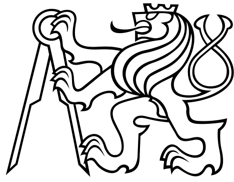
Master Thesis



Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Computer Graphics and Interaction

Efficient Sampling for Computing Complex Illumination in real-time

Bc. Karel Tomanec

Supervisor: doc. Ing. Jiří Bittner, Ph.D.
Field of study: Open Informatics
Subfield: Computer Graphics
May 2023

ČVUT
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Tomanec** Jméno: **Karel** Osobní číslo: **478155**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačové grafiky a interakce**

Studijní program: **Otevřená informatika**

Specializace: **Počítačová grafika**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Efektivní vzorkování pro výpočet komplexního osvětlení v reálném čase

Název diplomové práce anglicky:

Efficient sampling for computing complex illumination in real-time

Pokyny pro vypracování:

Prostudujte metody výpočtu komplexního přímého a nepřímého osvětlení v reálném čase. Zaměřte se na nedávno publikované články využívající vzorkování pomocí časově a prostorově sdíleného rezervoáru vzorků (ReSTIR). Implementujte metodu výpočtu komplexního dynamického přímého osvětlení s využitím hardwarově akcelerovaného vrhání paprsku (rozhraní RTX). Výsledky vyhodnoťte na nejméně třech různých scénách s různou složitostí přímého a nepřímého osvětlení. Důkladně zmapujte efektivitu jednotlivých částí implementace a navrhněte optimalizaci úzkých hrdel výpočtu.

Seznam doporučené literatury:

[1] Bitterli, B., Wyman, C., Pharr, M., Shirley, P., Lefohn, A., & Jarosz, W. (2020). Spatiotemporal reservoir resampling for real-time ray tracing with dynamic direct lighting. ACM Transactions on Graphics (TOG), 39(4), 148-1.
[2] Ouyang, Y., Liu, S., Kettunen, M., Pharr, M., & Pantaleoni, J. (2021, December). ReSTIR GI: Path Resampling for Real□Time Path Tracing. In Computer Graphics Forum (Vol. 40, No. 8, pp. 17-29).
[3] Lin, D., Wyman, C., & Yuksel, C. (2021). Fast volume rendering with spatiotemporal reservoir resampling. ACM Transactions on Graphics (TOG), 40(6), 1-18.
[4] Majercik, Z., Müller, T., Keller, A., Nowrouzezahrai, D., & McGuire, M. (2021). Dynamic Diffuse Global Illumination Resampling. In ACM SIGGRAPH 2021 Talks (pp. 1-2).
[5] Zeng, Z., Liu, S., Yang, J., Wang, L., & Yan, L. Q. (2021). Temporally Reliable Motion Vectors for Real□time Ray Tracing. In Computer Graphics Forum (Vol. 40, No. 2, pp. 79-90).
[6] Ritschel, T., Dachsbacher, C., Grosch, T., & Kautz, J. (2012). The state of the art in interactive global illumination. In Computer graphics forum (Vol. 31, No. 1, pp. 160-188). Oxford, UK: Blackwell Publishing Ltd.

Jméno a pracoviště vedoucí(ho) diplomové práce:

doc. Ing. Jiří Bittner, Ph.D. Katedra počítačové grafiky a interakce

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **12.09.2022** Termín odevzdání diplomové práce: **26.05.2023**

Platnost zadání diplomové práce: **19.02.2024**

doc. Ing. Jiří Bittner, Ph.D.
podpis vedoucí(ho) práce

podpis vedoucí(ho) ústavu/katedry

prof. Mgr. Petr Páta, Ph.D.
podpis děkana(ky)

© VUT v Praze, Design: VUT v Praze, VIC

III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací.
Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

.

Datum převzetí zadání

Podpis studenta

© VUT v Praze, Design: VUT v Praze, VIC

Acknowledgements

I would like to express my profound gratitude to doc. Ing. Jiří Bittner, Ph.D., for his valuable advice and guidance in supervising this work. I am immensely thankful to my family for their unflagging support and encouragement during my studies. Their unwavering belief in me has been a continuous source of inspiration.

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

In Prague, May 2023

Abstract

This thesis delves into advanced techniques for real-time computation of complex direct and indirect illumination, focusing on Reservoir-based SpatioTemporal Importance Resampling (ReSTIR). It offers a practical implementation of the ReSTIR algorithm for direct illumination, demonstrating a robust approach for biased and unbiased many-light sampling that leverages hardware-accelerated ray tracing. The study includes qualitative and performance tests validating the algorithm's effectiveness in various settings. Furthermore, we propose several optimizations to enhance computational efficiency and rendering quality. The thesis concludes with a discussion of potential future work, outlining promising directions for further improvements and expansions of the ReSTIR methodology.

Keywords: ReSTIR, real-time rendering, ray tracing, vzorkování rezervoáru, fotorealistický rendering

Supervisor: doc. Ing. Jiří Bittner, Ph.D.

Abstrakt

Tato práce se zabývá pokročilými technikami pro výpočet komplexního přímého a nepřímého osvětlení v reálném čase se zaměřením na metodu ReSTIR (Reservoir-based SpatioTemporal Importance Resampling). Součástí práce je praktická implementace algoritmu ReSTIR pro přímé osvětlení, která demonstruje robustní přístup pro vychýlené i nestranné vzorkování mnoha světel využívající hardwarově akcelerovaný ray tracing. Práce zahrnuje kvalitativní a výkonnostní testy, které ověřují účinnost algoritmu v různých nastaveních. Dále navrhujeme několik optimalizací pro zvýšení rychlosti výpočtu a kvality vykreslování. Práci uzavírá diskuze, která nastiňuje slibné směry pro další vylepšení a rozšíření metodiky ReSTIR.

Klíčová slova: ReSTIR, real-time rendering, ray tracing, reservoir sampling, photorealistic rendering

Překlad názvu: Efektivní vzorkování pro výpočet komplexního osvětlení v reálném čase

Contents

1 Introduction 1
1.1 Structure of the Thesis 2

2 Related Work 3
2.1 Problem Statement 3
2.1.1 Rendering Equation 3
2.1.2 Monte Carlo Importance Sampling 4
2.2 Previous Work 6
2.2.1 Lightcuts 6
2.2.2 Stochastic Lightcuts 7
2.2.3 Light BVH 7
2.2.4 Importance Resampling for Global Illumination 8
2.2.5 Denoising and Reconstruction Algorithms 9

3 ReSTIR Algorithm Overview 11
3.1 Resampled Importance Sampling 11
3.2 Streaming RIS Using Reservoir Sampling 13
3.3 Spatiotemporal Reuse 14
3.3.1 Visibility Reuse 15
3.4 Eliminating Bias 16

4 Implementation 21
4.1 Falcor Rendering Framework . . . 21
4.2 Test Scenes 22
4.3 Rendering Pipeline 23
4.3.1 V-buffer 24
4.3.2 Initial Candidates Generation 24
4.3.3 Temporal Resampling 26
4.3.4 Spatial Resampling 27
4.3.5 Shading 29
4.4 Target PDF 29
4.5 Ray Budget 30
4.6 Improving Cache Coherency 30
4.6.1 Light Tiles 30
4.7 True Spatiotemporal Reuse 32
4.8 Decoupled Shading 33
4.8.1 Cheaper Visibility 34
4.9 Checkerboard Rendering 34
4.10 Distribution of Samples among Light Types 35
4.11 Denoising and Upscaling 36
4.12 ReSTIR GI 36

5 Results 39
5.1 Qualitative Results 40
5.2 Performance Results 45
5.3 ReSTIR GI Results 50

6 Discussion and Future Work 53
6.1 Other ReSTIR applications 54

7 Conclusion 57

Bibliography 59

A Dependencies 63

B Electronic Contents 65

Figures

1.1 BEEPLE Zero-Day [Win19] scene rendered with path tracing in the Falcor rendering framework [KCK+22] 1

2.1 This diagram shows the complexity of the integrand $f(x)$, which depends on several factors..... 5

3.1 This diagram shows how bias arises when reusing samples from neighboring reservoirs 17

4.1 Falcor rendering framework [KCK+22]..... 22
4.2 Original ReSTIR pipeline [BWP+20]..... 24
4.3 Result after generating initial candidates and resamling..... 26
4.4 Illustration of reusing temporal samples..... 27
4.5 Comparison of using different number of spatial iterations after initial candidate generation (without temporal reuse), gathering 5 neighbors at each step 28
4.6 Temporal disocclusions under motion 28
4.7 Illustration of using light tiles with different pixel tile sizes after initial candidate generation (without spatial and temporal reuse) 32
4.8 True Spatiotemporal Reuse pipeline [WP21] 33
4.9 Decoupled pipeline [WP21]..... 33
4.10 Differences of using Original pipeline 4.2 (first row) and Decoupled pipeline 4.9 (second row)..... 34
4.11 High-level pipeline with integrated denoising and DLSS..... 36

5.1 Comparison of the results of the ReSTIR algorithm when sampling the emissive geometry (top) and sampling the environment maps (bottom)..... 40
5.2 Visualization of the origin of light samples for each individual pixel 41
5.3 Visualization of the count of resampled spatial samples..... 42
5.4 Equal-time comparison of Light BVH (left part of the pictures) and ReSTIR 4.2 (right part of the pictures)..... 43
5.5 Application of the SVGF denoiser [SSK+17] to the output of the ReSTIR algorithm..... 44
5.6 Perceptual image error produced by ꟻLIP [ANA+20] 45
5.7 Equal-time comparison of the efficiency of the ReSTIR GI algorithm 52

Tables

5.1 Scene information 39
5.2 Image Error versus Offline Reference 46
5.3 Performance results of various ReSTIR pipelines 47
5.4 Performance results with various light tiles settings of the biased Original pipeline 4.2 49
5.5 Performance results of Initial Candidates pass of the biased Original pipeline 4.2 for various resolutions 50

Chapter 1

Introduction

Over the past decade, path tracing has emerged as the standard algorithm for film production and the simulation of realistic optical effects. The film and game industries have begun to abandon empirical shading models in favor of physically-based ones. Moreover, the simplified model of light propagation via rasterization has necessitated filmmakers and developers to employ ray tracing methods for accurately calculating reflections, shadows, caustics, and indirect lighting.

Figure 1.1 : BEEPLE Zero-Day [Win19] scene rendered with path tracing in the Falcor rendering framework [KCK+22]. (Left) Rendered with path tracing with 1 sample per pixel. (Right) Rendered with path tracing with 1000 samples per pixel.

In recent years, path tracing for real-time applications has emerged as a significant challenge in computer graphics research. Even with the help of newly deployed graphic cards that enable hardware acceleration of ray tracing, it is currently possible to render only simple 3D scenes with a few rays per pixel to meet the requirements for a resolution 1920×1080 and a minimum frequency of 30 Hz.

The computation is significantly affected by the structure of the scene,

whether it is the geometry, the complexity of materials, or the lighting. Dynamic scenes, which may contain thousands of animated models, are common in real-time applications and can add further complexity to the overall computation. Given such constraints, we can only afford to cast a small number of shadow rays toward the light sources for each pixel. This results in an image output that contains a substantial amount of noise, which is typical for path tracing when using a limited number of samples.

An example of the noise resulting from a few samples can be seen in the left part of Figure 1.1. On the right, we see an image rendered with path tracing created by accumulating 1000 samples. Although rendering such an image may take a few seconds, upon closer inspection, subtle noise is still noticeable.

Denoising constitutes a separate area of research. As a subsequent block in the ray-tracing pipeline, it attempts to eliminate unwanted noise through reconstruction algorithms. The two primary denoising approaches involve filtering techniques and algorithms developed through training neural networks. The quality of the results produced by these algorithms is contingent on the amount of noise in the input image. Thus, it is necessary to develop methods that can minimize noise in the shortest possible time for the input of the denoising algorithm. Consequently, selecting the appropriate light sources for sampling, which most significantly affect the visual appearance of a given surface, is crucial for the quality of the final image.

This thesis explores state-of-the-art methods for real-time computation of complex direct and indirect illumination using Reservoir-based SpatioTemporal Importance Resampling (ReSTIR) [BWP+20]. Additionally, it provides a practical implementation of the ReSTIR algorithm for direct illumination. This method constitutes a compelling approach for biased and unbiased many-light sampling, employing hardware-accelerated ray tracing.

1.1 Structure of the Thesis

Chapter 2 outlines the problem addressed in this thesis, introduces fundamental concepts, and reviews previous work on this topic. Chapter 3 offers a theoretical description of the ReSTIR algorithm. Chapter 4 details our implementation of the ReSTIR algorithm for direct illumination, including a description of the optimizations utilized. In Chapter 5, we conduct qualitative and performance analyses of our implementation results. Chapter 6 allows us to discuss the results obtained in this work and suggest potential avenues for extending and enhancing the existing method. Finally, Chapter 7 provides a summary of this thesis's results.

Chapter 2

Related Work

Addressing the complexity of real-time direct and indirect illumination has been the focus of numerous contemporary research papers. The methodologies predominantly involve various sampling techniques or the development of mathematical models to tackle this challenge.

This chapter will introduce the essential theoretical concepts required to comprehend this subject matter. Additionally, we will review several noteworthy papers that have contributed to the field in recent years.

2.1 Problem Statement

In this section, we address the primary issue associated with ray tracing, which is the solution of the rendering equation, and we further discuss the use of Monte Carlo importance sampling in this context.

2.1.1 Rendering Equation

The fundamental problem of ray tracing is solving the rendering equation [Kaj86], which is an integral equation that describes the outgoing radiance at point y in direction $\boldsymbol{\omega}_o$

$$L_o(y, \boldsymbol{\omega}_o) = L_e(y, \boldsymbol{\omega}_o) + \int_\Omega L_i(y, \boldsymbol{\omega}_i)\, \rho(\boldsymbol{\omega}_o, \boldsymbol{\omega}_i)\, \langle \cos\theta_i \rangle\, d\omega_i \tag{2.1}$$

where L_o is the total outgoing radiance, L_e is the emitted radiance, Ω is the hemisphere of directions, L_i is the incoming radiance coming toward y from direction $\boldsymbol{\omega}_i$, ρ is the bidirectional scattering distribution function (BSDF), $\cos\theta_i$ is the clamped cosine of the angle between the surface normal at the point y and direction ω_i, and $d\omega_i$ is the solid angle.

The incoming radiance L_i can be written as the outgoing radiance at the visible surface along a ray from y in the direction ω_i:

$$L_i(y, \boldsymbol{\omega}_o) = L_o(r(y, \boldsymbol{\omega}_o), -\boldsymbol{\omega}_o) \tag{2.2}$$

where r is the ray casting function that returns the closest point on the surface hit by the ray. The problem in solving this equation is that it contains an unknown quantity L on both sides and also under the integral

sign. Therefore, this equation cannot be solved analytically in the general case.

In some cases, it may be helpful to split the integral of the rendering equation into two separate integrals over direct and indirect illumination:

$$L_o(y, \boldsymbol{\omega}_o) = L_{direct}(y, \boldsymbol{\omega}_o) + L_{indirect}(y, \boldsymbol{\omega}_o) \tag{2.3}$$

In the implementation part of the thesis, we will deal mainly with the solution of direct lighting. In this context, the equation can be rewritten as an integral over all light-emitting surfaces A:

$$L_{direct}(y, \boldsymbol{\omega}_o) = \int_A L_e(y, \overrightarrow{yx})\, \rho(\boldsymbol{\omega}_o, \overrightarrow{yx})\, G(y, x)\, V(y, x)\, dAx \tag{2.4}$$

where L_e is the emitted radiance, ρ is BSDF, G is the geometry term, and V is the mutual visibility between y and x. The geometry term typically contains inverse squared distance and cosine terms.

This equation will be written for brevity as:

$$L = \int_A f(x)\, dx \tag{2.5}$$

2.1.2 Monte Carlo Importance Sampling

Given that the rendering equation cannot be analytically solved in general scenarios, we use a numerical approximation of the solution. The conventional approach employs the Monte Carlo method, which utilizes the following estimator:

$$\langle L \rangle_{IS}^{N} = \frac{1}{N} \sum_{i=1}^{N} \frac{f(x_i)}{p(x_i)} \approx L \tag{2.6}$$

Employing the traditional Monte Carlo method can yield noisy results. To mitigate this noise, one could either increase the number of samples N or attempt to construct a distribution of samples p (Probability Density Function or PDF) that aligns more closely with the integrand. The first option is usually impractical in real-time applications due to high computational costs, while the second can be technically challenging. Optimally, a perfect importance sampler reduces N to 1, but requires $p \propto f$ and as PDFs must integrate to unity, means:

$$p(x) = \frac{f(x)}{\int f(x)\, dx} \tag{2.7}$$

but requires knowing L in advance.

The shape of the integrand $f(x)$ is influenced by many of factors, given that it is the product of several different functions. Each one of these constituent functions can potentially contribute to the presence of noise.

