

Incremental Performance and Quality Analysis of Hybrid Ray Tracing

Incremental Analysis in Vulkan Real-Time Rendering

De Meyer Luca

Graduation work 2025-2026



Contents

Abstract & Keywords	4
Preface	5
List of Figures	6
1 Introduction	7
2 Literature study / Theoretical framework	9
2.1 Rasterization Fundamentals	9
2.1.1 Rasterization Baseline for Hybrid Ray Tracing	9
2.2 Rendering Pipeline Variants	10
2.2.1 Forward Rendering	10
2.2.2 Deferred Rendering	11
2.3 GPU Parallelism and Pipeline Cost Factors	13
2.4 Ray Tracing & Hybrid Strategies	13
2.4.1 Ray Tracing Fundamentals	13
2.4.2 Vulkan raytracing VS DXR	14
2.4.3 Spatiotemporal Reconstruction for Low-Sample Ray Tracing	17
2.4.4 Production Denoisers: NRD	17
2.4.5 Hybrid Rendering Strategies	18
2.4.6 Perceptual Quality Metrics	19
2.4.7 Cost Characteristics of Ray-Traced Shadows and Re- flections	19
2.4.8 Limitations of Traditional Image Quality Metrics	20
2.5 Motivation for Hybrid Ray Tracing	21

3	Research	22
3.1	Research Questions & Hypotheses	22
3.2	Methodology	23
3.2.1	Test Environment and Hardware	23
3.2.2	Test Scenes	23
3.2.3	Independent and Dependent Variables	24
3.2.4	Instrumentation and Profiling	25
3.2.5	Pipeline Configurations	25
4	Implementation	26
4.1	Rasterization Baseline and Deferred Rendering	26
4.1.1	Fragment Shading in the Rasterization Pass	26
4.1.2	Deferred Rendering Architecture	27
4.1.3	The Lighting Pass and Baseline Features	27
4.1.4	Hybrid Ray Tracing Integration	28
4.2	G-buffer-Guided vs. Unguided Ray Generation	28
4.2.1	Ray-Traced Shadows	28
4.2.2	Ray-Traced Ambient Occlusion (RTAO)	29
4.2.3	Ray-Traced Reflections (RTR)	29
4.2.4	Dual-Path SVGF Reconstruction	30
5	Results	31
5.1	Rasterization Baseline Performance	31
5.2	Ray-Traced Shadow Performance	32
5.3	Ray-Traced Reflection Performance	34
5.4	Ray-Traced Ambient Occlusion Performance	36
5.5	Hybrid Configuration Comparison	37
5.5.1	Architectural Efficiency: Pre-Lighting vs. Post-Lighting (Sponza)	39
5.6	G-buffer-Guided vs. Unguided Ray Dispatching	40
5.7	Reflection Stress Test: Metal-Spheres Scene	43
5.7.1	Performance Overhead and Discretization Artifacts	43
5.7.2	The Failure of Post-Lighting on Specular Surfaces	44
5.8	Visual Quality and Perceptual Metrics	44
5.8.1	The Convergence of Raw Ray Tracing	45
5.8.2	The Denoising Plateau (SVGF)	46
5.8.3	Note on Metric Interpretation	47
5.9	Chapter Summary	48

6	Discussion and Limitations	49
6.1	Signal-Dependent Viability of Post-Lighting Integration	49
6.2	Practical Quality-per-Cost Tradeoffs	50
6.3	Feature Coverage versus Sample Density	50
6.4	Methodological Limitations	51
7	Conclusion	52
7.1	Summary of Key Findings	52
7.2	Final Remarks	54
8	Future Work	55
8.1	Albedo Remodulation and Energy Conservation	55
8.2	Global Illumination and Diffuse Interreflection	56
8.3	Machine Learning-Based Reconstruction	56
8.4	Dynamic Quality Scaling	57
8.5	Cross-Vendor Hardware Analysis	57
8.6	Human Perceptual Validation	58
9	Critical Reflection	59
9.1	Technical Challenges in Vulkan Integration	59
9.2	The Reality of Real-Time Ray Tracing	60
9.3	Methodological Growth	61
9.4	Final Conclusion	61

Abstract & Keywords

Real-time rendering has converged on hybrid pipelines that combine rasterization with selectively applied ray tracing, yet developers still lack quantitative guidance on how to prioritize ray-traced features under fixed frame-time budgets. While effects such as shadows, reflections, and global illumination can substantially improve visual fidelity, their performance costs and perceptual benefits vary widely across scenes, sampling strategies, and denoising configurations, complicating engine-level decision making.

This study evaluates the incremental performance cost and visual impact of ray-traced shadows and reflections in a Vulkan-based hybrid renderer. We analyze G-buffer-guided ray generation, SVGF denoising, and screen-space fallbacks across two representative scenes at 1080p resolution, targeting real-time frame budgets on contemporary mid-range GPUs. Visual quality is assessed against a path-traced reference using perceptual similarity metrics (LPIPS, SSIM), while GPU profiling is used to measure feature-level costs. By combining these measurements, we derive a "quality-per-millisecond" metric that enables direct comparison of hybrid rendering configurations. Our results demonstrate that G-buffer-guided ray tracing at extremely low sample counts (1 SPP), when paired with spatiotemporal denoising, offers the most favorable balance between perceptual quality and performance, rendering higher sampling rates redundant for real-time applications.

Keywords: Real-time rendering, Hybrid Ray Tracing, Vulkan, Shadows, Reflections, Performance Analysis, Perceptual Quality, LPIPS.

Preface

This graduation project investigates the incremental integration of ray tracing into real-time rendering pipelines, with a specific focus on balancing visual quality and performance under fixed frame-time constraints. Rather than pursuing fully path-traced rendering, the work explores hybrid strategies that selectively combine rasterization with ray-traced shadows and reflections, reflecting the practical design choices made in modern game engines. The project was motivated by a desire to better understand how emerging hardware-accelerated ray tracing features can be adopted pragmatically in real-time applications. While ray tracing offers clear improvements in lighting accuracy and visual realism, its cost varies significantly depending on scene characteristics, sampling strategies, and reconstruction techniques. This work aims to provide quantitative and qualitative insights that can inform engine-level decisions when integrating ray tracing incrementally.

From a personal perspective, the project served as an opportunity to deepen technical knowledge of Vulkan-based rendering systems, GPU profiling, and spatiotemporal reconstruction techniques. In particular, designing and instrumenting a hybrid renderer from first principles provided valuable insight into the performance characteristics of both rasterization and ray tracing workloads, as well as the trade-offs involved in real-world engine development.

The scope and structure of this thesis were shaped by both technical constraints and time limitations. As a result, the focus is placed on shadows and reflections as representative ray-traced effects, while more complex phenomena such as global illumination are left for future work. Despite these limitations, the project aims to present results that are both technically sound and practically relevant to developers working with hybrid real-time rendering pipelines.

List of Figures

*List of Figures

2.1	Graphics Rendering Pipeline	10
2.2	<i>Conceptual visualization of a G-Buffer layout created during the geometry pass.</i>	12
2.3	Ray tracing pipeline in Vulkan with acceleration structure traversal	16
3.1	The Sponza test scene, utilized for evaluating complex shadow occlusion and indirect lighting.	24
5.1	Performance distribution and timing clustering for the rasterization baseline configuration.	32
5.2	Isolated performance scaling and timing distribution of ray-traced shadow passes.	33
5.3	Granular pass-by-pass GPU timing breakdown for the full pipeline configuration at 8 SPP.	34
5.4	Performance scaling behavior of ray-traced reflections across varying sample counts.	35
5.5	Comprehensive performance-versus-quality trade-off across full pipeline configurations.	38
5.6	Sample count (SPP) scaling convergence chart demonstrating diminishing perceptual returns.	38
9.1	Visual comparison demonstrating the necessity of spatiotemporal reconstruction at low sample budgets.	60

Chapter 1

Introduction

Hardware-accelerated ray tracing is now a mature feature of contemporary consumer GPUs, forming a standard component of real-time rendering pipelines across both high-end engines and commercial games. Rather than replacing rasterization, ray tracing is typically deployed selectively within hybrid pipelines, where it augments traditional techniques to improve lighting fidelity, visibility accuracy, and geometric correctness under strict performance constraints.

Although recent hardware generations and advanced denoising techniques have enabled real-time path-traced rendering in controlled scenarios, fully converged path tracing remains impractical for most interactive applications without aggressive spatio-temporal reconstruction. Consequently, modern engines rely on incremental ray tracing adoption, allocating a limited ray budget to specific effects such as shadows and reflections while retaining rasterization for primary visibility and shading.

Determining how to allocate this budget presents a nontrivial optimization problem. The performance cost of individual ray-traced features varies significantly with scene complexity, sampling strategy, and acceleration structure usage, while the resulting perceptual benefit depends heavily on lighting conditions, material properties, and denoising quality. Existing evaluations primarily focus on fully path-traced pipelines or isolated techniques, providing limited guidance for hybrid approaches. DXR is discussed in the theoretical framework for context, but experimental evaluation in this work focuses on Vulkan.

Research Goals and Contributions:

- Quantitative measurement of incremental ray tracing costs for shadows and reflections individually and in combination.
- Perceptual quality analysis of different sampling rates and denoising strategies using LPIPS, SSIM metrics.
- Discussion of DXR as a related API model in the theoretical framework, with Vulkan used for experimental evaluation.
- Quality-per-millisecond ranking of hybrid rendering strategies (G-buffer-guided, screen-space fallbacks).
- Practical configuration recommendations for achieving 60 FPS targets on upper-midrange/enthusiast-class GPU hardware.

Chapter 2

Literature study / Theoretical framework

2.1 Rasterization Fundamentals

Rasterization is the dominant technique in real-time computer graphics, used in almost all 3D interactive applications. The fundamental principle involves converting 3D geometry into a 2D grid of pixels. By using the GPU, which is highly optimised to rasterize millions of triangles, rasterization is fast and efficient, forming the backbone of real-time rendering. Understanding rasterization requires a structured view of how modern GPUs transform scene data into pixels, which is formalized by the graphics rendering pipeline (Marschner and Shirley, 2016).

2.1.1 Rasterization Baseline for Hybrid Ray Tracing

The graphics rendering pipeline describes the process by which a three-dimensional scene is transformed into a two-dimensional image from a virtual camera. In modern real-time engines, rasterization serves as the primary mechanism for determining visible surfaces and for generating screen-space representations of geometry and materials. In hybrid rendering pipelines, these rasterized outputs form the foundation on which ray tracing and spatiotemporal reconstruction techniques operate.

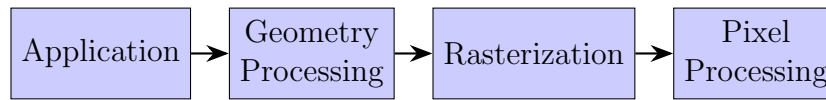


Figure 2.1: Graphics Rendering Pipeline

Rendering commands are assembled on the CPU and submitted to the GPU, where vertex processing transforms geometry into clip space. After clipping and viewport mapping, primitives are rasterized into screen-space fragments. During fragment shading, material properties are evaluated and per-pixel attributes are written either to the framebuffer or to intermediate render targets. Optional programmable stages such as tessellation or geometry shaders may be used for adaptive detail or procedural effects; however, these stages are not central to the hybrid rendering pipeline evaluated in this work and are therefore omitted from further discussion.

For the purposes of this thesis, the primary role of rasterization is not the production of a fully shaded image, but the efficient generation of screen-space buffers encoding geometric and material information required for deferred lighting, ray tracing, and spatiotemporal reconstruction (Akenine-Möller et al., 2018; de Vries, nd).

2.2 Rendering Pipeline Variants

2.2.1 Forward Rendering

In a forward rendering pipeline, scene geometry is rasterized and shaded in a single pass, with lighting computations performed directly during fragment shading. This approach is conceptually simple and aligns closely with the traditional graphics pipeline, making it straightforward to implement for scenes with a limited number of light sources.

However, the cost of forward rendering scales with both fragment count and the number of active lights, as each fragment must evaluate all relevant lighting equations. As a result, forward rendering becomes inefficient in scenes with many dynamic lights. Modern variants such as Forward+ mitigate this limitation through screen-space light culling, though lighting computations remain tightly coupled to fragment shading (Akenine-Möller et al., 2018; de Vries, nd).

