國立臺灣大學電機資訊學院資訊工程學研究所

碩士論文

Department of Computer Science and Information Engineering
College of Electrical Engineering and Computer Science
National Taiwan University

Master's Thesis

用於蒙地卡羅渲染的雙歷史時空濾波方法：以變異數引導的時空濾波為例

Two-history Approach of Spatiotemporal Filtering for Monte Carlo Rendering: Spatiotemporal Variance-Guided Filtering as Example

涂家銘

Chia-Ming Tu

指導教授：陳炳宇 博士、簡韶逸 博士

Advisor: Bing-Yu Chen Ph.D, Shao-Yi Chien Ph.D.

中華民國 113 年 05 月

May 2024

doi:10.6342/NTU202400987

國立臺灣大學碩士學位論文
口試委員會審定書
MASTER'S THESIS ACCEPTANCE CERTIFICATE
NATIONAL TAIWAN UNIVERSITY

用於蒙地卡羅渲染的雙歷史時空濾波方法：以變異數引導的時空濾波為例

Two-history Approach of Spatiotemporal Filtering for Monte Carlo Rendering: Spatiotemporal Variance-Guided Filtering as Example

本論文係涂家銘君（學號 R09922077）在國立臺灣大學資訊工程學系完成之碩士學位論文，於民國 113 年 5 月 17 日承下列考試委員審查通過及口試及格，特此證明。

The undersigned, appointed by the Department of Computer Science and Information Engineering on 17 May 2024 have examined a Master's thesis entitled above presented by CHIA-MING TU (student ID: R09922077) candidate and hereby certify that it is worthy of acceptance.

口試委員 Oral examination committee:

陳炳宇 簡韶逸

（指導教授 Advisor）

莊永裕 張鈞法

系主任/所長 Director: 陳祝嵩

致謝

首先感謝簡韶逸教授在我離開原實驗室時給予協助，成為我最迷茫時的一道曙光，也感謝陳炳宇教授及簡韶逸教授願意收留並指導我。

感謝見臻科技的夥伴，在實習期間給予我很大的信任以及工作上的彈性，使我能在實習與研究之間取得平衡，並學到很多東西。

最後感謝我的家人在這漫長的研究生涯中支持我，父母在我需要全心投入研究時給予我經濟上的援助，還有哥哥提供寶貴的經驗讓我在匆忙的籌備口試及論文時有所依靠。

2024.05.23 涂家銘

doi:10.6342/NTU202400987

中文摘要

透過時空重用增加有效採樣數量對於實時光線追踪的降噪至關重要。現有的方法通常假設輸入是由固定的每像素樣本數生成的，並在時間重用當中使用指數移動平均來合併顏色。然而，當每像素樣本數變化時，這些方法無法有效利用由高樣本數生成的顏色。在這項研究中，我們提出了一種新方法，稱為「雙歷史方法」，以增強現有的時空濾波器，提升其處理可變樣本數輸入的能力。我們使用這種方法改進了變異數引導的時空濾波（SVGF）算法，並通過簡單的注視點渲染管線評估其性能。

doi:10.6342/NTU202400987

Abstract

Spatio-temporal reusing samples is critical to reducing noise for realtime ray-tracing. Existing methods usually assume inputs are generated by constant sample-per-pixel and use exponential moving averages to aggregate colors for temporal reuse. However, when sample-per-pixel vary, these approaches fail to effectively utilize colors generated by high sample counts. In this study, we proposed a novel method termed the ”Two-history Approach” to augment existing spatio-temporal denoisers, enhancing their ability to handle inputs with variable sample count. We adapt the SVGF algorithm using our approach and evaluate its performance with a simple foveated rendering pipeline.

doi:10.6342/NTU202400987

Contents

doi:10.6342/NTU202400987

doi:10.6342/NTU202400987

List of Figures

doi:10.6342/NTU202400987

List of Tables

doi:10.6342/NTU202400987

Chapter 1

Introduction

Monte Carlo base path-tracing (or ray-tracing) integrates illuminance by randomly sampling different directions from each hit point. Compared with rasterisation based method, path-tracing can handle complex reflection and refraction in a more physically accurate way. Although path-tracing can provide renders with better quality, usually a large number of samples per pixel (spp) is required to converge to a stable result, which is difficult to reach realtime.

Hardware supports accelerate path-tracing a lots, however even with a modern GPU, we can still only trace several paths per-pixel at interactive frame rate. To address the limitation, previous research has developed many denoising algorithms tailored for scenarios with very few samples per pixel. Most of these denoisers reuse samples both spatially and temporally, i.e. spatio-temporal filtering, to increase effective sample counts.

Figure 1.1 illustrate a high-level common structure of spatio-temporal filters. In a spatio-temporal filter, we usually apply temporal filter before spatial filter. In temporal filters, a re-projection step (usually backward projection) is required to map current samples to history samples. To stabilize the temporal filter faster, it's common to feedback spatial filtered results into temporal filter as history. For spatial filter with multiple iterations, we may feedback result filtered by only first few layers of the iterations to prevent over-blurring.

doi:10.6342/NTU202400987

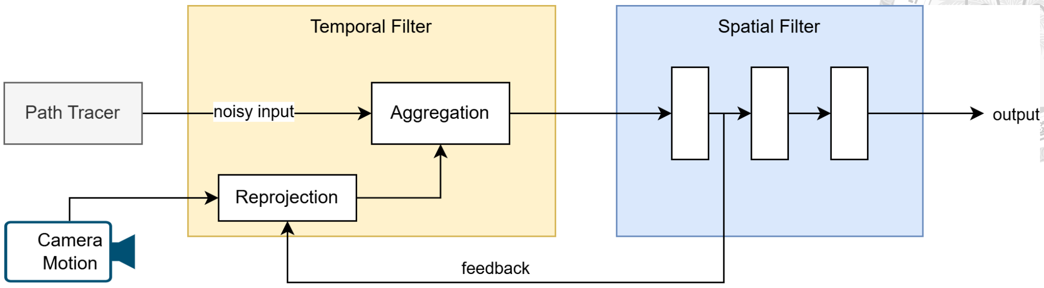

Figure 1.1: Common structure of spatio-temporal filters.

Although many powerful filters has been developed, when it comes to head-mounted devices such as a VR headset, realtime path-tracing are still challenging because rendering stereo view requires higher resolution and higher frame rates are required to prevent motion sickness. Fortunately, head-mounted devices are suitable for near distance eye-tracking, which makes foveated rendering an attractive solution. Foveated rendering spends more computing resources at foveal the area, where the user's gaze is focused, while conserving resources in the peripheral areas. This strategy works because human visual system only has higher resolution in a pretty small area around visual center. When applying to path-tracing, since path-tracing naturally support controlling sample count for each pixel by a sample map, a simplest foveated path-tracing algorithm would be taking more samples in the foveal region by manipulating the sample map.

However we found some problems might emerge when trying to combine foveated rendering (or other method that varies sample counts) with existing denoisers that contain spatio-temporal reuse. Ideally, to minimize variance of accumulated color, we should weight the colors by number of samples they took. But many denoisers for realtime path-tracing assumes constant spp and accumulate colors by exponential moving average without considering number of samples each color took, i.e:

$$x'_t = \beta x'_{t-1} + (1-\beta)x_t = \frac{\sum_{i=1}^{t} \beta^{t-i} x_i}{\sum_{i=1}^{t} \beta^{t-i}} \tag{1.1}$$

where β is the decay rate.

One may ask why not just multiply the weights by number of samples? Indeed

doi:10.6342/NTU202400987

we can account number of sample by modifying the formula to:

$$x'_t = \frac{\sum_{i=1}^{t} \beta^{t-i} x_i \cdot n_i}{\sum_{i=1}^{t} \beta^{t-i} \cdot n_i} \tag{1.2}$$

where n_i is the sample count at frame i. However, we found this method yields two problems:

1. Sometimes it is worse than original method.
2. Temporal bias between different pixels may differ.

Below we will see examples of the two problems. For convenience, we call formula (1.1) the **unweighted method**, and formula (1.2) the **weighed method** in this paper.

Figure 1.2 shows an example that weighted method can be worse than unweighted method. Consider $\beta = 0.5$ and the pixel has 1 frame of history (excluding current frame). In the first frame, the pixel was in peripheral area, taking 1 sample to generate the color. In the second frame, the pixel entered foveal area, taking 2 samples to generate the color. The un-normalized weights of unweighted and weighted method at frame i are denoted as w_i and $\tilde{w}_i$ respectively. As we can see in the figure, w_i is proportional to sample count n_i, which produces result with minimal variance, while $\tilde{w}_i$ is not.

