한국컴퓨터그래픽스학회
Korea Computer Graphics Society Vol. 31, No. 3, P. 171~181

모바일 플랫폼에서의 Vulkan 기반 광선 추적 렌더러의 성능 프로파일링

유택근[1] 윤성호[1] 조세희[1○] 서 웅[2] 임인성[1*]

서강대학교 컴퓨터공학과[1], 삼성전자[2]

{xorrms9025, sh3931, sehee138, ihm*}@sogang.ac.kr[1], woong.seo@samsung.com[2]

Performance Profiling of Vulkan-based Ray Tracing Renderers on Mobile Platforms

Taekgeun You[1] Sungho Yun[1] Sehee Cho[1○] Woong Seo[2] Insung Ihm[1*]

Sogang University[1], Samsung Electronics[2]

요 약

고성능 PC GPU의 등장으로 광선 추적 (ray tracing) 기법은 3D 게임을 비롯한 실시간 렌더링 분야에서 활발히 활용되고 있다. 하지만 모바일 기기에서는 전력 소모, 발열, 제한적인 자원 등으로 인해 실시간 광선 추적 기법을 적용하는 데에 어려움이 있다. 이에 따라 본 논문에서는 모바일 환경에서의 실시간 광선 추적에 최적화된 Vulkan 기반 렌더러를 개발하고 성능을 분석하였다. 렌더러는 렌더링 방식에 따라 전체 광선 추적 (full ray tracing) 렌더러와 래스터화에 기반한 하이브리드 광선 추적 (hybrid ray tracing) 렌더러로 구분하였다. 최적의 구현을 위해 각 렌더러에는 광선 추적 파이프라인 (ray tracing pipeline)과 광선 질의 확장 (ray query extension) 방식을 각각 적용하였으며, 이에 대한 광선 추적 성능을 비교하였다. 또한, 다수의 광원을 사용한 장면에 대해 그림자 광선 투사 및 조명에 따른 쉐이딩 비용을 줄이기 위해 광원의 영향 범위에 따라 쉐이딩을 제한하는 최적화 기법을 활용했다. 이 경우, 빛의 영향 범위를 최대한 작게 유지하면서도 빛의 세기를 적절히 유지하는 것이 중요하므로 이에 적합한 새로운 감쇠 함수를 제안하며, 이를 적용했을 때에 대한 성능을 평가하였다.

Abstract

The emergence of high-performance desktop GPU has enabled ray tracing techniques to be widely used in real-time rendering, including 3D gaming applications. However, applying real-time ray tracing on moblie devices remains challenging due to power consumption, thermal constraints, and limited resources. In this paper, we developed a Vulkan-based optimized renderer for real-time ray tracing on mobile platforms and analyzed its performance. The renderers are categorized into two types: a full ray tracing renderer and a rasterization-based hybrid ray tracing renderer. To achieve optimal implementation, we applied the ray tracing pipeline and ray query extension to each renderer and evaluated their performance. Furthermore, to reduce the shading cost caused by shadow rays in scenes with multiple lights, we adopted an optimization technique that restricts the shading based on the extent of each light's influence. In this context, maintaining the light's influence as small as possible while preserving appropriate intensity is critical. To this end, we propose a new attenuation function and assess its performance when applied.

키워드: 실시간 광선 추적, 모바일 플랫폼, Vulkan 광선 추적 파이프라인, 광선 질의, 적응적 빛의 감쇠, 성능 프로파일링
Keywords: Real-time ray tracing, mobile platform, Vulkan ray tracing pipeline, ray query, adaptive light attenuation, performance profiling

*corresponding author: Insung Ihm / Sogang University (ihm@sogang.ac.kr)

Received : 2025.06.13./ Review completed : 1st 2025.06.30. 2nd 2025.07.10./ Accepted : 2025.07.14.
DOI : 10.15701/kcgs.2025.31.3.171
ISSN : 1975-7883(Print)/2383-529X(Online)

1. 서론

광선 추적 기법 (ray tracing method)은 카메라로 향하는 광선의 경로를 역추적함으로써 빛의 경로를 모방하는 렌더링 기법으로, 래스터화 방식과 함께 컴퓨터 그래픽스 분야의 양대 축을 이루는 핵심 방법이다. 그림자, 반사, 굴절, 조명 효과 등을 정밀하게 구현하여 고품질의 영상을 생성할 수 있어 다양한 3D 그래픽 분야에서 각광받았지만, 광선-물체 교차 계산 비용 문제로 인해 실시간 렌더링 응용에는 한계가 있었다. 그러나, 수 년 전부터 광선-물체 교차 및 광선 추적 파이프라인을 가속하는 GPU 모듈이 등장함에 따라 PC 환경에서는 경로 추적 (path tracing) 정도의 간접 조명과 같은 상당한 수준의 광선 추적 효과도 실시간으로 생성할 수 있게 되었고, 대규모 고사양 게임에서는 광선 추적 효과가 필수적인 요소로 자리 잡았다.

한편, 게임 산업이 PC에서 모바일 중심으로 전환된 것에 이어, 확장현실 (XR) 콘텐츠의 수요가 증가함에 따라 최근 모바일 GPU는 Vulkan API를 기반으로 하는 광선 추적 플랫폼을 제공하기 시작하였다. 이를 이용하여 모바일 플랫폼에서 광선 추적 기술을 적용하는 방법은 크게 두 가지가 있다. 하나는 광선 추적 파이프라인 확장 (VK_KHR_ray_tracing_pipeline)을 사용하는 것으로, NVIDIA의 OptiX API와 같이 ray-gen, intersection, any-hit, closest-hit, miss와 callable 쉐이더 등을 통하여 광선 추적을 구현할 수 있다. 반면, 광선 추적 파이프라인을 지원하지 않는 모바일 GPU에서는 그래픽스 파이프라인 (graphics pipeline)이나 컴퓨트 파이프라인 (compute pipeline)을 기반으로 광선 질의 (VK_KHR_ray_query) 확장을 통해 광선 추적 효과를 생성할 수 있다. 비록 전자처럼 다양한 종류의 쉐이더를 통한 유연한 광선 추적 프로그래밍은 어렵지만 그래픽스 및 컴퓨트 파이프라인의 기본 쉐이더를 통하여 기본적인 광선 추적 효과를 생성할 수 있다.

모바일 플랫폼에서의 광선 추적 기술 적용은 상업적으로 중요한 키워드 중의 하나이지만, 아직 모바일 GPU의 한계로 인하여 어려움이 존재한다. GPU의 광선 추적 모듈의 주 기능은 광선-물체 교차 계산을 하드웨어적으로 가속하는 것으로, PC 플랫폼에서는 GPU 광선 추적 모듈이 제공하는 공간 가속 구조 (Bounding Volume Hierarchy, BVH) 탐색 및 교차 계산 모듈의 도움으로 렌더링 과정에서 광선-물체 교차 계산의 부담을 상당히 감소시킴으로써 계산 여력을 물리 기반 렌더링 (Physically Based Rendering, PBR)과 같은 고도의 쉐이딩 효과 생성에 활용할 수 있다. 반면, 모바일 GPU 상에서는 아직도 광선-물체 교차 계산의 비중이 상당히 큰 편이고, 이는 고속/고화질 광선 추적 효과 생성에 큰 제약으로 작용하고 있다. 이에 따라 모바일 GPU 상에서는 적응적으로 최대한 적은 수의 광선만 처리하는 한편, 알고리즘 기반의 소프트웨어적인 방법의 적용을 통하여 광선-물체 교차 계산을 최적화하는 것이 요구되고 있다.

