TU WIEN Informatics

Spatio-Temporal Filtering for Real-Time Path Tracing in Virtual Reality

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Visual Computing

eingereicht von

Lukas Gersthofer, BSc
Matrikelnummer 01325669

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Privatdoz. Mag.rer.nat. Dr.techn. Hannes Kaufmann
Mitwirkung: Mag. Dr.techn. Peter Kán

Wien, 5. März 2020

Lukas Gersthofer	Hannes Kaufmann

TU WIEN Informatics

Spatio-Temporal Filtering for Real-Time Path Tracing in Virtual Reality

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Visual Computing

by

Lukas Gersthofer, BSc
Registration Number 01325669

to the Faculty of Informatics

at the TU Wien

Advisor: Privatdoz. Mag.rer.nat. Dr.techn. Hannes Kaufmann
Assistance: Mag. Dr.techn. Peter Kán

Vienna, 5th March, 2020

Lukas Gersthofer	Hannes Kaufmann

Erklärung zur Verfassung der Arbeit

Lukas Gersthofer, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 5. März 2020

Lukas Gersthofer

Acknowledgements

I would like to express my gratitude to my supervisors Hannes Kaufmann and Peter Kán. Thank you very much for all the support and flexibility you provided me with. It helped me find my own path through this thesis.

A special thanks goes to Hiroyuki Sakai, Johannes Unterguggenberger and Bernhard Kerbl for having interesting and idea-giving discussions which provided me with new and exciting perspectives. I would also like to thank Bernhard Steiner for his helpful input regarding writing of this thesis.

Furthermore, my thanks goes to the faculty of informatics for granting me a funding scholarship to finance the necessary hardware.

I owe my deepest gratitude to my loving parents who always cared for my education. Without your steady support and motivation this journey would have been a thousand miles longer.

Last but not least, I am indebted to my girlfriend who constantly countered my complaining with disarming arguments giving me hope and motivation.

Kurzfassung

Die Einführung von NVIDIAs Turing Architektur kombiniert mit der Erweiterung der Grafikschnittstelle DirectX erlaubte erstmals den Einsatz von Echtzeit-Strahlenverfolgung in grafischen Echtzeit-Anwendungen. Gleichzeitig gibt es Grafikentwicklern die Möglichkeit globale Effekte wie Schatten oder Reflexionen physikalisch korrekt und trotzdem dynamisch zu berechnen. Die Integration in populäre Game-Engines wie *Unreal Engine*[1] und *Unity*[2] stellt diese neuen Funktionen einer großen Menge an Spiele- und Grafikentwicklern zur Verfügung.
Das Eintauchen in die virtuelle Realität (VR) mit Hilfe von geeigneten Systemen erfreute sich nicht nur großer Beliebtheit in der Entertainment Branche, sondern fand auch mehr und mehr Einsatz in medizinischen sowie Lehr- und Trainingsanwendungen. Auf besonderen Wert sollte hier nicht nur auf die geometrische Komplexität der virtuellen Szene, sondern auch auf deren physikalisch plausible Beleuchtung gelegt werden. Daher eignet sich der Einsatz der Echtzeit-Strahlenverfolgung in VR besonders um qualitativ hochwertige Beleuchtungsmodelle umsetzen zu können, die auch auf dynamische Änderungen der Beleuchtungssituation reagieren können. Allerdings muss dabei beachtet werden, dass der Einsatz dieser neuen und noch limitierten Technologie zu zusätzlichem Berechnungsaufwand führt, der in VR Applikationen bereits höher ist als bei Desktop Anwendungen.
Diese Arbeit beschäftigt sich mit der Integration der Lichtransportmethode *Pathtracing* in einem hybriden Ansatz. Dabei soll zunächst der erste Strahl jedes Pixels rasterisiert werden und anschließend eine geringe Anzahl an indirekten Strahlen verfolgt werden. Das daraus resultierende Bild enthält einen hohen Grad an Bildrauschen, das durch zeitliche Akkumulation und Filterung von benachbarten Pixel bereinigt werden soll. Zusätzlich wird eine Änderung in der Ermittlung von Varianz angedacht, die zur gänzlichen Bereinigung von Bildrauschen notwendig ist. Schließlich wird ein neues System zur Reduzierung der Anzahl an verfolgten Strahlen vorgeschlagen um das notwendige Zeitbugdet pro Bild einhalten zu können.

[1] `https://www.unrealengine.com/`
[2] `https://unity.com/`

Abstract

Ray tracing has raised its importance for real-time rendering since the presentation of NVIDIAs Turing architecture and Microsofts extension DirectX Raytracing (DXR). It allows for computing physically correct renderings dynamically for various global lighting effects such as shadows or reflection for example. Integration into major game engines such as *Unreal Engine*[3] and *Unity*[4] made these features quickly accessible to a wide range of game developers and graphics programmers.
At the same time, consumer-level virtual reality (VR) systems gained more attention not just in the entertainment industry but also for medical, educational and training purposes. Perception of virtual scenes depends not only on geometric complexity but also heavily on physically plausible lighting. Hence, the combination of aforementioned real-time ray tracing with VR to produce visually more plausible lighting scenarios and increase immersion seems natural. Unfortunately, rendering for VR systems requires more computational power than ordinary graphics applications for desktop systems and real-time ray tracing is still limited in its extent.
This thesis investigates if the light transport method - path tracing - is suitable to be used in a virtual reality setup in order to produce higher quality dynamic lighting. Therefore, a hybrid rendering pipeline is proposed combining path tracing with a low number of samples per pixel and spatio-temporal filtering of the noisy path tracer output. The pipeline comprises rasterization of first hits, tracing of a low amount of indirect rays per pixel, temporal accumulation of path tracing samples and, finally, de-noising of the highly noisy path traced image. In addition, this thesis introduces a novel masking system which provides a trade-off between the number of traced rays and the visual quality of dynamic objects. Furthermore, an improvement to the variance estimation of the chosen de-noising approach is proposed.
A prototype shows that path tracing together with de-noising can achieve interactive frame rates suitable for VR applications. Resulting quality is comparable to reference renderings. The proposed masking system allows for further tuning of performance without introducing noticeable artifacts, making this approach viable for a large range of hardware setups.

[3] `https://www.unrealengine.com/`
[4] `https://unity.com/`

Contents

CHAPTER 1

Introduction

Since the popular upcoming of consumer-level virtual reality (VR) systems, resolution and immersion factor of these setups have steadily increased with each new generation of head-mounted displays. This development led to an increased demand of high end graphics cards and algorithms designed for stereo rendering to keep them running at their default refresh rate of 90Hz (i.e. 11ms per frame). Additionally, higher quality rendering, i.e. better reconstruction of the real world in terms of surface materials and lighting, was desired to further improve immersion.

With recent advances in various game engines [1,2], almost photo-realistic rendering is possible with decent hardware. This is due to outsourcing many computations from runtime to a precomputation step, requiring specific methods to make use of this statically baked information [SL17, O'D18]. Unfortunately, most of these illusions break when viewed in VR, e.g. a precomputed reflection on an object does neither resemble the actual surroundings perfectly nor include dynamic objects. Furthermore, each visual effect needs its own special algorithm, making the whole rendering system complex and bulky. It also requires developers to actually implement a big variety of graphics algorithms and to fit them in their engines. Clearly, this can lead to long development times and is error prone.

Path tracing [Kaj86] is an approach to simulate light transport for rendering up to a degree, such that it is almost indistinguishable from images taken with a real camera. Moreover, path tracing achieves global visual effects like shadows or global illumination for free in terms of additional development effort. However, time to converge to noise-free images is far away from the tiny time budget available for real-time rendering. Usually, convergence of the Monte Carlo integration used in path tracing requires a few hundred to a few thousand samples per pixel. The observation, that even noisy images generated with a few stochastic samples contain the most important visual features led to the development

[1] https://www.unrealengine.com/

[2] https://unity.com/

of filtering techniques to remove or at least reduce the remaining noise by a substantial amount. Recent methods allow for almost photo-realistic renderings in real-time by filtering noisy path traced images over the spatial and temporal domain. For dynamic real-time applications such as demos, games or VR applications, temporal filtering is necessary to mitigate flickering between frames due to consistent movement of the camera.

Path tracing gives a powerful and flexible tool to achieve dynamism in scenes, whether it deals with shadows, reflections or indirect illumination. Although simple to implement, it cannot be used as a stand-alone solution due to the high time budget needed to compute noise-free images. Luckily, it is based on an iterative algorithm such that it can be stopped at any iteration, i.e. at any number of samples per pixel. This gives the opportunity to apply filtering to remove remaining noise and reveal an image that is very close to the ground truth.

1.1 Approach

In the course of this thesis, a prototype was developed in order to see how far we can go towards realistic and dynamic lighting with path tracing in real-time environments, especially in a computational challenging one - virtual reality. Hence not only high performance solutions to the individual problems were necessary but also new approaches for reduction of computational complexity. In the end, the developed application was capable of handling complex scenes with simple to moderate lighting situations dynamically and in real-time.

Two major challenges arise when path tracing is used as light transport method in real-time. First, tracing only a small amount of rays causes the image to be extremely noisy. Therefore, it is necessary to focus computations on the most significant rays and reach for better approximations through clever sampling strategies. However, even getting the most out of the limited number of rays traced is not enough to completely remove noise from the image, thus additional measures are necessary. The second challenge consists of dealing with resulting noise in the image. By utilizing spatial and temporal information noise levels in the image may be reduced below a perceivable threshold without introducing additional visual artifacts.

Our basic pipeline consists of 4 steps:

1. Rasterization of primary rays
2. Path tracing of a low number of indirect rays per pixel
3. Temporal sample accumulation, i.e. sample reprojection
4. Spatial sample accumulation, i.e. filtering/de-noising

First of all, primary rays, i.e. rays emanating from the camera, are rasterized instead of ray traced. Even though ray tracing operations are hardware accelerated on certain GPUs, rasterization on current hardware is still faster, if it can be applied. Additionally, this gives the opportunity to cheaply capture a noise-free representation of various per-pixel features which would be harder to obtain by using pure ray tracing. Examples include not only local information like position, normal or material parameters but also quantities defined only in screen space, e.g. derivatives of local features. The additional data acquired in this stage is later used for validation during sample accumulation.
For each valid pixel a number of rays is traced through the scene to estimate incoming light. Those indirect rays are sampled in a way, such that they represent a valuable contribution to the approximation of global illumination. Sampling is controlled via importance sampling of lights and bidirectional reflectance distribution functions, as well as correlation between samples to further reduce the resulting noise despite the low number of rays traced. In this work, only indirect rays corresponding to the first bounce were traced in order to keep the frame rate high enough for virtual reality. However, for a closer estimate or if future hardware is capable of it, more bounces can be taken into account.
To further balance performance and visual quality, a new masking system is proposed to reduce the number of rays traced per frame by introducing slight visual artifacts visible only under certain circumstances. A spatial mask determines which pixels are traced in the current frame. Missing information is filled in during later stages of filtering. The mask follows a pattern inspired by blue noise, i.e. noise that only contains high frequencies, which is not as apparent to the human eye as regular patterns.
Although a clever sampling strategy drastically decreases noise in the final image, it does not completely remove it. To effectively increase the amount of samples per pixel, information gathered over the temporal as well as the spatial domain can be exploited. Temporal accumulation got popular among the games industry via *Temporal Anti-Aliasing* [Kar14] acting as a cheap method for supersampling. In essence, pixels are accumulated over a certain period of time while considering movement of camera and objects and their resulting occlusions. This may directly increase the effective number of samples for temporally coherent scenes but at the cost of additional blurring or ghosting artifacts. Nonetheless, at the moment this proves to be vital for real-time de-noising of stochastically sampled domains.
Eventually, a majority of the samples exhibit spatial coherency which is advantageous when accumulating samples. Spatial accumulation can be thought of as a filter over a certain spatial domain, e.g. screen space or path space, comparing various features of samples and finally accumulating those which seem to be similar enough according to some metric. Screen space filters based on the Joint/Cross bilateral filter proved to provide a good trade-off between computational cost and visual quality in real-time de-noising. We employ a hierarchical à-trous wavelet transform using specific weight functions to prevent overblurring of edges based on geometric and visual properties.

1.2 Contributions

The contributions of this thesis can be summarized as follows:

- A hybrid rendering pipeline usable for ordinary desktop as well as for virtual reality applications.
- A novel masking system which reduces the number of rays traced per frame and thus it is increasing performance by a substantial amount. The emerging visual artifacts are mitigated by the spatio-temporal filter.
- Special optimizations for virtual reality applications by sharing samples between the stereoscopic views.
- An improvement to the robustness of variance estimation of a recently published work related to this thesis.

1.3 Outline

This thesis is structured in a way to first give a joint overview of the theoretical concepts and the current state-of-the-art in real-time path tracing in Chapter 2.
Chapter 3 covers the methodological approach for this work and gives more details on the various stages of the developed pipeline. Furthermore, the previously mentioned contributions of this thesis are described in detail.
Technical details of the implementation can be found in Chapter 4.
Eventually, the developed application is benchmarked with a variety of scene configurations and results are presented visually and quantitatively in Chapter 5.
Chapter 6 provides a short summary and drawn conclusions from this thesis. Major contributions and limitations of this work are recapitulated. Furthermore, a possible outlook for the future of this work and the area of real-time path tracing is given.

CHAPTER 2

Background and Related Work

Physically based light transport in virtual environments is a well-researched area. Its applications are crucial to a number of industries, e.g. movies, games and architecture. Until recently, light transport was mainly simulated offline. Due to the recent advent of graphics cards, supporting hardware accelerated tracing and intersection testing of rays, a number of visual algorithms have been revised and their - mostly overly complicated and error prone - rasterization counter parts are progressively substituted. This chapter provides an overview of how real-time ray tracing can replace current rasterization-based approximations for physically based light transport. However, on current hardware it is not possible to solely rely on ray tracing, instead it turns out that a combination of rasterization and ray tracing - called hybrid rendering - yields a decent balance between visual quality, physical correctness and performance.

2.1 Light Transport

The field of light transport simulations is concerned with predicting and calculating how light moves through space and interacts with objects in a physically plausible manner. This is not trivial since light is known to be an electromagnetic radiation behaving like a wave and a particle at the same time. Nonetheless, the particle character dominates as long as it interacts with objects much more than the wavelength of light, hence photons are merely described as the path they take, so-called rays.
Before introducing the basics of light transport used in computer graphics, it is necessary to describe the actual unit used to measure emitted, received or reflected light - *radiance*. Radiance denotes the amount of energy that is carried by a ray of light, when interacting with a surface. It accounts for normalization of projected area such that it corresponds to an infinitely thin ray and is further normalized by the incident angle to account for a decrease of energy when incident at a flat angle.

Tracing light rays through the pixels of an image was first described by Appel [App68] and further improved by Whitted [Whi79] to, recursively, also account for reflective and refractive surfaces. However, paths were always considered to be reflected or refracted on an idealized surface, e.g. a perfect mirror, into exactly one direction. In practice, surfaces might be rough and disperse incoming light in a range of directions depending on the surface material. In a further improvement Kajiya [Kaj86] and Immel et al. [ICG86] simultaneously developed a formulation to account for all incoming light contributing to the shading of a surface point instead of only relying on a single direction. This is called the rendering equation and is given in a simplified version as:

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

where

- $\mathbf{x}$ is a specific location in space
- $\boldsymbol{\omega_o}$ is the outgoing direction of light
- $\boldsymbol{\omega_i}$ is the incident direction of light
- $\mathbf{n}$ is the unit surface normal
- Ω is the unit hemisphere centered around $\mathbf{n}$
- $L_o(\mathbf{x}, \boldsymbol{\omega_o})$ is the outgoing radiance on a given point in space $\mathbf{x}$ towards a particular outgoing direction ω_o
- $L_e(\mathbf{x}, \boldsymbol{\omega_o})$ is the emitted radiance of the surface point $\mathbf{x}$
- $f_r(x, \boldsymbol{\omega_i}, \boldsymbol{\omega_o})$ is the bidirectional reflectance distribution function which describes surface materials as how much light they reflect based on incident and outgoing directions of light
- $L_i(\mathbf{x}, \boldsymbol{\omega_i})$ is the incoming radiance at a point $\mathbf{x}$ from a particular direction $\boldsymbol{\omega_i}$

A key observation of this equation is that it is defined recursively since the incoming radiance L_i at a particular point $\mathbf{x}$ may depend on the outgoing radiance L_o at a different point in space $\mathbf{x'}$ towards $\mathbf{x}$. A closed form solution of this equation for general virtual environments and arbitrary materials was not found yet, hence constituting a major challenge in solving or closely approximating it.

Although this compact equation can explain a lot of visual phenomena it also lacks some important aspects which for example are:

- Transmission through the surface of solids (e.g. glass) or through volumetric bodies (e.g. fog, water) since only interactions at the surface of objects are defined
- Discrepancies in entry and exit points of light on a surface, i.e. light is not directly reflected at the surface but within the solid (subsurface scattering). This can be observed on various materials like skin, wax or marble.
- Phenomena described by the wave character of light like polarization, fluorescence, interference or the relativistic Doppler effect.

Mainly two different techniques for approximating the rendering equation have been applied: finite elements and Monte Carlo integration. While the first one led to the development of the radiosity method [ICG86], the latter one was employed in a variety of algorithms to approximate the rendering equation more closely depending on the scene and material parameters. Examples of algorithms utilizing Monte Carlo integration are:

- Uni- and bidirectional path tracing
- Metropolis light transport
- Photon mapping

A more detailed list about Monte Carlo methods for light transport is given in Eric Veach's dissertation [Vea97].

This work among others which will be explained later, makes use of unidirectional path tracing due to its simplicity, flexibility and having the least algorithmic complexity out of the other mentioned algorithms. Although this comes with giving up potential improvements to visual quality for certain arrangements of scenes, it is better suited for real-time applications. Hence, unidirectional path tracing, further referred to as just path tracing, and Monte Carlo integration will be explained in more detail.

2.1.1 Monte Carlo Integration

Numerical integration of definite integrals proves as an important but difficult task for many application ranging from geometry, mechanics to almost every other natural sciences. Although, there exists a variety of schemes for numerical integration such as Riemann sum, Trapezoidal or Simpson's rule increasing in precision, they all require a uniform sampling, i.e. equal intervals. Depending on the integrand this can lead to aliasing or interference of the function with the sampling pattern and thus to errors or inaccuracy in the estimate of the integral. Furthermore, for higher dimensional integrals these methods suffer from the curse of dimensionality. Due to the inherently high dimensionality of the rendering equation, these methods are not applicable to obtain a useful approximation. In contrast, Monte Carlo integration utilizes random numbers for a statistical estimate of an integral. The basic correlation between definite integral and Monte Carlo estimate is given in the following equation:

$$\int_a^b f(x)\,\mathrm{d}x = (b-a)\mathrm{E}[f(x)] \approx \frac{(b-a)}{n}\sum_{i=1}^{n} f(X_i) \tag{2.2}$$

where $\mathrm{E}[f(x)]$ is the expected value of $f(x)$ in a statistical sense and X_i is a random number drawn from a uniform distribution.

Although, Monte Carlo integration generally converges slower than above mentioned numerical integration methods, it can handle arbitrary sample intervals. This can be advantageously used to distribute samples according to how much they contribute to the final estimate which is called importance sampling.

Importance Sampling

The original formulation of Monte Carlo integration makes use of any set of random numbers to approximate a definite integral which may slow down convergence and make the estimate oscillate around the reference value. An improvement of this method is to draw random samples from specific distributions and weighting them according to their contribution to the estimate. This distribution is called a probability density function which - in the ideal case - is proportional to the integrand. Unfortunately, this is rarely possible because in order to deduce this probability density function one has to sample the integrand which in most cases is unknown. However, for a lot of applications it is still possible to formulate a useful probability density function without knowing the integrand. Examples will be given later in Section 2.1.4. The new formulation of importance sampled Monte Carlo integration is stated as the following:

$$\int_a^b f(x)\,\mathrm{d}x = (b-a)\mathrm{E}[f(x)] \approx \frac{(b-a)}{n}\sum_{i=1}^{n} \frac{f(X_i)}{p(X_i)} \tag{2.3}$$

where $p(X_i)$ is the probability density function which roughly follows $p(X_i) \propto f(X_i)$.

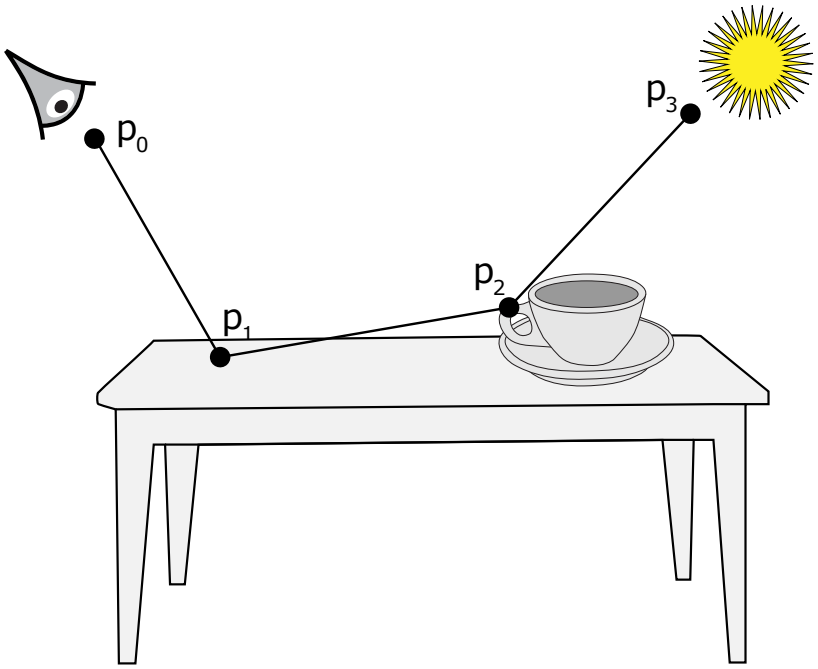


Figure 2.1: One exemplary path of light emitted emitted by a light source and bouncing twice before reaching the observer. Reprinted from [PJH16].

Bias

An important concept in the field of Monte Carlo integration is called bias. Bias is the intentional error made in the numerical integration in order to decrease variance and thus speed up or even solve certain integrals. Formally is it given as the following equation:

$$\beta = \mathrm{E}[f(x)] - \int f(x)\,\mathrm{d}x \tag{2.4}$$

Bias can take many forms but one of the most common ones used in rendering is correlated sampling where samples drawn for multiple events are correlated in order to reduce variance. This technique will be described in more detail in Section 2.1.5.
In general, unbiased methods for light transport may be more desirable to closely and correctly approximate the rendering equation. However, for certain arrangements of scenes an unbiased method will never or extremely rarely find a converged solution. Furthermore, in applications where performance plays a role, bias might further be exchanged with low initial variance.

2.1.2 Path Tracing

"Path tracing was the first general-purpose unbiased Monte Carlo light transport algorithm used in graphics" [PJH16]. Path tracing was first introduced by Kajiya [Kaj86] to provide a simple but powerful method of solving the rendering equation up to a degree such that in many cases the results seem to be indistinguishable from real photographs.
In path tracing the rendering equation is approximated via Monte Carlo integration, whereas each sample of the integrand is considered as a path of light moving through the scene, emitted by a light source and bouncing off various surfaces before reaching the observer eventually. Actual tracing of paths is done backwards, starting from the camera and ending on a light source. A single path is given as a visual example in Figure 2.1. Each intersection of the light ray and a surface that is part of the path is called a path vertex. In theory, an infinite number of paths with infinitely many variations are required to exactly compute the integral of the rendering equation. Since computers cannot deal with infinite quantities easily, both of these issues are addressed by path tracing:

- The first is an essential parameter in path tracing called samples per pixel or stochastic rays per pixel (spp). Increasing the number of paths traced per pixel increases the accuracy of the calculated pixel value and reduces its variance. An example of how this number affects the appearance of a path traced image is illustrated in Figure 2.2.
- To counter tracing rays with infinitely many intersections, it is common practice to stop after hitting a light source or a certain number of recursion steps, thus increasing bias. Especially implementations running on the GPU require a fixed budget of recursion steps in order to gain the most advantage out of the massive parallelization.

Algorithm 2.1 provides pseudo code of a naive path tracer to render a scene.

(a) 8 samples per pixel. (b) 1024 samples per pixel.

Figure 2.2: Path traced images with different number of samples per pixel. Adapted from [PJH16].

Algorithm 2.1: Simple path tracer.

Input: Number of samples per pixel *spp* and resolution image given as *width* and *height*
Output: Image I

```
for x ← 1 to width do
    for y ← 1 to height do
        pixel ← 0⃗;
        for s ← 1 to spp do
            ray ← GenerateRay(x, y);
            pixel ← pixel + TraceRay(ray, 0);
        end
        I(x, y) ← pixel / spp;
    end
end
return I
```

where GenerateRay(..) generates a ray starting from a pixel (x, y) with a direction according to the projection settings of the camera. TraceRay(..) performs the actual recursive tracing and is described in Algorithm 2.2.

Algorithm 2.2: Pseudo code for basic path tracer.

Input: Ray *ray* with origin *ray.origin* and direction *ray.direction*. Current recursion tracker *depth*.
Output: Color of traced ray

```
1  if depth > MaximumDepth then
2  |  return Black
3  end
4  intersection ← IntersectWithScene(ray);
5  if intersection.hit == false then
6  |  return Black
7  end
8  nextRay.origin ← intersection.hitPoint;
9  nextRay.direction ← SampleHemisphere(intersection.normal);
10 brdf ← EvaluateBRDF(intersection.material, ray, nextRay);
11 cosθ ← newRay.direction · intersection.normal;
12 L_e ← intersection.material.emittance;
13 L_i ← TraceRay(newRay, depth + 1);
14 return L_e + (brdf * L_i * cosθ)
```

Algorithm 2.2 introduces three new functions:

- IntersectWithScene(..) Deals with intersection of a given ray with all objects contained in the scene and returns the closest intersection along with additional information of the hit point.
- SampleHemisphere(..): Samples the unit hemisphere centered around a given normal to obtain a random direction.
- EvaluateBRDF(..): Evaluates the light-surface interaction taking the surface material into account.

SampleHemisphere(..) will be further described in Section 2.1.4 together with more sophisticated solutions dealing with clever sampling strategies in order to obtain only samples which contribute to the final result thus speeding up convergence. The theory of physically based light-surface interactions necessary for EvaluateBRDF(..) will be described in Section 2.1.3.