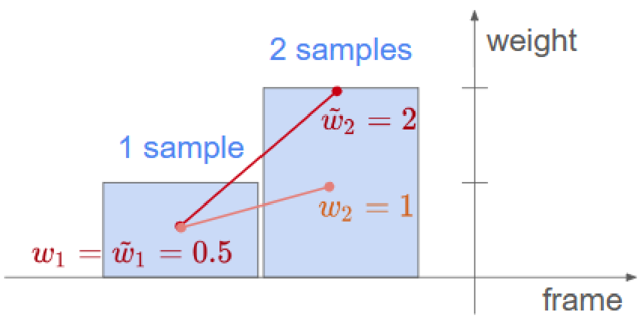

Figure 1.2: Example of weighted method being worse then unweighted method.

The second problem emerges when illuminance changes fast and n_i changes differently between pixels. Figure 1.3 shows an example that foveal area darken

doi:10.6342/NTU202400987

faster than peripheral area. In the example, the scene is darkening, and the foveal area was moving from bottom-left to upper-right. Pixels in foveal area has larger weights for recent frames, while pixels in peripheral rely more on old frames.

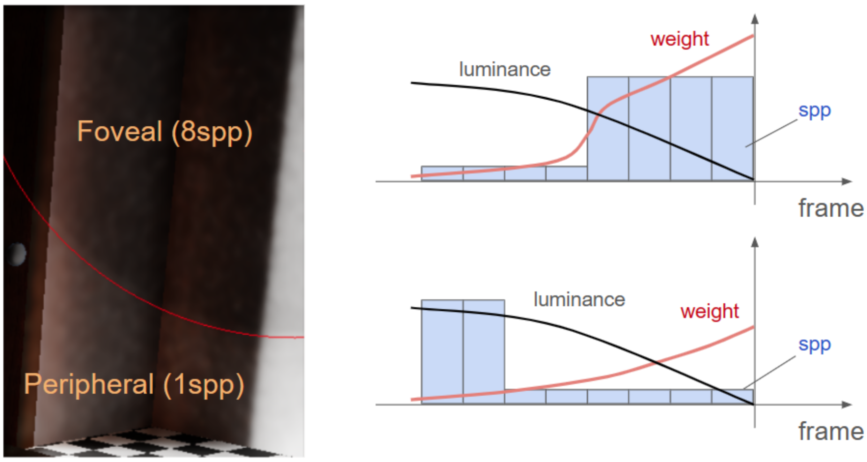

Figure 1.3: Example of difference of temporal bias.

In this paper, we aim to avoid the problems of weighted method by falling back to unweighted method whenever weighted method performs bad. We maintain separate histories for both unweighted and weighted method and blend them adaptively using information derived from sample counts and estimated gradient.

doi:10.6342/NTU202400987

Chapter 2

Related Works

2.1 Spatio-temporal Reuse

Although hardware support for ray-tracing has became more common in recent years, we can still only trace a few or even only 1 sample per pixel to achieve realtime. Thus many denoisers for very-few-sample path-tracing has emerged. In this paper we only focus on those with spatio-temporal reuse.

Schied et al. [1] proposed Spatio-temporal Variance-Guided Filtering (SVGF) for denoising 1-spp path-traced results. SVGF uses variance estimated in temporal filtering pass to guide parameters of spatial filter. In the temporal filtering part, simple exponential moving average $x'_t = (1-\alpha)\cdot x_{t-1} + \alpha \cdot x_t$ is used to invalidate old samples. For spatial filter, they use the edge avoiding À-trous wavelet transform developed by Dammertz et al. [2] to form hierarchical filters.

To achieve temporal stability, a small α is required ($\alpha = 0.05$ was chosen by [1]). However, small α may yields temporal over-blurring or delay. To reduce temporal over-blurring, Schield et al. [3] proposed A-SVGF that adaptively change α base on per-pixel gradient estimation. The larger the gradient is, the faster the algorithm invalidate old samples (i.e. larger α).

Bitterli et al. [4] proposed ReSTIR, a resampling based algorithm to path-tracing direct lighting with millions of light sources. The algorithm was later

doi:10.6342/NTU202400987

extended to world-space reuse [5; 6], volume rendering [7], and full path-tracing [8] (ReSTIR-PT).

ReSTIR also reuse sample both spatially and temporally to reduce noise. Similar to SVGF, they invalidated old samples by exponentially decay but in the sense of probability. Some other algorithms in the ReSTIR family like [6] determine whether to drop old samples by spatially down-sampled re-tracing, which is similar to A-SVGF.

All the methods mentioned above, either explicitly or implicitly, assume that the number of samples per pixel remains constant over screen space. Although algorithms like ReSTIR take multiple samples per pixel into consideration, they are similar to the weighted method mentioned in introduction section, which means changing sample count arbitrary for each pixel may cause inconsistent temporal bias.

The concept of spatio-temporal filtering can also be found in neural network based works. Kuznetsov et al. [9] introduced a CNN based method to do adaptive sampling and denoising using two networks. CNN naturally has functionality of spatial filtering but lack of temporal information. To enable temporal reuse, Chaitanya et al. [10] proposed to add recurrent links. Recurrent structures makes the network being able to retrieve temporal information. However, when the camera moves fast, we may need larger networks to gather information from re-projected position, which make it difficult to reduce model size. Hasselgren et al. [11] optimized this by explicitly implementing re-projection.

2.2 Foveated Rendering

To make ray-tracing realtime on head-mounted devices, Weier et al. [12] use both foveated rendering and temporal re-projection to reduce required number of samples by . Kim et al. [13] combined selective adaptive super sampling developed by Jin et al. [14] with temporal reuse and foveated rendering to produce image with

doi:10.6342/NTU202400987

up to effective 36 spp in foveal area and gradually reduce to 1 spp in peripheral.

Although these ray-tracing algorithms already combines foveated rendering and temporal reuse, they focus on reducing number of samples base on gaze and re-projected information. While our work aim to improve image quality without controlling sampling method.

Foveated rendering is a large topic, we only list a few of the works that are more relative to our research. For more about foveated rendering, Wang et al. [15] has made an exhaustive survey.

2.3 Temporal Gradients

Temporal gradients have been used to guide various spatio-temporal filtering algorithms.

Schield et al. [3], which is also mentioned in section 2.1, proposed a method to estimate temporal gradients for path-traced inputs. By apply their gradient estimation method on SVGF [1] and RAE [16] with additional temporal filter, they developed A-SVGF and A-RAE. In both of the applications, gradients are used to control decay rate of exponential moving average. To estimate gradients with higher coherence between consequent frames, they calculate additional samples using forward-projected random seed from previous frame. Since generating additional samples is costly, they calculate gradient sparsely and then reconstruct it to dense gradient map.

Although [3] works well, their implementation is somehow complex, posing difficulty to integrate with existing rendering pipelines. However such complexity is required because over estimating gradient is fatal to the applications. Whenever we drop old samples by applying an α larger than necessary due to noise, the old samples can never be restored.

Mueller et al. [17] takes the concept of guiding temporal reuse by gradient to adaptive shading (non ray-traced). They approximate gradient to estimate when

doi:10.6342/NTU202400987

will color change being large enough to be sensitive and re-shade only if that happens.

In this paper we also use temporal gradient to guide our denoiser. Similar to [17], we use gradient to estimate whether color changes are large enough to be sensible. Although we are handling path-traced input, our method won't invalidate more old samples forever due to over estimating gradient like [3], so we choose to apply a more simpler gradient estimation algorithm.

2.4 SVGF

In this section we briefly introduce the SVGF algorithm developed by Schied et al. [1] which is the example we will use in chapter 4. Figure 2.1 shows the high-level structure of SVGF.

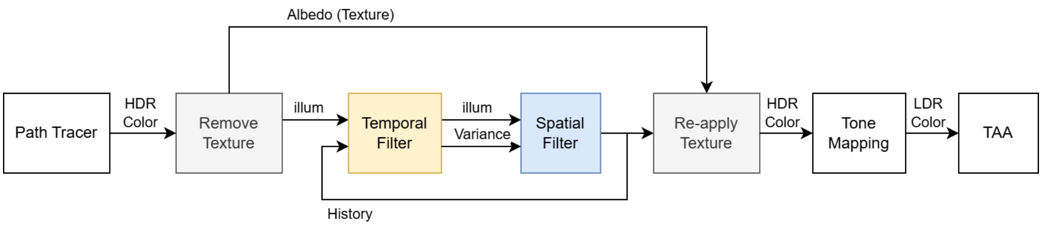

Figure 2.1: High-level structure of SVGF

To prevent blurring textures, they first remove textures by dividing the texture colors and re-apply (multiply) them after spatio-temporal filtering.

In the temporal filter pass, they use exponential moving average to integrate colors, i.e.:

$$x'_t = \beta^{t-i} x'_{i-1} + (1 - \beta) x_t$$

Here we re-describe the formula using $\beta = (1 - \alpha)$.

