Tampere University

MATIAS KOSKELA

Foveated Path Tracing

with Fast Reconstruction and Efficient Sample Distribution

Tampere University Dissertations 233

Tampere University Dissertations 233

MATIAS KOSKELA

Foveated Path Tracing

with Fast Reconstruction and Efficient Sample Distribution

ACADEMIC DISSERTATION
To be presented, with the permission of
the Faculty of Information Technology and Communication Sciences
of Tampere University,
for public discussion in the Auditorium TB109
of the Tietotalo, Korkeakoulunkatu 1, Tampere,
on 27th of March 2020, at 12 o'clock.

ACADEMIC DISSERTATION
Tampere University, Faculty of Information Technology and Communication Sciences
Finland

Responsible supervisor	Professor Jarmo Takala Tampere University Finland	
Supervisor and Custos	Assistant Professor Pekka Jääskeläinen Tampere University Finland	
Pre-examiners	Professor Tamy Boubekeur Telecom Paris France	Adjunct Professor Kari Pulli University of Oulu Finland
Opponent	Professor Ulf Assarsson Chalmers University of Technology Sweden	

Cover design: Roihu Inc.

ISBN 978-952-03-1513-9 (print)
ISBN 978-952-03-1514-6 (pdf)
ISSN 2489-9860 (print)
ISSN 2490-0028 (pdf)
http://urn.fi/URN:ISBN:978-952-03-1514-6

PunaMusta Oy – Yliopistopaino
Tampere 2020

ACKNOWLEDGEMENTS

This research was carried out at the Virtual reality and Graphics Architectures (VGA) group of Tampere University (formerly Tampere University of Technology) during the years 2015-2019. First, I would like to thank my supervisors Prof. *Jarmo Takala* for making the research possible and Asst. Prof. *Pekka Jääskeläinen* for daily guidance and help with all the practicalities of the research. It has been a pleasure to work at the VGA group with all my awesome colleagues. Especially, I would like to express my gratitude to Dr. *Timo Viitanen*, Mr. *Kalle Immonen*, M.Sc., Mr. *Atro Lotvonen*, B.Sc., Dr. *Markku Mäkitalo*, Mr. *Julius Ikkala*, B.Sc., Mr. *Petrus Kivi*, B.Sc., Mr. *Joonas Multanen*, M.Sc., and Mr. *Heikki Kultala*, M.Sc.

Also, I would like to thank Advanced Rendering and Compute group of Apple Inc. where I had the privilege to complete an internship during my Doctoral studies. In particular, I want to thank Mr. *Sean James*, B.Sc., Mr. *Teemu Rantalaiho*, M.Sc., Mr. *Dhruv Saksena*, M.Sc., Mrs. *Chalana Bezawada*, M.Sc., and Mr. *Max Yuan*.

I am thankful to Prof. *Alessandro Foi* for guidance in the denoising work and Mr. *Toimi Teelahti*, M.Sc., for grammatical corrections of the introductory part of this thesis.

In addition, I would like to thank the pre-examiners of this thesis Prof. *Tamy Boubekeur* and Adj. Prof. *Kari Pulli*. Also, I would like to thank Prof. *Ulf Assarsson* for agreeing to be the opponent in the public defense of this thesis.

I am grateful to Stanford 3D scanning repository for the dragon, Morgan McGuire for Cornell Box (CC BY 3.0), Frank Meinl for Crytek Sponza (CC BY 3.0), and Christophe Seux for Class room models used in the figures [McG17].

I am also grateful to all of the funding sources that made this research possible: Tampere University of Technology (TUT) Graduate school, Nokia Foundation, Emil Aaltonen Foundation, Finnish Foundation for Technology Promotion, Industrial research fund of TUT by Tuula and Yrjö Neuvo, Doctoral Education Network Intelligent Systems (DENIS), Doctoral training network in ELec-

tronics, Telecommunications and Automation (DELTA), TEKES project "Parallel Acceleration 3" (decision 1134/31/2015), ARTEMIS project ALMARVI (2013 GA 621439), Finnish Funding Agency for Technology and Innovation (decision 40142/14, FiDiPro-StreamPro), and ECSEL JU project FitOptiVis (project number 783162).

Finally, I want to thank my wife Mrs. *Sara Tulla-Koskela*, M.A., and my daughters *Lumi* and *Usva* for making my life awesome and making everything in my work possible even when it requires going to the ends of the Earth, literally.

ABSTRACT

Photo-realistic offline rendering is currently done with path tracing, because it naturally produces many real-life light effects such as reflections, refractions and caustics. These effects are hard to achieve with other rendering techniques. However, path tracing in real time is complicated due to its high computational demand. Therefore, current real-time path tracing systems can only generate very noisy estimate of the final frame, which is then denoised with a post-processing reconstruction filter.

A path tracing-based rendering system capable of filling the high resolution in the low latency requirements of mixed reality devices would generate a very immersive user experience. One possible solution for fulfilling these requirements could be foveated path tracing, wherein the rendering resolution is reduced in the periphery of the human visual system. The key challenge is that the foveated path tracing in the periphery is both sparse and noisy, placing high demands on the reconstruction filter.

This thesis proposes the first regression-based reconstruction filter for path tracing that runs in real time. The filter is designed for highly noisy one sample per pixel inputs. The fast execution is accomplished with blockwise processing and fast implementation of the regression. In addition, a novel Visual-Polar coordinate space which distributes the samples according to the contrast sensitivity model of the human visual system is proposed. The specialty of Visual-Polar space is that it reduces both path tracing and reconstruction work because both of them can be done with smaller resolution. These techniques enable a working prototype of a foveated path tracing system and may work as a stepping stone towards wider commercial adoption of photo-realistic real-time path tracing.

TIIVISTELMÄ

Polunseuranta on tietokonegrafiikan piirtotekniikka, jota on käytetty pääasiassa ei-reaaliaikaisen realistisen piirron tekemiseen. Polunseuranta tukee luonnostaan monia muilla tekniikoilla vaikeasti saavutettavia todellisen valon ilmiöitä kuten heijastuksia ja taittumista. Reaaliaikainen polunseuranta on hankalaa polunseurannan suuren laskentavaatimuksen takia. Siksi nykyiset reaaliaikaiset polunseurantasysteemi tuottavat erittäin kohinaisia kuvia, jotka tyypillisesti suodatetaan jälkikäsittelykohinanpoisto-suodattimilla.

Erittäin immersiivisiä käyttäjäkokemuksia voitaisiin luoda polunseurannalla, joka täyttäisi laajennetun todellisuuden vaatimukset suuresta resoluutiosta riittävän matalassa vasteajassa. Yksi mahdollinen ratkaisu näiden vaatimusten täyttämiseen voisi olla katsekeskeinen polunseuranta, jossa piirron resoluutiota vähennetään katseen reunoilla. Tämän johdosta piirron laatu on katseen reunoilla sekä harvaa että kohinaista, mikä asettaa suuren roolin lopullisen kuvan koostavalle suodattimelle.

Tässä työssä esitellään ensimmäinen reaaliajassa toimiva regressionsuodatin. Suodatin on suunniteltu kohinaisille kuville, joissa on yksi polunseurantanäyte pikseliä kohden. Nopea suoritus saavutetaan tiileissä käsittelemällä ja nopealla sovituksen toteutuksella. Lisäksi työssä esitellään Visual-Polar koordinaattiavaruus, joka jakaa polunseurantanäytteet siten, että niiden jakauma seuraa silmän herkkyysmallia. Visual-Polar-avaruuden etu muihin tekniikoiden nähden on että se vähentää työmäärää sekä polunseurannassa että suotimessa. Nämä tekniikat esittelevät toimivan prototyypin katsekeskeisestä polunseurannasta, ja saattavat toimia tienraivaajina laajamittaiselle realistisen reaaliaikaisen polunseurannan käyttöönotolle.

CONTENTS

List of Figures

List of Tables

ABBREVIATIONS

2D	Two-Dimensional
3D	Three-Dimensional
AR	Augmented Reality
BMFR	Blockwise Multi-order Feature Regression
BSDF	Bidirectional Scattering Distribution Function
BVH	Bounding Volume Hierarchy
CNN	Convolutional Neural Network
FOV	Field of View
fps	frames per second
GPU	Graphics Processing Unit
HLBVH	Hierarchical Linear BVH
HMD	Head Mounted Display
HVS	Human Visual System
k-NN	k-Nearest Neighbor
MIS	Multiple Importance Sampling
MR	Mixed Reality
MSAA	Multisample Anti-Aliasing
PLOC	Parallel Locally-Ordered Construction
RCNN	Recurrent Convolutional Neural Network
SAH	Surface Area Heuristic
SIMD	Single Instruction Multiple Data

SIMT	Single Instruction Multiple Thread
spp	samples per pixel
SSAA	Super Sampling Anti-Aliasing
SVGF	Spatiotemporal Variance-Guided Filtering
TAA	Temporal Anti-Aliasing
VR	Virtual Reality
VRS	Variable Rate Shading

NOMENCLATURE

Path Tracing (Section 2.4):

$\mathbf{x}$	Shaded 3D-point
$\mathbf{n}$	Surface normal at the point $\mathbf{x}$
ω_o	Outgoing light direction
ω_i	Incoming light direction
Ω	All possible directions
L_o	Outgoing luminance
L_i	Incoming luminance
L_e	Emitted luminance
f_r	bidirectional scattering distribution function
F	Weight of the Monte-Carlo integrant
p	Random value
q	Russian Roulette threshold

Bilateral Blur (Section 3.2):

w	Weight of the sample
(x, y)	Target pixel's coordinate
(x', y')	Sample pixel's coordinate
σ	Standard deviation
$I(x, y)$	Color of a pixel at (x, y)
$n(x, y)$	First bounce normal direction at pixel (x, y)
$Z(x, y)$	First bounce distance at pixel (x, y)

Regression (Section 3.5):

Z	Noisy path traced data
T_m	Feature buffer m
a_m	Weight for feature buffer m
M	Count of feature buffers
$\Omega_{i,j}$	Regression window around target pixel

Foveated Sample Distribution (Chapter 4):

L	Sampling probability
e	Eccentricity angle
f_l	Fovea limit in eccentricity degrees
p_l	Periphery limit in eccentricity degrees
P_p	Periphery sampling probability
ρ	Distance from the gaze point
ϕ	Angle around the gaze point
S	Sampling density
V	Visual acuity

ORIGINAL PUBLICATIONS

This thesis consists of introductory part and five original publications reproduced in the end of the thesis with kind permissions from the publishers.

P1 M. Koskela, T. Viitanen, P. Jääskeläinen and J. Takala. Foveated Path Tracing: A Literature Review and a Performance Gain Analysis. *Proceedings of International Symposium on Visual Computing*. Ed. by I. Daisuke and A. Sadagic. 2016. DOI: 10.1007/978-3-319-50835-1_65.

P2 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. *Transactions on Graphics* 38.5 (2019). DOI: 10.1145/3269978.

P3 M. Koskela, K. Immonen, T. Viitanen, P. Jääskeläinen, J. Multanen and J. Takala. Foveated Instant Preview for Progressive Rendering. *SIGGRAPH Asia Technical Briefs*. Ed. by D. Gutierrez. 2017. DOI: 10.1145/3145749.3149423.

P4 M. Koskela, K. Immonen, T. Viitanen, P. Jääskeläinen, J. Multanen and J. Takala. Instantaneous Foveated Preview for Progressive Monte Carlo Rendering. *Computational Visual Media* 4.3 (2018). DOI: 10.1007/s41095-018-0113-0.

P5 M. Koskela, A. Lotvonen, M. Mäkitalo, P. Kivi, T. Viitanen and P. Jääskeläinen. Foveated Real-Time Path Tracing in Visual-Polar Space. *Eurographics Symposium on Rendering (DL-only Track)*. Ed. by T. Boubekeur and P. Sen. 2019. DOI: 10.2312/sr.20191219.

1 INTRODUCTION

The creation of photo-realistic frames which are indistinguishable from real images has always been the main goal in the field of computer graphics. This goal has been achieved in offline context, where significant amounts of resources, both computer and human, may be used even on a single frame. Currently, research in the field is focused on bringing the same level of visual fidelity to real-time rendering. The main difference between real-time rendering and offline rendering is that in real-time rendering, the user may affect the image by moving the camera or the objects in the virtual 3D world. In other words, real-time rendering in tens of milliseconds is required if the application is interactive. Real-time photo-realistic rendering would, for example, provide more realistic training simulators, better medical applications as well as higher quality entertainment. In addition, real-time rendering is used by artists for previewing offline renderings and, therefore, better real-time quality also improves offline rendering workflow.

In the offline context, rendering is currently done with so-called *path tracing* [Kaj86; Kel+15]. One of the most important motivations to use path tracing is that the same unified rendering pipeline can be used to simulate most real-world light phenomena. An example of soft shadows, reflections and refractions produced by path tracing can be seen in Figure 1.1. Path tracing first generates a noisy estimation of the frame. As more and more light paths are averaged, the noise is reduced [PH10]. There are many algorithm modifications to pure path tracing such as next event estimation and importance sampling which make the noise reduction faster. One can also use a post processing filter to approximate the final result with fewer computations compared to actual simulation of a sufficient number of paths [Zwi+15]. We are going to see more and more real-time applications with this kind of visual fidelity since all major *Graphics Processing Unit* (GPU) manufacturers have released or announced GPUs with dedicated hardware for ray traversal [Kil+18], which is a primitive operation used by path tracing.

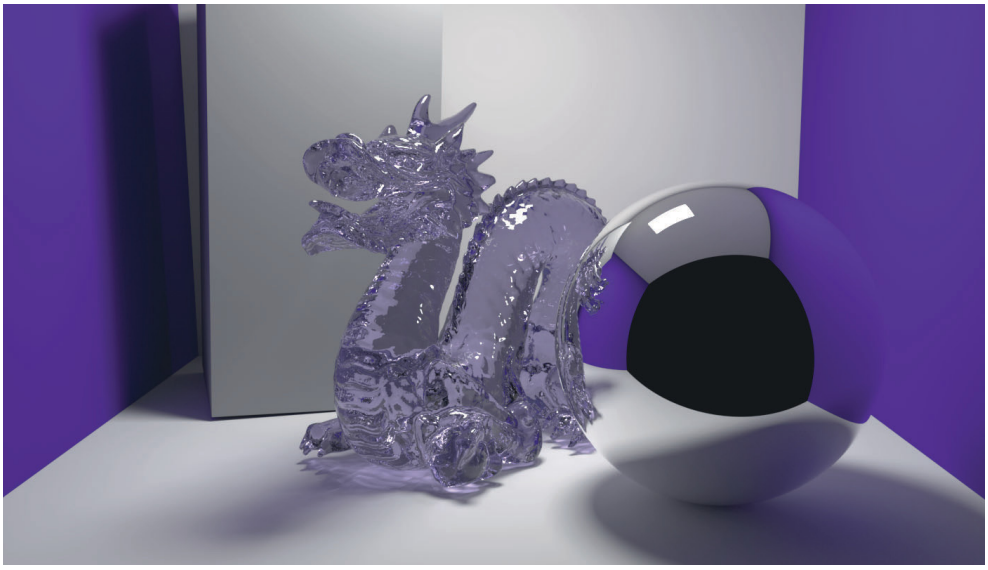


Figure 1.1 Example of path tracing a virtual 3D scene to generate a 2D frame. Notice how realistically the light interacts with the dragon made of virtual glass.

In recent years, there has been a lot of interest in *Virtual Reality* (VR) and *Augmented Reality* (AR) devices. A collective term for such devices is *Mixed Reality* (MR) devices. A commercial MR device using a *Head Mounted Display* (HMD) of sufficient quality would quite possibly make all existing screens in use obsolete because the MR device could render their content on the HMD. However, wide adoption of these devices is still waiting for devices of high enough quality and commercially interesting applications.

From a computer graphics perspective, MR devices have some interesting challenges. For better immersion and reduced simulation sickness, the rendering resolution and latency requirements for MR devices are very demanding [Abr14]. However, there is only one user per device and it can be measured with en eye tracking device at which point on the screen the user is looking [Kra+16]. Moreover, the potential gain of an eye-tracking based, so-called *foveated rendering* optimization, with a single user is high since human visual acuity drops significantly in the periphery of the vision.

1.1 Objectives and Scope of the Thesis

The objective of this thesis is to combine the two worlds of path tracing and foveated rendering. The motivation is to enable photo-realistic rendering for a single user in real-time. Combining these two worlds is not straightforward because it places extra challenges since real-time path-traced foveated frames are both noisy and sparse. Moreover, even with the sparse sampling of the periphery the results must be temporally stable.

Foveation requires the rendering to be done in real time. Before the start of this thesis project path tracing was mainly used only in the offline context, but currently real-time path tracing appears to be closer than ever. The first reconstruction methods which work on a sufficiently low path tracing sample budget to be usable in real time and which are still able to generate visually pleasing results have recently been presented [Mar+17; **P2**; Sch+17].

There has been large body of work on rasterized foveated graphics because rasterization has been the most commonly used rendering technique for real-time application due to its fast hardware support. More recently, some ray tracing based foveation research has emerged [PZB16; Sie+19; Wei+16; Wei+18a]. However, these works assume noise-free ray tracing algorithms and therefore they only need to consider the sparsity of the samples. In this thesis, the rendering in the periphery is both noisy and sparse.

This thesis proposes techniques for implementing an end to end foveated path tracing system. The research method used in this thesis was constructive research. This thesis includes five original publications [**P1**; **P2**; **P3**; **P4**; **P5**].

1.2 Thesis Contributions

The first contribution of the Thesis is an estimation of the upper bound of foveated rendering optimization to be 95% of reduced rendering work [**P1**]. The other contributions are a novel real-time path tracing reconstruction method [**P2**] and novel ways for distributing the path tracing samples so that their density follows the resolution of the human visual system [**P3**; **P4**; **P5**].

At the time of starting this thesis project, there were no methods fast enough to reconstruct path tracing in real-time, therefore a novel regression-based real-time re-

construction system for path tracing is proposed [**P2**]. Other work on real-time reconstruction is based on fast approximations of cross bilateral filter [Mar+17; Sch+17]. In offline context, regression has shown good results and, therefore, it is an interesting candidate for real-time filtering. In this thesis different ways of making regression orders of magnitude faster are introduced. For instance, stochastic regularization is used for getting rid of rank deficiencies cheaply. Moreover, augmented QR-decomposition as a regression method reduces GPU memory traffic significantly.

Also a foveated method for previewing offline progressive rendering is proposed [**P3**; **P4**]. In this method the results are not denoised, but the gaze contingent rendering with accurate following of human visual acuity makes the results to converge to noise-free image quicker. With this kind of system, artists can quickly preview their renderings in real-time without any artifacts from reconstruction filter.

Finally, a novel Visual-Polar space which distributes the samples according to human visual acuity is introduced [**P5**]. Specialty of visual-Polar space is that also reconstruction can be done in it before mapping the results back to screen space. Therefore, Visual-Polar space reduces both path tracing and reconstruction work significantly. The emphasis of the publication is on efficient path tracing and reconstruction and the idea is that it can be used with different methods that map the frames to the screen space.

1.3 The Author's Contributions

In this section, the Author's contributions to each included publication is described in detail. The Author was the main contributor and responsible of the actual publication writing in all the publications.

The basis of the first publication [**P1**] was mainly individual work of the Author, but other authors helped in the writing process of the publication.

The original idea of blockwise multi-order feature regression for a single frame was proposed by Prof. Alessandro Foi, and the task of the first three authors of [**P2**] was to make it temporally stable and study ways to make it fast enough for real-time. The Author led the GPU implementation. Therefore, he was responsible for steering the algorithm modifications towards real-time implementation. To give the reader an idea of the significance of this work, the first GPU implementation

produced from the algorithm description was more than hundred times slower than the final figures reported in the publication. The first GPU implementation was done according to an OpenCL best practices book [Sca12]. While working on the GPU implementation the Author constantly shared ideas with Mr. Kalle Immonen who was working on the MATLAB implementation in the same room.

The ideas and the code for the third publication [**P3**] and its journal extension in the fourth publication [**P4**] were mostly developed by the Author. However, the Author was discussing his ideas with Mr. Kalle Immonen throughout the process.

The main ideas of fifth publication [**P5**] were developed by the Author, inspired by the previous work [**P3**; **P4**] and log-polar space [Men+18]. The Author did most of the Visual-Polar space related algorithm development and most of the BMFR modifications in this paper. While working on the project, the Author constantly discussed his ideas with Mr. Atro Lotvonen and the Author received many fruitful comments from him.

There are also multiple other publications which could have been considered to be part of the same project as this PhD thesis, but which were not included into the thesis [**Kos+15**; **Kos+16**; **LKJ20**; **Mak+19**; **Vii+18a**]. The main reason for not including them was that they either did not fit under the same title as well or the contribution of the Author was not as significant as in the included publications. The citations to publications to which the Author has supervised or contributed to are marked in bold font within this thesis.

1.4 Structure of the Thesis

This thesis is comprised of an introductory part and original publications. First, Chapter 2 introduces some background about the human visual system and path tracing. The emphasis is on the techniques required for making path tracing fast enough for real-time applications. This chapter extends and updates the literature review published in [**P1**]. Next, the state-of-art of the real-time path tracing reconstruction and its relation to [**P2**] is explained in Chapter 3. Real-time reconstruction is a fundamental part of foveated path tracing because it removes noise from the path-traced frames and also improves temporal stability. Related foveated rendering work and their relation to [**P3**; **P4**; **P5**] are introduced in Chapter 4. Finally Chapter 5 concludes the introductory part, summarizes the main results of the thesis and cov-

ers some possible future work. All five original publications [**P1**; **P2**; **P3**; **P4**; **P5**] can be found at the end of the thesis.

2 BACKGROUND

This chapter provides some background of the topics related to foveated path tracing. If you are familiar with the human visual system and rendering, especially path tracing please skip directly to Chapter 3. The first part of this chapter is dedicated to how humans see the world (Section 2.1) and the rest is dedicated to generating frames of virtual 3D worlds in real time (Sections 2.2, 2.3 and 2.4).

2.1 Human Visual System

The *Human Visual System* (HVS) has many interesting features several of which can be taken into account when rendering images with a computer for it [SRJ11; Wan95]. The requirement of high fps, low latency, and high resolution makes real-time generation of frames for VR devices a very demanding task. Therefore, it would be useful to find limitations in the HVS which could be used make the rendering task less computationally heavy without perceivable quality decrease.

Humans have a horizontal *field of view* (FOV) of approximately 190 degrees [Wei+17]. Typical desktop display setups only cover a small portion of the total FOV. In contrast, when using a VR device users wear HMD, which react to their position and orientation so that the users feel immersed in a 3D world. For better immersion, devices covering almost the whole FOV of the HVS have been built [Vrg18]. The high FOV comes with a cost: the HVS can detect up to 60 pixels per degree [Wan95] and therefore the resolution of device must to be high. Otherwise the user can distinguish individual pixels.

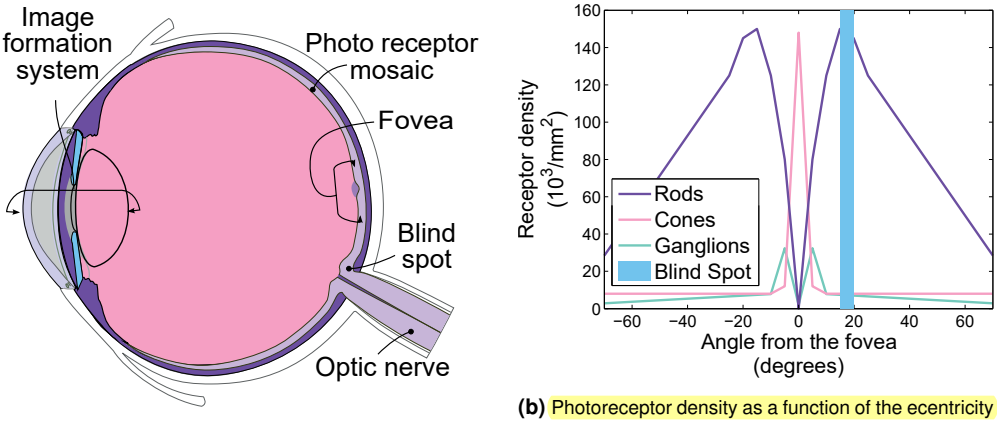
After having some idea of the scale of the required resolution for fully immersed VR experience it is important to know how quickly new frames are required. Approximately 15 *frames per second* (fps) is enough for performing perceptual tasks [CT07]. Lower fps is seen as sequence of still images. However, the motion appears to be smooth only with 24 fps or more and, therefore, movies have been using

that frame rate [Wil+15]. Typically, computer games are considered to be *real-time* if their frame rate is 30 fps or higher. However, higher fps improves the immersion especially with the VR devices. Moreover, fully immersive VR may require even 95 fps [Abr14]. Also in some cases VR system needs to have the total latency of less than 20 ms [Abr14]. However, another study measured that a total latency around 70ms is fine even if the system reacts to the user's gaze direction [Alb+17].

In general, eyes can be in three different states [Kow11]. Firstly, eyes can be in fixation, in other words focused on some object. Even during fixation, eyes make small movements called *microsaccades*, which are used to maintain visibility of the otherwise fading image [PR10]. Secondly, eyes can be smoothly pursuing some moving object and, thirdly, eyes can be in a fast movement called saccade from a fixation to another. During saccades, the human brain does not register the eye signal [HF04]. Therefore, even the orientation of the VR world can be slightly altered during the saccades [Sun+18]. In this manner, users can be tricked into thinking they are walking in a straight direction even though the system is making sure they don't walk into real world obstacles by altering the orientation. More interestingly in the context of rendering optimization the frame quality could be reduced during the saccades. However, occasional easements can be used just to save power usage and not to improve fps. Another option is not to reduce rendering quality, but instead predict where the gaze is going to land. Based on the prediction, the rendering can start before the saccade ends [Ara+17; Mor+18].

The human eye has three different types of photoreceptor cells: cone cells, rod cells, and ganglion cells. Cone cells can be further divided into three different types based on which wavelengths they detect. This mechanism is how humans sense colors. Rod cells are specialized in detecting the brightness. On their own, ganglion cells are only able to detect ambient brightness. However, the data from the cone and rod cells go through ganglion cells [DH14]. There are fewer ganglion cells in comparison to other photoreceptor cells [CA90; Cur+90] and therefore they act as low pass filter directly in the photoreceptor mosaic. The distribution of different photoreceptors can be seen in Figure 2.1b.

The photoreceptor cell distribution immediately shows one source for potential rendering optimization: the resolution of the HVS decreases significantly, when the objects are further away from the visual fixation point. The resolution decrease is mainly due to fewer photoreceptor cells in the periphery but also poorer optical



(a) Illustration of the human eye [CM07]

(b) Photoreceptor density as a function of the ecentricity angle [CA90; CG66; Cur+90; Wan95]

Figure 2.1 The human visual system in more detail

quality of the lenses in the edges of the image formation system reduces the resolution [Cog+18; Thi87]. There have been many studies measuring this resolution as a function of eccentricity angle, such as [AET96; Red97; Sch56]. This effect can be also called cortical magnification [RVN78; SRJ11]. If we assume a situation with the maximum contrast, in other words, the image changes from completely white to completely black in every other pixel, the detection resolution as a function eccentricity angle can be called visual acuity function [Red97]. If the same function is modeled as a function of the stimulus contrast instead of eccentricity, it can be called contrast sensitivity. The combination of the two functions is defined by W. Geisler and J. Perry [GP98].

As long as the user's gaze point can be measured, the reduced HVS resolution in the periphery allows reducing rendering quality in that area. This rendering optimization is called foveated rendering, because the area with the most accuracy is called the fovea, which can be seen Figure 2.1a. Based on the visual acuity model, theoretical upper bound to the resolution reduction states that 95% of the rendering work is excessive [**P1**]. This estimate assumes comparison to constant full resolution rendering over the whole FOV. However, in reality HVS is more complicated and just reducing the resolution in the periphery does not work perfectly [AGL19].

In the periphery, luminance information is more important than color information because there are fewer cone cells compared to rod cells. Moreover, the detection of temporal flickering artifacts stays uniformly about the same across the whole visual field [Kel84]. Therefore, temporal stability requires extra care in the peripheral

parts of a foveated rendering system where sparse sampling easily produces flickering.

What is interesting is that the HVS can detect the presence of a pattern in the periphery before actually resolving it [AET96; TCW87]. Therefore, the required rendering can be reduced even more if contrast is added to periphery, even though that might generate patterns which are not correct [Pat+16]. However, it is important that these patterns are temporally stable and fade quickly when the gaze point moves closer to them.

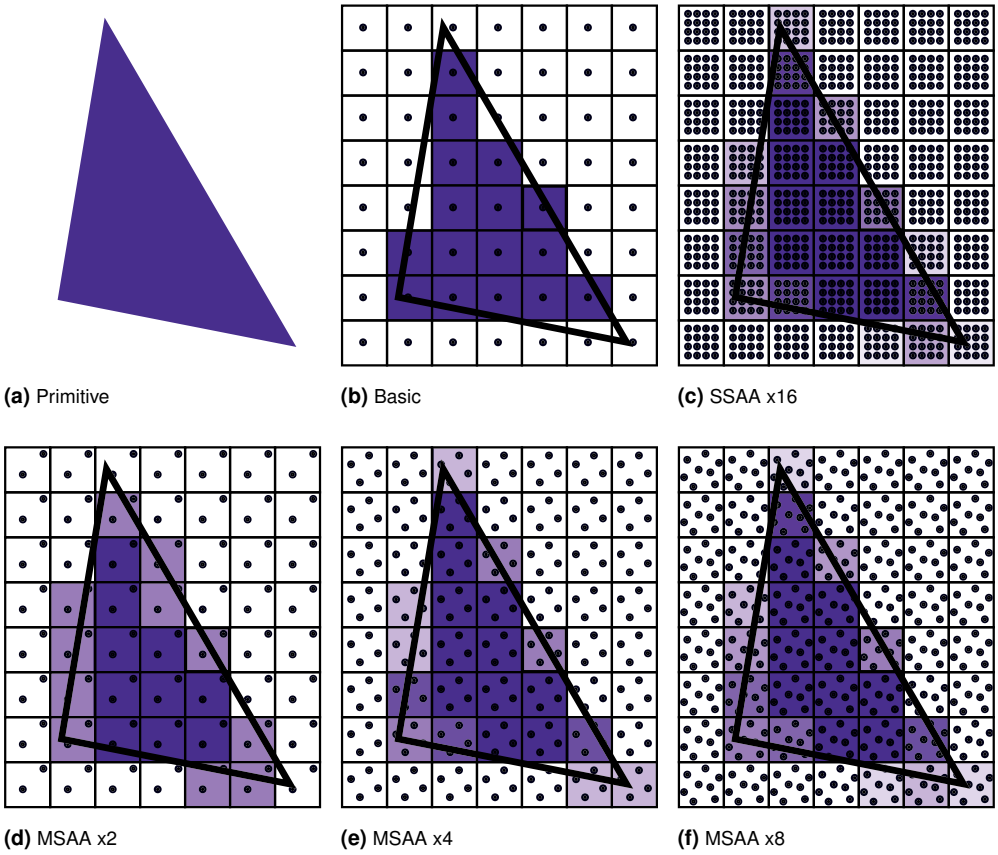
2.2 Rasterization

Rasterization is a way to generate images for the HVS in real time. In this thesis, rasterization is mostly not used, but it is important to know how it works since rasterization has been used for most real-time graphics since the launch of first consumer level GPUs. Therefore, most of the previous work on foveated rendering is rasterized. Also, the results of real-time path tracing are typically compared to rasterized results and current state-of-the-art real-time path tracing systems use hardware accelerated rasterization for primary ray traversal.

The idea of rasterization is to determine the visibility of a 3D primitive, for example, a triangle. The determination is done for a grid of samples, for example, pixels on the computer screen. Typical restrictions of rasterization are that all the samples must have a common origin and their directions need to be aligned in a perfect grid.

The common origin restriction can be relaxed by doing multiple passes of rasterization which can be used, for example, for environment map reflections of a car in a racing game [BN76]. This could be done by first rendering the same scene with a 360-degree camera in the location of the car. Then the main camera can be rendered and while it is shading the reflections of the car it can use colors from the previously rendered 360° frame. However, these techniques typically have problems with near objects [Hug+14, p. 550] and do not support showing the reflecting object itself in the reflection. There are also other ways to loosen the common origin restriction like multi view rendering extensions, but they do not give full flexibility to decide the origins of every sample completely freely.

GPU hardware also supports loosening the perfect grid alignment of the directions requirement. For instance, *Multisample Anti-Aliasing* (MSAA) generates softer



(a) Primitive **(b)** Basic **(c)** SSAA x16

(d) MSAA x2 **(e)** MSAA x4 **(f)** MSAA x8

Figure 2.2 Some of the hardware or driver accelerated rasterization visibility sampling patterns and the pixel colors they produce visualized. Black dots show the visibility sample locations. Only SSAA computes shading for every visibility shading location. Other techniques use one shading computation per pixel.

primitive edges by computing more visibility samples in the grid cells containing edges [Ake+18, pp. 139-143]. For instance, MSAA x4, which computes four visibility samples in those cells, uses a rotated grid sampling pattern. Visibility and shading are decoupled in MSAA. Even if there are multiple samples of the same primitive in the same pixel, only one shading is applied. Figure 2.2 shows some of the hardware accelerated rasterization visibility sampling patterns. *Super Sampling Anti-Aliasing* (SSAA) is equivalent to using a higher resolution and computing the average of each group of pixels. For instance, SSAA x16 multiplies height and width by four and computes the average of 16 adjacent samples. SSAA x16 is used as a reference in many of the anti-aliasing research. Current games typically use *Temporal Anti-*

Aliasing (TAA), which jitters the camera and does temporal accumulation [Kar14]. The main motivation for TAA is that anti-aliasing is done later in the pipeline, which reduces the count of shaded fragments.

In addition to hardware accelerated anti-aliasing, the newest generations of GPU hardware support *Variable Rate Shading* (VRS) which allows the application developer to control the sampling for every cell of the frame individually. The shading rate can be even set to be less than one sample per pixel [Har19]. In any case, the sample directions are still in some kind of grid, but the resolution of the grid can be altered on a coarse cell level.

2.3 Ray Tracing Basics

Both the common origin and the grid alignment restrictions introduced by rasterization are removed in ray tracing-based techniques, where rays can have any origin and any direction. In a sense, rasterization can be thought to be a subset of ray tracing which is limited in order to enable better hardware support. For a computer science perspective, the main difference is that the order of the loops is different in rasterization and ray tracing. In rasterization, it is determined which pixels should be colored for each primitive. In ray tracing, it is determined which primitive is the closest primitive in front of each pixel. In addition, ray tracing techniques can recursively continue ray tracing from the found ray object intersection point. Path tracing is a special category of ray tracing where some of the ray parameters are decided randomly.

Path tracing is a ray tracing-based technique which uses Monte Carlo integration to approximate the rendering equation [Kaj86]. What makes path tracing interesting is that it naturally supports all the effects which are hard for rasterization-based techniques, such as soft shadows, global illumination, reflections, and refractions.

Other commonly used ray tracing methods are *ray casting*, *Whitted-style ray tracing* [Whi80] and *distributed ray tracing* [CPC84]. Ray casting only sends out a primary ray from every camera pixel and does not include any recursion. If the ray traversal supports returning multiple intersections with the scene, ray casting can be used for rendering transparent data, for instance, in medical applications [Had+05]. Whitted style ray tracing introduces recursive secondary and shadow rays to ray casting [Whi80]. Therefore, it allows perfect mirror-like materials and hard shadows.

Distributed ray tracing is sometimes called *Cook-style ray tracing* after its inventor. Cook-style ray tracing extends Whitted style ray tracing to support glossy reflections and soft shadows by tracing multiple secondary rays and multiple shadow rays [Coo84]. However, the number of required rays grows exponentially making distributed ray tracing out of reach for general real-time applications. The advantage of path tracing compared to distributed ray tracing is that the number of maximum required rays per bounce is small and known beforehand. However, this comes with the price of having noise in the result.

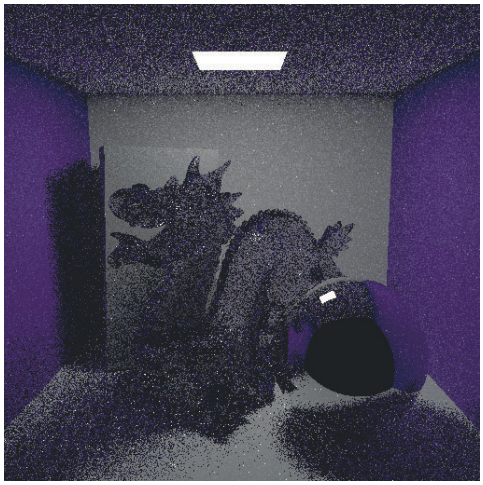
The first path-traced games were demonstrated already at the start of the 2010s [BS13]. However, at the time it required multiple GPUs to run the game and, therefore, path tracing was mainly used for offline rendering [Kel+15]. Just recently the first path-traced visually pleasing games on consumer level hardware have emerged [Sch19]. This is partly due to dedicated ray tracing hardware in consumer GPUs [Kil+18] and partly due to advances in the research on the field [Sch+17; YKL17]. However, rasterization is still a faster way to determine visibility for a regular grid of samples that have the common origin. Therefore, it is typical to use rasterization hardware for the primary rays and then continue recursive ray tracing based on the rasterized G-buffer data, which contains for instance position, normal and material details of the first encountered surface for every pixel [Bar18; Mar+17; **P2**; Sch+17].

2.4 Path Tracing Theory

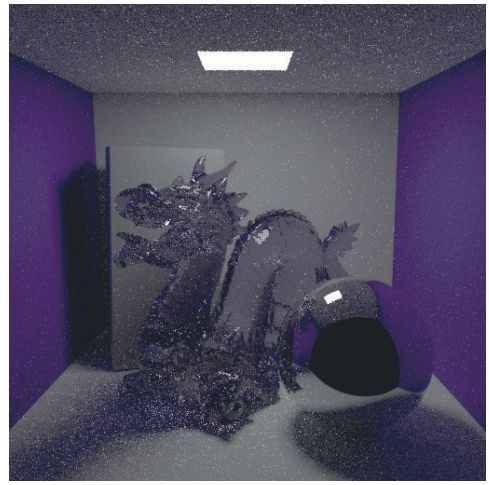
In this section, some basic principles of path tracing are presented. The emphasis is on techniques necessary to implement real-time path tracing. Therefore, factors such as the wavelength and the time have been omitted from this description. The resulting rendering equation can then be written as

$$L_o(\mathbf{x},\omega_o) = L_e(\mathbf{x},\omega_o) + \int_\Omega f_r(\mathbf{x},\omega_i,\omega_o) L_i(\mathbf{x},\omega_i)(\omega_i \cdot \mathbf{n})\, d\omega_i\,, \qquad (2.1)$$

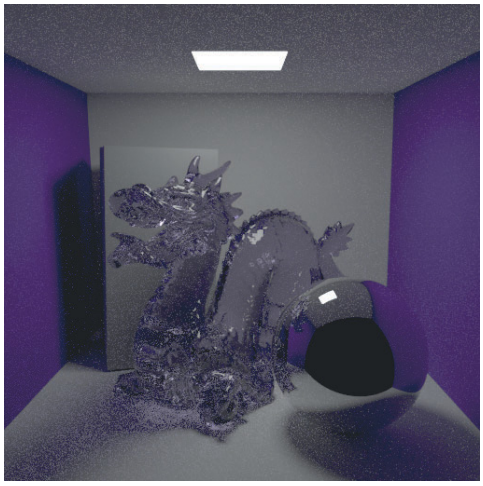
where $\mathbf{x}$ is a point in 3D space, ω_o is an outgoing light direction, Ω is all possible directions, ω_i is an incoming light direction, and $\mathbf{n}$ is a surface normal. Then the function $L_o(\mathbf{x},\omega_o)$ is the luminance going out from the point $\mathbf{x}$ towards ω_o direction, $L_e(\mathbf{x},\omega_o)$ is the luminance emitted to the direction, $f_r(\mathbf{x},\omega_i,\omega_o)$ is the material properties described in *bidirectional scattering distribution function* (BSDF),



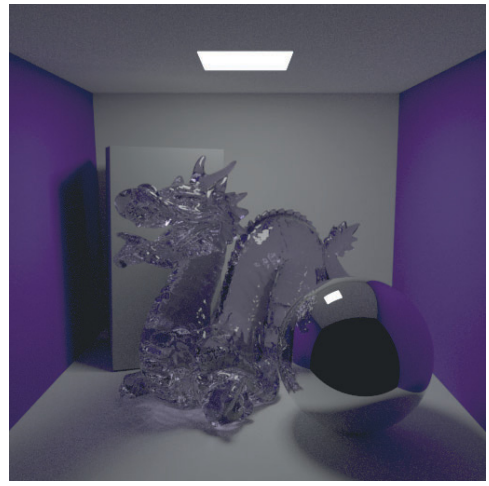
(a) 1 spp



(b) 16 spp



(c) 256 spp

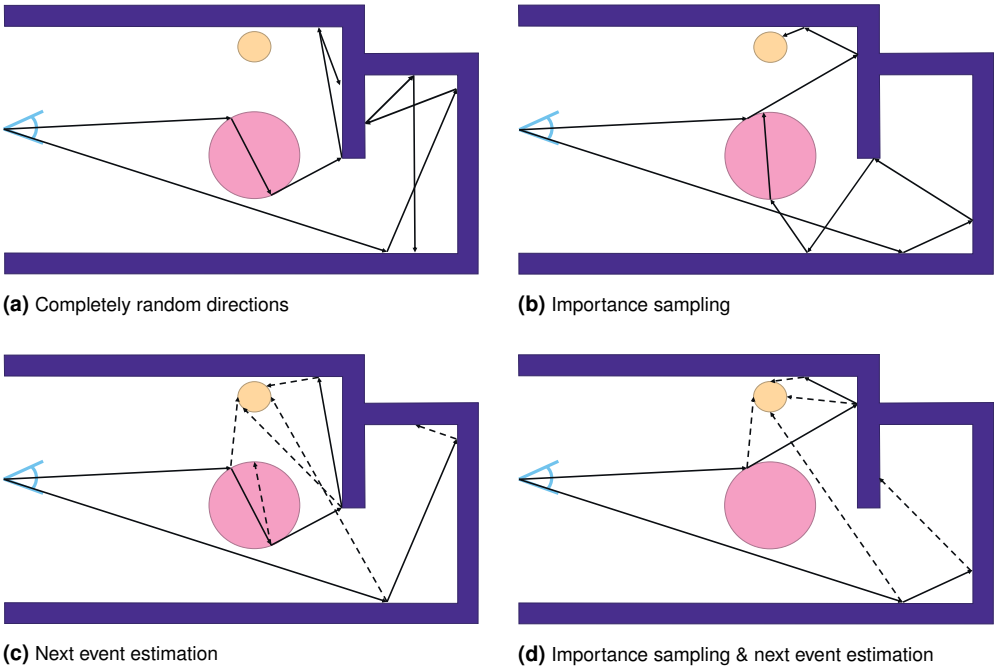


(d) 4096 spp

Figure 2.3 Example images with different *sample per pixel* spp counts depicting the same 3D scene. Every path was allowed to have a maximum of 12 bounces. Lower spp images seem darker because for visualization purposes the colors need to be clamped to low dynamic range image. Individual samples contain brighter data compared to the clamp maximum, which makes averaged colors brighter.