2.2.2 Deferred Rendering

Deferred rendering separates geometry processing from lighting evaluation by dividing the pipeline into two distinct passes: a geometry pass and a lighting pass. During the geometry pass, the scene is rasterized once and per-fragment attributes such as surface normals, albedo, material parameters, and depth are written to a set of render targets collectively referred to as the G-buffer. Depth testing is performed during this stage; optionally, a depth pre-pass may be executed prior to G-buffer generation to reduce overdraw and unnecessary fragment shading.

In the lighting pass, illumination is computed in screen space by sampling the G-buffer. Because lighting is evaluated only for visible fragments, the cost of shading scales primarily with screen resolution rather than scene complexity, making deferred rendering well suited for scenes with many dynamic lights (Fernando, 2004).

Deferred rendering introduces trade-offs, including increased memory usage and bandwidth requirements due to multiple render targets. Additionally, handling transparency and hardware multisample anti-aliasing (MSAA) is non-trivial, leading many real-time engines to employ hybrid solutions (Akenine-Möller et al., 2018; de Vries, nd).

G-Buffer as Input for Hybrid Ray Tracing

The G-buffer is a collection of screen-space render targets that store per-fragment geometric and material information required for subsequent lighting evaluation and hybrid ray tracing.

Typical contents include:

- **World-space position vectors:** Used to reconstruct the 3D location of each fragment for lighting and shading calculations.
- **Surface normals:** Required for accurate lighting, reflections, and shading computations.
- **Albedo (diffuse color):** Stores the base color of the surface without lighting applied.
- **Additional material properties (optional):** Such as roughness, metallicity, and specular intensity for physically based rendering workflows.

These buffers collectively allow the lighting pass to operate independently of scene geometry, enabling deferred shading and efficient evaluation of multiple light sources (Akenine-Möller et al., 2018; de Vries, nd).

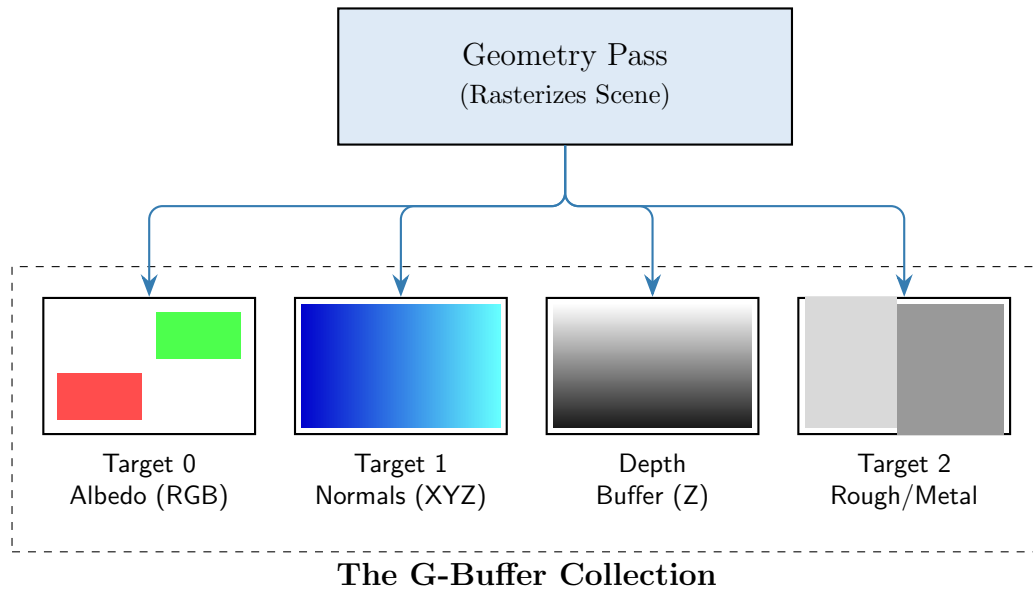


Figure 2.2: *Conceptual visualization of a G-Buffer layout created during the geometry pass.*

World-space positions may be stored directly or reconstructed from depth using inverse camera matrices (Akenine-Möller et al., 2018; de Vries, nd).

Lighting Pass

The lighting pass operates entirely in screen space, reconstructing surface properties from the G-buffer to compute lighting contributions for each visible pixel. Because only the closest visible fragments are stored, lighting calculations are performed solely for pixels that contribute to the final image, improving performance in scenes with high depth complexity.

Multiple light sources can be evaluated without re-rendering geometry, making the deferred lighting pass particularly well suited for dynamic lighting scenarios and for integration with hybrid ray tracing techniques.

2.3 GPU Parallelism and Pipeline Cost Factors

Modern GPUs exploit massive data parallelism by executing threads in lock-step groups, known as warps or wavefronts. While fixed-function stages like rasterization are highly optimized and throughput-oriented, programmable stages such as ray generation and intersection shaders are highly sensitive to divergence. In ray tracing, when secondary rays bounce in different directions (e.g., on rough specular surfaces), threads within the same warp take different execution paths and access disparate memory locations. This incoherent control flow and irregular memory access pattern drastically reduce effective throughput and place an immense bottleneck on memory bandwidth (Akenine-Möller et al., 2018).

To mitigate these architectural bottlenecks under strict real-time frame budgets, modern render engines leverage asynchronous compute. By overlapping ray tracing and compute workloads (e.g., denoising or BVH updates), with traditional graphics passes, developers can partially hide the latency of these bandwidth-heavy operations, provided that shared resources and synchronization requirements allow for it (Akenine-Möller et al., 2018).

2.4 Ray Tracing & Hybrid Strategies

2.4.1 Ray Tracing Fundamentals

Ray tracing is a rendering technique that simulates light transport by tracing rays through a scene and evaluating their interactions with geometry and materials. Rays are typically emitted from the camera to determine visible surfaces, after which secondary rays may be spawned to evaluate effects such as shadowing, reflections, and refractions.

A ray is commonly represented as a parametric half-line

$$P(t) = O + tD, \quad t \geq 0$$

where O denotes the ray origin and D its direction (Shirley, 2024). In practice, rays are traced over finite segments defined by application-specific minimum and maximum distances.

Ray tracing algorithms are typically structured around three conceptual stages: ray generation, ray-scene intersection, and shading. Ray generation defines the origin and direction of rays, intersection tests determine the closest geometry hit along each ray, and shading evaluates material response and lighting at the intersection point. To limit computational cost, recursive ray spawning is bounded by a maximum bounce depth.

Several ray tracing variants are commonly distinguished in the literature. As surveyed by Qu (2023), these methods fundamentally divide into forward and backward raytracing. Because forward tracing (simulating rays from light sources to the camera) is highly inefficient for image synthesis, modern real-time applications rely on backward ray tracing. Early backward methods like Whitted-style ray tracing assumed ideal specular surfaces and traced a limited set of reflection, refraction, and shadow rays. Stochastic ray tracing introduces probabilistic sampling to support effects such as glossy reflections and soft shadows. Path tracing generalizes this approach by stochastically sampling the full light transport equation, enabling physically based global illumination (Pharr et al., 2023). However, solving this equation at the extremely low sample counts required for real-time rendering introduces massive variance in the form of high-frequency stochastic noise (Wester, 2020). To limit computational cost, ray intersection is accelerated using Bounding Volume Hierarchies (BVH). As described by Galvan (2025), these structures partition scene geometry into a tree of axis-aligned bounding boxes (AABBs), allowing the traversal shader to cull vast amounts of geometry by testing against the boxes rather than individual triangles. Modern GPUs contain dedicated hardware units to traverse these Top-Level (TLAS) and Bottom-Level (BLAS) structures efficiently.

In real-time applications, fully converged path tracing is generally impractical under fixed frame-time budgets. Consequently, modern engines employ hybrid ray tracing pipelines in which rasterization determines primary visibility, while ray tracing is selectively applied to specific effects such as shadows and reflections (Haines and Akenine-Möller, 2019).

2.4.2 Vulkan raytracing VS DXR

Both Vulkan Ray Tracing and DirectX Raytracing (DXR) expose hardware-accelerated ray tracing through a common conceptual model consisting of acceleration structures, programmable ray tracing shaders, and shader binding tables. As described in Ray Tracing Gems, both APIs ultimately target the same fixed-function traversal hardware on modern GPUs and differ primarily

in their API design philosophy rather than their underlying execution model.

DXR is implemented as an extension to DirectX 12 and is designed to integrate ray tracing into an existing graphics pipeline (Luna, 2016; NVIDIA Developer Blog, 2018). It provides higher-level abstractions for ray tracing pipelines, shader binding, and acceleration structure management, prioritizing ease of integration and reduced boilerplate for applications already built on DirectX 12 (Marrs et al., 2021). Many resource management and synchronization details are handled implicitly by the API.

In contrast, Vulkan Ray Tracing follows Vulkan’s explicit control philosophy. Acceleration structures, shader binding tables, memory allocation, and synchronization are all exposed directly to the developer. Ray tracing pipelines are constructed in a manner similar to compute pipelines, allowing ray tracing workloads to be scheduled explicitly alongside rasterization and compute passes. This design favors predictability and fine-grained control, making Vulkan Ray Tracing particularly suitable for engine-level implementations and research-oriented systems.

Despite these API-level differences, both Vulkan Ray Tracing and DXR provide access to the same ray tracing shader stages, including ray generation, miss, closest-hit, any-hit, and intersection shaders. Ray Tracing Gems emphasizes that performance differences between the two APIs are generally attributable to application-level design choices rather than fundamental differences in the hardware ray tracing pipeline.

Both Vulkan Ray Tracing and DirectX Raytracing (DXR) expose hardware-accelerated ray traversal through similar conceptual models, including acceleration structures, ray tracing shader stages, and shader binding tables. Prior work (e.g., Ray Tracing Gems) emphasizes that performance characteristics are primarily driven by ray count, coherence, and traversal complexity rather than API choice. Consequently, this study focuses on Vulkan as a representative modern ray tracing API, with results expected to generalize to DXR-based pipelines under comparable workloads (Haines and Akenine-Möller, 2019; Khronos Group, nd; NVIDIA Developer Blog, 2018).

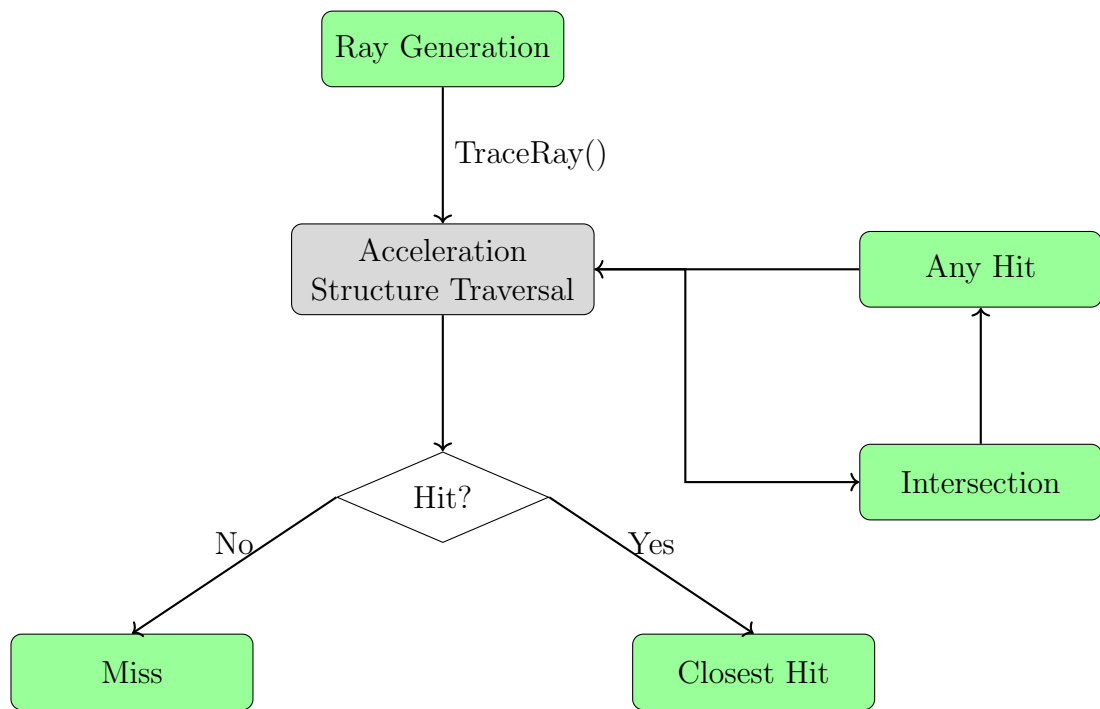


Figure 2.3: Ray tracing pipeline in Vulkan with acceleration structure traversal

2.4.3 Spatiotemporal Reconstruction for Low-Sample Ray Tracing

Real-time ray tracing at interactive frame rates is constrained to extremely low sample counts—often a single path per pixel. At such sampling rates, Monte Carlo noise obscures image structure entirely, making reconstruction filters essential for practical deployment.

