언리얼 페스트 2024 서울

나나이트의 여정

박창현
디벨로퍼 릴레이션 리드
에픽게임즈 코리아

나나이트 세계의 루멘

스테이트 오브 언리얼 2024
마블 1943 :라이즈 오브 하이드라

하나의 문장으로 표현한다면?

버추얼 텍스처의
지오메트리 버전

가상화된
지오메트리

나나이트의 여정

나나이트의 시작
나나이트의 기반 기술 소개

프로그래머블 래스터라이저
나나이트 지원 머티리얼 확장

스태틱/다이나믹 테셀레이션
스태틱 디스플레이스먼트 맵핑
런타임 테셀레이션 지원

컴퓨트 머티리얼
머티리얼 패스의 중요한 변경점

로드맵

나나이트의 시작

나나이트는 어떻게 동작하나

영화급 퀄리티를 위한 노력
고해상도 지오메트리

= 엄청나게 많은 트라이앵글

나나이트 제작 기조
It just works

성능 예산으로부터의 자유
- 폴리카운트, 드로 콜, 메모리

필름 퀄리티 에셋 사용
퀄리티 저하 X

나나이트가 사용한 기술

클러스터 빌드
2-패스 컬링
끊김없는 LOD
래스터라이저
머티리얼 패스
디퍼드 머티리얼


BASEPASS
STATIC LIGHTING / SHADOWS
POST PROCESSING
VISIBILITY
ADDITIONAL
PERFORMANCE
SCALABILITY EXAMPLE

기존 렌더링 파이프라인

지오메트리 가시성 검사

메시의 바운드 기준, 눈에 보이는지 여부 판단

프러스텀 컬링

차폐 컬링

하드웨어/소프트웨어 오클루전 컬링

기존 렌더링 파이프라인

지오메트리 가시성 검사

뎁스 프리패스 Depth Prepass

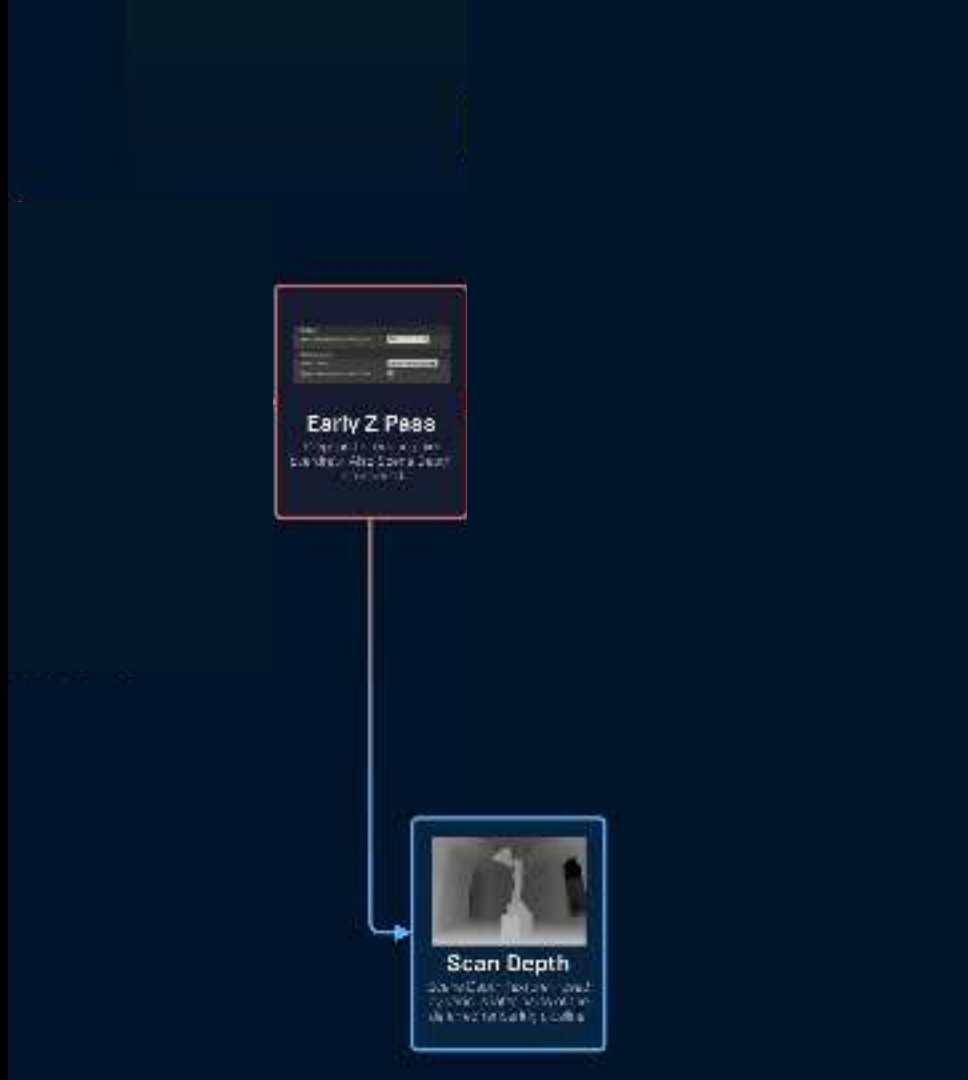
= Early-Z를 위한 Scene Depth 텍스처 렌더링

메시 정보와

실제 메시가 사용하지 않는 단순한 머티리얼을 사용해

깊이 값을 텍스처에 렌더링

➔ 후에 베이스패스 렌더링시 하드웨어 래스터라이저의 Early-Z 최적화에 활용



기존 렌더링 파이프라인

지오메트리 가시성 검사

뎁스 프리패스 Depth Prepass

베이스 패스 Base Pass

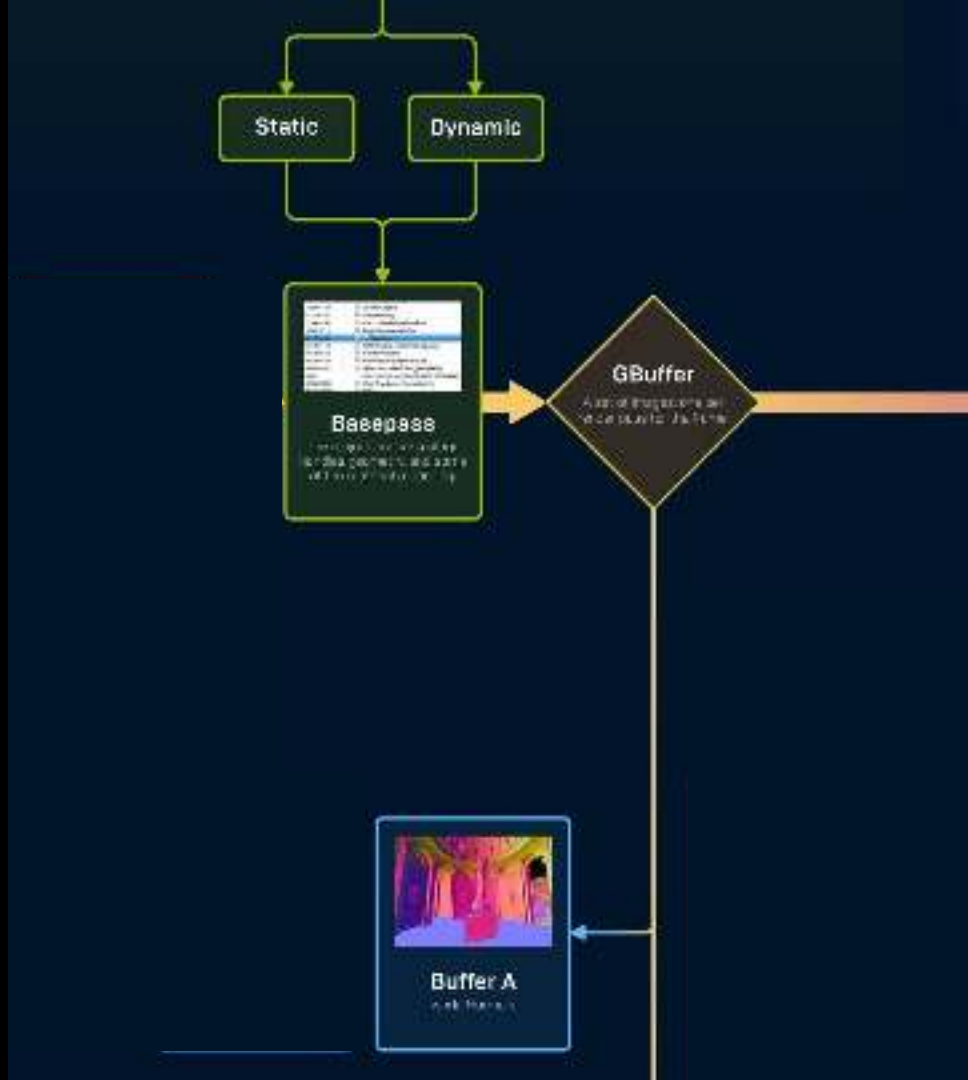
메시 정보와

실제 메시가 사용하는 머티리얼을 사용해

Early-Z를 통과한 픽셀에 한해

머티리얼 속성을 GBuffer A~E에 렌더링

➔ 후에 라이팅 패스에서 텍셀내 머티리얼 속성을 읽어와 라이팅 평가



프레임 버퍼

< “중세 게임 환경” 샘플 스크린샷 >

GBufferA ~ D

R 채널	G 채널	B 채널	A 채널	
Encoded World Normal			Per-Object Data	GBufferA(RGB10A2)
Metalic	Specular	Roughness	ShadingModeId / SelectiveOutputMask	GBufferB(RGBA8)
Base Color			AO	GBufferC(RGBA8)
CustomData				GBufferD(RGBA8)
Precomputed Shadow Factors				GBufferE(RGBA8)
Velocity				Scene Velocity (RGBA16)

< UE GBuffer 레이아웃 >

기존 렌더링 파이프라인

지오메트리 가시성 체크

뎁스 프리패스 Depth Prepass

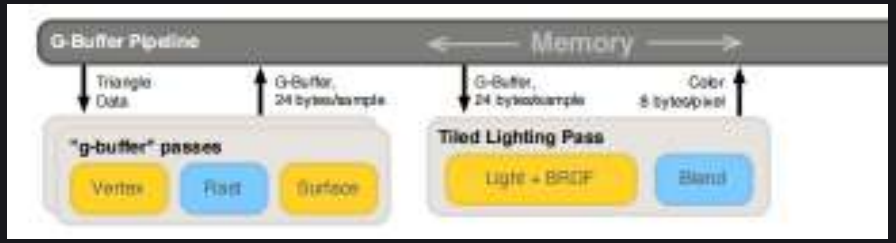
베이스 패스 Base Pass

라이팅 패스

...

기존 렌더링 파이프라인

머티리얼 다양성 및 복잡도 확대
GBuffer 읽기/쓰기 대역폭 증가



< 출처: https://habr.com/en/articles/655105/ >

예) QHD x 픽셀당 24바이트 x 60프레임 x 읽기/쓰기 2회
= 초당 약 10GB 대역폭 사용

기본 설정의 GBuffer와 가장 간단한 렌더러 가정

기존 렌더링 파이프라인

뎁스 프리패스 오버드로

모든 메시는 최소 2번 렌더링 된다

- 뎁스 프리패스 + 베이스 패스

무거운 과정을 두번이나 해야한다

픽셀 쿼드 비효율적

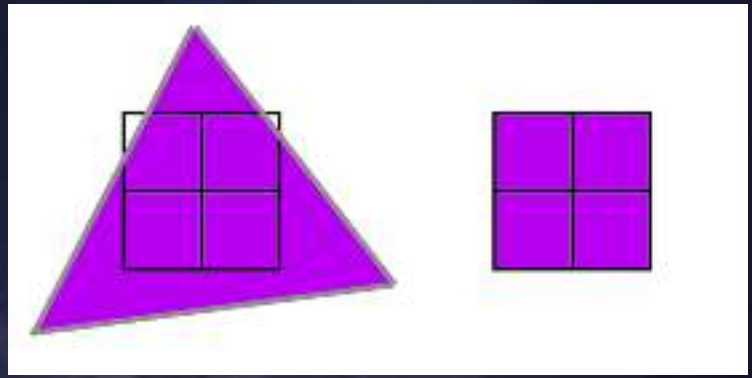
고밀도 메시의 경우, 래스터라이제이션 시 픽셀 쿼드 효율이 떨어진다

머티리얼 평가시 오버드로

래스터화 중 셰이더 변경

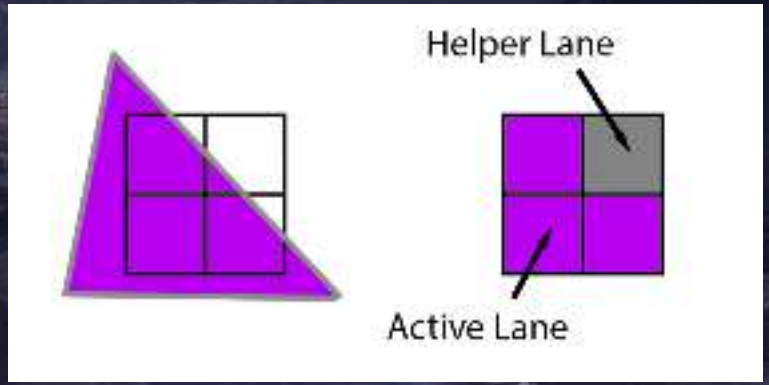
픽셀 쿼드

- 픽셀 셰이딩시 2x2 그룹으로 동작
- 2x2 그룹 = 쿼드
- 쿼드내 동일 셰이더 활용
- 텍스처 읽기시 4 UV 비교하여 밉맵 레벨 결정
- GPU에서 손쉽게 ddx/ddy/fwidth 유도