$L_i(\mathbf{x}, \omega_i)$ is the incoming luminance from direction ω_i, and $\omega_o \cdot \mathbf{n}$ is the attenuation factor. [Kaj86]

The interval of the integral in Equation 2.1 is over every possible direction. Moreover, the integral is recursive, meaning that in every possible visible surface point the



(a) Completely random directions

(b) Importance sampling

(c) Next event estimation

(d) Importance sampling & next event estimation

Figure 2.4 Different path tracing styles illustrated with two paths on an example scene. Only paths that find the light source contribute to the pixel color. Without next event estimation the path needs to be very lucky to find the light source.

same integral needs to be evaluated for all surfaces visible from those points. Therefore, Equation 2.1 does not have a closed form solution with scenes usable in real applications. In path tracing, the correct result of the rendering equation is approximated by taking random samples of the integral and computing the average of the samples. Different *sample per pixel* (spp) counts are visualized in Figure 2.3. Having 1 spp means that in a frame every pixel has traced one path. For correct results the average is also computed over both the spatial and the temporal domains. Spatial domain averaging is used to average different colors in the area covered by one pixel and it generates anti-aliased edges. In contrast, temporal domain averaging is used to simulate camera exposure time and it creates motion blurred results [Coo84].

In practice, using Monte Carlo integration to approximate the rendering equation means that a ray is traced from the point $\mathbf{x}$ towards one ω_i direction. If the ray finds an intersection with the scene, the Eq 2.1 is evaluated at that point and the same process restarts. Now the previous ω_i becomes the new ω_o and the new ω_i direction is decided randomly. Basically, this recursive loop should continue until

a material that does not send any luminance to the ω_o is encountered. If the path encounters any materials which emit light, the contribution of that light source to the pixel of the path can be computed. One example of this process is visualized in Figure 2.4a.

2.4.1 Russian Roulette

Instead of actually continuing the recursion until arriving at a dark material, one typical way is to use so-called Russian roulette method which randomly kills some of the paths [PH10, p. 680-681]. Russian roulette makes convergence of the path-traced image slower. However, it improves efficiency because after many bounces paths' contribution to the final color would be insignificant. Killing some of the paths requires weighting the integrand so that

$$F' = \begin{cases} \frac{F}{1-q} & p > q \\ 0 & \text{otherwise} \end{cases}, \tag{2.2}$$

where F' is the new weight of the integrand, F is the original weight, $p \in \mathbb{R} \mid 0 \leq p \leq 1$ is a random value, and q is parameter chosen by the implementer of the path tracer. $q = 0$ is equal to not having Russian roulette at all and greater q means more killed paths.

2.4.2 Importance Sampling

The *bidirectional scattering distribution function* (BSDF) f_r in Equation 2.1 depends on the material the point $\mathbf{x}$ is simulating. The idea of the BSDF is to tell how much the luminance from ω_i direction is going to affect the final color perceived from the direction ω_o.

The original idea of Monte Carlo integration in path tracing uses uniformly distributed random samples and weights the result based on their probability. The convergence can be made faster by changing the random sample distribution to follow the probability of the samples. Specifically, the samples that contribute more to the final color are more likely to be sampled. As an extreme example, if the material is a perfect mirror, then all the samples are sampled from the mirror reflection direction. Figure 2.4b shows a case where the path tracer has weighted the directions based on

the BSDF and randomly decided directions that are close to the reflection direction. [PH10, pp. 688-693] In addition, it is possible to do importance sampling of the light sources [EK18; EL18]. Moreover, *Multiple Importance Sampling* (MIS) is used when two or more importance sampling strategies are applied at the same time [VG95].

2.4.3 Next Event Estimation

Path tracing cannot produce any luminance if the path does not intersect any light sources, that is, a surface for which the L_e term is greater than zero. Therefore, one common way for making the convergence faster is to use next event estimation, which samples one random point in one random light source from every intersection found from the scene. This process is visualized in Figure 2.4c and Figure 2.4d. Most importantly, next event estimation does not introduce bias to the results [VG95].

2.4.4 Ray Traversal

Ray traversal is the process of finding the closest intersection for a ray or a group of rays. Typically this process is accelerated using a tree structure called *Bounding Volume Hierarchy* (BVH), which stores a bounding volume for each tree node. Entire branches of the tree can be rejected with a ray-bounding volume test because, if the ray misses the bounding volume, it is then known that it will not intersect any geometry within the branch.

Ray traversal is an important part of the path tracing process, because generating the first noisy estimation of a frame already requires millions of traced rays. Even for one bounce of path tracing four rays per pixel are required: one primary ray, one secondary ray, and two shadow rays one from each intersection with the scene. Therefore, there has been extensive research on the area of fast BVH traversal, for example, by using standard data types to store the information with fewer bits [**Kos+15**; **Kos+16**; **Kos15**] or by using a custom data type specifically designed for BVHs [Kee14; YKL17]. Even dedicated hardware units for BVH traversal have been proposed [Kee14; Lee+13; **Vii+16**].

Offline construction of high quality BVH for a static scenes is typically done with *Surface Area Heuristic* (SAH) [WBS07], which minimizes the total surface area of the bounding volumes on every level of the tree. The surface area estimates how likely

a random ray would hit the volumes.

In contrast, for dynamic content, the BVH quality is not as important as the speed of updating or rebuilding the BVH [Vii18]. Updating the BVH is adequate if the overall structure of the animated object does not change significantly [**Vii+17a**; Wal+09]. However, for keeping the BVH quality sufficient, for example, in an explosion animation, completely rebuilding the BVH is required. Some examples of quick build algorithms are *Hierarchical Linear BVH* (HLBVH) [PL10], which uses Morton order curve bit patterns of the triangle centroids for constructing the hierarchy, and *Parallel Locally-Ordered Construction* (PLOC) [MB18], which improves the quality by sweeping through the Morton ordered primitives and constructing the best BVH nodes within a small local window. Both algorithms are well suited for low-power hardware implementations [**Vii+15**; **Vii+17b**; **Vii+18b**].

2.4.5 Current Bottlenecks

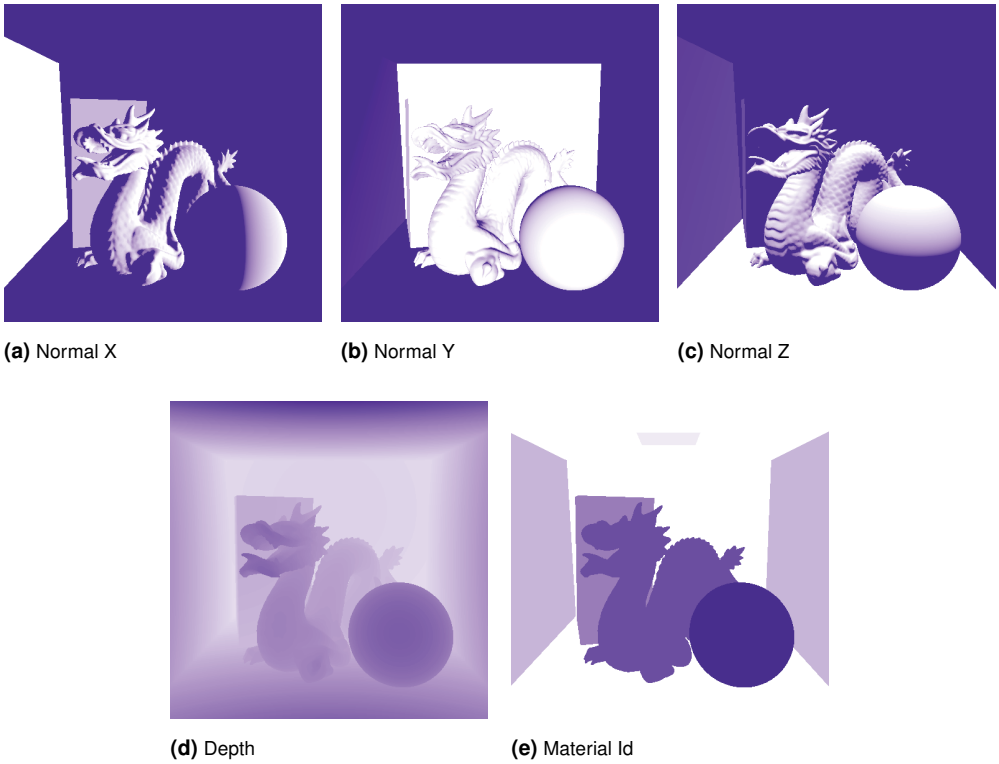
In the Author's experience, due to extensive research in the area of ray traversal, the material interaction computations, specifically shading, currently dominate the path tracing timings. Shading depends on the material of the surface and, therefore, depending on the path-traced scene it can be very divergent work. The amount of divergence can be reduced by sorting the rays [GL10]. However, even if the rays that intersect the same material are in the same *Single Instruction Multiple Data* (SIMD) or *Single Instruction Multiple Thread* (SIMT) lane it does not help with the divergence of the expensive texture fetches. In addition, shading work is typically modifiable by the developers and, therefore, it is hard to make any better dedicated hardware for them than programmable shading cores of the GPUs. Furthermore, current hardware accelerated ray tracing APIs hide the details of the ray traversal and BVH building from the developers. For these reasons, it is interesting to look at the different ways how one could reduce the amount of path tracing work in general. Some ideas for reduction, which will be covered in more detail below, are reconstructing a visually pleasing frame from just a few Monte Carlo samples as well as reducing paths in the peripheral parts of the user's vision.

3 REAL-TIME PATH TRACING RECONSTRUCTION

Monte Carlo integration in path tracing produces an estimation of a pixel's final color value which contains variance. The variance is seen as noise in the output frame. If multiple samples are averaged, the amount of noise decreases. Halving the signal-to-noise ratio requires quadrupling the number of samples [Vea97, p. 39]. Therefore, there is always a point when a denoising algorithm can generate a perceptually perfect image with less work compared to actually tracing more rays. In consequence, even offline movie renderings typically use denoisers for getting rid of barely visible noise after hundreds or even thousands of samples per pixel [God14]. Denoising is even more important part of the path tracing pipeline in the real-time context where the sample budget is significantly lower.

In this chapter, different denoising algorithms that are suitable for real-time path tracing are introduced. Most of the work relevant to only offline rendering is intentionally omitted since the scope of this thesis real-time path tracing. The system cannot know for sure beforehand where the user is going to look at and therefore, it is hard to optimize offline prerendering work with the idea of foveated rendering. Also since there is no hard timing limit in offline rendering, there is no need to do this kind of optimization with it. However, pointers to some of the most interesting offline reconstruction algorithms are provided, which could be bases for future real-time algorithms.

In the path tracing context, denoising is typically called *reconstruction*, because in contrast to conventional digital photo denoising, path tracing reconstruction has access to more data than just the output frame. A more in-depth survey of the different path tracing reconstruction work can be found in the survey paper by M. Zwicker et al. [Zwi+15].



(a) Normal X **(b)** Normal Y **(c)** Normal Z

(d) Depth **(e)** Material Id

Figure 3.1 Examples of different feature buffers produced as a side product of path tracing because the data is required for shading. The reconstruction algorithm can use these buffers, for example, to detect edges. Purple color means zero or less and white color means one or more. For instance, Normal X buffer is the X component of the first bounce surface normal. For visualization, Depth and Id buffers were scaled to be in the range from zero to one.

3.1 Concepts

This section introduces a few key concepts which can be used as basic building blocks with most of the real-time reconstruction algorithms described later in this chapter.

3.1.1 Feature Buffers

A path tracer can store information about the 3D scene and the reconstruction algorithms can use this information for guiding their reconstruction process. Most importantly this feature information is often completely *noise-free*. Examples of feature buffers are all G-buffer channels, specifically, surface normals, positions in the

3D world, surface roughness, material albedo, etc. Some examples of these buffers can be seen in Figure 3.1. For faster runtime in contemporary real-time applications feature buffers and primary rays are typically computed with rasterization hardware [Bar18; Mar+17; **P2**; Sch+17].

In contrast, photograph denoisers must rely completely on noisy 2D bitmap data. This could also be the case if the path tracer has motion blur or depth of field simulation, which generate noise also to the feature buffers. Some options for working without any noise-free data is to fit polynomials to the data or find similar areas from the image and use them to arrive at a noise-free estimate [DFE07]. However, at the time of writing, motion blur and depth of field are out of reach of real-time path tracing and they are generated with post-processing estimation techniques [GMN14; YWY10].

3.1.2 Motion Vectors

In path tracing, the exact parameters of the simulated camera are known and the world positions or depths of the first intersections can be stored. With this information, previous frames and other viewpoints can be projected to the current camera location and orientation. Motion vectors can also support simple animations as long as there is a way to find out where the point was in screen space in the previous frame. The position can be computed, for instance, if the animation is constructed from a set of basic matrix operations like translations, rotations and scalings. What makes the camera parameters *exact* is that the camera's parameters like the position are known down to the accuracy of the used data type. Similar information can be extracted from just a video stream [SB91]. However, this requires a lot of memory traffic and from the video it is hard to acquire the information as accurately and without noise.

Reprojection gives us per pixel motion vectors, which denote where in the screen space the world space position of a pixel was in the previous frame. With 1 spp frames, sampling history data based on motion vectors and computing the exponential moving average can give results that are similar to 10 spp frames [**Mak+19**]. This requires thresholds, for example in sample's normal and position temporal change for realizing if the point was occluded on the previous frame. Otherwise there are so called ghosting artifacts where the foreground data is mixed with the background.

Reprojection can also be used in the spatial domain if multiple views are generated, for example for a stereo HMD or for a light field display.

Motion vectors can be computed for different components of lighting separately [Zim+15], which preserves effects such as reflections. However, this complicates the motion vector and luminance computations, since components need to be stored separately and, therefore, it is difficult to use the technique in real-time with contemporary hardware.

There are at least two drawbacks with the use of reprojected data. Firstly, the quality varies across the screen, since reprojection cannot be done on areas that were occluded on the previous frames. To be more precise, if the reconstruction algorithm uses reprojected and accumulated frames it must support varying quality inputs. Secondly, using temporal previous frame data introduces temporal lag to the illumination changes. Depending on the parameters, the lag can, for example, be 10 frames long [**P2**]. A lag of 10 frames can be invisible to the user in some cases, but for example a light source flashing on every other frame would appear to be constant and half as bright as it really is.

There is a solution which removes the temporal lag [SPD18]. The idea is that one path tracing sample in every block of pixels is path-traced with the same random seed as in the previous frame. Using the same seed means that, if the illumination conditions are the same, the sample generates the same result as in the previous frame. If the result is different it means that the illumination has changed and, in that case, the temporal data can be discarded. Basically, the algorithm falls back to the first frame 1 spp quality in areas where there are changes. Interestingly this technique also removes ghosting from reflecting surfaces, because also they fall back to 1 spp quality when the camera is moving. However, current real-time reconstruction algorithms are not good enough with just 1 spp input and there will be artifacts. The severity and the type of the artifacts is determined by the used reconstruction algorithm. Another problem is that generating the same sample as in the previous frame requires altering the sub-pixel offsets per pixel, which is not supported by the fastest primary ray computation method of hardware accelerated rasterization.

Reprojection can also be used after the reconstruction algorithm. An extra reprojection step makes the results more temporally stable [**P2**; SPD18]. In addition, more temporal stability can be achieved with *Temporal Anti-Aliasing* (TAA) [All+17; Kar14; **P2**; Sch+17]. TAA uses temporal reprojection without discarding

the occluded data and instead it clamps the history sample's luminance to the current frame's neighboring pixels' luminance.

3.1.3 Component Separation

Monte Carlo integration is the sum over incoming light directions. Therefore, it is possible to reconstruct the samples in separate groups, without introducing bias to the results.

One idea is to compute the filtering parameters for two groups, each containing half of the samples, separately and then do so called *cross filtering* [RKZ12]. In cross filtering the parameters computed for the first half are used to reconstruct the second half and parameters computed for the second half are used to reconstruct the first half. The final result is the average of the two reconstructed images. The idea of cross filtering is to reduce over fitting of the filtering parameters. Currently, path tracing two different full resolution sets of samples is unfeasible [All+17; Bar18; **P2**; **P5**; Sch+17], but this could be one interesting direction in the near future, since it can produce good results in an offline context [Bit+16].

Another idea for reaching better quality is to filter the direct illumination and indirect illumination separately [Mar+17; Sch+17] or diffuse and specular component separately [Bak+17]. In those cases, there can be separate reconstruction algorithms specifically tuned for their inputs. For instance, the direct illumination can be generated with noise-free shadow mapping techniques and then there is no need to reconstruct it [Mar+17]. However, in some work [**P2**; SPD18] mainly for faster execution reasons, separate reconstruction was not found beneficial and both of the components are reconstructed at once.

3.2 Cross Bilateral Blur Variants

The first actual reconstruction algorithm introduced in this thesis is cross bilateral blur and its variants which has been optimized better runtime. Bilateral blur is an extension of the basic the Gaussian blur. The difference is that bilateral blur tries to preserve the edges of the content. The problem with bilateral blur is that it is not fast enough with big enough blur kernels for real-time path tracing.

One of the fundamental ways to blur an image is to use Gaussian blur. Gaussian

blur decides the weights of the neighboring samples based only on the spatial distance of the sample to the blurred pixel. The formula for one sample pixel's weight is

$$w(x',y') = e^{-\frac{(x'-x)^2+(y'-y)^2}{2\sigma^2}}, \tag{3.1}$$

where x is the blurred pixel coordinate on the x-axis, x' is the sample pixel coordinate on the x-axis, y and y' are the same variables on the y-axis, and σ is the wanted standard deviation of the Gaussian kernel.

At spatial distances further than 3σ from the blurred pixel, the Gaussian weight of the sample pixels are already more than thousand times lower compared to the center. Therefore, practical real-time implementations can limit sampling area which saves the memory bandwidth, without noticeable difference in resulting quality.

The basic version of Bilateral blur [TM98] extends this formula by introducing the color space distance to it

$$w_b(x',y') = e^{-\frac{(x'-x)^2+(y'-y)^2}{2\sigma_d^2} - \frac{|I(x,y)-I(x',y')|}{2\sigma_r^2}}, \tag{3.2}$$

where $I(x,y)$ is the color value at the blurred pixel. Note that there are separate standard deviation factors σ for the distance and the color value.

Bilateral blur can be extended to use other information than just the spatial and color space distance. This is called cross bilateral filtering [ED04; Pet+04]. Moreover, the color space distance varies a lot in path tracing noise and therefore it is not very useful information in the path tracing reconstruction case. So, for example, the distance from the camera to the first intersection and surface normals on that point typically contain useful information about the possible edges in the 3D scene [Dam+10]. The weight $w_b(x',y')$ from Equation 3.2 must be multiplied with the weights from these buffers

$$w_c(x',y') = w_b(x',y') \cdot w_z(x',y') \cdot w_n(x',y'), \tag{3.3}$$

where $w_n(x',y')$ is the weight from the normal buffer and $w_z(x',y')$ is the weight from distance to the first intersect buffer specifically the depth buffer.

One good way to compute the weight from the normal buffer is

$$w_n(x',y') = max(0, n(x,y) \cdot n(x',y'))^{\sigma_n}, \tag{3.4}$$

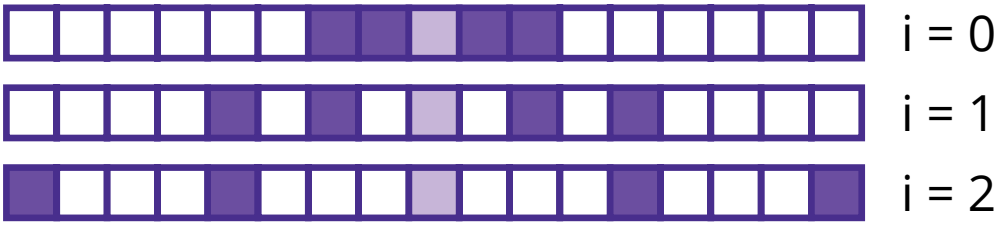


Figure 3.2 An illustration of sampling pattern on three first iterations of 1D À Trous filter with kernel window size of 5. The light purple pixel is the target pixel which also sampled from the input and where the bilateral blur result is stored in the output.

where $n(x,y)$ is the normal vector, in other words, three values $\in \mathbb{R} \mid -1 \leq p \leq 1$ of the closest surface in front of the pixel x,y [Sch+17].

The weight from the depth buffer can, for example, be

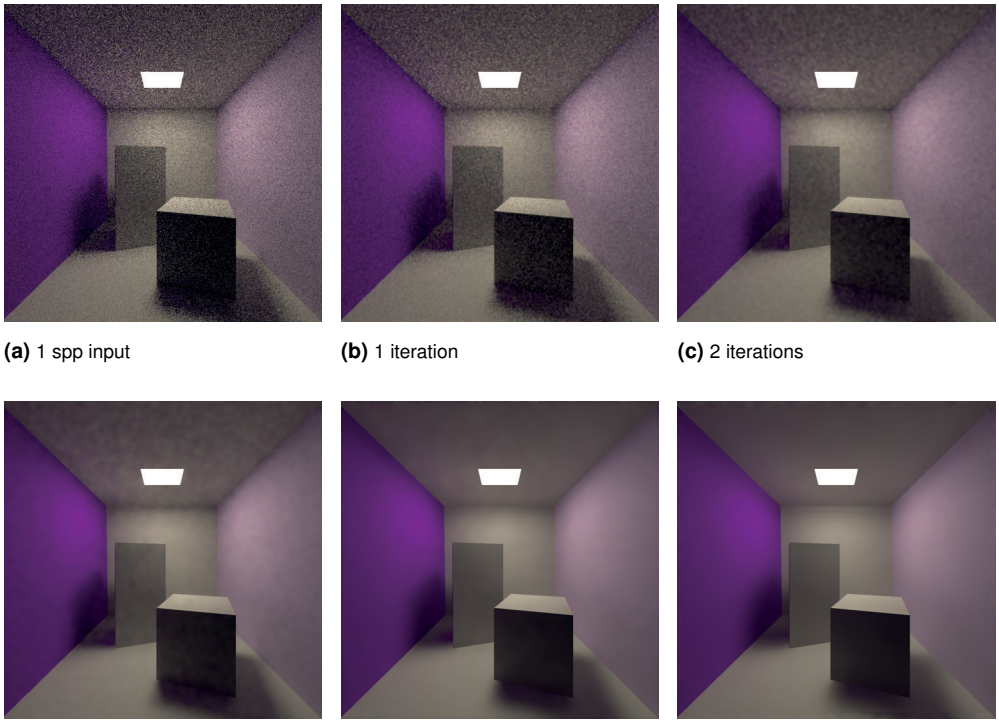
$$w_z(x',y') = e^{-\frac{|Z(x,y)-Z(x',y')|}{\sigma_z|\nabla Z(x,y)\cdot[x-x',y-y']^\mathsf{T}|+\epsilon}}, \tag{3.5}$$

where $Z(x,y)$ is the depth buffer value at the pixel x,y, $\nabla Z(x,y)$ is the gradient of the depth, and ϵ is used to avoid division by zero [Sch+17].

Multidimensional Gaussian blur can be optimized by separating it to separate passes per axis, which reduces the number of expensive memory access from $\mathcal{O}(n^m)$ to $\mathcal{O}(m \times n)$ where n is the blur kernel diameter and m is the count of dimensions. In contrast, Bilateral blur cannot be separated because blur on one axis affects many pixels on the other axis and one distance cannot be used in the spaces that are used for finding the edges. Therefore, fast timings require some other approximation of the Bilateral blur. One option is to use so called *adaptive manifolds* where the work can be shared between the neighboring pixels, but it requires deciding how many manifods are needed, which affects the quality and the runtime greatly [Bau+15; GO12]. Therefore, currently some sparse versions of the bilateral blur like the *À Trous Filter* are typically used [Dam+10; **Imm17**; Mar+17; Sch+17]. The fast timings are achieved by not sampling every intermediate pixel.

3.2.1 À Trous Filter

The idea of the À Trous filter [Bur81] is to run multiple passes over the image, which all blur different frequencies. Therefore, À Trous filter is also called a dis-



(a) 1 spp input **(b)** 1 iteration **(c)** 2 iterations

(d) 3 iterations **(e)** 4 iterations **(f)** 5 iterations

Figure 3.3 An example of how every iteration of the À Trous filter blurs lower frequencies of the 1 spp input frame. The kernel size is 5×5 and no variance or temporal data is used. The 5th iteration starts to show typical artifacts of basic À Trous filter around the area where the back wall and ceiling meet.

crete wavelet transform [Fow05]. Every pass uses the same bilateral blur window, but what changes between the iteration is the amount pixels in between the sampling locations. More specifically, every iteration makes the bilateral filter more sparse, as can be seen in Figure 3.2. Sparse filtering makes it possible to blur with a bigger blur window without actually sampling all of the values in the area. Also, À Trous can be extended to use the feature buffers for edge stopping [Dam+10]. In the Figure 3.3, À Trous filter is applied to a 1 spp frame using albedo, normals, and world position feature buffers as guidance. Sometimes the typical À Trous artifacts visible in the Figure 3.3 can be avoided if iterations are applied in reverse order [QWH12].

(a) 1 spp input **(b)** BMFR [**P2**] **(c)** SVGF [Sch+17]

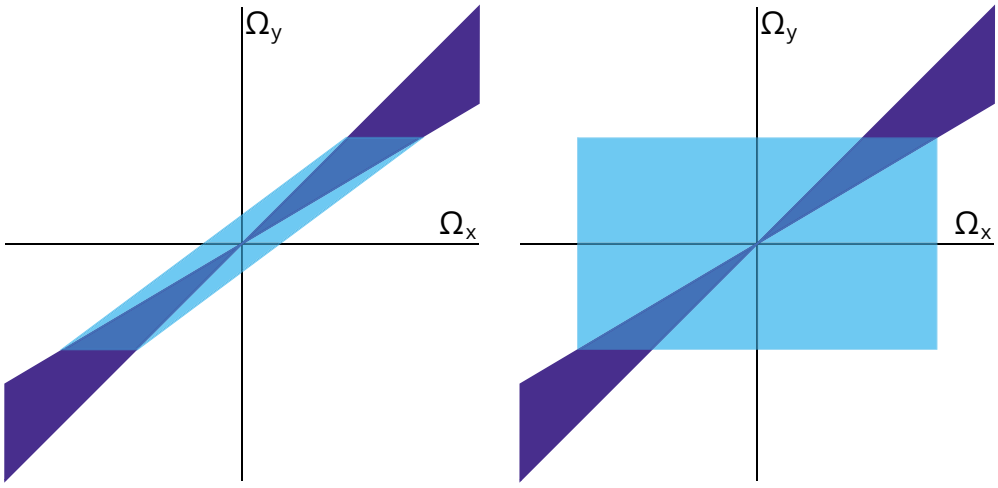
Figure 3.4 The same scene as in Figure 3.3 reconstructed with BMFR and SVGF algorithms. BMFR algorithm produces a halo around the light source that could be reduced by giving BMFR a feature buffer containing a similar pattern as the luminance in the reference such as 1-bit buffer indicating if the material is emitting light. SVGF produces some luminance bumps on the walls and À Trous style artifacts both of which could be reduced by tweaking the parameters.

3.2.2 Spatiotemporal Variance-Guided Filtering

In path tracing reconstruction one interesting edge stopping feature buffer is the per pixel variance. With the variance buffer the certainty of the sample luminance can be used to increase the sample's weight. If there is just one sample, variance cannot be estimated. Therefore, *Spatiotemporal Variance-Guided Filtering* (SVGF) [Sch+17] uses temporal data for the variance estimation and in case of occlusion the variance is estimated spatially in screen space. In addition, SVGF updates variance estimation on every iteration of the À Trous filter. An example of the quality of SVGF can be seen in Figure 3.4c. Notice how the use of temporal data and the variance estimation removes edge artifacts and improves the contact shadow quality of the smaller box compared to the original À Trous filter shown in Figure 3.3.

3.3 Sheared Filtering

Path tracing effects such as soft shadows produce noise spectra in the frequency domain which look like double sided wedges as visualized in Figure 3.5. The wedges can be filtered efficiently with so-called *sheared filtering*, where the filtering pattern is a parallelogram in the frequency domain [Ega+11]. Axis-aligned sheared filtering



(a) Sheared filtering

(b) Axis-aligned sheared filtering

Figure 3.5 Frequencies produced by for instance 1 spp noise of the soft shadow. The blue box represents the filtering pattern. While sheared filtering is the correct filtering pattern, the axis-aligned version of it is faster to compute.

[MWR12] is an optimization of sheared filtering, which can produce good quality results for soft shadows in real time [HA19, p. 314]. The speed is achieved by the ability to run the filter as a 2D filter in screen space. However, the filter must run for every light source separately.

The original sheared filtering algorithm constructs a 4D database of samples and then reconstructs the soft shadow using it. In contrast, in the axis-aligned version the frequency analysis is used to guide the 2D filtering pattern in screen space. Even with the axis-aligned optimization the image converges to the correct result. Axis-aligned sheared filtering can be extended to global illumination where there are multiple light sources and the illumination bounces from materials [Meh+13]. It can also be extended to filter effects that generate noise to the feature buffer [Meh+14].

3.4 Machine Learning

Learning-based methods are interesting in the domain of path tracing reconstruction as they can produce non-linear models and in theory, they can realize how many weights are needed for a linear model in the given situation. Despite the interesting

characteristics of the machine learning-based methods, the runtime of the inference seems to be the problem. Specifically, at the time of writing, there are no published real-time machine learning-based approaches for path tracing reconstruction. However, the authors of [All+17] have indicated that their method runs in real-time on current hardware [Kel+18]. In any case, this section covers some of the most interesting interactive and offline machine learning approaches to reconstruction.

3.4.1 Dataset Generation

Setting up the framework for researching neural networks for path tracing reconstruction is easy. Firstly, it is an image-based problem where one can use some of the thoroughly researched image neural network designs. In addition, it is easy to generate a lot of training data with a path tracer. All that is needed is a path tracer, 3D models and camera paths. Other than generating meaningful camera paths there is no requirement to have humans labeling the data.

There are many ways how data augmentation with path tracing reconstruction can be done [All+17]. For instance, the network can be taught with cropped frames and it is also possible to randomly rotate and flip the inputs. Moreover, better generalization to different camera paths is achieved with randomly stopping the camera path and randomly changing the direction in which they are played.

Many previous works [All+17; Bak+17; KBS15; Vog+18] use noise-free converged frames as the target frame. However, if the network training uses suitable loss function, a recent Noise2Noise idea can be used [Leh+18]. There both the sample and the target frames are noisy. Generating two noisy frames with different random seeds is thousands of times faster than generating a pair of noisy and noise-free frames. This idea could even be utilized in a system that learns to reconstruct the 3D scene in interactive frame rates while the user is flying in it.

3.4.2 Network Designs

The first neural network design proposed for path tracing reconstruction was fully-connected [KBS15]. However, lately *Convolutional Neural Networks* (CNN) have been preferred over fully-connected networks, because they have significantly fewer parameters to learn and they also do not overfit to screen space locations [All+17;

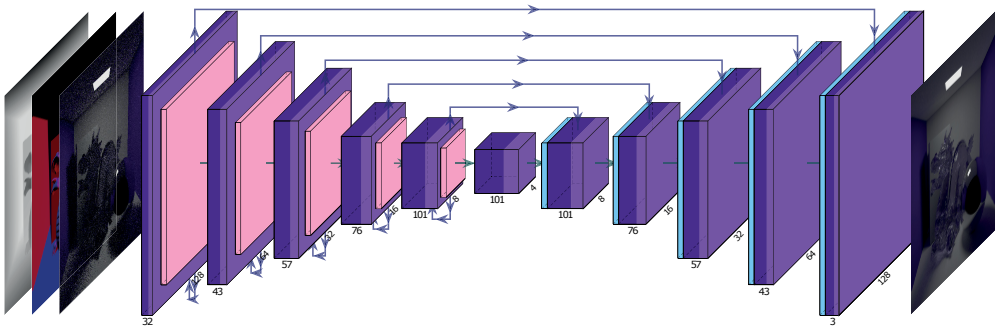


Figure 3.6 Example of U-Net design which can be used to reconstruct path tracing with interactive frame rates [All+17]. Purple shade means convolutions, pink downsampling, and blue upsampling. Note the recurrent connections on every encoder stage and the skip connections from the encoder stages to the decoder stages.

Bak+17; Vog+18]. Typically, feature buffers are given as extra input channels to the network and the network is allowed to learn how to best utilize them. However, also filtering feature buffers with extra convolutional layers and using a separate encoder stages just for them has been proposed [Yan+19].

Instead of directly outputting the final color the network can output kernel weights for every pixel which are then used to construct the final color [Bak+17]. The sum of the weights is forced to be exactly one and each weight is forced to be within range from zero to one by using a softmax activation. The advantage of this is that the final color is always a weighted sum of its neighborhood and not something completely different. In addition, the learning is faster because the kernel weights are scale independent.

The kernel prediction process is inverted in a recent *kernel-splatting network* [Gha+19]. Instead of predicting every output pixel as a weighted sum of noisy pixels, one can predict which pixels should be affected by the noisy samples. This is done individually to every path tracing sample instead of doing it to averaged color of many samples. It is easier to learn to silence outlier samples with the kernel-splatting approach compared to the kernel-prediction. However, the computation requirement is higher, because every sample is done individually.

One possibility from general image neural network literature is to use autoencoders, which are neural networks that combine encoder and decoder stages into one network. Autoencoders automatically learn to compress the data while preserving the most important features of it [LHY08]. Better handling of the high frequency de-

tails in an autoencoder can be achieved with skip connections [RFB15]. The idea of the skip connections is to connect the same sized layers from the encoder to decoder. The connections remove the possibility to use the autoencoder as a data compressor, but they work well in the denoising use case since adding these connections makes the learning faster, reduces vanishing gradient problem and improves the generalization of the network [He+16; RVL12; Yan+18]. In some sources, an autoencoder with skip connections is called a U-net because it can be visualized as a U-shaped graph.

The temporal stability of the network can be improved using a so-called *Recurrent CNN* (RCNN), which adds temporal recurrent connections to the convolutional network design [All+17]. An example of this kind of network design can be seen in Figure 3.6. The idea of the recurrent connection is to give the previous frame state a layer as an input to it or some other previous layer on the next frame. Recurrent connections can also use analytical reprojection for moving the most relevant data to the correct place in the layer [Vog+18]. In both cases the network learns, for example, based on the roughness feature buffer, not to use temporal data when the material suffers from temporal lag.

Also, other ideas can be used directly from image processing networks. One example of this is dropout [Sri+14], which improves generalization by randomly shutting down some of the neurons. Another example is transfer learning, where known to be good network design and weights are used as initial values for the teaching of the network that is tuned for the task in hand. Interestingly the base network can even be targeted to a very different task [TS07].

3.4.3 Loss Function

The selection of the loss function affects the learning of the network significantly. The loss selection is interesting in the real-time context because it is computed only when teaching the network. More specifically, even a very complicated loss will not slow down the inference timings.

S. Bako et al. [Bak+17] tested different loss functions and found out that the absolute value loss function, in other words, *L1* loss is the most robust and closest to perceptual difference. However, the analysis was limited to simple loss functions that can be evaluated quickly and locally. In the case of Noise2Noise training, L2

style loss must be used because its minimum is found at the mean of the samples, which makes it converge towards the correct result [Leh+18].

Temporal errors such as flickering can be penalized with a L1 loss which compares the absolute difference of temporal derivatives of the network output and the reference [All+17].

One idea to enhance fine details of the image is to use the L1 loss in the gradient-domain [All+17]. However, better results can be achieved by comparing internal representations of image classification neural network and using their difference as the loss [KHL19a; Zha+18].

Loss calculations can be improved by applying some non-linearity before the calculation. For instance logarithm makes the loss more robust against bright outliers [Bak+17; KHL19a; Vog+18] and modified gamma correction improves quality in the dark areas of the frame [All+17]. In the case of comparing the classification networks' internal representations, the loss calculation can be made more robust by applying random transformation to the denoiser output [KHL19b].

3.4.4 Optimizing Network for Fast Inference

Faster inference can be achieved by simplifying the overall network design [Ian+16]. Examples of simplifying the network include reducing the sizes of convolution kernels and working with less input and output data on every layer of the network. For example, C. Alla Chaitanya et al. [All+17] use only 7 channels of data as input to the network. Moreover, every convolutional layer uses a kernel size of only 3×3. In addition, the simplification can be done automatically with so-called pruning [LDS90].

A single convolution layer can be made faster by replacing it with a combination of purely spatial convolution and a pointwise matrix multiplication with weights along the depth axis [Sif14; Van14]. In addition, explicit upsampling layers can be made faster with subpixel convolution which increases the size of the output while applying the convolution kernels [Shi+16].

3.4.5 Optimizing Inference of Existing Network

CNNs operate on rather big sets of data and the internal operations are multiplications and additions. Therefore, CNNs can be made faster with dedicated hardware

using a data type that saves memory bandwidth and simplifies multiplication units by using fewer bits [CBD14]. Therefore, major hardware manufacturers have released their own dedicated machine learning cores.

On a general purpose hardware IEEE standard's half-precision floating-point numbers can be used for accelerating CNNs [CBD14; IEE08]. In addition, a dedicated 16-bit floating-point format called *bfloat16* with three extra exponent bits and three fewer mantissa bits has been proposed [HTH19]. The main motivation of *bfloat16* is reduced power consumption and physical chip area with the same network accuracy compared to the original 16-bit format of IEEE. Also the conversion to and from 32-bit single-precision floating-point format is simple since there are as many exponent bits in the both formats.

3.5 Regression

Before the emergence of the machine learning approaches, the state-of-the-art offline path tracing reconstruction algorithms used regression [Bit+16; Moo+16]. The idea of regression-based reconstruction algorithms is to fit the noise-free feature buffer data to noisy path-traced illumination data. Also fitting of polynomials can be used especially if there are no noise-free feature buffers available [Moo+16].

One example of a feature buffer fitting methods is to use simple least squares regression, where the squared difference is minimized. Let noise-free feature buffers be T_m and noisy path-traced data be Z, then least squares regression becomes

$$\hat{\alpha} = \underset{\alpha \in \mathbb{R}^M}{\operatorname{argmin}} \sum_{(p,q) \in \Omega_{i,j}} \left(Z(p,q) - \sum_{m=1}^{M} \alpha_m T_m(p,q) \right)^2, \tag{3.6}$$

where $\alpha = [\alpha_1 ... \alpha_M]$ contains the weight α_m for every feature buffer m, the total count of feature buffers is M, and $\Omega_{i,j}$ is some area around the pixel for which the reconstructed value is computed. Pixels in $\Omega_{i,j}$ can be weighted so that the pixels closer to the target pixel affect the result more.

Improving the quality requires making the model more complicated, for example, by adding more feature buffers that are tailored for a certain light phenomenon [Bak+17]. Typically, these additions improve the quality only slightly while slowing down the process significantly [**P2**]. Moreover, there is problem of overfitting and realizing when the model is overfitting. The pattern may seem like

overfitting, but the model might be following some complicated pattern in the luminance. In an extreme case, the regression model is so complicated that it represents the noisy frame exactly. This can be constrained by introducing penalty to the regression model, but it is yet another parameter to control.

3.5.1 Guided Image Filter

A guided image filter [HST13] is a general edge stopping filter. Guided image filter fits a guiding image to the noisy data. Therefore, it is well suited for path tracing reconstruction and can run with interactive frame rates [BEM11]. The downside is that it either requires dozens of moving window operations over the whole dataset or generating equally many summed-area tables. The moving window cannot be computed in parallel for every pixel, and summed-area tables require the use of a parallel scan pattern, which requires a lot of synchronization. The advantage is that the values are computed per pixel so there exists a potential for good quality. However, this would require using adaptive parameters [**P2**]. The guided filter can also be used for denoising the feature buffers if there is noise generated by depth of field or motion blur [Liu+17].

3.6 Thesis Contributions

In [**P2**], *Blockwise Multi-Order Feature Regression* (BMFR) is proposed. BMFR is a regression-based reconstruction method, which achieves real-time performance. The whole post-processing reconstruction pipeline of BMFR is visualized in the Figure 3.8. The pipeline consist of three different main stages, which all are covered in the subsections below.

The input to the BMFR post-processing pipeline is path traced frame and corresponding albedo, normal, and world position feature buffers. The target of this work is real-time and, therefore, 1 spp path tracing is used. Also the path configuration is chosen based on the real-time requirement. In every path after primary ray there are only one secondary ray and two next event estimation rays one from each intersection point. In other words, there are four rays per pixel and one of them, the primary ray, can be computed with rasterization hardware. The method has also been tested to work well with other spp counts and other path configurations.

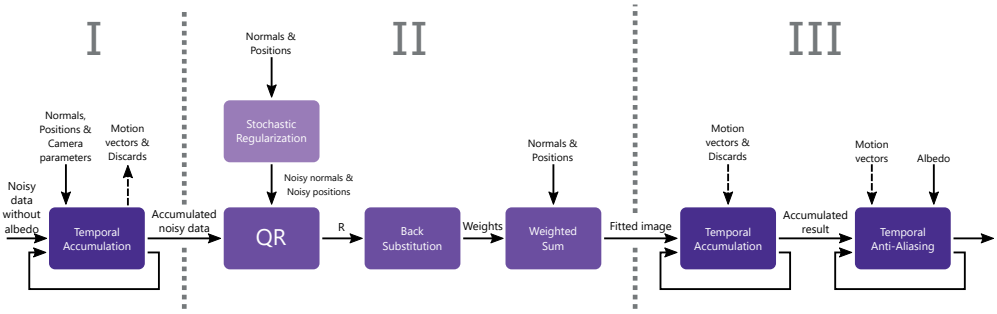


Figure 3.7 The pipeline of *Blockwise Multi-order Feature Regression* (BMFR) [**P2**]

However, best results would require adjusting some of the parameters like the used feature buffers and the size of the blocks.

3.6.1 Pipeline

In the first stage (denoted as I in the Figure 3.8), similarly to the previous work, the effective spp count of 1 spp path tracing is increased with reprojection and temporal accumulation. However, in BMFR temporal accumulation uses cumulative moving average before falling back to conventional exponential moving average when a threshold weight of 10% of new frame data is met. The use of cumulative moving average, which weights every sample equally, removes artifacts due to correct weighting of the first samples after an occlusion. Falling back to exponential moving average, used by the previous work, allows adaption to temporal effects such as lighting condition changes. The result of the stage I can be seen in Figure 3.8b.

The second stage of the BMFR pipeline (denoted as II in the Figure 3.8) does the actual filtering. Table 3.1 shows that the second stage is clearly the slowest stage of the pipeline. There noise-free feature buffers are fitted with least squares regression to noisy input image in a blockwise manner. The use of multiple orders of the feature buffers improves the quality especially in the soft shadows.

Real-time performance of the regression is achieved with stochastic regularization which adds noise to the input buffers and therefore removes rank deficiencies cheaply compared to traditional method of pivoting. Noise-free versions of the buffers are used to construct the final result instead of stochastically regularized versions of the buffers. The use of original noise-free buffers makes sure that the stochastic regularization noise does not leak into the results.

