UE4 모바일 렌더링 개요

에픽게임즈 코리아 박창현

Agenda

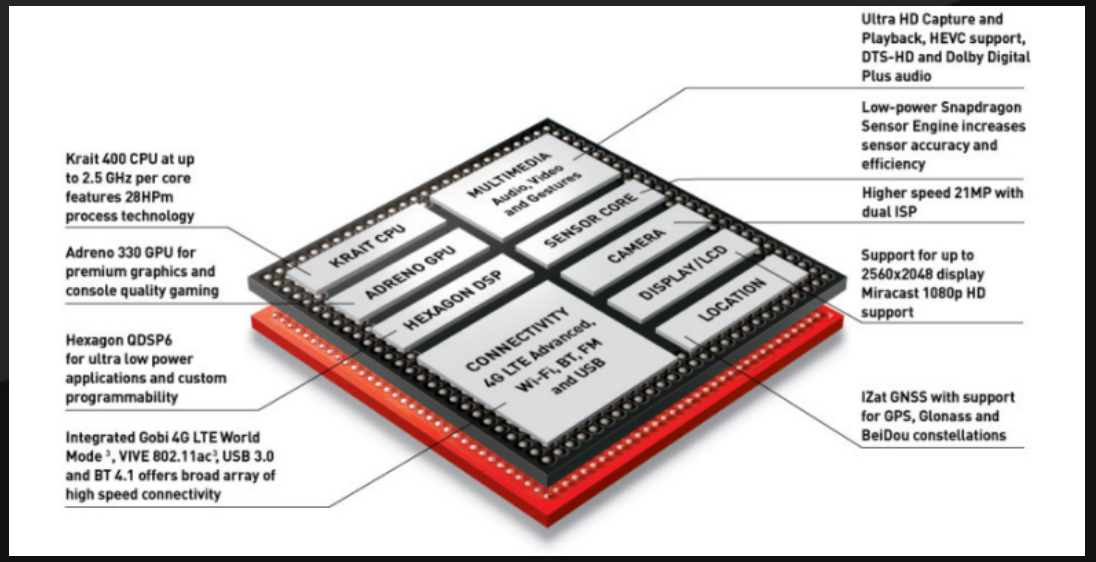
- 모바일 GPU
  - 모바일 디바이스
  - Tile-Based GPU / Early Z Test
  - 단편화
- UE4 모바일 씬 렌더러 (최신 4.19.2 기준)
  - Feature Level
  - 렌더러 분석
  - 라이팅과 그림자
- UE4 모바일 렌더링 관련 Tips

모바일 GPU

모바일 디바이스 / Tile-Based GPU / Early Z Test / 단편화

모바일 AP

- Mobile Application Processor
  - System-on-Chip
- 여러 GPU 제조사 & 여러 아키텍쳐
  - Arm - Mali [Utgard / Midgard / Bifrost]
  - ImgTec – Power VR [SGX / Rogue]
  - Apple – A11 Bionic
  - Qualcomm – Adreno
  - Etc
- 이후 내용들은 널리 통용되는 기종들 위주로



