DirectX 12 Meshlet과 IA 기반의 하이브리드 렌더링 파이프라인 구현

Implementation of a Hybrid Rendering Pipeline Based on Meshlets and the Input Assembler in DirectX 12

유현수
동명대학교 컴퓨터공학과

지도교수: 고정국

요 약

본 논문은 DirectX 12에서 Input Assembler(IA) 경로와 Amplification Shader(AS)·Mesh Shader(MS) 기반 Meshlet 경로를 하나의 Renderer에 통합하고, Mesh 그룹마다 두 경로 중 하나를 런타임에 선택하는 하이브리드 렌더링 파이프라인을 구현한다. 선택기는 환경마다 사전에 베이크한 GPU 비용 정보로 만드는 환경 특화형 모델로, 그룹의 컬링 전 Primitive 수 P와 카메라의 절두체 컬링 비율 C, 또는 위치별 공간 비용 Grid를 입력으로 쓴다.

11개 환경에서 선택기마다 IA 기본·MS 기본 정책을 비교한 결과, 모든 모델에서 MS 기본이 IA 기본보다 1.9~9.6% 빨랐다. 이에, 기본 경로를 MS로 두고 실제 혼합 시간으로 이득이 확인된 군집만 IA로 허용하는 MS 기본 K-Means를 제안하였다. 최종 평가에서 제안 모델은 GPU Scene 시간을 IA 전용 대비 15.83%, MS 전용 대비 3.01% 줄여 다섯 선택기 모델 중 유일하게 두 전용 경로를 모두 앞섰고, 같은 프레임 안의 IA·MS 혼합을 6개 환경에서 확인하였다. 이는 환경별 사전 베이크 기반의 하이브리드 선택이 단일 경로보다 짧은 GPU 시간을 낼 수 있음을 보여 준다.

Keywords : DirectX 12, Mesh Shader, Amplification Shader, Meshlet, 하이브리드 렌더링, 파이프라인 선택, 환경 특화 모델

1. 서론

1.1 연구 배경

전통적인 IA 파이프라인에서는 Fixed-function Input Assembler가 정점·인덱스 버퍼를 읽고, Vertex Shader가 모든 제출 정점을 변환한 뒤 Rasterizer 단계에서 화면 밖 삼각형을 컬링한다. 여기서 컬링(Culling)은 최종 화면에 기여하지 않을 Geometry를 판정하여 이후 단계의 처리 대상에서 제외하는 것을 말한다. 판정 기준에 따라 카메라 시야 절두체 밖의 Geometry를 제외하는 절두체 컬링(Frustum Culling), 카메라를 등진 면을 제외하는 후면 컬링(Back-face Culling), 다른 물체에 가려진 Geometry를 제외하는 오클루전 컬링(Occlusion Culling) 등으로 나뉘며, 판정을 파이프라인의 앞 단계에서 할수록 뒤 단계의 비용을 더 많이 아낄 수 있다. IA 경로의 Rasterizer 컬링은 정점 변환이 끝난 뒤에 일어나므로, 최종 화면에 기여하지 않는 Geometry도 정점 처리 비용을 모두 치른다.

Mesh Shader 파이프라인은 이 앞단을 Programmable Amplification Shader와 Mesh Shader로 대체한다[1][2]. Mesh를 수십 개 정점·삼각형 단위의 Meshlet으로 나누고, AS가 Meshlet 단위로 가시성을 검사해 살아남은 Meshlet만 MS로 넘기면 정점 처리 전에 보이지 않는 Geometry를 사전에 제거할 수 있다.

그러나, 두 경로 중 한쪽이 항상 빠르지는 않다. IA는 하드웨어 Instancing과 성숙한 정점 재사용 구조를 활용하고[3][4], Meshlet 경로는 컬링 이득 대신 AS 실행·경계 자료 읽기·Meshlet 간 정점 중복 처

리라는 고정 비용을 진다. 어느 쪽이 유리한지는 Geometry 규모와 화면 밖 비율, 카메라 구도에 따라 달라지므로, 화면의 Mesh 그룹마다 더 유리한 경로를 골라 한 프레임 안에서 섞어 쓰는 구조를 생각할 수 있다.

1.2 문제 정의

본 연구는 어떤 환경에나 그대로 적용되는 범용 모델이 아니라, 환경 특화형 모델을 다룬다. 게임의 맵은 개발 단계에서 고정되므로, 구역마다 Mesh 그룹의 GPU 비용을 미리 측정·베이크하고 그 환경에 맞춘 선택기를 만들 수 있다. 이 접근에는 세 가지 과제가 있다. 첫째, 선택은 매 프레임 현재 카메라 상태로 계산되어야 하며 그 비용이 이득보다 작아야 한다. 둘째, 베이크할 수 있는 비용은 그룹 단위지만 실제 프레임 시간은 여러 그룹이 섞인 배치의 비용이어서, 그룹 비용의 단순 합과 다를 수 있다. 셋째, 학습에 쓴 카메라 경로와 다른 경로로 같은 환경을 지나갈 때도 이득이 유지되어야 한다.

1.3 연구 질문

- RQ1. 한 화면에서 IA와 MS를 섞고, 현재 카메라 상태에 따라 같은 Mesh 그룹의 경로를 바꿀 수 있는가?
- RQ2. 환경별 베이크 정보로 만든 $P \cdot C$ 분류기, 공간 비용 Grid, 최소 비용 Chunk는 단일 경로와 비교해 실제 GPU 시간에서 어떤 이득을 보이며, 기본 정책에 따라 어떻게 달라지는가?
- RQ3. 제안한 MS 기본 K-Means는 IA 전용과 MS 전용을 모두 앞설 수 있는가?
- RQ4. 제안 정책의 평균 GPU 시간 이득은 5초 구간별 결과와 p95·p99에서도 유지되는가?
- T1 경로에서는 선택기 모델마다 IA 기본·MS 기본 정책을 비교하고(RQ1·RQ2), E1 경로에서는 제안

정책을 고정 P, Centroid가 같은 IA 기본 K-Means 및 대조군인 IA Only(모든 그룹을 IA로 고정)·MS Only(모든 그룹을 MS로 고정)와 비교한다(RQ3·RQ4). T1과 E1은 모두 11개 환경에서 수행한다(3.9절).

1.4 연구의 기여

1. IA 경로와 Meshlet 경로를 공통 Material·출력 구조에 연결하고, 같은 프레임의 Mesh 그룹마다 한 경로로 한 번만 제출하는 Renderer를 구현하였다.
2. 환경별로 사전 베이크한 GPU 비용으로 환경 특화 선택기(P·C 분류기, 공간 비용 Grid, 최소 비용 Chunk)를 만들고, 각각 IA 기본·MS 기본 정책으로 실제 구동하여 11개 환경에서 비교하였다.
3. 그룹 단위 비용과 실제 혼합 비용의 차이를 검증 경로의 실제 혼합 시간으로 보정하는 MS 기본 K-Means를 제안하였다.
4. 최종 평가에서 MS 기본 K-Means가 IA 전용과 MS 전용을 모두 앞섰음을 확인하고, 파이프라인 통계·화면 동등성·국소 분석으로 그 동작을 설명하였다.

1.5 논문의 구성

2장은 관련 연구를, 3장은 하이브리드 Renderer와 선택기의 설계를, 4장은 구현과 실험 환경을 설명한다. 5장은 T1·E1 결과를 분석하고, 6장은 결론을, 7장은 후속 연구를 제시한다. 환경별 상세 자료는 부록에 싣는다.

2. 관련 연구

2.1 IA 파이프라인과 정점 재사용

IA 경로의 정점 처리 비용은 인덱스 순서에 따른 정점 재사용에 크게 좌우된다. Sander 등은 삼각형 순서를 바꿔 Post-transform Vertex Cache 효율과 Overdraw를 함께 개선하는 알고리즘을 제안하였다[4]. Kerbl 등은 현대 GPU가 고전적인 FIFO 형태의 Post-Transform Cache가 아니라 배치 단위로 정점을 재사용한다는 점을 실측으로 보이고, 이에 맞춘 최적화가 더 효과적임을 보였다[3]. meshoptimizer는 이러한 정점 캐시·fetch 최적화를 라이브러리로 제공하며[5], 본 연구의 IA 경로도 이를 적용한다. 따라서, 본 연구의 IA 대조군은 최적화되지 않은 기준선이 아니다.

2.2 Mesh Shader 파이프라인과 Meshlet

Microsoft의 Mesh Shader 명세는 IA·VS 단계를 AS와 MS로 대체하는 Programmable Geometry 파이프라인을 정의한다[1]. AS는 실행할 MS 작업 수와 payload를 정하고, MS는 Thread Group 단위로 정점과 Primitive를 직접 출력한다. Kubisch는 NVIDIA Turing 아키텍처에서 이 구조를 소개하며, Meshlet 단위 컬링과 Meshlet 크기 선택의 지침을 제시하였다[2].

Meshlet을 어떻게 만드느냐도 성능에 영향을 준다. Jensen 등은 여러 Meshlet 생성 전략을 7만~3,900만 삼각형 규모의 Mesh로 비교하여, 지역적·탐욕적 알고리즘이 더 낮은 렌더 시간을 보였다고 보고하였다[6]. 본 연구는 Meshlet 생성을 meshoptimizer에 맡기고[5], 생성된 Meshlet을 어느 파이프라인에서 그릴지에 집중한다.

2.3 클러스터 컬링과 GPU 주도 렌더링

Mesh Shader 이전에도 Compute Shader로 클러스터·삼각형 단위 컬링을 수행하는 GPU 주도 렌더링이 널리 연구되었다. Haar와 Aaltonen은 CPU 왕복 없이 GPU가 Mesh 클러스터의 가시성을 판단하고 Depth 기반 오클루전 컬링을 수행하는 파이프라인을 소개하였다[7]. Wihlidal은 Rasterization 전에 Compute로 클러스터 Normal Cone 컬링과 삼각형 단위 컬링을 수행하는 방법을 제시하였다[8]. Nanite는 클러스터 계층과 LOD, 컬링, Rasterization을 GPU에서 일괄 처리하는 가상화 Geometry 시스템으로 발전하였다[9].

이 연구들은 대체로 Scene 전체를 하나의 Geometry 경로로 처리한다. 본 연구는 컬링 기법을 확장하지 않고, 기존 IA 경로와 Meshlet 경로를 그룹별로 섞는 선택 문제를 다룬다는 점에서 다르다.

2.4 비용 예측 기반 렌더링 선택

렌더링 비용을 예측해 표현 방식을 고르는 연구는 오래 전부터 있었다. Funkhouser와 Séquin은 객체별 LOD의 비용과 이득을 추정해 목표 프레임 시간 안에서 선택하는 적응형 표시 알고리즘을 제안하였다[10]. Wimmer와 Wonka는 시점과 잠재적 가시 객체 집합으로부터 렌더링 시간을 추정하는 틀을 제시하였다[11]. 본 연구의 선택기는 같은 방향의 문제를, 상세 수준(LOD)이 아니라 Geometry 파이프라인의 선택에 적용한다. 또한, 비용을 해석적으로 모델링하지 않고, 환경마다 사전에 측정·베이크한 GPU 시간으로 그 환경 전용의 선택기를 만든다.

2.5 물리 기반 셰이딩

PBR(Physically Based Rendering)은 미세면 이론(Microfacet Theory)에 기반한 BRDF(Bidirectional Reflectance Distribution Function)로 빛과 표면의 반사를 근사한다[12]. Diffuse 모델로는 Lambert와 거칠기를 반영한 Oren-Nayar[19]가 쓰였고, Matusik 등은 100개 실제 Material의 반사를 측정한 MERL BRDF Database를 공개하였다[18]. Burley는 이 데이터를 분석해 직관적인 파라미터를 갖는 Disney BRDF를 제안하였다[13]. Specular 항의 GGX 분포는 Walter 등이[14], Masking-Shadowing 함수의 이론적 근거는 Heitz가 정리하였다[15]. Schlick은 Fresnel 항의 효율적인 근사를 제시하였고[17], Frostbite는 이 요소들을 게임 엔진에 적용하였다[16]. 본 연구는 셰이딩 모델을 제안하지 않으며, 두 경로가 공유하는 사실적 셰이딩 부하로 사용한다. 사용한 BRDF 식과 각 항의 의미는 4.7절에 상세히 기술한다.

2.6 GPU 성능 측정과 통계적 평가

Direct3D 12는 커맨드 큐 Timestamp로 GPU 구간 시간을[20], 파이프라인 통계 쿼리로 단계별 처리량을 제공한다[21]. 성능 측정은 실행 순서나 환경처럼 사소해 보이는 조건에도 편향될 수 있으므로[25], 본 연구는 모델 실행 순서를 섞고 반복 코호트를 두었다. 신뢰구간은 코호트 단위 bootstrap[26]으로, 다중 비교는 Holm[27]과 Benjamini-Yekutieli[28] 보정으로 다룬다. 본 연구의 Timestamp 기록 위치와 측정 구간은 4.9절 그림 8에 나타낸다.

3. 설계

3.1 설계 목표

1. Scene · Material · Pixel Shader · Render Target을 공유하고 Geometry 파이프라인만 변수로 둔다.
2. 같은 프레임에서 Mesh 그룹마다 IA 경로 또는 Meshlet 경로 중 하나만 실행한다.
3. Meshlet · Bounding Sphere 생성, Texture 업로드, PSO(Pipeline State Object) 구축은 전처리로 분리한다. 단, 런타임의 C 획득 · 추론 · 정렬 · PSO 바인딩 비용은 남기고 별도로 기록한다.
4. 학습 · 검증 · 시험 경로를 분리하고, 시험 전에 모델을 보존한다.

3.2 전체 구조

Renderer는 오프라인 준비와 런타임 선택 · 제출로 나뉜다(그림 1). 오프라인에서는 YAML Scene과 FBX 모델을 읽어 Sub-Mesh로 나누고, IA용 최적화 정점 · 인덱스와 MS용 Meshlet · Bounding Sphere를 모두 준비한다. 학습 경로에서 그룹별 GPU 비용을 측정해 선택 모델과 비용표를 만들고 보존한다. 런타임에는 현재 카메라로 그룹별 경로를 정하고, 각 그룹을 해당 경로로 한 번 제출한 뒤 공통 Rasterizer와 PBR Pixel Shader로 합친다.

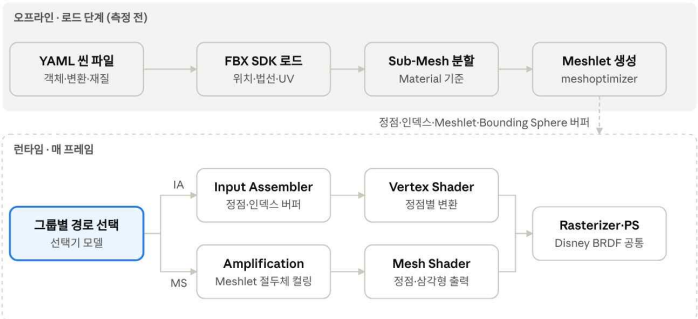


그림. 1. 하이브리드 Renderer의 전체 구조

3.3 두 Geometry 경로

IA 경로는 CPU가 정점·인덱스 버퍼를 바인딩하고 DrawIndexedInstanced를 호출한다. Input Assembler가 Primitive Assembly로 삼각형을 구성하고 Vertex Shader가 인스턴스별 월드 행렬로 정점을 변환한다. 제출된 모든 정점이 처리되며, 화면 밖 삼각형은 이후 Rasterizer 단계에서 컬링된다. IA 경로의 전처리와 실행 과정은 그림 2와 같다.



그림. 2. IA 경로의 전처리와 실행 과정

Meshlet 경로는 CPU가 DispatchMesh를 호출하면 AS 그룹이 Meshlet마다 Bounding Sphere를 절두체와 비교하고, 살아남은 Meshlet 번호를 payload에 담아 그 수만큼 MS 그룹을 실행한다. MS는 Meshlet의 정점과 삼각형을 출력한다. 즉, 화면 밖 Geometry를 정점 처리 전에 Meshlet 단위로 제거하는 대신, AS 실행과 경계 자료 읽기라는 고정 비용이 생긴다. Meshlet 경로의 전처리와 실행 과정은 그림 3과 같다.