Schied et al. introduced Spatiotemporal Variance-Guided Filtering (SVGF), a reconstruction algorithm designed specifically for 1 SPP path-traced input. The technique combines temporal accumulation across frames with a hierarchical à-trous wavelet filter, using per-pixel luminance variance estimates to adaptively control filter strength. By leveraging G-buffer attributes (depth, normals, mesh IDs) as edge-stopping functions, SVGF preserves geometric detail while aggressively filtering stochastic noise.

Key to SVGF’s effectiveness is its variance estimation strategy: temporal accumulation of the first and second moments of luminance provides a statistically meaningful variance estimate that would be impossible from a single spatial sample. This variance then modulates the spatial filter, preventing overblurring in low-variance regions while permitting aggressive filtering where noise dominates.

The authors report 10 ms reconstruction time at 1080p on contemporary hardware, with SSIM improvements of 5 - 47% over prior filtering approaches. However, SVGF exhibits limitations relevant to hybrid pipelines: over-blurred specular reflections when indirect lighting is undersampled, temporal lag under fast motion, and incompatibility with stochastic primary ray effects.

SVGF is used in this study as a reference denoiser due to its algorithmic transparency, ease of integration, and prevalence in academic literature, enabling controlled comparison with newer production denoisers (Schied et al., 2017).

2.4.4 Production Denoisers: NRD

Building on the foundations established by SVGF, NVIDIA Real-Time Denoisers (NRD) represents the current state of production-quality reconstruction for hybrid ray tracing pipelines. NRD is a spatio-temporal, API-agnostic library designed specifically for signals at 0.5–1 rays per pixel, and has been deployed in commercial titles including Watch Dogs: Legion, Dying Light 2, and Cyberpunk 2077 (Singh, 2020).

NRD provides two primary denoiser variants:

- **ReBLUR:** A blur-based denoiser using hit distance to guide spatial filtering, supporting diffuse and specular signals with built-in ambient/specular occlusion denoising.
- **ReLAX:** Designed for RTXDI (ReSTIR-based direct illumination), optimized for high-frequency shadow boundaries and specular detail.

Unlike SVGF’s unified approach, NRD processes diffuse and specular components separately, allowing signal-specific tuning. Both variants require demodulated irradiance input (albedo factored out) and standard G-buffer attributes. Performance at 1440p on RTX 4080 hardware is reported at 2.55 ms for ReBLUR_DIFFUSE_SPECULAR in default mode, representing approximately 50% improvement over equivalent SVGF configurations (NVIDIA-RTX, nd).

NRD’s API-agnostic design supports DX11, DX12, and Vulkan through pre-compiled DXBC, DXIL, and SPIR-V shaders respectively. While NRD represents the current production standard for commercial titles, its complex API requirements and pre-compiled shader integration place it outside the scope of this research project. Therefore, this experiment focuses on using SVGF as its algorithmic baseline to clearly measure the performance impact. This leaves an open question that this thesis addresses directly: given a simpler, fully transparent denoising baseline like SVGF rather than a production black-box like NRD, how much perceptual quality per millisecond of ray tracing overhead can a hybrid pipeline actually recover—and where does that budget best go, shadows or reflections?

2.4.5 Hybrid Rendering Strategies

To manage ray tracing cost under fixed frame-time budgets, modern renderers apply ray tracing selectively and combine it with rasterized buffers and screen-space methods. This thesis evaluates the following strategy families:

- **Full-screen ray tracing:** Dispatches rays for every pixel (low SPP) with aggressive denoising.
- **G-buffer-guided tracing:** Uses rasterized geometry buffers to selectively dispatch rays only where needed (e.g., glossy surfaces).
- **Screen-space first, fallback:** SSR/SSAO when RT reflection and RTAO are turned off.

2.4.6 Perceptual Quality Metrics

Hybrid ray tracing quality is evaluated against a path-traced reference. Because denoising introduces artifacts such as blur, temporal lag, and loss of high-frequency detail, simple pixel-wise error metrics often correlate poorly with perceived quality. We therefore use both a structural metric (SSIM) and a learned perceptual metric (LPIPS) to better capture visually meaningful differences in shadows and specular reflections.

- **LPIPS:** Learned Perceptual Image Patch Similarity. Uses deep neural networks trained on human perceptual judgments.
- **SSIM:** Structural Similarity Index. Measures structural information preservation.

2.4.7 Cost Characteristics of Ray-Traced Shadows and Reflections

The performance cost of ray tracing depends strongly on ray count, ray coherence, and scene traversal complexity. Shadow rays are typically short, single-bounce queries with a clear termination criterion (visibility to a light), making their cost primarily driven by light count, shadow distance, and BVH occlusion complexity. In hybrid pipelines, shadow denoising is often less expensive than reflection denoising due to lower signal variance and fewer high-frequency view-dependent features.

However, generating realistic soft shadows requires distributing multiple rays across an area light to accurately sample the penumbra (Hamza, 2024; Xie et al., 2007). At low sample counts, this introduces severe Monte Carlo noise (Wester, 2020). Furthermore, full-resolution ray-traced shadows can be prohibitively expensive; Anagnostou demonstrates that naively ray tracing shadows for a full frame can cost upwards of 92 ms, whereas a hybrid approach using edge-focused tracing and reconstruction can reduce this to 15 ms (Anagnostou, 2021). To mitigate this immense disparity without tracing full screens, early hybrid approaches like those proposed by Lauterbach et al. (2009) introduced selective ray tracing, utilizing rasterization for the umbra and fully lit areas, and only dispatching rays for potentially incorrect pixels at shadow boundaries.

Ray-traced reflections are generally more expensive because reflection rays are often less coherent (especially on rough surfaces), may require longer traversals, and produce high-variance specular signals that demand stronger

spatiotemporal reconstruction. Reflection cost is therefore influenced by material roughness distribution, view-dependent ray directions, and the percentage of pixels that require fallback beyond screen space. These differences motivate evaluating shadows and reflections separately and in combination when budgeting ray tracing time (Akenine-Möller et al., 2018; Haines and Akenine-Möller, 2019).

2.4.8 Limitations of Traditional Image Quality Metrics

Traditional image quality metrics such as MSE and PSNR operate on per-pixel differences and assume that small spatial shifts or changes in texture detail are perceptually minor. In hybrid ray tracing pipelines, however, denoisers can introduce structured artifacts such as temporal ghosting, specular smearing, and loss of high-frequency detail that may be visually objectionable despite low pixel-wise error. Conversely, stochastic noise may produce large pixel-wise error while remaining less perceptually disruptive once temporally filtered.

Because the goal of this thesis is to quantify perceptual improvement per millisecond of ray tracing overhead, evaluation requires metrics that better correlate with human judgments of image similarity. This motivates the use of SSIM for structural preservation and LPIPS for perceptual differences that arise from denoising and view-dependent specular effects (Zhang et al., 2018; Schied et al., 2017). This methodology aligns with recent industry evaluations, such as the comparative analysis of RTAO and GTAO (Waldner, 2021), which successfully relied on SSIM to quantify the visual noise introduced by hardware-accelerated ray tracing under strict millisecond budgets.

2.5 Motivation for Hybrid Ray Tracing

Hybrid rendering pipelines arise from a practical mismatch between the visual benefits of ray-traced effects and the computational constraints of real-time frame budgets. Rasterization remains highly efficient for primary visibility and material evaluation, while ray tracing provides more accurate visibility and light transport for effects that are difficult to approximate robustly in screen space, such as hard-to-predict reflections and complex shadowing.

Consequently, modern engines rely on incremental ray tracing adoption, allocating a limited ray budget to specific effects such as shadows and reflections. This hybrid approach has been successfully deployed in AAA titles such as *Call of Duty: Modern Warfare*, where ray-traced shadows were integrated alongside traditional rasterization to balance fidelity and performance (Olejník and Kozłowski, 2020). Similarly, academic studies have demonstrated that deferred hybrid pipelines can achieve frame rates 45–118% higher than naive full-scene ray tracing (Ki, 2023).

However, the cost of ray tracing scales with ray count, traversal complexity, and reconstruction overhead, and is further amplified by incoherent secondary rays and bandwidth-heavy denoising. As a result, most real-time engines treat ray tracing as a limited budget that must be allocated selectively across features and scenes. This motivates evaluating incremental adoption strategies that combine rasterized buffers (e.g., the G-buffer), selective ray dispatch, and spatiotemporal denoising to achieve the largest perceptual improvement per millisecond of overhead.

The remainder of this thesis investigates how ray-traced shadows and reflections contribute to perceptual quality under fixed budgets, using Vulkan Ray Tracing as the experimental platform. DXR is discussed as a theoretically equivalent API model to contextualize the results.

Chapter 3

Research

3.1 Research Questions & Hypotheses

Research Questions: To evaluate the viability of hybrid rendering pipelines under strict real-time constraints, this research addresses the following primary questions:

1. What is the performance overhead of integrating ray-traced effects (Shadows, Reflections, and Ambient Occlusion) into a deferred pipeline, and how is this cost distributed across individual GPU passes?
2. How do computational costs and perceptual quality metrics (such as LPIPS) scale across different ray budgets (1, 8, 16, and 32 Samples Per Pixel)?
3. How does integrating spatiotemporal denoising (SVGF) post-lighting compare to a traditional pre-lighting integration in terms of overall frame time and architectural efficiency?

Hypotheses:

Based on the theoretical framework and industry benchmarks, the following hypotheses are proposed:

- **H1 (Performance Scaling & Diminishing Returns):** The computational cost of hybrid ray tracing will scale sub-linearly with the

sample count due to the fixed overhead of spatiotemporal denoising. Consequently, a 1 SPP ray budget combined with SVGF will yield the optimal perceptual quality-per-millisecond ratio, with higher sample counts (8, 16, and 32 SPP) exhibiting steep diminishing returns.

- **H2 (Architectural Efficiency):** A unified Post-Lighting SVGF architecture will significantly reduce total GPU frame time compared to a Pre-Lighting architecture by eliminating redundant secondary G-buffer rendering passes, all while maintaining equivalent perceptual visual fidelity.

3.2 Methodology

3.2.1 Test Environment and Hardware

To provide accurate and reproducible profiling data, all performance metrics and frame timings were captured on a consistent hardware test bench. The evaluation was conducted on a system equipped with an AMD Ryzen 9 9900X CPU, 64GB of RAM, and an AMD RX 9070 XT GPU. This configuration represents a modern high-performance target, ensuring the millisecond timings reflect realistic consumer hardware constraints. All tests were rendered at a fixed internal resolution of 1920×1080 .

3.2.2 Test Scenes

The evaluation utilizes two distinct scenes to stress-test different ray tracing characteristics:

- **Indoor Scene (Sponza):** Selected to rigorously test complex shadow occlusion, contact hardening, and the high-frequency variance inherent to indirect lighting in a geometrically complex space.
- **Specular Scene (Metal-Spheres):** Selected specifically to evaluate the performance and denoising stability of ray-traced reflections on high-frequency glossy and mirrored surfaces.



Figure 3.1: The Sponza test scene, utilized for evaluating complex shadow occlusion and indirect lighting.

3.2.3 Independent and Dependent Variables

The experiment measures dependent performance and quality variables across a matrix of independent configurations.

Independent Variables:

- **Rendering Architecture:** Rasterization Baseline, SVGF Pre-Lighting, SVGF Post-Lighting, and RT No Denoiser (Raw).
- **Active Features:** Tested independently (Shadows only, Reflections only, Ambient Occlusion only) and concurrently (Full Pipeline).
- **Samples Per Pixel (SPP):** 1, 8, 16, and 32 SPP to measure the scaling of GPU cost versus noise reduction.