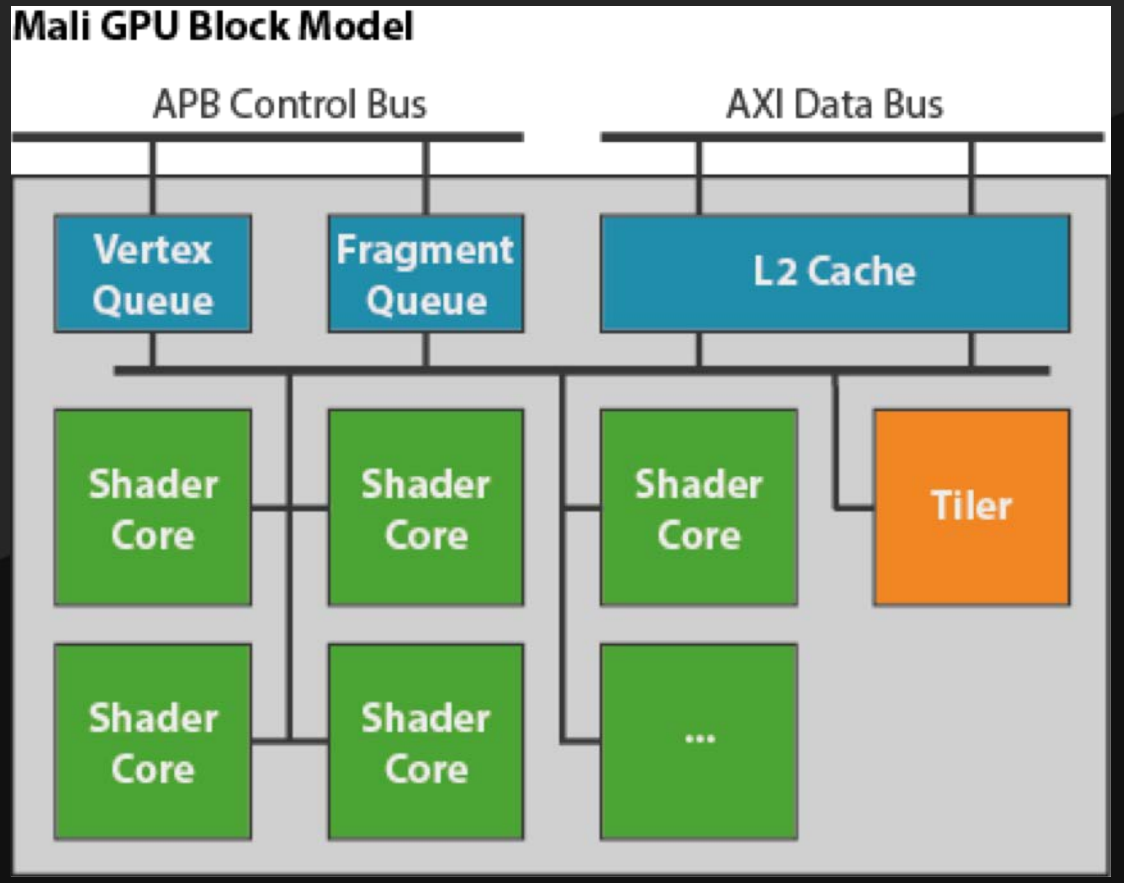
< 이미지 출저 : https://www.anandtech.com/show/7846/the-difference-between-snapdragon-800-and-801-clearing-up-confusion >

모바일 AP

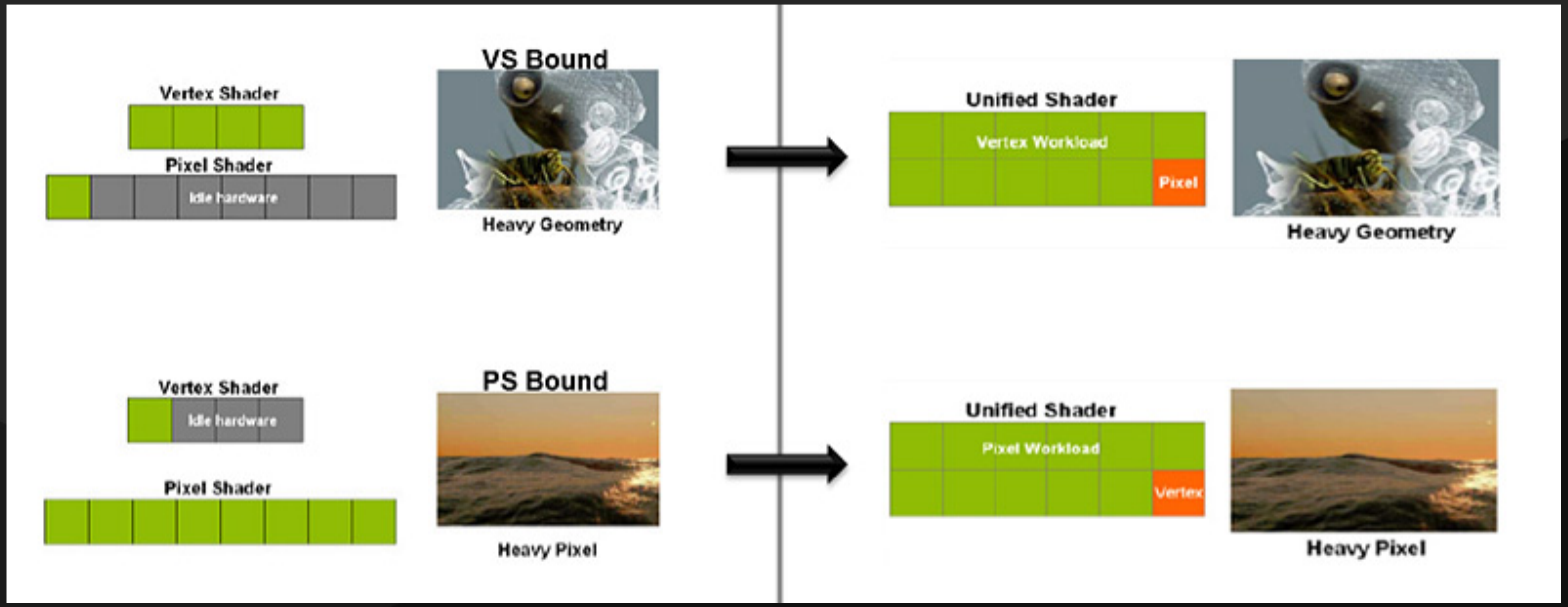
- 갤럭시 시리즈 : Exynos
  - Adreno / Mali GPU
- iPhone 시리즈 (6S, SE, 7) : A9, A10 Fusion
  - Power VR GPU
- iPhone 시리즈 (8, X) : A11 Bionic
  - 자체 개발 GPU (Metal 2 지원)