그림. 3. Meshlet 경로의 전처리와 실행 과정

meshopt_computeMeshletBounds는 Meshlet마다 두 가지 경계 자료를 만든다(그림 4). Bounding Sphere는 Meshlet의 모든 정점을 감싸는 구(중심 c, 반지름 r)다. 절두체는 여섯 평면(Normal n, 거리 d)으로 둘러싸인 영역이므로, 구 전체가 한 평면이라도 바깥쪽에 있으면($n \cdot c + d < -r$) Meshlet의 어떤 삼각형도 화면에 보일 수 없다. 이 판정은 정점을 하나도 읽지 않고 중심과 반지름만으로 끝나며, 구는 실제 형상보다 크므로 보이지 않는 Meshlet을 남길 수는 있어도 보이는 Meshlet을 잘못 버리지는 않는다. 평면에 걸친 구도 유지된다.

Normal Cone은 Meshlet 삼각형들의 면 Normal이 모여 있는 방향 범위를 꼭짓점(Apex), 축(Axis), cutoff로 나타낸 원뿔이다. 카메라가 원뿔의 뒤쪽에 있어 dot(normalize(apex − 카메라 위치), axis) ≥ cutoff가 성립하면 Meshlet의 모든 삼각형이 카메라를 등지므로, Meshlet 단위로 후면 컬링을 할 수 있다.

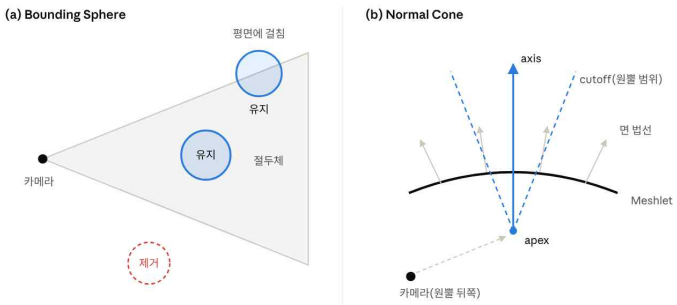


그림. 4. Bounding Sphere 절두체 판정(a)과 Normal Cone 후면 판정(b)

두 경로는 같은 출력 속성(위치, Normal, UV, MaterialID)을 공통 Pixel Shader에 넘기고 같은 Color · Depth Target에 쓴다. 따라서, 그룹마다 다른 PSO를 써도 후속 처리는 일관된다.

그림 5는 두 경로의 차이를 비교한다. IA 경로는 제출된 모든 정점을 Vertex Shader에서 변환한 뒤 Rasterizer 단계에서 화면 밖 삼각형을 컬링하고, Meshlet 경로는 AS가 정점 처리 전에 Meshlet 단위로 절두체 컬링을 수행한다. 두 경로는 Rasterizer 이후의 처리를 공유한다.

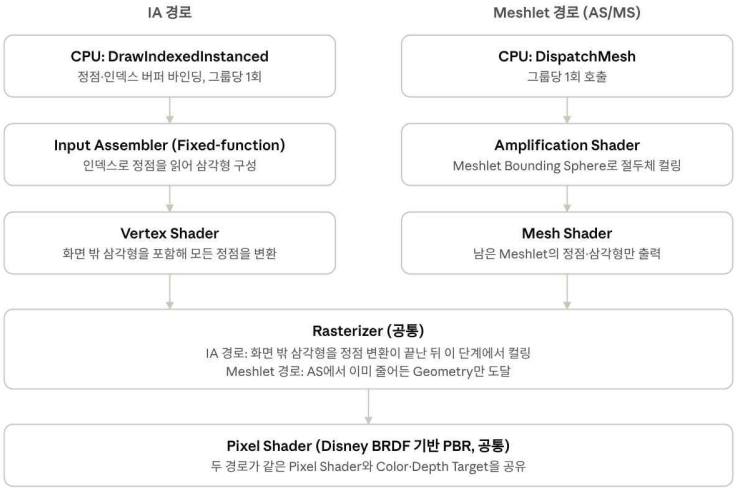


그림. 5. IA 경로와 Meshlet 경로의 처리 흐름과 컬링 위치

3.4 선택 단위: 논리 Mesh 그룹

선택 단위는 같은 모델을 공유하는 인스턴스 그룹과 Sub-Mesh의 조합인 논리 Mesh 그룹이다. 그룹 내 인스턴스는 같은 경로를 공유하며, IA는 DrawIndexedInstanced 한 번, MS는 DispatchMesh 한 번으로 모든 인스턴스를 그린다. 투명 오브젝트는 Depth 정렬을 위해 실제 드로우가 인스턴스별로 나뉘지만 새 선택 단위가 생기지는 않는다. 따라서, 논리 그룹 수와 실제 드로우 수는 다를 수 있다.

3.5 선택 특징: P와 C

그룹 g를 이루는 Meshlet · 인스턴스 집합을 M_g, Meshlet m의 Primitive 수를 p_m이라 하자. 현재 카메라에서 m의 Bounding Sphere가 절두체 판정을 통과하면 v_m은 1 아니면 0이며, P와 C는 식 1과 같다.

$$P_g = \sum_{m \in M_g} p_m, \qquad C_g = 1 - \frac{1}{P_g} \sum_{m \in M_g} p_m v_m \qquad \text{식 1}$$

P는 인스턴스를 포함한 컬링 전 삼각형 수로 그룹마다 고정되고, C는 카메라에 따라 변하는 Primitive 가중 컬링 비율이다($C = 0$은 모두 유지, $C = 1$은 모두 컬링). C는 Bounding Sphere 기준의 보수적 추정이며 픽셀 가시성이나 오클루전 비율과는 다르다. 모델 입력은 $\log_{10}(P)$와 C이다.

3.6 학습 라벨

학습 경로의 각 구도에서, 주변 Scene을 전부 IA 또는 전부 MS로 두고 그룹별 Timestamp로 GPU 시간을 측정한다. 한 블록은 8프레임 예열 후 IA·MS를 ABBA 또는 BAAB 순서로 번갈아 실행한 4프레임 묶음 8개(IA·MS 각 16프레임)이며, 이 블록을 서로 다른 프로세스에서 3번 독립적으로 반복한다. 세 블록을 $b=1,2,3$으로 번호 붙이면, b번째 블록의 시간 차이 δ_b와 세 블록의 평균 $\bar{\delta}$, 표준오차 SE는 식 2와 같다. δ_b가 양수면 MS가 빠르다.

$$\delta_b = T_{\mathrm{IA},b} - T_{\mathrm{MS},b}, \qquad \bar{\delta} = \frac{1}{3}\sum_{b=1}^{3}\delta_b, \qquad \mathrm{SE} = \frac{s_\delta}{\sqrt{3}} \qquad \text{식 2}$$

여기서 s_δ는 세 δ_b의 표준편차다. 지도 비용 g는 식 3과 같이 $\bar{\delta}$에 임계값 2SE의 소프트 임계화(Soft Thresholding)[32]를 적용해 불확실성을 반영한 값이다. 소프트 임계화는 잡음 수준에 비례하는 임계값으로 잡음에 묻힌 작은 값을 없애는 방법이며, 본 연구는 잡음 수준으로 반복 측정의 SE를 쓴다.

$$g = S_{2\mathrm{SE}}(\bar{\delta}) = \mathrm{sign}(\bar{\delta}) \cdot \max(|\bar{\delta}| - 2\mathrm{SE},\ 0) \qquad \text{식 3}$$

따라서, g는 $\bar{\delta}$와 2SE의 크기에 따라 식 4의 세 경우로 나뉜다. $|\bar{\delta}|$가 2SE 이하로 반복 측정의 흔들림과 구분되지 않으면 불확실($g=0$)로 두고, 그보다 크면 차이에서 2SE를 뺀 보수적인 크기만 남긴다.

$$g = \begin{cases} \bar{\delta} - 2\mathrm{SE}, & \bar{\delta} > 2\mathrm{SE} \quad (\mathrm{MS}\ 우세) \\ 0, & |\bar{\delta}| \le 2\mathrm{SE} \quad (불확실) \\ \bar{\delta} + 2\mathrm{SE}, & \bar{\delta} < -2\mathrm{SE} \quad (\mathrm{IA}\ 우세) \end{cases} \qquad \text{식 4}$$

선택기 학습에서는 상태를 맞혔는지가 아니라 잘못 고른 상태에서 잃는 시간을 손실로 쓴다. 상태의 우세 경로를 a^*, 선택기가 고른 경로를 a라 하면 손실은 식 5와 같다.

$$\ell(a) = |g| \cdot \mathbf{1}[a \neq a^*], \qquad a^* = \begin{cases} \mathrm{MS}, & g > 0 \\ \mathrm{IA}, & g < 0 \end{cases} \qquad \text{식 5}$$

즉, 시간 차이가 큰 상태를 틀릴수록 손실이 커지고, $g=0$인 상태는 손실에 기여하지 않는다.

11개 환경의 학습·검증 구도 22,155개에서 그룹 상태 4,397,704개를 취득했고, g에 따라 나누면 IA 우세 3,069,258개, MS 우세 369,297개, 불확실 959,149개였다.

3.7 선택 정책

표 1은 비교한 선택 정책의 입력, 선택 규칙과 후보 설정을 정리한 것이다.

표 1. 실험 대상 선택기 모델

정책	입력	선택 규칙	후보·설정
IA Only / MS Only	없음	모든 그룹을 한 경로로 고정	대조군
고정 P	P	$P \ge$ *Threshold*면 MS	환경별 *Threshold*
다항 로지스틱	$\log_{10}(P)$, C	다항 *score* > 0면 MS	차수 2~6×L2 3종 = 9개
LUT	$\log_{10}(P)$, C	C knot 별 $\log_{10}(P)$ *Threshold*를 선형 보간	knot 5·9·17 × 평활 3종 = 9개
공간 비용 Grid	카메라 위치·	베이크 비용을 보간해 작은 쪽	1·2.5·5·10 m

	시선		
최소 비용 Chunk	카메라 위치	셀별로 앞서 만든 Expert 하나를 실행	1·2.5·5·10 m
IA 기본 K-Means	$\log_{10}(P)$, C	최근접 Centroid에 배정한 동작	Centroid 2·4·8·16 = 4개
MS 기본 K-Means	$\log_{10}(P)$, C	근거가 충분한 군집만 IA	λ 3종 × τ 4종

- 다항 로지스틱 회귀. 정규화한 $x = \log_{10}(P)$, $y = C$에 대해 $a + b \le d$인 모든 항 $\beta_{ab} \cdot x^a \cdot y^b$를 더한 *score*를 쓴다. 비용 합 $A \cdot B$로 가중한 logistic 손실에 L2를 더해 Newton법으로 학습한다.
- LUT. C knot마다 $\log_{10}(P)$ *Threshold*를 두고, 잘못 고른 비용 합과 인접 *Threshold* 차의 평활 페널티를 동적 계획법으로 최소화한다.
- K-Means. 정규화한 ($\log_{10}(P)$, C) 공간에서 가중 k-means++ 초기화[29]와 가중 Lloyd 반복으로 Centroid를 구하고, 세 번의 재시작 중 빈도 가중 inertia가 가장 작은 결과를 쓴다. 이어, 군집 안의 부호 있는 비용의 합이 양수면 MS, 아니면 IA 동작을 부여한다. 근거가 부족한 군집도 IA로 남는다는 점에서, 본 논문은 이 모델을 IA 기본 K-Means라 하고, 3.8절의 MS 기본 K-Means와 함께 비교한다. 군집 형성의 가중치는 표본 빈도이며 GPU 비용은 동작 부여에만 쓴다.
- 공간 비용 Grid. Grid Point마다 여섯 축 방향의 그룹별 IA · MS 비용을 베이크하고, 현재 위치의 8꼭짓점 trilinear 가중치와 시선의 방향 가중치를 곱해 보간한다.
- 최소 비용 Chunk. 공간 비용 Grid 간격의 셀마다 앞서 만든 Expert(Logistic, LUT, IA 기본 K-Means, 해당 간격 공간 비용 Grid중 IA 대비 2SE를 넘는 근거가 있는 것을 고르고, 근거가 없으면 IA로 둔다.

K-Means는 3.6절에서 묶은 특징점 i를 입력으로 쓰며, 묶인 원시 표본 수 w_i를 가중치로 둔다. 두 축은 단위와 범위가 다르므로 w_i로 가중한 평균 μ와 표준편차 σ로 식 6과 같이 정규화한다.

$$\boldsymbol{x}_i = \left(\frac{\log_{10} P_i - \mu_P}{\sigma_P},\ \frac{C_i - \mu_C}{\sigma_C} \right) \qquad \text{식 6}$$

3.8 기본 정책: IA 기본과 MS 기본

K-Means 선택기는 근거가 부족한 군집을 어느 경로로 둘지에 따라 두 기본 정책으로 나뉜다. IA 기본 K-Means(3.7절)는 이를 IA로 두고, MS 기본 K-Means는 기본 경로를 MS로 두어 원래 IA였던 군집 중 식 7의 조건을 만족하는 것만 IA로 남긴다.

$$\mu_k > 0 \quad \text{and} \quad \mu_k - \lambda \cdot \mathrm{SE}_k > \tau, \qquad \lambda \in \{0,1,2\},\ \tau \in \{0,1,2,4\}\,\mu s \qquad \text{식 7}$$

그룹 단위 라벨은 IA 배정을 실제 혼합 비용보다 넓게 잡을 수 있으므로, MS 기본 정책은 IA 허용 여부를 실제 혼합 시간으로 확인한다. 식 7에서 μ_k와 $\mathrm{SE_k}$는 3.6절의 $\bar{\delta}$, SE와 같은 방식으로 세 블록에서 구한 군집 k의 평균 IA 절감량($T_{\mathrm{MS}} - \mathrm{T_{IA}}$)과 그 표준오차다. λ는 $\mathrm{SE_k}$에 곱하는 보수성 계수로, 클수록 측정의 흔들림을 넘는 확실한 이득만 인정한다. τ는 IA를 허용하는 데 필요한 최소 절감량(μs)으로, PSO 전환 비용 같은 물리량이 아니라 V에서 보정하는 경험적 *Threshold*다.

λ와 τ의 조합 12개(3 × 4) 가운데 환경마다 하나를 검증 경로(V)에서 다음 순서로 고른다.

1. IA로 허용되는 군집이 서로 다른 후보를 최대 3개 고른다.

2. 각 후보를 V에서 실제로 혼합 실행해 GPU 시간을 MS Only와 비교한다.
3. MS보다 빠름이 확인된 후보(짝차이 95% 하한 > 0, Holm 단측 5% 검정[27]) 중 가장 보수적인 것을 고른다.
4. 통과한 후보가 없으면 그 환경은 항상 MS로 둔다.

런타임에는 현재 $P \cdot C$로 최근접 Centroid를 찾아 보존된 동작(IA 또는 MS)을 조회하기만 한다. 그림 6은 이 오프라인 결정과 런타임 조회 과정을 나타낸다. MS 기본 K-Means가 본 논문의 제안 정책이며, IA 기본 K-Means는 비교 대상이다. 다른 모델도 같은 방식으로 두 기본 정책을 갖는다. Logistic · LUT는 $score$에, 공간 비용 Grid는 IA · MS 비용 차에 margin을 두고, 최소 비용 Chunk는 셀마다 $\lambda \cdot \tau$를 적용하여, 근거가 충분할 때만 기본 경로의 반대 경로를 허용한다.