(a) Demodulated noisy input **(b)** Reprojection and accumulation (Stage I) **(c)** Blockwise regression (Stage II) **(d)** Reprojection and TAA (Stage III)

Figure 3.8 The results of different BMFR pipeline stages. Here the camera is rotating leftwards. Therefore, leftmost pixels can use only current frame data in the reprojections and TAA on stages I and III.

Table 3.1 Average timings for different BMFR sample code kernels processing a 720p frame on AMD Radeon Vega FE.

Stage	Kernel	Runtime
I	Temporal accumulation	0.44 ms
II	QR & back substitution	1.55 ms
	Weighted sum	0.12 ms
III	Temporal accumulation	0.23 ms
	TAA	0.16 ms
Total		2.54 ms

Since after stochastic regularization there is no need to care about the rank deficiencies a relatively simple and fast regression method can be used. In this case the regression is made fast with an augmented implementation of the QR-decomposition. In the implementation, a matrix containing the noise-free buffers $T_1 ... T_M$ as columns is augmented with vector of noisy samples Z. In other words, Z is concatenated to be the last column of the matrix. Then Q matrix is not needed and there is only requirement for computing the small R matrix which reduces memory bandwidth usage significantly. While R is computed, $Q \times Z$ is in practice computed and back substitution of R gives one solution to the regression. Like in fast inference of CNNs, also in BMFR memory bandwidth usage is reduced even further by using half-precision

Table 3.2 GPU runtimes of different reconstruction methods as reported in the original articles and linearly scaled for 720p frame size.

Method	Timing	Hardware
BMFR (OpenCL) [**P2**]	2.4 ms 2.5 ms	NVIDIA Titan X (Pascal) AMD Radeon Vega FE
SVGF [Sch+17]	4.4 ms	NVIDIA Titan X (Pascal)
Sparse Bilateral [Mar+17]	9.2 ms	NVIDIA Titan X (Pascal)
Autoencoder [All+17]	55 ms	NVIDIA Titan X (Pascal)
Guided Filter [BEM11]	94 ms	NVIDIA GTX 285
Adaptive Manifolds [Bau+15]	260 ms	NVIDIA GTX 780
Kernel Prediction [Bak+17]	5.3 s	NVIDIA Quadro M6000
Kernel Splatting [Gha+19]	6.8 s	NVIDIA Titan X (Pascal)

floating-point format in intermediate storage of the regression.

After weighted sum on each block the output image is noise free, but there are severe block artifacts as can be seen in Figure 3.8c. These artifacts show the magnitude of the error in the regression. The input to the BMFR is very noisy and, therefore, the result will always contain some error. As can be seen in the figure there is more error in the blocks that contain data from multiple distinct surfaces. The worst case is that there are only a few pixels from another surface and their squared error is insignificant compared to all other pixels. In that case, least squares regression can make the whole block to follow the radiance of the main surface. However, a significant part of these artifacts is hidden in the next stage.

In the third stage (denoted as III in the Figure 3.8), the same reprojection which is typically used for increasing the effective spp before filtering, is used also after filtering. The reprojection combined with pseudo random jittering of the blocks removes most of the block artifacts. However, in areas that were occluded on the previous frame there is no history data and, therefore, block artifacts are left to the frame shown to the user. These artifacts fade away quickly once there is some history data from consecutive frames, which can be seen in left edge of Figure 3.8d. There the blocks are less visible from right to left. Moreover, these blocks can be hidden if the application can afford running the second stage twice with half block offset on the second iteration. Finally, after the second reprojection step TAA [Kar14] is applied because it improves the perceived visual quality.

3.6.2 Results

The most important novelty of BMFR in comparison to the competing methods is that it can achieve fast execution time. According to our measurements, it is almost twice as fast as the closest competitor, SVGF. Table 3.2 reports the timings of different reconstruction methods. The code used for the BMFR timings was developed on AMD hardware using OpenCL. Therefore, Nvidia timings are likely not optimal. The code is available in the supplementary material of [**P2**].

The resulting visual quality of BMFR can be compared to SVGF [Sch+17] in Figure 3.4. The scene in the figure represents one of the hardest cases for the algorithms since there are different penumbra sizes in the shadows and there are no albedo details which typically hide small errors effectively.

4 FOVEATED SAMPLE DISTRIBUTION

This chapter describes different kinds of foveated rendering systems relevant to the contributions of this thesis. For a more comprehensive literature review on the topic, see the article by M. Weier et al. [Wei+17]. Since path tracing has been out of reach for real-time applications, to the best of Author's knowledge there has been no previous work (before [**P3**; **P4**; **P5**]) on foveated path tracing. Therefore, this chapter introduces related work with other styles of rendering. The motivation is to consider some pointers how they could be modified to support path tracing.

4.1 Cartesian Coordinate Space

Typical display devices have pixels that are ordered in a Cartesian grid. Therefore, rendering systems, like rasterization, are designed for rendering samples in a grid where the samples are distributed uniformly. Naturally it is a good idea to utilize these well-optimized rendering systems and build the foveated rendering pipeline so that it uses as many uniformly distributed samples as possible.

4.1.1 Multiple Resolutions

One common way of foveated rendering with rasterization is to render the frame with *multiple resolutions* [Gue+12; LW90; Pat+16; Wat+97]. The principal idea is to render only a small rectangle around the gaze point with the display's maximum resolution. Then the rest of the screen is rendered with coarser pixels, in other words, using just a small resolution. The small resolution image is mapped to display pixels with, for example, bilinear sampling. The boundary between the two different renderings can be hard to perceive in the periphery [HMT18] and it can be well hidden by using an overlap and blending. For following eye resolution more accurately, the system can render three different images. An example of this kind of

Figure 4.1 An example image of how multiple resolution foveated rendering can be done. The left side shows the relative sizes of the three renderings and how the frame is constructed. The right side visualizes the sample coverage by using nearest neighbor interpolation in upsampling the lower quality renderings.

rendering can be seen in Figure 4.1. Using this method, rendering may be up to five times faster than full resolution rendering [Gue+12]. There is also display hardware which has physical multiple resolution foveation [Kim+19; Var18].

4.1.2 Variable Rate Shading

The recent addition of VRS to GPU hardware makes implementing multiple resolution foveated rendering easier [Bho18; Sal+17]. The system does not have to run multiple complete GPU passes in order to get multiple different resolutions rendered and finally blended with VRS. Instead, the desired sampling rate can be set to every tile of the frame before the rendering pass and then the rendering of the frame can be done in one pass [Har19]. However, there is no possibility to linearly blend between two sampling rates. Therefore, at the time of writing it is uncertain if the boundaries are going to be visible with drastic changes of shading rate between neighboring tiles. If the maximum rate of change is limited it reduces the possible gain of foveated rendering optimization.

4.1.3 Linear Fall-Off

A similar sample distribution as the multiple resolution distribution may also be used with ray-traced methods [LW90; Wei+16; Wei+18a]. Since ray tracing could use also other distributions this distribution is typically called *linear fall-off*. One way to implement it is to start the recursive ray tracing process for every display pixel. Then some of the rays are killed based on predefined probabilities. For example, one can use 100% of sampled pixels at fovea, 20% at periphery and linear blending of probabilities in between. In general form, this sampling probability as a function of eccentricity angle e can be written as

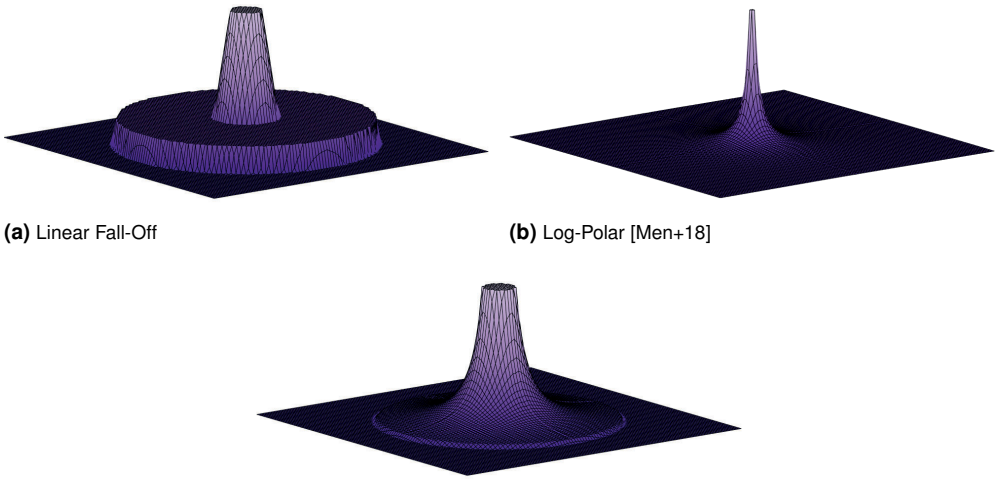
$$L(e) = \begin{cases} 1 & 0 \leq e \leq f_l \\ \frac{1-P_p}{f_l - p_l}(e - f_l) + 1 & f_l < e \leq p_l \\ P_p & p_l < e \end{cases}, \tag{4.1}$$

where f_l is the fovea limit, p_l is the periphery limit, and P_p is the periphery probability of path staying alive. In a typical implementation, probability 1 means one spp. However, nothing prevents the use of more than one spp. The sample distribution of this kind of probabilities is visualized in Figure 4.2a.

One drawback of linear fall-off is that some post-processing techniques, like denoising, require a full resolution G-buffer. In this case, one can first rasterize the G-buffer and use the probabilities for deciding if the path should continue recursively or not. It would also be possible to build hardware level support for this by decoupling shading and visibility [Vai+14]. The probabilities can also be modified based on the G-buffer data. For instance, better quality foveation can be achieved if more complicated shading is used in pixels which represent edges in the 3D scene [Ste+16].

4.2 Other Coordinate Spaces

In the recent years rendering in other screen coordinate spaces than Cartesian space have gained more interest [FRS19; Men+18]. One reason for the interest is that general purpose computing is currently fast enough for real-time applications, which allows other rendering methods than just hardware accelerated rasterization. For



(a) Linear Fall-Off

(b) Log-Polar [Men+18]

(c) Visual Acuity

Figure 4.2 Sample distributions of different foveated path tracing methods. Higher and at the same time brighter vertexes mean more samples. The fovea is always at the center of the highest sampling.

instance, in ray tracing the screen space sample locations can be decided freely. In contrast, using other coordinate spaces with rasterization requires, for example, first rasterizing the G-buffer in Cartesian space [Men+18] or tessellation. Then, the G-buffer data is mapped to the other coordinate space for deferred shading and finally the shaded results are mapped back to the Cartesian screen space pixels. Even with all this mapping, the other coordinate space can be beneficial since shading is typically the most expensive operation and the coordinate spaces being used are designed so that they reduce work significantly.

4.2.1 Polar-Space

One simple way to render more samples at the gaze point is to shade a uniform grid of samples in polar coordinates [Men+18]. Specifically, to render a 2D frame, where one coordinate is the distance ρ from the gaze point and the other coordinate is the angle ϕ around the gaze point.

The problem is that the distribution is not close to the visual acuity function. Specifically, there are as many samples as the maximum of pixels in the angle coordinate which has a property of $\rho = 0$. With a typical resolution, this means that

there are approximately a thousand samples at the gaze point. A huge sample count in just one pixel is wasteful because the fovea is bigger and, therefore, the user is also seeing worse quality rendering. In addition, there is always some error in eye-tracking systems. For instance, the error can be approximately one eccentricity degree [Wei+18b].

When moving away from the gaze point, the count of samples radically decreases. In fact, the sampling density on the eccentricity angle follows the function

$$S(e) = \frac{max(\phi)}{\rho + 1}, \tag{4.2}$$

which is visualized with its solid of revolution in Figure 4.2b. Note that here the maximum spp is $max(\phi)$, which is measured in pixels in the Polar-Space frame. In other words the height of the Polar-Space frame if angle coordinate ϕ is on vertical axis.

The distribution can be improved by using a nonlinear ρ axis. One way is to take the natural logarithm of the ρ coordinate [Men+18]. Another approach is to use a more complicated function to map the distribution to the visual acuity function [**P5**].

4.2.2 Visual Acuity Function

In some foveated ray tracing work the sample distribution follows the visual acuity function [Sie+19]. For instance the following function can be used:

$$V(e) = \begin{cases} 60.0 & 0 \leq e \leq 5.79 \\ \frac{449.4}{(0.3e+1)^2} & e > 5.79 \end{cases}, \tag{4.3}$$

where e is the eccentricity angle and $V(e)$ is the visual acuity in cycles per degree [Red97]. In other words, the maximum number of times the image can change from completely white to black. Using Equation 4.3 for path tracing sampling with a contemporary display device requires taking the minimum of the equation and the highest number cycles per degree the used device is capable of displaying. By using Equation 4.3 for guiding the sampling, the samples are used exactly where they are needed and the gain from the foveation can be greater, assuming that the eye tracking system is good enough. In case of poor eye tracking accuracy, the fovea area can be

made bigger to allow room for some error.

This kind of more complicated sample distribution like the visual acuity function can be achieved, for example, with precomputed sample locations [Sie+19] or by computing pseudorandom sample locations with the wanted distribution [**P3**; **P4**]. Both techniques have drawbacks. It is hard to move the precomputed sample locations so that they would contribute well to such temporal effects as TAA or progressive path tracing. On the other hand, completely random locations break the coherence of the ray traversal. Both problems are avoided if Polar-Space sample distribution is modified to follow visual acuity function like in Visual-Polar space [**P5**].

4.2.3 Combining Visual Acuity Function with Content Features

Even more sampling can be reduced if the sample location decision takes the content into account. For instance, on top of foveation M. Stengel et al. [Ste+16] determine which samples to shade based on silhouettes, object saliency and specular highlights. However, the decision process requires full resolution G-buffer. Then pull-push algorithm is used to fill data into in-between fragments. Similar sampling decision strategies can also be used with light field data [Sun+17]. Another approach makes the shading decisions with bigger granularity by using a low-resolution estimation of the frame [Tur+19]. The problem is that all the rendering passes must be doubled so that the low-resolution estimation can be initially generated.

Even though these methods can save 50-75% of sampling, all the extra steps combined with the sparse shading or compaction are not optimal. Therefore, the actual savings in the total timings are more modest or even negligible, at least in the case of simple fragment shaders.

4.3 Efficient Sample Distribution Implementations

For performance reasons, it is important to preserve the coherence of the primary rays and avoid sparse shading, in other words, fill all SIMT/SIMD lanes with meaningful work without the extra step of packing. This section describes two different ways that fulfill these requirements while still achieving the visual acuity function distribution of the samples.

One way to achieve visual acuity distributions is to stretch the grid of uniform sampling so that the area around the gaze point is magnified [FRS19]. The area near the gaze point gets more samples and peripheral areas get fewer samples compared to the original uniform sampling. A magnification function is needed for the sample distribution and its inverse is needed for mapping the samples back to the screen space.

Magnification can also be done with rasterization in a vertex shader. The new location of each vertex is decided with the magnification function based on the vertex's screen space location. However, this produces bent edges in the final screen space frame. Therefore, the primitive size should be kept small, which hides the bending effect. The size can be modified at run time either by selecting smaller LOD levels [Lue+03] or by splitting the primitive with a tessellation shader.

The benefits of magnification include preserving the coherence of the primary rays and filling all the SIMT/SIMD lines. The main drawback of magnification is that, like with polar spaces, when used with a rectangular grid of samples, some of them go outside of the screen area.

4.4 Mapping Other Spaces Back to Cartesian Space

When other coordinate spaces are mapped to screen space Cartesian pixels, both pixel minification and magnification are needed. In the fovea, multiple samples are mapped to one screen space pixel. On the other hand, in the periphery, one sample covers a bigger area than one screen space pixel. In consequence, mapping without artifacts requires both computing averages of multiple samples and filling areas between the samples.

4.4.1 Predefined Sampling Locations and k-Nearest Neighbor

One idea is to use predefined sample locations and store a lookup table where every pixel can fetch the addresses of its *k-Nearest Neighbor* (k-NN) samples [FH14]. The problem is that in the fovea, the maximum spp is limited by k. Also, the predefined sample map and the lookup table must be large enough to cover the screen area even if the user is looking at a corner of the screen. Predefined locations also lead to varying total sample budget based on the gaze point. The biggest sample budget is

required when the user looks at the very center of the screen.

4.4.2 Interpolation with Rasterization Hardware

Another option for mapping the predefined sample locations to screen space is to use rasterization hardware [Sie+19]. The idea is that every sample is a vertex in a mesh of triangles. When the triangles are rendered, the rasterization hardware can be used to automatically linearly interpolate between the samples in the periphery. If some anti-aliasing technique which computes multiple shading samples is used, this technique also supports averaging samples in the fovea. However, the maximum spp is limited by the anti-aliasing technique being used and it is complicated to generate a sample location map that perfectly matches the anti-aliasing sampling pattern. An interesting opportunity here would use variable rate shading when mapping the samples to screen space.

The triangle mesh can be generated, for example, with the Delaunay triangulation technique [Del34]. Its timings are in the order of seconds and therefore this technique can only be used with predefined sample locations [Cao+14; QCT12].

4.4.3 Push-Pull Technique

Dynamic sample locations can be supported for example with the push-pull technique [Gor+96]. Push-pull first maps the samples to the screen space image and marks pixels without samples. Then it generates an image pyramid by averaging every 2×2 window in the previous layer. The average is only computed from the available samples and if there are no samples at all in a window, the average is also marked to not be a sample. Once the whole pyramid is constructed it is processed from the smallest image to the largest image and every missing sample is set to be the average color which can be found from the smaller pyramid level. In the author's experience the timing of push-pull for one megapixel is in the order of 0.1 millisecond.

4.4.4 Sampling Mipmaps in Backwards Projection

3D model texturing typically uses trilinear filtering when both minifications and magnifications are needed [Ake+18, pp. 183-189]. The idea is to generate an image pyramid called a *mipmap* where every smaller level is an image where every pixel is an average of a 2×2 window from the bigger image. The difference to the push-pull algorithm is that this pyramid is done in the foveated sample distribution space and in push-pull the pyramid is done in screen space. Then the final color is decided by bilinear sampling from two pyramid levels and blending them based on the wanted sample size.

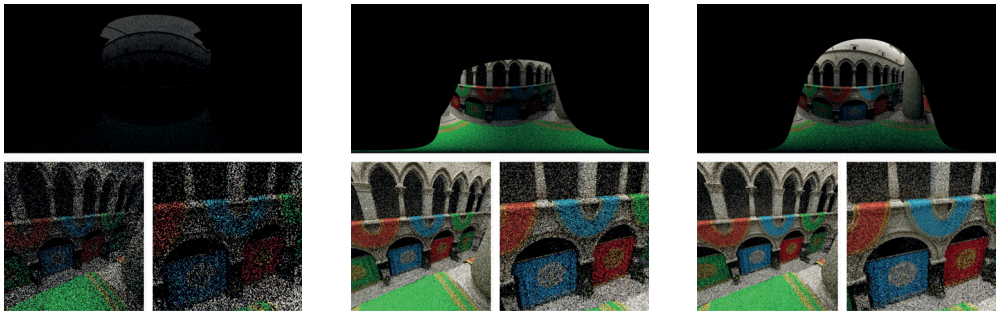
If the shape of the wanted sample is not a square, a more complicated sampling of *anisotropic* sampling can be used. The idea is to either take multiple samples from the original mipmaps or to generate mipmaps that are averages on only one of the two axes. These allow approximating other quad shapes than squares.

Trilinear and anisotropic sampling can be used to map foveated sampling space images to screen space [FRS19; Men+18]. That way both the fovea gets averaged samples and the periphery gets linear blending between the samples. Rendering APIs have function calls for generating mipmaps. Therefore, driver manufacturers have implemented very fast mipmap generation routines.

4.5 Improving the Quality with Post-Processing

Undersampling in the periphery of foveated rendering causes different kinds of spatial and temporal aliasing artifacts. Temporal artifacts are especially problematic in the periphery. One simple way to reduce them is to use a basic gaussian blur after mapping the frame to the screen space [**P5**]. Another idea for silencing temporal problems is to combine TAA with the upsampling to screen space [Gue+12]. However, the reduced contrast causes tunnel effect and therefore results are improved if contrast is added to the blurred areas [Pat+16]. In this case, magnified temporal flickering is silenced with a special version of TAA which has better outlier handling by computing the variance of the current frame neighborhood color instead of minimum and maximum.

A more complicated post-processing filtering can be used for improving the quality of the rendering at the same time. For instance, a post-processing depth of field



(a) Uniform 360° sampling

(b) Uniform sampling on the view-port area

(c) Foveated sampling according to visual acuity

Figure 4.3 Previewing offline rendering of a 360° path-traced image with different methods. In every subfigure the top figure is the whole 360° image, the left bottom figure is the view-port of HMD device, and the right bottom figure is magnification of the user's gaze point.

effect can be part of the foveation [Wei+18a]. Depth of field estimation requires more accuracy from the eye-tracking system because the focus distance of the eyes must be accurately estimated even when the user is looking at the edges of the objects.

4.6 Thesis Contributions

The foveated rendering related contributions of this thesis can be divided into two categories. First, research is conducted on previewing offline path tracing where the results are not denoised [**P3**; **P4**]. Then, after emergence of real-time path tracing reconstruction algorithms they are used with foveated rendering in [**P5**]. This work concentrates on distributing the samples so that path tracing and reconstruction of noise-free estimate can be done efficiently.

4.6.1 Foveated Preview

In [**P3**; **P4**] previewing offline path tracing renderings with foveated sample distribution is proposed. The system distributes the samples according to the visual acuity function. In this use case it is not necessary that the noise is removed before the frame is shown to the user. In the user study the participants reported that in the foveated preview they felt that the image converges to a noise-free image instanta-

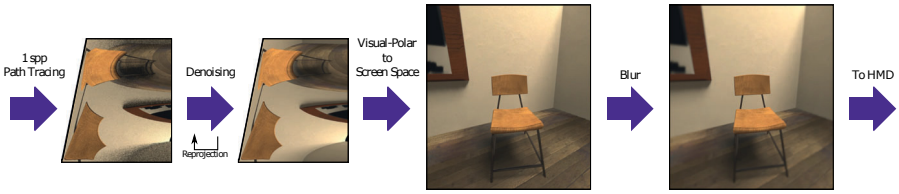


Figure 4.4 An example of a path tracing and denoising rendering pipeline using Visual-Polar [**P5**]

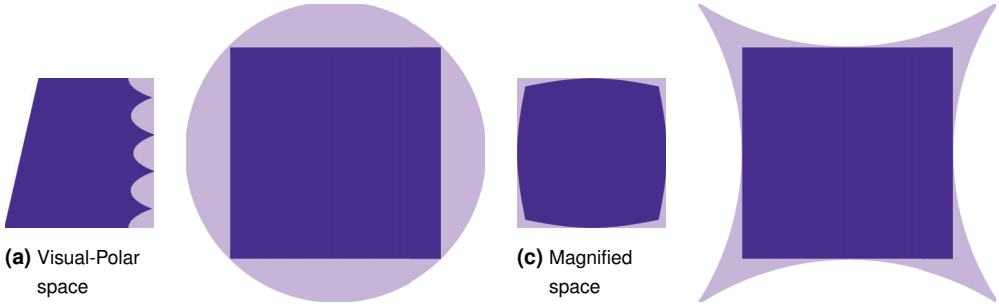
neously. Measured timings state that foveated sampling using visual acuity function is approximately three times faster compared to uniform sampling in the view-port area. In addition, foveated sampling is approximately nine times faster when compared to uniform sampling of 360° image. Comparison of different rendering methods after two seconds of rendering can be seen in Figure 4.3.

4.6.2 Visual-Polar Space

The Visual-Polar space [**P5**] is a combination of Polar-Space [Men+18] and sampling patterns that follow the visual acuity function [**P3**; **P4**]. The principal idea is that the ρ axis of the Polar-Space is scaled so that the sampling distribution is transformed to follow the visual acuity function in the periphery. In addition, a triangle shaped area is cut in the fovea to make the sample distribution uniform in the whole fovea area. Without the cutting there would be thousands of samples in one pixel at the center of the fovea. The cutting can be seen in Figure 4.4 which also shows an example pipeline of path tracing rendering in Visual-Polar space. The sample distribution of the Visual-Polar space can be seen in Figure 4.2c.

A key benefit of the Visual-Polar space is that primary rays are coherent and every SIMD/SIMT lane is doing meaningful work already without an extra step of compaction. Additional benefit is that the sample location jittering can be done in the Visual-Polar space allowing TAA.

Reconstruction of a noise-free image from the path-traced samples can be done directly in the Visual-Polar space. Typical real-time reconstructions filters like A-SVGF [SPD18], BMFR [**P2**] and SVGF [Sch+17] all work with Visual-Polar space. Reconstruction in Visual-Polar space requires only minor changes to the reconstruction algorithm. For instance, the out of bounds accesses must be wrapped around the ϕ axis. On the ρ axis at the fovea side, the out of bounds accesses could be handled properly by rotating them π radians, which equivalent to adding $max(\phi)/2$ to



(a) Visual-Polar space

(b) Visual-Polar image in screen space

(c) Magnified space

(d) Magnified image in screen space

Figure 4.5 A visualization of samples outside screen space (lighter purple) when the user looks at the center of the screen. The screen area is visualized with a darker shade of purple.

the ϕ value. However, just clamping to edge works well. Temporal reprojection accesses also need to take the Visual-Polar space into account. One way is to compute the Cartesian screen space location from the 3D world position, just like the original reconstruction algorithm did, and then map the value to the Visual-Polar space.

An interesting property of the Visual-Polar space is that it by its design adjusts the sampling area size of the reconstruction so that it matches the human visual system. The reconstruction is accessing the same number of samples in every part of the screen, but in fovea those samples map to a smaller screen space area.

In [**P5**] the last step of the pipeline is just a basic blur at the periphery. However, more complicated analytical filter or machine learning based solution which removes temporal artifacts more efficiently could also be used. In other words, Visual-Polar space can be used as a building block of a more complicated foveated rendering pipeline.

When comparing the use of magnification [FRS19] to the Visual-Polar space, there a few key differences. Firstly, there is a discontinuity on the ϕ axis in Visual-Polar space, which makes its use more complicated. Secondly, due to the polar coordinate space, the artifacts which the reconstruction algorithms generate are circular around the fovea in the Visual-Polar space. This has not been proved with a user study, but a hypothesis is that circular artifacts around the fovea are less disturbing to the user. Both methods have samples which go outside the screen area, which are visualized in Figure 4.5. How much the excessive samples and their location affect the timings of the path tracer and denoiser depends on their implementations.

When mapping Visual-Polar space frames to screen space there is no need to gen-

erate regular mipmap pyramid and use anisotropic filtering with it like in [FRS19; Men+18]. It is enough to generate the mipmaps over the ϕ axis, because ρ scaling is designed so that no minification is needed on that axis. This yields anisotropic style sampling by default.

5 CONCLUSIONS

High requirements for latency, resolution, and visual quality are drivers in the search for new rendering optimization strategies which could allow orders of magnitude better real-time rendering. Due to the rising number of HMDs with wide field of view, foveated rendering is considered to be one such optimization strategy. In order to obtain a good visual quality, one can turn into rendering techniques such as path tracing which are used in the movie industry for achieving rendering results that are indistinguishable from real camera footage [God14].

Low sample rate path tracing has recently been made fast enough for real-time applications. Firstly, dedicated ray tracing hardware [Kil+18] has made the available ray per pixel budget barely high enough to achieve a basic, but very noisy, path-traced estimate of a simulated camera view. Secondly, today there are multiple methods which can reconstruct noise-free frames from these approximations in real time [HA19; **P2**; Sch+17]. However, there is still major room for improvement in the real-time reconstruction of path tracing. All of the fast-enough methods contain some kind of quality issues: typical SVGF implementations show À Trous filter style artifacts [Bar18], sheared filtering can handle only one light at the time [HA19, p. 314], and BMFR shows its blockwise nature in occlusions and difficult soft shadow cases [**P2**]. Moreover, none of the methods can work without extensive use of history data. Gradient estimation [SPD18] removes this problem to some extent, but only if most of the pixels can use history data.

Current state-of-the-art machine learning based approaches are too slow for real-time [Gha+19; Vog+18]. Another problem is that the output is not temporally stable [All+17]. This could be improved by adding an extra reprojection step [SPD18], but changes in the illumination make the network fall back to temporally unstable single frame quality.

Path tracing has been too slow for real-time applications and there is no point of doing foveated rendering off-line since it would require predicting the user's possi-

ble points of interests. Therefore, most of the related work on foveated rendering has been rasterization-based. Moreover, due to the restrictions in the rasterization hardware, a typical approach has been to render multiple different resolution images with different FOV [Gue+12]. Similar sample distribution can be used in ray tracing-based techniques [LW90; Wei+16; Wei+18a], but since ray tracing allows flexible sampling in screen space, it is reasonable to use sampling patterns which more accurately model the human visual acuity.

One problem with simple implementations of more complex sample distribution in ray-traced foveation is that the coherence of primary rays and shadow rays of the first bounce are not preserved. Another problem is that some of the SIMD/SIMT lanes are not filled with meaningful work. Both problems diminish the possible benefits of foveation. Proposed solutions to these problems include using modified polar space [**P5**] or magnification of Cartesian space [FRS19].

5.1 Main Results

This thesis proposes novel techniques required for building an end-to-end foveated path tracing system. Firstly, estimate that the possible gain of foveated optimization is at maximum 95% is computed [**P1**]. This outlines the absolute maximum for the computational benefits available.

Secondly, to address the need for a real-time path tracing reconstruction method, a new path tracing reconstruction method called *Blockwise Multi-Order Feature Regression* (BMFR) is proposed [**P2**]. The method is the first regression-based path tracing reconstruction method that achieves real-time frame rates. Other real-time methods, also published after the start of this thesis project, are based on fast approximations of bilateral filter [Mar+17; Sch+17]. Moving to regression-based methods is a natural next step since they were dominating the offline reconstruction before machine learning-based methods. BMFR produces some artifacts in difficult soft shadow cases and in occlusions where it cannot use reprojected previous frame data. Therefore, it is not yet a perfect candidate, without any modification, for use with gradient estimation [SPD18]. However, the main motivation for using BMFR is its fast runtime with a reasonable quality.

Then, a method for previewing path-traced content in VR is introduced [**P3**; **P4**]. In this work the results are left noisy. However, thanks to gaze contingent path

tracing with visual acuity-based sample distribution the participants to the user study reported that the rendering converges instantly. Therefore, this method could be used to make a 3D-artist's workflow faster which in turn improves offline rendering quality.

Finally, the work in [**P2**; **P3**; **P4**] is put together with inspiration from [Men+18] in the form of a novel Visual-Polar space [**P5**]. The main idea is to develop a method to distribute the path tracing samples so that primary rays are coherent and all SIMD/ SIMT lanes are full of meaning full work which improves the utilization of the hardware. A further novelty is that, the path tracing reconstruction can be done directly in the Visual-Polar space which has a lower resolution and therefore also reduces the computing power required for reconstruction. The use of polar coordinates introduces discontinuity, which does not produce any additional artifacts, but it complicates the implementation. One benefit of the Visual-Polar space is that the artifacts produced by the reconstruction filters are circular around the fovea. Visual-Polar space shows one way of reducing the resolution in the periphery and it can be used as a building block of other foveated rendering systems which use more sophisticated filters or even machine learning for mapping the image to the screen space.

5.2 Future Work

At the time of this writing the first generation of ray traversal hardware has been released by one graphics processor manufacturer and other manufacturers are going to release their hardware in the near future. It is possible that ray traversal hardware follows the same path as rasterization hardware. In that case, we will see a couple of generations of faster dedicated ray traversal hardware units. After that ray traversal could be part of general-purpose computing units like proposed in [**Kos+16**]. So, it would be interesting to research more small modifications to general-purpose hardware, like proposed in [Kee14], which could make ray traversal almost equally fast compared to dedicated hardware.

It is likely that the real-time reconstruction will follow the same path that offline reconstruction has been following. Currently the research is focused on minor improvements to the À Trous filtering. It is possible that there will be some research on better and faster regression-based methods, like the one presented in this thesis

[**P2**], before there is sufficiently fast neural network hardware within consumer-level devices to allow full-scale use of machine-learning reconstruction algorithms in real time.

Thanks to variable rate shading, there will likely be no new foveated rendering publications on multi-resolution foveation. Most likely, in the future, the control of VRS will be improved and the shading rate can be adjusted more precisely. More freedom within rasterization-based techniques will also help ray tracing based techniques as then the G-buffer can be generated more freely. However, this likely does not allow complete reduced resolution for path tracing reconstruction like Visual-Polar space [**P5**].

In the future, we are likely going to see some interesting methods that use deep learning for improving the foveation quality [Kap+19]. To the best of the author's knowledge this is going to be the first machine learning based method for reconstructing foveated frames. The idea is that the network generates temporally stable details in between the samples so that they are not distracting in the peripheral vision of the user. Currently the results seem promising. However, the execution time of the inference is still the problem. The paper reports 9 ms inference on a cluster of 4× high-end GPUs. Therefore, likely the next step is to make a significantly faster version of the network. This kind of fast network could be combined with the efficient path tracing and reconstruction of Visual-Polar space [**LKJ20**].

REFERENCES

[Abr14] M. Abrash. *What VR Could, Should, and Almost Certainly Will Be within Two Years*. Steam Dev Days 4. 2014.

[AET96] R. Anderson, D. Evans and L. Thibos. Effect of Window Size on Detection Acuity and Resolution Acuity for Sinusoidal Gratings in Central and Peripheral Vision. *Journal of the Optical Society of America A* 13.4 (1996).

[AGL19] R. Albert, A. Godinez and D. Luebke. Reading Speed Decreases for Fast Readers Under Gaze-Contingent Rendering. *Proceedings of the Symposium on Applied Perception*. 2019.

[Ake+18] T. Akenine-Möller, E. Haines, N. Hoffman, A. Pesce, M. Iwanicki and S. Hillaire. *Real-Time Rendering*. 3rd. CRC Press, 2018.

[Alb+17] R. Albert, A. Patney, D. Luebke and J. Kim. Latency Requirements for Foveated Rendering in Virtual Reality. *Transactions on Applied Perception* 14.4 (2017).

[All+17] C. Alla Chaitanya, A. Kaplanyan, C. Schied, M. Salvi, A. Lefohn, D. Nowrouzezahrai and T. Aila. Interactive Reconstruction of Monte Carlo Image Sequences Using a Recurrent Denoising Autoencoder. *Transactions on Graphics* 36.4 (2017).

[Ara+17] E. Arabadzhiyska, O. Tursun, K. Myszkowski, H.-P. Seidel and P. Didyk. Saccade Landing Position Prediction for Gaze-Contingent Rendering. *Transactions on Graphics* 36.4 (2017).

[Bak+17] S. Bako, T. Vogels, B. McWilliams, M. Meyer, J. Novák, A. Harvill, P. Sen, T. Derose and F. Rousselle. Kernel-Predicting Convolutional Networks for Denoising Monte Carlo Renderings. *Transactions on Graphics* 36.4 (2017).

[Bar18] C. Barré-Brisebois. *Game Ray Tracing: State-of-the-Art and Open Problems*. High Performance Graphics Keynote. 2018.

[Bau+15] P. Bauszat, M. Eisemann, S. John and M. Magnor. Sample-Based Manifold Filtering for Interactive Global Illumination and Depth of Field. *Computer Graphics Forum* 34.1 (2015).

[BEM11] P. Bauszat, M. Eisemann and M. Magnor. Guided Image Filtering for Interactive High-quality Global Illumination. *Computer Graphics Forum* 30.4 (2011).

[Bho18] S. Bhonde. *Turing Variable Rate Shading in VRWorks*. `https://devblogs.nvidia.com/turing-variable-rate-shading-vrworks/` accessed: 2019-05-12. 2018.

[Bit+16] B. Bitterli, F. Rousselle, B. Moon, J. A. Iglesias-Guitián, D. Adler, K. Mitchell, W. Jarosz and J. Novák. Nonlinearly Weighted First-order Regression for Denoising Monte Carlo Renderings. *Computer Graphics Forum* 35.4 (2016).

[BN76] J. Blinn and M. Newell. Texture and Reflection in Computer Generated Images. *Communications of the ACM* 19.10 (1976).

[BS13] J. Bikker and J. van Schijndel. The Brigade Renderer: A Path Tracer for Real-Time Games. *International Journal of Computer Games Technology* (2013).

[Bur81] P. Burt. Fast Filter Transform for Image Processing. *Computer Graphics and Image Processing* 16.1 (1981).

[CA90] C. Curcio and K. Allen. Topography of Ganglion Cells in Human Retina. *Journal of Comparative Neurology* 300.1 (1990).

[Cao+14] T.-T. Cao, A. Nanjappa, M. Gao and T.-S. Tan. A GPU Accelerated Algorithm for 3D Delaunay Triangulation. *Proceedings of the Symposium on Interactive 3D Graphics and Games*. 2014.

[CBD14] M. Courbariaux, Y. Bengio and J.-P. David. Training Deep Neural Networks with Low Precision Multiplications. *arXiv preprint arXiv:1412.7024* (2014).

[CG66] F. Campbell and R. Gubisch. Optical Quality of the Human Eye. *The Journal of Physiology* 186.3 (1966).

[CM07] R. Castilhos and J. Marchn. *Schematic diagram of the human eye*. `https://commons.wikimedia.org/wiki/File:Schematic_diagram_of_the_human_eye_en.svg`, accessed: 2019-09-04, Licensed under the Creative Commons Attribution-Share Alike 3.0 Unported license. 2007.

[Cog+18] T. E. Cognard, A. Goncharov, N. Devaney, C. Dainty and P. Corcoran. A Review of Resolution Losses for AR/VR Foveated Imaging Applications. *Proceedings of Games, Entertainment, Media Conference*. 2018.

[Coo84] R. Cook. Shade Trees. *SIGGRAPH Computer Graphics* 18.3 (1984).

[CPC84] R. Cook, T. Porter and L. Carpenter. Distributed Ray Tracing. *SIGGRAPH Computer Graphics*. Vol. 18. 3. 1984.

[CT07] J. Chen and J. Thropp. Review of Low Frame Rate Effects on Human Performance. *Transactions on Systems* 37.6 (2007).

[Cur+90] C. Curcio, K. Sloan, R. Kalina and A. Hendrickson. Human Photoreceptor Topography. *Journal of Comparative Neurology* 292.4 (1990).

[Dam+10] H. Dammertz, D. Sewtz, J. Hanika and H. Lensch. Edge-avoiding À-Trous Wavelet Transform for Fast Global Illumination Filtering. *Proceedings of the High Performance Graphics*. 2010.

[Del34] B. Delaunay. Sur la Sphere Vide. *Izv. Akad. Nauk SSSR, Otdelenie Matematicheskii i Estestvennyka Nauk* 7.793-800 (1934).

[DFE07] K. Dabov, A. Foi and K. Egiazarian. Video Denoising by Sparse 3D Transform-Domain Collaborative Filtering. *Proceedings of the European Signal Processing Conference*. 2007.

[DH14] O. Dhande and A. Huberman. Retinal Ganglion Cell Maps in the Brain: Implications for Visual Processing. *Current Opinion in Neurobiology* 24 (2014).

[ED04] E. Eisemann and F. Durand. Flash Photography Enhancement via Intrinsic Relighting. *Transactions on graphics* 23.3 (2004).

[Ega+11] K. Egan, F. Hecht, F. Durand and R. Ramamoorthi. Frequency Analysis and Sheared Filtering for Shadow Light Fields of Complex Occluders. *Transactions on Graphics* 30.2 (2011).

[EK18] A. C. Estevez and C. Kulla. Importance Sampling of Many Lights with Adaptive Tree Splitting. *Proceedings of the ACM on Computer Graphics and Interactive Techniques* 1.2 (2018).

[EL18] A. C. Estevez and P. Lecocq. Fast Product Importance Sampling of Environment Maps. *SIGGRAPH 2018 Talks*. 2018.

[FH14] M. Fujita and T. Harada. *Foveated Real-Time Ray Tracing for Virtual Reality Headset*. Tech. rep. Light Transport Entertainment Research, 2014.

[Fow05] J. Fowler. The Redundant Discrete Wavelet Transform and Additive Noise. *Signal Processing Letters* 12.9 (2005).

[FRS19] S. Friston, T. Ritschel and A. Steed. Perceptual Rasterization for Head-Mounted Display Image Synthesis. *Transactions on Graphics* 38.4 (2019).

[Gha+19] M. Gharbi, T.-M. Li, M. Aittala, J. Lehtinen and F. Durand. Sample-Based Monte Carlo Denoising Using a Kernel-Splatting Network. *Transactions on Graphics* 38.4 (2019).

[GL10] K. Garanzha and C. Loop. Fast Ray Sorting and Breadth-First Packet Traversal for GPU Ray Tracing. *Computer Graphics Forum*. Vol. 29. 2. 2010.

[GMN14] J.-P. Guertin, M. McGuire and D. Nowrouzezahrai. A Fast and Stable Feature-Aware Motion Blur Filter. *Proceedings of High Performance Graphics*. 2014.

[GO12] E. Gastal and M. Oliveira. Adaptive Manifolds for Real-Time High-Dimensional Filtering. *Transactions on Graphics* 31.4 (2012).

[God14] L. Goddard. Silencing the Noise on Elysium. *SIGGRAPH 2014 Talks*. 2014.

[Gor+96] S. Gortler, R. Grzeszczuk, R. Szeliski and M. Cohen. The Lumigraph. *Proceedings of Conference on Computer Graphics and Interactive Techniques*. 1996.

[GP98] W. Geisler and J. Perry. Real-Time Foveated Multiresolution System for Low-Bandwidth Video Communication. *Human Vision and Electronic Imaging III*. Vol. 3299. International Society for Optics and Photonics. 1998.

[Gue+12] B. Guenter, M. Finch, S. Drucker, D. Tan and J. Snyder. Foveated 3D graphics. *Transactions on Graphics* 31.6 (2012).

[HA19] E. Haines and T. Akenine-Möller, eds. *Ray Tracing Gems*. Available: http://raytracinggems.com Accessed: 2019-04-02. Apress, 2019.

[Had+05] M. Hadwiger, C. Sigg, H. Scharsach, K. Bühler and M. Gross. Real-Time Ray-Casting and Advanced Shading of Discrete Isosurfaces. *Computer Graphics Forum* 24.3 (2005).

[Har19] S. Hargreaves. *DirectX Specs: Variable Rate Shading*. d23b62c3. Available: https://github.com/Microsoft/DirectX-Specs/blob/master/d3d/VariableRateShading.md Accessed: 2019-04-10. Microsoft. Mar. 2019.

[He+16] K. He, X. Zhang, S. Ren and J. Sun. Deep Residual Learning for Image Recognition. *Proceedings of the Conference on Computer Vision and Pattern Recognition*. 2016.

[HF04] J. Hopp and A. Fuchs. The Characteristics and Neuronal Substrate of Saccadic Eye Movement Plasticity. *Progress in Neurobiology* 72.1 (2004).

[HMT18] D. Hoffman, Z. Meraz and E. Turner. Limits of peripheral acuity and implications for VR system design. *Journal of the Society for Information Display* 26.8 (2018).