For instance, the emitted radiance function L_e can pose complexity, as a scene may contain numerous lights with significant intensity variations (Figure 2.1 (a)). Certain lights might be extremely bright, while others may be dim, contributing insignificantly to the overall lighting.

The Bidirectional Scattering Distribution Function (BSDF), denoted by ρ, can also exhibit complexity as it might contain high frequencies that are dependent on the viewing angle (Figure 2.1 (b)). This is often observed in scenes with models that incorporate glossy materials.

The geometric term G can be another source of noise as it depends on the distance between the light source and the shading point, as well as the orientation of their respective surfaces (Figure 2.1 (c)).

The visibility function V can pose challenges as well, given that many of the selected samples may not be visible from a specific point (Figure 2.1 (d)). This could be the case in a scene with a building with light sources in separate rooms.

These observations suggest that effectively sampling the product of these functions can be an immensely challenging, if not impossible, task.

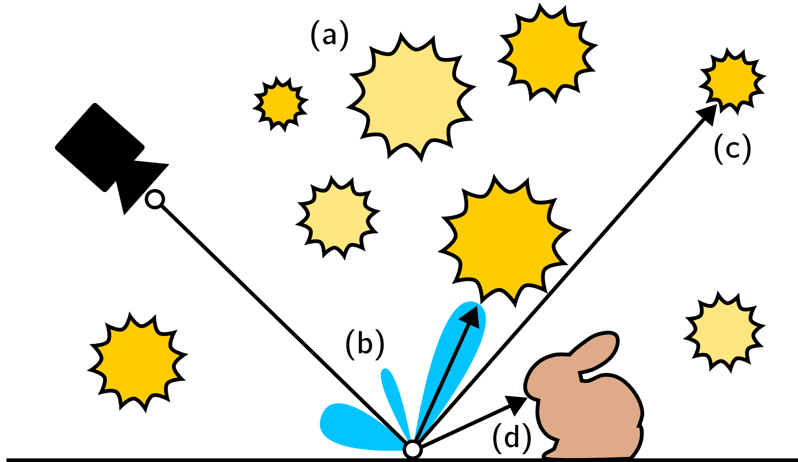


Figure 2.1 : This diagram shows the complexity of the integrand $f(x)$, which depends on several factors.

The presence of noise can be mitigated by sampling the individual functions constituting the product $f(x)$ separately. We can select M distinct sampling strategies—for instance, light sampling and BSDF sampling—and select N_s samples from a specific strategy s.

This approach is called *Multiple importance sampling* (MIS) and is described by the following weighted estimator:

$$\langle L \rangle_{MIS}^{M,N} = \sum_{s=1}^{M} \frac{1}{N_s} \sum_{i=1}^{N_s} \mathrm{w}_s(x_i) \frac{f(x_i)}{p_s(x_i)} \tag{2.8}$$

This estimate remains unbiased if w_s holds for the weights $\sum_{s=1}^{M} w_s(x) = 1$.

Monte Carlo ray tracing provides the ability to render scenes with many light sources. This is possible as light sources can be sampled stochastically, and tracing rays can evaluate shadowing effects. This method offers an advantage over shadow maps, a common practice in rasterization, which don't scale effectively when dealing with many light sources.

2.2 Previous Work

Efficient sampling for the computation of complex direct and indirect illumination is especially critical for scenes encompassing many light sources. This challenge is known within the computer graphics community as "The Many-Lights Problem." Over the years, numerous research papers have addressed this issue, focusing on both scenarios involving analytical light sources, such as point lights or realistic scenes containing area lights.

This section will explore various previously proposed solutions to this challenge. Furthermore, we will introduce the crucial role of denoising and reconstruction algorithms, which are essential complements for the real-time application of path tracing. These algorithms aid in reducing noise and improving the visual quality of the rendered images, thereby facilitating real-time performance.

2.2.1 Lightcuts

This approach has inspired numerous other methods aimed at computing realistic illumination. At its heart is an algorithm approximating illumination from many point lights at a significantly sublinear cost [WFA+05].

At the onset of each frame, a so-called *light tree* is constructed. This binary tree is global for the whole computation process and contains individual lights in each leaf. The interior nodes represent clusters of lights, encompassing the lights below them in the subtree. The quality of these clusters reaches its peak when they comprise lights that share high similarity in their material, geometric, and visibility terms. The authors of this paper address this by grouping lights based on spatial proximity and orientation similarity.

During the selection of lights for shading, a so-called *cut* through the light tree is chosen for each pixel, corresponding to a valid cluster partitioning. The lights corresponding to a chosen cut are used in shading, substantially reducing the overall complexity and rendering it sublinear. Each cut presents a unique cluster partitioning and thus varies in cost and quality. To address this, the authors employ a relative error criterion that determines whether the resulting cut is appropriate for use in a given context based on an estimate of total radiance.

The construction of the tree is a stochastic process to circumvent bias. However, once the tree is built, the same tree is used for the entire image. Moreover, each internal node contains a representative light shared with one of its child nodes. The issue with this method is that the lights at the top of

the tree are more likely to be sampled. For instance, the representative light of the root node is continuously sampled. This leads to sampling correlation and temporal instability. Furthermore, this method is unsuitable for real-time rendering as it still requires a relatively large number of samples to achieve quality results.

2.2.2 Stochastic Lightcuts

Another approach for addressing complex illumination is Stochastic Lightcuts [Yuk20]. This method is directly inspired by Light Cuts and attempts to remove its shortcomings, such as temporal instability and high noise, when using few samples.

The central concept involves the removal of representative lights from the internal nodes. Instead, each internal node is treated as an aggregation of light sources beneath its subtree. Then, when sampling a given node, a light sample is randomly selected. This process effectively removes the flickering, replacing it with noise.

The light sample is chosen using *Hierarchical Importance Sampling*. Individual nodes are assigned a probability that determines the likelihood of their selection during sampling. Each node's probability is given by a weight that incorporates the parameters of the subtree. These parameters include the total light intensity of the subtree, reflectance bound (incorporating both material and geometric terms), and the minimum distance to the bounding box of the subtree.

Although this approach was initially designed for offline rendering, the authors implemented simple modifications to make it more GPU-friendly [LY20]. The first modification involves using a perfect binary light tree, which is highly efficient to construct and traverse on the GPU. The second modification entails sharing cuts between blocks of k×k pixels rather than calculating a cut for each pixel separately—a process that proves costly for real-time rendering. The computation pass is executed for each pixel block, where one of the k×k pixels is selected as the representative pixel. This representative pixel's material and geometric properties are used to create the cut.

2.2.3 Light BVH

Several researchers have investigated approaches based on building bounding hierarchies over the light sources and traversing them to sample lights. Conty Estévez and Kulla [CEK18] proposed constructing a light bounding volume hierarchy (Light BVH) that aggregates energy, spatial, and orientation data from the emitters. This facilitates the accurate prediction of the effect a cluster of lights may have on any given shading point. As a result, instead of calculating these terms for all light sources, it enables the hierarchical approximation of these quantities, thus reducing the per-sample complexity to $\mathcal{O}(\log n)$.

While the original method does not accommodate dynamic light sources, Moreau et al. [MPC22] proposed the creation of a two-level acceleration structure. Prior approaches utilized a single bounding volume hierarchy (BVH) that needed complete rebuilding if even a single light source was moved or had its parameters, such as intensity, altered. This is not ideal for real-time rendering. The issue with dynamic lights is connected to managing data structures for ray-intersection testing in dynamic scenes.

Ray tracing APIs employ two-level BVHs that store collections of geometry (e.g., meshes) in a bottom-level acceleration structure (BLAS) and maintain a top-level acceleration structure (TLAS) housing the BLASes. The same principle can be applied to emissive meshes. The authors suggest storing each emissive mesh in its own BLAS; if the emissive geometry moves, only its own BLAS and the TLAS must be rebuilt. They further enhance performance by updating the light BVH via refitting directly on the GPU, preserving its original topology.

The sampling of the two-level light tree is executed by initially traversing the top-level acceleration structure (TLAS) down to a leaf node. This is achieved by evaluating an importance function for each of the current node's children and stochastically selecting one. Each leaf node of the TLAS points to a bottom-level acceleration structure (BLAS), and the same technique is employed to select a light within it. The probability of sampling a light is the product of the probability of sampling a given BLAS in the TLAS and the probability of sampling it within its own BLAS.

2.2.4 Importance Resampling for Global Illumination

To reduce noise in Monte Carlo integration, we typically use importance sampling. In order to use it, we need to be able to generate samples with a distribution according to the probability density function. The more closely the PDF aligns with the integrand, the lower the variance of the Monte Carlo estimate. Three commonly employed techniques include generating samples using the inversion of the cumulative distribution function (CDF), rejection sampling, and Metropolis sampling. In their paper, Talbot et al. [TCE05] introduce a fourth technique, called *Importance Resampling*, which they combine with importance sampling to present a novel variance reduction method named *Resampled Importance Sampling* (RIS).

This method allows for better sampling of the function f when the source PDF q is a poor approximation of this function. The authors show that we can choose an unnormalized function $\hat{p}$ (e.g., $\hat{p} \propto \rho \cdot L_e \cdot G$) that is a good approximation of f and guide the sampling with it.

This approach significantly reduces variance compared to standard importance sampling, mainly when dealing with common rendering challenges such as direct illumination sampling or BRDF sampling. Furthermore, this method is the foundational basis for the technique we will discuss in the next chapter.

2.2.5 Denoising and Reconstruction Algorithms

While it is still impossible to generate noise-free images in real-time ray tracing applications using a single GPU, recent research efforts have concentrated on eliminating noise in the resulting image while maintaining interactivity. These strategies are orthogonal to the ones mentioned previously and can be employed to enhance the output of prior methods.

Denoising algorithms commonly employ three techniques: spatial filtering, temporal accumulation, and machine learning-based reconstruction. Spatial filtering leverages the often-present similarities in neighboring pixels. Its advantage is that it does not induce temporal lag, allowing it to respond quickly to changes in the scene. However, this method can result in blurring and temporal instability, leading to flickering. Temporal filtering utilizes information from previous frames through pixel position reprojection, thereby avoiding blurring and maintaining temporal stability.

These two techniques are frequently combined, as exemplified in Spatiotemporal Variance-Guided Filtering (SVGF), where the authors utilize variance estimation to differentiate between noise and detail [SSK+17]. It is also common in these algorithms to use geometric information from the scene, such as normal maps or depth, for edge detection as guidance for the denoising algorithm, which helps prevent image blurring [BEM11].

Direct and indirect illumination is frequently filtered separately using Equation 2.1.1. This is because the noise patterns of direct and indirect illumination exhibit different characteristics. Direct lighting often contains high frequencies, while indirect lighting appears smoother in areas with similar geometry.

NVIDIA has introduced NVIDIA Real-Time Denoisers (NRD), a collection of spatio-temporal denoisers that are easily integrated into the rendering pipeline [nrd22]. NRD offers better image quality than SVGF and supports denoising various signals, including diffuse reflections, specular reflections, and shadows.

Machine learning and deep learning algorithms employ neural networks for signal reconstruction. These networks are trained using a variety of noisy and reference signals. These algorithms can be temporally unstable and may require temporal information for guidance. An example of such a denoiser is the NVIDIA OptiX™ AI-Accelerated Denoiser, based on [CKS+17], which enables the reconstruction of global illumination with extremely low sampling budgets at nearly-interactive frame rates.

Chapter 3

ReSTIR Algorithm Overview

In this chapter, we delve into the details of the algorithm presented in the paper by Bitterli et al. [BWP+20], which forms the foundation of the implementation aspect of this thesis. This algorithm, primarily designed for the computation of direct illumination, enables the rendering of fully dynamic scenes with numerous light sources. Unlike methods previously published, it does not require complex spatial data structures and utilizes a constant amount of memory that solely depends on the resolution of the resultant image. Furthermore, the computational complexity remains constant across frames, ensuring no fluctuation in the rendering frequency during sudden changes in the scene.

3.1 Resampled Importance Sampling

This technique was described in the paper as mentioned earlier by Talbot et al. [TCE05]. Initially, we generate $M \geq 0$ initial samples, also referred to as candidates, according to the *source* PDF q, which is sub-optimal with respect to f, but easy to sample from (e.g., $q \propto L_e$). Subsequently, we randomly select one candidate, represented by index z, for which the following holds:

$$p(z \mid \{x_1, \ldots, x_M\}) = \frac{w(x_z)}{\sum_{i=1}^{M} w(x_i)} \tag{3.1}$$

$$w(x) = \frac{\hat{p}(x)}{q(x)} \tag{3.2}$$

where $\hat{p}(x)$ is the *target* PDF, for which no practical sampling algorithm may exist, and it does not have to be normalized, so we have much freedom in its choice (e.g., $\hat{p} \propto L_e \cdot G \cdot \rho$).

The sample $y := x_z$ is selected, and repeating this process and averaging the results leads to the following N-sample RIS estimator:

$$\langle L \rangle_{RIS}^{N,M} = \frac{1}{N} \sum_{i=1}^{N} \left(\frac{f(y_i)}{\hat{p}(y_i)} \cdot \left(\frac{1}{M} \sum_{j=1}^{M} w(x_{ij}) \right) \right) \approx L \tag{3.3}$$

This process can be seen as approximating perfect importance sampling 2.7 by iteratively applying Monte Carlo integration:

$$p(x) = \frac{\hat{p}(x)}{\int \hat{p}(x)dx} \approx \frac{\hat{p}(x)}{\frac{1}{M}\sum_j \frac{\hat{p}(x_j)}{q(x_j)}} \tag{3.4}$$

Merging Equation 3.4 into estimator 2.6 gives the RIS estimator 3.3.
Let us define the function $W(x, z)$:

$$W(x, z) = \frac{1}{\hat{p}(x_z)}\left(\frac{1}{M}\sum_{j=1}^{M} w(x_j)\right) \tag{3.5}$$

which allows the Equation 3.3 to be written as

$$\langle L\rangle_{RIS}^{N,M} = \frac{1}{N}\sum_{i=1}^{N} f(x_i)W(x, i) \tag{3.6}$$

The function W can be viewed as a weighting factor of the resulting sample y evaluated by the integrand f.
The estimate above remains unbiased when:

$$\mathbb{E}[W(x, z)] = \frac{1}{p(x_z)} = \frac{\int \hat{p}(x)dx}{\hat{p}(x_z)} \tag{3.7}$$

which requires $\mathbb{E}[\frac{1}{M}\sum \frac{\hat{p}(x_i)}{q(x_i)}] = \int \hat{p}(x)dx$.
The pseudocode for this procedure, where $N = 1$, can be seen in Algorithm 1.

Algorithm 1: Resampled Importance Sampling (RIS)

Input: $M \geq 1$ number of candidates to generate for pixel t
Output: Sample y and the sum of RIS weights $\sum_{i=1}^{M} w(x_i)$

```
1  x ← ∅
2  w ← ∅
3  w_sum ← 0
4  for i ← 1 to M do
5  |   generate x_i ~ q
6  |   x ← x ∪ {x_i}
7  |   w_i ← p̂_q(x_i)/q(x_i)
8  |   w_sum ← w_sum + w_i
9  |   w ← w ∪ {w_i}
10 compute normalized CDF C from w
11 draw random index z ∈ [0, M) using C to sample ∝ w_z
12 y ← x_z
13 return y, w_sum
```

In [WP21], Wyman et al. explore edge cases of the estimator 3.3. For $M = 1$, RIS becomes a standard Monte Carlo estimator. For small values of M, the quality of q is crucial for the final estimate. As M grows, RIS better samples the target PDF $\hat{p}$, and for $M \to \infty$, the quality of q becomes irrelevant as RIS perfectly samples $\hat{p}$.

3.2 Streaming RIS Using Reservoir Sampling

The issue with RIS is that all M candidates must be stored to select the final sample. Bitterli et al. [BWP+20] solved this by combining RIS with *Weighted Reservoir Sampling* (WRS).

Reservoir sampling is a family of algorithms that, given a stream of N elements, can randomly select a K-element subset in a single pass. Often, K is defined as a small constant, and there is no need to know N in advance. Each sample is assigned a weight $w(x_i)$, such that the sample x_i is selected with a probability proportional to its weight:

$$P_i = \frac{w(x_i)}{\sum_{j=1}^{N} w(x_j)} \tag{3.8}$$

Thus, WRS processes each element only once and retains only the required number of elements, K, in the reservoir. The reservoir is updated after processing each element, maintaining the invariant that after N samples, the sample x_i will remain in the resulting reservoir with a probability proportional to its weight.

In this discussion, we are focusing on the variant of the algorithm where the reservoir size $K = 1$.

This algorithm is described by pseudocode 2 as a function inside the `Reservoir` class. Initially, an empty reservoir is created. Then, the sample stream is traversed to update the reservoir with the chosen samples.