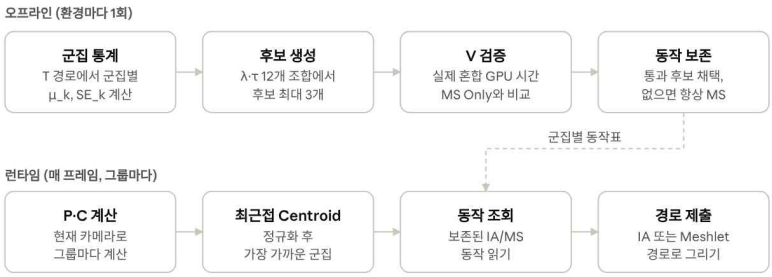


그림. 6. MS 기본 K−Means의 오프라인 결정(위)과 런타임 조회(아래)

그림 7은 Studio_D에서 같은 Centroid · 정규화를 쓴 채 군집 동작만 바꾼 결과다. IA 기본 K-Means에서 넓게 퍼져 있던 IA 영역이 MS 기본 정책에서는 근거가 충분한 일부 영역으로 좁아졌다.

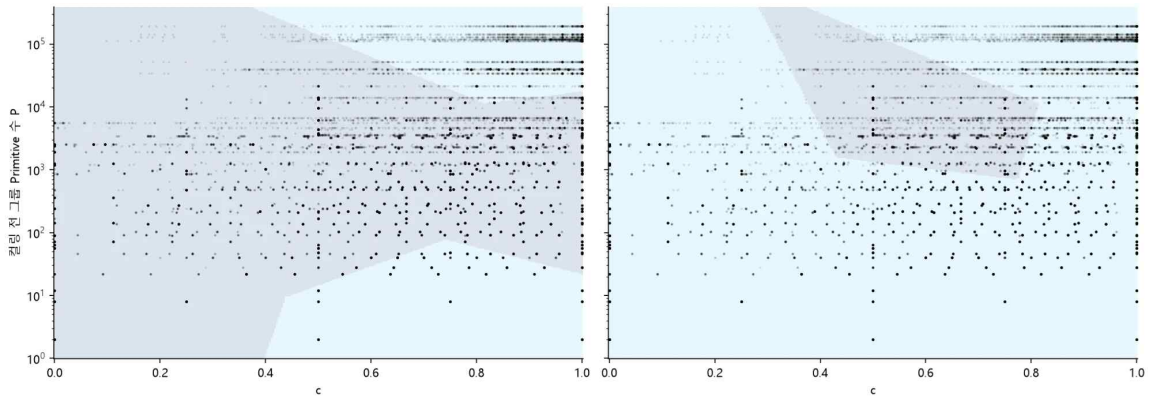


그림. 7. Studio_D의 결정 영역. IA 기본 K−Means(왼쪽), MS 기본 K−Means(오른쪽) / 회색은 IA, 청색은 MS, 검은 점은 E1 입력 분포 / 가로축은 C, 세로축은 컬링 전 그룹 Primitive 수 $\log_{10}(P)$.

3.9 자료 분할과 보존

표 2는 학습 · 검증 · 시험 경로 자료의 역할을 정리한 것이다.

표 2. 자료별 역할

자료	역할
T(학습)	모델 계수와 군집 손익 통계
V(검증)	모델별 후보 선정(그룹 비용 손실이 가장 작은 후보), 고정 P의 $Threshold$ 선정, MS 기

	본 정책의 $\lambda \cdot \tau$를 실제 혼합 GPU 시간으로 검증
T1(대표 시험)	선택기 11개 모델의 IA·MS 기본 정책을 포함한 26개 모델의 동일 경로 비교
E1(최종 시험)	제안 정책을 고정 P, IA 기본 K−Means, 대조군(IA Only·MS Only)과 비교

T, V, T1, E1은 환경을 나눈 것이 아니라 각 환경 안의 카메라 경로를 나눈 것이다. 따라서, 11개 환경(표 9) 모두가 네 경로를 각각 갖고, T1과 E1 측정에는 11개 환경이 모두 포함된다. 모든 모델은 환경마다 따로 학습·보존되며, E1 경로는 학습에 쓰지 않은 경로다.

4. 구현

4.1 개발 기반과 실행 환경

Renderer는 Scene·인스턴스 그룹, 두 Geometry 경로, 그룹별 선택, 공통 Material, 프레임 자원 관리, 계측으로 구성된다. 조명은 단일 색상 Render Target에 직접 계산하는 Forward 방식이다. 개발 및 실행 환경은 표 3과 같다.

표 3. 개발 및 실행 환경

항목	설정
GPU	NVIDIA GeForce RTX 4070 Laptop GPU, 드라이버 560.94
개발 도구	Visual Studio(v143), Windows SDK 10.0.26100.0, Agility SDK 1.618.5
셰이더 컴파일	DXC(IDxcCompiler3), VS·PS·AS 6.6, MS 6.7
라이브러리	Autodesk FBX SDK, meshoptimizer, yaml−cpp, stb_image, ImGui
렌더 해상도	1280×720
Swap Chain	FLIP_DISCARD, Back Buffer 2개, tearing 허용, Present interval 0
Command Queue	DIRECT 큐 1개
카메라	세로 화각 45° near 10 cm, far 1,000,000 cm

IA와 MS의 혼합은 비동기 Compute 큐가 아니라 같은 ID3D12GraphicsCommandList 안의 서로 다른 Draw와 Mesh Dispatch다.

4.2 Scene 구성: YAML과 FBX

YAML Scene 시스템. 맵은 코드가 아니라 YAML 파일로 관리한다. 로더는 scene.yaml의 Objects 배열을 순회하며 각 오브젝트의 Name, UseMeshlet, Position, Rotation, Scale, MeshPath, ShaderPath, Roughness, Metallic, WorldMatrix를 읽는다. Material은 별도의 textured_materials.yaml이 담당한다. 이 파일의 Materials는 Material마다 Texture 4장의 경로·BaseColor·계수·UV 변환·블렌드 정보를, Bindings는 객체·슬롯과 Material의 대응을 정의한다. Scene 자료의 계층은 표 4와 같다.

표 4. Scene 자료 계층

계층	파일	역할
Scene 배치	scene.yaml	객체의 Mesh·셰이더 경로와 변환
Geometry	FBX	삼각형과 정점 속성
Material	textured_materials.yaml	Texture, UV, Material 계수, 블렌드
Polygon Link	textured_polygon_slots.csv	삼각형과 Material 슬롯의 대응

코드 1과 코드 2는 StudioB의 실제 설정 파일에서 발췌한 예시다. 코드 1의 두 오브젝트는 같은 Mesh를 쓰고 WorldMatrix만 다르므로, 로딩 후 같은 인스턴스 그룹으로 묶인다. WorldMatrix가 주어지면 Position·Rotation·Scale보다 우선한다. 코드 2에서는 Materials 배열의 순서가 MaterialID이고, Bindings는 오브젝트 이름과 슬롯으로 MaterialID를 연결한다. Textures는 순서대로 BaseColor, Normal,

Roughness, Metallic이며, Shader는 Roughness의 G 채널과 Metallic의 B 채널을 읽는다.

코드 1. StudioB의 scene.yaml 발췌

```
Objects:
  - Name: "StudioB_0"
    Position: [0, 0, 0]
    Rotation: [0, 0, 0]
    Scale: [1, 1, 1]
    MeshPath: "…/meshes/SM_ModularKit_01_Floor_20x20_48f18c0dbc.fbx"
    ShaderPath: "Shaders/main.hlsl"
    Roughness: 0.7
    Metallic: 0.0
    WorldMatrix: [0.0, 0.0, 1.0, 0.0, −1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0]
  - Name: "StudioB_1"
    UseMeshlet: true
    Position: [0, 0, 0]
    Rotation: [0, 0, 0]
    Scale: [1, 1, 1]
    MeshPath: "…/meshes/SM_ModularKit_01_Floor_20x20_48f18c0dbc.fbx"
    ShaderPath: "Shaders/main.hlsl"
    Roughness: 0.7
    Metallic: 0.0
    WorldMatrix: [0.0, 0.0, 1.0, 0.0, −1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 600.0, 0.0, 0.0, 1.0]
```

코드 2. StudioB의 textured_materials.yaml 발췌

```
Materials:
  - Source: "…/MI_Black_Glossy_Pattern"
    Textures: ["…/tx_77efedce….png", "…/tx_0d0d4c1e….png",
               "…/tx_f855e078….png", "…/tx_f855e078….png"]
    BaseColor: [1, 1, 1, 1]
    Factors: [1, 1, 1, 0]
    UVU:
      - [1.0, −0.0, 0, 0]
      - [1.0, −0.0, 0, 0]
      - [1.0, −0.0, 0, 0]
      - [1.0, −0.0, 0, 0]
    UVV:
      - [0.0, 1.0, 0, 0]
      - [0.0, 1.0, 0, 0]
      - [0.0, 1.0, 0, 0]
      - [0.0, 1.0, 0, 0]
    TexCoordSets: [0, 0, 0, 0]
    Visual:
      blend_mode: 0
      unlit: false
      effect_kind: 0
      cutoff_or_fade_m: 0.5
```

```
Bindings:
  - Object: "StudioB_0"
    Slot: 0
    MaterialID: 0
    ExpectedOriginalMaterial: "MI_Wood_Planks"
  - Object: "StudioB_1"
    Slot: 0
    MaterialID: 0
    ExpectedOriginalMaterial: "MI_Wood_Planks"

    Metallic: 0.0
    WorldMatrix: [0.0, 0.0, 1.0, 0.0, -1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 600.0, 0.0, 0.0, 1.0]
```

IA로 시작하는 그룹도 MS로 전환할 수 있어야 하므로 모든 그룹이 Meshlet 자원을 미리 보유한다.

FBX 로드. Autodesk FBX SDK의 importer로 Scene을 읽고 노드를 재귀적으로 순회하며 Mesh 노드를 처리한다. 노드 변환과 geometric transform을 합성해 위치를, 그 역전치 행렬로 Normal을 변환한다.

4.3 IA 경로의 전처리

원본 정점은 meshopt_generateVertexRemap으로 속성이 완전히 같은 중복 정점을 합친다. IA 전용 사본에는 추가로 meshopt_optimizeVertexCache와 meshopt_optimizeVertexFetch를 적용하고, MS는 remap한 원본 정점과 Meshlet 인덱스를 사용한다.

4.4 Meshlet 구축: meshoptimizer의 입력과 출력

Meshlet은 Sub-Mesh마다 로드 단계에서 한 번 만든다. 상한은 정점 64개, 삼각형 124개, cone weight는 0이다[2]. 사용한 meshoptimizer 함수의 입력과 출력은 표 5와 같다.

표 5. meshoptimizer 함수의 입력과 출력

단계	함수
1. 상한 계산	meshopt_buildMeshletsBound
2. Meshlet 분할	meshopt_buildMeshlets
3. 경계 계산	meshopt_computeMeshletBounds

Meshlet 레코드는 VertCount, VertOffset, PrimCount, PrimOffset 네 개의 uint(16 B)이다. Local Index 배열은 삼각형마다 Meshlet 내부 정점 번호 3개를 1바이트씩 저장하고 끝을 4바이트 경계에 맞춘다. 두 경로에서 CPU가 준비해 GPU에 전달하는 자료의 메모리 형식은 표 6과 같다.

표 6. CPU 자료 메모리 형식

자료	크기	내용
Vertex	52 B	Position, Normal, UV0, Material 슬롯, UV1, UV2
Meshlet	16 B	정점·삼각형 개수와 오프셋
MeshletBounds	48 B	Bounding Sphere와 Normal Cone
InstanceData	64 B	전치 월드 행렬
ObjectConstants	256 B	ViewProj, 카메라, 조명, 인스턴스 오프셋, 절두체 평면 6개
Material 레코드	192 B	Texture Descriptor 4개, BaseColor, 계수, UV 변환, 플래그

AS payload	132 B	AS가 MS에 넘기는 인스턴스 번호 1개와 생존 Meshlet 번호 최대 32개

4.5 AS와 MS 셰이더

CPU는 MS 그룹에 대해 DispatchMesh를 호출한다. AS는 32-thread Thread Group이며, 그룹 ID의 y가 인스턴스를, x와 스레드 번호가 Meshlet을 정한다. 각 스레드는 자기 Meshlet의 Bounding Sphere 중심을 월드 좌표 c_w로 변환하고, 반지름에 월드 행렬 3축 길이의 최댓값을 곱해 r_w로 키운 뒤, 식 8의 조건이 하나라도 참이면 그 Meshlet을 제거한다.

$$\boldsymbol{n}_i \cdot \boldsymbol{c}_w + d_i < -r_w, \qquad i = 1,\ldots,6 \qquad \text{식 8}$$

여기서 n_i와 d_i는 표 6의 ObjectConstants에 담긴 절두체 평면 i의 평면 방정식 $\boldsymbol{n}_i \cdot \boldsymbol{c}_w + d_i = 0$의 Normal과 상수항이다. Normal은 절두체 안쪽을 향하는 단위 벡터이므로, 좌변은 구 중심과 평면 사이의 부호 있는 거리다. r_w는 위에서 월드 좌표로 키운 반지름이다. 따라서, 어느 한 평면에서라도 이 거리가 $-r_w$보다 작으면 구 전체가 절두체 밖에 있다. 살아남은 Meshlet 번호는 InterlockedAdd로 payload에 모이며, AS는 그 수만큼 MS 그룹을 실행한다. MS는 128-thread Thread Group이고, 선언된 출력 크기는 정점 64개 · 삼각형 128개다. SetMeshOutputCounts 호출 후 각 스레드는 Unique Vertex Index로 원본 정점을 읽어 변환 · 출력하고, Local Index 3개로 삼각형을 출력한다.

4.6 리소스 바인딩

두 경로는 하나의 Root Signature를 공유한다[22]. Root Signature의 구성은 표 7과 같다.

표 7. Root Signature 구성

레지스터	자원	형태
b0	ObjectConstants	Root CBV
t0 ~t3	Vertex, Meshlet, Unique Vertex Index, Local Index Array	Descriptor Table
t8	MeshletBounds	Root SRV
t9	InstanceData	Root SRV
t10	Material 레코드(192 B)	Root SRV
t11	인스턴스·슬롯별 Material 번호	Root SRV
s0	Sampler	Sampler

4.7 Material과 셰이딩

Texture와 Normal. Texture는 stb_image로 RGBA8에 디코딩해 mip 1단계의 R8G8B8A8_UNORM 자원으로 올린다. Albedo는 셰이더에서 sRGB 변환 함수로 선형화하고, Roughness는 G 채널, Metallic는 B 채널에 Material 계수를 곱해 얻는다.

BRDF PBR. 셰이딩 모델은 BRDF Explorer[30]에서 Lambert · Blinn-Phong, Oren-Nayar[19], Disney[13] 모델을 MERL BRDF Database[18]의 실측 Material과 비교해 골랐다. Lambert는 가장자리가 항상 어둡고, Oren-Nayar는 거칠기를 반영하지만 grazing retro-reflection이 약했다. Disney diffuse는 거칠기에 따라 가장자리 반사율을 0.5~2.5배로 조절해 실측에 가장 가까웠고, Metallic-Roughness Texture를 그대로 쓸 수 있었다.

