Incremental Performance and Quality Analysis of Hybrid Ray Tracing: Vulkan vs DXR in Real-Time Rendering

De Meyer Luca
Howest-Digital Arts and Entertainment
`luca.de.meyer@student.howest.be`

December 27, 2025

Abstract

Real-time rendering is transitioning from pure rasterization to hybrid ray tracing, yet developers lack quantitative guidance on which ray-traced features to adopt first. While shadows, reflections, and global illumination each offer visual improvements, their performance costs and perceptual quality gains remain poorly characterized in production contexts. This study measures the incremental costs and visual quality of ray-traced shadows and reflections in Vulkan Ray Tracing and DirectX Raytracing (DXR), identifying configurations that achieve 60 FPS on mid-range hardware. We evaluate multiple hybrid strategies including G-buffer-guided tracing, adaptive sampling, and screen-space fallbacks across three representative scenes at 1080p and 1440p resolutions. Using perceptual metrics (LPIPS, SSIM) and GPU profiling, we quantify the quality-per-millisecond ratio for each approach. Our results provide practical recommendations for incremental ray tracing adoption in real-time applications, demonstrating that [placeholder for key finding] achieves the optimal balance between visual fidelity and performance.

1 Introduction

The advent of hardware-accelerated ray tracing in consumer GPUs (NVIDIA RTX 20-series and AMD RDNA 2) has transformed real-time rendering from a purely rasterization-based paradigm to a hybrid approach combining traditional techniques with ray-traced effects. While full path-traced rendering remains computationally prohibitive for interactive framerates, selective use of ray tracing for shadows, reflections, and ambient occlusion offers a practical middle ground between rasterization artifacts and photorealistic quality.

However, adopting ray tracing incrementally presents a complex optimization problem. Each ray-traced feature incurs a measurable performance cost, yet the perceptual quality improvement varies significantly based on scene characteristics, sampling strategies, and denoising techniques. Developers transitioning to hybrid rendering face critical questions: Which features should be ray-traced first? What sampling rates are perceptually sufficient? How do different hybrid strategies compare in quality-per-millisecond efficiency?

Existing benchmarks and research typically evaluate either full ray tracing pipelines or isolated techniques, leaving a gap in practical guidance for incremental adoption. Furthermore, the performance parity between Vulkan Ray Tracing and DirectX Raytracing (DXR) remains under-documented, despite both APIs targeting the same underlying hardware acceleration structures.

This study addresses these gaps through systematic experimentation across three representative scenes, measuring the individual and combined costs of ray-traced shadows and reflections. We quantify visual quality using perceptual metrics (LPIPS, SSIM, PSNR) against path-traced ground truth and derive a quality-per-millisecond metric to rank hybrid strategies. Our experiments span multiple resolutions, sampling rates, and optimization techniques on current RTX hardware.

1.1 Contributions

Our work makes the following 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, and PSNR metrics
- Direct performance comparison of Vulkan Ray Tracing and DXR on identical hardware and workloads
- Quality-per-millisecond ranking of hybrid rendering strategies (G-buffer-guided, adaptive sampling, screen-space fallbacks)
- Practical configuration recommendations for achieving 60 FPS targets on mid-range RTX hardware

2 Background and Related Work

2.1 Rasterization Fundamentals

Rasterization is the dominant rendering technique in real-time computer graphics, used in almost all 3D interactive applications. Fundamental principle involes converting 3D geometry into a 2D grid pixels. By using the GPU, which is highly optimized to rasterize millions of triangle, rasterization is fast and efficient which is why its the backbone of real-time rendering. In simple terms, it takes mathematically described shapes and turns the into colored pixels we can see.

2.1.1 The Graphics Rendering Pipeline

The graphics rendering pipeline or "the pipeline" is the core component of any rendering technique. Its main function is to "render" a 2D image given a virtual camera, 3D objects, light sources and more. (real time rendering 4th edition) A pipeline consists of several stages which all perform parts of a larger task. In the next section we will briefly go over each stage.

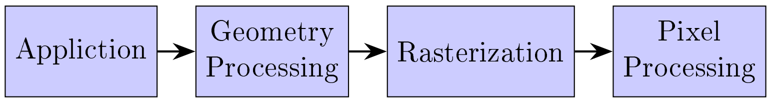

Figure 1: Graphics Rendering Pipeline

2.1.2 The Application Stage

The application stage tkeas place on the CPU meaning we as the developers have full control meaning we can modify and optimize as much as our hearts content, however some application work can be on the GPU in the form of compute shaders. These treat the GPU as a highly parallel general processor ignoring its specialization meant for graphics. At the end of the application stage the geometry to be rendered is sent to the geometry processing stage.

