Karlsruhe Institute of Technology

Adaptive Path Space Filtering

Masterarbeit von

Alexander Schipek

An der Fakultät für Informatik
Institut für Visualisierung und Datenanalyse,
Lehrstuhl für Computergrafik

October 13, 2022

Erstgutachter:	Prof. Dr.-Ing. Carsten Dachsbacher
Zweitgutachter:	Prof. Dr. Hartmut Prautzsch
Betreuender Mitarbeiter:	M.Sc. Addis Dittebrandt

Contents

Abstract

English

As modern graphics hardware improves in the field of ray tracing acceleration, it becomes increasingly apparent that it will be used more frequently in real-time applications. Path tracing thus marks the current direction of research regarding modern real-time rendering. Even with modern hardware it is still not feasible to trace more than one path per pixel. This leads to a severe amount of variance which manifests itself as noise. One way to reduce that noise is by filtering. Filtering spatial regions results in the technique called path space filtering. Filtering over spatial regions results in a biased estimator. The image quality is thus determined by a variance-bias trade-off. This thesis introduces two techniques to control this variance-bias trade-off by using the spatial structure provided by path space filtering to estimate the variance of a spatial region. Based on these variance estimations this paper derives different ideas to both improve the image quality and also improve frame times. The first technique introduces path survival and interpolation between path tracing and path space filtering by analyzing the variance on the primary surface, hence the name adaptive path space filtering for primary surfaces. It achieves to improve the frame times of path space filtering beyond the frame times of path tracing while also generating better image quality in some cases. The second technique analyzes the variance along paths and terminates into a spatial cell once the variance exceeds a given threshold. It is thus called adaptive path graphs. While being computationally heavy, it generates interesting results regarding the variance-bias trade-off and can handle difficult situations, especially if combined with the first technique.

Deutsch

Heutige Grafikkarten, welche über Hardwarebeschleunigungen für Ray Tracing verfügen, sorgen dafür, dass Ray Tracing immer beliebter wird. Es ist demnach offensichtlich, dass es immer öfter für Echtzeitapplikationen Verwendung findet. Path Tracing gibt dem entsprechend, was Ray Tracing angeht, die Richtung momentaner Forschungen im Themengebiet Echtzeit-Rendering an. Jedoch ist es momentan nicht für Echtzeitprogramme umsetzbar mehr als einen Pfad pro Pixel zu verfolgen. Dies führt zu einer hohen Varianz, welche sich als Rauschen im Bild manifestiert. Eine Möglichkeit, um dieses Rauschen zu verringern, ist es einen Filter anzuwenden. Filtert man über räumliche Regionen, so resultiert das in einer Technik namens Path Space Filtering. Das Problem hierbei ist jedoch, dass das räumliche Filtern zwar Varianz verringert, jedoch zu einem Bias führt. Die Bildqualität ist demnach durch einen Ausgleich zwischen Varianz und Bias definiert. Diese Arbeit führt zwei Techniken ein, um diesen Ausgleich zwischen Varianz und Bias zu steuern. Dabei wird die räumliche Datenstruktur, welche durch Path Space Filtering eingeführt wird, verwendet, um die Varianz in der räumlichen Region zu schätzen. Anhand dieser Schätzung werden in dieser Arbeit dann verschiedene Ideen besprochen, um sowohl die Qualität des Bildes als auch die Laufzeit des Programmes zu verbessern. Die erste dieser Techniken führt Methoden für das Pfadüberleben und der Interpolation zwischen Path Tracing und Path Space Filtering anhand des geschätzten Varianzwertes an primären Oberflächen ein. Daraus folgt dann der Name Adaptives Path Space Filtering für Primäre Oberflächen. Diese Technik erzielt eine Verbesserung der Laufzeit von Path Space Filtering, welche sogar die Laufzeit von Path Tracing überbietet. Des Weiteren erzeugt die Technik dabei noch eine bessere Bildqualität für manche Situationen. Die zweite Technik analysiert die Varianz entlang der Pfade und terminiert diese, sobald diese geschätzte Varianz einen gewissen Schwellwert übersteigt. Sobald die Varianz überschritten wird, verwendet die Technik den Beitrag der räumlichen Zelle für das Endergebnis. Diese Technik wird dem entsprechend Adaptive Path Graphs genannt. Obwohl sie auf der einen Seite rechenaufwändiger ist, erzeugt sie dennoch interessante Ergebnisse im Zusammenhang mit dem Varianz-Bias-Ausgleich und kann vor allem in Kombination mit der ersten Technik mit schwierigen Situationen zurecht kommen.

1. Introduction

Realistic light transport is a keystone to rendering realistic scenes. Path tracing is therefore the go-to method in modern day cinema and similar offline media. In order to implement this method, it is mandatory to have an efficient implementation of ray tracing. That is why modern graphics cards all come with hardware accelerated ray tracing cores. This enables path tracing for interactive applications.

Path tracing generates remarkable results if enough samples can be generated. As interactive applications have a low budget of frame time, it is not feasible to render more than one sample per pixel. The resulting image is therefore noisy by nature.

In order to suppress this noise, there are many techniques for denoising an image. Yet, the problem persists, as they introduce computational overhead and cannot remove the noise entirely. One example would be screen-space denoisers like OptiX. These denoisers work well for less noisy images, but the noisier it gets, the blurrier the denoised image gets. Therefore, it is important to reduce the induced noise in the first place, by using different sampling techniques in order to improve sample quality, or by increasing the number of samples.

One of these techniques is called path space filtering [KDB16]. The main idea is to accumulate the outgoing radiance of spatial regions inside a spatial structure. The result is thereby filtered by taking the average over the accumulated values for a given region.

Unfortunately this comes at the price of an induced bias. This bias manifests itself on many occasions, as high frequency details get filtered away.

This thesis introduces new adaptive techniques in order to improve upon path space filtering. The goal is to reduce the bias introduced by path space filtering by also carrying the variance inside the spatial structure. By using this variance estimate for a given spatial region, it is possible to improve frame times, while also introducing control over the variance-bias trade-off.

The introduced techniques are separated into two techniques for primary surfaces and one so called adaptive path graphs approach which binds the overall variance of a path below a given threshold.

Chapter 2 explains the basis of this thesis starting with the the rendering equation 2.1, followed by Monte Carlo integration 2.2 and path tracing 2.3 and ends with the introduction of path space filtering 2.4. The next chapter 3 introduces related work and provides short insights for these techniques and concepts. That is followed by the main chapter 4, which introduces the techniques of adaptive path space filtering for primary surfaces 4.1 and adaptive path graphs 4.2. Chapter 5 discusses implementation detail and chapter 6 discusses results regarding image quality and frame time analysis. At last, the conclusion is given in chapter 7.

2. Background

First of all, it is important to understand the underlying techniques used to implement path space filtering. This background section deduces path tracing by first introducing the rendering equation. After explaining what a Monte Carlo integral is, it is easy to see how path tracing solves the rendering equation as a Monte Carlo integral. Finally, this chapter introduces the technique in which this thesis is based on: path space filtering.

2.1 Rendering Equation

In order to analyze the problem mathematically, it is necessary to have a mathematical expression for correct light transport. This expression is called the rendering equation and was first introduced by Kajiya [Kaj86]. It describes how the reflected radiance of a given surface point, as seen in figure 2.1, is calculated:

$$L(x, \omega_o) = L_e(x, \omega_o) + \int_\Omega f(\omega_i, x, \omega_o) L_i(x, \omega_i) |\cos\theta_i| \mathrm{d}\omega_i \tag{2.1}$$

where ω_o is the reflected direction, $L_e(x, \omega_o)$ is the emitted light in direction ω_o, Ω is the hemisphere along the normal of the surface, $f(\omega_i, x, \omega_o)$ is the bidirectional reflectance distribution function (short BRDF) and $L_i(x, \omega_i)$ is the incoming radiance from direction ω_i. The term $|\cos\theta_i|$ weakens the contribution of incoming radiance with respect to the solid angle θ_i. It can also be written as $|\cos\theta_i| = |\omega_i \cdot N_x|$.

This equation is hard to solve as it contains itself recursively inside an integral.

A common way to rewrite this integral is to write it in path space. The following equations are taken from the lecture slides of Dr. Johannes Schudeiske [Sch22].

$$I_p = \int_P h_p(X) \cdot f(X) \mathrm{d}X \tag{2.2}$$

where $X = (x_1, x_2, ..., x_k) \in P$ refers to a path with vertices $x_1, x_2, ..., x_k$, $h_p(X)$ is a function that selects paths per pixel and $f(X)$ is the measurement contribution function in product area measure $\mathrm{d}X = \prod_i \mathrm{d}x_i$. $h_p(X)$ is the reconstruction filter.

The measurement contribution function can then be written as:

$$f(X) = L_e G(x_{k-1}, x_k) \left(\prod_{i=2}^{k-1} f_r(x_{i-1}, x_i, x_{i+1}) G(x_{i-1}, x_i) \right) W \tag{2.3}$$

with the geometry term G and the camera responsivity function W.

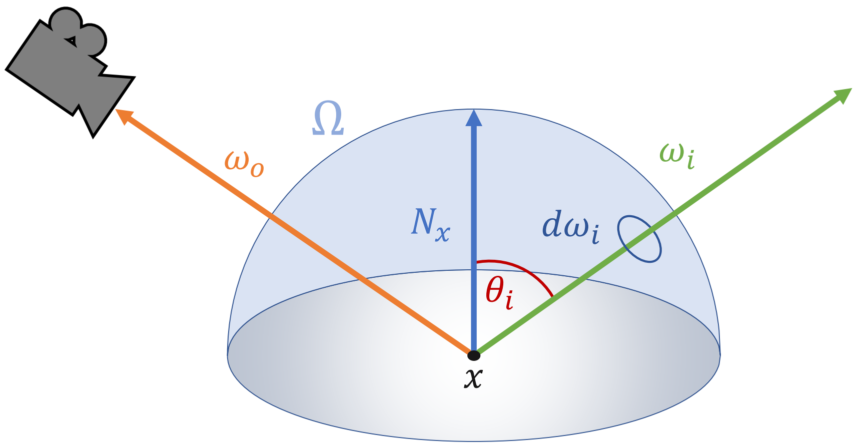


Figure 2.1: Depiction of variables used in the rendering equation 2.1.

2.2 Monte Carlo Integration

Solving the rendering equation requires solving an integral. This equation does not have a closed form solution, hence the need for an integration method.

Monte Carlo integration can solve this problem. Let S be the integration domain, in order to evaluate a Monte Carlo integral given $f : S \to \mathbb{R}$, $x \sim p$, a random variable x with probability density $p(x)$ and $x_1, \ldots, x_N \in S$ random samples for:

$$\mu_f \equiv E[f(x)] = \int_S f(x)p(x)\mathrm{d}x \approx \frac{1}{N}\sum_{i=1}^{N} f(x_i) \equiv \overline{\mu_f} \tag{2.4}$$

So in order to approximate an integral I of a function f, it is possible to write:

$$I = \int_S f(x)\mathrm{d}x \approx \frac{1}{N}\sum_{i=1}^{N} \frac{f(x_i)}{p(x_i)} \tag{2.5}$$

Calculating the standard error is nothing else but calculating the variance of the difference between μ_f and $\overline{\mu_f}$:

$$\begin{aligned} V[\overline{\mu_f} - \mu_f] &= V[\overline{\mu_f}] - V[\mu_f] \\ &= V[\overline{\mu_f}] \\ &= V\left[\frac{1}{N}\sum_{i=1}^{N} f(x_i)\right] \\ &= \frac{1}{N^2}\sum_{i=1}^{N} V[f(x_i)] \\ &= \frac{1}{N^2} N \cdot V[f] = \frac{V[f]}{N} \end{aligned} \tag{2.6}$$

As the variance is nothing but the squared standard error, we get a standard error of

$$\frac{\sigma}{\sqrt{N}} \tag{2.7}$$

for Monte Carlo integration, where σ is the standard error of the function f. This error manifests itself as noise. It can be reduced by either increasing the number of samples or improving the variance of the function. The latter can be achieved by proper sampling strategies.

The main benefit of Monte Carlo integration compared to other integration methods, such as Quadrature, is that the variance of the approximated integral is independent of the dimension of the integral. It therefore performs better for high dimensional integrals.

2.3 Path Tracing

The rendering equation is a high dimensional integral. Path tracing approximates the rendering equation with Monte Carlo integration.

In order to perform Path Tracing, the camera shoots a ray into the scene. Upon hitting a surface, a random direction is chosen in the hemisphere of said surface point and a new ray is shot from there. This is done until either a light source or the environment map is hit. For reasons of performance, the maximum path length is kept at a predefined constant value. This can be seen in figure 2.2. There are also other possibilities to terminate paths, for instance Russian Roulette.

Another optimization is to shoot a second ray at every intersection for primary illumination at that spot. If the scene consists of many light sources, shooting one ray per intersection per light source becomes expensive quickly. Therefore it is better to choose one light source at random and only trace a ray towards that light source. This is called next-event estimation. This yields a trade-off between induced variance from the stochastic process of choosing a light source and the performance benefit of tracing less rays.

A resulting image can be seen in figure 2.3. The same image but with the average over 1000 samples per pixel can be seen in figure 2.4.

In order to reduce the variance, it is better to use importance sampling techniques. Rather than choosing a direction from a uniform distribution at any intersection, one should choose a distribution that resembles the local BRDF interaction. This makes sense as the BRDF interaction is a factor inside the integral of equation 2.1. Sampling proportionally to this factor is thus called BRDF sampling. It ensures that highly reflective surfaces will get more samples towards the reflected direction and will therefore lead to a better result at said surfaces.

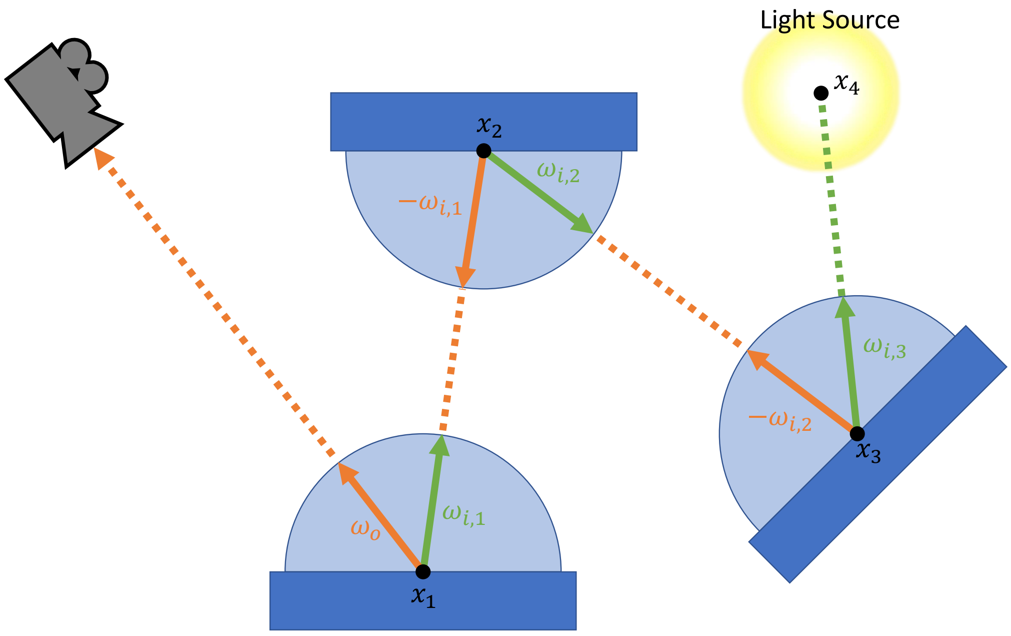


Figure 2.2: Two dimensional illustration of path tracing.

Figure 2.3: Example of an image generated by path tracing.

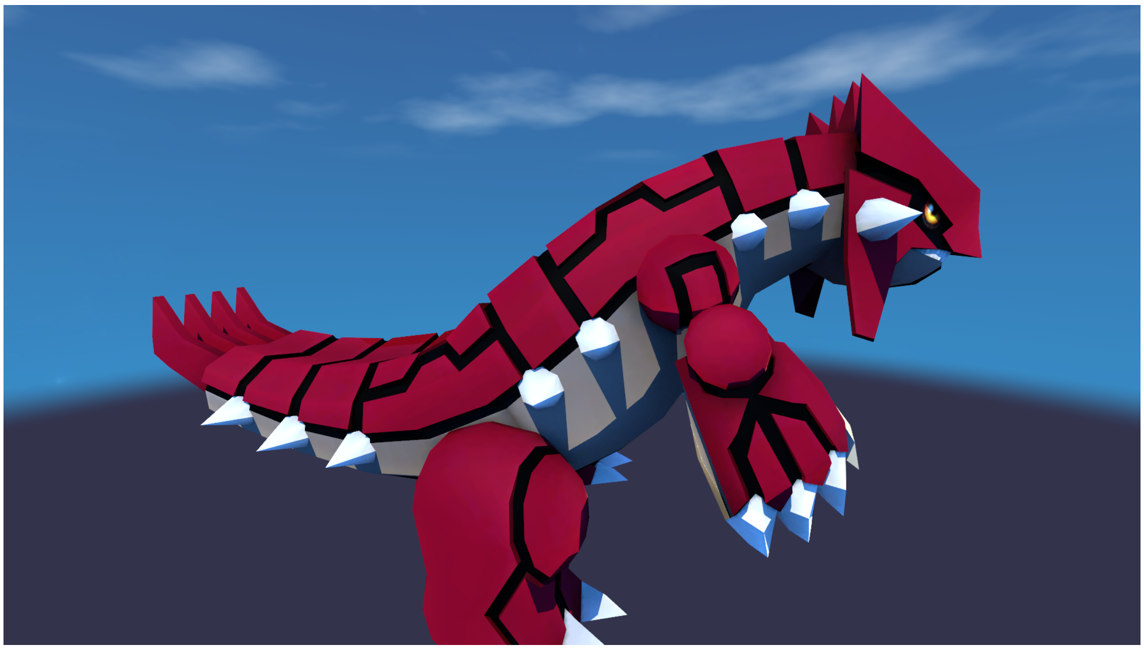


Figure 2.4: Accumulated path traced image after 1000 frames of accumulation.

2.4 Path Space Filtering

As interactive applications do not have a big frame time budget, it is not practicable to use more than one sample per pixel per frame. Using such a low number of samples per pixel, introduces a lot of noise to the image. This can be improved by accumulating spatial regions.

If spatially adjacent surfaces are considered to produce similar results, it is possible to work with regional averages. This certainly does introduce bias, as there could be small highlights for example that get averaged away, but the image will be less noisy.

This is the fundamental idea behind a method first introduced by Keller et al. [KDB16] named path space filtering.

2.4.1 Mathematical Origin

Binder et al. [BFK19] provide a mathematical demonstration for their technique derived from the rendering equation 2.1.

First the rendering equation is split into emitted and reflected radiance.

$$\begin{aligned} L(x,\omega_o) &= L_e(x,\omega_o) + \int_\Omega f(\omega_i,x,\omega_o)L_i(x,\omega_i)|\cos\theta_i|\mathrm{d}\omega_i \\ L(x,\omega_o) &= L_e(x,\omega_o) + L_r(x,\omega_o) \end{aligned} \tag{2.8}$$

Binder et al. [BFK19] rewrite the reflected radiance as follows

$$L_r(x,\omega_o) = \int_{\partial V} V(x,y)f(\omega_i,x,\omega_o)L_i(x,\omega_i)\cos\theta_x\frac{\cos\theta_y}{|x-y|^2}\mathrm{d}y \tag{2.9}$$

This changes the integration domain from solid angle to surface area measure. The scene surfaces ∂V are now the integration domain for next-event estimation and subpath connection. $V(x, y)$ describes the visibility function between the two surface points x and y. The last term of the equation is referred to as the geometric term.

Taking the average over local neighborhoods results in

$$\begin{aligned} L_r(x, \omega_o) = \lim_{r(x)\to 0} \int_{\partial V} \int_{\Omega_y} & \frac{\chi_B(\|x - y\|, r(x))}{\pi r(x)^2} \cdot \\ & \cdot f(\omega_i, y, \omega_o) L_i(y, \omega_i) \cos\theta_y \mathrm{d}\omega_i \mathrm{d}y \end{aligned} \tag{2.10}$$

where $r(x)$ is the radius of the sphere and

$$\chi_B(d, r) := \begin{cases} 1 & d^2 < r^2 \\ 0 & \text{otherwise} \end{cases} \tag{2.11}$$

the characteristic function of the sphere. Dividing by the area of the intersection circle between the surface and the sphere yields a density estimation also referred to as photon mapping.

A similar equation can be formulated for path space filtering. The big difference is that a local average is calculated and normalized by the integral of all weights in the neighborhood. This results in

$$L_r(x, \omega_o) = \lim_{r(x)\to 0} \int_{\Omega} \frac{\int_{\partial V} \chi_B(\|x - x'\|, r(x)) w(x, x') L_i(x', \omega_i) f(\omega_i, x, \omega_o) \cos\theta_{x'} \mathrm{d}x'}{\int_{\partial V} \chi_B(\|x - x'\|, r(x)) w(x, x') \mathrm{d}x'} \mathrm{d}\omega_i \tag{2.12}$$

the finalized equation for path space filtering formulated by Binder et al. [BFK19]. It is immanent that this technique produces bias for $r(x) > 0$.