모바일 GPU

- Unified Shader Architecture
  - 모든 Shader Processing Unit이 어떤 타입의 셰이딩 연산이든 처리
  - VS, GS, TS, PS, CS, …



< 이미지 출저 : https://community.arm.com/graphics/b/blog/posts/the-mali-gpu-an-abstract-machine-part-3---the-midgard-shader-core >



< 이미지 출저 : http://www.embedded-computing.com/embedded-computing-design/understand-the-mobile-graphics-processing-unit

모바일 GPU

- Unified Memory Architecture
  - CPU와 같은 Physical Memory
  - CPU와 다른 Logical Memory

  - 과거에는
    - GPU만 접근 가능한 Window를 지정
    - GPU <-> CPU 메모리 접근 : 복사 발생

  ⇒ CPU MMU와 GPU의 MMU가 같은 주소 공간을 사용

모바일 하드웨어

- Battery
  - 같은 성능이라도 저전력이라면 더 선호
  - 메인 메모리도 DDR이 아닌 LPDDR을 사용
  - 메모리 대역폭 사용이 많으면, 배터리 소모도 크다

- Device Temperature
  - 발열이 심하면 자동으로 성능 Throttling
  - CPU / GPU의 성능을 크게 낮춰 기기가 쉴 수 있게 한다