[HST13] K. He, J. Sun and X. Tang. Guided Image Filtering. *Transactions on Pattern Analysis and Machine Intelligence* 35.6 (2013).

[HTH19] G. Henry, P. Tang and A. Heinecke. Leveraging the bfloat16 Artificial Intelligence Datatype For Higher-Precision Computations. *arXiv preprint arXiv:1904.06376* (2019).

[Hug+14] J. Hughes, A. Van Dam, M. McGuire, D. Sklar, J. Foley, S. Feiner and K. Akeley. *Computer Graphics: Principles and Practice*. 3rd. Addison-Wesley Professional, 2014.

[Ian+16] F. Iandola, S. Han, M. Moskewicz, K. Ashraf, W. Dally and K. Keutzer. SqueezeNet: AlexNet-Level Accuracy with 50x Fewer Parameters and <0.5 MB Model Size. *arXiv preprint arXiv:1602.07360* (2016).

[IEE08] IEEE. *IEEE 754 Half-Precision Binary Floating-Point Format*. 2008.

[**Imm17**] K. Immonen. Real-Time Noise Removal in Foveated Path Tracing. M.Sc. thesis. Tampere University of Technology, Finland, 2017.

[Kaj86] J. Kajiya. The Rendering Equation. *SIGGRAPH Computer Graphics* 20.4 (1986).

[Kap+19] A. Kaplanyan, A. Sochenov, T. Leimkuehler, M. Okunev, T. Goodall and R. Gizem. DeepFovea: Neural Reconstruction for Foveated Rendering and Video Compression using Learned Statistics of Natural Videos. *Transactions on Graphics* 38.4 (2019).

[Kar14] B. Karis. High-quality Temporal Supersampling. *SIGGRAPH 2014, Advances in Real-Time Rendering in Games*. 2014.

[KBS15] N. Kalantari, S. Bako and P. Sen. A Machine Learning Approach for Filtering Monte Carlo Noise. *Transactions on Graphics* 34.4 (2015).

[Kee14] S. Keely. Reduced Precision Hardware for Ray Tracing. *Proceedings of High Performance Graphics* (2014).

[Kel+15] A. Keller, L. Fascione, M. Fajardo, I. Georgiev, P. Christensen, J. Hanika, C. Eisenacher and G. Nichols. The Path Tracing Revolution in the Movie Industry. *SIGGRAPH 2015 Courses*. 2015.

[Kel+18] A. Keller, J. Křivánek, J. Novák, A. Kaplanyan and M. Salvi. Machine Learning and Rendering. *SIGGRAPH 2018 Courses*. 2018.

[Kel84] D. Kelly. Retinal Inhomogeneity. I. Spatiotemporal Contrast Sensitivity. *Journal of the Optical Society of America A* 1.1 (1984).

[KHL19a] M. Kettunen, E. Härkönen and J. Lehtinen. Deep Convolutional Reconstruction for Gradient-Domain Rendering. *Transactions on Graphics* 38.4 (2019).

[KHL19b] M. Kettunen, E. Härkönen and J. Lehtinen. E-LPIPS: Robust Perceptual Image Similarity via Random Transformation Ensembles. *arXiv preprint arXiv:1906.03973* (2019).

[Kil+18] E. Kilgariff, H. Moreton, N. Stam and B. Bell. *NVIDIA Turing Architecture In-Depth*. `https://devblogs.nvidia.com/nvidia-turing-architecture-in-depth/` accessed: 2019-01-08. 2018.

[Kim+19] J. Kim, Y. Jeong, M. Stengel, K. Akşit, R. Albert, B. Boudaoud, T. Greer, J. Kim, W. Lopes, Z. Majercik, P. Shirley, J. Spjut, M. McGuire and D. Luebke. Foveated AR: Dynamically-Foveated Augmented Reality Display. *Transactions on Graphics* 38.4 (2019).

[**Kos+15**] M. Koskela, T. Viitanen, P. Jääskeläinen, J. Takala and K. Cameron. Using Half-Precision Floating-Point Numbers for Storing Bounding Volume Hierarchies. *Proceedings of the Computer Graphics International Conference*. 2015.

[**Kos+16**] M. Koskela, T. Viitanen, P. Jääskeläinen and J. Takala. Half-precision Floating-point Ray Traversal. *Proceedings of the International Conference on Computer Graphics Theory and Applications*. 2016.

[**Kos15**] M. Koskela. Software-Based Ray Tracing for Mobile Devices. M.Sc. thesis. Tampere University of Technology, Finland, 2015.

[Kow11] E. Kowler. Eye Movements: the Past 25 Years. *Vision Research* 51.13 (2011).

[Kra+16] K. Krafka, A. Khosla, P. Kellnhofer, H. Kannan, S. Bhandarkar, W. Matusik and A. Torralba. Eye Tracking for Everyone. *Proceedings of the Conference on Computer Vision and Pattern Recognition*. 2016.

[LDS90] Y. LeCun, J. Denker and S. Solla. *Advances in Neural Information Processing Systems 2*. Morgan Kaufmann Publishers Inc, 1990. Chap. Optimal Brain Damage.

[Lee+13] W.-J. Lee, Y. Shin, J. Lee, J.-W. Kim, J.-H. Nah, S. Jung, S. Lee, H.-S. Park and T.-D. Han. SGRT: A Mobile GPU Architecture for Real-Time Ray Tracing. *Proceedings of High Performance Graphics*. 2013.

[Leh+18] J. Lehtinen, J. Munkberg, J. Hasselgren, S. Laine, T. Karras, M. Aittala and T. Aila. Noise2Noise: Learning Image Restoration without Clean Data. *Proceedings of the International Conference on Machine Learning*. 2018.

[LHY08] C.-Y. Liou, J.-C. Huang and W.-C. Yang. Modeling Word Perception Using the Elman Network. *Neurocomputing* 71.16-18 (2008).

[Liu+17] Y. Liu, C. Zheng, Q. Zheng and H. Yuan. Removing Monte Carlo Noise Using a Sobel Operator and a Guided Image Filter. *The Visual Computer* 34.4 (2017).

[**LKJ**20] A. Lotvonen, M. Koskela and P. Jääskeläinen. Machine Learning Is the Solution Also for Foveated Path Tracing Reconstruction. *Accepted to Proceedings of the International Conference on Computer Graphics Theory and Applications*. 2020.

[Lue+03] D. Luebke, M. Reddy, J. Cohen, A. Varshney, B. Watson and R. Huebner. *Level of Detail for 3D Graphics*. Morgan Kaufmann, 2003.

[LW90] M. Levoy and R. Whitaker. Gaze-Directed Volume Rendering. *SIGGRAPH Computer Graphics*. Vol. 24. 2. 1990.

[**Mak+19**] M. Mäkitalo, P. Kivi, M. Koskela and P. Jääskeläinen. Reducing Computational Complexity of Real-Time Stereoscopic Ray Tracing with Spatiotemporal Sample Reprojection. *Proceedings of the International Conference on Computer Graphics Theory and Applications*. 2019.

[Mar+17] M. Mara, M. McGuire, B. Bitterli and W. Jarosz. An Efficient Denoising Algorithm for Global Illumination. *Proceedings of the High Performance Graphics*. 2017.

[MB18] D. Meister and J. Bittner. Parallel Locally-Ordered Clustering for Bounding Volume Hierarchy Construction. *Transactions on Visualization and Computer Graphics* 24.3 (2018).

[McG17] M. McGuire. *Computer Graphics Archive*. `https://casual-effects.com/data` accessed: 2019-09-23. 2017.

[Meh+13] S. Mehta, B. Wang, R. Ramamoorthi and F. Durand. Axis-Aligned Filtering for Interactive Physically-Based Diffuse Indirect Lighting. *Transactions on Graphics* 32.4 (2013).

[Meh+14] S. Mehta, J. Yao, R. Ramamoorthi and F. Durand. Factored Axis-Aligned Filtering for Rendering Multiple Distribution Effects. *Transactions on Graphics* 33.4 (2014).

[Men+18] X. Meng, R. Du, M. Zwicker and A. Varshney. Kernel Foveated Rendering. *Proceedings of the ACM on Computer Graphics and Interactive Techniques* 1.1 (2018).

[Moo+16] B. Moon, S. McDonagh, K. Mitchell and M. Gross. Adaptive Polynomial Rendering. *Transactions on Graphics* 35.4 (2016).

[Mor+18] A. Morales, F. Costela, R. Tolosana and R. Woods. Saccade Landing Point Prediction: A Novel Approach based on Recurrent Neural Networks. *Proceedings of the International Conference on Machine Learning Technologies*. 2018.

[MWR12] S. Mehta, B. Wang and R. Ramamoorthi. Axis-Aligned Filtering for Interactive Sampled Soft Shadows. *Transactions on Graphics* 31.6 (2012).

[**P1**] M. Koskela, T. Viitanen, P. Jääskeläinen and J. Takala. Foveated Path Tracing: A Literature Review and a Performance Gain Analysis. *Proceedings of International Symposium on Visual Computing*. 2016.

[**P2**] 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. *Transactions on Graphics* 38.5 (2019).

[**P3**] M. Koskela, K. Immonen, T. Viitanen, P. Jääskeläinen, J. Multanen and J. Takala. Foveated Instant Preview for Progressive Rendering. *SIGGRAPH Asia Technical Briefs*. 2017.

[**P4**] M. Koskela, K. Immonen, T. Viitanen, P. Jääskeläinen, J. Multanen and J. Takala. Instantaneous Foveated Preview for Progressive Monte Carlo Rendering. *Computational Visual Media* 4.3 (2018).

[**P5**] M. Koskela, A. Lotvonen, M. Mäkitalo, P. Kivi, T. Viitanen and P. Jääskeläinen. Foveated Real-Time Path Tracing in Visual-Polar Space. *Eurographics Symposium on Rendering (DL-only Track)*. 2019.

[Pat+16] 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. *Transactions on Graphics* 35.6 (2016).

[Pet+04] G. Petschnigg, R. Szeliski, M. Agrawala, M. Cohen, H. Hoppe and K. Toyama. Digital Photography with Flash and No-Flash Image Pairs. *Transactions on graphics* 23.3 (2004).

[PH10] M. Pharr and G. Humphreys. *Physically Based Rendering: From Theory to Implementation*. 2nd. Morgan Kaufmann, 2010.

[PL10] J. Pantaleoni and D. Luebke. HLBVH: Hierarchical LBVH Construction for Real-Time Ray Tracing of Dynamic Geometry. *Proceedings of High Performance Graphics*. 2010.

[PR10] M. Poletti and M. Rucci. Eye Movements Under Various Conditions of Image Fading. *Journal of Vision* 10.3 (2010).

[PZB16] D. Pohl, X. Zhang and A. Bulling. Combining Eye Tracking with Optimizations for Lens Astigmatism in Modern Wide-Angle HMDs. *Proceedings of the International Conference on Virtual Reality*. 2016.

[QCT12] M. Qi, T.-T. Cao and T.-S. Tan. Computing 2D Constrained Delaunay Triangulation Using the GPU. *Transactions on Visualization and Computer Graphics* 19.5 (2012).

[QWH12] B. Qi, T. Wu and H. He. A Novel Edge-Aware À-Trous Filter for Single Image Dehazing. *International Conference on Information Science and Technology*. 2012.

[Red97] M. Reddy. Perceptually Modulated Level of Detail for Virtual Environments. PhD thesis. University of Edinburgh, United Kingdom, 1997.

[RFB15] O. Ronneberger, P. Fischer and T. Brox. U-net: Convolutional Networks for Biomedical Image Segmentation. *Proceedings of the International Conference on Medical Image Computing and Computer-Assisted Intervention*. 2015.

[RKZ12] F. Rousselle, C. Knaus and M. Zwicker. Adaptive Rendering with Non-Local Means Filtering. *Transactions on Graphics* 31.6 (2012).

[RVL12] T. Raiko, H. Valpola and Y. LeCun. Deep Learning Made Easier by Linear Transformations in Perceptrons. *Proceedings of Machine Learning Research*. 2012.

[RVN78] J. Rovamo, V. Virsu and R. Näsänen. Cortical Magnification Factor Predicts the Photopic Contrast Sensitivity of Peripheral Vision. *Nature* 271.5640 (1978).

[Sal+17] S. Saleh, C. Brennan, A. Pomianowski and R. Wu. Variable Rate Shading. United States Patent Application 20190066371. 2017.

[SB91] G. Sullivan and R. Baker. Motion Compensation for Video Compression Using Control Grid Interpolation. *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*. 1991.

[Sca12] M. Scarpino. *OpenCL in Action*. Manning Publications Co, 2012.

[Sch+17] C. Schied, A. Kaplanyan, C. Wyman, A. Patney, C. R. A. Chaitanya, J. Burgess, S. Liu, C. Dachsbacher, A. Lefohn and M. Salvi. Spatiotemporal Variance-Guided Filtering: Real-Time Reconstruction for Path-Traced Global Illumination. *Proceedings of the High Performance Graphics*. 2017.

[Sch19] C. Schied. *Q2VKPT*. `http://brechpunkt.de/q2vkpt/` accessed: 2019-04-17. 2019.

[Sch56] O. Schade. Optical and Photoelectric Analog of the Eye. *Journal of the Optical Society of America* 46.9 (1956).

[Shi+16] W. Shi, J. Caballero, F. Huszár, J. Totz, A. Aitken, R. Bishop, D. Rueckert and Z. Wang. Real-time Single Image and Video Super-Resolution Using an Efficient Sub-Pixel Convolutional Neural Network. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016.

[Sie+19] A. Siekawa, M. Chwesiuk, R. Mantiuk and R. Piórkowski. Foveated Ray Tracing for VR Headsets. *International Conference on Multimedia Modeling*. 2019.

[Sif14] L. Sifre. Rigid-Motion Scattering for Image Classification. PhD thesis. Ecole Polytechnique, France, 2014.

[SPD18] 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 (2018).

[Sri+14] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever and R. Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *The Journal of Machine Learning Research* 15.1 (2014).

[SRJ11] H. Strasburger, I. Rentschler and M. Jüttner. Peripheral Vision and Pattern Recognition: A Review. *Journal of Vision* 11.5 (2011).

[Ste+16] M. Stengel, S. Grogorick, M. Eisemann and M. Magnor. Adaptive Image-Space Sampling for Gaze-Contingent Real-Time Rendering. *Computer Graphics Forum* 35.4 (2016).

[Sun+17] Q. Sun, F.-C. Huang, J. Kim, L.-Y. Wei, D. Luebke and A. Kaufman. Perceptually-Guided Foveation for Light Field Displays. *Transactions on Graphics* 36.6 (2017).

[Sun+18] Q. Sun, A. Patney, L.-Y. Wei, O. Shapira, J. Lu, P. Asente, S. Zhu, M. McGuire, D. Luebke and A. Kaufman. Towards Virtual Reality Infinite Walking: Dynamic Saccadic Redirection. *Transactions on Graphics* 37.4 (2018).

[TCW87] L. Thibos, F. Cheney and D. Walsh. Retinal Limits to the Detection and Resolution of Gratings. *Journal of the Optical Society of America A* 4.8 (1987).

[Thi87] L. Thibos. Calculation of the Influence of Lateral Chromatic Aberration on Image Quality Across the Visual Field. *Journal of the Optical Society of America A* 4.8 (1987).

[TM98] C. Tomasi and R. Manduchi. Bilateral Filtering for Gray and Color Images. *Proceedings of the International Conference on Computer Vision*. 1998.

[TS07] M. Taylor and P. Stone. Cross-Domain Transfer for Reinforcement Learning. *Proceedings of the international conference on Machine learning*. 2007.

[Tur+19] O. Tursun, E. Arabadzhiyska-Koleva, M. Wernikowski, R. Mantiuk, H.-P. Seidel, K. Myszkowski and P. Didyk. Luminance-Contrast-Aware Foveated Rendering. *Transactions on Graphics* 38.4 (2019).

[Vai+14] 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. *Proceedings of High Performance Graphics*. 2014.

[Van14] V. Vanhoucke. *Learning Visual Representations at Scale*. ICLR invited talk. 2014.

[Var18] Varjo. *VR-1: The Only Human-Eye Resolution Headset*. Available: `https://varjo.com/products/vr-1/` Accessed: 2019-10-10. 2018.

[Vea97] E. Veach. Robust Monte Carlo Methods for Light Transport Simulation. PhD thesis. 1997.

[VG95] E. Veach and L. Guibas. Optimally Combining Sampling Techniques for Monte Carlo Rendering. *Proceedings of the Conference on Computer graphics and interactive techniques*. 1995.

[**Vii+15**] T. Viitanen, M. Koskela, P. Jääskeläinen, H. Kultala and J. Takala. MergeTree: A HLBVH Constructor for Mobile Systems. *SIGGRAPH Asia Technical Briefs*. 2015.

[**Vii+16**] T. Viitanen, M. Koskela, P. Jääskeläinen and J. Takala. Multi Bounding Volume Hierarchies for Ray Tracing Pipelines. *SIGGRAPH Asia Technical Briefs*. 2016.

[**Vii+17a**] T. Viitanen, M. Koskela, P. Jääskeläinen, K. Immonen and J. Takala. Fast Hardware Construction and Refitting of Quantized Bounding Volume Hierarchies. *Computer Graphics Forum* 36.4 (2017).

[**Vii+17b**] T. Viitanen, M. Koskela, P. Jääskeläinen, H. Kultala and J. Takala. MergeTree: A Fast Hardware HLBVH Constructor for Animated Ray Tracing. *Transactions on Graphics* 36.5 (2017).

[**Vii+18a**] T. Viitanen, M. Koskela, K. Immonen, M. Mäkitalo, P. Jääskeläinen and J. Takala. Sparse Sampling for Real-time Ray Tracing. *Proceedings of the International Conference on Computer Graphics Theory and Applications*. 2018.

[**Vii+18b**] T. Viitanen, M. Koskela, P. Jääskeläinen, A. Tervo and J. Takala. PLOCTree: A Fast, High-Quality Hardware BVH Builder. *Proceedings of the ACM on Computer Graphics and Interactive Techniques* 1.2 (2018).

[Vii18] T. Viitanen. Hardware Accelerators for Animated Ray Tracing. PhD thesis. 2018.

[Vog+18] T. Vogels, F. Rousselle, B. McWilliams, G. Röthlin, A. Harvill, D. Adler, M. Meyer and J. Novák. Denoising with Kernel Prediction and Asymmetric Loss Functions. *Transactions on Graphics* 37.4 (2018).

[Vrg18] Vrgineers. *XTAL Technical Specification*. Available: `https://vrgineers.com/xtal/technical-specification/` Accessed: 2019-09-11. 2018.

[Wal+09] I. Wald, W. Mark, J. Günther, S. Boulos, T. Ize, W. Hunt, S. Parker and P. Shirley. State of the Art in Ray Tracing Animated Scenes. *Computer Graphics Forum* 28.6 (2009).

[Wan95] B. Wandell. *Foundations of Vision*. Available: `https://foundationsofvision.stanford.edu/` Accessed: 2019-04-17. Sinauer Associates, 1995.

[Wat+97] B. Watson, N. Walker, L. Hodges and A. Worden. Managing Level of Detail Through Peripheral Degradation: Effects on Search Performance with a Head-Mounted Display. *Transactions on Computer-Human Interaction* 4.4 (1997).

[WBS07] I. Wald, S. Boulos and P. Shirley. Ray Tracing Deformable Scenes Using Dynamic Bounding Volume Hierarchies. *Transactions on Graphics* 26.1 (2007), 6.

[Wei+16] 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 (2016).

[Wei+17] 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 (2017).

[Wei+18a] M. Weier, T. Roth, A. Hinkenjann and P. Slusallek. Foveated Depth-of-Field Filtering in Head-Mounted Displays. *Transactions on Applied Perception* 15.4 (2018).

[Wei+18b] M. Weier, T. Roth, A. Hinkenjann and P. Slusallek. Predicting the Gaze Depth in Head-Mounted Displays Using Multiple Feature Regression. *Proceedings of the ACM Symposium on Eye Tracking Research & Applications*. 2018.

[Whi80] T. Whitted. An Improved Illumination Model for Shaded Display. *Communications of the ACM* 23.6 (1980).

[Wil+15] L. Wilcox, R. Allison, J. Helliker, B. Dunk and R. Anthony. Evidence That Viewers Prefer Higher Frame-Rate Film. *Transactions on Applied Perception* 12.4 (2015).

[Yan+18] X. Yang, D. Wang, W. Hu, L. Zhao, X. Piao, D. Zhou, Q. Zhang, B. Yin, Q. Cai and X. Wei. Fast Reconstruction for Monte Carlo Rendering Using Deep Convolutional Networks. *IEEE Access* 7 (2018).

[Yan+19] X. Yang, W. Hu, D. Wang, L. Zhao, B. Yin, Q. Zhang, X. Wei and H. Fu. DEMC: A Deep Dual-Encoder Network for Denoising Monte Carlo Rendering. *arXiv preprint arXiv:1905.03908* (2019).

[YKL17] H. Ylitie, T. Karras and S. Laine. Efficient Incoherent Ray Traversal on GPUs Through Compressed Wide BVHs. *Proceedings of High Performance Graphics*. 2017.

[YWY10] X. Yu, R. Wang and J. Yu. Real-Time Depth of Field Rendering via Dynamic Light Field Generation and Filtering. *Computer Graphics Forum* 29.7 (2010).

[Zha+18] R. Zhang, P. Isola, A. Efros, E. Shechtman and O. Wang. The Unreasonable Effectiveness of Deep Features as a Perceptual Metric. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018.

[Zim+15] H. Zimmer, F. Rousselle, W. Jakob, O. Wang, D. Adler, W. Jarosz, O. Sorkine-Hornung and A. Sorkine-Hornung. Path-space Motion Estimation and Decomposition for Robust Animation Filtering. *Computer Graphics Forum* 34.4 (2015).

[Zwi+15] M. Zwicker, W. Jarosz, J. Lehtinen, B. Moon, R. Ramamoorthi, F. Rousselle, P. Sen, C. Soler and S.-E. Yoon. Recent Advances in Adaptive Sampling and Reconstruction for Monte Carlo Rendering. *Computer Graphics Forum* 34.2 (2015).

PUBLICATIONS

PUBLICATION 1

[P1]

Foveated Path Tracing: A Literature Review and a Performance Gain Analysis

M. Koskela, T. Viitanen, P. Jääskeläinen and J. Takala

Proceedings of International Symposium on Visual Computing. Ed. by I. Daisuke and A. Sadagic. 2016

DOI: 10.1007/978-3-319-50835-1_65

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, Tampere, Finland
matias.koskela@tut.fi

Abstract. Virtual Reality (VR) places demanding requirements on the rendering pipeline: the rendering is stereoscopic and the refresh rate should be as high as 95 Hz 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.

VR and AR devices require refresh rates as high as 95 Hz and maximum latency of 20 ms 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 Sect. 2. Then we explain foveated rendering in Sect. 3. We conduct the theoretical performance gain estimation in Sect. 4 and conclude the paper in Sect. 5.

G. Bebis et al. (Eds.): ISVC 2016, Part I, LNCS 10072, pp. 723–732, 2016.
DOI: 10.1007/978-3-319-50835-1_65

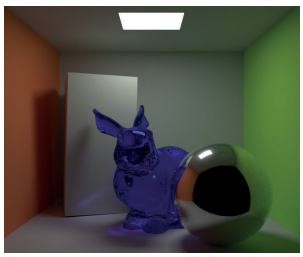


Fig. 1. Example of a path traced image

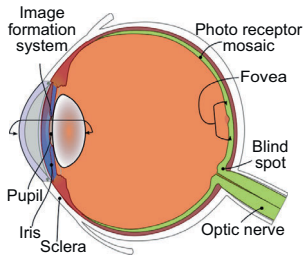


Fig. 2. Different parts of the human eye

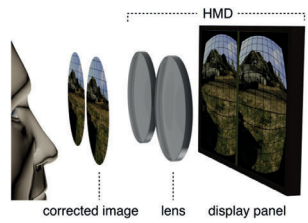


Fig. 3. Barrel distortion is used when rendering VR frames (Image by Daniel Pohl, licensed under https://creativecommons.org/licenses/by/4.0/)

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

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.

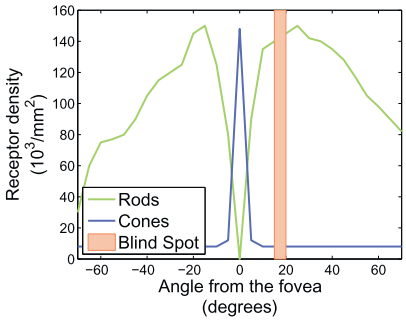


Fig. 4. Example of (*rod*) and (*cone*) density as a function of the angle to the center of the fovea

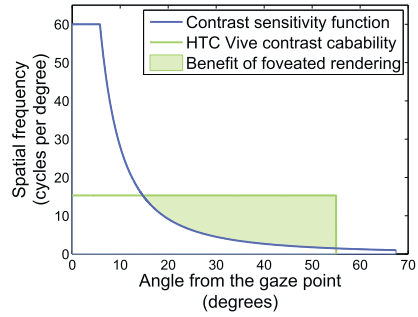


Fig. 5. A slice of the (*Contrast sensitivity function*), which models how much details an eye is able to resolve

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° on horizontal and 135° 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° 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. Guenter 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. Swafford 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 min to 15 min 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 c/deg, 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 60 c/deg details with the oval FOV of 160° 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 60 c/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 15.3 c/deg [32]. We limit the Eq. 2 to this number and calculate the integral over a circular area with radius of 110°. 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 Guenter 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 90 Hz and a resolution of 2160×1200 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.

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 Steam Dev Days, Seattle (2014)
2. Wald, I., Benthin, C., Boulos, S.: Getting rid of packets - efficient SIMD single-ray traversal using multi-branching BVHs. In: Proceedings of the IEEE Symposium on Interactive Ray Tracing (2008)
3. Laine, S., Karras, T., Aila, T.: Megakernels considered harmful: wavefront path tracing on GPUs. In: Proceedings of the High-Performance Graphics (2013)
4. Garanzha, K., Premože, S., Bely, A., Galaktionov, V.: Grid-based SAH BVH construction on a GPU. Vis. Comput. **27**(6–8), 697–706 (2011)
5. Pantaleoni, J., Luebke, D.: HLBVH: hierarchical LBVH construction for real-time ray tracing of dynamic geometry. In: Proceedings of the High-Performance Graphics (2010)
6. Karras, T.: Maximizing parallelism in the construction of BVHs, octrees, and k-d trees. In: Proceedings of the High-Performance Graphics (2012)
7. Keely, S.: Reduced precision hardware for ray tracing. In: Proceedings 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. Graph. **33**(4), 143 (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. Graph. **29**(4), 66 (2010)
10. AMD: RadeonRays SDK (2016). https://github.com/GPUOpenLibrariesAndSDKs/RadeonRays_SDK. Accessed 6 Oct 2016
11. Bikker, J., van Schijndel, J.: The brigade renderer: a path tracer for real-time games. Int. J. Comput. Games Technol. **2013**, 1–14 (2013). https://www.hindawi.com/journals/ijcgt/2013/578269/
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. Comput. Graph. Forum **34**(2), 667–681 (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 Comput. Graph. Appl. **21**(5), 68–75 (2001)

16. Bowman, D.A., Kruijff, E., LaViola Jr., J.J., Poupyrev, I.: 3D User Interfaces: Theory and Practice. Addison-Wesley, New York (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: Proceedings of the ACM Symposium on User Interface Software and Technology (2015)
18. Hua, H.: Integration of eye tracking capability into optical see-through head-mounted displays. In: Proceedings of 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: Proceedings of the ACM International Conference on Multimedia (2015)
20. Salvucci, D.D., Goldberg, J.H.: Identifying fixations and saccades in eye-tracking protocols. In: Proceedings of the Eye Tracking Research and Applications (2000)
21. Ohshima, T., Yamamoto, H., Tamura, H.: Gaze-directed adaptive rendering for interacting with virtual space. In: Proceedings of the VR Annual International Symposium (1996)
22. Shic, F., Scassellati, B., Chawarska, K.: The incomplete fixation measure. In: Proceedings of the 2008 Symposium on Eye Tracking Research and Applications (2008)
23. Ji, Q., Yang, X.: Real time visual cues extraction for monitoring driver vigilance. In: Schiele, B., Sagerer, Gerhard (eds.) ICVS 2001. LNCS, vol. 2095, pp. 107–124. Springer, Heidelberg (2001). doi:10.1007/3-540-48222-9_8
24. Murphy, H.A., Duchowski, A.T., Tyrrell, R.A.: Hybrid image/model-based gaze-contingent rendering. ACM Trans. Appl. Percept. **5**(4), 22 (2009)
25. Siekawa, A., Mantiuk, S.R.: Gaze-dependent ray tracing. In: Proceedings of Central European Seminar on Computer Graphics (non-peer-reviewed) (2014)
26. Mantiuk, R., Janus, S.: Gaze-dependent ambient occlusion. In: Bebis, G., et al. (eds.) ISVC 2012. LNCS, vol. 7431, pp. 523–532. Springer, Heidelberg (2012). doi:10.1007/978-3-642-33179-4_50
27. Guenter, B., Finch, M., Drucker, S., Tan, D., Snyder, J.: Foveated 3D graphics. ACM Trans. Graph. **31**(6), 164 (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: Proceedings 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: Proceedings of the ACM Symposium on Applied Perception (2016)
31. Luebke, D., Hallen, B.: Perceptually driven simplification for interactive rendering. In: Proceedings of the Eurographics Workshop (2001)
32. Vlochos, A.: Advanced VR rendering Game Developers Conference, San Francisco (2015)
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: Proceedings 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: Proceedings of the ACM Symposium on VR Software and Technology (2013)

PUBLICATION 2

[P2]

Blockwise Multi-Order Feature Regression for Real-Time Path Tracing Reconstruction

M. Koskela, K. Immonen, M. Mäkitalo, A. Foi, T. Viitanen, P. Jääskeläinen, H. Kultala and J. Takala

Transactions on Graphics 38.5 (2019)
DOI: 10.1145/3269978

Blockwise Multi-Order Feature Regression for Real-Time Path Tracing Reconstruction

MATIAS KOSKELA, KALLE IMMONEN, MARKKU MÄKITALO, ALESSANDRO FOI, TIMO VIITANEN, PEKKA JÄÄSKELÄINEN, HEIKKI KULTALA, and JARMO TAKALA, Tampere University, Finland

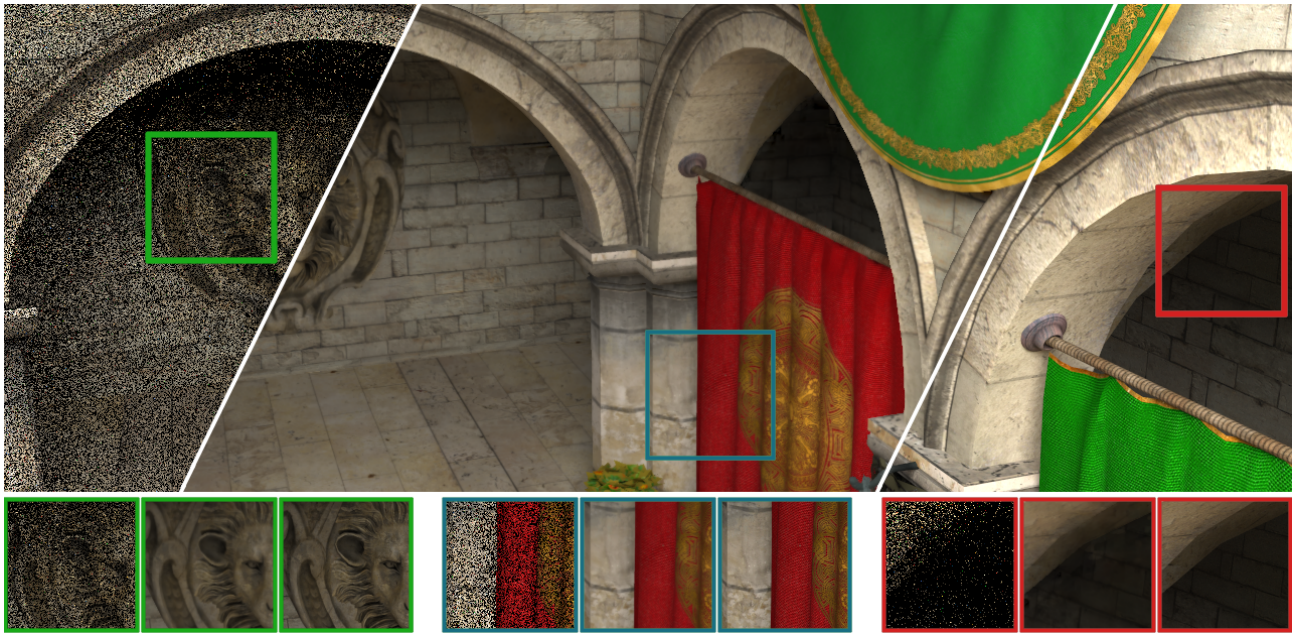


Fig. 1. In all image sets, left: 1 sample per pixel path-traced input, center: result of the proposed post-processing denoising/reconstruction pipeline, and right: 4096 samples per pixel reference. Leftmost highlights: the lion is barely visible in the input, but the proposed pipeline is able to produce realistic illumination results without blurring the edges and high-frequency albedo details. Center highlights: the best case for the pipeline is geometry with sufficient light in the input. Rightmost highlights: the worst case for the pipeline is the one with occlusions and almost no light, resulting in blurry artifacts.

Path tracing produces realistic results including global illumination using a unified simple rendering pipeline. Reducing the amount of noise to imperceptible levels without post-processing requires thousands of *samples per pixel* (spp), while currently it is only possible to render extremely noisy 1 spp frames in real time with desktop GPUs. However, post-processing can utilize feature buffers, which contain noise-free auxiliary data available in the rendering pipeline. Previously, regression-based noise filtering methods have only been used in offline rendering due to their high computational cost. In this paper we propose a novel regression-based reconstruction pipeline, called *Blockwise Multi-Order Feature Regression* (BMFR), tailored for path-traced 1 spp inputs that runs in real time. The high speed is achieved with a fast implementation of augmented QR factorization and by using stochastic regularization to address rank-deficient feature data. The proposed algorithm is 1.8× faster than the previous state-of-the-art real-time path tracing reconstruction method while producing better quality frame sequences.

CCS Concepts: • **Computing methodologies** → **Ray tracing**; *Rendering*; *Image processing*;

Additional Key Words and Phrases: path tracing, reconstruction, regression, real-time

ACM Reference Format:
Matias Koskela, Kalle Immonen, Markku Mäkitalo, Alessandro Foi, Timo Viitanen, Pekka Jääskeläinen, Heikki Kultala, and Jarmo Takala. 2019. Blockwise Multi-Order Feature Regression for Real-Time Path Tracing Reconstruction. *ACM Trans. Graph.* X, Y, Article Z (May 2019), 14 pages. https://doi.org/0000001.0000001

Authors' address: Matias Koskela, matias.koskela@tuni.fi; Kalle Immonen, kalle.immonen@aspekt.fi; Markku Mäkitalo, markku.makitalo@tuni.fi; Alessandro Foi, alessandro.foi@tuni.fi; Timo Viitanen, viitanet@gmail.com; Pekka Jääskeläinen, pekka.jaaskelainen@tuni.fi; Heikki Kultala, heikki.kultala@tuni.fi; Jarmo Takala, jarmo.takala@tuni.fi, Tampere University, Tampere, 33720, Finland.

1 INTRODUCTION

Real-time path tracing has been a long-standing goal of graphics rendering research due to its ability to produce natural soft shadows, reflections, refractions, and global illumination effects using a conceptually simple unified drawing method. However, its computational complexity is a major challenge; contemporary ray tracing frameworks [AMD 2017; Parker et al. 2010; Wald et al. 2014]

are able to produce only around one path tracing *sample per pixel* (spp) at real-time frame rates on desktop-class hardware. It is expected that the real-time performance will increase in the near future as new generations of high-end GPUs will integrate hardware acceleration for ray tracing [Patel 2018]. Nevertheless, a linear improvement in rendering quality requires a quadratic increase in computational complexity: to halve the signal-to-noise ratio in path tracing, the number of samples per pixel has to be quadrupled [Pharr and Humphreys 2010]. Consequently, reducing the amount of noise to imperceptible levels without post-processing requires thousands of samples per pixel and, therefore, denoising filters are used even in offline path-traced movie rendering [Goddard 2014].

The trend of rising resolutions and refresh rates, driven especially by the needs of virtual reality immersion, increases the amount of required computations at the same rate as the computing hardware is improved. As a consequence, it is unrealistic to expect the computing hardware performance to improve fast enough to support real-time path tracing at high frame rates. It seems that the achievable real-time path tracing sample rates will remain around 1 spp with the consumer devices of the near future [Alla Chaitanya et al. 2017; Schied et al. 2017; Viitanen et al. 2018]. Therefore, there is an urgent need for novel real-time post-processing denoising methods that are targeted for 1 spp path-traced inputs.

Constructing high quality results from a 1 spp starting point is hard even when done offline without strict real-time constraints. The input has an extreme amount of noise, much more than conventional image denoising algorithms can handle. However, the reconstruction results can be improved by utilizing *feature buffers*, which contain noise-free auxiliary data available from the path tracer. The buffers can include useful information such as surface normals and texture albedo colors. As is essential for the real-time goal, this information can be extracted from a path tracer with little performance overhead. Utilizing feature buffers allows reconstruction filters to, e.g., avoid blurring samples across geometry edges, which is a very disturbing artifact for the human eye, or it can reduce smearing the details in the textures.

Moreover, fast path tracers can reproject and accumulate samples from multiple previous frames to reduce temporal noise that varies between successive frames. Flickering artifacts are especially noticeable by the end users. Real-time denoising algorithms must specifically account for the temporal noise as there is no option of simply adding more samples per pixel and the denoising needs be fast enough to fit in the time slot left over from the rendering.

In this article we propose a new regression-based reconstruction pipeline optimized for 1 spp input images that runs in real time on desktop GPUs. The proposed method is 1.8× faster and has better objective quality than the previous state-of-the-art real-time path tracing reconstruction method. The article presents the following contributions:

- A novel *Blockwise Multi-Order Feature Regression* (BMFR) algorithm, where multiple versions of the feature buffers of different orders are used for fitting.
- A fast GPU-based implementation of the BMFR algorithm.
- Proposal to use stochastic regularization to address the possible rank-deficiency of the blockwise features, avoiding numerical instabilities without the extra complexity of pivoting.

In other words, the proposed algorithm combines a completely novel concept (multi-order feature buffers) with a few established concepts (feature regression, QR factorization). Regression-based methods have typical had execution times in order of seconds [Moon et al. 2016] and have been considered to be applicable only in offline context [Alla Chaitanya et al. 2017; Schied et al. 2017]. However, we do regression in an unusual way (blockwise processing, augmented factorization with stochastic regularization) and, therefore, the proposed method is the first regression-based method to achieve real-time performance.

2 RELATED WORK

Path tracing reconstruction methods are covered in a recent comprehensive survey article [Zwicker et al. 2015]. In general, the methods can be divided into three categories based on their complexity: offline methods, interactive methods, and real-time methods. Real-time methods are closest to the context of this article, but we also draw ideas from and compare to methods from the other categories.

Naturally, the best reconstruction quality for path tracing can be achieved with offline methods. Since there is no strict time budget, offline methods can use complicated and slow algorithms. Furthermore, as they are not constrained by real-time deadlines, their execution time can vary heavily based on the input data. Typically, offline methods target inputs that have more than 1 spp, because it is not a problem to generate more path tracing samples if the filtering itself takes a comparatively long time. In offline methods it is also possible for the filtering to guide the sample generation process in path tracing so that more samples are generated at problematic areas in screen space [Li et al. 2012]. Offline reconstruction can be implemented, for example, with general edge-preserving image filters like guided image filtering [He et al. 2013; Liu et al. 2017] or bilateral filtering [Tomasi and Manduchi 1998], which are guided with feature buffers. Another approach is to use a neural network [Kalantari et al. 2015], which can be trained even with a complete set of frames from a feature-length movie [Bako et al. 2017]. A third approach is to fit the noise-free feature buffers to the noisy image data [Bitterli et al. 2016; Moon et al. 2014, 2015].

Neural networks can also be used at interactive frame rates as shown recently by Alla Chaitanya et al. [2017]. Since the quality of the interactive methods is not as good as in offline methods, extra care needs to be taken to address temporal stability of the results. One way to address temporal noise is to use recurrent connections in each neural network layer [Alla Chaitanya et al. 2017]. Sheared filtering is another approach to achieve interactive frame rates [Yan et al. 2015]. In contrast to the neural network approach, sheared filtering also supports effects that produce noise to the feature buffers, such as motion blur [Egan et al. 2009].

Reconstruction based on the guided image filter is the closest method in the literature to the proposed one which can also reach interactive frame rates [Bauszat et al. 2011]. However, it is not an appealing approach for real-time implementation on modern GPUs,

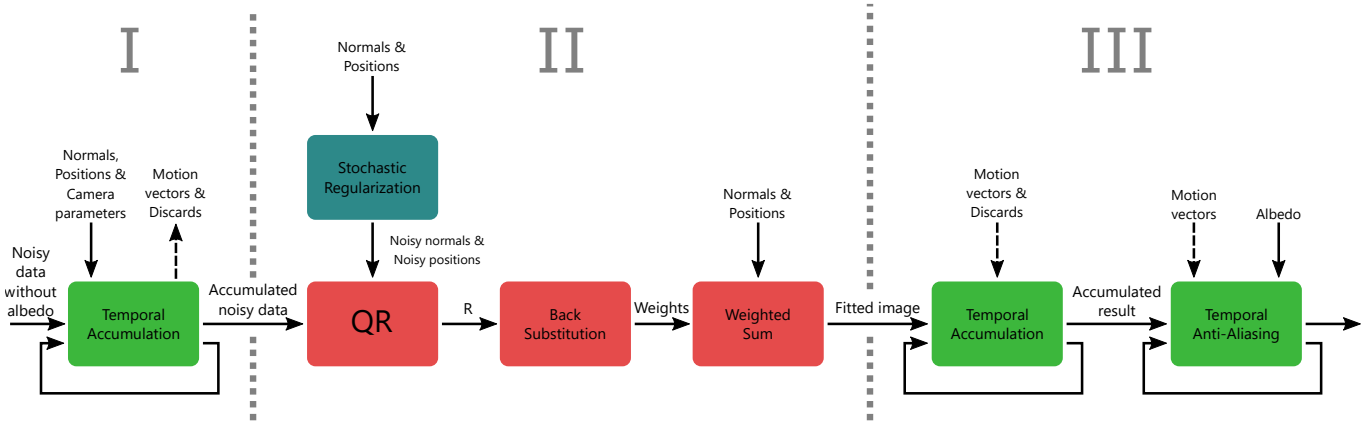


Fig. 2. Overview of the proposed reconstruction pipeline. The pipeline inputs a noisy 1 spp path-traced frame and the corresponding normal and world-space position buffers. It outputs a noise-free image with a good approximation of global illumination. Without the stochastic regularization, the back substitution block produces NaNs and Infs due to rank deficiency.

since it requires either dozens of moving window operations or generating as many summed-area tables. Moving window operations involve several orders of magnitude less parallel work than a modern GPU can process concurrently, whereas generating summed-area tables requires an expensive parallel scan pattern and higher precision values stored in the buffers.