For a voxel-based implementation, it is necessary to define the characteristic function of a voxel, given by

$$\chi_V(k, k') := \begin{cases} 1 & \lfloor s(k)k \rfloor = \lfloor s(k')k' \rfloor \wedge s(k) = s(k') \\ 0 & \text{otherwise} \end{cases} \tag{2.13}$$

for a resolution selection function $s(k)$ with key k. Using voxels in equation 2.12 yields the following equation

$$L_r(x, \omega_o) \approx \int_{\Omega} \frac{\int_{\partial V} \chi_V(k, k') L_i(x', \omega_i) f(\omega_i, x, \omega_o) \cos\theta_{x'} \mathrm{d}x'}{\int_{\partial V} \chi_V(k, k') \mathrm{d}x'} \mathrm{d}\omega_i \tag{2.14}$$

Assuming $f(\omega_i, x, \omega_o)$ can be written as $f_r(\omega_o, x) \cdot f_i(x, \omega_i)$ and $f_i(x, \omega_i) \approx$ const inside the voxel, it is possible to rearrange equation 2.14 as follows

$$L_r(x, \omega_o) \approx f_r(\omega_o, x) \int_{\Omega} \frac{\int_{\partial V} \chi_V(k, k') L_i(x', \omega_i) f(x', \omega_i) \cos\theta_{x'} \mathrm{d}x'}{\int_{\partial V} \chi_V(k, k') \mathrm{d}x'} \mathrm{d}\omega_i \tag{2.15}$$

It is therefore possible to apply the albedo outside the voxel accumulation. This is very important for high frequency textures, as they would otherwise lose a lot of detail.

2.4.2 Implementation

The method has been optimized by Binder et al. [BFK18, BFK19]. Path space filtering consists of two passes. Firstly, the path tracer that generates samples and accumulates them in a hash map, and secondly, a pass-through shader that reads from said hash map and outputs the final image.

The hash map uses the position as hash input for the key. A second hash function, additionally taking the surface normal and incident direction into account, is used for linear probing of the hash map in order to generate spatial coherence in the data structure. The insertion is managed by atomic floating-point addition.

In order to carry values from earlier frames, a compute shader executes an eviction strategy. It erases a cell if the time stamp is too old and applies an exponential moving average for cells that keep getting filled.

As this approach induces discretization artifacts, Binder et al. [BFK18] provide a jitter approach for reading from the hash map. This approach induces noise again but this time the noise has a smaller variance, because the neighboring cells are rather similar to each other. It is therefore easier to denoise the noise induced by the jittering than the noise generated by the path tracer.

2.4.3 Adaptive Resolution and Jittering

Binder et al. [BFK19] provide a very simple implementation of adaptive resolution by setting a proper $s(k)$ in equation 2.13. $s(k)$ is given by a level of detail function taking the distance between a point and the camera. Taking the second order logarithm of that distance yields the level of detail. The key generation and jittering is then described by algorithm 1. The spatial structure and the provided jittering approach are visualized in figure 2.5. Figure 2.6 and figure 2.7 depict the difference between an image with and without jittering.

Algorithm 1: Computation of the two hashes used for lookup. Note that the arguments of a hash function, which form the key, may be extended to refine clustering. This algorithm is provided by Binder et al. [BFK19]

```
Input: Location $x$ of the vertex, the normal $n$, the position of the camera $p_{\text{cam}}$,
       and the scale $s$.
Output: Hash $i$ to determine the position in the hash table and hash $f$ for
        fingerprinting.
$l \leftarrow$ level_of_detail($\|p_{\text{cam}} - x\|$)
$x' \leftarrow x +$ jitter($n$) $\cdot s \cdot 2^l$
$l' \leftarrow$ level_of_detail($\|p_{\text{cam}} - x'\|$)
$\tilde{x} \leftarrow \lfloor \frac{x'}{s \cdot 2^{l'}} \rfloor$
$i \leftarrow$ hash($\tilde{x}, \dots$)
$f \leftarrow$ hash2($\tilde{x}, n, \dots$)
```

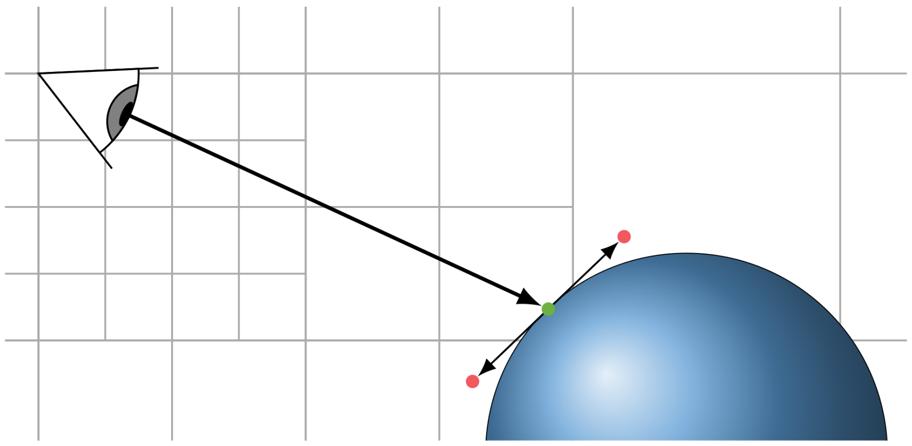


Figure 2.5: Spatial depiction of the hash map structure provided by Binder et al. [BFK19], including jittering.

Figure 2.6: Example image of path space filtering.

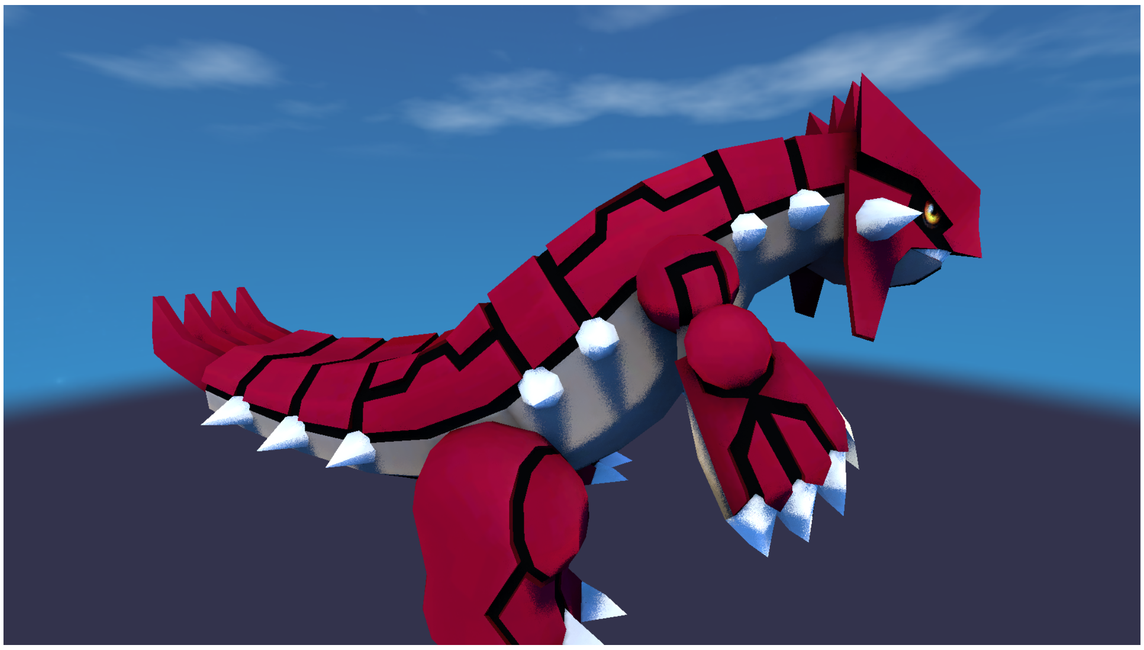


Figure 2.7: Example image of path space filtering with jittering.

3. Related Work

This chapter introduces a few methods that are related to this thesis. These methods are briefly explained and discussed here.

3.1 An Error Metric for Monte Carlo Ray Tracing

Path tracing is not a new technique. There are many papers that analyze path tracing in a mathematical manner. One of these papers published in 1997 has been written by Bolin and Meyer [BM97].

An error metric regarding path tracing is introduced in this paper in order to improve the decision boundaries for Russian Roulette and splitting.

This error metric is deduced by evaluating the Monte Carlo integral of a given path tracer:

$$E[f] \approx \overline{f} = \frac{1}{N}\sum_{i=1}^{N} f(x_0^{(i)}, x_1^{(i)}, x_2^{(i)}, ...) \tag{3.1}$$

where $f(x_0^{(i)}, x_1^{(i)}, x_2^{(i)}, ...)$ is a function calculating the radiance of a given path with path vertices $(x_0^{(i)}, x_1^{(i)}, x_2^{(i)}, ...)$. The variance can be written as:

$$\begin{aligned} V[f] &= E[(f - E[f])^2] \\ &= E[f^2 - 2fE[f] + E[f]^2] \\ &= E[f^2] - 2E[f]E[f] + E[f]^2 \\ &= E[f^2] - E[f]^2 \end{aligned} \tag{3.2}$$

Calculating the variance of the path tracer thus results in:

$$V\left[\overline{f}\right] = \frac{V[f]}{N} \tag{3.3}$$

as calculated before in equation 2.6. The next step is to express the overall variance of the path tracer with respect to each level of the path tree. This leads to the following formula:

$$V[f] = V\left[E\left[f|x_0\right]\right] + E\left[V\left[f|x_0\right]\right] \tag{3.4}$$

applying this formula recursively yields:

$$V[f] = VE\left[f|x_0\right] + EV\left[E\left[f|x_0x_1\right]|x_0\right] + EV\left[E\left[f|x_0x_1x_2\right]|x_0x_1\right] + \dots \tag{3.5}$$

or rather

$$V[f] = VE\left[f|x_0\right] + \sum_j EV\left[E\left[f|x_0...x_j\right]|x_0...x_{j-1}\right] \tag{3.6}$$

This equation states how the variance, generated by the j-th layer in the path tree, affects the overall variance of the path tracer. It implies that the variance of the path tracer is nothing but the sum over the variance generated by each level in the path tree. The first term $VE\left[f|x_0\right]$ describes the aliasing of a pixel. Bolin and Meyer [BM97] use this equation in order to calculate an optimal splitting and Russian Roulette scheme.

3.2 Variance Aware Path Guiding

Variance aware path guiding is a technique developed by Rath et al. [RGH+20] in order to estimate the target densities of guiding methods in order to achieve optimal performance. Rath et al. [RGH+20] describe a generic procedure to calculate these optimal target densities for local path guiding and apply it in two applications: unidirectional path tracing and light source selection for many lights.

The first step of deducing this target density estimation is based on the assumption that only a single decision along the path can be guided. The chosen quantity is the irradiance $E(x)$ of the surface point x. The typical estimator for $E(x)$ can be written as:

$$\langle E(x)\rangle = \frac{\langle L_i(\omega_i, x)\rangle|\cos\theta_i|}{p(\omega_i|x)} \tag{3.7}$$

the target function $p(\omega_i|x)$ is usually set to:

$$p(\omega_i|x) \propto L_i(\omega_i, x)|\cos\theta_i| = E[\langle L_i(\omega_i, x)\rangle]|\cos\theta_i| \tag{3.8}$$

the problem with this target function is that it neglects the variance generated by $\langle L_i\rangle$. A better target density can be found if the variance of the irradiance estimator is taken into consideration. This can be done by using the second moment, which increases if either the mean or the variance gets bigger. Rath et al. [RGH+20] thus deduce the target density to be:

$$p_E(\omega_i|x) \propto \sqrt{E[\langle L_i(\omega_i, x)\rangle^2]}|\cos\theta_i| \tag{3.9}$$

The same deduction can be used to generate a target density for marginalized product sampling. The goal for this step is to guide an estimator for the reflected radiance:

$$\langle L_o(x, \omega_o)\rangle = \frac{f(\omega_i, x, \omega_o)\langle L_i(\omega_i, x)\rangle|\cos\theta_i|}{p(\omega_i|x)} \tag{3.10}$$

by using the same steps as before Rath et al. [RGH+20] deduce the target density to be:

$$p_{L_o}(\omega_i|x) \propto \sqrt{E[f(\omega_i, x, \omega_o)^2\langle L_i(\omega_i, x)\rangle^2]}|\cos\theta_i| \tag{3.11}$$

In order to implement these guiding distributions, Rath et al. [RGH$^+$20] propose a spatial cache approach. These spatial cache cells are thus trained to represent the target density of a region by averaging over all the points inside that region.

Rath et al. [RGH$^+$20] also deduce a target density in order to minimize the image error. This is done by minimizing the error of every pixel. In order to do so, it is necessary to consider the contribution of a path to a pixel, which is the product of the sensor response and the path to the pixel.

Minimizing the pixel error thus consists of including the path contribution of a pixel in the estimator. As learning one density per pixel is unfeasible, Rath et al. [RGH$^+$20] propose a target density for minimizing the mean squared error instead. This is done by minimizing the mean variance over all pixels of the image.

The problem of minimizing the mean squared error is that darker pixels are neglected in favor of brighter pixels. This is also a problem of the adaptive techniques discussed later in this paper.

In order to solve this, Rath et al. [RGH$^+$20] minimize the relative mean squared error instead by dividing the mean squared error by the squared ground truth value of the pixel. This value becomes thus independent of the pixel luminance. The ground truth is obviously unknown, but Rath et al. [RGH$^+$20] approximate it by denoising or filtering the previous frame.

3.3 Path Graphs: Iterative Path Space Filtering

Deng et al. [DHC$^+$21] introduce a technique that improves path space filtering by constructing a graph. This technique is thus called path graphs. This graph operates in world space and consists of light and surface points. These points serve as the vertices of the graph. As for the edges, there are three distinct edges. There are neighbor edges, light edges and continuation edges. An illustration of the graph can be seen in figure 3.1.

This graph is constructed during the path tracing phase of the pipeline. In order to generate neighbor edges, the algorithm connects each shading point to approximately $K-1$ nearby shading points. In order to do that, the shading points are distributed upon K clusters. Points inside a cluster are then interconnected with neighbor edges.

Hereafter, an aggregation and propagation scheme is performed on the graph. This scheme refines the graph in an iterative manner. The aggregation operator improves the radiance estimates of the vertices by combining them over the neighborhoods generated in the graph. Direct and indirect light aggregation are handled separately. The propagation operator updates incoming indirect radiance estimates by copying the outgoing radiance from vertices connected via continuation edges. These two operators are combined in order to refine the graph iteratively.

At last, the graph is gathered and the final image is generated. This last step is thus called the final gather. This step is needed as this method generates strong correlations between nearby points. In order to fix that, the cluster size K is set to 1 in order to compute the final pixel values. Doing so gets rid of these correlations. Another benefit of this step is that it can be used as input for a denoiser.

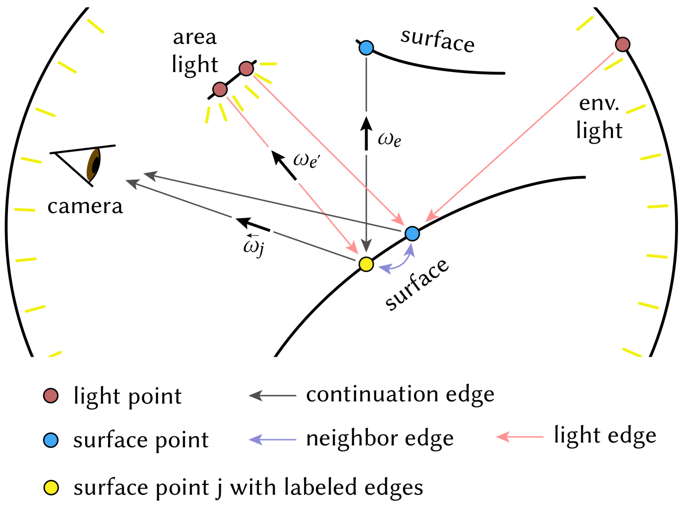


Figure 3.1: Illustration of a path graph by Deng et al. [DHC+21]

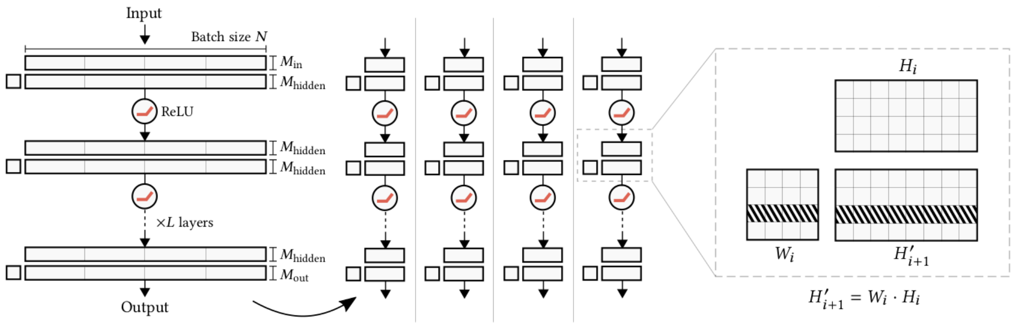
3.4 Real-Time Neural Radiance Caching for Path Tracing

Müller et al. [MRNK21] present a radiance caching method for real-time applications using a neural network. The neural network is not pretrained but rather adapts while rendering. This leads to so called generalization via adaptation. It maps spatio-directional coordinates to radiance values.

The algorithm provided by Müller et al. [MRNK21] works by rendering one short path per pixel per frame. These paths are terminated once the radiance cache provides a sufficiently accurate value. Next-event estimation is performed for each vertex of the path. The last vertex contribution is calculated using the radiance approximation provided by the neural radiance cache.

In order to train the cache, a fraction of these short paths are extended by a few vertices. These vertices are the so called training suffix. Most of the time this suffix consists of only one vertex. Müller et al. [MRNK21] use the radiance estimate collected by all the vertices along these longer training paths as reference values. These reference values are thus used to train the neural radiance cache.

Müller et al. [MRNK21] also provide an efficient implementation of the neural network on the graphics card. The main idea behind that implementation can be seen in figure 3.2.



(a) Batched neural network evaluation **(b)** Distribution of a batch over thread blocks **(c)** Per-thread-block matrix multiplication

Figure 3.2: Structure of the neural radiance cache by Müller et al. [MRNK21]

3.5 Adjoint-Driven Russian Roulette and Splitting in Light Transport Simulation

Russian Roulette and splitting are two techniques to improve the efficiency of a Monte Carlo Renderer. While Russian Roulette improves the render time by terminating paths that have small contributions, splitting improves the quality by dividing paths. Vorba and Křivánek [VK16] aim to improve these techniques by terminating or splitting paths depending on the expected image contribution.

The decision whether to Russian Roulette or split is handled by a single real value $q > 0$. If $q < 1$ the path will be terminated with a probability of $1 - q$. If $q > 1$ the path is split into q new paths. The weight ν_i of a particle thus needs to be rewritten in order to compensate for the decision:

$$\hat{\nu}(y, \omega_i) = \frac{\nu_i(y, \omega_i)}{q(y, \omega_i)} \tag{3.12}$$

A depiction of this concept can be seen in figure 3.3. Vorba and Křivánek [VK16] state that the factor q should be chosen to be proportional to the total expected contribution $E[c(y, \omega_i)]$ divided by the computed measurement I. I could be for instance a pixel value. This leads to the main equation of the paper:

$$q(y, \omega_i) = \frac{E[c(y, \omega_i)]}{I} = \frac{\nu_i(y, \omega_i)\Psi_o^r(y, \omega_i)}{I} \tag{3.13}$$

where the adjoint Ψ stands for the visual importance for a path traced from a light source. Adjoint-driven Russian Roulette and splitting is designed such that the invariant:

$$\hat{\nu}(y, \omega_i) = \frac{I}{\Psi_o^r(y, \omega_i)} \tag{3.14}$$

is maintained. This is done by using weight windows such that the particle weight is always in the weight window center. Vorba and Křivánek [VK16] state that these weights thus oscillate around the ideal value $I/\Psi_o^r(y, \omega_i)$.

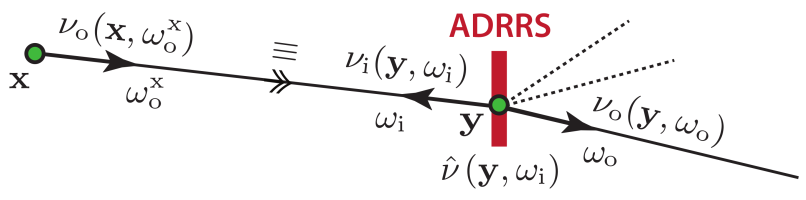

$\nu_o(\mathbf{x}, \omega_o^x)$
$\mathbf{x}$
ω_o^x
$\equiv$
ADRRS
$\nu_i(\mathbf{y}, \omega_i)$
ω_i
$\mathbf{y}$
$\hat{\nu}(\mathbf{y}, \omega_i)$
$\nu_o(\mathbf{y}, \omega_o)$
ω_o

Figure 3.3: Illustration of adjoint-driven RR and splitting by Vorba and Křivánek [VK16]

4. Adaptive Path Space Filtering

The main problem of path space filtering is the visible induced bias. While jittering removes some artifacts resulting from this bias, it cannot fix all of them. One example for that would be high frequency detail on specular surfaces or high frequency shadows.