Algorithm 2: Weighted Reservoir Sampling (WRS)

```
   Data: Set of samples 𝕊
 1 class Reservoir
 2 |   Sample y
 3 |   w_sum ← 0
 4 |   M ← 0
 5 |   W ← 0
 6 |   function update(x_i, w_i)
 7 |   |   w_sum ← w_sum + w_i
 8 |   |   M ← M + 1
 9 |   |   if rand() < (w_i / w_sum) then
10 |   |   |   y ← x_i
11 function reservoirSampling(𝕊)
12 |   Reservoir r
13 |   for i ← 1 to M do
14 |   |   r.update(𝕊[i], weight(𝕊[i]))
15 |   return r
```

The combination of RIS and WRS enables selection of a desired number of samples from a random data stream into the resulting reservoir. The pseudocode for this combined approach can be seen in 3. In this case, we

generate a separate reservoir for each image pixel; we then traverse the stream of M samples, each of which we generate with a probability q. For each of these samples, we compute the weight according to equation 3.2, which is then used to update the reservoir. After processing the entire sample stream, we compute the final reservoir weight W (as per Eq. 3.1), which serves as a weighting factor for pixel shading (Eq. 3.1).

Algorithm 3: Streaming RIS using weighted reservoir sampling

```
   Data: Image I
 1 foreach pixel t ∈ I do
 2 |   I[t] ← shadePixel(RIS(t), t)
 3 function RIS(t)
 4 |   Reservoir r
 5 |   for i ← 1 to M do
 6 |   |   generate x_i ∼ q
 7 |   |   r.update(x_i, p̂_t(x_i)/p(x_i))
 8 |   r.W = 1/p̂_q(r.y) · ( 1/r.M · r.w_sum )
 9 |   return r
10 function shadePixel(Reservoir r, t)
11 |   return f_t(r.y) · r.W
```

Indeed, this combined approach of Resampled Importance Sampling (RIS) and Weighted Reservoir Sampling (WRS) significantly reduces the spatial complexity from the original RIS, which is $\mathcal{O}(M)$, to a much more manageable $\mathcal{O}(1)$. However, the time complexity remains at $\mathcal{O}(M)$. This is particularly beneficial as it necessitates storing only one reservoir for each pixel, thereby limiting the total size of the data structure to the size of the image resolution. This ensures the process remains efficient and manageable, even with high-resolution images.

3.3 Spatiotemporal Reuse

Streaming RIS combined with Weighted Reservoir Sampling efficiently selects relatively good samples. However, generating many samples to achieve a good approximation of the function $\hat{p}$ (and consequently f) can be time-consuming. This is especially critical in real-time scenarios where computational resources and time are typically limited.

Bitterli et al. [BWP+20] exploit the observation of a significant correlation between the PDFs of $\hat{p}$ for adjacent pixels. In other words, for $\hat{p} = L_e \cdot G \cdot \rho$ (unshadowed illumination), neighboring pixels are likely to exhibit similar illumination patterns and possess similar material and geometric properties. Denoising algorithms frequently leverage this correlation.

In real-time applications, camera motion generates a sequence of frames, allowing for the utilization of information from prior frames to generate a new one. By employing motion vectors, we can determine the position of

Algorithm 4: Combining the streams of multiple reservoirs

```
   Input: Reservoirs r_i to combine
   Output: A combined reservoir s
1  function combineReservoirs(t, r_1, ..., r_k)
2      Reservoir s
3      foreach r ∈ {r_1, ... r_k} do
4          s.update(r.y, p̂_t(r.y) · r.W · r.M)
5      s.M ← r_1.M + ... + r_k.M
6      s.W = 1/p̂_t(s.y) · (1/s.M · s.w_sum)
7      return s
```

the current pixel in the previous frame (a technique also used by temporal anti-aliasing methods, such as TAA). This enables us to use the samples stored in the reservoir from previous frames.

The challenge lies in the need to store all generated samples along with their respective weights to reuse them spatially or temporally. However, Bitterli et al. [BWP+20] demonstrate that it is possible to consider the sample y in the reservoir as a newly generated sample with a weight of w_{sum}. This operation is mathematically equivalent to performing reservoir sampling on the combined input streams of two reservoirs. Algorithm 4 presents the pseudocode for combining the input streams of k reservoirs, and it executes in $\mathcal{O}(k)$.

Through this process, we can achieve quality sampling where each pixel is presented with kM candidates and a time complexity of $\mathcal{O}(M + k)$. Furthermore, the spatial reuse can be executed in n iterations, resulting in each pixel being presented with knM candidates and a time complexity of $\mathcal{O}(M + nk)$. The complete pseudocode for this algorithm is depicted in Algorithm 5.

3.3.1 Visibility Reuse

While we can obtain a considerable number of samples using spatiotemporal reuse, $\hat{p}$ does not perfectly sample the integrand f. This is because $\hat{p}$ represents an unshadowed contribution. We could integrate visibility into $\hat{p}$, which would be computationally demanding in real-time.

To mitigate this, after generating M candidates, we perform a visibility test for each pixel, setting the reservoir's weight W to 0 if the test fails. Conducting the visibility test immediately after the initial candidates are generated also results in the reuse of visibility, as samples in shadowed areas are not propagated to neighboring reservoirs. Moreover, by identifying early that a sample in the reservoir is in shadow, the reservoir has a better chance of acquiring the correct sample in subsequent spatial and temporal reuse. Both of these outcomes significantly reduce noise.

Algorithm 5: RIS with spatiotemporal reuse

```
   Data: Image I
   Input: Image sized buffer containing the previous frame's reservoirs
   Output: The current frame's reservoirs
 1 function reservoirReuse(prevReservoirs)
 2     reservoirs ← new Array[ImageSize]
       // Generate initial candidates
 3     foreach pixel t ∈ I do
 4         reservoirs[t] ← RIS(t)
       // Evaluate visibility for initial candidates
 5     foreach pixel t ∈ I do
 6         if shadowed(reservoirs[t].y) then
 7             reservoirs[t].W ← 0
       // Temporal reuse
 8     foreach pixel t ∈ I do
 9         t' ←pickTemporalNeighbor(t)
10         reservoirs[t]← combineReservoirs(t, reservoirs[t],
            prevReservoirs[t'])
       // Spatial reuse
11     for i ← 1 to n do
12         foreach pixel t ∈ I do
13             Q ← pickSpatialNeighbors(t)
14             ℝ ← { reservoirs[t'] | t' ∈ Q}
15             reservoirs[t]←combineReservoirs(t, reservoirs[t],ℝ)
       // Shade pixel
16     foreach pixel t ∈ I do
17         I[t]←shadePixel(reservoirs[t], t)
18     return reservoirs
```

3.4 Eliminating Bias

The previous section describes how ReSTIR can provide an approximately perfect importance sampler. However, one detail that needs to be addressed is that each pixel uses a different integration domain and target distribution $\hat{p}$, which could introduce bias.

Bitterli et al. [BWP+20] demonstrate that the estimate as outlined in Equation 1 remains unbiased when $q(x) > 0$ whenever $\hat{p}(x) > 0$. If source PDFs vary per sample, then:

$$\mathbb{E}[W(x,z)] = \frac{1}{p(x_z)} \frac{|Z(x_z)|}{M} \tag{3.9}$$

where $|Z(x_z)|$ counts the source PDFs where $q_j(x_z) > 0$. This means we introduce bias if we reuse a sample x for which $q_j(x) = 0$. Equation 3.9 can be used directly for debiasing:

$$\mathbb{E}\left[W(x,z)\frac{M}{|Z(x_z)|}\right] = \frac{1}{p(x_z)} \tag{3.10}$$

which leads to the unbiased version of the estimator 3.3:

$$\langle L\rangle_{RIS}^{N,M} = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{f(x_i)}{\hat{p}(x_i)}\cdot\left(\frac{1}{|Z(x_i)|}\sum_{j=1}^{M}w(x_{ij})\right)\right) \approx L \tag{3.11}$$

Note that when combining reservoirs, we are actually using a multi-iteration RIS estimator:

$$\frac{1}{N}\sum_{i=1}^{N}\left(\frac{f(x_i)}{\hat{p}_0(x_i)}\cdot\frac{1}{|Z(x_i)|}\sum_{j=1}^{M_0}\left(\frac{\hat{p}_0(x_{ij})}{\hat{p}_1(x_{ij})}\cdot\frac{1}{|Z(x_{ij})|}\sum_{k=1}^{M_1}\left(\dots\right)\right)\right) \approx L \tag{3.12}$$

where $|Z(x_i)|$ counts the number of $\hat{p}_1(x_{ij}) > 0$ and $|Z(x_{ij})|$ counts the number of $\hat{p}_2(x_{ijk}) > 0$, etc.

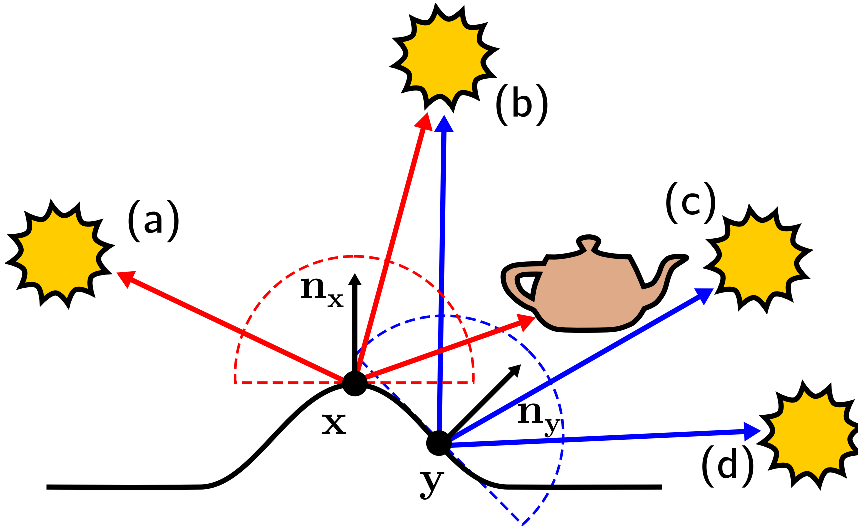


Figure 3.1 : This diagram shows how bias arises when reusing samples from neighboring reservoirs. The points $\mathbf{x}$ and $\mathbf{y}$ simply cannot generate the same samples as the visibility of the individual samples from these points is different, and the rotation of their normals $\mathbf{n_x}$ and $\mathbf{n_y}$ is also different, causing bias when reusing $\mathbf{x}$ at $\mathbf{y}$ or vice versa.

Figure 3.1 provides a more intuitive view of how bias can emerge. Shading points x and y, which belong to different pixels, cannot generate the same light samples represented by point lights in the figure. Only point x can generate a sample from light (a) since the light is located outside the hemisphere of point y defined by the normal n_y. Light (b) can generate a sample for both points, as it is located within their respective hemispheres and visible to both

points. Light (c) can generate a sample only for point y, as point x is in the shadow of the teapot. Lastly, light (d) can be sampled only by point y, as it is not within the visible hemisphere of point x. This inequality in sample generation may introduce bias into the final rendered image.

Indeed, point **x** can generate samples (a) and (b), resulting in a total of $M_{\mathbf{x}} = 2$ candidates, whereas point **y** can generate samples (b), (c), and (d), yielding a total of $M_{\mathbf{y}} = 3$ candidates. If we perform a reuse from point **y** in favor of point **x**, we would have a total of $M_{\mathbf{xy}} = M_{\mathbf{x}} + M_{\mathbf{y}} = 5$ candidates.

However, if we end up selecting the sample (a) during reuse, we introduce bias when normalizing with $M_{\mathbf{xy}}$. In this case, we need to normalize with $M_{\mathbf{x}}$ instead to ensure unbiasedness. If we neglect this, we may end up with high values of M, leading to the darkening of the resulting pixels in areas with substantial differences in depth, normal, or visibility.

Algorithm 6: Unbiased combination of multiple reservoirs

Input: Reservoirs r_i to combine and the pixels t_i they originate from.
Output: A combined reservoir s

```
1  function combineReservoirsUnbiased(t_1, ..., t_k, r_1, ..., r_k)
2      Reservoir s
3      foreach r ∈ {r_1, ... r_k} do
4          s.update(r.y, p̂_t(r.y) · r.W · r.M)
5      s.M ← r_1.M + ... + r_k.M
6      Z ← 0
7      foreach i ∈ {1, ... k} do
8          if p̂_{t_i}(s.y) > 0 then
9              Z ← Z + r_i.M
10     m ← 1/Z  s.W = 1/p̂_t(s.y) (m · s.w_sum)
11     return s
```

The unbiased reservoir combination can be seen in Algorithm 6. The changes compared to the biased reservoir combination (Algorithm 4) are highlighted in blue. Compared to the biased combination, we can see that one more for loop is needed to check whether the selected sample can be generated by the corresponding pixel t_i. For spatial reuse, this means an extra for loop over the total number of spatial samples. Note that if we want to guarantee unbiasedness even in the visibility framework, we need to perform a ray cast in Algorithm 6 on line 8 within $\hat{p}_{t_i}(s.y)$.

If we aim to utilize a biased estimator while minimizing darkening as much as possible, we must employ heuristics when choosing neighboring pixels. These heuristics could consider factors such as differences in depth, the orientation of the normals, or the similarity of materials.

Bitterli et al. [BWP+20] also illustrate that the normalization term in Equation 3.1 can be substituted with an arbitrary weight $m(x_z)$:

$$W(x,z) = \frac{1}{\hat{p}(x_z)}\left(m(x_z)\sum_{j=1}^{M} w(x_j)\right) \tag{3.13}$$

This modification permits unbiased reuse given the condition that $\sum_{i \in Z(x)} m(x_i) = 1$. This approach enables the application of Multiple Importance Sampling (MIS) - utilizing balanced heuristics for candidate PDFs.

Chapter 4

Implementation

This chapter outlines our implementation of the ReSTIR algorithm for direct illumination, grounded in insights from the original paper [BWP+20] and a subsequent study [WP21]. The latter aims to address the shortcomings, identify bottlenecks, and optimize the execution time of the initial algorithm.

Since the ReSTIR algorithm is primarily designed for real-time raytracing and allows efficient implementation for GPUs, we have decided to use DirectX Ray Tracing (DXR), a feature of the DirectX API. DXR brings to the API an abstraction over the ray tracing acceleration structure optimized for use on the GPU and provides several kinds of shaders that can be used to describe the computation flow of the ray tracing algorithm.

Given that our implementation is focused on a specific segment of the rendering process, it was fitting to opt for a rendering framework capable of handling common graphic operations, such as model loading, building, and utilizing acceleration structures. This framework should include additional auxiliary tools, such as a profiler and animation player. Consequently, we selected NVIDIA's Falcor framework, which not only supports DirectX RayTracing but is also specifically designed to rapidly prototype ray tracing algorithms.

4.1 Falcor Rendering Framework

Falcor is an open-source real-time rendering framework developed by NVIDIA for rapid prototyping and productivity enhancements in research [KCK+22]. It offers many features and is designed to abstract common graphic operations like shader compilation, model loading, and the detailed aspects of scene rendering. This design empowers developers to concentrate on the specific problem at hand.

Falcor supports real-time ray tracing through an abstraction layer over DX12 and Vulkan APIs. Beyond standalone applications, Falcor enables the creation of render passes that can be incorporated into the Render Graph System, thus facilitating a modular, flexible, and easily editable rendering pipeline. The framework encompasses several basic and advanced render passes that can be readily extended, with individual render graphs represented via a Python script. Scene scripting is another feature of Falcor, proving

useful for quickly incorporating additional components into a scene, such as lights, environment maps, cameras, animations, and other scene elements. The framework is capable of loading standard graphic formats like FBX or OBJ, including animated scenes, and supports numerous NVIDIA RTX SDKs, such as DLSS, RTXDI, RTXGI, and NRD. Additionally, the editor includes a feature for straightforward recording of screenshots and video sequences, facilitating quick and easy comparisons of results. The user interface of Falcor can be seen in Figure 4.1.

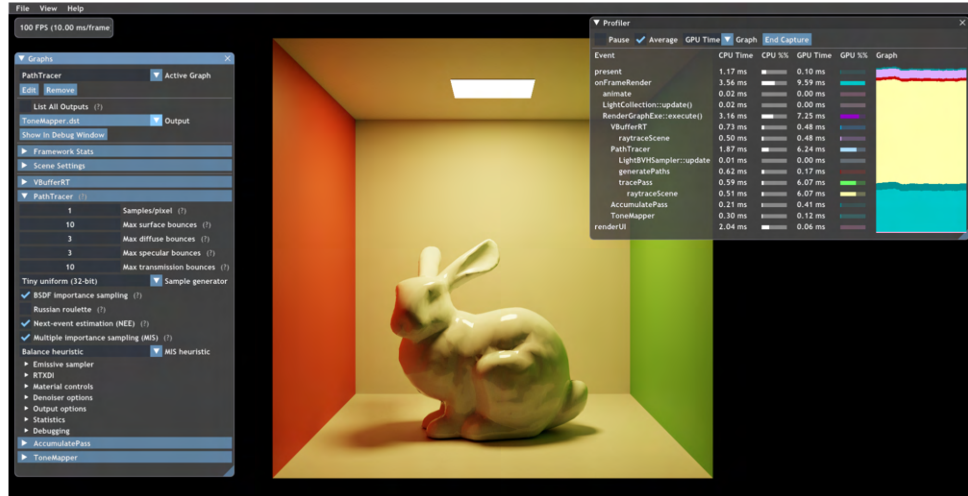


Figure 4.1 : Falcor rendering framework [KCK+22].