Dependent Variables:

- **Performance:** Total GPU time (ms) and individual GPU pass timings (e.g., Geometry, Ray Traversal, Spatial Filtering, Temporal Accumulation).
- **Perceptual Quality:** Learned Perceptual Image Patch Similarity (LPIPS), alongside traditional metrics (SSIM, PSNR), calculated against a 1000 SPP ground truth.

3.2.4 Instrumentation and Profiling

Performance data was gathered using hardware timestamp queries injected directly into the Vulkan command buffers. Timestamps were isolated for every individual stage of the pipeline to allow for granular pass breakdown analysis.

To eliminate temporal outliers and ensure steady-state measurements, an automated testing framework was utilized. For each variable configuration, 100 consecutive frames were recorded after the engine completed its initial warm-up phase. The middle 60 frames were extracted, averaged, and exported to a master CSV file for evaluation.

3.2.5 Pipeline Configurations

To isolate the performance overhead and visual impact of both the ray-traced features and the reconstruction algorithms, the renderer was evaluated under four distinct pipeline configurations:

- **Rasterization Baseline:** A traditional deferred pipeline utilizing Screen-Space Ambient Occlusion (SSAO), Screen-Space Reflections (SSR), and rasterized shadow mapping.
- **RT No Denoiser (Raw Baseline):** Ray-traced features evaluated without any spatial or temporal filtering, serving as a baseline to visualize raw Monte Carlo variance.
- **SVGF Pre-Lighting:** Ray-traced signals are denoised before the deferred lighting pass. This requires secondary data generation passes to store the denoised signals before material evaluation.
- **SVGF Post-Lighting:** The proposed optimized architecture where ray-traced signals and deferred lighting are evaluated simultaneously, with SVGF operating on the fully lit output.

Chapter 4

Implementation

4.1 Rasterization Baseline and Deferred Rendering

This section describes the rasterization-based foundation of the renderer and its critical role within the hybrid ray-tracing pipeline. Traditional rasterization is utilized to determine primary visibility and to generate the screen-space buffers (G-buffer) required for deferred lighting, ray generation and spatiotemporal reconstruction. This rasterization baseline serves as the strict reference configuration against which all ray-traced features are incrementally evaluated.

4.1.1 Fragment Shading in the Rasterization Pass

During the initial rasterization phase, scene geometry is processed through a traditional pipeline utilizing dedicated vertex and fragment shaders (**geometry.vert** and **geometry.frag**). The vertex stage transforms mesh vertices into clip space using the current view and projection matrices, while the fragment stage evaluates material properties and outputs per-pixel attributes to multiple render targets.

Rather than producing a fully shaded forward image, fragment shading in this pass focuses exclusively on exporting the material and geometric data required for later stages. These outputs include, surface albedo, normals, packed

material parameter (roughness and metallicity), and motion vectors. by deferring lighting calculations to the subsequent **DeferredLightingPass.cpp**, the initial rasterization pass remains highly efficient and entirely independent of the number of active light sources in the scene. Furthermore, this pass provides high-precision motion vectors and depth information, which are mathematically essential for temporal reprojection in both Temporal Anti-Aliasing (**TAAPass.cpp**) and SVGF-based denoising.

4.1.2 Deferred Rendering Architecture

The custom Vulkan renderer employs a strict deferred rendering architecture governed by a modular render graph (**RenderGraph.cpp**). The baseline pipeline consists of a depth pre-pass (**DepthPrePass.cpp**), a geometry pass (**GeometryPass.cpp**), and a screen-space deferred lighting pass. In the depth pre-pass, scene geometry is rasterized utilizing **depth.vert** and **depth.frag** without shading overhead to heavily reduce overdraw for subsequent stages.

The geometry pass writes per-pixel data to the G-buffer, successfully decoupling geometry processing from the lighting evaluation. This structural decoupling is the backbone of the hybrid pipeline: the G-buffer generated during rasterization is aggressively reused to guide ray-tracing origins, provide surface properties for shading ray hits, and supply the critical edge-stopping variance data needed for the SVGF filters. Because geometry processing and lighting are decoupled, the cost of additional ray-traced effects can be isolated to specific passes without affecting the rasterization workload.

4.1.3 The Lighting Pass and Baseline Features

Lighting is evaluated in a full-screen compute pass (**lighting.frag**) that operates entirely in screen space. For each pixel, surface properties are reconstructed from the G-buffer and combined with lighting data to compute the final shaded color.

To establish a baseline for performance comparisons, the engine implements traditional screen-space equivalents for all advanced lighting features. When ray tracing is disabled, the renderer natively falls back to Screen-Space Ambient Occlusion (**SSAOPass.cpp**), Screen-Space Reflections (**SSRPass.cpp**), and traditional rasterized shadow mapping (**ShadowPass.cpp** and **PointShadowPass.cpp**). This centralized design allows for a direct, 1:1 comparison between screen-space fallbacks and ray-traced techniques under identical lighting conditions.

4.1.4 Hybrid Ray Tracing Integration

Hybrid ray tracing is integrated into the renderer as a set of highly modular compute passes that augment the rasterization-based lighting. Ray tracing is applied selectively to ambient occlusion, shadows, and reflections, while rasterization remains strictly responsible for primary visibility and material evaluation.

4.2 G-buffer–Guided vs. Unguided Ray Generation

A core architectural feature of this engine is the implementation of both G-buffer-guided and unguided ray generation strategies for comparative analysis.

In the guided approach (**RTAOPass.cpp**, **RTRPass.cpp**, and **RTShadowPass.cpp**), ray generation is explicitly guided by the G-buffer to ensure that rays are only spawned for definitively visible pixels. Rays originate from surfaces identified during the rasterization pass, utilizing the reconstructed world-space positions and surface normals. This strategy drastically improves Bounding Volume Hierarchy (BVH) traversal coherence and heavily reduces unnecessary ray workloads compared to blind, object-space ray dispatching. Conversely, the engine also implements unguided variants (**RTUnguidedAOPass.cpp**, **RTUnguidedRTRPass.cpp**, and **RTUnguidedShadowPass.cpp**) to actively profile the performance overhead and architectural benefits of the guided visibility approach.

4.2.1 Ray-Traced Shadows

Ray-traced shadows are implemented by casting shadow rays from visible surface points directly toward light sources (**RTShadow.rgen** and **RTShadow.rchit**). Each shadow ray performs a single traversal of the acceleration structure (**AccelerationStructure.cpp**) and terminates immediately upon the first intersection. The result is a binary visibility signal indicating whether the surface point is occluded. While shadow rays are short and single-bounce—exhibiting relatively low initial variance—temporal accumulation and spatial filtering are heavily applied to stabilize the soft-shadow penumbras under strict 1 SPP budgets.

4.2.2 Ray-Traced Ambient Occlusion (RTAO)

To ground objects physically within the scene, Ray-Traced Ambient Occlusion (**RTAO.rgen**) replaces traditional SSAO. RTAO casts short, hemisphere-distributed cosine-weighted rays from the G-buffer surface normals to detect local geometric occlusion. Because it relies on true scene geometry via the Vulkan ray-tracing pipeline rather than depth-buffer approximations, RTAO eliminates the screen-space halo artifacts and off-screen occlusion failures inherent to rasterized SSAO.

4.2.3 Ray-Traced Reflections (RTR)

Ray-traced reflections are generated by emitting rays from visible surface points based on the camera view direction and the G-buffer surface normal (**RTR.rgen**, **RTR.rhit**, and **RTR.rmiss**). Unlike shadow or AO rays, reflection rays traverse significantly larger portions of the scene and inherently exhibit lower coherence, particularly when scattering across rough metallic surfaces.

Because the reflected radiance returned by these rays is highly volatile at low sample counts, reflections require the most aggressive spatiotemporal reconstruction in the pipeline.

4.2.4 Dual-Path SVGF Reconstruction

To resolve the Monte Carlo noise produced by low-sample ray tracing, the engine utilizes a custom Spatiotemporal Variance-Guided Filtering (SVGF) implementation. This reconstruction pipeline is divided into dedicated temporal accumulation (**SVGFTemporalPass.cpp** and **svgf_temporal.comp**) and spatial filtering passes (**SVGFSpatialPass.cpp** and **svgf_spatial.comp**). Crucially, the engine architecture does not rigidly enforce a single denoising pipeline; instead, it dynamically supports both ****Pre-Lighting**** and ****Post-Lighting**** SVGF integrations to facilitate direct comparative analysis.

- **Pre-Lighting Integration:** In this traditional configuration, the raw ray-traced shadow and reflection signals are denoised in isolation. The clean signals are then stored in secondary intermediate buffers before being integrated into the deferred lighting evaluation.
- **Post-Lighting Integration:** In this experimental configuration, the spatial and temporal filters operate directly on the combined, fully lit signal, utilizing variance estimation to preserve high-frequency edges while aggressively blurring noise.

This modular, dual-path design allows the renderer to explicitly profile the structural overhead, Vulkan memory footprint, and overall performance differences between the two distinct SVGF integration strategies.

Chapter 5

Results

5.1 Rasterization Baseline Performance

Table 5.1 summarizes the total GPU timings for the rasterization-only reference configuration. Total GPU time remained highly stable at approximately 1.35 ms in the tested scene, indicating a steady-state workload after the engine warm-up period.

Configuration	Total GPU Time (ms)
Rasterization Baseline (1 SPP)	1.35

Table 5.1: Rasterization-only baseline total GPU timings (Sponza, 1920×1080)

Traditional raster costs remain tightly clustered, as illustrated in the rasterization baseline performance distribution in Figure 5.1. The baseline cost is heavily dominated by traditional rasterized shadow mapping and SSAO. All measurements in this chapter were collected in the Sponza test scene at 1920×1080 . This 1.35 ms baseline serves as the stable reference point for evaluating the incremental overhead of ray-traced features in the following sections.

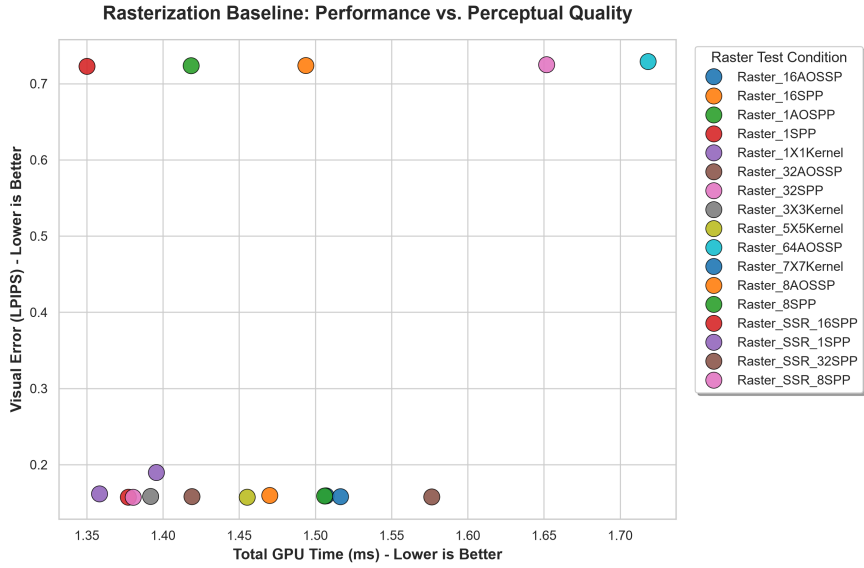


Figure 5.1: Performance distribution and timing clustering for the rasterization baseline configuration.

5.2 Ray-Traced Shadow Performance

This configuration replaces rasterized shadow mapping with ray-traced shadow queries while keeping all other pipeline stages identical to the rasterization baseline. Raster shadow passes are disabled, and a single ray-traced shadow pass followed by SVGF-based temporal filtering is enabled.

Configuration	Total GPU Time (ms)	Overhead vs Baseline
RT Shadows (1 SPP + SVGF)	1.88	+ 0.53 ms
RT Shadows (8 SPP + SVGF)	6.06	+ 4.71 ms
RT Shadows (16 SPP + SVGF)	7.49	+ 6.14 ms
RT Shadows (32 SPP + SVGF)	10.91	+ 9.56 ms

Table 5.2: Total GPU timings with ray-traced shadows enabled (SVGF Post-Lighting, Sponza, 1920×1080)

Compared to the rasterization baseline (1.35 ms), 1 SPP ray-traced shadows introduce an additional overhead of approximately 0.53 ms per frame. The isolated performance scaling of this effect is mapped in Figure 5.2.