이에 본 논문에서는 모바일 환경에서 효과적인 실시간 광선 추적 방법을 탐색하고자 Vulkan 기반의 다양한 렌더러를 설계 및 구현하였으며 각 렌더러에 대한 성능 평가를 수행했다. 모바일 환경에서 광선 추적 비용과 파이프라인 복잡도 간 상충관계 (trade-off)를 비교하기 위해 모든 픽셀에 광선을 투사해 렌더링을 수행하는 전체 광선 추적 렌더러와 래스터화에 기반해 선택적으로 광선을 추적하는 하이브리드 렌더러를 구현했다. 모든 렌더러에는 최근 컴퓨터 그래픽스 분야에서 표준으로 자리 잡은 물리 기반 렌더링 기법을 적용했으며, 다양한 텍스춰를 활용해 보다 현실감 있는 조명 효과를 구현했다. 또한, Vulkan API에서 광선 추적을 위해 제공하는 두 가지 확장을 각각 적용하고 성능을 평가함으로써 모바일 환경에 더 적합한 구현 방식을 탐색했다. 마지막으로, 다수의 광원에 대한 그림자 광선 및 조명에 따른 쉐이딩 연산을 선택적으로 처리하기 위해 빛의 영향 범위를 효율적으로 유지할 수 있도록 하는 새로운 감쇠 함수를 제안한다.

2. 관련 연구

2.1 Whitted 스타일 광선 추적

Whitted 스타일 광선 추적 (Whitted-style ray tracing)은 Turner Whitted에 의해 제안된 고전적인 광선 추적 방식으로, 광학적 정확성을 갖춘 반사, 굴절, 그림자 효과를 표현할 수 있는 최초의 컴퓨터 그래픽스 렌더링 모델 중 하나이다 [1]. 이 방식은 단순히 주 광선 (primary ray)만을 처리하는 래스터화 기반 접근법과는 달리, 표면과의 교차 이후 재귀적으로 추가적인 이차 광선 (secondary ray)을 생성하는 방식으로 조명 및 시각 효과를 모사한다.

Whitted 스타일 접근법은 금속, 유리 등 반사/굴절 특성을 가지는 물체의 사실적인 묘사에 강점을 가진다. 하지만 재귀적 계산 구조로 인해 연산량이 기하급수적으로 증가하며, 이에 따라 실시간 렌더링 환경에서는 적용이 제한적이었다. 최근 GPU 기반의 가속 기법 및 OptiX, Vulkan, DirectX 광선 추적과 같은 API의 등장으로, Whitted 스타일의 광선 추적 모델을 실시간 환경에서도 활용할 수 있게 되었고, 특히 하이브리드 렌더러나 반사/굴절 효과를 구현하는 렌더러에서 주요 기술로 다시 주목받고 있다.

2.2 GPU 기반 광선 추적 파이프라인

고성능 실시간 광선 추적의 발전은 하드웨어 및 소프트웨어의 통합을 통해 이루어지고 있으며, RTX GPU의 광선 추적 모듈을 활용한 하드웨어 가속 기반의 파이프라인을 제공한다 [2, 3]. 이 파이프라인의 핵심은 공간 가속 구조 기반의 장면 탐색과 기하 교차 연산을 GPU에서 전용 유닛으로 가속하는 구조에 있으며, 이는 광선과의 교차 연산에 소요되는 비용을 획기적으로 줄여준다. 이를 통해 복잡한 장면에서도 실시간 수준의 성능을 확보할 수 있으며, 다양한 광선과 쉐이딩을 병렬적으로 처리할 수 있다.

광선 추적 파이프라인은 전통적인 그래픽스 파이프라인과 달리, ray-gen, intersection, any-hit, closest-hit, miss 등 여러 종류

의 쉐이더를 통해 광선 추적을 세분화하여 처리할 수 있는 구조를 갖는다. 이러한 쉐이더 프로그램들은 쉐이더 바인딩 테이블 (Shader Binding Table, SBT)을 통해 개별 기하 객체와 동적으로 연결되며, 각 쉐이더에 필요한 정보 및 함수 포인터들을 전달한다. 결과적으로 광선 추적 파이프라인은 사용자 정의 연산의 유연성과 하드웨어 가속의 효율성을 동시에 제공하며, 복잡한 재질 모델, 반사·굴절, 전역 조명 등의 고급 광선 추적 기반 렌더링 기법에 적합한 실행 환경을 제공한다.

이러한 광선 추적 표준 파이프라인 (Figure 1 참조)은 고성능 GPU에 최적화되어 있어 고정된 흐름에서 복잡한 효과를 효율적으로 표현할 수 있으나, 모바일 환경에서는 파이프라인 자체의 오버헤드로 인해 성능 저하의 원인이 될 수 있다. 특히 any-hit 쉐이더를 사용하여 나뭇잎과 같은 정교한 물체를 표현할 때 연산량이 증가하여 최근 NVIDIA에서는 불투명도 마이크로맵 (opacity micromap)과 같은 새로운 대안을 제시하고 있는 실정이다 [4].

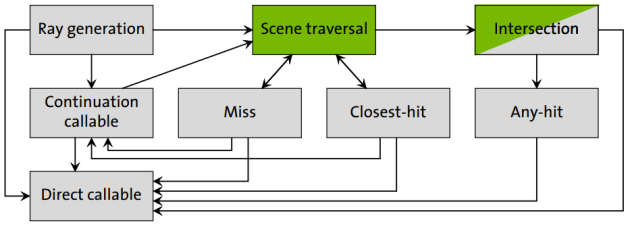


Figure 1: Ray tracing pipeline [3]

광선 질의는 Vulkan API에 VK_KHR_ray_query 확장으로 정의되어 있으며, 기존의 래스터화 기반 쉐이더 프로그램에서 광선 교차 검사를 수행할 수 있도록 지원한다. 이는 GPU 내부의 기존 파이프라인에 광선 추적 효과를 삽입함으로써, 반사나 굴절, 그림자와 같은 효과를 통합할 수 있다는 장점이 있다 [5]. 광선 질의는 쉐이더 내에서의 유연한 제어 흐름, 낮은 오버헤드와 파이프라인 전환 비용 등의 이유로 성능 이점을 제공하여 실시간 렌더링 환경에서 실용적인 대안으로 주목받고 있다.

2.3 지연 렌더링 (Deferred rendering) 기반 광선 추적

실시간 광선 추적 기술의 발전과 함께, 기존의 래스터화 기반 지연 렌더링 구조와의 결합이 활발히 연구되고 있다. 특히 기하 버퍼 (G-buffer)를 활용하여 정보를 저장하고, 이후 단계에서 반사 및 그림자 계산을 수행하는 방식은 복잡한 조명 처리를 가능하게 한다. 이는 광선 추적 파이프라인 전체를 구성하지 않고도 광선 추적 효과를 달성할 수 있는 효율적인 대안으로 평가받고 있다. 이 방식에서는 주 광선 투사를 래스터화에 기반한 기하 단계 (geometry pass)로 대체하고, 조명 단계 (lighting pass)에서는 기하 버퍼의 정보를 바탕으로 필요한 경우 선택적으로 광선을 투사한다 [6]. 이러한 구조는 특히 모바일 환경에서 연산 자원의 부담을 줄이며 우수한 성능을 제공한다.

2.4 물리 기반 렌더링

물리 기반 렌더링은 현실 세계의 조명과 재질 간의 물리적 상호작용을 수학적 근거가 있는 모델을 통해 근사함으로써, 시각적으로 일관된 고품질 이미지를 생성하는 렌더링 접근 방식이다. 전통적인 가시광 기반의 가중치 모델과 다르게 물리 기반 렌더링은 에너지 보존, 마이크로패싯 (micro-facet) 이론, 프레넬 방정식 (Fresnel equation), 양방향 반사 분포 함수 (Bidirectional Reflectance Distribution Function, BRDF) 등의 물리 법칙을 기반으로 구현된다. 이에 따라 동일한 재질 파라미터가 다양한 광원 조건에서도 일관된 반응을 보장하며, 현실 세계 기반의 스캔 데이터와도 정합된다 [7]. 본 논문에서는 물리 기반 렌더링을 사용하여 보다 사실적인 장면을 표현한다.

