze-contingent Rendering. In *4th Symposium on Applied Perception in Graphics and Visualization* (2007), ACM, pp. 107–114. 2

[Miu86] MIURA T.: Coping with situational demands: A study of eye movements and peripheral vision performance. *Vision in Vehicles* (1986), 206–216. 2, 8, 9

[MMB97] MARK W. R., MCMILLAN L., BISHOP G.: Post-rendering 3D Warping. In *Proceedings of the 1997 Symposium on Interactive 3D Graphics* (1997), ACM, pp. 7–ff. 3, 5

[MN84] MCKEE S., NAKAYAMA K.: The detection of motion in the peripheral visual field. *Vision research 24*, 1 (jan 1984), 25–32. 2

[NSL*07] NEHAB D., SANDER P. V., LAWRENCE J., TATARCHUK N., ISIDORO J. R.: Accelerating Real-time Shading with Reverse Reprojection Caching. In *22Nd ACM Symposium on Graphics Hardware* (2007), Eurographics Association, pp. 25–35. 3

[ODH02] O'SULLIVAN C., DINGLIANA J., HOWLETT S.: Gaze-contingent algorithms for interactive graphics. *Mind's Eyes: Cognitive and Applied Aspects of Eye Movement Research* (2002). 2

[PBNG15] POHL D., BOLKART T., NICKELS S., GRAU O.: Using astigmatism in wide angle HMDs to improve rendering. In *Virtual Reality* (2015), IEEE, pp. 263–264. 2

[Ree15] REED N.: *Gameworks VR*. Tech. rep., NVIDIA, 2015. 6

[RLMS03] REINGOLD E. M., LOSCHKY L. C., MCCONKIE G. W., STAMPE D. M.: Gaze-Contingent Multiresolutional Displays: An Integrative Review. *Human Factors 45* (2003), 307–328. 2

[SFD09] STICH M., FRIEDRICH H., DIETRICH A.: Spatial Splits in Bounding Volume Hierarchies. In *High Performance Graphics 2009* (2009), ACM, pp. 7–13. 3

[SGEM16] STENGEL M., GROGORICK S., EISEMANN M., MAGNOR M.: Adaptive image-space sampling for gaze-contingent real-time rendering. *Eurographics Symposium on Rendering 35*, 4 (2016). 2, 6

[Sim00] SIMMONS M.: *A Dynamic Mesh Display Representation for the Holodeck Ray Cache System*. Tech. Rep. UCB/CSD-00-1090, EECS Department, University of California, Berkeley, Jan 2000. 3

[SRJ11] STRASBURGER H., RENTSCHLER I., JÜTTNER M.: Peripheral vision and pattern recognition: A review. *Journal of vision 11*, 5 (jan 2011), 1–82. 3

[SS00] SIMMONS M., SÉQUIN C. H.: Tapestry: A Dynamic Mesh-based Display Representation for Interactive Rendering. In *Eurographics Workshop on Rendering Techniques 2000* (2000), pp. 329–340. 3, 4

[TPWG02] TOLE P., PELLACINI F., WALTER B., GREENBERG D. P.: Interactive Global Illumination in Dynamic Scenes. *ACM Trans. Graph. 21*, 3 (July 2002), 537–546. 3, 4

[Wan95] WANDELL B. A.: *Foundations of Vision*. Stanford University, 1995. 2

[WDP99] WALTER B., DRETTAKIS G., PARKER S.: Interactive Rendering using the Render Cache. In *Proceedings of the Eurographics Workshop on Rendering* (Jun 1999), vol. 10, pp. 235–246. 3

[WS99] WARD G., SIMMONS M.: The Holodeck Ray Cache: An Interactive Rendering System for Global Illumination in Nondiffuse Environments. *ACM Trans. Graph. 18*, 4 (Oct. 1999), 361–368. 3

[WWH04] WATSON B., WALKER N., HODGES L. F.: Supra-threshold Control of Peripheral LOD. *ACM Trans. Graph. 23*, 3 (Aug. 2004), 750–759. 2

[YNS*09] YANG L., NEHAB D. F., SANDER P. V., SITTHI-AMORN P., LAWRENCE J., HOPPE H.: Amortized supersampling. *ACM Trans. Graph. 28*, 5 (2009), 135:1–135:12. 3, 5

[YPG01] YEE H., PATTANAIK S., GREENBERG D. P.: Spatiotemporal Sensitivity and Visual Attention for Efficient Rendering of Dynamic Environments. *ACM Trans. Graph. 20*, 1 (Jan. 2001), 39–65. 2