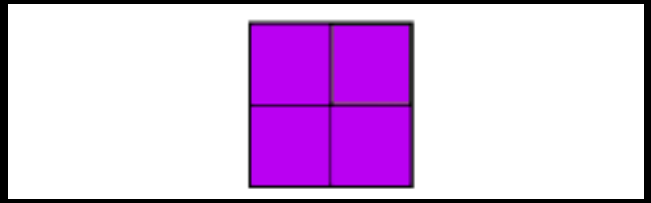


픽셀 쿼드 유틸라이제이션

- 액티브 레인 / 헬퍼 레인
- Extrapolation 후 동작

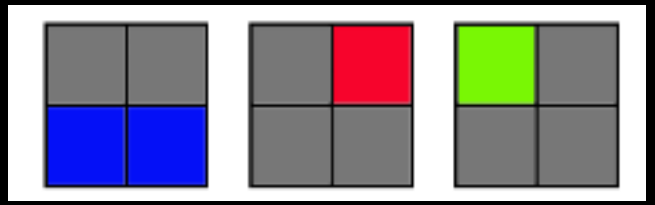


마이크로 폴리곤 픽셀 쿼드 효율

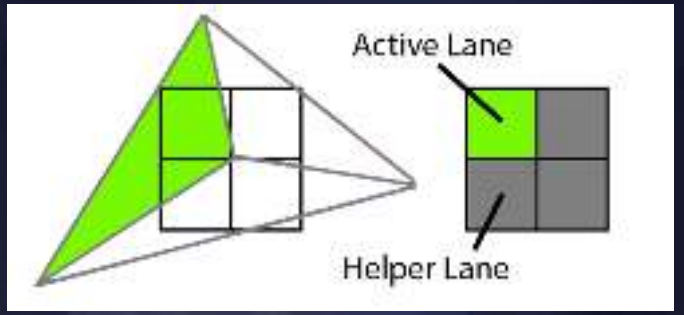


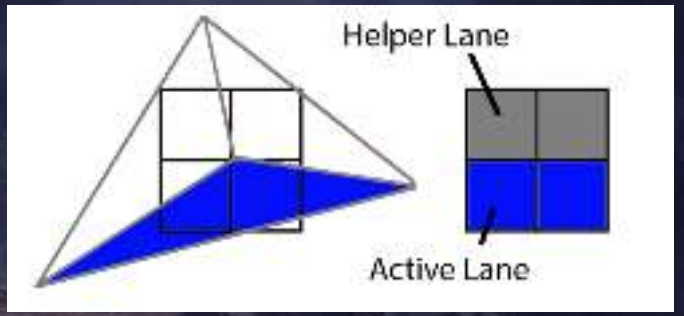
100%

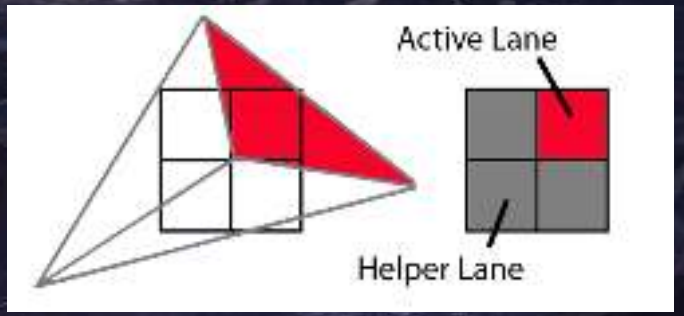
VS



33%







기존 렌더링 파이프라인

뎁스 프리패스 오버드로

모든 메시는 최소 2번 렌더링 된다

- 뎁스 프리패스 + 베이스 패스

무거운 과정을 두번이나 해야한다

픽셀 쿼드 비효율적

고밀도 메시의 경우, 래스터라이제이션 시 픽셀 쿼드 효율이 떨어진다

머티리얼 평가시 오버드로

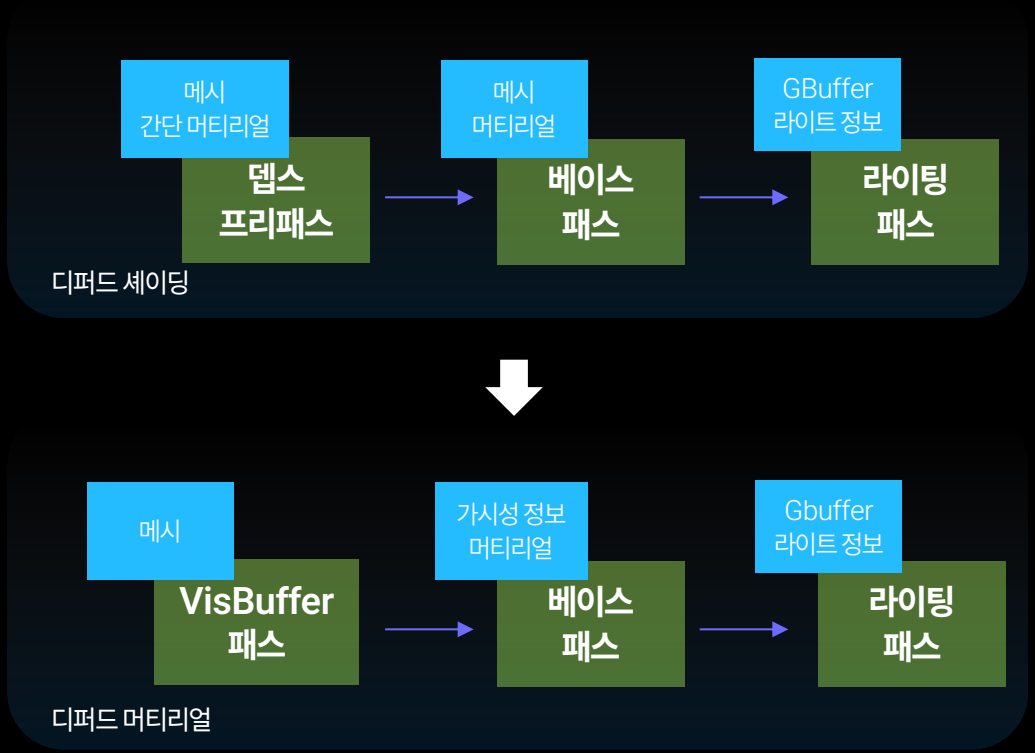
래스터화 중 셰이더 변경

디퍼드 머티리얼

머티리얼에서 가시성 분리

기존 디퍼드 셰이딩
- 베이스 패스는 Early-Z를 통과한 픽셀에 한해 머티리얼 평가

새로운 디퍼드 머티리얼
- 메시의 가시성 검사 이후 머티리얼 평가 진행
- 무거운 고밀도 메시는 한번만 처리



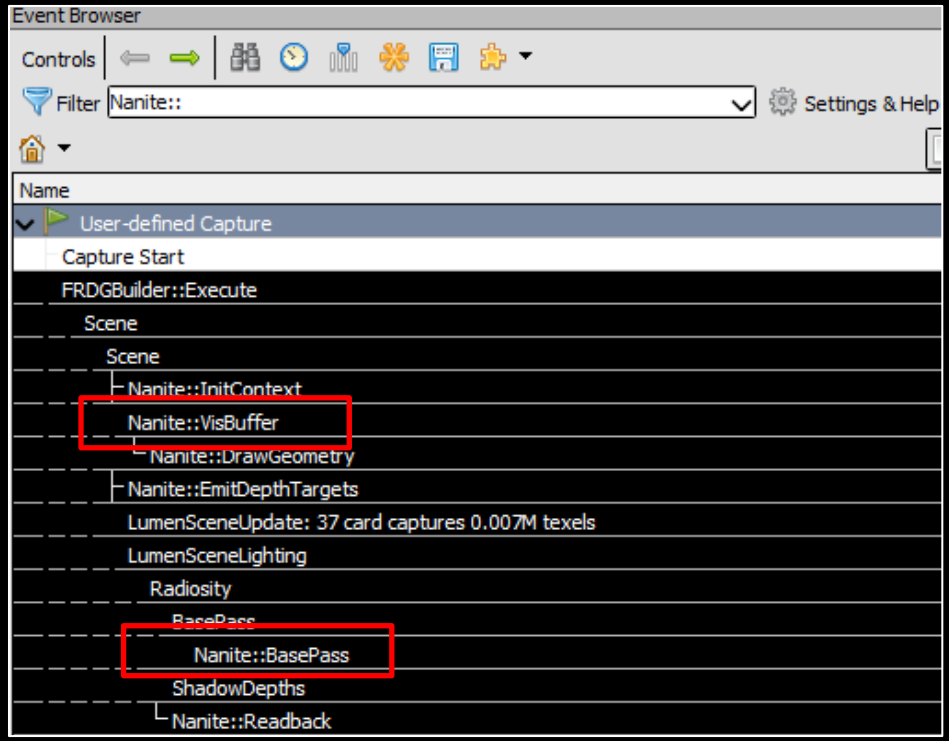
디퍼드 머티리얼

머티리얼에서 픽셀 가시성 분리

실제 렌더링 흐름

나나이트 VisBuffer
- Visibility Buffer 출력

나나이트 BasePass
- Visibiliby Buffer 활용
- GBuffer 출력



디퍼드 머티리얼

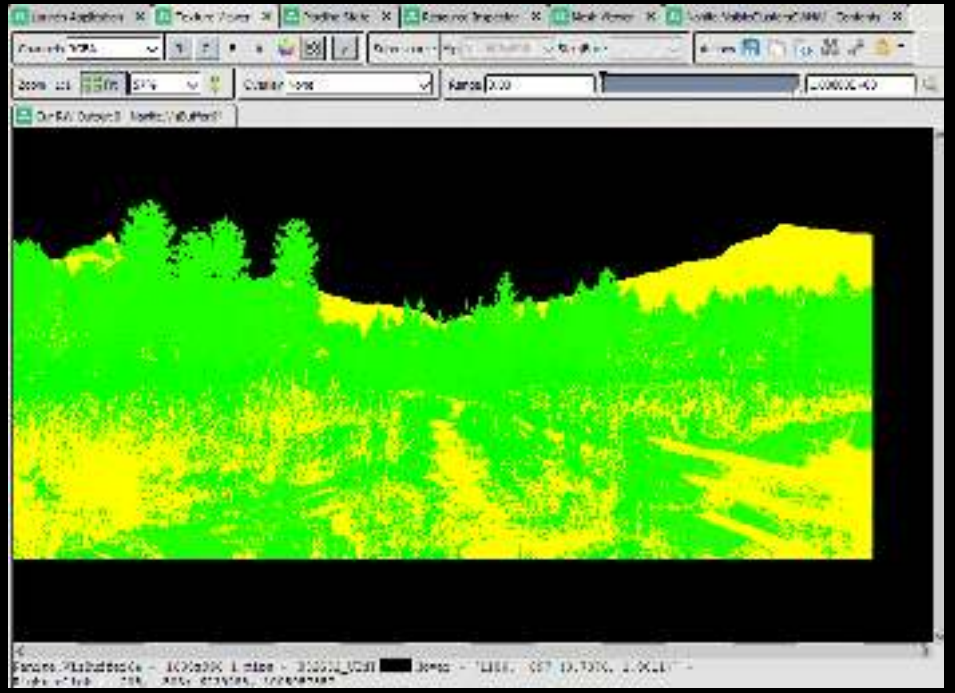
머티리얼에서 픽셀 가시성 분리

나나이트 VisBuffer

- 클러스터 ID + 트라이앵글 ID + 뎁스

나나이트 BasePass

- VisBuffer 정보 + 클러스터내 패킹된 머티리얼 정보를 통해 머티리얼 ID 유도



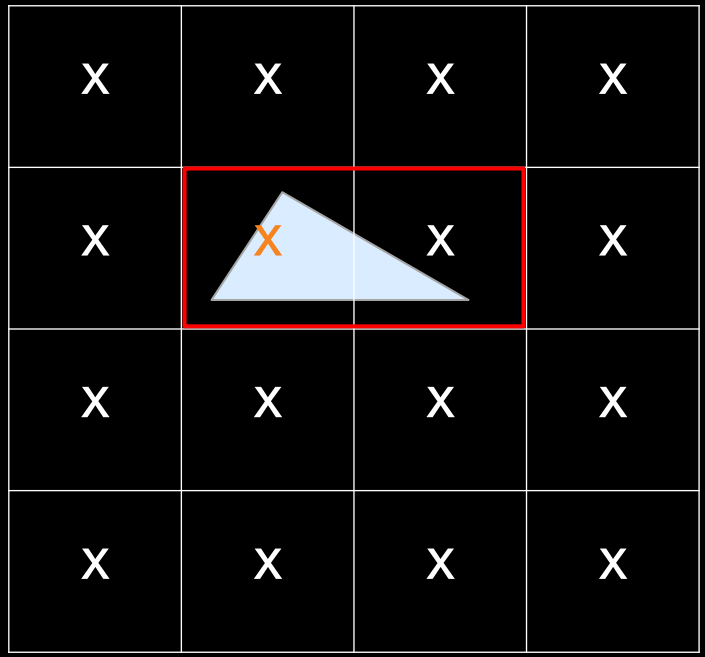
VisBuffer64 / R32G32_UINT

마이크로폴리 래스터라이저

하드웨어 가속/최적화 X
…성능?
SW 래스터라이저로 3배 이상 빠른 속도

SW 래스터라이저
- 적은 픽셀을 커버하는 많은 트라이앵글에 특화
- 픽셀 크기 트라이앵글

⇔ HW 래스터라이저
- 많은 픽셀을 커버하는 커다란 트라이앵글에 특화
- 일반적인 트라이앵글 크기 = 10픽셀 크기



하이브리드 래스터라이저

클러스터에 따라 HW/SW 선택

- 클러스터 변의 길이가 32픽셀 이하면 소프트웨어 래스터라이저