2.5 빛 감쇠 (Light attenuation)

빛 감쇠는 거리 증가에 따른 조명 강도의 감소를 정량적으로 설명하는 물리적 개념이다. 대표적인 예로 OpenGL을 비롯한 다양한 렌더링 엔진에서는 $f(d) = \frac{1}{k_c + k_l d + k_q d^2}$ 형태의 다항식 기반 감쇠 함수를 활용한다 [8]. 여기서 k_c, k_l, k_q는 각각 상수항, 일차항, 이차항 감쇠 계수로, 광원의 유효 범위와 강도 분포를 조정하는 데 사용된다. 하지만 광원이 다수 존재하는 장면에서는 해당 함수의 급격한 감쇠로 인해 지나치게 어두운 영역을 생성하는 단점이 있다. 특히, 실내 장면이나 광원이 다수 존재하는 환경에서는 이러한 어둠의 확산으로 인해 조명의 시각적 균형이 무너지고, 결과적으로 사실적인 조명 표현에 어려움이 따른다. 따라서 본 논문은 해당 문제를 해결하기 위해 쉐이딩 기법과 감쇠 함수의 확장을 제시한다.

3. Vulkan 기반 모바일 광선 추적 렌더러의 설계 및 구현

3.1 개발 렌더러 요약

본 논문에서는 Table 1에서 보인 6가지 렌더러를 개발했다. 각 렌더러는 광선 추적 파이프라인을 사용한 방식과 광선 질의를 사용한 방식으로 나뉜다. 두 방식은 각각 전체 광선 추적 (RTP/RQ-FRT), 하이브리드 광선 추적 (RTP/RQ-HRT), 그림자 맵을 포함한 하이브리드 광선 추적 (RTP/RQ-HRT-SM)으로 다시 나뉜다.

전체 광선 추적 렌더러는 Whitted 스타일 광선 추적 방식의 주 광선과 이차 광선을 모두 투사하는 방식의 렌더러이다. 하이브리드 광선 추적의 경우 2.3절에서 서술한 바와 같이 두 단계로 구성된다. 먼저 기하 단계에서 래스터화를 통해 최전방 물체의 위치, 법선 벡터, 색상 등의 정보를 텍스춰 형태로 저장함으로써

Table 1: List of developed renderers

광선 추적 방식	구현 광선 추적 방식 [약어]
Ray tracing pipeline	Full ray tracing [RTP-FRT]
	Hybrid ray tracing [RTP-HRT]
	Hybrid ray tracing with shadow mapping [RTP-HRT-SM]
Ray query feature	Full ray tracing [RQ-FRT]
	Hybrid ray tracing [RQ-HRT]
	Hybrid ray tracing with shadow mapping [RQ-HRT-SM]

주 광선을 대체한다. 이후 조명 단계에서 저장해놓은 정보들로 쉐이딩을 수행하고 반사체 또는 굴절체의 경우 반사 광선 및 굴절 광선을 추가로 투사한다. 마지막으로 그림자 맵을 포함한 하이브리드 광선 추적의 경우 하이브리드 광선 추적 방식에서의 그림자 광선을 그림자 맵으로 대체한 방식이다. 이 방식은 그림자 맵을 생성하는 래스터화 단계, 최전방 물체들의 정보를 저장하는 래스터화 단계, 쉐이딩을 포함하여 반사체 및 굴절체에 대한 이차 광선을 투사하는 광선 추적 단계의 3단계로 구성된다.

3.2 전체 광선 추적과 하이브리드 광선 추적

전체 광선 추적 렌더러는 픽셀마다 주 광선과 이차 광선을 모두 투사함에 따라 투사되는 광선의 개수가 가장 많다. 이는 광선 추적 기법에서 연산량이 가장 많다고 알려진 광선-물체 교차 계산이 가장 많이 수행된다고 할 수 있다. 이때 광선-물체 교차 계산은 그래픽 하드웨어에 의해 공간 가속 구조를 구축함으로써 가속화가 이루어지지만, 모바일 환경에서는 여전히 성능 부담이 큰 상황이다. 따라서 모든 픽셀에 투사되는 주 광선을 래스터화 방식으로 대체한 하이브리드 광선 추적 방식을 통해 투사하는 광선 수를 상당히 감소시켰다. 또한 연산량이 많은 any-hit 쉐이더를 대신하여 최전방 물체 정보를 저장하는 래스터화 단계의 알파 텍스춰 처리를 수행함에 따라 성능이 보존되는 효과를 확인하였다.

그러나 모든 크기의 장면에 대해 성능 향상이 이루어지지는 않았다. 장면을 구성하는 삼각형의 개수가 많아질 수록 대체적으로 성능 향상의 정도가 감소했다. 이는 래스터화 방식과 광선 추적 방식의 구조적 차이 및 파이프라인 복잡도 간 상충관계에 의한 결과로 볼 수 있다. 래스터화 방식에서는 early-z 컬링 기법을 적용해도 종종 한 픽셀에 대하여 여러 번의 고비용 프래그먼트 쉐이딩 계산을 수행한다. 반면 광선 추적 방식에서는 쿼드 트리 형태의 공간 가속 구조를 탐색하며 광선-물체 교차 계산을 수행하고, 이를 통해 결정된 최전방 물체에 대해서만 쉐이딩 연산을 수행한다. 즉 래스터화 방식에서는 삼각형 개수에 정비례하게, 광선 추적 방식에서는 일반적으로 $\log N$에 비례하여 연산 비용이 증가한다. 이에 따라 삼각형의 개수가 늘어날수록 하이브리드 광선 추적 렌더러의 래스터화 파이프라인 연산 비중이 증가하여 성능 향상의 정도가 감소한 것으로 파악된다.

하이브리드 광선 추적 렌더러의 경우 래스터화 단계가 주 광선의 역할을 대체하기 때문에, 전체 광선 추적 렌더러와 같은 전통적인 광선 추적 기법과 달리 슈퍼샘플링 (supersampling)을 통한 안티-에일리어싱 (anti-aliasing)을 적용하기 어렵다는 한계가 있다. 그러나 본 연구의 모든 렌더러에서는 제한된 자원만을 활용해야 하는 모바일 환경에서의 한계로 인해 픽셀 당 1개의 광선을 투사하기 때문에, 전체 광선 추적 렌더러와 하이브리드 광선 추적 렌더러에서 품질 측면에서의 차이는 느낄 수 없었다.

3.3 3-Pass 하이브리드 광선 추적

투사하는 광선의 개수를 더욱 줄이기 위해 하이브리드 광선 추적 방식에서의 그림자 광선을 그림자 맵으로 대체했다. 그림자 맵은 광원별로 생성되는데, 방향 광원일 경우 광원 당 1장의 텍스춰, 점 광원일 경우 광원 당 6장의 텍스춰가 생성된다. 또한 그림자 맵에는 각 픽셀로 보이는 장면의 최전방 물체와 광원 간 거리를 저장하기 때문에, 카메라가 이동하여 최전방 물체가 바뀌거나 광원이 이동하는 경우 그림자 맵을 재생성해주어야 한다. 이러한 이유로 정적 방향 광원이 소수 있는 장면에서는 그림자 광선을 그림자 맵으로 대체함에 따라 15–77% 가량의 성능 향상이 있었지만, 동적 광원이 존재하는 광원 혹은 점 광원이 다수 있는 장면 등에서는 유의미한 성능 향상을 확인할 수 없었다.

품질 측면에서는 그림자의 경계 부분에 에일리어싱 (aliasing)이 관찰되어 눈에 띄게 품질이 떨어졌다. 이를 완화하기 위해서는 경계 부분에만 그림자 광선을 투사하고 나머지 부분은 그림자 맵을 활용함으로써, 그림자 광선의 개수를 줄여 성능을 보존하고 품질은 높이는 방법을 적용해볼 수 있다.

3.4 광선 질의 방식의 구현

Algorithm 1은 광선 질의 방식을 통해 광선 추적을 구현하기 위해 본 연구에서 설계한 알고리즘이다. 특히 알파 텍스춰를 처리하는 방식을 구현한 사례가 거의 없어 이를 중심으로 서술하였다. 기본적으로 VK_KHR_ray_query 확장에서 제공하는 함수를 사용하고, Algorithm 1에서는 각 함수의 이름을 축약하여 표현하였다.