There is recent research interest on algorithms that can perform path tracing reconstruction in real time. A way to achieve required execution speed is to use approximations or variants of the bilateral filter, such as a sparse bilateral filter [Mara et al. 2017], or a hierarchical filter with multiple iterations [Burt 1981] expanded with customized edge-stopping functions [Dammertz et al. 2010; Schied et al. 2017].

Real-time methods are typically targeted for 1 spp inputs because the motivation for attempting to perform the reconstruction in real time is low if the input frames must be computed offline anyway. In case of 1 spp inputs and fast lower quality reconstruction, even higher degree of variation is expected in the results, making temporal stability an even bigger problem with real-time methods.

Temporal stability can be improved by accumulating projected frames [Yang et al. 2009], which produces a greater effective spp and more static noise in world coordinate locations. A similar idea can also be used for dealing with ambient occlusions [Jiménez et al. 2016]. However, in these reprojection-based techniques some of the rendered pixels cannot utilize the accumulated data because they were occluded in the previous frame. Such *disocclusion events* can be recognized, for example, based on inconsistencies in the world-space position or normal data in the feature buffers for the subsequent frames. Interestingly, the reprojection method can also support, for example, rigid body animations if there is a way to find out where the current pixel was in the previous frame [Rosado 2007]. Temporal stability can be further improved, e.g., with simple *Temporal Anti-Aliasing* (TAA) [Karis 2014], which uses colors from the neighborhood of the pixel in the current frame to adjust the data sampled from the previous frame. The idea of using temporal data in anti-aliasing was introduced in *Enhanced Subpixel Morphological Antialiasing* (SMAA) [Jimenez et al. 2012].

As in previous work, the proposed reconstruction algorithm also utilizes TAA, and also reprojects and accumulates noisy data from previous frames. However, we dynamically change the weight of the new frame so that first samples after an occlusion do not get overweighted. Moreover, we add an additional step of data accumulation after filtering to increase temporal stability and to avoid artifacts. Moreover, instead of using the typical approximations of the bilateral filter we use regression-based reconstruction, which has been previously considered too slow for real-time use cases [Alla Chaitanya et al. 2017; Schied et al. 2017]. By means of applying augmented QR factorization and stochastic regularization we made the regression fast enough for real-time use. Finally, we introduce BMFR, where multiple versions of the feature buffers of different orders are used for fitting, improving the chances for the fitting to succeed.

3 RECONSTRUCTION PIPELINE

The proposed reconstruction pipeline can be divided into three main phases: preprocessing, feature fitting and post-processing. The phases, marked with roman numerals, are illustrated in Fig. 2 and explained in subsections below. The proposed algorithm does not need to guide the path tracing process in any way.

3.1 Input

The input for the real-time reconstruction filter is a 1 spp path-traced frame and its accompanying feature buffers. The 1 spp frames are generated by using a rasterizer for producing the primary rays and feature buffers. We use mipmapped textures in albedo. Next, we do so-called next event estimation: we trace one shadow ray towards a random point in one random light source and then continue path tracing by sending one secondary ray to a random direction. Namely, we use multiple importance sampling [Veach and Guibas 1995]. The direction of the secondary ray is decided based on importance sampling. We also trace a second shadow ray from the intersection point of the secondary ray. Consequently, the 1 spp

pixel input has one rasterized primary ray, one ray-traced secondary ray and two ray-traced shadow rays. The random numbers in the path tracer were generated with Wang hash [Wang et al. 2008]. The ray configuration was chosen because it can be path traced in real time and is able to reproduce effects like realistic global illumination, soft shadows, and reflections. Every time we refer to *1 spp data* in this article, we refer to this ray configuration.

Before inputting the 1 spp input into our post-processing pipeline, we remove first bounce surface albedo from it. Reconstructing without albedo is a common practice [Alla Chaitanya et al. 2017; Bako et al. 2017; Mara et al. 2017; Schied et al. 2017] because it ensures that high-frequency details in first-bounce textures are not blurred by the filter. The other commonly used idea is to decompose the lighting contribution to a direct and indirect component [Bauszat et al. 2011; Mara et al. 2017]. However, we do not do the separation, because it typically assumes that the direct lighting component is noise-free. Instead, we have 1 spp path-traced soft shadows in the direct component and we filter both components at once. Filtering two noisy components separately would require running the pipeline twice, which does not double the runtime since heaviest parts of the work can be shared. However, we did not find significant quality increase and the slowdown is unacceptable in our real-time context.

If the scene contains multilayer materials, the pipeline has to be run separately for every material's illumination component. However, all illumination components can be combined and filtered at once if the albedo is the same for every layer. An optimization opportunity for multilayer materials is to compute a weighted sum of different albedos and illuminations and filter all illuminations at once [Schied et al. 2017]. Even though combining the illuminations before filtering does not produce a physically correct result, this approach can be used as a fast approximation.

3.2 Preprocessing

The preprocessing phase (I) consists of temporal accumulation of the noisy 1 spp data, which reprojects the previous accumulated data to the new camera frame. In the reprojection process, world-space positions and shading normals are used to test whether we can accumulate previous data or have to fall back to the current frame's 1 spp path-traced result. Because of accumulation, in most of the pixels the effective spp can be greater than 1 even though the individual frame inputs are 1 spp. In addition, accumulation improves temporal stability of the noise.

Following a previous work [Schied et al. 2017; Yang et al. 2009], we compute an *exponential moving average* and mix 80% of the history data with 20% of the current frame data. However, we apply one significant modification compared to the previous work: we start by computing a *cumulative moving average* of the samples, and use the exponential moving average only after the cumulative moving average weight of the new sample would be less than 20%. The use of regular average on the first frames and after occlusions makes sure that the first samples do not get an excessively high weight, and limiting the weight to a minimum of 20% makes sure that the aged data fades away.

Computing the cumulative moving average requires that we store and update the sample count of every pixel. Since we are interested in the sample count only if the count is small, the values can be stored in just a few bits. Loading and storing, for example, 8-bit integers is insignificant compared to other memory accesses of the temporal accumulation.

We use bilinear sampling of the history data and do a discard test for each pixel separately. The final color is normalized by the sum of the accepted sample weights only, thus the discarded pixels do not affect the brightness of the sample. Also the sample count data is sampled using the same custom bilinear sampling and the result is rounded to the closest integer value.

3.3 Blockwise Multi-Order Feature Regression (BMFR)

The feature fitting phase (II) is based on the following feature regression operated on non-overlapping image blocks, covering the entire single frame.

Let $F = \left[F_1, \ldots, F_N\right]$ denote a set of available noise-free feature buffers, such as the world-space positions and shading normals. Typically these buffers are created as a side product of the path tracing. However, they could contain artificially created data like gradients. Every buffer in F has the same resolution as the noisy frame. We consider an *extended* set T of M feature buffers:

$$T = \left[F_{n_1}^{\gamma_1}, \ldots, F_{n_m}^{\gamma_m}, \ldots, F_{n_M}^{\gamma_M}\right], \tag{1}$$

where $n_m \in \{1, \ldots, N\}$, $m = 1, \ldots, M$, $\gamma_1 = 0$, and $\gamma_m > 0$, $m = 2, \ldots, M$, are positive exponents. The first buffer in T is a constant buffer $F_{n_1}^0 = \mathbf{1}$. Note that γ_m, $m > 1$, need not be an integer and can be larger as well as smaller than 1.

Denoting by $\Omega_{i,j}$ the set of absolute coordinates of the pixels within an image block located at position (i, j), the *Blockwise Multi-Order Feature Regression* (BMFR) problem can be formulated like a standard least-squares expression with respect to the multi-order features T as

$$\hat{\alpha}^{(c)} = \underset{\alpha^{(c)} \in \mathbb{R}^M}{\operatorname{argmin}} \sum_{(p,q) \in \Omega_{i,j}} \left(Z^{(c)}(p,q) - \sum_{m=1}^{M} \alpha_m^{(c)} F_{n_m}^{\gamma_m}(p,q) \right)^2, \tag{2}$$

where $Z^{(c)}$ is the c channel of the noisy path-traced input which can be temporally accumulated (e.g., c may be red, green, blue, or any relevant luminance or chrominance component). The estimate of the noise-free scene Y for channel c and block $\Omega_{i,j}$ is thus

$$\hat{Y}^{(c)}(p,q) = \sum_{m=1}^{M} \hat{\alpha}_m^{(c)} F_{n_m}^{\gamma_m}(p,q)\,. \tag{3}$$

While being a simple linear solution, it is non-linear w.r.t. the features F, which makes it more flexible and capable of better fit to the data than established methods based on linear regression on F.

3.4 Feature Fitting with Stochastic Regularization

We solve the least-squares problem (2) by the Householder QR factorization [Heath 1997]. Specifically, and using matrix-vector notation, let us reshape the M blockwise feature buffers $F_{n_m}^{\gamma_m}(p,q)$, $(p,q) \in \Omega_{i,j}$, $m = 1, \ldots, M$, as column vectors of length W, where W is the number of pixels in the block $\Omega_{i,j}$, and let $\mathbf{T}$ be the $W \times M$

(a) 1 spp (b) World-space positions (c) Normals (d) World positions [2] (e) Just constant buffer in fitting (f) + world positions (g) + normals (h) + world positions [2] (i) Reference

Fig. 3. Different buffers and results with a single 64×64 block of BMFR. Notice how adding world-space positions squared allows the fitting to generate a more realistic soft shadow under the edge. In the fast implementation we use 32×32 blocks, but the larger blocks visualize the benefit in a single block more clearly. The results get closer to the reference when temporal accumulation averages multiple blocks from different displacements.

matrix obtained by horizontal concatenation of such column vectors. Further, let

$$\widetilde{\mathbf{T}}^{(c)} = \left[\mathbf{T}, \mathbf{z}^{(c)}\right] \tag{4}$$

be the $W \times (M+1)$ matrix obtained by augmenting $\mathbf{T}$ with $\mathbf{z}^{(c)}$, which is $Z^{(c)}(p,q)$, reshaped into a column vector of length W. We expect $W \gg M$, meaning that each block has much more pixels than there are feature buffers.

Assuming that $\widetilde{\mathbf{T}}^{(c)}$ is full rank, the Householder QR factorization yields an $(M+1) \times (M+1)$ upper triangular matrix $\widetilde{\mathbf{R}}^{(c)}$ such that $\widetilde{\mathbf{T}}^{(c)} = \widetilde{\mathbf{Q}}^{(c)}\widetilde{\mathbf{R}}^{(c)}$, where $\widetilde{\mathbf{Q}}^{(c)}$ is a $W \times (M+1)$ matrix with orthonormal columns. Given $\widetilde{\mathbf{R}}^{(c)}$, there is no need to compute $\widetilde{\mathbf{Q}}^{(c)}$ for solving the linear least squares problem; instead, we can solve the transformed system contained in $\widetilde{\mathbf{R}}^{(c)}$ [Heath 1997, pp. 92-93]. By dealing just with the smaller matrix $\widetilde{\mathbf{R}}^{(c)}$ we get a significant performance improvement.

Specifically, if we denote by $\mathbf{R}$ and by $\mathbf{r}^{(c)}$, respectively, the top-left $M \times M$ sub-matrix and the top-right $M \times 1$ sub-column of $\widetilde{\mathbf{R}}^{(c)}$, the solution $\hat{\alpha}^{(c)}$ of (2) is given as

$$\mathbf{R}\hat{\boldsymbol{\alpha}}^{(c)} = \mathbf{r}^{(c)}, \tag{5}$$

which can be solved, for example, via back substitution, which is simple and fast. Hence, $\hat{Y}^{(c)}(p,q)$, $(p,q) \in \Omega_{i,j}$, (3) is obtained as

$$\hat{\mathbf{y}}^{(c)} = \mathbf{T}\hat{\boldsymbol{\alpha}}^{(c)}, \tag{6}$$

where $\hat{\mathbf{y}}^{(c)}$ is $\hat{Y}^{(c)}$ reshaped into a column vector of length W. Observe that $\mathbf{R}$ (and its inverse) does not depend on $\mathbf{z}^{(c)}$, and that $\mathbf{r}^{(c)}$ can be computed for different channels without recalculating $\mathbf{R}$, which allows to process multiple channels with minimal extra cost.

In practice, $\widetilde{\mathbf{T}}^{(c)}$ may be rank-deficient, leading to numerical instabilities that break the factorization. While the rank-deficiency is typically managed by pivoting, we employ *stochastic regularization*. That is, we add noise to the input buffers, which makes them linearly independent, i.e., (4) becomes

$$\widetilde{\mathbf{T}}^{(c)} = \left[\mathbf{T} + \mathbf{N}, \mathbf{z}^{(c)}\right], \tag{7}$$

where $\mathbf{N}$ is a $W \times M$ matrix of zero-mean independent and identically distributed noise. T within every block is scaled to be in range $[-1, 1]$, before this addition. Since the average of the noise is zero, we can expect that this regularization does not increase the fitting bias. The synthesis (6) always uses the noise-free buffers $\mathbf{T}$, so the noise itself is not visible in the reconstructed estimate. In our implementation, we use zero-mean uniformly distributed noise over an interval $[-\varepsilon, \varepsilon]$, thus having variance $\varepsilon^2/3$. The value of ε that worked with all our tested scenes was 0.01. Much stronger noise ($\varepsilon \approx 1.0$) caused visibly too bright and dark constant blocks, whereas much weaker noise ($\varepsilon \approx 0.0001$) failed to regularize, leading to divisions by zero in the factorization.

3.5 Post-processing

The purpose of the post-processing phase (III) is to increase temporal stability and the perceived visual quality.

First, the fitted frame is temporally accumulated, which reduces blocky artifacts caused by operating the BMFR algorithm on non-overlapping blocks and improves temporal stability. Importantly, small fitting errors caused by the stochastic regularization can be expected to cancel out when multiple frames are accumulated because the injected noise has a zero mean. To aid the reduction of blockiness, BMFR processes each frame over a grid of non-overlapping blocks which is displaced with random offsets. These offsets prevent the artifacts that would arise from reusing same block positions on a static scene with a static camera.

This post-processing phase is essentially the same process that was done in the preprocessing step to increase the effective spp. However, the process is faster because bandwidth can be saved by reusing the motion vectors and discard decisions from the preprocessing phase. By loading for every pixel just 2 floats and 4 booleans, we avoid loading all 5 world-space positions and shading normals again, all containing three channels (1 from current frame, and 4 for bilinear sampling of the previous frame).

In this second temporal accumulation we use 10% of new data and 90% old data because these values hide the block place variations. Similarly to the first temporal accumulation we use the cumulative moving average until the weight of the new sample has reached the chosen 10%. Using the cumulative moving average in this second temporal accumulation is crucial since the first block fitted after an occlusion is more likely to contain outlier data and with the cumulative moving average it is mixed with subsequent frames more quickly. For example, if we used the exponential moving average, after three frames the weight of the very first fitted data would still be more than half. With cumulative moving average the weight is the same as in a regular average: one third.

As a last step of the pipeline, TAA [Karis 2014] is used. While in many of the test scenes TAA decreases the quality measured by the

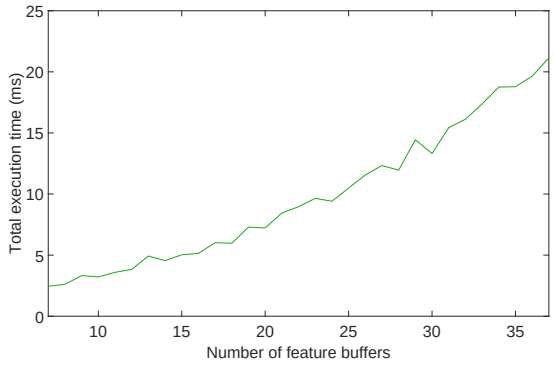


Fig. 4. The execution time of the whole pipeline with different counts of feature buffers. The QR block size used in this measurement was 32×32. In the rest of the runtime results we use 10 feature buffers. All of the test scenes have a similar runtime since the runtime varies only in the stages that access previous frame data from pixels stated by the motion vectors.

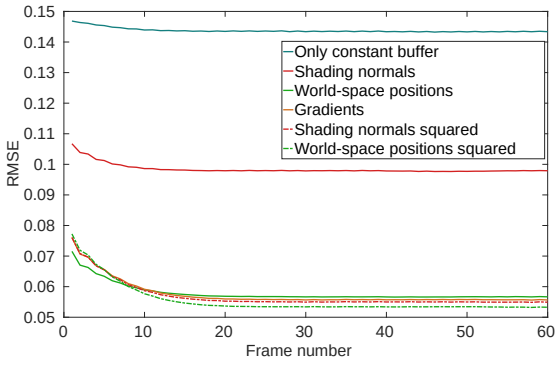


Fig. 5. The effect on denoising quality as more sets of buffers are added cumulatively, measured by *Root Mean Square Error* (RMSE) (lower is better) for the Sponza test scene with a static camera. The buffers are greedily added in the order specified in the legend from top to bottom.

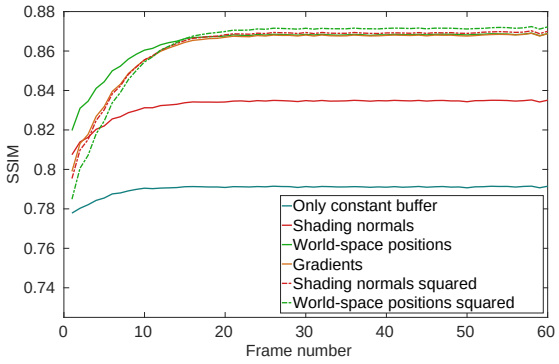


Fig. 6. The effect on denoising quality as more sets of buffers are added cumulatively, measured by *Structural SIMilarity* (SSIM) [Wang et al. 2004] (higher is better) for the Sponza test scene with a static camera. The buffers are greedily added in the order specified in the legend from top to bottom.

objective quality metrics, in our experience it gives more visually pleasing results.

4 COMPLEXITY ANALYSIS

Phases I and II in the proposed pipeline can be implemented using the parallel map and parallel stencil patterns. Thus, the execution time of these phases is linearly dependent on the number of pixels in the input image. In these phases adding more feature buffers only increases the amount of data stored in first accumulation stage. In other words, the processing can be parallelized easily because the result pixels are independent of each other. However, adding more computing hardware is likely to quickly reach its limits because all the stages are mostly memory bound.

The most important stage in the pipeline regarding the complexity analysis is the QR stage. When the number of pixels in the input image is increased, the number of QR blocks grows linearly. The blocks do not affect each other in any way, so all of them can be loaded and processed in parallel, and therefore performance scales linearly. In contrast, if one feature buffer is added, it must be transformed by all of the previous feature buffers. The transform requirement comes from the Householder reflections method: the number of required transforms is $O(M(M+1)/2) = O(M^2)$, where M is the number of feature buffers. However, the work per each feature buffer in the proposed method is quite small, which can be seen in Fig. 4. With a reasonable number of feature buffers, the execution time increase is almost linear. For comparison, guided filter's [Bauszat et al. 2011] requirement is to generate $O(M^2)$ summed-area tables. Therefore, we can include more feature buffers in the same execution time to produce results that have a higher visual quality.

5 FEATURE BUFFER SELECTION

The choice of which feature buffers to include in the filtering is crucial. Including additional feature buffers increases the computational complexity by $O(M^2)$, but the resulting quality improvement varies dramatically based on the buffer type. It is thus essential in real-time filtering to include only the most beneficial feature buffers.

To this end, we measured the effects of different buffer types by greedily adding all available set of buffers to find the ones that helped the most. Greedy addition means that we tested every available buffer and added the one that improved the objective quality the most. After each addition we started the same process again with the rest of the available buffers. Fig. 5 and Fig. 6 show the obtained results for the Sponza test scene with a static camera; the corresponding results for our other test scenes yield similar conclusions.

We also experimented by adding horizontal and vertical gradient buffers consisting of a horizontal or a vertical gradient from 0 to 1 for each block, respectively. The idea was to provide more freedom for the feature regression (2). However, the gradient buffers yielded only minor quality improvements, as Fig. 5 and Fig. 6 also show.

Only minor quality improvements make sense because typically there is always gradient-like data available in the world position buffers.

Every channel of each feature buffer was added at once even though some channels might have not contributed much to the result, because otherwise the feature selection would have suffered from overfitting to camera orientations.

Based on the aforementioned measurement, we adopted the following multi-order set of feature buffers:

$$T = \left[\mathbf{1}, \mathbf{n}_x, \mathbf{n}_y, \mathbf{n}_z, \mathbf{w}_x, \mathbf{w}_y, \mathbf{w}_z, \mathbf{w}_x^2, \mathbf{w}_y^2, \mathbf{w}_z^2\right], \tag{8}$$

where $\mathbf{n}_x, \mathbf{n}_y, \mathbf{n}_z$ are the three channels of shading normals, and $\mathbf{w}_x, \mathbf{w}_y, \mathbf{w}_z$ are the three channels of world-space positions. This set of buffers was selected because, as can be seen in the figures, the error is decreased the most by adding the normals and the world-space positions. The benefit of adding further buffers appears to get negligible compared to increased execution time.

However, the computational error metrics do not reveal small problematic areas in the result, and therefore, after visual examination, we decided to add the second-order world-space positions. Fig. 3 illustrates the reason for this choice; world-space positions are particularly useful for getting more convincing soft shadows.

In the proposed method, the specular highlight is generated from a feature buffer that happens to have data similar to the highlight. If the highlight is not well presented by the available feature data, the result improves when multiple block locations from successive frames are accumulated. Adding material roughness to the set of feature buffers allows illumination to vary between regions inside a block, which only helps if there are materials that have a roughness texture with fine details. However, the constant feature buffer generates the same result if regions of the input larger than block size have uniform roughness.

6 TEST SETUP

We measured the visual quality and execution speed of the proposed algorithm while rendering animations. To provide the algorithm with a realistic amount of accumulated frame data, which is also hindered by occlusions, all except two of the test inputs had continuously moving cameras. Each frame of these animations can be found in the supplementary material of this article. One frame consists of 1 spp input data, the corresponding feature buffers, and a 4096 spp reference rendering.

In the following we describe our test setup, which includes an example implementation of the proposed algorithm and a set of compared algorithms.

6.1 GPU Implementation

To measure the performance of the proposed algorithm with realistic hardware, we implemented the algorithm using OpenCL and optimized it for a contemporary high-end desktop GPU, AMD Radeon Vega Frontier Edition. The code we wrote for the measurements is available as supplementary material of the article. The primary implementation choices that affect performance as it pertains to our target hardware are described next.

The block size was chosen to be 32×32 because even though we found that the best quality is achieved with a 64×64 block, 32×32 block gives us four times more parallel work to improve the processing element utilization. Moreover, we need to synchronize within the block, and synchronization can be done in groups of 256 parallel work items in the targeted hardware. Consequently, already with the 32×32 block, the code needs to be unrolled four times between the synchronization points.

Table 1. The number of memory accesses in parallel reduction. *Level* means combining two values into one value. *Iteration* is one code block without synchronization between parallel workers.

Levels per iteration	1	2	3	4	5
Elements per iteration	2	4	8	16	32
Memory accesses	3	5	9	17	33
Accesses per level	3	2.5	3	4.25	6.6
Accesses for 1024 elements	3069	1705	1317	1161	1089

For the displacement, we used a static sequence of 16 random offsets, uniformly distributed over the whole set of possible offsets. The displacement is done both horizontally and vertically. This number gives enough variety of displacements with the chosen blendings of history data and the new frame in temporal accumulations.

After avoiding the heavy matrix multiplications by just computing $\widetilde{\mathbf{R}}$, the computation on the targeted hardware was limited by the speed of accessing the data and performing reduction in local memory, i.e., computing the sum of all concurrently processed elements in local memory[1], which is the fastest memory space visible to the whole block.

The reduction calculations are needed for the sum calculations of the dot products and vector norms, both of which are computed multiple times in the Householder algorithm. Reduction is also used in every block to find out the local minimum and maximum of every feature, which are used to scale the values to be in the same range in the fitting. We implemented the reduction with parallel reduction, where all parallel processing elements process more than two elements on every iteration. The number of memory accesses per iteration for different counts of elements processed at once can be seen in Table 1. The fastest alternative for reduction of $32 \times 32 = 1024$ elements on our target hardware was experimentally determined to consist of summing 4 elements on the first two iterations and 8 elements on the last two iterations. This approach appears to be a good compromise between the parallelism available and the total amount of memory accesses. Fewer levels per iteration gives more parallel work. In contrast, more levels per iteration results in less memory accesses in total.

The largest implemented kernel was fitting, which contains almost all the stages of phase II. In contrast to what was found by Laine et al. [2013], in this case a single "megakernel" which included the heaviest stages of phase II was the fastest because the intermediate data could be passed through fast local memory and registers.

For faster data access we use half-precision floating-point numbers as the temporal storage type and order the pixels such that every 32×32 block is at consecutive addresses in memory. Thanks

[1] We use OpenCL terminology and call this memory space *local memory*. The corresponding CUDA term is *shared memory*.

to the memory layout, the hardware can load and store the data with faster vector accesses. It is also possible that the path tracer stores the data directly in this format because many path tracers render square blocks of pixels in one work group since then there is more cache locality in the primary rays [Aila and Karras 2010].

6.2 Compared Algorithms

We compared the proposed algorithm to five state-of-the-art algorithms: 1) The neural network denoiser which is freely available in the OptiX 5.0 SDK. In this article we refer to it as the *OptiX Neural Network Denoiser* (ONND). 2) A recent state-of-the-art real-time Monte Carlo reconstruction algorithm, *Spatiotemporal Variance-Guided Filtering* (SVGF) [Schied et al. 2017]. 3) *Guided Filtering* (GF) [Bauszat et al. 2011], which we consider the algorithm-wise ancestor of the proposed work. 4) An off-line reconstruction algorithm called *Nonlinearly Weighted First-order Regression* (NFOR) [Bitterli et al. 2016]. 5) Another real-time reconstruction method, namely *An Efficient Denoising Algorithm for Global Illumination* (EDAGI) [Mara et al. 2017], which is separately compared in Subsection 7.3.

The ONND implementation is based on the interactive reconstruction from the article by Alla Chaitanya et al. [2017], but differs in a few ways. Most importantly, every frame is denoised individually, which causes low temporal stability. The OptiX implementation also does not separate albedo from the input before filtering. Moreover, it uses a different set of feature buffers than the original article. We attempted to use the filter with temporally accumulated noisy data similar to our method but found that with the default training set the filter is not able to discriminate between detail and noisy data due to changes in noise characteristics caused by accumulation. Consequently, we had to use a 1 spp input with this denoiser. Furthermore, ONND requires that the input is tone-mapped and gamma-encoded.

The authors of SVGF did not provide an implementation for accurately reproducing the results of their article. Therefore we used a freely available implementation of the algorithm in the quality assessments.[2] We modified the implementation to follow the original article's algorithm more closely by running it separately for direct and indirect lighting and by removing albedo before filtering. We also changed it to use the same TAA [Karis 2014] as in the SVGF article.

We used our own code for the Guided Filter implementation. Our implementation is based on the MATLAB code provided by the authors of the original article on guided filter [He et al. 2013] but has been extended to allow an arbitrary number of feature buffers. As in the article by Bauszat et al. [2011], we used a 4-dimensional guidance image consisting of three normal channels and depth. In the article, only indirect illumination is filtered. For the indirect component, we used radius 24 and epsilon 0.01 as suggested in the article. Because in our dataset also the direct illumination component is noisy, we filtered it as well with guided filter. We used a smaller filter size (radius 12) to cause less blurring, and therefore to improve the results. The epsilon used for direct illumination was the same as for indirect illumination. Finally, we extended the method with albedo removal and accumulation of noisy data.

For NFOR we used the freely avaible code released by the original authors [Bitterli et al. 2016]. For comparison fairness, instead of using 1 spp inputs, we used the reprojected and accumulated inputs and reprojected running variances, which improved the quality significantly. We also applied TAA to the results because it improved subjective quality in all test scenes and objective quality in approximately half of the tests scenes.

7 RESULTS

This section reports the performance of the algorithm in terms of the visual quality of the result and the execution speed with the test setup described in the previous section.

7.1 Objective Quality

We used four different metrics to measure the objective quality of our method compared to the other methods: *Root Mean Square Error* (RMSE), *Structural SIMilarity* (SSIM) [Wang et al. 2004], temporal error [Schied et al. 2017], and *Video Multi-Method Assessment Fusion* (VMAF)[3] [Aaron et al. 2015; Li et al. 2016]. The results of our measurements are presented in Table 2 and Table 3, and comparison images of all the methods are shown in Fig. 7. The known limitations of the proposed method are further discussed in Section 8.

As expected, the offline comparison method NFOR is able to obtain best results in most of the scenes with most of the metrics. However, the results of the proposed method are close to the NFOR results with more than ten thousand times faster runtime. NFOR is not originally designed for 1 spp inputs, but when we give it reprojected inputs, the effective spp count gets close to the counts used in the original paper.

In the majority of the test scenes, our method outperforms the previous real-time methods in terms of RMSE, SSIM and VMAF. In the remaining scenes our results are still generally comparable to the other real-time methods, with only marginal differences at the top. In the few cases where our results are average in terms of one metric, such as for RMSE in the moving light Sponza, another metric still ranks us at the top, in this case VMAF. Hence, in such cases the performance difference can be at least partially attributed to inherent limitations in the simple metrics, as they disagree with each other to some extent; therefore, we provide the results for several metrics. Moreover, our results could be improved if only optimizing these metrics by skipping TAA in phase III, since it introduces some blur in the results and thus affects RMSE, SSIM and VMAF negatively. Nevertheless, we chose to apply it due to it producing visually more pleasing results to our eyes.

In terms of temporal error [Schied et al. 2017], our results are overall similar to those of the guided filter and ONND. SVGF yields the lowest temporal error in all of the scenes, with our method being on par with it in the static scene. However, the used temporal error metric is rather simple, as it only considers the average per-pixel luminance differences between adjacent frames, so its correlation with subjectively perceived temporal quality variance is not immediately evident. This observation is further corroborated by the fact

[2]The SVGF version which was used as a base of our modifications can be found at https://github.com/ruba/RadeonProRender-Baikal, Git commit hash ed2a7e2d929653551f8a93ada5b164d2f9f624e7.

[3]The VMAF model used in the comparison was v0.6.1.

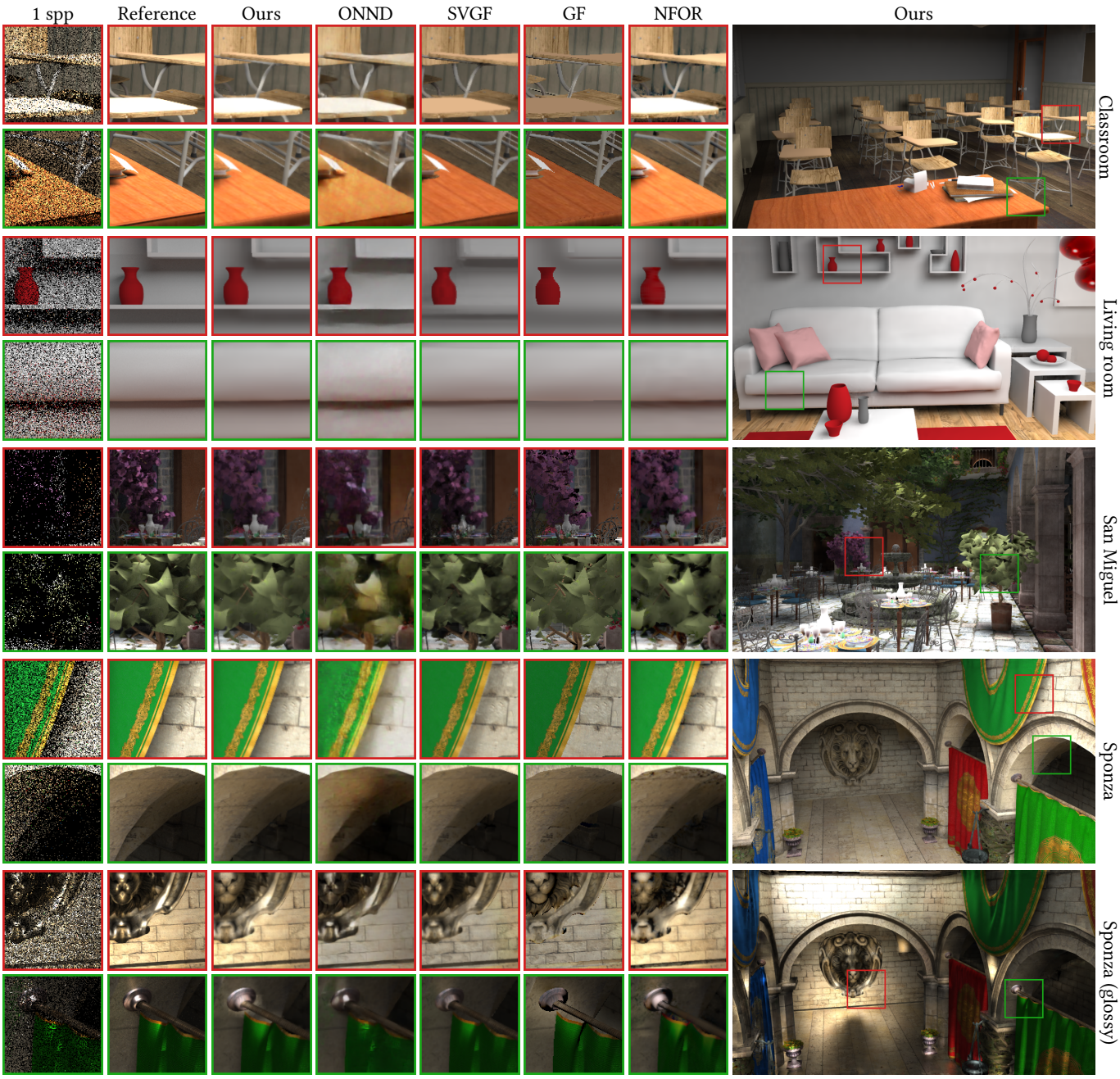


Fig. 7. Closeups highlighting the quality differences between the proposed pipeline and the comparison methods taken from animated sequences after 30 frames. Detailed description of the insets is in Subsection 7.2. Reference is 4096 spp and the comparison methods are *OptiX Neural Network Denoiser* (ONND) which is based on Alla Chaitanya et al. [2017], *Spatiotemporal Variance-Guided Filtering*, (SVGF) [Schied et al. 2017], *Guided Filtering* (GF) which is based on Bauszat et al. [2011], and *Nonlinearly Weighted First-order Regression* (NFOR) [Bitterli et al. 2016].

Table 2. Objective quality measurements for the results measured with RMSE (lower is better) and SSIM (higher is better), each averaged over all 60 frames. For brevity, Sponza with a static camera is referred to as "Sponza (static)", and Sponza with a moving light as "Sponza (light)".

	Average RMSE					Average SSIM				
	Proposed	ONND	SVGF	GF	NFOR	Proposed	ONND	SVGF	GF	NFOR
Classroom	0.0356	0.0431	0.0561	0.0586	**0.0321**	0.960	0.938	0.957	0.937	**0.961**
Living room	0.0315	0.0526	0.0434	0.0608	**0.0272**	0.953	0.927	0.936	0.918	**0.957**
San Miguel	0.0895	0.0982	0.1160	0.1011	**0.0812**	0.753	0.669	0.745	0.742	**0.757**
Sponza	**0.0282**	0.0591	0.0661	0.0509	0.0306	**0.965**	0.901	0.943	0.961	0.957
Sponza (glossy)	0.0564	0.0671	0.0900	0.0736	**0.0503**	**0.906**	0.847	0.893	0.885	0.899
Sponza (static camera)	**0.0317**	0.0756	0.1159	0.0939	0.0394	**0.973**	0.876	0.906	0.932	0.948
Sponza (moving light)	0.1450	**0.0773**	0.1418	0.0936	0.0811	0.835	0.846	0.855	**0.913**	0.892

Table 3. Objective quality measurements for the results obtained with various algorithms, measured with a simple temporal error as in [Schied et al. 2017], averaged over all 60 frames. The quality is also measured with VMAF (higher is better).

	Average temporal error						VMAF				
	Ref.	Proposed	ONND	SVGF	GF	NFOR	Proposed	ONND	SVGF	GF	NFOR
Classroom	0.043	0.037	0.040	0.035	0.039	0.040	53.14	52.78	41.51	24.90	**61.11**
Living room	0.024	0.021	0.027	0.019	0.019	0.022	57.85	61.53	46.78	21.58	**64.61**
San Miguel	0.082	0.060	0.061	0.056	0.073	0.067	21.68	21.63	10.77	25.73	**28.03**
Sponza	0.059	0.051	0.048	0.045	0.056	0.052	**74.81**	49.66	50.58	58.04	70.54
Sponza (glossy)	0.058	0.050	0.043	0.044	0.053	0.054	38.50	32.87	22.73	24.22	**44.87**
Sponza (static camera)	0.004	0.001	0.019	0.001	0.003	0.001	**70.46**	32.71	30.53	24.97	65.52
Sponza (moving light)	0.011	0.007	0.022	0.006	0.010	0.009	32.19	32.46	19.09	26.40	**49.24**

that, as Table 3 shows, the temporal error of the reference itself is typically higher than that of the reconstructed result. Hence, instead of merely focusing on the absolute error, it may also be useful to consider how close the error of the reconstructed result is to the error of the reference. However, similar temporal error readings can be caused by completely different changes in the consecutive frames. On the other hand, VMAF demonstrably correlates well with subjective quality [Li et al. 2016], and in most cases our method yields significantly higher VMAF results than the other real-time methods.

7.2 Subjective Quality

Subjective quality of the proposed method can be evaluated with Fig. 7. Moreover, all full resolution frames and a video are available in the supplementary material of this article.

In Fig. 7 the insets of the Living room and Classroom scenes represent cases where our algorithm is able to outperform the comparison methods. ONND sometimes starts to generate details that are not present in the reference at all. Due to its À-Trous nature, SVGF generates sometimes light artifacts that are typical to À-Trous based methods. These are visible for example in the red inset of the Living room scene. On the other hand, GF often overblurs the illumination, which might be due to poor parameter selection. We used the best parameters according to the original authors.

Insets of the San Miguel scene show different foliage cases. Our method produces results which are visually pleasing and believable, though somewhat overblurred.

One of the main motivations of our work is visible in the red insets of the Sponza scene. The proposed method can produce in real time dynamic soft shadows that are very close to the reference. The green insets of the same scene represent a case where there is just a small amount of light and our algorithm must rely on 1 spp data due to occlusions (camera is moving back and rightwards). In this case the result contains some blurred artifacts.

The roughness in the Sponza (glossy) scene is 0.1 for every material. As can be seen in the red insets of the Sponza (glossy) scene in Fig. 7, our algorithm can perform well with even quite complex specular highlights. On the other hand, the green insets of the same scene represent a hard case where all of the methods fail and it is up to the viewer to decide which type of imperfection is the least disturbing. More discussion on the limitations of the specular highlights can be found in Section 8.

7.3 Comparison to Noise-Free Direct Lighting

In this subsection we report a separate comparison with EDAGI [Mara et al. 2017]. This method is treated separately because it assumes a rasterized noise-free direct lighting component. Thus, it is incompatible with the stochastic noisy direct lighting in our input dataset, preventing an objective comparison to the fully path-traced reference like that in Tables 2 and 3.

Fig. 8 presents some of the test scenes from the original EDAGI work, as reconstructed by the proposed method from a fully stochastic path-traced lighting. When comparing these images to those in their online supplementary material, it is visible how realistic

Fig. 8. Some of the scenes from [Mara et al. 2017] reconstructed with the proposed work.

Table 4. Average execution times of different stages in the proposed pipeline on AMD Radeon Vega Frontier Edition. The division to OpenCL kernels is different than the stages in the Fig. 2. The division was chosen for performance and code readability reasons. The total runtime is measured from the beginning of the first kernel to the end of last kernel. All of the scenes and camera paths yield similar timings, because only the runtime of accumulation and TAA kernels is affected by the input data.

Phase	Kernel	Runtime
I	Temporal accumulation	0.44 ms
II	QR & back substitution	1.55 ms
	Weighted sum	0.12 ms
III	Temporal accumulation	0.23 ms
	TAA	0.16 ms
Total		2.54 ms

Table 5. GPU runtimes of different comparison methods for 720p frames as reported in the original articles. NVIDIA GeForce Titan X (Pascal) runtime was measured with the OpenCL implementation, which was developed on AMD Radeon Vega Frontier Edition. Consequently, there is likely room for optimizations on NVIDIA platforms.

Method	Runtime	Hardware
Proposed (OpenCL)	2.5 ms	AMD Radeon Vega FE
	2.4 ms	NVIDIA Titan X (Pascal)
[Alla Chaitanya et al. 2017]	55 ms	NVIDIA Titan X (Pascal)
SVGF [Schied et al. 2017]	4.4 ms	NVIDIA Titan X (Pascal)
GF [Bauszat et al. 2011]	94 ms	NVIDIA GTX 285
EDAGI [Mara et al. 2017]	9.2 ms	NVIDIA Titan X (Pascal)

soft shadows produced by the stochastic direct lighting make the proposed kind of rendering compelling.

7.4 Execution Speed

The average execution times of different parts of the proposed pipeline can be seen in Table 4. In the measurements we assumed that the path traced 1 spp input and feature buffers are in GPU buffers when we start the timer and that the result can be left to another GPU buffer. That is, we model a scenario where a GPU-based path tracer has left its data to GPU buffers and at the end, the results are written to the frame buffer.

All of the runtimes reported in this section are with 1280×720 frames. We also confirmed with measurements that, as analyzed in Section 4, the runtime scales linearly relative to the number of pixels.

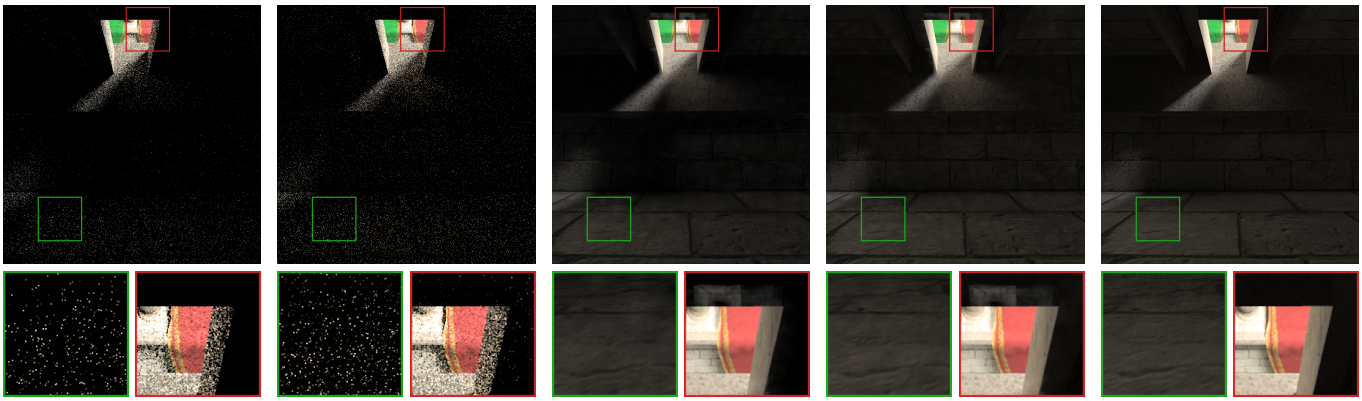
The execution time of the proposed pipeline was stable on AMD Vega Frontier Edition (variation approximately ± 0.04 ms) across different scenes and animation frames. The only pipeline stages where runtimes are affected by the input data are the ones with temporal accumulation. The runtime variation is due to cache misses of dispersed loads and early exits, e.g., in case of projected pixels that are detected to fall outside the new frame.