- 픽셀 크랙 X = DX의 래스터라이제이션 룰에 맞추어 SW도 제작

- 하드웨어 래스터라이저
- SW와 동일한 동작

= 뎁스 테스트 X + 렌더타겟이 아닌 UAV에 저장

나나이트 시각화 래스터모드
파란색 : SW / 빨간색 : HW

고해상도 지오메트리

- 클러스터 방식
- 메시 전체가 아닌 클러스터 단위 연산

클러스터

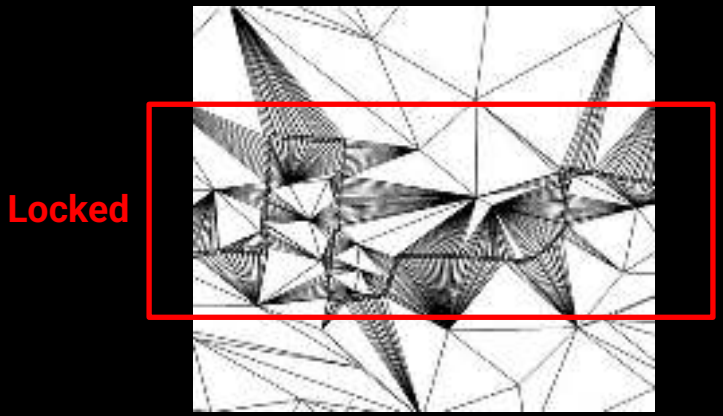
- 트라이앵글 묶음
- 1 클러스터 = 128 트라이앵글
- 지원: 클러스터 LOD + 클러스터 컬링

클러스터 LOD

- 끊김없는 LOD 전환
- 크랙이 없음

일반적인 메시 단순화 방식

- 외곽선 등 필요한 엣지를 Lock
- Locked 엣지를 포함한 폴리곤 정점 줄이기



클러스터 LOD

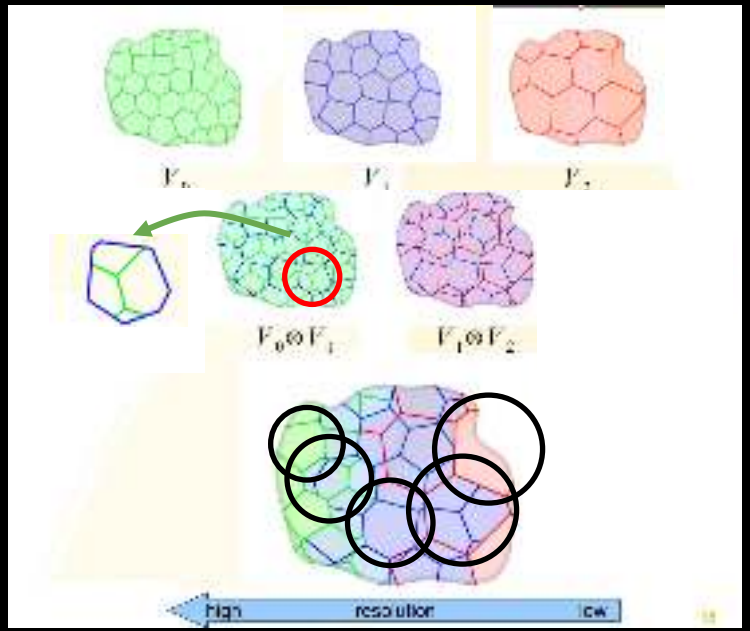
나나이트 클러스터 LOD 방식

- 클러스터 엣지에 제한 X
- LOD 레벨마다 전혀 다른 클러스터 구조

예시) LOD간 자연스러운 전환

- V0, V1, V2 = 디테일 레벨
- V1ⓧ V2 = V2의 외곽선 V1 나누어짐
- V0ⓧ V1 = V1의 외곽선 V0 나누어짐

- 왼쪽으로 가면서 낮은 LOD부터 높은 LOD로 전환
- V2 ➔ V1ⓧ V2 ➔ V1 ➔ V0 ⓧ V1 ➔ V0

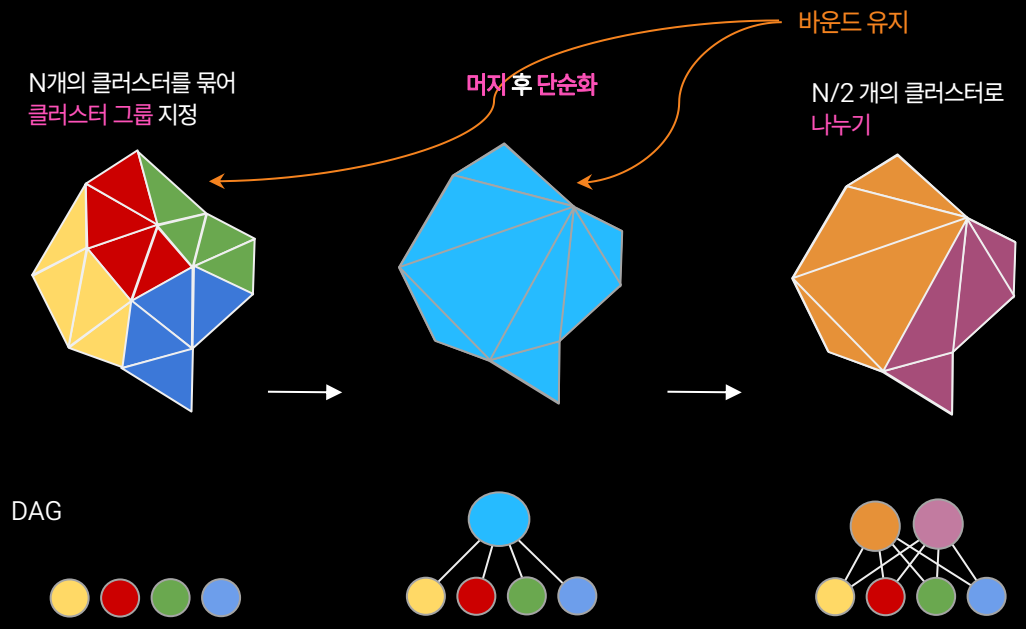


< 출저: http://vcg.isti.cnr.it/Publications/2005/CGGMPS05/Slide_BatchedMT_Vis05.pdf >

클러스터 LOD

클러스터 빌드

- 순서: Leaf에서 Root로
- 클러스터 그룹 선정
- 머지 ➔ 단순화 ➔ 나누기 반복
- 클러스터 그룹의 바운드 유지



클러스터 LOD

클러스터 빌드

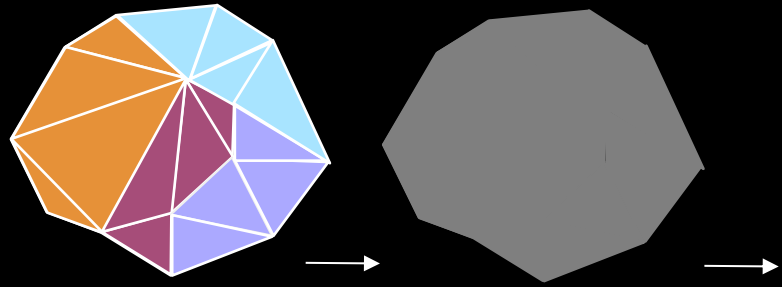
- 순서: Leaf에서 Root로
- 클러스터 그룹 선정
- 머지 ➔ 단순화 ➔ 나누기
- 클러스터 그룹의 바운드 유지

- 루트가 하나의 클러스터가 될때까지 반복

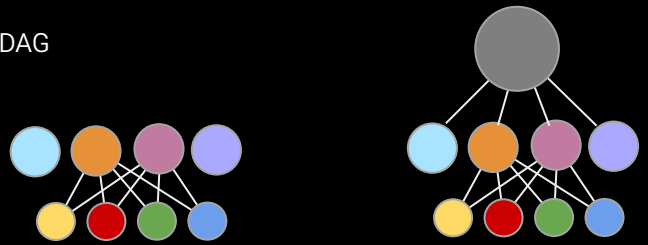
N개의 클러스터를 묶어
클러스터 그룹 지정

머지 후 단순화

N/2 개의 클러스터로
나누기



DAG



클러스터 LOD

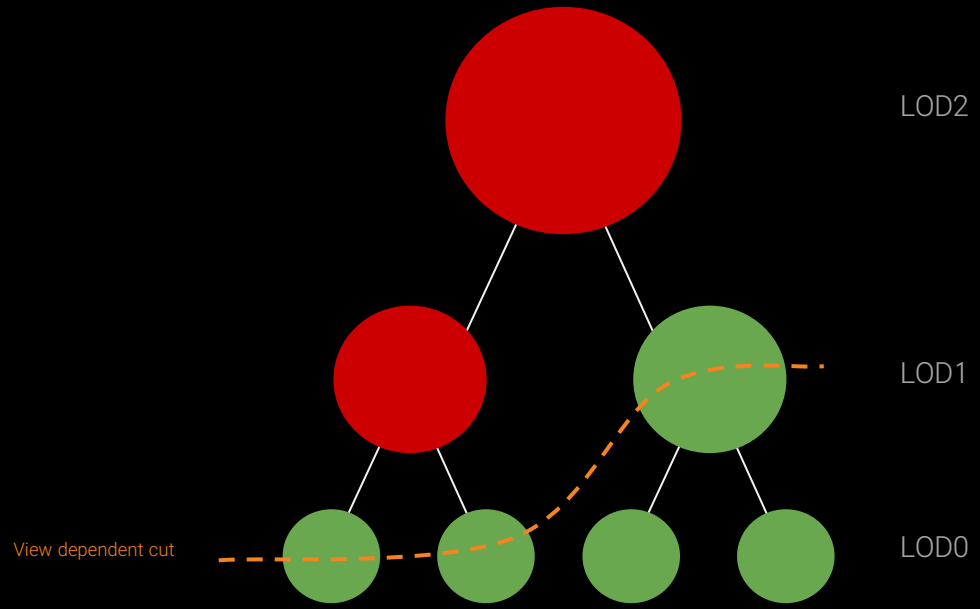
클러스터 LOD 선택

- LOD 선별 기준?
- 비교: 단순화 과정 중 계산한/저장된 클러스터별 에러 vs. 스크린 스페이스 에러

- 부모 노드의 에러가 너무 큰가?
- 현재 노드의 에러는 충분히 작은가?

클러스터 그룹끼리 다른 LOD 선택?