For this naive implementation of the path tracing algorithm to result in converged images, i.e. they contain no visible noise, a high number of samples and recursion depth would

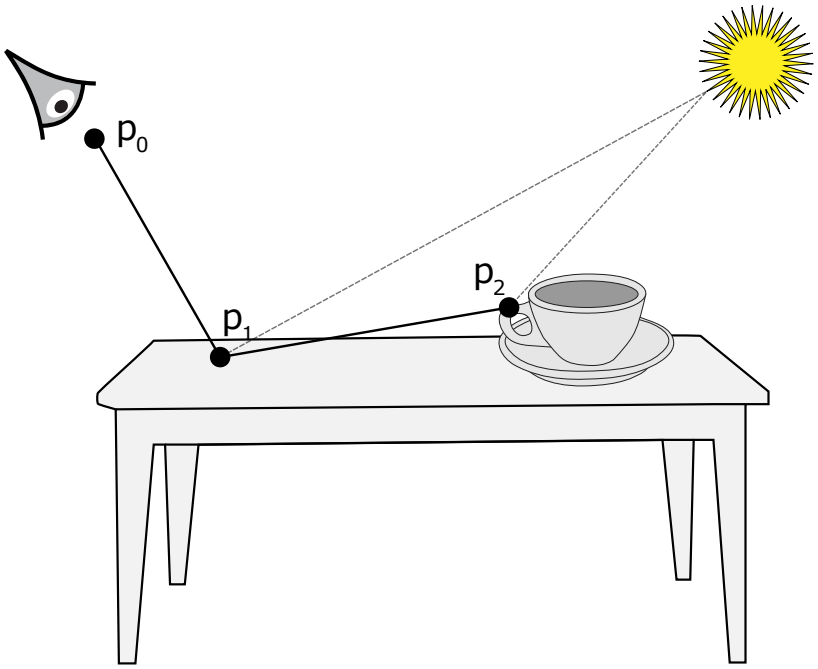


Figure 2.3: Example path consisting of 2 bounces. Each path vertex can be directly connected to the light source to cheaply evaluate a considerable amount of incident light. Adapted from [PJH16].

be necessary. Besides importance sampling there are a few more techniques to speed up convergence, some introduce bias or integration error, some do not. Correlating samples between path vertices or pixels can extremely reduce convergence time by introducing bias. In addition to that, Quasi-Monte Carlo integration is an extension to regular Monte Carlo integration by restricting the set of random numbers to a low-discrepancy sequence. Both methods change the distribution from which samples are drawn. They will be further explained later on in Section 2.1.5. An unbiased method to achieve lower convergence times is called next-event estimation. Separating direct light from light sources and light that indirectly bounces off other surfaces allows for immediate evaluation of a considerable amount of illumination at a given surface point. At each path vertex, light directly incident from a light source can be evaluated and already provides an initial approximation of the illumination, see Figure 2.3. To fit into the rendering equation, contribution from direct light has to be either weighted like an indirect sample or possible paths pointing directly at light sources have to be forbidden during sampling of indirect directions.

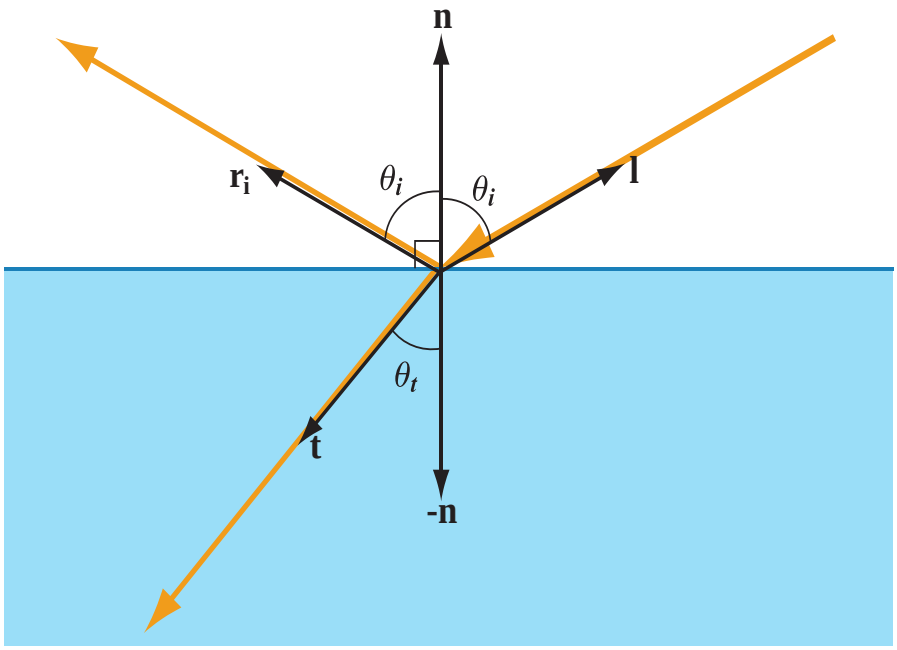


Figure 2.4: An incident light ray scatters into a reflection and a refraction ray when it intersects with a perfectly flat plane. The solution of the Maxwell equations in this simple scenario is called Snell's law. Reprinted from [AMHH18].

2.1.3 Surface Interaction

This section will mainly explain the foundations of how light interacts with surfaces of real-life objects and how those phenomena can be formulated formally in a physically plausible manner. Note, that in this work only light-surface interaction will be explained assuming homogeneous media, i.e. light paths will only change when hitting a surface acting as an interface between two different media. On the other hand, traversal of light through heterogeneous media, like fog, milk or marble, where light paths constantly change throughout a medium will be left out, since this is not easily integrated into the path tracing algorithm, especially for real-time purposes. Furthermore, only local light interaction will be considered, i.e. reflection and refraction which happens in close vicinity of the intersection point.

Physics of Light-Surface Interactions

Light is an electromagnetic wave and its behavior can be explained via the famous Maxwell equations. While for most situations these equations do not have an analytical solution, there is one special case relevant for shading [Hof13]. Although it might seem artificial, the problem of intersection of light with a perfectly flat surface has an analytical solution. In reality, there is no perfectly flat surface and usually the small bumps in the scale of a few hundred nanometers on the surface would lead to wave interference, thus

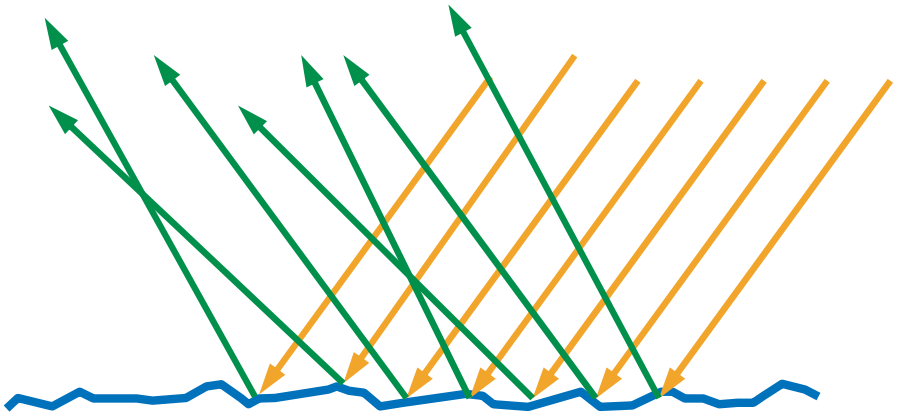


Figure 2.5: Total reflection off a rough surface is the aggregate of multiple planar surface reflections. Reprinted from [AMHH18].

making the shading computations too complex for real-time applications. Instead, the solution to this pure mathematical problem, i.e. the Fresnel equations, was adopted. When light hits a perfectly flat surface, it scatters into exactly two rays whereas one is reflected off the surface and one is refracted below. This is illustrated in Figure 2.4. For reflection, the incident ray is reflected along the normal according to the formula:

$$\mathbf{r_i} = \mathbf{l} - 2\mathbf{n}(\mathbf{l} \cdot \mathbf{n}) \tag{2.5}$$

Refraction additionally depends on the refractive indices of both surfaces stating how strong light is slowed down in the medium. This slowdown leads to a weaker or stronger bending into the media and its precise effect is explained via Snell's law. Its vector form is given as:

$$\mathbf{t} = r\mathbf{l} + (rc - \sqrt{1 - r^2(1 - c^2)})\mathbf{n} \tag{2.6}$$

where $r = \frac{\eta_1}{\eta_2}$ is the fraction of refractive indices and $c = \cos\theta_i = -\mathbf{l} \cdot \mathbf{n}$ is the cosine of the incident angle.

As mentioned earlier, no real surface is perfectly flat and in order to accurately model how light interacts with rough surfaces, one has to distinguish between the size of bumps on the surface. Bumps within a few hundred nanometers, i.e. below the wave length of visible light, induce interference of waves and thus leading to effects which are not easily approximated. For bumps bigger than the wavelength of light but still too small to be directly modeled as geometry, the surface can be considered as a connection of perfectly flat surfaces. The overall appearance of such a rough shape with bumps bigger than wave length of visible light is then described as the aggregate of all planar surface reflections which is visualized in Figure 2.5.

Bidirectional Reflectance Distribution Function

The concept of describing rough surfaces for shading is also referred to as microgeometry or microfacet theory. These models describe how light bounces off various surfaces which are only statistically defined instead of providing the full 3-dimensional geometry. An early microfacet model was introduced by Torrance and Sparrow [TS67] which, in a simplified and adapted version, is given as:

$$f_r(\boldsymbol{\omega_o}, \boldsymbol{\omega_i}) = f_r(\mathbf{v}, \mathbf{l}) = \frac{D(\mathbf{h})G(\mathbf{v}, \mathbf{l})F_r(\mathbf{v})}{4(\mathbf{n} \cdot \mathbf{l})(\mathbf{n} \cdot \mathbf{v})} \tag{2.7}$$

where

- $f_r(\mathbf{v}, \mathbf{l})$ is the bidirectional reflectance distribution function (BRDF) describing the amount of radiance reflected into direction of the observer $\mathbf{v}$ when light is incident from direction $\mathbf{l}$
- $\mathbf{h}$ is the half-angle vector and is equal to $\frac{\mathbf{l}+\mathbf{v}}{\|\mathbf{l}+\mathbf{v}\|}$
- $D(\mathbf{h})$ is the normal distribution function which statistically models the alignment of normals of the microfacets allowing for reflection in the direction $\mathbf{v}$
- $G(\mathbf{v}, \mathbf{l})$ is called the geometry term and describes statistically geometric self shadowing and masking
- $F_r(\mathbf{v})$ is the Fresnel term describing how much light is reflected from an optically flat surface in the given direction $\mathbf{v}$

The bidirectional reflectance distribution function is an essential part of the rendering equation (see Equation 2.1) and thus of the path tracing algorithm. Its purpose is to define how exactly light bounces off a surface depending on its material. Another way to view it is that it provides the amount of light reflected towards a viewer given a light-surface interaction. An example of these two views is illustrated in Figure 2.6. Furthermore, the contribution of a BRDF is commonly separated into the sum of diffuse

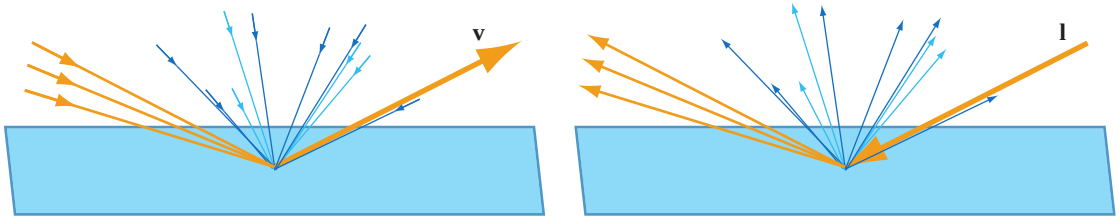


Figure 2.6: On the left, the BRDF describes the total amount of light reflected towards the viewer. On the right, the BRDF shows its scattering behavior based on incident light. Reprinted from [AMHH18].

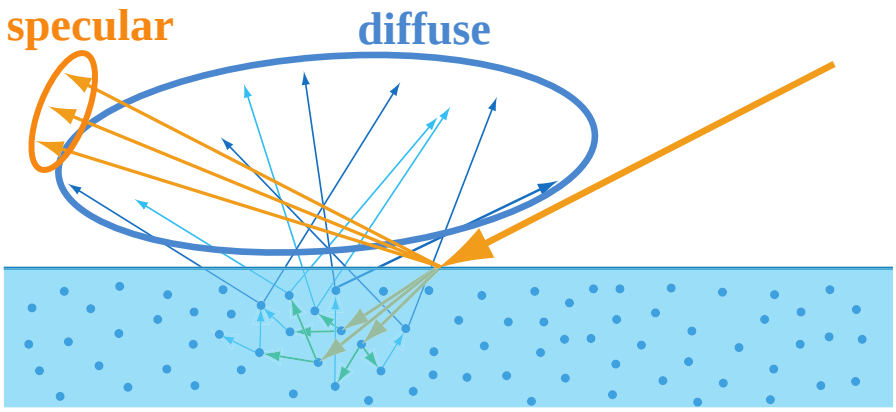


Figure 2.7: Diffuse and specular parts of a BRDF. Light that penetrates the surface but exits again in close vicinity to the original intersection is commonly referred to as diffuse light. Light reflected directly at the surface is called specular light. Reprinted from [AMHH18].

and specular, which is depicted in Figure 2.7. Any incident light which is directly reflected on the surface is referred to as specular light. When light penetrates the surface, scatters within a solid body and exits again in close vicinity to the original intersection (subsurface scattering), it is called diffuse light. Depending on the material of an object one of those parts can be weaker or even be zero. To state an example, metals - or more general conductors - typically only reflect light on the surface and no to very little light scatters below the surface. On the contrary, chalk is an excellent example for an insulator which exhibits almost no reflection on the surface but instead light is scattered in every direction due to local subsurface scattering.

Not any function describing light scattering behavior is a valid candidate for a BRDF. In order to be physically plausible the BRDF has to meet two requirements [Hof13]:

- Reciprocity: $f_r(\mathbf{v}, \mathbf{l}) = f_r(\mathbf{l}, \mathbf{v})$
- Energy conservation: $\forall \mathbf{l}, \int_\Omega f_r(\mathbf{v}, \mathbf{l})(\mathbf{n} \cdot \mathbf{v})\, \mathrm{d}\mathbf{v} \leq 1$

Various valid options for BRDFs emerged throughout the years, falling into the two categories of accuracy and performance. While more accurate solutions were mostly bound to higher computation cost and desired in the field of offline rendering (e.g. movie industry), the latter ones were designed for real-time purposes and were mainly driven by the games industry. For the purpose of this work, only options used in real-time applications will be described here.

Options for the Fresnel term are quite limited when aiming for high-performance. A commonly used form is called Schlick's approximation (see Equation 2.8) and was

published by Christophe Schlick [Sch94]. Cheaper approximations based on spherical Gaussians were presented by Sébastien Lagarde [Lag12], Brian Karis [KG13] and Lazarov Dimitra [Laz13].

$$\begin{aligned} F_r(\mathbf{v}) &= F_0 + (1 - F_0)\big(1 - (\mathbf{n} \cdot \mathbf{v})\big)^5 \\ F_0 &= \left(\frac{\eta_1 - \eta_2}{\eta_1 + \eta_2}\right)^2 \end{aligned} \tag{2.8}$$

On the contrary, there are a lot of options for the normal distribution and the geometry function. Approaches were presented by Trowbridge and Reitz [TR75], Cook and Torrance [CT81], Ashikhmin and Sherley [AS00], Kelemen and Szirmay-Kalos [KSK01], Kurt et al. [KSKK10], Bagher et al. [BSH12], Löw et al. [LKYU12] and Brent Burley [BS12]. The work by Trowbridge and Reitz [TR75] was further extended by a physically correct normalization term by Walter et al. [WMLT07] and is well known as the so-called GGX BRDF.

So far, presented options focused on surface reflection, i.e. the specular term of a BRDF. Methods for approximating the diffuse reflection range from simple constant BRDFs, modeling a uniform reflection in every direction [Lam60], to approaches that model the trade-off of energy between diffuse and specular reflection. Examples of the latter one can be found in [AS00], [KSK01] and [SSHL97]. In addition, some more sophisticated methods also take surface roughness and other properties into account when approximating the local subsurface scattering, see [HK93], [ON94] and [BS12]. The latter one known as Disney's principled BRDF was later reworked to account for correct normalization in its diffuse term by Lagarde and Rousiers [LdR14]. Earl Hammon [HJ17] recently presented a new Oren-Nayar-based [ON94] diffuse term based on the assumptions of the specular GGX BRDF [WMLT07].

Formulas for all normal distribution and geometry functions were omitted due to its huge overhead, but can be found in the respective papers. Due to its relevance to this thesis, Disney's principled BRDF is explained in more detail. It utilizes a diffuse term taking into account roughness and energy trade-off via a slightly adapted Schlick's approximation and is stated below:

$$\begin{aligned} f_d(\mathbf{v}, \mathbf{l}) &= \frac{albedo}{\pi}\big(1 + (F_{D90} - 1)\big(1 - (\mathbf{n} \cdot \mathbf{l})\big)^5\big)\big(1 + (F_{D90} - 1)\big(1 - (\mathbf{n} \cdot \mathbf{v})\big)^5\big) \\ F_{D90} &= 0.5 + 2(\mathbf{l} \cdot \mathbf{h})^2 roughness \end{aligned} \tag{2.9}$$

where

- *albedo* is the base color when viewed at a 90° angle
- *roughness* is an artistic linear parameter in the range $[0, 1]$ controlling the roughness of the surface

Along Schlick's approximation for the Fresnel term, its specular BRDF consists of a geometry term derived by GGX [WMLT07] but including a remapped roughness parameter in order to better fit measured data and artists' needs [BS12]. In addition, it incorporates a generalized version of the Trowbridge-Reitz normal distribution function. Both terms are given below with respect to Equation 2.7:

$$\begin{aligned} D(\mathbf{h}) &= \frac{(\gamma-1)(\alpha^2-1)}{\pi(1-(\alpha^2)^{1-\gamma})}\frac{1}{(1+(\alpha^2-1)(\mathbf{n}\cdot\mathbf{h}))^2)^\gamma} \\ G(\mathbf{v},\mathbf{l}) &= G_1(\mathbf{l})G_1(\mathbf{v}) \\ G_1(\mathbf{w}) &= \frac{2(\mathbf{n}\cdot\mathbf{w})}{(\mathbf{n}\cdot\mathbf{w})+\sqrt{\alpha^2+(1-\alpha^2)(\mathbf{n}\cdot\mathbf{w})^2}} \\ \alpha &= (0.5+0.5roughness)^2 \end{aligned} \tag{2.10}$$

As a final note, BRDFs based on the microfacet theory are neither the only way to model the behavior of light-surface intersections in a physically plausible manner nor applicable to every material. A more modern approach is to measure the BRDF for a whole variety of real materials and use the information as lookup tables stored in material databases [MPBM03]. Since a BRDF is comprised of 4 dimensions (2 angles per direction), measurements generate too much data and the information thus has to be encoded, e.g. using a limited set of Fourier basis functions [PJH16]. While simple materials can easily be represented by a single BRDF a whole variety of materials we encounter in daily life cannot, e.g. coated surfaces. These so-called layered materials need extra attention when taking local subsurface scattering, especially between layers, into account, see for example Jakob et al. [Jak14]. Furthermore, microfacet theory cannot handle wave optic phenomena which are typically caused by nano-scale bumps on the surface. Nonetheless, BRDFs derived from the microfacet theory proved to yield visually convincing and physically plausible results for a wide range of materials while still being relatively easy to compute, making it a perfect fit for real-time applications.

2.1.4 Sampling of Lights and BRDFs

Although importance sampling already provides an effective reduction of noise, convergence rates still can be quite long depending on the distribution samples are drawn from. A mirror, for example, only reflects light from the perfect reflection direction or at least within a very small cone. Therefore, sampling a hemisphere uniformly when hitting a mirror will not prove to yield good results since only very few rays will match the reflection direction. Instead, the reflection direction along with a small cone of possible light contributions could be determined and then used for sampling. This concept can be extended to various surface materials and furthermore to the BRDFs themselves. This clever sampling strategy can lead in most cases to an incredible speedup in convergence since for each intersection only a limited set of reflection directions has to be considered for sampling. However, diffuse surfaces such as chalk or dry walls still reflect incident light in almost every direction which leads to little to no improvement on those surfaces.

Section 2.1.3 introduced Disney's principled BRDF [BS12] as the combination of a Fresnel-roughness-based diffuse term and a specular term derived from GGX. First, importance sampling requires a suitable probability density function which is given below:

$$\begin{aligned} pdf_{diff} &= \frac{\mathbf{n} \cdot \mathbf{l}}{\pi} \\ pdf_{spec} &= \frac{D(\mathbf{h})(\mathbf{n} \cdot \mathbf{h})}{4(\mathbf{l} \cdot \mathbf{h})} \end{aligned} \tag{2.11}$$

Second, as stated before, it is recommended to draw samples according to this probability density function. This is done by computing the cumulative distribution function as the integral of the probability density function, inverting it and then sampling the obtained function. Given the normal $\mathbf{n}$, tangent $\mathbf{t}$ and bitangent $\mathbf{b}$ of an intersection point and two random numbers $\xi_1, \xi_2 \in [0, 1]$, the analytic solution for an importance sampled reflection direction $\mathbf{r'}$ of Disney's principled BRDF is stated in Equation 2.12 for its diffuse and in Equation 2.13 for its specular term.

$$\begin{aligned} \mathbf{r}'_{diff} &= \sqrt{\xi_2}\cos\phi\mathbf{t} + \sqrt{\xi_2}\sin\phi\mathbf{b} + \sqrt{1-\xi_2}\mathbf{n} \\ \phi &= 2\pi\xi_1 \end{aligned} \tag{2.12}$$

$$\begin{aligned} \mathbf{r}'_{spec} &= 2(\mathbf{v} \cdot \mathbf{h'})\mathbf{h}' - \mathbf{v} \\ \mathbf{h}' &= \sin\theta_h\cos\phi\mathbf{t} + \sin\theta_h\sin\phi\mathbf{b} + \cos\theta_h\mathbf{n} \\ \phi &= 2\pi\xi_1 \\ \cos\theta_h &= \sqrt{\frac{1 - \left[(\alpha^2)^{1-\gamma}(1-\xi_2) + \xi_2\right]^{\frac{1}{1-\gamma}}}{1-\alpha^2}} \\ \sin\theta_h &= \sqrt{1 - \cos^2\theta_h} \end{aligned} \tag{2.13}$$

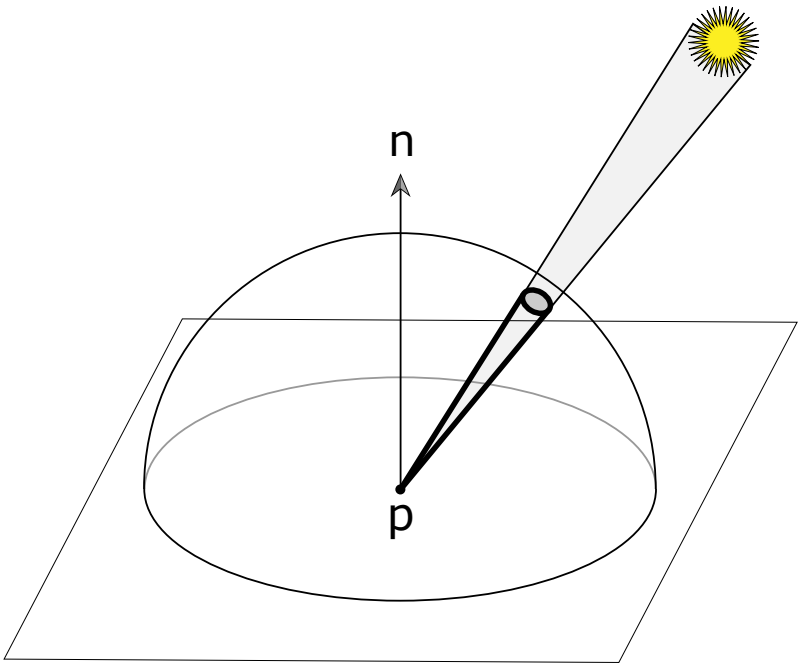


Figure 2.8: The contribution of physically plausible light sources to the illumination at a given point is given as an integral over all directions pointing at all surface points of the light source. Sphere lights are evaluated by integrating over a cone of directions. Reprinted from [PJH16].

If light-surface interactions are treated in a physically based manner, so should light sources be. In the early days of computer graphics, light sources used in the graphics domain were described as pure mathematical constructs which would never occur in reality. Popular ones are directional lights to simulate perfect parallel light, point lights defined as infinitesimal small points in space emitting light in all directions and spot lights being essentially point lights restricted to a certain cone. Sampling those lights proves to be very easy. That was the reason why they were so popular, especially for real-time rendering. In reality, light is not emitted by a single point but rather from a shape. Instead of abstracting a light bulb as an infinitesimal small point, a closer match of reality would be to represent it by a sphere. This minor change not only allows for more accurate descriptions of light sources through real physical quantities but also enables shadows to appear soft. The introduction of physically plausible light sources including the property of area also requires an additional integral to be solved during evaluation of directly incident light. Instead of just a single reflection direction a whole domain of directions has to be taken into account when computing the direct illumination - see Figure 2.8. This issue can be solved in the same way as other integrals are approximated in path tracing - Monte Carlo integration. By sampling the light source many times,

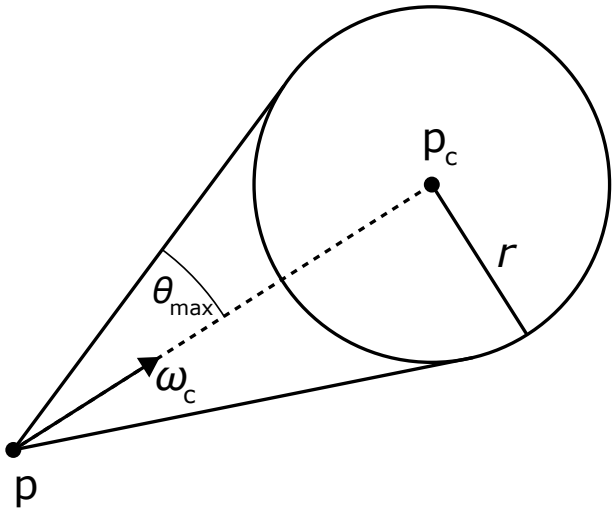


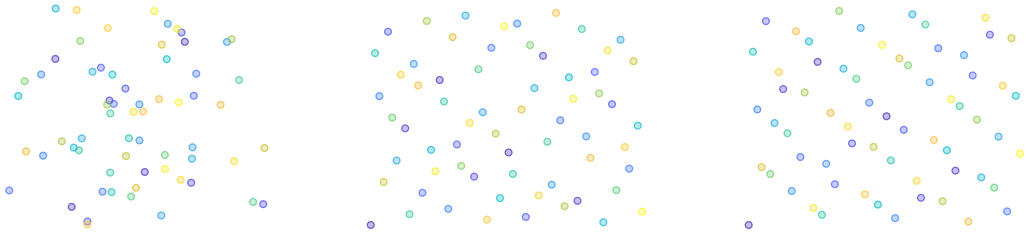
Figure 2.9: Sampling a random direction to a spherical light corresponds to choosing a random direction in a cone with an angle depending on the radius and distance of the sphere. Reprinted from [PJH16].

a decent estimate of the direct light integral can be computed. Again, a solution for sampling the most relevant directions to the light source is necessary for fast convergence, similar to the one introduced for BRDFs earlier. An example of how a random direction $\mathbf{l}'$ to a spherical light source is sampled, is depicted in Figure 2.9 and the according formulas are stated below:

$$\begin{aligned} \mathbf{l}' &= \sin\theta\cos\phi\mathbf{t} + \sin\theta\sin\phi\mathbf{b} + \cos\theta\mathbf{n} \\ \phi &= 2\pi\xi_1 \\ \cos\theta &= (1-\xi_2) + \xi_2\cos\theta_{max} \\ \sin\theta &= \sqrt{1-\cos^2\theta} \\ \cos\theta_{max} &= \arcsin\left(\frac{r}{\|\mathbf{p}-\mathbf{p_c}\|}\right) \end{aligned} \tag{2.14}$$

where $\mathbf{t}$ and $\mathbf{b}$ are mutually perpendicular to $\mathbf{n}$, r denotes the radius and $\mathbf{p_c}$ the center position of the spherical light source.