직접 조명 식. 최종 Pixel Shader의 BRDF f는 식 9와 같이 Disney diffuse 항[13]과 specular 항

$F \cdot D \cdot V$의 합이다. F는 Schlick Fresnel[17], D는 GGX 분포[14], V는 Masking-Shadowing 함수 G를 분모 $4(\boldsymbol{n}\cdot\boldsymbol{l})(\boldsymbol{n}\cdot\boldsymbol{v})$와 묶은 높이 상관 Smith GGX 가시성 항이다[15][16][31]. $\boldsymbol{l}$은 광원 방향, $\boldsymbol{v}$는 시선, $\boldsymbol{n}$은 Normal, $\boldsymbol{h}=normalize(\boldsymbol{l}+\boldsymbol{v})$는 하프 벡터, r은 roughness, m은 metallic, $\alpha=r^2$이다. F_{D90}은 grazing 각도의 diffuse 반사율로 거칠수록 가장자리를 밝게 하고, F_0은 수직 입사 반사율(비금속 0.04, 금속 baseColor)이다. 금속은 diffuse 반사가 없으므로 c_{diff}에 $(1-m)$을 곱한다. 각 항은 식 10, 11과 같다.

$$f(\boldsymbol{l},\boldsymbol{v})=\frac{c_{\mathrm{diff}}}{\pi}\big(1+(F_{D90}-1)(1-\boldsymbol{n}\cdot\boldsymbol{l})^5\big)\big(1+(F_{D90}-1)(1-\boldsymbol{n}\cdot\boldsymbol{v})^5\big)+F\cdot D\cdot V \qquad \text{식 9}$$

$$F_{D90}=0.5+2r(\boldsymbol{l}\cdot\boldsymbol{h})^2,\quad F=F_0+(1-F_0)(1-\boldsymbol{v}\cdot\boldsymbol{h})^5,\quad D=\frac{\alpha^2}{\pi\big((\boldsymbol{n}\cdot\boldsymbol{h})^2(\alpha^2-1)+1\big)^2} \qquad \text{식 10}$$

$$c_{\mathrm{diff}}=baseColor\,(1-m),\qquad F_0=lerp(0.04,\ baseColor,\ m) \qquad \text{식 11}$$

V는 식 12의 Smith GGX로 계산한다. $n_l=saturate(\boldsymbol{n}\cdot\boldsymbol{l})$, $n_v=|n\cdot v|+10^{-5}$이다.

$$S=n_l\sqrt{\alpha^2+(1-\alpha^2)n_v^2}+n_v\sqrt{\alpha^2+(1-\alpha^2)n_l^2},\qquad V=\begin{cases}\dfrac{1}{2S}, & S>0\\ 0, & S\le 0\end{cases} \qquad \text{식 12}$$

4.8 PSO와 그리기 순서

IA와 MS 각각 opaque · alpha · additive용 PSO를 두어 총 6종을 미리 구축한다[24]. IA PSO는 Input Layout과 VS를, MS PSO는 AS와 MS를 쓰며 Pixel Shader · Root Signature · Render Target 형식은 같다. 각 PSO의 구성은 표 8과 같다.

표 8. PSO 구성

PSO	Color Blend	Depth Write
IA·MS opaque	Off	On
IA·MS alpha	SrcAlpha / InvSrcAlpha	Off
IA·MS additive	SrcAlpha / One	Off

4.9 프레임 구조와 GPU 계측

FrameResource 3개가 Command Allocator, Constant Buffer, Timestamp Readback, Fence 값을 묶어 순환하고, Back Buffer 2개는 별도로 순환한다. 해당 슬롯의 Fence 완료를 확인한 뒤에만 Constant Buffer와 Allocator를 재사용하며[23], IA↔MS 전환 때마다 별도 대기를 넣지 않는다. 한 프레임의 실행 순서는 다음과 같다.

1. 다음 FrameResource를 고르고 필요하면 Fence 완료를 기다린다.
2. 카메라 · 절두체 평면 · C · 그룹 선택을 갱신하고 Constant Buffer를 쓴다.
3. Clear 후 불투명 목록을 그룹 선택에 따라 IA 또는 MS로 기록하고, Depth 복사 후 투명 목록을 기록한다.
4. Timestamp와 통계 쿼리를 resolve하고 제출 · Present · signal한다.
5. 같은 슬롯을 다시 쓸 때 완료된 쿼리 결과를 원래 프레임에 연결한다.

ID3D12GraphicsCommandList에는 Timestamp 4개를 기록한다. $q2$는 Clear 직전, $q0$은 Draw 시작, $q1$은 Draw 종료, $q3$은 출력 관련 명령 뒤이며, 차이를 큐의 Timestamp Frequency로 나눠 시간으로 바꾼다[20]. 주 지표는 GPU Scene 시간($q1-q2$)이며 Clear와 모든 Scene Draw를 포함하고 CPU 준비·Present 대기는 포함하지 않는다. 또한, 파이프라인 통계 쿼리[21]로 CInvocations·CPrimitives와 IA·VS·AS·MS·PS 실행 수를 수집했다. 이와 함께, 프레임마다 실제 IA·MS 드로우 수, 계획·실행 배정 해시, 그리기 순서 해시를 기록하여 선택한 대로 실행되었는지 검증한다.

그림 8은 한 프레임의 ID3D12GraphicsCommandList에서 네 Timestamp가 기록되는 위치와 각 측정 구간을 보여 준다. 주 지표인 GPU Scene 시간은 Clear 직전($q2$)부터 마지막 Scene Draw 종료($q1$)까지이며, 출력 전환($q1$~$q3$)과 CPU 준비·Present 대기는 포함하지 않는다.

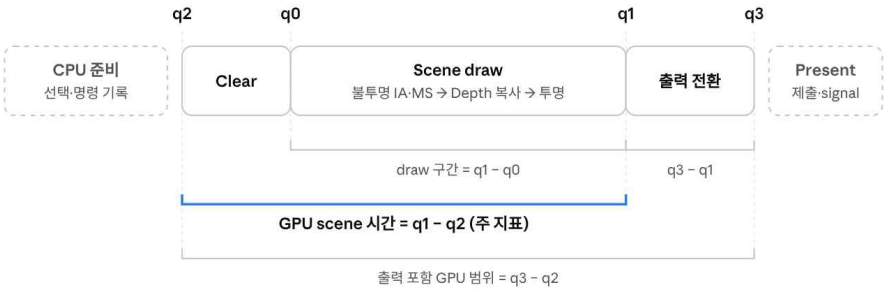


그림. 8. 한 프레임의 GPU 명령과 Timestamp 기록 위치

4.10 실험 환경과 카메라 경로

카메라는 승인된 통행 공간을 보행 최대 1.4 m/s, 회전 최대 90°/s로 순회하며 30 Hz 구도열을 끝까지 재생한다. 환경별 규모와 E1 경로는 표 9, 참조 화면과 경로 조감도는 부록 E와 같다.

표 9. 실험 환경의 규모와 E1 경로 (T1·E1 공통 11개 환경, P 내림차순)

환경	자산 묶음	논리 그룹	컬링 전 P	경로 길이(m)	구도 수
LV_Soul_Slum	Soul: City	529	5,110,300	141.3	3,986
Highrise	Shooter Game	390	1,894,664	957.6	26,580
Studio_D	Virtual Studio	68	995,624	107.5	3,243
StudioC	Virtual Studio	67	706,919	69.3	2,064
LV_Soul_Cave	Soul: Cave	129	683,169	128.3	4,312
SunTemple	Sun Temple	51	618,599	153.5	4,473
Platformer_StreetSection	Platformer Game	191	573,783	362.3	8,953
TrackerlessStudio	Virtual Studio	54	504,020	101.2	2,891
StudioA	Virtual Studio	98	444,163	96.2	3,075
StudioB	Virtual Studio	61	380,987	71.2	2,142
Sanctuary	Shooter Game	13	108,775	694.3	19,926

5. 결과

5.1 평가 방법

표 10은 T1과 E1의 측정 조건이다.

표 10. T1·E1 측정 조건

항목	T1 선택기 모델 비교	E1 최종 평가
환경	11개 전체(표 9)	11개 전체(표 9)
비교 모델	26개(IA, MS, 고정 P, 11개 모델 × IA·MS 기본, MS 기본 K-Means)	5개(IA, MS, 고정 P, IA 기본 K-Means, MS 기본 K-Means)
반복(코호트)	8	8
카메라 구도	80,938	81,645
측정 프레임	16,835,104	3,265,800

각 모델은 같은 전체 경로의 모든 30 Hz 구도를 끝까지 재생하며, 모델 실행 순서는 사전 seed로 섞고 뒤집었다. 주 지표는 GPU Scene 시간이다. 반복별 경로 합계를 짝지어 비교하고, 95% 신뢰구간은 환경 안에서 코호트를 재표집하는 paired bootstrap[26]으로 구했다. 이하 감소율은 양수가 개선이다.

5.2 두 경로의 화면 동등성

성능 비교의 전제는 두 경로가 같은 화면을 만드는 것이다. 환경마다 한 구도를 같은 카메라·Scene·셰이더로 IA Only와 MS Only에서 촬영해 비교하였다(1280×720, 921,600픽셀). 11개 구도 모두 Depth는 모든 픽셀에서 bit 단위로 같았고 Geometry coverage 차이도 0이었다. RGB는 일부 구도에서 작은 차이가 있었고, 8단계를 넘는 픽셀은 최대 806개(화면의 0.0875%)였다(표 11).

표 11. 동일 구도 IA/MS 화면 차이

환경	RGB 평균 절대차(0~255)	RGB 최대치	RGB 8 초과 픽셀	Depth 8 초과 픽셀
LV_Soul_Slum	0.035130	187	806	0
Highrise	0.017057	65	87	0
TrackerlessStudio	0.003041	76	15	0
StudioA	0.000495	4	0	0
Platformer_StreetSection	0.000391	93	8	0
Studio_D	0.000245	200	2	0
StudioB	0.000075	12	1	0
LV_Soul_Cave	0.000037	12	2	0
SunTemple	0	0	0	0
StudioC	0	0	0	0
Sanctuary	0	0	0	0

그림 9는 RGB 차이가 가장 컸던 구도의 두 화면이다.



그림. 9. RGB 차이가 가장 컸던 LV_Soul_Slum 구도의 IA Only(왼쪽)와 MS Only(오른쪽)

5.3 Geometry 처리량: 파이프라인 통계

같은 구도열과 보존 모델로 파이프라인 통계 쿼리[21]를 수집하였다(T1 1,133,132프레임, E1 408,225프레임). CInvocations는 Rasterizer에 들어간 Primitive 수, CPrimitives는 Rasterizer 단계의 컬링 뒤 실제로 렌더링된 Primitive 수다. E1의 결과는 표 12와 같고, T1의 결과는 부록 표 B1에 싣는다.

표 12. E1 프레임당 평균 파이프라인 통계

방법	CInvocations	IA 대비 감소	CPrimitives	Meshlet 경로 Primitive	PS 실행
IA	1,127,736.7	0.00%	170,080.0	0	3,480,416
MS	200,082.6	82.26%	170,080.0	200,082.6	3,548,168
고정 P	638,432.8	43.39%	170,080.0	96,488.3	3,512,960
IA 기본 K-Means	635,829.2	43.62%	170,080.0	81,408.0	3,488,515
MS 기본 K-Means	248,809.6	77.94%	170,080.0	180,766.5	3,480,500

AS 컬링은 Rasterizer 입력을 IA 대비 82.26% 줄였지만, Rasterizer 단계의 컬링 뒤 Primitive 수는 세 방법이 사실상 같았다. 즉, AS는 최종 화면 Geometry를 바꾸지 않고, Rasterizer가 어차피 컬링할 Geometry를 앞 단계에서 미리 컬링한다. 주목할 점은 MS 기본 K-Means가 MS Only보다 Rasterizer 입력을 24.35% 더 보내는데도 GPU 시간은 3.01% 짧았다는 것이다(5.5절). 따라서, Primitive 수만으로 가장 빠른 방법을 정할 수 없다.

5.4 T1: 선택기 모델과 기본 정책의 비교

T1에서는 선택기 11개 모델(Logistic, LUT, K-Means, 공간 비용 Grid 4종, 최소 비용 Chunk 4종)마다 IA 기본·MS 기본 정책을 두고, IA Only·MS Only·고정 P·제안 모델을 더한 26개 모델을 같은 경로에서 8회씩 측정하였다(모델당 647,504프레임). 표 13은 모델별 결과이며, 전체 경로 합계의 감소율과 95% 구간은 그림 10과 같다. 대조군에서는 MS Only가 IA Only보다 13.94% 빨랐고, 고정 P는 MS Only보다 4.01% 느렸다.

표 13. T1 모델별 기본 정책의 MS Only 대비 GPU Scene 시간 감소율

모델	IA 기본	MS 기본	MS 기본의 IA 기본 대비	MS 기본 실제 혼합 환경
Logistic	−4.28%	−0.28%	+3.83%	1
LUT	−5.05%	+0.07%	+4.87%	0
K-Means	−1.96%	−0.05%	+1.87%	2
공간 비용 Grid 1 m	−3.79%	+0.13%	+3.77%	3
공간 비용 Grid 2.5 m	−2.23%	+0.95%	+3.11%	3
공간 비용 Grid 5 m	−2.56%	+0.91%	+3.39%	3
공간 비용 Grid 10 m	−2.46%	+0.83%	+3.21%	2
최소 비용 Chunk 1 m	−8.05%	+0.73%	+8.12%	2
최소 비용 Chunk 2.5 m	−6.71%	+0.67%	+6.91%	2
최소 비용 Chunk 5 m	−11.10%	−0.43%	+9.61%	2
최소 비용 Chunk 10 m	−4.63%	+0.11%	+4.53%	0

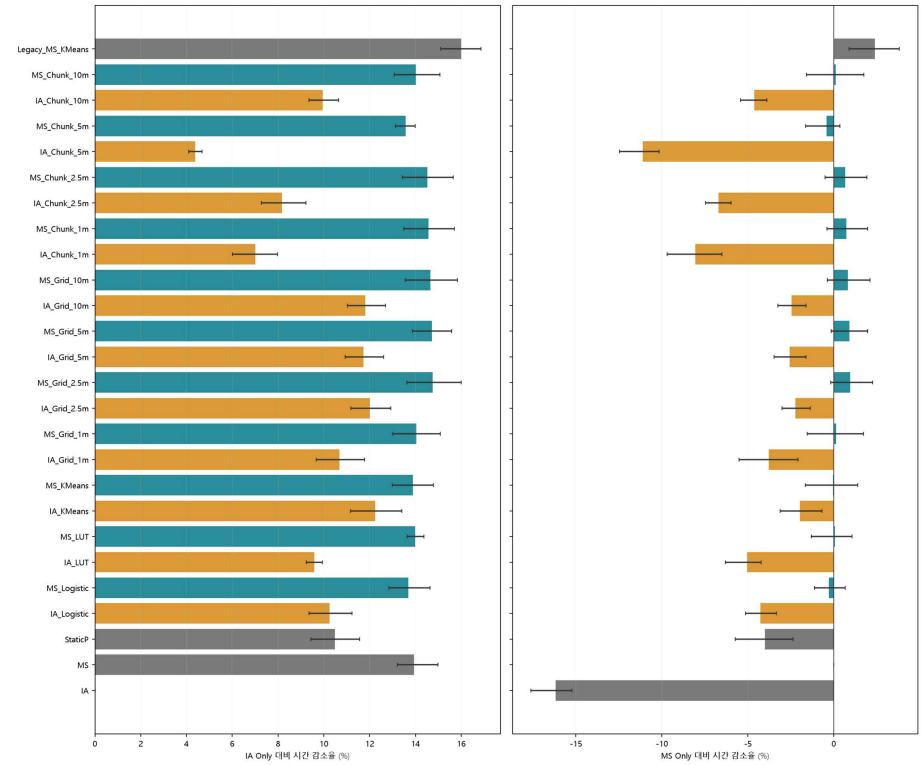


그림. 10. T1 26개 모델의 IA Only 대비(왼쪽)·MS Only 대비(오른쪽) / 전체 경로 합계 감소율과 95% 반복 코호트 구간 / 주황은 IA 기본, 청록은 MS 기본 / Legacy_MS_KMeans는 제안 모델, StaticP는 고정 *P*다.

모든 모델에서 MS 기본이 IA 기본보다 1.87~9.61% 빨랐다. IA 기본 정책은 근거가 부족한 그룹을 IA로 두므로 모든 모델이 MS Only보다 1.96~11.10% 느렸다. 반면, MS 기본 모델은 MS Only와 −0.43~+0.95% 차이로 95% 구간이 모두 0을 포함하였다. 이들은 대부분 환경에서 검증을 통과한 후보가 없어 MS로 고정되었고, 실제 혼합은 많아야 3개 환경에서 일어났다. 표의 K-Means 행은 다른 모델과 같은 절차로 보정한 모델이며, 제안 모델은 3.8절의 절차로 보존한 모델이다.