The proposed pipeline clearly outperforms the other algorithms (listed in Table 5) in terms of execution speed. NFOR runtime is left out from the table because it is in order of minutes rather than milliseconds. SVGF, the previous state-of-the-art real-time method, reports average execution times of 4.4 ms on NVIDIA Titan X (Pascal). Our 2.4 ms execution time is thus 1.8× faster. Moreover, SVGF's execution time depends more on the input data because they fall back to a slower method with harder inputs. The other real-time method [Mara et al. 2017] has an average execution time of 9 ms on NVIDIA Titan X (Pascal). However, they expect noise-free direct lighting which makes the comparison difficult. Alla Chaitanya et al. [2017] report runtimes of 55 ms on NVIDIA Titan X (Pascal), which means the proposed pipeline is 22× faster. Guided filter [Bauszat et al. 2011] execution time linearly scaled to 720p frame is 94 ms on NVIDIA 285 GTX, and even for that number, noise-free direct lighting is required. However, the article where the number was reported is already a bit old and uses a previous generation GPU, thus there could be room for improvement if the algorithm was optimized for a modern GPU.

8 LIMITATIONS

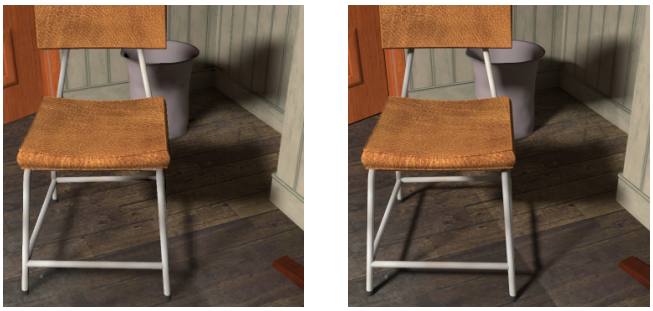
We have observed three different categories of imperfections in the results of the proposed method, which we plan to address in our future work:

1) Because of the fixed sizes of the blocks, the algorithm can sometimes have difficulty constructing illumination that is not visible in the feature data and is smaller than the block size. Example of small soft shadows can be seen in Fig. 10. Another example of this is specular highlights. High values in a small area are typically blurred as can be seen in the last row of Fig. 7. However, different order versions of the feature buffers and block place variation reduces the problem significantly. Moreover, the quality can be improved by using feature buffers containing noise-free data related to the cause of the problematic illumination. The effect of adding this kind of a buffer can be seen in Fig. 12.



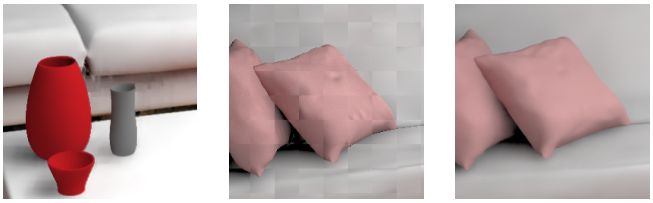
(a) 1-bounce 1 spp (b) 9-bounces 1 spp (c) 1-bounce 1 spp BMFR (d) 9-bounce 1 spp BMFR (e) 9-bounce reference

Fig. 9. An example of how the proposed method handles inputs which have more than one bounce. In this mostly indirect illumination case the 1-bounce 1 spp BMFR is slightly too dark in the green inset since it is very unlikely that the only secondary ray finds its way out from the opening. The 9 bounce 1 spp BMFR is already close to the reference. However, in the red inset the fireflies on the dark wall next to the opening cause more brightness to bleed to the wrong side of the corner. In these figures the gamma correction was modified so that the problems standout more clearly.



(a) Proposed algorithm (b) Reference

Fig. 10. If a shadow smaller than the block size is not represented in the feature buffers, the resulting shadow can be too soft. However, bigger shadows like the contact shadow of the trash can follow the reference quite closely.

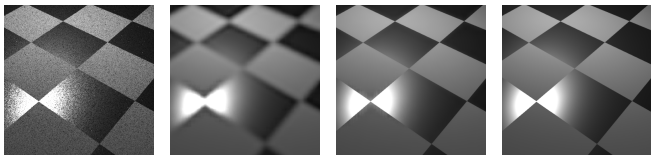


(a) Artifacts can be seen on the occluded areas. This is the worst case since the camera is moving to the right with a high speed.

(b) First frame without any accumulated data (effective 1 spp input) shows the blockwise nature of the algorithm.

(c) Accumulation and block place variation removes the blocky look from the same pillows as in Fig. 11b

Fig. 11. Different artifacts from the proposed pipeline. The lack of detailed texturing in the scene makes the artifacts stand out more than usual.



(a) 1 spp (b) The proposed BMFR (c) BMFR with material ID feature buffer (d) Reference

Fig. 12. In this example the only difference in the flat surface is its roughness. Also, albedo is constant for the whole surface but the black background makes the smoother surface seem darker. The only feature data which is not constant are the two world position axes and BMFR has to construct the final image from them. In Fig. 12c we add the material ID feature buffer, which allows BMFR to differentiate between the two materials and, therefore, improves the results significantly.

2) The proposed algorithm is affected by the same problems as the previous works that use reprojected temporal data. Since the reprojection is done to the first bounce intersection world-space position, e.g., reflections and specular highlights get overblurred. However, if the material is a completely reflecting mirror, the problem can be fixed by using a virtual world-space position, but if there are both a reflecting and a non-reflecting component in the material, we would have to store and reproject those separately [Zimmer et al. 2015]. Occlusions with the reprojected data also cause the input to have different amounts of effective spp in different screen space areas. Different effective spp causes the quality of the output of our algorithm to be decreased in the occluded areas as can be seen in Fig. 11a. The visibility of these artifacts on a still frame does not correspond to their visibility on a moving scene, due to how perception works. The artifacts are stronger in case of fast camera movement causing larger disocclusions, but those cases are also the ones where

(a) Frame 15 of the proposed algorithm. (b) Frame 15 of the reference. (c) Frame 25 of the proposed algorithm.

Fig. 13. A scene with a light moving towards the camera shows the temporal lag caused by the temporal accumulation. With the proposed temporal accumulation parameters, it takes approximately 10 frames for the proposed method to produce the most similar lighting.

artifacts get harder to be noticed by the user's perception [Reibman and Poole 2007].

Reprojected temporal data also causes lag in the lighting changes caused by animations. Fig. 13 shows a scene with a moving light. With the proposed setup (where 20% is from the newest path tracing samples and 10% is from the newest fitted frame), it takes approximately 10 frames for the image to converge to a similar appearance as the reference. However, in a real use case where the 4096 spp reference is not available, the lag is hard to notice since 10 frames is not a long time with the frame rates the proposed algorithm is able to generate. One solution to the temporal lag problem was provided in a concurrent work [Schied et al. 2018]. However, the same algorithm cannot be directly used with the proposed work because it would generate blocking artifacts to areas where illumination changes drastically.

3) The blockwise nature of the algorithm causes blocking artifacts visible in Fig. 11b. These can be seen on the first frame when there is no accumulated data in the input and no block displacement in BMFR. On the first frame, the problem could be fixed by running the fitting phase (II) twice with two different grid locations and smoothly blending the overlapping pixels from one block to another. Moreover, during the first frames in a completely new camera location it is hard for the human visual system to perceive artifacts [Reibman and Poole 2007]. However, the issue can be adequately resolved by using a fade-in effect over a few frames when the camera is "teleported" to a completely new location.

We have also tested the proposed method with more than one bounce of path tracing. An example of this is shown in Fig. 9. The only limiting factor is that the radiance of fireflies is only propagated within a single block area, which is defined in screen space. This limitation is not visible in typical scenes, but it can be a problem in dedicated test scenes where a path to the light is very unlikely to be found. However, temporal accumulation after the fitting phase robustly removes temporal artifacts caused by the fireflies. If the fireflies are very rare and there is a need for some illumination in the results, it might be possible to use path space regularization techniques [Kaplanyan and Dachsbacher 2013].

During prototyping the algorithm, we noticed that using multiple iterations of BMFR with multiple orders of features, different block locations, and different block sizes on each iteration, can reduce the artifacts discussed in this section. However, having a single iteration with fixed-sized blocks was best suited for our real-time implementation. Akin to multivariate monomials, the extended set of feature buffers in Eq. 1 may also include generic products of the form $F_{n_j}^{\gamma_j} F_{n_k}^{\gamma_k}$, however this opportunity has not been investigated for this work.

One more limitation of our algorithm is that noise in the feature buffers, due to, e.g., motion blur or depth of field, is visible in the results. These kinds of effects would require denoising the feature buffers first. However, in both examples we can compute how much the data in the feature buffer should be blurred to follow the physical phenomenon. We plan to address the problem of noisy feature buffers in future work.

9 CONCLUSIONS

In this article, we introduced *Blockwise Multi-Order Feature Regression* (BMFR). In BMFR, different powers of the feature buffers are used for blockwise regression in path-tracing reconstruction. We show that a real time GPU-based implementation of BMFR is possible; the evaluated example implementation processes a 720p frame in 2.4 ms on a modern GPU, making it 1.8× faster than the previous state-of-the-art real-time path tracing reconstruction algorithm with better quality in almost all the used metrics. The code and the data to reproduce our results is available in the supplementary material of this article.

The high execution speed of the proposed algorithm is achieved by augmented QR factorization and the use of stochastic regularization, which addresses rank-deficiencies and avoids numerical instabilities without the extra complexity of pivoting. Like in previous work, our algorithm relies on reprojecting and accumulating previous frames, which increases the effective samples-per-pixelcount in our input. Instead of using exponential moving average for the data accumulation all the time, on the first frames and after an occlusion we use a cumulative moving average of the samples. Cumulative moving average does not give an excessive weight to the very first samples and, therefore, reduces artifacts. In our algorithm we use similar accumulation also after the regression to increase the temporal stability and to decrease the amount of artifacts in the results.

ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers, Petrus Kivi, and Atro Lotvonen for their fruitful comments. We are also grateful to Aleksandr Diment for letting us use his workstation for the NVIDIA Titan X (Pascal) measurements, Dmytro Rubalskyi for his open-source SVGF implementation used in the comparison, and to the model makers: Frank Meinl for Sponza (CC BY 3.0, [McGuire 2017]), Guillermo M. Leal Llaguno for San Miguel (CC BY 3.0, [McGuire 2017]), Christophe Seux for Classroom (CC0), and Wig42 for Living room (CC BY 3.0, [Bitterli 2016]). Finally, this work was supported by the funding from TUT Graduate School, Emil Aaltonen Foundation, Finnish Foundation for Technology Promotion, Nokia Foundation, Business Finland (funding decision 40142/14, FiDiPro-StreamPro), Academy of Finland (funding decisions 297548, 310411) and ECSEL JU project FitOptiVis (project number 783162).

REFERENCES

Anne Aaron, Zhi Li, Megha Manohara, Joe Yuchieh Lin, Eddy Chi-Hao Wu, and C.-C Jay Kuo. 2015. Challenges in Cloud Based Ingest and Encoding for High Quality Streaming Media. In *Proceedings of the Image Processing*.

Timo Aila and Tero Karras. 2010. Architecture Considerations for Tracing Incoherent Rays. In *Proceedings of the High Performance Graphics*.

Chakravarty Alla Chaitanya, Anton Kaplanyan, Christoph Schied, Marco Salvi, Aaron Lefohn, Derek Nowrouzezahrai, and Timo Aila. 2017. Interactive Reconstruction of Monte Carlo Image Sequences Using a Recurrent Denoising Autoencoder. *Transactions on Graphics* 36, 4 (2017).

AMD. 2017. RadeonRays SDK. Online. (2017). Available: https://github.com/GPUOpen-LibrariesAndSDKs/RadeonRays_SDK, Referenced: January 23 2018.

Steve Bako, Thijs Vogels, Brian Mcwilliams, Mark Meyer, Jan NováK, Alex Harvill, Pradeep Sen, Tony Derose, and Fabrice Rousselle. 2017. Kernel-predicting convolutional networks for denoising Monte Carlo renderings. *Transactions on Graphics* 36, 4 (2017).

Pablo Bauszat, Martin Eisemann, and Marcus Magnor. 2011. Guided Image Filtering for Interactive High-quality Global Illumination. *Computer Graphics Forum* 30, 4 (2011).

Benedikt Bitterli. 2016. Rendering resources. (2016). https://benedikt-bitterli.me/resources/.

Benedikt Bitterli, Fabrice Rousselle, Bochang Moon, José A Iglesias-Guitián, David Adler, Kenny Mitchell, Wojciech Jarosz, and Jan Novák. 2016. Nonlinearly Weighted First-order Regression for Denoising Monte Carlo Renderings. *Computer Graphics Forum* 35, 4 (2016).

Peter Burt. 1981. Fast Filter Transform for Image Processing. *Computer Graphics and Image Processing* 16, 1 (1981).

Holger Dammertz, Daniel Sewtz, Johannes Hanika, and Hendrik Lensch. 2010. Edge-avoiding À-Trous Wavelet Transform for Fast Global Illumination Filtering. In *Proceedings of the High Performance Graphics*.

Kevin Egan, Yu-Ting Tseng, Nicolas Holzschuch, Frédo Durand, and Ravi Ramamoorthi. 2009. Frequency analysis and sheared reconstruction for rendering motion blur. *Transactions on Graphics* 28, 3 (2009), 93.

Luke Goddard. 2014. Silencing the Noise on Elysium. In *ACM SIGGRAPH 2014 Talks*.

Kaiming He, Jian Sun, and Xiaoou Tang. 2013. Guided Image Filtering. *Transactions on Pattern Analysis and Machine Intelligence* 35, 6 (2013).

Michael Heath. 1997. *Scientific Computing*. McGraw-Hill.

Jorge Jimenez, Jose I. Echevarria, Tiago Sousa, and Diego Gutierrez. 2012. SMAA: Enhanced Morphological Antialiasing. *Computer Graphics Forum (Proc. EUROGRAPHICS 2012)* 31, 2 (2012).

Jorge Jiménez, X Wu, A Pesce, and A Jarabo. 2016. Practical real-time strategies for accurate indirect occlusion. *SIGGRAPH 2016 Courses: Physically Based Shading in Theory and Practice* (2016).

Nima Khademi Kalantari, Steve Bako, and Pradeep Sen. 2015. A Machine Learning Approach for Filtering Monte Carlo Noise. *Transactions on Graphics* 34, 4 (2015).

Anton Kaplanyan and Carsten Dachsbacher. 2013. Path space regularization for holistic and robust light transport. *Computer Graphics Forum* 32, 2pt1 (2013).

Brian Karis. 2014. High-quality Temporal Supersampling. In *ACM SIGGRAPH 2014, Advances in Real-Time Rendering in Games*.

Samuli Laine, Tero Karras, and Timo Aila. 2013. Megakernels Considered Harmful: Wavefront Path Tracing on GPUs. In *Proceedings of the High Performance Graphics*.

Tzu-Mao Li, Yu-Ting Wu, and Yung-Yu Chuang. 2012. SURE-based Optimization for Adaptive Sampling and Reconstruction. *Transactions on Graphics* 31, 6 (2012).

Zhi Li, Anne Aaron, Ioannis Katsavounidis, Anush Moorthy, and Megha Manohara. 2016. Toward a Practical Perceptual Video Quality Metric. Online. (2016). Available: https://medium.com/netflix-techblog/toward-a-practical-perceptual-video-quality-metric-653f208b9652, Referenced: January 23 2018.

Yu Liu, Changwen Zheng, Quan Zheng, and Hongliang Yuan. 2017. Removing Monte Carlo Noise Using a Sobel Operator and a Guided Image Filter. *The Visual Computer* 34, 4 (2017).

Michael Mara, Morgan McGuire, Benedikt Bitterli, and Wojciech Jarosz. 2017. An Efficient Denoising Algorithm for Global Illumination. In *Proceedings of the High Performance Graphics*. http://casual-effects.com/research/Mara2017Denoise/

Morgan McGuire. 2017. Computer Graphics Archive. (2017). https://casual-effects.com/data.

Bochang Moon, Nathan Carr, and Sung-Eui Yoon. 2014. Adaptive Rendering Based on Weighted Local Regression. *Transactions on Graphics* 33, 5 (2014).

Bochang Moon, Jose A Iglesias-Guitian, Sung-Eui Yoon, and Kenny Mitchell. 2015. Adaptive Rendering with Linear Predictions. *Transactions on Graphics* 34, 4 (2015).

Bochang Moon, Steven McDonagh, Kenny Mitchell, and Markus Gross. 2016. Adaptive Polynomial Rendering. *Transactions on Graphics* 35, 4 (2016).

Steven G Parker, James Bigler, Andreas Dietrich, Heiko Friedrich, Jared Hoberock, David Luebke, David McAllister, Morgan McGuire, Keith Morley, Austin Robison, et al. 2010. Optix: a General Purpose Ray Tracing Engine. *Transactions on Graphics* 29, 4 (2010).

Amar Patel. 2018. *D3D12 Raytracing Functional Spec, v0.09*. Microsoft. Available: http://forums.directxtech.com/index.php?topic=5860.0, Referenced: March 23 2018.

Matt Pharr and Greg Humphreys. 2010. *Physically Based Rendering: From Theory to Implementation* (2nd ed.). Morgan Kaufmann.

Amy R Reibman and David Poole. 2007. Predicting packet-loss visibility using scene characteristics. In *Proceedings of the Packet Video*.

Gilberto Rosado. 2007. *GPU gems 3*. Addison-Wesley Professional, Chapter 27. Motion Blur as a Post-Processing Effect.

Christoph Schied, Anton Kaplanyan, Chris Wyman, Anjul Patney, Chakravarty R Alla Chaitanya, John Burgess, Shiqiu Liu, Carsten Dachsbacher, Aaron Lefohn, and Marco Salvi. 2017. Spatiotemporal Variance-guided Filtering: Real-time Reconstruction for Path-traced Global Illumination. In *Proceedings of the High Performance Graphics*.

Christoph Schied, Christoph Peters, and Carsten Dachsbacher. 2018. Gradient Estimation for Real-Time Adaptive Temporal Filtering. *Proceedings of the ACM on Computer Graphics and Interactive Techniques* 1, 2 (2018), 24.

Carlo Tomasi and Roberto Manduchi. 1998. Bilateral Filtering for Gray and Color Images. In *Proceedings of the Computer Vision*.

Eric Veach and Leonidas J Guibas. 1995. Optimally combining sampling techniques for Monte Carlo rendering. In *Proceedings of the Computer graphics and interactive techniques*.

Timo Viitanen, Matias Koskela, Kalle Immonen, Markku Mäkitalo, Pekka Jääskeläinen, and Jarmo Takala. 2018. Sparse Sampling for Real-time Ray Tracing. In *Proceedings of the GRAPP*.

Ingo Wald, Sven Woop, Carsten Benthin, Gregory S. Johnson, and Manfred Ernst. 2014. Embree: A Kernel Framework for Efficient CPU Ray Tracing. *Transactions on Graphics* 33, 4 (2014).

Yong Wang, Xiaofeng Liao, Di Xiao, and Kwok-Wo Wong. 2008. One-way hash function construction based on 2D coupled map lattices. *Information Sciences* 178, 5 (2008).

Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. 2004. Image Quality Assessment: from Error Visibility to Structural Similarity. *Transactions on Image Processing* 13, 4 (2004).

Ling-Qi Yan, Soham Uday Mehta, Ravi Ramamoorthi, and Fredo Durand. 2015. Fast 4D sheared filtering for interactive rendering of distribution effects. *Transactions on Graphics* 35, 1 (2015), 7.

Lei Yang, Diego Nehab, Pedro V Sander, Pitchaya Sitthi-amorn, Jason Lawrence, and Hugues Hoppe. 2009. Amortized Supersampling. *Transactions on Graphics* 28, 5 (2009).

Henning Zimmer, Fabrice Rousselle, Wenzel Jakob, Oliver Wang, David Adler, Wojciech Jarosz, Olga Sorkine-Hornung, and Alexander Sorkine-Hornung. 2015. Path-space Motion Estimation and Decomposition for Robust Animation Filtering. 34, 4 (2015).

Matthias Zwicker, Wojciech Jarosz, Jaakko Lehtinen, Bochang Moon, Ravi Ramamoorthi, Fabrice Rousselle, Pradeep Sen, Cyril Soler, and S-E Yoon. 2015. Recent Advances in Adaptive Sampling and Reconstruction for Monte Carlo Rendering. *Computer Graphics Forum* 34, 2 (2015).

PUBLICATION 3

[P3]

Foveated Instant Preview for Progressive Rendering

M. Koskela, K. Immonen, T. Viitanen, P. Jääskeläinen, J. Multanen and J. Takala

SIGGRAPH Asia Technical Briefs. Ed. by D. Gutierrez. 2017
DOI: 10.1145/3145749.3149423

Foveated Instant Preview for Progressive Rendering

Matias Koskela
Tampere University of Technology

Kalle Immonen
Tampere University of Technology

Timo Viitanen
Tampere University of Technology

Pekka Jääskeläinen
Tampere University of Technology

Joonas Multanen
Tampere University of Technology

Jarmo Takala
Tampere University of Technology

ABSTRACT

Progressive rendering, for example Monte Carlo rendering of 360° content for virtual reality headsets, is a time-consuming task. If the 3D artist notices an error while previewing the rendering, he or she must return to editing mode, do the required changes, and restart rendering. Restart is required because the rendering system cannot know which pixels are affected by the change. We propose the use of eye-tracking-based optimization to significantly speed up previewing the artist's points of interest. Moreover, we derive an optimized version of the visual acuity model, which follows the original model more accurately than previous work. The proposed optimization was tested with a comprehensive user study. The participants felt that preview with the proposed method converged instantly, and the recorded split times show that the preview is 10 times faster than conventional preview. In addition, the system does not have measurable drawbacks on computational performance.

CCS CONCEPTS

• **Computing methodologies** → **Rendering**; *Graphics systems and interfaces*; Ray tracing;

KEYWORDS

Progressive rendering, Preview, Foveated rendering, Eye tracking

ACM Reference Format:
Matias Koskela, Kalle Immonen, Timo Viitanen, Pekka Jääskeläinen, Joonas Multanen, and Jarmo Takala. 2017. Foveated Instant Preview for Progressive Rendering. In *SA '17 Technical Briefs: SIGGRAPH Asia 2017 Technical Briefs, November 27–30, 2017, Bangkok, Thailand*. ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3145749.3149423

1 INTRODUCTION

Virtual Reality (VR) devices are getting more and more common for both work and entertainment applications. However, one of the challenges of VR is how to easily generate 360° content, because of its high resolution, and the requirement of having meaningful interesting content in every direction. Rendering high resolution images with realistic-looking progressive rendering methods typically takes hours to complete. Noisy images of the rendering are produced quickly. However, for example in Monte Carlo rendering, reducing the error of the estimator to half at any point requires the number of samples to be quadrupled [Pharr and Humphreys 2010].

If the artist notices during the preview that something was wrong in the scene, he or she must cancel the rendering, make the required changes, and start the rendering all over again. Restart is required because the system cannot know what pixels are affected by the change. Typically, the artist can create a rough estimate of the scene with a faster rendering method, but the error-free version becomes visible only after the slow progressive rendering process, especially if the scene has reflections, transparency, or soft shadows. If the artist can preview the rendering faster, it directly transfers to the speed of the whole content generation process. Compared to conventional rendering, the high resolution of 360° content makes the preview even slower, which makes its optimization more important.

In this paper, we propose a method for optimizing preview of progressive rendering by applying *foveated rendering*, i.e., reducing the quality of rendering in the peripheral regions of vision. Quality

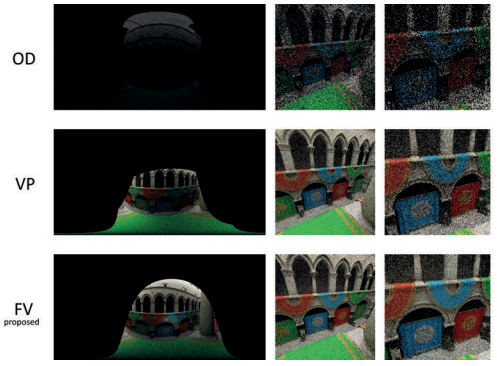


Figure 1: Results after two seconds of rendering with a static point of interest from left to right: rendering buffer, preview, and magnification of the point of interest. For the differences between the methods, see Sec. 4. Note how the point of interest already starts to converge in FV, but the edges of the preview have more noise than VP.

The work has been financially supported by the TUT Graduate School, Nokia Foundation, Emil Aaltonen Foundation, Finnish Foundation for Technology Promotion, Academy of Finland (funding decision 297548), Finnish Funding Agency for Technology and Innovation (funding decision 40142/14, FiDiPro-StreamPro) and ARTEMIS joint undertaking under grant agreement no 621439 (ALMARVI) .

SA '17 Technical Briefs, November 27–30, 2017, Bangkok, Thailand

ACM ISBN 978-1-4503-5406-6/17/11…$15.00
https://doi.org/10.1145/3145749.3149423

can be reduced because the human visual system cannot detect fine detail outside the center of the vision. Moreover, it has been predicted that more than 90% of real-time path tracing samples can be omitted by employing foveated rendering [Koskela et al. 2016]. This paper's novel contributions are:

(1) We derive an optimized version of the human visual acuity model, which can be followed to accurately generate path tracing samples.
(2) We propose the use of foveated rendering to speed up preview of progressive rendering, and validate with a user study that the proposed method is 10 times faster than conventional preview.

2 RELATED WORK

There is a large body of work on real-time foveated rendering, which is summarized by a recent comprehensive literature review [Weier et al. 2017]. Foveated rendering is very appealing with *Head Mounted Displays* (HMD), which typically have a wider field of view than desktop monitors, and only a single observer per display [Shibata 2002].

Current hardware supports only a fixed, predefined resolution for rasterization. Therefore, foveated rendering can be implemented more easily with ray-tracing-based techniques, because they support arbitrary sampling patterns in screen space. Consequently, foveated ray tracing has gained academic interest [Murphy et al. 2009; Weier et al. 2016]. An intuitive idea would be to distribute samples according to a model of the human visual acuity's smallest detectable spatial frequency:

$$m(e) = \begin{cases} 1.0, & e \leq 5.79 \\ \frac{7.49}{(0.3e+1)^2}, & e > 5.79 \end{cases}, \tag{1}$$

where $e \geq 0$ and it is the eccentricity angle, i.e., the angle from the gaze direction [Reddy 2001]. This model is derived from various psychophysical studies. The equation describes just one radius of the visual acuity, and the actual 2D model is obtained by taking a solid of revolution of the equation.

Due to the complexity of the visual acuity model, linear denominator models can be used instead of the quadratic denominator model shown in Eq. 1. However, they are not as accurate on the peripheral parts of the vision [Guenter et al. 2012]. A further simplified version is to use a linear fall-off between full detail and the minimum sampling probability [Stengel et al. 2016; Vaidyanathan et al. 2014; Weier et al. 2016], or even a static probability with respect to eye tracking [Pohl et al. 2016].

The context of previewing is closely related to real-time rendering, since the preview needs to be updated in real-time. Moreover, preview of a region of interest needs to converge as quickly as possible, because then the artist can cancel the rendering earlier, and make the required changes faster. One way to vary the convergence rates is to apply so-called *guided preview*, and have more samples in an area chosen by the artist with a pointing device [Roth et al. 2015]. Another idea is to select an area of the image where the sample computation is concentrated [Pixar 2017]. Importance masking [LuxRender 2013] is an advanced version of area selection.

In this paper, we utilize the idea of guided preview, and use one of the most intuitive ways for guiding, i.e., the point where the user is looking at. Moreover, we use the quadratic denominator visual acuity model instead of the significant simplifications of it.

3 PROPOSED METHOD

The idea of our preview method is to render images for VR and to give the artist an instant preview. The method tracks the eye of the user and generates more samples around the gaze direction. Sampling according to the visual acuity model does not decrease the user experience of previewing, because resolution can be reduced significantly on the peripheral parts of vision without affecting search task performance [Duchowski et al. 2009].

Sampling the world according to the visual acuity model requires random positions to be generated with probability density equal to Eq. 1. Note that the equation from Reddy [2001] is not consistent with the definition of the probability density function because its integral over the entire space is not equal to one. However, we will fix the equation so that it follows the definition in Eq. 5.

As we show below, generating random numbers according to the solid of revolution of Eq. 1 would have been too complicated for the targeted real-time preview method. Therefore, we simplify the generation by producing polar coordinates: one uniformly distributed for the angular coordinate ϕ, and another for the radial coordinate r, which is the distance from the center of the vision. To achieve correct distribution for r, the probability density of Eq. 1 must be slightly modified based on the circumference of circle $2\pi R$ (where R is the radius):

$$g(e) = 2\pi e\, m(e) = \begin{cases} 2\pi e, & e \leq 5.79 \\ \frac{14.98\pi e}{(0.3e+1)^2}, & e > 5.79 \end{cases}. \tag{2}$$

The angle can be generated by one of the many algorithms available for quickly generating uniformly distributed random numbers. In addition, uniform distribution can be transformed to any other distribution with the so-called inversion method [Devroye 1986]:

$$r = f^{-1}(u), \tag{3}$$

where u is a uniformly distributed random number in interval $[0, 1]$, f is the desired cumulative distribution function, and r is a random number that has a cumulative distribution f. The inversion method requires us to derive the cumulative distribution function from the probability density defined in Eq. 2 by taking the integral of $g(e)$ in interval $[0, x]$:

$$h(x) = \int_0^x g(e)\, de = \begin{cases} \pi x^2, & x \leq 5.79 \\ \frac{14.98}{0.09}\pi\big(\frac{1}{0.3x+1} + \ln(0.3x+1)\big) - 612.256, & x > 5.79 \end{cases}. \tag{4}$$

We chose the upper limit of the function at 80°, because the model starts to reach zero around 80°. Finally, the integral needs to be modified to be consistent with the cumulative distribution function definition that runs from 0 to 1 in y-axis:

$$f(x) = \frac{h(x)}{G(80)}, \tag{5}$$

where $G(e) = \int g(e)\, de$.

Eq. 3 requires the inverse of f in Eq. 5. However, it cannot be expressed in terms of standard mathematical functions and Lambert W-function [Weisstein 2002] would be needed. We simplified the

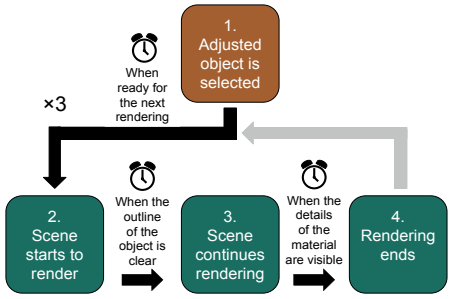


Figure 2: Illustration of the single scene procedure in the user study. Boxes are the states and arrows are the participant's actions for transitioning to other states. Clocks represent points where the system saved split times.

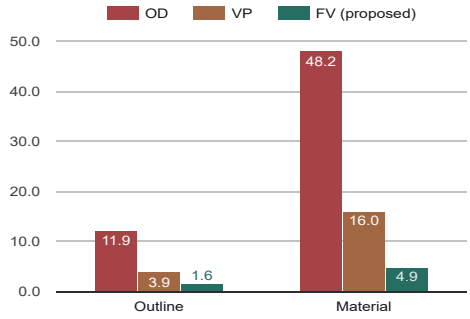


Figure 3: Geometric mean split times (less is better) over all 5 test scenes of the *visible outline* and *visible material* criteria for each of the three preview methods.

function by approximating it with a fitted fourth-order polynomial that was determined numerically by least squares regression:

$$f^{-1}(u) \approx 80 \times \begin{cases} 0.2330\sqrt{u}, & u \leq 0.0965 \\ 0.3136u^4 + 0.0021u^3 + \\ \quad 0.3451u^2 + 0.2984u + 0.0404, & u > 0.0965 \end{cases}. \quad (6)$$

The maximum error of Eq. 6 to Eq. 1 is 1.8% and integral of their difference is less than 0.04%, which are very small especially if compared to approximations from previous work.

In the proposed method the users preview the results with a VR HMD that has eye tracking capability, but also a desktop setup could be used. We chose the HMD because a VR headset gives better spatial awareness and enjoyment [MacQuarrie and Steed 2017] and, therefore, it is likely that the artist wants to preview the scene with a device similar to what the end users will use.

4 USER STUDY

To measure the subjective performance of our proposed instant preview method, we conducted a user study. The study used five different scenes and three different preview methods in random order. We chose the scenes to represent different 360° rendering scenarios. The preview methods were:

- *Omnidirectional preview* (OD): In this method samples were distributed uniformly to every possible point in an equirectangular image. This method represents conventional baseline rendering without any preview optimizations.
- *Viewport preview* (VP): This method distributed samples uniformly to the area currently viewable with the HMD. The idea of this method was to simulate sampling similar to rectangle area selection tool used in some rendering engines.
- *Foveated preview* (FV): This is the proposed method which distributed samples according to the visual acuity model centered on the gaze point of the eye-tracked user.

Procedure for each 3D scene can be seen in Fig. 2. In every scene, we asked the participants to play the role of a 3D artist, and to choose an object in the 3D world. They were told that the object represents an object that they have just adjusted. Adjustment could have been, for example, changing the orientation of the object or changing its material parameters.

After the selection, the rendering started, and the participants recorded split times. The first split was recorded at the point where the participants thought that they could determine if transform or rotation of the object was successful. The second split represented the time when the participant was able to determine if the material adjustment was successful. The idea was that at these points the artist could cancel the rendering and go back to editing mode.

We chose unidirectional path tracing with importance sampling as the progressive rendering method. AMD RadeonRays [AMD 2017] was used for ray traversal and the path tracer ran on an AMD Fury X GPU. The FOVE 0 VR headset was used as a viewing device in the study due to its eye tracking capability. The system generated equirectangular images and the previewing used trilinear filtering to cancel flickering near poles.

5 RESULTS

We conducted the user study with 16 participants, of whom 11 were male and 5 were female. The ages of the participants varied between 22 and 37. Two of the participants knew details about the test set-up beforehand, but their results were close to the average of the other results.

The geometric means of all timings are shown in Fig. 3 and arithmetic means of each scene over all participants are listed in Table 1. The results show that the proposed method required only around 10% time compared to the baseline method of OD. The time saving translates directly to the speed of the artist's feedback loop, since he or she can quit the rendering and start making the required changes 10 times faster. Equivalent comparison states that previewing with VP takes around 30% time compared to OD.

In an open discussion after the test, many participants reported that FV was so fast that it was hard to record the first split at the right time. They also stated that it felt that the FV method converged instantly. On the other hand, several participants stated that slowness of OD might have caused them to get bored, inducing them to mark a split time at a lower quality than with the other

Table 1: Arithmetic mean (μ) split times and their standard deviations (σ) from the user study for each scene. The results of the FV (proposed) and VP are compared to the OD and pp stands for percent point. Big σ values in OD are caused by the participants selecting different kind of objects.

Split type	Outline visible						Material visible					
Preview method	OD		VP		FV		OD		VP		FV	
Value type	μ (s)	σ (s)	μ (%)	σ (pp)	μ (%)	σ (pp)	μ (s)	σ (s)	μ (%)	σ (pp)	μ (%)	σ (pp)
BMW	6.1	7.6	41.8	9.8	20.8	1.9	29.9	16.9	33.6	15.1	11.8	5.3
Classroom	15.6	7.7	27.1	3.1	11.6	2.1	67.9	79.2	30.8	35.7	7.9	4.6
Conference	41.9	59.7	29.7	8.6	7.6	2.3	137.2	197.4	33.9	46.0	8.2	8.6
Sibenik	24.7	20.8	29.5	9.2	11.4	3.9	76.7	55.2	34.3	29.0	10.9	10.1
Sponza	16.1	13.8	25.9	4.9	9.1	2.1	60.4	46.8	30.1	18.9	7.5	4.0

methods. Most of the participants also stated that they did not realize that eye tracking was used, and instead thought that the actual rendering was somehow faster. Not noticing the eye tracking means that the visual acuity model is a good way to distribute the samples.

Measurement of the computational performance of the three different methods states that they are computationally equally good. On the target machine, according to AMD CodeXL, it takes around 0.17 ms to launch 65,536 primary rays with all of the preview methods. The launch includes generating random pixel coordinates for the rays and calculating the ray origin and direction based on them. In the case of the proposed FV extra work is done to change the random number distribution with the inversion method (Eq. 6). The timing implies that the extra work is hidden by the latencies of memory accesses and the kernel launch.

The ray tracing performance is dependent on the user's gaze or head direction with the FV and VP methods, respectively. In contrast, OD has the same ray tracing performance no matter where the user is looking at. While OD yields a larger number of samples per second than the other methods, this result can be misleading because many of the rays are sent to directions that are easier to ray trace, e.g., straight to a skybox.

6 CONCLUSIONS

In this paper we presented a foveation-based preview system for progressive rendering. The system tracks the user's gaze and distributes samples according to a visual acuity model.

Thanks to our optimized visual acuity model, the image converges at the user's point of interest 10 times faster than with conventional uniform sampling over the whole 360° image area. Quick convergence enables the 3D artist to cancel the rendering 10 times earlier, reducing the length of the feedback loop significantly. We recorded these timings in a user study with 16 participants. The study measured when the users could detect both if a change in the transformation of an object was successful, and if a change in the material parameters was successful. Generating uniform random numbers according to the visual acuity model did not have a measurable difference on the computation performance.

The proposed system uses a head mounted display, but it could be extended to support a desktop display with eye tracking. In the future, we are interested in exploring more ways to make content generation for virtual reality devices easier and faster.

ACKNOWLEDGMENTS

The authors would like to thank the creators of the 3D models used in the user study especially Frank Meinl for the Crytek Sponza model seen in Fig. 1. In addition, the authors would like to thank Heli Väätäjä, Chelsea Kelling, and Otto Kauhanen for helpful discussions.

REFERENCES

AMD. 2017. RadeonRays SDK. Online. (2017). Available: https://github.com/GPUOpen-LibrariesAndSDKs/RadeonRays_SDK, Referenced: 24th of May 2017.

Luc Devroye. 1986. *Non-Uniform Random Variate Generation.* Springer.

Andrew Duchowski, David Bate, Paris Stringfellow, Kaveri Thakur, Brian Melloy, and Anand Gramopadhye. 2009. On Spatiochromatic Visual Sensitivity and Peripheral Color LOD Management. *ACM Transactions on Applied Perception* 6, 2 (2009).

Brian Guenter, Mark Finch, Steven Drucker, Desney Tan, and John Snyder. 2012. Foveated 3D Graphics. *ACM Transactions on Graphics* 31, 6 (2012).

Matias Koskela, Timo Viitanen, Pekka Jääskeläinen, and Jarmo Takala. 2016. Foveated Path Tracing. In *Proceedings of the International Symposium on Visual Computing*.

The community of LuxRender. 2013. *LuxRender Documentation: Refine Brush.* Available: http://www.luxrender.net/wiki/Refine_Brush, Referenced: 26th of May 2017.

Andrew MacQuarrie and Anthony Steed. 2017. Cinematic Virtual Reality: Evaluating the Effect of Display Type on the Viewing Experience for Panoramic Video. In *Proceedings of the IEEE Virtual Reality*.

Hunter Murphy, Andrew Duchowski, and Richard Tyrrell. 2009. Hybrid Image/Model-Based Gaze-Contingent Rendering. *ACM Transactions on Applied Perception* 5, 4 (2009).

Matt Pharr and Greg Humphreys. 2010. *Physically Based Rendering: From Theory to Implementation* (2nd ed.). Morgan Kaufmann.

Pixar. 2017. *RenderMan 20 Documentation: Rendering Efficiently.* Available: https://renderman.pixar.com/resources/RenderMan_20/tutorialRenderingEfficiently.html, Referenced: 26th of May 2017.

Daniel Pohl, Xucong Zhang, and Andreas Bulling. 2016. Combining Eye Tracking with Optimizations for Lens Astigmatism in Modern Wide-Angle HMDs. In *Proceedings of the Virtual Reality*.

Martin Reddy. 2001. Perceptually Optimized 3D Graphics. *IEEE computer Graphics and Applications* 21, 5 (2001).

Thorsten Roth, Martin Weier, Jens Maiero, André Hinkenjann, and Yongmin Li. 2015. Guided High-Quality Rendering. In *Proceedings of the International Symposium on Visual Computing*.

Takashi Shibata. 2002. Head mounted display. *Displays* 23, 1–2 (2002).

Michael Stengel, Steve Grogorick, Martin Eisemann, and Marcus Magnor. 2016. Adaptive Image-Space Sampling for Gaze-Contingent Real-time Rendering. *Computer Graphics Forum* 35, 4 (2016).

Karthik Vaidyanathan, Marco Salvi, Robert Toth, Tim Foley, Tomas Akenine-Möller, Jim Nilsson, Jacob Munkberg, Jon Hasselgren, Masamichi Sugihara, Petrik Clarberg, et al. 2014. Coarse Pixel Shading. In *Proceedings of High Performance Graphics*.

Martin Weier, Thorsten Roth, Ernst Kruijff, André Hinkenjann, Arsène Pérard-Gayot, Philipp Slusallek, and Yongmin Li. 2016. Foveated Real-Time Ray Tracing for Head-Mounted Displays. 35, 7 (2016).

Martin Weier, Michael Stengel, Thorsten Roth, Piotr Didyk, Elmar Eisemann, Martin Eisemann, Steve Grogorick, André Hinkenjann, Ernst Kruijff, Marcus Magnor, Karol Myszkowski, and Philipp Slusallek. 2017. Perception-driven Accelerated Rendering. *Computer Graphics Forum* 36, 2 (2017).

Eric Weisstein. 2002. Lambert W-function. Online. (2002). Available: http://mathworld.wolfram.com/LambertW-Function.html, Referenced: 1st of June 2017.

PUBLICATION 4

[P4]

Instantaneous Foveated Preview for Progressive Monte Carlo Rendering

M. Koskela, K. Immonen, T. Viitanen, P. Jääskeläinen, J. Multanen and J. Takala

Computational Visual Media 4.3 (2018)
DOI: 10.1007/s41095-018-0113-0

Computational Visual Media
https://doi.org/10.1007/s41095-018-0113-0 Vol. 4, No. 3, September 2018, 267–276
Research Article

Instantaneous foveated preview for progressive Monte Carlo rendering

Matias K. Koskela[1] (✉), **Kalle V. Immonen**[1], **Timo T. Viitanen**[1], **Pekka O. Jääskeläinen**[1], **Joonas I. Multanen**[1], **and Jarmo H. Takala**[1]

Abstract Progressive rendering, for example Monte Carlo rendering of 360° content for virtual reality headsets, is a time-consuming task. If the 3D artist notices an error while previewing the rendering, they must return to editing mode, make the required changes, and restart rendering. We propose the use of eye-tracking-based optimization to significantly speed up previewing of the artist's points of interest. The speed of the preview is further improved by sampling with a distribution that closely follows the experimentally measured visual acuity of the human eye, unlike the piecewise linear models used in previous work. In a comprehensive user study, the perceived convergence of our proposed method was 10 times faster than that of a conventional preview, and often appeared to be instantaneous. In addition, the participants rated the method to have only marginally more artifacts in areas where it had to start rendering from scratch, compared to conventional rendering methods that had already generated image content in those areas.