2.1.3 Geometry Processing stage

The geometry stage handles most per-triangle and per-vertex operations, it is further devided into vertex shading, projection, clipping and screen mapping stages.

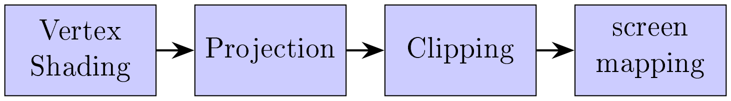

Figure 2: Geometry processing

2.1.4 Vertex Shading

To keep things short and simple, vertex shading has 2 tasks. Compute vertex positions and evaluate what we as programmer want out vertex output data to contain for example normals, UV-coordinates. Traditionally the shade of an object was computed by applying lights to vertex locations and normal and only storing the resulting color, this is why this stage is called vertex shading however, with the modren GPU and a large part of shading taking place per pixel, the vertex stage is more general and may not even evaluate any shading.

2.1.5 Rasterization

Rasterization also known as scan conversion, is the process of converting our 2D-vertices in screen space with a z-value and shading information into pixels on the screen. we find all pixels inside the primitives and go over our next 2 stages. primitive assembly also known as triangle setup and trianlge traversal, given the names not only triangles can be procesed, these stages can also handle lines and points but since a trianlge is the most common form they have triangle in their name.

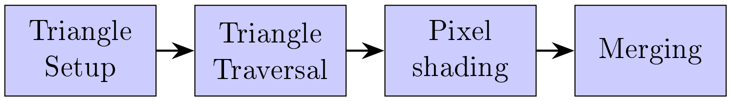

Figure 3: Rasterization stage

2.2 Ray Tracing Fundamentals

Ray tracing simulates light transport by casting rays from the camera through each pixel and testing for intersections with scene geometry. Modern hardware acceleration structures (Bounding Volume Hierarchies) enable real-time traversal, though full path tracing remains expensive. Hybrid rendering selectively applies ray tracing to effects where rasterization produces visible artifacts: hard shadows from small light sources, accurate reflections on curved surfaces, and inter-object occlusion.

2.3 Vulkan Ray Tracing vs DirectX Raytracing

Both Vulkan Ray Tracing (VK_KHR_ray_tracing) and DirectX Raytracing (DXR) expose hardware ray tracing through similar abstractions: acceleration structures, shader binding tables, and ray generation/intersection/miss shaders. Vulkan offers explicit control over memory and synchronization, while DXR provides higher-level abstractions. Both compile to identical GPU instructions on NVIDIA and AMD hardware, leading to expectations of performance parity with minor driver overhead differences.

2.4 Hybrid Rendering Strategies

Several hybrid approaches have emerged in production rendering:

Full-screen ray tracing dispatches rays for every pixel but uses low sample counts (1-2 SPP) with aggressive denoising. This is the most straightforward approach but potentially wasteful on pixels where screen-space techniques suffice.

G-buffer-guided tracing uses rasterized geometry buffers to selectively dispatch rays only where needed (e.g., on glossy surfaces for reflections or in shadow penumbrae). This reduces ray count but requires additional logic to determine ray spawn conditions.

Screen-space first, ray-traced fallback attempts cheaper screen-space techniques (SSR, SSAO) and only invokes ray tracing when screen-space fails (off-screen reflections, occluded regions). This minimizes ray tracing overhead but introduces complexity in blending techniques.

Adaptive sampling varies ray count per pixel based on material roughness, motion vectors, or edge detection. Smooth regions receive fewer samples while edges and specular highlights receive more, optimizing the sample budget.

Prior work has evaluated these strategies in isolation, but comprehensive quality-per-cost comparisons across multiple scenes and features remain limited.

2.5 Perceptual Quality Metrics

Traditional image quality metrics like PSNR and SSIM capture structural differences but correlate weakly with human perception for rendered images. LPIPS (Learned Perceptual Image Patch Similarity) uses deep neural networks trained on human perceptual judgments, providing better correlation with subjective quality assessment. We employ all three metrics for comprehensive evaluation, with LPIPS as the primary perceptual measure.

3 Research Questions and Hypotheses

We investigate the following research questions:

1. What are the individual and combined performance costs of ray-traced shadows and reflections?
2. How do these costs scale with resolution and scene complexity?
3. Which hybrid strategies offer the best quality-per-millisecond ratio?
4. Do Vulkan and DXR achieve performance parity on identical workloads?

Based on preliminary profiling and prior literature, we formulate the following hypotheses:

3.1 Performance Costs (H1)

H1a: Ray-traced shadows cost less than reflections, with global illumination being most expensive. We expect shadows to add approximately 2ms, reflections 4-6ms, and combined features 7-9ms at 1080p on RTX 3060.

H1b: Rendering costs scale super-linearly with resolution due to increased ray coherence loss. We predict 1080p to 1440p incurs 1.8× cost, and 1440p to 4K incurs 2.2× cost.