제안한 MS 기본 K-Means는 IA Only 대비 15.99%, MS Only 대비 2.39%(95% 구간 0.88~3.82%) 시간을 줄여 26개 모델 중 가장 빨랐고, 구간 전체가 0보다 큰 유일한 모델이었다(RQ2). 이 모델은 6개 환경에서 실제 혼합을 수행했으며, 환경마다 IA Only와 MS Only 중 하나만 V로 골라 고정한 기준선보다도 1.33% 빨랐다. 따라서, 이득은 환경 단위의 경로 선택이 아니라 화면 안의 그룹 혼합에서 나온다. 환경별로는 Studio_D(+9.74%), StudioC(+7.85%), Sanctuary(+3.84%), Platformer_StreetSection(+2.41%)에서 이득이 컸고 TrackerlessStudio만 1.72% 느렸으며, 이는 E1의 환경별 경향(표 16)과 같다. 항상 MS로 동작하는 4개 환경의 차이는 ±0.9% 이내로 측정 변동 수준이다(부록 C). E1에서는 가장 단순한 고정 P와, 제안 모델과 Centroid가 같은 IA 기본 K-Means를 비교 대상으로 삼는다.

제안 모델의 국소 구간도 전수 탐색하였다. 구간 2,694개 가운데 IA Only와 MS Only보다 각각 20% 이상 짧고 8회 반복의 모든 프레임에서 실제 혼합이 일어난 구간은 6개 환경의 39개였다(표 14, 부록 F). 그림 11은 이 중 Platformer_StreetSection 18~19초 구간으로, 8회 반복 모두에서 MS 대비 24% 이상 짧았다.

표 14. 제안 모델의 환경별 국소 구간

환경	전체 구간	조건 충족	비율	대표 구간(IA / MS 대비)
Studio_D	108	13	12.04%	20~21초 (29.99% / 27.49%)

StudioC	68	9	13.24%	19~20초 (30.95% / 31.60%)
Platformer_StreetSection	298	10	3.36%	18~19초 (51.15% / 27.32%)
StudioB	65	4	6.15%	51~52초 (34.57% / 34.30%)
StudioA	102	2	1.96%	47~48초 (29.53% / 28.77%)
TrackerlessStudio	96	1	1.04%	51~52초 (21.56% / 22.68%)
나머지 5개 환경	1,957	0	0.00%	—

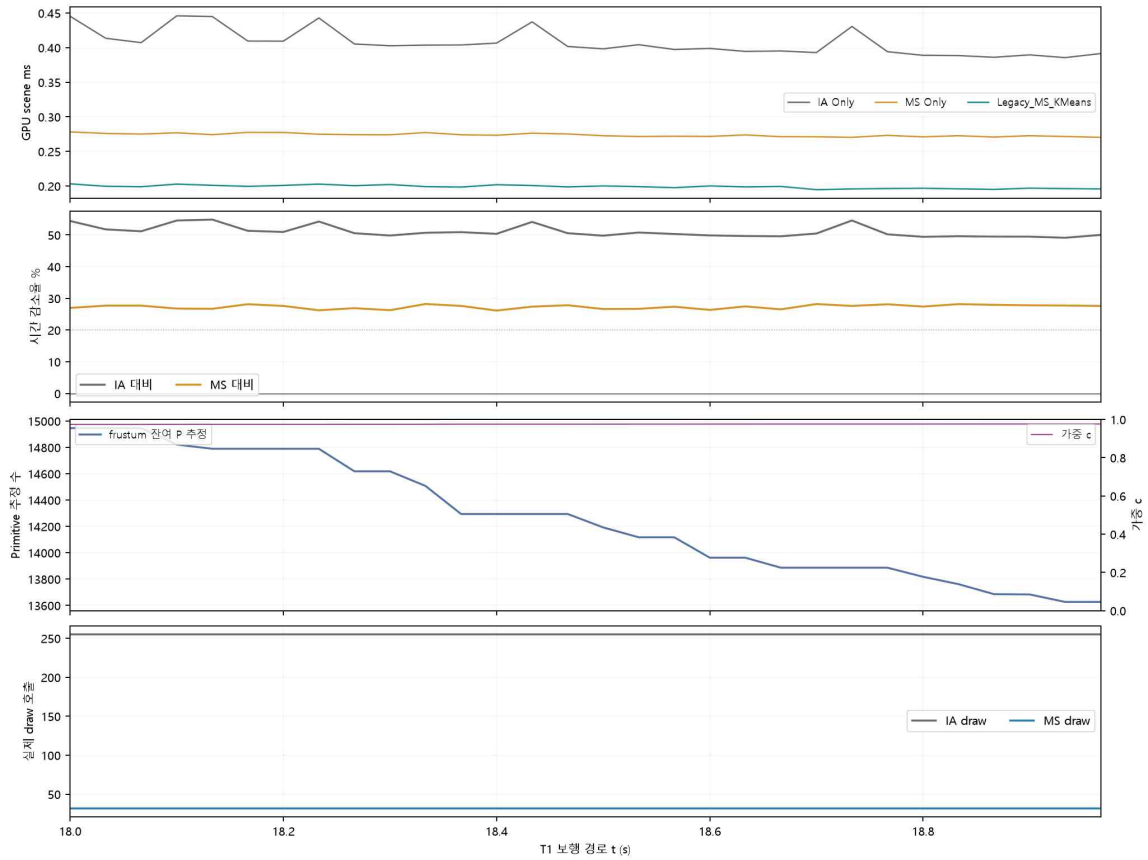


그림. 11. Platformer_StreetSection 18~19초 구간의 제안 모델(Legacy_MS_KMeans) / 위부터 GPU Scene 시간, 구도별 감소율, frustum 잔여 P 추정과 C, 실제 IA·MS 드로우 수

5.5 E1: 제안 정책의 평가

E1에서 MS 기본 K-Means는 전반적으로 IA Only 대비 15.83%, MS Only 대비 3.01% 시간을 줄여, 다섯 선택기 모델 중 유일하게 두 전용 경로를 모두 앞섰다. MS 비교의 95% paired bootstrap 구간은 2.78~3.26%로 전 구간이 0보다 크다(RQ3). 평균뿐 아니라 중앙값과 p99에서도 가장 낮았고, 환경을 동등 가중해도 MS 대비 2.32% 빨랐다. Centroid와 정규화가 같은 IA 기본 K-Means보다도 5.80% 짧았다. 반면, 고정 P와 IA 기본 K-Means는 MS보다 각각 3.87%, 2.96% 느렸다. 전체 경로 합계의 감소율은 그림 12, GPU 시간 분포는 표 15와 같다.

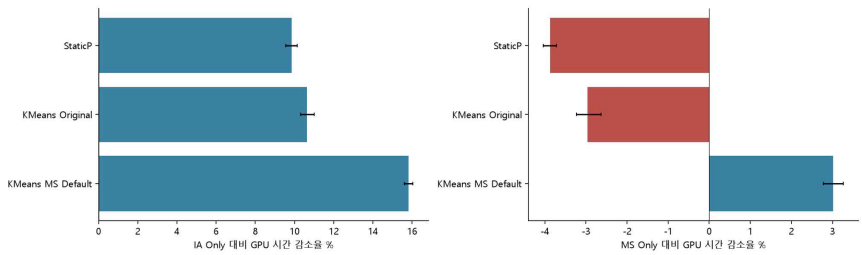


그림. 12. E1 전체 경로 합계의 감소율과 95% 반복 코호트 구간

표 15. E1 GPU Scene 시간 분포 (방법당 653,160프레임, ms)

모델	중앙값	평균	표준편차	p99	CV(%)	IA 대비	MS 대비
IA Only	0.666	0.750	0.412	2.154	54.97	—	−15.22%
MS Only	0.571	0.651	0.383	1.939	58.91	+13.21%	—
고정 P	0.611	0.676	0.363	1.932	53.72	+9.85%	−3.87%
IA 기본 K-Means	0.598	0.670	0.371	1.949	55.44	+10.64%	−2.96%
MS 기본 K-Means	0.55	0.631	0.363	1.896	57.56	+15.83%	+3.01%

반복 안정성은 환경별 8개 코호트 평균의 CV로 보며, 제안 정책에서 0.66~6.41%였다(부록 A). 환경별 실행 형태와 성능은 표 16과 그림 13에 나타낸다.

표 16. E1 환경별 MS 기본 K-Means의 실행 형태와 성능 (MS 대비 내림차순)

환경	실행 형태	혼합 프레임 비율	IA 대비	MS 대비	IA→MS / MS→IA 전환
Studio_D	실제 혼합	96.70%	+15.56%	+9.65%	361 / 351
Sanctuary	전부 IA	0.00%	+0.11%	+6.87%	0 / 0
StudioC	실제 혼합	100.00%	+6.87%	+4.61%	262 / 255
StudioA	실제 혼합	100.00%	+5.20%	+3.14%	235 / 240
Platformer_StreetSection	실제 혼합	100.00%	+17.13%	+2.11%	279 / 279
StudioB	실제 혼합	100.00%	+4.24%	+1.22%	184 / 184
Highrise	MS 유지	0.00%	+22.60%	+0.70%	0 / 0
LV_Soul_Cave	MS 유지	0.00%	+6.28%	+0.04%	0 / 0
LV_Soul_Slum	MS 유지	0.00%	+39.31%	−0.06%	0 / 0
SunTemple	MS 유지	0.00%	+16.22%	−0.49%	0 / 0
TrackerlessStudio	실제 혼합	100.00%	+3.31%	−2.27%	241 / 239

같은 프레임 안의 IA·MS 동시 혼합은 6개 환경에서 관찰되었고, 카메라 이동에 따른 논리 그룹 전환은 IA→MS 1,562회, MS→IA 1,548회였다(RQ1). 실행 여부는 실제 드로우 수와 계획·실행 배정 해시의 일치로 확인하였다. 나머지 4개 환경은 V에서 통과 후보가 없어 항상 MS로, Sanctuary는 허용된 IA 군집만 선택되어 전부 IA로 동작하였다.

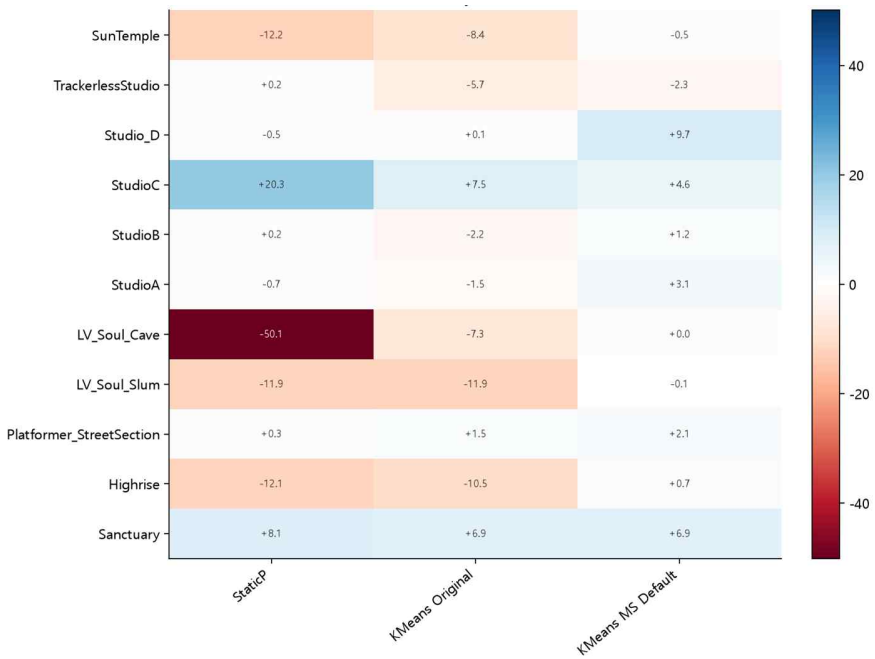


그림. 13. E1 환경별 MS Only 대비 GPU 시간 감소율(%)

5.6 이득의 분해

- 한 번의 전체 경로 재생에서 순절감 1,602.40 ms 중 53.2%는 전부 IA로 동작한 Sanctuary에서, 40.2%는 실제 혼합 6개 환경에서 나왔다. 나머지는 MS와 배정이 같은 환경의 반복 변동이다.
- 실제 혼합 6개 환경을 합산하면 MS 대비 3.79% 감소다.
- MS와 배정이 같은 4개 환경의 차이를 0으로 두면 전체 감소율은 2.82%다.
- 환경을 하나씩 빼도 감소율은 1.84~4.03%로 양수였다. Sanctuary를 뺀 가장 낮은 경우에도 1.84%로, 이득이 특정 환경 하나에만 기대지 않는다. 환경별 이득·손실 구도의 누적 기여는 그림 14와 같다.

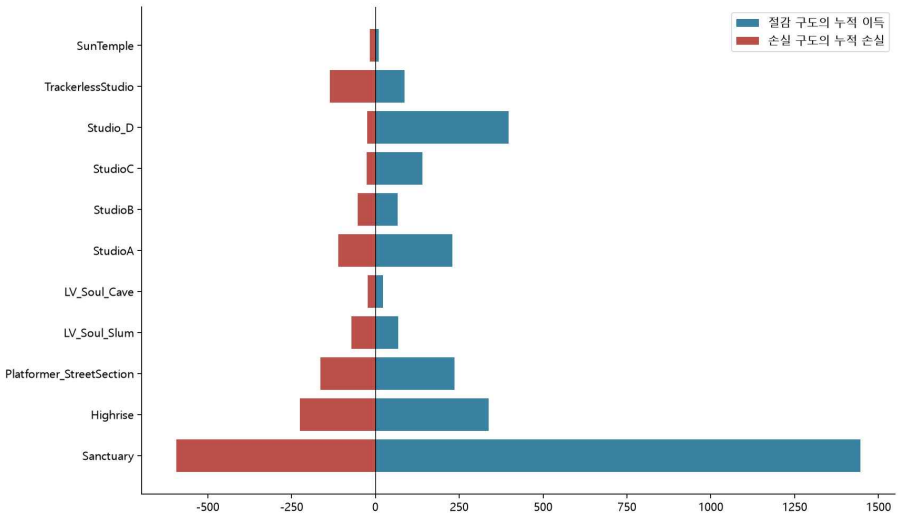


그림. 14. 한 경로 재생의 GPU 시간 합계 ms

5.7 국소 구간과 Tail Latency

구간 538개 중 MS 기본 K-Means는 64.68%에서 MS보다, 57.43%에서 IA와 MS 양쪽보다 짧아, 절반이 넘는 구간에서 두 전용 경로를 모두 앞섰다(RQ4). MS보다 3% 이상 긴 구간은 16.17%였고, 가장 큰 손실은 Sanctuary의 일부 5초 구간(MS 대비 최대 −47.69%)에서 나타났다. 이 환경은 정책상 전부 IA로 동작했으므로, 손실은 IA↔MS 전환 비용이 아니라 구역에 따라 IA가 불리해지는 Scene 상태에서 왔다. 맵을 더 작은 구역으로 나누어 구역마다 정책을 베이크하면 줄일 수 있는 부분이다(7장).

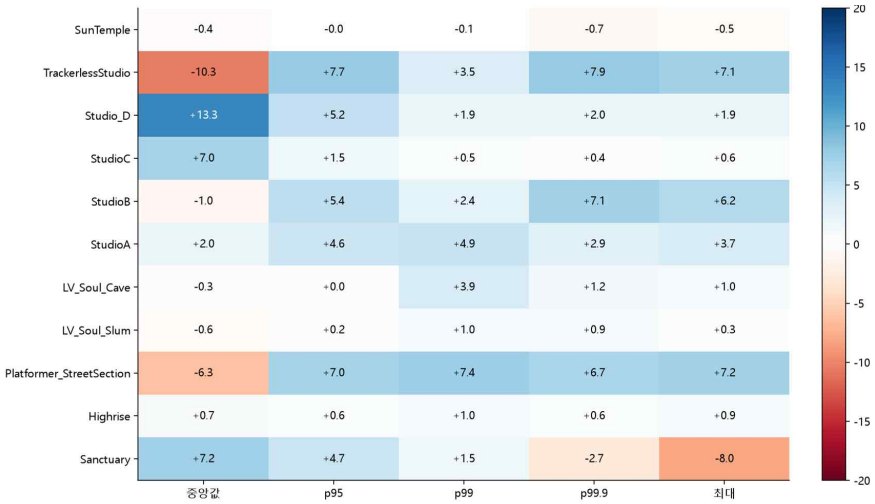


그림. 15. E1 GPU 분포 통계별 MS 대비 감소율 %

Tail Latency에서도 대부분의 환경이 개선되었다. 11개 환경 중 p95는 9개, p99는 10개 환경에서 MS보다 낮았고, 평균이 2.27% 느렸던 TrackerlessStudio도 p95와 p99는 각각 7.72%, 3.53% 개선되었다. 반면, Sanctuary는 평균 6.87% 개선과 함께 최대 GPU 시간이 평균 7.98% 늘어, Tail Latency를 함께 고려하는 선택으로 더 다듬을 여지가 있다(그림 15).