광선 추적 파이프라인 방식에서의 모든 쉐이더는 광선 질의 방식에서 하나의 쉐이더로 처리된다. 광선 추적 파이프라인의 raygen 쉐이더에서 traceRayEXT 함수를 호출하는 것은 광선 질의 방식의 rayQueryProceedEXT 함수에 대응된다. 다만 광선 추적 파이프라인에서는 intersection, any-hit, closest-hit 쉐이더에서 나누어 처리하는 것을 광선 질의 방식에서는 후술할 함수들을 통해 필요한 정보를 확인하고 처리한다.

Algorithm 1의 line 9부터 살펴보면, rayQueryEXT 타입의 변

수 *rQuery*를 rayQueryInitializeEXT 함수를 통해 초기화할 수 있다. 초기화 단계에서 변수 *rQuery*에 탐색할 공간 가속 구조, 광선의 원점 및 방향 등 탐색에 필요한 정보들을 설정한다. rayQueryProceedEXT는 탐색을 한 단계 수행하는 함수로, 전체 가속구조에 대한 탐색이 완료되지 않았다면 true를, 완료되었다면 false를 반환함으로써 교차 탐색이 완료될 때까지 탐색을 수행할 수 있도록 한다. 이후 rayQueryGetIntersectionTypeEXT를 통해 교차 물체가 삼각형 기하로 이루어졌는지를 확인한다. 본 연구에서는 삼각형 기하로 이루어진 장면만을 사용하여 별도의 추가 작업을 수행하지 않지만, 사용자가 직접 정의한 물체를 추가할 경우 이에 대한 교차 여부를 확인해야한다.

Algorithm 1 Alpha texture process using ray query

```
1:  procedure ISOPAQUE(rQuery)
        ▷ Get the geometry index of intersection.
2:      uint geomIdx = GETINTERSECTGEOMIDX(rQuery);

        ▷ Check whether it is opaque or not.
3:      if ISOPAQUEFRAGMENT(geomIdx) then
4:          return true;
5:      else
6:          return false;
7:      end if
8:  end procedure

9:  procedure PROCESS OF EACH BOUNCE OF RAY
        ▷ Find an intersection.
10:     rayQueryEXT rQuery;
11:     while PROCEED(rQuery) do
12:         if GETINTERSECT(rQuery) == Triangle then
13:             if ISOPAQUE(rQuery) then
14:                 CONFIRMINTERSECTION(rQuery);
15:             end if
16:         end if
17:     end while

        ▷ If an intersection exists, compute shading.
18:     if ExistConfirmedIntersection then
19:         Compute local shading.
20:         continue;
21:     else
22:         break;
23:     end if
24: end procedure
```

본 알고리즘에서는 교차를 바로 확정하지 않고 교차 지점 텍스춰의 불투명도를 추가로 확인하여 교차 여부를 결정하는데, 이를 수행하는 부분이 procedure ISOPAQUE(*rQuery*)이다. rayQueryGetIntersectionGeometryIndexEXT, rayQueryGetIntersectionPrimitiveIndexEXT, rayQueryGetIntersectionBarycentricsEXT는 각각 현재 교차한 기하 인덱스, 기하를 이루는 삼각형 프리미티브 인덱스, 교차 삼각형 내 정확한 교차 지점을 반환하는 함수로, 이러한 정보들을 바탕으로 텍스춰의 알파 값을 확인하고, 해당 지점이 불투명하다면 rayQueryConfirmIntersectionEXT 함수를 통해 교차를 확정한다.

광선 질의에서 교차는 후보군 (candidate)과 확정군 (committed)으로 분류된다. 후보군은 탐색 과정에서 교차했지만 확정되지 않은 교차를, 확정군은 rayQueryConfirmIntersectionEXT 함수를 통해 확정한 교차를 의미한다. rayQueryGet...EXT 형태의 함수에 bool 타입 인자를 전달함으로써 후보군과 확정군 각각에 대한 정보를 얻을 수 있다. 인자 값으로 false를 전달하면 해당 탐색에서 발견한 확정되지 않은 후보군 정보를, true를 전달하면 확정군에 대한 교차 정보를 얻는다. 따라서 가장 가까운 교차가 확정된 이후 쉐이딩 연산을 하는 시점에 해당 함수를 호출하며 true 값을 전달하면 확정군의 교차 정보를 얻을 수 있다.

성능 측면에서 광선 추적 파이프라인에 비해 광선 질의 방식이 유의미한 향상을 보였다. 또한, 광선 질의 방식은 광선 추적 파이프라인에 비해 상대적으로 다양한 하드웨어 및 플랫폼에서 폭넓게 지원되므로 실제 응용에서 보다 유연하게 활용 가능하다.

4. 적응적 그림자 광선 샘플링을 위한 빛의 감쇠 함수 확장

4.1 적응적 그림자 광선 샘플링

기존 광선 추적 렌더러에서는 장면 내 모든 광원에 대해 그림자 광선을 투사하므로, 광원의 영향을 거의 받지 않는 물체에 불필요한 쉐이딩 연산이 수행된다. 이에 광원의 개수가 증가하면 그림자 광선의 비용이 급격히 증가하며, 특히 모바일 환경에서 성능 저하의 주된 원인이 된다. 따라서 실시간 렌더링이나 모바일 같은 자원이 제한된 환경에서는 불필요한 광선 투사 및 쉐이딩을 줄여 효율적으로 그림자를 생성하기 위한 방안이 필요하다.

최근, 광원의 유효 범위를 설정하고, 해당 범위 내에 포함되지 않는 픽셀에 대해서는 그림자 광선을 투사하지 않는 방식의 광원 범위 제한 (light culling) 기반 최적화 기법이 도입되고 있다. 픽셀마다 모든 광원에 대해 그림자 광선을 투사하여 그림자 여부를 검사하는 대신, 거리 기반 필터링과 같은 간단한 조건 검사를 통해 광원의 유효 범위 여부를 사전에 결정함으로써 고비용의 광선 추적 연산 비용을 효과적으로 절감할 수 있다. 구체적으로, 각 광원에 대해 감쇠 함숫값이 일정 임계값 이하가 되는 거리를 유효 거리 (d_{max})로 정의하여, 광원을 중심으로 구 형태의 범위를 설정한다. 그리고 광원 중심과 픽셀 위치 사이의 거리가 유효 범위를 초과하는 경우 해당 픽셀에서는 그림자 광선을 투사하지 않는다.

4.2 빛의 감쇠 함수의 확장

일반적으로 사용되는 빛의 감쇠 함수는 이차 함수의 역수 형태로, 초반에 급격히 감소한 후 완만하게 유지되는 곡선의 형태를 갖는다. 이러한 특성으로, 광원과 거리가 매우 먼 지점에서도 육안으로 구분하기 어려운 미세한 수준의 조명 효과를 쉐이딩해 불필요한 쉐이딩 연산을 수행하게 된다. 하지만, 무의미한 쉐이딩 연산을 피하기 위해 앞서 기술한 바와 같이 광원의 쉐이딩 범위를 구의 형태로 제안하면, Figure 2의 (a)와 같이 광원과 매우 가까운 영역에서만 밝기가 유지되어 장면이 전체적으로 어두워지는 문제가 발생한다. 또한, 이를 위해 감쇠 함수의 임계값을 높일 경우, Figure 2의 (b)와 같이 광원의 영향 범위가 뚜렷하게 드러나 부자연스러운 렌더링 결과가 나타난다. 이에 본 연구에서는 제한된 범위를 갖는 광원에 적합한 새로운 빛의 감쇠 함수를 제안한다.