Since a large β is used (0.95 by the original paper) to increase temporal stability, the resulting x'_t might rely too much on the first frame after dis-occlusion, which yields larger variance. To deal with this problem, they use simple average in the

doi:10.6342/NTU202400987

first few frames. Thus we may re-write the formula in a more general form:

$$x'_t = \beta_i{}^{t-i} x'_{i-1} + (1 - \beta_i) x_t$$

That is, each frame may has it's own β value. To do simple average in the first few frame, they let $\beta_i = \max\left(\frac{i-1}{i}, \beta_0\right)$, where β_0 is the chosen base decay rate.

The spatial filter consist of multiple iterations basis on à-trous wavelet transform. Each iteration increase the effective footprint (area of reuse) width by an order. Figure 2.2 shows an 1-d example of the hierarchy.

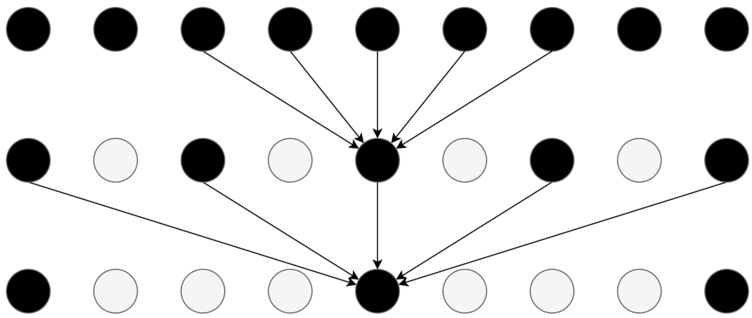

Figure 2.2: An 1-d example of à-trous wavelet transform.

The algorithms guides spatial filter using temporal variance estimation calculated in temporal filter pass. The larger the temporal variance is, the more aggressive the spatial filter would be. To estimate the temporal variance, the raw moments of illuminance are calculated in temporal filter pass using exponential moving average. Formally,

$$\mu_1(t) = \beta\mu_1(t-1) + (1-\beta)x_i$$
$$\mu_2(t) = \beta\mu_2(t-1) + (1-\beta)x_i^2$$

, where the decay rate β is recommended to be a smaller value so as to response faster to illumination changes. Then the variance estimation would be $\mu_2 - \mu_1^2$. Similar to filtering illuminance temporally, the accumulated moments might be too noisy when history is short. So they filter the moments spatially with a 7x7 bilateral filter when the history is short (< 4 frames).

doi:10.6342/NTU202400987

After re-applying texture, the color would be tone-mapped to low dynamic range. Finally, a temporal anti-aliasing (TAA) [18; 19] pass is used to clean-up remaining noise.

Chapter 3

Two-history Approach

3.1 Overview

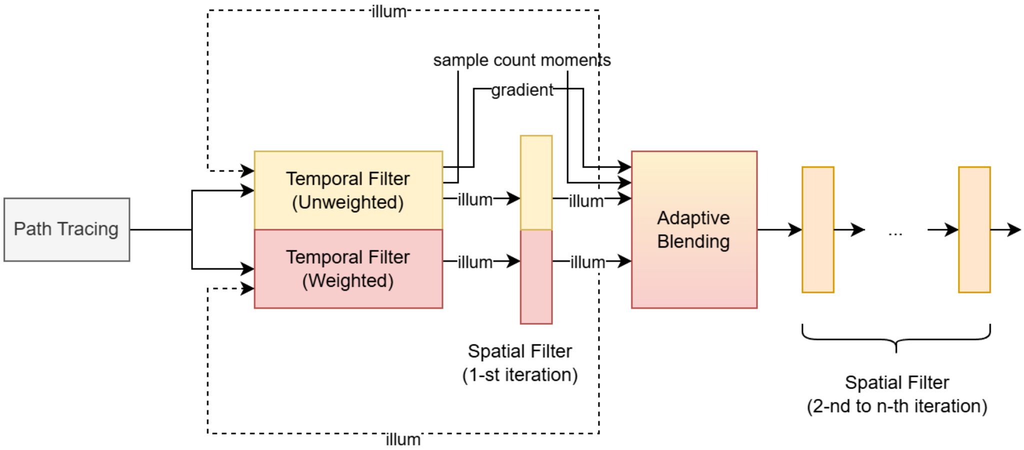

Figure 3.1: Overview of our method

Given a spatio-temporal filter, our method duplicate the temporal filter and one iteration of spatial filter to maintain two histories I_u and I_w, produced by unweighted and weighted method respectively. The accumulation results will be blended together by linear interpolation $I_b = \text{lerp}(I_w, I_u, r)$, where lerp is a linear interpolation function that $\text{lerp}(a, b, r) = (1 - r) \cdot a + r \cdot b$. The blending ratio r is computed separately for each pixel of each frame.

The idea of our approach is to rely more on unweighted method whenever any

doi:10.6342/NTU202400987

of the two problems might appear in weighted method, i.e. when (1) weighted method might be noisier than unweighted method, or (2) temporal bias of weighted method is significant.

We conceptually blend the histories in two phases to deal with the two problems respectively. In the first phase, we find a optimal blending factor p that minimize variance of blended result:

$$I_{opt} = \text{lerp}(I_w, I_u, p). \tag{3.1}$$

In the second phase, we blend I_{opt} with I_u using linear interpolation, formally:

$$I_b = \text{lerp}(I_{opt}, I_u, q), \tag{3.2}$$

where q is correlated to magnitude of luminance change.

Then equation 3.1 and 3.2 can be combined together:

$$\begin{aligned} I_b &= \text{lerp}(I_w, I_u, \text{lerp}(q, 1, p)) \\ \Rightarrow r &= \text{lerp}(q, 1, p) \end{aligned}$$

In section 3.2 and 3.3 we will explain details of each phase of blending with only considering temporal filters. In 3.4, we will put spatial filter into the pipeline and discuss why we put the blending pass after first iteration of spatial filter.

3.2 First Phase of Blending

The purpose of first phase of blending is to prevent using I_w when it is noisier than I_u as shown by the sample in figure 1.2. Instead of using I_u directly when I_w is bad, we found that blending them together with correct ratio may further reduce noise. Figure 3.2 shows an example that blended result may has smaller variance.

To decide the best blending ratio p, we have to solve the following minimization problem:

$$\begin{aligned} \underset{p}{\text{minimize}} \quad & \text{Var}[\text{lerp}(I_w, I_u, p)] \\ \text{subject to} \quad & 0 \le p \le 1 \end{aligned}$$

doi:10.6342/NTU202400987

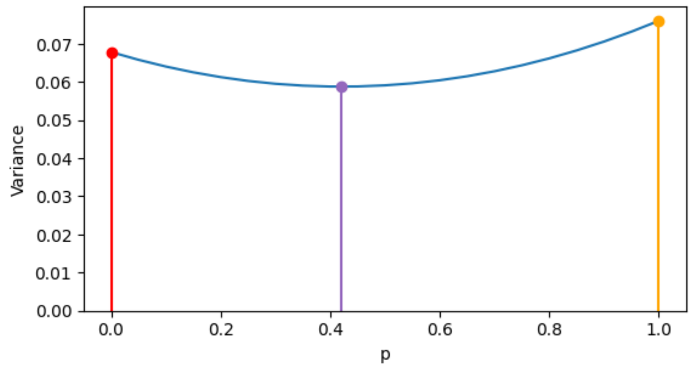

Figure 3.2: **Blended result may has smaller variance.** $p = 0.0$ (red) means fully relying on I_w and $p = 1.0$ means fully relying on I_u.

Note that the variance comes from random sampling and does not effected by illuminance change over time.

In the rest of this section, we describe how we solve the problem. Readers less interested in the proof may skip to equation 3.3 for the final formula.

To simplify the proof, only a single pixel without camera motion is considered. We start from formalizing the accumulated colors as random variables. For each frame i, let X_i be the random variable of taking one sample, n_i be the number of samples to take, and $\bar{X}_i$ be the random variable of averaging n_i samples taken from X_i. Then the accumulated colors I_u and I_w can be written as:

$$I_u = \frac{\sum_{i=1}^{t} B_i \bar{X}_i}{\sum_{i=1}^{t} B_i} = \frac{\sum_{i=1}^{t} B_i \bar{X}_i}{W_u}$$
$$I_w = \frac{\sum_{i=1}^{t} B_i n_i \bar{X}_i}{\sum_{i=1}^{t} B_i n_i} = \frac{\sum_{i=1}^{t} B_i n_i \bar{X}_i}{W_w}$$

where t is the number of frames after dis-occlusion and $B_i = \prod_{j=i+1}^{t} \beta_j$ is the product of decay rates applied to color of frame i. $W_u = \sum_i^t B_i$ and $W_w = \sum_i^t B_i n_i$ are un-normalized total weight of unweighted and weighted history respectively.