Eventually, in the context of importance sampling, multiple importance sampling [Vea97] has to be mentioned. It provides utility to weigh samples drawn from multiple distribution used for estimating individual terms of a single integral. In the long term, this provides an additional variance reduction method.



(a) Uniform random samples. (b) Halton sequence. (c) Sobol sequence.

Figure 2.10: Distribution of 64 samples drawn from different 2d point sets.

2.1.5 Low-Discrepancy Sequences and Correlated Sampling

Monte Carlo integration relies on drawing samples, i.e. random numbers, from a uniform distribution. However, random numbers following a uniform distribution come with their own problems such as clumping which in some cases initially increase variance in the estimate of the integrand and increase the convergence time.
Stratified sampling represents a first attempt at obtaining better distributed samples. Therefore, the domain is first subdivided into n non-overlapping regions called strata where n is the number of samples which ought to be drawn. Then, a regular random sample is placed in every strata. Although this alleviates clumping, it suffers from the curse of dimensionality since the computation cost increases exponentially with number of dimensions.
Another choice to tackle this problem is called Quasi-Monte Carlo integration which uses low-discrepancy sequences instead of uniformly distributed random numbers. A low-discrepancy sequence is characterized by its equidistributed appearance, i.e. it is quite similar to the appearance of a equidistant point sampling but with enough randomness in it to be useful for Monte Carlo integration. Another great advantage for most of those point sets lies in the computational simplicity. A major drawback is that, depending on the type of the sequence, it is not easily or impossibly extendable to an arbitrary number of dimensions. Although, this can be circumvented in many situations by employing scrambling, i.e. random permutations of sample indices. Popular examples for low-discrepancy sequences are van der Corput, Hammersley, Halton and Sobol sequences. A comparison of uniform random numbers, Halton and Sobol sequence is shown in Figure 2.10.

While careful sample placement as described earlier and low-discrepancy sequences already explore the domain of the integrand pretty well, issues arise for paths which rely on a specific sequence of bounce events. Caustics are an excellent example which describe light phenomena where light rays are focused through reflection or refraction to form a beautiful pattern on a mostly diffuse surface. To obtain caustics during rendering, paths have to be traced which in the simplest case first hit a diffuse surface, then are reflected

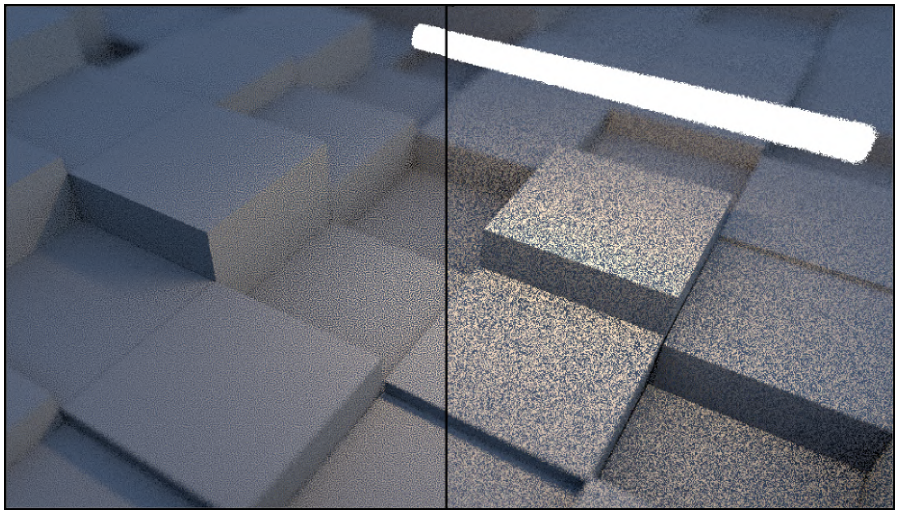


Figure 2.11: Monte Carlo error is distributed as blue noise using a correlated low-discrepancy sampler in the left image. A random low-discrepancy sampler was used for the right image. Adapted from [HBO+19].

toward a specific point on a reflective or refractive surface where light is directly incident. With conventional methods described so far, it is almost impossible to render noise-free images of caustics since sampling the first diffuse surface hit is very unlikely to reflect the light in exactly the few directions which are needed for the caustic. An improvement can be made by correlating samples between multiple events along a single path. Sample correlation, however, may increase bias and introduce certain patterns of noise in the final image which have to be tackled but usually require less effort than reducing Monte Carlo noise. An excellent overview of sample correlation techniques was recently published by Singh et al. [SÖA+19]. Blue noise sampling patterns, i.e. noise that consists only of high frequencies in its spectral representation, play an important role in improving observed visual quality, especially when Monte Carlo errors are distributed in blue noise patterns in the final image. This was demonstrated first by Georgiev and Fajardo [GF16] and further extended by Heitz et al. [HBO+19] providing a low-discrepancy sampler achieving the same effect. The latter one is compared visually with an uncorrelated low-discrepancy sampler in Figure 2.11.

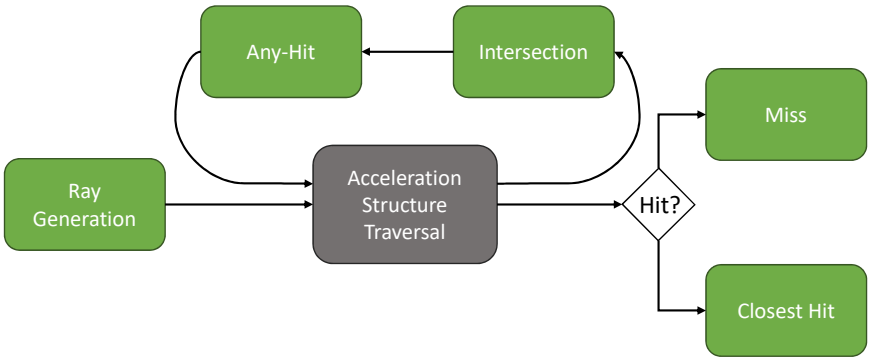


Figure 2.12: Green sections mark new programmable shader stages while the grey one is the actual hardware accelerated stage. First, rays need to be generated by the ray generation shader but are not restricted in number or direction. Afterwards, all rays are traversed through and intersected against a predefined scene. Intersection against triangles, and hence triangle meshes, is built-in but this behavior can be overwritten in order to intersect with arbitrary shapes. Tracing can either process every single intersection along a path (any-hit shader) or only return the closest hit for shading and further processing. If the ray does not intersect with any object in the scene, the miss shader is invoked. Shaders can trace further rays up to a given predefined recursion depth.

2.2 Real-time Ray Tracing and Hybrid Rendering

Ray Tracing in real-time applications has gained high popularity since the release of NVIDIA's Turing architecture providing hardware acceleration to certain parts of ray tracing. Although, the presented hardware was not the first to provide acceleration for ray tracing, it can be considered as the first consumer-level hardware, available to a wide variety of users. Former solutions for hardware accelerated ray tracing were initially designed for volume rendering ([PHK+99], [MKW+02]) and were further extended for a more general usage of ray tracing ([SWS02], [SWW+04]). A more in-depth description of the Turing architecture was published by NVIDIA [Nvi18] and a recent analysis of its performance by Sanzharov et al. [SGFV19]. At the same time, Microsoft developed and published DirectX Raytracing (DXR) as an extension to the low-overhead graphics API DirectX 12. DXR gave graphics programmers the opportunity to explore what is possible with the new hardware accelerated ray tracing. Since DXR played a critical role in the development of this work, an overview is given in the following section.

2.2.1 DirectX Raytracing

DXR introduces a new workflow for real-time ray tracing on GPUs with additional programmable shader stages. A brief visual overview of the GPU pipeline and the newly

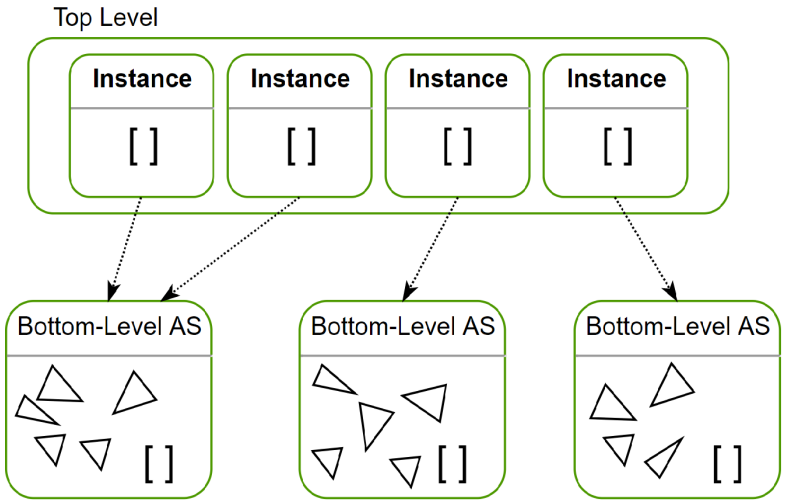


Figure 2.13: Illustration of two-level acceleration structure. While BLAS entries store geometry information, TLAS entries store instances of BLAS entries together with their transformations. Reprinted from NVIDIA Raytracing tutorial [2].

involved shaders is given in Figure 2.12. To carry information from one shader stage to the next, rays are accompanied by user-defined chunks of data named payloads. These payloads may store data about color or light intensity, recursion depth, random seeds or any other numeric quantity.
DXR does not rely on graphics hardware with hardwired functionality for ray tracing. Instead, traversal and intersection can also be emulated in shaders as it is for example necessary for the older NVIDIA Pascal architecture. However, hardware acceleration provided by the Turing architecture, in particular the new RT cores, allows for speed-ups by up to 10x when performing ray tracing [1] and thus is essential to a real-time ray tracing application.

Besides the new programmable shader stages, DXR also introduces:

- A compact GPU scene description split into a two-level tree-based acceleration structure
- Shading binding tables for assignment of DXR shaders to their respective stages
- Built-in ray-triangle intersection shader

[1] https://www.nvidia.com/en-us/geforce/news/geforce-gtx-ray-tracing-coming-soon/
[2] https://developer.nvidia.com/rtx/raytracing/dxr/DX12-Raytracing-tutorial-Part-1

In order to achieve fast traversal of rays through a scene, DXR provides an acceleration structure comprised of a bottom-level and a top-level structure to speed up intersection tests. The bottom-level acceleration structure (BLAS) directly stores shapes such as triangle meshes or any procedural primitives used for intersection along with additional shape data like normals, texture coordinates and similar. The top-level acceleration structure (TLAS) stores a hierarchy of BLAS entries along with a transformation property. Figure 2.13 provides an illustration of this two-level acceleration structure.

As a last note on this very brief introduction to DirectX Raytracing, some performance considerations are presented. Although, ray tracing is hardware accelerated on Turing GPUs, this does not mean rasterization can be completely replaced by ray tracing. Currently, the number of RT cores are quite limited and only permit for a low number of rays to be traced per pixel per frame depending on the complexity of shaders. Furthermore, incorrect or careless usage of the DXR API can cause dramatic performance problems. Examples are wrong usage of tracing flags or too large ray payloads. Furthermore, the more rays diverge between neighboring pixels, the more incoherent memory accesses get and thus caching looses its value rather rapidly.
For a more detailed view on the DXR API, the functional specs are found on GitHub [Mic18]. A more abstract and code-based description can be found in the *Ray Tracing Gems* [HAM19, Chpt. 3].

2.2.2 Hybrid Rendering

Due to the limitations of the number of rays that can be traced per frame, a new strategy in real-time rendering arose: Hybrid Rendering. Rasterization and ray tracing working together on the GPU through a single interface. Combining the speed of rasterization and the flexibility of ray tracing led to a number of reworked real-time global effects such as shadows, reflections, ambient occlusion and diffuse global illumination. Surprisingly, these effects have already been integrated into major game engines such as *Unreal Engine*[3] and *Unity*[4] despite the short period of time the necessary features have been around. Furthermore, a growing number of AAA-games such as *Battlefield V* ([ED18]), *Control* ([Ent19]), *Metro Exodus* ([Gam19]) or *Shadow of the Tomb Raider* ([CD18]) were shipped with support of ray traced shadows, reflections, ambient occlusion and/or even first bounce diffuse global illumination. First used as a research project in the field of real-time de-noising of path traced images [SPD18], the graphics back end of the classic shooter *Quake II* was completely overhauled to make use of real-time path tracing and de-noising [NVI19].

One of the most popular approaches is based on deferred shading. First, data necessary for shading, such as positions, normals and material parameters, is written into a set of

[3] https://devblogs.nvidia.com/introduction-ray-tracing-unreal-engine-422/
[4] https://docs.unity3d.com/Packages/com.unity.render-pipelines.high-definition@7.1/manual/Ray-Tracing-Getting-Started.html

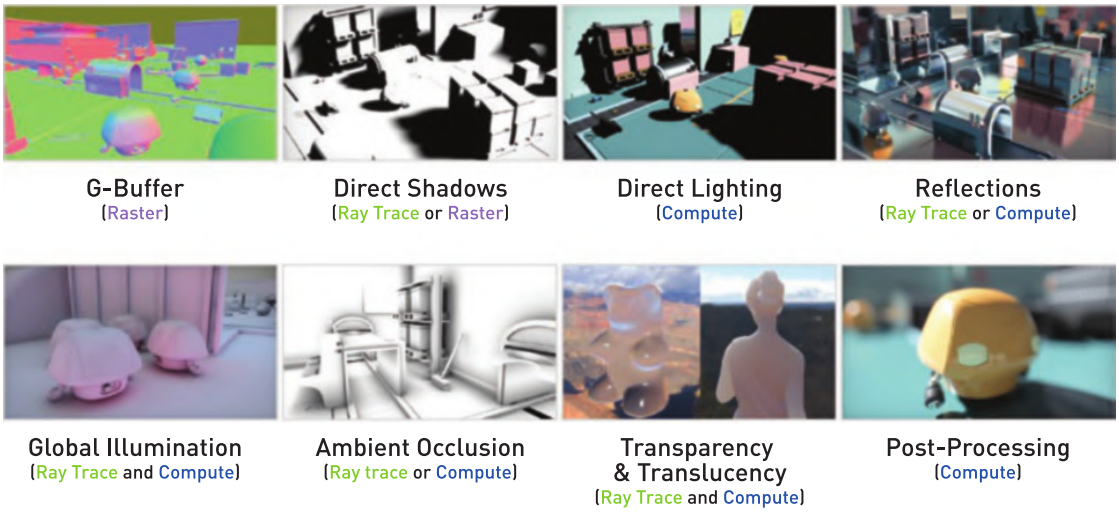


Figure 2.14: Overview of the pipeline of a hybrid renderer. Reprinted from [HAM19, Chpt. 25].

textures - the G-Buffer. In a usual deferred renderer, the shading computations would then be evaluated for each pixel, whereas missing parts, such as visibility and higher order light bounces, are approximated by rasterization-based methods. Shadow Mapping [Wil78] is an excellent example for a rasterization-based method, which captures the global scene visibility from the perspective of a light source in order to determine whether a given position is actually lit by this light source or not. Such discrete scene representations for shading purposes are heavily depending on the resolution of the discretization and hence may suffer from aliasing artifacts during resampling. An algorithm to compute the visibility based on ray tracing instead, is as simple, as testing whether a ray towards the light source is blocked beforehand or reaches the light source. Soft shadows produced by light sources with area prove to be a complication since not a single but a whole domain of rays have to be traced to determine how much light is actually incident to a given point. Stochastically sampling the light source gives a constant amount of work to do per point to shade but introduces noise in the shadow. Hence, the noisy visibility is gathered in a separate texture which is de-noised afterwards. De-noising strategies will be described later on in Section 2.3 but it is important to note here that de-noising requires additional information about the surroundings which is provided via several noise-free buffers contained in the G-Buffer. Eventually, the de-noised visibility information can be used for shading computations. Similarly, specular reflections can be traced for glossy surfaces by sampling the BRDF as described in Section 2.1.4, de-noised and used during shading.
Hybrid rendering extends further and can be applied to more phenomena as discussed here. An overview of some effects and how they can be more accurately computed in real-time rendering is shown in Figure 2.14. Note, that this is still an early overview and GPU ray tracing will likely be applied to various other real-time graphics problems in the near future.

2.3 Filtering of Monte Carlo Noise

Due to the low sample count and despite all sampling techniques, the path traced output contains a high amount of noise and thus cannot be presented directly to a user. Even the slightest changes in the random number sequence can lead to major differences between neighboring pixels, although they might even be of the same material. These differences occur in the temporal and spatial domain and are also referred to as high frequency changes. Smoothing out these high frequencies between pixels in both domains is accomplished by applying a suitable filter accounting for changes in geometry and material between samples. A temporal filter makes use of the information of previous frames to average a certain amount of pixels corresponding to the same point in the scene. Neighboring samples, independent of their current domain, are likely to contain coherent information. Therefore, a spatial filter may be applied to samples that are similar in terms of geometry and material. Filters for Monte Carlo noise, often referred to as de-noisers, simply try to cheaply approximate the effect of additional samples computed during path tracing by reusing existing ones that have similar conditions.

Related to the earlier mentioned hybrid rendering pipeline - see Section 2.2.2 - methods capable of robustly de-noising specific features arose in the industry of real-time rendering. Bokansky et al. [HAM19, Chpt. 13] trace a limited amount of rays in combination with a spatio-temporal filter to estimate the visibility term of direct lighting in order to obtain noise-free soft shadows. Heitz et al. [HHM18] address the issue of correctly recombining direct illumination with de-noised ray traced shadows. Edward Liu et al. [HAM19, Chpt. 19] demonstrate how shadows, reflections, ambient occlusion, indirect diffuse lighting and translucency are rendered in a stochastic manner and de-noised separately in *Unreal Engine* to produce real-time ray traced cinematics. Barré-Brisebois et al. [HAM19, Chpt. 25] present their hybrid pipeline capable of rendering soft shadows, glossy reflections, ambient occlusion and first bounce indirect diffuse illumination in a real-time ray tracing experiment called *PICA PICA*. In conclusion, de-noising just specific parts of the rendering equation eases implementation effort in already existing engines and achieves astonishing real-time results. Nonetheless, de-noising the fully path traced image gives the opportunity for an even simpler and more accurate result while further unifying shading related stages in the rendering pipeline.
A general de-noising algorithm, capable of eliminating visual noise in low sample path traced images in real-time, does not come without drawbacks. First and foremost, such filtering methods are tied to a certain rendering pipeline and require additional information either from first hit rasterization or tracing of indirect rays. In addition to that, the filters are designed to deal with very specific surface materials and their bouncing behavior, hence limiting the artistic freedom.

The following sections will give an overview about real-time methods for temporal and spatial filtering of path traced images and their final places in a de-noising pipeline.

2.3.1 Temporal Filtering

When rendering a static image that does not contain any movement of objects or the camera, temporal accumulation is as simple as averaging the same pixels. The whole history comprising all frames rendered so far may be employed. As soon as something is moving, the correspondence between pixels describing the same surface points has to be reevaluated. Even if the parameters are known how to transform one pixel from one frame to the corresponding pixel in a previous or succeeding frame, these pixels would not likely describe the very same point of a surface. The point might have been occluded or is now. Furthermore, when ignoring the integration over the area of the pixel, its value is solely computed from a single sample. Just a small movement would cause the point samples of corresponding pixels to be not aligned anymore. However, those samples are considered to be close enough to be accumulated.

The transformation to determine where a pixel of one frame would land in a previous or succeeding frame is called reverse or forward reprojection respectively [NSL+07, YNS+09]. During forward reprojection, samples are scattered into the next frame yielding gaps and occlusion artifacts which have to be manually resolved [NSL+07]. On the contrary, reverse reprojection follows a gathering approach and accumulates valid samples, but in the worst case, only the results of the current frame are available.

Estimating the transformation involved in reprojection and thus the position of a pixel in a temporally neighboring frame includes the computation of a discrete motion field. Per-pixel vectors describe the offsets between corresponding pixels in a screen-affine space. Usually, computing these motion vectors does not require more effort than storing all transformation matrices of the previous frame and passing them to the respective shader. By simply transforming each vertex twice, once with the previous transformation and once with the current one, the offset is calculated as the difference of both positions in screen space. Eventually, the current position along with the motion vector are used to access data of previous frames in later stages of the pipeline.

Determining whether shading information of the current frame can be mixed with data from a previous frame plays a crucial role in preventing ghosting, i.e. visible artifacts caused by the remnants of previously not occluded surfaces that are still visible on top of their occluders. An example of changing occlusion between two frames is shown in Figure 2.15. Detecting occlusions between two corresponding pixels requires utilizing auxiliary information. Suitable features for validation can be of geometric nature, e.g. depth or normal, or based on the surface itself, e.g. unique identifiers of objects or material parameters. As mentioned previously, corresponding pixels are unlikely to represent the very same surface point. During reprojection, the sampled point might even not directly land in the center of another pixel but instead between multiple. In such a case, samples of multiple valid pixels can be reused and bilinearly interpolated.

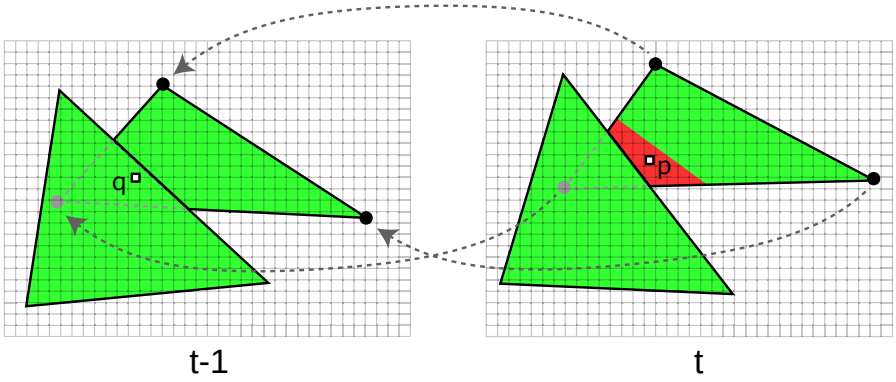


Figure 2.15: Due to movement of one triangle, the pixel labeled q that was previously occluded in frame $t-1$ is revealed in frame t as pixel p. There is no history available for p, hence only shading information generated in frame t is available. Reprinted from [NSL+07].

Until now, sample accumulation was used as a broad term of combining a set of similar samples in a meaningful way. To follow Monte Carlo integration, averaging of samples would be perfectly fine to increase accuracy in the estimate. In practice, however, this would require to keep track of all sample values along with the number of currently contributing samples. Employing a moving average alleviated the need of storing all samples separately. To further improve on that, an exponentially moving average allows for controlling the trade-off between variance reduction and ghosting [YNS+09]. Given sample c_t of frame t and sample c_{t-1} of frame $t-1$, the exponentially moving average is defined as:

$$\bar{c} = \alpha c_t + (1-\alpha)c_{t-1} \tag{2.15}$$

where $\alpha \in [0,1]$ regulates how much history is used to complement shading information of the current frame. According to that, the contribution of single samples decreases exponentially over time but in theory never reaches zero.
A constant value of the temporal accumulation factor α cannot account for changes in illumination on the surface. For example, if a light source is slowly turned on and off, changes of the illumination on nearby surfaces would be delayed. This lag also appears when light sources and their respective bright spots on objects are moving. To counter this issue, Schied et al. [SPD18] introduce a variable factor α based on the temporal gradient of the pixel color.

Temporal filtering exploits coherence between a series of frames to effectively increase the number of samples per pixel. It is, however, of uttermost importance that points that were either previously occluded or outside of the screen *do not benefit* from this filtering at all since there is no valid history to rely on. The sample count of pixels in such areas have to be increased by employing a spatial filter as explained next.

2.3.2 Spatial Filtering

Accumulation of spatially close and similar samples helps in effectively increasing the number of samples per pixel depending on the density of the local vicinity. The spatial domain describes a space containing specific primitives which may be tested of similarity and proximity. Screen space, texture space and path space are examples of spatial domains with pixels, texels and rays as their respective primitives. While the two latter ones are mostly employed in offline de-noising, especially filtering in screen space gained a lot of popularity in real-time de-noising algorithms due to its GPU-friendly implementation. However, Binder et al. [BFK19] recently proposed a path space filter which is massively parallelizable. Nonetheless, an excellent overview of spatial filters for Monte Carlo noise reduction is given by Zwicker et al. [ZJL+15] and Sen et al. [SZR+15]. Unfortunately, the proposed methods are designed to work with a way higher number of samples than are available in real-time applications. Therefore, new filters have to be deduced from existing ones or even developed from scratch to deal with the extremely noisy images produced by real-time path tracing.

In recent years, a lot of different strategies have been employed to design real-time de-noising algorithms in screen space. Proposed approaches range from machine learning [CKS+17], analytic functions discovered by genetic programming [KDK17], feature regression [KIM+19] to discrete convolution [DSHL10, GO12, MMBJ17, SKW+17, HAM19, Chpt. 26]. The method developed in this thesis is based on the latter ones, especially the one presented by Schied et al. [SKW+17], hence these will be explained in more detail. The convolution-based approaches have in common that they filter a single pixel by considering neighboring pixels in a certain pattern - the filter kernel - and evaluating similarity through weight functions on certain features. These weight functions are designed to smooth out the high frequencies in regions which are similar but do not blur across edges, thus they are named edge-stopping functions. This procedure may be applied a single time or multiple times with varying patterns or kernel dimensions.

