Real-time Path Traced Global Illumination for Games by

Path Space Filtering

Master Thesis - Game & Media Technology

student	Dustin Meijer
number	ICA-5726328
supervisor	Alexander Keller (NVIDIA)
first examiner	Jacco Bikker
second examiner	Frank van der Stappen

Utrecht University

Acknowledgements

I would like to thank Alexander Keller for the amazing opportunity and guidance during this thesis work.
I am super grateful to Nikolaus Binder for all the advice, amazing discussions, fun and being an overall amazing colleague and mentor.
A big thanks to Petrik Clarberg, Nir Benty, Pierre Moreau and the all the Falcor users for helping me out whenever I had questions about or issues with the Falcor framework.
Matthijs Van keirsbilck and Gonçalo Gordido, thank you guys for making lunch-time so entertaining and for being willing to explore Berlin with me.
I had a lot of fun in Berlin thanks to all the great colleagues there, I won't forget the Wednesdays!
Jacco Bikker and Frank van der Stappen, thank you so much for willing to serve as supervisors and examiners for this thesis work.
Thanks to Douwe van Gijn for supporting me during the thesis writing process, without him this thesis would not be completed.
Finally, a big thanks to my wife and family for supporting me during my time in Berlin. I could not have done this without all the amazing people around me.

Abstract

This thesis is about approximating global illumination for games in real-time using ray tracing. We have found that not pre-multiplying the albedo during tracing to avoid having to divide it out is numerically more stable due to floating point arithmetic and avoids a special case where information is lost.

Furthermore, we investigate the benefits of using world-space neighbourhood search in path space filtering with a screen-space search, called screen-space path space filtering (SSPSF). In addition, we implement and test multiple extensions of the hashed path space filtering (HPSF) method that were proposed in the original report "Fast Path Space Filtering by Jittered Spatial Hashing" by Binder *et al.* [1]. Based on HPSF, we develop a novel technique called recursive hashed path space filtering (nHPSF), which filters at multiple selected bounces along a light transport path, which allows for reducing variance. It can transport information between hash cells during scattering as well, which further increases efficiency.

Our main result is an algorithm that amortizes the cost of tracing long light transport paths over multiple frames and combines it with the nHPSF technique. It hence is named Amortized nHPSF (A-nHPSF).

The thesis includes an extensive survey of the state of the art in the industry and reviews important related literature. It closes with a set of important and promising directions for future research.

Contents

List of Figures

List of Algorithms

1 Introduction

Games have been the driving force behind the improvements of real-time computer graphics since the early 1990's and are all about delivering a constant real-time interactive experience. Games could historically get away with many approximations for global lighting effects because other components in the scene were visually more distracting, e.g. the low polygonal count in the 3D models. The main rendering technique that has been used over the past two decades for games is called rasterization which has historically needed involved techniques to approximate or fake global lighting effects. With the constant strive for increased image fidelity, the techniques for achieving global lighting effects such as reflections, soft shadows and more, have grown to become computationally expensive and complex to manage in a rendering pipeline.

In contrast, ray tracing is a method of rendering where global light effects are achieved by simulating light transport. However, this method of rendering has traditionally been too computationally expensive when compared to rasterization and was mainly reserved for off-line rendering purposes. The physically correct version of ray tracing, called path tracing, has been the method of choice in rendering photoreal imagery in production rendering for visual effects for movies and the like for over a decade.

With the release of NVIDIA's RTX technology, real-time ray tracing and path tracing on consumer hardware is now in the realm of possiblities. At the time of writing, ray tracing is being combined with traditional rasterization to utilize the strengths of both techniques and to attempt to overcome their separate limitations. The main examples of this hybrid approach are techniques that use ray tracing as a drop-in replacement for various effects, e.g. reflections, shadows, ambient occulusion, and more. This is mentioned in greater detail in subsection 3.2.

Due to the stochastic nature of path tracing there is visible noise in the output when it is not yet converged. This would be alleviated by taking many samples, but in games there is a strict frame budget for computation that needs to be adhered to which makes it prohibitive to do so. This is the main problem with path tracing that makes it unappealing for use in games, even though it can now be performed in real-time on consumer hardware.

If we want to have physically correct lighting effects from path tracing in real-time in games, we need to look to methods other than taking more samples to deal with the noise that is present in the output.

1.1 The Goal of this Thesis

The main objective of this thesis is to achieve path traced global illumination in real-time for games by employing filtering techniques, specifically filters based on "Path space filtering for integro-approximation problems" by Keller *et al.* [2].

The strict frame budget we need to adhere to is determined by the cost of the rendering work per frame. This is in turn dependent on the complexity of the scene that is being rendered and the rendering techniques that are being used. There are many ways one can define what 'real-time' is for games. In general, the bare minimum performance requires the game to run at 30 frames per second (fps). However, most gamers expect games to run at a minimum of 60 fps. The minimum resolution that is expected of modern games is 1280 by 720 pixels (HD Resolution), e.g. the screen resolution of the Nintendo Switch [3].

We aim to achieve a resolution of 1920 by 1080 pixels (Full-HD Resolution) at 60 fps.

We believe the best image fidelity comes from the effects of global illumination, which is the summation of the direct and indirect light that lands on a surface. Convincingly accurate lighting with shadows and reflections ground the 3D environment into something akin to reality for any art direction that does not completely ignore the physics behind light transport. We believe that the next big step in image fidelity will be convincingly plausible lighting in games using path tracing.

The main focus for games is the real-time requirement, which means that any shortcuts that yield

a little bias in the image are allowed. This is quite different from the production rendering requirements where image fidelity is the main focus.

For games, it is a reasonable assumption that we must make due with at most 7 rays per pixel consisting of 3 scatter rays and 4 direct light sampling rays, while delivering 60 frames per second on a 1920x1080 resolution. This assumption is based on the performance of the Quake II RTX path tracer which has this exact ray budget [4], [5].

1.2 Structure

In the remainder of this thesis, we will first introduce the unexperienced reader to computer graphics in section 2. Then we give the context of our work with respect to previous research and existing applications in section 3. This is followed by an in depth description of the algorithms used in our experiments in section 4. In particular, we describe our novel algorithms, Recursive HPSF (nHPSF) and Amortized nHPSF (A-nHPSF), in subsection 4.3 and 4.4, respectively. We will then show the results of our experiments in section 5 which are then discussed in section 6. Finally, we conclude with our findings and describe possible directions for future work in section 7.

1.3 Contributions

The contributions of this thesis are the following :

1. We show that not pre-multiplying the albedo during tracing to avoid having to divide it out is numerically more stable due to floating point arithmetic and avoids a special case where information is lost.
2. We introduce a new simple method that replaces the world-space neighbourhood search in path space filtering with a screen-space search, called screen-space path space filtering (SSPSF).
3. We implemented and tested multiple extensions to the hashed path space filtering (HPSF) method that were proposed in the original paper.
4. We introduce a novel extension of the HPSF method called recursive hashed path space filtering (nHPSF), which filters on multiple valid bounces during a light transport path. It also transports information between hash cells during scattering.
5. We introduce a novel method that amortizes the cost of tracing long light transport paths over multiple frames and combine it with nHPSF called Amortized nHPSF (A-nHPSF).

2 Preliminaries

This section briefly reviews the main issues that are researched in computer graphics. The entire research area is too large to cover in its entirety, thus this section is a short collection of preliminaries to cover the basics of the field. First, we will go over what the goal is of computer graphics and what problems are tackled in the field. Then, we will describe the existing methods that attempt solve these problems, as well as their shortcomings.

2.1 The Goal of Rendering

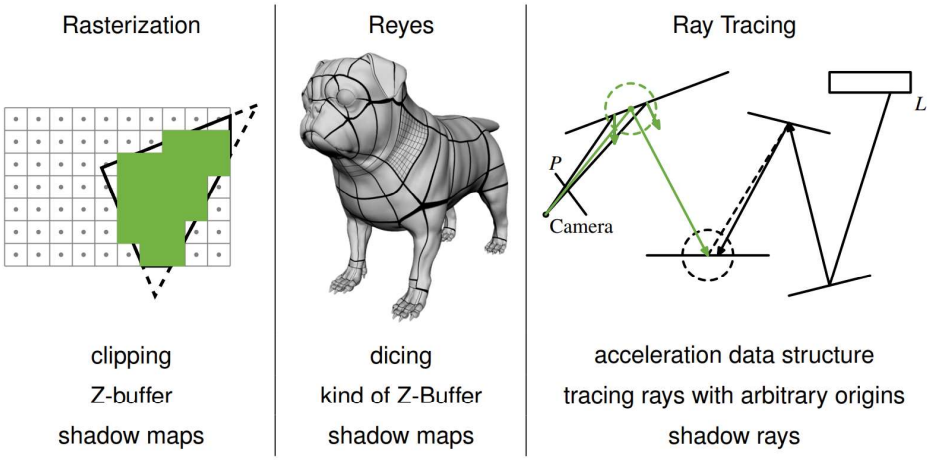


Figure 1: This figure illustrates the principles of the different rendering methods. Image reproduced with permission from Alexander Keller, NVIDIA.

In computer graphics we aim to create two-dimensional (2D) images given some description of a scene or environment. This information usually includes at least geometry, light emitters, material information, and a camera. In the terms for 3D graphics, the geometry is usually described by triangles which are rendered to a 2D image from the viewpoint of a camera by processing the scene information for each pixel of our 2D output image. In the most simplest of terms, we are trying to solve a visibility problem (see Figure 1). There are multiple algorithms that go about different ways of solving this, each with its own strengths and detractions. But, most importantly, for all the algorithms it is too computationally expensive to simply consider all triangles during rendering. To make rendering feasible, each algorithm attempts to clip away triangles that do not directly influence the output of an image.

2.2 Rasterization (Fixed Function Pipeline)

Historically for games, rasterization-based rendering has been the only way to create three-dimensional (3D) environments for gamers to enjoy in real-time. The 3D environment is described by triangles which are rendered to a 2D image from the viewpoint of a camera by processing each triangle. The algorithm scales linearly with the amount of triangles and can be trivially parallelized, which is why many of the operations for rasterization have been accelerated by hardware.

The triangles themselves are made up of a combination of vertices. The vertices that lie within the View Frustum of the camera are projected to a 2D view. For each triangle and overlapping pixel, the distance from the camera is determined and only the closest distance is kept in a Z-Buffer. This distance is also known as the depth. The depth information indicates what pixels see what fragments of the triangles. Each fragment can then be processed to determine what color it is. This process is called shading.

The shading can be done in many ways, but in general the idea is to give an illusion of realistic illumination and color. The triangles usually carry more information than just their position. In most cases the direction that is being faced is also stored in what is called the normal. Even though the normal of a triangle can technically be calculated given its vertices, other normal information is used instead. The vertices of a triangle can carry per-vertex normals or coordinate information to access a normal map. This type of normal information can be used to simulate none flat triangles like smooth surfaces or even further detail, e.g. wood grain. Furthermore, the triangles usually carry some reference to or description of a material (e.g. red car paint). Using the normals, the material, and information about lights in the 3D environment, the fragments can be shaded in a fashion dependent on the artistic direction of the rendering.

Issues in Rasterization: The goal for graphics in games is not to be correct, but to be convincing and in line with the respective art direction. Processing triangles in isolation is fast, but introduces a fundamental restriction: The information available for rendering is only what is locally available as fragments. This causes problems for any effect that depends on global information e.g. transparency, reflections, and shadows.

Geometry in an environment is not only lit by direct light from light sources, but also by indirect light that is reflected from other surfaces. Because the rasterization algorithm only has local information, i.e. only what is on the screen and not occluded, many effects are difficult to achieve convincingly or physically accurate. Reflections may reflect geometry that is not in the view frustum of the camera. Shadows can be cast by geometry outside of the view frustum onto what is in the view frustum of the camera.

Assuming a static scene, many effects can be precomputed and applied by performing a look up of that precomputed information. Otherwise, if the scene is dynamic, the scene might have to be rendered multiple times from different perspectives to collect the information required to achieve these effects. For example :

- With light baking, shadows and indirect lighting can be precomputed for everything that is static in the scene [6].
- Transparency can be achieved by rendering what is behind transparent fragments and blending the rendering results together with some blending factor [7].
- Screen-space reflections approximate reflections under the assumption that what is being reflected is present on the screen [8]. The reflection is achieved by transforming what is already rendered on the screen to the location of where the reflection would be. When this assumption is not true, the reflection can be and oftentimes is incomplete.
- With reflection probes, reflections are created by rendering from the perspective of the reflection and projecting the results on the location of where the reflection should be [9].
- With shadow maps, we render from the perspective of the lights to see what geometry is occluded [10], [11]. This information is gathered in a shadow map which can then be transformed to the camera perspective and applied. The computation cost scales linearly with the amount of lights.

Achieving these effects can become difficult and computationally expensive while the fidelity of the results remain unsatisfactory [7]. The strength of Rasterization lies in its parallel nature and speed for primary visibility. The strength of ray tracing comes from the fact that many effects are the natural result of simulating how light moves through a 3D environment which requires using global information, e.g. shadows, translucency, and reflectance. In current modern games, effects that require global information can be achieved by combining the strengths of rasterization and ray tracing. For instance, the 2016 game "Inside"* uses ray tracing (specifically ray marching) for volumetric light effects like fog and water.

With the advent of physically-based Rendering, all these effects can be achieved realistically. These effects are the natural results of the correct description and simulation of light transport according

*"Low Complexity, High Fidelity - INSIDE Rendering" by Gjoel *et al.* [12]

to the rules that describe what happens in nature and reality. Nonetheless, this is computationally expensive, so we have to be smart about how the way light moves through a 3D environment is simulated.

2.3 Ray Tracing

In contrast to rasterization, ray tracing clips away information on a per ray basis instead of clipping at the global ensemble. This naturally allows ray tracing to access information beyond that of which is only directly visible in screen-space. "An Improved Illumination Model for Shaded Display" by Whitted [13] introduced recursive ray tracing. With this algorithm a lot more of the light transport effects are considered, e.g shadows, reflections, refractions, but not all of them.