Next, we find the unconstrained extreme value by taking derivative. Below we

doi:10.6342/NTU202400987

denote $V_i = \text{Var}[X_i]$.

$$\begin{aligned}
&\frac{d}{dp}\text{Var}[\text{lerp}(I_w, I_u, p)] = 0\\
&\Rightarrow \frac{d}{dp}\sum_{i=1}^{t}\left((1-p)\frac{B_i n_i}{W_w} + p\frac{B_i}{W_u}\right)^2 \frac{1}{n_i}V_i = 0\\
&\Rightarrow 2\sum_{i=1}^{t}\left((1-p)\frac{B_i n_i}{W_w} + p\frac{B_i}{W_u}\right)\left(\frac{B_i}{W_u} - \frac{B_i n_i}{W_w}\right)\frac{1}{n_i}V_i = 0\\
&\Rightarrow \sum_{i=1}^{t}\left(W_u(1-p)B_i n_i + W_w p B_i\right)\left(W_w B_i - W_u B_i n_i\right)\frac{1}{n_i}V_i = 0\\
&\Rightarrow p = \frac{W_u^2\sum_{i=1}^{t}B_i^2 n_i V_i - W_w W_u \sum_{i=1}^{t}B_i^2 V_i}{W_u^2\sum_{i=1}^{t}B_i^2 n_i V_i - 2W_w W_u\sum_{i=1}^{t}B_i^2 V_i + W_w^2\sum_{i=1}^{t}B_i^2 n_i^{-1}V_i}
\end{aligned}$$

To simplify the formula, we define $S_k = \sum_{i=1}^{t} B_i^2 n_i^k V_i$, then the formula can be written as:

$$p = \frac{W_u^2 S_1 - W_w W_u S_0}{W_u^2 S_1 - 2W_w W_u S_0 + W_w^2 S_{-1}}$$

In order to get the exact solution, we have to know V_i for each frame i. Since it's difficult to know exact value of V_i in most of the cases, we may instead use an estimator $\tilde{V}_i$ to approximate it. In this paper, we let $\tilde{V}_i$ be constant for simplicity and efficiency, i.e. $\tilde{V}_i = \tilde{V}_j$ for all i, j. With this assumption the equation can be rewritten to:

$$\begin{aligned}
p &= \frac{W_u^2 \tilde{S}_1 - W_w W_u \tilde{S}_0}{W_u^2 \tilde{S}_1 - 2W_w W_u \tilde{S}_0 + W_w^2 \tilde{S}_{-1}}\\
\text{where} \quad \tilde{S}_k &= \sum_{i=1}^{t} B_i^2 n_i^k
\end{aligned}$$

To prove the extreme value is actually a minimum, we show that $\text{Var}[\text{lerp}(I_w, I_u, p)]$

is a quadratic function of p with the coefficient of p^2 being non-negative.

$$\begin{aligned}\text{Var}[\text{lerp}(I_w, I_u, p)] &= \sum_{i=1}^{t}\left((1-p)\frac{B_i n_i}{W_w} + p\frac{B_i}{W_u}\right)^2 \frac{1}{n_i}V_i \\ &= \sum_{i=1}^{t}\left(\frac{B_i n_i}{W_w} + p\left(\frac{B_i}{W_u} - \frac{B_i n_i}{W_w}\right)\right)^2 \frac{1}{n_i}V_i\end{aligned}$$

Then the coefficient of p^2 would be $\sum_{i=1}^{t}\left(\frac{B_i}{W_u} - \frac{B_i n_i}{W_w}\right)^2 \frac{1}{n}V_i$. By definition, both n_i and V_i must be non-negative, thus the coefficient of p^2 is also non-negative.

Finally, combining with constrained solutions, we get the final formula of blending ratio:

$$p = \text{clamp}\left(\frac{W_u^2\tilde{S}_1 - W_w W_u \tilde{S}_0}{W_u^2\tilde{S}_1 - 2W_w W_u \tilde{S}_0 + W_w^2 \tilde{S}_{-1}}, 0, 1\right) \tag{3.3}$$

From the perspective of implementation, all of W_u, W_w and $\tilde{S}_k$ can be calculated through temporal accumulation. When combining these components into p, we can rearrange the terms by letting $C = W_u(W_u\tilde{S}_1 - W_w\tilde{S}_0)$ and $D = W_w(W_u\tilde{S}_0 - W_w\tilde{S}_{-1})$ to reduce the number of multiplication operations. The formula after rearranging would be $p = \frac{C}{C-D}$.

3.3 Second Phase of Blending

In the second phase of blending, we try to rely less on I_w when temporal bias difference between pixels might be significant. Since difference in temporal bias can only be perceived if illuminance changes, we use more I_u when illuminance change is larger.

The blending formula is then designed as follow:

$$\begin{aligned}I_b &= (1-q)I_{opt} + qI_u \\ q &= \text{clamp}\left(S \cdot \frac{\text{lum}(\text{abs}(g))}{\tilde{\sigma}}, 0, 1\right)\end{aligned}$$

where g is the approximated temporal gradient, $\text{abs}(\cdot)$ calculates element-wise absolute value, S is a scaling factor, $\text{lum}(\cdot)$ is a function mapping RGB color to

doi:10.6342/NTU202400987

luminance, and $\tilde{\sigma}$ is the estimated standard deviation. We will explain the details below.

We start from approximating temporal gradient to estimate how fast the illumination change. To calculate gradient estimation g, we temporally accumulate $x_i - x_{i-1}$ using exponential moving average, and then apply an edge aware bilateral filter that is similar to a single layer of spatio-filter in SVGF. Since the gradients may have negative elements, we take the element-wise absolute value of gradients. Then to squash three channels into one, we apply the luminance function lum, which is a linear function mapping high dynamic range RGB colors to relative luminance.

Then to normalize the luminance change, we divide $\text{lum}(\text{abs}(g))$ by $\tilde{\sigma}$. $\tilde{\sigma}$ is the weighted standard deviation of $\text{lum}(x_i)$ calculated by exponential moving average, formally:

$$\tilde{\sigma} = \sqrt{\frac{\sum_{i=1}^{t} \text{lum}(x_i)^2 B_i}{\sum_{i=1}^{t} B_i} - \left(\frac{\sum_{i=1}^{t} \text{lum}(x_i) B_i}{\sum_{i=1}^{t} B_i}\right)^2}$$

$$\text{where} \quad B_i = \prod_{j=i+1}^{t} \beta_j$$

Note that unlike the variance in first phase of blending which estimate variance of random variable X_i, this standard deviation is calculated from actual colors x_i we have generated.

The normalization factor has these good properties:

- When we scale the inputs x_i, $\frac{\text{lum}(\text{abs}(g))}{\tilde{\sigma}}$ remains unchanged, which is an essential property of relative change.
- The noisier the input is, the larger the $\tilde{\sigma}$ would be. This property make I_b tends to use I_{opt} (which might be less noisier) when the input is more noisy.

The parameter S is a scaling factor that can be controlled by developers. If we know the scene won't have significant luminance change (including those of glossy reflection), than we can set S to a smaller value to better reducing noise.

doi:10.6342/NTU202400987

Conversely, if the scene may has drastic luminance change, than we can set S to a larger value to against temporal bias difference. We found that $S = 1$ make a good balance among the scenes we tested.

Lastly, we clamp the value to $[0, 1]$ range, then we get the final blending ratio q.

3.4 Combine with Spatial Filters

Previously we have only considered temporal filters in sections. In this section we will discuss how to put spatial filter into the pipeline.

The most intuitive way might be concatenating spatial filter after the blending pass directly. However, we can also run the spatial filter for two histories separately and then blend the histories after spatial filter. If the spatial filter consist of multiple iterations, it's also possible to place blending pass after arbitrary number of iterations of spatial filter.

Here we choose to place blending after the first iteration and put at least one iteration of spatial filter after blending if possible. We will discuss the reasons bellow.

Why don't we place the blending pass before the first iteration? As described in introduction, spatio-temporal filters usually feedback color denoised by some layers of spatial filters, so as to expand the reusing area when the history length growth and also stabilize the temporal filter faster. If we apply blending before any spatio-filtering, then it's not possible to feedback spatially filtered result into the two histories because blending breaks the properties of unweighted and weighted history. To maintain the properties of each history while feeding back result filtered by at least one iteration of spatial filter, the blending pass should be placed after some layers of spatial filter.