H1c: Vulkan and DXR perform within ±5-10% on identical hardware, as both compile to the same GPU ray tracing instructions with minor driver overhead differences.

3.2 Quality Optimization (H2)

H2a: One sample per pixel (1 SPP) with temporal denoising achieves superior visual quality per millisecond compared to 4 SPP without denoising, due to temporal accumulation effectively raising the sample count.

H2b: Adaptive sampling based on edge detection reduces cost by 40-60% without perceptible quality loss, as most pixels require minimal sampling.

H2c: G-buffer-guided tracing provides the best quality-cost ratio, reducing cost by approximately 50% with less than 10% quality degradation.

3.3 Hybrid Strategy Effectiveness (H3)

We predict the following ranking by quality-per-millisecond:

1. G-buffer guided + 1 SPP + temporal denoising
2. Screen-space first with ray-traced fallback
3. Adaptive sampling (motion/roughness based)
4. Full-screen ray tracing with low samples
5. Full-screen ray tracing with high samples

3.4 Scene Complexity Scaling (H4)

H4a: Shadow costs scale linearly O(n) with light count. We expect 1 light: 2ms, 4 lights: 8ms.

H4b: Reflection costs scale logarithmically O(log n) with BVH depth. Doubling triangle count should increase cost by approximately 30%, not 100%.

H4c: Material complexity affects shading more than ray tracing overhead. Complex PBR shading should add approximately 2ms regardless of ray tracing usage.

4 Methodology

4.1 Experimental Design

Our experiment measures incremental performance costs and visual quality improvements when transitioning from pure rasterization to hybrid ray tracing. We employ a controlled experimental design with systematic variation of rendering features, sampling strategies, and scene characteristics.

4.2 Test Scenes

Three representative scenes are selected to cover different performance and quality scenarios:

Indoor Scene: Medium geometric complexity (1-2M triangles) featuring several glossy materials and 2-3 dynamic light sources. This scene tests reflection accuracy on varied surface roughness and multi-light shadow performance.

Outdoor Scene: High geometric complexity (10M triangles) with vegetation, small occluders, and one dominant directional light. This scene stresses BVH traversal and tests shadow quality with complex occluder geometry.

Table 1: Experimental Variables and Test Values

Category	**Variable**	**Values Tested**
Resolution	Internal rendering resolution	1080p, 1440p
Ray-Traced Features	Enabled feature set	None, Shadows only, Reflections only, Both
Samples Per Pixel	Ray sample count	1, 2, 4, 8
Denoiser	Temporal/spatial filtering	None, Simple temporal, SVGF-like, NRD
Hybrid Strategy	Ray dispatch method	Full-screen, G-buffer guided, SSR fallback, Adaptive
Trace Resolution	RT buffer resolution	Full, Half, Quarter
Light Count	Active light sources	1, 4, 8
Scene Complexity	BVH size / geometry density	Base, $2\times$ triangles, $4\times$ triangles

Specular Scene: Low triangle count but multiple mirror and water-like surfaces. This scene isolates reflection quality and tests handling of highly reflective materials where screen-space techniques fail.

Each scene includes a scripted camera path ensuring reproducible viewpoints across all test runs. Ground-truth reference images are generated using 1024 SPP path tracing for perceptual quality comparison.

4.3 Independent Variables

Table 1 summarizes the experimental variables tested in our study.

4.4 Hardware and Software Configuration

Experiments are conducted on three representative hardware configurations:

- **Mid-range laptop:** RTX 3060 Mobile (6GB), Ryzen 5800H, 16GB RAM
- **High-end laptop:** RTX 4090 Mobile (16GB), i9-13980HX, 32GB RAM
- **Desktop:** AMD Radeon RX 9070 XT (16GB), Ryzen 7950X, 32GB RAM

Software environment: Windows 11, Vulkan SDK 1.3.x, DXR via DirectX 12. All drivers updated to latest stable versions. Power settings configured for maximum performance with adequate cooling to prevent thermal throttling.

4.5 Instrumentation and Data Collection

GPU timings are collected using timestamp queries inserted at key pipeline stages:

1. Rasterization pass (G-buffer generation)
2. Shadow ray tracing dispatch
3. Reflection ray tracing dispatch
4. Denoising pass
5. Post-processing and final composite

Each configuration executes for 120 consecutive frames following a 60-frame warm-up period to ensure thermal and workload stability. The middle 60 frames are analyzed to exclude transients. Each test repeats three times to measure variance and compute confidence intervals.

Metrics recorded per frame include:

- Total frame time (ms)
- Per-stage GPU time (ms)
- GPU temperature and power draw (via nvidia-smi / AMD metrics)
- Ray count and average traversal depth (when available via profiler APIs)

Image buffers (final color, G-buffer components, motion vectors, depth) are captured for all test configurations to enable offline quality analysis.