5.8 선택 계산 비용

표 17은 T1에서 측정한 방법별 프레임당 CPU 선택 비용과 PSO 바인딩 수다. CPU 선택 비용은 특징 획득·정책 준비·선택기 실행 시간의 합이며 정렬 시간은 제외하였다.

표 17. T1 방법별 프레임당 CPU 선택 비용과 PSO 바인딩 (μs)

방법	특징 획득	wnsql	선택기	합계	POS 바인딩(횟수)
IA Only / MS Only	0.00	0.00	0.00	0.00	4.32
고정 P	0.00	0.00	0.00	0.00	5.32
MS 기본 공간 비용 Grid 2.5 m	0.00	0.00	2.11	2.11	4.43
MS 기본 최소 비용 Chunk 2.5 m	0.11	0.81	0.18	1.10	4.36
IA 기본 K-Means	47.45	0.00	6.93	54.38	5.58
MS 기본 K-Means	11.93	0.00	1.35	13.28	5.50

평균에는 상수 경로로 귀결되어 계산을 생략한 환경의 0이 포함된다. 제안 모델은 K-Means를 실행하는 7개 환경에서 프레임당 25.5 μs였고, 항상 MS인 4개 환경에서는 C 계산과 추론을 생략하였다. MS 기본 Grid·Chunk의 평균 비용이 작은 것은 대부분 환경에서 상수 MS로 동작했기 때문이며, 실행한 환경만 보면 각각 21.6 μs, 14.9 μs였다. CPU와 GPU 구간은 겹치므로 이 비용을 GPU 시간에 더하지 않는다.

5.9 검증 경로에서 시험 경로로의 변화

같은 환경이라도 검증 경로(V)와 시험 경로(E1)의 카메라 분포가 다르면 이득의 크기가 달라졌다. Studio_D는 V의 6.28%에서 E1의 9.65%로 이득이 커졌고, TrackerlessStudio는 V의 15.34%에서 E1의 −2.27%로 바뀌었다. 같은 군집 안에서도 경로에 따라 보이는 그룹과 Material 구성이 달라지기 때문이며, 보조 진단에서 일부 IA 군집을 MS로 바꾸면 오히려 빨라진 것도 같은 맥락이다. 이는 환경 특화 모델의

정보를 맵 전체가 아니라 더 작은 구역 단위로 나누어 베이크하면 이득을 더 안정적으로 끌어낼 수 있음을 시사한다.

5.10 논의

그룹 비용과 혼합 비용. 그룹 단위 비용 라벨만으로 IA를 배정하는 IA 기본 선택기는 실제 혼합 실행에서 정렬 순서·PSO 전환·캐시 상태가 달라져 MS 전용보다 느렸지만, 이 차이를 검증 경로의 실제 혼합 시간으로 보정한 MS 기본 K-Means는 두 전용 경로를 모두 앞섰다. 같은 MS 기본 정책을 다른 모델에 적용하면 모든 모델에서 IA 기본보다 빨랐으나, 대부분 환경에서 검증을 통과한 후보가 없어 MS 전용과 비슷한 수준에 머물렀다(5.4절). 다만, 공간 비용 Grid는 제안 모델이 MS보다 느렸던 TrackerlessStudio에서 MS 대비 6.11~7.76% 빨라, 환경에 따라 모델을 고르는 확장의 여지가 있다.

Geometry 양과 시간의 관계. AS는 Rasterizer 입력을 크게 줄이지만(5.3절), Primitive를 더 많이 보낸 MS 기본 K-Means가 MS Only보다 빨랐다. IA의 하드웨어 Instancing과 정점 재사용, AS의 고정비, Meshlet 간 정점 중복 처리가 함께 작용하기 때문이다. 따라서, Geometry 양만으로 정하는 일반 규칙보다, 환경별 실측 비용을 베이크해 쓰는 본 연구의 접근이 실제 GPU 시간을 더 잘 반영한다.

평가 범위. 주 지표는 Clear와 Draw의 GPU Scene 시간이며 그림자·후처리·게임플레이를 포함한 전체 프레임 시간은 아니다. 실험은 한 종류의 GPU와 1280×720 해상도에서 수행되었고 객체 단위 CPU 컬링·오클루전 컬링·LOD는 포함하지 않았다. 선택기는 환경 특화 모델이므로, 다른 환경이나 GPU에는 같은 절차로 그 조건의 정보를 새로 베이크하여 적용한다. 또한, 같은 조건에서도 실행 순서가 측정을 편향시킬 수 있어[25], 배정이 같은 환경에서도 반복 차이(최대 0.70%)가 관측되었다.

6. 결론

본 논문은 DirectX 12에서 IA 경로와 Meshlet 경로를 하나의 Renderer에 통합하고, 환경마다 사전에 베이크한 정보로 논리 Mesh 그룹별 경로를 고르는 하이브리드 파이프라인을 구현하였다. 연구 질문에 대한 답은 표 18과 같다.

표 18. 연구 질문별 결론

연구 질문	답	근거
RQ1 동일 프레임 혼합과 전환	가능하다	6개 환경에서 동시 혼합, IA→MS 1,562회·MS→IA 1,548회 전환, Depth 화면 일치
RQ2 환경 특화 선택기의 이득	모든 모델에서 MS 기본이 IA 기본보다 빨랐고, MS 기본 K-Means만 MS 전용을 확실히 앞섰다	T1 26개 모델에서 MS 기본이 IA 기본 대비 1.87~9.61% 빠름, MS 기본 K-Means는 MS 대비 2.39%(95% 구간 0.88~3.82%)
RQ3 제안 정책	두 전용 경로를 모두 앞섰다	E1에서 IA 대비 15.83%, MS 대비 3.01%(95% 구간 2.78~3.26%) 감소, 다섯 선택기 모델 중 유일, 같은 Centroid의 IA 기본 K-Means보다 5.80% 빠름
RQ4 국소·Tail Latency 유지	대체로 유지된다	5초 구간의 57.43%에서 양쪽보다 빠름, p95는 9개·p99는 10개 환경에서 MS보다 낮음

핵심 결과는 환경마다 베이크한 정보로 만든 하이브리드 선택기가 IA 전용과 MS 전용을 모두 앞설 수 있다는 것이다. 특히, 그룹 단위 비용과 실제 혼합 비용의 차이를 검증 경로의 실제 혼합 시간으로

보정한 MS 기본 K-Means는 단순한 구조로도 두 전용 경로보다 짧은 GPU 시간을 보였고, 중앙값·p99에서도 가장 좋았다. 또한, Primitive를 더 보낸 정책이 더 빨랐다는 관찰은, Geometry 양 규칙보다 환경별 실측 비용을 베이크하는 접근이 필요함을 뒷받침한다.

이 결과는 맵이나 그보다 작은 구역마다 비용 정보를 미리 구워 두는 실제 게임 파이프라인으로 확장할 가능성을 보여 준다. 사례로 제시한 국소 구간처럼 경로 선택이 Scene에 잘 맞을 때의 이득이 크므로, 구역 단위 특화와 혼합 비용 기반 학습을 더하면 이득을 더 키울 여지가 있다. 본 논문에서 관측한 이득은 11개 환경의 GPU Scene 시간에 대한 것이며, 전체 게임 FPS로 바로 환산하지는 않는다.

7. 후속 연구

1. **구역 단위 특화**. 맵을 더 작은 구역으로 나누어 구역마다 MS 기본 정책을 베이크하고, 카메라가 있는 구역의 정책을 쓰도록 최소 비용 Chunk 구조를 확장한다. 구역에 따라 IA가 불리해지는 Scene(예: Sanctuary)의 손실을 줄일 수 있다.
2. **혼합 배치 비용 기반 라벨**. 그룹 단위 비용 대신 실제 혼합 배치의 비용을 측정하거나 보간하여 베이크한다.
3. **환경별 모델 선택**. 제안 모델이 약한 환경(예: TrackerlessStudio)에서 공간 비용 Grid가 빨랐던 결과를 바탕으로, 환경이나 구역마다 K-Means와 Grid 중 하나를 고르는 구조를 검증한다.
4. **Tail Latency와 전환 제어**. 평균과 함께 p99·최대 지연을 선택 기준에 넣고, 전환 빈도를 제어하는 히스테리시스의 효과를 검증한다.
5. **선택 비용의 분리**. 같은 배정과 그리기 순서를 재생하는 대조 실험으로 C 획득·추론·PSO 바인딩 비용과 배치 효과를 나눈다.
6. **베이크 자동화와 이식**. 비용 취득·학습·검증·보존을 한 번에 수행하는 도구로 만들어, 다른 환경·GPU·해상도에서 같은 절차로 환경 특화 모델을 만든다. 학습만 완료한 0.5~5초 간격 모델의 실측 비교도 포함한다.
7. **컬링 확장**. Normal Cone 후면 컬링, Depth 기반 오클루전 컬링[7], Meshlet 단위 LOD[9]를 추가하고 본 측정 체계로 그 효과를 평가한다.
8. **실제 게임 부하 포함**. 그림자·후처리·다중 광원을 포함한 환경에서 전체 프레임 시간 기준으로 평가한다.

부록

부록 A 반복 안정성

E1 MS 기본 K-Means의 변동을 세 수준으로 나눈다. 프레임 CV는 경로에 따른 작업량 변화를 포함하고, 코호트 평균 CV와 같은 구도 반복 CV는 반복 산포를 더 직접 보여 준다. 변동 계수는 표 A1, 반복 코호트별 감소율은 그림 A1과 같다.

표 A1. E1 MS 기본 K-Means의 변동 계수(%)

환경	프레임 CV	코호트 평균 CV	같은 구도 반복 CV 중앙값
SunTemple	32.44	0.79	2.28
TrackerlessStudio	51.76	2.34	3.34
Studio_D	33.83	2.99	6.00
StudioC	39.65	1.76	6.03

StudioB	46.21	0.84	1.77
StudioA	40.66	0.74	7.05
LV_Soul_Cave	48.98	0.66	2.21
LV_Soul_Slum	62.33	1.01	3.40
Platformer_StreetSection	36.95	1.18	2.39
Highrise	29.82	0.75	2.45
Sanctuary	45.34	6.41	6.89

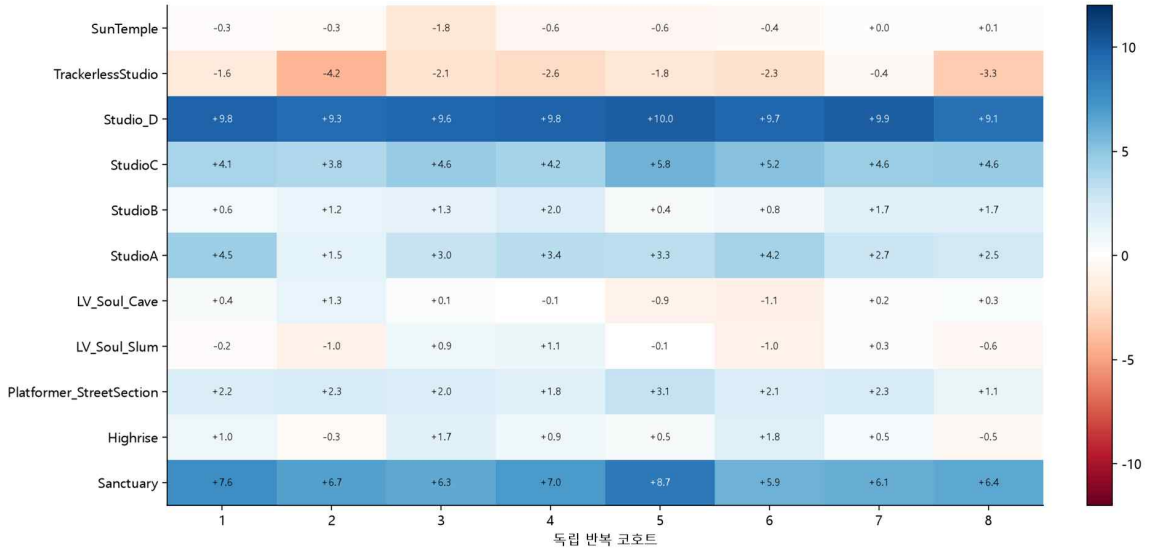


그림. A1. E1 반복별 MS 대비 GPU 시간 감소율(%)

부록 B T1 파이프라인 통계

표 B1은 T1 재생에서 수집한 프레임당 평균 파이프라인 통계다.

표 B1. T1 프레임당 평균 파이프라인 통계 (CInvocations 내림차순)

방법	CInvocations	IA 대비 감소	Meshlet 경로 Primitive	PS 실행
IA Only	1,132,493.0	0.00%	0	3,469,642
최소 비용 Chunk 1 m	794,147.9	29.88%	52,707.3	3,517,894
최소 비용 Chunk 2.5 m	770,531.4	31.96%	54,058.4	3,517,990
최소 비용 Chunk 5 m	747,886.8	33.96%	55,714.1	3,513,917
최소 비용 Chunk 10 m	739,390.8	34.71%	55,855.2	3,540,264
Logistic	667,117.8	41.09%	82,516.9	3,493,727
LUT	658,395.1	41.86%	86,935.5	3,512,116
고정 P	641,442.9	43.36%	97,405.9	3,497,236
IA 기본 K-Means	638,527.9	43.62%	83,112.7	3,475,803
공간 비용 Grid 1 m	630,232.6	44.35%	93,243.8	3,556,886
공간 비용 Grid 2.5 m	621,895.1	45.09%	91,283.2	3,544,226
공간 비용 Grid 5 m	610,926.0	46.05%	90,999.1	3,545,115
공간 비용 Grid 10 m	603,001.2	46.75%	89,372.4	3,574,681
MS 기본 K-Means	249,603.7	77.96%	181,820.8	3,469,930
MS Only	200,175.8	82.32%	200,175.8	3,537,719

CPrimitives는 모든 방법에서 170,201.5로 같았다.

부록 C 환경별 MS 대비 감소율

환경별 감소율 히트맵이다. 양수가 GPU 시간 감소이다. 그림 C1·C2는 T1 26개 모델이며, 그림 안의 IA_·MS_ 접두사는 IA 기본·MS 기본 정책, StaticP는 고정 P, Legacy_MS_KMeans는 제안 모델이다. 그림 C3은 E1 결과이며, 그림 안의 KMeans는 IA 기본 K-Means, KMeans MS Default는 제안 모델이다. E1의 MS 대비 결과는 본문 그림 13에 있다.