Filter Kernels and Patterns

A naive way of filtering the spatial area around a given pixel is to simply test all pixels for similarity within a certain radius and average those with according weights. An example of the well known cross/joint-bilateral filter [PSA+04, ED04] following exactly this naive approach is given in the equation below:

$$I(\mathbf{s})' = \frac{1}{k(\mathbf{s})} \sum_{\mathbf{p} \in \Omega} f(\mathbf{p} - \mathbf{s}) g(I(\mathbf{p}) - I(\mathbf{s})) I(\mathbf{p}) \tag{2.16}$$

where $I(\mathbf{i})$ denotes the value of pixel $\mathbf{i}$, Ω is the filter domain, $k(\mathbf{s})$ is a normalization factor for pixel $\mathbf{s}$ given by the sum of all weights within Ω and f and g are Gaussians to penalize larger differences in pixel values or spatial location. Introduction of further weighting terms based on other features allows for even better validation of similarity

between pixels.
Unfortunately, this method requires a huge amount of per-pixel computations and is mostly bandwidth bound, hence not qualified for a real-time approach. Instead, a sparse filter kernel contains just a subset of all sample points within a certain radius. While this boosts performance based on the sparsity, it also reduces the number of samples taken into account. Furthermore, it may lead to situations where no samples at all can be accumulated because similar samples fall through the "holes" of the filter. However, by alternating the filter pattern between iterations or frames, the pixel value will consider all its neighbors during filtering at some point, considering temporal accumulation works properly.
Mara et al. [MMBJ17] apply multiple sparse bilateral filters in their pipeline with temporally varying patterns. Filter patterns contain roughly 0.5% of possible samples in their respective disk and origin from a previous work [MMNL16]. Similarly, Willberger et al. [HAM19, Chpt. 26] propose a multi-pass bilateral filter where the randomness of each pattern is optimized, such that in a 3x3 pixel window the diversity of samples is maximized during temporal filtering.

Discrete wavelet transforms constitute a different approach of image filtering by relying on the computation of multiple convolutions with increasing filter sizes [DSHL10]. The *algorithme à-trous* introduced by Holschneider et al. [HKMMT90] provides an iterative construction scheme for performing the discrete wavelet transform of a signal c as follows [DSHL10]:

1. For iteration $i = 0$ use the input signal $c_0(p)$
2. $c_{i+1}(p) = c_i(p) * h_i$, where $*$ denotes a discrete convolution and h_i are the kernel weights based on a B_3 spline interpolation with values $(\frac{1}{16}, \frac{1}{4}, \frac{3}{8}, \frac{1}{4}, \frac{1}{16})$.
3. $d_i(p) = c_{i+1}(p) - c_i(p)$, where d_i describes the wavelet coefficients of iteration i
4. if $i < N$, go to step 2 if maximum number of iterations N has not been reached
5. The wavelet transform is given by the coefficients $d_0, d_1, ..., d_{N-1}, c_N$ and the signal reconstruction is given by $c = c_N + \sum_{i=N-1}^{0} d_i$

The influence of the filter and which pixels contribute during convolution is exemplary depicted in Figure 2.16.

Incorporating edge-stopping functions changes the convolution in step 2 to:

$$c_{i+1}(\mathbf{p}) = \frac{1}{k(\mathbf{p})} \sum_{\mathbf{q} \in \Omega} c_i(\mathbf{p}) w(\mathbf{p}, \mathbf{q}) h_i(\mathbf{q}) \tag{2.17}$$

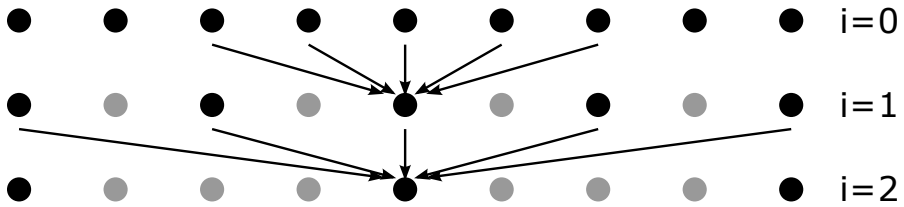


Figure 2.16: Three iterations of the *algorithme à-trous* in 1D. Black dots correspond to pixels which contribute during convolution, gray dots mark pixels which are skipped. Therefore, the filter grows in size by a factor of 2 per iteration. Reprinted from [DSHL10].

where the weight function w comprises the product of all edge-stopping functions and the normalization factor k is given as:

$$k(\mathbf{p}) = \sum_{\mathbf{q}\in\Omega} h_i(\mathbf{q}) w(\mathbf{p}, \mathbf{q}) \tag{2.18}$$

From another perspective, the à-trous wavelet transform seems like a multi-pass, sparse bilateral filter with additional weights and increasing filter footprint. This allows for an efficient implementation on the GPU. It has been applied to Monte Carlo filtering by Dammertz et al. [DSHL10] and Schied et al. [SKW+17, SPD18].

Edge-Stopping Functions

While the regular bilateral filter only made use of differences in pixel values and positions in the input, different weighting functions may be employed to achieve a desired degree of smoothness based on various other features. Possible features might be:

- Geometric features: position or normal, linear or perspective depth.
- Surface properties: material parameters, mesh or material IDs.
- Intermediate results: color, brightness.

Dammertz et al. [DSHL10] propose to use a combination of weighting functions based on illumination, positions and normals. The according functions are stated below:

$$\begin{aligned} w(\mathbf{p}, \mathbf{q}) &= w_{rt} w_n w_x \\ w_{rt} &= \exp\left(- \frac{\|I(\mathbf{p}) - I(\mathbf{q})\|}{\sigma_{rt}^2} \right) \end{aligned} \tag{2.19}$$

where I_p and I_q are illumination values of pixels p and q respectively and σ_{rt} is an estimated variance of changes in the noisy image which reduces with each iteration of the filter. The weights w_n and w_x are formulated in the same fashion with their own σ_n and σ_x respectively.

Schied et al. [SKW+17] replace the position-based weighting function by a depth-based weight. Furthermore, illumination is only given as luminance [Poy98] and weight functions are formulated slightly differently which are shown below:

$$\begin{aligned} w(\mathbf{p},\mathbf{q}) &= w_z w_n w_l \\ w_z &= \exp\left(- \frac{|z(\mathbf{p}) - z(\mathbf{q})|}{\sigma_z |\nabla z(\mathbf{p}) \cdot (\mathbf{p} - \mathbf{q})| + \epsilon} \right) \\ w_n &= \max\left(0, n(\mathbf{p}) \cdot n(\mathbf{q})\right)^{\sigma_n} \\ w_l &= \exp\left(- \frac{|l(\mathbf{p}) - l(\mathbf{q})|}{\sigma_l \sqrt{g_{3x3}(Var(l(\mathbf{p})))} + \epsilon} \right) \end{aligned} \tag{2.20}$$

The depth-based weight function introduces $z(\mathbf{p})$ as the clip-space depth of the pixel $\mathbf{p}$. In addition, accepted depth discrepancy is scaled by the natural change of depth across pixels, e.g. a tilted surface. A locally linear depth model based on screen space partial derivatives is assumed to approximate this range of depth.
Normals are not weighted as the length of their differences anymore, instead their angle changes get penalized directly. The allowed range of angles is further restricted by the parameter σ_n.
One of the more important changes concerns the weight function of illumination. Illumination is replaced by the scalar quantity luminance and the impact of larger changes depends on the local variance of luminance. An estimate of variance is first deduced during the temporal filter stage and averaged over a 3x3 pixel window. This adaptive filter weight leads to stronger blurring for pixels with high variance and preserves areas that already exhibit low variance.
Last but not least, w_z and w_l introduce a small ϵ to avoid division by zero.

2.3.3 De-noising Pipeline

An effective real-time de-noising algorithm does not solely rely on the temporal or spatial filtering but instead merges them to work together as a spatio-temporal filter. This section will provide a brief overview of three different real-time de-noising pipelines proposed in recent years ordered chronologically.

Mara et al. [MMBJ17] factor BRDFs into diffuse and specular parts and approximate their estimators assuming that albedo and Fresnel term vary slowly when changing incident and outgoing light directions. Both terms are then separately filtered. The whole pipeline is shown in Figure 2.17.
Diffuse illumination undergoes smoothing by a joint spatio-temporal filter and subsequently by a final spatial filter to remove the last remains of noise. The loop between step (3) and (4) is due to temporally alternating filter patterns of the sparse bilateral filter. The authors claim that this combination of temporal accumulation and sparse filter patterns give the same results as a dense bilateral filter with equal filter radius

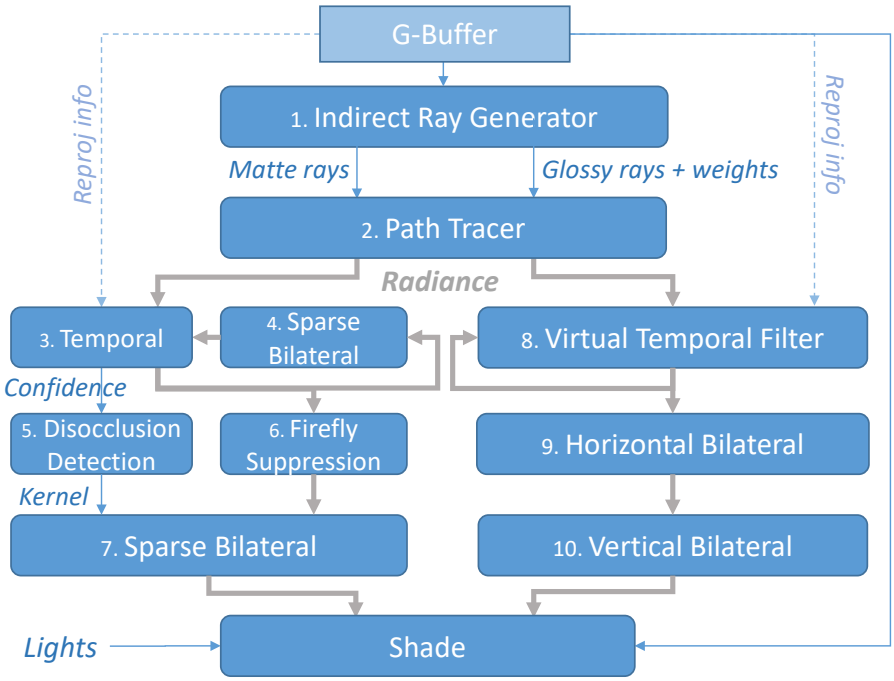


Figure 2.17: Real-time path tracing and de-noising pipeline proposed by Mara et al. Reprinted from [MMBJ17].

but at a fraction of the computation cost [MMBJ17]. Additionally, a 3x3 median filter is involved to filter out fireflies, i.e. statistical outliers that appear extremely bright in relation to the local neighborhood. Step (7) gives the chance of taking disocclusions during the temporal stage into account and fill the noisy holes.
At the same time, specular illumination is temporally reprojected and accumulated by instead using virtual positions of reflected objects [ZJL+15] to avoid ghosting. To avoid artifacts due to the sparse sampling pattern, a dense bilateral filter separated into two 1D passes is applied.

Schied et al. [SKW+17] proposed *Spatiotemporal Variance-Guided Filtering* – SVGF – as a de-noising pipeline introducing novel edge-stopping functions partly driven by a per-pixel variance estimate. In contrast to the work by Mara et al. [MMBJ17] a hierarchical à-trous wavelet transform is employed instead of bilateral filters. Furthermore, instead of splitting diffuse and specular components, direct and indirect illumination are filtered separately which does not require any approximations during evaluation of illumination. Due to separation, shadows can be more faithfully de-noised without overblurring them. The full pipeline is given in Figure 2.18.

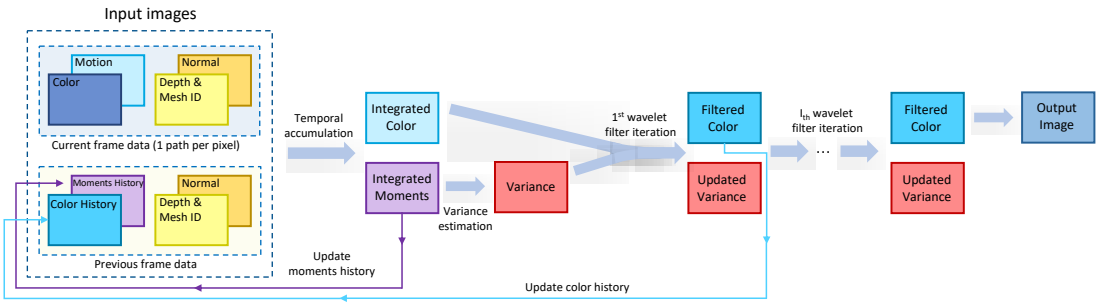


Figure 2.18: Full pipeline describing the SVGF de-noising algorithm proposed by Schied et al. Reprinted from [SKW+17].

Variance estimation is based on a fundamental equivalence in statistics such that variance for a given random variable X with mean $\mu = E[X]$ is given as:

$$Var[X] = E[(X - \mu)^2] = E[X^2] - E[X]^2 \tag{2.21}$$

In the discrete case it can be further formulated as:

$$Var[X] = \frac{1}{n}\sum_{i=1}^{n} x_i^2 - \left(\frac{1}{n}\sum_{i=1}^{n} x_i\right)^2 \tag{2.22}$$

According to that, luminance is treated as a representative random variable for estimation of variance of Monte Carlo integration. Therefore, luminance and the square of luminance - further referred to as moments - are temporally accumulated to approximate the expected value. If temporal history is not sufficient, moments are spatially accumulated through a small cross-bilateral filter. Eventually, the estimated variance drives the luminance-based edge-stopping function during the spatial filtering process to also accumulate bigger differences between samples if variance is high and vice versa.
After the first level of the à-trous wavelet transform, the still noisy image is written back into the history buffer to be used for accumulation in the next frame. After that an arbitrary amount of iterations can be performed to de-noise the image. However, the authors chose a five-level wavelet transform which gives an effective filter footprint of 65x65 [SKW+17]. Estimated variance is reduced after each iteration by filtering it similarly to the noisy signal:

$$Var(c_{i+1}(\mathbf{p})) = \frac{1}{k(\mathbf{s})^2}\sum_{\mathbf{q}\in\Omega} h(\mathbf{q})^2 w(\mathbf{p},\mathbf{q})^2 Var(c_i(\mathbf{q})) \tag{2.23}$$

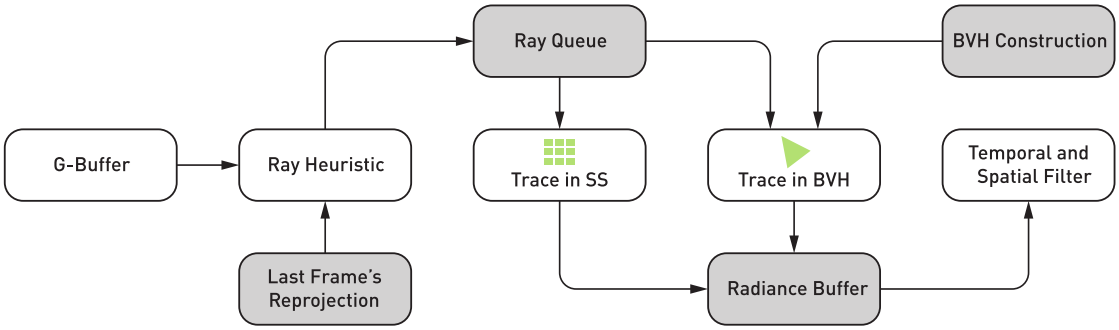


Figure 2.19: Hybrid path tracing approach proposed by Willberger et al. Rays are either traced cheaply in screen space (SS) or through an acceleration structure (BVH) to further speed up path tracing. Reprinted from [HAM19, Chpt. 26].

Willberger er al. [HAM19, Chpt. 26] focus on an efficient combination of rasterization and ray tracing to fall back to the cheaper screen space method if possible. Material parameters are additionally baked into the acceleration structure to avoid multiple accesses to material textures during path tracing. The remaining noise is eliminated via a multi-pass sparse bilateral filter and a temporal accumulation stage. Similarly to Mara et al. [MMBJ17], illumination is split into its diffuse and specular components which are treated slightly differently during the filtering process. An overview of the pipeline is shown in Figure 2.19.
In addition, they propose an adaption of their pipeline to be used for virtual reality. One eye is marked as dominant and its diffuse lighting is updated as usual. Diffuse illumination for the other eye, however, is only reprojected from the previous frame and blended with the rendered and reprojected image of the other eye. The decision on the dominant eye flips with each frame.

2.4 Rendering for Head-Mounted Displays

Virtual reality headsets, also called head-mounted displays (HMD), exploit the stereoscopic vision of humans. Stereoscopic vision or stereopsis refers to delivering a 3 dimensional impression through a pair of 2D images made from specific viewpoints - the eyes. Two, mostly parallel, screen panels requiring a special perspective projection enable stereoscopic vision in order to trick the human brain and convey depth.
Rendering for virtual reality is considered to be especially challenging due to the large overall resolution. Newest generation HMDs[5,6] support a native resolution of 1600x1440 per eye which corresponds to about 2.3 million pixels. In comparison, a full HD (1080p) image contains roughly 2 million pixels, hence rendering for virtual reality is similar to rendering a bit more than two full HD images in terms of computational power. High-quality business HMDs[7,8] provide even higher native resolutions, e.g. 3840x2160 per eye or higher, requiring immense computational power.
In addition to the high resolution, current HMDs work with refresh rates ranging from 80 - 144Hz, higher than the 60Hz rate of usual desktop screens. Aiming for a 90Hz refresh rate, which represents the default rate of most VR headsets, reduces the frame time from 16.67ms for 60Hz to 11.11ms. But this already low frame time is reduced further by applying necessary methods to counter chromatic abberation and lens distortion caused by the optics. However, if rendering takes longer than the given time budget, refresh rate is halved and every other frame is reprojected from the previous frame. This is usually performed via the according VR API and does not require the developer to interfere.
As a last performance limiting factor, rendering directly with just the native resolution typically leads to geometric aliasing due to uneven distribution of shading rates caused by the lenses. A naive way of tackling this issue is to employ multisample anti-aliasing (MSAA).

This just gave a small glimpse of which computational problems arise in rendering for VR. Topics like tracking and prediction of movement, frame reprojection if frame time is exceeded or improved timings for GPU command submission have to be considered as well for stable VR rendering. However, a range of solutions is already provided by special APIs[9] and hence mentioned techniques will not be further explained here. Instead, this section focuses on techniques that may be employed to further reduce the computational complexity.

[5] https://www.valvesoftware.com/en/index/headset
[6] https://www.vive.com/us/product/vive-pro/
[7] https://www.pimax.com/
[8] https://varjo.com/products/vr-2-pro/
[9] https://store.steampowered.com/steamvr?l=english

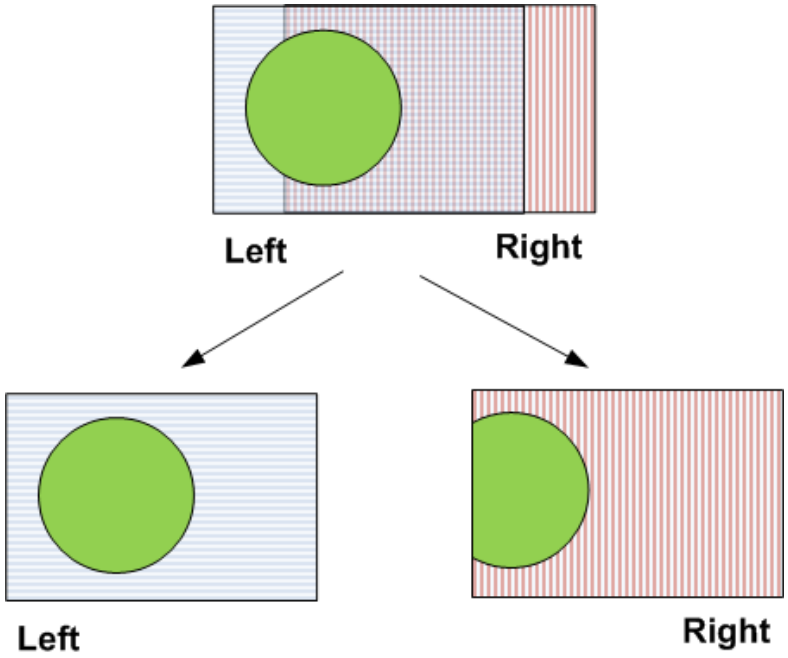


Figure 2.20: Visual overlaps between a pair of stereo images. For smaller field of views, objects might be fully visible in one image but only partially visible in the other. This can lead to binocular rivalry resulting in distraction. Reprinted from [Bog16].

2.4.1 Visual Overlaps

Stereo images for human vision exhibit a large amount of visual overlap, see Figure 2.20. A natural way of exploiting this phenomenon would be to reuse shading calculations of one eye for the other eye. However, care must be taken, since illumination of a pixel may contain view-dependent terms, e.g. glossy surfaces. Such shading information cannot be reprojected and reused since shading changes heavily when the viewpoint is varied. On the other hand, diffuse illumination is, depending on the chosen BRDF, view-independent and may be reused. Although Disneys principled BRDF for diffuse surfaces as stated in Equation 2.9 contains a term using the view direction, it generally varies slowly when changing the position of the viewer. Since the positions of both eyes are not far apart, the according view directions to the same point are similar enough such that diffuse illumination is approximately the same for both viewpoints. Differences between view directions and thus the respective diffuse illumination reach their maximum when an object is viewed up close. Despite the big overlaps, not every point of one image may be reprojected freely into the other one because some might be occluded or simply not visible. This generally leads to occluded regions where no shading information from the other image is available. Based on this observation, a major amount of diffuse illumination can be shared between both stereo images without introducing too big visual discrepancies.

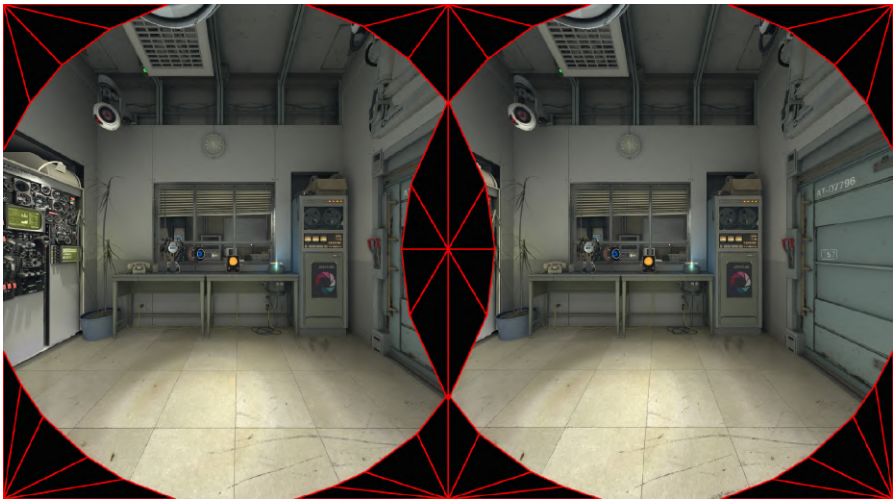


Figure 2.21: Simplified mesh to approximate the areas where pixels from the displays are not visible to a user due to the optics. The hole in the middle corresponds to the region which is typically occluded by the nose. Reprinted from [Vla15].

2.4.2 Invisible Pixels

The optical system - namely the lenses - of an HMD distorts the viewed displays such that certain regions near the borders of the rectangular screen are not captured by the lenses. The radially symmetric distortion leaves only a circular cutout visible. This essentially leads to regions of pixels which are wasted since they will not be visible from the perspective of the user. Depending on the actual HMD and its optical system, around 15% of the entire pixels are dead in this sense [Vla15].
To avoid loosing computational power due to rendering invisible pixels, these regions can be masked out before doing costly shading calculations. Alex Vlachos [Vla15] presents a simple technique to avoid shading a great amount of dead pixels by masking them out via the depth or stencil buffer. A mesh approximating the visible parts of the screen is rendered either to stencil out the pixels the user will not see, or the nearest depth value. Before execution of shading computations these pixels are rejected by the early stencil or early-z test respectively. An example of this so-called *hidden area mesh* is illustrated in Figure 2.21.

2.4.3 Uneven Sampling Rate and Foveated Rendering

Although displays in HMDs are rectangular, images are warped to match the characteristics of the optical system. The warping effect pushes the borders inwards towards the center and thus lets outer regions be denser sampled than inner ones. This imbalance in shading rate is necessary to counter the effects of lens distortion. This effect is visualized in Figure 2.22. However, human vision does not perceive its entire field of view as a sharp image, only the region that falls into the *fovea*. Everything else is blurred such

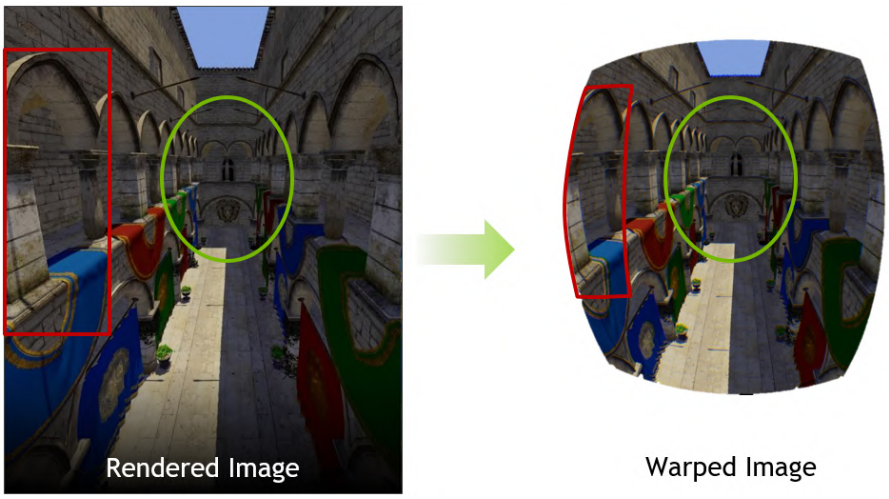


Figure 2.22: The rectangular image is warped to counter lens distortion effects. After the warping process, shading rate is unevenly distributed leading to overshading in outer regions (red) while inner regions almost stay untouched (green). Reprinted from [Nvia].

that minor changes are not noticeable. This leads to the observation that too much computational power is used to compute shading at regions which the user does not focus on, so-called overshading. Therefore, shading rate should be handled gaze-dependent, i.e. regions which are focused by the user should be sampled with a higher rate than peripheral regions. This is called *foveated rendering* and requires tracking of eye movements. Although there exist VR solutions including eye trackers[10], using the center of the image as a fixed focus point gives satisfactory results with decent performance improvements. *Fixed foveated rendering* is illustrated in Figure 2.23.

Leaving away limitations of the optical system, several methods have been proposed to adapt shading rate in a GPU-friendly manner to perform *fixed foveated rendering*. *Multi-Res Shading* [Nvia] tries to reduce overshading by splitting the image into multiple viewports, rendering those farther out with lower resolution. Although this effectively reduces shading rate, it also reduces the spatial sampling rate of the geometry leading to geometric aliasing. *Variable Rate Shading*[Nvic], introduced with the new Turing architecture of NVIDIA, was especially designed to give developers more choice in controlling the distribution of shading rate without losing the advantages of higher sampling rate during rasterization. Unfortunately, this new system is only applicable to hardware rasterization and not ray tracing. Alex Vlachos [Vla16] presents a different GPU-friendly solution by rendering the lower resolution region by skipping every other 2x2 pixel block creating effectively a checkerboard pattern. Missing pixels are retrieved by simple bilinear interpolation between neighboring rendered blocks.

