Foveated Path Tracing

Foveated Path Tracing
a Literature Review and a Performance Gain Analysis

Matias Koskela, Timo Viitanen, Pekka Jääskeläinen, and Jarmo Takala

Department of Pervasive Computing, Tampere University of Technology, Finland

Abstract. Virtual Reality (VR) places demanding requirements on the rendering pipeline: the rendering is stereoscopic and the refresh rate should be as high as 95Hz to make VR immersive. One promising technique for making the final push to meet these requirements is foveated rendering, where the rendering effort is prioritized on the areas where the user's gaze lies. This requires rapid adjustment of level of detail based on screen space coordinates. Path tracing allows this kind of changes without much extra work. However, real-time path tracing is fairly new concept. This paper is a literature review of techniques related to optimizing path tracing with foveated rendering. In addition, we provide a theoretical estimation of performance gains available and calculate that 94% of the paths could be omitted. For this reason we predict that path tracing can soon meet the demanding rendering requirements of VR.

1 Introduction

Not long ago it was uncommon to own a smartphone. Nowadays everyone is accessing the web wirelessly from all over the world, finding places on their vacation trips without carrying maps and connecting to their relatives with video calls. All this is done with the help of mobile devices. *Virtual Reality* (VR) and *Augmented Reality* (AR), in other words, applications that create non-existing 3D worlds and applications that lay extra content on top of the real world, are starting to introduce societal changes of similar scale.

[1] Image by Daniel Pohl, licensed under https://creativecommons.org/licenses/by/4.0/

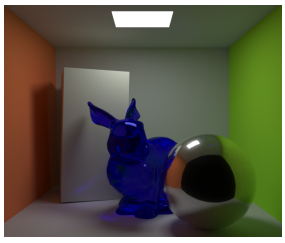

Fig. 1. Example of a path traced image

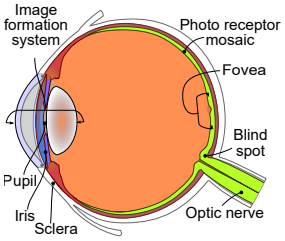

Fig. 2. Different parts of the human eye

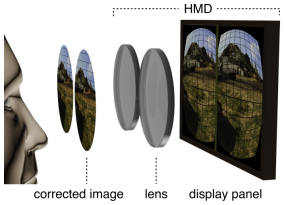

Fig. 3. Barrel distortion is used when rendering VR frames[1]

VR and AR devices require refresh rates as high as 95Hz and maximum latency of 20ms from user action to last photons, caused by the action, to be sent from displays [1]. When these requirements are met, users have reported to experience immersion, that is, the feeling of being present in another world. Consequently, rendering hardware and software will have to see major improvements to keep up with these requirements.

In this paper, we present a literature review on foveated path tracing, a promising technique which exploits eye tracking to reduce the computational cost of rendering. We also present a theoretical estimate of benefits available on contemporary and future VR devices. We start by briefly covering path tracing and fields that are most essentially connected to foveated rendering in Section 2. Then we explain foveated rendering in Section 3. We conduct the theoretical performance gain estimation in Section 4 and conclude the paper in Section 5.

2 Background

2.1 Path Tracing

Path tracing is a rendering method often used for offline, photorealistic rendering. In practice, basic forward path tracing renders images by shooting virtual photons from the camera into the scene, which then rebound at random from scene objects until they hit a light source. At each rebound, the light sample is weighed by the *Bidirectional Reflectance Distribution Function* (BRDF) of the surface material. Typically many such samples are taken per pixel and averaged before the image reaches good quality: as a Monte Carlo method, path tracing has square root convergence. Path tracing naturally models visual effects such as diffuse lighting, reflections, refractions, shadows, focal blur and caustics, which are approximated with special techniques in rasterization-based rendering.