모바일 성능

- 다른 플랫폼 대비 낮은
  - Memory Bandwidth
  - Fillrate (Pixel/Texel)
  - GFLOPS

- 성능을 올리기 위해
  - Tile-Based GPU
  - Early Z Testing

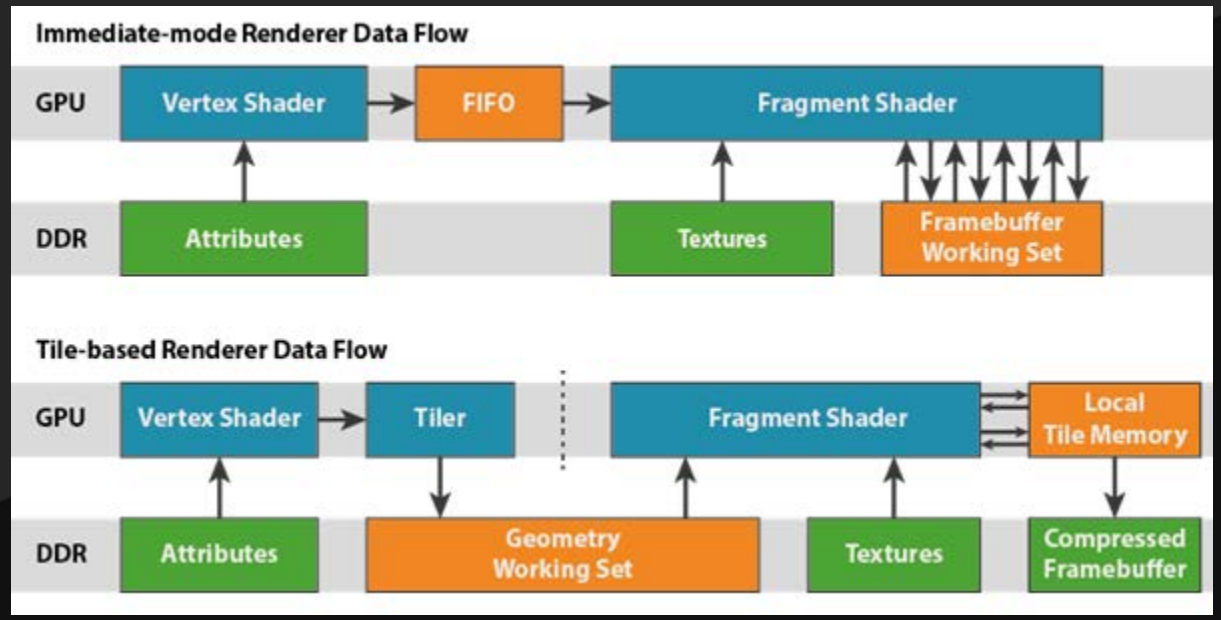
모바일 GPU – Tile-Based

- 일반적인 GPU
  - "한 장면"을 그리기 위해 필요한 리소스를 GPU 메모리에 올려놓는다
  - 리소스: SceneColor / DepthStencil / ...
- Tile-based GPU
  - 모든 렌더링을 타일 단위로 진행
  - Ex. 1024 x 768 ⇒ (32 x 32) 768 Tiles
    ※ Tile 크기는 제조사마다 다름