[10] https://enterprise.vive.com/us/product/vive-pro-eye/

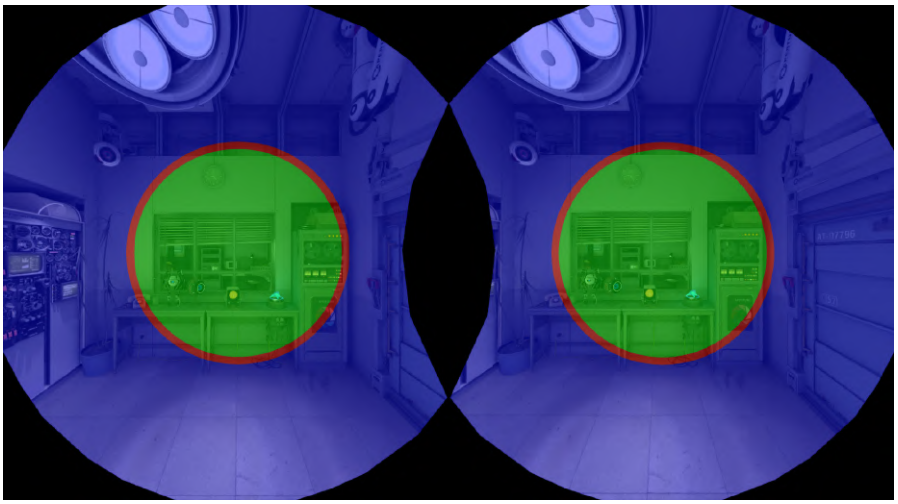


Figure 2.23: Fixed foveated rendering adapts the shading rate to focus on regions directly viewed shown in green. Over-shaded peripheral regions are shown in blue. Reprinted from [Vla16].

CHAPTER 3

Methodology

This chapter describes the methodological approach taken in this thesis. First, the overall pipeline is visualized and discussed. Then, light is brought into every single stage. Last but not least, an explanation and reflection of the various design decisions is given for each stage.

3.1 Pipeline Overview

The hybrid rendering pipeline developed in this thesis is inspired mostly by the works of Schied et al. [SKW+17] and Mara et al. [MMBJ17]. However, two additional stages only applicable in virtual reality setups are added. In addition, the number of overall paths traced per frame can be greatly reduced due to a novel screen space masking method giving a decent performance boost on current hardware. The overall pipeline is illustrated in Figure 3.1. The color of each stage corresponds to the type of hardware acceleration it uses: orange for rasterization, green for ray tracing and blue for compute. The single stages are explained in further detail in the upcoming sections along with a discussion of which design choices they provide.

The pipeline is essentially ran through twice, once per eye, although certain stages can be merged to reduce the overhead of invoking GPU computations for each eye. Furthermore, it requires twice the amount of GPU memory since most pipeline related resources are stored once per eye. This may easily lead to performance bottlenecks due to limited bandwidth (Chapter 5).

3.2 First Hit Rasterization

In an initial rasterization phase, the primary ray hits are rasterized and per-hit data is stored in the G-Buffer. Prior to that, the hidden area mesh introduced in Section 2.4.2

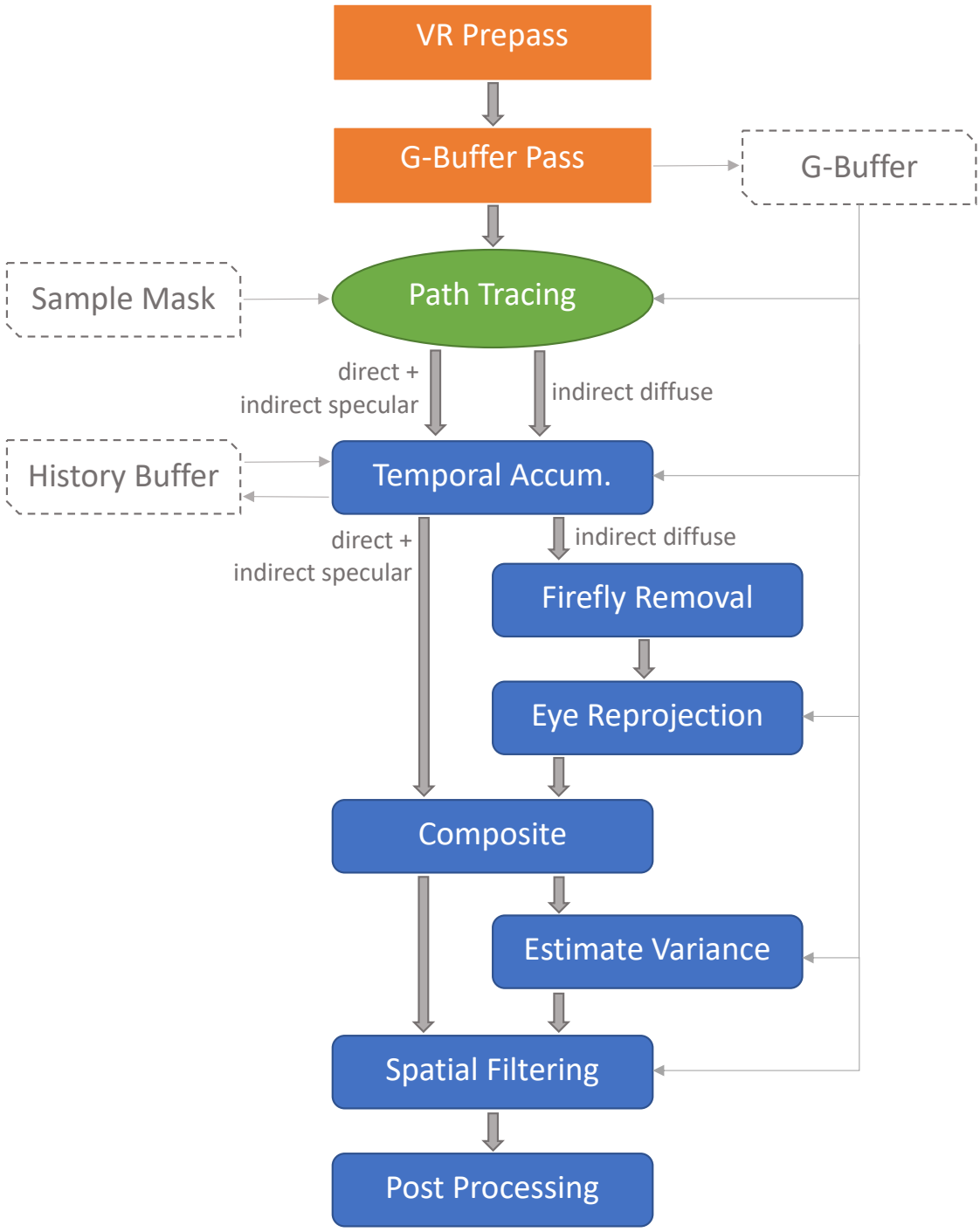


Figure 3.1: VR hybrid rendering pipeline developed in this thesis. The color of a stage denotes the type of hardware acceleration: orange for rasterization, green for ray tracing and blue for compute. Boxes with a dashed gray border represent important GPU resources which are read and written in various stages.

is rendered into the stencil buffer (*VR Prepass*) to prevent performing unnecessary computations in regions which will not be visible when displayed. During the *G-Buffer Pass*, all scene objects are rasterized and for each pixel, data about geometry, material and movement is stored. The full list of features captured in the G-Buffer is shown below:

- Geometric: position and normal
- Surface material: albedo, emissive color, roughness and Fresnel specular color (F0)
- Movement: motion vectors
- Filtering: linear depth and its maximum screen space deviation, mesh id

As explained earlier, a major advantage of rasterizing the primary rays is not only due to the performance increase but also because a noise-free representation of various feature buffers is obtained which is later utilized during spatio-temporal filtering.

3.3 Real-Time Path Tracing

To maintain real-time frame rates in VR rendering, only a small amount of rays per pixel may be traced on current hardware. Hence, in our method, for each pixel only three rays are traced:

1. One shadow ray towards a random light source to evaluate direct lighting of the primary hit point
2. One indirect ray sampled from the surface material
3. One shadow ray to evaluate direct lighting of the indirect hit point

The resulting path consists of a primary ray and a single indirect bounce together with their shadow rays. Direct contributions to surface hits are instantly evaluated via next event estimation described in Section 2.1.2. Therefore, only first bounce global illumination is achieved. This setup is similar to the work by Schied et al. [SKW+17].

To effectively capture the most important features of the integrand, those rays are sampled from representative distributions as explained in Section 2.1.4. Shadow rays are directed towards a sample point on a random light source in the scene. Indirect rays are sampled from the BRDF of the primary hit point. For composite BRDFs consisting of a diffuse and a specular term, for example the Disney principled BRDF, both lobes are sampled. Furthermore, the generation of random samples is driven by the correlated low-discrepancy sampler introduced by Heitz et al. [HBO+19] which distributes Monte Carlo noise in a blue-noise pattern.

Multiple importance sampling [Vea97] weighs sampling of the light source and the BRDF to effectively reduce variance in the long run. It turned out, however, that for such a low amount of samples per pixel, initial variance was actually increased. Therefore, multiple importance sampling was dropped as an additional noise reduction method in our work. In contrast to Schied et al. [SKW+17] this work does not make use of path space regularization presented by Kaplanyan and Dachsbacher [KD13].

Illumination is split to a *direct + indirect specular* component and an *indirect diffuse* component. Opposite to SVGF [SKW+17] this is just for separation to further process indirect diffuse illumination. Eventually, both components are recombined by addition in the *Composite* pass and the whole illumination is filtered at once.

3.3.1 Reduction of Traced Rays through Blue-Noise Masking

Similar to the hidden area mesh technique used for primary hit rasterization, only rays in visible regions need to be traced. Still, this would result in a dense tracing of rays, i.e. tracing a number of rays for each pixel. In the case of a static scene or a dynamic one with only slow movements, most pixels will stay visible for more than just a couple of frames after reprojection. In such scenes, per-pixel temporal history grows larger and larger giving an already high number of samples per pixel. On the other hand, since most of the reprojected pixels will stay visible for a period typically longer than the temporal accumulation period, new shading information is generated and aggregated although it might not introduce any changes.
This led to the observation, that the ray tracing rate per pixel can be reduced while temporal accumulation is still aggregating enough history. However, regions which were previously occluded will take longer to converge, causing visual artifacts due to the lack of shading information. The faster movements are, the larger those occlusion regions grow in a single frame. Although this constitutes a general problem of temporal reprojection and accumulation, it has an even more pronounced negative effect when further reducing the temporal shading rate in certain regions. Therefore, reduction of ray tracing and thus shading rate should be only considered in regions which already contain enough temporal history.

First, a spatial sampling pattern has to be determined to reduce the total number of rays traced per frame while still tracing each pixel at least once in a given period. Most regular patterns result in visual artifacts appearing artificially and thus being prominent in perception. The works of Georgiev and Fajardo [GF16] and Heitz et al. [HBO+19] showed that distribution of Monte Carlo error in a blue noise pattern gives visually more accepting images despite the noise. Similarly, we use a blue noise mask as sampling pattern to determine which rays should be traced in each frame. Therefore, a small tile-able blue noise texture is thresholded to result in n mutually exclusive masks which exhibit a uniformly distributed irregular pattern. Masks are then repeated to match the dimensions of the image. The union of all binary images covers the whole image,

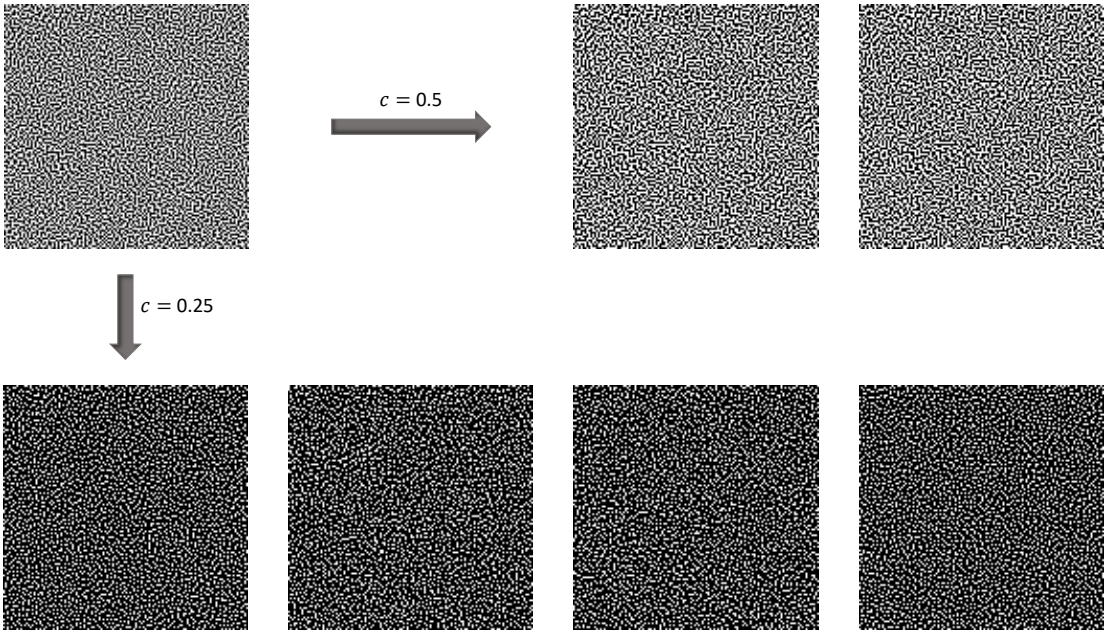


Figure 3.2: Thresholding of a 128x128 blue noise tile. Sample masks generated by 2 coverage factors, $c = 0.5$ and $c = 0.25$, are shown. White dots correspond to pixels traced, black dots represent *empty* pixels which have no shading information after path tracing.

hence alternating masks each frame and only tracing rays contained in the current mask lets every pixel be traced after n frames. Based on the number of threshold operations applied, various coverage factors with $c = \frac{1}{n}$ may be achieved, giving direct control of performance gained versus slow-down of convergence speed. Applying uniformly distributed thresholds does not result in masks containing the same amount of samples. However, these differences are so small that further balancing is not necessary. For an illustration of the resulting blue noise masks refer to Figure 3.2.

Assuming that pixels might be *empty*, this has to be taken into account in subsequent stages. Further assuming that empty pixels will have no color associated, i.e. they are black, previously occluded regions appear darker. To naively counter this issue, the intensity of a pixel $\mathbf{p}$ contained in the current mask for which reprojection failed is scaled by the inverse coverage factor c as follows:

$$I'(\mathbf{p}) = \frac{I(\mathbf{p})}{c} \tag{3.1}$$

Intensity is then distributed to neighboring pixels through temporal and spatial filtering. Unfortunately, this directly conflicts with the 3x3 median filter employed as firefly removal on indirect diffuse illumination. Therefore, the additional firefly removal stage is disabled if masking is enabled.
A more sophisticated approach to tackle this issue is to detect occlusions early – before

path tracing – and generate the sampling mask dynamically to include every previously occluded pixel. A more elaborate discussion concerning this approach is provided in Chapter 6.

Similarly, blue noise masks may intentionally cover certain regions better than others which we did not investigate further. An example might be to assure fast convergence rates in the area which is currently focused by the user. This corresponds to the previously introduced foveated rendering in Section 2.4.3 but instead of increasing the spatial shading rate, the temporal shading rate is not further reduced.

3.4 Sample Accumulation and Spatio-Temporal Filtering

The filtering pipeline constitutes the largest part of our rendering pipeline consisting of multiple different passes. Starting with *Temporal Accumulation*, samples are reprojected and temporally aggregated to effectively increase the number of samples per pixel. Again outputting a *direct* and *indirect* component as the path tracer did, lets the further stages only process indirect diffuse illumination. A 3x3 median filter is run to remove fireflies which usually result in the indirect diffuse illumination due to caustics [MMBJ17]. Following the *Firefly Removal*, pixels are projected to the other eye's view and accumulated if possible in the *Eye Reprojection* pass. Eventually, both illumination components are recombined in a *Composite* pass. Working with the full shading information, the *Estimate Variance* pass approximates the temporal or spatial variance used to guide the spatial filter. As a last stage in the filtering pipeline, *Spatial Filtering* via a 5-level à-trous wavelet transform using a custom set of edge-stopping functions is applied.

3.4.1 Temporal Accumulation

All pixels containing valid shading information are reverse reprojected into the previous frame using clip-space motion vectors. Motion vectors are calculated during first-hit rasterization using the objects current and previous transformation matrix. After reprojection, a pixel might land between the raster points of the previous frame, see Figure 3.3. Then, the reprojected point is tested against the four neighboring pixels contained in the history buffer. During validation, mesh IDs and world space normals are compared such that IDs must be equal and normals must be within a certain angle. To obtain a single color value for the previous frame, bilinear interpolation of valid pixels is performed.
In the case that no pixel passes validation checks, a small 3x3 cross-bilateral filter extends the radius in which similar samples are looked for. This especially pays off for thin structures which only occupy one to two pixels in width. The filter makes use of the same validation criteria, i.e. mesh ID and world space normal. If still no valid samples are found, history for this pixel is reset and only shading information from the current frame may be used.
Illumination from the history buffer is combined with illumination of the current frame

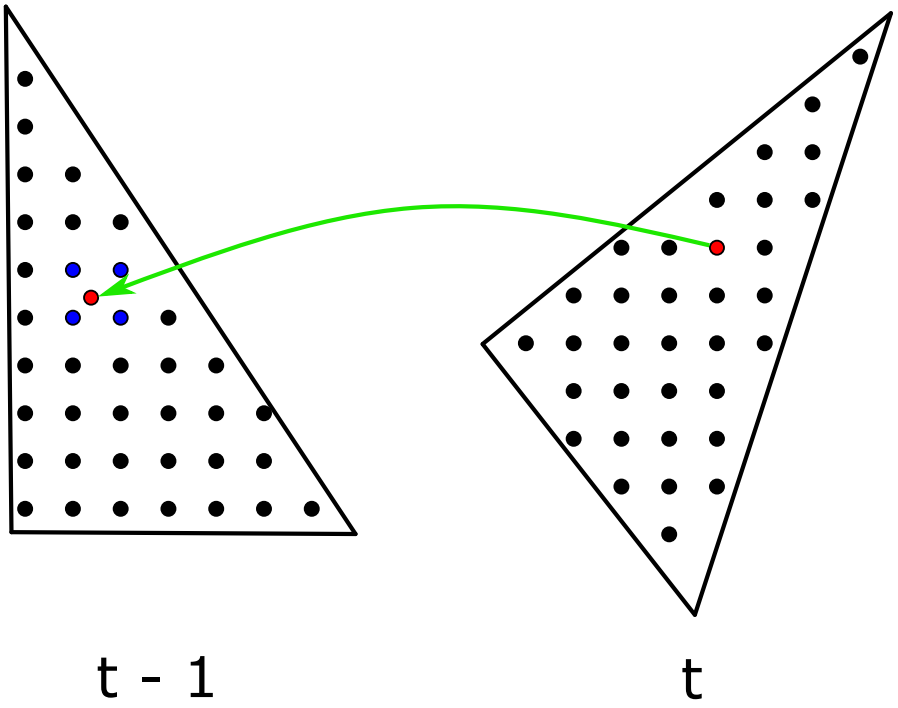


Figure 3.3: Reverse reprojecting a pixel of the current frame to the previous one might cause it to not perfectly land on the center of another pixel. Validation is then performed on the four closest pixels shown in blue. Eventually, the color of the reprojected point is bilinearly interpolated between all valid pixels.

via an exponential moving average. Experimentally, we found out that, if reprojection succeeded, a fixed aggregation factor of $\alpha = 0.2$, i.e. taking 20% of the current frame and 80% of history information, worked well. Although this effectively enables lag for reflections and shading for moving lights and observer, anti-lag techniques were not in the scope of this thesis, but will be further discussed in Chapter 6

Along with color information, moments for variance estimation and additional noisy features used for spatial filtering are accumulated. The original method (SVGF) [SKW+17] estimates variance based on the luminance of illumination. Instead, we employ CIE lightness (L^* in CIELAB color space) [Poy98] which better captures differences in dark regions, thus improving the stability of variance estimation in regions with bad lighting conditions. In addition, other features obtained during path tracing, inherently containing noise, may be accumulated. We make use of the indirect ray lengths, i.e. the distance which indirect rays traveled before hitting another surface, to further guide the spatial filter later on to reliably smooth out reflections without overblurring edges contained in them.

To keep track of the number of contributing samples, an additional counter is stored in the history buffer. This counter is later employed during variance estimation to check whether enough temporal information is available.

3.4.2 Additional Sample Accumulation for Indirect Diffuse Light

Indirect diffuse illumination proved to be the hardest to de-noise completely in our experiments. This is due to the almost hemispherical lobe that is sampled with only a single ray. Independent from the extreme amounts of noise, some indirect diffuse rays might connect to a specular surface on the secondary hit point. If this surface is aligned such that the diffuse ray is reflected towards a light source, the ray would carry a high amount of radiance. This is considered as caustics but unfortunately only appear as fireflies in low sample path tracing. To counteract flickering due to those rare bright spots, a spatial 3x3 median filter is employed in the *Firefly Removal* stage. A negative side effect is that caustics cannot be reconstructed and thus lead to an energy loss. As mentioned earlier, increasing illumination in previously occluded regions to deal with darkening when using blue noise masking conflicts with firefly removal because its local neighborhood will either contain black or extremely bright pixels.

Exploiting the overlaps between the stereoscopic views allows for sharing shading information. However, only view-independent illumination may be shared, i.e. the diffuse component. Due to the large lobe that is sampled, indirect diffuse illumination exhibits the largest amount of variance and noise.
Therefore, we propose in our *Eye Reprojection* stage to project pixels from one view into the other and accumulate indirect diffuse illumination if the reprojection succeeded. Validation follows the principles from temporal accumulation. First, rasterized points from the source view are transformed from world space to screen space of the target view. Similarly to temporal reprojection, the point might not land directly on a rasterized point and thus has to be bilinearly interpolated from the four neighboring pixels.
Although the diffuse component of some BRDFs, e.g. Disney's principled BRDF, includes a view-dependent factor, its contribution is low enough such that the discrepancy between the two view directions only plays a role if the object is viewed from very close. We experienced, however, that the subtle differences in indirect diffuse were not perceivable and thus ignored this fact to effectively double the sample count for successfully reprojected pixels.

3.4.3 Variance-driven Spatial Filtering

To guide the spatial filter, an estimate of per-pixel variance is computed during the *Estimate Variance* pass. Using accumulated moments and history count gathered during temporal accumulation, an estimation of temporal variance is given by Equation 2.22. If history count is not sufficient (< 4), a 3x3 cross-bilateral filter is employed to estimate variance spatially. The filter is guided by differences in depth, normal and mesh ID. As

mentioned earlier, variance is not directly estimated on color information, instead we utilize CIE lightness (L^* in CIELAB color space) which is still easy to compute but responds better to differences in dark regions than regular luminance.

In this work, we employ a hierarchical à-trous wavlet transform similar to SVGF but in order to save performance due to bandwidth limitation, our spatial filter works with the full path tracing output, i.e. direct and indirect illumination. Additionally, we introduce edge-stopping functions based on information obtained from the indirect bounce to prevent overblurring specular reflections. Due to correlated sampling, we do not loop back output from the first iteration into the history buffer.

In addition to the edge-stopping functions of SVGF discussed in Equation 2.20, we propose two additional weighting functions to specifically deal with glossy reflections:

$$\begin{aligned} w_{zi} &= \exp\left(- \frac{|z_i(\mathbf{p}) - z_i(\mathbf{q})|(1 - roughness(\mathbf{p}))}{\sqrt{g_{3x3}(Var(l(\mathbf{p})))} + \epsilon} \right) \\ w_r &= 0.01 + \sqrt{roughness(\mathbf{p})} \end{aligned} \tag{3.2}$$

where $z_i(\mathbf{p})$ denotes the accumulated travel distance of the indirect ray. w_{zi} penalizes blurring over depth differences visible through specular reflections. Since indirect ray lengths contain noise and are thus not optimal as a stable feature, the weight is further scaled by the variance estimate to make the weight negligible for noisy regions. w_r is only active for glossy BRDFs to increase blurring of reflections for rough surfaces to further avoid fireflies.

Last but not least, to avoid blurring of possible high-frequency color textures, like albedo, during spatial filtering, only untextured illumination is filtered and texturing is reapplied afterwards [SKW+17]. This albedo demodulation is performed during path tracing and albedo is remodulated after spatial filtering.

3.5 Post Processing

To wrap up the proposed pipeline, *Post Processing* is performed to deal with cheap anti-aliasing and tone mapping. Rendering at native resolution causes the center region of the image to be undersampled due to lens distortion. This region then exhibits geometric aliasing, i.e. undersampling of geometry, especially noticeable on thin structures. Post process anti-aliasing provides a cheap tool to mitigate some geometric aliasing problems. Fast approximate anti-aliasing (FXAA) [Lot09] runs a simple edge detection based on luminance differences to smooth out aliased edges. More sophisticated options include morphological (MLAA) [JME+11] and the recently popular temporal anti-aliasing (TAA) [Kar14].

Although post process anti-aliasing is rather cheap compared to the rest of the pipeline, it does not give good results for the difficult cases, e.g. a branch in the distance only appearing as a few disconnected pixels. Instead of reactively dealing with geometric aliasing, a more proactive style is desired. This can either be achieved naively by increasing the resolution, i.e. super sampling, or employing solutions which adapt and only use up more computational power where really needed. A method which fits this description and also fits a ray tracing based pipeline was presented by Marrs et al. [MSG+18] which extends TAA by tracing additional rays in critical regions. Unfortunately, we did not investigate further whether integration of this approach would be feasible in terms of maintaining interactive frame rates. Due to simplicity we employ FXAA as a fast post processing step to counter a wide range of simple problems caused by geometric aliasing.

Due to the high dynamic range of the path traced image, tone mapping is necessary to map intensities to a range of displayable colors. Since post process anti-aliasing might depend on linear intensities, tone mapping is performed as the last step in the pipeline. We tested various options of tone mapping algorithms, including Reinhardt and modified Reinhardt tone mapping [Rei02], Uncharted 2 filmic tonemapping [Hab10] and ACES filmic tone mapping [Nar]. Eventually, we opted for the latter one because it preserved contrast better than the other ones for our chosen scenes.