A single sample in path tracing consist of tracing multiple rays in the scene. For this reason, path tracing can be made faster with two different main strategies: firstly, ray traversal can be sped up, or secondly, the amount of rays can be reduced. Ray traversal typically means finding out closest intersection of a single ray and the 3D geometry of the scene. There have been major leaps forward with the ray traversal thanks to improved algorithm design to exploit parallel hardware resources [2–4] and thanks to algorithmic improvements [5–7]. These improvements have paved the road for the real-time ray tracing frameworks [8–10]. However, these frameworks still require high-end desktop hardware to reach real-time frame rates. In 2013 it was estimated that 8 to 16 times more computation power is needed to enable path traced games [11].

In addition to improving ray traversal throughput, there is a large literature on reducing the number of rays needed for acceptable image quality. In rough terms, importance sampling and adaptive sampling techniques aim to select the traversed rays efficiently, while reconstruction filters out noise after rendering. There is a recent survey of related techniques by Zwichker et. al. [12]. The number of rays can also be reduced with *foveated rendering*, which focuses the main

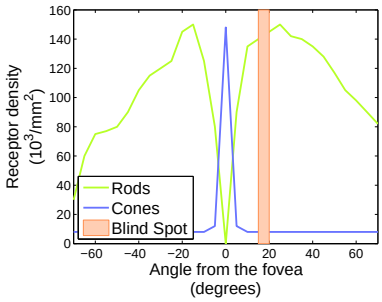

Fig. 4. Example of (*rod*) and (*cone*) density as a function of the angle to the center of the fovea

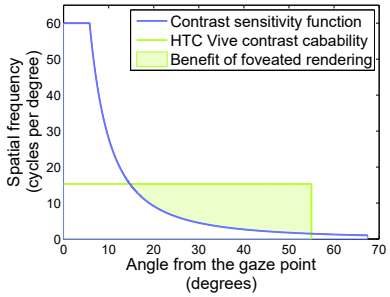

Fig. 5. A slice of the (*Contrast sensitivity function*), which models how much details an eye is able to resolve

rendering effort around the user's gaze, measured using eye tracking equipment. In fact, the main and the only user for virtually all rendering tasks is the human eye [13] and that is why it is important to know how human eyes work.

2.2 Human Eye

The human eye is a complex system. In simplified terms, it consists of two main components: the image formation system and the photoreceptor mosaic. This structure is visible in Fig 2. Photons travel through the image formation system to the photoreceptor mosaic, which sends the measured light data to the brain via the optic nerve [14].

The image formation system, like all optical systems, is not perfect, which means that the image will be somewhat blurred [14]. However, the system satisfies homogeneity and superposition, consequently, a linear system can be constructed which maps the input light density into the image projected on the photoreceptor mosaic. Thanks to this property there are accurate models of human eyes [13]. Moreover, the linearity means that there are no flaws, like inaccuracies, in the optical system, which could be used to optimize rendering.

On the other hand, the photoreceptor mosaic consists of more than 100 million light sensitive cells [14]. There are two types of cells: color sensitive *cones* and luminance sensitive *rods*. Cones require brighter lighting conditions to function. In contrast, rods stop working at bright lighting conditions.

The center of the human photoreceptor mosaic contains only color sensitive cones. This area is called *fovea* [14] and its size is around ten degrees. The lack of rods means that dim light sources can only be seen when viewer is not looking directly at them. More importantly, in the areas where the viewer's gaze is not fixed there are only few cones. Consequently, edges of the vision sense mostly changes in the brightness and mainly at dim lighting conditions. The distribution of the cones and rods can be seen in Fig. 4. The point where the optic nerve is attached to is called the *blind spot*, because there are no photo receptor cells in that area.

The amount of details the human eye is able to detect at certain point relative to gaze direction can be estimated with *Contrast Sensitivity Function* (CSF) visible in Fig. 5. The function has been deduced from measurements of human eyes and it is tested in user studies. It is a kind of worst case estimate for the use in computer graphics, meaning that it estimates the maximum amount of details most people are able to see. [15]

Photons of a computer generated images are sent to the human eyes with various types of display devices. Conventional displays may have multiple users, making it difficult to take advantage of characteristics of a single human eye. However, there is a sub-class of displays called *head-mounted displays*, where each display has only one user.