This chapter discusses techniques to control the variance-bias trade-off between path tracing and path space filtering. The first technique does that by interpolating between the path tracing result and the path space filtering result depending on the primary surface 4.1.2.

A technique for variance control along the path is also explained in this chapter. It is called adaptive path graphs 4.2.

Another major concern is performance. As the values inside the spatial structure get accumulated over time, it is not necessary to trace one sample per pixel anymore. If a given first bounce has a properly converged cell inside the spatial structure, it is reasonable to generate less samples for that region in order to improve the performance.

This chapter provides a technique for stochastic path survival depending on the primary surface 4.1.1.

Implementing adaptive solutions for these problems consists of using some kind of performance metric. A good idea would thus be to estimate the variance of a given spatial cell. If the values accumulated inside the cell are roughly the same, the variance would be small. Such cells would not need too many additional samples. In contrast to that a cell with a high variance accumulates strongly deviating values. Such a cell would need a higher number of samples in order to display a proper result.

4.1 Methods Regarding Primary Surfaces

The first methods consist of looking at the primary surfaces only. As path space filtering relies on looking up the spatial structure for the first surface only, it is not a bad idea to make decisions based on the value inside that spatial cell.

This section describes two ways to improve the path tracing step with the information provided by the spatial cell from previous frames. The spatial cell from previous frames is used in order to provide an analogical implementation of the methods, as the path survival cannot read the values written in the current frame.

4.1.1 Variance Aware Path Survival

The current implementation of path space filtering traces one path per pixel. This gets expensive for big resolutions. It is also not really necessary as the displayed value is read from the spatial structure anyways and the spatial cells get accumulated temporarily, thus generating a high number of samples per cell.

Some cells do not benefit that much from getting this many samples. Most of the time these cells are almost completely converged.

Detecting properly converged spatial cells can be achieved by estimating their variance. A cell with a high variance will need more samples than a cell with a small variance. In this case, the variance of the radiance emitted from this cell is evaluated.

Given this variance measure, it is possible to tweak the probability of culling a path for cells that have a small variance. This makes sense, as these cells are probably decently converged already. The better a cell is converged, the less additional samples it will need to generate a proper radiance estimate.

Culling these paths would automatically lead to improved computational performance, without drastically deteriorating the visual fidelity. Nonetheless, it would not be a good idea to remove paths completely. Doing so would starve these cells, thus generating problems for dynamic scenes.

A probabilistic approach is therefore proposed in order to still guarantee that no paths get starved from samples, while also lowering the overall number of unnecessary paths generated per pixel. This is done by utilizing probability functions.

In order to generate the probability of a path not being culled, a so-called path survival function needs to be generated. It takes the variance as input and generates a given probability. Controlling this function can be achieved by manually tweaking so-called variance threshold v_0 and v_1.

This function should generate a higher probability for higher variances. It is therefore easy to assume a linear function as seen in figure 4.1a:

$$p(v) = \text{clamp}\left(0, 1, \frac{v - v_0}{v_1 - v_0}\right) \tag{4.1}$$

As hard borders can result in harsh artifacts, a smooth linear variant is proposed in figure 4.1b:

$$p(v) = \text{smoothstep}\left(0, 1 - \frac{v_1 - v_0}{8}, \frac{v - (v_1 + v_0) \cdot 0.5}{(v_1 - v_0) \cdot 5} + \left(1 - \frac{v_1 - v_0}{8}\right) \cdot 0.5\right) \tag{4.2}$$

with

$$\text{smoothstep}\,(e_0, e_1, x) = \text{clamp}\left(0, 1, \frac{x - e_0}{e_1 - e_0}\right)^2 \cdot \left(3 - 2 \cdot \text{clamp}\left(0, 1, \frac{x - e_0}{e_1 - e_0}\right)\right) \tag{4.3}$$

The smooth linear function has been tailored by manually tweaking the values such that the shape of the function looks smooth.

4.1.2 Variance Aware Spatial Cell Interpolation

Spatial regions with low variance will probably not change their radiance too much. One example would be a diffusely lit wall. It is a good idea to use the accumulated value from the spatial cell in this case. Spatial regions with high variance will also benefit from the accumulation inside the spatial cell, as they induce a big amount of noise, which is hard to get rid of by denoising the image.

Spatial regions in the medium variance range suffer from the bias induced by path space filtering while not contributing too much noise themselves. It would therefore be beneficial to use the unbiased result generated by the path tracer instead.

These low, medium and high variance regions are user defined regions.

Interpolating between path tracer and spatial cell contributions can be achieved by using a linear interpolation. In order to determine the blending factor, a function just like the one for the variance aware path survival 4.1.1 is used. Note that it is not a probability we generate in this case, but a blending factor.

The function should also be controlled by manually setting variance thresholds v_0 and v_1. It thus generates a blending value between zero and one.

The radiance generated by the path tracer is best used for medium range variance. A step function would generate this exact behavior. An example can be seen in figure 4.1c:

$$p(v) = \begin{cases} 0 & v_0 < v < v_1 \\ 1 & \text{otherwise} \end{cases} \tag{4.4}$$

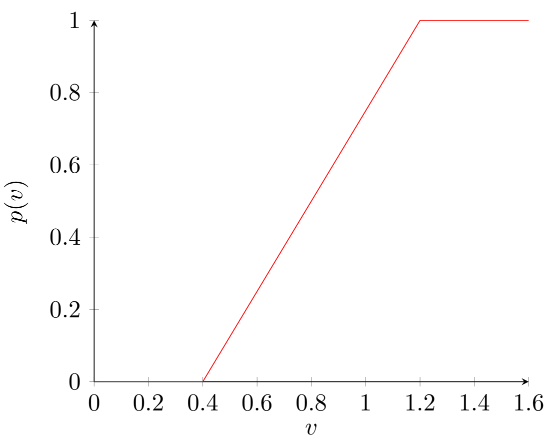
Having hard thresholds would result in visible artifacts. It is therefore a better idea to use something like figure 4.1d. Note that the function is continuous. This is beneficial as it allows to merge the path tracing contribution with the path space filtering contribution, resulting in an overall smoother image.

$$p(v) = \text{clamp}\left(0, 1, 1 - \frac{(1 - \frac{v_1 - v_0}{8})}{\left(\frac{v - (v_1 + v_0)\cdot 0.5}{(v_1 - v_0)\cdot 2}\right)^2 \cdot 16 + 1}\right) \tag{4.5}$$

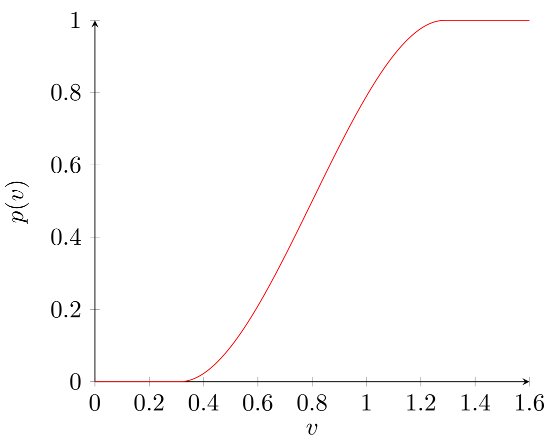
This one over x squared function has also been tailored by manually tweaking the values such that the shape of the function looks decent.

Using these techniques yields the image seen in figure 4.2. Figure 4.3 shows how the different functions change the result of path space filtering.

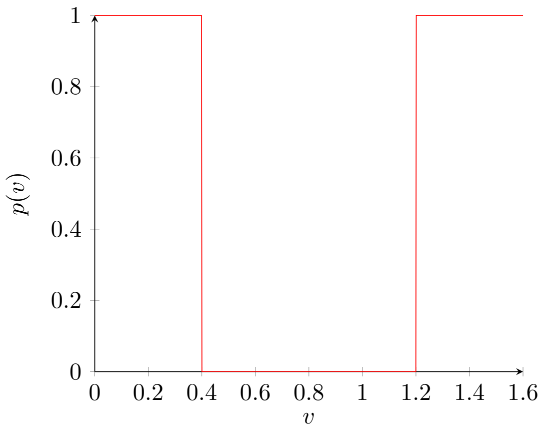
The interaction between variance aware path survival and spatial cell interpolation is illustrated by figure 4.4.



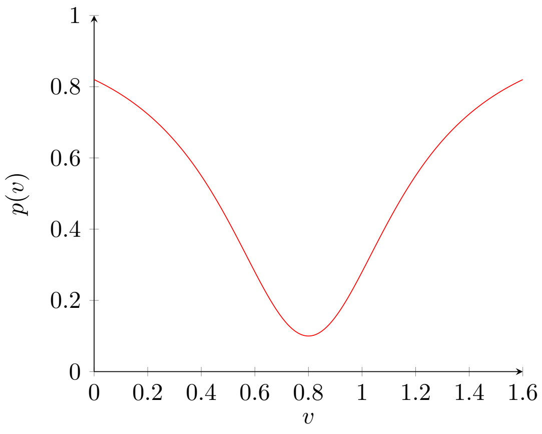
(a) Equation 4.1 for $v_0 = 0.4$ and $v_1 = 1.2$



(b) Equation 4.2 for $v_0 = 0.7$ and $v_1 = 0.9$



(c) Equation 4.4 for $v_0 = 0.4$ and $v_1 = 1.2$



(d) Equation 4.5 for $v_0 = 0.4$ and $v_1 = 1.2$

Figure 4.1

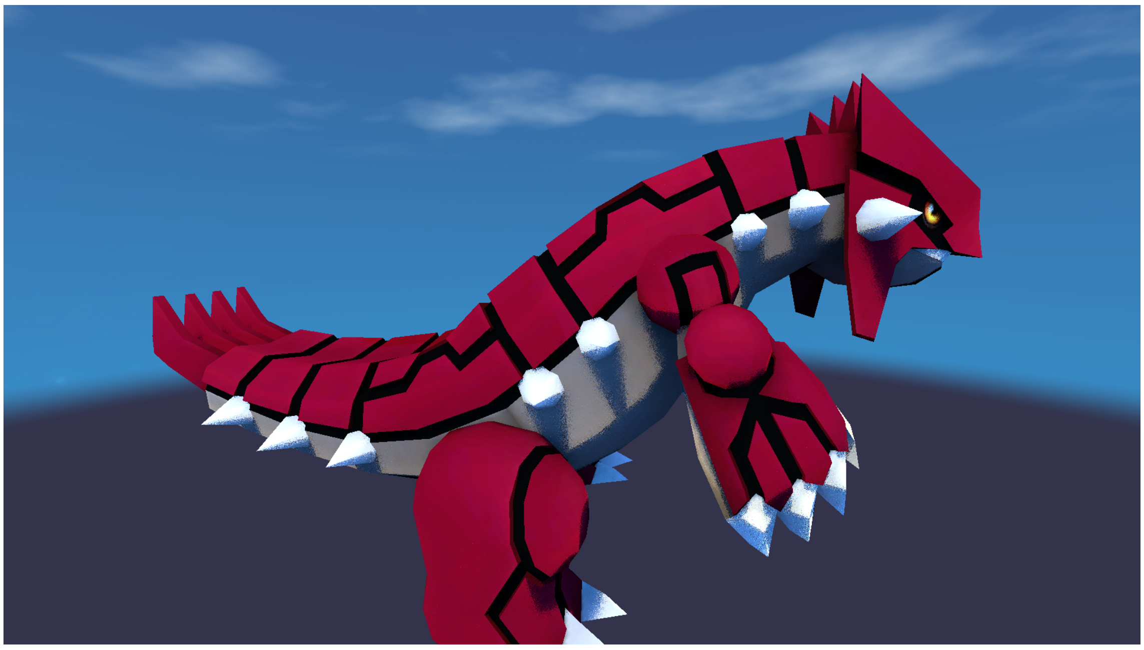
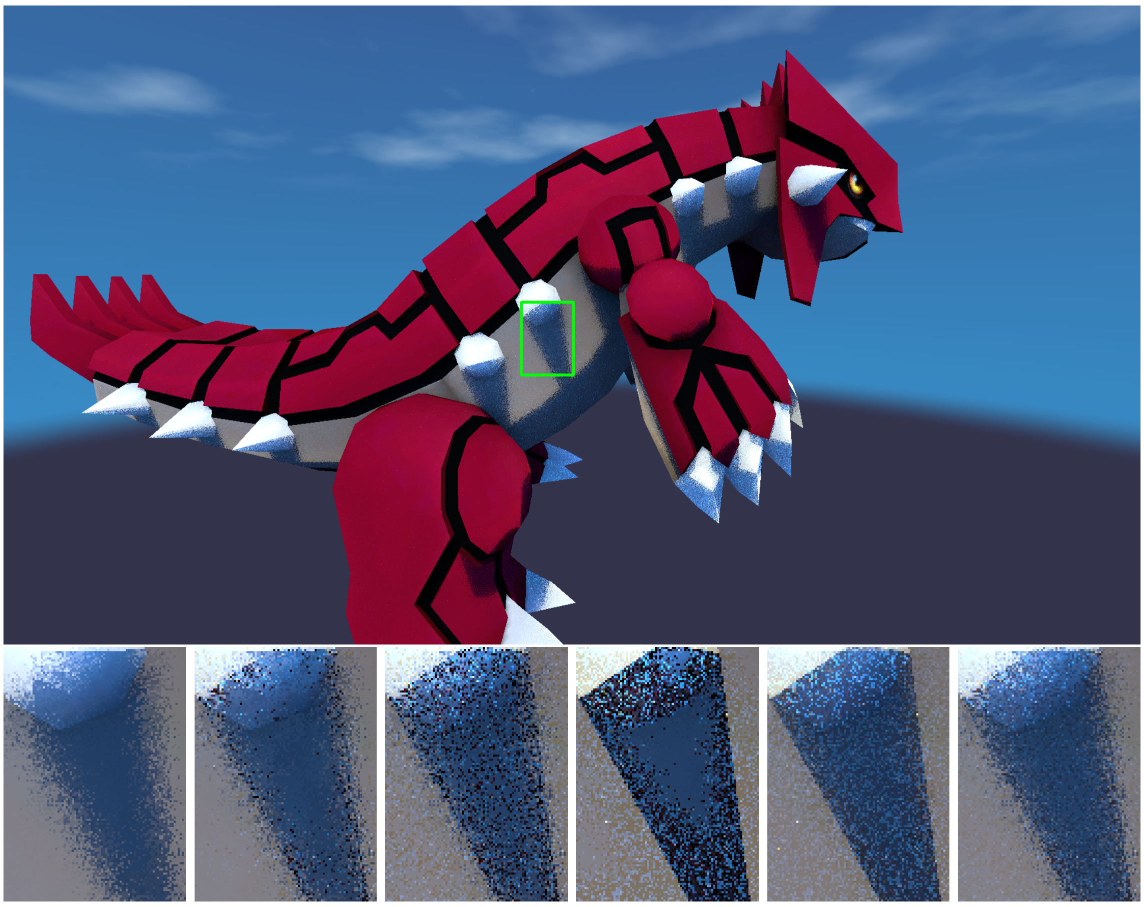


Figure 4.2: Example image of adaptive path space filtering for primary surfaces.



(a) (b) (c) (d) (e) (f)

Figure 4.3: Comparison between different combinations of functions used for primary methods variance aware spatial cell interpolation (va-interpolation) and variance aware path survival (va path survival): (a) Path space filtering as a reference, (b) uses zero for va interpolation and the linear function 4.1 for va path survival, (c) uses zero for va interpolation and the smooth linear function 4.2 for va path survival, (d) uses the step function 4.4 for va interpolation and one for variance aware path survival, (e) uses the one over x squared function 4.5 for va interpolation and one for va path survival and (f) uses the one over x squared function 4.5 for va interpolation and the smooth linear function 4.2 for va path survival.

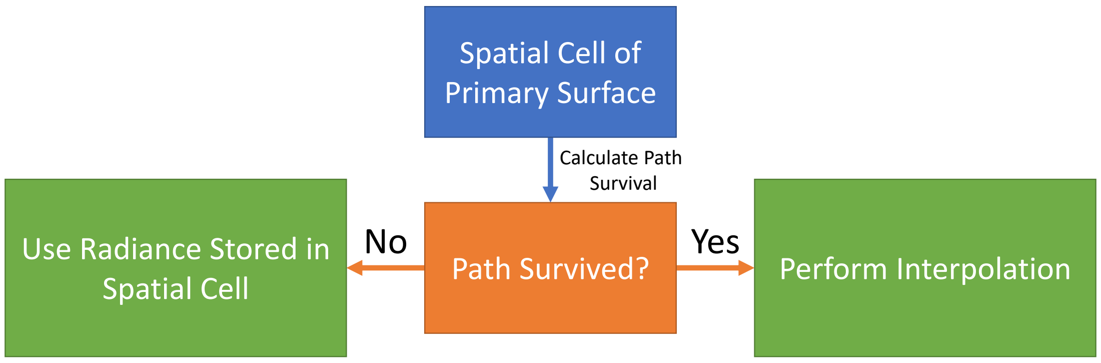

Spatial Cell of Primary Surface
Calculate Path Survival
Use Radiance Stored in Spatial Cell
No
Path Survived?
Yes
Perform Interpolation

Figure 4.4: Illustration of the interaction between variance aware path survival and spatial cell interpolation.

4.2 Adaptive Path Graphs

Another way to utilize the variance measure is by embedding it into the path tracer itself. The idea consists of using the variance measure provided by Bolin and Meyer [BM97]. By utilizing equation 3.6, it is possible to limit the variance induced by the path tracer with a constant value such that $V[f] < \text{const}$. Assuming $V[f] \geq \text{const}$:

$$\begin{aligned} &\text{const} \leq V[f] \\ \Leftrightarrow\ &\text{const} \leq VE\left[f|x_0\right] + \sum_j EV\left[E\left[f|x_0 ... x_j\right]|x_0 ... x_{j-1}\right] \end{aligned} \tag{4.6}$$

it is possible to find the largest possible index L such that:

$$\text{const} > VE\left[f|x_0\right] + \sum_j^L EV\left[E\left[f|x_0 ... x_j\right]|x_0 ... x_{j-1}\right] \tag{4.7}$$

The measurement contribution function f from equation 2.3 depends on $x_0, ..., x_L, ..., x_k$. It is therefore possible to bind the variance below a threshold by cutting the contributions after the index L. A better solution than removing them would be replacing them with zero variance contributions. These zero variance contributions can be drawn from the spatial structure as they only induce bias and not variance.

Using the linearity of expectation and removing the aliasing term $VE\left[f|x_0\right]$ yields:

$$\text{const} > E\left[\sum_j^L V\left[E\left[f|x_0 ... x_j\right]|x_0 ... x_{j-1}\right]\right] \tag{4.8}$$

This formula no longer operates on separate layers but on separate paths instead. As Aliasing induces little variance, it is possible to remove the aliasing term. It will thus not be taken into consideration for this thesis.

Equation 4.8 holds true, especially if the variance of each path is smaller than the given constant. This can be written as:

$$\forall (x_0, ..., x_L, ... x_k) \in P : \text{const} > \sum_j^L V\left[E\left[f|x_0 ... x_j\right]|x_0 ... x_{j-1}\right] \tag{4.9}$$

The last problem consists of finding the index L. This can be done by summing up the variance for every path iteration like this:

$$V_i[f] = V_{i-1}[f] + V\left[E\left[f|x_0 ... x_i\right]|x_0 ... x_{i-1}\right] \tag{4.10}$$

the index L is reached once the variance exceeds the given constant after being increased.

In order to use this statement for a path tracer, it is necessary to evaluate the term $V\left[E\left[f|x_0 ... x_i\right]|x_0 ... x_{i-1}\right]$. This is done by utilizing the measurement contribution provided by equation 2.3. One starts by separating the measurement contribution into a suffix- and postfix-term at an index j:

$$\begin{aligned} f(X) &= W\left(\prod_{i=2}^{k-1} f_r(x_{i-1}, x_i, x_{i+1}) G(x_{i-1}, x_i)\right) G(x_{k-1}, x_k) L_e \\ &= f_{pre}(X, j) \cdot f_{post}(X, j) \end{aligned} \tag{4.11}$$

The prefix-term is thus calculated as:

$$f_{pre}(X, j) = W\left(\prod_{i=2}^{j} f_r(x_{i-1}, x_i, x_{i+1}) G(x_{i-1}, x_i)\right) = W \cdot t_j(X) \tag{4.12}$$

with $t_j(X)$ being the throughput at index j. The postfix-term is:

$$f_{post}(X, j) = \left(\prod_{i=j+1}^{k-1} f_r(x_{i-1}, x_i, x_{i+1}) G(x_{i-1}, x_i)\right) G(x_{k-1}, x_k) L_e \tag{4.13}$$

using this separation, it is possible to write the variance of a path iteration i as:

$$\begin{aligned} V_i[f] &= V_{i-1}[f] + V\left[E\left[f(X) | x_0 ... x_i\right] | x_0 ... x_{i-1}\right] \\ &= V_{i-1}[f] + V\left[E\left[f_{pre}(X, i-1) \cdot f_{post}(X, i-1) | x_0 ... x_i\right] | x_0 ... x_{i-1}\right] \\ &= V_{i-1}[f] + f_{pre}^2(X, i-1) \cdot V\left[E\left[f_{post}(X, i-1) | x_0 ... x_i\right] | x_0 ... x_{i-1}\right] \\ &= V_{i-1}[f] + W^2 t_{i-1}(X)^2 \cdot \underbrace{V\left[E\left[f_{post}(X, i-1) | x_0 ... x_i\right] | x_0 ... x_{i-1}\right]}_{\text{variance estimate provided by the spatial cell}} \end{aligned} \tag{4.14}$$

Clamping the generated variance below a given threshold would thus consist of carrying a throughput and variance variable. For each step, the variance gets incremented by the throughput squared times the estimated variance of the spatial cell, and the throughput gets multiplied by the current bsdf weight afterwards.