Why don't we place the blending pass after more iterations? Since we need to duplicate all layers of spatial filters before blending, the more iterations of spatial filter are before blending, the more computing time would be spent. Thus

doi:10.6342/NTU202400987

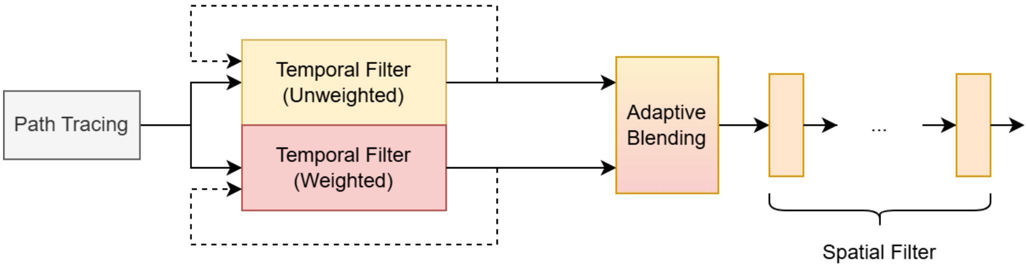

Figure 3.3: Blending before spatial filter.

we put the blending pass as early as possible.

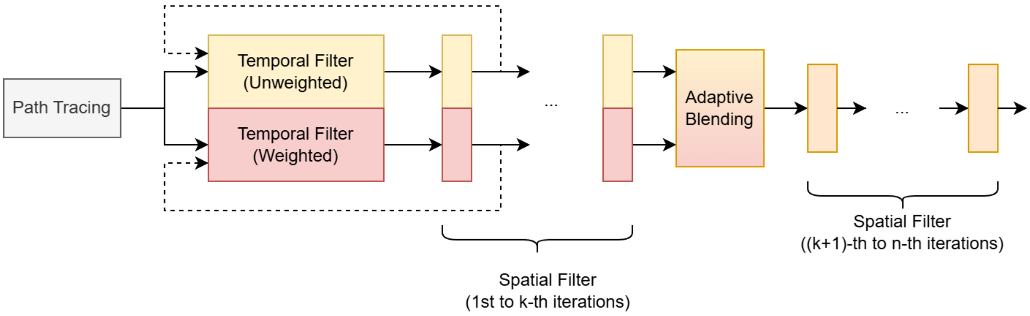

Figure 3.4: Blending after more iterations.

Why at least one iteration after blending is recommended? This is because our gradient estimation is noisy and the blended output may reflect such noise. We observe that one iteration of spatial filter is usually enough for easing out noise inherits from gradients. If a better gradient estimation method is used (such as method in [3]), then we might not need the additional layer after blending.

doi:10.6342/NTU202400987

Chapter 4

Experiments

4.1 Experiment Design

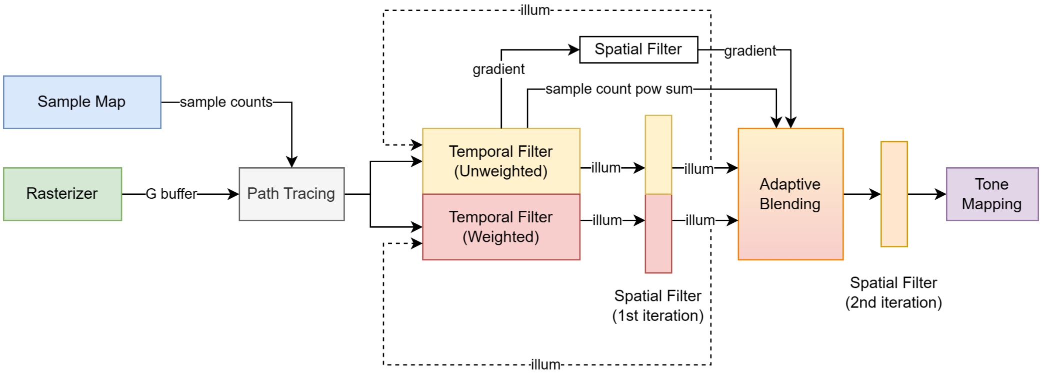

Figure 4.1: **The rendering pipeline for experiments.** We omit moment filtering, texture removal, and texture re-applying in this graph.

We modify SVGF by our two-history method and combine it with a simple foveated rendering algorithm. Figure 4.1 shows the pipeline structure.

The temporal filter and the first iteration of spatial filter are duplicated to maintain weighted history. Since SVGF has an additional spatial filter for moments, we also run that filter separately for unweighted and weighted histories. Improvement in temporal filter would be less significant after spatial-filtering due to blurring, so we use only 2 iterations instead of 4 that recommended by the original paper. The final TAA pass is also stripped for simplicity.

We render the images at 1280x720 resolution and set the foveal area to be inside a circle of 200px radius around gaze point. 8 spp are taken in foveal area and 1 spp is taken in peripheral area. The gaze point is simulated by moving along a Lissajous curve, as shown in Figure 4.2.

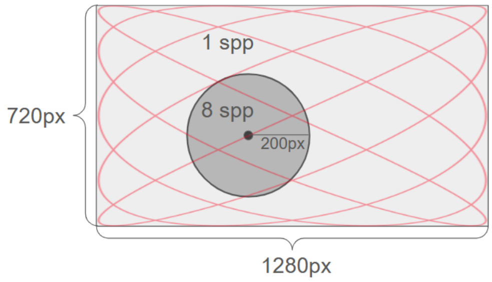

Figure 4.2: Foveated rendering settings

Two different metrics are used to measure the image quality for HDR images and tone-mapped images respectively. For HDR image, we use relative Mean Square Error (relMSE) $\frac{1}{|P|}\sum_{t=1}^{|P|}\frac{(y_i-r_i)}{(\epsilon+r_i^2)}$ to measure the error. For tone-mapped images, we use SSIM [20]. Reference images are generated by 128 sample-per-pixel path-tracing, denoised by un-modified SVGF with decay rate and number of spatial iterations set to the same as in foveated rendering pipeline.

Our implementation is based on the Falcor framework [21] (v5.2). The path-tracing pass and the tone-mapping pass in our pipeline remain default configuration provided by the Falcor framework.

The scenes we use for experiments are donated by [22], [23], [24], [25], and [26].

4.2 Image Quality

First we measure the average relMSE and SSIM of whole frames in the whole animations. Table 4.2 shows the metrics for unweighted, weighted, and two-history method. Table 4.3 shows results relative to unweighted method.