CHAPTER 4

Implementation

To test and evaluate the de-noising pipeline proposed in Chapter 3, a prototype was developed. The application is based on the *Falcor* framework [BYF+18] written in C++. We decided to use Falcor because it was the only open source framework meeting all the requirements necessary for this thesis:

- Customization of rendering pipeline
- Support for hardware ray tracing
- Tracking and rendering for VR
- Scene and material description along with loading utilities
- Simple UI for tweaking parameters
- Debugging and profiling tools

Another option for the implementation could have been *Unreal Engine*. However, at the point of development *Unreal Engine* provided only an experimental version of ray tracing but it proved too difficult to rewrite the entire rendering pipeline. Writing a custom engine for this purpose including all the above mentioned features would have been infeasible in the scope of this thesis.
Figure 4.1 depicts a screen shot of the developed prototype. Besides mirroring the current view of the HMD, a simple UI for parameter tweaking and profiling is shown. The following sections will provide more details on Falcor and implementation details of the proposed pipeline.

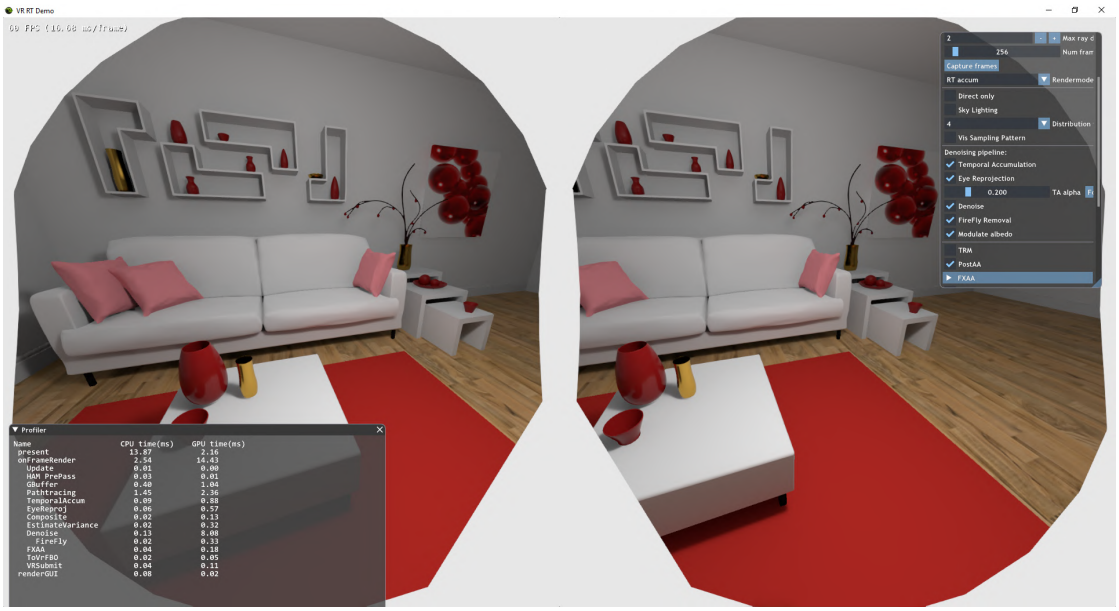


Figure 4.1: Screen shot of prototype application. The current view displayed in the plugged in HMD is mirrored. On top, a simple UI is rendered for parameter tweaking and profiling.

4.1 Falcor

Falcor is developed and maintained directly by NVIDIA where it is used for research and development. It provides an abstraction layer on top of *DirectX 12* and *Vulkan* supporting a range of helpful features as they are stated on the frameworks homepage[1]:

- Support for DirectX ray tracing (DXR)
- Physically-based shading system
- Thin abstraction layers on top of DirectX 12 and Vulkan
- Modular library of modern graphics techniques
- Stereo rendering using OpenVR
- Animation paths for cameras, lights, and meshes
- Support for common 3D asset formats, including FBX and OBJ
- Scene editor for quickly arranging meshes, placing lights and cameras, and authoring animation paths.

[1] `https://developer.nvidia.com/falcor`

Texture	R	G	B	A	Format
0	position			valid geometry	16-bit float
1	normal			-	16-bit float
2	albedo			-	8-bit unsigned int
3	emissive color			-	8-bit unsigned int
4	material ID	linear roughness	metalness	-	8-bit unsigned int
5	clip space motion vector		-	-	16-bit float
6	linear z	max z deviation	oct normal	mesh ID	32-bit float

Figure 4.2: Layout of G-Buffer using 7 render targets. Cells marked with a dash are empty and may be used for further material or geometric information. Due to bandwidth limitations, precision of textures was reduced to the minimum without introducing visual artifacts. In texture 6, *oct normal* denotes an octahedron normal vector [MSS+10].

A number of high-quality scenes in the Falcor scene format are freely available such as the Amazon Lumberyard Bistro [Lum17], NVIDIA Emerald Square [HB17] or Unreal Engine Sun Temple [Gam17].

During development, some issues related to VR rendering occurred, which were eventually fixed in a custom version of the framework. Examples of those problems include view-frustum culling only considering the left eye or a slight lag due to the delayed submission of rendered images to the HMD.

4.2 Implementation Details of De-noising Pipeline

Figure 3.1 states how the various stages of our pipeline use hardware acceleration. Rasterization is only used for the first two stages to obtain information about the primary ray hits. Due to the optics of an HMD some pixels will never be visible and thus can be ignored during rendering. This is accomplished by rendering the hidden area mesh introduced in Section 2.4.2 into the stencil buffer, only letting pixels in the visible areas pass the stencil test. The hidden area mesh is provided by OpenVR based on the currently plugged in HMD.
Afterwards, the scene is rasterized for each eye individually and primary hit information is stored in the G-Buffer. Although, computational cost of rasterization can be reduced via layered rendering using a geometry shader or *Single Pass Stereo* [Nvib], this stage was not considered as a limiting factor of the pipeline as will be shown in Chapter 5. During the G-Buffer pass, textures are sampled with trilinear filtering, i.e. bilinear filtering in addition to linear interpolation between the nearest mipmaps. Information needed in further stages is stored in various textures contained in the G-Buffer. The exact layout of the G-Buffer is shown in Figure 4.2. This layout is not tightly packed but it was optimized such that each stage uses the least amount of texture reads possible.

Path tracing is based on the recently introduced *DXR* API which is further integrated and abstracted in Falcor. A ray generation shader first evaluates direct light by sampling a random light source and computes its direct contribution. The developed prototype only supports sphere and sky (hemisphere) lights, but in theory other types of area lights may be used as well. In addition, a shadow ray is traced towards the same sampled point. Shadow rays only test whether a surface is hit, i.e. they only consist of a miss shader. To estimate indirect illumination, one indirect ray is traced based on the material. In the case of a composite or layered material, like the Disney principled BRDF, one indirect ray per component or layer is traced. Hit points of indirect rays are shaded exactly like primary hit points including a single shadow ray. Indirect hit points require the surface material to be resampled. However, in contrast to primary hit points, these material textures can only be sampled using bilinear filtering since there is no intrinsic mechanism, like screen space derivatives for example, to determine the required mipmap level. We employ bilinear filtering, although this effectively may introduce texture aliasing and decrease performance due to poor texture cache usage. Better solutions to this problem are described by Akenine-Möller et al. [HAM19, Chpt. 20].
Eventually, direct and indirect specular illumination is calculated and stored in a single texture. Indirect diffuse illumination along with indirect ray lengths are stored in a separate texture for further processing.
If blue noise masking is enabled, only pixels contained in the current mask are traced. A 128x128 blue noise texture is generated offline according to Georgiev and Fajardo [GF16]. In a preprocess step, blue noise sampling masks are generated and converted into GPU buffers containing linear indices of pixel positions. These buffers are then read in the ray generation shader and converted into 2D pixel coordinates. Storing and using linear indices instead of screen space textures allows full utilization of all threads within a compute unit since no invocations for empty pixels occur.

During temporal accumulation new information coming from the previous stages is mixed with history accumulated over several frames. The *history buffer* is comprised of five textures:

- Direct illumination (rgb) and history length (a)
- Indirect illumination (rgb)
- Moments for variance estimation (rg)
- Validation data consisting of linear depth (r), octohedron normal vector (g) and mesh ID (b)
- Extra features like indirect ray length (r)

After successful reprojection the history buffer consists of mainly accumulated data with the sole exception of validation data where only information of the current frame is stored.

Before illumination components are recombined and spatial filtering is performed, indirect diffuse illumination is prefiltered to remove fireflies. This is done by a 3x3 median filter which is implemented according to the fast and GPU-friendly version presented by Morgan McGuire [McG08]. Afterwards, pixels from one view are projected into the other by transforming world space positions stored in the G-Buffer of one eye with the view-projection matrix of the other eye. Similar to temporal accumulation, samples are accumulated if validated successfully. To maximize differences in sample placement, the index within the low-discrepancy set is offset by 64 for one eye.

After recombination of illumination components and variance estimation, the spatial filter implemented in a compute shader is invoked five times with increasing kernel sizes. Due to the high number of required texture reads, the spatial filtering pass is considered as strictly bandwidth limited. As mentioned earlier, textures contained in the G-Buffer are reduced in precision to reduce the performance loss without introducing noticeable artifacts.

For post processing, *Falcor* provides a list of already implemented and integrated effects. Post process anti-aliasing offers FXAA and TAA. We decided to use FXAA due to its simplicity and overall good results. TAA sadly suffered from overblurring of edges which was even more apparent in VR. Furthermore, *Falcor* includes a list of tone mappers which can easily be switched during runtime. For most scenes, the ACES filmic tone mapper [Nar] gave the best results.

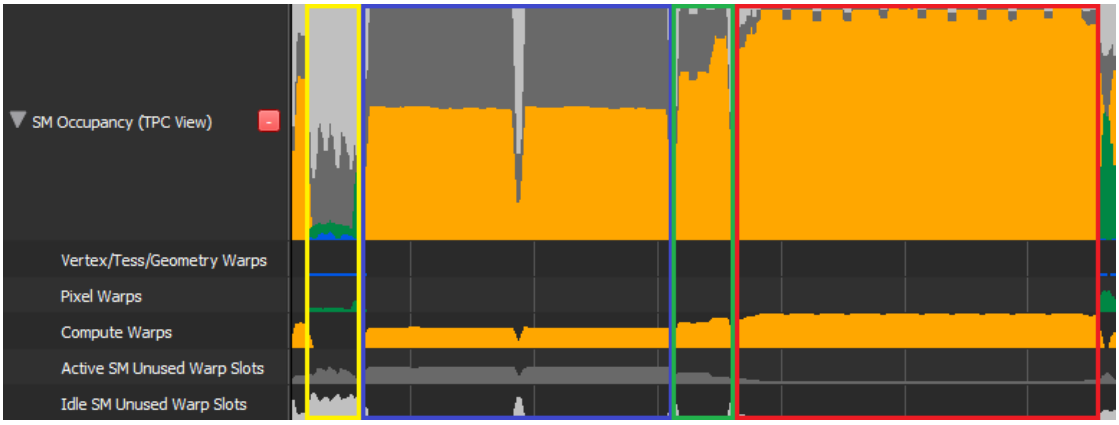


Figure 4.3: Snapshot of occupancy of streaming multiprocessors (SM) in *NVIDIA Nsight Graphics*.

4.2.1 GPU Occupancy

Figure 4.3 gives an overview of the actual occupancy of GPU processors of a sample scene. Grey regions depict idle threads indicating that some stages are not fully utilizing available processors or are bound by other factors such as bandwidth. During rasterization (yellow) the GPU mainly performs texture lookups and copies certain information into the G-Buffer. Although this uses bandwidth almost to its capacity, processors are not performing many computations and thus around 50% of streaming multiprocessors are completely idle. Path tracing (blue) performs a lot of computation on regular compute threads. While RT cores are used to its capacity (not visible in Figure 4.3), there are some idle compute threads. This suggests that shading computations are stalled by ray tracing operations performed on the RT cores. Temporal reprojection and eye reprojection (green) utilize roughly 70-80% of available compute threads. Due to conditional spatial filters, some threads may take more time to complete and thus stall all other threads in this warp. Eventually, spatial filtering (red) almost fully utilizes processing power. Bandwidth limitation was almost completely eliminated for spatial filtering due to reduction of precision of all textures in the G-Buffer.

CHAPTER 5

Results and Limitations

In this chapter, the developed prototype undergoes an extensive evaluation. Multiple scenes with varying complexity in geometric details and lighting conditions are used to evaluate performance and flexibility. Visual and quantitative comparisons of selected views are given to demonstrate which regions proved to be problematic to de-noise. Moreover, proposed contributions of this thesis are presented and compared visually. In addition, frame times and memory usage are discussed. Last but not least, focus is put on the consequences of design decisions and general limitations of the proposed methodology and protoype.

5.1 Comparison of De-noised and Reference Rendering

Before going into detail about results and evaluation, a short note on the way how renderings were generated is necessary. For better visibility and comparability, a few changes were made:

- Most renderings will not include the hidden area mesh optimization in order to show the full image. However, runtimes are measured with the prepass enabled.
- Renderings shown correspond to the view of a single eye and if necessary both views are shown. Otherwise, only the view for the left eye is present.
- HMDs require the screens to be rendered using an off-axis projection and a bigger field of view as usual. For illustration purposes only, a regular perspective projection using a smaller field of view is used.
- Any post processing is omitted. This includes pipeline specific post processing, but also correction of lens distortion and chromatic aberration carried out by the VR API.

For evaluation four scenes were chosen, showcasing different lighting conditions and varying geometric complexity:

1. **RedRoom** contains an indoor scene with a single light source and is comprised of 449783 vertices and 786058 triangles. Surfaces are made out of diffuse, metallic and non-metallic materials using Disney's principled BRDF. The scene was adapted from Schied et al. [SKW+17].

2. **CornellBox** follows a simple but well-known rendering scene with slight adaptations. It contains 8090 vertices and 16164 triangles together with a single light source. Materials are mostly chosen to be diffuse-only with high roughness except for the metallic sphere in the middle.

3. **Sponza** shows the popular Crytek Sponza variant [McG17] including 139952 vertices, 239727 triangles and 2 light sources. Most surfaces are considered to be diffuse. The material of curtains varies from diffuse for the fabric to metallic for the embossing.

4. **Bistro** contains a reduced version of the interior of the Amazon Lumberyard Bistro scene [Lum17]. With 453268 vertices, 806540 triangles and 3 light sources it is the most complex scene used for testing and evaluation. It includes a variety of materials including diffuse-only, rough metallic and non-metallic surfaces.

An equal time rendering of the **RedRoom** scene is shown in Figure 5.1 to further motivate the use of de-noising. Tracing one additional sample per pixel takes the same amount of time as de-noising the initial single sample image. However, the path traced image still contains immense amounts of noise. To completely remove noise, hundreds to thousands of samples would be required which defeats the purpose of real-time rendering.

When looking at the de-noised image closely, a few issues can be immediately spotted. These problems are better visible when comparing the de-noised image with the reference path tracing shown in Figure 5.2. Note that reference images still make use of rasterization of primary rays in order to obtain equal amounts of geometric aliasing.
At a first glance, the difference in brightness constitutes a major issue. Since we rely only on a single indirect bounce, energy is lost during the process. As can be observed in Figure 5.3 this mainly happens for scenes containing bright surface colors such that more light is reflected instead of absorbed by surface materials. The red rectangle in Figure 5.2 shows this effect in regions only illuminated indirectly. Closely related is the effect of color bleeding where reflected light transports color information to a nearby hit surface as it is observable in the blue rectangle. Although this effect is basically visible using only a single bounce, it is not as pronounced as if the full paths are evaluated. A further limitation due to the reduced number of bounces is depicted in the green rectangle. Specular surfaces obtain the majority of their shaded color through indirect

Figure 5.1: Equal time comparison of the **RedRoom** scene. Path tracing with 2 samples per pixel (left) takes the same amount of time (~25ms) as de-noising a single sample (right) but contains strongly visible noise.

light. However, when the secondary hit point is also only indirectly lit, i.e. lies in shadow, it appears dark in the reflection.

While all the previously mentioned issues are due to the limited recursion depth of a path, the yellow rectangle shows an actual issue of the proposed de-noising pipeline. Sample validation during temporal and spatial filtering rely mainly on geometric properties such as depth and normal. The baseboard visible in the yellow rectangle, however, constitutes a problem because depth varies non-linearly around the tube-like structure and normals change constantly on curved surfaces. Therefore, certain regions may exhibit properties which make it very hard or even impossible to accumulate enough samples in the temporal and spatial neighborhood in order to completely get rid of visible noise.
Another problem that is caused by the spatial filtering is overblurring of shadows in darker regions, for example visible on the ceiling of the **Bistro** scene shown in Figure 5.4. This is mainly due to combining direct and indirect illumination to save bandwidth, thus reducing the effectiveness of the variance-guided luminance weight during filtering.

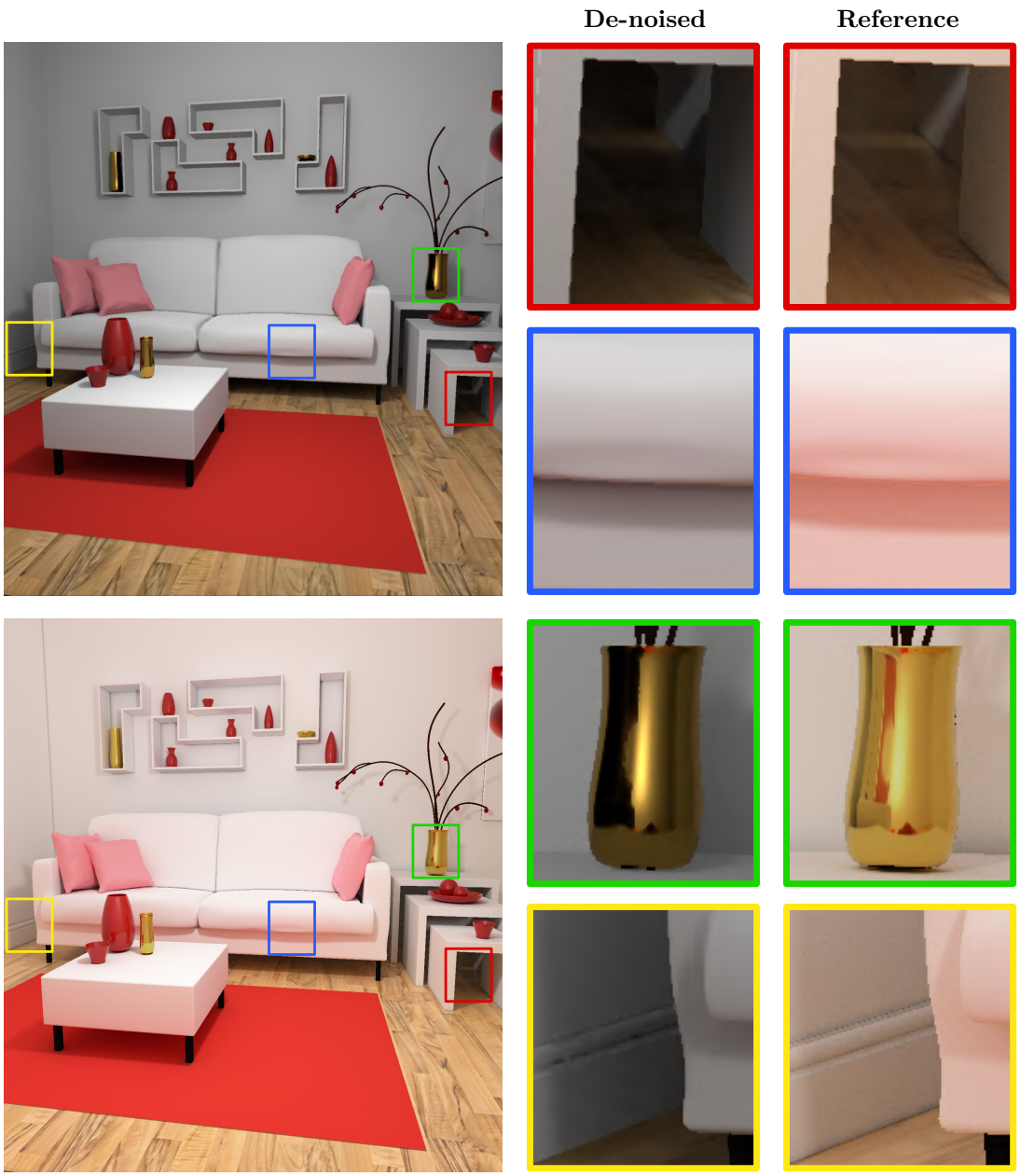


Figure 5.2: Comparison of de-noised rendering (top left) with reference image (bottom left) of the **RedRoom** scene. The reference was obtained by offline path tracing using 1024 samples per pixel. Computation of the de-noised image took about 25 milliseconds whereas the reference image took roughly 10 minutes on the GPU.

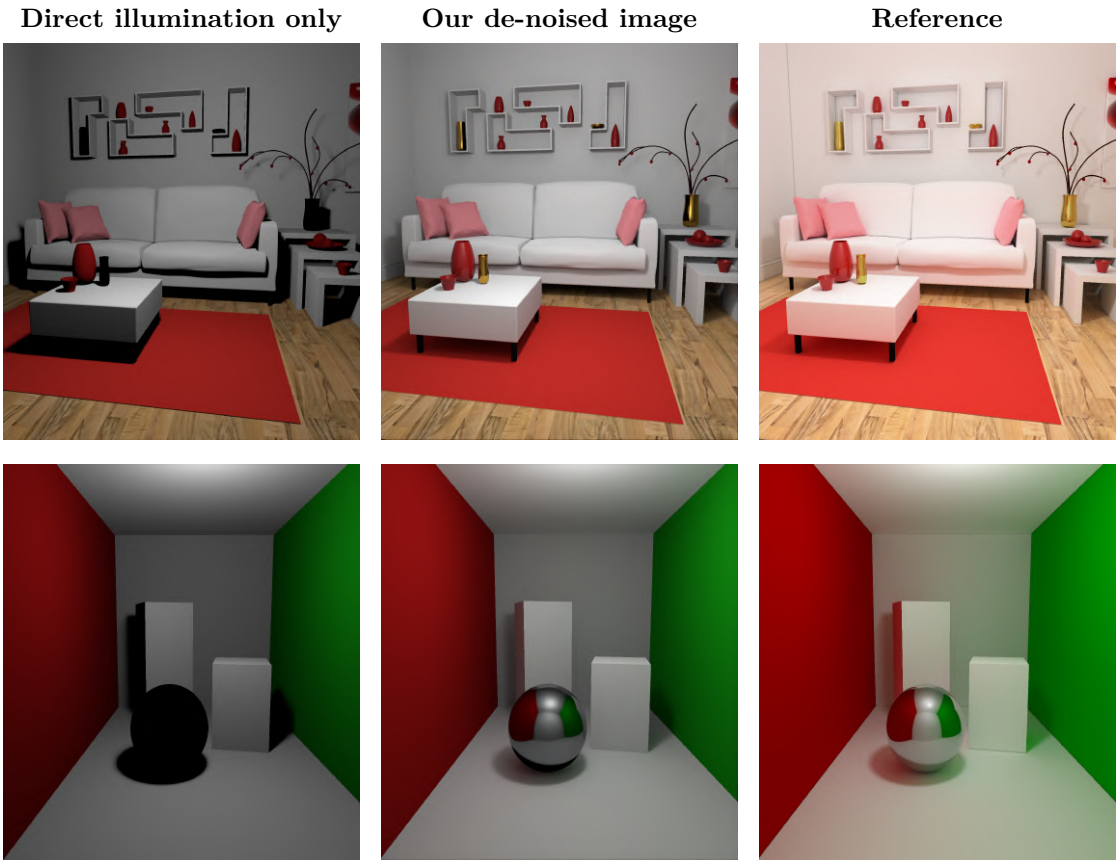


Figure 5.3: Comparison of **RedRoom** (top) and **CornellBox** (bottom) scenes. From left to right: direct illumination only, our de-noised image with one indirect bounce, reference image with 1024 samples per pixel. Execution times for our de-noised image are around 20-25ms and between 8 and 10 minutes for the reference image. Note, that metallic surfaces appear black in the most left image since their shading is mainly comprised from a direct specular highlight as well as indirect illumination.

To further evaluate differences between the proposed de-noising pipeline and a regular reference path tracer, images are compared using a number of metrics such as *Mean Square Error (MSE)*, *Structral Similarity Index (SSIM)* and *HDR VDP* [NMDSLC15]. The latter one gives a more perceptional measure of how noticeable differences between the two images are. An overview of this quantitative evaluation is given in Table 5.1. Scenes with intense color bleeding and stronger indirect illumination effects tend to differ more than darker scenes. **RedRoom** and **CornellBox** fall in the former category and show bigger differences, especially in indirect illumination, color bleeding and brightness of reflections. **Sponza** and **Bistro** contain materials which tend to be overall darker, thus absorbing more light with each bounce. Nonetheless, all scenes reach a score of around 80 for the HDR VDP quality which has a theoretical range of $[0, 100]$ with higher values marking less perceptible quality degradation.

Figure 5.4: Comparison of **Bistro** (top) and **Sponza** (bottom) scenes. From left to right: direct illumination only, our de-noised image with one indirect bounce, reference image with 1024 samples per pixel. Execution times for our de-noised image are around 30-40ms and between 12 and 14 minutes for the reference image. Note, that metallic surfaces appear black in the most left image since their shading is mainly comprised from a direct specular highlight as well as indirect illumination.