"The Rendering Equation" by Kajiya [14] introduced a mathematical formulation of what needs to be solved to achieve Light Transport correctly and also introduced Path Tracing as a method to solve this. The rendering equation can be formulated in many ways, but let us consider the following formulation in Equation 1.

$$L_o(x, \omega_o) = L_e(x, \omega_o) + \int_\Omega f_s(x, \omega_i, \omega_o) L_i(x, \omega_i)(\omega_i \cdot n)\, \mathrm{d}\omega_i \tag{1}$$

Consider a single ray that hits a point x, the radiance that leaves point x in the direction ω_o is $L_o(x, \omega_o)$. The radiance that is directly emitted from point x is $L_e(x, \omega_o)$. The incoming radiance that is reflected is more involved, for this we need to integrate over all the possible incoming directions over the hemisphere Ω. For an incoming direction ω_i, the reflected radiance towards ω_o is in the incoming radiance $L_i(x, \omega_i)$ on the surface per unit area, which is called the irradiance, by multiplying it by the cosine of the angle between ω_i and the surface normal n. Lastly, the irradiance is multiplied by the *BSDF* (Bidirectional scattering distribution function) [15], which describes how incoming irradiance is scattered to outgoing radiance for a pair of incoming and outgoing directions, which encapsulates the properties of the material at point x.

Path Space Integral: There is also a different formulation of the rendering equation which considers *Path Space* called the *Path Space Integral* formulated by Veach and Guibas[16]. We use the simple formulation of this form from Hachisuka, Pantaleoni, and Jensen [17] in Equation 2.

$$I_j = \int_P f_j(\bar{x})\, \mathrm{d}\mu(\bar{x}) \tag{2}$$

In this equation, we are integrating over the domain of all light transport paths of all path lengths P, where f_j is the contribution measurement and $\mu\bar{x}$ is the operator for constructing and selecting paths. In simple terms, this allows us to describe and consider paths individually.

Global Illumination is a commonly used term, but not extremely descriptive. Consider how light arrives at a camera in a scene, if we look at the first intersection (see Figure 3). As the Rendering Equation (Equation 1) describes, light can simultaneously be emitted by a surface and the light that arrives at the surface can and mostly is reflected.

A point can receive light from many different light transport paths of different lengths. The light that is received can be classified into direct and indirect light. The direct light that a point can receive consists of two components. The first component is light that is directly emitted and not reflected. And the second component is the light that is reflected exactly once on a surface. This light is from short light transport paths, paths with a length of a maximum length of 1.

Light that is emitted from somewhere and is first reflected by a different surface before arriving at the surface we are looking at is called indirect illumination. In other words, light that arrives light transport paths of a length that is longer than 1. Global illumination considers light from all light transport path lengths. However, this direct and indirect illumination can considered recursively

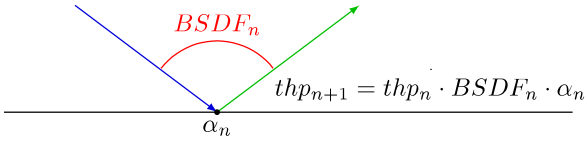


Figure 2: A visualization of updating the path tracing throughput. Here, thp_n is the throughput along a path of length n that is updated at each path vertex, the $BSDF_n$ is the Bidirectional Scattering Distribution Function as described in [15], and α_n is the albedo at the path vertex n.

for every surface that reflects illumination. This leads to the global illumination problem: given all the light emitters in a scene, what is the amount of light that is received on a point in a scene?

With the form of equation 1 is presented in, scattering to a different surface is measured only once, we can however scatter recursively as many times as we want by making $L_i = L_o$. Sampling the negative hemisphere can be also avoided by replacing $(\omega_i \cdot n)$ with $\max(0, \omega_i \cdot n)$, since sampling the negative hemisphere only makes sense if rays can pass through the material of that surface.

In a path tracer, the BSDF at a path vertex, which includes the albedo, is called the attenuation. The product of the attenuations along a path is usually stored in a variable we call the throughput. The radiance that is reflected towards the camera is then the product of the attenuations of the path vertices along the path multiplied with the received irradiance at a path vertex (see Figure 2).

Equation 1 does not yet cover all possible effects of light transport and can be extended and/or adjusted in many ways, for example;

- To include the interactions of wavelengths of light [18]–[20].
- To include the interactions of different volume media [21], [22].
- To include finite speeds [23].
- Or even, to simulate sound waves [24].

With the rendering equation formulated, we can now describe how we can measure the color value that a camera sensor would receive given the simulated Light Transport. Every measurement would then be in the form of Equation 3:

$$c = \int_{A \times \Omega} W(x, \omega) L_i(x, \omega) \, \mathrm{d}x \, \mathrm{d}\omega \tag{3}$$

With this equation, a sensor in the 3D environment can be modeled that, depending on its sensitivity $W(x, \omega)$, integrates over the surface of the sensor A and all possible incoming directions Ω. Based on this equation multiple types of cameras can be modeled, these correspond to different forms and collections of the measurement equation mentioned here. For instance, a pinhole camera would have a sensor for each pixel where each position on the sensor would only respond to one direction. By extending the equation to integrate over a surface, like a lens, effects like *depth of field* [25] are achieved. And by integrating over time, *motion blur* can be modeled [25], [26].

The main issue with the rendering equation is that there is no closed form solution available in the general case which means that we need to approximate this integral numerically. This is done using Monte Carlo simulation. However, as this is a subject much too large and detailed to cover in this thesis, instead the basics will be introduced that are required for this thesis and for more detailed information we refer to other works.

Monte-Carlo Integration:

$$\int_A^B f(x)dx \approx \frac{B-A}{N} \sum_{i=1}^{N} f(X_i), \text{where} X_1, ..., X_N \in [A, B] \tag{4}$$

The idea of Monte Carlo Integration is that every random sample is part of a stochastic experiment where the expected value is the actual value of the integral we wish to know. By defining a consistent estimator as the stochastic experiment, the average value will converge to the correct value with sufficient samples. By being consistent, it is guaranteed that in the limit, variance will decrease and the result will converge to the correct value. In other words, when the amount of samples approaches infinity the standard deviation will approach 0 and the average of our samples will equal the actual value of the integrand. However, an estimator can also be biased while remaining consistent, which means that the expected error of the estimator is either higher (positive bias) or lower (negative bias) than the actual value. If an estimator is biased, but consistent, the bias will disappear in the limit.

By allowing bias and not demanding to be consistent, there is a much larger class of algorithms available of which some can be faster and still look plausible, but will not converge to the correct answer. For instance, noise filtering can be more efficient than variance reduction [27].

Note that an estimator can be unbiased, but inconsistent, thus it will not converge. For further reading about these topics, we refer the reader to [28], [29].

Variance, Error, and the Need to reduce it all: The standard deviation of the outcome of Equation 4 decreases by $\frac{1}{N}$, this means that prohibitively more samples are needed if N is large to converge to the correct answer. Variance, in simple and mostly correct terms, is the squared standard deviation, which can usually be regarded as the error which decreases with more samples. That means that the error will decrease to $\frac{1}{N^2}$; so to half the error, four times the amount of samples are needed [27]. This means that Moore's law [30] will not help, as having more computational power to take more samples, will only have diminishing returns.

In the environment of real-time graphics, we simply do not have the time to take more samples. This means we need to be clever regarding what samples we take and what we do with the few samples that are available to us. Furthermore, it is important to note that variance can also indicate that there is high-frequency detail between samples in an area and is not just simple error (e.g. imagine a shadow boundary).

Sampling what is Important: Importance sampling takes samples proportional to their significance to arrive to the correct answer faster with fewer samples. If the significance has unknown factors, heuristics must be employed to estimate the expected significance of the samples. Figuratively speaking, the idea is to try to choose a sample distribution whose shape is similar to the function that is being integrated. By drawing samples randomly according to this distribution, the samples are more likely to be placed where the most important information is expected. This focuses the majority of the samples on places of the integrand where the contribution is highest, at least according to the distribution, which can reduce variance [28], [29]. To keep the sampling process consistent and fair, all the contributions that were found by using this technique must be reweighted by the probability that they were chosen [31].

However, we do not know what distribution is ideal when we are sampling the light arriving over the hemisphere of an intersection point beforehand. Some distributions work better for certain scenarios than others, e.g. the diffuse BSDF scatters light proportional to the projected solid angle, which means that the radiance to irradiance conversion is used as an estimate for the sample distribution for importance sampling, which is a cosine hemisphere distribution. Furthermore, if the wrong distribution is used, instead of decreasing variance the opposite might happen instead.

Sampling can be done with correlated or uncorrelated random numbers. By allowing correlated samples, we can trade artifacts, e.g. structured noise, for the ability to re-use data in computation or to guarantee a certain sampling pattern. Correlated sampling can positively improve the convergence rate of the estimator, e.g. if there is low frequency detail in a pixel that is sampled well with a regular pattern [32]–[34]. Furthermore, correlated samples can allow for gradients and covariance to be calculated and can be used as input to increase the quality of denoisers. However, one must be careful for the structured noise pattern this type of sampling can introduce.

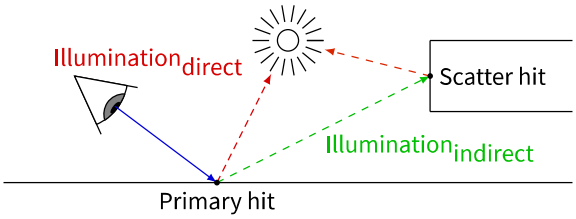


Figure 3: Path Tracing with Next Event Estimation (NEE)

Next Event Estimation (NEE) and Multiple Importance Sampling (MIS): Next event estimation tries to estimate the next most important event. The event is usually assumed to be a path that directly connects a path vertex to a light source [28], [35], [36]. Sometimes, the light source and path vertex do not have a direct path because something is blocking the light. In that case, a zero contribution is added. Using this explicit sampling, variance may be significantly reduced. This technique can be expressed in a form of the rendering equation, see Equation 5.

$$L(s \leftarrow x) = L_e(s \leftarrow x) + \int_A f_s(s \leftarrow x \leftarrow x')L(x \leftarrow x')G(x \leftrightarrow x')\,\mathrm{d}A(x') \tag{5}$$

Compared to Equation 1, G is introduced by the change of measure from the exitant angle $\mathrm{d}\omega$ to the surfaces $\mathrm{d}A(x')$. Furthermore, one could sample proportional to G because G is an independent factor. This could reduce variance even further. In practice, NEE is used to divide the domain of the integrand into direct and indirect illumination (see Figure 3).

Estimators, like the NEE estimator mentioned above, are usually good at estimating only one part of the entire integrand and are bad at estimating other parts of the integrand. Veach and Guibas, and Lafortune and Willems realized that combining multiple estimators would decrease variance. The estimators are then only used on parts of the integrand that they specialize in. The framework for combining estimators is called Multiple Importance Sampling (MIS), which is now ubiquitous to combine all sorts of estimation techniques [37], [38].

There are many different approaches and extensions to sampling with Path tracing that aim at reducing the Monte Carlo noise and speed up convergence [28], [35]–[43].

Even considering all of these techniques, if there is a direct limit on the amount of samples that can be taken and the image is not yet converged, noise will remain.

Filtering the Remaining Noise: Path tracing is a point sampling technique where sampling a point in high dimensional path space can be seen as synonymous to sampling a single path, where the first two dimensions of the high dimensional sample space equate to a point in a pixel. By considering the ensemble of pixels and sharing information between them, we can attempt to identify and filter noise. Denoising and reconstructing the image with filters introduces bias, while smoothing out our output image. The bias from this type of filtering can introduce artifacts that remove sharp detail, e.g. overly smoothed edges and blurred texture detail. To keep these types of artifacts to a minimum, the filters are usually supplied with extra information from our sampling, e.g normals, depth, etc., to identify features that need to be retained. Bias is traded for a reduction in visually disturbing noise, making it a very powerful tool to create images that are convincingly correct. This will be discussed further in Subsection 3.3

The interested reader who wants to learn more about path tracing, is referred to *Physically based rendering: From theory to implementation* by Pharr *et al.* [44] and "Guest Editor's Introduction: Special Issue on Production Rendering" by Pharr [45].

3 Related Work

The research area surrounding global illumination techniques and real-time filtering of monte carlo noise are prohibitively large to cover in its entirety, instead the most relevant and recent research works are covered here.

3.1 Global Illumination in Production Rendering

Production renderers are used to create computer generated imagery for use in movies and commercials by solving the global illumination problem [45], [46]. Achieving the realism and artistic direction required for modern day animated features is a complicated and computationally expensive task. We can learn from the work that has gone into the creation of the production renderers and what their strategies are to tackle the global illumination problem. Recently, most of the major production renderers released papers detailing their considerations and what is implemented in their renderers. All of the production renderers discussed use a form of Path Tracing [45], [47].

It is important to note that the constraints for games are very different compared to those of the production renderers [27]. The major difference in the problem area between games and production rendering is the amount of samples and computation time that is available for creating a final frame. Production renderers have 256 samples per pixel or more available per frame at almost arbitrarily long path lengths, and much more computation time to their disposal. Furthermore, games can make due with much coarser approximations than what is available to production renderers. The complexity, fidelity and number of the assets and light sources in scenes for games is at least an order of magnitude less.

The lessons from the production renderers that are transferable to our work are the following :

1. Both “Arnold: A Brute-Force Production Path Tracer” by Georgiev *et al.* [48] and “Sony Pictures Imageworks Arnold” by Kulla *et al.* [49] employ very aggressive variance reduction techniques. The roughness value is clamped to the highest value encountered during the tracing of path to reduce the impact that noisy caustics may have.
2. “RenderMan: An Advanced Path-Tracing Architecture for Movie Rendering” by Christensen *et al.* [50] uses denoising on every subdivision of illumination that is available to them and plan to further implement denoising during rendering.
3. “Manuka: A Batch-Shading Architecture for Spectral Path Tracing in Movie Production” by Fascione *et al.* [20] uses path guiding for adaptively sampling light sources, which attempts to learn the radiance distribution.
4. It also uses variance estimates of not only the pixel itself, but also the neighbourhood of the pixel. The estimates can be used to steer post-process filtering or adaptive sampling.
5. “The Design and Evolution of Disney’s Hyperion Renderer” by Burley *et al.* [51] adaptively samples based on variance estimates that are blurred to suppress the noise contained in the estimates themselves.
6. It furthermore takes a very aggressive approach to denoising fireflies (single bright pixels) by removing them entirely and replacing them with neighbourhood information.
7. “The Iray Light Transport Simulation and Rendering System” by Keller *et al.* [52] and “RenderMan: An Advanced Path-Tracing Architecture for Movie Rendering” by Christensen *et al.* [50] show the use of deterministic random sampling, which allows for the re-evaluation of samples by simply storing what random seed was used. This can be used to detect if information from the previous frame is still valid for re-use by simply checking if the result of what should be exactly the same evaluation is different, like “Gradient Estimation for Real-Time Adaptive Temporal Filtering” by Schied *et al.* [53].