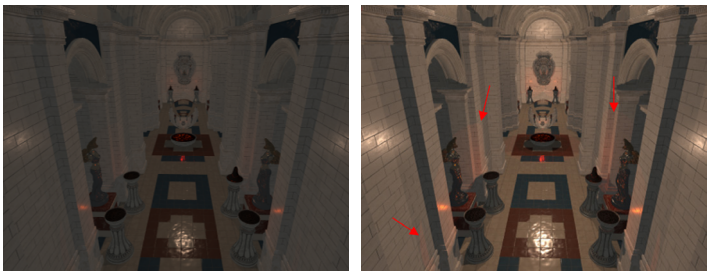
$$f(x) = \begin{cases} \dfrac{1}{ax^2 + bx + 1} & \text{if } 0 < x < \alpha d_{max} \\ \dfrac{1}{h\left(x - \alpha d_{max}\right)^2 + \dfrac{1}{\beta}} & \text{if } \alpha d_{max} < x < d_{max} \end{cases}$$

여기서 $a = \left(1 - \dfrac{1}{\beta}\right) \dfrac{1}{\alpha^2 d_{max}^2}$,

$$b = -2\left(1 - \frac{1}{\beta}\right) \frac{1}{\alpha d_{max}} \quad ,$$

$$h = \frac{1}{(1-\alpha)^2 d_{max}^2} \left(\frac{1}{\gamma} - \frac{1}{\beta}\right)$$

광원의 유효거리 d_{max}에 대하여 α, β, γ $(0 \leq \alpha, \beta, \gamma \leq 1)$ 그리고 I 등 4개의 인자를 사용하여 함수의 형태를 조절한다. $f(x)$와 $g(x)$가 만나는 교점은 거리와 밝기가 각각 α, β이고, d_{max}는 광원의 유의미한 범위를 나타낸다. 또한 γ는 해당 범위에서의 광원의 밝기를 의미하고, 마지막으로 I는 각 광원의 색 (r, g, b)중 가장 큰 값이다. 이러한 형태의 그래프를 통해 유의미한 범위에서만 쉐이딩 연산을 수행함과 동시에 그 결과가 자연스럽게 보여질 수 있다. 또한 각 인자를 적절히 조절함으로써 광원마다 서로 다른 형태의 감쇠 효과를 적용할 수 있으며, 해당 인자들을 통해 광원의 밝기와 빛 감쇠 정도를 직관적으로 조절 가능하다.



(a) Too dark rendering result (b) Exposed light boundaries

Figure 2: Rendering results of existing attenuation function

5. 실험 결과

5.1 실험 환경 및 데이터 구성

본 실험에서는 Xclipse 950과 Adreno 830 GPU가 탑재된 모바일 기기 두 대를 실험 장비로 사용하였다. 기본적으로 Table 1의 모든 렌더러에 대해 성능을 측정하였으나, Adreno 830이 탑재된 모바일 기기는 Vulkan의 광선 추적 파이프라인 확장을 지원하지 않아 광선 질의 렌더러만을 사용하여 측정하였다. Xclipse 950이 탑재된 모바일 기기는 삼성전자로부터 지원받은 테스트 기기로, 발열 및 전력에 따른 성능 편차를 줄이기 위해 GPU 클럭 주파수를 816MHz로 고정하고 실험하였다. Adreno 830이 탑재된 모바일 기기는 상용 스마트폰으로, GPU 클럭 주파수 고정이 불가능하여 성능 측정 결과상으로는 더 빠른 렌더링 속도를 보인다. 하지만 Xclips 950이 탑재된 모바일 기기의 성능 제한을 해제하면 두 기기가 거의 유사한 성능을 보였다. 렌더링 해상도는 일반적인 모바일 기기 해상도에 맞춘 1440 x 2560으로 설정하였고, 렌더러 및 장면 별 프레임 속도 외에도 AMD의 라데온 GPU 프로파일러 (Radeon GPU Profiler, RGP) [9]를 통해 렌더러에 대한 프로파일링을 수행하였다. Figure 3은 라데온 GPU 프로파일러를 통해 확인한 광선 추적 파이프라인의 다이어그램 예시를 보여준다.

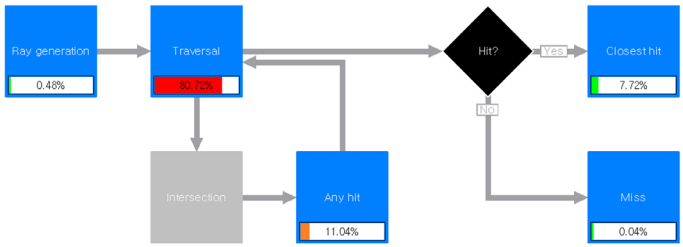
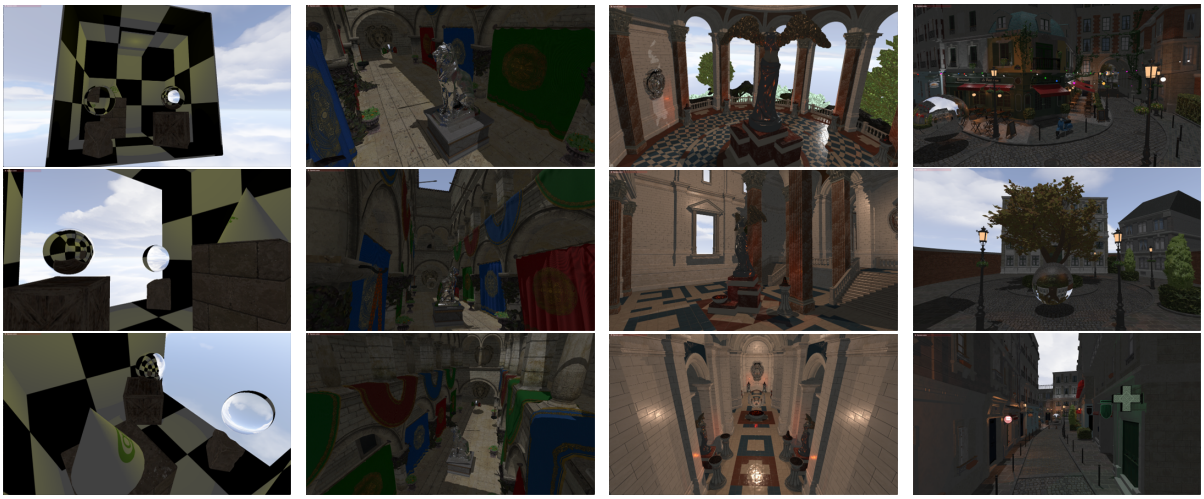


Figure 3: Example of ray tracing pipeline profiling

각 렌더러는 Whitted 스타일의 광선 추적 방식으로 렌더링을 수행하며, 전체 광선 추적 렌더러와 하이브리드 광선 추적 렌더러에 대한 최대 바운스는 각각 5와 4로 설정하였다. 실험 장면으로는 각각 17K, 304K, 608K, 2,829K 개의 삼각형으로 이루어진 Cornell Box, Sponza, Sun Temple, Bistro를 사용하였으며, 각각 1개의 정적 광원, 1개의 동적 광원, 1개의 동적 광원과 31개의 정적 광원, 그리고 48개의 정적 광원을 배치하였다. 실험에 사용된 장면에 대한 구체적인 정보는 Table 2에 제시되어 있다. 장면별로 세 가지 카메라 뷰에 대한 프레임 속도를 평균 내어 측정하였으며, Figure 4는 각 카메라 뷰에 대한 렌더링 결과이다.

Table 2: Sizes of example 3D scenes

장면	Cornell Box	Sponza	Sun Temple	Bistro
정점	10,887	229,940	569,427	2,929,933
삼각형	17,648	304,310	608,406	2,829,226
광원	1	1	32	48
텍스쳐	12	71	103	406
BVH 크기 (MiB)	3.0	47.0	94.0	439.0



(a) Cornell Box (b) Sponza (c) Sun Temple (d) Bistro

Figure 4: Tested views of example scenes

Bistro 장면은 대량의 삼각형 및 텍스처를 포함하는 고품질 대용량 데이터로, 모바일 기기에서는 메모리 제한으로 인해 원본 데이터를 사용할 수 없어 텍스터 압축을 통해 데이터 크기를 줄임으로써 렌더링을 수행하였다. 기하 물체의 재질 및 조명 효과는 컴퓨터 그래픽스 분야의 현대적 흐름에 맞추어 물리 기반 쉐이딩 기법을 통해 표현하였다. 각 장면에는 나뭇잎, 쇠사슬 등과 같은 다량의 세밀한 물체가 포함되어 있으나, 기본적으로는 알파 클리핑 (alpha clipping)을 위한 any-hit 쉐이더를 적용하지 않고 성능을 측정하였다. 그림자 맵핑 기법 적용 시에에는 각 점 광원에 대해 1024 × 1024 크기의 그림자 텍스춰 6개를 생성하였다.