	MSE	**SSIM**	**HDR VDP**
RedRoom	0.0428	0.6258	81.7200
CornellBox	0.0087	0.6918	82.6606
Sponza	0.0017	0.9441	81.8759
Bistro	0.0028	0.8990	80.0415

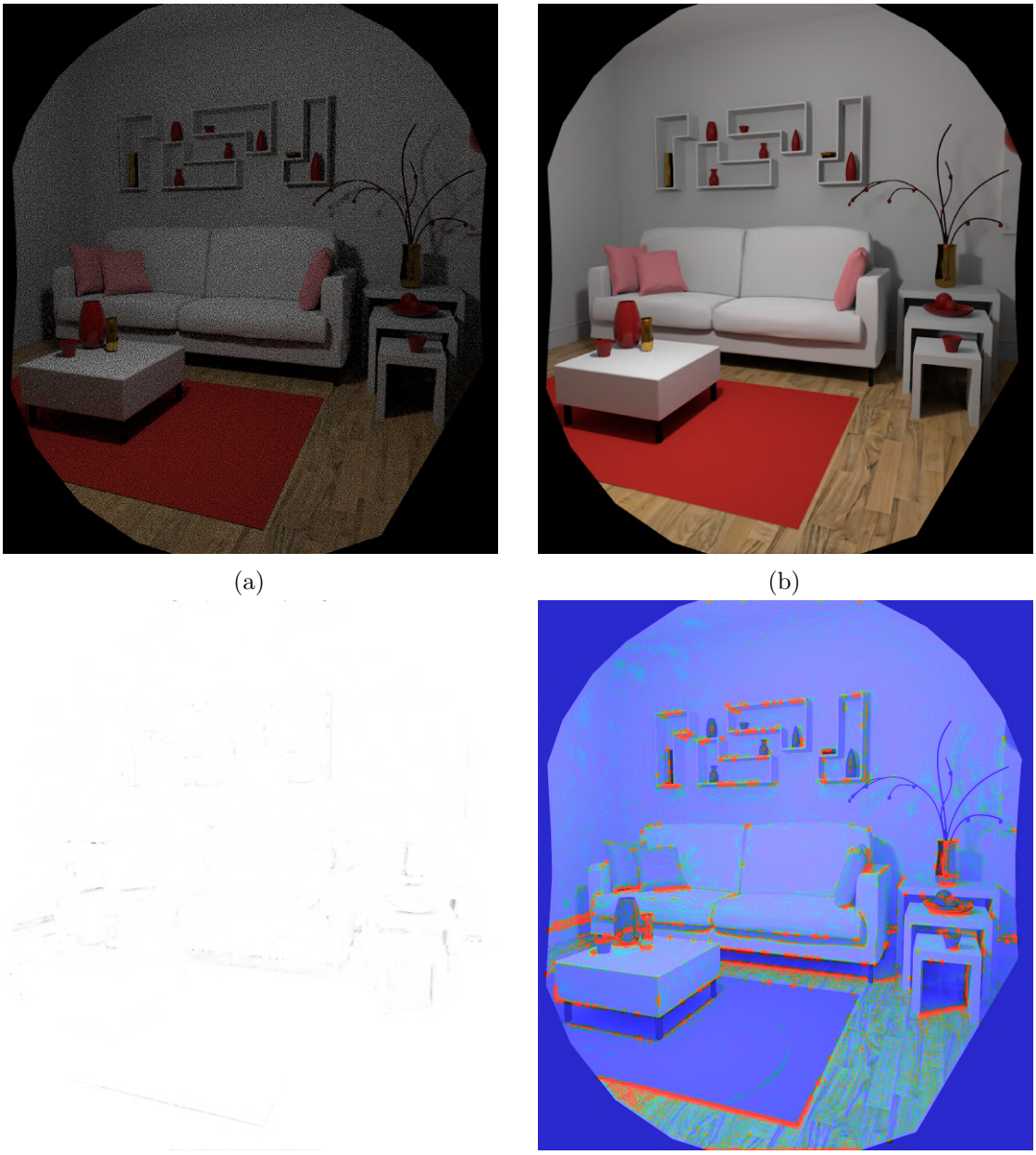
Table 5.1: Mean Square Error (MSE), Structural Similarity Index (SSIM) and HDR VDP Quality to measure differences between de-noised image and its reference for all tested scenes.

Coverage factor	MSE	SSIM	HDR VDP
0.5	2.2023e−5	0.9978	86.9189
0.25	5.578e−5	0.9956	86.1223
0.125	1.446e−4	0.9905	85.4053

Table 5.2: Mean Square Error (MSE), Structural Similarity Index (SSIM) and HDR VDP Quality to measure differences between the fully traced denoised image and denoised image with blue noise masking of the **RedRoom** scene. The high values for SSIM and HDR VDP quality (both max. 100) indicate that there are no noticeable artifacts with blue noise masking with little to no movement.

5.2 Evaluation of Blue Noise Masking

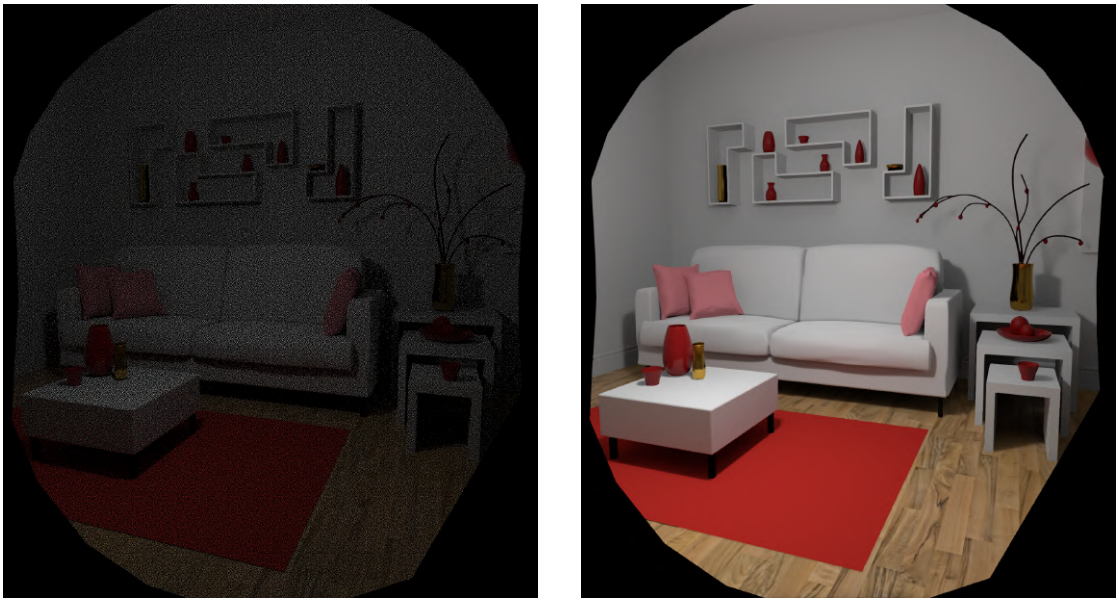
The novel blue noise masking system which substantially reduces the number of traced rays per frame is evaluated in two steps. First, little to no movement is assumed and it is investigated whether any visual artifacts appear. After that, movement of objects and the observer is added and again evaluated. In addition, it is shown how artifacts apparent during movement can be mostly mitigated.
Figure 5.5, Figure 5.6 and Figure 5.7 show raw path traced output with varying coverage factors and their respective de-noised images. Alongside, a visualization shows differences according to SSIM and HDR VDP whereas dark regions in the SSIM map correspond to bigger visual differences and red regions in the HDR VDP map denote differences which are detectable by users. It can be observed that most differences are located at shadow boundaries and reflections. The according measured differences are listed in Table 5.2.
During movement, the temporally decreased shading rate due to masking leaves behind dark artifacts in regions which were previously occluded. By increasing the intensity of newly traced rays in these regions - see Equation 3.1 - these artifacts are mostly avoided and only visible during very fast movements. Comparisons with object and observer movement are given in Figure 5.8.



(a) (b)

(c) (d)

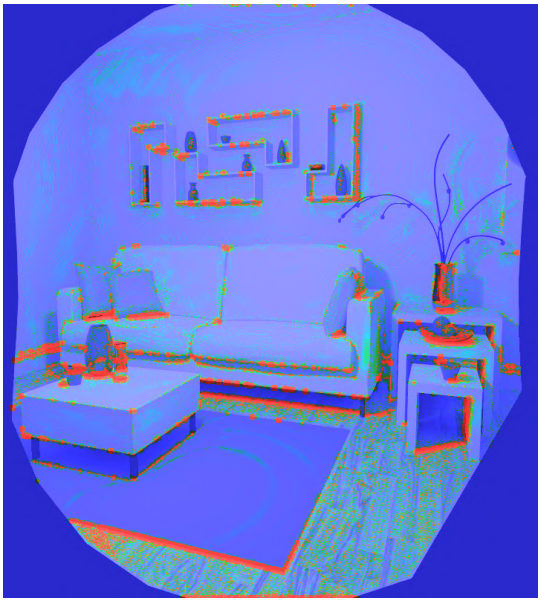
Figure 5.5: Denoising with coverage factor of 0.5. Top left (a) shows the reduced output of the path tracer. The denoised image is visible in the top right (b). SSIM and HDR VDP maps, with respect to the fully traced and de-noised image, are visualized in (c) and (d) respectively.



(a)

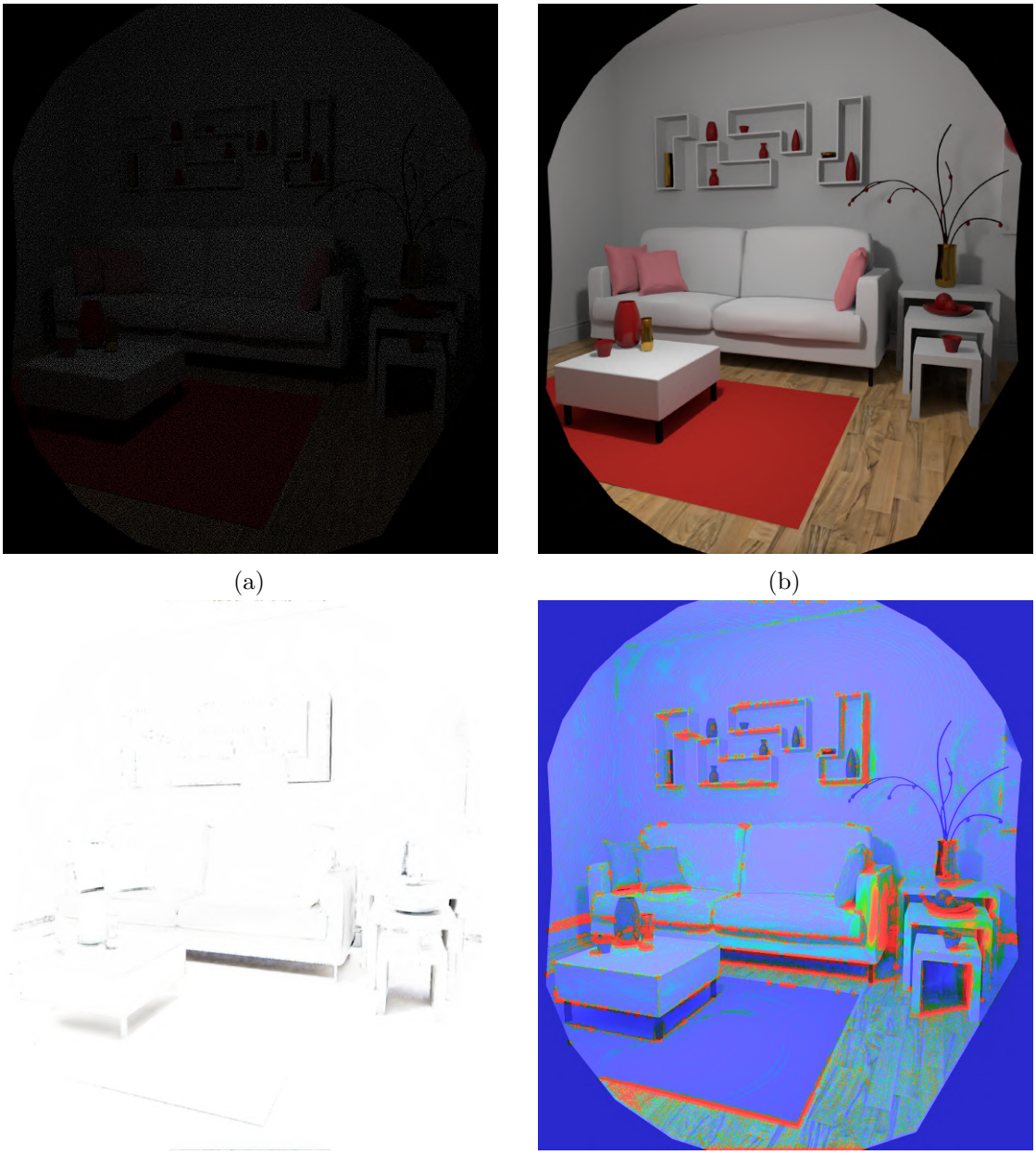
(b)

(c)



(d)

Figure 5.6: Denoising with coverage factor of 0.25. Top left (a) shows the reduced output of the path tracer. The denoised image is visible in the top right (b). SSIM and HDR VDP maps, with respect to the fully traced and de-noised image, are visualized in (c) and (d) respectively.



(a) (b)

(c) (d)

Figure 5.7: Denoising with coverage factor of 0.125. Top left (a) shows the reduced output of the path tracer. The denoised image is visible in the top right (b). SSIM and HDR VDP maps, with respect to the fully traced and de-noised image, are visualized in (c) and (d) respectively.

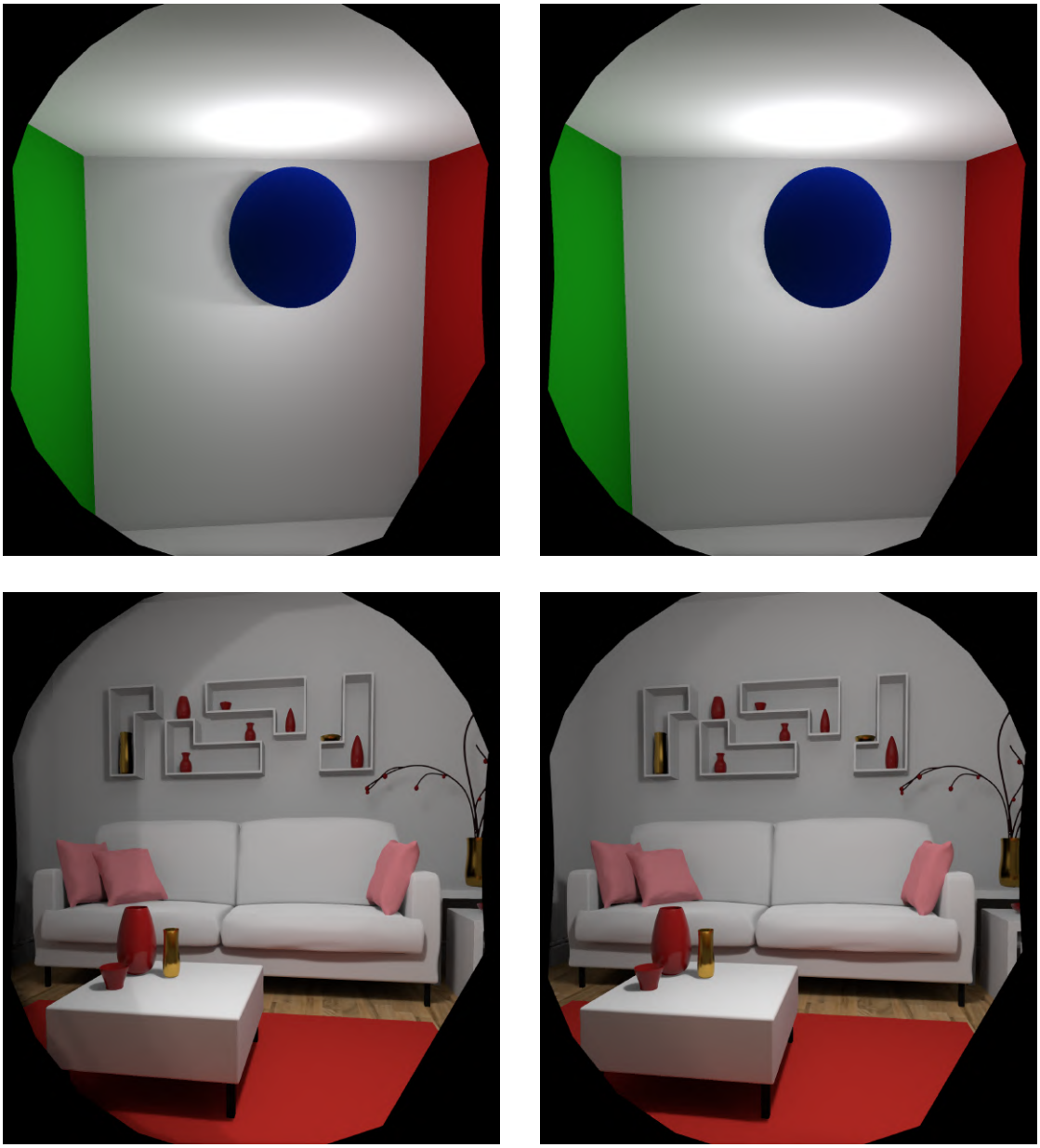
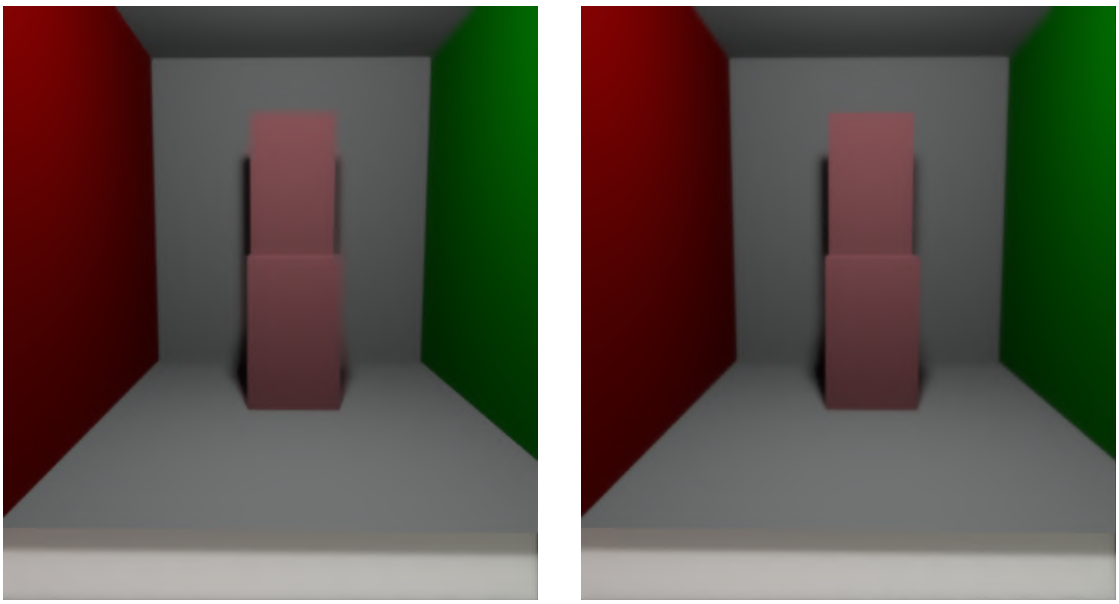


Figure 5.8: Images rendered using blue noise mask with coverage factor of 0.25. When objects move (top row) or the camera is rotated to the left (bottom row), new regions become visible. Without increasing the intensity of traced pixels which were previously occluded, movement leaves a dark shadow behind due to *empty* pixels in between them considered as valid samples during filtering (left). Boosting the intensity by the inverse of the coverage factor (Equation 3.1) which is then distributed during filtering mitigates a large amount of darkening (right).

5.3 Evaluation of Edge-Stopping Functions for Glossy Surfaces

A further contribution of this work consists of a more faithful reconstruction of glossy reflections. Previous work mostly uses features of primary hit points during spatial filtering while we also make use of information of the indirect rays and their hit points. We propose two additional weight functions (see Equation 3.2) to avoid overblurring of depth edges visible through reflections. A visualization of proposed weights and a comparison of filtering with and without them is shown in Figure 5.9. It can be observed that color is blurred across depth edges visible through an almost specular reflection. Our additional edge-stopping functions avoid this unwanted blur depending on the roughness of the surface. However, for certain values of roughness near zero, this might prevent filtering although there is a slight amount of noise present.



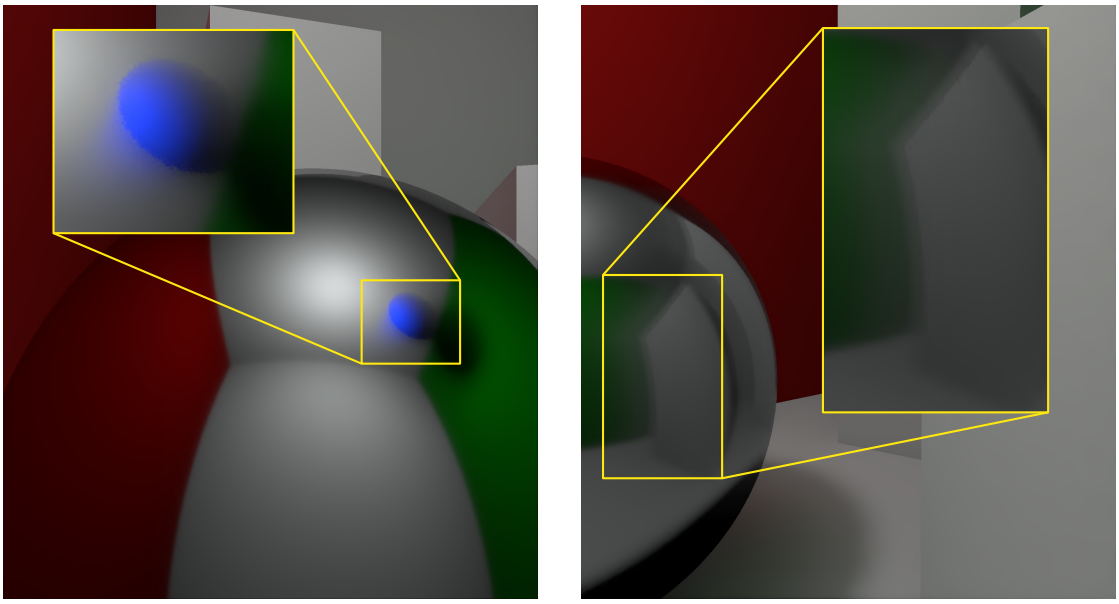
(a) (b)

Figure 5.9: Scene viewed from the reflection of a specular surface with low roughness. Denoising without proposed edge stopping functions yields slight ghosting and overblurring across edges in the reflection (a). Employing additional weights removes mentioned artifacts (b).

5.4 Further Limitations

Two more limitations are given that state a problem and may distract the user. These issues are not inherent to the pipeline structure but rather constitute problems that appear in single stages. Therefore, they are not considered as major issues such as the limited number of bounces or overblurring of shadows.

First, temporal accumulation takes only geometric information into account. Reprojection of pixels showing a reflective surface uses motion vectors according to that surface. However, reflected points move differently than the surface they are reflected in. This leads inevitably to ghosting since no shading-related information is employed during sample validation. The problem is illustrated in Figure 5.10 for object and camera movement.
Second, lag occurs for lighting changes, e.g. for a moving light source. When a light moves and a pixels brightness decreases, it actually lags behind a bit. This is best observed in specular highlights which are then stretched out.



(a) (b)

Figure 5.10: Pixels that contain specular reflection need extra attention during temporal reprojection. Reflected pixels are not reprojected correctly when using per-pixel motion vectors of the reflectors and are not invalidated since only primary hit information is taken into account. This leads to ghosting when objects (a) or the observer (b) move while viewing a specular surface.

Rendering at the native resolution of an HMD display causes visible geometric aliasing which is even magnified by the big field of view and lens distortion in VR. Post process anti-aliasing is not sufficient, which can be seen in Figure 5.11. The thin structures are undersampled and contain holes. This is impossible to resolve in a post process step without higher resolution during rasterization. However, increasing the resolution would dramatically reduce performance since path tracing and filtering are considered as the main bottlenecks.

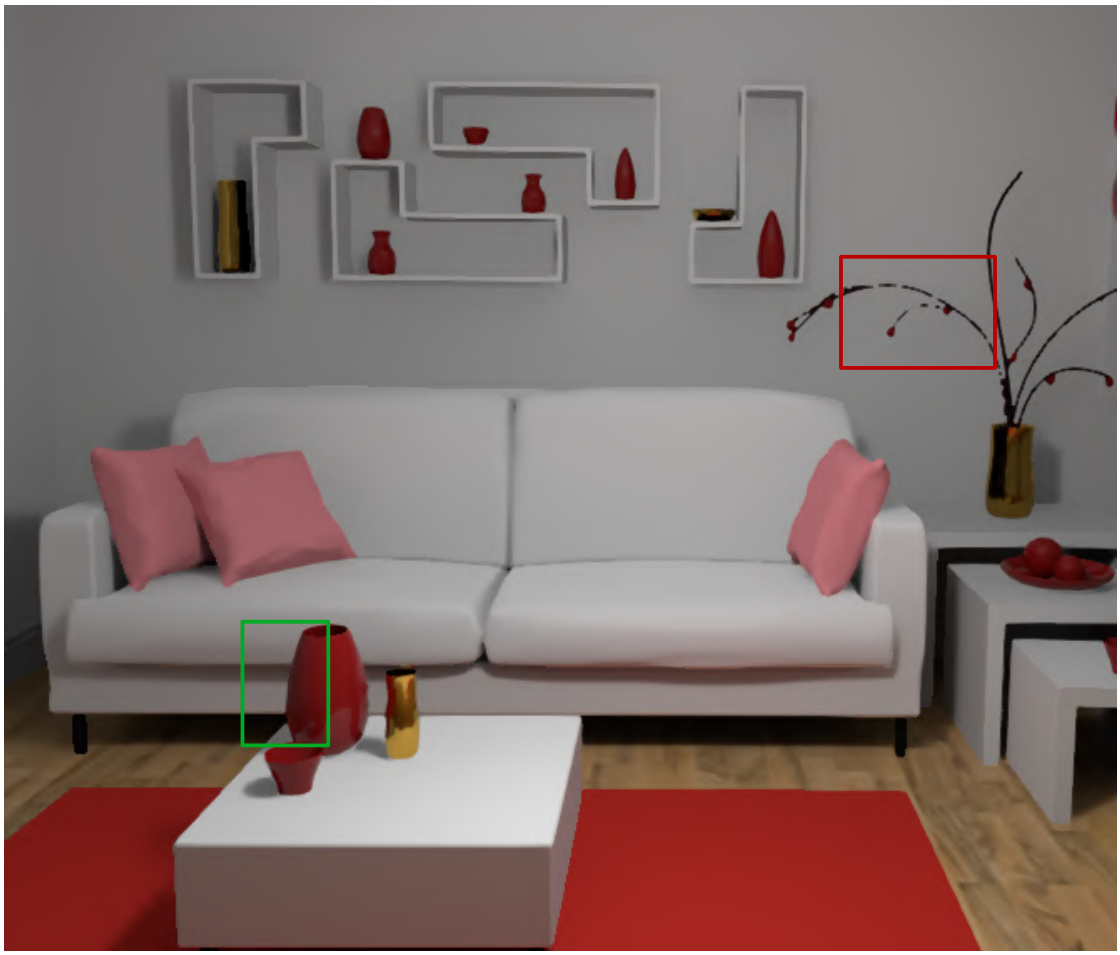


Figure 5.11: Post process anti-aliasing achieves decent quality in some regions, e.g. green rectangle, but completely fails in others, e.g. red rectangle. Undersampling happening during rasterization cannot be countered in a post process stage anymore.

	Scene			
Pipeline Stage	**RedRoom**	**CornellBox**	**Sponza**	**Bistro**
G-Buffer + Prepass	1.40	0.65	1.53	1.60
Path Tracing	8.73	5.40	24.10	17.13
Temporal Accumulation	1.13	1.10	1.28	1.25
Eye Reprojection	0.95	1.00	1.03	1.02
Firefly Removal	0.49	0.48	0.48	0.48
Composite	0.23	0.22	0.30	0.28
Variance Estimation	0.36	0.34	0.35	0.35
Spatial Filtering	9.87	9.92	9.92	9.62
VR Submission	0.20	0.20	0.20	0.20
Total Frametime (ms)	23.36	19.31	39.19	31.93
Frames per Second	42.81	51.79	25.52	31.32

Table 5.3: Averaged frame times measured for each pipeline stage in milliseconds. The majority of stages is independent of scene complexity. Some stages such as temporal accumulation or variance estimation may vary a bit due to conditional spatial filters.