All the renderers that were mentioned here attempt to solve the global illumination problem using path tracing in some form, with the claim that it makes the life and work-flow of the artist much

easier. Secondly, and just as important, path tracing seems to be the most feasible way to solve the global illumination problem in a high quality manner given the time and budget constraints of production rendering. Hyperion shows, that even if the artistic direction does not aim for photo-real graphics, anything that requires realistic lighting will benefit from the proper simulation of Light Transport [51]. Eventually, games will have to adopt the use of path tracing for realistic lighting [54]. Even though the quality of the final frame is extremely important, the production renderers use approximations and allow for errors that still seem perceptually correct instead of full convergence.

The major observation shared by all production renderers is that there is not enough time to render images to convergence by taking more samples, even while employing adaptive sampling. This stresses the fact that more samples is not the solution for getting rid of noise, but that we must look to other strategies to handle and reduce noise. Most of the renderers, combine advanced importance sampling techniques with the simplification and approximation of difficult effects that are major sources of noise, which most notably seems to be caustics [48], [49].

3.2 Ray Tracing & Path Tracing in Games

In research there are efforts to make path tracing available in real-time for games.

Most of these works are related to the techniques of Radiosity and Photon Mapping, which are worth revisiting with the new capabilities of the current day hardware. The most recent work is on "Dynamic Diffuse Global Illumination with Ray-Traced Irradiance Fields" by Majercik *et al.* [55]. Light Probes themselves have been used extensively in games to create global illumination solutions, but required a lot of manual tweaking by artists to avoid artifacts [55].

While ray tracing has been the method of choice for production rendering, it has historically been too computationally expensive to use in real-time interactive media like games. The seminal work "Ray Tracing for Real-Time Games" by Bikker [36] made major efforts to make ray tracing the rendering technique of choice for games. This later resulted in the "The Brigade renderer: A path tracer for real-time games" by Bikker *et al.* [56], which was a renderer aimed at using Path Tracing in Real-time for Games, albeit at low resolutions and running on multiple GPU's. At the time of that work there was no dedicated hardware acceleration available yet for ray tracing workloads.

The middleware solution called Enlighten [57], [58] has been using a form of real-time radiosity to achieve global illumination in games. This has seen use in many modern games.

Recently, the Ray Tracing Gems book by Haines *et al.* [59] has been released with a collection of topics to help developers on their way with creating real-time rendering solutions with ray tracing.

At the time of writing, multiple games and game engines have already adopted ray tracing for some effects or even as their main rendering technique :

1. *Q2VKPT* by Schied [60] and *Quake II RTX* by NVIDIA Lightspeed Studios [5] are the first games that are fully path traced in real-time with 1080p resolution [61].
2. *Battlefield V* by EA DICE [62] uses RTX for reflections and DLSS [63].
3. *MechWarrior 5 : Mercenaries* by Games [64] uses RTX for reflections, raytraced ambient occlusion and DLSS [65].
4. *Shadow of the Tomb Raider* by Eidos-Montréal and Crystal Dynamics [66] uses RTX for shadows and DLSS [67].
5. *Metro : Exodus* by 4A Games [68] uses RTX for diffuse global illumination and DLSS [69].
6. *Control* by Remedy Entertainment [70] uses RTX for indirect diffuse and specular illumination [69].
7. *Cyberpunk 2077* by CD Projekt Red [71] will use RTX for reflections [72].
8. *Unreal Engine 4* by Epic Games [73] and *Unity* by Unity Technologies [74] has RTX support implemented in their engines [75], [76].

As the list above shows, there is major interest by game developers to use ray tracing in their games in a hybrid format to get the strengths of both rasterization and ray tracing.

In subsection 3.1, filtering is still required even with the computational budgets alloted to production rendering. While remaining robust, temporally stable and consistently performant under much harsher frame budgets, the remaining noise of sampling in ray and path tracing must be dealt with. Moreover, somehow noise must be removed by increasing the sample count per pixel available to us in some manner while minding the diminishing returns of simply adding more samples. Filtering and reconstructing where and whenever possible seems to be the only way forward to obtain the goal of real-time path traced global illumination with current hardware.

3.3 Real-time Filtering of Monte Carlo Noise

The research area of filtering and reconstruction of noisy images is prohibitively large to cover in its entirety. In this section we focus on the basics of filtering and reconstruction that are required to understand the work of this thesis. Secondly, we focus on the current state of the art of filtering and denoising techniques for Monte Carlo rendering that are suitable for real-time use.

As Koskela *et al.* [77] and Zwicker *et al.* [78] mention, filtering and reconstruction methods may be divided into three categories : offline methods, interactive methods and real-time methods. Real-time methods are those that run above 30 frames per second and off-line methods have computational times that prevent an interactive workflow, and everything in between is classified as interactive.

Zwicker *et al.* [78] also makes an important distinction between methods that can work as a post-process which are called a posteriori methods and methods that need deep integration into the rendering pipeline which are called a priori methods. In a generalized sense, a posteriori methods work on the sampled information from the renderer. While a priori methods work on the information from the analytical analysis of the light transport equations that are employed by the renderer.

Reconstruction filters work by sharing information between pixels, which aggregates information and thus samples over larger image regions, while trying to avoid overly blurring the output [78].

The information can be shared in different domains. For instance, simple filtering can be done in image space where the input of the filter is only the pixels of the image. However, we can also consider exploiting the spatial coherence of samples and filtering in path space or other spatial domains. Furthermore, we can also exploit an assumption that information from previous frames is most likely still valid for the current frame. This allows for the reprojection of information from a previous frame to the current frame, allowing for filtering over the temporal domain.

For a more complete introduction and overview of state of the art of denoising images, we refer the reader to the literature study by Voorhuis [79], the overview paper by Zwicker *et al.* [78], the work by Bitterli *et al.* [80], and the recent work by Vogels *et al.* [81].

Many of the techniques for denoising and filtering are too computationally expensive to be used in a real-time setting [78], [82]. There are contemporary filters that are nearly real-time and are worth mentioning.

Koskela *et al.* [77] uses a regression-based reconstruction pipeline that combines multi-order feature buffers with QR factorization and feature regression that is processed in a blockwise manner, the method targets 1 sample per pixel input. All of their results were based on 720p resolutions and their filter runs at approximately 2.5ms, but suffers from overblurring, structural artifacts due to the block processing scheme and visual lag due to the reprojection that is employed.

Mara *et al.* [83] created a framework that combines rasterization, path tracing and multiple filters, but assumes that direct illumination is noise-free which only happens if the direct light samples that are taken are greater than or equal to the amount of light sources in the scene.

Chaitanya *et al.* [84] used a recurrent auto-encoder for denoising, but suffers from temporal instability and struggles to maintain real-time performance that is needed for games.

Schied *et al.* [53] continued the work of Chaitanya *et al.* [84] by developing a temporal adaptive filtering term to increase the visual quality in dynamic scenes.

In the same paper Schied *et al.* [53] extended on their previous work [82], the hierarchical filter with custom edge stopping functions, with temporal adaptive filtering. This work was the basis of the *Q2VKPT* by Schied [60] which was later extended to become *Quake II RTX* by NVIDIA Lightspeed Studios [5] [61]. This is the first work that targets 1 sample per pixel (spp) on 60 frames per second on a Full HD resolution that creates noise-free temporally stable images in real-time.

In a separate concurrent work to Schied *et al.* [53], Voorhuis [79] extended the original filter by Schied *et al.* [82] to handle reflections by considering reflected motion vectors for reprojection using a novel application of a diamond search algorithm.

These works are all limited by the maximum path depth that can be traced in a single frame to obtain the indirect illumination.

Secondly, the SVGF methods work screen-space and rely on the reprojection of samples across frames, which maps information from the previous frames to the current frame. Then, with some similarity function, it can be determined if the information samples are similar enough for re-use. All the screen-space methods can only work the information that is available on the screen and even it is vulnerable to multiple types of artifacts, for example :

- Incorrect Shadows, imagine an object that is in motion that casts a shadow, with naive Motion Vectors, the shadow will incorrectly move with the object. ASVGF alleviates this with the temporal adaptive filtering term.
- Ghosting, imagine a reflection of a moving object and a moving camera, if motion vectors are used naively the reflection will incorrectly move with the camera movement. This is what Voorhuis [79] combats with the reflected motion vectors.
- Searching, Motion Vectors give an indication where you can find a pixel that had the sample of the previous frame, but you must still search to find the correct one. This is illustrated in “Beyond Spatiotemporal Variance-Guided Filtering: Temporally Stable Filtering of Path-Traced Reflections in Real-Time” by Voorhuis [79] with the use of the diamond search algorithm to find the correct sample to reproject for reflections.

However, path space filtering methods do not suffer from the same screen-space issues. In the next section we will discuss the path space filtering methods in more detail because these form the basis of the algorithms that we have developed during our research.

3.4 Path Space Filtering (PSF)

As discussed before, filtering is required to enable ray and path tracing in real-time for games. Filtering can be done in screen space, as is done in Q2RTX with their implementation of ASVGF [53], requiring to re-project information from previous frames to the current frame. However, we can also filter in a different domain, such as path space.

Normally during tracing, a path is created by sampling illumination and other properties at intersections with geometry, these intersection locations can be seen as path vertices. While a path usually consists of multiple path vertices, normally no information is shared between neighbouring path vertices.

“Path space filtering for integro-approximation problems” by Keller *et al.* [2] works by averaging radiance information between a path vertex and path vertices are neighbouring in world space that fulfill similarity criteria. These similarity measures are e.g. world-space neighbours, similar normals or material layer properties.

The intuition of the process is that any path vertex that is neighbouring sufficiently closely enough will in all likelihood receive similar radiance. Of course this is not true for all vertices, which is why similarity measures are employed. For each path vertex, a neighbourhood search is performed by centering a sphere on the vertex with a radius and looking what other vertices lie within the boundaries of said sphere. Then, a consistent weighted average is calculated with every neighbouring

input at 1 sample per pixel | path space filtered | filter iterated 4 times

Figure 4: Iterated weighted averaging very efficiently smooths the solution by relaxation at the cost of losing some detail. Obviously, path space filtering brightens up the image by replacing black pixels of the input with the weighted averages. Model courtesy of G. M. Leal LLaaguno. Original image and caption reproduced with permission from Keller *et al.* [2].

vertex that passes the similarity tests, e.g. having a similar normal. The radius of the sphere is determined by the amount of samples the path vertex has received, shrinking with more samples. This means that in the limit we converge to what the actual radiance should be at the path vertex. The averaged radiance can then be multiplied by the correct attenuation at the filtered vertex, restoring texture quality.

It is important the albedo is not multiplied in during the averaging process. This seperation of terms is required so that crisp texture detail is not blurred.

We will discuss the details of the method by borrowing the math from the original paper by Keller *et al.* [2]. When considering the i-th of a total of n paths, vertex x_i is selected for filtering the radiance contribution c_i of the light path segment towards x_i by a filter criteria, which is recommended to be the first sufficiently diffuse path vertex. The reason for this is that the diffuse illumination is independent of the viewing angle. During the sampling of path space the attenuation α_i along the path segment towards the camera is determined as well.

With path space being sampled in batches of $b^m \in \mathbb{N}$, for each path one tuple of (x_i, α_i, c_i) is stored for path space filtering. The batch of b^m paths is processed starting at index $s_i := \lfloor \frac{i}{b^m} \rfloor b^m$ where the output is generated by accumulating $\alpha_i \cdot \overline{c}_i$. As the memory consumption is proportional to the batch size b^m, the maximum natural number m is determined given the size of the tuples and the maximum size of a memory block. The $\overline{c}_i$ is the weighted average of contributions c_{s_i+j} of all vertices in a ball $\beta(n)$ of radius $r(n)$ centered in x_i normalized by the sum of weights $w_{i,j}$ as is calculated in Equation 6.

$$\overline{c}_i := \frac{\sum_{j=0}^{b^m-1} \chi_{\beta(n)} \cdot (x_{s_i+j} - x_i) \cdot \omega_{i,j} \cdot c_{s_i+j}}{\sum_{j=0}^{b^m-1} \chi_{\beta(n)} \cdot (x_{s_i+j} - x_i) \cdot \omega_{i,j}} \tag{6}$$

The ball $\beta(n)$ centered in x_i has an initial radius r_0, given that we have non-zero weights $w_{i,j}$ the radius is determined by Equation 7.

$$r(n) = \frac{r_0}{n^\gamma} \text{ for } \gamma \in (0, 1) \tag{7}$$

With the radius of the ball being determined in Equation 7 and the i-th path is always included in the characteristic function $\chi_{\beta(n)}$, it follows that the radius vanishes with the total number of n paths, which guarantees that $\lim_{n\to\infty} \overline{c}_i = c_i$ and thus consistency.

In other words, the consequence is that all the artifacts will vanish with more samples. This also still holds when the filter is iterated, thus creating an iterated weighted average. Keller *et al.* [2] note that, having a large radius to include as many contributions as possible, directly competes with, having the smallest possible radius to hide all the transient artifacts. The benefits and the consequences of iterated filtering can be seen in Figure 4 and the results of progressive path space filtering can be seen in Figure 5.