doi:10.6342/NTU202400987

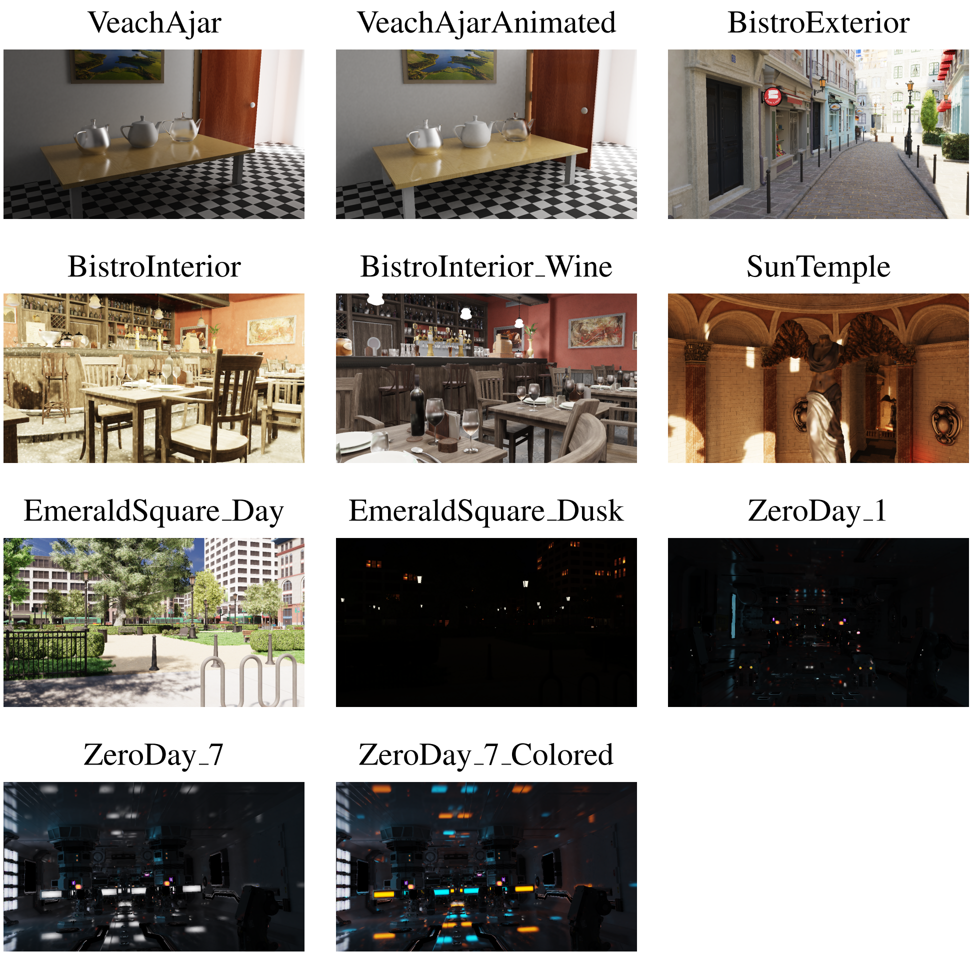

Table 4.1: Scenes for experiments

Scene	relMSE Unweighted	relMSE Weighted	relMSE Two-history	SSIM Unweighted	SSIM Weighted	SSIM Two-history
VeachAjar	4.435e-2	**4.137e-2**	4.213e-2	0.9038	**0.9222**	0.9180
VeachAjarAnimated	3.559e-2	3.716e-2	**3.457e-2**	0.9135	**0.9257**	0.9229
BistroExterior	1.507e-1	1.505e-1	**1.492e-1**	0.9264	0.9297	**0.9315**
BistroInterior	2.128e+2	**2.098e+2**	2.111e+2	0.8271	0.8281	**0.8295**
BistroInterior_Wine	1.683e+0	**1.678e+0**	1.678e+0	0.9180	0.9219	**0.9234**
SunTemple	1.246e+0	**1.199e+0**	1.212e+0	0.8592	0.8718	**0.8728**
EmeraldSquare_Day	1.690e+0	**1.642e+0**	1.668e+0	0.8563	0.8589	**0.8605**
EmeraldSquare_Dusk	1.580e-4	1.617e-4	**1.558e-4**	0.9832	0.9837	**0.9838**
ZeroDay_1	1.491e-2	1.473e-2	**1.311e-2**	0.9660	0.9649	**0.9675**
ZeroDay_7	**9.159e-3**	1.250e-2	9.181e-3	0.9631	0.9610	**0.9640**
ZeroDay_7_Colored	1.542e-2	2.161e-2	**1.540e-2**	0.9502	0.9488	**0.9518**

Table 4.2: **Average relMSE and SSIM.** Best results among the three methods are colored green.

Scene	relMSE Weighted	relMSE Two-history	SSIM Weighted	SSIM Two-history
VeachAjar	**-6.722%**	**-5.015%**	**+2.0355%**	**+1.5710%**
VeachAjarAnimated	+4.409%	**-2.862%**	**+1.3420%**	**+1.0317%**
BistroExterior	**-0.196%**	**-1.014%**	**+0.3485%**	**+0.5418%**
BistroInterior	**-1.419%**	**-0.797%**	**+0.1120%**	**+0.2893%**
BistroInterior_Wine	**-0.333%**	**-0.308%**	**+0.4294%**	**+0.5926%**
SunTemple	**-3.759%**	**-2.724%**	**+1.4693%**	**+1.5802%**
EmeraldSquare_Day	**-2.813%**	**-1.266%**	**+0.2990%**	**+0.4823%**
EmeraldSquare_Dusk	+2.397%	**-1.379%**	**+0.0455%**	**+0.0574%**
ZeroDay_1	**-1.196%**	**-12.114%**	-0.1138%	**+0.1495%**
ZeroDay_7	+36.487%	+0.238%	-0.2114%	**+0.0936%**
ZeroDay_7_Colored	+40.137%	**-0.125%**	-0.1533%	**+0.1664%**

Table 4.3: **Average relMSE and SSIM relative to unweighted method.** Improvements are colored green and worsen results are colored red.

doi:10.6342/NTU202400987

By comparing scenes VeachAjar (static) and VeachAjarAnimated, we found that weighted method has lower error in static scenes and higher error in animated scene. This is because that animated scene has drastic illuminance change which makes temporal bias difference significant. When weighted method performs better than unweighted method (e.g. scene VeachAjar), two-history method also performs better than unweighted method. When weighted method performs worse than unweighted method, two-history method can prevent most of the deterioration (e.g. ZeroDay_7) and sometimes has better overall quality than unweighted method (e.g. ZeroDay_7_Colored).

In figure 4.3 and 4.4, we plot relMSE and SSIM over time to take a closer look. We can see that our method is always closer to the better one among unweighted and weighted method. For the VeachAjar scene (figure 4.3), weighted method has better overall quality in most of the time, and our method has quality similar to weighted method. In the VeacAjarAnimated scene (figure 4.4), the scene darkens drastically from frame 200 to frame 250, which makes weighted method suffered from temporal bias difference. We can see the quality of weighted method becomes significantly worse than unweighted method in that duration, while our method stick with unweighted method to keep its quality.

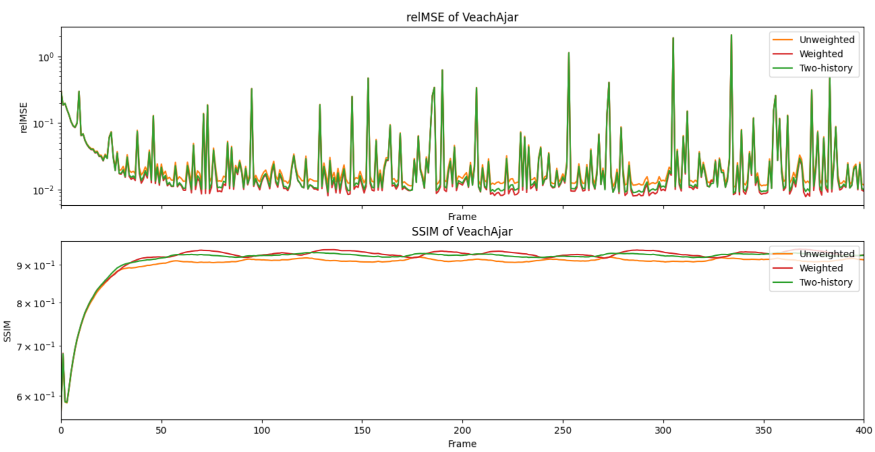

Figure 4.3: RelMSE and SSIM over time of the VeachAjar scene.

doi:10.6342/NTU202400987

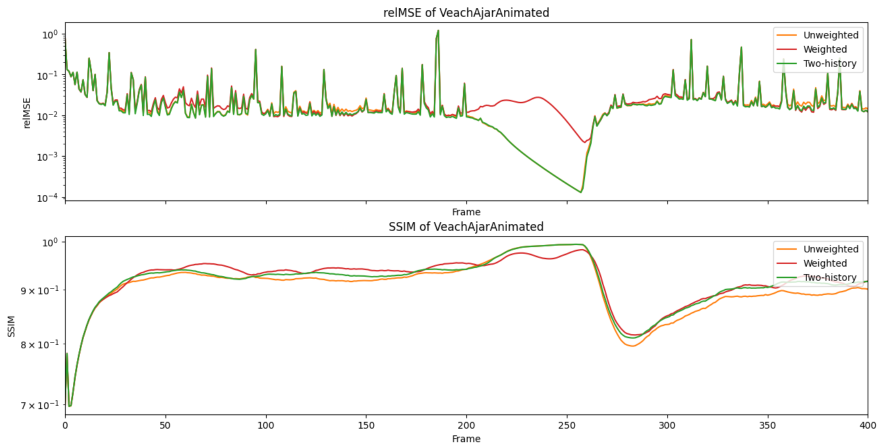

Figure 4.4: RelMSE and SSIM over time of the VeachAjarAnimated scene.

Since quality in foveal area is more important in foveated rendering, we also measure the performance for foveal area, shown in table 4.4 and table 4.5. We can see that weighted method usually performs bad in foveal area, even in static scenes like VeachAjar. This is because weighted method usually has variance larger than unweighted method has when recent frames have larger sample counts. In contrast, our method produces smaller error than unweighted method in all of the scenes we have tested. Although the SSIM values of our method decreases in many of the scenes, their magnitudes of difference are small enough to be ignored.

In Figure 4.5, weighted method looks noisier than unweighted method, while our method avoids noisier output by weighted method and rely more on unweighted method. This shows effect of first phase of blending.