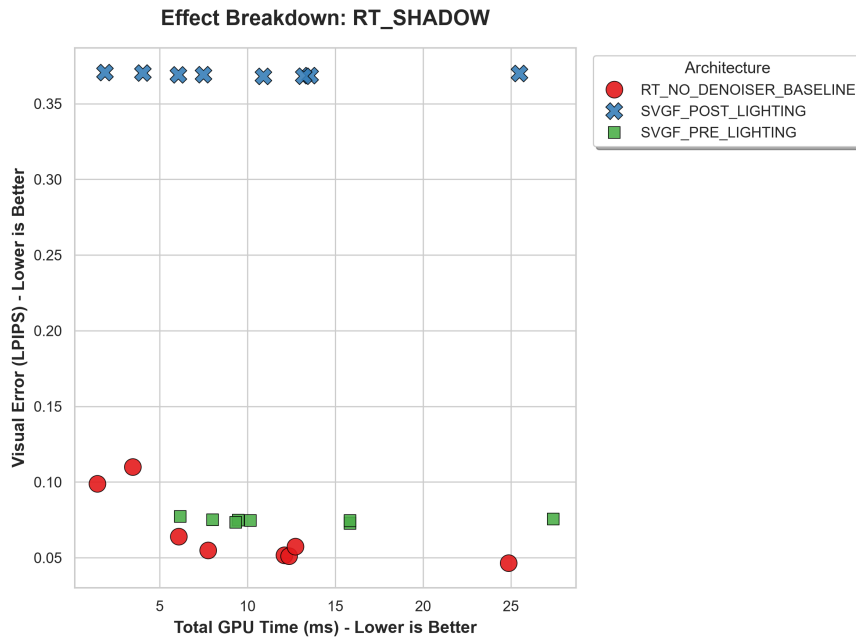


Figure 5.2: Isolated performance scaling and timing distribution of ray-traced shadow passes.

While the raw ray traversal cost is moderate, a granular analysis of individual GPU passes (depicted in Figure 5.3) confirms that a significant portion of the overhead is attributed to SVGF-based spatiotemporal denoising, which is required to stabilize the low-sample shadow signal. Despite the increased computational cost, ray-traced shadows provide physically accurate contact hardening and completely eliminate common shadow map artifacts such as aliasing and light leaking.

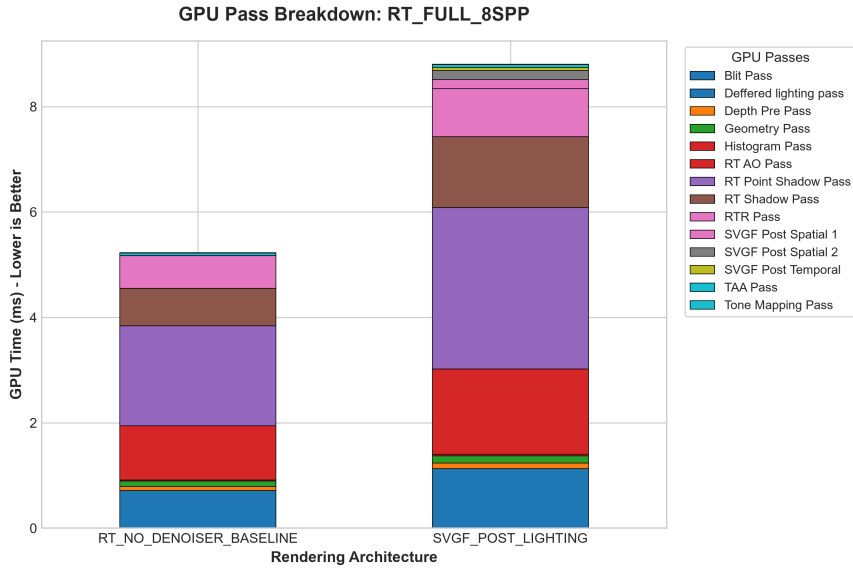


Figure 5.3: Granular pass-by-pass GPU timing breakdown for the full pipeline configuration at 8 SPP.

5.3 Ray-Traced Reflection Performance

Ray-traced reflections replace screen-space reflections (SSR) while preserving the same deferred lighting and material evaluation pipeline. Reflection rays are generated using G-buffer surface data and denoised using SVGF.

Configuration	Total GPU Time (ms)	Overhead vs Baseline
RT Reflections (1 SPP + SVGF)	2.05	+ 0.70 ms
RT Reflections (8 SPP + SVGF)	3.84	+ 2.49 ms
RT Reflections (16 SPP + SVGF)	5.81	+ 4.46 ms
RT Reflections (32 SPP + SVGF)	9.88	+ 8.53 ms

Table 5.3: Total GPU timings for ray-traced reflections at varying sample counts (SVGF Post-Lighting, Sponza, 1920×1080)

Compared to the raster baseline—which utilizes rudimentary screen-space reflections at a negligible cost—1 SPP ray-traced reflections introduce a controllable overhead of 0.70 ms. As visualized in the individual effect breakdown in Figure 5.4, reflection cost scales sub-linearly with sample count due to the fixed-cost amortization of spatiotemporal denoising.

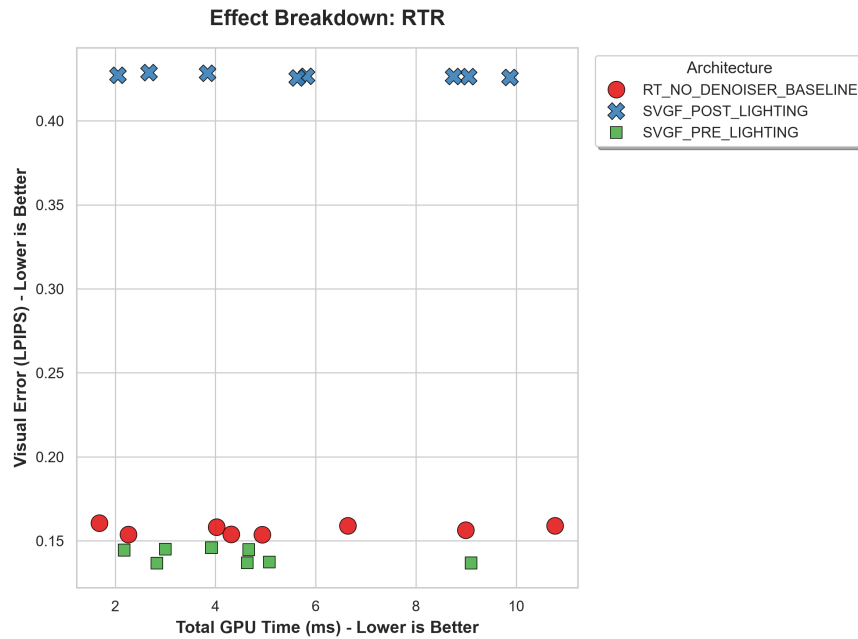


Figure 5.4: Performance scaling behavior of ray-traced reflections across varying sample counts.

While higher sample counts reduce initial noise, the marginal visual quality improvement diminishes rapidly relative to the added GPU cost. These results indicate that a 1 SPP configuration combined with aggressive temporal accumulation provides the most favorable balance between real-time performance and perceptual visual quality.

5.4 Ray-Traced Ambient Occlusion Performance

Ray-traced ambient occlusion (RTAO) replaces the traditional screen-space ambient occlusion (SSAO) pass. Unlike SSAO, which relies on depth buffer approximations and is prone to off-screen occlusion failures, RTAO casts hemisphere-distributed rays to calculate physically accurate local geometric occlusion. The raw RTAO signal is then stabilized using SVGF.

Configuration	Total GPU Time (ms)	Overhead vs Baseline
RTAO (1 SPP + SVGF)	2.31	+ 0.96 ms
RTAO (8 SPP + SVGF)	5.77	+ 4.42 ms
RTAO (16 SPP + SVGF)	9.44	+ 8.09 ms
RTAO (32 SPP + SVGF)	16.82	+ 15.47 ms

Table 5.4: Total GPU timings for Ray-Traced Ambient Occlusion at varying sample counts (SVGF Post-Lighting, Sponza, 1920×1080)

Compared to the 1.35 ms rasterization baseline, enabling 1 SPP RTAO introduces a moderate overhead of 0.96 ms per frame. Similar to the trends observed in ray-traced shadows and reflections, the performance cost of RTAO scales sub-linearly with the sample count due to the fixed overhead of the spatiotemporal denoising passes.

However, scaling the ray budget to 32 SPP increases the total frame time to an unviable 16.82 ms. This demonstrates the same severe diminishing returns seen in the other effects: pushing raw sample counts incurs massive traversal penalties that far outweigh the benefits, proving that relying heavily on the temporal accumulation of a 1 SPP signal is structurally superior for real-time budgets.

5.5 Hybrid Configuration Comparison

To evaluate the effectiveness of different hybrid rendering configurations, the full rasterization baseline is compared against the full combined ray-traced pipeline (Shadows, Reflections, and RTAO active simultaneously). Table 5.5 summarizes total GPU frame time and incremental cost relative to the raster-only baseline.

Configuration	Total GPU (ms)	Overhead vs Baseline
Rasterization Only	1.35	—
RT Full Pipeline (1 SPP + SVGF)	2.24	+ 0.89 ms
RT Full Pipeline (8 SPP + SVGF)	8.80	+ 7.45 ms
RT Full Pipeline (16 SPP + SVGF)	10.01	+ 8.66 ms
RT Full Pipeline (32 SPP + SVGF)	20.06	+ 18.71 ms

Table 5.5: Comparison of hybrid rendering configurations (SVGF Post-Lighting Full Pipeline, Sponza, 1920×1080)

All configurations were measured using identical scenes, resolution, and capture methodology. This overarching performance-versus-quality trade-off is visually captured in the full pipeline scatter plot (Figure 5.5) and explicitly detailed in the convergence curve (Figure 5.6).

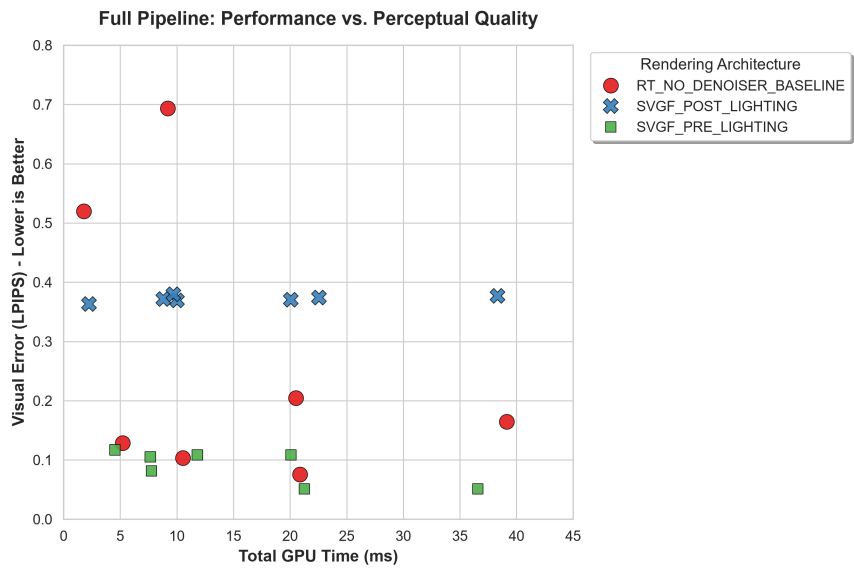


Figure 5.5: Comprehensive performance-versus-quality trade-off across full pipeline configurations.