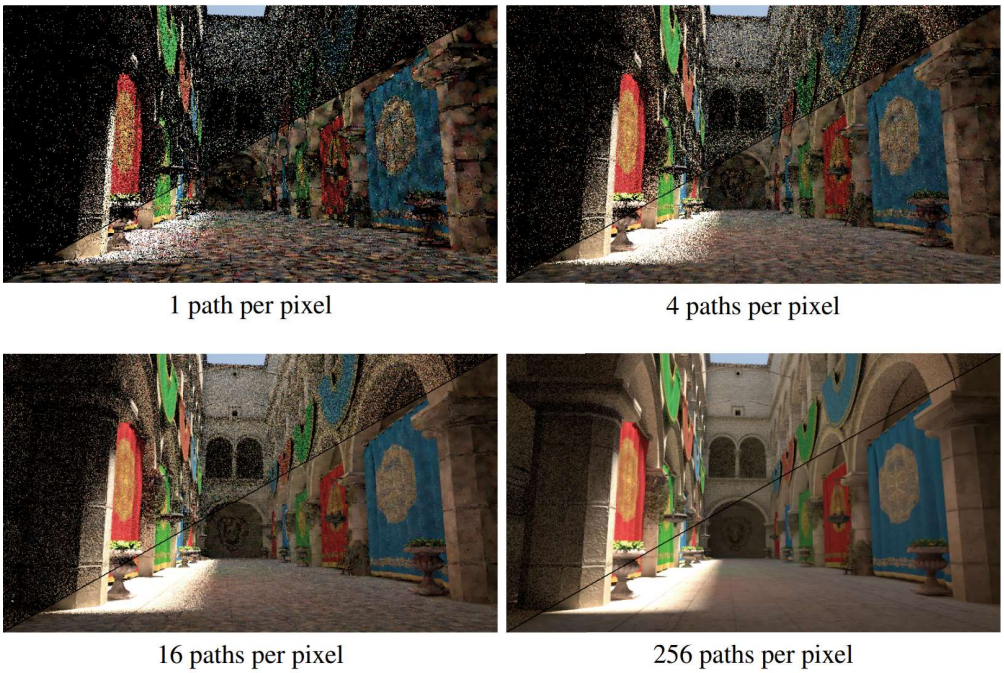


Figure 5: The series of images illustrates progressive path space filtering. Each image shows the unfiltered input above and the accumulation of weighted averages below the diagonal. As more and more batches of paths are processed, the splotchy artifacts vanish due to the consistency of the algorithm as guaranteed by the decreasing range search radius $r(n)$. Model courtesy of M. Dabrovic and Crytek. Original image and caption reproduced with permission from Keller *et al.* [2].

The paper proposes further applications in light transport simulation such as animations, motion blur, spectral rendering, translucency, participating media, and more.

The most costly part of using this filtering technique is the neighbourhood search. The original paper does not mention any performance statistics, but only qualitative results in the form of figures. Path space filtering was extended upon by Gautron *et al.* [85], where they proposed a similarity estimation of light transport paths using Fourier Histrogram Descriptors. This was implemented in the NVIDIA Iray Interactive renderer[52] and searches on pixel windows to determine the Fourier histogram descriptors. The results of this technique were reported to be on 1280 x 720 resolution the histograms took 32ms to generate on a 5x5 pixel grid and the subsequent filtering using a 21x21 pixel window required 85ms on a NVIDIA Tesla K20c GPU. This is still a stretch away from having a filtering technique that allows us to run on 1920x1080 with 60 fps in 16.66ms. In the defense of Gautron *et al.* [85], their work was done with a production renderer using production rendering quality assets as opposed to game quality assets. Furthermore, the path lengths and the amount of lights were not mentioned in the results.

4 Methods

In this section we describe the research that was performed for this thesis and the methods that were implemented. First, we will explore a method that investigates the benefits of a screen-space search for finding path space neighbours called Screen-Space Path Space Filtering (SSPSF) in Subsection 4.1. Then, we will describe the Hashed Path Space Filtering (HPSF) method that forms the basis of the remaining algorithms that are presented in this work, as well as the extensions to this method that were implemented in Subsection 4.2. Furthermore, we will present a novel new technique that filters on multiple selected bounces along a light transport path which allows for reducing variance called recursive hashed path space filtering (nHPSF) in Subsection 4.3. It can also transport information between hash cells during scattering. Lastly, we describe our main result of the thesis, which is a novel algorithm that amortizes the cost of tracing long light transport paths over multiple frames and combines it with the nHPSF technique called Amortized nHPSF (A-nHPSF) in Subsection 4.4.

4.1 Screen-Space Path Space Filtering (SSPSF)

Inspired by the work of Gautron *et al.* [85] and the idea of testing out how good a screen-space search could be for finding path space neighbours, SSPSF has been created.

We have discovered that it is important that the albedo is not pre-multiplied during the BSDF calculations, which is done in other works [82]. We run into numerical floating point stability issues if the albedo is multiplied in and has to be divided out. Furthermore, if the albedo is zero in a color channel, radiance information is unrecoverably lost and the calculation is undefined. For example, if two path space neighbours have spatially differing albedos, they can still benefit from the radiance that is received on one another. If clamping was used to avoid division by zero, there would still be precision loss during the division step. Secondly, the operation is unnecessary, by not premultiplying a division and multiplication operation is avoided.

Besides that this prevents the blurring of texture detail, this is needed so that the terms can be separated cleanly during tracing when the filter criteria is met.

Like standard PSF, the terms are seperated, averaged and recombined, but the expensive world space neighbourhood search is replaced by a screen-space search for neighbours using similarity measures e.g. world space location and normals. Our tracing process for this method is illustrated in Figure 6 and algorithm 1.

The filter criteria is simply the first sufficiently diffuse path vertex, as was used by Gautron *et al.* [85]. When the filter criteria is met, the world position, the normal and the throughput of the path segment multiplied with the albedo of the path vertex is stored for recombination after filtering. Furthermore, the path throughput for the upcoming segments is set to the BSDF of the path vertex and the path is flagged for filtering. Then after tracing is completed, for every pixel a 9x9 pixel

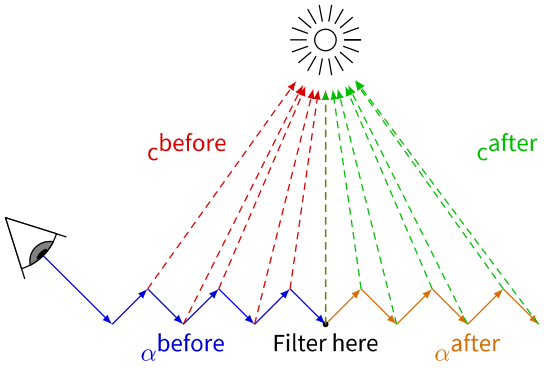


Figure 6: Path space filtering - Seperation of terms.

Input: World position $\texttt{pos}_\texttt{w}$, Path segment origin $\texttt{pos}_\texttt{o}$, Unfiltered path travel distance $\texttt{d}_\texttt{up}$, Diffuse BSDF Lobe $\texttt{BSDF}_\texttt{d}$, Specular BSDF Lobe $\texttt{BSDF}_\texttt{s}$, Incident radiance `Li`, Current path throughput `currThp`, Radiance before filtering `LradBF`, Attenuation in filter point `attenIF`, Radiance after filtering `LradAF`,
Data: Albedo `albedo`, Light Source Visible `visible`,

```
bool filterCriteria ← any(BSDF_d · albedo) ;
if ¬ filterCriteria then
    if visible then
        float3 LradBF ← LradBF + currThp · ((Ldiff · albedo) + Lspec)
    float3 currThp ← currThp · ((BSDF_d · albedo) + BSDF_s);
    float3 d  ← abs(pos_w − pos_o);
    float d_up ← d_up + max(d.x, d.y, d.z);
else
    if visible then
        float3 LradBF ← LradBF + max(0,  Lspec · currThp) ;
        float3 LradAF ← LradAF + Ldiff;
    float3 attenIF ← (BSDF_d · albedo) + BSDF_s;
    float3 currThp ← float3(1) ;
end
```

Algorithm 1: PSF pseudocode: Path tracing.

window search is performed in screen-space. In the search a similarity check is used on the world position, normal, and whether the neighbour was flagged for filtering. A weighted average for the radiance contribution is created, which is recombined with the stored terms and used for generating the output.

4.2 Hashed Path Space Filtering (HPSF)

Rendering a noise-free image with Path Tracing has been shown to take considerable amounts of time and is not suitable for real-time applications like games just out of the box. Path space filtering is a denoising technique that functions in path space, but the main drawback of the technique is the expensive k-Nearest Neighbour search that takes place in world space. HPSF tackles this limitation by replacing the nearest neighbour search by a constant time look-up and linear probe in a hash map by novel use of quantizations, path vertex descriptors, and hashing.

4.2.1 Basic HPSF

The algorithm works in two major steps :

1. Path vertex descriptor generation and quantization
2. Hash map storage and querying

First, a filter criteria is defined, our criteria is the first path vertex that samples the diffuse BSDF lobe. Then, to perform filtering, for every path vertex that meets the filter criteria a path vertex descriptor is created that at least contains the world position. This descriptor is then quantized which creates an implicit voxelization in world-space (see Algorithm 3). The descriptor is then hashed twice separately by using two different hashing functions to create a hash key for indexing and a hash key for verification (see Algorithm 2). It is very important that two different hashing functions are employed, so that the differences in the hashing functions can be exploited. The hash index is used to index into the hash map, while the verification hash is used to resolve collisions and has more advanced uses (see Section 4.2.2). A verification hash is employed, instead of storing the full key as the full key can be rather large, depending on what information is used to create the key. Every entry in the hash map can be considered a cell or voxel.

To store information, it is necessary for the intended cell to either be free or have a matching verification hash. If the cell is not free and the verification hashes do not match, a collision occurs. Collisions are resolved with limited linear probing over the subsequent indices in the hash map to find either a free spot or a cell with matching verification hashes (see Algorithm 4). Due to cache line behaviour where blocks of memory are prefetched, this linear probing is nearly free to use [1].

Retrieving information from the hash map is then as simple as storing what hash map index and verification hashes were used per pixel that are flagged for filtering, which can then be used to do a single look up to find the filtered radiance. If no filtering occurred, the raw radiance can be used for that pixel as a fallback strategy (see Algorithm 5).

4.2.2 HPSF Advanced Uses & Extensions

The quantization and linear probing can also be used in advanced ways. For instance, by including quantized normals as information for creating the hashes, we can prevent filtering across normals that are significantly different.

Quantizing Normals: By reserving a certain amount of bits in the hash keys for quantized normals, we can implicitly filter with respect to different normals (see algorithm 2). The amount of normal bits used would then create 2^b different cells where b is the number of bits for the quantized normal information.

It is also possible to include the normal information in the verification hash but not in the hash index key and instead mask out the least significant bits of the index hash. This would cause the descriptors that are similar in world space to hash towards the same starting cell but have different verification hashes. The collision would then allow us to linear probe over different quantized normals that are in the same quantized world position.

In section 5 we show the difference between using normal information in the hash keys with linear probing over different quantized normals and filtering without such information.

Quantization Artifacts: The implicit voxelization due to the quantization creates structured artifacts visible as boundaries of the voxels (see Figure 9). To alleviate these artifacts, jittering can be employed. Instead of directly quantizing the world position of a hit, we can randomly jitter the world position in the tangent plane given its normal before quantizing and storing it. This will replace the structured artifacts by uniform noise.

Moreover, this jittering can be employed multiple times on the same sample to allow its information to end up in multiple hash cells. Naturally doing this will results in more noise but less visible quantization artifacts. This specific extension was not explored in this thesis.

Level of Detail: The quantization of the path vertex descriptors can also be used to handle level of detail changes. By including information of the projected size of the desired filter width on the screen and the travel distance of the path until it was filtered, the size of the quantization can dynamically be adjusted proportionally to the travel distance of the path. Voxels that are further away from the camera, or had a longer travel distance will then become larger, and thus share more neighbour information and vice versa. This extension is implemented in the results of this work.

Temporal Accumulation & Eviction: Even with the sharing of radiance information between many neighbours, being noise-free is not guaranteed within a single frame, this is where temporal accumulation comes in. By using two versions of the hash map, one for the previous frame and one that is used to collect all the new information for this frame, we can perform accumulation. At the beginning of every frame, the hash map are swapped and the one intended for the current frame is cleared. For every hash cell that is used in the current frame, the hash cell can be found in

Input: Quantized position $\mathtt{pos}_\mathtt{q}$, Level of Detail $\mathtt{l}$, Normal $\mathtt{n}$, Iteration $\mathtt{i}$, Pixel Index $\mathtt{p}$, Normal Bits $\mathtt{b}$, Jitter Normal $\mathtt{j}$
Output: Hash key $\mathtt{key}$

$\mathtt{uint\ key} \leftarrow hash(\mathtt{pos}_\mathtt{q}.\mathtt{z});$
$\mathtt{key} \leftarrow hash(\mathtt{key} + (\mathtt{pos}_\mathtt{q}.\mathtt{x}));$
$\mathtt{key} \leftarrow hash(\mathtt{key} + quantizeNormal(\mathtt{n}, \mathtt{i}, \mathtt{p}, \mathtt{b}, \mathtt{j}))\,;$ /* Optional */
$\mathtt{key} \leftarrow hash(\mathtt{key} + (\mathtt{pos}_\mathtt{q}.\mathtt{y}));$
$\mathtt{key} \leftarrow hash(\mathtt{key} + \mathtt{l});$
return *key*

Algorithm 2: HPSF hash key creation pseudocode, with e.g. WangHash or XorShift32.

the hash map from the previous to be summed together with information from the current frame. This forces a very strict implicit eviction strategy, that cells that are not touched between frames are not copied over and thus cleared. There are also other options for eviction strategies, but these were not explored in this thesis research.

In the case of switching to a different level of detail, we may to end up in a cell with little to no information, since we are reading from and writing to different cells in the hash map. This case can be handled by searching across different levels of detail and retrieving information there, which is basically jittering or dithering over different levels of detail.

4.3 Recursive HPSF (nHPSF)

Though the original HPSF filters on the first valid bounce, it is also possible to filter on every valid diffuse bounce of a path. This can be done in a recursive fashion, extending the original HPSF to a slightly different technique we call nHPSF.