4.6 Image Quality Evaluation

Perceptual quality is evaluated against path-traced ground truth using three complementary metrics:

LPIPS (Learned Perceptual Image Patch Similarity): Primary metric for perceptual quality, computed using the pretrained AlexNet-based model from Zhang et al. Lower values indicate higher perceptual similarity.

SSIM (Structural Similarity Index): Measures structural information preservation. Computed over 11×11 windows with standard parameters.

PSNR (Peak Signal-to-Noise Ratio): Traditional pixel-wise difference metric, included for completeness and comparison with prior work.

Temporal stability is measured by computing frame-to-frame LPIPS differences along the camera path, detecting flickering and ghosting artifacts.

Additionally, A/B visual comparison tests are conducted where observers rate perceived quality differences between configurations on a 5-point scale, validating that metric-based conclusions align with subjective perception.

4.7 Quality-Per-Millisecond Metric

To directly compare configurations with different performance and quality characteristics, we derive a quality-per-millisecond (QPM) metric:

$$\text{QPM} = \frac{\text{Normalized Quality Gain}}{\text{RT Overhead (ms)}} \tag{1}$$

where Normalized Quality Gain is defined as:

$$\text{NQG} = 1 - \frac{\text{LPIPS}_{\text{config}}}{\text{LPIPS}_{\text{raster}}} \tag{2}$$

This metric quantifies the perceptual improvement per millisecond of ray tracing cost, enabling direct comparison across diverse configurations. Higher QPM indicates more efficient quality gains.

4.8 Data Analysis

For each configuration, we compute mean, median, and 90th percentile (P90) frame times per pipeline stage. Data is aggregated into CSV logs for batch processing and statistical analysis.

The following visualizations are generated:

- Stacked bar charts showing frame time breakdown by rendering stage
- Line plots of RT cost vs. resolution scaling
- Scatter plots of RT cost vs. triangle count (scene complexity)
- Quality-cost scatter plots (LPIPS/SSIM vs. ms) with Pareto frontier highlighting
- Heatmaps showing quality-per-millisecond across sampling rates and hybrid strategies

Vulkan and DXR implementations are compared directly using identical scene data, shaders (translated via SPIR-V Cross), and timing methodology to isolate API-level performance differences.

Statistical significance of performance differences is assessed using paired t-tests with Bonferroni correction for multiple comparisons. Effect sizes are reported using Cohen's d to distinguish practically significant differences from merely statistically significant ones.

5 Expected Results

Based on preliminary profiling and hypothesis formulation, we anticipate the following outcomes:

Performance Costs: Ray-traced shadows will add approximately 2ms per light at 1080p, while reflections will cost 4-6ms depending on material roughness distribution. Combined, we expect 7-9ms overhead, slightly less than the sum due to shared BVH traversal setup costs.

Resolution Scaling: The transition from 1080p to 1440p is expected to increase ray tracing costs by approximately $1.8\times$ due to the $1.77\times$ increase in pixel count plus reduced ray coherence. The 1440p to 4K transition will incur a steeper $2.2\times$ cost increase.

Quality Optimization: We expect 1 SPP with temporal denoising to achieve LPIPS scores within 10% of 4 SPP raw while costing 75% less, validating the effectiveness of temporal accumulation. Adaptive sampling should reduce costs by 40-60% with minimal ($< 5\%$) quality degradation.

API Parity: Vulkan and DXR are expected to perform within $\pm$5-10% on identical workloads, with Vulkan potentially showing slight advantages in CPU-limited scenarios due to lower driver overhead and explicit synchronization control.

Hybrid Strategy Ranking: G-buffer-guided tracing with 1 SPP and temporal denoising is predicted to achieve the highest QPM score, offering approximately 50% cost reduction compared to full-screen tracing with less than 10% quality loss.

6 Discussion

7 Limitations

Our study focuses on shadows and reflections, excluding global illumination and ambient occlusion which have different performance characteristics. The three test scenes, while representative, cannot capture all possible scene types and characteristics. Hardware testing is limited to NVIDIA RTX and AMD RDNA architectures; results may not generalize to future GPU generations or alternative vendors.

The perceptual quality metrics, while validated in prior work, represent a specific model of human perception. Individual subjective preferences may vary, particularly for temporal artifacts where LPIPS may not fully capture discomfort from flickering or ghosting.

8 Conclusion

9 Future Work

Future research directions include extending this analysis to global illumination and ambient occlusion, evaluating next-generation denoising techniques (including ML-based approaches), and investigating dynamic quality scaling systems that adjust ray tracing configurations based on GPU load and frame time targets.

References