Algorithm 2 does exactly that. It checks if the given path vertex induces enough variance such that it would exceed the given threshold. In that case the algorithm would opt to use the radiance given by the spatial cell which is said to have a bias but no variance.

At last, the local radiance and the radiance from the next intersection, provided by the spatial structure, are written to the spatial structure for the current intersection. This leads to algorithm 2. The effects on the spatial structure can be seen in figure 4.5.

The values inserted into the spatial cells are divided by the albedo of the current path vertex. This is done in order to maintain high frequency texture details. It is not possible to demodulate it as done in path space filtering, as the assumption provided for equation 2.15 does not hold here. Equation 2.15 handles diffuse surfaces only and can thus not be used for non-diffuse surfaces.

The first benefit is that it is easy to implement this technique in a standard path tracer. The path space filtering pipeline already covers all necessary computation outside the path tracer. Implementing this method thus only persists of adding the code depicted in algorithm 2.

The main benefit is that the variance of the contribution approximately never exceeds the given constant. It is therefore a great tool to tune the variance-bias trade-off between path tracing and path space filtering. An example of how the variance constant changes the result of adaptive path graphs is depicted in figure 4.6.

In order to have a smoother transition, the contribution of the path tracer gets blended with the contribution of the spatial cell, such that the variance reaches exactly the variance threshold once it overshoots. This is done by introducing an alpha value that weights the usage of the variance inducing contribution, such that it induces exactly the correct amount of variance. Let `var` be the accumulated variance, `alpha` be the sought alpha blend variable, `tp` the throughput and E the estimated postfix provided by the spatial cell:

$$\begin{aligned}\texttt{var} + V[\texttt{alpha} \cdot \texttt{tp} \cdot E] &= \texttt{const}\\ \texttt{var} + \texttt{alpha}^2 \cdot \texttt{tp}^2 \cdot V[E] &= \texttt{const}\\ \texttt{alpha} &= \sqrt{\frac{\texttt{const} - \texttt{var}}{\texttt{tp}^2 \cdot V[E]}}\end{aligned} \tag{4.15}$$

Multiplying this alpha value with the incoming contribution and weighting the zero variance contribution of the spatial cell with $(1 - \texttt{alpha})$ yields the desired blending.

One remaining problem is that the spatial cells could end up in cyclic dependencies. This could cause cells to build up artifacts by continuously affecting each other. In order to prevent this, the current level of indirection is used to separate the cells in the spatial structure. It is thereby guaranteed that no cyclic dependency can arise but the number of samples per indirection is reduced.

Figure 4.5 shows that adaptive path graphs generates path information by a segment wise update. It therefore does not need to store the path contributions separately. It can access and insert these contributions with local information only. This leads to the last benefit of having a small register pressure.

Algorithm 2: Adding adaptive path graphs to a given default path tracer

```
1:   con ← 0 // contribution
2:   tp ← 1 // throughput
+3:  var ← 0 // variance
+4:  terminated ← false
+5:  alpha ← 1

6:   ray ← cameraray()
7:   currentIT ← intersect(ray)

8:   for i in (0, maxLength - 1) do
         // check variance
+9:      if not terminated then
             // calculate using first and second moment, see equation 3.2
+10:         varLocal ← $\texttt{tp}^2 \cdot (\texttt{currentIT.hm.mom2 - currentIT.hm.mom1}^2)$
+11:         if var + varLocal > const then
+12:             alpha ← $\sqrt{\frac{\texttt{const}-\texttt{var}}{\texttt{varLocal}}}$ // blending such that var = const
+13:             con ← con + $(1 - \texttt{alpha}) \cdot \texttt{tp} \cdot$ currentIT.hm.raidance

+14:             terminated ← true
+15:             var ← const // var = const, see equation 4.15
+16:         else
+17:             var ← var + varLocal

         // direct
18:      conNEE ← NEE(currentIT) // contribution from next-event estimation
+19:     con ← con+ alpha$\cdot$ tp $\cdot$ conNEE

         // indirect
20:      ray, bsdfWeight ← sampleBSDF(currentIT)
21:      tp ← tp $\cdot$ bsdfWeight
22:      nextIT ← intersect(ray)

+23:     currentIt.hm.insert(conNEE + bsdfWeight $\cdot$ nextIT.hm.radiance)
24:      currentIT ← nextIT

+25:     if terminated then
+26:         alpha ← 0 // disable contributions from the path tracer
```

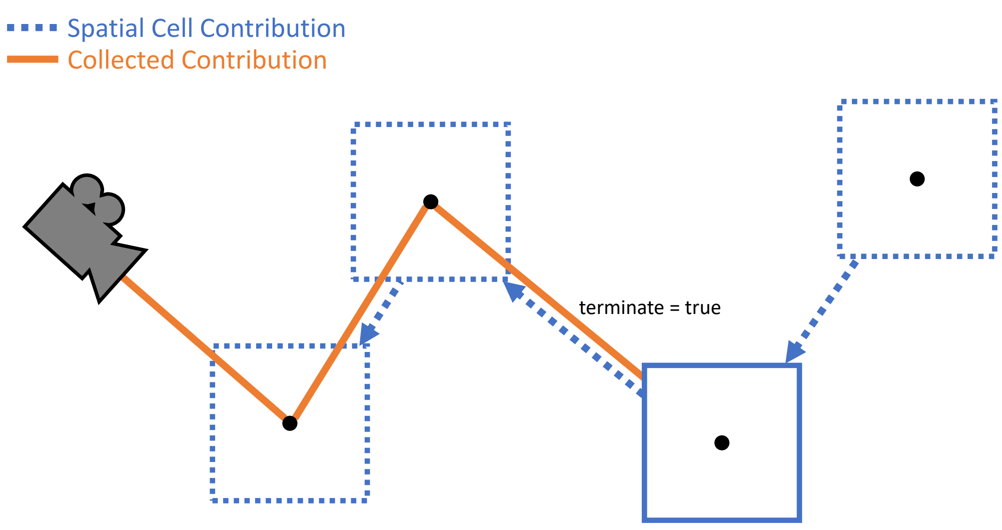


Figure 4.5: Illustration of adaptive path graphs.

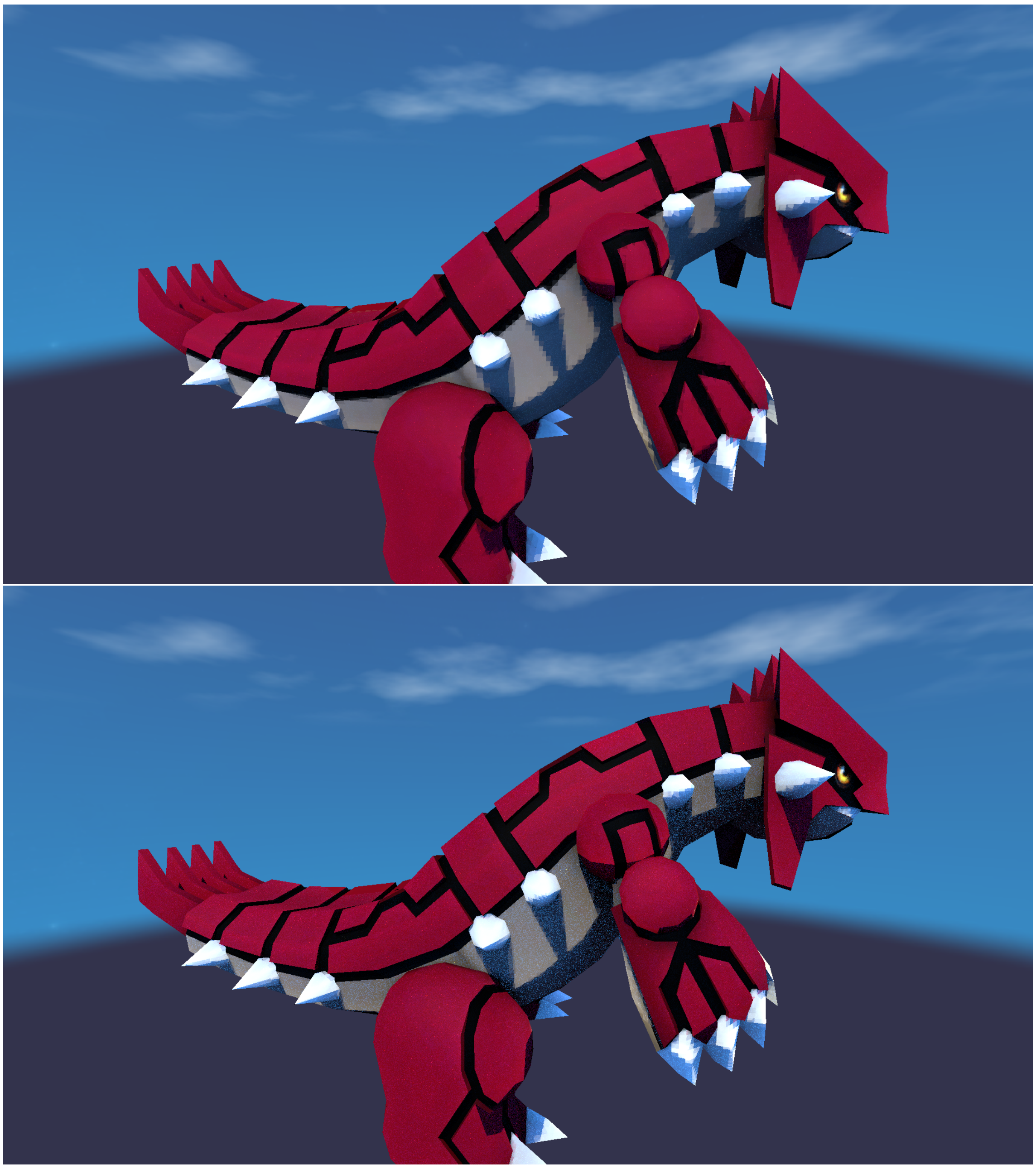


Figure 4.6: Illustration how the variance constant changes the image for non-jittered adaptive path graphs.

5. Implementation

This chapter discusses implementation details of the program used to evaluate and compare the given methods.

5.1 Graphics Pipeline

The overall structure of the graphics pipeline can be seen in figure 5.1. The pipeline starts with the generation of a sky box, then it builds the necessary structures for rendering, after that it uses a deferred pass in order to handle path tracing, next it uses an optional denoiser and finalizes by drawing the user interface. It is separated into three parts, two Vulkan command buffers and a CUDA call. This separation is necessary for the OptiX denoiser, as it is implemented in CUDA.

The first Vulkan command buffer generates the image that needs to be denoised. In order to do that it first generates a sky box. The sky box generation consists of drawing a cube map for the night sky and two spheres, one for the blue sky, sun and moon and one for the clouds, which consists of perlin noise. The sky is drawn to a cube map which is done by rendering it for each face of the cube map. This generates the resulting dynamic environment map.

After that it rebuilds the top level ray tracing acceleration structure. The bottom level acceleration structure is already generated after loading the scene. As this program does not provide compatibility for animation other than matrix transformation of whole instances, there is no need to rebuild the bottom level acceleration structures.

The next step is to apply the hash map eviction. Details regarding these steps can be seen in 5.3.

As this program uses a deferred renderer, the next step is the G-Buffer prepass. The layout of the G-Buffer can be seen in figure 5.2.

This is followed by the deferred ray tracing pass, which is further discussed in section 5.2.

The path space filtering pass is also a deferred ray tracing pass, which is executed next. It is explained in 5.3.

These steps yield the image that needs to be denoised next. In order to do that, the image is copied into an external buffer that can be read from Vulkan and from CUDA as well. This is achieved with a compute shader. This concludes the first Vulkan command buffer.

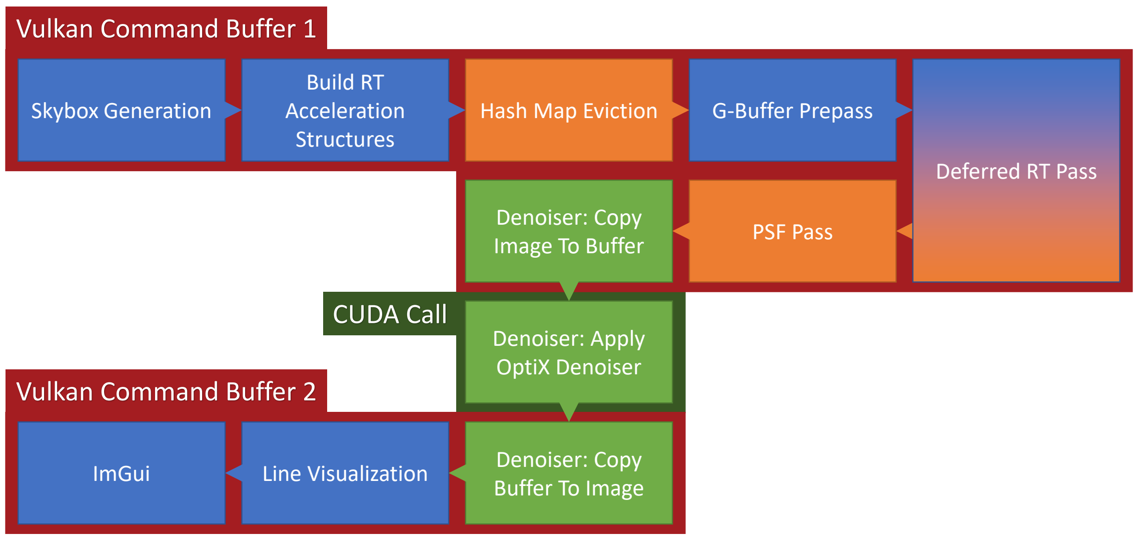


Figure 5.1: The current pipeline of this project. The synchronization between Vulkan and CUDA is handled by an external semaphore.

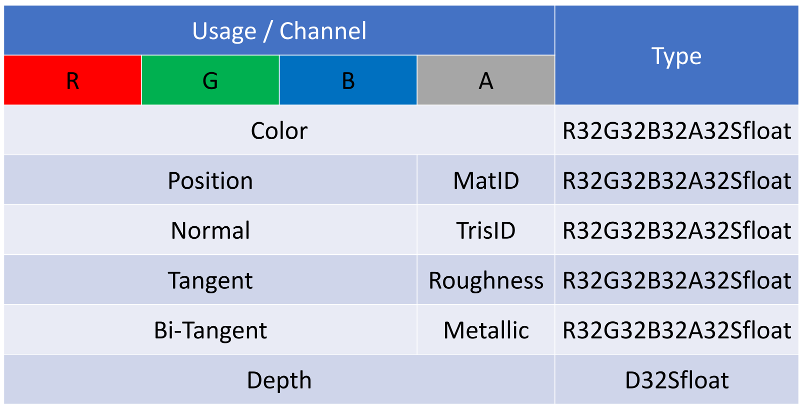
The OptiX denoiser is executed with CUDA. In order to synchronize Vulkan and CUDA, an external semaphore is used. This follows the implementation provided by NVIDIA[1].

Once the denoising is finished, the result is written to the external buffer. The second Vulkan command buffer is now executed.

The execution starts with a compute shader in order to write the buffer data back to an image.

A line visualization used for debug purposes is rendered and after that ImGui is rendered to the resulting image. The image gets written to the swap chain image, which concludes the graphics pipeline.

[1] `https://github.com/nvpro-samples/vk_denoise`



Usage / Channel				Type
R	G	B	A	
Color				R32G32B32A32Sfloat
Position			MatID	R32G32B32A32Sfloat
Normal			TrisID	R32G32B32A32Sfloat
Tangent			Roughness	R32G32B32A32Sfloat
Bi-Tangent			Metallic	R32G32B32A32Sfloat
Depth				D32Sfloat

Figure 5.2: The current layout of the G-Buffer. This can definitely be optimized.

5.2 Ray Tracing

Ray tracing is executed by utilizing the ray query extension inside deferred shaders. The main benefit is that the main bounce can be executed faster by the rasterization pipeline. The fragment shader for the prepass generates many fragments that are not visible. Running computationally expensive code like ray queries for every fragment would thus be disadvantageous. A pass-through shader with one fragment per pixel is thus used for ray query computation.

Hitting a triangle with a ray query returns an instance ID, a triangle ID and barycentric coordinates. By binding the index buffer, the instance offset buffer and the vertex buffer to this shader, it is easy to get the correctly interpolated hit vertex. Realizing whether the ray missed or not can be achieved by checking the intersection type.

5.3 Path Space Filtering

In order to implement path space filtering properly, it is necessary to read and write to a hash map in the same frame. This is not a problem for path space filtering as it works with two passes, but the adaptive techniques provided by this paper need to access the hash map in the same path tracer pass. The hash maps are written to by atomic float operations. If done so, the hash map cell that gets written cannot be read in the same pass as it will have corrupted values, for instance, while the counter has been increased the radiance was not added yet.

In order to fix this problem, a dual buffering scheme is used. While the first hash map will be written in this frame, the second one is read. After every frame, the hash map, previously written to, gets copied into the other hash map in the eviction shader, after applying the eviction and exponential moving average.

Path space filtering thus persists of three main passes: the hash map eviction, the path tracer and the path space filtering pass.

The hash map eviction is a compute shader pass that not only copies the written hash map into the other hash map, but it also evicts cells that have not been used for a longer period of time.

Before copying the hash map, an exponential moving average for a given constant maximum cell size is applied. This exponential moving average needs to work on batches as many hash map insertions could have occurred in the previous frame. This is done by declaring a maximum cell size N and multiplying the values in the written cells by N over the number of accumulated samples. In order to evict cells, they also carry a time stamp which is only updated if the cell has been used before. If the time stamp on a cell deviates too much from the current time, it gets cleared.

The hash map also uses a dynamic size. The hash maps are written with a fixed size in VRAM but the actually used size is handled by a constant. This constant is dynamically re-sized depending on current settings like hash map cell size or screen resolution. This drastically improves performance for the eviction shader, as it only works on the dynamic size and not the whole allocated hash map.

The path tracer accumulates the contribution over every vertex along the path. It does not apply the local color as this is done later on in the path space filtering pass. After accumulating the values they are inserted in the hash map.

The path space filter shader reads those values from the hash map at the first bounce generated by the G-buffer. It applies the albedo color from the texture and displays the resulting image.

The path space filtering pass also serves some debug purposes like displaying the current hash map occupation.

6. Results

This chapter discusses the results generated by the two introduced techniques and compares them to path space filtering and path tracing. The first comparison is based on image quality, which is measured in terms of mean squared error (MSE). This comparison is performed and analyzed in section 6.1.

The other comparison consists of a frame time analysis, measured in milli-seconds, which is evaluated in 6.2.

At last, the relation between terminated path depth and variance constant for adaptive path graphs 4.2 is analyzed in section 6.3.

All resulting values and images are generated on a system with an Intel core i7-9700k, a NVIDIA GeForce RTX 3080 Ti and 16GB RAM.

6.1 Image Quality

The image quality is measured in terms of mean squared error (MSE). The equation for the MSE can be written as:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(Y_i - \hat{Y}_i)^2 \tag{6.1}$$

where Y_i describes the correct value drawn from a reference and $\hat{Y}_i$ the estimated value.

The reference is generated by accumulating the result of an unbiased technique, in this case path tracing. Doing so yields the results seen in figures 6.1 - 6.4.

Figure 6.1 shows that the techniques presented by this thesis produce roughly the same MSE as path space filtering. The same pattern holds true for figure 6.2 and 6.3. There are some instances in which the adaptive techniques generate a smaller MSE and some in which they produce a higher MSE than path space filtering. The MSE of the adaptive techniques in figure 6.4 is always a little bit lower than the MSE of path space filtering.

As the adaptive techniques opt to provide a trade-off between bias and variance, it is not surprising to see this result, as both bias and variance raise the mean squared error.

In order to analyze this trade-off, it is necessary to calculate the bias and the variance of the techniques.

The bias is calculated by calculating the MSE between a path tracing reference and an accumulated result of the given technique. Accumulating the result of the technique rids it from the induced variance, therefore the only error remaining is caused by the bias. This can be seen in figures 6.5 - 6.8.

Figure 6.5 shows that adaptive path space filtering for primary surfaces outperforms both adaptive path graphs and path space filtering. Adaptive path graphs generates a smaller bias than path space filtering.

The first crop in figure 6.6 shows the strongest benefit of adaptive path graphs, as the shadow of the lamp is only visible when applying this technique. The other two examples show artifacts of variance for adaptive path graphs, which leads to the assumption that 1000 frames are not enough for this scenario. Adaptive path graphs is the best technique for all three crops and adaptive path space filtering for primary surfaces has a slight edge over path space filtering.