The crux of this method lies in the filtering and transporting radiance information from the hash cell of the scattered bounce to the hash cell of the primary filter hit. The diffuse reflected radiance from the scattered filter hit will contain less high-frequency information and can thus be filtered more aggressively. In the most naive way, this would be performed by storing all of the radiance information that is found along a path and processing it in reverse, starting at the scattered hit, then the primary hit etc. However, this would introduce multiple synchronization dependencies, increase the memory bandwidth required during tracing and ultimately slowing the entire tracing process down.

Instead, this is implemented by a scheme that transports hashed radiance information from the previous frame to the current frame, by using the scatter hit location to access the hashmap and transport the information to the primary filter hit. The results of this method can be seen in subsection 5.3.

4.4 Progressive Global Illumination through Amortized nHPSF (A-nHPSF)

The problem with the previous filter methods and the current state of the art methods, is the limit on the path length of tracing. Even with the filtering techniques discussed in the previous chapters, we only access a single indirect bounce of light. Due to the strict frame budget, a single pixel only has a few rays available to be cast to create a path.

Recall that global illumination effects come from the summation of direct and indirect light. The indirect light can be reflected many times in a recursive manner. This means that if we have an upper limit to how deep the paths can be, then there are complex light interactions that we will not find.

With the previous filter method, nHPSF, we introduced the capability of transporting radiance between hash cells across frames. The HPSF methods also temporally accumulates information

Input: World Position $\mathtt{pos_w}$, Normal $\mathtt{n}$, View Plane Distance $\mathtt{d_{vp}}$, Pixel Filter Width $\mathtt{f_p}$, Jitter Position $\mathtt{pos'_w}$, Path Segment Origin $\mathtt{pos_o}$, Unfiltered path travel distance $\mathtt{d_{up}}$, Pixel Index $\mathtt{i}_p$, Iteration $\mathtt{i}$

Output: Quantized World Position $\mathtt{pos_q}$, Level of Detail $\mathtt{lod}$, Jittered Quantized World Position $\mathtt{pos_{qj}}$, Jittered Level of Detail $\mathtt{lod'}$

$\mathtt{vec3\ d_{cp}} \leftarrow abs(\mathtt{pos_o} - \mathtt{pos_w})$; /* Travel Distance of Current Path Segment */
$\mathtt{float\ d_t} \leftarrow \mathtt{d_{up}}$;
if $(d_{cp}.x > d_{cp}.y) \wedge (d_{cp}.x > d_{cp}.z)$ **then**
  $\mathtt{d_t} \leftarrow \mathtt{d_t} + \mathtt{d_{cp}.x}$;
else if $d_{cp}.y > d_{cp}.z$ **then**
  $\mathtt{d_t} \leftarrow \mathtt{d_t} + \mathtt{d_{cp}.y}$;
else
  $\mathtt{d_t} \leftarrow \mathtt{d_t} + \mathtt{d_{cp}.z}$;
end
$\mathtt{float\ width} \leftarrow \mathtt{f_p}/\mathtt{d_{vp}} \cdot \mathtt{d_t}$;
$\mathtt{uint\ lod} \leftarrow asint(\mathtt{width})\&\mathtt{0xFF800000}$; /* Extract exponent so result is 2^m */
$\mathtt{float\ width_q} \leftarrow asfloat(\mathtt{lod})$; /* Quantized Width */
/* Position Jittering */
$\mathtt{uint\ h} \leftarrow hash(\mathtt{i} + 104729 \cdot \mathtt{i_p})$; /* Rank 1 lattice sequence to decorrelate */
$\mathtt{float3}\ \delta \leftarrow \mathtt{float3}(-1.0)$;
$\delta \leftarrow \delta + 2.0 \cdot \mathtt{float3}(((\mathtt{h}\&1023)/1024.0), ((\mathtt{h}>>10)\&1023)/1024.0), ((\mathtt{h}>>20)\&1023)/1024.0)$;
$\delta \leftarrow \delta - (|\mathtt{n}| \cdot \delta)$; /* Gram-Schmitt, removes all tangential parts */
/* If jittering offset is used, filter width becomes wrong */
$\mathtt{float3\ pos'_w} \leftarrow \mathtt{pos_w} + \mathtt{j_{pos}} \cdot \mathtt{width_q} \cdot \delta$;
$\mathtt{float3\ d'_{cp}} \leftarrow |\mathtt{pos_o} - \mathtt{pos'_w}|$;
$\mathtt{float\ d'_t} \leftarrow \mathtt{d_{up}}$;
if $(d'_{cp}.x > d'_{cp}.y) \wedge (d'_{cp}.x > d'_{cp}.z)$ **then**
  $\mathtt{d'_t} \leftarrow \mathtt{d'_t} + \mathtt{d'_{cp}.x}$;
else if $d'_{cp}.y > d'_{cp}.z$ **then**
  $\mathtt{d'_t} \leftarrow \mathtt{d'_t} + \mathtt{d'_{cp}.y}$;
else
  $\mathtt{d'_t} \leftarrow \mathtt{d'_t} + \mathtt{d'_{cp}.z}$;
end
$\mathtt{float\ width'} \leftarrow \mathtt{f_p}/\mathtt{d_{vp}} \cdot \mathtt{d'_t}$;
$\mathtt{uint\ lod'} \leftarrow asint(\mathtt{width'})\&\mathtt{0xFF800000}$; /* Extract exponent so result is 2^m */
$\mathtt{uint\ width'_q} \leftarrow asfloat(\mathtt{lod'})$; /* Quantized Filter Width considering jittering */
$\mathtt{float3\ pos_q} \leftarrow \mathtt{int3}(\mathtt{pos_w}/\mathtt{width_q})$;
$\mathtt{float3\ pos'_q} \leftarrow \mathtt{int3}(\mathtt{pos'_w}/\mathtt{width'_q})$;
return pos_q, l, pos'_q, l'

Algorithm 3: HPSF pseudocode: Vertex descriptor quantization.

in path space using the hashmaps. With these two things, it is possible to transport information across frames.

We propose a new method that exploits these two things, which requires two rays to be available in the ray budget per pixel to continue the path across frames. This allows us to amortize the cost of tracing a deep path over multiple frames where every path vertex is filtered with nHSPF. The radiance that this path will find can be transported back over multiple frames in the same manner that nHPSF transports radiance between cells.

This method was implemented using a wavefront path tracer. For the sake of simplicity, the method is currently focused on diffuse lighting interactions, but can be extended to deal with

Input: Index hash $\texttt{key}_{\texttt{Idx}}$, Verification hash $\texttt{key}_{\texttt{v}}$, Radiance rad, Radiance hash map $\texttt{hm}_{\texttt{rad}}$, Verification hash map $\texttt{hm}_{\texttt{v}}$, Hash map capacity $\texttt{hm}_{\texttt{cap}}$, Probing width probe,
Output: Table Index tableIndex

uint firstFree $\leftarrow \neg 0$;
uint tableIndex $\leftarrow \neg 0$;
for *uint probeIndex* $\leftarrow 0$ **to** *probeIndex* $<$ *probe* **do**
  tableIndex $\leftarrow (\texttt{key}_{\texttt{Idx}} + \texttt{probeIndex}) \mod \texttt{hm}_{\texttt{cap}}$;
  if $hm_v.Load(tableIndex) == key_v$ **then**
    $\texttt{hm}_{\texttt{rad}}.AtomicAdd(\texttt{tableIndex}, \texttt{rad})$;
  else if $hm_v.Load(tableIndex) == 0$ **then**
    firstfree $\leftarrow$ probeIndex;
    break;
  end
end
for *uint startIndex* $\leftarrow$ *firstFree* **to** *startIndex* $<$ *probe* $+$ *firstFree* **do**
  tableIndex $\leftarrow (\texttt{key}_{\texttt{Idx}} + \texttt{startIndex}) \mod \texttt{hm}_{\texttt{cap}}$;
  uint originalVal $\leftarrow \neg 0$;
  uint compareVal $\leftarrow 0$;
  $\texttt{hm}_{\texttt{v}}.AtomicCompareExchange(\texttt{tableIndex}, \texttt{compareVal}, \texttt{key}_{\texttt{v}}, \texttt{originalVal})$;
  if $originalVal == 0 \vee originalValue == key_v$ **then**
    $\texttt{hm}_{\texttt{rad}}.AtomicAdd(\texttt{tableIndex}, \texttt{rad})$;
    break;
  end
end
return *tableIndex*

Algorithm 4: HPSF pseudocode : Collect contributions

Input: Table index i, Verification hash keyv, Radiance before filtering radBF, Filter Radiance radAF, Attenuation attenuation, Radiance hash map $\texttt{hm}_{\texttt{rad}}$, Verification hash map $\texttt{hm}_{\texttt{v}}$
Output: Output Color outputColor

float4 outputColor $\leftarrow$ float4(0) ;
if $valid(i) \wedge valid(key_v)$ **then**
  radAF $\leftarrow readRadianceHashMap(\texttt{i}, \texttt{key}_{\texttt{v}}, \texttt{hm}_{\texttt{rad}}, \texttt{hm}_{\texttt{v}})$;
end
outputColor $\leftarrow$ radAF;
outputColor $\leftarrow$ outputColor $\cdot$ attenuation;
outputColor $\leftarrow$ outputColor/outputColor.w;
outputColor $\leftarrow$ outputColor $+$ radBF;
return *outputColor*

Algorithm 5: HPSF pseudocode: Generate output

specular reflections. We will now discuss the rendering pipeline that was used, which is split into 6 parts.

1. GBuffer shader, generates the primary hit and the material information of that hit.
2. GeneratePaths shader, which processes the primary hit and generates the NEE and scatter ray. It also generates a hash key and checks if it is valid by trying to find a empty or matching cell from the previous frame. If it is valid and the filtering criteria is met, we know that the hash map can be used to query or store radiance. Lastly, it updates the scatter state for A-nHPSF path continuation.
3. The raytracing loop continues until maximum path depth is reached and consists of two parts:

(a) Trace Rays shader, traces the scatter and NEE rays.

(b) Process Paths shader, accumulates the NEE rays and the reflected emission from the scatter ray. It also performs the filtering with the splitting of terms and transports the radiance to the ray origin from the newly found scatter location by storing the radiance found in the hashmap. Lastly, it updates the scatter state for A-nHPSF path continuation.

4. Trace A-nHPSF path continuation, given that we have a path that can be continued, the rays of this path are traced.
5. Process A-nHPSF path continuation, processes the path in the same manner as the Process Path Shader.
6. Write output, given a validly filtered primary hit, the output is generated in exactly the same way as it is done in nHPSF.

To allow the path to continue over multiple frames, the precedence order for updating the A-nHPSF path continuation state is the deepest valid path that is still below the maximum path continuation depth.

In the previous filter methods the eviction strategy is simpler, mainly because it is expected that rays will end up in similar cells across frames. This assumption does not hold with this method, as paths can continue randomly and it is not garantueed that a cell will have a contribution added to it in a frame. But, the cells can contain information that we want to keep so that it can be transported across frames. This also highlights the difficulty of making this method capable of handling dynamic scene content, since it becomes difficult to detect what radiance information is still valid.

5 Results

In this section we discuss the results that were obtained during this research. All of the experiments are built upon the work of "Importance Sampling of Many Lights on the GPU" by Moreau *et al.* [86] which was implemented in the Falcor framework [87]. The experiments were run on a machine that had a i7-5960X processor @ 3.50GHz and a RTX 2080 Ti. The Falcor framework and the ray tracer that was used to conduct experiments was still under development and was ready to be optimized. For this reason we could not take representative performance measurements. What can be said, is that a synchronized hash-map write for all threads on a 1920x1080 resolution takes an average of 1.5ms on the GPU.

All experiments were run at frame 30 of the Bistro Exterior scene and frame 30 of the Classroom Scene where applicable and all the scenes where fully diffuse.

We will first show the results of the separate filters and then compare the best performing configurations with one another using Figure 18. The results are compared using the settings that yielded the most visibly pleasing result of each filter and are then compared given 1, 16 and 32 frames of information. The goal of this comparison is to show the differing strengths and weaknesses of each filter point. A-nHPSF is an exception, since there are only results of the filter working on 32 frames, but has been added for the sake of completeness.

5.1 SSPSF

(a) raw 1 spp input (b) 1 spp (c) 2 spp

(d) 16 spp (e) 32 spp (f) raw 8192 spp ground truth

Figure 7: SSPSF results with multiple spp configurations using a 9x9 pixel kernel compared to unfiltered results.

The results of SSPSF can be seen in Figure 7 and 8.

There are blocky artifacts that are introduced with SSPSF due to its pixel neighbourhood search. Furthermore, the shadows are blurred out compared to the ground truth. The artifact surrounding the edges of the backrests of the chairs in shadow in the classroom scene (see Figure 8e) is interesting, there is a clear edge of light surrounding them. However this artifact does seem to be present in the Bistro scene.

Even so, it can be seen that with more samples per pixel the filter quickly approximates the ground truth and produces less noisy images on 1 spp. However, texture detail is fully kept because only the radiance is filtered.

(a) raw 1 spp input (b) 1 spp (c) 2 spp

(d) 16 spp (e) 32 spp (f) raw 8192 spp ground truth

Figure 8: SSPSF results with multiple spp configurations using a 9x9 pixel kernel compared to unfiltered results.

(a) Width 4 (b) Width 8 (c) Width 16

(d) Width 24 (e) Width 32 (f) Width 64

Figure 9: HPSF results by filtering on the first valid bounce using different filter widths with 1spp.

5.2 HPSF

The results of HPSF can be seen in Figure 9, 10 and 11.

The big artifact that HPSF introduces is the blockiness of the voxels that are created where radiance is averaged together. When the filter width is increased the noise decreases, but detail is lost and the block pattern artifact becomes larger. On a filter width above 32, noise is almost no longer present, but the blocks are enormous. Which can be seen for instance in the shadow underneath the moped on filter width 64 in the Bistro scene (see Figure 9f).

(a) Width 4 (b) Width 8 (c) Width 16

(d) Width 24 (e) Width 32 (f) Width 64

Figure 10: HPSF results by filtering on the first valid bounce using different filter widths and jittering the filter hit by 0.6 along the tangent plane with 1spp.