5.2 프로파일링을 통한 성능 분석

Table 3은 Xclipse 950이 탑재된 모바일 기기에서 측정된 광선 추적 파이프라인의 각 단계 별 지연 시간과 그에 따른 비중을 나타낸 것이다. 해당 Table에서 instruction cost 항목은 총 지연 시간 중 해당 쉐이더가 차지하는 비율을 의미하며, total latency는 실제 지연 시간을 의미한다. traversal 항목의 instrution cost는 any-hit 적용 시 약 80–86%, 미적용 시 약 64–85%로 광선 추적 파이프라인 전체 계산 비용의 대부분을 차지하는 것을 확인할 수 있다. 앞서 서술한 바와 같이, 고성능 GPU 광선 추적 모듈의 도움을 받을 수 있는 PC와는 달리, 모바일에서는 여전히 광선-물체 교차 계산 및 탐색 비용이 상당한 부담이며 이를 줄일 수 있는 소프트웨어적인 노력이 필요하다고 결론지을 수 있다.

any-hit 적용 여부 또한 성능에 큰 영향을 미치므로 신중한 선택이 요구된다. Table 4는 Xclipse 950에서 RTP-FRT에 대해 any-hit 쉐이더 적용 여부에 따른 프레임 속도를 비교한 결과이다. any-hit 쉐이더는 전체 광선 추적 파이프라인 내에서 7.95–14.39% 정도의 적지 않은 비중을 차지하지만, 이와 같은 비중에도 불구하고 해당 비율을 훨씬 상회하는 만큼의 프레임 속도 차이가 나는 것을 확인할 수 있다. 이는 any-hit 쉐이더의 적용에 따라 탐색 시간이 1.3–4.3 배까지 증가했기 때문이다. any-hit 쉐이더를 적용하면, 매 광선-물체 교차마다 알파값에 따라 무시되는 삼각형이 생겨나므로 필연적으로 더 많은 삼각형을 탐색해야 하며, 이로인해 교차 계산 시간이 전반적으로 늘어난다. 또한, 매 탐색마다 any-hit 쉐이더가 호출되는 한편, 해당 연산 결과가 이후 탐색의 범위에 영향을 미치게 되어 GPU 병렬성에도 부정적인 영향을 미친다. 모바일 기기의 경우 PC에 비해 상대적으로 GPU의 성능이 낮고, 광선 추적 파이프라인이 최적화되지 않았기 때문에 any-hit 쉐이더의 적용 시에 성능 저하가 더욱 두드러지게 나타난다. 2장에서 언급한 바와 같이, NVIDIA에서는 미세 불투명도 맵을 통해 any-hit 쉐이더의 부담을 줄이는 방안을 제시한 바 있다. 그러나 해당 기법은 모바일 환경에는 적용이 어려워, 모바일의 경우 소프트웨어적 최적화 방안 마련이 여전히 요구된다. 본 논문에서 제안한 하이브리드 광선 추적 렌더러는 주 광선에 대한 any-hit 계산을 래스터화 단계의 프래그먼트 쉐이더에서 처리함으로써 알파 클리핑의 효과를 일부 재현하면서도 좋은 성능을 유지할 수 있었다. 다만 해당 효과는 주 광선에만 한정되어 그림자 및 반사, 굴절 등의 이차 광선에는 적용되지 않기 때문에 해당 영역에 대해서는 품질 저하가 발생하였다.

Table 5는 두 모바일 기기에서 각 장면에 대한 렌더러의 프레임 속도를 측정한 결과이다. 하이브리드 광선 추적 렌더러는 전체 광선 추적 렌더러에 비해 전체적으로 좋은 성능을 보였다. 모든 기하 물체에 대해 투사하는 주 광선을 래스터화 과정으로 대체하면서 그림자 광선 및 일부 픽셀에 대해서만 반사 및 굴절 광선을 투사함에 따라서 주 광선에 대한 광선-물체 교차 계산 비용을 절감했기 때문이다. 하지만, 장면의 크기가 커질수록 성능 향상 비율이 미미해지는데, 이는 앞서 언급한대로 장면의 크기가 커질수록 래스터화의 비용이 매우 커지기 때문이다. Table 6는 하이브리드 광선 추적 렌더러의 래스터화 파이프라인과 광선 추

Table 3: Instruction cost and total latency comparison

장면			ray-gen	traversal	closest-hit	any-hit	miss
any-hit 적용	CornellBox	instruction cost (%)	2.31	79.66	9.49	7.95	0.59
		total latency (clks)	13,672,557	472,367,692	56,284,792	47,149,415	3,482,236
	Sponza	instruction cost (%)	0.62	79.73	5.26	14.39	0.00
		total latency (clks)	11,824,336	1,519,432,700	100,211,922	274,197,693	0.00
	Sun Temple	instruction cost (%)	0.48	80.72	7.72	11.04	0.04
		total latency (clks)	7,528,461	1,268,385,673	121,310,608	173,487,690	652,947
	Bistro	instruction cost (%)	0.26	86.33	3.9	9.5	0.0
		total latency (clks)	3,395,776	1,111,100,736	50,178,084	122,266,480	23756
any-hit 미적용	CornellBox	instruction cost (%)	6.27	63.89	28.24	0.00	1.61
		total latency (clks)	11,476,644	117,013,208	51,714,499	0.00	2,942,533
	Sponza	instruction cost (%)	2.71	73.87	23.4	0.00	0.02
		total latency (clks)	12,887,548	350,896,858	111,131,870	0.00	75,019
	Sun Temple	instruction cost (%)	1.51	71.89	26.51	0.00	0.1
		total latency (clks)	9,656,437	461,127,769	170,039,365	0.00	648,344
	Bistro	instruction cost (%)	0.72	85.03	14.24	0.00	0.01
		total latency (clks)	7,492,994	881,970,189	6,477,980	0.00	46,459

Table 4: Cost of any-hit shader in RTP-FRT

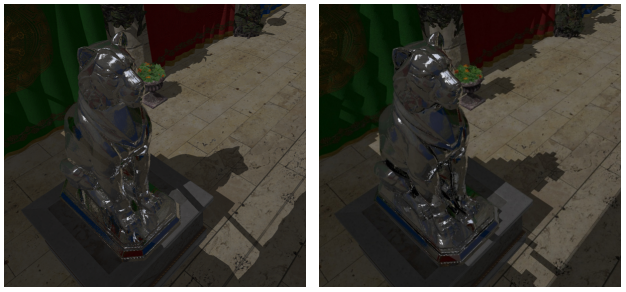
단위: FPS(ms)

	장면	RTP-FRT	RTP-HRT
any-hit 적용	Cornell Box	9.7 (106.1)	24.7 (41.1)
	Sponza	4.1 (242.0)	8.3 (122.5)
	Sun Temple	3.1 (321.6)	4.9 (205.4)
	Bistro	2.2 (469.6)	2.9 (398.9)
any-hit 미적용	Cornell Box	26.7 (38.5)	73.2 (14.0)
	Sponza	14.0 (71.7)	30.8 (32.8)
	Sun Temple	9.0 (112.0)	19.2 (52.9)
	Bistro	8.8 (127.0)	6.1 (191.5)

적 파이프라인의 소요 시간 및 그에 따른 비율을 나타낸 것으로, 장면의 크기가 커짐에 따라 광선 추적 파이프라인의 상대적인 비중이 줄어들어 절반 이하로 떨어진 것을 확인할 수 있다.