- 크랙 발생!
- 그룹내 클러스터가 모두 동일한 클러스터 LOD를 갖도록 그룹내 “클러스터 에러”를 동일하게 설정

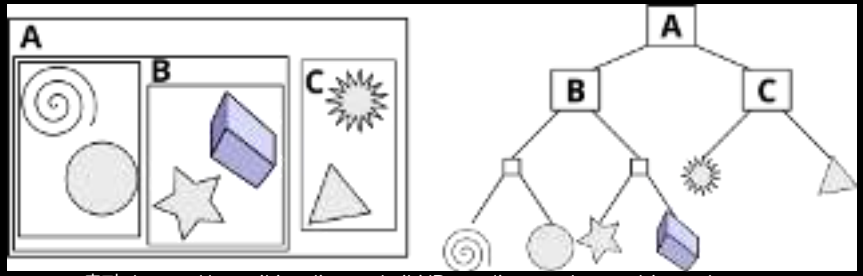


클러스터 LOD

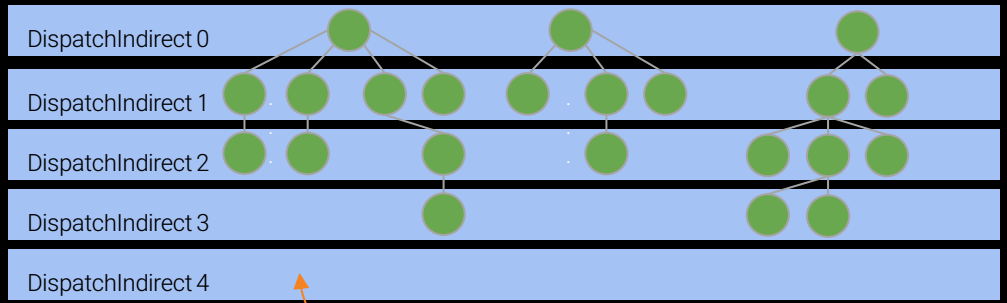
클러스터 LOD 선택
- 클러스터 그룹이 동일한 LOD 레벨을 갖어야 한다
- Hierarchical Culling 과정에서 결정

계층 컬링 Hierarchy Culling
- DAG 탐색 복잡 ➔ BVH 도입
- 모든 클러스터 바운드 + 에러 정보 저장

- 퍼시스턴트 스레드 활용
- 부모에서 시작, 테스트 후 자식을 잡 큐에 추가



< 출저: https://en.wikipedia.org/wiki/Bounding_volume_hierarchy >



CPU는 얼마나 많은 레벨이 필요한지 알수 없다.
비어있는 Dispatch가 실행될 가능성 O

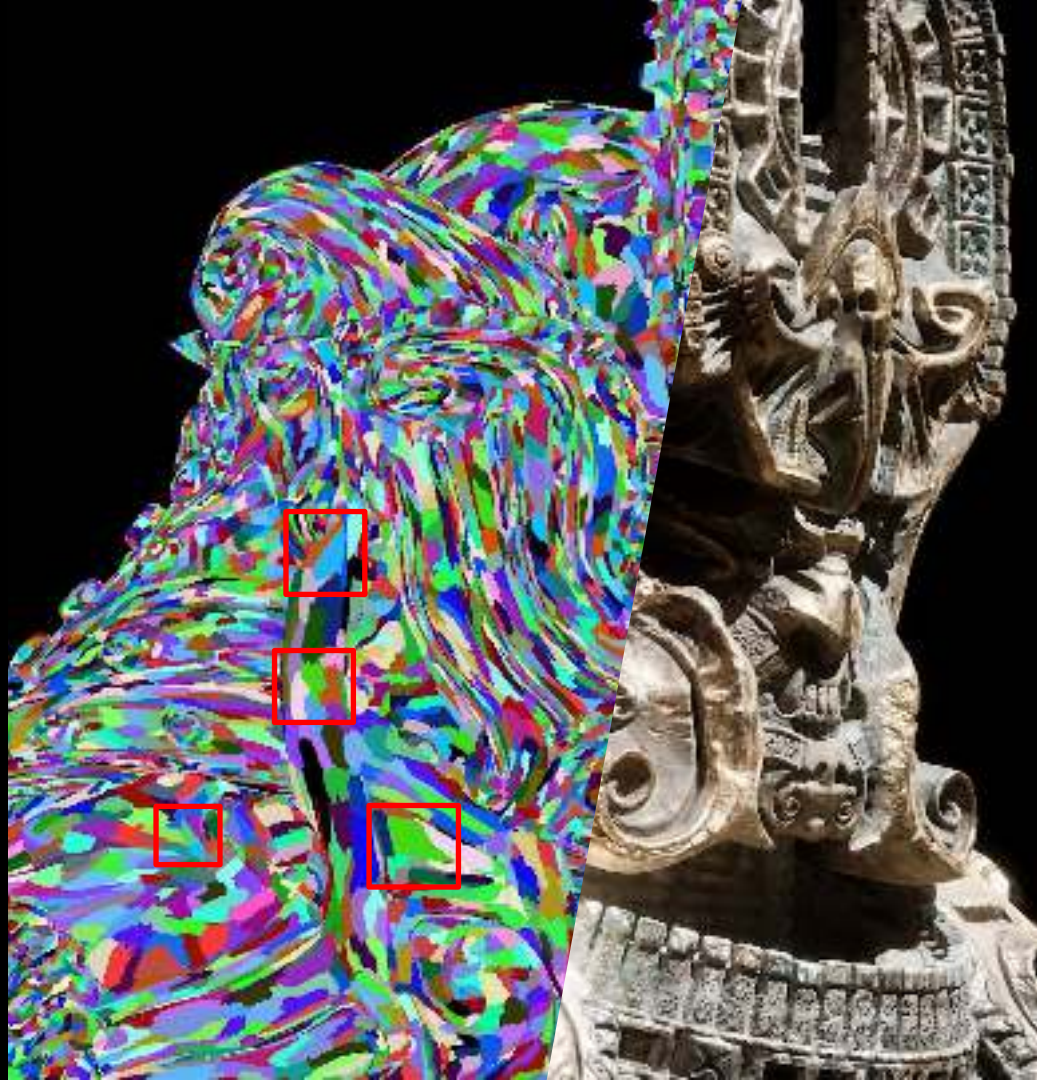
클러스터 컬링

오클루전 컬링

- 메시가 아닌 클러스터별 바운딩 박스
- 메시 = 여러 클러스터의 집합
- 생각보다 엄청나게 많은 클러스터

※ 클러스터 후면 제거 X

- 후면은 클러스터 컬링 단계에서 제거



클러스터 컬링

Hierarchical Z-Buffer (HZB)

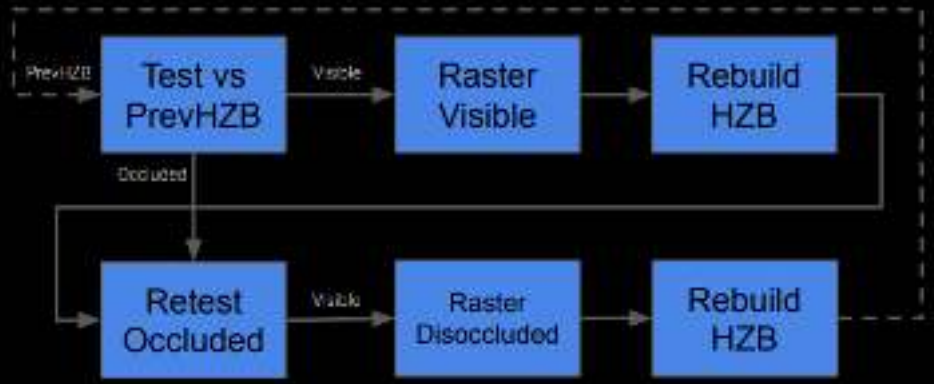
- 클러스터 바운드의 스크린 크기 계산
- 스크린 크기가 4x4 보다 적은 밉맵 선택
- 뎁스 테스트

클러스터 컬링

2 패스로 진행

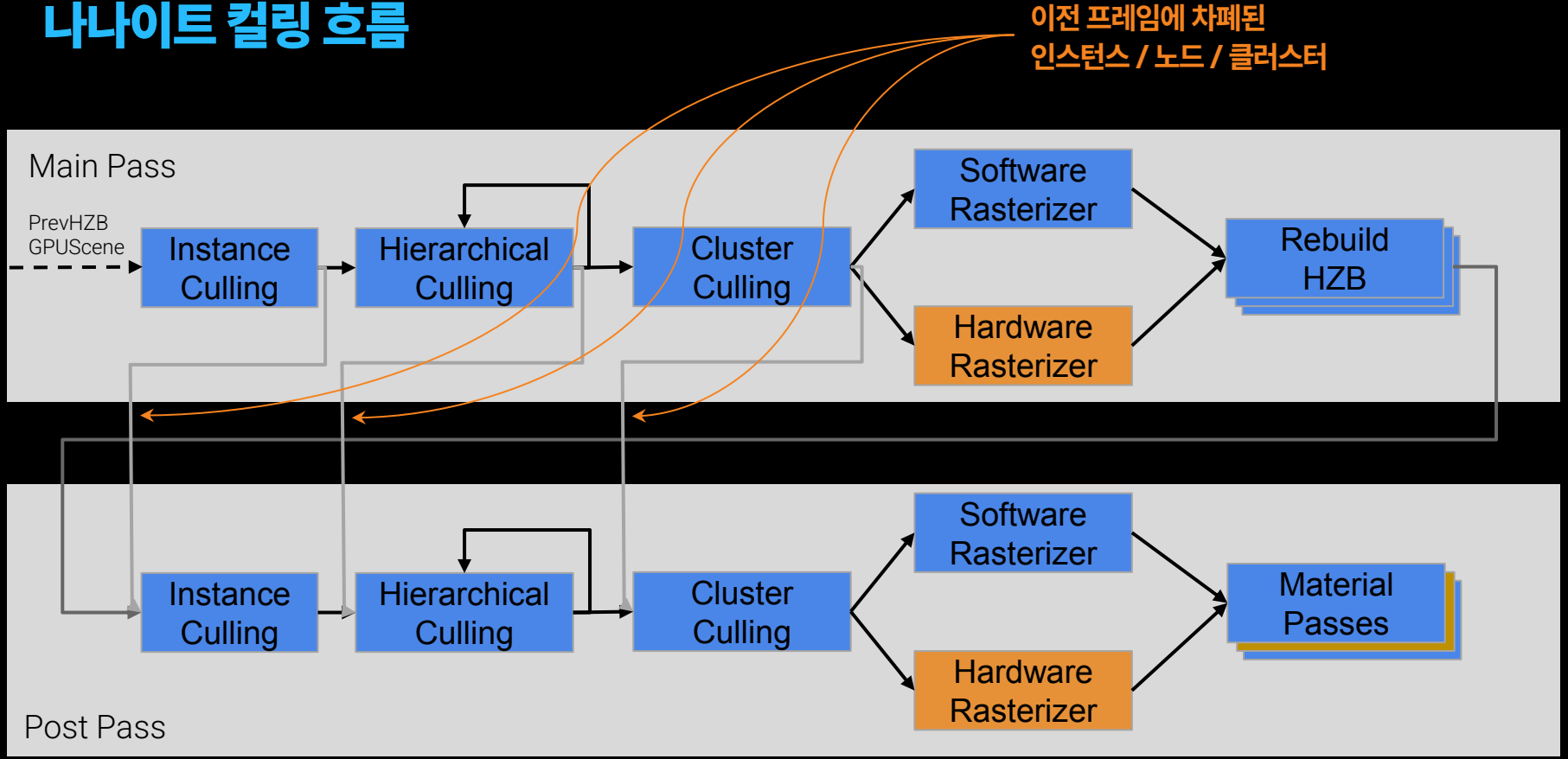
- 앞 프레임에서 보이는 것이 현재에도 보일 가능성이 높다

- 앞선 프레임의 Visible Set을 유지하기 어렵다
- 2번에 나눠서 오클루전 컬링 진행



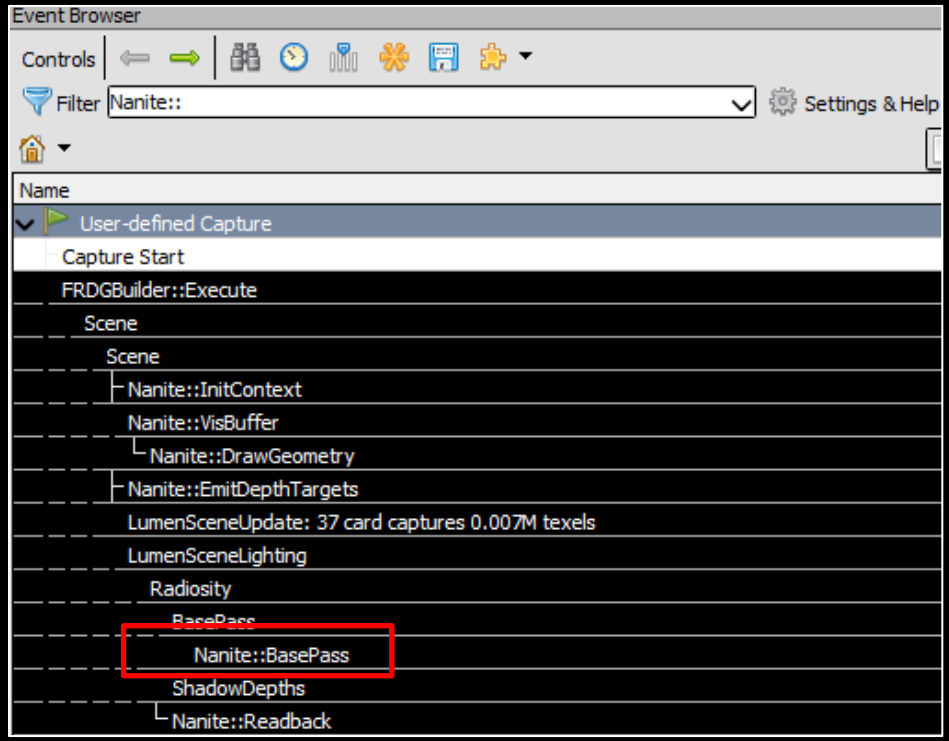
< 출저: A Deep Dive into Nanite Virtualized Geometry >

나나이트 컬링 흐름


이전 프레임에 차폐된
인스턴스 / 노드 / 클러스터
Main Pass
PrevHZB
GPUScene
Instance Culling
Hierarchical Culling
Cluster Culling
Software Rasterizer
Hardware Rasterizer
Rebuild HZB
Instance Culling
Hierarchical Culling
Cluster Culling
Software Rasterizer
Hardware Rasterizer
Material Passes
Post Pass

머티리얼 패스
= 디퍼드 머티리얼
= Nanite::BasePass

머티리얼 정보가 필요하다

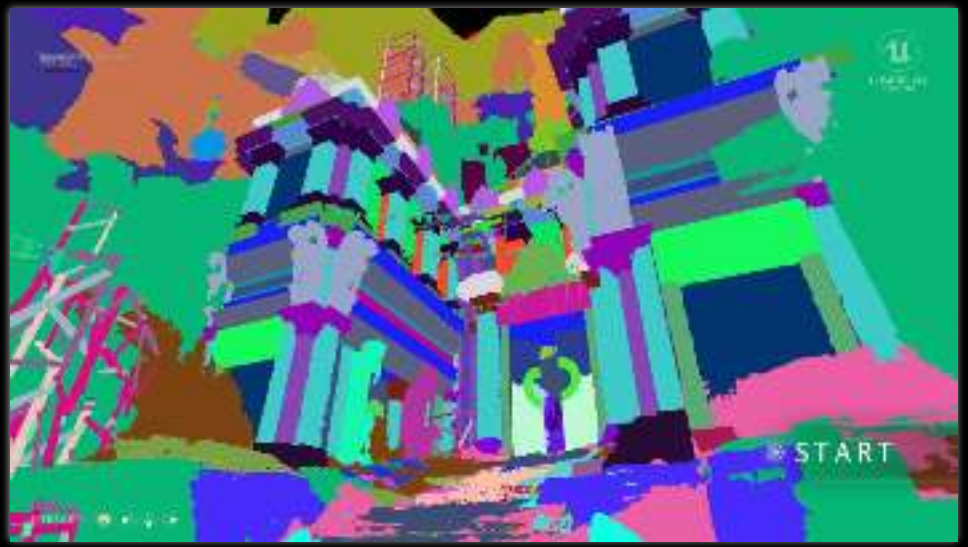


머티리얼 패스

어느 픽셀에 어떤 머티리얼?