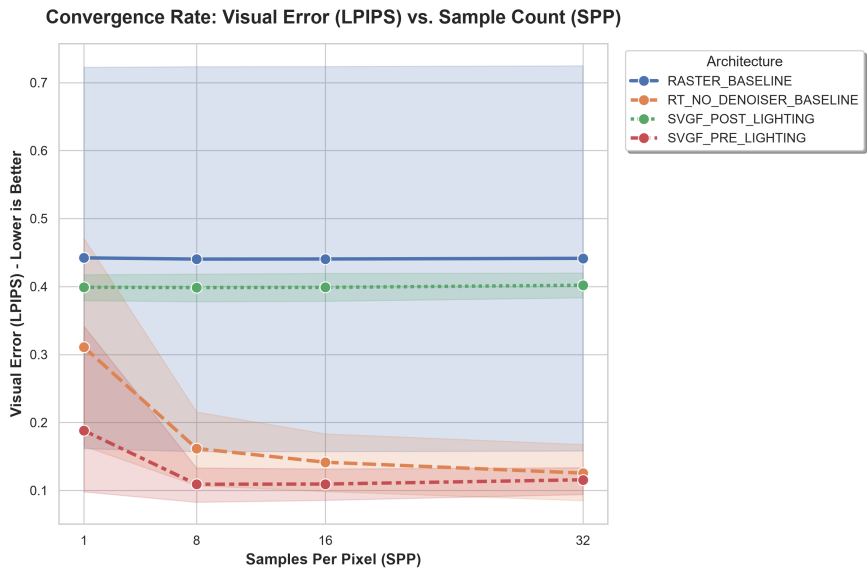


Figure 5.6: Sample count (SPP) scaling convergence chart demonstrating diminishing perceptual returns.

The quantitative data shows that a complete hybrid ray-tracing pipeline can be integrated at a highly efficient 2.24 ms per frame when restricted to 1 SPP. This represents a total pipeline overhead of roughly +0.89 ms over traditional rasterization while providing drastically superior lighting, shadows, and reflections.

The SPP convergence charts demonstrate a steep law of diminishing returns when pushing sample counts to 8 SPP, 16 SPP, and beyond. While an 8 SPP configuration lowers raw temporal variance before denoising, the total GPU time spikes to 8.80 ms. The additional traversal cost grows significantly faster than the perceptual LPIPS improvement after SVGF reconstruction. This suggests that, at least within the tested scene, temporal accumulation is the primary factor stabilizing low-sample ray-traced signals, with additional per-frame samples contributing comparatively little beyond that point.

5.5.1 Architectural Efficiency: Pre-Lighting vs. Post-Lighting (Sponza)

To evaluate Hypothesis 2, the total GPU frame time of the traditional Pre-Lighting SVGF architecture was directly compared against the unified Post-Lighting architecture running the complete hybrid pipeline in the Sponza environment.

Architecture (Full Pipeline, Guided 1 SPP)	Total GPU Time (ms)
Pre-Lighting SVGF	4.52
Post-Lighting SVGF	2.24

Table 5.6: Architectural performance comparison for the full hybrid pipeline (Sponza, 1920×1080)

As shown in Table 5.6, transitioning from Pre-Lighting to Post-Lighting cuts the total ray-tracing rendering overhead in half. Because the Post-Lighting architecture eliminates the need for secondary intermediate buffers to store the isolated denoised signals, it drastically reduces memory bandwidth bottlenecks, dropping the full pipeline cost from 4.52 ms down to a highly efficient 2.24 ms. This confirms the performance advantage hypothesized for diffuse and shadow-dominated environments.

5.6 G-buffer-Guided vs. Unguided Ray Dispatching

To validate the architectural decision of utilizing the G-buffer to guide ray generation, a direct performance comparison was conducted between the standard G-buffer-guided pipeline and the unguided fallback variants. In the unguided configuration, rays are dispatched blindly into the scene without prior rasterized visibility culling or normal/roughness data pre-caching.

Table 5.7 isolates the total GPU timings for both individual reflections (RTR) and the Full Ray-Traced Pipeline under the `SVGF_PRE_LIGHTING` architecture at 1 SPP and 8 SPP.

Configuration	Guided (G-buffer)	Unguided	Performance Penalty
RT Reflections (1 SPP)	2.18 ms	2.83 ms	+ 0.65 ms
RT Reflections (8 SPP)	3.00 ms	4.64 ms	+ 1.64 ms
RT Full Pipeline (1 SPP)	4.52 ms	7.75 ms	+ 3.23 ms
RT Full Pipeline (8 SPP)	7.66 ms	21.26 ms	+ 13.60 ms

Table 5.7: Total GPU timings comparing Guided vs. Unguided ray generation (Sponza, 1920×1080)

As the data explicitly demonstrates, relying on the G-buffer for primary visibility and ray origin calculation dramatically reduces the Bounding Volume Hierarchy (BVH) traversal overhead. At a budget of 1 SPP, blindly dispatching unguided rays for the full pipeline nearly doubles the rendering cost (from 4.52 ms up to 7.75 ms). The architectural flaw of unguided dispatching becomes catastrophic at higher sample counts; pushing the full unguided pipeline to 8 SPP results in a massive 21.26 ms frame time, rendering it entirely unviable for real-time applications (running nearly three times slower than its guided counterpart at 7.66 ms). By providing accurate surface normals and spatial bounds upfront, the G-buffer-guided approach severely restricts the volume of the scene the ray tracing cores must evaluate.

While Table 5.7 focuses on reflections and the full pipeline, isolated per-feature profiling reveals a more nuanced picture than a uniform "guided is always faster" pattern. Ray-traced shadows broadly follow the same trend at moderate-to-high sample counts, with guided dispatch outperforming unguided at 8, 16, and 32 SPP, table 5.8. At 1 SPP, however, the guided shadow path is marginally slower than its unguided counterpart (8.01 ms vs. 6.17 ms), suggesting that at very low sample counts the fixed overhead of the additional G-buffer lookups briefly outweighs the traversal coherence gained.

RTAO does not follow this pattern at all: the guided variant is consistently more expensive than the unguided variant across every tested sample count, table 5.9, by as much as $2.6\times$ at 32 SPP. This divergence is fundamentally rooted in the differing Vulkan API execution models utilized by each path. While both implementations dispatch an identical sample budget per pixel, the unguided path leverages Inline Ray Tracing (rayQueryEXT), evaluating intersections directly within the compute shader and bypassing the Shader Binding Table (SBT) entirely. In contrast, the guided path utilizes standard Pipeline Ray Tracing (traceRayEXT), incurring the overhead of SBT lookups and context switching between shader stages. Furthermore, the unguided implementation computes a highly simplified, view-facing occlusion estimate rather than utilizing accurate geometric normals. Consequently, the traversal-coherence benefits of G-buffer guidance are eclipsed by the architectural overhead of the ray tracing pipeline compared to lightweight inline queries.

Configuration	Guided (G-buffer)	Unguided	Delta
RT Shadows (1 SPP)	8.01 ms	6.17 ms	+1.84 ms (guided slower)
RT Shadows (8 SPP)	9.32 ms	10.16 ms	-0.84 ms (guided faster)
RT Shadows (16 SPP)	9.48 ms	15.84 ms	-6.36 ms (guided faster)
RT Shadows (32 SPP)	15.84 ms	27.41 ms	-11.57 ms (guided faster)

Table 5.8: Total GPU timings comparing Guided vs. Unguided ray generation for shadows (SVGF Pre-Lighting, Sponza, 1920×1080)

Configuration	Guided (G-buffer)	Unguided	Delta
RTAO (1 SPP)	2.66 ms	2.63 ms	+0.03 ms (guided slower)
RTAO (8 SPP)	6.16 ms	4.33 ms	+1.83 ms (guided slower)
RTAO (16 SPP)	10.37 ms	4.61 ms	+5.76 ms (guided slower)
RTAO (32 SPP)	14.82 ms	5.76 ms	+9.06 ms (guided slower)

Table 5.9: Total GPU timings comparing Guided vs. Unguided ray generation for RTAO (SVGF Pre-Lighting, Sponza, 1920×1080)

Conclusion:

Within the tested scene, the configuration combining G-buffer-guided ray tracing at 1 SPP with spatiotemporal denoising offers the strongest measured balance between GPU cost and image quality. These benchmark results support the hybrid rendering strategy proposed earlier: selective ray tracing guided by rasterized visibility, combined with a 1 SPP budget and temporal denoising, outperformed both pure screen-space techniques and higher-sample ray tracing in terms of perceptual quality relative to GPU cost, under the conditions tested here.

5.7 Reflection Stress Test: Metal-Spheres Scene

To evaluate the stability of ray-traced reflections on high-frequency glossy and mirrored surfaces, the Metal-Spheres scene was analyzed. Unlike the diffuse-heavy Sponza environment, this scene rigorously tests the denoiser’s ability to reconstruct sharp specular details and the traversal costs associated with incoherent secondary rays.

5.7.1 Performance Overhead and Discretization Artifacts

Table 5.10 summarizes the total GPU timings and perceptual metrics for reflections under the traditional Pre-Lighting SVGF architecture, comparing G-buffer-guided dispatch against unguided object-space dispatch.

Configuration (Pre-Lighting)	Total GPU (ms)	RTR Pass (ms)	LPIPS ($\downarrow$)
Raster SSR Baseline (1 SPP)	1.74	0.08	0.0796
RTR Guided (1 SPP)	2.06	0.10	0.0913
RTR Guided (8 SPP)	2.64	0.63	0.0919
RTR Guided (16 SPP)	3.26	1.24	0.0918
RTR Guided (32 SPP)	4.46	2.44	0.0913
RTR Unguided (1 SPP)	2.23	0.21	0.0143
RTR Unguided (8 SPP)	2.88	0.82	0.0103
RTR Unguided (16 SPP)	3.58	1.53	0.0076
RTR Unguided (32 SPP)	4.93	2.90	0.0066

Table 5.10: Performance and quality metrics for specular reflections (Metal-Spheres, 1920×1080)

As observed in the Sponza benchmarks, G-buffer-guided ray generation remains computationally superior. At a budget of 1 SPP, the unguided reflection pass (0.21 ms) takes double the computation time of the guided pass (0.10 ms). However, the unguided approach yields significantly better perceptual quality on mathematically curved surfaces (improving LPIPS from 0.0913 to 0.0143). This is because unguided primary rays intersect the acceleration structure directly, bypassing the rasterization discretization and normal interpolation artifacts inherent to the G-buffer representation of spheres.

5.7.2 The Failure of Post-Lighting on Specular Surfaces

While the Post-Lighting SVGF architecture proved highly efficient for combined rendering in Sponza, the Metal-Spheres stress test reveals a critical architectural limitation when handling pure specular signals.

SVGF Architecture (Guided, 1 SPP)	Total GPU Time (ms)	LPIPS (↓)
Pre-Lighting SVGF	2.06	0.0913
Post-Lighting SVGF	2.12	0.5583

Table 5.11: Architectural comparison of SVGF integration on specular surfaces

As the empirical data explicitly demonstrates, applying spatiotemporal filtering after the deferred lighting pass completely destroys high-frequency specular reflections, resulting in an unusable LPIPS error of 0.5583. Because Post-Lighting SVGF filters the fully lit signal—blending sharp reflections across neighboring pixels based on geometric variance—it fundamentally fails to preserve the view-dependent nature of mirror reflections. Pre-Lighting integration remains the absolute requirement for physically accurate ray-traced reflections, as it allows the denoiser to operate strictly on the raw reflection vector data before material shading.

5.8 Visual Quality and Perceptual Metrics

To objectively quantify the visual fidelity of the hybrid pipeline, rendering outputs were evaluated against a high-sample ground truth using three standard metrics: Peak Signal-to-Noise Ratio (PSNR), Structural Similarity Index Measure (SSIM), and Learned Perceptual Image Patch Similarity (LPIPS). While PSNR and SSIM measure strict mathematical pixel-level differences, LPIPS utilizes deep learning networks to evaluate how humans perceive structural noise and blurring. For PSNR and SSIM, higher values are better. For LPIPS, lower values indicate less perceptual error.

5.8.1 The Convergence of Raw Ray Tracing

To establish a baseline for how ray sampling scales without interference, the Full Ray-Traced Pipeline was first evaluated completely raw, without any SVGF spatial or temporal filtering applied.

Configuration (No Denoiser)	LPIPS (↓)	PSNR (↑)	SSIM (↑)
RT Full Pipeline (1 SPP)	0.5195	17.138	0.5560
RT Full Pipeline (8 SPP)	0.1283	21.146	0.7888
RT Full Pipeline (16 SPP)	0.1031	21.447	0.8232
RT Full Pipeline (32 SPP)	0.0753	21.614	0.8557

Table 5.12: Quality Metrics for Raw Ray Tracing (No Denoiser) (Sponza, 1920×1080)

As expected, raw Monte Carlo rendering exhibits standard mathematical convergence. At 1 SPP, the image is overwhelmed by high-frequency variance, resulting in a poor LPIPS score of 0.5195. As the ray budget increases to 8 SPP, the variance plummets, drastically improving the LPIPS score to 0.1283. Pushing the engine to 32 SPP yields a near-pristine image with an LPIPS of 0.0753 and an SSIM of 0.8557. However, as established in Section 5.5, achieving this raw quality requires an unviable 20+ ms per frame.