< 이미지 출저: https://www.imgtec.com/powervr/graphics/architecture/ >

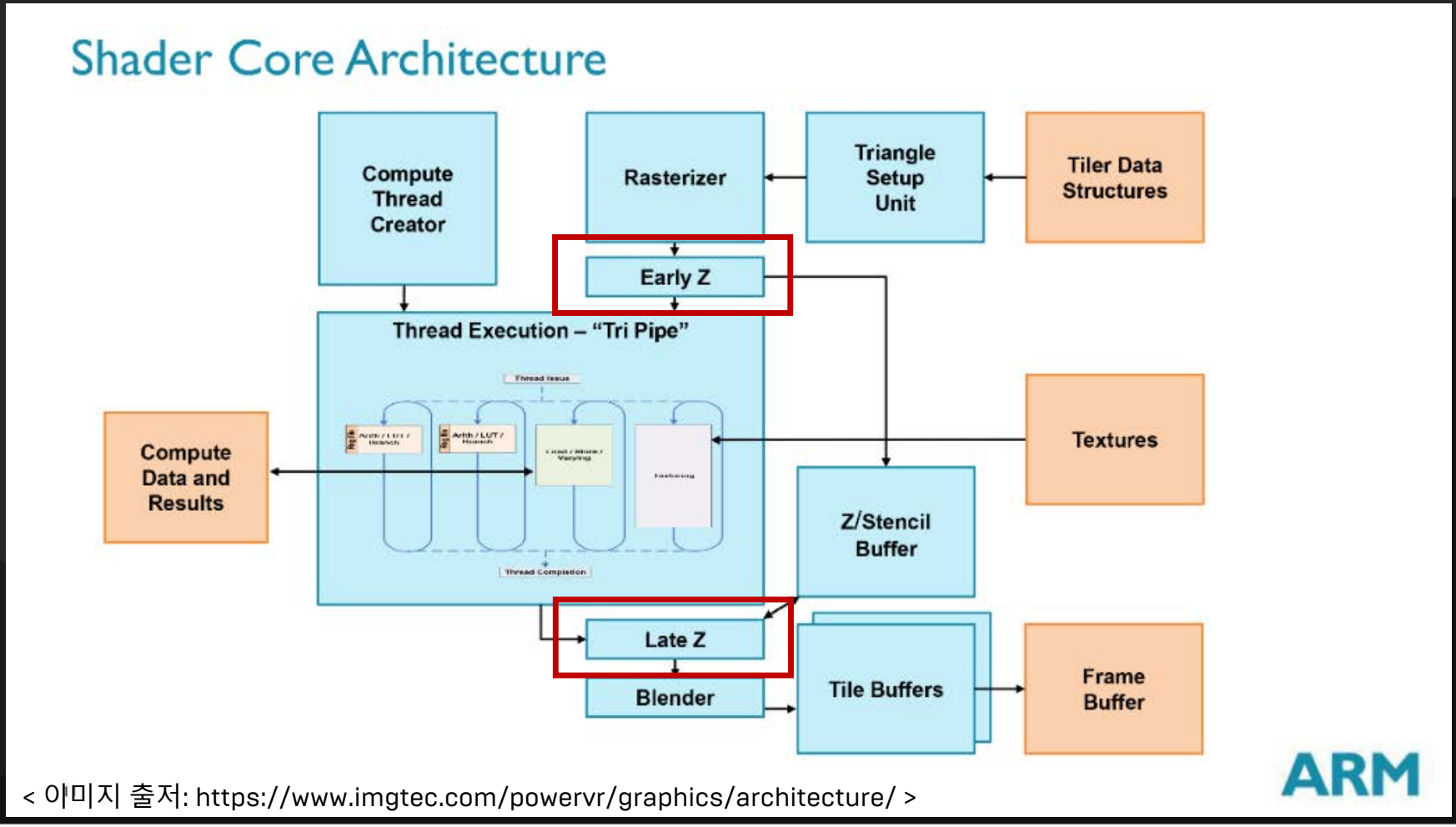
모바일 GPU – Tile-Based

- Tile-based GPU
  - “한 타일”을 그리기 위해 필요한 리소스만을 “in-chip 메모리”에 올려놓는다
  - In-chip 메모리와 ShaderCore간의 데이터 전송은 아주 저렴
- 단점
  - 한 장면을 여러 타일로 나누고 관리하기 위한 추가적인 관리 비용



< 이미지 출저: https://www.anandtech.com/show/8234/arms-mali-midgard-architecture-explored/4 >