Keywords foveated rendering; progressive rendering; Monte Carlo rendering; preview; 360° content

1 Introduction

Virtual reality (VR) is increasingly used for both work and entertainment. One challenge posed by VR is the generation of 360° content, especially due to the high resolution requirements of VR devices, and the need for meaningful interesting content in every direction in 3D space. Rendering high resolution images with progressive photorealistic methods typically takes hours to complete, while approximate preliminary results can be produced much faster. Moreover, in *Monte Carlo* rendering, halving the error in the rendered images requires quadrupling the number of rendered samples [1]: the payoff obtained from additional rendering time reduces quickly.

If the artist notices during previewing that something is wrong with the scene, they must abort rendering, make the required changes, and restart rendering all over again. Restarting the rendering process from scratch is required: for example, changing the illumination conditions potentially affects every pixel of the image. In many cases, the artist can create an approximation of the scene with a faster rendering method, but it typically lacks photorealistic effects such as reflections and indirect lighting, which require slow, offline methods to render. If the artist can preview the rendering sooner, it directly improves the speed of the content creation process.

In this paper, we propose a method for speeding up the preview of progressively rendered images by applying *foveated rendering* to reduce the quality in the peripheral regions of vision. Quality can be reduced because visual acuity decreases with increasing eccentricity, as a consequence of drop in the density of rod and cone cells in the retina off-axis [2]. It has been estimated that more than 90% of real-time path tracing samples can be omitted by employing foveated rendering [3].

1 Tampere University of Technology, Tampere, 33720, Finland. E-mail: M. K. Koskela, matias.koskela@tut.fi (✉); K. V. Immonen, kalle.immonen@tut.fi; T. T. Viitanen, timo.2.viitanen@tut.fi; P. O. Jääskeläinen, pekka.jaaskelainen@tut.fi; J. I. Multanen, joonas.multanen@tut.fi; J. H. Takala, jarmo.takala@tut.fi.

Manuscript received: 2017-12-23; accepted: 2018-02-17

We make the following novel contributions in this article:

1. an optimized human visual acuity model, which can be used to accurately generate path tracing samples;
2. an evaluation of the benefits of foveated rendering in speeding up progressive rendering previews, validated by a user study showing that the proposed method is 10 times faster than a conventional previewing approach.

The results of the proposed preview framework are shown in Fig. 1.

We extend our previous work [4] with an additional evaluation of the questionnaire presented to the participants of the user study, and an evaluation of how the participants assessed artifacts in the results.

2 Related work

The idea in foveated rendering is to adapt the rendered visual quality to the physiological abilities of the human visual system. Foveated rendering requires predicting or measuring the direction of the user's gaze. Consequently, a real-time requirement is imposed on rendering. There is a large body of work on real-time foveated rendering, which is summarized by a recent comprehensive literature review by Weier et al. [5]. Foveated rendering is very appealing when using *head-mounted displays* (HMDs), which typically have a wider *field of view* (FOV) than desktop monitors, and only a single observer per display [6]. The wider FOV means that the user can see clearly only a proportionally smaller area of the screen. In addition, HMDs require low latency rendering to reduce motion sickness, which calls for greater optimization than for a desktop setup. Moreover, accuracy of eye tracking is better with an HMD setup because the camera used to measure the gaze direction is fixed to the head of the user [7].

One method to perform foveated rendering is to rasterize the scene at multiple resolutions [8]. The system renders only the region centered on the gaze direction at the highest resolution, and uses larger pixels in the user's peripheral vision. Another approach is to include foveated rendering into a deferred shading pipeline by shading only some pixels, and by interpolating results for the remainder of the pixels [9, 10].

Current hardware supports only a fixed, predefined resolution for rasterization. Therefore, foveated rendering can be implemented more easily with ray-tracing-based techniques because they support arbitrary sampling patterns in screen space. Consequently, foveated ray tracing has gained academic interest in recent years [11, 12]. An intuitive idea is to distribute samples according to the smallest detectable spatial frequency according to a model of human visual acuity:

$$m(e) = \begin{cases} 1.0, & 0 \leqslant e \leqslant 5.79 \\ 7.49/(0.3e+1)^2, & e > 5.79 \end{cases} \quad (1)$$

where e is the eccentricity angle, i.e., the angle from the gaze direction [13]. This model is derived

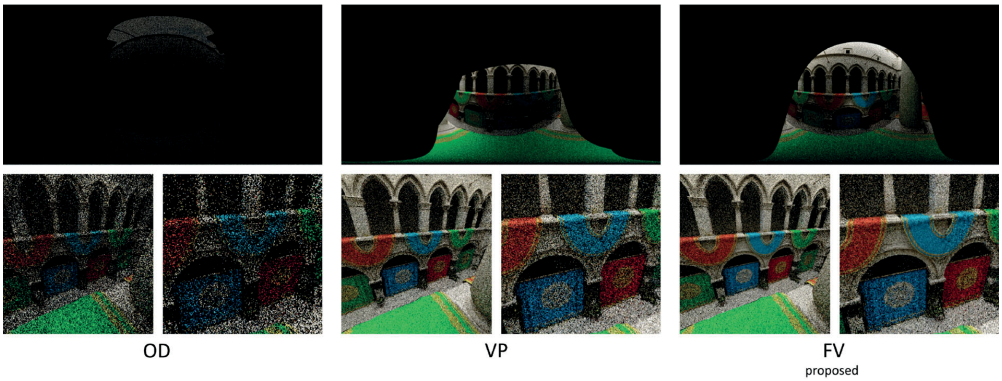


Fig. 1 Results after two seconds of rendering with a static point of interest. Above: rendering buffer. Below: preview on screen, and close-up of the point of interest. Note how it already starts to converge in our proposed foveated preview approach (FV) which uses eye tracking and a human visual acuity model. On the other hand, the edges in FV are noisier than when uniformly sampling the viewport area (VP). Uniform sampling of the whole 360° image (OD) is noisier than the other methods.

from various psychophysical studies. The equation describes just one radius of visual acuity, and the actual 2D model is obtained by taking a solid of revolution determined by the equation, where the axis of revolution is at $e = 0$.

Due to the complexity of the visual acuity model, linear denominator models may be used instead of the quadratic denominator model in Eq. (1). However, they are not as accurate in the peripheral parts of vision [8]. A simplified version uses a linear fall-off between the maximum and the minimum sampling probability [9–11], or even a static probability with respect to gaze direction [14].

When previewing progressively rendered images, rendering of the region of interest needs to converge as quickly as possible to allow the artist to abort the rendering as soon as possible, when needed, and to make the required changes sooner. One way to vary the convergence rate is to apply a so-called *guided preview*, and have more samples in an area chosen by the artist with a pointing device [15]. Another idea is to select an area of the image where the sample computation is concentrated [16]. Importance masking [17] is an advanced version of area selection.

In this paper, we utilize the idea of a guided preview, and use one of the most intuitive kinds of guidance: the point at which the user is looking. This means that there is no need to manually select the region of interest, and instead the system automatically detects the user's point of interest. Moreover, we use the quadratic denominator visual acuity model instead of coarser models. Compared to coarser models, the more accurate model places fewer samples in the peripheral regions of vision, and therefore allows faster convergence in the fovea. In addition, previous work on foveated rendering has concentrated on real-time rendering, while we propose its use to preview off-line rendering.

3 Proposed method

The aim of our previewing method is to render images for VR and to give the artist an instant preview. The method tracks the eye of the user and generates samples according to the visual acuity model. Doing so does not worsen the user experience because resolution can be reduced significantly in the peripheral parts of vision without affecting search task performance [18]. In other words, the user can find the area of interest in equal time compared to when using a conventional preview. However, the area of interest converges to the final result significantly faster than when the image is uniformly sampled.

Sampling the world according to a visual acuity model requires random image space positions to be generated with probability density given by Eq. (1). Note that the equation from Reddy [13] is not directly usable as a probability density function because its integral over the entire space is not equal to one. We show later how to transform the equation to fulfil the constraint in Eq. (6).

Progressive rendering produces the correct color only after averaging many samples. Instead of clamping the model to one sample per pixel, we keep its value as cycles per degree. Using cycles per degree makes sure that the image converges quickly in the gaze direction. In other words, more than one sample may be placed into a single pixel during rendering of the frame. This in turn ensures, for example, that better anti-aliasing occurs faster in the area of interest. Due to the probabilistic nature of sampling, some pixels may be completely unsampled for the first few frames. While unsampled areas could be reconstructed [19], because the pixels are likely to be sampled quickly thereafter, we do not attempt to reconstruct the missing pixels.

Generating random numbers analytically according to the solid of revolution of Eq. (1) is not feasible for the targeted real-time preview method. Therefore, we simplify the generation by using polar coordinates: a uniformly distributed angular coordinate ϕ, and a radial coordinate r, which is the distance from the center of the vision, i.e., eccentricity angle e. The angle ϕ can be generated by one of the many algorithms available for quickly generating uniformly distributed random numbers. To achieve correct distribution for r, the probability density of Eq. (1) must be modified based on the circumference of circle $2\pi R$ (where R is the radius):

$$g(e) = 2\pi e m(e) = \begin{cases} 2\pi e, & e \leqslant 5.79 \\ 14.98\pi e/(0.3e+1)^2, & e > 5.79 \end{cases} \tag{2}$$

A uniform distribution can be transformed to any other distribution using the *inversion method* [20]:

$$r = f^{-1}(u) \tag{3}$$

where u is a uniformly distributed random number in $[0, 1]$, f is the desired cumulative distribution

function, and r is a random number with cumulative distribution f. The inversion method requires us to derive the cumulative distribution function from the probability density defined in Eq. (2) by integrating $g(e)$ over the interval $[0, x]$:

$$h(x) = \int_0^x g(e)\mathrm{d}e \tag{4}$$

so

$$h(x) = \begin{cases} \pi x^2, & x \leqslant 5.79 \\ \left(\dfrac{1}{0.3x+1} + \ln(0.3x+1)\right) & \\ \quad \times 166.4\pi - 612.3, & x > 5.79 \end{cases} \tag{5}$$

We chose the upper limit of the function at an eccentricity angle of 80° because at that point the model begins to reach zero. In addition, such an angle suffices to cover all typical HMD FOVs. Finally, the integral needs to be made consistent with the requirement that a cumulative distribution function runs from 0 to 1:

$$f(x) = \frac{h(x)}{G(80)} \tag{6}$$

where $G(e) = \int g(e)\mathrm{d}e$.

Equation (3) requires the inverse of f in Eq. (6). However, it cannot be expressed in terms of standard mathematical functions and the Lambert W-function [21] is needed. We simplify the function by approximating it with a fitted fourth-order polynomial determined numerically by least squares regression:

$$f^{-1}(u) \approx \begin{cases} 18.64\sqrt{u}, & u \leqslant 0.0965 \\ 25.09u^4 + 0.1680u^3 & \\ \quad + 27.61u^2 + 23.87u & u > 0.0965 \\ \quad + 3.232, & \end{cases} \tag{7}$$

The maximum approximation error in Eq. (7) is 1.8% and the integral of the difference is less than 0.04%, which are very small, especially in comparison to coarser approximations in previous work. Small error means that the model generates fewer unneeded samples in the peripheral visual regions and more in the center, leading to faster convergence.

In addition to utilizing the sampling pattern shown in Eq. (7), the proposed method allows eye tracking to be frozen. This feature is used if the user wants to look around and still generate most new samples around a certain point of interest.

In the proposed method the users preview the results with a VR HMD with eye tracking capability, but also a desktop setup could be used. We chose an HMD because a VR headset gives better spatial awareness and enjoyment [22] and, therefore, it is likely for an artist to preview the scene with a device similar to the ones used by the consumers of the rendered content. Moreover, future versions of 3D design tools might include user interfaces where the design is done partially or completely in a virtual environment using an HMD [23].

4 User study

To measure the subjective performance of our proposed instant preview method, we conducted a user study. It started with a questionnaire on a five-point Likert scale concerning the participant's background in 3D graphics. The questions posed are listed in Table 1.

The study used five different scenes and three different preview methods, in random order. The test scenes were BMW, Classroom, Conference, Sibenik, and Sponza. A sample view of each scene can be seen in Fig. 2. We chose the scenes to represent different 360° rendering scenarios.

4.1 Preview methods

Our study compared three different preview methods:

- *Omnidirectional preview* (OD): Samples were distributed uniformly to every possible point in an equirectangular image. This method represents conventional baseline rendering without preview optimization.

Table 1 Arithmetic mean (μ) and standard deviation (σ) of answers to the background questionnaire

Question	μ	σ
1. Age?	28.5	4.3
2. Gender? (5 = female, 1 = male)	2.3	1.9
3. How much previous experience do you have using applications with 3D graphics (like 3D games)?	3.5	1.3
4. How much previous experience do you have with offline 3D rendering (like Blender)?	2.2	1.2
5. How much previous experience do you have with virtual reality or augmented reality devices?	2.4	1.1
6. Have you experienced virtual reality sickness?	2.9	1.4

Fig. 2 Sample views of test scenes used in the user study. Left to right: BMW, Classroom, Conference, Sibenik, and Sponza.

- *Viewport preview* (VP): Samples were distributed uniformly in the area currently viewable with the HMD. The idea was to simulate the rectangular area sampling used in some rendering engines.
- *Foveated preview* (FV): This was the proposed method, which distributed samples according to the visual acuity model centered on the gaze point of the eye-tracked user.

4.2 Single scene procedure

The procedure for each 3D scene can be seen in Fig. 3. We asked the participants to play the role of a 3D artist, and to choose an object in the 3D world. They were told that the object represents an object that they have just adjusted. Adjustment could have been, for example, changing the orientation of the object or changing its material parameters. Examples of both can be seen in Fig. 4.

After object selection, the rendering started, and the participants recorded rendering times. The first event was recorded at the point where the participants thought that they could determine if translation or rotation of the object was successful. The second measured time represented the event when the participants were able to determine if the material adjustment was successful. The idea was that at these points the artist could cancel the rendering, go back to editing mode, and make the required changes. In each scene, the procedure described above was repeated for each preview method. The order of preview methods was randomized.

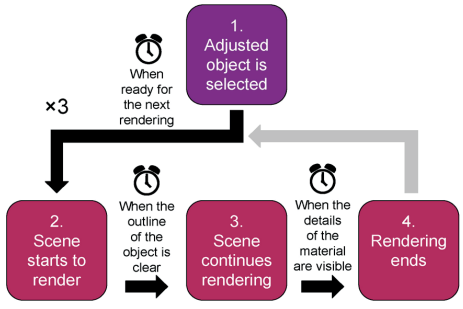


Fig. 3 Single scene procedure in the user study. Boxes are stages and arrows are participant's actions triggering transitions to other stages. Clocks represent points where the system saved timing.

We told the participants that it was important to record the timing at a similar rendering quality in each preview method. If the participant felt that even a single timing failed substantially, the whole rendering method in that scene was timed again.

After time for a single method was measured, we asked the participants to look around in the 360° image and to rate the prevalence of disturbing artifacts in the other areas of the image. The value was recorded on a five-point Likert scale, where one meant no artifacts were present and five meant so many were present that the scene was not discernible at all.

4.3 Rendering

We chose unidirectional path tracing with importance sampling as the progressive rendering method. AMD RadeonRays [24] was used for ray traversal and the path tracer ran on an AMD Fury X GPU. The host code ran on an Intel Core i7-6700K CPU with 16 GB of memory. The FOVE 0 VR headset was used as a

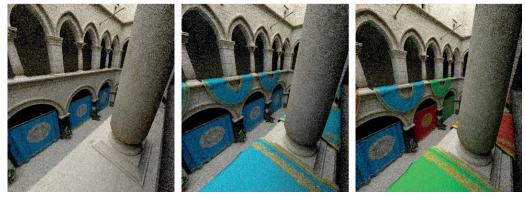


Fig. 4 Example of a 3D artist's workflow. First the artist places U-shaped cloths, which might require many previews of the positioning with the slow progressive rendering method, especially if the objects are transparent or reflective. Then the artist modifies the material of the objects and previews the changes.

viewing device in the study due to its eye tracking capability, needed for the proposed method.

Translation of the virtual camera was disabled as camera motion would have invalidated the progressive rendering samples. Thus, the assumed starting point was that the 3D artist had already chosen the camera position by utilizing a faster rendering method.

The system generated equirectangular images because they are common in VR applications, in the authors' experience. The preview method used trilinear filtering to eliminate flickering near the poles. With an unoptimized GPU implementation, generating and sampling mipmaps on the target machine took only about 1 ms of additional time compared to bilinear filtering. This drawback was reasonable since the target was 14 ms per preview frame to achieve the 70 Hz refresh rate needed for FOVE 0.

5 Results

The user study included 16 participants, of whom 11 were male and 5 were female. Their ages varied between 22 and 37. Two of the participants knew details of the test set-up beforehand, but their results were so close to the average of the other results that we concluded that this did not affect the results.

5.1 Questionnaire

Statistics for answers to the questionnaire can be found in Table 1. The average answer to questions regarding the participants' background in 3D graphics has $P = 0.196$ compared to the speedup of foveated rendering. This P value means that there is no significant correlation in the values, which was expected since the study should test the human visual system and not the person's experience in 3D graphics. In addition, the P value suggests that a user study with random users should give similar results to a user study with actual 360° rendering artists. The questions used in the calculation were questions 3 and 4. The timing used in the calculation was the difference between OD and FV.

5.2 User timing

The geometric means of all timing are shown in Fig. 5 and arithmetic means for each scene over all participants are listed in Table 2. The results show that the proposed method required only around 10% of the time required by the baseline method OD. The time savings directly translate to the frequency of the artist's feedback loop since rendering can be aborted 10 times faster. An equivalent comparison shows that previewing with VP is 3 times faster than with OD. Likewise, comparison of VP to FV shows that when rendering regular images rather than 360° images, foveated previewing can provide a 3× speedup of the previewing task.

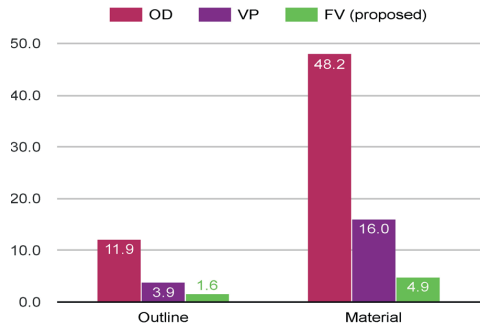


Fig. 5 Geometric mean of time measured in seconds over all 5 test scenes using the *visible outline* and *visible material* criteria for each of the three preview methods (smaller time is better).

5.3 Artifacts

Results of the assessment of artifacts can be seen in Fig. 6. The original idea of this measurement was to assess the reduction in quality for methods other than OD in directions away from the point of interest. These areas are not rendered at all in VP and FV because the user was looking at the point of interest throughout the test. We found out that this measurement was hard to record because every participant had a completely different idea about what should be considered a disturbing artifact. However, from the results we can see that the fast convergence of FV is perceived to have almost the same quality as OD. In contrast, VP clearly has the most artifacts. Note that if the slightly lower quality of other areas in FV is a problem in a progressive rendering system, then the system could be modified to use a hybrid of FV and OD.

5.4 Subjective observations

In an open discussion after the test, many participants reported that FV was so fast that it was hard to record the first timing at the right time. They also stated that it felt that the FV method converged instantly. On the other hand, several participants stated that the perceived slowness of OD might

Table 2 Arithmetic mean (μ) and standard deviations (σ) of the measured time for each scene in the user study. The results of the FV (proposed) and VP methods are compared to OD; pp stands for percent point. Large values of σ in OD are caused by the participants selecting different kinds of objects

Timing type	Outline visible						Material visible					
Preview method	OD		VP		FV		OD		VP		FV	
Value type	μ (s)	σ (s)	μ (%)	σ (pp)	μ (%)	σ (pp)	μ (s)	σ (s)	μ (%)	σ (pp)	μ (%)	σ (pp)
BMW	6.1	7.6	41.8	9.8	20.8	1.9	29.9	16.9	33.6	15.1	11.8	5.3
Classroom	15.6	7.7	27.1	3.1	11.6	2.1	67.9	79.2	30.8	35.7	7.9	4.6
Conference	41.9	59.7	29.7	8.6	7.6	2.3	137.2	197.4	33.9	46.0	8.2	8.6
Sibenik	24.7	20.8	29.5	9.2	11.4	3.9	76.7	55.2	34.3	29.0	10.9	10.1
Sponza	16.1	13.8	25.9	4.9	9.1	2.1	60.4	46.8	30.1	18.9	7.5	4.0

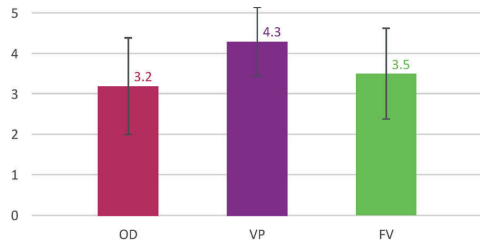


Fig. 6 Arithmetic means and the standard deviations of the amount of artifacts the participants saw when looking around. These numbers were measured after the users were satisfied with the rendering of their point of interest.

have caused them to get bored, inducing them to mark timing at a lower quality than with the other methods. These participants simply did not have enough patience to wait for the image to converge to the same level as with the other measurements. Hence, the results are skewed in favor of OD. Most of the participants also stated that they did not realize that eye tracking was used, and instead thought that the actual rendering was somehow faster. Not noticing the eye tracking indicates that the visual acuity model is a good way to distribute the samples.

5.5 Performance

All three methods showed similar computational performance. On the target machine, according to AMD CodeXL, it takes around 0.17 ms to launch 65,536 primary rays with all preview methods. This includes generating random pixel coordinates for the rays and calculating the ray origin and direction based on the random pixel coordinate. In the case of our proposed FV method, modifying the random number distribution with the inversion method requires some extra work (Eq. (7)). However, our measurements showed that the extra work done in FV to generate non-uniform random numbers is entirely hidden by the latencies of memory accesses and the kernel launch.

The ray tracing performance is dependent on the user's gaze or head direction with the FV and VP methods, respectively. In contrast, OD has the same ray tracing performance independent of where the user is looking at. While OD yields a larger number of samples per second than the other methods, this result can be misleading because many of the rays are sent to directions that are easier to ray trace, e.g., directly to a skybox.

5.6 Latency

Because the users preview the content with an HMD device, the latency should be kept low enough to not hinder the rendering experience. None of the participants of the user study mentioned any issues with latency. Only a few reported some minor VR motion sickness, but they had also had similar symptoms in other VR experiments. Moreover, for most participants, the latency was low enough for them not to realize that the system reacted to their gaze direction.

The components adding up to the total rendering latency are shown in Fig. 7. The HMD device has a screen refresh rate of 70 Hz, which means that we need to have a new preview frame ready every 14.3 ms. Otherwise the HMD displays the same frame for 28.6 ms, thus causing a *frame drop*. To avoid reducing the quality of experience, our code is designed so that it always meets the 14 ms target. All work other than progressive rendering itself takes around 4 ms in our code. This work includes, for example, updating the UI, checking inputs, generating the mipmap, sampling it, and sending the image to the screen buffer with DirectX. In the 10 ms time left from the 14 ms target after all other work, the progressive renderer is able to path trace a batch of approximately 100,000 samples.

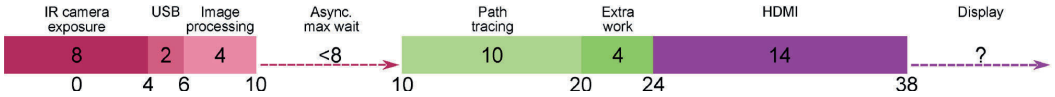


Fig. 7 Breakdown of the system latency in milliseconds. Best-case latency is approximately 38 ms. Worst-case latency is hard to determine, since we do not know all the internals of the HMD used. Moreover, green timing is the only one we can affect without changing the HMD.

The actual end-to-end latency of foveated rendering, from eye movement to pixels being illuminated in the HMD display, is hard to determine because we do not know all the internals of the FOVE 0 driver and hardware. To the best of our knowledge, the exposure time of the eye-tracking camera is 8 ms, transferring the data to the driver takes 2 ms, and processing the data takes approximately 4 ms [25]. However, the eye-tracking device used in FOVE 0 has a refresh rate of 120 Hz [26], meaning that the exposure and processing of different frames occur in parallel. At the beginning of every frame our code queries the driver for the latest eye position, which means that if the frames of the display and eye tracking are not synchronized by the driver, in the worst case the eye position data used is that from an image processing phase that ended 8 ms ago.

It should also be noted that the image captured by the eye-tracking camera may show motion blur due to the movement of the eye. To simplify analysis, we assumed the eye to be moving at a constant speed, and also assumed that the eye position estimate produced by the image processing phase corresponds to the eye's position at the midpoint of the exposure interval. Consequently, we started our latency timing 4 ms after the start of the exposure.

After processing ends, our code swaps the image to the displays. We have not measured how long it takes for the FOVE 0 display to illuminate the pixels after the buffer swap. Since we meet the 14 ms timing requirement it is likely that the frame is moved almost immediately to the screen. FOVE 0 moves the frame via HDMI 1.4 [26], and with typical transfer speeds it should take 14 ms to move the frame to the display.

6 Conclusions

In this paper we have proposed a foveation-based preview system for progressive rendering. The system tracks the user's gaze and distributes samples according to a visual acuity model. Generating the sample locations with the proposed method did not show a measurable overhead in computational performance.

We measured the benefits of the system in a user study with 16 participants, who were asked to indicate how fast the different preview methods reached specified levels of quality. The targets used in the study were (i) when the users could detect if a change in the transformation of an object was successful, and (ii) when they could detect if a change in material parameters was successful.

The results showed that the rendered image converges at the user's point of interest 10 times faster than with conventional uniform sampling over the whole 360° image area. In practice this means that the 3D artist can abort rendering 10 times earlier, shortening the artist's feedback loop time and thereby improving working efficiency significantly.

Most of the user study participants did not realize that eye tracking was used, and instead thought that the rendering process itself was faster, which was the desired end result. In addition, participants rated the proposed system to have only slightly more artifacts than in areas where conventional rendering has already rendered image content progressively for several seconds and the proposed method needs to start from scratch. This is likely due to the speed of the proposed method, which is supported by the fact that many participants stated that the proposed method appears to converge instantly. The perception of foveated rendering did not have significant correlation with the participant's background in 3D graphics.

Acknowledgements

The authors would like to thank the creators of the 3D models used in the user study: Christophe Seux for Classroom, Anat Grynberg and Greg Ward for Conference, Marko Dabrovic for Sibenik (License: CC BY-NC), and Frank Meinl for Crytek Sponza (License: CC BY 3.0). In addition, the authors would like to thank Heli Väätäjä, Chelsea Kelling, and Otto Kauhanen for helpful discussions.

The work was financially supported by the TUT Graduate School, Nokia Foundation, Emil Aaltonen Foundation, Finnish Foundation for Technology Promotion, Academy of Finland (funding decision 297548), Finnish Funding Agency for Technology and Innovation (funding decision 40142/14, FiDiPro-StreamPro) and ARTEMIS joint undertaking under grant agreement no 621439 (ALMARVI).

Electronic Supplementary Material Supplementary material is available in the online version of this article at https://doi.org/10.1007/s41095-018-0113-0.

References

[1] Pharr, M.; Jakob, W.; Humphreys, G. *Physically Based Rendering: From Theory to Implementation,* 2nd edn. Morgan Kaufmann, 2010.

[2] Strasburger, H.; Rentschler, I.; Jüttner, M. Peripheral vision and pattern recognition: A review. *Journal of Vision* Vol. 11, No. 5, 13, 2011.

[3] Koskela, M.; Viitanen, T.; Jääskeläinen, P.; Takala, J. Foveated path tracing. In: *Advances in Visual Computing. Lecture Notes in Computer Science, Vol. 10072.* Bebis, G.; Boyle, R.; Parvin, B. et al. Eds. Springer Cham, 723–732, 2016.

[4] Koskela, M.; Immonen, K.; Viitanen, T.; Jääskeläinen, P.; Multanen, J.; Takala, J. Foveated instant preview for progressive rendering. In: Proceedings of the SIGGRAPH Asia 2017 Technical Briefs, Article No. 10, 2017.

[5] Weier, M.; Stengel, M.; Roth, T.; Didyk, P.; Eisemann, E.; Eisemann, M.; Grogorick, S.; Hinkenjann, A.; Kruijff, E.; Magnor, M.; Myszkowski, K.; Slusallek, P. Perception-driven accelerated rendering. *Computer Graphics Forum* Vol. 36, No. 2, 611–643, 2017.

[6] Shibata, T. Head mounted display. *Displays* Vol. 23, Nos. 1–2, 57–64, 2002.

[7] Lee, E. C.; Park, K. R. A robust eye gaze tracking method based on a virtual eyeball model. *Machine Vision and Applications* Vol. 20, No. 5, 319–337, 2009.

[8] Guenter, B.; Finch, M.; Drucker, S.; Tan, D.; Snyder, J. Foveated 3D graphics. *ACM Transactions on Graphics* Vol. 31, No. 6, Article No. 164, 2012.

[9] Vaidyanathan, K.; Salvi, M.; Toth, R.; Foley, T.; Akenine-Möller, T.; Nilsson, J.; Munkberg, J.; Hasselgren, J.; Sugihara, M.; Clarberg, P.; Janczak, T.; Lefohn, A. Coarse pixel shading. In: Proceedings of High Performance Graphics, 9–18, 2014.

[10] Stengel, M.; Grogorick, S.; Eisemann, M.; Magnor, M. Adaptive image-space sampling for gaze-contingent real-time rendering. *Computer Graphics Forum* Vol. 35, No. 4, 129–139, 2016.

[11] Weier, M.; Roth, T.; Kruijff, E.; Hinkenjann, A.; Pérard-Gayot, A.; Slusallek, P.; Li, Y. Foveated real-time ray tracing for head-mounted displays. *Computer Graphics Forum* Vol. 35, No. 7, 289–298, 2016.

[12] Murphy, H. A.; Duchowski, A. T.; Tyrrell, R. A. Hybrid image/model-based gaze-contingent rendering. *ACM Transactions on Applied Perception* Vol. 5, No. 4, Article No. 22, 2009.

[13] Reddy, M. Perceptually optimized 3D graphics. *IEEE Computer Graphics and Applications* Vol. 21, No. 5, 68–75, 2001.

[14] Pohl, D.; Zhang, X.; Bulling, A. Combining eye tracking with optimizations for lens astigmatism in modern wide-angle HMDs. In: Proceedings of the IEEE Virtual Reality, 269–270, 2016.

[15] Roth, T.; Weier, M.; Maiero, J.; Hinkenjann, A.; Li, Y. Guided high-quality rendering. In: *Advances in Visual Computing. Lecture Notes in Computer Science, Vol. 9475.* Bebis, G.; Boyle, R.; Parvin, B. et al. Eds. Springer Cham, 115–125, 2015.

[16] Pixar. Renderman 20 documentation: Rendering efficiently. 2017. Available at `https://renderman.pixar.com/resources/RenderMan_20/tutorialRenderingEfficiently.html`.

[17] The community of LuxRender. LuxRender documentation: Refine brush. 2013. Available at `http://www.luxrender.net/wiki/Refine_Brush`.

[18] Duchowski, A. T.; Bate, D.; Stringfellow, P.; Thakur, K.; Melloy, B. J.; Gramopadhye, A. K. On spatiochromatic visual sensitivity and peripheral color LOD management. *ACM Transactions on Applied Perception* Vol. 6, No. 2, Article No. 9, 2009.

[19] Viitanen, T.; Koskela, M.; Immonen, K.; Mäkitalo, M.; Jääskeläinen, P.; Takala, J. Sparse sampling for real-time ray tracing. In: Proceedings of the 13th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications, Vol. 1, 295–302, 2018.

[20] Devroye, L. *Non-Uniform Random Variate Generation.* Springer-Verlag, 1986.

[21] Weisstein, E. Lambert W-function. Available at `http://mathworld.wolfram.com/LambertW-Function.html`.

[22] MacQuarrie, A.; Steed, A. Cinematic virtual reality: Evaluating the effect of display type on the viewing experience for panoramic video. In: Proceedings of the IEEE Virtual Reality, 45–54, 2017.

[23] Stark, R.; Israel, J. H.; Wöhler, T. Towards hybrid modelling environments—Merging desktop-CAD and virtual reality-technologies. *CIRP Annals* Vol. 59, No. 1, 179–182, 2010.

[24] AMD. Radeon Rays SDK. Available at `https://github.com/GPUOpen-LibrariesAndSDKs/RadeonRays_SDK`.

[25] Rodrigo. What is the latency of FOVE eye tracking? Available at `https://support.getfove.com/hc/en-us/articles/115000733714-What-is-the-Latency-of-FOVE-Eye-Tracking-`.

[26] FOVE Inc. Tech specs. Available at `https://www.getfov.com/`.

Matias K. Koskela received his bachelor and master degrees with honors from Tampere University of Technology in 2014 and 2015, respectively. His research interests include optimizations and parallelism in real-time rendering, especially in real-time ray tracing.

Kalle V. Immonen received his M.Sc. (Tech.) degree in pervasive systems from Tampere University of Technology (TUT) in 2017. He is now working as a project researcher at TUT. His research interests include computer graphics methods and algorithms.

Timo T. Viitanen received his M.Sc. degree in embedded systems from Tampere University of Technology (TUT) in 2013, and is now a graduate student in the Laboratory of Pervasive Computing, TUT. He is the recipient of a TUT graduate school position and was awarded a Nokia Scholarship in 2014. His research interests include computer architecture and computer graphics.

Pekka O. Jääskeläinen (Adjunct Professor) has worked on heterogeneous platform customization and programming since the early 2000s. He has published over 60 academic papers in journals and conferences, and is an active contributor to various heterogeneous parallel platform related open source projects. In addition to his ongoing research on methods and tools to reduce the engineering effort involved in design and programming of heterogeneous platforms, he is interested in next generation programmable graphics architectures for photorealistic real-time rendering in the context of small form factor VR/AR products of the future.

Joonas I. Multanen received his M.Sc. degree in electrical engineering from Tampere University of Technology (TUT) in 2015. He is currently a graduate student at the Laboratory of Pervasive Computing, TUT. His research interests include energy efficient computer architectures and computer graphics.

Jarmo H. Takala received his M.Sc. (hons) degree in electrical engineering and Dr.Tech. degree in information technology from Tampere University of Technology (TUT), Tampere, Finland, in 1987 and 1999, respectively. From 1992 to 1995, he was a research scientist at VTT-Automation, Tampere, Finland. Between 1995 and 1996, he was a senior research engineer at Nokia Research Center, Tampere, Finland. From 1996 to 1999, he was a researcher at TUT. Since 2000, he has been a professor in computer engineering at TUT and is currently vice president of TUT. Dr. Takala is Co-Editor-in-Chief for *Journal of Signal Processing Systems*. During 2007–2011 he was Associate Editor and Area Editor for *IEEE Transactions on Signal Processing*. His research interests include circuit techniques, parallel architectures, and design methodologies.

PUBLICATION 5

[P5]

Foveated Real-Time Path Tracing in Visual-Polar Space

M. Koskela, A. Lotvonen, M. Mäkitalo, P. Kivi, T. Viitanen and P. Jääskeläinen

Eurographics Symposium on Rendering (DL-only Track). Ed. by T. Boubekeur and P. Sen.
2019
DOI: 10.2312/sr.20191219

Eurographics Symposium on Rendering (DL-only Track) (2019)
T. Boubekeur and P. Sen (Editors)

Foveated Real-Time Path Tracing in Visual-Polar Space

M. Koskela[1],A. Lotvonen[1], M. Mäkitalo[1], P. Kivi[1], T. Viitanen[1,2], and P. Jääskeläinen[1]

[1]Tampere University, Tampere, Finland
[2]Now with Nvidia, Helsinki, Finland

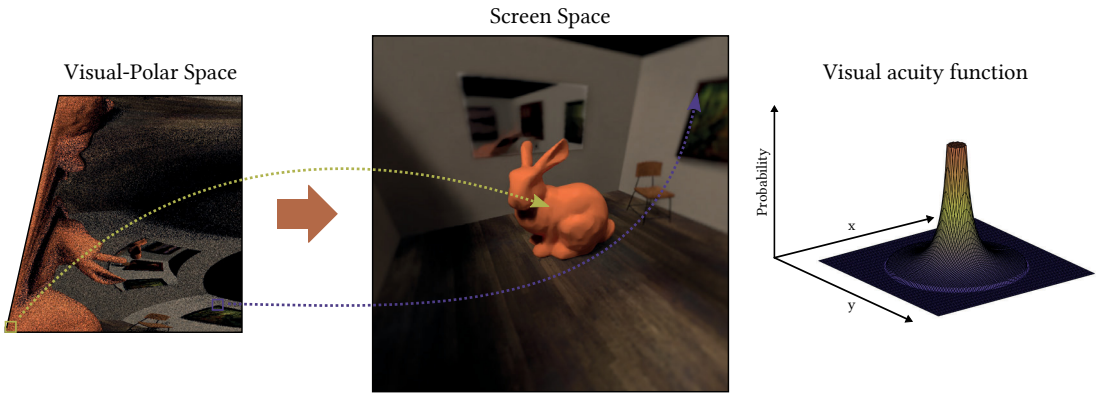


Figure 1: *Illustration of a path traced frame in Visual-Polar space, the denoised result transformed into Cartesian screen space, and the distribution of the path tracing samples in screen space. Path tracing and denoising in Visual-Polar space makes both* $2.5\times$ *faster.*

Abstract

Computing power is still the limiting factor in photorealistic real-time rendering. Foveated rendering improves perceived quality by focusing the rendering effort on where the user is looking at. Applying foveated rendering to real-time path tracing where we must work on a very small number of samples per pixel introduces additional challenges; the rendering result is thoroughly noisy and sparse in the periphery. In this paper we demonstrate foveated real-time path tracing system and propose a novel Visual-Polar space in which both real-time path tracing and denoising is done before mapping to screen space. When path tracing a regular grid of samples in Visual-Polar space, the screen space sample distribution follows the human visual acuity model, making both the rendering and denoising $2.5\times$ *faster with similar perceived quality. In addition, when using Visual-Polar space, primary rays stay more coherent, leading to improved utilization of the GPU resources and, therefore, making ray traversal* $1.3-1.5\times$ *faster. Moreover, Visual-Polar space improves 1 sample per pixel denoising quality in the fovea. We show that Visual-Polar based path tracing enables real-time rendering for contemporary virtual reality devices even without dedicated ray tracing hardware acceleration.*

CCS Concepts

• ***Computing methodologies*** → ***Perception;*** *Ray tracing; Virtual reality;*

1. Introduction

In order to produce an immersive and comfortable *virtual reality* (VR) or gaming experience with the evolving *head mounted displays* (HMD), the ability to generate high resolution content with a very high frame rate is essential. Computing power remains to be the limiting factor in generating realistic content for these devices. Some approaches to overcome this problem include having a spatially varying shading rate [VST*14; HGF14] and temporally varying shading locations with reprojection based on camera movement and animations [HEMS10; XLV18]. Another idea is to optimize rendering based on the fact that the human visual system recog-

DOI: 10.2312/sr.20191219

nizes details accurately only on a small area around the gaze point. Using this information in rendering optimization is typically called foveated rendering [WSR*17].

Path tracing is one of the most interesting options for generating realistic content. It simulates how photons interact with the scene and is thus able to naturally generate real life effects such as soft shadows, global illumination and reflections. Even though there is now dedicated acceleration hardware in consumer desktop GPUs for ray tracing [KMSB18], the achievable real-time path tracing rendering budget is still below 1 *sample per pixel* (spp) [Bar18]. The need for high resolution and refresh rate further reduces the available sample budget. In addition, higher spp counts are needed for rendering more complex materials and producing effects such as depth of field and motion blur.

In this article, we optimize path tracing rendering using foveation based methods. Our system utilizes a novel human visual system inspired sample coordinate space we call the Visual-Polar space. The main idea of the proposed method can be seen in Figure 1. Our rendering demonstrator uses only 0.4 spp and is still able to generate visually pleasing fully denoised results. To the best of our knowledge this is the first time foveated denoised path tracing is demonstrated in real-time with full resolution of contemporary VR device. The main contributions of this article are:

1. We propose a novel Visual-Polar space, which saves 61% of the rendering work compared to Cartesian screen space and allows coherent primary rays with improved SIMD/SIMT utilization. The only additional overhead is mapping back to the Cartesian screen space which with our test setup takes only 1.6 ms for a contemporary 1280×1440 VR HMD resolution.
2. We show that state-of-the-art real-time path tracing denoisers such as A-SVGF [SPD18], BMFR [KIM*19], and SVGF [SKW*17] can all operate in the proposed Visual-Polar space, which saves 61% of the denoising work and requires only minor changes to the denoiser.
3. We show that in the fovea the denoised output quality improves beyond conventional screen space quality, because when the denoiser is applied in the Visual-Polar space it automatically adapts to higher than 1 spp in the fovea.

2. Related Work

Path tracing rendering evaluates the rendering equation via Monte Carlo integration; therefore, it converges to the correct result when more and more noisy samples are averaged [Kaj86]. Even on recent GPUs with hardware acceleration for ray traversal, in real time we can only path trace approximately 1 spp [Bar18], which results in a highly noisy image. Two basic approaches to improving image quality are to apply denoising filters, and to reuse and accumulate samples from previous frames, resulting in a higher effective sample count. Recent real-time ray tracing methods combine both approaches to cope with 1 spp inputs.

One option for the real-time filtering is a wavelet-based filter called *Spatiotemporal Variance-Guided Filtering* (SVGF) [SKW*17]. To achieve real-time denoising, SVGF uses temporal accumulation to have an increased effective sample count and spatiotemporal luminance variance estimations for wavelet-based spatial filtering. SVGF's advanced version (A-SVGF) [SPD18] derives adaptive temporal accumulation factors to add support for temporal effects such as moving lights. It also improves the quality of materials, such as mirrors, where the first bounce motion vectors produce blurred results. On the other hand, regression-based methods have previously shown good denoising results in offline rendering [BRM*16], and a recent real-time work, called *Blockwise Multi-order Feature Regression* (BMFR), achieved even faster performance than wavelets [KIM*19]. The idea behind BMFR is to do fitting of the feature data to noisy input in relatively big blocks instead of deciding every pixel's color individually.

Path traced frames are typically viewed by a human visual system and an interesting characteristic of the system is that it can only resolve details accurately in a very small area around the gaze point. The number of cycles per eccentricity degree a human eye can resolve is described in the so-called visual acuity function as

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

where e is the eccentricity angle, and the result tells how many times per degree the image can change from completely white to completely black [Red97]. The function has been determined in user studies. If the change is not from completely white to completely black the resolvable cycles per degree is even less. The figure for showing different resolvable cycles per degree as a function of contrast is called the *Contrast Sensitivity Function* (CSF) [SRJ11].