(a) Position unjittered without normal bits (b) Position unjittered with 2 normal bits

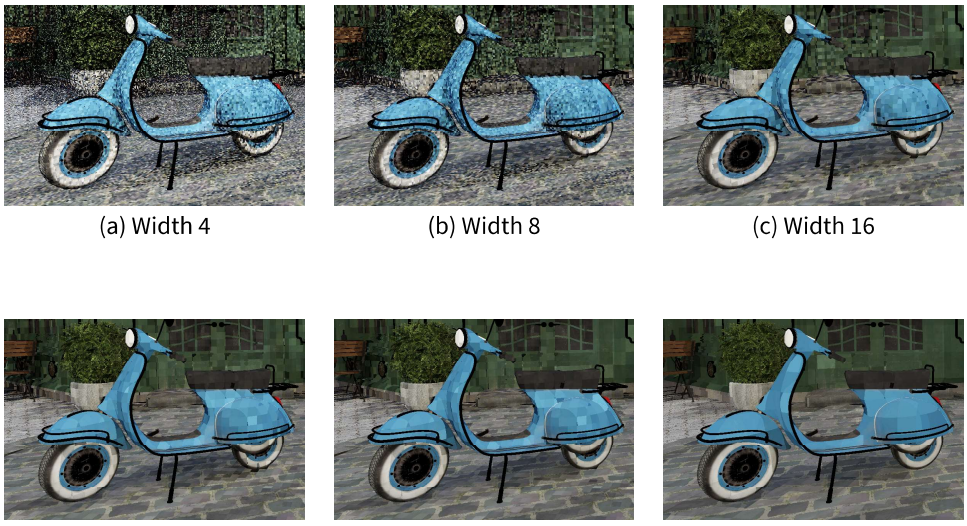
(c) Position unjittered with 2 normal bits, normal jittering of 0.2 (d) Position jittered with 0.6, with 2 normal bits, normal jittering of 0.2

Figure 11: HPSF results with 2 normal bits being used in the hash descriptor with 1 spp.

This block pattern artifact can somewhat be hidden by employing jittering, although this introduces uniform noise. When this is employed, even a filter width of 16 already seems acceptable considering that it only has 1 spp of input available (see Figure 10c). Nonetheless, shadow detail is lost, too, due to the blurring of radiance. However, like SSPSF, texture detail is fully preserved as only the radiance is filtered.

When normal information is used during filtering, detail in areas with different normals is kept, in particular in areas with shadows. This can be seen on the door behind the moped in Figure 11b. When we jitter the normals, there is still detail kept that is not present in the results without the normal bits. However, the jittering of the normal and position loses detail in the images while attempting to hide the quantization artifacts.

5.3 nHPSF



(a) Width 4 (b) Width 8 (c) Width 16

(d) Width 24 (e) Width 32 (f) Width 64

Figure 12: nHPSF results by filtering on the first and second valid bounce using different filter widths with 2 frames of information.

The results of nHPSF can be seen in Figure 12, 13, and 14.

The filter causes blocky artifacts due to the voxelization that is employed. What is noticable is the fact that the shadows are too bright. There seems to be too much radiance available.

When comparing HPSF to nHPSF, there are is a stark difference in the radiance levels when there is more than 1 frame of information available for nHPSF. The shadow on the door behind the moped is missing and so is the shadow of the fence in front of the moped. With that said, after 16 spp nHPSF looks virtually noise-free and only the discretization artifacts are still visible.

When normal information is used during filtering, the radiance no longer seems overblurred and shadow detail is kept. However, the quantization artifacts become quite pronounced when there is no jittering applied (see Figure 14b). When jittering of the normal is used, then even more shadow detail is traded for more uniform noise. For instance, this is visible in the shadow of the spare tire of the moped and in the shadow of the awning of the Bistro (see Figure 14c). When both the normal and position is jittered, there is quite some noise but the most of the quantization artifacts are hidden (see Figure 14d).

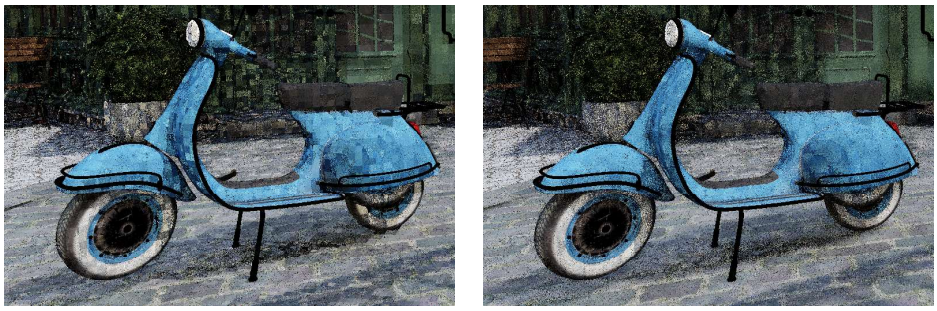
When the results of HPSF and nHPSF with normal information are compared, nHPSF has more shadow detail but also more uniform noise due to jittering taking place on both the first and secondary filter hits. When comparing HPSF to nHPSF, there are is a stark difference in the radiance

(a) Width 4 (b) Width 8 (c) Width 16

(d) Width 24 (e) Width 32 (f) Width 64

Figure 13: nHPSF results by filtering on the first and second valid bounce using different filter widths and jittering the filter hit by 0.6 along the tangent plane with 2 frames of information.

(a) Position unjittered without normal bits (b) Position unjittered with 2 normal bits

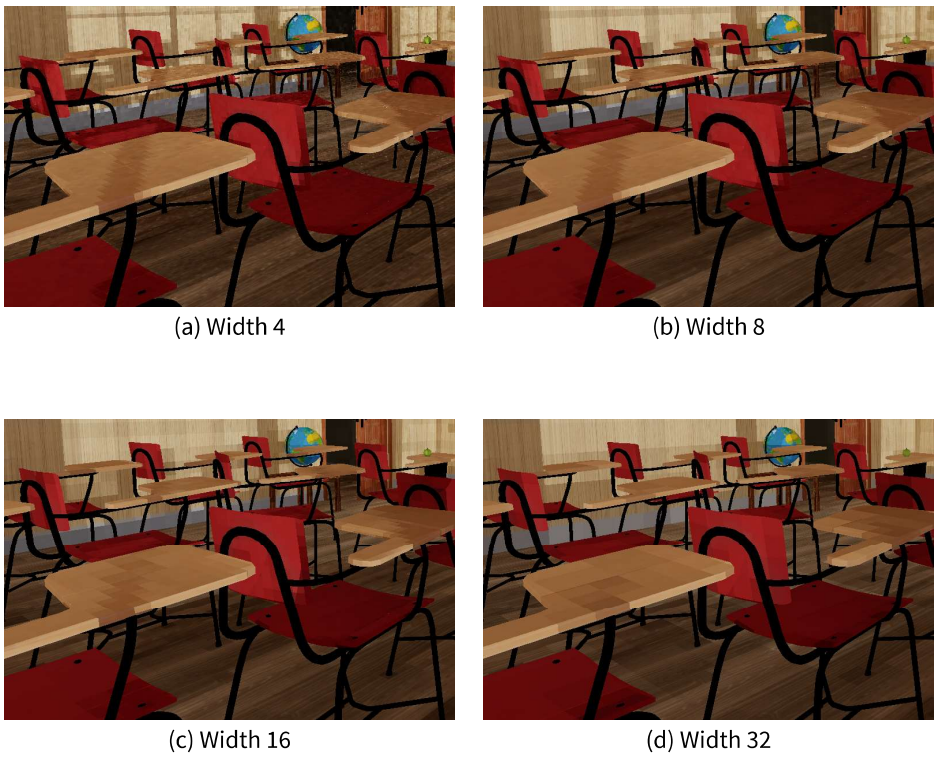


(c) Position unjittered with 2 normal bits, normal jittering of 0.2 (d) Position jittered with 0.6, with 2 normal bits, normal jittering of 0.2

Figure 14: nHPSF results with 2 normal bits being used in the hash descriptors with 2 frames of information.

levels when there is more than 1 frame of information available for nHPSF. If this is compared to the raw input, it is noticable that there is some form of information potentially being overly averaged and transported around, dramatically changing the look of the scene. The shadow on the door behind the moped is missing and so is the shadow of the fence in front of the moped. With that said, after 16 spp nHPSF looks virtually noise-free and only the discretization artifacts are still visible.

5.4 A-nHPSF



(a) Width 4 (b) Width 8

(c) Width 16 (d) Width 32

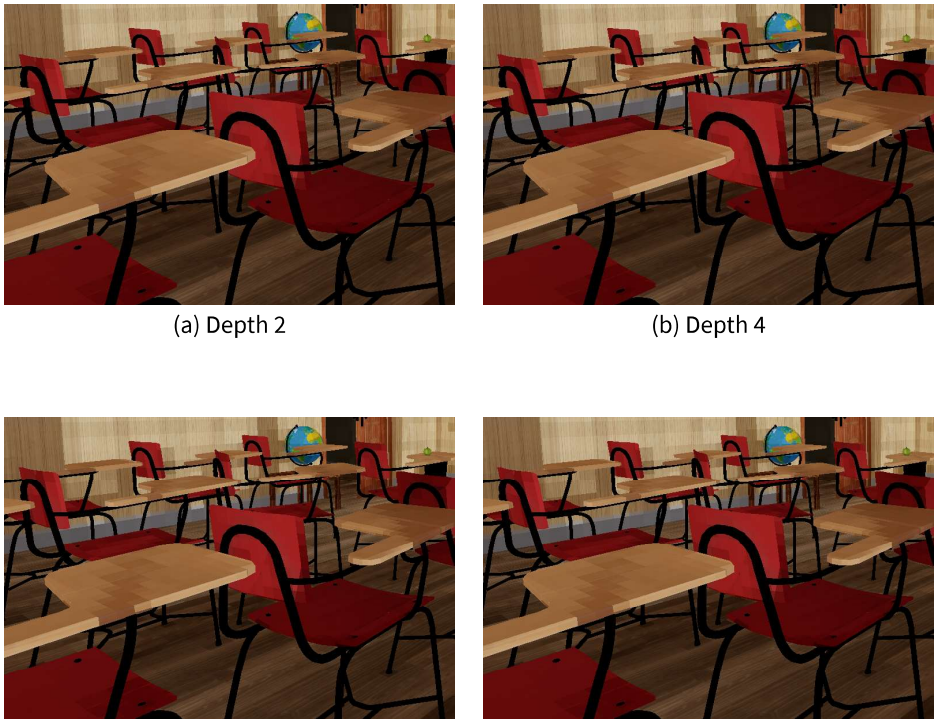
Figure 15: A-nHPSF results over 32 frames, with a maximum path depth of 4 and different filter widths.

The results for A-nHPSF can be seen in Figure 15, 16, and 17.

Due to the nature of the technique, all of the results shown are from 32 frames of information, since the technique is meant to amortize the cost of tracing across multiple frames. This technique, like all the other filter techniques shown thus far, suffers from the same blocky artifacts and loss of strength of shadows due to the radiance averaging that is happening in the cells. As the consequence of the longer path lengths of this technique the classroom scene seems brightened enough that the floor and wall behind the door are visible (see 16).

When comparing A-nHPSF to the other HPSF methods, it shows that it suffers from the same discretization artifacts. Unlike nHPSF, A-nHPSF still retains most of the look that can be seen in the raw results. However, when A-nHPSF is position jittered, the shadows disappear as is visible in Figure 17.

Note that A-nHPSF has a deeper path depth when compared to these other results, as it amortizes the continuation of deeper paths over frames.



(a) Depth 2 (b) Depth 4

(c) Depth 8 (d) Depth 16

Figure 16: A-nHPSF results with filter width 16 over 32 frames, with varying maximum path depths.

(a) Width 4 (b) Width 8

(c) Width 16 (d) Width 32

Figure 17: A-nHPSF results over 32 frames, with a maximum path depth of 4 and different filter widths and position jittering of 0.6.

(a) raw 1 spp input (b) raw 16 spp input (c) raw 32 spp input

(d) SSPSF 1 spp (e) SSPSF 16 spp (f) SSPSF 32 spp

(g) HPSF Width 16, 1spp (h) HPSF Width 16, 16 spp (i) HPSF Width 16, 32 spp

(j) nHPSF Width 16, 1spp (k) nHPSF Width 16, 16 spp (l) nHPSF Width 16, 32spp

(m) A-nHPSF Depth 4, 32spp

Figure 18: All filters compared on 1, 16, and 32 frames of input with exception of A-nHPSF.

6 Discussion

The only advantage of SSPSF compared to the other filtering methods is that it lacks discretization artifacts given enough samples.

HPSF suffers from discretization issues that jittering does not completely hide. Jittering reintroduces uniform noise which is exactly what we are trying to remove, thus dealing with the discretization artifacts is difficult. These same discretization artifacts cause crisp shadow detail at the shadow boundaries to be lost. Shadows also lose their deep darkness due to the averaging of radiance in the hash cells. Nevertheless, the resulting noise is much less than the noise in the input data.

When normal information is used in the HPSF and nHPSF methods, shadows keep more of their detail. This is because areas in shadow in general have a significantly different quantized normal compared to areas that are not in shadow.

The HPSF method with 1 spp is not a noise-free image, but the method is very powerful in how it deals with temporal accumulation. Since this happens in the hash cells, it is not reliant on reprojecting samples in screen-space across frames. The temporal accumulation only suffers from disocculusion and occulusion artifacts if samples do not reach the same cells that were used in the previous frames.

However, the same temporal accumulation introduces another problem when dealing with dynamic content. How does one detect when temporal information is no longer valid? If information in the hash cells is stored indefinetly, then new information that needs to be stored will have no free hash cells available to do so, thus a form of eviction is required. The simple eviction strategy that is implemented in HPSF is too naive. If a cell is not written to in the current frame, it is evicted. This is too aggressive since the information could still be valid in the next frame. However, with animated scene content like moving lights this will result in lagging shadows and other artifacts.

For both nHPSF and A-nHPSF it is possible to tweak how aggressively the subsequent bounce information is filtered, compared to the primary filter hit. This was not experimented with during this thesis project.

Because the scattering of longer paths is amortized over multiple frames, the method is extremely sensitive to the eviction strategy. To transport the radiance that is found during the amortized scattering process, the radiance needs to remain available for enough frames such that other samples can find and use it.