2.3 Head-Mounted Displays

The idea of *Head-Mounted Displays* (HMD) is to have displays affixed to the head of the user. By tracking the head motion and rendering so that the virtual camera moves correspondingly, HMDs can produce a sense of immersion in a virtual world. Therefore, HMDs are typically used with VR and AR applications.

An important property of a HMD is its *Field Of View* (FOV), which measures how much area of the sight of the user they cover [16]. The HMD's FOV is not to be mixed with a human's FOV, which tells how great angle human is able to see without rotating his/her head. A typical FOV varies from person to another, but usually it is around 160 degrees on horizontal and 135 degrees on vertical axis. Increasing the FOV of an HMD device enhances immersion, but might cause more motion sickness [17]. Usually immersion begins when the FOV of the HMD is around 80 degrees and deepens rapidly when the FOV is increased [1].

2.4 Eye Tracking

Eye tracking is the task of measuring what the user is currently looking at. The task can be divided into two subtasks: how to measure which direction the gaze of the user is pointing at and how to interpret the direction samples.

There are multiple ways to measure the direction of the user's gaze [18]. Typically there is some kind of a camera taking pictures of the eye. The camera may be, for example, an infrared camera combined with a bright infrared light [19]. Signal processing is used to determine which pixels correspond to different parts of the eye, e.g. the pupil, the iris and the sclera. What part of the eye is actually used in tracking depends on the camera configuration used [18]. All that is left to do is to map the tracked part's coordinates in the captured images to screen space locations on the display the user is looking at.

Coordinates in the image can be mapped to screen locations by calibrating the system at the beginning of eye tracking [18]. The user can be asked to look at different locations on the screen, and the screen space coordinates are connected to the tracking results. Another option is to accurately measure the position of the eye relative to the camera and calculate the calibration results. The main difficulty with calibrations is that they can gradually lose accuracy. For

example, if a head mounted eye tracking device changes its relative orientation to the user's head, this causes drifting in the tracking results. One solution is to track the relative position of the device on the head. Another is to use multiple different eye tracking methods from different angles. When the calibration is in place, the device is able to obtain accurate estimate of gaze coordinates on the screen space. This raises the problem of interpreting the coordinates.

The other subtask of eye tracking is interpreting the gaze coordinate data. Often the application wants to know so called *fixation* points of human sight, which are the points where sight pauses to look at informative regions of interest. Rapid movements between fixations are called *saccades*. The distinction between fixations and saccades is important, because only little or no visual processing is done by the brain during saccades. [20, 21]

One of the easiest ways to classify tracking data to fixations and saccades is based on the velocity [20]. However, this is very vulnerable to noise in the data and the selection of parameters can change the fixation points completely [22]. The problem of noise can be overcome by looking at a window of tracking samples at once or using filtering such as Kalman filter [23].

Eye tracking enables interesting optimizations for real-time rendering tasks because rendering can concentrate on the area where the user is looking at. Some sources refer to this as *gaze-directed* or *gaze-contingent rendering* [21, 24–26], but nowadays *foveated rendering* [1, 27–29] seems to be more commonly used.

3 Foveated Rendering

Foveated rendering means that only those details are rendered which the user is actually looking at, based on eye tracking data. There is a large body of work in optimizing rasterized rendering based on gaze-direction. In contrast, foveated path tracing has not gained as much interest, maybe because path tracing, at the time of the writing, has not been widely used in real-time applications.