- VisBuffer 디코드하여 머티리얼 정보 유도
- 클러스터 ID + 트라이앵글 ID + 뎁스

- 클러스터 ID ⇒ 클러스터 ⇒ 프리미티브 ID
- 클러스터 + 트라이앵글 ID ⇒ Material ID
- 프리미티브 ID + Material ID ⇒ 머티리얼 슬롯 ID



< 프리미티브 시각화 >

머티리얼 패스

머티리얼 셰이딩
- 머티리얼 당 스크린 쿼드 드로
- 스크린 쿼드 = 화면 크기의 사각형

머티리얼 컬링
- 모든 머티리얼에 대해 모든 픽셀에 대해
- 머티리얼 ID 동일한지 테스트

➔ 너무 느리다

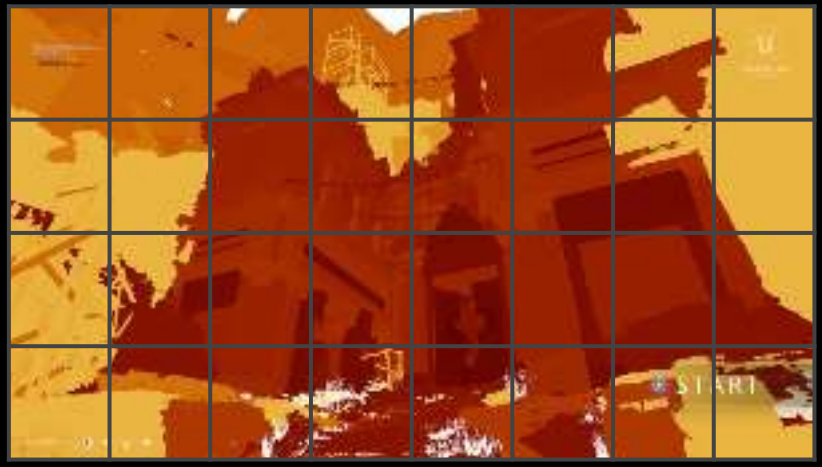
머티리얼 패스

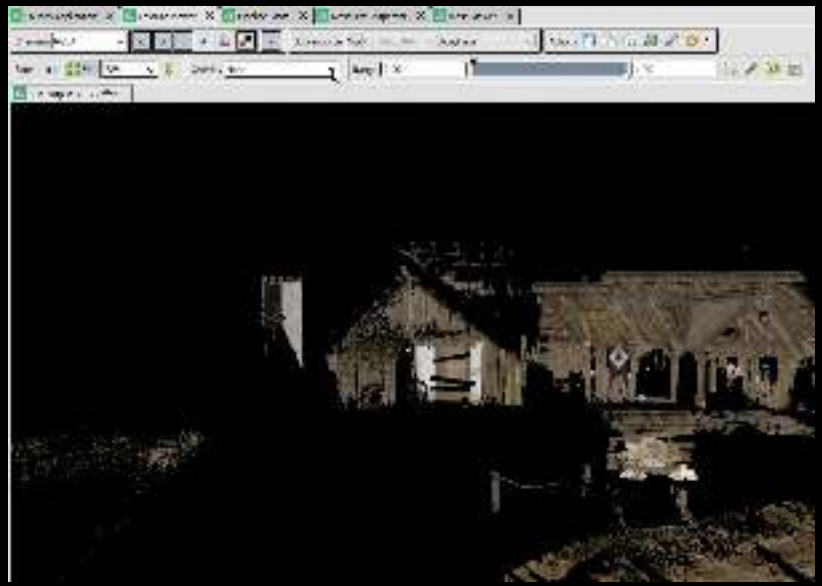
스크린 쿼드가 아닌
타일 단위 드로

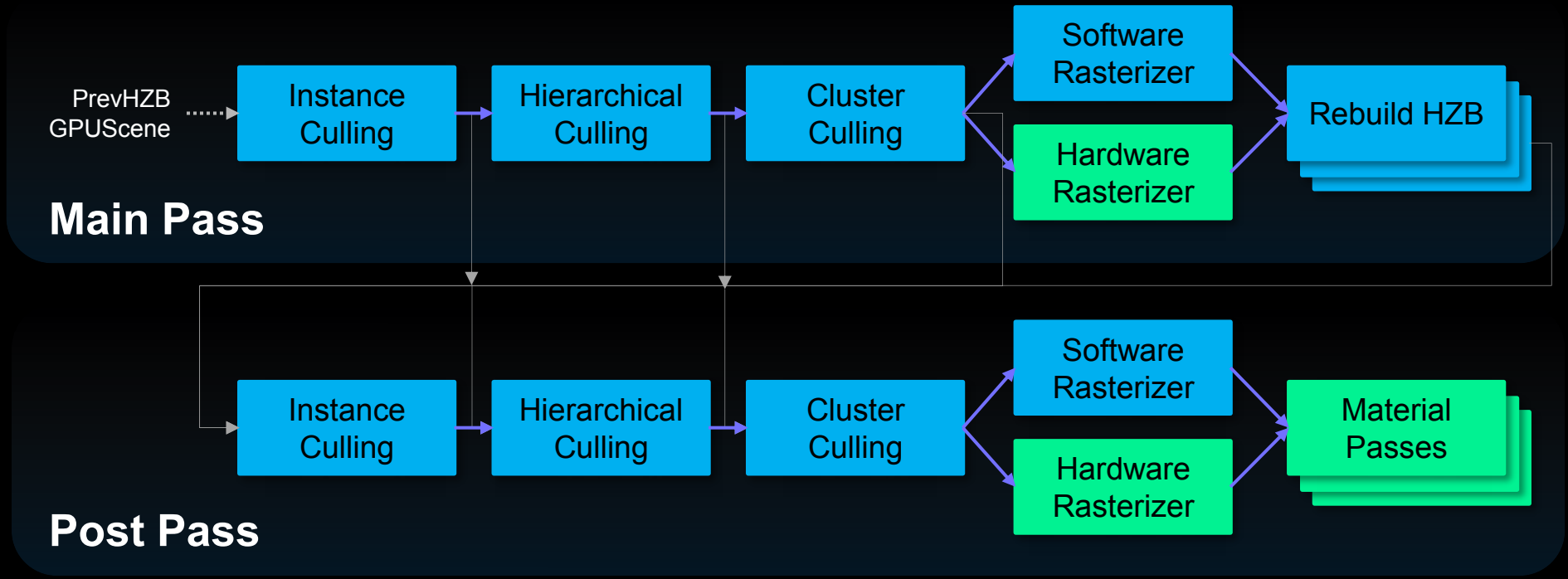
- 풀 스크린 쿼드가 아닌 8 x 4 타일 활용
- 타일 단위 컬링

머티리얼 깊이 버퍼

- 머티리얼 ID를 뎁스 값으로 활용
- 뎁스 테스트로 하드웨어 가속






PrevHZB
GPUScene
Instance Culling
Hierarchical Culling
Cluster Culling
Software Rasterizer
Hardware Rasterizer
Rebuild HZB
Main Pass
Instance Culling
Hierarchical Culling
Cluster Culling
Software Rasterizer
Hardware Rasterizer
Material Passes
Post Pass

프로그래머블 래스터라이저

모든 것의 나나이트화

나나이트 미지원
(5.0 기준)

Deformation, Displacement

Skinning

Masked

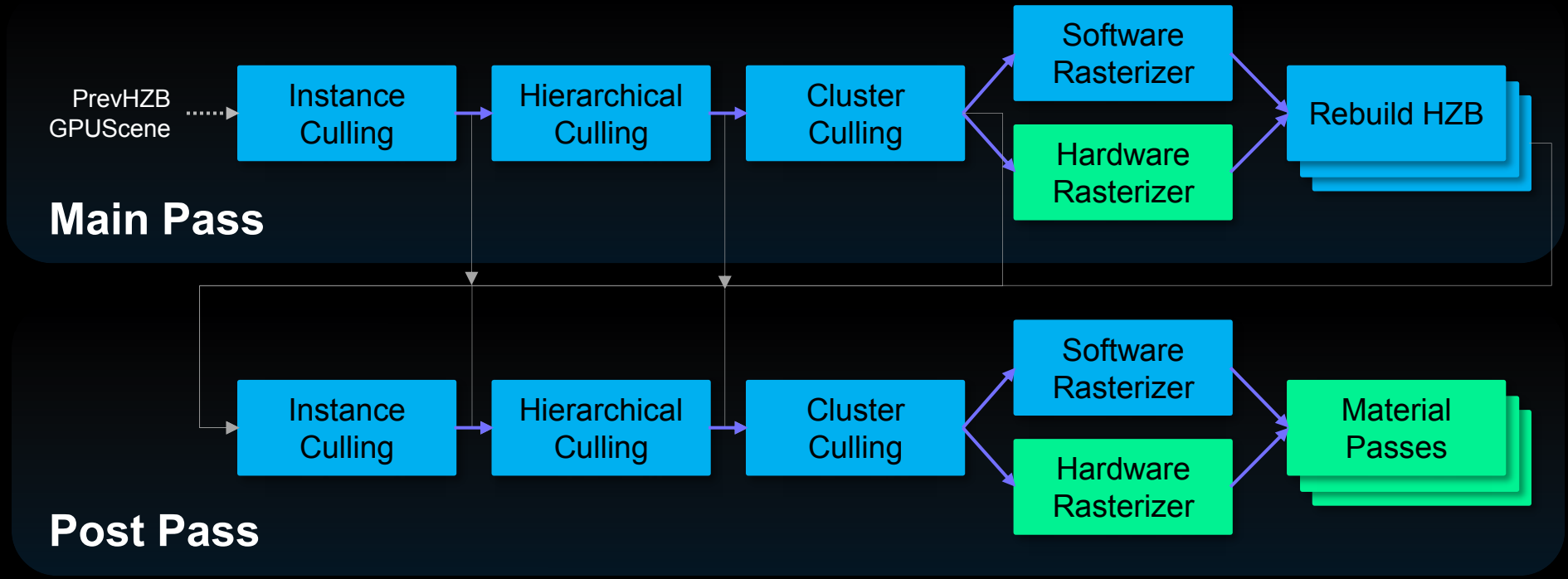
World Position Offset

Custom UVs

Pixel Depth Offset

Two-sided

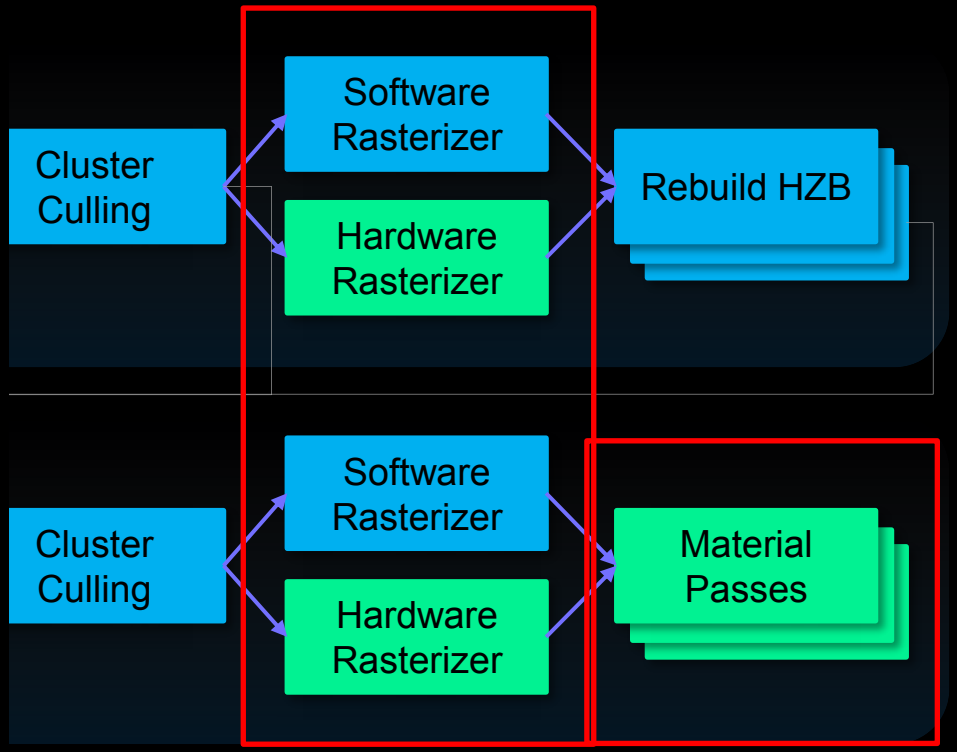
...