Figure 6.7 shows another example where adaptive path graphs outperforms the other techniques. It has the smallest bias for both the first and last crop. Adaptive path space filtering for primary surfaces has, as always, a smaller bias than path space filtering.

At last, figure 6.8 shows more examples in which adaptive techniques show a lower bias than path space filtering.

There is another way to analyze the bias of a technique by using the mean squared error. It is done by accumulating a scene with a different number of samples per pixel and comparing this result against an unbiased reference rendered with many samples per pixel. The MSE of biased techniques will thus converge towards the bias of said technique. In order to get a good result, a strongly converged reference is necessary. A path tracing result with 100000 samples per pixel is thus used. The resulting graph can be seen in figure 6.9. Note that the entire scene is evaluated and not just small crops. Adaptive path graphs has the smallest bias of the biased techniques. Adaptive path space filtering for primary surfaces has a very similar curve to path space filtering, which makes sense, as it only interpolates between path space filtering and path tracing. Using the path survival does not introduce bias.

The last analysis in the context of image quality is the variance comparison. The variance can be received by calculating the MSE between the accumulated result of the technique and the one sample per pixel result of said technique. The accumulated result would thus serve as the mean μ which results in the following equation:

$$\begin{aligned} MSE &= \frac{1}{N}\sum_{i=1}^{N}(Y_i - \hat{Y}_i)^2 \\ &= \frac{1}{N}\sum_{i=1}^{N}(\mu - \hat{Y}_i)^2 = V[\hat{Y}] \end{aligned} \tag{6.2}$$

which is exactly the definition of the variance.

The variances of the currently discussed crops are depicted in figures 6.10 - 6.13.

Figure 6.10 shows the obvious result, adaptive techniques introduce more variance than path space filtering. More interesting is how the variance constant affects the variance for adaptive path graphs. Note that the variance calculated here can be higher than the variance constant, as adaptive path graphs estimates the variance by utilizing the spatial structure. This estimation can thus lead to higher variance. In this case it is easy to see that the variance constant of 0.0001 does not bind the variance of the crops below itself.

Looking at figure 6.11 one can see that the variance of adaptive path graphs is indeed lower than the variance constant 0.04349. One can also see that adaptive techniques still introduce more variance than path space filtering.

Adaptive path graphs interestingly produces less variance than path space filtering for the first crop in figure 6.12.

Figure 6.13 shows similar results to figure 6.10.

It is therefore apparent that adaptive techniques introduce a variance-bias trade-off. The bias is lower than the bias of path space filtering but the variance is bigger in general.

The variance-bias trade-off is also analyzed in figure 6.14 by calculating the bias and variance as previously discussed for different variance constant values. The higher the variance constant, the more variance is allowed, therefore the variance gets bigger. The bias on the other hand gets smaller the higher the variance constant gets. This is exactly the trade-off discussed before.

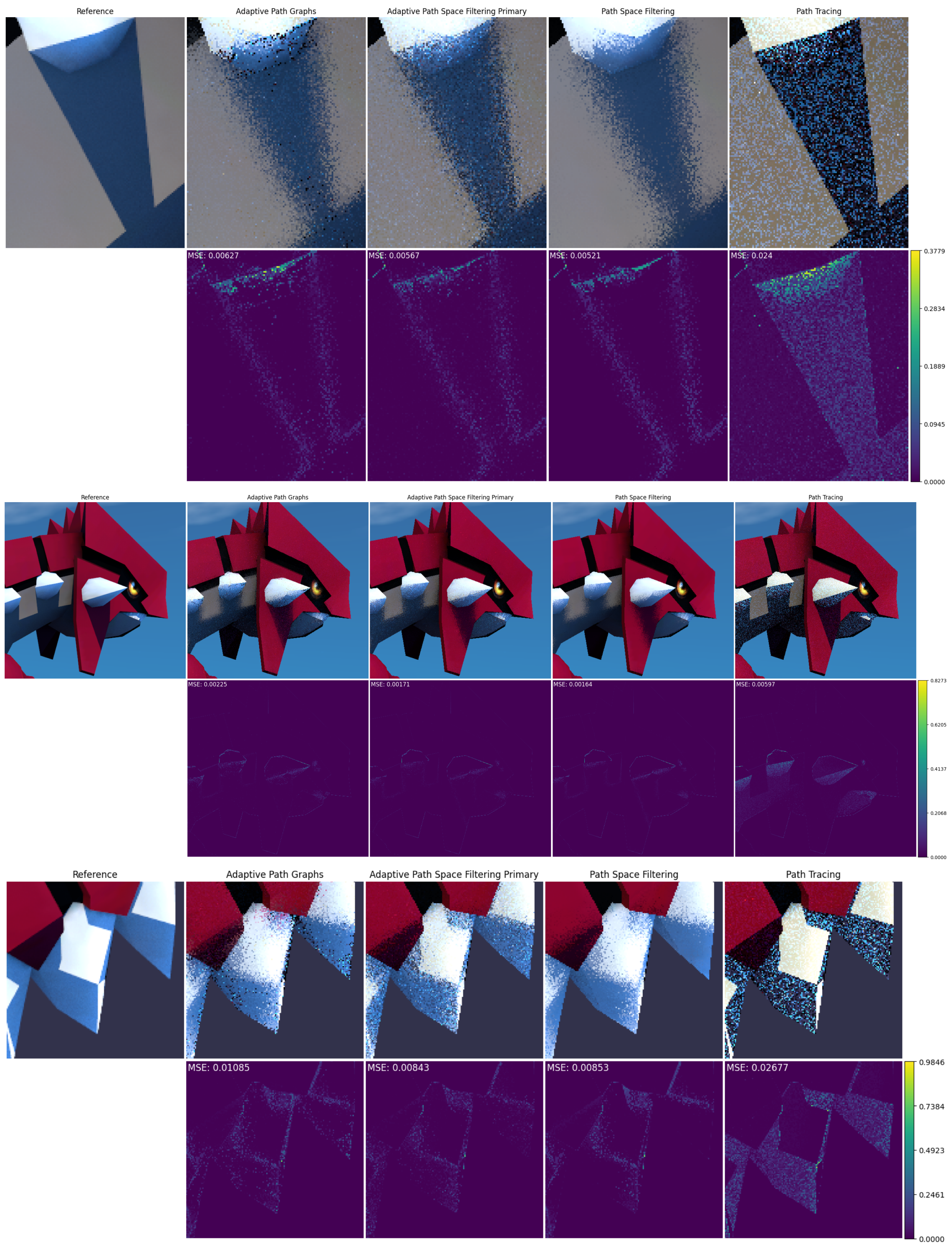


Figure 6.1: MSE comparison, the reference is generated by accumulating 1000 frames with path tracing.

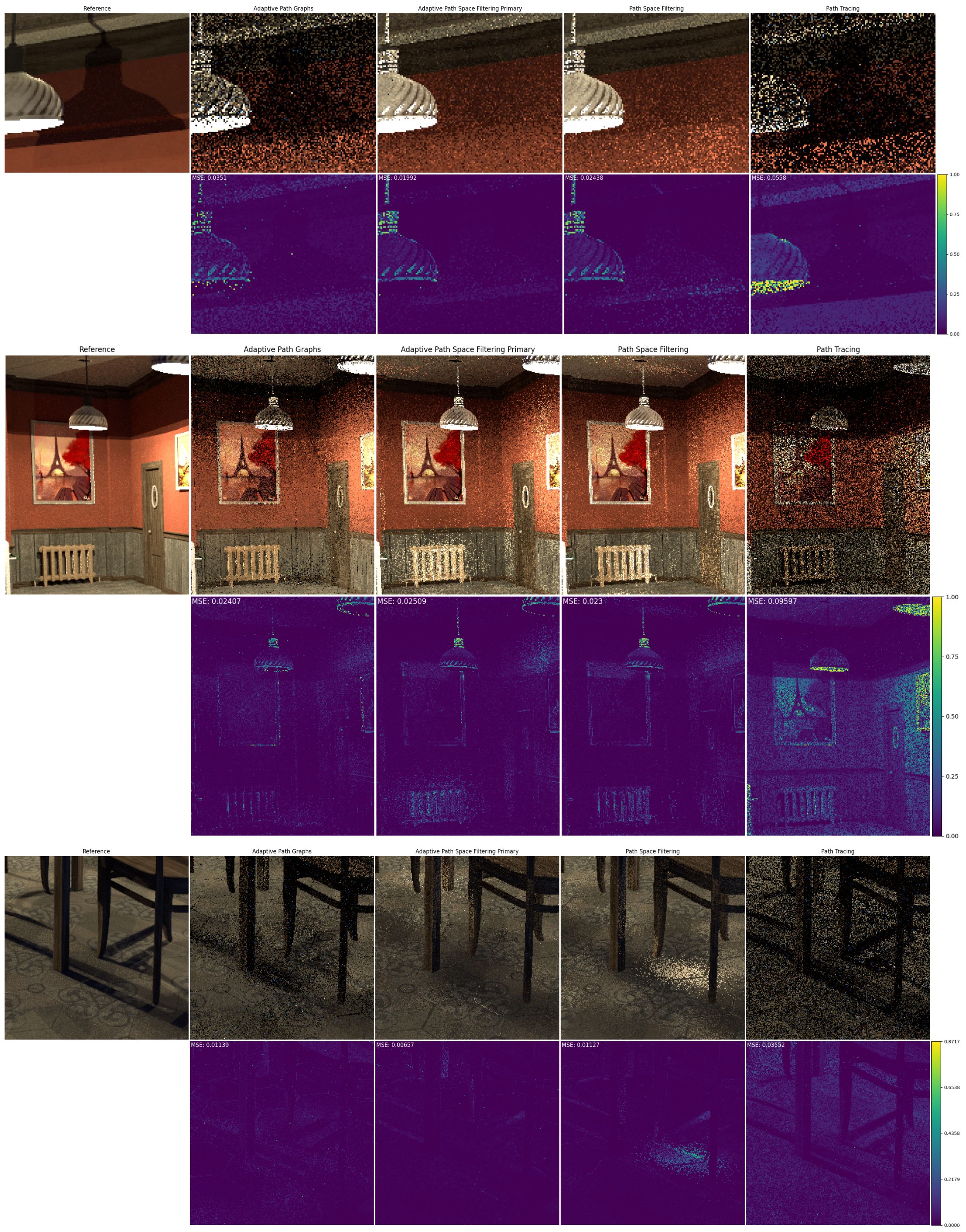


Figure 6.2: MSE comparison, the reference is generated by accumulating 100000 frames with path tracing.

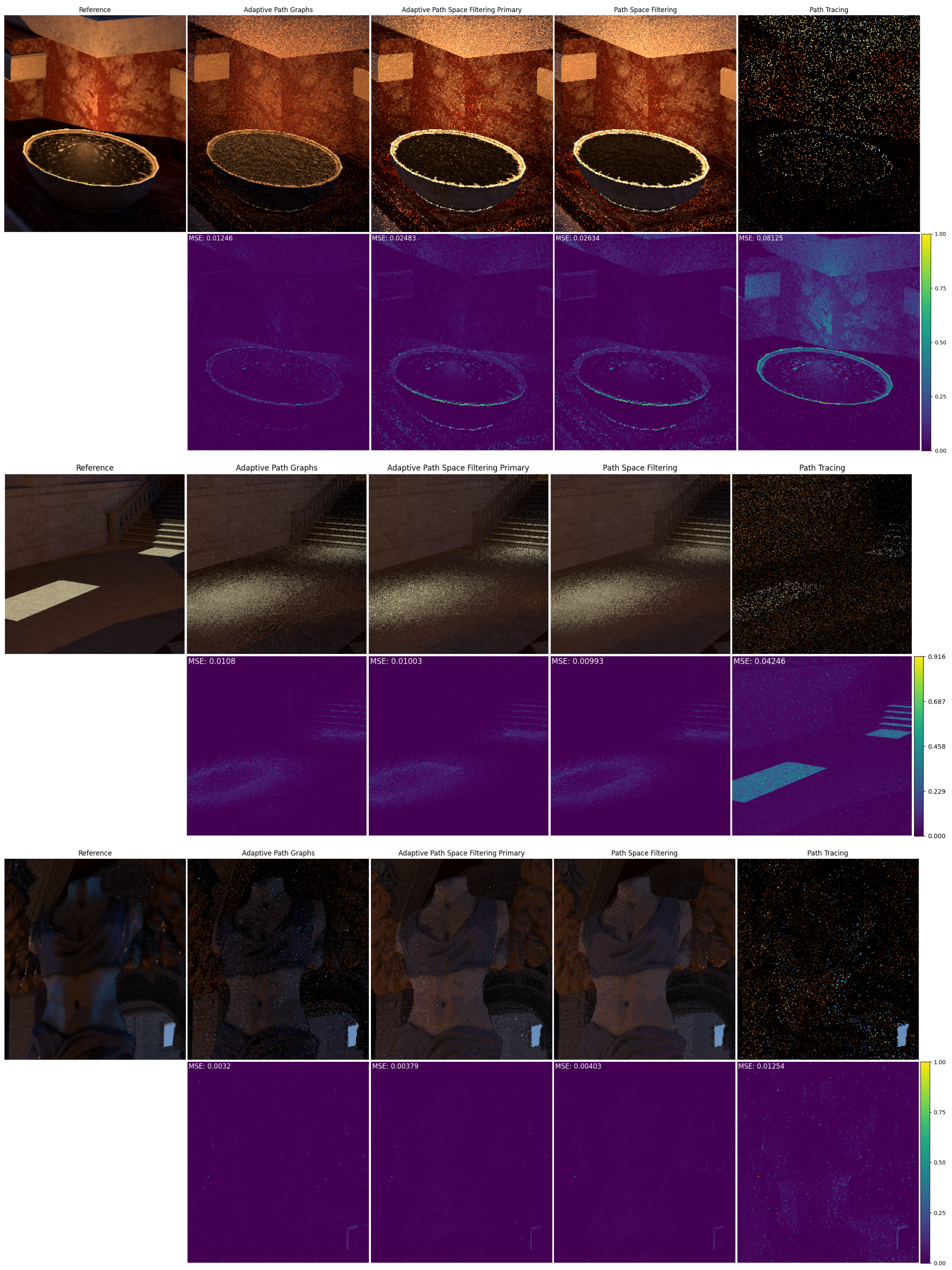


Figure 6.3: MSE comparison, the reference is generated by accumulating 1000 frames with path tracing.

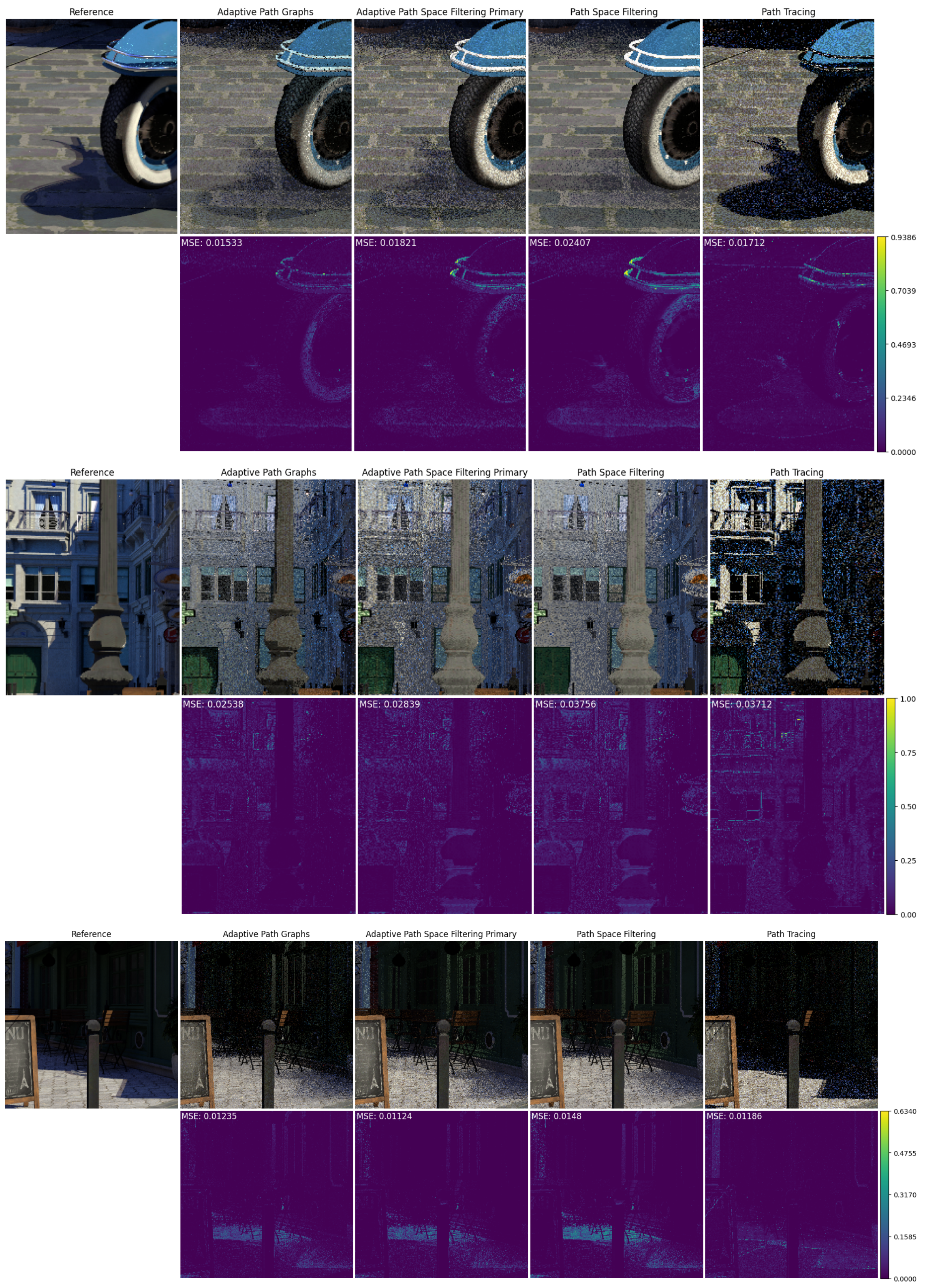


Figure 6.4: MSE comparison, the reference is generated by accumulating 1000 frames with path tracing. The screen shots are taken by a moving camera.