3.1 Rasterized Foveated Rendering

One approach for adding foveated rendering to an existing rendering pipeline is use the gaze direction as an input to complex fragment shaders. The shader code can then run a simplified version, if it realises that the user is not looking at the current target pixel. For example, fragment shading for the uninteresting parts can be done with fewer ambient occlusion samples [26, 30]. This technique increases divergence in the shader code, but neighbouring pixels are always almost as far from the gaze point, so they usually follow the same code paths.

Significant performance gain can be achieved if the whole rendering pipeline is designed around the estimate on how much detail the eye sees in different angles to the gaze point, that is the CSF function. The gaze direction can be given as an input to the *Level of Detail* (LoD) algorithms [21]. The idea of LoD is to replace distant geometry with a simplified version that has fewer triangles. In an extreme case, a distant model with thousands of triangles might take up

only a few on-screen pixels. The basic idea is straightforward, but there is a large literature on techniques to make the transition between levels of detail seamless and avoid visual artifacts. In CSF based LoD, the level of detail is based on eye-tracking data in addition to distance. This requires having multiple versions of each model in memory, rather than changing the model only once when distance to the object changes. Moreover, changing a model that covers great portion of the display area often causes flickering [31].

Another idea is to reduce the amount of samples in the screen space. In theory, perfect results could be achieved by sampling the 3D world according to the CSF. However, such resampling is difficult to map to a rasterization pipeline. Guenther et al. [27] render sections of the image at multiple resolutions. A final rendering pass blends the sections into a foveated image. This approach has the drawback that sections need to overlap. In addition, vertex and geometry shaders are re-run for each section, but the savings in rasterized and shaded pixels are enough to improve performance by a factor of 5-6.

Along with the gaze direction, also the gaze speed should be used as an input to the detail reduction algorithm. In an extreme case, Ohshima et al. [21] are not updating the image at all during saccades. A more commonly used idea is to reduce the quality more dramatically when the eye is moving [15, 27, 31].

Foveated rendering can achieve significant speedups. Guenter et al. [27] reported that a 100x speedup is possible with a FOV of 70%. Moreover, they state that increasing the display's FOV, which usually has a quadratic effect on rendering requirements, has a linear effect on foveated rendering, because the added extra display area is only adding an even lower level of quality rendering.

In summary, there are still two major difficulties with rasterization and foveated rendering: Firstly, it is hard to sample according to the CSF in screen space. Secondly, changing LoD based on gaze direction causes quick model changes all the time and, therefore, requires having multiple levels of details versions of the same model in memory.

3.2 Foveated Path Tracing

In contrast to fixed resolution of rasterization, in path tracing, rays are sent from screen locations. It is straightforward to distribute these rays according to the CSF. This optimization is one idea of making path tracing faster by reducing the total ray count.

There are already a few publications of techniques for foveated ray tracing. Murphy et al. [24] chose so called ray casting, which sends out only one ray per pixel, because it suited better for their test cases, where they change both the image space sampling rates and the model quality. Zhang et al. [28] use a screen-space ray tracing technique based on depth peeling the scene, but this approach is approximate and limited to simple scenes. Swardford et al. [30] test different amounts of quality reduction with foveated ray casting using multi-layer relief mapping. In fourth found paper Fujita et al. [29] call their ray tracing foveated, even though they do not utilize eye tracking and, therefore, they have the best quality always on the center of the screen. Siekawa et al. [25] reduce the rendering

time of a single path traced frame from 48 minutes to 15 minutes by introducing a simulated static gaze point to their non-real-time rendering.

4 Theoretical Performance Gain Analysis

The motivation of this analysis is to find a lower bound on the speed up foveation can give to path tracing. CSF estimates how much details the human eye is able to resolve as a function of the angle from the gaze fixation point. The size of the detail is expressed as so called *spatial frequency* and the unit is cycles per degree (c/deg). That is, how many of given sized details fit into one degree of human vision. By using CSF it is possible to find out how many rays we can omit when using foveated path tracing. Approximation model of CSF can be divided into two separate parts