그림자 맵핑과 그림자 광선 투사 여부에 따른 성능은 GPU 아키텍춰에 따라 편차를 보였지만, 장면의 크기가 커지고, 광원의 수가 많아질수록 성능이 감소하거나 성능의 증가폭이 줄어들었다. 그림자 맵핑 방식의 경우, 정적 광원에 대해 전처리 과정에서 그림자 맵을 생성하고, 동적 광원에 대해서만 매 프레임마다 그림자 맵을 생성한다. 장면 내에 동적인 물체가 없고, 광원의 수가 매우 적다면 우수한 성능을 보이나, 게임과 같은 다량의 동적 물체 및 광원이 사용되는 분야에서는 적용하기 힘들뿐더러 렌더링된 그림자의 품질이 매우 좋지 않았다.

Figure 5는 그림자 광선과 그림자 맵핑의 렌더링 결과 차이를 도시한 것으로, 그림자 맵핑 적용 시에는 낮은 그림자 맵 해상도로 인해 에일리어싱 효과가 두드러진다. 그림자 텍스춰의 해상도를 높여 그림자의 품질을 향상시킬 수 있지만, 메모리 크기가 한정된 모바일 기기에서는 다량의 고품질 그림자 텍스춰를 사용하는 것이 큰 부담으로 작용한다.



(a) Shadow ray (b) Shadow map

Figure 5: Comparison of shadow generation

광선 질의 확장을 적용할 경우, 광선 추적 파이프라인 확장을 적용했을 때보다 성능이 비약적으로 향상된 것을 확인할 수 있다. 현재 AMD의 GPU 드라이버에서는 광선 추적 쉐이더에 대해 간접 (indirect) 모드와 통합 (unified) 모드라는 두 가지 컴파일 모드를 제공하고 있다 [10]. 간접 모드는 광선 추적에 사용되는 쉐이더를 분리하여 컴파일한 후 개별적으로 호출하는 방식이며, 통합 모드는 모든 쉐이더를 하나의 코드 블록으로 통합하여 처리하는 방식으로, 일반적으로 통합 모드로 컴파일 하는 것이 간접 모드에 비해 좋은 성능을 보인다. 광선 질의 확장의 경우, 프로파일러가 제공하는 정보가 제한적이기 때문에 파이프라인 내에서의 광선 추적 탐색 비용에 대해서는 확인할 수 없었다. 하지만 광선 추적 파이프라인의 any-hit, closest-hit, miss 등의 쉐이더 내에서 재귀적으로 광선 추적 함수를 호출할 경우 간접 모드로 컴파일을 수행하며 본 연구에서 광선 추적 파이프라인 확장을 사용한

Table 5: Performance comparison of ray tracing renderers

단위: FPS(ms)

	Scene	RTP-FRT	RTP-HRT	RTP-HRT-SM	RQ-FRT	RQ-HRT	RQ-HRT-SM
Xclipse 950	Cornell Box	26.7 (38.5)	73.2 (14.0)	180.3 (5.6)	109.0 (9.7)	144.9 (7.1)	204.3 (5.0)
	Sponza	14.0 (71.7)	30.8 (32.8)	29.8 (33.6)	46.2 (21.8)	45.8 (22.0)	29.6 (33.8)
	Sun Temple	9.0 (112.0)	19.2 (52.9)	17.5 (57.2)	29.8 (33.9)	32.2 (31.4)	18.3 (54.5)
	Bistro	8.8 (127.0)	6.1 (191.5)	8.3 (130.8)	16.4 (74.2)	7.2 (161.2)	8.5 (126.8)
Adreno 830	Cornell Box	-	-	-	162.1 (6.6)	162.5 (6.3)	550.7 (1.9)
	Sponza	-	-	-	55.7 (18.1)	62.4 (16.2)	85.0 (11.8)
	Sun Temple	-	-	-	37.4 (26.9)	43.8 (22.9)	46.4 (21.6)
	Bistro	-	-	-	23.7 (59.0)	16.5 (73.8)	36.6 (28.9)

Table 6: Duration and percentage of two pipelines in RTP-HRT

Scene	Rasterization pipeline		Ray tracing pipeline	
	Duration (μs)	Portion (%)	Duration (μs)	Portion (%)
Cornell Box	1,557.7	15.2	8,719.8	84.8
Sponza	10,720.6	27.1	28,896.5	72.9
Sun Temple	11,140.6	22.8	37,711.7	77.2
Bistro	113,943.7	51.3	107,278.1	48.7

렌더러 또한 간접 모드로 컴파일되는 것을 확인하였다. 이에 따라 분리되어 컴파일된 쉐이더 간 상호 연산이 모바일 기기에 큰 부담으로 작용하며, 이를 단일 쉐이더 내에서의 연산으로 전환한 것이 큰 성능 향상에 기여한 것으로 추측한다.

5.3 적응적 그림자 광선 샘플링 적용 결과

Table 7은 Xclipse 950이 탑재된 모바일 기기에서 적응적 그림자 광선 샘플링 적용 여부에 따른 프레임 속도를 측정한 결과이다. 적응적 그림자 광선 샘플링을 적용할 경우, 광원의 영향 범위 외부에서는 그림자 광선의 투사를 생략함으로써 광선 추적 성능이 비약적으로 (446.2–1078.5%) 향상된 것을 확인할 수 있다. 이는 모바일 GPU의 제한된 연산 자원 하에서 해당 기법이 특히 효과적임을 시사한다. 동시에, 장면에 따라 적절한 인자를 설정할 경우 기존 함수 대비 광량 표현의 자연스러움은 유지되었으며, 광원 경계의 불연속 현상도 관찰되지 않았다.

Table 7: Performance comparison of adaptive shadow ray

단위: FPS(ms)

	Scene	RQ-FRT	RQ-HRT
Fixed	Sum Temple	4.7(213.3)	4.6(219.1)
	Bistro	1.4(797.2)	1.3(847.0)
Adaptive	Sum Temple	29.0(34.7)	31.6(31.9)
	Bistro	16.5(73.9)	7.1(164.6)

Figure 6는 빛의 감쇠 효과 함수에서 각 인자의 값에 따른 함수의 개형을 시각적으로 나타낸 것이다. 각 함수의 렌더링 결과는 Figure 7에서 확인할 수 있다. 결과적으로, 본 감쇠 함수는 성능-품질 균형을 효과적으로 달성할 수 있는 수단이며, 향후 실시간 렌더링 환경에서의 광원 처리 전략으로 활용될 수 있다.