Due to the nature of A-nHPSF, it is necessary that some form of temporal adaptivity is built-in, which is currently lacking. The naive temporal accumulation that is currently built into this method will be challenged by moving lights.

7 Conclusion & Future Work

The goal of this thesis was to achieve real-time path traced global illumination in games using path space filtering techniques. As was mentioned in section 5, we could not give performance results. Nonetheless, we argue that this class of filters can serve in real-time due to the minimal overhead of reading and writing to the hashmaps on the GPU.

The filters that use HPSF that were introduced in this thesis still suffer from many artifacts and have many problems that need to be overcome. We argue that there is great potential for these filters to be extended in ways to overcome the various artifacts, with the exception of SSPSF.

SSPSF provides a tool to see if the separation of terms that is neccessary for the other PSF methods was performed correctly.

We believe that temporal adaptivity, eviction policies and extensions to deal with glossy surfaces are the most important subjects for future work. Further investigation into adding more information into the hash descriptor can also be interesting. For instance, radiance could be averaged on multiple material layer levels by adding information of different material layers into the hash descriptor. In this thesis it was not explored how to deal with glossy surfaces with the HPSF methods, adding material layer information might be a method on how to filter for glossy surfaces. Further inspiration on how to deal with glossy surfaces could be found in "Beyond Spatiotemporal Variance-Guided Filtering: Temporally Stable Filtering of Path-Traced Reflections in Real-Time" by Voorhuis [79], "Dynamic Diffuse Global Illumination with Ray-Traced Irradiance Fields" by Majercik *et al.* [55] and "Real-Time Path Tracing and Denoising in Quake 2 (Presented by NVIDIA)" by Schied *et al.* [4].

An interesting way of dealing with temporal adaptivity would be to store the parameterized rays from a previous frame to be re-traced in the current frame. This would allow us to check if the result of the re-cast ray is within given error bounds. If this is done sparsely at each hash cell, we can construct a measure of reliability and use it to inform a temporal blending factor. The temporal blending factor would then be used in the temporal accumulation stage of the pipeline to reliably drop invalid radiance information from hash cells.

One artifact that can be noticed for all HPSF methods that were shown in this thesis, is that there is a boundary between the switching of the different levels of detail, the cells at that boundary overlap. The idea behind the automatic level of detail switching that is employed by HPSF, is to approximate when radiance can be filtered more aggressively. More research into methods to deal with this automatic level of detail switching is required.

All the HPSF methods could also look into iterating the filtering process. This would work by jittering and storing the sample an iterated amount of times. The expected result would be that the radiance that is filtered iteratively will be overly blurred, but it would be computationally cheap to perform. This could be interesting to do when for the first thread that stores to an empty cell. Such an extension would add to the jittering across multiple levels of detail.

A-nHPSF is the only method that can retrieve indirect illumination from long paths in real-time since the tracing of rays is amortized of multiple frames. This makes it the method with the most potential. The method can be extended to cast multiple types of rays which can also be amortized over frames. The extensions that were discussed for the HPSF methods also apply to this method.

A unique extension to A-nHPSF would be the ability to transfer between multiple cells at the same time after continuing a path. This could be performed by keeping track of the hash keys and attenuations of all the path vertices along the full continued path. This would mean storing 20 bytes (2 unsigned integers and 3 floats) for every path vertex along the path. On every scatter, we then have the information required to transfer radiance between each hash cell that each path vertex allows us to access. It is to be expected that accessing the hashmap many times will be more computationally expensive, but at the same time we would transfer radiance from deeper paths more quickly.

Another extension to A-nHPSF would be to learn the distribution of which hash cells are visible from which hash cell. This extension could be considered an extension of the uniformed scattering

from hash cells extension. Hopefully, it would cause important cells to transport radiance more often. This is related to the work of "Learning Light Transport the Reinforced Way" by Dahm *et al.* [39].

References

[1] N. Binder, S. Fricke, and A. Keller, "Fast path space filtering by jittered spatial hashing," in *ACM SIGGRAPH 2018 Talks*, ser. SIGGRAPH '18, Vancouver, British Columbia, Canada: ACM, 2018, 71:1–71:2, ISBN: 978-1-4503-5820-0. DOI: 10.1145/3214745.3214806. [Online]. Available: http://doi.acm.org/10.1145/3214745.3214806.

[2] A. Keller, K. Dahm, and N. Binder, "Path space filtering for integro-approximation problems," *Proc. of MCQMC*, 2014. DOI: https://doi.org/10.1007/978-3-319-33507-0_21.

[3] Nintendo, *Nintendo Switch Technical Specs*, Mar. 2017. [Online]. Available: https://www.nintendo.com/switch/features/tech-specs/ (visited on 12/18/2018).

[4] C. Schied and A. Panteleev, "Real-Time Path Tracing and Denoising in Quake 2 (Presented by NVIDIA)," in *GDC 2019*, San Francisco, California: GDC Vault, 2019. [Online]. Available: https://schedule.gdconf.com/session/real-time-path-tracing-and-denoising-in-quake-2-presented-by-nvidia/865647 (visited on 08/12/2019).

[5] NVIDIA Lightspeed Studios, *Quake II RTX*, Jun. 2019. [Online]. Available: https://store.steampowered.com/app/1089130/Quake_II_RTX (visited on 07/11/2019).

[6] Y. Tokuyoshi, T. Sekine, and S. Ogaki, "Fast global illumination baking via ray-bundles," in *SIGGRAPH Asia 2011 Sketches*, ser. SA '11, Hong Kong, China: ACM, 2011, 25:1–25:2, ISBN: 978-1-4503-1138-0. DOI: 10.1145/2077378.2077410. [Online]. Available: http://doi.acm.org/10.1145/2077378.2077410.

[7] T. Akenine-Möller, E. Haines, and N. Hoffman, *Real-time rendering*. AK Peters/CRC Press, 2018.

[8] T. Stachowiak and Y. Uludag, "Stochastic screen-space reflections," *ACM SIGGRAPH Courses 2015: Advances in Real-Time Rendering in Games*, 2015.

[9] C. Brennan, "Accurate environment mapped reflections and refractions by adjusting for object distance," *ATI Research*, 2002.

[10] R. Dimitrov, "Cascaded shadow maps," *Developer Documentation, NVIDIA Corp*, 2007.

[11] H. Kolivand, M. S. Sunar, A. Altameem, A. Rehman, and M. Uddin, "Shadow mapping algorithms: Applications and limitations," *Applied Mathematics & Information Sciences*, vol. 9, no. 3, p. 1307, 2015.

[12] M. Gjoel and M. Svendsen, "Low Complexity, High Fidelity - INSIDE Rendering," in *GDC 2016*, San Francisco, California: GDC Vault, 2016. [Online]. Available: https://www.gdcvault.com/play/1023002/Low-Complexity-High-Fidelity-INSIDE.

[13] T. Whitted, "An improved illumination model for shaded display," *Commun. ACM*, vol. 23, no. 6, pp. 343–349, Jun. 1980, ISSN: 0001-0782. DOI: 10.1145/358876.358882. [Online]. Available: http://doi.acm.org/10.1145/358876.358882.

[14] J. Kajiya, "The rendering equation," *SIGGRAPH Comput. Graph.*, vol. 20, no. 4, pp. 143–150, Aug. 1986, ISSN: 0097-8930. DOI: 10.1145/15886.15902. [Online]. Available: http://doi.acm.org/10.1145/15886.15902.

[15] F. Bartell, E. Dereniak, and W. Wolfe, "The theory and measurement of bidirectional reflectance distribution function (brdf) and bidirectional transmittance distribution function (btdf)," in *Radiation scattering in optical systems*, International Society for Optics and Photonics, vol. 257, 1981, pp. 154–160.

[16] E. Veach and L. Guibas, "Metropolis light transport," in *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '97, New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 1997, pp. 65–76, ISBN: 0-89791-896-7. DOI: 10.1145/258734.258775. [Online]. Available: https://doi.org/10.1145/258734.258775.

[17] T. Hachisuka, J. Pantaleoni, and H. W. Jensen, "A path space extension for robust light transport simulation," *ACM Trans. Graph.*, vol. 31, no. 6, 191:1–191:10, Nov. 2012, ISSN: 0730-0301. DOI: 10.1145/2366145.2366210. [Online]. Available: http://doi.acm.org/10.1145/2366145.2366210.

[18] M. Peercy, “Linear color representations for full speed spectral rendering,” in *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, ACM, 1993, pp. 191–198.
[19] L. Belcour and P. Barla, “A practical extension to microfacet theory for the modeling of varying iridescence,” *ACM Transactions on Graphics (TOG)*, vol. 36, no. 4, p. 65, 2017.
[20] L. Fascione, J. Hanika, M. Leone, M. Droske, J. Schwarzhaupt, T. Davidovič, A. Weidlich, and J. Meng, “Manuka: A batch-shading architecture for spectral path tracing in movie production,” *ACM Trans. Graph.*, vol. 37, no. 3, 31:1–31:18, Aug. 2018, ISSN: 0730-0301. DOI: 10.1145/3182161. [Online]. Available: http://doi.acm.org/10.1145/3182161.
[21] E. Lafortune and Y. Willems, “Rendering participating media with bidirectional path tracing,” in *Proceedings of the Eurographics Workshop on Rendering Techniques ’96*, Porto, Portugal: Springer-Verlag, 1996, pp. 91–100, ISBN: 3-211-82883-4. [Online]. Available: http://dl.acm.org/citation.cfm?id=275458.275468.
[22] J. Kajiya and B. Von Herzen, “Ray tracing volume densities,” *SIGGRAPH Comput. Graph.*, vol. 18, no. 3, pp. 165–174, Jan. 1984, ISSN: 0097-8930. DOI: 10.1145/964965.808594. [Online]. Available: http://doi.acm.org/10.1145/964965.808594.
[23] A. Smith, J. Skorupski, and J. Davis, “Transient rendering,” 2008.
[24] C. Cao, Z. Ren, C. Schissler, D. Manocha, and K. Zhou, “Interactive sound propagation with bidirectional path tracing,” *ACM Transactions on Graphics (TOG)*, vol. 35, no. 6, p. 180, 2016.
[25] R. L. Cook, T. Porter, and L. Carpenter, “Distributed ray tracing,” *SIGGRAPH Comput. Graph.*, vol. 18, no. 3, pp. 137–145, Jan. 1984, ISSN: 0097-8930. DOI: 10.1145/964965.808590. [Online]. Available: http://doi.acm.org/10.1145/964965.808590.
[26] M. Potmesil and I. Chakravarty, “Modeling motion blur in computer-generated images,” *SIGGRAPH Comput. Graph.*, vol. 17, no. 3, pp. 389–399, Jul. 1983, ISSN: 0097-8930. DOI: 10.1145/964967.801169. [Online]. Available: http://doi.acm.org/10.1145/964967.801169.
[27] M. Pharr, *Adopting lessons from offline ray tracing to real-time raytracing for practical pipelines*, http://advances.realtimerendering.com/s2018/index.htm https://www.jendrikillner.com/post/graphics-programming-weekly-issue-53/, Aug. 2018. [Online]. Available: http://advances.realtimerendering.com/s2018/Pharr%20-%20Advances%20in%20RTR%20-%20Real-time%20Ray%20Tracing.pdf.
[28] G. Fishman, *Monte Carlo: concepts, algorithms, and applications*. Springer Science & Business Media, 2013.
[29] L. Szirmay-Kalos, *Monte Carlo Methods in Global Illumination - Photo-realistic Rendering with Randomization*. Saarbrücken, Germany, Germany: VDM Verlag, 2008, ISBN: 9783836479196.
[30] R. R. Schaller, “Moore’s law: Past, present and future,” *IEEE spectrum*, vol. 34, no. 6, pp. 52–59, 1997.
[31] P. Efraimidis and P. Spirakis, “Weighted random sampling,” in *Encyclopedia of Algorithms*, M.-Y. Kao, Ed. Boston, MA: Springer US, 2008, pp. 1024–1027, ISBN: 978-0-387-30162-4. DOI: 10.1007/978-0-387-30162-4_478. [Online]. Available: https://doi.org/10.1007/978-0-387-30162-4_478.
[32] A. Keller, “A quasi-Monte Carlo algorithm for the global illumination problem in the radiosity setting,” in *Monte Carlo and Quasi-Monte Carlo Methods in Scientific Computing*, Springer, 1995, pp. 239–251.
[33] ——, “Quasi-Monte Carlo methods in computer graphics,” *Zeitschrift fur Angewandte Mathematik und Mechanik*, vol. 76, pp. 109–112, 1996.
[34] ——, “Quasi-Monte Carlo radiosity,” in *Rendering Techniques’ 96*, Springer, 1996, pp. 101–110.
[35] R. Coveyou, C. V. R, and K. Yost, “Adjoint and importance in monte carlo application,” *Nuclear Science and Engineering*, vol. 27, no. 2, pp. 219–234, 1967.
[36] J. Bikker, “Ray tracing for real-time games,” 2012.
[37] E. Veach and L. J. Guibas, “Optimally combining sampling techniques for monte carlo rendering,” in *Proceedings of the 22Nd Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH ’95, New York, NY, USA: ACM, 1995, pp. 419–428, ISBN: 0-89791-701-4. DOI: 10.1145/218380.218498. [Online]. Available: http://doi.acm.org/10.1145/218380.218498.

[38] E. Lafortune and Y. Willems, "Bi-directional path tracing," in *Proceedings of Third International Conference on Computational Graphics and Visualization Techniques (Compugraphics '93)*, Alvor, Portugal, Dec. 1993, pp. 145–153.

[39] K. Dahm and A. Keller, "Learning light transport the reinforced way," in *Monte Carlo and Quasi-Monte Carlo Methods. MCQMC 2016. Proceedings in Mathematics & Statistics*, A. Owen and P. Glynn, Eds., vol. 241, Springer, 2018, pp. 181–195.

[40] K. Dahm and A. Keller, "Learning light transport the reinforced way," in *ACM SIGGRAPH 2017 Talks*, ser. SIGGRAPH '17, Los Angeles, California: ACM, 2017, 73:1–73:2, ISBN: 978-1-4503-5008-2. DOI: 10.1145/3084363.3085032. [Online]. Available: http://doi.acm.org/10.1145/3084363.3085032.