The features offered by the Falcor framework make it an excellent candidate for prototyping the ReSTIR algorithm. The algorithm's various components can be decomposed into render passes, accelerating debugging and development processes. The algorithm's final form can be implemented as a single render pass, providing a more efficient platform for bottleneck optimization within the computation process.

4.2 Test Scenes

For implementation and measurement, it was necessary to select test scenes comprising varying numbers of emissive triangles. The following list provides a basic description of these scenes:

Cornell Box [Bit16]: Traditionally utilized by the computer graphics community to test ray tracing algorithms, this scene consists of a room with five walls, two boxes in the middle, and a light source composed of two emissive triangles. Rendered images of this scene are often compared to photographs of the physical model, allowing for evaluating the rendering algorithm's quality. It is apt for the initial development stage as the simplest among the selected scenes.

Arcade [KCK+22]: This scene, simpler in design, features a room with an arcade machine. The machine includes several dozen emissive triangles forming a display, buttons, and a signboard. Unlike the Cornell Box scene, this scene's objects exhibit more complex and realistic materials.

Amazon Bistro [Lum17]: This collection comprises two scenes – an interior (bistro) and an exterior (street). Both scenes contain thousands of emissive triangles, multicolored and animated in the exterior scene. The implemented method can be readily tested under these conditions with an animated camera in both scenes.

BEEPLE ZERO DAY [Win19]: This scene contains a mix of glossy and diffuse materials and is illuminated by thousands of animated emissive triangles. Rapid animations of emissive objects and the camera result in swift changes in lighting conditions, making this scene a significant challenge for real-time path tracing.

Emerald Square [NHB17]: Composed of a park surrounded by roads and buildings, the scene's light sources include street lights, windows, traffic lights, and vehicles. Unique among the other scenes, it features a substantial amount of vegetation formed by dense triangle meshes.

Veach Door [Bit16]: This scene is composed of a room with a table that holds three teapots, each made of a different material. A partially opened door permits a modest amount of direct light to infiltrate the room. This particular setup poses a significant challenge for global illumination algorithms, as the majority of the scene is lit primarily by indirect illumination. This scene serves as a testing ground for our experimental implementation of the ReSTIR GI algorithm.

4.3 Rendering Pipeline

We subdivided the algorithm into several successive rendering passes, forming a rendering pipeline 4.2. This segmentation simplifies the implementation process, as each part of the pipeline can be implemented independently. It also allows for a clear definition of each render pass's inputs and outputs and facilitates the detection of potential computational bottlenecks. This pipeline aligns with the algorithm's description in the paper [BWP+20]. The rest of this section is dedicated to describing the individual components of the rendering pipeline.

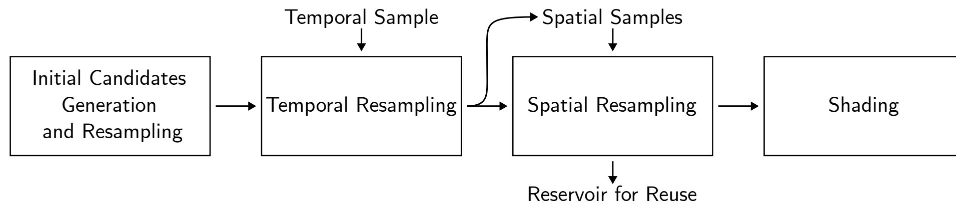


Figure 4.2 : Original ReSTIR pipeline [BWP+20].

4.3.1 V-buffer

The first segment of the rendering pipeline involves generating a visibility buffer (V-buffer). During this stage, primary rays identify intersections with scene objects, creating a visibility buffer that consolidates the object and triangle IDs into a single value. Additionally, this process stores the intersection's barycentric coordinates within the given triangle, furnishing all necessary information to fetch any other parameters about the intersection point.

Like the G-buffer, the V-buffer is part of a deferred shading strategy. As opposed to forward lighting, deferred shading reduces the number of fragment shader runs in scenes with high-depth complexity. Unlike the V-buffer, the G-buffer typically requires 16-32 bytes per visibility sample, leading to the potential wastage of texture bandwidth and storage resources. However, if a parameter is used multiple times in a pipeline, the G-buffer can serve as a handy alternative by saving the computation time of multiple fetches. Falcor also offers the flexibility to use a rasterized variant of the V-buffer and G-buffer passes, with the raytraced variant potentially being more efficient in scenes with numerous occluded objects.

We employed a ray-traced V-buffer since most required parameters are only used once during the rendering process. If specific parameters need to be used multiple times, we resort to custom buffers that allow for more compact storage.

4.3.2 Initial Candidates Generation

The initial stage of the ReSTIR pipeline involves generating the initial candidates. This stage is executed as a compute shader, with computations for each image pixel. The input for this stage is a V-buffer, which provides all the information about the surface hit by the primary ray.

The compute shader produces samples of three different types of lights—emissive triangles, environment, and analytic lights—with the number of samples for each type specified by the user. A separate reservoir is created for each light type, in which generated candidates are resampled. Consequently, each reservoir retains one sample of a given light type after this stage. These reservoirs are integrated into a single reservoir following Algorithm 4, and a visibility test is conducted to assign a zero weight to the resulting reservoir if the sample is occluded. As previously noted, this step is crucial for propagating

visibility in spatial and temporal reuse. Since this pass is implemented as a compute shader, we utilize the inline raytracing feature of DXR 1.1, which enables the casting of rays from any type of shader.

We use the source PDF $q = L_e$ to generate the candidates, implying that the probability of generating a sample is proportional to the incoming intensity. We employ an alias method [Wal77], [AMW21] to sample this discrete probability distribution, which ensures a constant lookup, unlike the commonly used inverse of the cumulative distribution function.

The alias table is precomputed on the CPU side and loaded into the GPU's global memory. This method proves advantageous when precomputation is feasible, and the distribution remains unchanged, which would otherwise necessitate the table's precomputation for each frame. The `LightSampler` class, which is used for light sampling, stores three alias tables—each designated for emissive triangles, environment maps, and analytical lights, respectively. This class enables access to the individual elements of the alias tables and also incorporates sampling methods. Additionally, the LightSampler class is designed to handle different light types and their associated properties, allowing for greater flexibility and functionality in the rendering pipeline. It simplifies the process of sampling different light sources and ensures efficient utilization of resources by managing these alias tables.

The reservoir structure looks as follows:

```
struct Reservoir
{
    MinimalLightSample sample;  ///< Output sample.
    float weightSum;            ///< Sum of weights.
    float W;                    ///< Weight of the reservoir.
    uint M;                     ///< Number of samples seen
                                ///  so far.
}
```

To enhance cache coherency and reduce bandwidth, this structure is compressed into a 16-byte `PackedReservoir` when stored in a structured buffer.

The `MinimalLightSample` structure comprises a light type, an index into a global array that stores further information about the light, and the local coordinates of the sampled point within the light (for instance, barycentric coordinates in the case of an emissive triangle). When the reservoir is packed, this structure amounts to 8 bytes per pixel.

Further information can be derived from the `MinimalLightSample` in the form of a `LightSample` structure by referring to the global array of lights:

```
struct LightSample
{
    Type lightType; ///< Type of light (Area, Distant, Point).
    float3 posDir;  ///< Position or direction
                    ///  (depends on type).
    float3 normal;  ///< Normal vector at the sample point.
```

```
    float pdf;       ///< Source PDF of the light sample.
    float Le;        ///< Emission.
}
```

In addition, this stage creates the SurfaceData structure needed for later pipeline stages to compute the target PDF for a given pixel and sample. This structure contains the position of the hit surface, the normal vector at the hit location, depth (distance from the camera), specular and diffuse weight, and specular roughness.

The resulting render, produced after executing this pass, is depicted in Figure 4.3.

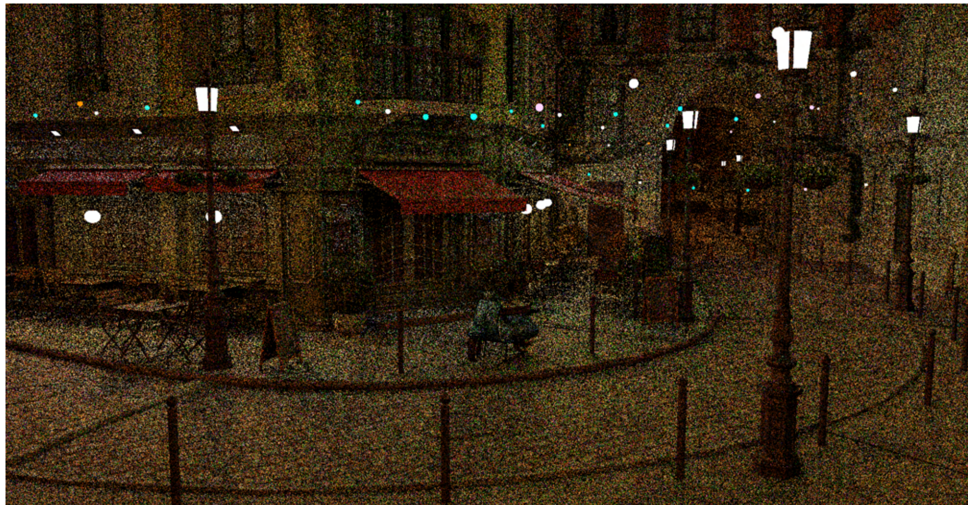


Figure 4.3 : Result after generating initial candidates and resamling.

4.3.3 Temporal Resampling

The second stage in the ReSTIR pipeline involves temporal resampling. The primary inputs for this stage are the reservoirs established in the preceding stage and those from previous frames, as depicted in Figure 4.2.

Another critical input is the 2D motion vector, which indicates the pixel's position in the previous frame. These motion vectors are generated by considering the movements of both the objects and the camera. The previous pixel's position allows us to identify the location of the reservoir from the preceding frame, which we then use for resampling.

This stage is crucial in enhancing the temporal stability of the ReSTIR algorithm. It reuses lighting information from previous frames, which can reduce noise and improve the quality of the rendered image, especially in dynamic scenes where light sources or camera views change over time.

After finding the previous pixel, we use a simple heuristic as a validity test to reduce the bias. In this test, we compare if the normal vectors of the two pixels have similar directions and if the difference in depth between the two pixels is not significant:

$$
\begin{aligned}
\tau_{normal} &< \vec{n_x} \cdot \vec{n_y} \\
\tau_{depth} &> \frac{|z_x - z_y|}{\max\left(z_x, z_y\right)}
\end{aligned}
\tag{4.1}
$$

These heuristics are similar to edge-stopping functions in filtering techniques such as SVGF [SSK+17].

As the number of samples in previous reservoirs can increase without bound, we introduce a constraint known as M-cap on the maximum value of M. This cap partially curtails the influence of temporal samples, providing new candidates with a better opportunity to be chosen during resampling. Implementing a reasonable M-cap is also necessary to limit correlations between frames. The M-cap process, illustrated in Equation 4.2, has proven convenient in our implementation, where we set the history limit to 20.

$$
previous.M = min(previous.M, current.M * historyLimit) \tag{4.2}
$$

Thus, this part of the rendering pipeline outputs the reservoir from the previous part on which temporal reuse was performed. After generating the initial candidates and applying this pass, the resulting render can be viewed in Figure 4.4.

Figure 4.4 : Illustration of reusing temporal samples.

4.3.4 Spatial Resampling

Spatial resampling, the subsequent stage in the rendering pipeline, is implemented as a compute shader, similar to temporal resampling. The inputs for this phase are derived from the samples of the preceding phase, wherein temporal reuse has been executed, as illustrated in Figure 4.2.

During this process, reservoirs within the current pixel's neighborhood are selected randomly. This neighborhood is typically set to a default radius of 30 pixels. To ensure the validity of a specific reservoir, the same heuristics applied in temporal resampling are utilized, as referenced in Equation 4.1.

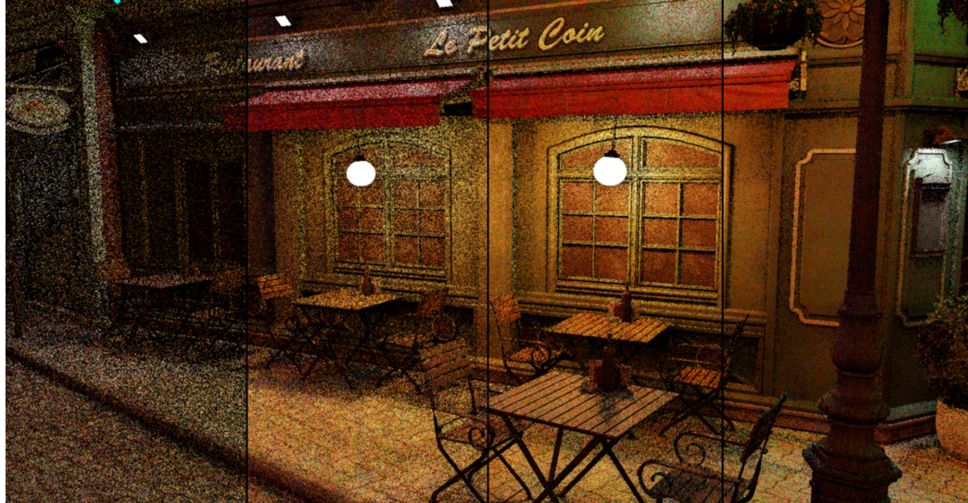


Figure 4.5 : Comparison of using different number of spatial iterations after initial candidate generation (without temporal reuse), gathering 5 neighbors at each step. Left-to-right: no spatial reuse, 1 iteration, 2 iterations, 4 iterations

Spatial resampling can be performed in multiple iterations. However, a challenge lies in the fact that each pass through this stage necessitates global synchronization, as the reservoir array from the previous iteration serves as input. This synchronization has the potential to introduce latency, yet it becomes critical in situations where the quality of samples from temporal resampling is not sufficient.

Figure 4.5 compares varying numbers of spatial resampling iterations, each selecting five neighboring samples. In this comparison, spatial resampling is applied immediately following the generation of initial candidates, with no application of temporal resampling.

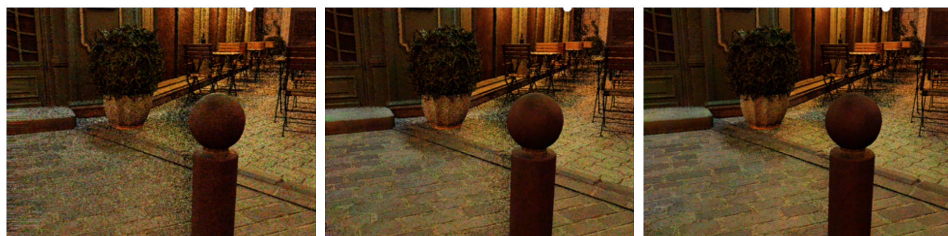


Figure 4.6 : Temporal disocclusions under motion. (Left) Temporal reuse, (center) temporal reuse and one spatial sample, (right) temporal reuse and five spatial samples.

Figure 4.6 illustrates the impact of increasing spatial samples on mitigating temporal disocclusion. It is evident that even a single spatial sample significantly reduces noise compared to relying solely on temporal reuse. Although

using five samples further diminishes the noise, it necessitates extended computation time, which may not substantially enhance the quality of the results.

4.3.5 Shading

The final stage of the rendering pipeline is shading. It takes as input a direct illumination sample generated from the preceding stages of the ReSTIR pipeline. For shading, we utilize Falcor's standard surface Bidirectional Scattering Distribution Function (BSDF), which comprises the following lobes:

- Delta reflection (ideal specular reflection)
- Specular reflection utilizing a GGX microfacet model
- Diffuse reflection using Disney's diffuse Bidirectional Reflectance Distribution Function (BRDF)
- Delta transmission (ideal specular transmission)
- Specular transmission using a GGX microfacet model
- Diffuse transmission

The BSDF is a linear combination of these lobes.

This shading phase also incorporates a visibility test for the sample in the reservoir. As this component is also implemented as a compute shader, we utilize the inline raytracing feature offered by DXR 1.1.