In Figure 4.6, the weighted method suffer from temporal bias difference. The shown scene was darkening because the door was closing, and the foveal area (green circle) has moved from bottom-right to top-left. For weighted method, pixels passed through by foveal area darkens faster then other pixels, leaving a darker area behinds the foveal area. In such situation, our method rely more on unweighted history, avoiding temporal bias difference successfully, showing effects of second phase of blending.

doi:10.6342/NTU202400987

Scene	relMSE Unweighted	relMSE Weighted	relMSE Two-history	SSIM Unweighted	SSIM Weighted	SSIM Two-history
VeachAjar	7.639e-3	1.076e-2	**7.540e-3**	0.9517	0.9039	**0.9519**
VeachAjarAnimated	6.272e-3	2.378e-2	**6.223e-3**	0.9541	0.9048	**0.9542**
BistroExterior	2.618e-2	3.532e-2	**2.591e-2**	**0.9666**	0.9359	0.9666
BistroInterior	7.221e+1	**4.989e+1**	6.156e+1	**0.8361**	0.8094	0.8360
BistroInterior_Wine	2.438e-1	2.461e-1	**2.385e-1**	**0.9500**	0.9183	0.9500
SunTemple	2.181e-1	2.798e-1	**2.180e-1**	**0.8981**	0.8418	0.8981
EmeraldSquare_Day	6.582e-1	**2.088e-1**	4.815e-1	0.9313	0.9058	**0.9314**
EmeraldSquare_Dusk	3.576e-5	6.161e-5	**3.484e-5**	0.9911	0.9900	**0.9912**
ZeroDay_1	4.377e-3	9.542e-3	**4.290e-3**	**0.9817**	0.9640	0.9817
ZeroDay_7	2.919e-3	8.988e-3	**2.870e-3**	**0.9830**	0.9675	0.9830
ZeroDay_7_Colored	5.900e-3	1.563e-2	**5.720e-3**	**0.9754**	0.9565	0.9753

Table 4.4: **Average relMSE and SSIM in the foveal area.** Best results among the three methods are colored green.

Scene	relMSE Weighted	relMSE Two-history	SSIM Weighted	SSIM Two-history
VeachAjar	+40.837%	**-1.287%**	-3.5940%	**+0.0116%**
VeachAjarAnimated	+279.064%	**-0.790%**	-3.9232%	**+0.0078%**
BistroExterior	+34.906%	**-1.025%**	-2.7954%	-0.0034%
BistroInterior	**-30.906%**	**-14.741%**	-2.8883%	-0.0147%
BistroInterior_Wine	+0.955%	**-2.143%**	-2.7475%	-0.0054%
SunTemple	+28.294%	**-0.012%**	-5.3940%	-0.0060%
EmeraldSquare_Day	**-68.272%**	**-26.852%**	-2.3976%	**+0.0084%**
EmeraldSquare_Dusk	+72.301%	**-2.559%**	-0.1615%	**+0.0145%**
ZeroDay_1	+118.000%	**-1.987%**	-2.0680%	-0.0025%
ZeroDay_7	+207.912%	**-1.670%**	-1.7441%	-0.0024%
ZeroDay_7_Colored	+164.914%	**-3.043%**	-2.0783%	-0.0017%

Table 4.5: **Average relMSE and SSIM in the foveal area relative to unweighted method.** Improvements are colored green and worsen results are colored red.

doi:10.6342/NTU202400987

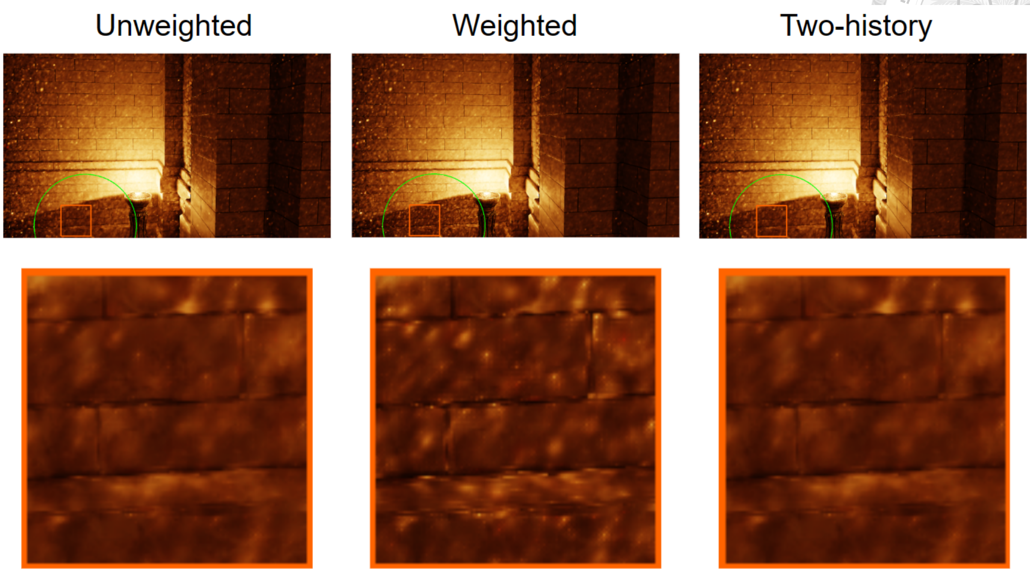

Figure 4.5: When weighted method is noisier than unweighted method.

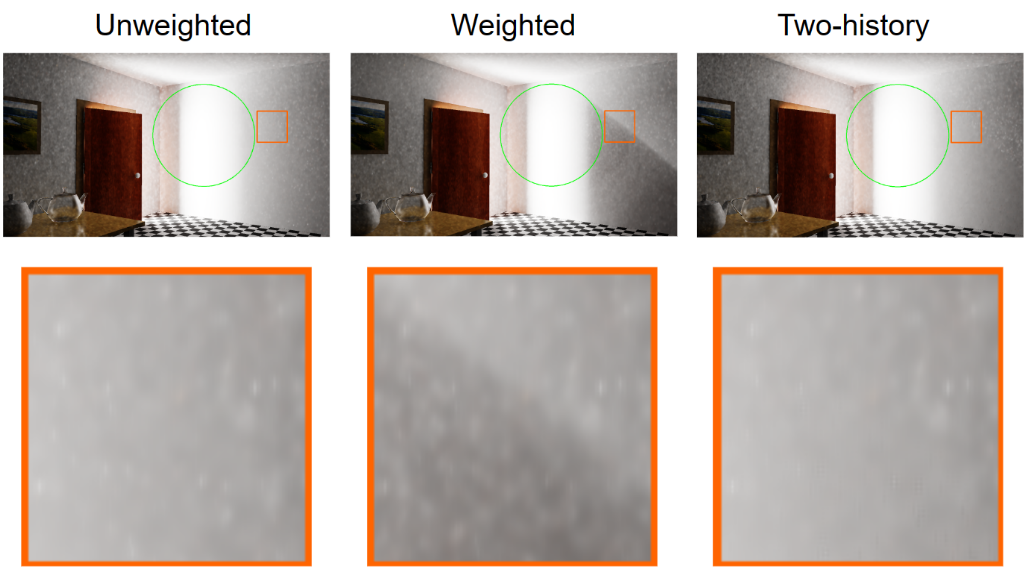

Figure 4.6: When difference in temporal bias is significant

4.3 Runtime

The two-history SVGF takes extra time to maintain additional history and calculate W_u, W_w, S_k and gradient. Such extra execution time is proportional to frame resolution. Table 4.6 compares the execution times at different resolutions. Usually the total time cost by two-history method is no more than twice the time cost by

doi:10.6342/NTU202400987

original (unweighted) method, and the ratio of extra time cost may decrease if we use more layers of spatio filters.

Although our modification costs additional time, foveated rendering has potential to saves much more GPU time than the extra cost due to our modification. So for scenes requiring larger number of samples to produce good visual effect, it might still be worthy to add foveated rendering combined with our method.

Resolution	**Pixel Count**	**Time (unweighted)**	**Time (Two-history)**
1280 x 720	921,600	1.336 ms	2.528 ms
1600 x 900	1,440,000	2.088 ms	4.153 ms
1920 x 1080	2,073,600	3.038 ms	5.572 ms

Table 4.6: Execution time at different resolution

Process	Original SVGF	Two-history SVGF
LinearZAndNormal	0.065 ms	0.065 ms
TemporalFilter	0.245 ms	0.471 ms
FilterGradients	-	0.299 ms
FilterMoments	0.145 ms	0.250 ms
SpatioFilter	0.650 ms	1.079 ms
Two-history	-	0.130 ms
FinalModulate	0.105 ms	0.106 ms
Others	0.126 ms	0.129 ms
Total time	1.336 ms	2.528 ms

Table 4.7: Runtime breakdown

doi:10.6342/NTU202400987

Figure 4.7: GPU time of filtering 720p image on RTX 3070ti GPU

4.4 Memory Usage

Since we maintain additional history and parameters, we also cost extra memory. Here are the extra fields we need for each pixel:

```
float3 curWeightedIllumination, prevWeightedIllumination
float2 curTotalWeights, prevTotalWeights
float3 curGradient, prevGradient
float3 curSampleCountPowerSums, prevSampleCountPowerSums
```

By float we means 32-bit floating point. The appended number means the dimension of vector.