[41] T. Müller, B. McWilliams, F. Rousselle, M. Gross, and J. Novák, "Neural importance sampling," *ACM Transactions on Graphics*, 2019.

[42] P. Clarberg, W. Jarosz, T. Akenine-Möller, and H. W. Jensen, "Wavelet importance sampling: Efficiently evaluating products of complex functions," *ACM Trans. Graph.*, vol. 24, no. 3, pp. 1166–1175, Jul. 2005, ISSN: 0730-0301. DOI: 10.1145/1073204.1073328. [Online]. Available: http://doi.acm.org/10.1145/1073204.1073328.

[43] P. Clarberg and T. Akenine-Möller, "Practical product importance sampling for direct illumination," in *Computer Graphics Forum*, Wiley Online Library, vol. 27, 2008, pp. 681–690.

[44] M. Pharr, W. Jakob, and G. Humphreys, *Physically based rendering: From theory to implementation*. Morgan Kaufmann, 2016.

[45] M. Pharr, "Guest editor's introduction: Special issue on production rendering," *ACM Trans. Graph.*, vol. 37, no. 3, 28:1–28:4, Jul. 2018, ISSN: 0730-0301. DOI: 10.1145/3212511. [Online]. Available: http://doi.acm.org/10.1145/3212511.

[46] E. Catalano, R. Yasui-Schöffel, K. Dahm, N. Binder, and A. Keller, "Gi next: Global illumination for production rendering on gpus," in *ACM SIGGRAPH 2016 Talks*, ser. SIGGRAPH '16, Anaheim, California: ACM, 2016, 69:1–69:2, ISBN: 978-1-4503-4282-7. DOI: 10.1145/2897839.2927452. [Online]. Available: http://doi.acm.org/10.1145/2897839.2927452.

[47] L. Fascione, J. Hanika, R. Piekè, R. Villemin, C. Hery, M. Gamito, L. Emrose, and A. Mazzone, "Path tracing in production," in *ACM SIGGRAPH 2018 Courses*, ser. SIGGRAPH '18, Vancouver, British Columbia, Canada: ACM, 2018, 15:1–15:79, ISBN: 978-1-4503-5809-5. DOI: 10.1145/3214834.3214864. [Online]. Available: http://doi.acm.org/10.1145/3214834.3214864.

[48] I. Georgiev, T. Ize, M. Farnsworth, R. Montoya-Vozmediano, A. King, B. V. Lommel, A. Jimenez, O. Anson, S. Ogaki, E. Johnston, A. Herubel, D. Russell, F. Servant, and M. Fajardo, "Arnold: A brute-force production path tracer," *ACM Trans. Graph.*, vol. 37, no. 3, 32:1–32:12, Aug. 2018, ISSN: 0730-0301. DOI: 10.1145/3182160. [Online]. Available: http://doi.acm.org/10.1145/3182160.

[49] C. Kulla, A. Conty, C. Stein, and L. Gritz, "Sony pictures imageworks arnold," *ACM Trans. Graph.*, vol. 37, no. 3, 29:1–29:18, Aug. 2018, ISSN: 0730-0301. DOI: 10.1145/3180495. [Online]. Available: http://doi.acm.org/10.1145/3180495.

[50] P. Christensen, J. Fong, J. Shade, W. Wooten, B. Schubert, A. Kensler, S. Friedman, C. Kilpatrick, C. Ramshaw, M. Bannister, B. Rayner, J. Brouillat, and M. Liani, "Renderman: An advanced path-tracing architecture for movie rendering," *ACM Trans. Graph.*, vol. 37, no. 3, 30:1–30:21, Aug. 2018, ISSN: 0730-0301. DOI: 10.1145/3182162. [Online]. Available: http://doi.acm.org/10.1145/3182162.

[51] B. Burley, D. Adler, M. J.-Y. Chiang, H. Driskill, R. Habel, P. Kelly, P. Kutz, Y. K. Li, and D. Teece, "The design and evolution of disney's hyperion renderer," *ACM Trans. Graph.*, vol. 37, no. 3, 33:1–33:22, Jul. 2018, ISSN: 0730-0301. DOI: 10.1145/3182159. [Online]. Available: http://doi.acm.org/10.1145/3182159.

[52] A. Keller, C. Wächter, M. Raab, D. Seibert, D. van Antwerpen, J. Korndörfer, and L. Kettner, "The iray light transport simulation and rendering system," *CoRR*, vol. abs/1705.01263, 2017. arXiv: 1705.01263. [Online]. Available: http://arxiv.org/abs/1705.01263.

[53] C. Schied, C. Peters, and C. Dachsbacher, "Gradient estimation for real-time adaptive temporal filtering," *Proceedings of the ACM on Computer Graphics and Interactive Techniques*, vol. 1, no. 2, p. 24, 2018.

[54] C. Barré-Brisebois, *Game ray tracing: State-of-the-art and open problems*, https://www.ea.com/seed/news/hpg-2018-keynote, retrieved on 2018-09-27, 2018.

[55] Z. Majercik, J.-P. Guertin, D. Nowrouzezahrai, and M. McGuire, "Dynamic diffuse global illumination with ray-traced irradiance fields," *Journal of Computer Graphics Techniques (JCGT)*, vol. 8, no. 2, pp. 1–30, Jun. 2019, ISSN: 2331-7418. [Online]. Available: http://jcgt.org/published/0008/02/01/.

[56] J. Bikker and J. van Schijndel, "The brigade renderer: A path tracer for real-time games," *International Journal of Computer Games Technology*, vol. 2013, 2013.

[57] S. Studio, *Enlighten : Real-time global illumination across all platforms*, https://www.siliconstudio.co.jp/middleware/enlighten/en/, retrieved on 2019-07-10, 2019.

[58] S. Martin, "Enlighten real-time radiosity," in *ACM SIGGRAPH 2011 Computer Animation Festival*, ACM, 2011, pp. 97–97.

[59] E. Haines and T. Akenine-Möller, *Ray Tracing Gems*. Apress, Berkeley, CA, 2019, ISBN: 978-1-4842-4427-2. [Online]. Available: https://doi.org/10.1007/978-1-4842-4427-2.

[60] C. Schied, *Q2vkpt*, https://www.brechpunkt.de/q2vkpt, retrieved on 2019-01-24, 2019.

[61] A. Burnes, "Quake II RTX Available Now : Download the Ray Traced Remaster of the classic Quake II for free," *NVIDIA News*, Jun. 6, 2019. [Online]. Available: https://www.nvidia.com/en-us/geforce/news/quake-ii-rtx-ray-traced-remaster-out-now-for-free/ (visited on 07/11/2019).

[62] EA DICE, *Battlefield v*, Nov. 2018.

[63] A. Burnes, "Battlefield V DXR Real-Time Ray Tracing Available Now," *GeForce News*, Nov. 14, 2018. [Online]. Available: https://www.nvidia.com/en-us/geforce/news/battlefield-v-rtx-ray-tracing-out-now/ (visited on 12/18/2018).

[64] P. Games, *MechWarrior 5 : Mercenaries*, Sep. 2019. [Online]. Available: https://mw5mercs.com/ (visited on 07/10/2019).

[65] A. Burnes, "MechWarrior 5: Mercenaries' Mechs Are The Best-Looking Mechs To Date Thanks To NVIDIA RTX Ray Tracing," *GeForce News*, Nov. 20, 2018. [Online]. Available: https://www.nvidia.com/en-us/geforce/news/mechwarrior-5-mercenaries-rtx-ray-tracing-technologies/ (visited on 12/18/2018).

[66] Eidos-Montréal and Crystal Dynamics, *Shadow of the Tomb Raider*, Nov. 2018. [Online]. Available: https://tombraider.square-enix-games.com/en-us (visited on 07/10/2019).

[67] D. James, "Battlefield 5 and Shadow of the Tomb Raider to deliver Nvidia's real-time ray tracing this year," *PCGamesN*, Aug. 21, 2018. [Online]. Available: https://www.nvidia.com/en-us/geforce/news/justice-online-geforce-rtx-ray-tracing-dlss/ (visited on 12/18/2018).

[68] 4A Games, *Metro : Exodus*, Feb. 2019. [Online]. Available: https://www.metrothegame.com/en-gb/ (visited on 07/10/2019).

[69] NVIDIA, "Ray Tracing in 4A's Metro Exodus and Remedy's Control," *NVIDIA Developer News Center*, Jun. 17, 2019. [Online]. Available: https://news.developer.nvidia.com/ray-tracing-in-4as-metro-exodus-and-remedys-control/ (visited on 07/10/2019).

[70] Remedy Entertainment, *Control*, Aug. 2019. [Online]. Available: https://controlgame.com/ (visited on 07/10/2019).

[71] CD Projekt Red, *Cyberpunk 2077*, Apr. 2020. [Online]. Available: https://www.cyberpunk.net/ (visited on 07/10/2019).

[72] A. Burnes, "Cyberpunk 2077: NVIDIA Partnership Brings Ray Tracing To Hugely-Anticipated Game," *GeForce News*, Jun. 11, 2019. [Online]. Available: https://www.nvidia.com/en-us/geforce/news/cyberpunk-2077-nvidia-partnership-ray-tracing/ (visited on 07/10/2019).

[73] Epic Games, *Unreal Engine 4*, Mar. 2014. [Online]. Available: https://www.unrealengine.com/en-US/ (visited on 07/10/2019).

[74] Unity Technologies, *Unity*, Jun. 2005. [Online]. Available: https://www.unity.com/ (visited on 07/10/2019).

[75] NVIDIA, "Real-Time Ray Tracing Has Come to Unreal Engine with the Release of UE4.22," *NVIDIA Developer News Center*, Mar. 2, 2019. [Online]. Available: https://www.nvidia.com/en-us/geforce/news/cyberpunk-2077-nvidia-partnership-ray-tracing/ (visited on 07/10/2019).

[76] ——, "Real-Time Ray Tracing Integrated into Unreal Engine, Unity and Top 1st-Party AAA Game Engines," *NVIDIA News*, Apr. 18, 2019. [Online]. Available: https://nvidianews.

nvidia.com/news/nvidia-microsoft-epic-games-unity-and-leading-developers-kick-start-next-gen-gaming-at-gdc-2019 (visited on 07/10/2019).

[77] M. Koskela, K. Immonen, M. Mäkitalo, A. Foi, T. Viitanen, P. Jääskeläinen, H. Kultala, and J. Takala, "Blockwise multi-order feature regression for real-time path-tracing reconstruction," *ACM Transactions on Graphics (TOG)*, vol. 38, no. 5, p. 138, 2019. [Online]. Available: https://github.com/maZZZu/bmfr.

[78] M. Zwicker, W. Jarosz, J. Lehtinen, B. Moon, R. Ramamoorthi, F. Rousselle, P. Sen, C. Soler, and S.-E. Yoon, "Recent advances in adaptive sampling and reconstruction for monte carlo rendering," in *Computer Graphics Forum*, Wiley Online Library, vol. 34, 2015, pp. 667–681.

[79] V. Voorhuis, "Beyond spatiotemporal variance-guided filtering: Temporally stable filtering of path-traced reflections in real-time," Master's thesis, Utrecht University, 2018.

[80] B. Bitterli, F. Rousselle, B. Moon, J. A. Iglesias-Guitián, D. Adler, K. Mitchell, W. Jarosz, and J. Novák, "Nonlinearly weighted first-order regression for denoising monte carlo renderings," in *Computer Graphics Forum*, Wiley Online Library, vol. 35, 2016, pp. 107–117.

[81] T. Vogels, F. Rousselle, B. Mcwilliams, G. Röthlin, A. Harvill, D. Adler, M. Meyer, and J. Novák, "Denoising with kernel prediction and asymmetric loss functions," *ACM Trans. Graph.*, vol. 37, no. 4, 124:1–124:15, Jul. 2018, ISSN: 0730-0301. DOI: 10.1145/3197517.3201388. [Online]. Available: http://doi.acm.org/10.1145/3197517.3201388.

[82] C. Schied, A. Kaplanyan, C. Wyman, A. Patney, C. R. A. Chaitanya, J. Burgess, S. Liu, C. Dachsbacher, A. Lefohn, and M. Salvi, "Spatiotemporal variance-guided filtering: Real-time reconstruction for path-traced global illumination," in *Proceedings of High Performance Graphics*, ser. HPG '17, Los Angeles, California: ACM, 2017, 2:1–2:12, ISBN: 978-1-4503-5101-0. DOI: 10.1145/3105762.3105770. [Online]. Available: http://doi.acm.org/10.1145/3105762.3105770.

[83] M. Mara, M. McGuire, B. Bitterli, and W. Jarosz, "An efficient denoising algorithm for global illumination," in *Proceedings of High Performance Graphics*, ser. HPG '17, Los Angeles, California: ACM, 2017, 3:1–3:7, ISBN: 978-1-4503-5101-0. DOI: 10.1145/3105762.3105774. [Online]. Available: http://doi.acm.org/10.1145/3105762.3105774.

[84] C. A. Chaitanya, A. Kaplanyan, C. Schied, M. Salvi, A. Lefohn, D. Nowrouzezahrai, and T. Aila, "Interactive reconstruction of monte carlo image sequences using a recurrent denoising autoencoder," *ACM Trans. Graph.*, vol. 36, no. 4, 98:1–98:12, Jul. 2017, ISSN: 0730-0301. DOI: 10.1145/3072959.3073601. [Online]. Available: http://doi.acm.org/10.1145/3072959.3073601.

[85] P. Gautron, M. Droske, C. Wächter, L. Kettner, A. Keller, N. Binder, and K. Dahm, "Path space similarity determined by fourier histogram descriptors.," in *SIGGRAPH Talks*, 2014, pp. 39–1.

[86] P. Moreau and P. Clarberg, "Importance sampling of many lights on the gpu," in *Ray Tracing Gems*, Springer, 2019, pp. 255–283.

[87] N. Benty, K.-H. Yao, T. Foley, C. Lavelle, and C. Wyman, *The Falcor rendering framework*, https://github.com/NVIDIAGameWorks/Falcor, Jul. 2017. [Online]. Available: https://github.com/NVIDIAGameWorks/Falcor.