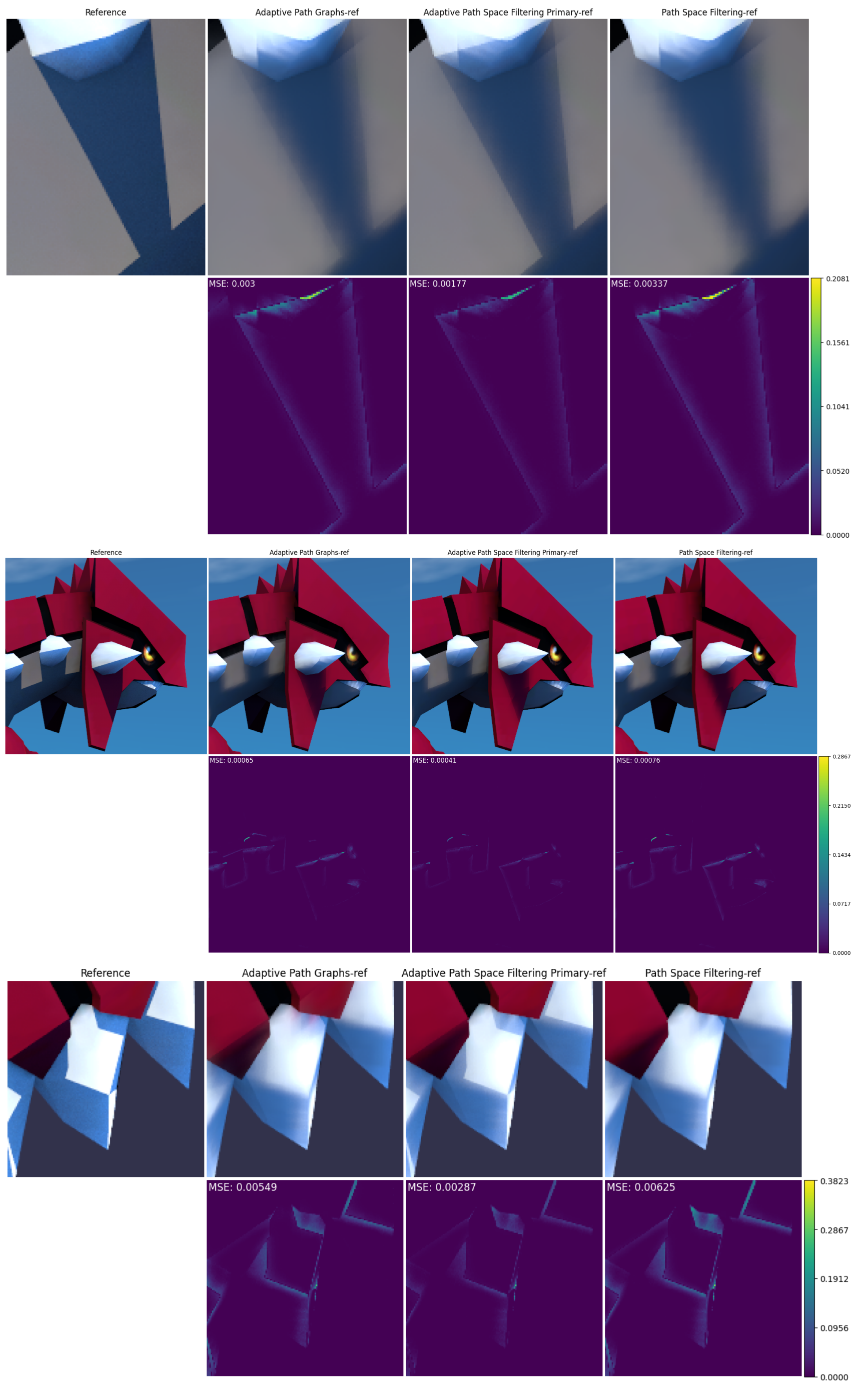


Figure 6.5: Bias comparison, the reference is generated by accumulating 1000 frames with path tracing. The techniques are also accumulated over 1000 frames

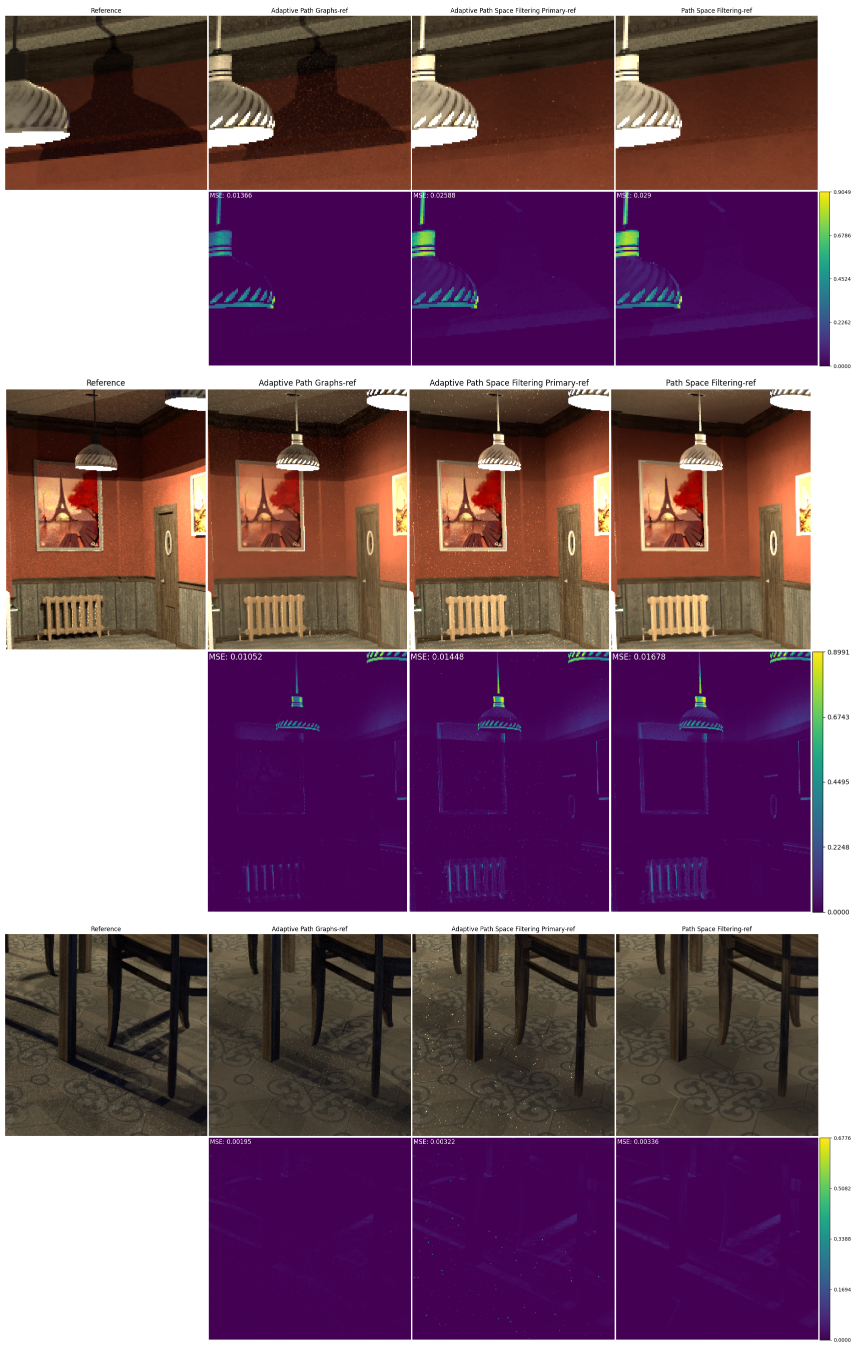


Figure 6.6: Bias comparison, the reference is generated by accumulating 100000 frames with path tracing. The techniques are accumulated over 1000 frames

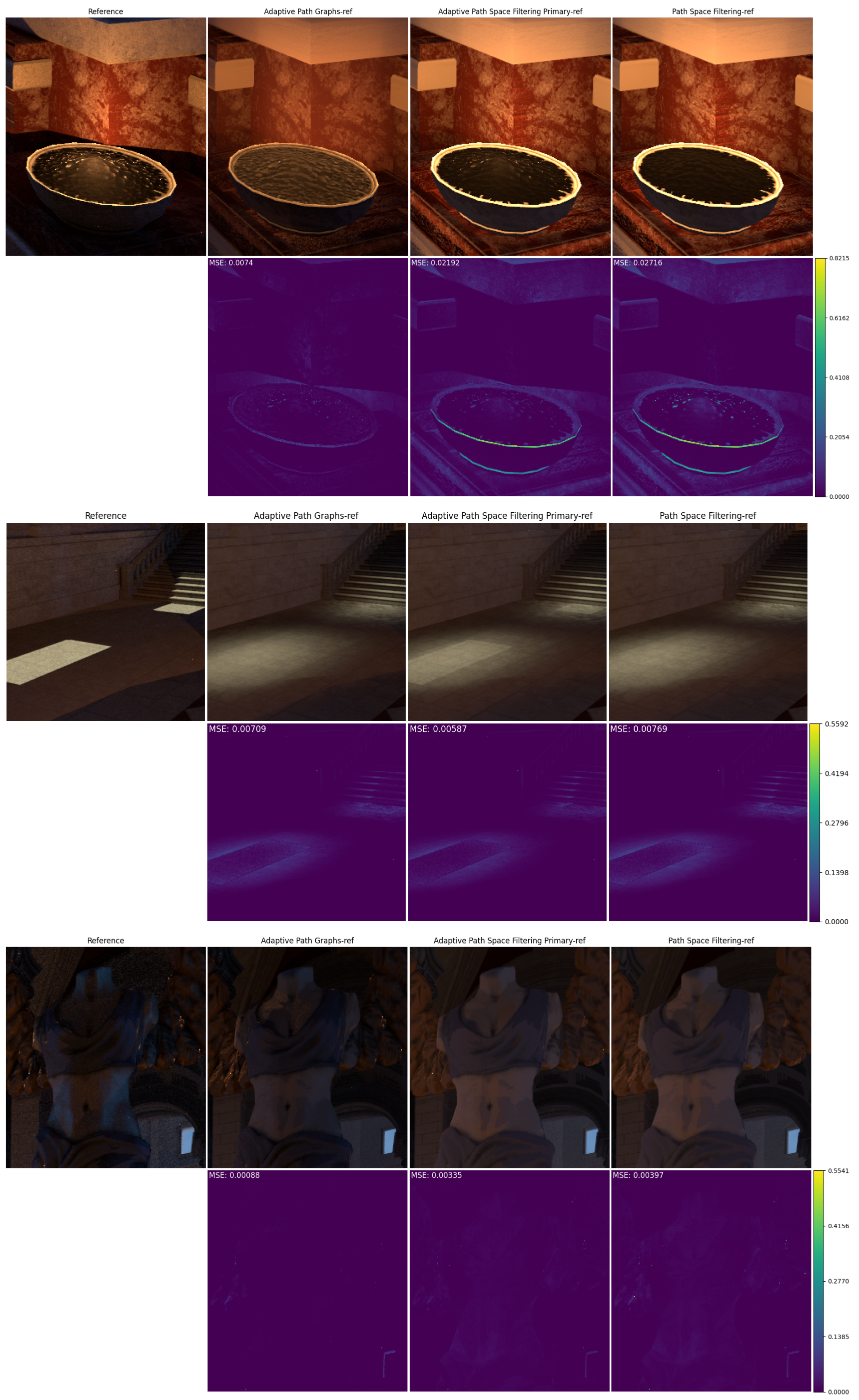


Figure 6.7: Bias comparison, the reference is generated by accumulating 1000 frames with path tracing. The techniques are also accumulated over 1000 frames

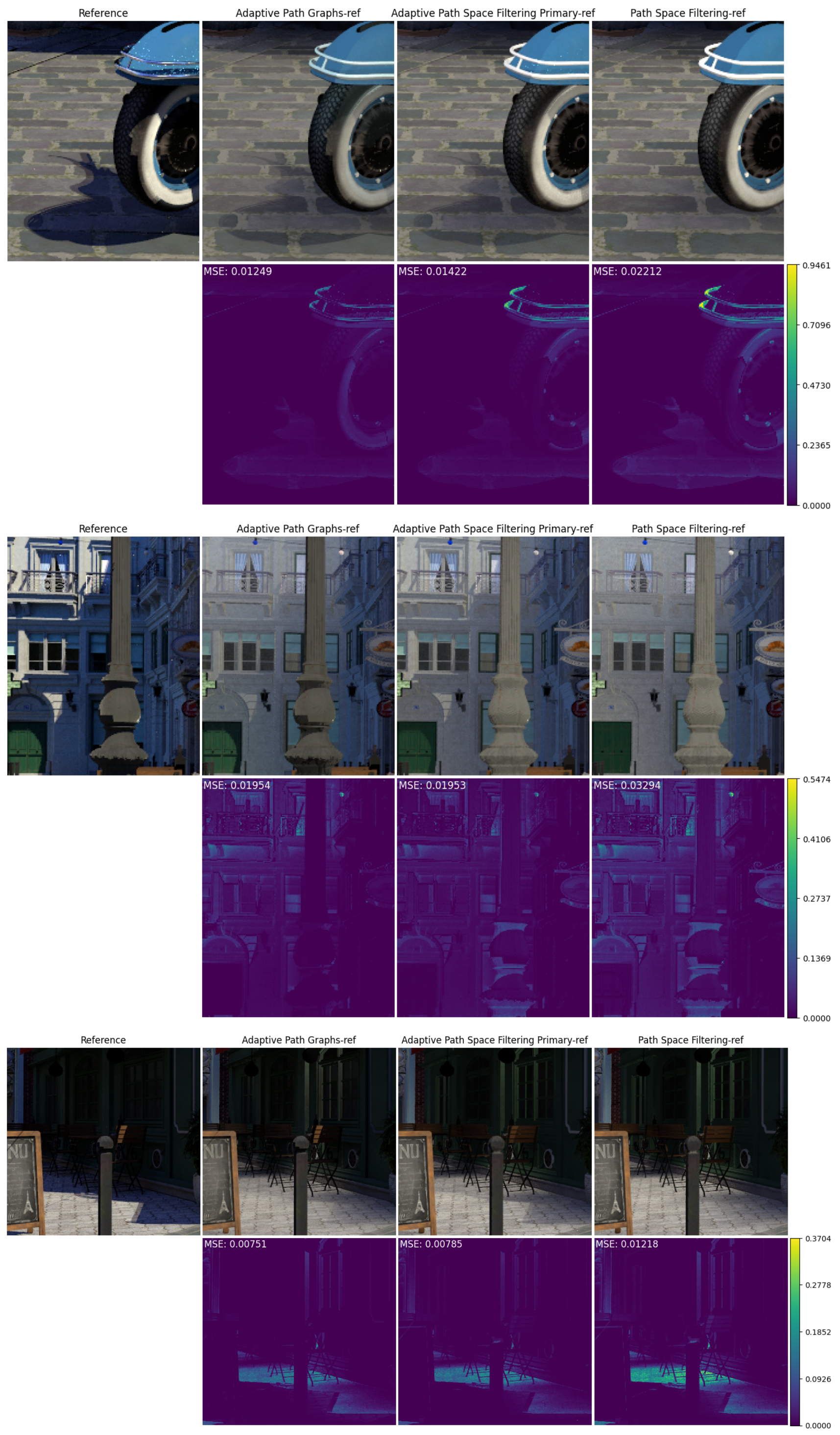


Figure 6.8: Bias comparison, the reference is generated by accumulating 1000 frames with path tracing. The techniques are also accumulated over 1000 frames. The screen shots are taken by a moving camera.

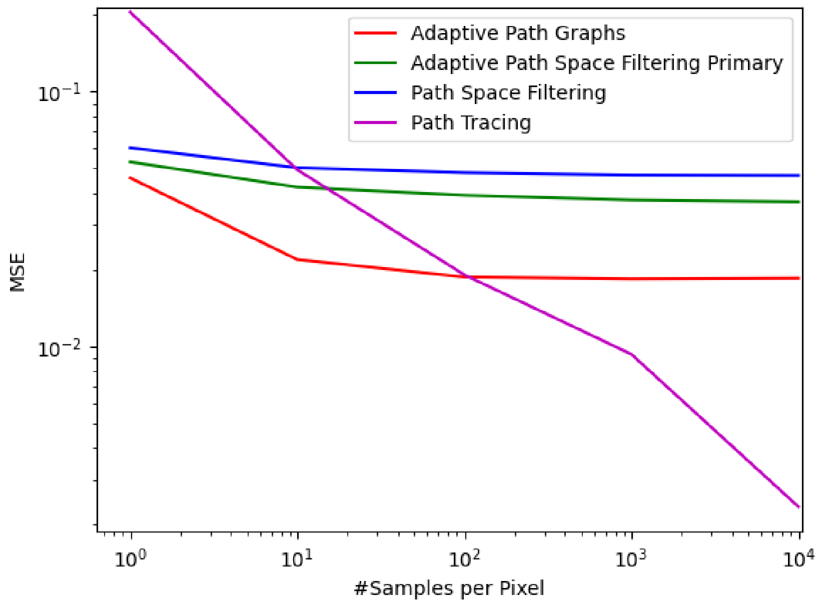


Figure 6.9: MSE for different number of accumulated samples per pixel. Biased methods converge toward that bias. The scene is the Bistro Interior scene with a resolution of 1920×1080 and the whole screen is evaluated for MSE.

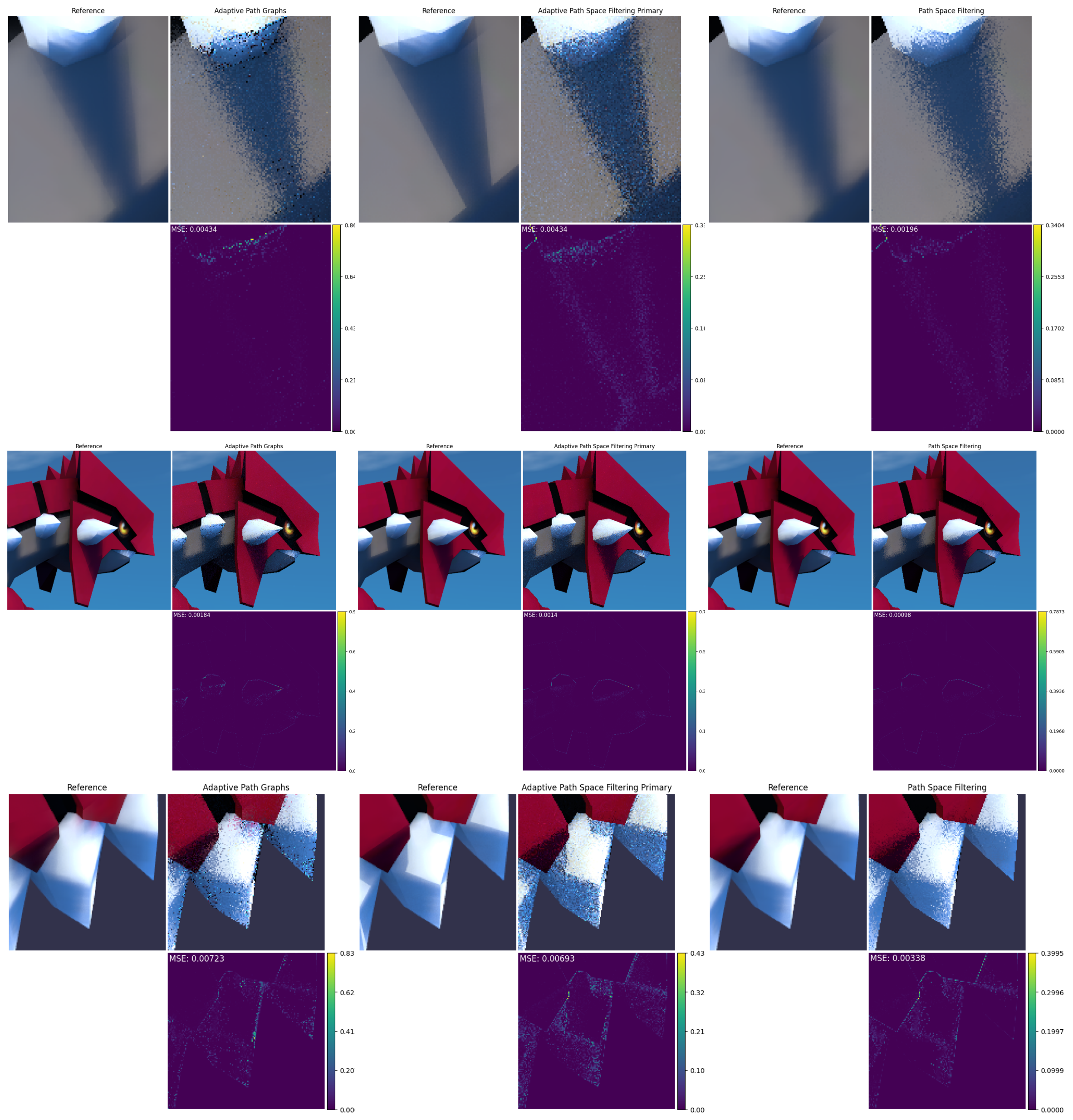


Figure 6.10: Variance comparison, the reference is generated by accumulating 1000 frames with the given technique. The variance constant for adaptive path graphs is set to 0.0001.

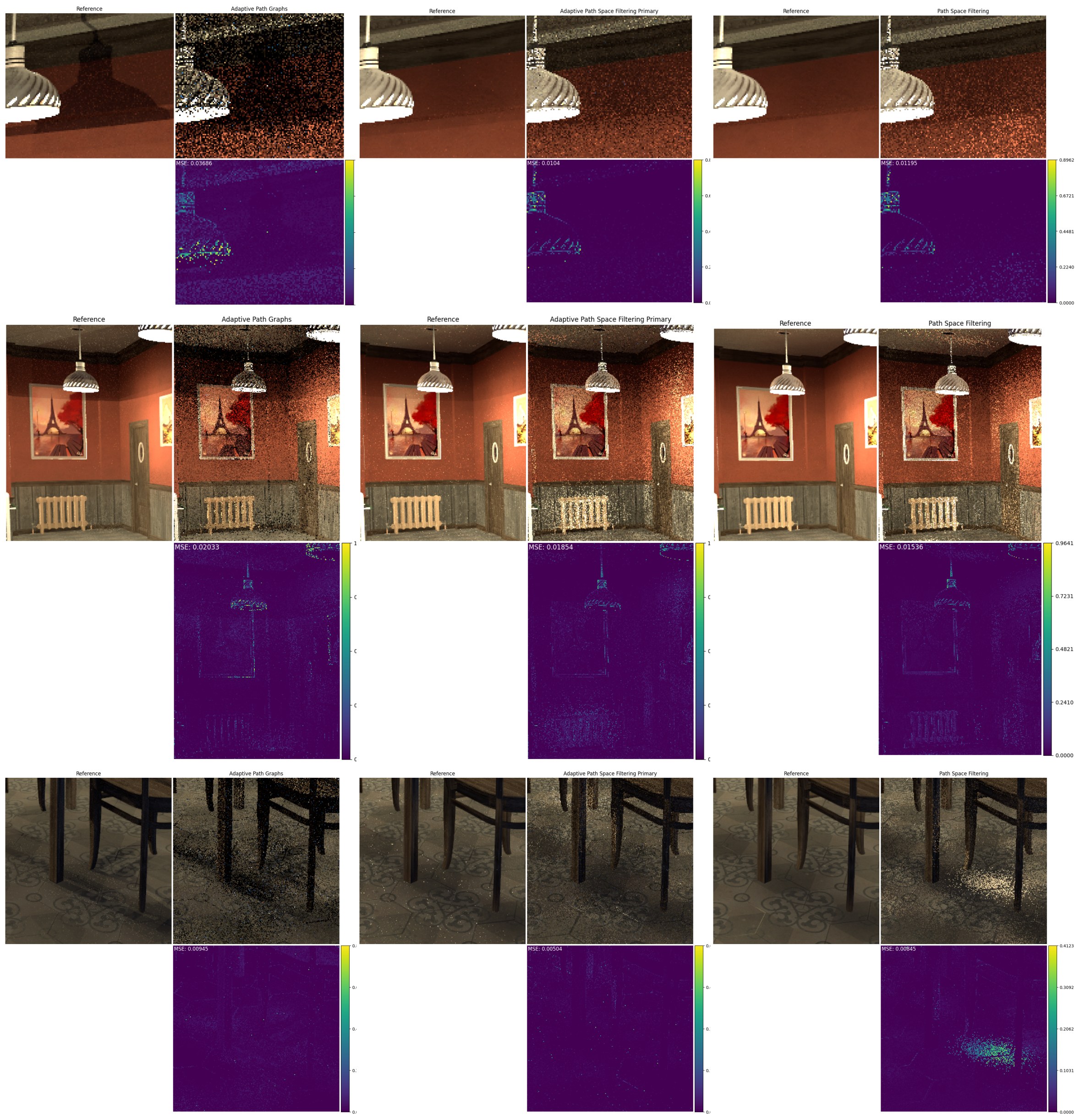


Figure 6.11: Variance comparison, the reference is generated by accumulating 1000 frames with the given technique. The variance constant for adaptive path graphs is set to 0.04349.