4.4 Target PDF

A crucial factor influencing the quality and performance of the ReSTIR method is the selection of the target Probability Density Function (PDF), denoted as $\hat{p}$. This function is evaluated at every resampling step, including the initial candidate generation phase and during spatial and temporal reuse. Given that this function does not require normalization, we have considerable flexibility in its choice. Bitterli et al. [BWP+20] proposed the use of $\hat{p} = L_e \cdot G \cdot \rho$.

The computation of L_e is inexpensive since this information is directly included in the reservoir sample and does not depend on material properties or geometric characteristics.

The geometric term, denoted as G, requires minor computation. To calculate G, one only needs the cosine of the light normal with the vector pointing from the light source to the surface point and the inverse-square law:

$$G = \frac{-\vec{v} \cdot \vec{n}_{light}}{d^2} \tag{4.3}$$

The ρ function can take various forms as it often determines the overall complexity of the $\hat{p}$. Rather than computing the complete BSDF, we utilize a simplified model based on the Trowbridge–Reitz (GGX) microfacet distribution. We can afford to perform the complete BSDF calculation if we render offline. However, a simpler BSDF model can be employed in case of performance constraints.

For the biased version of ReSTIR, we utilize the unshadowed path contribution inside $\hat{p}$, as casting rays for shadow computation is computationally expensive and can take significant rendering time. However, to ensure complete unbiasedness, we need to verify the visibility of the samples during resampling, which requires ray casting.

4.5 Ray Budget

Bitterli et al. [BWP+20] propose storing 4 reservoirs per pixel ($N = 4$). This results in a total of 5 shadow rays per pixel (1 for initial candidates and 4 during shading). In today's real-time applications, developers have a budget of ≤ 1 rays per pixel. Wyman et al. [WP21] conclude that reducing the number of reservoirs to $N = 1$ does not significantly reduce image quality. At $N = 4$, the samples of all the reservoirs were often identical, especially in areas of the scene where shadows are present, which are the places where we would expect that a higher number of samples would help reduce noise. Based on these observations, we also choose $N = 1$ with a total ray budget of 2. However, as previously mentioned, for the unbiased variant of the ReSTIR algorithm, extra ray casting is needed during temporal and spatial reuse.

4.6 Improving Cache Coherency

The ReSTIR algorithm theoretically guarantees constant time complexity. Wyman et al. [WP21] discovered this is not the case when dealing with scenes that incorporate numerous light sources. We did performance tests on scenes with many emissive triangles, and in our implementation, we noticed that the lighting cost varied by up to 30× between scenes.

This difference in performance across different scenes can be attributed to distinct caching behaviors. In the initial candidate sampling phase, light samples are chosen randomly. For instance, this method uses cache coherently for a scene like the Cornell Box, which only contains two emissive triangles. In contrast, for scenes with complex lighting configurations, such as Emerald Square, which has 89 thousand emissive triangles, the sampling process leads to a significant number of cache misses.

4.6.1 Light Tiles

Wyman et al. [WP21] proposed using a degenerate RIS estimator that introduces random stratification and divides the sampling process into two

steps. In the first step, several light subsets $\mathbb{S}$ (termed as *light tiles*) of the scene lights L are created, meaning many cache misses occur in this step. In the second step, each pixel samples the initial candidates from the selected subset, significantly reducing the occurrence of cache misses. This approach shifts the incoherent memory access from the inner loop of the algorithm to the outer loop. However, to eliminate the incoherence, selecting a subset size $|S| \ll |P|$ is essential, where $|P|$ denotes the number of pixels.

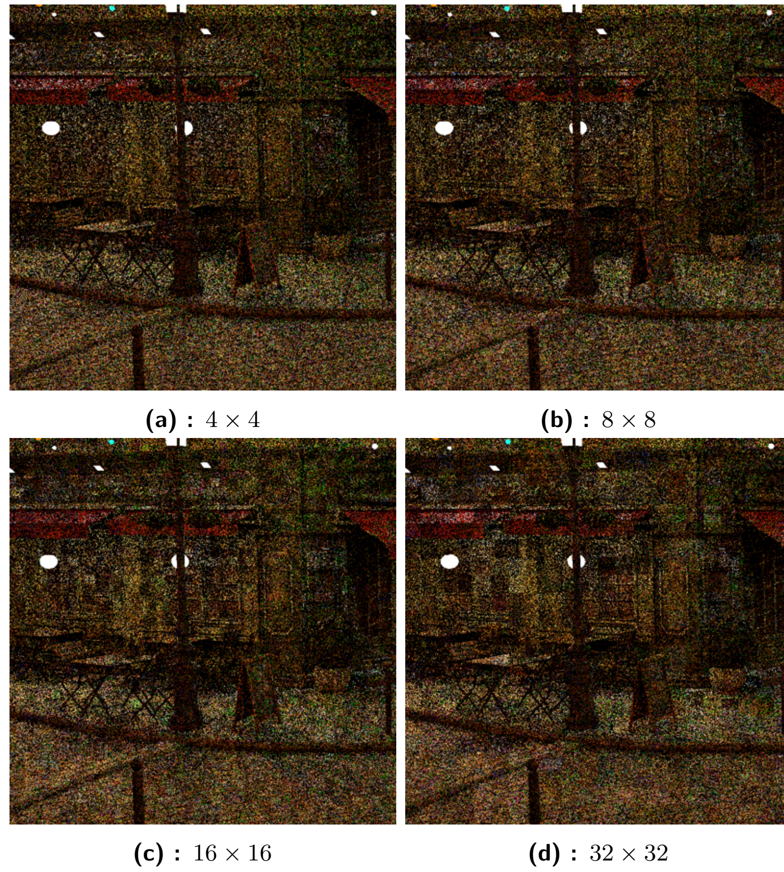
Algorithm 7: Creating light tiles

```
   Data: Set of scene lights L
 1 function reservoirReuse(...)
 2     // Create light tiles
 3     for i ← 1 to 𝕊 do
 4         generate |S_i| light samples ~ q from L
 5         S_i ← set of |S_i| samples
 6     reservoirs ← new Array[ImageSize]
       // Generate initial candidates
 7     foreach pixel t ∈ I do
 8         S ← randomSubsetForTile(t, 𝕊)
 9         reservoirs[t] ← RIS(t, S)
10     ⋮
11     return reservoirs
12 function RIS(t, S)
13     Reservoir r
14     for i ← 1 to M do
15         pick x_i from S uniformly
16         r.update(x_i, p̂_t(x_i)/p(x_i))
17     r.W = 1/p̂_q(r.y) · (1/r.M · r.w_sum)
18     return r
```

Even better coherence can be achieved at the level of warps, which are groups of threads on the GPU running simultaneously. It is ideal for each warp to have its individual threads accessing a similar section of memory, as the warp runs until it needs to wait for data (from device memory); otherwise, another warp has a turn. The image can be divided into tiles of size $k \times k$ (*pixel tiles*), and each tile is assigned a random subset $S \in \mathbb{S}$. This process can be seen in Alg. 7; changes compared to Alg. 3 and Alg. 5 are highlighted in blue.

We implemented the creation of light tiles using a compute shader, where each thread samples light according to q. The total number of threads equals the product of the number of light tiles $|\mathbb{S}|$ and the number of samples in each light tile S_i. The number of shader executions is typically significantly lower than per-pixel executions, contributing to this pass's rapidity. Reasonable parameters may be, for example $\mathbb{S} = 256$ and $S_i = 1024$.

Figure 4.7 showcases the impact of varying pixel tile sizes after generating



(a) : 4×4 **(b) :** 8×8 **(c) :** 16×16 **(d) :** 32×32

Figure 4.7 : Illustration of using light tiles with different pixel tile sizes after initial candidate generation (without spatial and temporal reuse). Parameters: $\mathbb{S} = 256$, $S_i = 1024$

initial candidates. It is evident that as the tile size increases, the sampling correlation between pixels within the same tile also increases. For instance, some light tiles can receive extremely bright light samples, while adjacent tiles receive considerably darker ones. This can result in a visible edge delineating the boundary between the tiles. Spatio-temporal reuse subsequently smooths these edges, but to prevent flickering effects, we recommend maintaining the tile size for an image resolution of 1920×1080 to be smaller than 16×16.

4.7 True Spatiotemporal Reuse

Wyman et al. [WP21] introduced the concept of eliminating the global synchronization between temporal and spatial reuse in the original pipeline, as shown in Figure 4.2. The challenge here is that, in order to perform spatial reuse, it is necessary to wait until all threads from the previous pipeline stage have completed their operations to access neighboring reservoirs. However, instead of waiting for the current reservoirs, we can utilize the reservoirs from the previous frame for spatial reuse, implementing what is referred to

as *True Spatiotemporal Reuse*. This substitution eradicates the single global synchronization from the original pipeline, leading to a new pipeline, depicted in Figure 4.8, that lacks global synchronization and can be implemented as a single kernel.

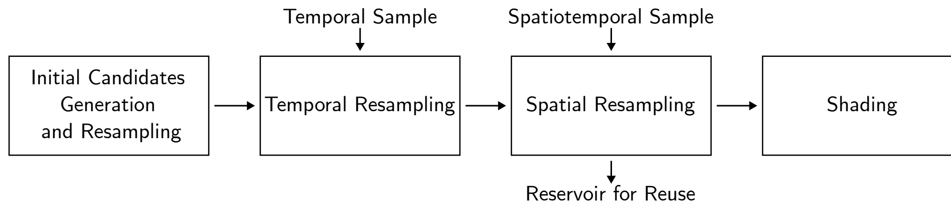


Figure 4.8 : True Spatiotemporal Reuse pipeline [WP21]

The spatial samples are thereby delayed by one frame, which is generally imperceptible. However, during fast animations or rapid camera movements, when a new scene segment is unveiled, it might be noticeable that the initial candidates fail to provide high-quality samples, leading to a noisy portion of the image. In scenarios where rapid image convergence is required, it remains necessary to reintroduce global synchronization, using the current reservoirs for spatial reuse.

4.8 Decoupled Shading

Wyman et al. [WP21] came up with another alternative version of the pipeline that preserves the overall ray budget (2 rays per pixel) while providing a better quality of the resulting image. In this pipeline, shading is separated from reuse 4.9. After generating the initial candidates, we perform a visibility test for the resulting candidate and find the temporal and spatiotemporal samples, so we have three samples that we can use for shading instead of one. However, a visibility test needs to be performed for the temporal and spatiotemporal samples giving us a total of three shadow rays. After shading, these three samples are reused as in the original pipeline and stored as temporal reservoirs for the next frame.

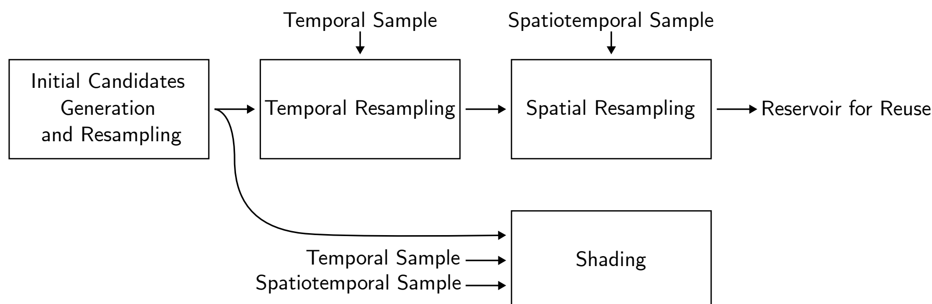


Figure 4.9 : Decoupled pipeline [WP21].

This approach dramatically reduces the black pixels representing failed

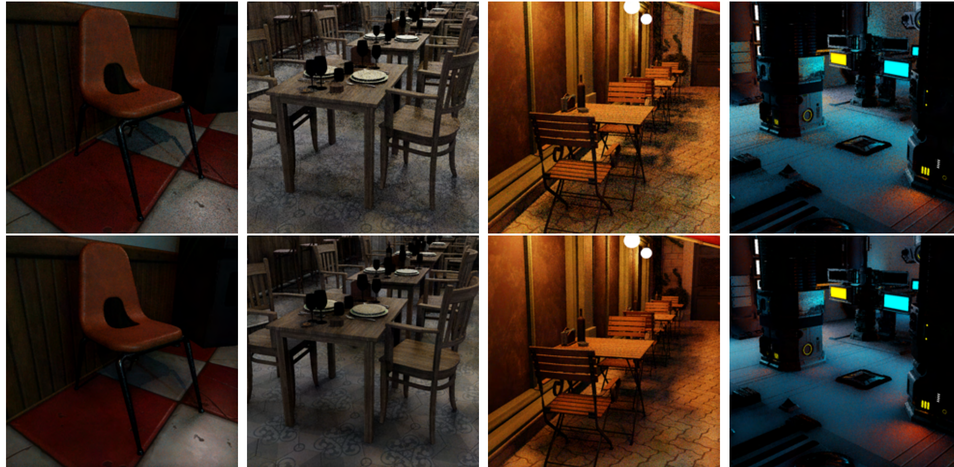


Figure 4.10 : Differences of using Original pipeline 4.2 (first row) and Decoupled pipeline 4.9 (second row).

samples that cannot be used due to visibility. The effect of decoupling shading from resampling can be seen in Figure 4.10.

4.8.1 Cheaper Visibility

The number of shadow rays per pixel in this pipeline can be reduced under certain conditions or by accepting a minor decrease in quality.

When the scene is static, the visibility of the temporal sample does not change, allowing us to save one shadow ray per pixel. If the scene is animated, reusing visibility without a visibility test introduces a one-frame shadow lag, which is almost imperceptible. Under these circumstances, the shadow ray can still be omitted.

Omitting the visibility test for the spatiotemporal sample can result in a noticeable decrease in quality. However, if we aim to trim our ray budget further, we can designate a distance for the spatiotemporal sample beyond which we will not cast the shadow ray. This approach is based on the rationale that the current pixel's close neighbors are less likely to be shadowed than distant ones. Doing so prevents potential light leakage in shadowed areas and can save many shadow rays.

4.9 Checkerboard Rendering

Generating initial candidates represents the most computationally demanding part of the ReSTIR algorithm. Bitterli et al. [BWP+20] suggested exploring the use of reservoir resolutions lower than the image resolution, such as assigning a single reservoir for every four neighboring pixels. Given that the rest of the pipeline does not directly depend on the origin of the initial candidates, it is feasible for some pixels to bypass candidate generation and borrow reservoirs from their neighboring pixels. A rendering technique known

as checkerboard rendering can be employed here. This technique is based on partitioning the screen pixels following the color pattern of a checkerboard. The first type of pixels will go through the entire pipeline and carry out most of the work, while the second type will only perform temporal reuse, spatial reuse, and shading. Moreover, the pixels alternate the workload after each frame. Therefore, in an odd frame, a pixel goes through the entire pipeline, whereas in an even frame, it only goes through a part of it.

With this approach, only half of the pixels must generate initial candidates. This signifies saving 1/4 of the total shadow rays for the original pipeline. If we aspire to push this further, we can adopt an even sparser checkerboard rendering, generating initial candidates for only one out of every four pixels.

4.10 Distribution of Samples among Light Types

ReSTIR offers the capability to sample lights irrespective of their type. It can sample emissive geometry, analytical light sources, environment maps, and even light probes. For scenes that include more than one of these light types, it becomes necessary to determine how many samples will be drawn from each type of light set. To maintain real-time frame rates, we choose fewer than 36 light samples and further need to decide how these samples will be distributed among the various light types.

Our implementation enables the sampling of three types of light sources - analytical, emissive triangles, and environment maps. We empirically determined the distribution of light samples based on the following considerations. In scenes intended for realistic rendering, there is usually only one analytical light source - a directional light. This represents an infinitely distant light source and simulates, for instance, the Sun. Therefore, we consider selecting only one initial candidate from the analytical light sources optimal.

We need to allocate more samples for sampling the emissive geometry to achieve high-quality results. Emissive geometry is more challenging to sample than other light types and is widely used in scenes aimed at realistic rendering. These scenes often comprise tens of thousands of emissive triangles. To stay within our sampling budget, we found allocating 24 samples for emissive triangles beneficial.

We distribute the remaining samples (8-11) for sampling environment maps. This number is sufficient, even for environment maps with significant intensity differences.

In our current implementation, the sample distribution can be manually configured. Nevertheless, it is conceivable to implement a heuristic that adapts the number of samples based on the content of the light sources within the scene. However, this approach necessitates further investigation and analysis.

4.11 Denoising and Upscaling

Spatio-temporal denoisers require an input signal with the lowest possible variance and a signal that can converge in the shortest time possible. To evaluate our implementation with denoisers, we used the SVGF denoiser, executed as a separate render pass in Falcor. This denoiser requires input from the G-buffer, such as position, normal vector, depth, and motion vector, as well as results from the path tracer, such as direct/indirect lighting and albedo.