PrevHZB
GPUScene
Instance Culling
Hierarchical Culling
Cluster Culling
Software Rasterizer
Hardware Rasterizer
Rebuild HZB
Main Pass
Instance Culling
Hierarchical Culling
Cluster Culling
Software Rasterizer
Hardware Rasterizer
Material Passes
Post Pass

머티리얼 그래프 지원

UE5.0 나나이트
- 머티리얼 패스? 문제 없음
- 아티스트가 정의한 셰이더 그래프 지원

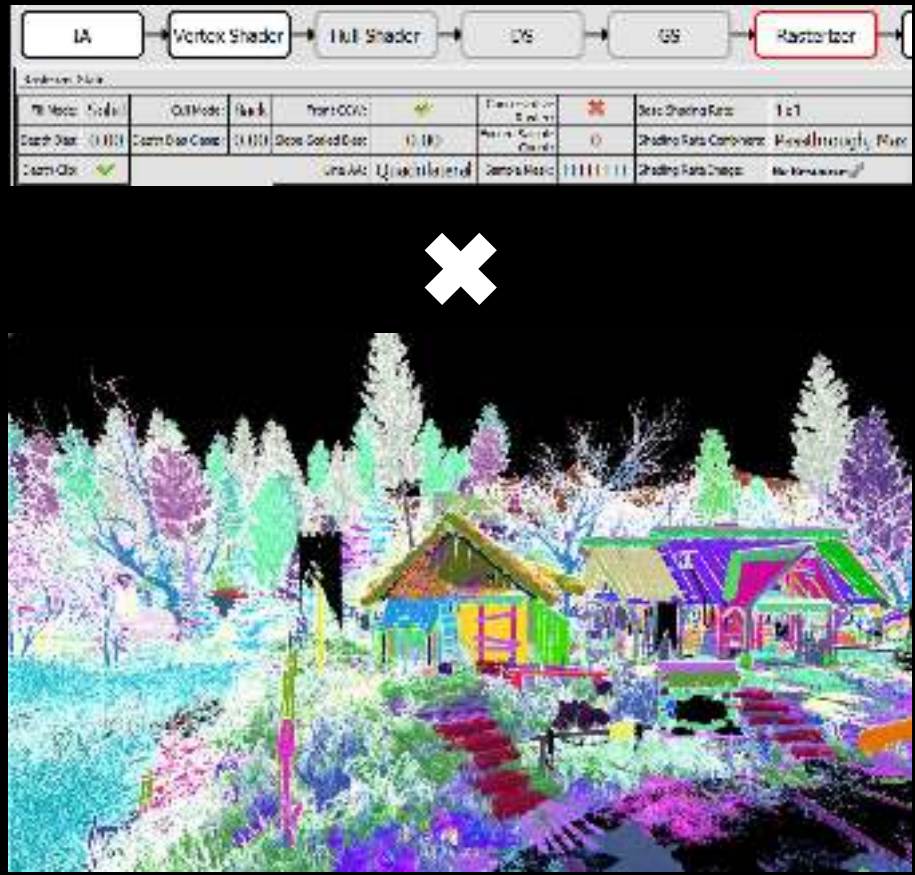
- 래스터라이저?
- 프로그래머가 하드 코딩한 글로벌 셰이더들
- 사실상 고정 함수



래스터라이저 기능 확대

머티리얼 그래프 주도 래스터라이저
- 래스터할 대상을 분류해서 모으자
- 분류? 셰이더와 래스터라이저 상태의 고유 조합

래스터 빈 Raster Bin
- 고유 조합별로 “분리 수거함”에 넣기
- 다양한/복잡한 셰이더 그래프를 지원하기 위한 사전 준비
- 그 후 래스터라이저에 전달



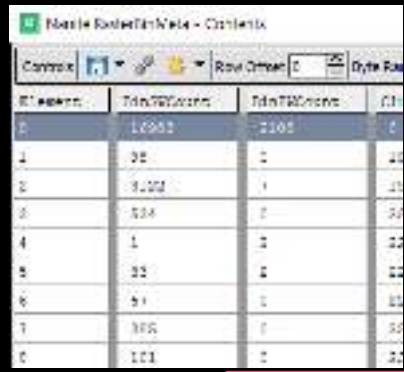
래스터라이저 기능 확대

래스터 비닝 Raster Binning

- 클러스터를 분류하기 위한 과정
- 3단계로 진행
- Count ➔ Reserve ➔ Scatter

- Count: 클러스터마다 참조하는 머티리얼 확인
- Reserve: 래스터 빈마다 글로벌 버퍼에 공간 확보
- Scatter: 클러스터마다 사용할 래스터 빈 분류

※ 하나의 클러스터에는 여러 머티리얼이 사용될 수 있다
= 하나의 클러스터가 여러 래스터 빈에 포함될 수 있다
= 하나의 클러스터가 여러번 래스터라이즈 될 수 있다



Type	Width	Height	Depth	Array Size
RWStructuredBuffer[2730]	65520	0	0	2730

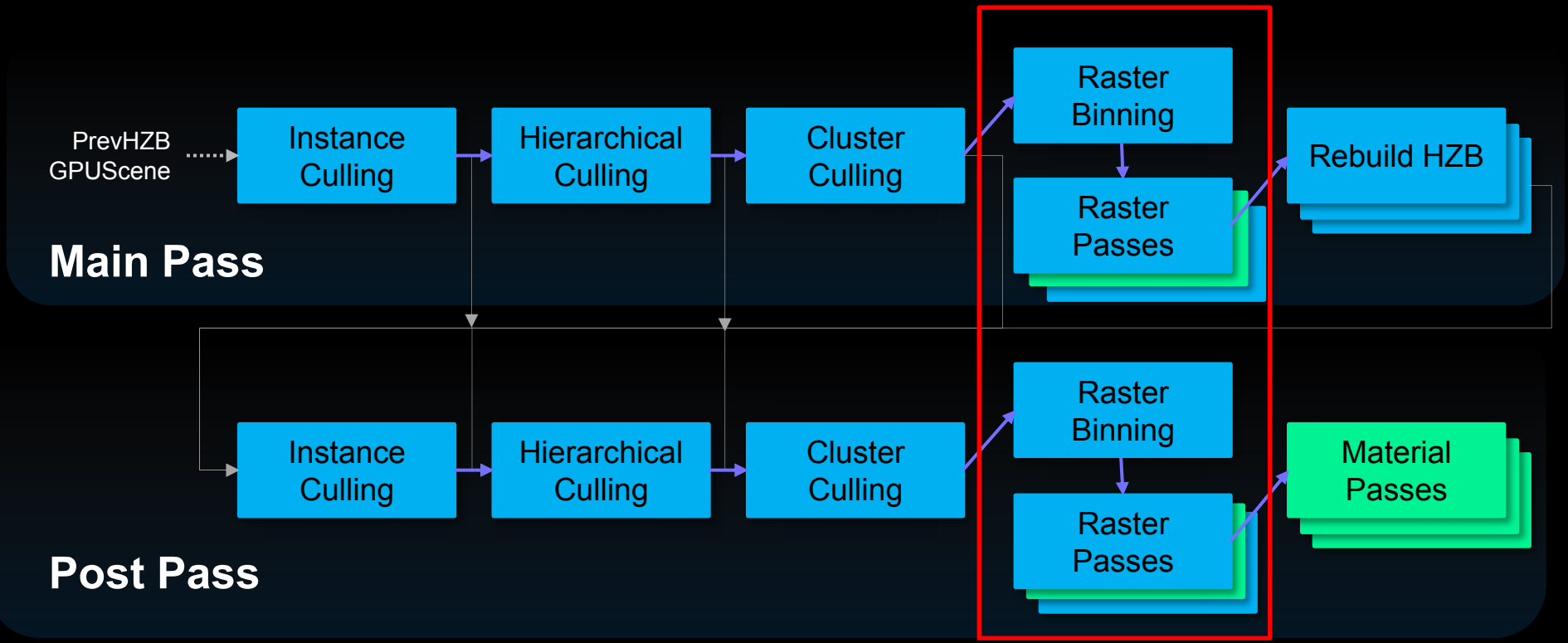
래스터라이저 기능 확대

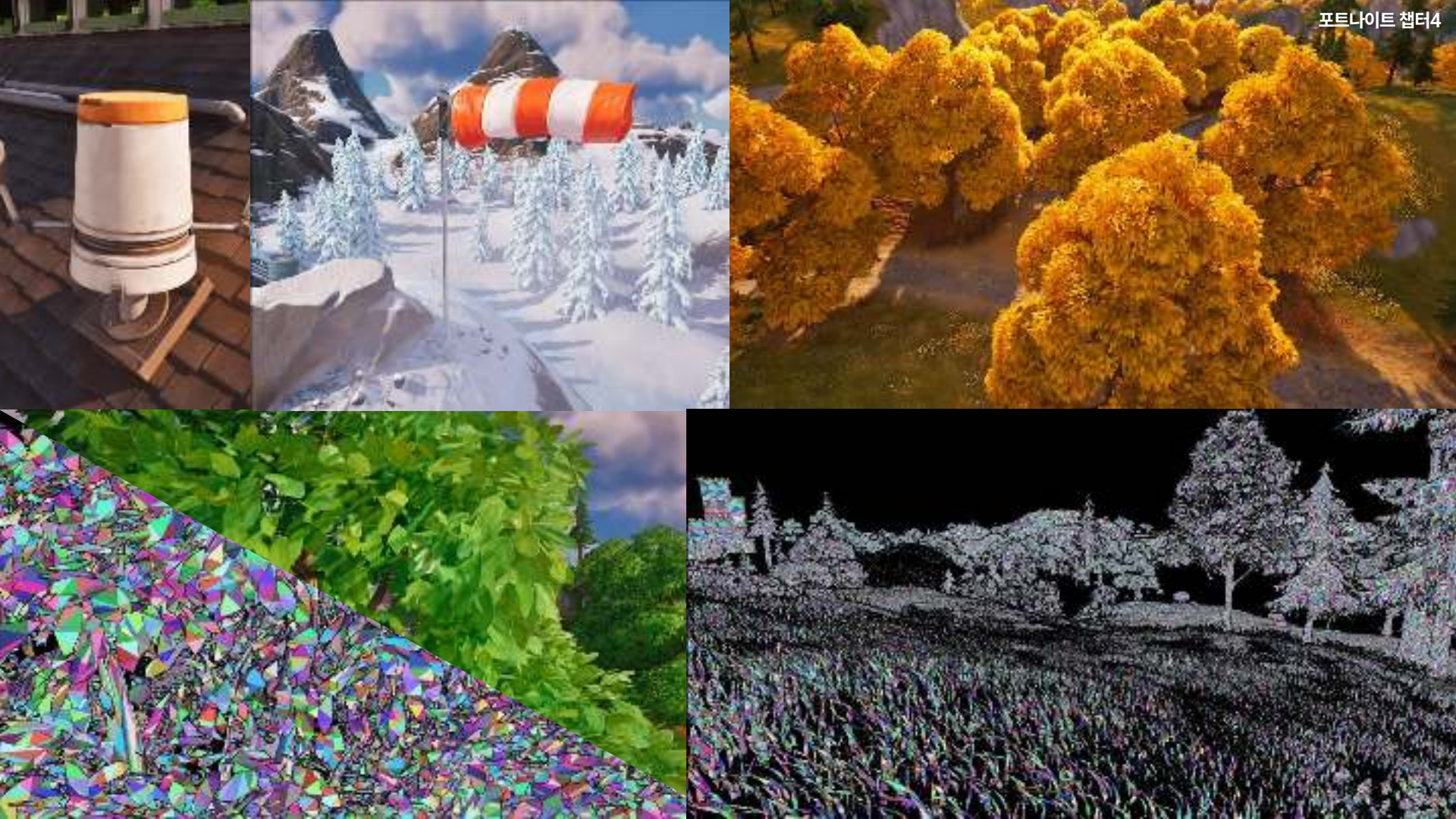
래스터라이저 셰이더

- 래스터라이저가 직접 머티리얼의 함수 호출

예) WPO 머티리얼

- 버텍스 셰이더가 Material 관련 셰이더 파라미터를 채운다
- 래스터라이저 셰이더가 함수를 호출해 WPO 정보에 접근한다


PrevHZB
GPUScene
Instance Culling
Hierarchical Culling
Cluster Culling
Raster Binning
Raster Passes
Rebuild HZB
Main Pass
Instance Culling
Hierarchical Culling
Cluster Culling
Raster Binning
Raster Passes
Material Passes
Post Pass