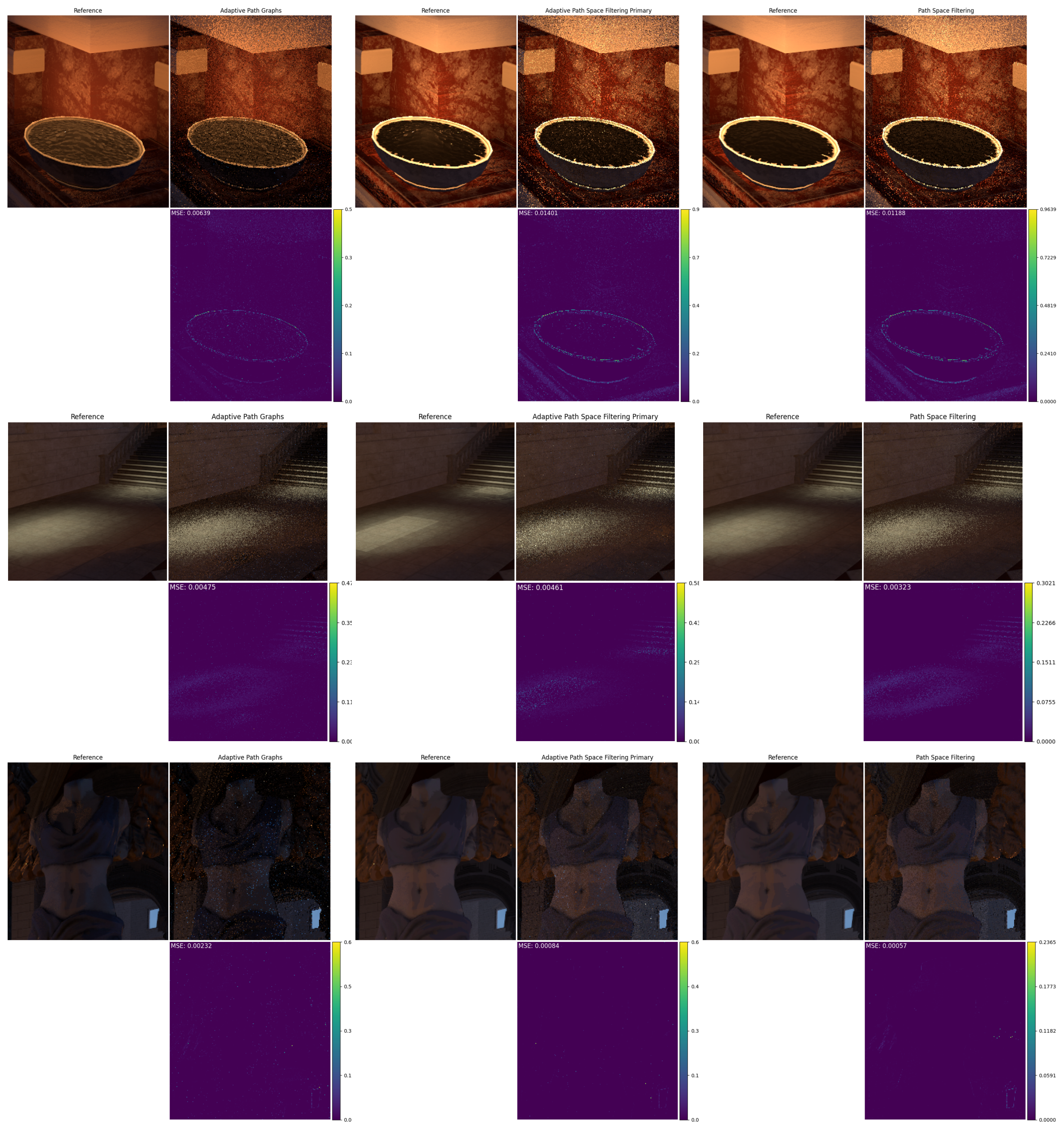


Figure 6.12: Variance comparison, the reference is generated by accumulating 1000 frames with the given technique. The variance constant for adaptive path graphs is set to 0.00015.

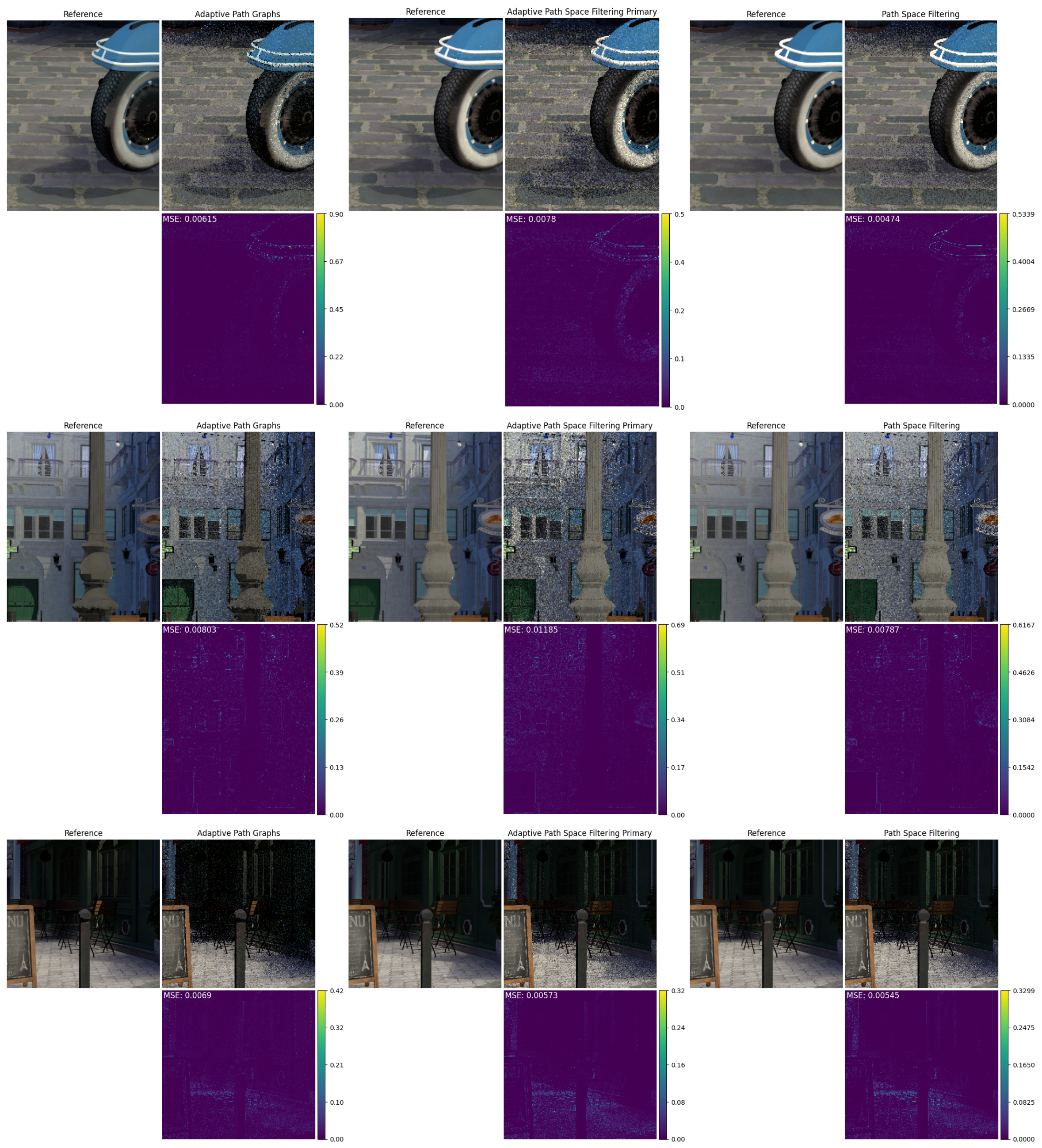


Figure 6.13: Variance comparison, the reference is generated by accumulating 1000 frames with the given technique. The variance constant for adaptive path graphs is set to 0.00093. The screen shots are taken by a moving camera.

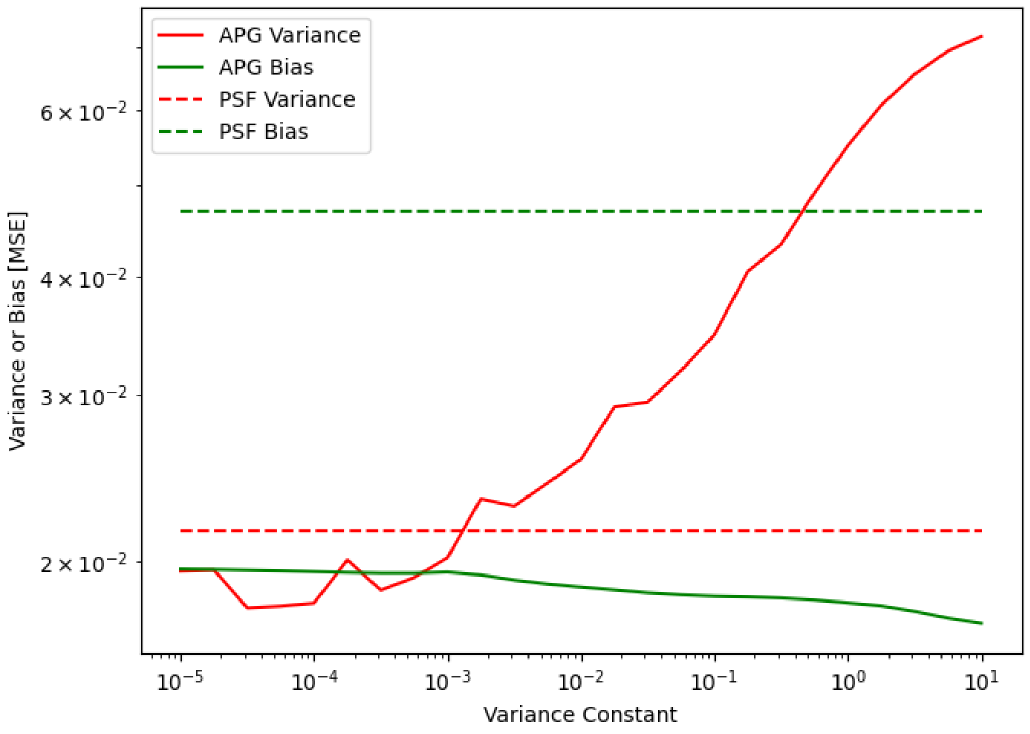


Figure 6.14: The variance and bias of an entire screen shot of the Bistro Interior scene calculated as discussed for different variance constants.

6.2 Frame Time

It is important to also compare the frame time, especially for interactive programs. In order to do that, Vulkan provides time stamp queries. It is therefore possible to generate precise measurements for frame timings of the different steps. The steps are discussed in section 5.1 and depicted in figure 5.1.

The graphs in figures 6.15 - 6.18 depict cumulative graphs and a comparison between the different techniques over 1000 frames. Note that the comparisons do not include the time needed for the ImGui and Denoiser step, as these two steps are optional.

All graphs have one thing in common, the ranking of the different techniques. Adaptive path space filtering for primary surfaces is the fastest technique, even faster than path tracing. The second place goes to path tracing, closely followed by path space filtering. Adaptive path graphs is the slowest technique and is improved by also utilizing adaptive path space filtering for primary surfaces.

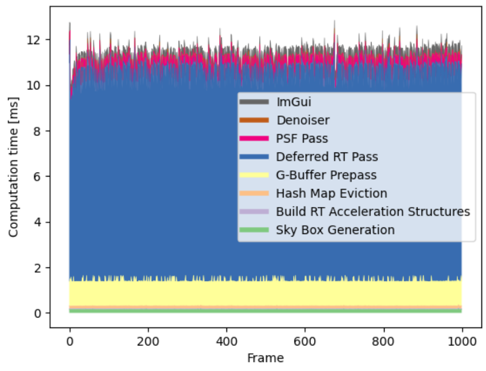
Looking at figure 6.15, using only two levels of indirection for 1920×1080 every technique remains interactive.

Figures 6.16 and 6.17b imply that the levels of indirection have a high impact on adaptive path graphs, which makes sense as it has to look into more spatial cells for more levels of indirection.

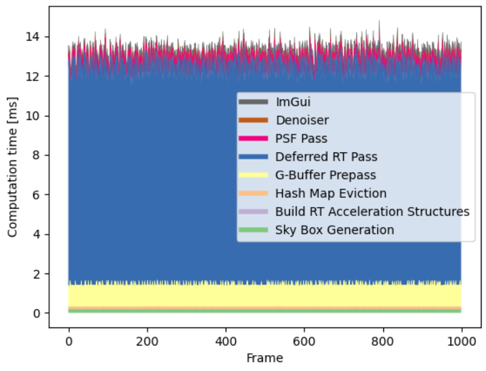
The techniques also behave rather similar for moving cameras as seen in figures 6.17a and 6.17b.

A higher resolution also leads to more hash map look-ups in the case of adaptive path graphs. The impact can be seen in figures 6.18a and 6.18b.

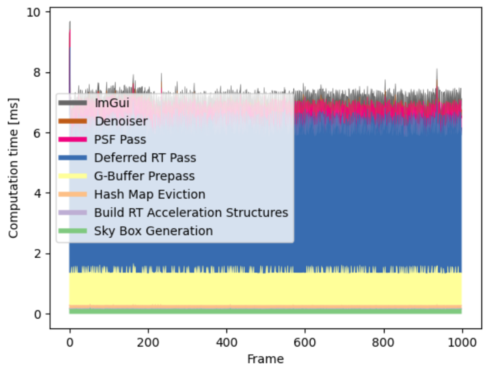
Adaptive path graphs is thus not suited to work well with too many levels of indirection or too high of a resolution. Adaptive path space filtering for primary surfaces on the other hand is a good technique to improve upon the frame time of path space filtering.



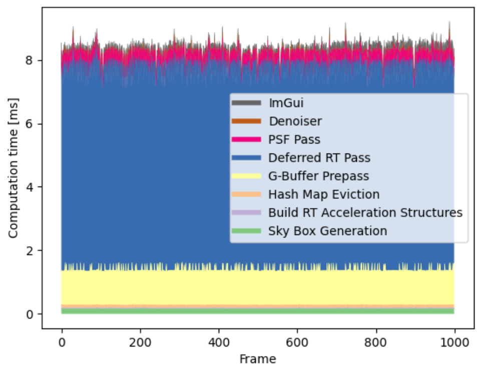
(a) Adaptive Path Graphs + Primary



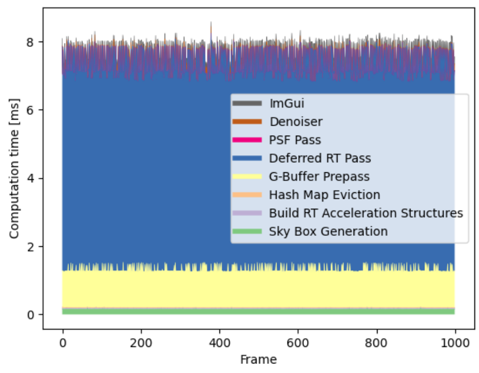
(b) Adaptive Path Graphs



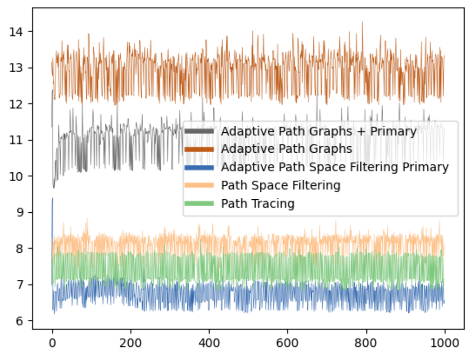
(c) Adaptive Path Space Filtering Primary



(d) Path Space Filtering

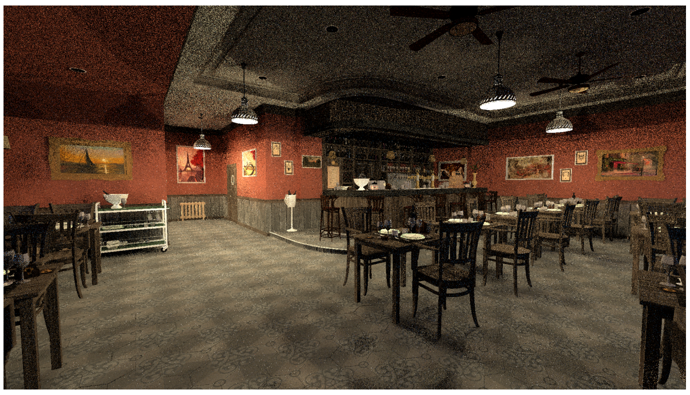


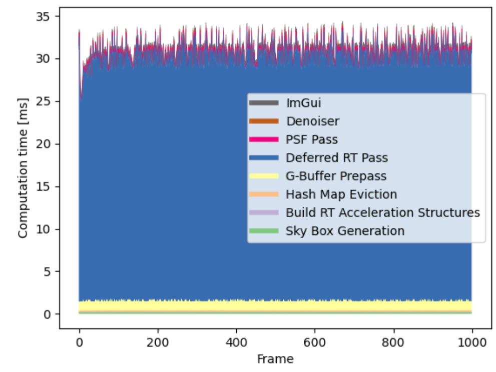
(e) Path Tracing



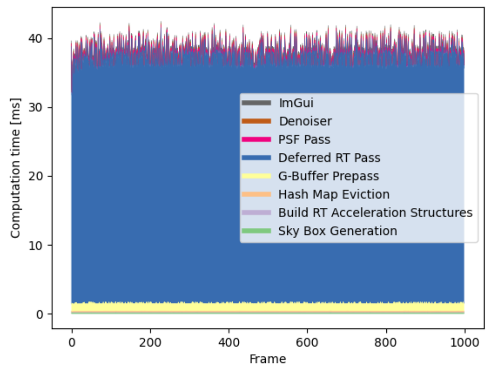
(f) Comparison

Figure 6.15: Bistro Interior, 2 levels of indirection, 1920 × 1080 resolution

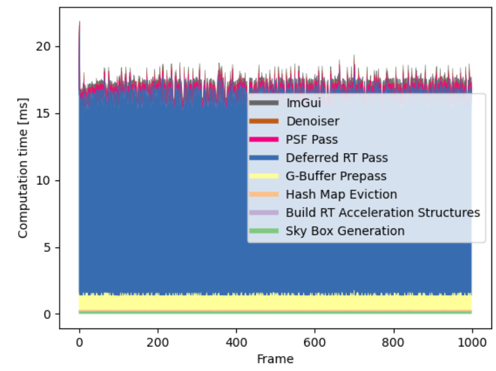




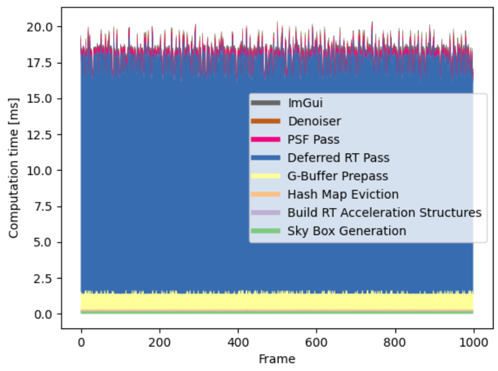
(a) Adaptive Path Graphs + Primary



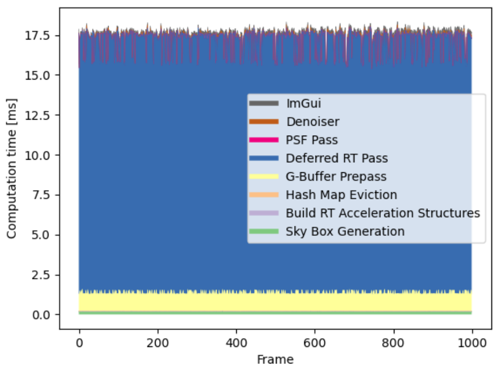
(b) Adaptive Path Graphs



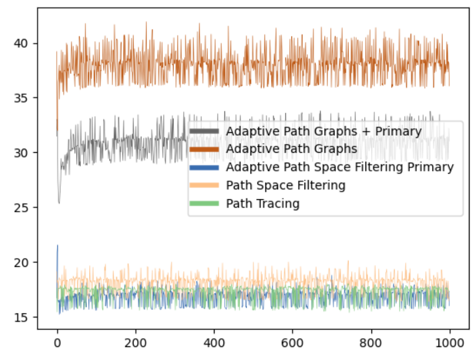
(c) Adaptive Path Space Filtering Primary