5.8.2 The Denoising Plateau (SVGF)

When the Post-Lighting SVGF pipeline is engaged, the scaling behavior of the perceptual metrics changes fundamentally.

Configuration (SVGF Enabled)	LPIPS ($\downarrow$)	PSNR ($\uparrow$)	SSIM ($\uparrow$)
RT Full Pipeline (1 SPP)	0.3632	10.495	0.4040
RT Full Pipeline (8 SPP)	0.3716	10.237	0.3843
RT Full Pipeline (16 SPP)	0.3693	10.218	0.3830
RT Full Pipeline (32 SPP)	0.3704	10.213	0.3839

Table 5.13: Quality Metrics with Post-Lighting SVGF Enabled (Sponza, 1920×1080)

Comparing the 1 SPP configurations, the SVGF filter successfully rescues the noisy raw signal, improving the LPIPS error from 0.5195 down to 0.3632, producing a perceptually stable, interactive image. However, Table 5.13 reveals a critical architectural bottleneck: the quality metrics completely plateau beyond 1 SPP. When the ray budget is increased to 8, 16, or even 32 SPP, the LPIPS score stagnates at approximately 0.37, while PSNR and SSIM show no mathematical improvement.

5.8.3 Note on Metric Interpretation

It should be noted that the absolute PSNR and SSIM values reported in Table 5.13 are affected by a known limitation in the current albedo demodulation implementation. The Post-Lighting SVGF pipeline demodulates the diffuse albedo prior to filtering—separating irradiance from surface reflectance so that spatial and temporal filtering operates on a smoother, lower-frequency signal—and re-multiplies the albedo back in after denoising.

In the current implementation, this remodulation step does not fully preserve the original image brightness, resulting in a systematic darkening of the final output relative to the ground truth reference. This explains why LPIPS, which evaluates perceptual and structural similarity via deep feature comparison, remains relatively stable and low across sample counts, while PSNR and SSIM—both of which are highly sensitive to absolute pixel-level intensity differences—report a large error even at the point of visual convergence.

The relative trends observed across sample counts (i.e., the plateau beyond 1 SPP) therefore remain valid, since the darkening bias is constant across all configurations in Table 5.13. However, the absolute PSNR/SSIM values in this table should not be directly compared against the raw, undenoised results in Table 5.12, nor interpreted as representative of SVGF’s theoretical quality ceiling. A corrected remodulation step—likely involving a review of the albedo epsilon clamping and the linear/sRGB colorspace consistency between the demodulation and remodulation stages—is identified as a direct avenue for future work.

5.9 Chapter Summary

These results indicate that the SVGF spatiotemporal filter, rather than the ray sample budget, is the dominant factor governing final image quality in the hybrid pipeline. The LPIPS scores in Table 5.13 remain essentially flat across 1, 8, 16, and 32 SPP, in contrast to the clear monotonic convergence observed in the undenoised results of Table 5.12. Since LPIPS is comparatively robust to the global brightness bias introduced by the current albedo remodulation implementation (see above), this plateau is interpreted as a genuine property of the à-trous filtering stage: once temporal accumulation and spatial filtering have stabilized the signal, additional ray samples provide diminishing returns in perceptual terms, while GPU cost continues to scale sharply (up to 20.06 ms at 32 SPP, as shown in Table 5.5).

The absolute PSNR and SSIM values in Table 5.13 are not used to support this conclusion, as they are confounded by the remodulation-induced darkening discussed above. A precise characterization of the filter’s intrinsic blurring—independent of the current brightness bias—is left as future work once the remodulation step is corrected.

Taken together, these findings support 1 SPP as the most defensible sampling budget for the current implementation: it captures the large majority of the perceptual quality improvement SVGF is capable of providing at a fraction of the GPU cost of higher sample counts, and the available evidence does not show that additional samples meaningfully change that trade-off.

It is important to note that this quality-per-cost advantage applies specifically to the diffuse and shadow signals evaluated in the Sponza scene; Section 5.7.2 demonstrates that Post-Lighting integration is unsuitable for specular reflection signals, where Pre-Lighting integration remains necessary.

Chapter 6

Discussion and Limitations

6.1 Signal-Dependent Viability of Post-Lighting Integration

While Section 5.5 demonstrates that Post-Lighting SVGF offers substantial architectural and performance advantages for the combined Sponza pipeline, Section 5.7 reveals a critical limitation on the Metal-Spheres scene: filtering the fully-lit signal collapses high-frequency specular detail, increasing LPIPS from 0.091 (Pre-Lighting) to 0.558 (Post-Lighting) under otherwise identical conditions. This indicates that Post-Lighting SVGF is not a universally superior architecture, but one whose viability depends on the frequency characteristics of the dominant signal in a given scene. Diffuse-dominated and shadow-dominated content — where geometric edge-stopping functions correlate well with the underlying signal — benefit from the simplified, single-pass Post-Lighting integration. Specular- and reflection-dominated content, where the signal is highly view-dependent and decorrelated from surface geometry, requires the raw reflection vector data to be denoised before modulation with lighting, i.e., Pre-Lighting integration.

6.2 Practical Quality-per-Cost Tradeoffs

Although perceptual metrics quantify visual error against a high-sample baseline, the practical optimization of a hybrid rendering engine relies on balancing visual stability against strict frame-time budgets. The experimental results demonstrate that spatiotemporal denoising (SVGF) is the primary driver of perceived image quality at low sample counts.

For ray-traced shadows, reflections, and ambient occlusion, increasing the sampling rate beyond 1 SPP (to 8, 16, or 32 SPP) primarily serves to reduce Monte Carlo variance in the raw, pre-filtered signal. However, it is the spatiotemporal filter that executes the vast majority of the visual convergence across frames. Consequently, under interactive frame budgets, the marginal visual improvement obtained by increasing the sample count to 8 SPP or beyond diminishes rapidly. The spatial and temporal stability gained by brute-forcing 8 to 32 samples per pixel fundamentally fails to justify the exponential rise in raw ray traversal overhead.

6.3 Feature Coverage versus Sample Density

A central finding of this evaluation is that enabling additional ray-traced effects at 1 SPP provides a drastically greater perceptual upgrade than spending that same millisecond budget on increasing the sample density of a single existing effect.

As demonstrated in the results, the unified Post-Lighting architecture is capable of running the entire hybrid pipeline (shadows, reflections, and ambient occlusion concurrently) at a highly efficient 2.24 ms when restricted to 1 SPP. Conversely, pushing that exact same pipeline to 8 SPP skyrockets the total GPU cost to 8.80 ms, while yielding negligible visual gains post-reconstruction.

These results dictate a clear design guideline for real-time engine architecture: hybrid pipelines should prioritize feature coverage over sample density. Allocating computational budgets toward selective 1 SPP dispatches across multiple features maximizes the visual value returned per millisecond. Higher sample counts remain mathematically and computationally inefficient for real-time rendering and are strictly better suited for non-real-time or offline rendering scenarios.

6.4 Methodological Limitations

While these findings offer clear architectural guidance for hybrid engines, certain limitations of this study must be acknowledged:

- **Scene Complexity and Diversity:** The evaluation was conducted using the Sponza environment. While this scene rigorously tests complex shadow occlusion, soft penumbras, and indirect lighting bounces in a highly detailed geometric space, it is fundamentally a static indoor environment. It does not fully capture the Bounding Volume Hierarchy (BVH) rebuild costs, dynamic object updates, or specialized traversal bottlenecks associated with massive, foliage-heavy open-world scenes.
- **Hardware Sensitivity:** Timings were gathered on a fixed upper-midrange to enthusiast-tier GPU baseline (AMD Radeon RX 9070 XT). While the relative scaling trends (e.g., sub-linear traversal cost versus constant SVGF overhead) and the architectural advantages of Post-Lighting integration remain objectively valid across hardware targets, absolute millisecond costs and RT-core scaling will vary across different GPU microarchitectures and vendor-specific traversal pipelines.

Chapter 7

Conclusion

This research set out to evaluate the performance overhead, perceptual trade-offs, and architectural efficiency of integrating hardware-accelerated ray tracing into a deferred rasterization pipeline. By meticulously profiling GPU timings, evaluating spatiotemporal reconstruction, and proposing a unified filtering architecture, this work establishes clear operational boundaries for budgeting ray-traced shadows, reflections, and ambient occlusion under real-time constraints.

7.1 Summary of Key Findings

- **Architectural Efficiency of Post-Lighting SVGF:** For diffuse and shadow-dominated signals, the proposed Post-Lighting SVGF architecture eliminates the need for secondary G-buffer storage passes, reducing pipeline complexity relative to Pre-Lighting integration. However, this benefit does not extend to specular reflections: the Metal-Spheres stress test (section 5.7.2) shows Post-Lighting integration destroys high-frequency reflection detail, and Pre-Lighting integration remains necessary for physically plausible mirror-like surfaces. A production hybrid pipeline should therefore apply Post-Lighting SVGF selectively, per-signal, rather than as a blanket architecture.
- **The Cost of Pipeline Overhead:** While G-buffer guidance vastly improves traversal coherence for complex effects like reflections and

shadows, it is not universally optimal. For short, localized queries like Ambient Occlusion, the fixed architectural overhead of Pipeline Ray Tracing (Shader Binding Tables and context switching) outweighs the coherence benefits. In these specific cases, lightweight Inline Ray Tracing without G-buffer guidance proved significantly faster, indicating that hybrid engines must match the API execution model to the specific ray profile.

- **Low Overhead Integration:** Hybrid ray tracing can be integrated into a Vulkan deferred rasterizer with remarkable efficiency. The experimental data proves that a complete hybrid pipeline—simultaneously running ray-traced shadows, reflections, and ambient occlusion at 1 SPP—can be achieved for a total frame-time of just 2.24 ms. This represents an incremental overhead of only +0.89 ms over the traditional rasterization baseline (1.35 ms) on modern enthusiast-tier hardware.
- **Dominance of Reconstruction:** Spatiotemporal filtering (SVGF) acts as the fundamental backbone of low-sample hybrid rendering. Because spatial and temporal accumulation aggressively eliminates high-frequency noise without requiring additional ray queries, a 1 SPP + SVGF configuration consistently yields the absolute optimal perceptual quality-per-millisecond ratio.
- **Diminishing Returns of Ray Density:** Increasing ray sampling density from 1 SPP to 8, 16, or 32 SPP introduces a severe performance penalty. For example, pushing the full pipeline from 1 SPP to 8 SPP spikes the GPU cost from 2.24 ms to 8.80 ms, while offering negligible visual stability improvements post-denoising.

7.2 Final Remarks

Ultimately, this work confirms that naive, full-resolution ray tracing remains fundamentally unnecessary for achieving modern real-time visual fidelity. The most computationally efficient and scalable strategy for real-time applications relies on a highly tailored hybrid approach: combining traditional rasterization for primary visibility with selective, 1 SPP ray dispatches. Crucially, the execution of these dispatches must be adapted to the signal—utilizing G-buffer-guided Pipeline Ray Tracing for complex traversal (reflections, shadows) and lightweight Inline Ray Tracing for short-range queries (ambient occlusion). When backed by signal-specific spatiotemporal reconstruction (Post-Lighting for diffuse, Pre-Lighting for specular), this architecture delivers immense perceptual value well within strict interactive frame budgets

Chapter 8

Future Work

The research presented in this thesis focused on the incremental cost of hard-surface reflections and direct shadowing. While these provide the most immediate visual upgrade for hybrid pipelines, several avenues exist to extend this work toward a fully comprehensive hybrid renderer.

8.1 Albedo Remodulation and Energy Conservation

As identified during the evaluation of perceptual metrics, the current Post-Lighting SVGF implementation introduces a systematic darkening bias. This energy loss occurs during the albedo demodulation and remodulation phases, where irradiance is temporarily separated from surface reflectance to allow the spatiotemporal filter to operate on a smoother, texture-independent signal.