포트나이트 챕터4

스태틱/다이나믹 테셀레이션

스태틱 디스플레이스먼트
런타임 테셀레이션

나나이트
스태틱 테셀레이션

오프라인 적응형 테셀레이터

- 에디팅 타임, 나나이트 빌드시 사전 계산
- 텍스처 기반 접근

- 디스플레이스먼트 맵을 구워 트라이앵글 적응형 배치
- 에러가 가장 적은 최적화된 메시 생성
- 크랙 X = 동일한 위치를 갖는 버텍스의 평균 변위 사용

나나이트
다이나믹 테셀레이션

다이나믹 프로그래머블 디스플레이스먼트

- 런타임, 완전 동적
- UE4의 하드웨어 테셀레이션과 동일

- 그러나 더 고밀도로!
- UE5 나나이트 테셀레이션 = 2픽셀 크기 트라이앵글

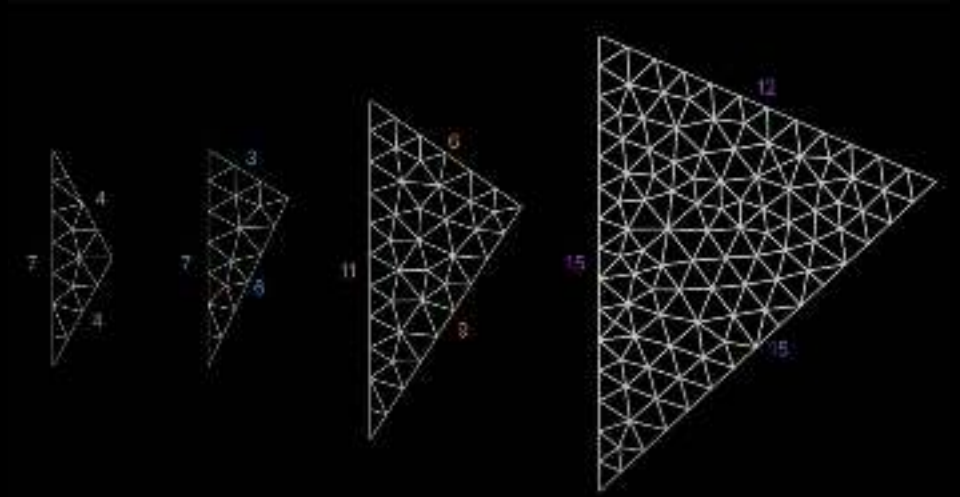
- 콘텐츠 적응형 X: 평평한 지오메트리도
  마이크로폴리곤으로 테셀레이션

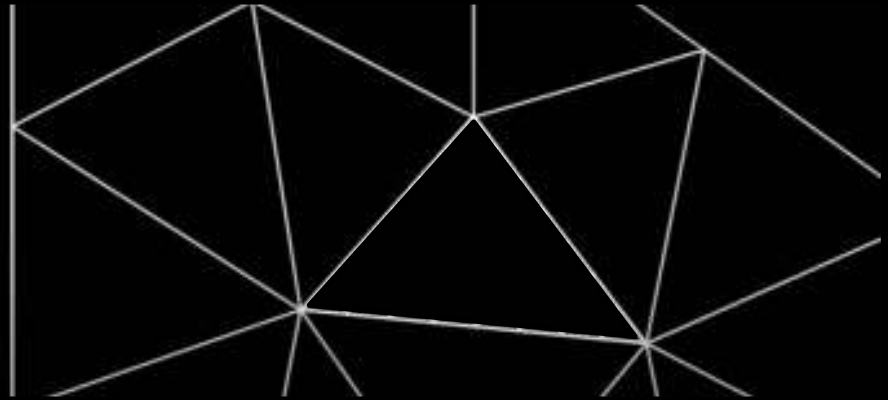
나나이트
다이나믹 테셀레이션

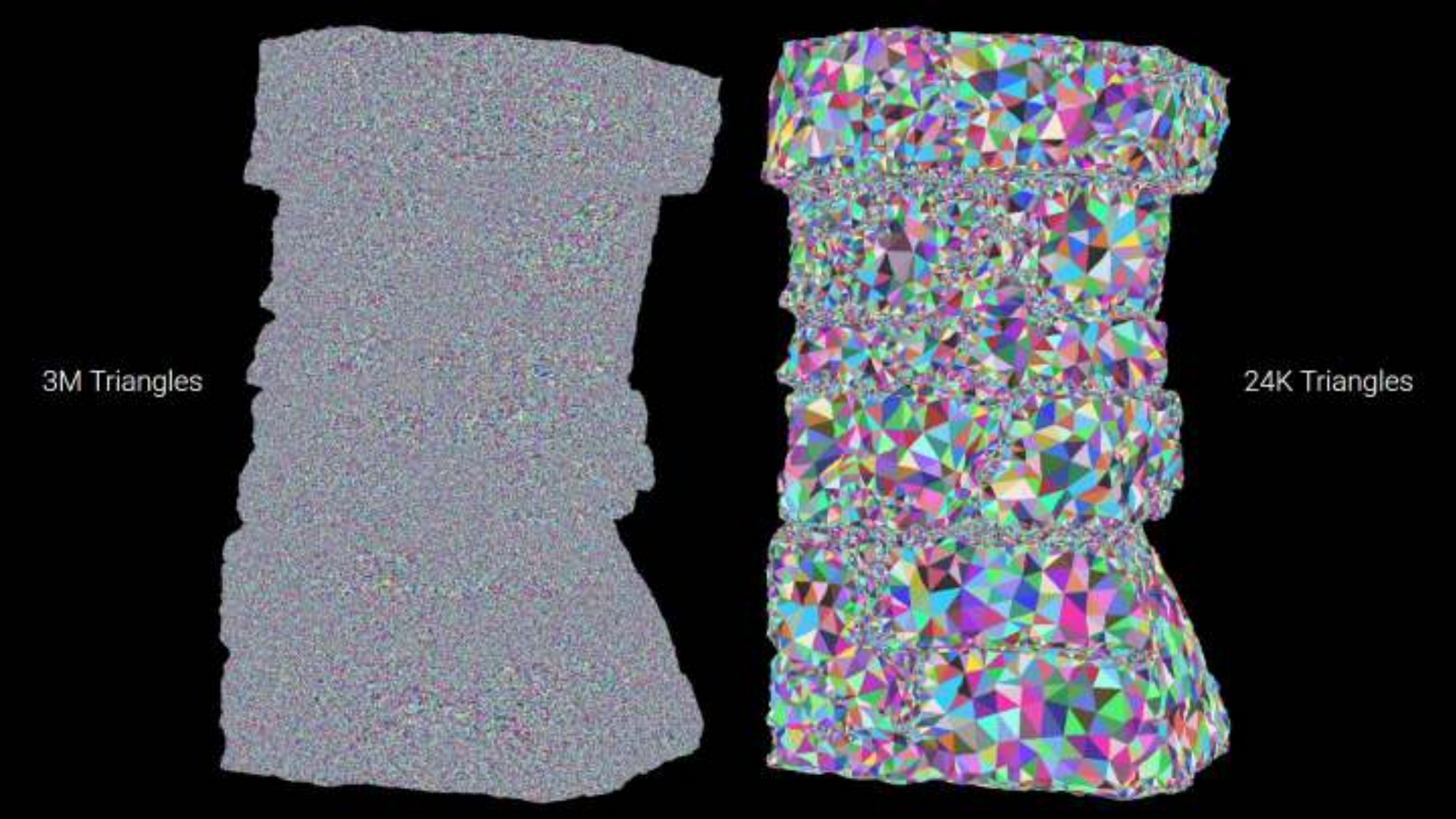
다이나믹 프로그래머블 디스플레이스먼트
- 어떻게? 테셀레이션 룩업 테이블

- 패치는 원래 메시의 커다란 트라이앵글
- 패치의 크기, 모양에 따라 사전 테셀레이션된 메시 탐색
- 패치를 해당 메시로 치환
- 그리고 디스플레이스먼트 적용

- 룩업 테이블에 없는 (커다란) 메시는?
- 해당 메시의 트라이앵글을 하나 찾아서
- 재귀적으로 메시 탐색 & 치환 반복



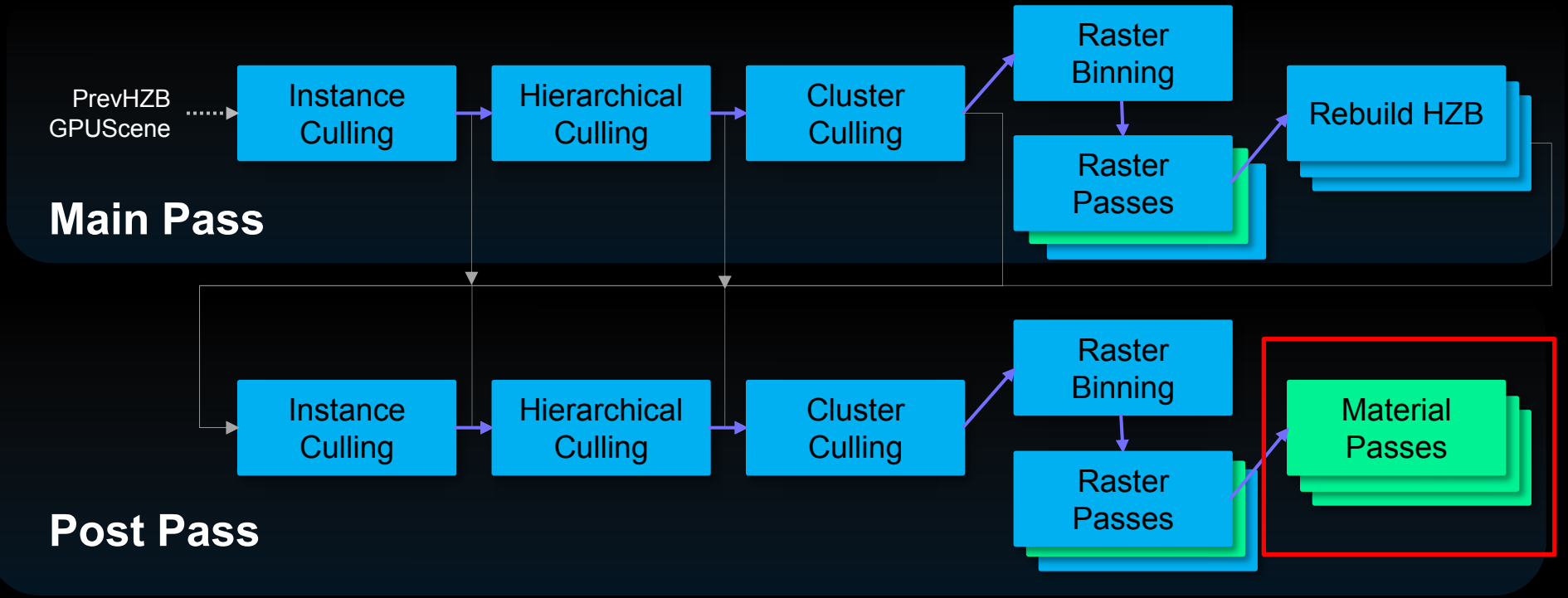



3M Triangles
24K Triangles


27.48MB Mesh
>6x
3.45MB Normal
0.38MB Displace
0.32MB Mesh
4.15MB Total

컴퓨트 머티리얼

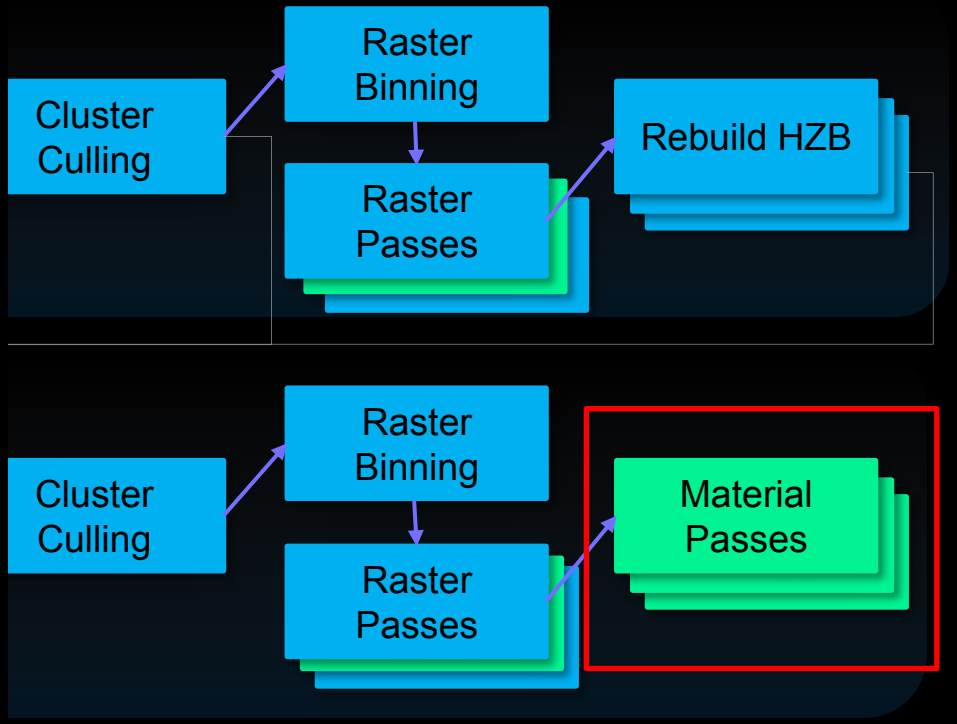
머티리얼 패스의 중요한 변경점


PrevHZB
GPUScene
Instance Culling
Hierarchical Culling
Cluster Culling
Raster Binning
Raster Passes
Rebuild HZB
Main Pass
Instance Culling
Hierarchical Culling
Cluster Culling
Raster Binning
Raster Passes
Material Passes
Post Pass