(d) Path Space Filtering

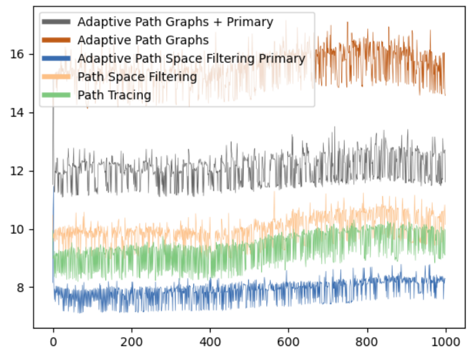


(e) Path Tracing

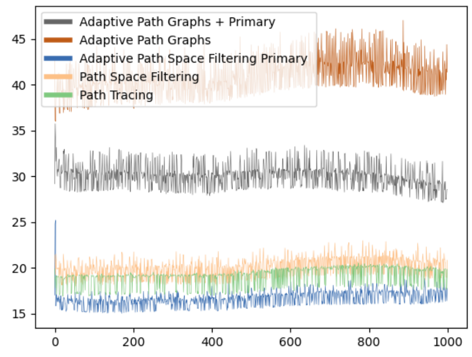


(f) Comparison

Figure 6.16: Bistro Interior, 10 levels of indirection, 1920 × 1080 resolution

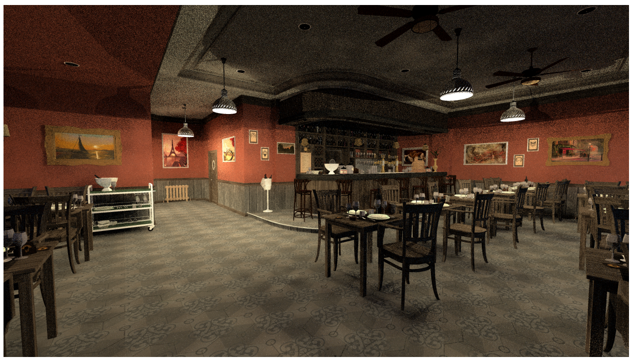

Adaptive Path Graphs + Primary
Adaptive Path Graphs
Adaptive Path Space Filtering Primary
Path Space Filtering
Path Tracing

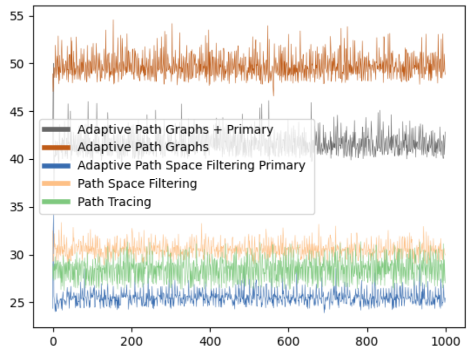
(a) 2 levels of indirection



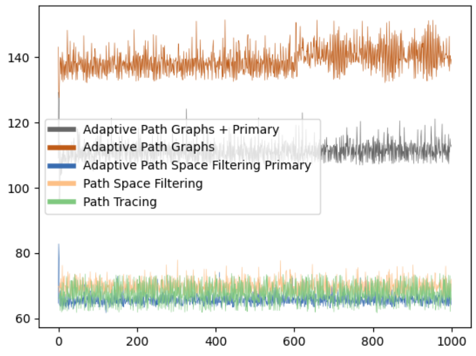
(b) 10 levels of indirection

Figure 6.17: Bistro Exterior, 1920 × 1080 resolution, moving camera




Adaptive Path Graphs + Primary
Adaptive Path Graphs
Adaptive Path Space Filtering Primary
Path Space Filtering
Path Tracing

(a) 2 levels of indirection



(b) 10 levels of indirection

Figure 6.18: Bistro Interior, 3840 × 2160 resolution

6.3 Adaptive Path Graphs: Path Termination Depth

The last analysis consists of measuring how the variance constant affects the path termination depth for path graphs. In order to analyze this, a scene with a light source that can only be seen after some levels of indirection is used. Figure 6.19 depicts such a scene. The resulting images and MSE are depicted in figure 6.20. The convergence of adaptive path graphs combined with adaptive path space filtering for primary surface can be seen in figure 6.21. It takes a little time for the rear spatial cells to reach the front spatial cells.

In order to see the path termination a little bit better, the jittering is deactivated. The path termination depth can be seen in figure 6.22. Plotting the average path termination depth against the variance constant yields the graphs in figure 6.23.

In real scenes it is hard to generate path termination depths other than 0 and 10, as the variance induced by the first surface hit is either so big that it already exceeds the variance constant or so low that the following path does not induce enough variance to cause an early termination.

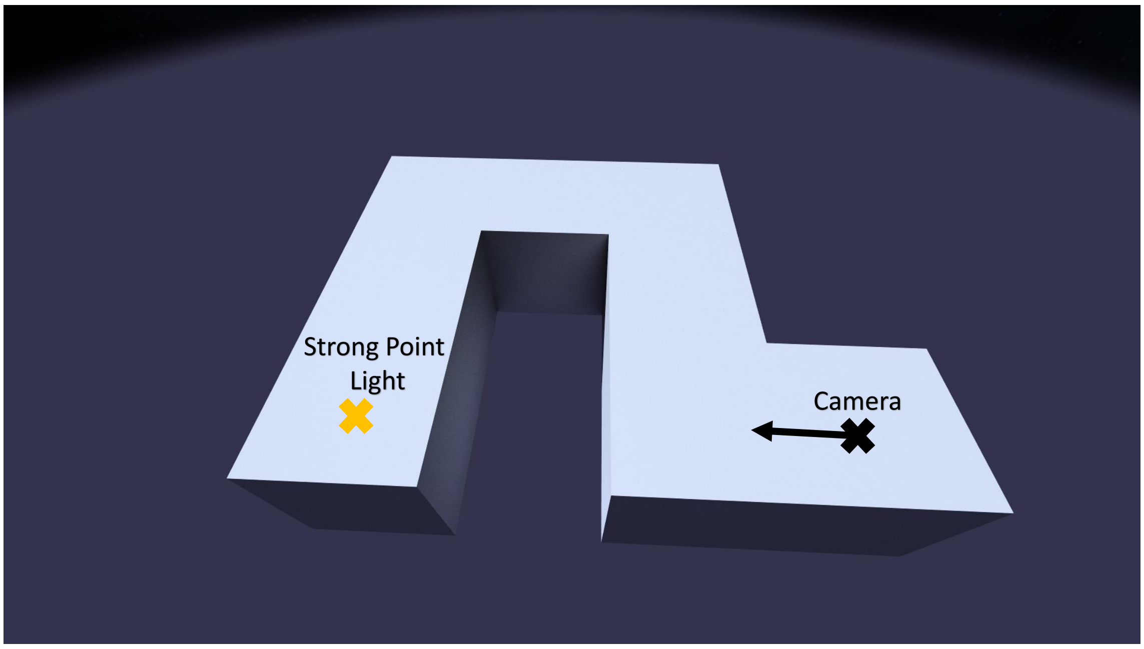


Figure 6.19: A corridor scene where the light can only be seen by using a higher level of indirection.

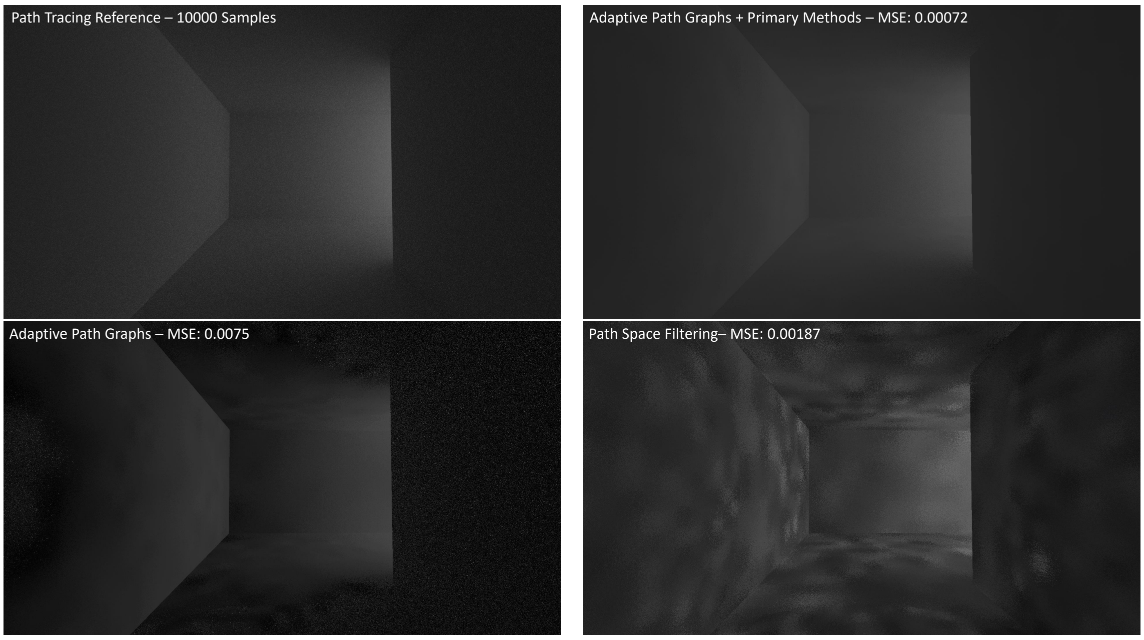

Path Tracing Reference – 10000 Samples
Adaptive Path Graphs + Primary Methods – MSE: 0.00072
Adaptive Path Graphs – MSE: 0.0075
Path Space Filtering– MSE: 0.00187

Figure 6.20: Comparison of techniques for the corridor scene from figure 6.19, a variance constant of 0.00163 is used.

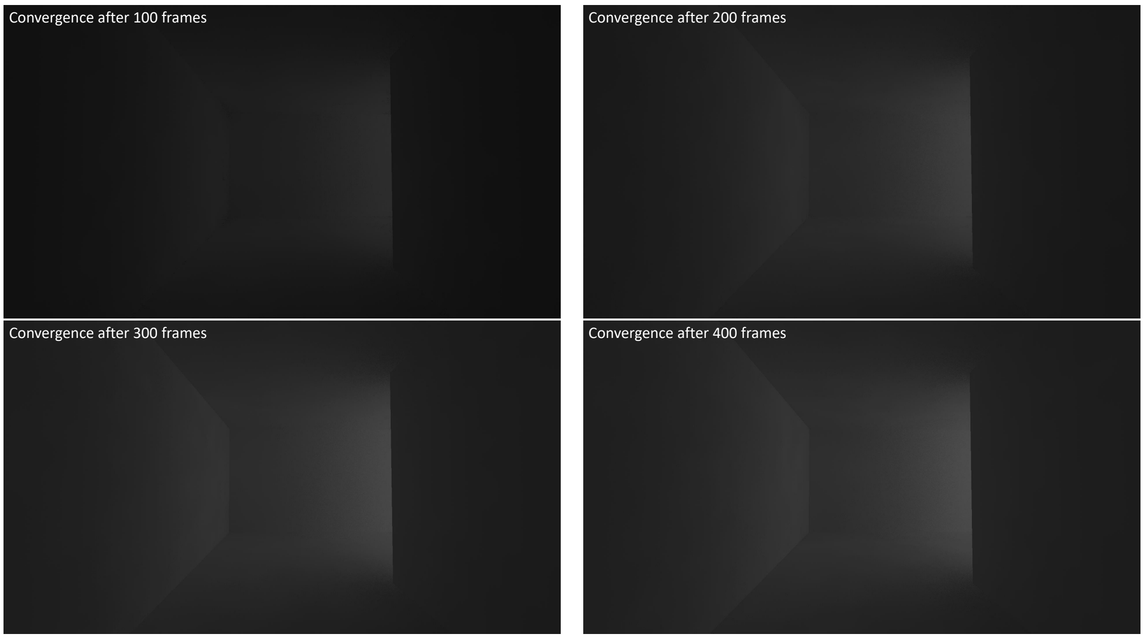

Convergence after 100 frames
Convergence after 200 frames
Convergence after 300 frames
Convergence after 400 frames

Figure 6.21: Convergence of adaptive path graphs combined with adaptive path space filtering for primary surfaces, for the corridor scene from figure 6.19 at one sample per pixel. A variance constant of 0.00163, a one function for va-interpolation and a smooth linear function for va path survival are used.

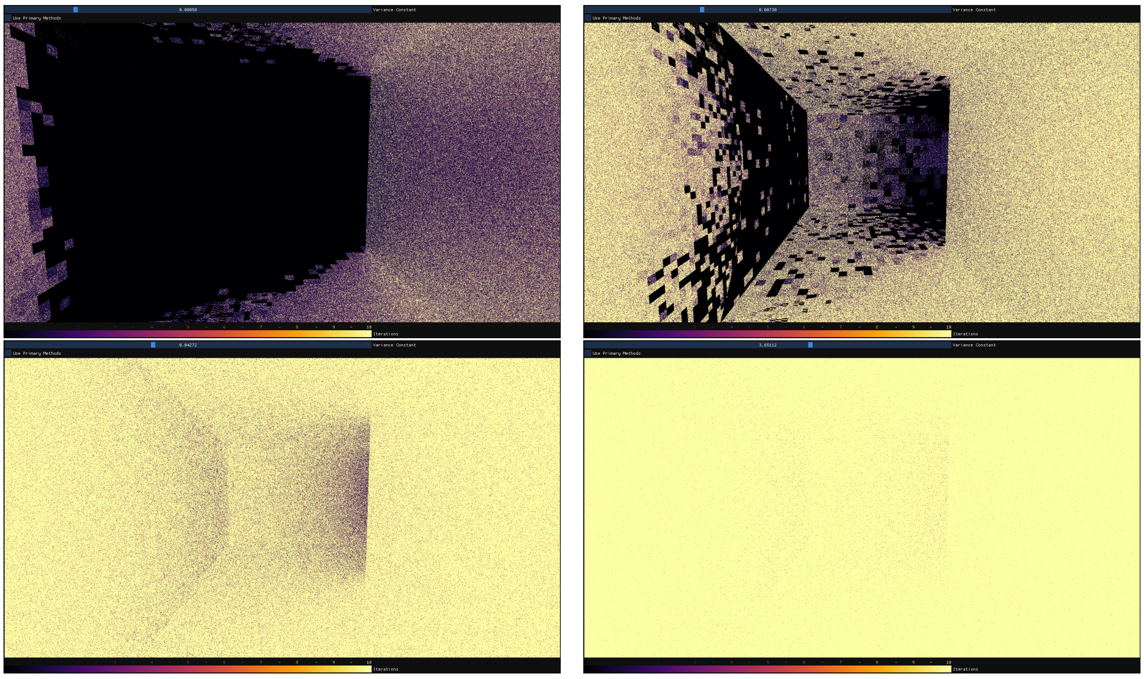


Figure 6.22: Path termination depth for adaptive path graphs given different variance constants.

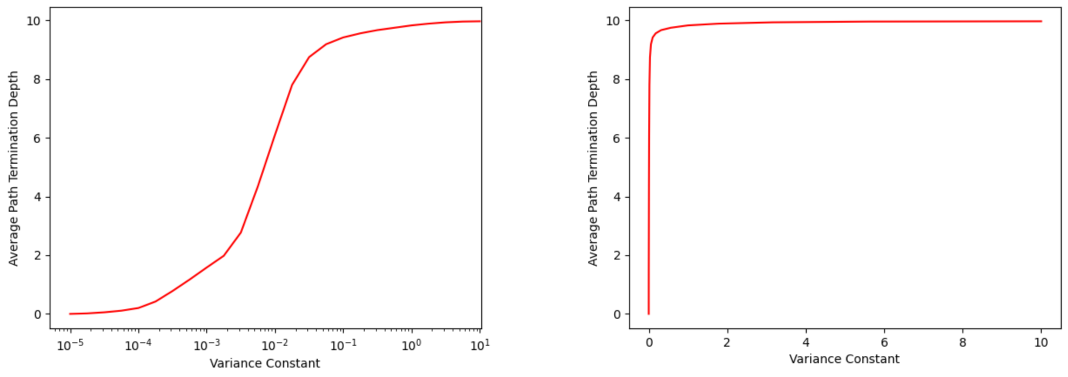


Figure 6.23: Average Path termination depth for adaptive path graphs for given variance constants.

7. Conclusions

As a conclusion, after analyzing the image quality and frame time, the techniques provided by this paper do a good job of allowing the user to control the variance-bias trade-off between path space filtering and path tracing. These adaptive techniques are easy to implement for a given implementation of path space filtering. They also run at interactive frame times and adaptive path space filtering for primary surfaces has even shown that it can surpass a path tracer. Another benefit is that an existing path tracer can easily be extended to use these techniques, which keeps the programming overhead low.

The techniques still suffer from some artifacts caused by path space filtering. Having a camera move through the world still causes the cache boundaries to become visible in some scenarios, fireflies can still generate very high spatial cell contributions and many more. But they do improve some scenes where the bias of path space filtering removes entire shadows as seen in the previous chapter.

Adaptive path space filtering for primary surfaces also provides a simple framework for variance based decisions other than spatial cell utilization or path survival. It could also be used to maybe introduce some splitting scheme or similar ideas.

Analyzing how other functions would influence the results of this technique or maybe learning these functions while rendering is work for future research.

Adaptive path graphs is still very computationally heavy, especially for high levels of indirection. Despite that, it provides promising results for some scenes, especially in combination with adaptive path space filtering for primary surfaces.

It would be interesting to see how these techniques would perform if the variance would not depend on the luminance of the pixels by using a similar scheme as Rath et al. [RGH+20]. One could for instance use the denoised image from the previous frame or a filtered version of that in order to normalize the variance such that it becomes independent of the luminance.

Looking forward to future research on these techniques, there is still head room for optimization.

8. Acknowledgments

First and foremost, I would like to thank Addis Dittebrandt for helping me write this thesis by creating ideas, giving me good feedback and always providing me with helpful comments on how to improve the implementation of the techniques and the framework of this thesis.

I also thank Christian Bender and Kilian Kraut for spell-checking my thesis.

Bibliography

[BFK18] N. Binder, S. Fricke, and A. Keller, "Fast path space filtering by jittered spatial hashing," in *ACM SIGGRAPH 2018 Talks*, ser. SIGGRAPH '18. New York, NY, USA: Association for Computing Machinery, 2018. [Online]. Available: https://doi.org/10.1145/3214745.3214806

[BFK19] N. Binder, S. Fricke, and A. Keller, "Massively parallel path space filtering," 2019. [Online]. Available: https://arxiv.org/abs/1902.05942

[BM97] M. R. Bolin and G. W. Meyer, "An error metric for monte carlo ray tracing," in *Proceedings of the Eurographics Workshop on Rendering Techniques '97*. Berlin, Heidelberg: Springer-Verlag, 1997, p. 57–68.

[DHC+21] X. Deng, M. Hašan, N. Carr, Z. Xu, and S. Marschner, "Path graphs: Iterative path space filtering," *ACM Trans. Graph.*, vol. 40, no. 6, dec 2021. [Online]. Available: https://doi.org/10.1145/3478513.3480547

[Kaj86] J. T. Kajiya, "The rendering equation," in *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '86. New York, NY, USA: Association for Computing Machinery, 1986, p. 143–150. [Online]. Available: https://doi.org/10.1145/15922.15902

[KDB16] A. Keller, K. Dahm, and N. Binder, *Path Space Filtering*, 01 2016, pp. 423–436.

[MRNK21] T. Müller, F. Rousselle, J. Novák, and A. Keller, "Real-time neural radiance caching for path tracing," *ACM Trans. Graph.*, vol. 40, no. 4, jul 2021. [Online]. Available: https://doi.org/10.1145/3450626.3459812

[RGH+20] A. Rath, P. Grittmann, S. Herholz, P. Vévoda, P. Slusallek, and J. Křivánek, "Variance-aware path guiding," *ACM Trans. Graph.*, vol. 39, no. 4, jul 2020. [Online]. Available: https://doi.org/10.1145/3386569.3392441

[Sch22] J. Schudeiske, "Photorealistic image synthesis lecture in the summer term 2022," 2022.

[VK16] J. Vorba and J. Křivánek, "Adjoint-driven russian roulette and splitting in light transport simulation," *ACM Trans. Graph.*, vol. 35, no. 4, jul 2016. [Online]. Available: https://doi.org/10.1145/2897824.2925912

Erklärung

Ich versichere, dass ich die Arbeit selbstständig verfasst habe und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, die wörtlich oder inhaltlich übernommenen Stellen als solche kenntlich gemacht und die Satzung des KIT zur Sicherung guter wissenschaftlicher Praxis in der jeweils gültigen Fassung beachtet habe. Die Arbeit wurde in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt und von dieser als Teil einer Prüfungsleistung angenommen.

Karlsruhe, den October 13, 2022

(Alexander Schipek)