$$H(e, v) = M(e) \times G(v) \tag{1}$$

where $M(e)$ is a function of angle e to the center of the gaze fixation point and $G(v)$ depends on the velocity v of the eye rotation [15]. Increasing velocity reduces the amount of contrast human eye is able to detect. Since we are trying to find the minimum amount of quality we can omit, we set the velocity to its most pessimistic value of zero. That is, the situation when the eye is focused and seeing as much details as it can, in other words $G(0) = 1$. Taking into account that the smallest detail humans are able to resolve is ca. 60 cycles per degree, the equation from [15] can be simplified to

$$H(e, 0) = M(e) = \begin{cases} 60.0 & 0 \leq e \leq 5.79 \\ \frac{449.4}{(0.3e+1)^2} & e > 5.79 \end{cases}. \tag{2}$$

First we examine performance gain on a perfect HMD device capable of showing as much details as human eyes are able to resolve. A perfect HMD device would be one capable of displaying this $60c/deg$ details with the oval FOV of 160 degrees horizontal and 135 degrees vertical. The biggest amount details need to be rendered when the user is looking at the center of the screen. For this reason, to provide lower bound estimate, we substitute the angle $e = 0$ of the Eq. 2 to the center of the perfect HMD's FOV. In that case, e tells the angle from the center of the FOV. One slice of the FOV area is shown in Fig. 5.

Then we integrate the Eq. 2 over the whole area of the oval FOV. When the volume is compared to the maximum amount of resolvable details $60c/deg$ on the same area, the result is that 94% of the details are unresolvable.

Another interesting case to estimate in the theoretical examination is calculating the same number for one of a contemporary consumer grade VR helmet, the HTC Vive, which is able to display details with around $15.3c/deg$ [32]. We limit the Eq. 2 to this number and calculate the integral over a circular area with radius of 110 degrees. The result is that at least 70% of the details can be omitted. The area of these omitted details is highlighted in Fig. 5. Remember that this is just one slice of the solid of revolution around the vertical axis.

The ratio of the details that could be omitted might not linearly correlate to the speed up gains possible with path tracing. However, there are many ways how details can be omitted, for example, there can be fewer paths for each pixel or paths can, e.g., use simplified lighting instead of full path tracing. The use of less samples produces higher frequency noise, but this can be reduced with more intensive filtering on areas where the user is not looking at. For example, foveated rasterized rendering can use anti-alias sampling to reduce artefacts [27].

What is the best way to reduce path tracing quality with eye-tracking is currently an open research question and therefore the ratio of extra details can be used as rough performance gain estimate. 94% performance gain is a bit better than the numbers Gunther et al. [27] found in their user studies as a number of pixels that can be reduced with rasterized rendering. Since their display is far from the perfect HMD, their pixel saving results should be lower. Moreover, since the FOV of their desktop display is smaller, the theoretical number of 70% savings on HTC Vive is a lower bound and in reality higher numbers are possible.

Path tracing a arbitrary scene might require hundreds of rays per pixel. However, a scene for adequate quality path tracing could be built so that it requires for example around 11 rays per pixel [33, 34]. This can be achieved by using only simple materials or by using more complex post processing. HTC Vive has a refresh rate of 90Hz and a resolution of 2160x1200 with 15% of the pixels invisible to the user [32]. This results in a required number of rays per second of around 2 180 MRays/s. According to the worst case estimate above, at least 70% of the rays could be omitted, which makes the requirement into 654 MRays/s. This number should be reachable with a modern high-end GPU setup [35]. In addition, the pipeline step of distortion handling [36], visualised in Fig. 3, can be greatly simplified or even avoided with path tracing. Reduced ray counts reflect savings in rendering computations and memory bandwidth usage.