Future iterations of this rendering pipeline must address this limitation to ensure strict energy conservation. Specifically, future work should investigate potential discrepancies in linear-to-sRGB colorspace conversions occurring between the G-buffer generation and the final composition passes. Additionally, a rigorous review of the epsilon clamping values—used to prevent divide-by-zero errors when isolating irradiance—is required. Correcting this step is essential not only to preserve the physical accuracy of the lighting model, but also to allow traditional pixel-wise metrics like PSNR and SSIM

to accurately reflect the true reconstruction quality ceiling of the denoiser without being confounded by global brightness shifts.

8.2 Global Illumination and Diffuse Inter-reflection

This study was limited to direct lighting and specular reflections. However, the most significant visual gap between rasterization and path tracing lies in indirect diffuse illumination (Global Illumination). Future research should evaluate the integration of probe-based solutions (such as DDGI) or screen-space history reservoirs (ReSTIR GI) alongside the G-buffer-guided techniques implemented here. Analyzing whether the "1 SPP + Denoising" strategy holds for low-frequency indirect diffuse light which typically exhibits different variance characteristics than high-frequency specular reflections would be a valuable continuation of this performance analysis.

8.3 Machine Learning-Based Reconstruction

Our implementation relied on SVGF, a traditional wavelet-based filter. While effective, it struggled with ghosting in high-motion scenarios and blurring of fine specular details. Modern production pipelines are increasingly adopting machine learning-based reconstruction (e.g., NVIDIA DLSS Ray Reconstruction) or advanced heuristics (NVIDIA NRD). A comparative analysis benchmarking SVGF against these "black box" production denoisers would provide deeper insight into whether the overhead of neural inference networks pays off in terms of LPIPS quality compared to the relatively cheap wavelet filter used in this work.

8.4 Dynamic Quality Scaling

Currently, the renderer uses fixed sampling rates (1, 4, or 8 SPP) selected at compile or configuration time. A more robust engine implementation would employ dynamic quality scaling. Future work could investigate an adaptive sampler that uses the G-buffer's roughness channel or the previous frame's variance estimate to locally vary the ray count per pixel. For example, perfectly smooth surfaces might require only 1 SPP, while rougher dielectrics could trigger 2-4 SPP to stabilize the denoiser. Implementing such a "variance-guided ray dispatch" system could potentially lower the total frame time further by allocating the ray budget only where strictly necessary.

8.5 Cross-Vendor Hardware Analysis

All performance metrics and timestamp queries in this thesis were collected on an AMD Radeon (RDNA) architecture. Because hardware vendors employ fundamentally different microarchitectural approaches to ray tracing—such as AMD's Ray Accelerators embedded within Compute Units compared to NVIDIA's dedicated RT Cores or Intel's Ray Tracing Units (RTUs)—traversal throughput, bounding box intersection efficiency, and cache behavior vary significantly across platforms.

Consequently, the specific millisecond overheads and scaling factors measured in this study may not generalize linearly to NVIDIA RTX or Intel Arc hardware. Repeating these isolated pass breakdowns across alternative GPU architectures represents an important avenue of future research to establish an empirical, hardware-agnostic baseline for the cost of hybrid ray tracing.

8.6 Human Perceptual Validation

While LPIPS serves as the current academic gold standard for measuring perceptual quality having been trained on massive datasets of human judgments it remains an algorithmic proxy. The mathematical quality plateaus identified in this study strongly suggest that additional ray samples provide negligible visual benefit, but this conclusion inherently relies on the accuracy of the metric.

A crucial next step for this research is empirical validation. Future work should conduct live, double-blind A/B testing with human subjects to verify whether the structural convergence measured by LPIPS perfectly correlates with what a player actually perceives during active gameplay. Gathering direct user feedback would definitively confirm whether the 1 SPP denoised configuration is truly visually indistinguishable from higher sampling rates in a dynamic production context.

Chapter 9

Critical Reflection

9.1 Technical Challenges in Vulkan Integration

The transition from a theoretical understanding of ray tracing to a practical Vulkan implementation proved to be the most significant challenge of this graduation work. Unlike higher-level graphics APIs, Vulkan demands explicit and unforgiving management of synchronization. As noted in *Mastering Graphics Programming with Vulkan*, the responsibility for hazard tracking is shifted entirely to the developer, making the correct implementation of `VkImageMemoryBarrier` and pipeline stages a primary source of complexity in modern engine development (Castorina and Sassone, 2023).

In the early stages of implementing the G-buffer-guided path, I frequently encountered synchronization validation errors and “black screen” race conditions where resources were being read before write operations had completed. Debugging these issues required a fundamental shift in mindset—from viewing rendering as a linear sequence of draw calls to understanding it as an asynchronous graph of dependencies.

This steep learning curve is ultimately what made the implementation of the modular render graph architecture so invaluable. By abstracting the explicit management of memory barriers and layout transitions into a data-driven graph, I could automatically and safely hand off the G-buffer from the rasterization pass to the ray generation shaders without brittle manual hazard tracking. This pivotal structural improvement highlighted that in

hybrid rendering, the core complexity lies not strictly in the ray traversal itself, but in the safe, automated, and efficient sharing of data between the raster and compute pipelines.

Integrating hardware-accelerated ray tracing introduced a distinct architectural challenge. Setting up Shader Binding Tables (SBTs) and managing Bounding Volume Hierarchy (BVH) construction proved significantly more difficult than initially anticipated. Furthermore, while the implementation is stable for the scope of the Sponza test scene evaluated in this thesis, scaling the architecture to massive, production-level environments like the Amazon Bistro scene resulted in frequent GPU memory crashes. This starkly highlighted the fragility of a solo, research-grade engine implementation compared to a battle-tested commercial production engine.

9.2 The Reality of Real-Time Ray Tracing

Another critical realization was the massive discrepancy between raw ray-tracing performance and perceived image quality. Initially, I underestimated the absolute necessity of denoising. My early implementations of 1 SPP reflections produced mathematically correct but visually unusable Monte Carlo noise.



(a) Raw Ray Tracing (1 SPP) - Extreme Noise



(b) SVGF Denoised (1 SPP) - Stabilized Output

Figure 9.1: Visual comparison demonstrating the necessity of spatiotemporal reconstruction at low sample budgets.

Integrating the SVGF pipeline demonstrated that for real-time applications, the actual “ray tracing” is often the easiest part of the pipeline; the “reconstruction” is where visual fidelity is truly earned. This project forced me to

dive deep into signal processing concepts—variance estimation, temporal accumulation, and à-trous wavelet filtering—which were not part of my initial scope but became essential to proving my hypothesis. Realizing that a 2.0 ms reflection pass actually consists of merely 0.5 ms of ray traversal and 1.5 ms of intensive spatiotemporal filtering fundamentally changed how I approach rendering architecture and performance budgeting.

9.3 Methodological Growth

Finally, this project deeply refined my approach to performance analysis. At the onset, I relied on coarse frame-rate (FPS) counters, which proved entirely insufficient for isolating specific algorithmic costs. Implementing a robust GPU timestamping system allowed me to dissect the frame at a microsecond level and see the true, unhidden cost of individual compute passes. This granularity was the backbone of my results chapter; without it, I would not have been able to definitively prove that increasing ray sample counts yields diminishing returns compared to the fixed overhead of the denoiser.

9.4 Final Conclusion

While the final implementation successfully validates the efficiency of G-buffer-guided ray tracing and the Post-Lighting SVGF architecture, the journey highlighted the immense engineering overhead required to ship hybrid features. This graduation project successfully bridged the gap between academic ray-tracing theory and the harsh, low-level engineering reality of Vulkan, providing a formidable technical foundation for my future work in rendering and engine programming.

Bibliography

- Akenine-Möller, T., Haines, E., and Hoffman, N. (2018). *Real-Time Rendering*. CRC Press / Taylor & Francis, 4 edition. 4th edition.
- Anagnostou, K. (2021). Experiments in hybrid raytraced shadows. Interplay of Light. <https://interplayoflight.wordpress.com/2021/05/15/experiments-in-hybrid-raytraced-shadows/>.
- Castorina, M. and Sassone, G. (2023). *Mastering Graphics Programming with Vulkan: Develop a Modern Rendering Engine from First Principles to State-Of-the-art Techniques*. Packt Publishing.
- de Vries, J. (n.d.). *Learn OpenGL: Graphics Programming*. Online book (PDF), accessed via LearnOpenGL.
- Fernando, R., editor (2004). *GPU Gems: Programming Techniques, Tips and Tricks for Real-Time Graphics*. Addison-Wesley Professional. <https://developer.nvidia.com/gpugems/gpugems/contributors>.
- Galvan, A. (2025). Ray tracing acceleration structures. Alain.xyz. <https://alain.xyz/blog/ray-tracing-acceleration-structures>.
- Haines, E. and Akenine-Möller, T., editors (2019). *Ray Tracing Gems: High-Quality and Real-Time Rendering with DXR and Other APIs*. Apress.
- Hamza, A. A. G. (2024). Realistic shadows in computer graphics. Online Publication.
- Khronos Group (n.d.). Vulkan guide: Ray tracing extensions (vk_khr_ray_tracing_pipeline, etc.). Online documentation.

- Ki, R. (2023). Real-time ray tracing reflections and shadows implementation using directx raytracing. *Journal of Theoretical and Applied Information Technology*, 101(10). <https://www.jatit.org/volumes/Vol101No10/9Vol101No10.pdf>.
- Lauterbach, C., Mo, Q., and Manocha, D. (2009). Fast hard and soft shadow generation on complex models using selective ray tracing. In *Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games*.
- Luna, F. (2016). *Introduction to 3D Game Programming with DirectX 12*. Mercury Learning & Information. Online PDF accessed via deadnet.se.
- Marrs, A., Shirley, P., and Wald, I., editors (2021). *Ray Tracing Gems II: Next Generation Real-Time Rendering with DXR, Vulkan, and OptiX*. Apress.
- Marschner, S. M. and Shirley, P. S. (2016). *Fundamentals of Computer Graphics*. CRC Press, 4th edition. <https://repo.darmajaya.ac.id/5422/1/Fundamentals%20of%20Computer%20Graphics.pdf>.
- NVIDIA Developer Blog (2018). Introduction to nvidia rtx and directx ray tracing. Online article.
- NVIDIA-RTX (n.d.). Nrd: Nvidia real-time denoising library. <https://github.com/NVIDIA-RTX/NRD>.
- Olejniak, M. and Kozlowski, P. (2020). Raytraced shadows in call of duty: Modern warfare. Digital Dragons. https://www.activision.com/cdn/research/Raytraced_Shadows_in_Call_of_Duty_Modern_Warfare.pdf.
- Pharr, M., Jakob, W., and Humphreys, G. (2023). *Physically Based Rendering: From Theory to Implementation*. The MIT Press, 4th edition. <https://pbr-book.org/>.
- Qu, C. (2023). Research and analysis of ray tracing methods. *Highlights in Science, Engineering and Technology*, 8.
- Schied, C., Kaplanyan, A., Wyman, C., Patney, A., Chaitanya, C. R. A., Burgess, J., Liu, S., and Dachsbacher, C. (2017). Spatiotemporal variance-guided filtering: Real-time reconstruction for path-traced global illumination. In *Proceedings of High Performance Graphics (HPG)*.
- Shirley, P. (2024). *Ray Tracing in One Weekend*. <https://raytracing.github.io/books/RayTracingInOneWeekend.html>.

- Singh, P. (2020). Nvidia real-time denoiser delivers best-in-class denoising in ubisoft's watch dogs legion. Online article.
- Waldner, F. (2021). Real-time ray traced ambient occlusion and animation: Image quality and performance of hardware. Master's thesis, KTH Royal Institute of Technology. <https://www.diva-portal.org/smash/get/diva2:1574351/FULLTEXT01.pdf>.
- Wester, A. (2020). Ray-tracing soft shadows in real-time. Medium. <https://medium.com/@alexander.wester/ray-tracing-soft-shadows-in-real-time-a53b836d123b>.
- Xie, F., Tabellion, E., and Pearce, A. (2007). Soft shadows by ray tracing multilayer transparent shadow maps. In *Eurographics Symposium on Rendering (EGSR)*.
- Zhang, R., Isola, P., Efros, A. A., Shechtman, E., and Wang, O. (2018). The unreasonable effectiveness of deep features as a perceptual metric. In *CVPR*.