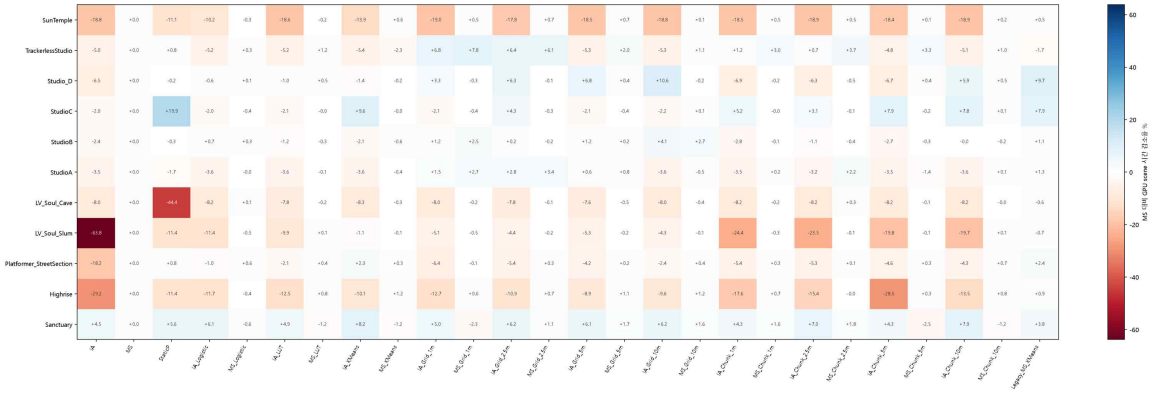


그림. C1. T1 환경별 MS Only 대비 시간 감소율(%)

제안 모델은 Studio_D·StudioC·Sanctuary·Platformer_StreetSection에서 이득이 컸고, 항상 MS로 동작하는 환경의 차이는 ±0.9% 이내였다.



그림. C2. T1 환경별 IA Only 대비 시간 감소율(%)

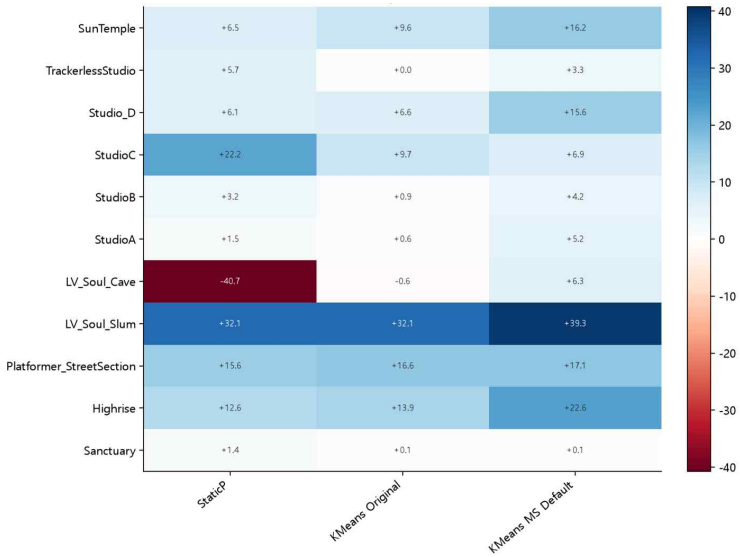


그림. C3. E1 환경별 IA Only 대비 시간 감소율(%)

부록 D 환경별 선정 파라미터

표 D1에 환경별로 모델 파라미터를 정리한다.

표 D1. 모델 파라미터

환경	고정 P *Threshold*	Logistic 차수 / L2	LUT knot / 평활	IA 기본 K-Means Centroid	MS 기본 정책
SunTemple	7,296	2 / 0.1	9 / 1.0	8	항상 MS
TrackerlessStudio	15,660	6 / 0.001	9 / 0.3	16	$\lambda=2,\ \tau=0$
Studio_D	51,728	6 / 0.001	9 / 0.3	8	$\lambda=2,\ \tau=1$
StudioC	1,200	5 / 0.001	5 / 1.0	8	$\lambda=2,\ \tau=0$
StudioB	33,904	5 / 0.001	17 / 0.3	16	$\lambda=2,\ \tau=1$
StudioA	47,304	5 / 0.1	5 / 0.0	16	$\lambda=2,\ \tau=1$
LV_Soul_Cave	42,188	2 / 0.001	5 / 0.0	16	항상 MS
LV_Soul_Slum	3,741,928	6 / 0.001	17 / 0.0	16	항상 MS
Platformer_StreetSection	4,736	4 / 0.1	17 / 0.0	16	$\lambda=2,\ \tau=1$
Highrise	37,872	3 / 0.01	17 / 0.0	16	항상 MS
Sanctuary	23,870	2 / 0.1	9 / 0.3	2	$\lambda=2,\ \tau=1$

부록 E 실험 환경과 카메라 경로

환경마다 (a) E1 경로에서 균등하게 고른 중앙 구도의 자체 Renderer IA 참조 화면과 (b) 카메라 경로 조감도를 싣는다. 조감도는 일정 높이 위의 Geometry를 제거한 CPU Mesh 투영이며, 경로 색은 보행 시간, 별은 시작점, 검은 사각형은 끝점이다. 원본 게임의 조명 화면이 아니며 성능 상위 구도를 고른 것도 아니다. 각 환경의 그림은 그림 E1~E11에 싣는다.



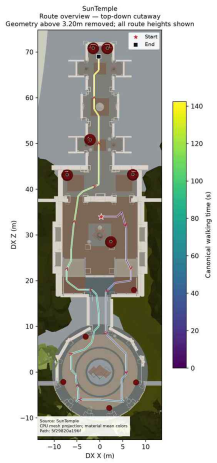


그림. E1. SunTemple: 참조 화면(왼쪽)과 카메라 경로 조감도(오른쪽)

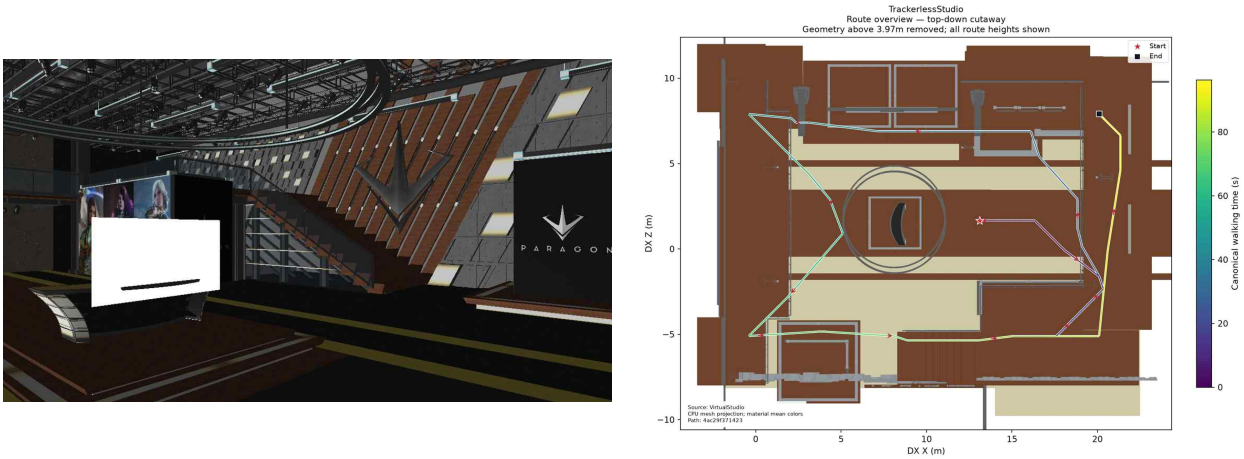


그림. E2. TrackerlessStudio: 참조 화면(왼쪽)과 카메라 경로 조감도(오른쪽)

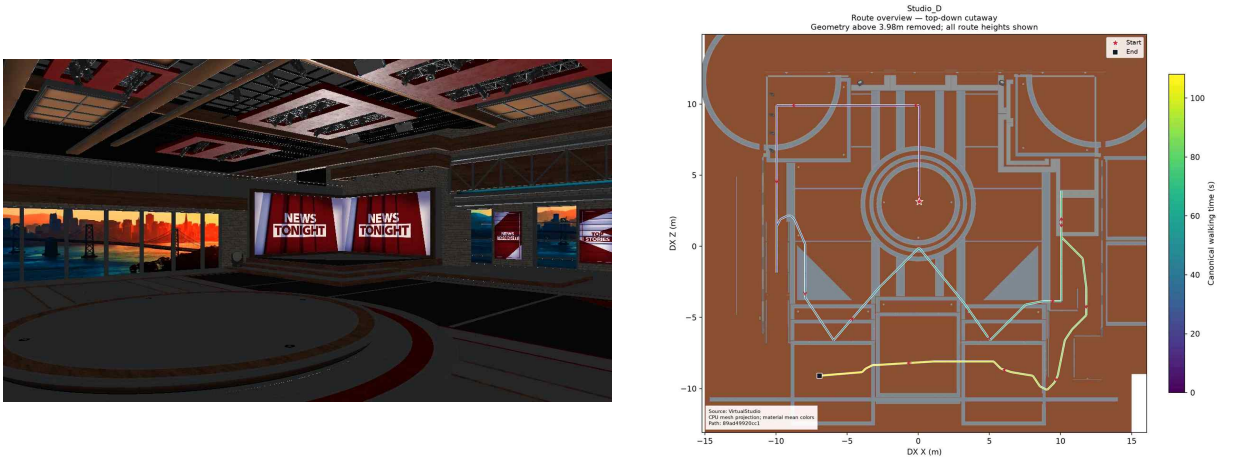


그림. E3. Studio_D: 참조 화면(왼쪽)과 카메라 경로 조감도(오른쪽)

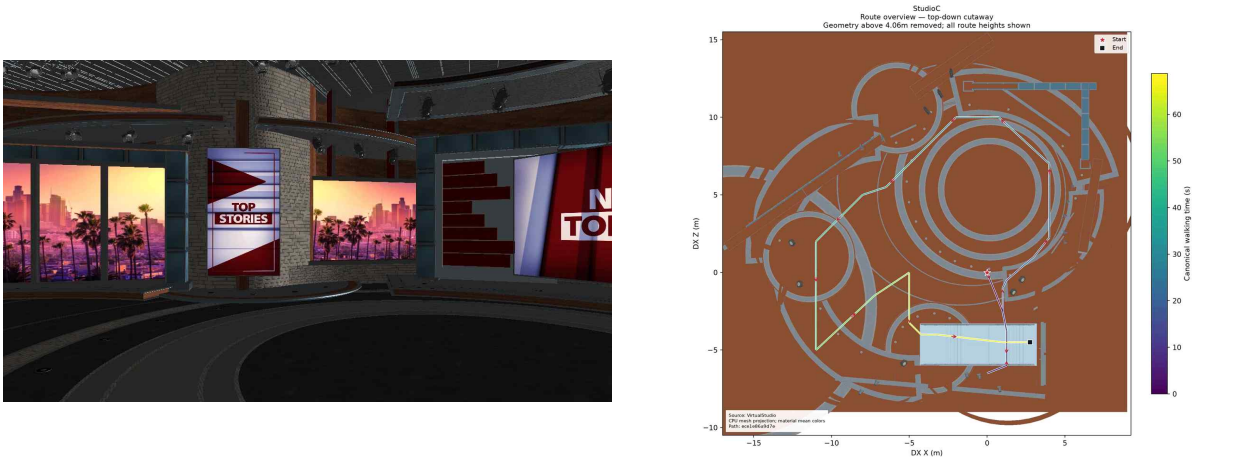


그림. E4. StudioC: 참조 화면(왼쪽)과 카메라 경로 조감도(오른쪽)

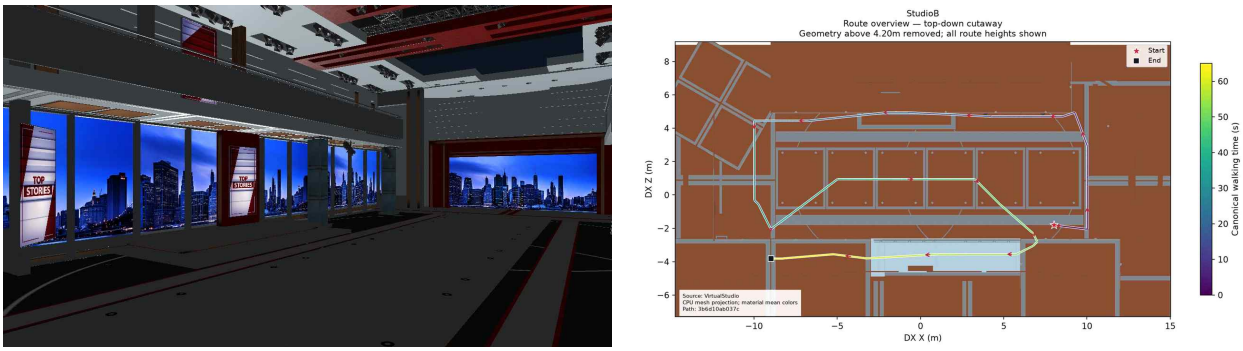


그림. E5. StudioB: 참조 화면(왼쪽)과 카메라 경로 조감도(오른쪽)

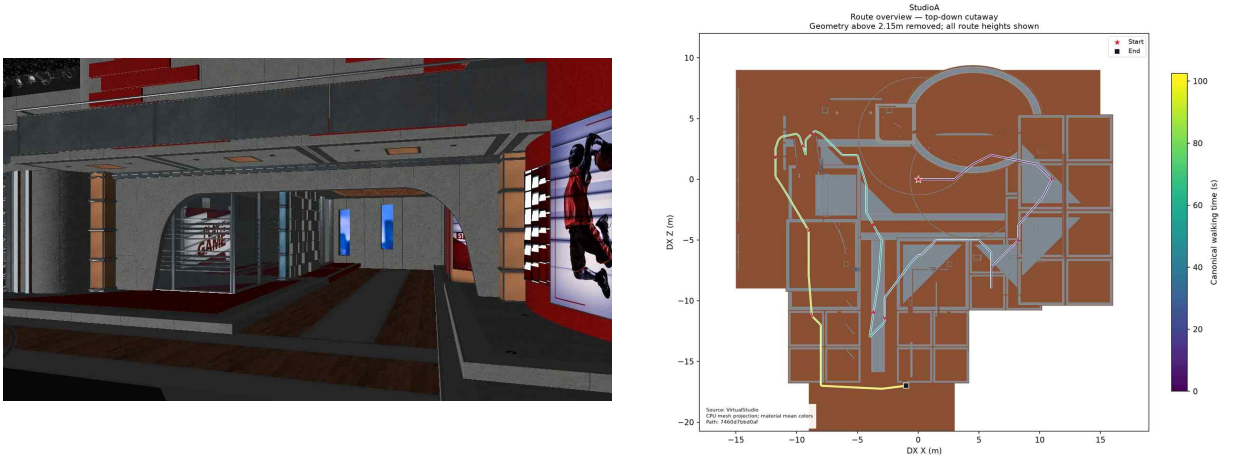


그림. E6. StudioA: 참조 화면(왼쪽)과 카메라 경로 조감도(오른쪽)

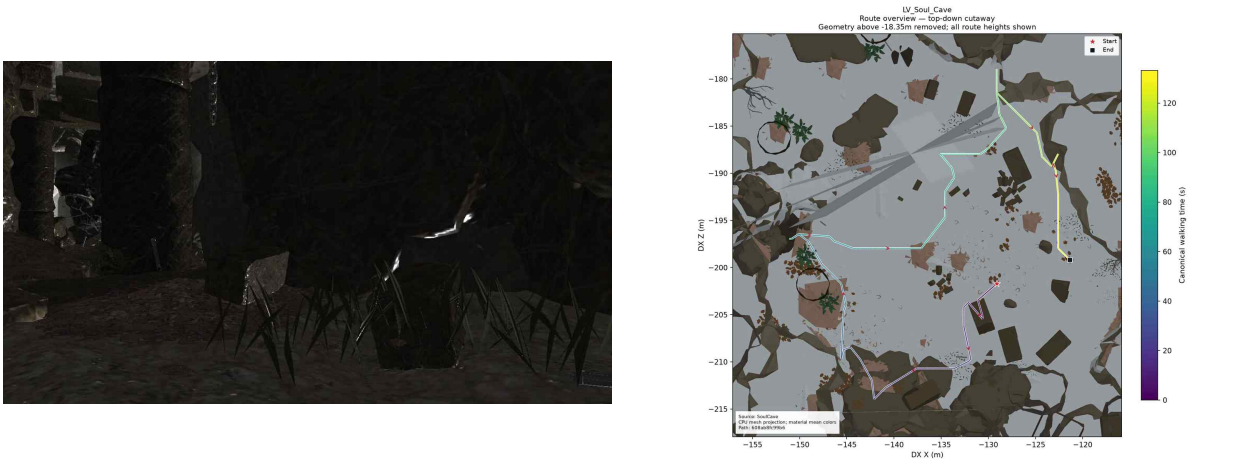


그림. E7. LV_Soul_Cave: 참조 화면(왼쪽)과 카메라 경로 조감도(오른쪽)