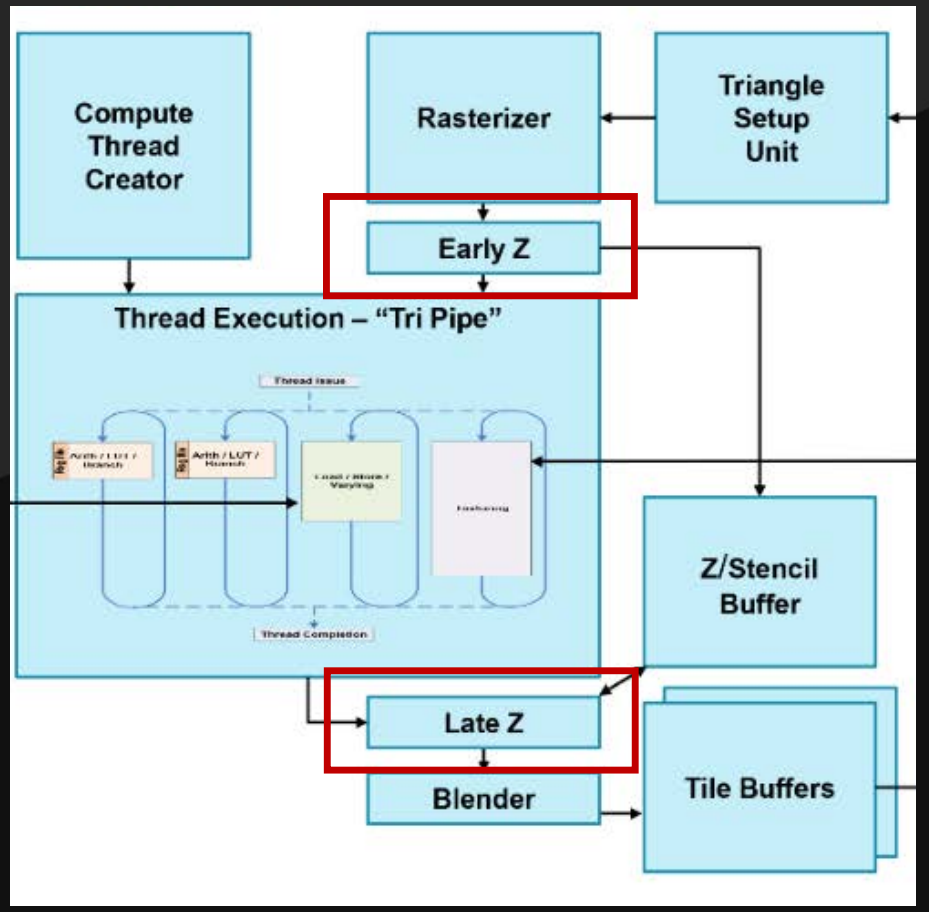
모바일 GPU – Early Z Test



< 이미지 출저: https://www.imgtec.com/powervr/graphics/architecture/ >

모바일 GPU – Early Z Test

- Z Test = Depth Test
- 일반적인 Depth Test
  - PS 후 렌더링 결과물을 타겟에 블렌딩할지 결정
  - Test 실패시 Drop
- Early Z Test ⇔ 일반적인 Depth Test
  - Pixel Shading 전에 미리 Drop여부 판단
  - Vendor/Architecture마다 다른 접근법을 추가
    - Mali GPU – Forward Pixel Kill
    - PowerVR GPU – Hidden Surface Removal



< 이미지 출저: https://www.imgtec.com/powervr/graphics/architecture/ >

모바일 GPU – 단편화

- Ex. 같은 ES3.1인데, 어느 기종은 되고 다른 기종은 안되고
- Ex. 같은 제조사 같은 모델이지만, 어느 디바이스는 되고 다른 디바이스는 안되고

- GPU 제조사
- GPU Architecture
- 모바일 AP / 디바이스 제조사

- 그래픽스 드라이버 스택
  - Ex. GPU는 ES3.1을 지원하지만, 드라이버가 지원 X
  - Ex. 빌드버전 A에서는 안되었지만, B에서는 동작

모바일 GPU – 단편화

- Metal
  - 단편화가 거의 없음.
  - 하나의 제조사가 모든 것을 관리
- OpenGL ES
  - glGetString(...)
  - GL_VENDOR / GL_RENDERER / GL_VERSION / GL_SHADING_LANGUAGE_VERSION / GL_EXTENSIONS
- UE4
  - FPlatformOpenGLDevice::Init() / FOpenGLBase::ProcessExtensions(...)
    - UE4 렌더링 그래픽스 API 결정 : Ex. OpenGL ES2.0과 ES3.1
    - UE4 디바이스별 Feature 결정 : Ex. GL_OES_texture_half_float
  - ES2 fallback

UE4 모바일 씬 렌더러

Feature Level / 렌더러 분석 / 라이팅과 그림자

플랫폼별 지원 그래픽스 렌더링 API

- Desktop
  - Windows = D3D / OpenGL / Vulkan
  - macOS = Metal / OpenGL / Vulkan
  - Linux = OpenGL / Vulkan
- Mobile
  - Android = OpenGL ES / Vulkan
  - iOS = Metal / OpenGL ES 3.0 / Vulkan
- HTML = WebGL

UE4 지원 그래픽스 렌더링 API

- Desktop
  - Windows = D3D / OpenGL / Vulkan(Experimental)
  - macOS = Metal / OpenGL / ~~Vulkan~~
  - Linux = OpenGL / Vulkan(Experimental)
- Mobile
  - Android = OpenGL ES / Vulkan(Experimental)
  - iOS = Metal ~~/ OpenGL ES 3.0 / Vulkan~~
- HTML = WebGL