In total, we need 88 bytes/pixel extra memory. Table 4.8 shows required memory at different resolutions. For desktop computers that have stronger GPUs, such memory cost may not be a problem. For VR devices however, as they have lesser GPU resource and higher resolution is required, such extra memory cost might be too costly.

doi:10.6342/NTU202400987

Resolution	Memory
1280 x 720	81.1 MB
1920 x 1080	182.5 MB
3840 x 2160	729.9 MB
2064 x 2208 x 2 (Quest 3)	802.1 MB

Table 4.8: Extra memory cost for different resolutions

doi:10.6342/NTU202400987

Chapter 5

Limitation and Future Work

For areas that sample counts rarely change, our method has nearly no effect. However, we still need to spend extra time to maintain two-history for those area. We leave reducing runtime for those area as future work.

Another limitations is that extra memory cost of our modification might be too expensive for VR devices. Further optimization like compressing the fields, or down-sampling gradient textures might mitigate the problem.

In the first phase of blending, we use a trivial variance estimator that only output a constant value, which might be too simple and may failed to provided best blending ratio especially when the luminance changes. There exists many techniques to estimate variance developed by adaptive sampling works. Choosing a proper estimator may improve the quality of our approach.

In the experiments part, we only test our method on SVGF combined with simple foveated rendering. We believe our approach is general enough to apply on other denoisers with spatio-temporal reusing such as ReSTIR, and also combine with other adaptive sampling algorithms.

Another issue of our experiment is absence of user study. Although we have measured the quality with some metrics, whether the improvement is worthy to spend extra execution time and memory depends on user experiments.

Finally, we think an interesting extension would be maintaining two histories using other different strategies. For example, by assigning different decay rates to the histories, we may adaptively adjust the decay rate by controlling the blending ratio. Previous works like A-SVGF [3] trying to change decay rate adaptively base on temporal gradient requires complicated method to estimate accurate gradients. This because over estimating gradient would invalidate old samples forever, effecting image quality in future frames. However, by maintaining two histories and blending between them, noise in gradients won't effect future frames, implying that using simpler gradient estimation method is possible.

doi:10.6342/NTU202400987

Chapter 6

Conclusion

In this paper we observed that many previous spatio-temporal filters use exponential moving average for temporal accumulation, which failed to put weight in colors generated by high sample count when combining with adaptive sampling methods such as foveated rendering.

To utilize colors generated by more samples, we introduced the ”Two-history Approach” which maintains both unweighted and weighted histories and blends them adaptively. When artifacts from weighted method are less significant, our method utilize it to reduce noise. When artifacts from weighted method are more significant, our method produce results similar to unweighted method.

Our modification cost roughly equal or less than twice of the time cost by unmodified filter, which might be not significant compared to the runtime saved by foveated rendering. Also additional 88 bytes per pixel memory cost is required, which is fine for desktops with stronger GPUs but might be a burden for VR devices.

doi:10.6342/NTU202400987

Reference

[1] 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. New York, NY, USA: Association for Computing Machinery, 2017. 5, 7, 8

[2] H. Dammertz, D. Sewtz, J. Hanika, and H. P. A. Lensch, "Edge-avoiding À-trous wavelet transform for fast global illumination filtering," in *Proceedings of the Conference on High Performance Graphics*, ser. HPG '10. Goslar, DEU: Eurographics Association, 2010, p. 67–75. 5

[3] C. Schied, C. Peters, and C. Dachsbacher, "Gradient estimation for real-time adaptive temporal filtering," *Proc. ACM Comput. Graph. Interact. Tech.*, vol. 1, no. 2, aug 2018. 5, 7, 8, 18, 31

[4] B. Bitterli, C. Wyman, M. Pharr, P. Shirley, A. Lefohn, and W. Jarosz, "Spatiotemporal reservoir resampling for real-time ray tracing with dynamic direct lighting," *ACM Trans. Graph.*, vol. 39, no. 4, aug 2020. 5

[5] G. Boissé, "World-space spatiotemporal reservoir reuse for ray-traced global illumination," in *SIGGRAPH Asia 2021 Technical Communications*, ser. SA '21. New York, NY, USA: Association for Computing Machinery, 2021. https://doi.org/10.1145/3478512.3488613 6

[6] Y. Ouyang, S. Liu, M. Kettunen, M. Pharr, and J. Pantaleoni, "Restir gi: Path

doi:10.6342/NTU202400987

resampling for real-time path tracing," *Computer Graphics Forum*, vol. 40, no. 8, pp. 17–29, 2021. 6

[7] D. Lin, C. Wyman, and C. Yuksel, "Fast volume rendering with spatiotemporal reservoir resampling," *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia 2021)*, vol. 40, no. 6, pp. 278:1–278:18, 12 2021. http://doi.acm.org/10.1145/3478513.3480499 6

[8] D. Lin, M. Kettunen, B. Bitterli, J. Pantaleoni, C. Yuksel, and C. Wyman, "Generalized resampled importance sampling: foundations of restir," *ACM Trans. Graph.*, vol. 41, no. 4, jul 2022. 6

[9] A. Kuznetsov, N. K. Kalantari, and R. Ramamoorthi, "Deep adaptive sampling for low sample count rendering," *Computer Graphics Forum*, vol. 37, pp. 35–44, 2018. 6

[10] C. R. A. Chaitanya, A. S. 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, jul 2017. https://doi.org/10.1145/3072959.3073601 6

[11] J. Hasselgren, J. Munkberg, M. Salvi, A. Patney, and A. Lefohn, "Neural temporal adaptive sampling and denoising," *Computer Graphics Forum*, vol. 39, no. 2, pp. 147–155, 2020. https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.13919 6

[12] M. Weier, T. Roth, E. Kruijff, A. Hinkenjann, A. Pérard-Gayot, P. Slusallek, and Y. Li, "Foveated real-time ray tracing for head-mounted displays," *Comput. Graph. Forum*, vol. 35, no. 7, p. 289–298, oct 2016. 6

[13] Y. Kim, Y. Ko, and I. Ihm, "Selective foveated ray tracing for head-mounted displays," in *2021 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, 2021, pp. 413–421. 6

[14] B. Jin, I. Ihm, B. Chang, C. Park, W. Lee, and S. Jung, "Selective and adaptive supersampling for real-time ray tracing," in *Proceedings of the Conference on High Performance Graphics 2009*, ser. HPG '09. New York, NY, USA: Association for Computing Machinery, 2009, p. 117–125. 6

[15] L. Wang, X. Shi, and Y. Liu, "Foveated rendering: A state-of-the-art survey," *Computational Visual Media*, vol. 9, no. 2, pp. 195–228, Jun 2023. 7

[16] C. R. A. Chaitanya, A. S. 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, jul 2017. https://doi.org/10.1145/3072959.3073601 7

[17] J. H. Mueller, T. Neff, P. Voglreiter, M. Steinberger, and D. Schmalstieg, "Temporally adaptive shading reuse for real-time rendering and virtual reality," vol. 40, no. 2, apr 2021. https://doi.org/10.1145/3446790 7, 8

[18] L. Yang, D. Nehab, P. V. Sander, P. Sitthi-amorn, J. Lawrence, and H. Hoppe, "Amortized supersampling," *ACM Trans. Graph.*, vol. 28, no. 5, p. 1–12, dec 2009. https://doi.org/10.1145/1618452.1618481 10

[19] B. Karis, "High-qality temporal supersampling," 2014. 10

[20] Z. Wang, A. Bovik, H. Sheikh, and E. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, 2004. 20

[21] S. Kallweit, P. Clarberg, C. Kolb, T. Davidovič, K.-H. Yao, T. Foley, Y. He, L. Wu, L. Chen, T. Akenine-Möller, C. Wyman, C. Crassin, and N. Benty, "The Falcor rendering framework," 8 2022. https://github.com/NVIDIAGameWorks/Falcor 20

[22] A. Lumberyard, "Amazon lumberyard bistro, open research con-

tent archive (orca)," July 2017. http://developer.nvidia.com/orca/amazon-lumberyard-bistro 20

[23] K. A. Nicholas Hull and N. Benty, "Nvidia emerald square, open research content archive (orca)," July 2017. http://developer.nvidia.com/orca/nvidia-emerald-square 20

[24] E. Games, "Unreal engine sun temple, open research content archive (orca)," October 2017. http://developer.nvidia.com/orca/epic-games-sun-temple 20

[25] M. Winkelmann, "Zero-day, open research content archive (orca)," November 2019. https://developer.nvidia.com/orca/beeple-zero-day 20

[26] B. Bitterli, "Rendering resources," 2016, https://benedikt-bitterli.me/resources/. 20

doi:10.6342/NTU202400987