그림. E8. LV_Soul_Slum: 참조 화면(왼쪽)과 카메라 경로 조감도(오른쪽)

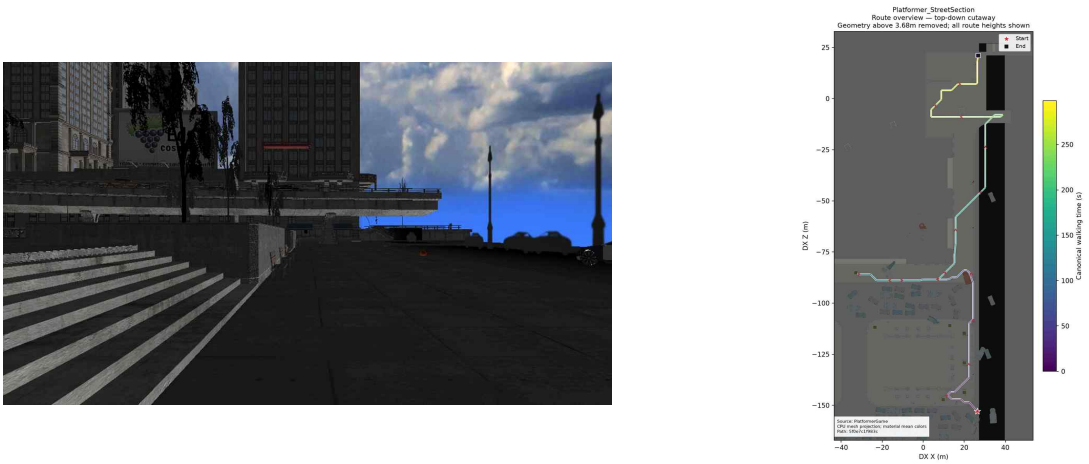


그림. E9. Platformer_StreetSection: 참조 화면(왼쪽)과 카메라 경로 조감도(오른쪽)

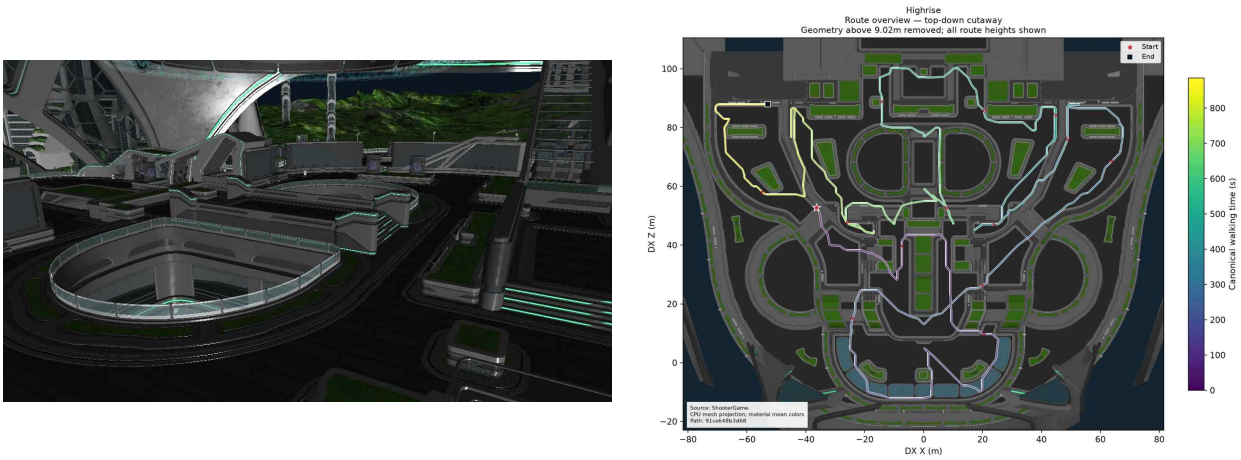


그림. E10. Highrise: 참조 화면(왼쪽)과 카메라 경로 조감도(오른쪽)

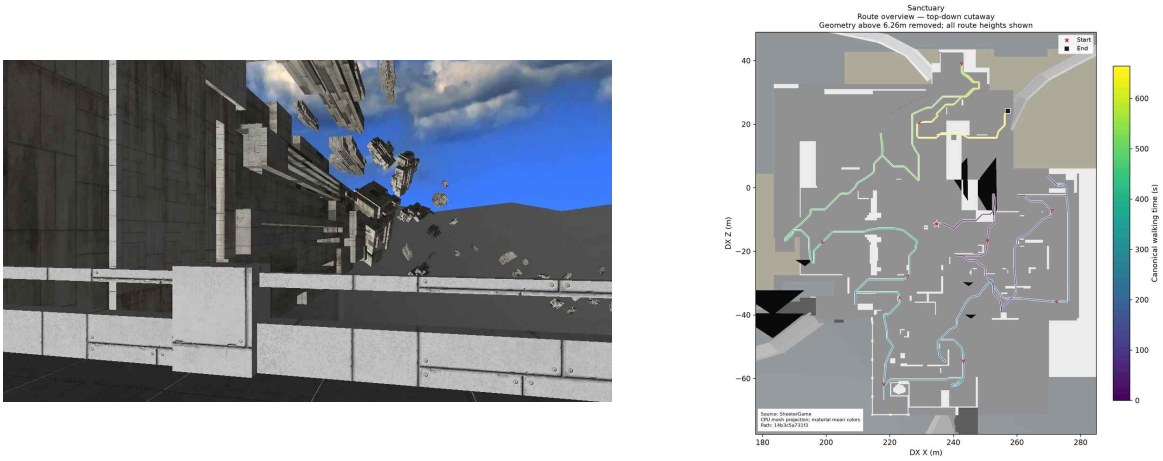


그림. E11. Sanctuary: 참조 화면(왼쪽)과 카메라 경로 조감도(오른쪽)

부록 F 제안 모델의 국소 구간 목록

T1 경로의 구간마다 8회 반복의 구간 평균을 구하고, IA Only와 MS Only 대비 감소율이 각각 20% 이상이며 8회 반복의 모든 프레임에서 IA·MS 드로우가 함께 실행된 구간을 골랐다. 구간은 사후에 탐색한 것이며, 감소율은 구간 평균이지 모든 프레임의 최소 이득이 아니다. 모든 구간의 BY 보정 q값은 0.002 미만이었다[28]. 표 F1은 39개 구간 전체다.

표 F1. 제안 모델의 국소 구간

환경	구간(초)	IA / MS / 혼합(ms)	IA 대비	MS 대비
Studio_D	18~19	1.5384 / 1.4869 / 1.1646	24.30%	21.68%
Studio_D	19~20	1.5317 / 1.4920 / 1.1514	24.83%	22.83%
Studio_D	20~21	1.5374 / 1.4844 / 1.0763	29.99%	27.49%
Studio_D	21~22	1.4760 / 1.4541 / 1.0975	25.64%	24.52%
Studio_D	22~23	1.4192 / 1.4043 / 1.0356	27.03%	26.26%
Studio_D	23~24	1.3077 / 1.2796 / 0.9976	23.71%	22.04%
Studio_D	36~37	1.4436 / 1.3870 / 1.0744	25.58%	22.54%
Studio_D	52~53	1.6758 / 1.5840 / 1.2371	26.18%	21.90%
Studio_D	53~54	1.6109 / 1.5166 / 1.1842	26.49%	21.92%
Studio_D	54~55	1.5452 / 1.4438 / 1.1002	28.79%	23.79%
Studio_D	55~56	1.4233 / 1.3581 / 0.9729	31.64%	28.36%
Studio_D	56~57	1.2119 / 1.2017 / 0.8398	30.70%	30.12%
Studio_D	57~58	1.1029 / 1.0987 / 0.7900	28.37%	28.10%
StudioC	10~11	0.9325 / 0.9552 / 0.7436	20.26%	22.16%
StudioC	18~19	1.1599 / 1.1833 / 0.8284	28.58%	29.99%
StudioC	19~20	1.0560 / 1.0660 / 0.7292	30.95%	31.60%
StudioC	20~21	1.0389 / 1.0416 / 0.7461	28.18%	28.37%
StudioC	31~32	1.3745 / 1.3541 / 1.0678	22.31%	21.14%
StudioC	60~61	0.8706 / 0.8737 / 0.6311	27.51%	27.77%
StudioC	61~62	0.9888 / 0.9437 / 0.7235	26.83%	23.33%
StudioC	63~64	1.2669 / 1.2534 / 0.9734	23.17%	22.34%
StudioC	64~65	1.0179 / 0.9643 / 0.7499	26.33%	22.23%
Platformer_StreetSection	4~5	0.6666 / 0.5830 / 0.4612	30.82%	20.89%
Platformer_StreetSection	7~8	0.6272 / 0.5538 / 0.4210	32.88%	23.98%
Platformer_StreetSection	8~9	0.6569 / 0.5718 / 0.4360	33.63%	23.76%
Platformer_StreetSection	9~10	0.6189 / 0.5461 / 0.4285	30.76%	21.53%
Platformer_StreetSection	14~15	0.6431 / 0.5460 / 0.4220	34.38%	22.71%
Platformer_StreetSection	15~16	0.6361 / 0.5378 / 0.4188	34.15%	22.13%
Platformer_StreetSection	17~18	0.4670 / 0.3937 / 0.3081	34.02%	21.75%

Platformer_StreetSection	18~19	0.4072 / 0.2737 / 0.1989	51.15%	27.32%
Platformer_StreetSection	19~20	0.3755 / 0.2332 / 0.1843	50.92%	20.99%
Platformer_StreetSection	295~296	0.5936 / 0.5888 / 0.4701	20.82%	20.17%
StudioB	32~33	0.4209 / 0.3750 / 0.2939	30.17%	21.61%
StudioB	51~52	0.5357 / 0.5335 / 0.3505	34.57%	34.30%
StudioB	53~54	0.8800 / 0.9142 / 0.6366	27.66%	30.36%
StudioB	54~55	0.8334 / 0.8575 / 0.6248	25.03%	27.14%
StudioA	47~48	2.0305 / 2.0087 / 1.4309	29.53%	28.77%
StudioA	48~49	1.9494 / 1.9258 / 1.4399	26.14%	25.23%
TrackerlessStudio	51~52	1.6320 / 1.6556 / 1.2801	21.56%	22.68%

참고문헌

[1] Microsoft, “DirectX Mesh Shader Specification.”

[2] C. Kubisch, “Introduction to Turing Mesh Shaders,” NVIDIA Developer Blog, 2018.

[3] B. Kerbl, M. Kenzel, E. Ivanchenko, D. Schmalstieg, M. Steinberger, “Revisiting the Vertex Cache: Understanding and Optimizing Vertex Processing on the Modern GPU,” Proc. ACM Comput. Graph. Interact. Tech., vol. 1, no. 2, art. 29, 2018 (HPG 2018). doi:10.1145/3233302

[4] P. V. Sander, D. Nehab, J. Barczak, “Fast Triangle Reordering for Vertex Locality and Reduced Overdraw,” ACM Trans. Graph., vol. 26, no. 3, 2007 (SIGGRAPH 2007). doi:10.1145/1276377.1276489

[5] A. Kapoulkine, “meshoptimizer,” open-source library.

[6] M. B. Jensen, J. R. Frisvad, J. A. Bærentzen, “Performance Comparison of Meshlet Generation Strategies,” Journal of Computer Graphics Techniques, vol. 12, no. 2, 2023.

[7] U. Haar, S. Aaltonen, “GPU-Driven Rendering Pipelines,” SIGGRAPH 2015 Course: Advances in Real-Time Rendering in Games, 2015.

[8] G. Wihlidal, “Optimizing the Graphics Pipeline with Compute,” Game Developers Conference, 2016.

[9] B. Karis, R. Stubbe, G. Wihlidal, “A Deep Dive into Nanite Virtualized Geometry,” SIGGRAPH 2021 Course: Advances in Real-Time Rendering in Games, 2021.

[10] T. A. Funkhouser, C. H. Séquin, “Adaptive Display Algorithm for Interactive Frame Rates During Visualization of Complex Virtual Environments,” Proc. SIGGRAPH 93, pp. 247–254, 1993.

[11] M. Wimmer, P. Wonka, “Rendering Time Estimation for Real-Time Rendering,” Eurographics Symposium on Rendering, pp. 118–129, 2003.

[12] N. Hoffman, “Background: Physics and Math of Shading,” SIGGRAPH 2013 Course: Physically Based Shading in Theory and Practice, 2013.

[13] B. Burley, “Physically Based Shading at Disney,” SIGGRAPH 2012 Course: Practical Physically Based Shading in Film and Game Production, 2012.

[14] B. Walter, S. R. Marschner, H. Li, K. E. Torrance, “Microfacet Models for Refraction through Rough Surfaces,” Eurographics Symposium on Rendering, pp. 195–206, 2007.

[15] E. Heitz, “Understanding the Masking-Shadowing Function in Microfacet-Based BRDFs,” Journal of Computer Graphics Techniques, vol. 3, no. 2, pp. 32–91, 2014.

[16] S. Lagarde, C. de Rousiers, “Moving Frostbite to Physically Based Rendering 3.0,” SIGGRAPH 2014 Course: Physically Based Shading in Theory and Practice, 2014.

[17] C. Schlick, “An Inexpensive BRDF Model for Physically-based Rendering,” Computer Graphics Forum, vol. 13, no. 3, pp. 233–246, 1994. doi:10.1111/1467-8659.1330233

[18] W. Matusik, H. Pfister, M. Brand, L. McMillan, “A Data-Driven Reflectance Model,” ACM Trans. Graph., vol. 22, no. 3, 2003 (SIGGRAPH 2003).

[19] M. Oren, S. K. Nayar, “Generalization of Lambert’s Reflectance Model,” Proc. SIGGRAPH 94, 1994.

[20] Microsoft, “Timing (Direct3D 12 Graphics).”

[21] Microsoft, “D3D12_QUERY_DATA_PIPELINE_STATISTICS” 및 “D3D12_QUERY_DATA_PIPELINE_STATISTICS1.”

[22] Microsoft, “Root Signatures Overview.”

[23] Microsoft, “Fence-Based Resource Management.”

[24] Microsoft, “Managing Graphics Pipeline State in Direct3D 12.”
[25] T. Mytkowicz, A. Diwan, M. Hauswirth, P. F. Sweeney, “Producing Wrong Data Without Doing Anything Obviously Wrong!,” Proc. ASPLOS, pp. 265–276, 2009. doi:10.1145/1508244.1508275
[26] B. Efron, “Bootstrap Methods: Another Look at the Jackknife,” Annals of Statistics, vol. 7, no. 1, pp. 1–26, 1979. doi:10.1214/aos/1176344552
[27] S. Holm, “A Simple Sequentially Rejective Multiple Test Procedure,” Scandinavian Journal of Statistics, vol. 6, no. 2, pp. 65–70, 1979.
[28] Y. Benjamini, D. Yekutieli, “The Control of the False Discovery Rate in Multiple Testing under Dependency,” Annals of Statistics, vol. 29, no. 4, pp. 1165–1188, 2001. doi:10.1214/aos/1013699998
[29] D. Arthur, S. Vassilvitskii, “k-means++: The Advantages of Careful Seeding,” Proc. ACM-SIAM SODA, pp. 1027–1035, 2007.
[30] Patapom, “BRDF Explorer (WebGL).”
[31] Google, “Filament: Physically Based Rendering,” Filament documentation.
[32] D. L. Donoho, “De-noising by soft-thresholding,” IEEE Transactions on Information Theory, vol. 41, no. 3, pp. 613–627, 1995. doi:10.1109/18.382009

이 논문은 동명대학교 학사학위 졸업논문으로 제출되었으며, 참여한 저자들의 연락처는 다음과 같습니다.

유현수		
lioie6478@gmail.com		