기존 머티리얼 파이프라인

스크린 쿼드를 통한 렌더링
- 픽셀 쿼드 비효율이 완전히 제거된 것이 아니다
- 연속적인 빈 드로 커맨드 발생. 오버헤드가 크다

래스터라이저에 집중
- Fast path 등 특화된 최적화로 인한 복잡성 증가
- 셰이딩의 비효율성은 그대로 방치

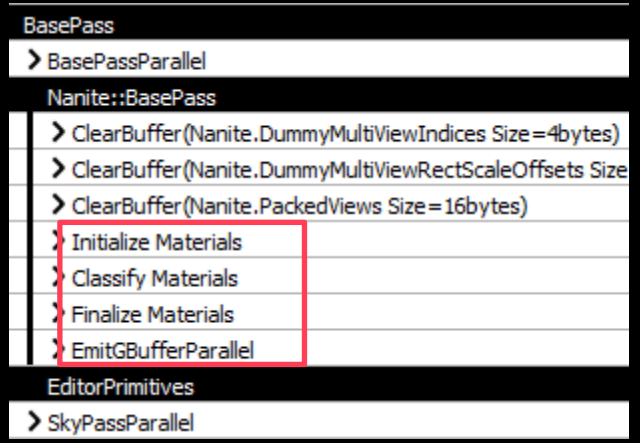


컴퓨트 머티리얼
= GBuffer를 컴퓨트 셰이더로!

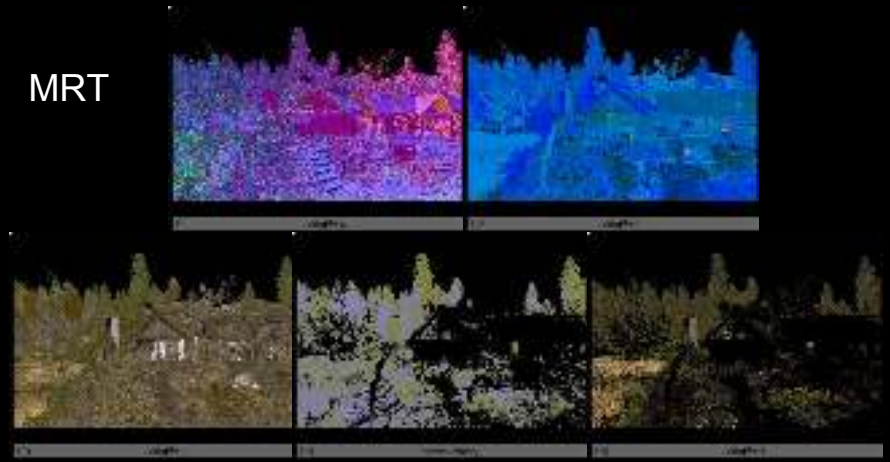
기존 디퍼드 방식
- 멀티 렌더 타겟(MRT) 기능 활용
- 동시에 여러 렌더 타겟에 출력
- 하드웨어 지원 / 고정함수

컴퓨트 셰이더
- 렌더 타겟이 아닌 UAV 출력
- GBuffer의 UAV로 변환하면 간단? 그렇지 않다

UE5.3
머티리얼 패스

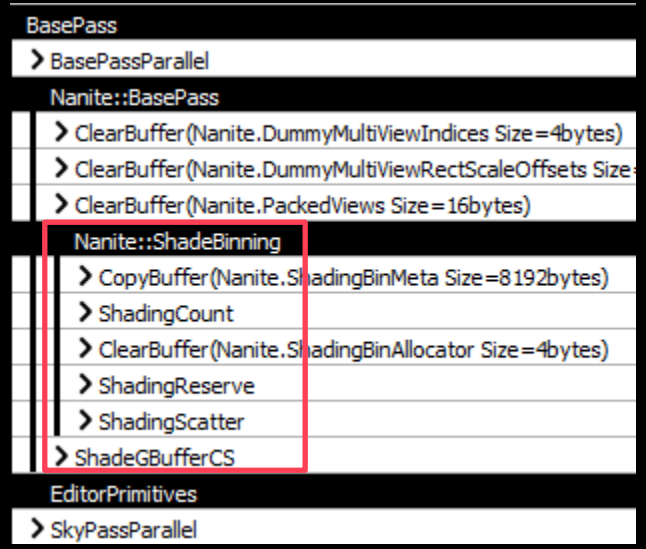


MRT



컴퓨트 머티리얼

UE5.4
머티리얼 패스



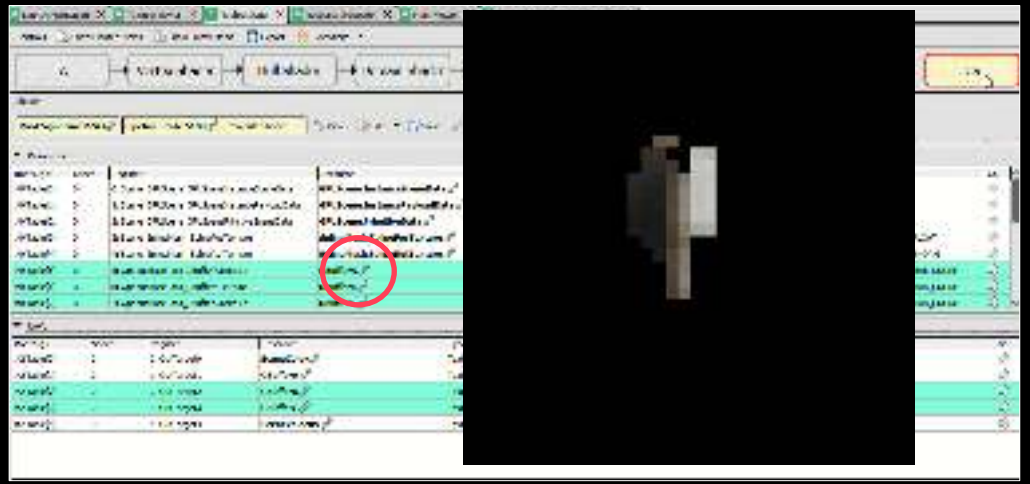
이 머티리얼이 어떤 픽셀에 그려질까?

- 셰이딩할 대상을 분류해서 모으자
- 분류? 동일 셰이딩이 필요한 픽셀 좌표 모으기

셰이딩 빈 Shading Binning

- 래스터 빈과 유사한 컨셉
- Shading Bin ID에 따라 VisBuffer에서 픽셀 분류
- 그 후 GBuffer를 업데이트하는 패스에 전달

- 3단계
- Count ➔ Reserve ➔ Scatter




UNREAL
ENGINE
START
INTERACT:

컴퓨트 머티리얼

셰이딩 패스
- 셰이딩 빈 당 스레드 하나
- 각 빈에 indirect dispatch

자동 그래디언트 불가
- ie. ddx/ddy/Sample/…
- 분류시 2x2 픽셀 쿼드를 저장하도록 확장
- DX12 SM6.6의 Compute Shader Derivatives에 의존

잘못된 밉 샘플링으로 깜빡임 발생


Menu
Fly
Sprint
Dismiss Controls
10% Helper Lanes


Menu
Fly
Sprint
Dismiss Controls
4% Helper Lanes

컴퓨트 머티리얼

비효율성 개선

- 다수의 Helper Lane 제거
- 7% 성능 향상
- 4.92 ms ➔ 4.62 ms

PS Total	4.92ms
Classify	0.04ms
Shading	4.88ms

➔

CS Total	4.62ms
Binning	0.18ms
Shading	4.44ms

가변 레이트 셰이딩

가변 레이트 셰이딩 Variable Rate Shading
- 최신 하드웨어 지원 피처
- 셰이딩 레이트 이미지 생성하여 하드웨어에 입력

셰이딩 레이트 이미지 생성
- 전 프레임의 컨트라스트와 비교
- 낮은 컨트라스트 영역에서는 낮은 셰이딩 레이트
- 셰이딩을 줄여도 인지가 안되도록

⇒ 컨트라스트 적응형 셰이딩 Contrast adaptive shading

< 출저: Moving Gears to Tier 2 Variable Rate Shading >

가변 레이트 셰이딩

하드웨어 셰이딩 레이트 방식
- 최소 VRS 타일 크기 제한이 존재
- AMD 8x8, NV 16x16
- 같은 타일은 같은 셰이딩 레이트로 동작

우리는 컴퓨트 셰이딩
- 필요에 따라 쿼드와 픽셀 구분
- 쿼드 모드에서 2x2 타일 VRS 테스트

⇒ 45% 셰이딩 감소

2x2 SW VRS 타일 사용 / 45% 셰이딩 감소

최적화를 위한 노력

SW VRS 지원을 위한 노력
- 셰이딩 비닝 변경
- 셰이더 패스 변경

비어있는 빈에 의한 GPU 오버헤드
- CPU는 어떤 머티리얼이 뷰에 있는지 모른다
- 모든 셰이딩 빈에 Dispatch 실행
- 커맨드 버퍼를 압축해서 부하를 줄이자!

최적화를 위한 노력

결과

- 인지적으로 문제 X
- 약 40% 성능 향상
- 4.92 ms ➔ 3.05 ms

PS Total	4.92ms
Classify	0.04ms
Shading	4.88ms

CS Total	4.62ms
Binning	0.18ms
Shading	4.44ms

CS Total	3.93ms
Binning	0.18ms
Shading	3.75ms

2x2 VRS: N
Compact: Y

CS Total	3.05ms
Binning	0.38ms
Shading	2.67ms

2x2 VRS: Y
Compact: Y

로드맵

개발 예정이거나 진행 중

나나이트 미지원
(5.4 기준)

Skinning
제한적 WPO
모프 타겟
인스턴스 버텍스 페인팅
...

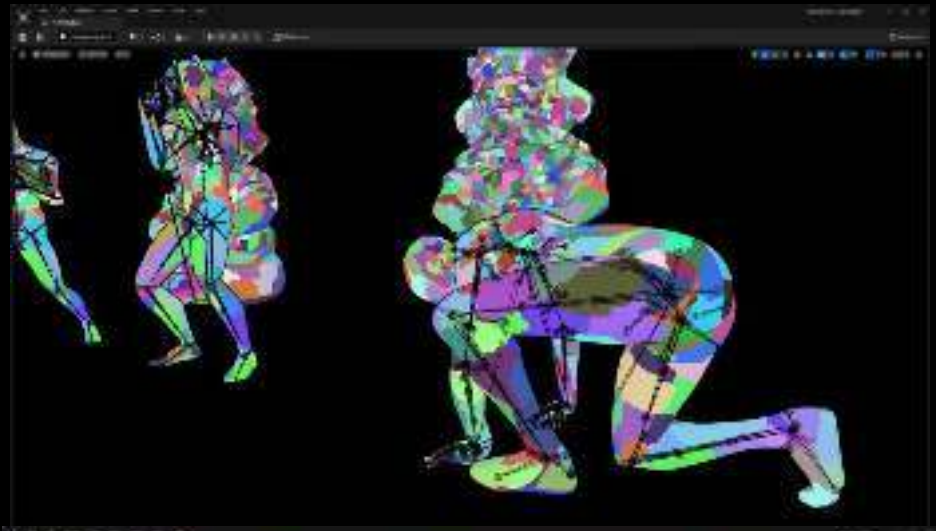
나나이트 스키닝

[개발 중] 스켈레탈 애니메이션

- Main 브랜치에 포함 / 디폴트 활성화
- 클로스 X, 모프 X

- 5.5 실험기능 (예상)
- 프로덕션에 활용하지 마세요

※ 출시 시기와 개발 방향은 예고없이 변경될 수 있습니다.



레거시 머티리얼 패스 제거

새로운 컴퓨트 머티리얼 방식

- 더 효율적이고 더 유지보수가 편리하다
- 미래에 있을 GPU주도 루멘씬과 통합 용이
- 레거시와의 기능 패리티 충족

- UE5.5에 적용 예정

※ 출시 시기와 개발 방향은 예고없이 변경될 수 있습니다.

팁: 나나이트 디버그 툴

디버그를 위한 시각화 기능 제공

- r.Nanite.Visualize Picking
- 마우스 픽킹된 타겟을 찾아 디버그 정보 출력

- r.Nanite.Picking.Domain 0-3
- Triangle ➔ Cluster ➔Instance ➔ Primitive


감사합니다.
에픽게임즈 코리아