As mentioned, Falcor also includes a DLSS SDK intended to upscale the final image. Consequently, we have integrated upscaling into our implementation as another alternative rendering pipeline, applying it directly after denoising. This allows us to create a V-buffer/G-buffer, execute the ReSTIR algorithm, perform the denoising all at a lower resolution (1920×1080), and then upscale the image to a higher resolution (3840×2160). This high-level pipeline is illustrated in Figure 4.11.

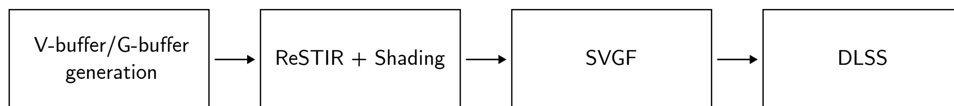


Figure 4.11 : High-level pipeline with integrated denoising and DLSS.

4.12 ReSTIR GI

The ReSTIR algorithm can also be extended for global illumination. This algorithm variant is called ReSTIR GI and is described in a paper by Ouyang et al. [OLK+21]. Out of curiosity, we tried to extend our implementation with this algorithm.

The core distinction between the ReSTIR for direct illumination (ReSTIR DI) and ReSTIR GI lies primarily in the samples' structure and the initial candidates' creation process. This approach generates initial samples by sampling random directions and tracing rays to find the closest intersections instead of generating random samples on the lights in the scene. Reflected radiance is computed at these intersections with path tracing. The sample structure in the context of ReSTIR GI is represented as follows:

```
struct SampleGI
{
    float3 visiblePoint;    ///< Visible point.
    float3 visibleNormal;   ///< Surface normal of the visible
                            ///  point
    float3 samplePoint;     ///< Sample point.
```

```
    float3 sampleNormal;     ///< Surface normal of the sample.
    float3 Le;               ///< Outgoing radiance at sample
                             ///  point.
}
```

The `visiblePoint` in this structure denotes the point on the surface that is observable from the camera, which is also the sample's origin. Alongside this point, the normal at this location is stored as well. On the other hand, the `samplePoint` refers to the intersection of the ray that originates from the `visiblePoint`, which is stored concurrently with its respective normal. The total radiance is then computed from this `samplePoint`, using path tracing with an arbitrary number of bounces and stored in the `Le`.

In the ReSTIR GI algorithm, generating each candidate necessitates casting a ray. Therefore, it is recommended to generate only one candidate at this stage. In our implementation, this stage is termed as the Trace pass. We use path tracing with one or two bounces and next event estimation (NEE) to compute the total radiance at the sample point.

Subsequently, we execute a temporal reuse process, closely mirroring the operations of ReSTIR DI. The outcome of this pass are temporal reservoirs.

Once the temporal reuse is completed, we move on to the spatial reuse pass. Here, it is crucial to adjust the target PDF of the spatial samples, given that the source PDFs of each sample vary. This variation stems from the fact that the sampling scheme is based on the position of the visible point and the normal at that point. Such a correction was not required for ReSTIR DI because, in that case, the lights were directly sampled, irrespective of the pixel's local geometry.

If we aim to reuse a sample from pixel q at pixel r, we must transform its solid angle PDF to match the solid angle of the current pixel. We can accomplish this by dividing the target PDF of the pixel q by the Jacobian determinant:

$$|J_{q\to r}| = \frac{|cos(\phi_2^r)|}{|cos(\phi_2^q)|} \cdot \frac{\|x_1^q - x_2^q\|^2}{\|x_1^r - x_2^q\|^2} \tag{4.4}$$

where x_1^q and x_2^q are the first and second vertex of the reused path, x_1^r is the visible point from the destination pixel, and ϕ_2^q and ϕ_2^r are the angles formed by the vectors $x_1^q - x_2^q$ and $x_1^r - x_2^q$ with the normal at x_2^q [OLK+21].

We adopt uniform hemisphere sampling as our primary sampling technique, aligning with the methodology outlined by the authors of the original paper, rather than resorting to cosine-weighted BSDF sampling. Our decision is rooted in the observation that cosine-weighted sampling is less inclined to generate directions at grazing angles. While this might be advantageous under specific circumstances, it can also produce noisy results due to the disproportionate focus on directions aligned closely with the surface normal.

Uniform hemisphere sampling, on the other hand, provides a comprehensive and evenly spread coverage over the hemisphere, which can significantly reduce the potential for noise in the final rendered images by not overlooking illumination from light sources situated at grazing angles.

Since this algorithm only utilizes a few initial candidates, employing a greater number of spatial samples in multiple iterations is recommended. In our implementation, we opted for 5 spatial samples across 2 iterations. To ensure satisfactory convergence and allow the substitution of existing samples with new ones, we applied an M-cap of 30 for temporal reservoirs and 300 for spatial reservoirs.

In our implementation, we opt for a spatial radius equivalent to 10% of the image resolution to select spatial candidates. In situations where the selection of spatial candidates is unsuccessful, we proceed by reducing this radius by half. The rationale behind choosing a larger radius lies in the fact that generally speaking, changes in indirect illumination are relatively subtle compared to those in direct illumination.

On the recommendation of the original ReSTIR GI paper [OLK+21], we chose the simpler $\hat{p} = Le$ as our target PDF, disregarding the BRDF of a specific surface. This decision ensures that the reservoir weights are not influenced by the view vector, which would otherwise impact the BRDF, and there would not be effective reuse of samples on specular surfaces.

Chapter 5

Results

In this chapter, we carry out both a qualitative and performance-based evaluation of the ReSTIR algorithm as implemented as part of this thesis. This includes a comprehensive analysis of the visual quality of the rendered images, assessing factors such as lighting accuracy and noise levels. We also examine the performance of our implementation in terms of computational speed and efficiency, focusing on scalability across various scene complexities. Additionally, we compare our results with those obtained from traditional rendering techniques to highlight the advantages and potential limitations of the ReSTIR approach. This analysis allows us to draw meaningful conclusions about the effectiveness of the ReSTIR algorithm in real-time rendering scenarios.

	Cornell Box	**Arcade**	**Bistro Interior**
Scene Info			
Scene Triangle Count	36	7,798	1,320,323
Emissive Triangle Count	2	124	3,576

	Bistro Exterior	**Zero-Day**	**Emerald Square**
Scene Info			
Scene Triangle Count	2,832,120	6,076,464	10,046,405
Emissive Triangle Count	20,638	10,965	89,279

Table 5.1 : Scene information.

In Table 5.1, we present information relevant to our following measurements about the scenes described in Chapter 4. This data indicates the complexity of each scene, both in terms of geometry and lighting.

5.1 Qualitative Results

To evaluate the qualitative results, we compare our results with reference images, the emissive power sampler, and the Light BVH method, detailed in Chapter 2.

Figure 5.1 : Comparison of the results of the ReSTIR algorithm when sampling the emissive geometry (top) and sampling the environment maps (bottom).

Figure 5.1 presents the outcome of employing the Original ReSTIR pipeline 4.2 on the Bistro scene. In the top part of the figure, the night scene is lit exclusively by the emissive geometry. In contrast, the scene is illuminated solely by the environment map in the bottom part. In both cases, 32 initial candidates are generated, upon which resampling is performed, followed

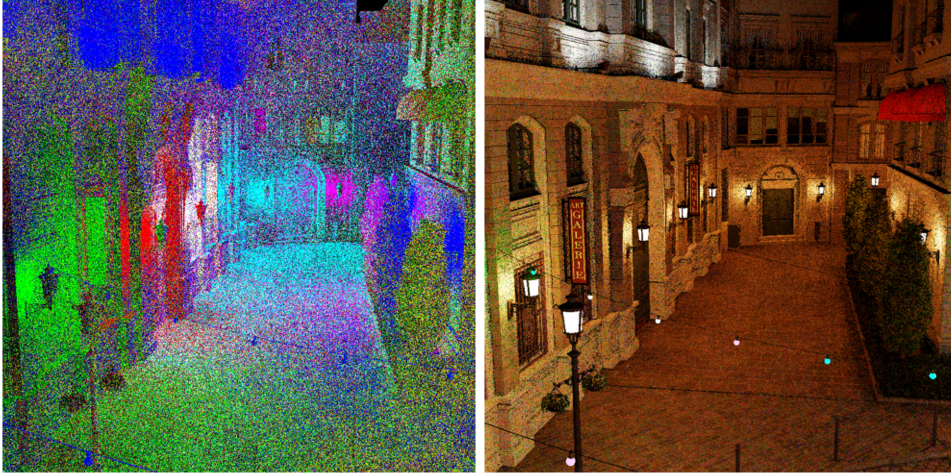


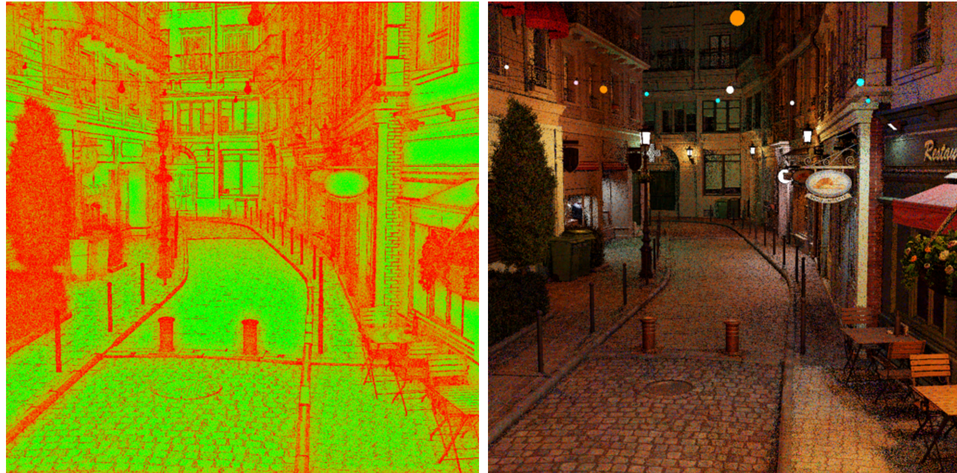
Figure 5.2 : Visualization of the origin of light samples for each individual pixel. The colors of the pixels are assigned based on the indices of the light sources, utilizing a low discrepancy RGB sequence.

by a visibility test on one selected sample. Subsequently, temporal reuse is conducted conventionally with the history limit of the past 20 frames, succeeded by a single iteration with a single spatial sample. The spatial sample is selected within a thirty-pixel radius, and a heuristic is used for its selection, which considers the angle between the surface normals and the distance of the surface from the camera. Note that this is a biased variant of the Original pipeline 4.2 that employs only two shadow rays per pixel. More shadow rays would be required for an unbiased version. The biased version might appear slightly darker in the results compared to the unbiased version, an issue that can be addressed by refining the rules of the spatial sampling heuristics.

In Figure 5.2, we can see how the algorithm performs in choosing the light samples. The individual pixels are colored here according to the light index to which the selected sample belongs. We can see that the algorithm performs very well in selecting light samples in the neighborhood of the light sources and concerning their visibility.

Figure 5.3 presents a visualization of the number of spatial samples successfully identified out of five. Green signifies more successfully detected spatial samples, while a red represents a low number. It can be observed that the identification of a new sample is more successful in geometrically continuous regions, where there is no significant variation in depth or normal vectors between surrounding pixels. Spatial resampling is less successful for vegetation, which in this context consists of dense geometry where the normal vectors between neighboring pixels are likely to differ substantially. Some houses in the scene are visible at steep angles, resulting in considerable depth variation between surrounding pixels, contributing to less successful spatial resampling.

In Figure 5.4, we observe a comparison between the Light BVH sampling algorithm and the ReSTIR algorithm. Both results were obtained within the



The green color represents a high number of successfully identified spatial samples, whereas the red color indicates a low number.

Figure 5.3 : Visualization of the count of resampled spatial samples

same time frame. With Light BVH, recognizing the contours of some objects becomes challenging due to the significant amount of noise. Conversely, the ReSTIR algorithm converges noticeably faster, yielding an image in which the scene's structure can be clearly discerned. Moreover, the ReSTIR algorithm performs better in handling complex light interactions, providing cleaner, more accurate results in areas with intricate lighting conditions. This further emphasizes the efficacy of the ReSTIR approach in maintaining image quality while ensuring faster convergence.

The application of denoisers is standard practice in real-time raytracing scenarios, given that they significantly enhance the visual quality by reducing the noise generated by the sampling process. To demonstrate their usefulness, we employed the SVGF algorithm, a highly effective spatio-temporal denoising algorithm, on the output of our ReSTIR implementation. This approach leverages temporal coherence and SVGF's iterative process to smooth out the noisy estimates and achieve a more visually pleasing result.

The outcomes of this process can be seen in Figure 5.5. Here, the SVGF denoiser significantly enhances the clarity and detail of the image. It is important to note that while the denoiser substantially improves the visual output, it respects the lighting conditions and does not artificially alter the scene's natural lighting distribution, which is a testament to the effectiveness of our ReSTIR implementation. In this instance, the parameters for the ReSTIR algorithm remain the same as in the case illustrated in Figure 5.1.

To evaluate image quality, we employed the perceptual metric ꟻLIP [ANA+20], which affirms that we uphold or enhance quality compared to other sampling methodologies. Figure 5.6 presents the perceptual image error produced by ꟻLIP, illustrating the differences between the biased and unbiased versions of the Original pipeline relative to the reference image. It is apparent that the most significant disparities occur in locations where

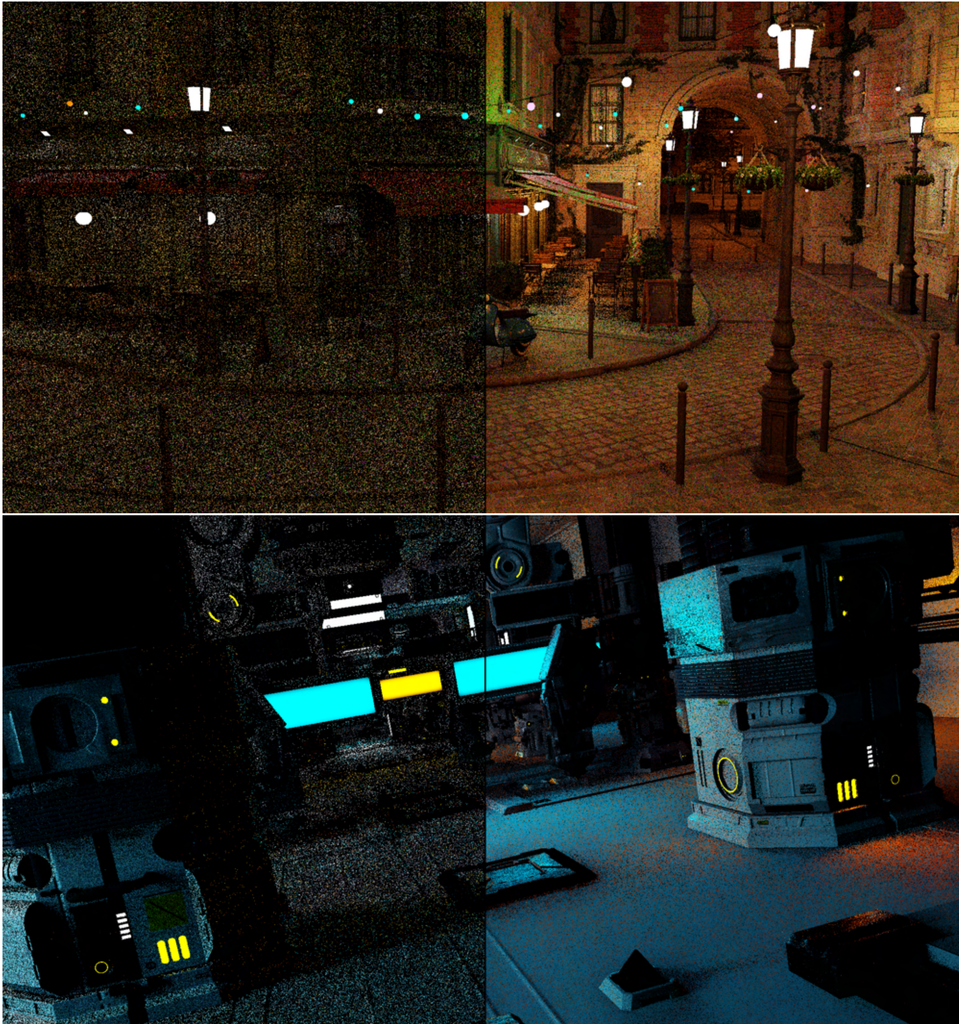


Figure 5.4 : Equal-time comparison of Light BVH (left part of the pictures) and ReSTIR 4.2 (right part of the pictures).

visibility determination is challenging, such as beneath the awnings of the bistro. These areas experience a darkening effect, which is further pronounced in the biased variant as we do not carry out correct normalization while computing the reservoir weight in this case.