5.5 Runtime

In addition to the evaluation of visuals, frame times and memory usage were recorded for the presented scenes. Although variances in frame time may occur due to faster ray tracing depending on the current view, selected renderings were considered as almost worst case scenarios when it comes to ray tracing, i.e. a majority of the scene geometry is contained in the current view frustum. Therefore, presented numbers correspond to the lowest frame rates achievable in the given scene. All measurements were performed on an *NVIDIA RTX 2070 Super* along with the VR HMD *Valve Index* using its native resolution of 1440x1600 per eye.
Table 5.3 gives an overview of time spent in the various pipeline stages for each scene. Furthermore, the amount of GPU memory required just for the pipeline is stated in Table 5.4. Frame times were averaged over a period of 100 frames.
It is clearly visible, that path tracing and spatial filtering constitute the main bottleneck of this approach. During path tracing, the number of rays that can be efficiently traced depends on the spatial coherency between them. Indirect rays require resampling of material textures at their hit points. Although rays reflected on a mirror-like surface exhibit spatial coherency, this does not hold true for rays from glossy reflections or indirect diffuse illumination. This leads to incoherent memory accesses resulting in a bad utilization of the texture cache. A further limiting factor during ray tracing lies in the balance of the bounding volume hierarchy. As an example, the **Sponza** scene is not more complex than the **RedRoom** scene in terms of geometry, but its bounding volume hierarchy is degenerated such that rays take on average longer to trace and find an intersection point.

Blue noise masking alleviates these issues by reducing the overall number of rays traced in one frame. However, reducing the number of rays does not increase performance by the same factor. This is connected to a certain overhead that is involved when invoking the ray tracing command on a GPU. The exact performance gains for various coverage factors are given in Table 5.5.

	G-Buffer	History Buffer	Mask Buffers	Intermediate	Total
Memory (MiB)	246.09	36.47	7.18	281.25	570.99

Table 5.4: Scene independent memory usage of proposed pipeline in mebibyte (MiB). This includes memory needed for both eyes.

Scene		*Coverage Factor*			
		1.00	**0.50**	**0.25**	**0.13**
RedRoom	*Time (ms)*	8.73	5.55	3.54	2.36
	Speed-up	-	1.57x	2.47x	3.70x
CornellBox	*Time (ms)*	5.4	3.45	2.25	1.56
	Speed-up	-	1.57x	2.40x	3.46x
Sponza	*Time (ms)*	24.1	14.66	8.34	4.86
	Speed-up	-	1.64x	2.89x	4.96x
Bistro	*Time (ms)*	17.13	10.50	6.08	3.60
	Speed-up	-	1.63x	2.82x	4.76x
	Min. Speed-up	-	1.57x	2.40x	3.46x
	Max. Speed-up	-	1.64x	2.89x	4.96x
	Avg. Speed-up	-	1.60x	2.64x	4.22x

Table 5.5: Time spent in path tracing stage can be significantly reduced with the proposed blue noise masking system. A coverage factor of 0.5 yields on average 50% better performance and creates little to no visual artifacts. The best performance-quality ratio in our experience is achieved with a coverage factor of 0.25, needing little less than half the original time and only resulting in slightly visible visual artifacts in the periphery during fast head movement. Further reducing the coverage factor still increases performance during path tracing but leads to noticeable visual artifacts.

CHAPTER 6

Conclusion and Future Work

To conclude, this chapter gives an overview of work and contributions presented in this thesis. Furthermore, extensive thoughts about future work and possible improvements of limitations shown in Chapter 5 are presented.

6.1 Conclusion

We presented a hybrid real-time path tracing and de-noising pipeline especially designed for virtual reality applications but with minor adaptations also usable for desktop applications. The proposed pipeline makes use of hardware rasterization, ray tracing and compute capabilities to maximize hardware utilization. It can reliably reconstruct first bounce global illumination following physically based principles. That includes soft shadows with contact hardening, glossy reflections and indirect diffuse illumination mostly visible as color bleeding. In addition to improving stability of variance estimation of related work we propose a spatial masking system to reduce the number of rays traced per frame as well as a system to share view-independent samples between stereoscopic views to further reduce noise. Our system proves to run stably and quite fast on modern mid-tier graphics hardware achieving interactive frame rates.
Nonetheless, its most important drawback lies in the energy loss due to a limitation of bounces which is best observed in scenes with bright material colors. While indirect diffuse illumination may be darker compared to reference renderings, glossy reflection might even show completely dark spots since only direct illumination is evaluated on the second hit point. Furthermore, due to spatial filtering of combined direct and indirect illumination, shadows are slightly overblurred and thus too bright. At last, this work did not elaborate on anti-lag or anti-ghosting to eliminate artifacts due to temporal reprojection.

6.2 Future Work

If hardware ray tracing is improved further in upcoming GPU generations, e.g. next *NVIDIA* generation [1], the amount of rays that can be traced may be significantly increased. This might go towards higher sample counts per pixel while maintaining bounce limitations to reduce noise during path tracing and simplify de-noising. On the other hand, this could be used to trace longer paths to close the gap between the de-noised image and the fully converged reference. However, this may lead to more incoherency in memory accesses as has been discussed in Chapter 5. Willberger et al. [HAM19, Chpt. 26] propose to additionally bake material information into vertex data, such that resampling on indirect hits does not require further texture lookups.

Although our approach scales well with varying geometric complexity, it suffers from noise when adding too many lights. Since only the direct contribution of a single light is evaluated with each intersection, variance is directly increased with a higher number of lights. Improvements to the sampling strategy when choosing a light source would give better results with a many-light setup. Moreau and Clarberg [HAM19, Chpt. 18] show how an additional hierarchical data structure over the light sources can be utilized to get better estimates of direct light.

Our novel blue noise masking system is able to reduce time spent on path tracing significantly but suffers from visual artifacts in previously occluded region. Despite the simple solution we presented, these artifacts still occur for fast movements or lower coverage factors. Additionally, temporal shading rate is effectively reduced causing ghosting in shading when objects or lights move. To further improve on that an interesting starting point would be to determine which regions were previously occluded such that those can be definitely included in the current mask. For instance, a pre-temporal reprojection stage after primary hit rasterization could evaluate reprojection and check for occlusions. Masks then could be generated on the fly or augmented by the obtained occlusion information. A different approach could consist of forward reprojection of pixels from the previous frame into the current one and tracing missing pixels as well as updating outdated pixels. Deprecating shading information might depend on time in combination with blue noise masking to update many pixels at once each frame or use a different system like temporal gradients as proposed by Schied et al. [SPD18].

Concerning reprojection, the previous chapter described how lag in reflections and shading may occur. To further address this problem, motion of reflected pixels may be approximated better with the virtual positions of the according object [MMBJ17]. Barré-Brisebois et al. [HAM19, Chpt. 25] present a method to combine motion vector for reflectors and reflected objects to achieve stable reprojection for reflections.

[1] https://wccftech.com/nvidia-ampere-rumors-massive-rt-performance-uplift-higher-clocks-more-vram-lower-tdps-vs-turing/

However, shading on any surface might also change when motion is involved, either of the object itself or of light sources in its surroundings. In addition to geometric data, shading information has to be employed to avoid ghosting in these scenarios. An example was presented by Schied et al. [SPD18] to replace the constant accumulation factor by a dynamic one which adapts to changes in illumination over time.

Post process anti-aliasing is not capable of completely removing geometric aliasing which is even more apparent in VR. Increasing the resolution during rendering also considerable increases the cost of path tracing and de-noising and is thus not feasible. Furthermore, the extent of the spatial filter might depend on the overall resolution of the image and thus varying image sizes may introduce instabilities during filtering. A similar idea as shown by Marrs et al. [MSG+18] might be able to better resolve geometric aliasing in problematic areas. During temporal reprojection, a conservative mask in screen space is computed stating in which areas post process anti-aliasing is likely to fail. By tracing more than just a single primary ray in these areas, a better estimate of the final pixel color may be obtained, thus eliminating geometric aliasing.

At last, methods to improve performance or quality in VR based on the actual perception of humans may be employed. Denes et al. [DMAM19] propose to reduce resolution of one eye each frame and balancing the resulting artifacts by augmenting the remaining full resolution image. By doing so, performance improvements of $37 - 49\%$ may be achieved without disturbing side effects. Zhong et al. [ZKD+19] propose to cheaply convey better local contrast via stereoscopic vision to improve visual quality.

List of Figures

List of Tables

List of Algorithms

Bibliography

[AMHH18] Tomas Akenine-Moller, Eric Haines, and Naty Hoffman. *Real-time rendering*. AK Peters/CRC Press, 2018.

[App68] Arthur Appel. Some techniques for shading machine renderings of solids. In *Proceedings of the April 30–May 2, 1968, spring joint computer conference*, pages 37–45. ACM, 1968.

[AS00] Michael Ashikhmin and Peter Shirley. An anisotropic phong brdf model. *Journal of graphics tools*, 5(2):25–32, 2000.

[BFK19] Nikolaus Binder, Sascha Fricke, and Alexander Keller. Massively parallel path space filtering. *arXiv preprint arXiv:1902.05942*, 2019.

[Bog16] Yuval Boger. Understanding binocular overlap and why it's important for vr headsets, 2016. [Online; accessed 24-January-2020].

[BS12] Brent Burley and Walt Disney Animation Studios. Physically-based shading at disney. In *ACM SIGGRAPH*, volume 2012, pages 1–7, 2012.

[BSH12] Mahdi M Bagher, Cyril Soler, and Nicolas Holzschuch. Accurate fitting of measured reflectances using a shifted gamma micro-facet distribution. In *Computer Graphics Forum*, volume 31, pages 1509–1518. Wiley Online Library, 2012.

[BYF+18] Nir Benty, Kai-Hwa Yao, Tim Foley, Matthew Oakes, Conor Lavelle, and Chris Wyman. The Falcor rendering framework, 05 2018. `https://github.com/NVIDIAGameWorks/Falcor`.

[CD18] Nixxes Software Crystal Dynamics, Eidos Montréal. Metro exodus, 2018.

[CKS+17] Chakravarty R Alla Chaitanya, Anton S Kaplanyan, Christoph Schied, Marco Salvi, Aaron Lefohn, Derek Nowrouzezahrai, and Timo Aila. Interactive reconstruction of monte carlo image sequences using a recurrent denoising autoencoder. *ACM Transactions on Graphics (TOG)*, 36(4):98, 2017.

[CT81] Robert L Cook and Kenneth E Torrance. A reflectance model for computer graphics. In *ACM Siggraph Computer Graphics*, volume 15, pages 307–316. ACM, 1981.

[DMAM19] G. Denes, K. Maruszczyk, G. Ash, and R. Mantiuk. Temporal resolution multiplexing: Exploiting the limitations of spatio-temporal vision for more efficient vr rendering. *IEEE Transactions on Visualization and Computer Graphics*, pages 1–1, 2019.

[DSHL10] Holger Dammertz, Daniel Sewtz, Johannes Hanika, and Hendrik Lensch. Edge-avoiding à-trous wavelet transform for fast global illumination filtering. In *Proceedings of the Conference on High Performance Graphics*, pages 67–75. Eurographics Association, 2010.

[ED04] Elmar Eisemann and Frédo Durand. Flash photography enhancement via intrinsic relighting. *ACM transactions on graphics (TOG)*, 23(3):673–678, 2004.

[ED18] Danger Close EA DICE, Criterion Games. Battlefield v, 2018.

[Ent19] Remedy Entertainment. Control, 2019.

[Gam17] Epic Games. Unreal engine sun temple, open research content archive (orca), October 2017. `http://developer.nvidia.com/orca/epic-games-sun-temple`.

[Gam19] 4A Games. Metro exodus, 2019.

[GF16] Iliyan Georgiev and Marcos Fajardo. Blue-noise dithered sampling. In *ACM SIGGRAPH 2016 Talks*, page 35. ACM, 2016.

[GO12] Eduardo SL Gastal and Manuel M Oliveira. Adaptive manifolds for real-time high-dimensional filtering. *ACM Transactions on Graphics (TOG)*, 31(4):33, 2012.

[Hab10] John Hable. Uncharted 2: Hdr lighting. Games, Game Developers Conference, 2010.

[HAM19] Eric Haines and Tomas Akenine-Moller. *Ray Tracing Gems: High-Quality and with DXR and Other APIs*. Apress, 1.7 pdf edition, 2019.

[HB17] Nicholas Hull and Nir Benty. Nvidia emerald square, open research content archive (orca), July 2017. `http://developer.nvidia.com/orca/nvidia-emerald-square`.

[HBO+19] Eric Heitz, Laurent Belcour, Victor Ostromoukhov, David Coeurjolly, and Jean-Claude Iehl. A low-discrepancy sampler that distributes monte carlo errors as a blue noise in screen space. 2019.

[HHM18] Eric Heitz, Stephen Hill, and Morgan McGuire. Combining analytic direct illumination and stochastic shadows. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, page 2. ACM, 2018.

[HJ17] Earl Hammon Jr. Pbr diffuse lighting for ggx+ smith microsurfaces. 2017.

[HK93] Pat Hanrahan and Wolfgang Krueger. Reflection from layered surfaces due to subsurface scattering. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pages 165–174. Citeseer, 1993.

[HKMMT90] Matthias Holschneider, Richard Kronland-Martinet, Jean Morlet, and Ph Tchamitchian. A real-time algorithm for signal analysis with the help of the wavelet transform. In *Wavelets*, pages 286–297. Springer, 1990.

[Hof13] Naty Hoffman. Background: physics and math of shading. *Physically Based Shading in Theory and Practice*, 24(3):211–223, 2013.

[ICG86] David S Immel, Michael F Cohen, and Donald P Greenberg. A radiosity method for non-diffuse environments. In *Acm Siggraph Computer Graphics*, volume 20, pages 133–142. ACM, 1986.

[Jak14] Atelier Otto Jakob. A comprehensive framework for rendering layered materials. *ACM Transactions on Graphics (TOG)*, 33(4), 2014.

[JME+11] Jorge Jimenez, Belen Masia, Jose I. Echevarria, Fernando Navarro, and Diego Gutierrez. *GPU Pro 2*, chapter Practical Morphological Anti-Aliasing. AK Peters Ltd., 2011.

[Kaj86] James T Kajiya. The rendering equation. In *ACM SIGGRAPH computer graphics*, volume 20, pages 143–150. ACM, 1986.

[Kar14] B Karis. High quality temporal anti-aliasing. *Advances in Real-Time Rendering for Games, SIGGRAPH Courses*, 2014.

[KD13] Anton S Kaplanyan and Carsten Dachsbacher. Path space regularization for holistic and robust light transport. In *Computer Graphics Forum*, volume 32, pages 63–72. Wiley Online Library, 2013.

[KDK17] Peter Kán, Maxim Davletaliyev, and Hannes Kaufmann. Discovering new monte carlo noise filters with genetic programming. *EG 2017-Short Papers*, 2017.

[KG13] Brian Karis and Epic Games. Real shading in unreal engine 4. *Proc. Physically Based Shading Theory Practice*, 4, 2013.

[KIM+19] Matias Koskela, Kalle Immonen, Markku Mäkitalo, Alessandro Foi, Timo Viitanen, Pekka Jääskeläinen, Heikki Kultala, and Jarmo Takala. Blockwise multi-order feature regression for real-time path-tracing reconstruction. *ACM Transactions on Graphics (TOG)*, 38(5):138, 2019.

[KSK01] Csaba Kelemen and Laszlo Szirmay-Kalos. A microfacet based coupled specular-matte brdf model with importance sampling. In *Eurographics Short Presentations*, volume 25, page 34, 2001.

[KSKK10] Murat Kurt, László Szirmay-Kalos, and Jaroslav Křivánek. An anisotropic brdf model for fitting and monte carlo rendering. *ACM SIGGRAPH Computer Graphics*, 44(1):3, 2010.

[Lag12] Sébastien Lagarde. Spherical gaussian approximation for blinn-phong, phong and fresnel. *Random Thoughts about Graphics in Games blog, June*, 3, 2012.

[Lam60] Jean-Henri Lambert. *JH Lambert,... Photometria, sive de Mensura et gradibus luminis, colorum et umbrae*. sumptibus viduae E. Klett, 1760.

[Laz13] Dimitar Lazarov. Getting more physical in call of duty: Black ops ii. *part of "Physically Based Shading in Theory and Practice," SIGGRAPH*, 2013.

[LdR14] Sébastien Lagarde and Charles de Rousiers. Moving frostbite to physically based rendering. In *SIGGRAPH 2014 Conference, Vancouver*, 2014.

[LKYU12] Joakim Löw, Joel Kronander, Anders Ynnerman, and Jonas Unger. Brdf models for accurate and efficient rendering of glossy surfaces. *ACM Transactions on Graphics (TOG)*, 31(1):9, 2012.

[Lot09] Timothy Lottes. Fxaa. *White paper, Nvidia, Febuary*, 2009.

[Lum17] Amazon Lumberyard. Amazon lumberyard bistro, open research content archive (orca), July 2017. `http://developer.nvidia.com/orca/amazon-lumberyard-bistro`.

[McG08] Morgan McGuire. A fast, small-radius GPU median filter. In *Published in ShaderX6.*, February 2008. ShaderX6.

[McG17] Morgan McGuire. Computer graphics archive, July 2017. `https://casual-effects.com/data`.

[Mic18] Microsoft. DirectX Raytracing (DXR) Functional Spec. `https://microsoft.github.io/DirectX-Specs/d3d/Raytracing.html`, 2018. [Online; accessed 13-January-2020].

[MKW+02] Michael Meißner, Urs Kanus, Gregor Wetekam, Johannes Hirche, Alexander Ehlert, Wolfgang Straßer, Michael Doggett, P Forthmann, and R Proksa. Vizard ii: a reconfigurable interactive volume rendering system. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 137–146. Eurographics Association, 2002.

[MMBJ17] Michael Mara, Morgan McGuire, Benedikt Bitterli, and Wojciech Jarosz. An efficient denoising algorithm for global illumination. In *High Performance Graphics*, pages 3–1, 2017.

[MMNL16] Michael Mara, Morgan McGuire, Derek Nowrouzezahrai, and David P Luebke. Deep g-buffers for stable global illumination approximation. In *High Performance Graphics*, pages 87–98, 2016.

[MPBM03] Wojciech Matusik, Hanspeter Pfister, Matt Brand, and Leonard McMillan. A data-driven reflectance model. *ACM Transactions on Graphics*, 22(3):759–769, July 2003.

[MSG+18] Adam Marrs, Josef Spjut, Holger Gruen, Rahul Sathe, and Morgan McGuire. Adaptive temporal antialiasing. In *Proceedings of the Conference on High-Performance Graphics*, page 1. ACM, 2018.

[MSS+10] Quirin Meyer, Jochen Süßmuth, Gerd Sußner, Marc Stamminger, and Günther Greiner. On floating-point normal vectors. In *Computer Graphics Forum*, volume 29, pages 1405–1409. Wiley Online Library, 2010.

[Nar] Krzysztof Narkowicz. Aces filmic tone mapping curve. `https://knarkowicz.wordpress.com/2016/01/06/aces-filmic-tone-mapping-curve/`. [Online; accessed 31-January-2020].

[NMDSLC15] Manish Narwaria, Rafal Mantiuk, Mattheiu P Da Silva, and Patrick Le Callet. Hdr-vdp-2.2: a calibrated method for objective quality prediction of high-dynamic range and standard images. *Journal of Electronic Imaging*, 24(1):010501, 2015.

[NSL+07] Diego Nehab, Pedro V Sander, Jason Lawrence, Natalya Tatarchuk, and John R Isidoro. Accelerating real-time shading with reverse reprojection caching. In *Graphics hardware*, volume 41, pages 61–62, 2007.

[Nvia] Nvidia. VRWorks - Multi-Res Shading. `https://developer.nvidia.com/vrworks/graphics/multiresshading`. [Online; accessed 25-January-2020].

[Nvib] Nvidia. VRWorks - Single Pass Stereo. `https://developer.nvidia.com/vrworks/graphics/singlepassstereo`. [Online; accessed 04-February-2020].

[Nvic] Nvidia. VRWorks - Variable Rate Shading. https://developer.nvidia.com/vrworks/graphics/variablerateshading. [Online; accessed 25-January-2020].

[Nvi18] Nvidia. Nvidia turing architecture whitepaper. https://www.nvidia.com/content/dam/en-zz/Solutions/design-visualization/technologies/turing-architecture/NVIDIA-Turing-Architecture-Whitepaper.pdf, 2018.

[NVI19] Lightspeed Studios NVIDIA. Quake II RTX. https://www.nvidia.com/de-de/geforce/campaigns/quake-II-rtx/, 2019.

[O'D18] Yuriy O'Donnell. Precomputed Global Illumination in Frostbite. https://www.ea.com/frostbite/news/precomputed-global-illumination-in-frostbite, 2018. [Online; accessed 05-April-2019].

[ON94] Michael Oren and Shree K Nayar. Generalization of lambert's reflectance model. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 239–246. ACM, 1994.

[PHK+99] Hanspeter Pfister, Jan Hardenbergh, Jim Knittel, Hugh Lauer, and Larry Seiler. The volumepro real-time ray-casting system. 1999.

[PJH16] Matt Pharr, Wenzel Jakob, and Greg Humphreys. *Physically based rendering: From theory to implementation*. Morgan Kaufmann, 2016.

[Poy98] Charles Poynton. Frequently asked questions about gamma. *Rapport Technique, janvier*, 152, 1998.

[PSA+04] Georg Petschnigg, Richard Szeliski, Maneesh Agrawala, Michael Cohen, Hugues Hoppe, and Kentaro Toyama. Digital photography with flash and no-flash image pairs. *ACM transactions on graphics (TOG)*, 23(3):664–672, 2004.

[Rei02] Erik Reinhard. Parameter estimation for photographic tone reproduction. *Journal of graphics tools*, 7(1):45–51, 2002.

[Sch94] Christophe Schlick. An inexpensive brdf model for physically-based rendering. In *Computer graphics forum*, volume 13, pages 233–246. Wiley Online Library, 1994.

[SGFV19] VV Sanzharov, AI Gorbonosov, VA Frolov, and AG Voloboy. Examination of the nvidia rtx. 2019.

[SKW+17] Christoph Schied, Anton Kaplanyan, Chris Wyman, Anjul Patney, Chakravarty R Alla Chaitanya, John Burgess, Shiqiu Liu, Carsten Dachsbacher, Aaron Lefohn, and Marco Salvi. Spatiotemporal variance-guided

filtering: real-time reconstruction for path-traced global illumination. In *Proceedings of High Performance Graphics*, page 2. ACM, 2017.

[SL17] Ari Silvennoinen and Jaakko Lehtinen. Real-time global illumination by precomputed local reconstruction from sparse radiance probes. *ACM Transactions on Graphics (TOG)*, 36(6):230, 2017.

[SÖA+19] Gurprit Singh, Cengiz Öztireli, Abdalla GM Ahmed, David Coeurjolly, Kartic Subr, Oliver Deussen, Victor Ostromoukhov, Ravi Ramamoorthi, and Wojciech Jarosz. Analysis of sample correlations for monte carlo rendering. In *Computer Graphics Forum*, volume 38, pages 473–491. Wiley Online Library, 2019.

[SPD18] Christoph Schied, Christoph Peters, and Carsten Dachsbacher. Gradient estimation for real-time adaptive temporal filtering. *Proceedings of the ACM on Computer Graphics and Interactive Techniques*, 1(2):24, 2018.

[SSHL97] Peter Shirley, Brian Smits, Helen Hu, and Eric Lafortune. A practitioners' assessment of light reflection models. In *Proceedings The Fifth Pacific Conference on Computer Graphics and Applications*, pages 40–49. IEEE, 1997.

[SWS02] Jörg Schmittler, Ingo Wald, and Philipp Slusallek. Saarcor: a hardware architecture for ray tracing. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 27–36. Eurographics Association, 2002.

[SWW+04] Jörg Schmittler, Sven Woop, Daniel Wagner, Wolfgang J Paul, and Philipp Slusallek. Realtime ray tracing of dynamic scenes on an fpga chip. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 95–106. ACM, 2004.

[SZR+15] Pradeep Sen, Matthias Zwicker, Fabrice Rousselle, Sung-Eui Yoon, and Nima Khademi Kalantari. Denoising your monte carlo renders: recent advances in image-space adaptive sampling and reconstruction. In *ACM Siggraph 2015 Courses*, pages 1–255. 2015.

[TR75] TS Trowbridge and Karl P Reitz. Average irregularity representation of a rough surface for ray reflection. *JOSA*, 65(5):531–536, 1975.

[TS67] Kenneth E Torrance and Ephraim M Sparrow. Theory for off-specular reflection from roughened surfaces. *Josa*, 57(9):1105–1114, 1967.

[Vea97] Eric Veach. *Robust Monte Carlo methods for light transport simulation*, volume 1610. Stanford University PhD thesis, 1997.

[Vla15] Alex Vlachos. Advanced vr rendering. *Game Developers Conference*, 2015.

[Vla16] Alex Vlachos. Advanced vr rendering performance. *Game Developers Conference*, 2016.

[Whi79] Turner Whitted. An improved illumination model for shaded display. In *ACM SIGGRAPH Computer Graphics*, volume 13, page 14. ACM, 1979.

[Wil78] Lance Williams. Casting curved shadows on curved surfaces. In *ACM Siggraph Computer Graphics*, volume 12, pages 270–274. ACM, 1978.

[WMLT07] Bruce Walter, Stephen R Marschner, Hongsong Li, and Kenneth E Torrance. Microfacet models for refraction through rough surfaces. In *Proceedings of the 18th Eurographics conference on Rendering Techniques*, pages 195–206. Eurographics Association, 2007.

[YNS+09] Lei Yang, Diego Nehab, Pedro V Sander, Pitchaya Sitthi-amorn, Jason Lawrence, and Hugues Hoppe. Amortized supersampling. In *ACM Transactions on Graphics (TOG)*, volume 28, page 135. ACM, 2009.

[ZJL+15] Matthias Zwicker, Wojciech Jarosz, Jaakko Lehtinen, Bochang Moon, Ravi Ramamoorthi, Fabrice Rousselle, Pradeep Sen, Cyril Soler, and S-E Yoon. Recent advances in adaptive sampling and reconstruction for monte carlo rendering. In *Computer Graphics Forum*, volume 34, pages 667–681. Wiley Online Library, 2015.

[ZKD+19] Fangcheng Zhong, George Alex Koulieris, George Drettakis, Martin S Banks, Mathieu Chambe, Frédo Durand, and Rafał K Mantiuk. Dice: dichoptic contrast enhancement for vr and stereo displays. *ACM Transactions on Graphics (TOG)*, 38(6):1–13, 2019.