Interestingly, it follows from the visual acuity function that if we had a rendering system capable of showing 60 cycles per degree, 95% of the rendered detail would be excessive [KVJT16]. On contemporary HMD devices this figure is around 75% depending on the resolution and the *field of view* (FOV). However, simply reducing sampling according to the visual acuity function causes both spatial and temporal aliasing artifacts in peripheral parts of the vision. Peripheral parts of the vision are sensitive especially to temporal artifacts [WSR*17] and, therefore, overly simple periphery quality reduction methods without temporal filtering are easily detectable by the user.

Foveated rendering utilizes these known features of the human visual system to reduce computational costs with a minimal noticeable quality decrease. The literature review [WSR*17] gives a comprehensive summary on previous foveated rendering research. The basic idea is to approximate the visual acuity of the human visual system in the distribution of samples. Foveated sample distribution can also be combined with other sample importance metrics, e.g., it is more important to shade the pixels around the object silhouettes [SGEM16].

With a rasterization type rendering, a coarse approximation of foveated sampling can be achieved by rendering multiple views of the scene at different resolutions [PSK*16; GFD*12; LW90; WWHW97]. The viewport is rendered fully only with a low resolution and a smaller image with greater pixel density is rendered and overlaid at the gaze region. Typically, there is some overlap in the

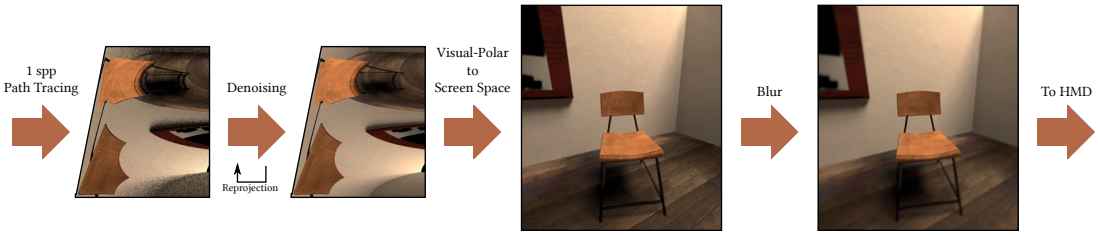


Figure 2: *The foveated Visual-Polar path tracing rendering pipeline is described in Sec. 3.*

rendered areas and interpolation is used to make the transition between resolutions smooth. For more accurate approximations there can also be a third intermediate resolution image [GFD*12].

Ray tracing is a good option for foveated rendering since it allows flexible sampling in screen space [KVJT16]. With rasterization, flexible sampling would require modifying the rasterization algorithm, which is typically done with dedicated hardware accelerators. The new Nvidia Turing architecture supports variable rate shading in screen space. However, the shading rate needs to be decided for a square block of pixels and completely free shading locations are not allowed [KMSB18]. However, due to how different work-items are scheduled to the processing elements of a programmable GPU, even with ray tracing it is difficult to do flexible sampling so that the whole computing capacity of the hardware is utilized.

For example, a common sample distribution of "linear falloff" uses a rendering probability for the peripheral part and does linear blending to full rendering in the fovea [WRK*16; WRHS18a]. Randomly killing some of the rays in the periphery causes idling lanes with both SIMD and SIMT hardware and is therefore suboptimal in terms of hardware utilization.

Sampling can also follow the visual acuity function more closely like in [KIV*18]. In this case, the GPU utilization is high, but the ray distribution in lanes is completely random, which makes primary rays completely incoherent and thus reduces cache locality. Another approach is to have a predefined sampling map [SCMP19]. They achieve fast interpolation from sparse ray tracing locations to full screen resolution, by using a precomputed triangulated mesh. In addition, one way to sample is to use spatially varying pixel density based on the distance to the shifting foveation point like in [RFS18].

An interesting option is to use log-polar space for the rendering. One downside of polar spaces is that they have discontinuity. However, artifacts can be avoided if discontinuity is taken into account, e.g., by using wrap around accesses. if ray tracing was done directly in log-polar space, the primary rays are coherent, and the rendered pixels are not sparse. One option is to first rasterize the G-buffer in Cartesian space, then map the result to log-polar space for deferred shading, and finally map the shading back to Cartesian screen space [MDZV18]. However, in the previous work log-polar distribution was not compared to the human visual system and even with the introduced kernel function all the tested parameters pack more samples to the gaze point than required.

In this paper we apply foveation-based rendering to path tracing and introduce the Visual-Polar space, which distributes the samples according to the visual acuity function. It also has coherent primary rays and all the lanes of SIMD/SIMT hardware are used, resulting in full utilization of all hardware resources. We use 1 spp real-time path tracing to generate the frames and denoise them with A-SVGF, BMFR, and SVGF. In contrast to previous work, we denoise in the Visual-Polar space before mapping the image to screen space, which means that the denoiser only needs to handle the lower resolution, and any possible denoising artifacts get circularly bent around the fovea.

3. Visual-Polar Space

The pipeline of path tracing rendering in the proposed Visual-Polar space is shown in Figure 2. The pipeline stages are described in the following subsections.

3.1. Path Tracing Setup

In this paper the 1 spp path tracing is done similarly as described in BMFR [KIM*19] and SVGF [SKW*17]. That is, we have one primary ray from every pixel and from the closest intersection point in the 3D space we trace one secondary indirect ray. From the hit points of the both rays we trace one shadow ray towards a random point in a random light.

The ray traversal, in other words, finding the closest intersection of each ray typically takes around 20% of the execution time in the path tracer used in our experiments. The coherence of the rays is quite high, with the path tracing setup used by this paper. Two out of four rays are either highly coherent primary rays or shadow rays traced from the first intersection towards the lights. With just one light these shadow rays are also highly coherent. Ray traversal of incoherent rays is significantly slower than ray traversal of coherent rays [Bar18]. Also, shader execution in the hit surface is faster with the coherent rays, because in a typical scene nearby areas have the same material. The best hardware utilization is achieved if all the work-items in a wavefront execute the same material code and, therefore, execute the same branches and load the same data. For these reasons, it is important that our foveated path tracing has similarly coherent primary rays as the Cartesian path tracing. Otherwise some of the gain from the foveation is lost in the inefficient ray traversal.

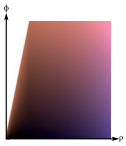


Figure 3: *Visualization of the axes ranges. Distance coordinate ρ runs from zero (black) to maximum field of view (purple) on horizontal axis. Angle coordinate ϕ runs from zero degrees (black) to 360 degrees (orange) on vertical axis. Notice how in the fovea area even with the triangular clipping the whole range of degrees is still present. When a uniform grid of samples is path traced in this space, the distribution shown in Figure 1 is achieved.*

3.2. Rendering

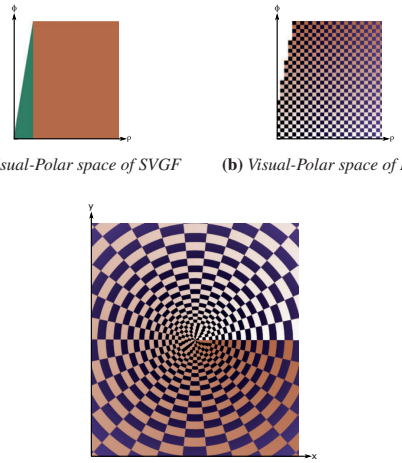
There were two main requirements we placed for the designed ray tracing sample distribution function. First, the ray traced samples should be picked in such a way that the primary rays (and also the first shadow rays) are coherent, which means that the rays in a single wavefront have approximately the same origin and traverse to approximately the same direction and, therefore, access the same *bounding volume hierarchy* (BVH) nodes as often as possible. Second, it should be feasible to perform denoising locally in the designed space that has less pixels to process than in the screen space.

An interesting option that fulfills both of the requirements is to ray trace in a polar coordinate space, so that the gaze point is always at the zero of the radius coordinate and the angle coordinate rotates around the gaze point. That is, if ray tracing a uniform grid in polar space, its sample distribution would be $\frac{1}{e}$, where e is the eccentricity angle. The problem is that this distribution does not accurately model the visual acuity function of the human eye. There are at least two simple ways to improve the distribution: Either adjust the number of samples on the angle coordinate ϕ, or change the scaling of the radius coordinate ρ.

Adjusting the number of samples on the angle coordinate requires a varying resolution on the ϕ-axis. For example, a constant sample distribution could be achieved by clipping the polar space along the $\phi = 2\pi\rho$ line, and with a more complicated clipping pattern we can match sampling with the visual acuity sample distribution. However, single sample coverage in the peripheral parts becomes stretched, which produces significant artifacts.

In the other option of scaling the radius coordinate ρ to follow the visual acuity distribution, the cumulative distribution function of the desired distribution and its inverse are needed [Dev86]. The inverse of the cumulative distribution function of the visual acuity function is too complex for a real-time implementation to be computed online [KIV*18]. However, we found that a fitted polynomial can approximate it efficiently enough for our use case. The downside of scaling the radius coordinate is that if the polar coordinate space has a constant resolution on the ϕ-axis, the first columns of samples are mapped to cover to whole fovea area and there are major stretching artifacts.

In summary, just varying the resolution of the ϕ-axis results in artifacts in peripheral vision, and just scaling the ρ-axis results in artifacts at the gaze point. Therefore, we chose a distribution where we combine the best parts of both approaches: In the fovea area we vary the resolution in the ϕ-axis and in the peripheral area we scale the ρ-axis.



(a) *Visual-Polar space of SVGF* **(b)** *Visual-Polar space of BMFR*

(c) *Screen space showing BMFR block distribution*

Figure 4: *Visual-Polar spaces for both of the denoisers. ρ is on the horizontal axis and ϕ is on the vertical axis. BMFR's blocks are shown in the screen space image. In the BMFR figures lighter orange means smaller ϕ values and darker purple means smaller ρ values.*

In order to have a constant distribution of samples in the fovea area, we use linear mapping on the ρ-axis and clip a triangular area off from the ϕ-axis. The fovea area is marked with green color in Figure 4a. The white clipped area above the green triangle reduces 12% of the original space size. We conservatively chose the clipping boundary so that the fovea has more than 1 spp in screen space.

In the peripheral part we use scaling of the ρ-axis to achieve the visual acuity falloff. The cumulative distribution function of the visual acuity is

$$\left(\frac{1}{0.3d+1} + \ln(0.3d+1)\right) \times 166.4\pi - 612.3, \qquad (2)$$

where d is the distance to the gaze point [KIV*18]. Outside the fovea the ρ-axis is scaled with Eq. 2 when mapping from screen space to the Visual-Polar space. In the actual implementation we avoid the logarithm and the division by using a least squares fitted polynomial as an approximation of this function.

The fitted inverse of the cumulative distribution function which is used when mapping from the Visual-Polar space to screen space is

$$25.09\rho^4 + 0.1680\rho^3 + 27.61\rho^2 + 23.87\rho + 3.232. \qquad (3)$$

This scaling is also visualized in the vertical axis of Figure 3.

On top of the visual acuity based scaling in the ρ-axis, we also scale the ρ-axis so that we keep the number of rendered paths constant no matter where the user is looking at on the screen. This is done by finding the greatest distance from the gaze point to the screen corners. The distance is used to scale the axis so that the maximum ρ is equal to it. This scaling dynamically changes the size of the fovea to be larger when the user looks at the edges of the vision, which compensates for the typically worse eye tracking accuracy in these areas [RWH*17].

For the path tracing itself we render a regular grid of pixels in the Visual-Polar (ρ, ϕ) space. When we compute the origins and directions for the primary rays, we do the mapping from the Visual-Polar space to screen space and then compute the origin and direction as is typically done in conventional screen space path tracing.

3.3. Denoising

An interesting feature of real-time path tracing denoisers like SVGF and BMFR is their use of temporal data that also reduces temporal flickering caused by the changing sample locations in the peripheral parts of the foveated rendering. We found that adapting these state-of-the-art real-time denoisers to the Visual-Polar space requires only minor modifications to them, which we describe in the following.

Out-of-bounds access handling: All the sampling in Visual-Polar space needs to be wrap around on the ϕ-axis. On the ρ-axis we used clamp-to-edge edge handling, but one could also use proper edge handling which rotates the sample to its correct position on the other side of the gaze point.

Temporal accumulation: Accessing temporal data needs to take the Visual-Polar space into account. Conventionally, the temporally-aware denoisers calculate from the 3D world positions of the current frames which screen space location they need to access in the previous frame. We simply apply the screen space to Visual-Polar space transformation to these locations and access data with bilinear sampling. When performing bilinear interpolation from four samples in the Visual-Polar space frame it needs to be taken into account that the height of the ϕ-axis can be different on both of the accessed columns. As in the original implementations of the denoisers, we decide separately for each sample whether to discard it due to, e.g., disocclusions.

Denoise filter sampling: With SVGF bilateral sampling and A-Trous sampling is done so that we consider the possibly smaller resolution in the ϕ-axis. With BMFR we do not scale the blocks based on the smaller resolution, but instead we reshape the Visual-Polar space so that the height of the ϕ-axis is always the same within one block column area. Synching the clipping with the BMFR blocks produces a staircase-like clipping boundary, which can be seen in Fig 4b. In addition, we keep the location of the clipping boundary synced with BMFR's constantly pseudo-randomly changing block locations.

Since the Visual-Polar space maps more pixels to the gaze point, it naturally scales the screen space A-Trous blur radius of SVGF and the screen space block size of BMFR. BMFR block size in screen space can be seen in Figure 4c. A smaller screen space area with the same number of path tracing samples in the Visual-Polar space enables the denoisers to produce better quality results with more difficult cases like reflections, which can be seen in Figure 5. The case is challenging for the denoisers, because the data of the world seen in the reflections is not in any way present in the feature buffers (G-buffer). For example, if the phenomenon BMFR tries to reconstruct is not present in the feature buffers, it reconstructs the result from gradient-like data available, e.g., in the world position buffer. Reconstructing a detailed sharp reflection from gradients yields a blurred result. In our experiments, SVGF performs better with sharp reflections, because it also uses color data in guiding the blurring. However, with the natural size reduction caused by the Visual-Polar space both of the denoisers have good quality in fovea. In the Visual-Polar space the size stays the same and, therefore, there is always the same number of path tracing samples affecting the denoising of a single pixel.

3.4. Mapping to Screen Space

The Visual-Polar space image can be mapped to screen space in one pass without synchronization. The mapping can be done with backwards projection, i.e., every screen space pixel samples their color from the Visual-Polar space pixels. Also in this case we handle out-of-bounds accesses with wrap around on the ϕ-axis and clamp-to-edge edge on the ρ-axis. We found that the wrap around on the ϕ-axis hides the discontinuity of the Visual-Polar space efficiently.

On ρ-axis the mapping is always either one-to-one or magnification. Therefore, bilinear sampling is enough for sufficient quality. In contrast, on ϕ-axis one screen space pixel can map to many Visual-Polar space pixels. Unlike in previous work, for the trilinear mapping from Visual-Polar space to screen space it is enough to generate just mipmaps on one axis of the smaller resolution image. Therefore, we use custom mipmap generation and sampling code, which generates mipmaps only on the ϕ-axis. Our unoptimized version of the code with the test setup takes less than 0.5 ms. This code could be highly optimized since the mipmaps are only required in the fovea area and the sampling positions can be precomputed.

3.5. Blur

After mapping to the screen space, we apply a moderate Gaussian blur. The idea is to blur just enough to remove most of the spatial aliasing problems but not too much to cause tunnel vision. We used zero blurring on the fovea area, and after the edge of the fovea we linearly increase the amount of blurring until the eccentricity angle of 30 degrees where the amount of blur is $\sigma = 6.8$ pixels (resolution 1280×1440). For performance reasons the blur was implemented in two separated passes. In our experiments, we decided to use linear falloff over visual acuity function with the parameters, because it gave more intuitive control over the parameters.

4. Experiments

This section describes the different user tests we ran with the Visual-Polar rendering. The results of each experiment are listed at the end of each subsection. The purpose of the experiments was to test different parameters for the foveation methods and to compare them with each other.

(a) *Screen space BMFR with full resolution* **(b)** *Screen space BMFR with the same number of samples as in Visual-Polar* **(c)** *Visual-Polar BMFR, with the gaze point in the middle of the left square* **(d)** *Visual-Polar BMFR, with the gaze point in the middle of the right square*

(e) *Screen space SVGF with full resolution* **(f)** *Screen space SVGF with the same number of samples as in Visual-Polar* **(g)** *Visual-Polar SVGF, with the gaze point in the middle of the left square* **(h)** *Visual-Polar SVGF, with the gaze point in the middle of the right square*

Figure 5: *Example on how the denoisers can preserve reflections and details better when they are applied in the Visual-Polar space.*

4.1. Test Set-Up

In the experiments we used the FOVE 0 HMD which is equipped with an eye tracker. The computer used in the experiments has a single AMD Vega Frontier Edition GPU, an Intel Core i7-6700K CPU, and 32GB of memory.

Even though the display of the FOVE 0 has only 1280×1440 pixels per eye, the driver requires a 1792×2016 frame per eye because it performs warping automatically. Our test system renders the actual resolution of the display for one eye and shows the same image for both eyes. In other words, we do not have binocular disparity. In addition, upsampling and warping our 1280×1440 frame to the internal representation of the driver introduces some minor sampling artifacts to the results. We decided to render the actual resolution for just one viewpoint because it reduces the required path tracing and denoising by 74%, which allowed us to use more complicated scenes even without dedicated ray tracing hardware acceleration.

The eye tracking accuracy of the system is roughly ± 1 degrees [WRHS18b]. The minimum latency of the system is at least 38 ms from eye movement to rendered pixels [KIV*18]. The users of a foveated rendering system can tolerate up to 70 ms end to end latency [APLK17].

(a) *Music room* **(b)** *Gray white room*

(c) *Dining room* **(d)** *Breakfast room*

(e) *Staircase* **(f)** *Pabellon*

Figure 6: *Example views of typical directions the participants decided to look at in the sampling experiment.*

4.2. Sampling Experiment

The purpose of this test was to compare different parameter sets of the proposed method to a full screen space resolution and this way find good parameters to use in the latter path tracing experiment. As the used hardware cannot path trace and denoise the full screen space resolution of 1280×1440 with a reasonable frame rate, the test was done by sampling prerendered converged omnidirectional images. Prerendering was done by path tracing equirectangular images. The images were fully converged so only the sample location changes might cause temporal artifacts. These artifacts are similar to the artifacts that the denoisers are generating, since they discard temporal data projected from other objects. For this experiment we removed the translation of the camera, because it would have required either many equirectangular images or reprojection and filling with depth information. For the full resolution screen space rendering, the equirectangular images were sampled in real-time with the full Cartesian resolution of the HMD. For the proposed method we sampled the prerendered images from the same positions from which path tracing with that distribution would have sampled the world.

4.2.1. Procedure

In the user study we showed the participants a pair of two renderings of the same scene and their task was to compare the quality of them. In case different quality was experienced, the users were asked to choose which one of the renderings was better. In every pair one of the renderings was a full resolution Cartesian rendering. The other one was the result of a randomly picked parameter set of the proposed method. The order of the two renderings in the pair was randomly selected.

The parameter sets in the study where different sample reductions and different blurring parameters. These sets where found out by testing the system with the authors first during the development. After the trial runs, some participants noticed that one of the renderings in the pairs was always the full resolution Cartesian rendering. To address this, we added random pairs of two foveated renderings of the same method. These random pairs were not used in the results. The idea was to get the users to focus on the quality and not to find ways to "cheat the test" by finding out if the rendering reacted to eye-tracking by quickly moving their eyes back and forth multiple times.

We used six different scenes, which are shown in Figure 6. As can be seen in the images, we included many hard cases where the scene contains patterns that are intentionally almost checkerboard patterns. The test lasted for approximately 40 minutes per test subject. In the experiment we had eight participants all with normal or corrected to normal vision.

4.2.2. Results

The answer distribution of the questionnaire of the sampling experiment can be seen in Figure 7. The names of the methods tell how much path tracing and denoising work was reduced. For example, 61% is achieved by reducing both the the width and the height of the resolution by 37%. The answers show that we can reduce the rendered pixels by approximately 60-70%, after which the users start to see too many artifacts. A possible cause for the lack of "Better than reference" answers in the 61% reduction option is that different blur settings work the best with different reductions. According to paired t-tests, the 88% reduction showed significant difference in users' responses compared with both of the smaller reductions (p-value < 0.05). Between 61% and 71% reductions, no significant difference was found. Based on these results, in the second test described below, we decided to use 61% reduction since we wanted to be conservative with our parameter choices.

It is also important to note that this test did not utilize temporal reprojection of samples. In other words, there were different temporal artifacts, caused by the moving gaze point and moving HMD, compared to results where a temporal-aware denoiser is used. Therefore, the results should not be treated as absolute numbers but only compared to each other. However, the results are very

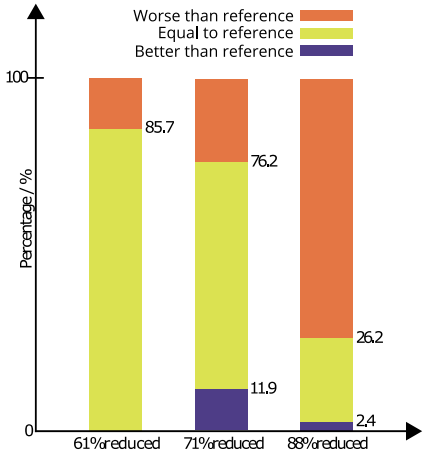
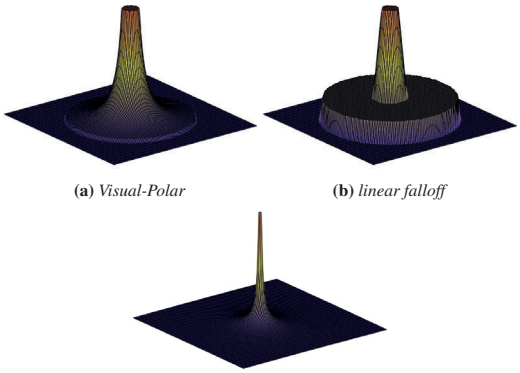


Figure 7: *Answer distribution of the sampling experiment with the best performing blurring settings for every reduction.*



(a) *Visual-Polar* **(b)** *linear falloff*

(c) *log-polar*

Figure 8: *Sample distribution around the gaze point with different methods in the path tracing experiment. Linear falloff requires a lot more samples in the periphery and, therefore, we cannot afford to give it as many samples in the fovea with our real-time budget. In reality the spike of the log-polar distribution is orders of magnitude higher than the highest values in other distributions.*

similar and this is the closest imposter of 1 spp denoised full Cartesian resolution we can run in real time. The main idea of this experiment was to get blurring and sample count reduction parameters for the real-time path tracing where we have temporal reprojection in the denoisers.

One of the participants marked every single time the foveated rendering as worse than Cartesian rendering which did not help with relative comparisons of the tested methods. In other words, this participant thought that the renderings are the same 0% of the time, and in contrast, the other participants thought 86% of the time that the renderings are the same. Since the behavior was a clear outlier we concluded that the eye tracking did not work properly for this one user and removed him/her from the results. Different eye-tracking behavior could be caused by astigmatism this participant had in his/her other eye.

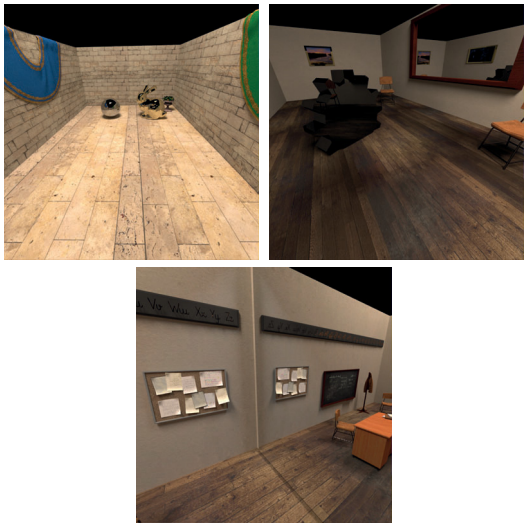
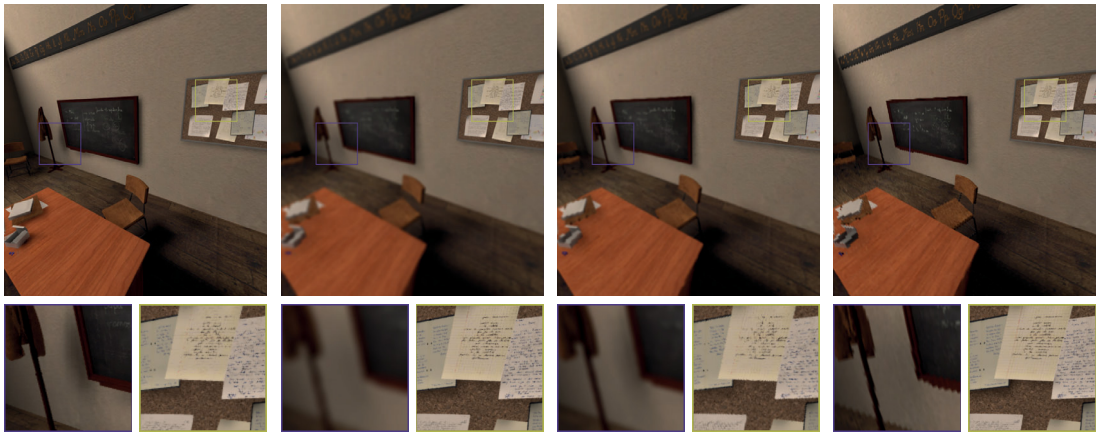


Figure 9: *Three scenes used in the path tracing experiment.*

4.3. Real-Time Path Tracing Experiment

The purpose of the second experiment was to compare different sample distributions in a path tracing scenario with with the same full resolution 1280×1440 as in the previous user study, but with a number of samples per pixel that can be rendered in real-time. In other words, we had to leave full resolution screen space rendering out of this test. The compared sample distributions were 1) uniform screen space, 2) linear falloff as described in [WRK*16], 3) log-polar [MDZV18], and 4) the proposed Visual-Polar distribution. The sample distribution around the gaze point of each of the foveated methods can be seen in Figure 8.

In the uniform distribution case, we used the same number of rendered pixels as in the proposed method because it started to reach the limit of how many path tracing samples we can produce in real-time on the test machine. In the linear falloff we used a sampling probability of 20% in the peripheral parts, fovea radius of 10 eccentricity degrees and periphery starting at 20 eccentricity degrees. This means it has a larger fovea and a higher sampling probability in the periphery. For this reason, we gave it $1.5\times$ more samples compared to the Visual-Polar method. To make the comparison fairer we also added blurring to the linear falloff method. In log-polar we rendered directly in log-polar space and used the parameter setup ($\sigma = 1.8$ $\alpha = 4.0$) described by the original paper [MDZV18]. To make the comparison fair we rendered the same



(a) *Screen space* **(b)** *Visual-Polar* **(c)** *Linear falloff* **(d)** *Log-polar*

Figure 10: *Different methods tested in the path tracing experiment.*

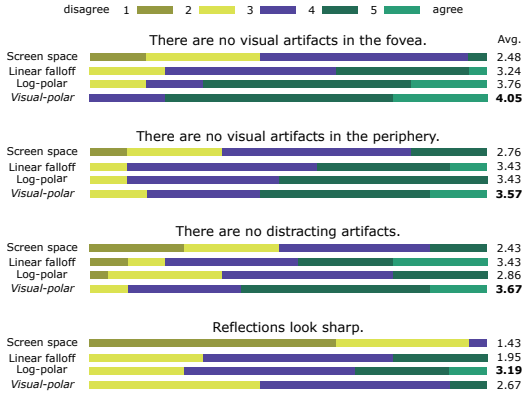


Figure 11: *Answer distribution of the path tracing user study.*

number of pixels with the proposed method as in the log-polar method. In this experiment we used BMFR with all the methods because it is faster than SVGF. Using just one denoiser reduced the number of parameter permutations in the test making it shorter.

4.3.1. Procedure

In the user study, we rendered a scene with a random rendering method and asked the participants to rate on a five-point Likert scale how much they agree with the following statements.

1. There are no visual artifacts in the fovea.
2. There are no visual artifacts in the periphery.
3. There are no distracting artifacts.
4. Reflections look sharp.

Table 1: *P-values from Mann-Whitney U test for every question in the real-time path tracing experiment. The comparison is against the proposed method on every row.*

Comparison	1	2	3	4
Cartesian	**8.2e-07**	**0.0093**	**3.8e-04**	**3.6e-06**
Linear falloff	**0.0023**	0.53	0.59	**0.0049**
Log-polar	0.41	0.58	**0.0059**	**0.060**

In question (1) the users were asked to assess the quality at their gaze point. The idea of question (2) was to measure both temporal artifacts and tunnel vision. In question (3) both the fovea and the periphery were considered. Finally we added question (4), because we wanted to measure the better quality in the reflections visualized in Figure 5.

In total we had three scenes which are seen in Figure 9 and four different rendering methods shown in Figure 10. The scenes were chosen so that they contain different kinds of reflective objects and not too much geometry, just to make sure that without dedicated ray tracing hardware the ray traversal is not the bottleneck. The test took around 35 minutes to complete depending on how quickly the participant decided their rating. In the experiment we had seven participants all with normal or corrected to normal vision.

4.3.2. Results

The answer distribution of the path tracing experiment can be seen in Figure 11. The proposed method has the best average answer in all other questions than the reflection sharpness related question (4). In that question log-polar is better because it packs so many samples to the center of the gaze point as can be seen in Figure 8. Therefore, it is able to generate very sharp reflections on small area around the gaze point. Sampling the same number of samples with

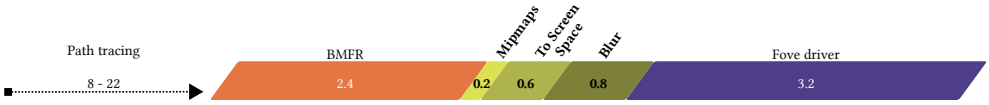


Figure 12: *The latency of different pipeline stages (ms) for screen space resolution of* 1280×1440 *and Visual-Polar space resolution of* 853×960*. Both path tracing and denoising (BMFR) are* $2.5\times$ *faster due to reduced resolution in the Visual-Polar space. Yellow parts are the ones added by the proposed method.*

the regular grid in screen space is clearly the worst of all the benchmarked methods. It also got bad ratings on the question (2) about the peripheral quality even though it has more samples in the periphery than other methods. The poor peripheral rating is likely caused by the poor resolution in the fovea, which made the users think that the rendering is overall bad.

We also performed paired Mann-Whitney U tests for the results between the proposed method and the other tested methods. The p-values for the statistical significance tests are shown in Table 1. Significant difference was found for each question between the proposed method and the same resolution Cartesian. Compared with Linear falloff, significant difference was found in questions (1) and (4). Significant difference with Log-polar was found in questions (3) and (4).

5. Execution Time

The execution time breakdown of our Visual-Polar rendering pipeline with the test machine can be seen in Figure 12. On our single consumer GPU computer without dedicated ray tracing hardware it takes 8-22 ms to path trace the scene at 1 spp in the Visual-Polar space resolution of 853×960, which maps to a 1280×1440 screen space resolution. The path tracing execution time varies a lot because it is heavily dependent on the scene. The Visual-Polar space saves 61% of the path tracing execution time. In other words, path tracing full screen space resolution of 1280×1440 would take around 20-56 ms, which is too much for the real-time frame budget.

The denoising takes only 2.4 ms in the Visual-Polar space, and the space also saves 61% of the denoising work. The extra steps added by the proposed method on top of path tracing are generating mipmaps (0.2 ms), mapping to the screen space (0.6 ms) and blurring (0.8 ms). These execution times are measured with our unoptimized OpenCL code. All in all, the Visual-Polar space saves 16-38 ms on path tracing and denoising while adding only about 1.6 ms.

6. Ray Coherence

In addition to the sample reduction, the Visual-Polar space also speeds up ray traversal because the primary rays are coherent and the SIMT/SIMD lanes have high utilization. On our test hardware if we ray trace coherent rays by completely randomizing the starting points as in [KIV*18] the ray traversal is approximately $1.5\times$ slower. This affects the ray traversal, which is typically 20% of the path tracing execution time on our test machine. In addition, if collisions are not prevented, there would be an additional cost of summing the samples using atomic operations and handling the unsampled locations.

On the other hand, a typical way to do linear falloff defined by sampling probability is to launch path tracing with full resolution and kill paths randomly in periphery. With random killing the ray traversal for the same number of samples is approximately $1.3\times$ slower compared to the Visual-Polar space.

7. Limitations and Future Work

We evaluated using contrast enhancement [PSK*16] both after the blur pipeline stage and as a replacement of it. However, we were not able to find parameters that would not make the users see less artifacts in the periphery. This was likely caused by the fact that we perform TAA [Kar14] in the Visual-Polar space and not after the contrast enhancer. We also tested applying TAA after mapping to screen space, but without modifications, it jitters the sample locations too much in the fovea and too little in the periphery. As a future work, it would be interesting to design a dedicated TAA method for Visual-Polar space rendering, which is applied after mapping to the screen space and therefore can be run after the contrast enhancer.

One drawback of Visual-Polar space is that adding fine-grained screen space sampling strategies, as in [SGEM16], would require sparse sampling with increased resolution. Sparse sampling was the thing we wanted to get rid of in the first place to fully utilize the SIMD/SIMT hardware.

Distortion correction (the counter operation of the HMD lenses) of the FOVE 0 is a black box to us, which we cannot disable. We provide the driver with an image rendered to a plane. In the future, it would be interesting to generate primary rays in the already warped Visual-Polar space, potentially further reducing the sampling requirement and eliminating the need for a separate distortion correction step. This would require an HMD driver which allows directly displaying the rendered pixels on the screen of the device.

The Visual-Polar space execution time could be further optimized by not rendering and denoising areas on the far edges of the vision, which are clipped by the viewport of the HMD. For this implementation the denoisers should handle these areas so that the outside data does not bleed into the viewport area.

8. Conclusions

We proposed the Visual-Polar space, which produces visually pleasing foveated real-time path tracing. The Visual-Polar space coordinates are a modification of the polar coordinates, so that the

sample distribution follows the visual acuity model of the human visual system. The wanted distribution is achieved by scaling and cutting the polar space. Compared to the previous work on polar foveated rendering our sample distribution follows human eye resolution more closely. This leads to getting more out from the very limited real-time path tracing sample budget.

Another key benefit is the ability to do both rendering and denoising in the Visual-Polar space making both of the stages 2.5× faster. In addition, the primary rays are coherent, which contributes to improved SIMD/SIMT hardware utilization, ending up with a $1.3 - 1.5\times$ faster ray traversal. We proposed the first system that directly path traces in a modified polar space and demonstrates that also denoising can be done directly in the space making both faster.

The Visual-Polar foveated path tracing was evaluated in two user studies. In the first study we compared different parameters for the proposed sampling pattern. In the second user study we compared the proposed method with other foveated path tracing sampling patterns. The proposed method had the best average answer in all artifact related questions.

To the best of our knowledge, our system is the first one that can do path traced and denoised foveated rendering in real-time for contemporary VR headset resolutions.

Acknowledgements

The authors would like to thank anonymous reviewers and Julius Ikkala for fruitful comments. In addition, the authors are thankful to user test participants and model makers: Christophe Seux for Class room (CC0), Stanford 3D scanning repository for Stanford Bunny, Frank Meinl for Crytek Sponza (CC-BY), Wig42 for Music room (CC-BY), Gray white room (CC-BY), Dining room (CC-BY), Breakfast room (CC-BY), Staircase (CC-BY), and Hamza Cheggour for Pabellon (CC-BY).

The work was financially supported by the Tampere University (of Technology) Graduate School, Emil Aaltonen Foundation, Finnish Foundation for Technology Promotion, Nokia Foundation, Business Finland (funding decision 40142/14, FiDiPro-StreamPro), Academy of Finland (funding decisions 297548, 310411) and ECSEL JU project FitOptiVis (project number 783162).

References

[APLK17] ALBERT, RACHEL, PATNEY, ANJUL, LUEBKE, DAVID, and KIM, JOOHWAN. "Latency requirements for foveated rendering in virtual reality". *Transactions on Applied Perception* 14.4 (2017) 6.

[Bar18] BARRÉ-BRISEBOIS, COLIN. *Game Ray Tracing: State-of-the-Art and Open Problems*. High Performance Graphics Keynote. 2018 2, 3.

[BRM*16] BITTERLI, BENEDIKT, ROUSSELLE, FABRICE, MOON, BOCHANG, et al. "Nonlinearly Weighted First-order Regression for Denoising Monte Carlo Renderings". *Computer Graphics Forum*. Vol. 35. 4. 2016 2.

[Dev86] DEVROYE, LUC. *Non-Uniform Random Variate Generation*. Springer, 1986 4.

[GFD*12] GUENTER, BRIAN, FINCH, MARK, DRUCKER, STEVEN, et al. "Foveated 3D graphics". *Transactions on Graphics* 31.6 (2012) 2, 3.

[HEMS10] HERZOG, ROBERT, EISEMANN, ELMAR, MYSZKOWSKI, KAROL, and SEIDEL, H-P. "Spatio-temporal upsampling on the GPU". *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*. 2010 1.

[HGF14] HE, YONG, GU, YAN, and FATAHALIAN, KAYVON. "Extending the graphics pipeline with adaptive, multi-rate shading". *Transactions on Graphics* 33.4 (2014) 1.

[Kaj86] KAJIYA, JAMES. "The rendering equation". *ACM Siggraph Computer Graphics*. Vol. 20. 4. 1986 2.

[Kar14] KARIS, BRIAN. "High-quality Temporal Supersampling". *ACM SIGGRAPH 2014, Advances in Real-Time Rendering in Games*. 2014 10.

[KIM*19] KOSKELA, MATIAS, IMMONEN, KALLE, MÄKITALO, MARKKU, et al. "Blockwise Multi-Order Feature Regression for Real-Time Path Tracing Reconstruction". *accepted to Transactions on Graphics* (2019) 2, 3.

[KIV*18] KOSKELA, MATIAS, IMMONEN, KALLE, VIITANEN, TIMO, et al. "Instantaneous foveated preview for progressive Monte Carlo rendering". *Computational Visual Media* (2018) 3, 4, 6, 10.

[KMSB18] KILGARIFF, EMMETT, MORETON, HENRY, STAM, NICK, and BELL, BRANDON. *NVIDIA Turing Architecture In-Depth*. https://devblogs.nvidia.com/nvidia-turing-architecture-in-depth/ accessed 8th of January 2019. 2018 2, 3.

[KVJT16] KOSKELA, MATIAS, VIITANEN, TIMO, JÄÄSKELÄINEN, PEKKA, and TAKALA, JARMO. "Foveated path tracing". *Proceedings of International Symposium on Visual Computing*. 2016 2, 3.

[LW90] LEVOY, MARC and WHITAKER, ROSS. "Gaze-directed volume rendering". *ACM SIGGRAPH Computer Graphics*. Vol. 24. 2. 1990 2.

[MDZV18] MENG, XIAOXU, DU, RUOFEI, ZWICKER, MATTHIAS, and VARSHNEY, AMITABH. "Kernel Foveated Rendering". *Proceedings of the ACM on Computer Graphics and Interactive Techniques* 1.1 (2018) 3, 8.

[PSK*16] PATNEY, ANJUL, SALVI, MARCO, KIM, JOOHWAN, et al. "Towards foveated rendering for gaze-tracked virtual reality". *Transactions on Graphics* 35.6 (2016) 2, 10.

[Red97] REDDY, MARTIN. "Perceptually modulated level of detail for virtual environments". PhD thesis. College of Science and Engineering, University of Edinburgh, 1997 2.

[RFS18] RITSCHEL, TOBIAS, FRISTON, SEBASTIAN, and STEED, ANTHONY. "Perceptual Rasterization for Head-mounted Display Image Synthesis". *arXiv preprint arXiv:1806.05385* (2018) 3.

[RWH*17] ROTH, THORSTEN, WEIER, MARTIN, HINKENJANN, ANDRÉ, et al. "A Quality-Centered Analysis of Eye Tracking Data in Foveated Rendering". *Journal of Eye Movement Research* 10.5 (2017) 5.

[SCMP19] SIEKAWA, ADAM, CHWESIUK, MICHAŁ, MANTIUK, RADOSŁAW, and PIÓRKOWSKI, RAFAŁ. "Foveated Ray Tracing for VR Headsets". *International Conference on Multimedia Modeling*. 2019 3.

[SGEM16] STENGEL, MICHAEL, GROGORICK, STEVE, EISEMANN, MARTIN, and MAGNOR, MARCUS. "Adaptive image-space sampling for gaze-contingent real-time rendering". *Computer Graphics Forum*. Vol. 35. 4. 2016 2, 10.

[SKW*17] SCHIED, CHRISTOPH, KAPLANYAN, ANTON, WYMAN, CHRIS, et al. "Spatiotemporal variance-guided filtering: real-time reconstruction for path-traced global illumination". *Proceedings of High Performance Graphics*. 2017 2, 3.

[SPD18] SCHIED, CHRISTOPH, PETERS, CHRISTOPH, and DACHSBACHER, CARSTEN. "Gradient Estimation for Real-Time Adaptive Temporal Filtering". *Proceedings of the Computer Graphics and Interactive Techniques* 1.2 (2018) 2.

[SRJ11] STRASBURGER, HANS, RENTSCHLER, INGO, and JÜTTNER, MARTIN. "Peripheral vision and pattern recognition: A review". *Journal of vision* 11.5 (2011) 2.

[VST*14] VAIDYANATHAN, KARTHIK, SALVI, MARCO, TOTH, ROBERT, et al. "Coarse pixel shading". *Proceedings of High Performance Graphics*. 2014 1.

[WRHS18a] WEIER, MARTIN, ROTH, THORSTEN, HINKENJANN, ANDRÉ, and SLUSALLEK, PHILIPP. "Foveated depth-of-field filtering in head-mounted displays". *Transactions on Applied Perception* 15.4 (2018) 3.

[WRHS18b] WEIER, MARTIN, ROTH, THORSTEN, HINKENJANN, ANDRÉ, and SLUSALLEK, PHILIPP. "Predicting the gaze depth in head-mounted displays using multiple feature regression". *Proceedings of the Symposium on Eye Tracking Research & Applications*. 2018 6.

[WRK*16] WEIER, MARTIN, ROTH, THORSTEN, KRUIJFF, ERNST, et al. "Foveated Real-Time Ray Tracing for Head-Mounted Displays". *Computer Graphics Forum* 35.7 (2016) 3, 8.

[WSR*17] WEIER, MARTIN, STENGEL, MICHAEL, ROTH, THORSTEN, et al. "Perception-driven Accelerated Rendering". *Computer Graphics Forum*. Vol. 36. 2. 2017 2.

[WWHW97] WATSON, BENJAMIN, WALKER, NEFF, HODGES, LARRY F, and WORDEN, AILEEN. "Managing level of detail through peripheral degradation: Effects on search performance with a head-mounted display". *Transactions on Computer-Human Interaction* 4.4 (1997) 2.

[XLV18] XIAO, KAI, LIKTOR, GABOR, and VAIDYANATHAN, KARTHIK. "Coarse pixel shading with temporal supersampling". *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*. 2018 1.