Table 5.2 presents a comparative analysis of the emissive power sampler, light BVH sampler, Original pipeline (both biased and unbiased), and the decoupled pipeline (biased) across different scenes, employing the RMAE and ꟻLIP metrics. For both metrics, the smaller the value, the smaller the error between the compared image and the reference. In all cases, the ReSTIR algorithm demonstrates a significant advantage over the initial two methods, underlining its effectiveness.

We observe that optimizing the Original pipeline using the checkerboard rendering technique does not significantly reduce the visual quality, as these metrics indicate. However, the resulting speedup of the pipeline can be

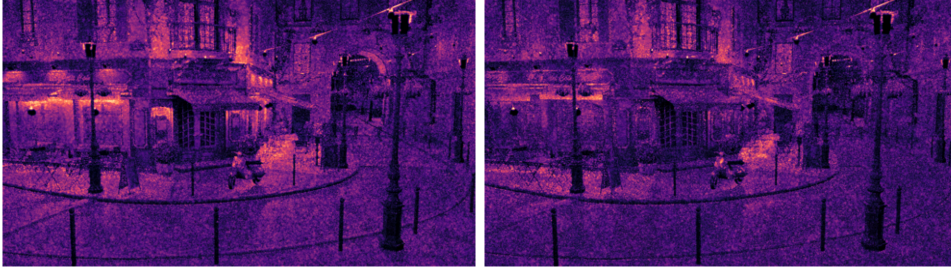
Figure 5.5 : Application of the SVGF denoiser [SSK+17] to the output of the ReSTIR algorithm.

substantial, as we will see in the next section.

The Decoupled pipeline enhances the image quality compared to the Original pipeline. This improvement is attributed to the fact that we shade three samples (initial, temporal, and spatiotemporal) in a single pass rather than just one.

The Unbiased Original pipeline exhibits the highest image error among all ReSTIR pipelines. Although this pipeline converges more slowly, it ultimately achieves the correct result, unlike its biased counterparts. The unbiased nature of the pipeline means it does not cut corners for speed, striving instead for accuracy, even if that means slower convergence. These findings underscore the trade-offs in balancing speed and accuracy when implementing the ReSTIR pipeline for real-time rendering.

Rendered images for each pipeline across all scenes, along with reference images and error visualization via the ꟻLIP, are included in the appendix of



(Left) Biased version of the Original pipeline. (Right) Unbiased version of the Original pipeline.

Figure 5.6 : Perceptual image error produced by ꟻLIP [ANA+20]

this thesis.

5.2 Performance Results

We have organized the performance measures into three separate tables. Measurements were primarily performed on the RTX 4070Ti, and for comparison, we also utilized the Quadro RTX 3000 Mobile laptop GPU to illustrate performance on portable devices.

Table 5.3 compares the total lighting computation time across four different ReSTIR pipelines. We present a more detailed measurement for the biased Original pipeline, highlighting the computational demands of individual render passes.

Indeed, generating initial candidates is the most computationally intensive part of the entire pipeline. This is reflected in the fact that a significant portion of the optimizations were applied to this stage. The time complexity of other pipeline parts does not dramatically increase with the scene's complexity. However, the time complexity of initial candidate generation increases, as it is directly related to the number of lights in the scene. By optimizing cache access with light tiles, as described in the previous chapter, we successfully managed to reduce the computation time of this render pass significantly. This was achieved in exchange for adding a new render pass for light tile creation, which did not exceed 0.03 milliseconds across all scenes on the RTX 4070 Ti. The computation time for Initial Candidates now demonstrates minimal scaling with the scene's complexity when processed on a desktop GPU. For a laptop GPU, however, this is not the case; the computation time for this render pass significantly increases depending on the scene complexity. We attribute this behavior to the difference in L1 and L2 cache sizes. The RTX 4070 Ti features 128 KB L1 cache and 48 MB L2 cache, whereas the Quadro RTX 3000 Mobile has a smaller 64 KB L1 cache and 3 MB L2 cache.

The efficiency of the ReSTIR algorithm primarily stems from its spatiotemporal reuse, which in most cases does not incur more cost than the creation of initial candidates. The computational time for these two passes is not as influenced by scene complexity but more by set parameters such as the

	Cornell Box	Arcade	Bistro Interior
Relative Mean Absolute Error (RMAE)			
Emissive Power Sampler**	0.0962	0.3785	1.0841
Light BVH Sampler**	0.0961	0.4723	1.0792
Original Pipeline*	0.0222	0.0540	0.2572
Original Pipeline w/ Checkerboard*	0.0201	0.0512	0.2501
Decoupled Pipeline*	0.0263	0.0508	0.2491
Original Pipeline**	0.0371	0.0501	0.2681
ꟻLIP [ANA+20]			
Emissive Power Sampler**	0.0455	0.0940	0.4327
Light BVH Sampler**	0.0457	0.1101	0.3551
Original Pipeline*	0.0227	0.0255	0.1559
Original Pipeline w/ Checkerboard*	0.0258	0.0279	0.1582
Decoupled Pipeline*	0.0211	0.0230	0.1552
Original Pipeline**	0.0399	0.0225	0.1346

	Bistro Exterior	Zero-Day	Emerald Square
Relative Mean Absolute Error (RMAE)			
Emissive Power Sampler**	1.0256	0.8843	0.6841
Light BVH Sampler**	1.1436	0.8135	0.8081
Original Pipeline*	0.5548	0.3832	0.2473
Original Pipeline w/ Checkerboard*	0.5598	0.3854	0.2499
Decoupled Pipeline*	0.5097	0.3261	0.2471
Original Pipeline**	0.5861	0.4132	0.2752
ꟻLIP [ANA+20]			
Emissive Power Sampler**	0.6347	0.3715	0.3666
Light BVH Sampler**	0.5284	0.3259	0.3110
Original Pipeline*	0.2469	0.1540	0.1582
Original Pipeline w/ Checkerboard*	0.2488	0.1582	0.1625
Decoupled Pipeline*	0.2369	0.1532	0.1519
Original Pipeline**	0.2213	0.1432	0.1322

Table 5.2 : Image Error versus Offline Reference. ReSTIR setup: Number of initial candidates = 32, Temporal history limit = 20, Number of spatial samples = 1, Number of spatial iterations = 1, Spatial neighborhood radius = 30px, Pixel tile size = 8×8, $|S_i| = 1024$, $|\mathbb{S}| = 128$, Normal threshold = 0.9, Depth threshold = 0.1.
* Biased, ** Unbiased

number of iterations or spatial samples. In the previous chapter, we noted that a small number of spatial samples and a single iteration are typically sufficient. This is the reason why we keep the number of spatial samples to a minimum in this analysis.

The subsequent pipeline maintains the same architecture as the Original

	Cornell Box	Arcade	Bistro Interior
Original Pipeline* 4.2			
Creating Light Tiles	0.01/0.03	0.01/0.10	0.02/0.16
Initial Candidates	0.47/1.53	1.05/4.47	1.13/4.49
Temporal Reuse	0.42/0.36	0.49/1.07	0.49/1.70
Spatial Reuse	0.22/0.34	0.31/1.34	0.31/1.73
Shading with Visibility	0.15/0.62	0.36/1.83	0.45/2.67
Total Lighting	1.27/2.88	2.22/8.81	2.40/10.75
Original Pipeline 4.2 **with Checkerboard***			
Total Lighting	1.01/2.55	1.87/7.55	1.98/9.15
Decoupled Pipeline* 4.9			
Total Lighting	0.84/3.09	2.04/9.96	2.29/11.33
Original Pipeline** 4.2			
Total Lighting	1.17/3.40	2.58/12.08	2.94/16.03

	Bistro Exterior	Zero-Day	Emerald Square
Original Pipeline* 4.2			
Creating Light Tiles	0.02/0.09	0.01/0.07	0.03/0.11
Initial Candidates	1.46/11.63	1.10/9.12	2.54/18.96
Temporal Reuse	0.46/2.00	0.48/1.07	0.49/1.71
Spatial Reuse	0.31/1.92	0.31/1.23	0.33/1.56
Shading with Visibility	0.71/3.88	0.43/2.77	1.78/11.95
Total Lighting	2.96/19.52	2.33/14.26	5.17/34.29
Original Pipeline 4.2 **with Checkerboard***			
Total Lighting	2.41/15.29	1.96/10.90	4.02/25.40
Decoupled Pipeline* 4.9			
Total Lighting	2.02/19.75	2.21/13.67	4.88/30.22
Original Pipeline** 4.2			
Total Lighting	4.06/27.75	2.93/19.48	7.93/55.49

Table 5.3 : Performance results of various ReSTIR pipelines. Times in milliseconds on RTX 4070 Ti / Quadro RTX 3000 Mobile. ReSTIR setup: Resolution 1920×1080, Number of initial candidates = 32, Temporal history limit = 20, Number of spatial samples = 1, Number of spatial iterations = 1, Spatial neighborhood radius = 30px, Pixel tile size = 8×8, $|S_i| = 1024$, $|\mathbb{S}| = 128$, Normal threshold = 0.9, Depth threshold = 0.1.
* Biased, ** Unbiased

pipeline but incorporates checkerboard rendering optimizations. This optimization cuts the computational time for generating initial candidates by approximately half. In this report, we present only the total illumination computation time, which roughly corresponds to the Original pipeline's time reduced by half the time it takes to generate the initial candidates.

The Decoupled pipeline typically outperforms the Original pipeline in terms of speed. This efficiency stems from the Decoupled pipeline being implemented as a single kernel, except for the light tiles creation process. As a result, the global synchronization that follows temporal reuse does not

impede it, leading to a more streamlined execution.

The unbiased version exhibits the poorest performance among all pipelines. This pipeline necessitates more noise reduction, requiring casting additional shadow rays. Compared to its biased counterpart, this pipeline is approximately 1.2 to 2 times slower.

Table 5.4 compares different configurations of light tiles for the biased Original pipeline.

The initial set of measurements explores different pixel tile sizes. Pixel Tile Size is the square tile of pixels comprising a group accessing the same light tile. We fixed the remaining two parameters to measure this parameter at $|S_i| = 1024$, $|\mathbb{S}| = 128$. At first glance, the performance in generating initial candidates appears to increase with the size of the pixel tile. This is attributed to a larger number of adjacent pixels accessing the same memory section, thereby reducing the number of cache misses. For a resolution of 1920×1080, we suggest a tile size of 8×8, as with larger tile resolutions, the correlation of light samples between neighboring pixels may become more pronounced, leading to more noticeable tile borders in the final image. However, a greater tile size may be appropriate for larger image resolutions.

The size of the light tile determines the number of light samples stored in each tile. As observed, lower values of this parameter lead to a decrease in the computation time for generating initial candidates. This is due to the high possibility of light samples from each light tile being cached for neighboring pixels during memory access. The computation time for generating light tiles marginally increases with the size of the light tile and is also influenced by the total number of emissive lights in the scene. The larger the specified size of the light tile, the lower the correlation will be between the chosen samples of neighboring pixels. This parameter can significantly impact the final image quality, and for lower values, individual Pixel Light Tiles may become discernible in the image, even after performing temporal and spatial resampling.

As suggested by its name, the number of light tiles determines the total quantity of light tiles produced. This parameter does not substantially influence the computation time for the generation of initial candidates but primarily impacts the creation time of the light tiles. By augmenting the number of light tiles, we can increase the likelihood of individual pixel light tiles selecting diverse sets of light samples, thereby introducing a greater variety of desirable samples into the image.

The final Table 5.5 delves into the performance of the Initial Candidates pass for three different resolutions. The computation of Initial Candidates is directly tied to the image's resolution. Achieving real-time frame rates is feasible for a 1920×1080 resolution, even on a laptop GPU. However, maintaining these frame rates becomes challenging as resolution escalates.

One potential approach to mitigate this issue is increasing the pixel tile size with the rise in image resolution. However, further experimentation would be necessary to fully understand this method's implications. Alternatively, DLSS (Deep Learning Super Sampling) upscaling technology might offer a practical

	Cornell Box	Arcade	Bistro Interior
Varying Pixel Tile Size			
1×1	1.63/10.35	4.34/27.95	4.33/29.85
2×2	0.94/3.97	2.23/11.05	2.26/10.38
4×4	0.61/2.36	1.54/6.95	1.44/6.71
8×8	0.47/1.53	1.05/4.47	1.13/4.49
16×16	0.37/1.43	0.92/4.18	0.90/4.08
$\|S_i\| = 1024$, $\|\mathbb{S}\| = 128$ - Initial Candidates			
Varying Size of Light Tiles $\|S_i\|$			
$\|S_i\| = 256$	0.36 (0.01)	0.91 (0.01)	0.86 (0.01)
$\|S_i\| = 1024$	0.47 (0.01)	1.05 (0.01)	1.13 (0.02)
$\|S_i\| = 4096$	0.71 (0.01)	1.69 (0.01)	1.55 (0.03)
$\|\mathbb{S}\| = 128$ - Initial Candidates (Presampling), on RTX 4070 Ti			
Varying Number of Light Tiles $\|\mathbb{S}\|$			
$\|\mathbb{S}\| = 128$	0.47 (0.01)	1.05 (0.01)	1.13 (0.02)
$\|\mathbb{S}\| = 512$	0.48 (0.01)	1.06 (0.02)	1.12 (0.03)
$\|\mathbb{S}\| = 1024$	0.51 (0.01)	1.07 (0.02)	1.05 (0.04)
$\|S_i\| = 1024$ - Initial Candidates (Presampling), on RTX 4070 Ti			

	Bistro Exterior	Zero-Day	Emerald Square
Varying Pixel Tile Size			
1×1	3.63/18.38	4.40/15.76	4.03/24.10
2×2	2.92/17.69	2.56/14.94	3.59/26.32
4×4	1.88/14.94	1.55/12.28	2.80/23.42
8×8	1.46/11.63	1.10/9.12	2.54/18.96
16×16	1.26/8.57	0.95/6.17	2.14/15.52
$\|S_i\| = 1024$, $\|\mathbb{S}\| = 128$ - Initial Candidates			
Varying Size of Light Tiles $\|S_i\|$			
$\|S_i\| = 256$	1.15 (0.01)	0.89 (0.01)	2.01 (0.02)
$\|S_i\| = 1024$	1.46 (0.02)	1.10 (0.01)	2.54 (0.03)
$\|S_i\| = 4096$	2.02 (0.03)	1.68 (0.02)	2.97 (0.04)
$\|\mathbb{S}\| = 128$ - Initial Candidates (Presampling), on RTX 4070 Ti			
Varying Number of Light Tiles $\|\mathbb{S}\|$			
$\|\mathbb{S}\| = 128$	1.46 (0.02)	1.10 (0.01)	2.54 (0.03)
$\|\mathbb{S}\| = 512$	1.49 (0.03)	1.09 (0.03)	2.44 (0.05)
$\|\mathbb{S}\| = 1024$	1.48 (0.05)	1.10 (0.05)	2.57 (0.07)
$\|S_i\| = 1024$ - Initial Candidates (Presampling), on RTX 4070 Ti			

Table 5.4 : Performance results with various light tiles settings of the biased Original pipeline 4.2. Times in milliseconds on RTX 4070 Ti / Quadro RTX 3000 Mobile. ReSTIR setup: Resolution 1920×1080, Number of initial candidates = 32, Temporal history limit = 20, Number of spatial samples = 1, Number of spatial iterations = 1, Spatial neighborhood radius = 30px, Normal threshold = 0.9, Depth threshold = 0.1.

solution, as mentioned in the previous chapter 4.11. This approach entails rendering the image at a 1920×1080 resolution, applying denoising, and then leveraging DLSS for upscaling to 3840×2160. This yields satisfactory results even at higher resolutions without significantly compromising visual quality or performance.

Another alternative may be to employ a reservoir grid resolution lower than

	Cornell Box	Arcade	Bistro Interior
Resolution Scaling			
1280×720	0.22/1.13	0.49/1.89	0.51/2.04
1920×1080	0.47/1.53	1.05/4.47	1.13/4.49
3840×2160	1.78/7.62	3.72/19.21	3.92/18.33

	Bistro Exterior	Zero-Day	Emerald Square
Resolution Scaling			
1280×720	0.73/5.33	0.56/3.93	1.16/7.29
1920×1080	1.46/11.63	1.10/9.12	2.54/18.96
3840×2160	5.84/44.12	4.18/37.55	8.65/59.88

Table 5.5 : Performance results of Initial Candidates pass of the biased Original pipeline 4.2 for various resolutions. Times in milliseconds on RTX 4070 Ti (and Quadro RTX 3000 Mobile). ReSTIR setup: Number of initial candidates = 32, Temporal history limit = 20, Number of spatial samples = 1, Number of spatial iterations = 1, Spatial neighborhood radius = 30px, Pixel tile size = 8×8, $|S_i| = 1024$, $|\mathbb{S}| = 128$, Normal threshold = 0.9, Depth threshold = 0.1.

the image resolution. This can be implemented by simply halving the grid in both dimensions or utilizing the aforementioned checkerboard rendering technique. For instance, instead of associating each reservoir with a pixel, multiple pixels could share a single reservoir. This approach could reduce the computational cost while maintaining an acceptable render quality. However, these techniques would require careful implementation and testing to ensure they do not introduce artifacts or negatively affect the quality of the final image.