5 Conclusions

Foveated path tracing is a promising technique for rendering VR applications. In foveated rendering the computation effort is focused mostly to screen space area where the user is looking at. With rasterized rendering this has already shown to improve performance by a factor of 5-6. However, foveated rendering is even more suited for path tracing, which is done by sending rays from screen space locations. Recently, real-time ray tracing has been made feasible on high-end hardware. Foveation could enable real-time path tracing on consumer devices, especially on hand-held mobile devices. Furthermore, typically VR and AR applications use HMD devices. Given that a HMD is specific to a single user, and covers wide field of view, the idea of foveated rendering is even more appealing.

We derived from a theoretical worst case model that foveation can omit at least 94% of rays required for the path tracing on future VR device which is capable of showing as much details as humans are able to perceive. Already on today's VR device at least 70% rays can be omitted. Moreover, thanks to reduced rendering work provided by the foveation, the very demanding rendering

requirements of VR could be met today with high-end GPUs. For these reasons we believe that path tracing is a very promising choice of rendering technique in the future of VR. As a future work we are interested in building the proposed system to empirically validate the numerical estimates proposed in this paper.

6 Acknowledgement

We would like to thank anonymous reviewers for fruitful comments and Williams College and the Stanford University scanning repository for the 3D models. In addition, we are thankful to our funding sources: TUT graduate school, TEKES (project "Parallel Acceleration 3", funding decision 1134/31/2015), European Commission in the context of ARTEMIS project ALMARVI (ARTEMIS 2013 GA 621439) and industrial research fund of TUT by Tuula and Yrjö Neuvo.

References

1. Abrash, M.: What VR could, should, and almost certainly will be within two years (2014) Steam Dev Days, Seattle.
2. Wald, I., Benthin, C., Boulos, S.: Getting rid of packets – efficient SIMD single-ray traversal using multi-branching BVHs. In: Proc. of the IEEE Symp. on Interactive Ray Tracing. (2008)
3. Laine, S., Karras, T., Aila, T.: Megakernels considered harmful: Wavefront path tracing on GPUs. In: Proc. of the High-Performance Graphics. (2013)
4. Garanzha, K., Premože, S., Bely, A., Galaktionov, V.: Grid-based sah bvh construction on a gpu. The Visual Computer **27** (2011)
5. Pantaleoni, J., Luebke, D.: HLBVH: Hierarchical LBVH construction for real-time ray tracing of dynamic geometry. In: Proc. of the High-Performance Graphics. (2010)
6. Karras, T.: Maximizing parallelism in the construction of BVHs, octrees, and k-d trees. In: Proc. of the High-Performance Graphics. (2012)
7. Keely, S.: Reduced precision hardware for ray tracing. In: Proc. of the High-Performance Graphics. (2014)
8. Wald, I., Woop, S., Benthin, C., Johnson, G.S., Ernst, M.: Embree: A kernel framework for efficient CPU ray tracing. ACM Trans. on Graphics **33** (2014)
9. Parker, S.G., Bigler, J., Dietrich, A., Friedrich, H., Hoberock, J., Luebke, D., McAllister, D., McGuire, M., Morley, K., Robison, A., et al.: Optix: a general purpose ray tracing engine. ACM Trans. on Graphics **29** (2010)
10. AMD: RadeonRays SDK. Online (2016) Available: https://github.com/GPUOpen-LibrariesAndSDKs/RadeonRays_SDK, Referenced: 6th of October 2016.
11. Bikker, J., van Schijndel, J.: The Brigade renderer: A path tracer for real-time games. Int. Jour. of Computer Games Technology **2013** (2013)
12. Zwicker, M., Jarosz, W., Lehtinen, J., Moon, B., Ramamoorthi, R., Rousselle, F., Sen, P., Soler, C., Yoon, S.E.: Recent advances in adaptive sampling and reconstruction for monte carlo rendering. **34** (2015)
13. Deering, M.F.: A photon accurate model of the human eye. In: ACM SIGGRAPH Papers. (2005)