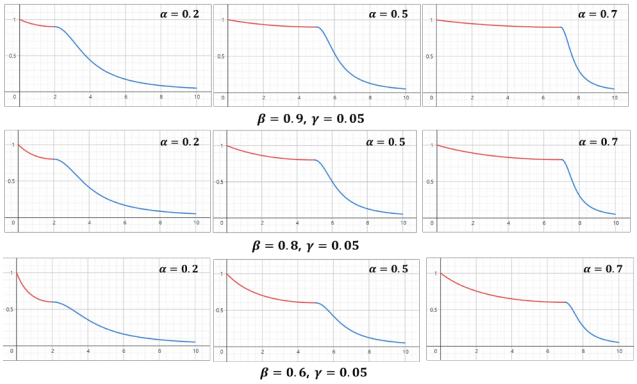


Figure 6: Attenuation function control using alpha, beta, and gamma parameters

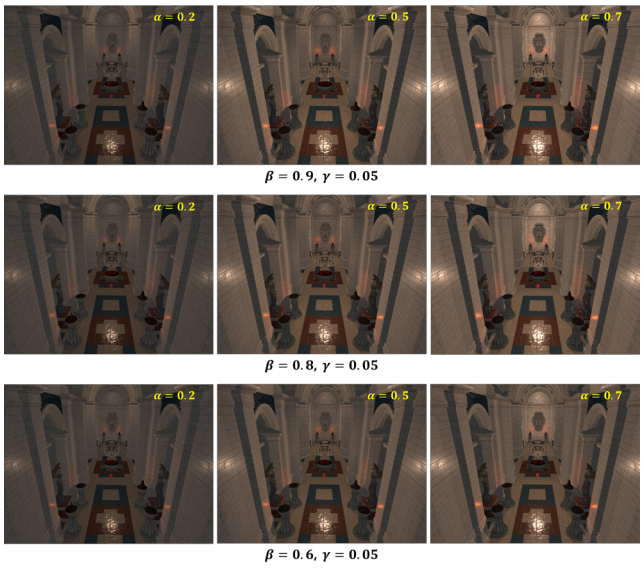


Figure 7: Adaptive control of proposed light attenuation function

6. 결론

본 논문에서는 모바일 플랫폼에서 Vulkan 기반의 효율적인 광선 추적 렌더러를 다양한 방식으로 설계하고 그 성능을 분석함으로써 자원이 부족한 환경에서 수행되는 광선 추적 작업의 부하 및 특성을 파악하였다. 일반적으로는 전체 광선 추적 렌더러에 비해 하이브리드 광선 추적 렌더러의 성능이 우수하였다. 그러나 장면을 구성하는 삼각형 개수가 증가할수록 하이브리드 광선 추적 렌더러의 기하 단계 비용이 선형적으로 증가하여 성능 향상 정도가 감소하였다. 또한 광선 질의 방식이 광선 추적 파이프라인 방식에 비해 더 우수한 성능을 보였다.

앞에서 언급한 다양한 방식의 렌더러 설계 및 실험을 통해 자원이 제한된 모바일 플랫폼에서는 광선의 개수가 성능에 지배적인 영향을 미친다는 사실을 확인하였다. 따라서 그림자 광선을 선택적으로 투사하기 위해 5장에서 언급한 '적응적 그림자 광선 샘플링' 기법을 도입하였다. 광원의 감쇠 효과에 따른 유의미한 쉐이딩 범위를 고려하여 해당 범위에 속하는 경우에만 그림자 광선을 투사하도록 함으로써 상당한 성능 향상을 확인하였다. 또한 성능 측면에서 효율적인 감쇠 효과 적용을 위해 확장한 감쇠 효과 함수를 제안하였다.

본 논문은 모바일 플랫폼을 겨냥하여 다양한 방식의 광선 추적 렌더러를 설계하고 그 성능을 비교하였다. 그래픽 하드웨어의 성능이 나날이 발전하고 있지만 모바일 환경에서 광선 추적 기법을 자유롭게 사용하는 것은 성능적 한계가 있어, 여전히 소프트웨어적인 최적화가 필요한 상황이다. 또한 광선 질의 방식을 활용한 구현의 경우 광선 추적 파이프라인보다 성능이 뛰어난 반면 관련 자료가 매우 부족하다. 본 논문의 실험 결과가 모바일 광선 추적 렌더러의 미래 연구에 큰 길잡이가 될 것으로 보인다.

향후에는 광선 추적을 위한 공간 가속 구조를 탐색하기에 효율적인 구조로 분할하여 구축함으로써 성능에 지배적인 영향을 미치는 광선-물체 교차 계산을 줄이기 위한 연구가 이루어지길 기대한다. 또한 본 연구의 모바일 광선 추적 렌더러를 기반으로 최근 활발히 연구가 진행되고 있는 3D Gaussian Splat 등의 새로운 유형의 장면에 대해서도 모바일 환경으로 이식 후 성능을 유지하기 위한 연구가 진행 중이다 [11].

감사의 글

본 연구는 삼성전자의 지원을 받아 수행한 연구 결과임. 또한 , 일부는 과학기술정보통신부 및 정보통신기획평가원의 메타버스 융합 대학원 연구 결과로 수행되었음 (RS-2022-00156318).

References

[1] T. Whitted, "An improved illumination model for shaded display," *Communications of the ACM*, vol. 23, no. 6, pp. 343–349, 1980.

[2] S. G. Parker, J. Bigler, A. Dietrich, H. Friedrich, J. Hoberock, D. Luebke, D. McAllister, M. McGuire, K. Morley, A. Robison, and M. Stich, "OptiX: A General Purpose Ray Tracing Engine," *ACM Transactions on Graphics (TOG)*, vol. 29, no. 66, pp. 1–13, 2010.

[3] NVIDIA, *NVIDIA OptiX 8.1 Programming Guide*. NVIDIA Corporation, 2024.

[4] NVIDIA, "NVIDIA Ada Lovelace Architecture." `https://images.nvidia.com/aem-dam/Solutions/geforce/ada/nvidia-ada-gpu-architecture.pdf`, 2022.

[5] Khronos Group, "Vulkan ray tracing specification, khr_ray_query extension," 2020.

[6] C. Barré-Brisebois, H. Halén, G. Wihlidal, A. Lauritzen, J. Bekkers, T. Stachowiak, and J. Andersson, "Hybrid rendering for real-time ray tracing," in *Ray Tracing Gems*, pp. 437–473, Apress, 2019.

[7] B. Burley, "Physically-based shading at disney," in *ACM SIGGRAPH 2012 Course Notes*, (Los Angeles, CA), ACM, 2012.

[8] Khronos Group, *OpenGL 4.6 Core Profile Specification*. 2022. Version 4.6 (Core Profile), released May 2022.

[9] AMD, *Radeon GPU Profiler*. Advanced Micro Devices, Inc., 2024. `https://gpuopen.com/rgp/`.

[10] AMD, *RGP documentation*. Advanced Micro Devices, Inc., 2025. `https://gpuopen.com/manuals/rgp_manual/rgp_manual-events_windows/`.

[11] N. Moenne-Loccoz, A. Mirzaei, O. Perel, R. de Lutio, J. Martinez Esturo, G. State, S. Fidler, N. Sharp, and Z. Gojcic, "3D Gaussian Ray Tracing: Fast tracing of particle scenes," *ACM Trans. Graph.*, vol. 43, Nov. 2024.

〈저자소개〉

유택근
- 2024년 서강대학교 공과대학 컴퓨터공학과 학사
- 2024년~현재 서강대학교 소프트웨어융합대학 컴퓨터공학과 석사과정
- 관심 분야: 광선 추적법, 모바일 렌더링, 실시간 렌더링
- https://orcid.org/0009-0000-2287-9595

윤성호
- 2024년 서강대학교 공과대학 컴퓨터공학과 학사
- 2024년~현재 서강대학교 소프트웨어융합대학 컴퓨터공학과 석사과정
- 관심분야: 렌더링 파이프라인, 실시간 렌더링, 광선 추적법
- https://orcid.org/0009-0003-9745-9212

조세희
- 2024년 서강대학교 공과대학 컴퓨터공학과 학사
- 2024년~현재 서강대학교 소프트웨어융합대학 컴퓨터공학과 석사과정
- 관심분야: 모바일 렌더링, 가상/증강 현실, GPGPU
- https://orcid.org/0009-0007-1678-9244

서웅
- 2012년 한국외국어대학교 공과대학 전자공학과 학사
- 2014년 서강대학교 공과대학 컴퓨터공학과 석사
- 2021년 서강대학교 공과대학 컴퓨터공학과 박사
- 2021년~2022년 LG전자 연구원
- 2022년~현재 삼성전자 연구원
- 관심분야: 컴퓨터 그래픽스, 모바일 렌더링, 광선 추적법
- https://orcid.org/0000-0001-8272-9169

임인성
- 1985년 2월 서울대학교 자연과학대학 계산통계학과 이학사
- 1987년 5월 Rutgers - The State University of New Jersey 컴퓨터학과 이학석사
- 1991년 7월 Purdue University 컴퓨터학과 이학박사
- 1999년 7월~2000년 7월 University of Texas at Austin 연구 교수
- 1993년 3월~현재 서강대학교 공과대학 컴퓨터공학과 교수
- 관심 분야: 실시간 렌더링, 가상/증강 현실, GPGPU 컴퓨팅
- https://orcid.org/0000-0002-5611-925X