5.3 ReSTIR GI Results

Since the practical part of this project encompasses a basic implementation of the ReSTIR GI algorithm, we have chosen to separate its results from those of the ReSTIR DI algorithm. Given that our implementation of the ReSTIR GI algorithm is in its experimental stages, we opted to test it on a single scene - the Veach Door. A comparison of the efficiency of the ReSTIR GI algorithm can be observed in Figure 5.7.

From the results, we can observe that although this scene presents a significant challenge for path tracing, given that the direct illumination is concentrated in a small region, the ReSTIR GI proves to be highly efficient due to its spatiotemporal sample reuse capabilities.

The total computation time for indirect illumination using the ReSTIR GI for this scene is 5.06 ms on the RTX 4070 Ti for the biased variant and 6.25 ms for the unbiased variant. On the Quadro RTX 3000 Mobile, these times increase to 34.11 ms for the biased variant and 47.06 ms for the unbiased variant.

This method shares similar limitations with ReSTIR DI. A prevalent issue is the reliance on screen-space buffers for reservoir storage, which may not

be sufficient during fast camera movements, leading to possible artifacts or inaccuracies. Another substantial challenge is the longer computation time compared to the ReSTIR DI algorithm, which might be impractical for real-time applications. Current games allocate minimal frame time for calculating indirect lighting, often leveraging techniques such as light mapping or light probes that require considerably less computational time compared to this method. Therefore, despite its promising results, the practical adoption of the ReSTIR GI approach in real-time game settings would require addressing these constraints.

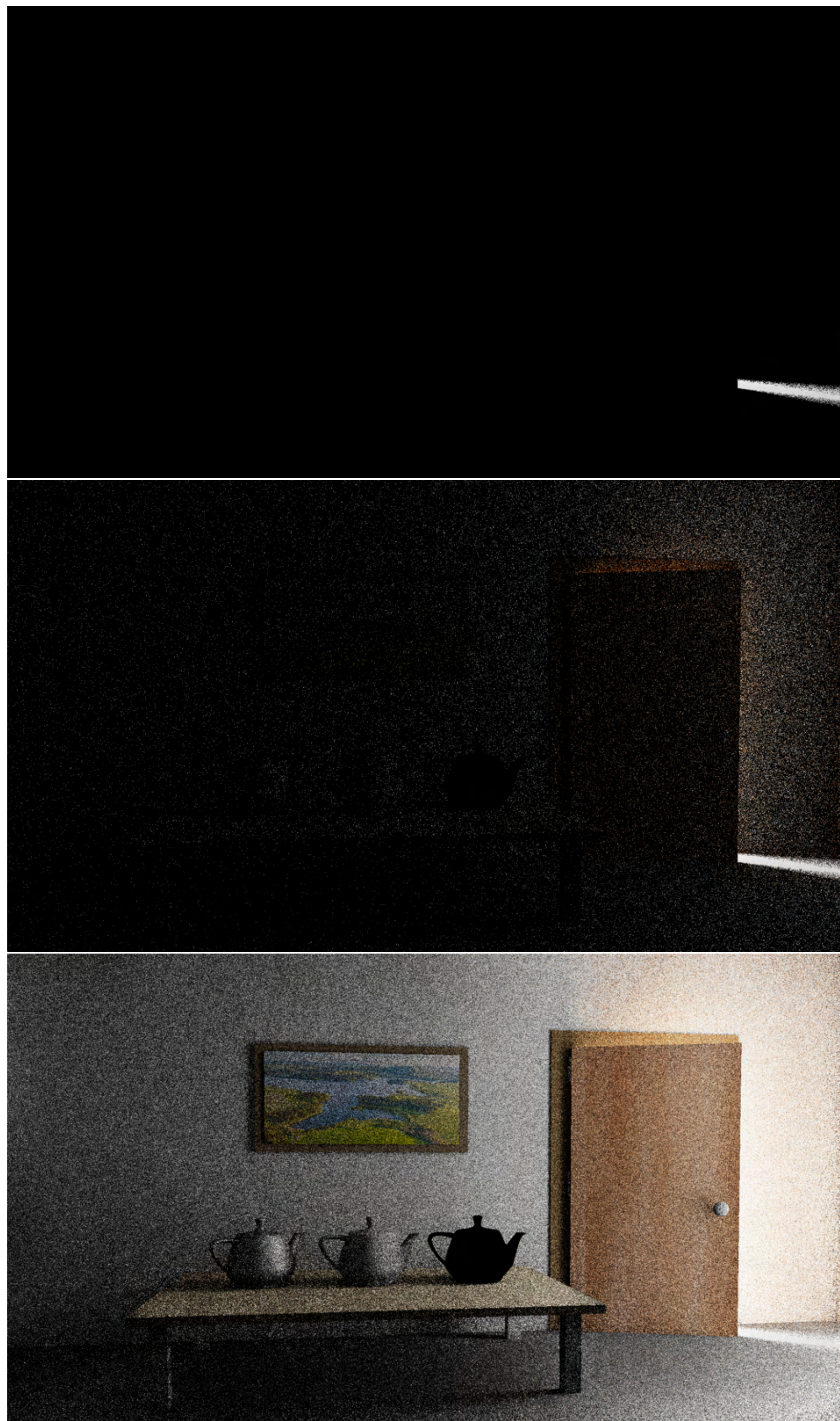


Figure 5.7 : Equal-time comparison of the efficiency of the ReSTIR GI algorithm. (Top) Direct illumination only. (Middle) Output of a Path Tracer with two bounces and NEE. (Bottom) Output generated by the unbiased ReSTIR DI+GI.

Chapter 6

Discussion and Future Work

ReSTIR is proving to be a powerful tool for solving the many-lights problem. Its clever utilization of spatio-temporal resampling enables the selection of high-quality light samples, even when computational resources are limited. Thanks to this feature, ReSTIR can be used for raytracing both in games and in other applications that run in real-time in the near future. By employing the ReSTIR algorithm to sample emissive geometry, the scene creation process for creators and developers can be significantly simplified. There is no longer a need to approximate these lights with analytical lights that often require manual placement.

In future research, it would be valuable to explore using ReSTIR with more complex materials, such as those featuring detailed normal maps or complex geometry consisting of dense triangular meshes (such as vegetation or scanned assets). However, one potential challenge in using ReSTIR with such materials is the granularity of the reservoir grid, which may not be adequate to handle just one reservoir per pixel. Therefore, further investigation is required to determine how to optimize the use of ReSTIR in rendering these complex materials. This could potentially involve implementing a more sophisticated data structure or refining the existing reservoir grid to better accommodate the needs of rendering in such scenarios.

One of the disadvantages, which is also a benefit, is that ReSTIR operates in a screen-space. The advantage of this approach is the simplicity of implementation and the constant amount of memory independent of the scene. The disadvantage is that ReSTIR cannot find good samples quickly during fast animations or camera movements (common in computer games) because there are simply no reservoirs outside the screen. Boksansky et al.[BJW21] have extended the use of the ReSTIR algorithm to world-space. Their sampling algorithm uses a uniform grid structure storing reservoirs in world-space. The sampling of lights is divided into two steps. In the first step, individual cells are assigned samples that can truthfully contribute to that region. Resampling is performed on the pool of these samples, and samples with zero contribution for a given cell are culled. In the second step, the resampling is performed according to the BRDF contribution at the shaded point. This method proves to be particularly suitable for use with secondary rays. Screen-space ReSTIR is preferable for primary rays as it uses a smaller

reservoir granularity. A combination of both methods may be a suitable solution for many applications.

6.1 Other ReSTIR applications

The utility of the ReSTIR algorithm extends beyond addressing many-light problems and direct illumination, as demonstrated throughout our discussions. It has potential applications in a variety of lighting situations, including, but not limited to, indirect illumination, as shown in our experimental implementation of ReSTIR GI. Its robustness in complex lighting scenarios makes it an effective technique to explore in advancing real-time rendering methodologies.

ReSTIR can also be utilized for offline rendering purposes. When rendering an animated sequence is parallelized across multiple computers, temporal resampling may not be available. In such scenarios, relying on spatial samples and increasing the number of iterations is recommended to compensate for the absence of temporal samples and accelerate the overall convergence. By adapting the approach this way, the rendering process can be optimized for offline use, ensuring efficient and high-quality results.

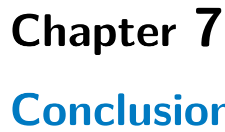
One recent paper that addresses ReSTIR-based samplers is by Daqi Lin et al. [LKB+22]. In this paper, the authors introduce generalized resampled importance sampling (GRIS), which extends the RIS theory and enhances its underlying principles. The authors successfully present a path-traced resampler (ReSTIR PT) that operates interactively on complex scenes, capturing many-bounce diffuse and specular lighting with just one path per pixel.

ReSTIR applies to traditional surface-based rendering and can also be effectively employed in volumetric rendering. This was addressed in a recent paper by Daqi Lin et al. [LWY21]. In this work, the authors applied ReSTIR to volumetric rendering. They demonstrated the ability to achieve low-noise, interactive volumetric path tracing while maintaining efficient performance even on high-resolution volumes. By utilizing ReSTIR for volumetric rendering, this work contributes to the field and opens up new possibilities for real-time rendering applications.

ReSTIR can also be employed for cluster rendering, but dealing with global synchronization and shared memory accesses during spatial reuse can pose a challenge. True spatiotemporal resampling may be suitable to address this issue, as used in the decoupled pipeline approach. This eliminates the need for global synchronization during computation, considerably simplifying implementation.

An intriguing potential application of ReSTIR could be for ray-tracing in virtual reality environments. One approach could involve generating light samples for the left-eye image and applying spatiotemporal resampling. The resulting reservoirs can then be mapped to the corresponding pixels of the right-eye image, potentially mitigating an issue known as specular flickering, which can distort depth perception. However, this method may encounter problems when one eye has a significantly different viewpoint than the other.

Both eyes may require independent light sample generation and processing in such cases.



Chapter 7

Conclusion

This thesis has addressed the topic of efficiently sampling complex illumination for real-time rendering applications. We have provided a comprehensive overview of various algorithms that have been developed to tackle this issue, with a specific focus on the ReSTIR algorithm, which offers a highly promising approach to this problem.

We have presented our implementation of ReSTIR in the Falcor rendering framework, including optimizations inspired by previous work, and demonstrated its superior performance and quality compared to existing techniques. Our results demonstrate that ReSTIR is a highly effective approach for achieving high-quality real-time rendering while maintaining interactive performance.

Moreover, we have explored the potential applications of ReSTIR in various contexts provided by recent papers, including volumetric rendering, offline rendering, cluster rendering, and global illumination. Our analysis has highlighted the strengths and limitations of ReSTIR, and we have identified potential areas for future research to enhance its effectiveness further and extend its capabilities.

Bibliography

[AMW21] Peter Shirley Adam Marrs and Ingo Wald (eds.), *Ray tracing gems ii*, Apress, 2021, `http://raytracinggems.com/rtg2`.

[ANA+20] Pontus Andersson, Jim Nilsson, Tomas Akenine-Möller, Magnus Oskarsson, Kalle Åström, and Mark D. Fairchild, *Flip: A difference evaluator for alternating images*, Proceedings of the ACM on Computer Graphics and Interactive Techniques **3** (2020), no. 2, 15:1–15:23.

[BEM11] Pablo Bauszat, Martin Eisemann, and Marcus Magnor, *Guided image filtering for interactive high-quality global illumination*, 1361–1368.

[Bit16] Benedikt Bitterli, *Rendering resources*, 2016, https://benedikt-bitterli.me/resources/.

[BJW21] Jakub Boksansky, Paula Jukarainen, and Chris Wyman, *Rendering many lights with grid-based reservoirs*, pp. 351–365, Apress, Berkeley, CA, 2021.

[BWP+20] Benedikt Bitterli, Chris Wyman, Matt Pharr, Peter Shirley, Aaron Lefohn, and Wojciech Jarosz, *Spatiotemporal reservoir resampling for real-time ray tracing with dynamic direct lighting*, ACM Transactions on Graphics (Proceedings of SIGGRAPH) **39** (2020), no. 4.

[CEK18] Alejandro Conty Estevez and Christopher Kulla, *Importance sampling of many lights with adaptive tree splitting*, Proc. ACM Comput. Graph. Interact. Tech. **1** (2018), no. 2.

[CKS+17] Chakravarty R. Alla Chaitanya, Anton S. Kaplanyan, Christoph Schied, Marco Salvi, Aaron Lefohn, Derek Nowrouzezahrai, and Timo Aila, *Interactive reconstruction of monte carlo image sequences using a recurrent denoising autoencoder*, ACM Trans. Graph. **36** (2017), no. 4.

[Kaj86] James T. Kajiya, *The rendering equation*, SIGGRAPH Comput. Graph. **20** (1986), no. 4, 143–150.

[KCK+22] Simon Kallweit, Petrik Clarberg, Craig Kolb, Tom'aš Davidovič, Kai-Hwa Yao, Theresa Foley, Yong He, Lifan Wu, Lucy Chen, Tomas Akenine-Möller, Chris Wyman, Cyril Crassin, and Nir Benty, *The Falcor rendering framework*, 8 2022, `https://github.com/NVIDIAGameWorks/Falcor`.

[LKB+22] Daqi Lin*, Markus Kettunen*, Benedikt Bitterli, Jacopo Pantaleoni, Cem Yuksel, and Chris Wyman, *Generalized resampled importance sampling: Foundations of restir*, ACM Transactions on Graphics (Proceedings of SIGGRAPH 2022) **41** (2022), no. 4, 75:1–75:23, (*Joint First Authors).

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

[LWY21] Daqi Lin, Chris Wyman, and Cem Yuksel, *Fast volume rendering with spatiotemporal reservoir resampling*, ACM Trans. Graph. **40** (2021), no. 6.

[LY20] Daqi Lin and Cem Yuksel, *Real-time stochastic lightcuts*, Proc. ACM Comput. Graph. Interact. Tech. (Proceedings of I3D 2020) **3** (2020), no. 1.

[MPC22] P. Moreau, M. Pharr, and P. Clarberg, *Dynamic many-light sampling for real-time ray tracing*, Proceedings of the Conference on High-Performance Graphics (Goslar, DEU), HPG '19, Eurographics Association, 2022, p. 21–26.

[NHB17] Kate Anderson Nicholas Hull and Nir Benty, *Nvidia emerald square, open research content archive (orca)*, July 2017, `http://developer.nvidia.com/orca/nvidia-emerald-square`.

[nrd22] *Nvidia real-time denoisers (nrd)*, Nov 2022.

[OLK+21] Y. Ouyang, S. Liu, M. Kettunen, M. Pharr, and Jacopo Pantaleoni, *Restir gi: Path resampling for real-time path tracing*, Computer Graphics Forum **40** (2021), 17–29.

[SSK+17] Christoph Schied, Marco Salvi, Anton Kaplanyan, Chris Wyman, Anjul Patney, Chakravarty Chaitanya, John Burgess, Shiqiu Liu, Carsten Dachsbacher, and Aaron Lefohn, *Spatiotemporal variance-guided filtering: real-time reconstruction for path-traced global illumination*, 1–12.

[TCE05] Justin Talbot, David Cline, and Parris Egbert, *Importance Resampling for Global Illumination*.

[Wal77] Alastair J. Walker, *An efficient method for generating discrete random variables with general distributions*, ACM Trans. Math. Softw. **3** (1977), no. 3, 253–256.

[WFA+05] Bruce Walter, Sebastian Fernandez, Adam Arbree, Kavita Bala, Michael Donikian, and Donald P. Greenberg, *Lightcuts: A scalable approach to illumination*, ACM Trans. Graph. **24** (2005), no. 3, 1098–1107.

[Win19] Mike Winkelmann, *Zero-day, open research content archive (orca)*, November 2019, https://developer.nvidia.com/orca/beeple-zero-day.

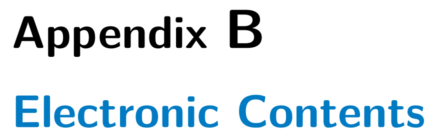
[WP21] Chris Wyman and Alexey Panteleev, *Rearchitecting Spatiotemporal Resampling for Production.*

[Yuk20] Cem Yuksel, *Stochastic lightcuts for sampling many lights*, IEEE Transactions on Visualization and Computer Graphics (2020).

Appendix A

Dependencies

NVIDIA Falcor: Real-Time Rendering Framework [KCK+22]



Appendix B

Electronic Contents

```
/
├── aplication
│   ├── bin
│   └── data
│       ├── render_graphs
│       └── scenes
├── source_code
├── documentation
│   ├── tex
│   │   └── images
│   └── thesis.pdf
└── results
    ├── images
    └── videos
```