14. Wandell, B.A.: Foundations of Vision. Sinauer Associates (1995)
15. Reddy, M.: Perceptually optimized 3D graphics. IEEE Computer Graphics and Applications **21** (2001)
16. Bowman, D.A., Kruijff, E., LaViola Jr, J.J., Poupyrev, I.: 3D user interfaces: theory and practice. Addison-Wesley (2004)
17. Benko, H., Ofek, E., Zheng, F., Wilson, A.D.: Fovear: Combining an optically see-through near-eye display with projector-based spatial augmented reality. In: Proc. of the ACM Symp. on User Interface Software & Technology. (2015)
18. Hua, H.: Integration of eye tracking capability into optical see-through head-mounted displays. In: Proc. SPIE. (2001)
19. Stengel, M., Grogorick, S., Eisemann, M., Eisemann, E., Magnor, M.A.: An affordable solution for binocular eye tracking and calibration in head-mounted displays. In: Proc. of the ACM Int. Conf. on Multimedia. (2015)
20. Salvucci, D.D., Goldberg, J.H.: Identifying fixations and saccades in eye-tracking protocols. In: Proc. of the Eye Tracking Research & Applications. (2000)
21. Ohshima, T., Yamamoto, H., Tamura, H.: Gaze-directed adaptive rendering for interacting with virtual space. In: Proc. of the VR Annual Int. Symp. (1996)
22. Shic, F., Scassellati, B., Chawarska, K.: The incomplete fixation measure. In: Proc. of the 2008 Symp. on Eye Tracking Research & Applications. (2008)
23. Ji, Q., Yang, X.: Real time visual cues extraction for monitoring driver vigilance. In: Proc. of the Int. Computer Vision Systems. (2001)
24. Murphy, H.A., Duchowski, A.T., Tyrrell, R.A.: Hybrid image/model-based gaze-contingent rendering. ACM Trans. Appl. Percept. **5** (2009)
25. Siekawa, A., Mantiuk, S.R.: Gaze-dependent ray tracing. In: Proc. Central European Seminar on Computer Graphics (non-peer-reviewed). (2014)
26. Mantiuk, R., Janus, S.: Gaze-dependent ambient occlusion. In: Proc. Int. Symp. Advances in Visual Computing. (2012)
27. Guenter, B., Finch, M., Drucker, S., Tan, D., Snyder, J.: Foveated 3D graphics. ACM Trans. on Graphics **31** (2012)
28. Zhang, X., Chen, W., Yang, Z., Zhu, C., Peng, Q.: A new foveation ray casting approach for real-time rendering of 3D scenes. In: Proc. of the Computer-Aided Design and Computer Graphics. (2011)
29. Fujita, M., Harada, T.: Foveated real-time ray tracing for virtual reality headset. Technical report, Light Transport Entertainment Research (2014)
30. Swafford, N.T., Iglesias-Guitian, J.A., Koniaris, C., Moon, B., Cosker, D., Mitchell, K.: User, metric, and computational evaluation of foveated rendering methods. In: Proc. of the ACM Symp. on Applied Perception. (2016)
31. Luebke, D., Hallen, B.: Perceptually driven simplification for interactive rendering. In: Proc. of the Eurographics Workshop. (2001)
32. Vlochos, A.: Advanced VR rendering (2015) Game Developers Conference, San Francisco.
33. Sen, P., Darabi, S.: Implementation of random parameter filtering. Technical report (2011)
34. Lee, W.J., Shin, Y., Lee, J., Kim, J.W., Nah, J.H., Jung, S., Lee, S., Park, H.S., Han, T.D.: SGRT: A mobile GPU architecture for real-time ray tracing. In: Proc. of the High-Performance Graphics. (2013)
35. Aila, T., Laine, S., Karras, T.: Understanding the efficiency of ray traversal on GPUs–Kepler and Fermi addendum. Technical report, NVIDIA Corporation (2012)
36. Pohl, D., Johnson, G.S., Bolkart, T.: Improved pre-warping for wide angle, head mounted displays. In: Proc. of the ACM Symp. on VR Software and Technology. (2013)