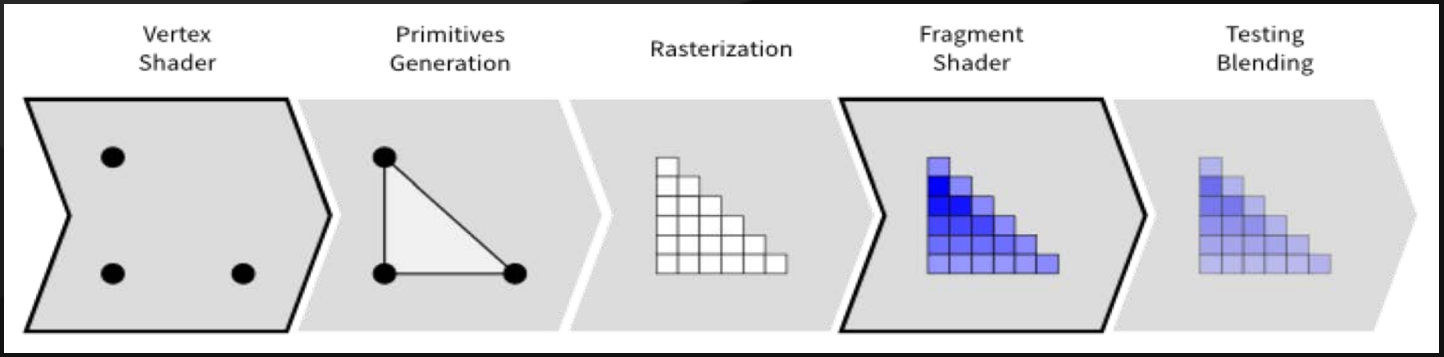
UE4 지원 그래픽스 렌더링 API

- 언리얼 엔진에서 관리법
  - 플랫폼 및 버전을 떠나 **지원되는 기능 단위 구분**

- Feature Level
  - ES2 : OpenGL ES2.0
  - ES3_1 : OpenGL ES3.1 & Metal & Vulkan
  - SM4 : DX10 & OpenGL 3
  - SM5 : DX11+ & OpenGL 4+

UE4 렌더러

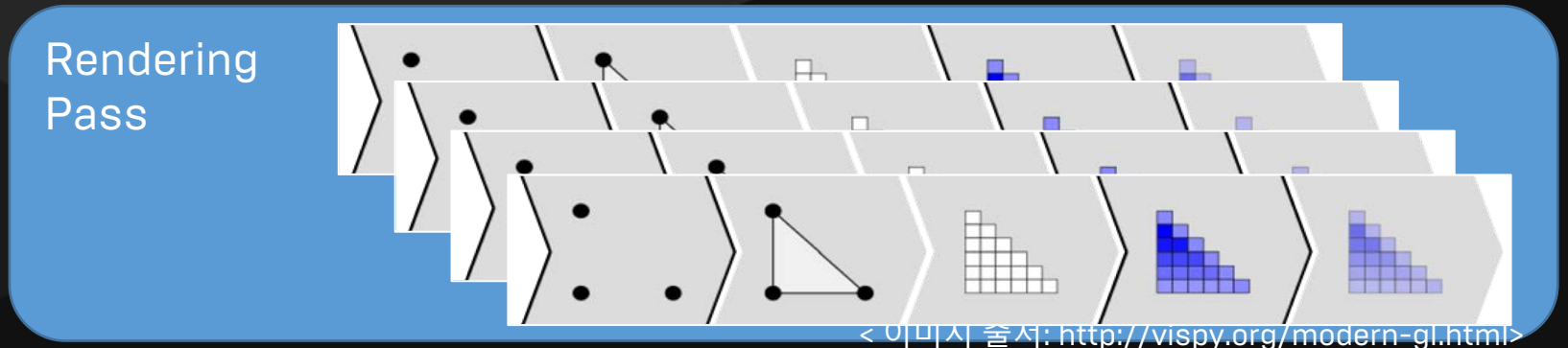
- 렌더러?
  - 렌더링 파이프라인
  - 하나의 Geometry를 그리는 과정
    - RenderState 설정
    - Draw Call로 시작
    - 최종 결과물 RenderTarget은 텍스쳐 혹은 프레임 버퍼



< 이미지 출저: https://open.gl/drawing >

UE4 렌더러

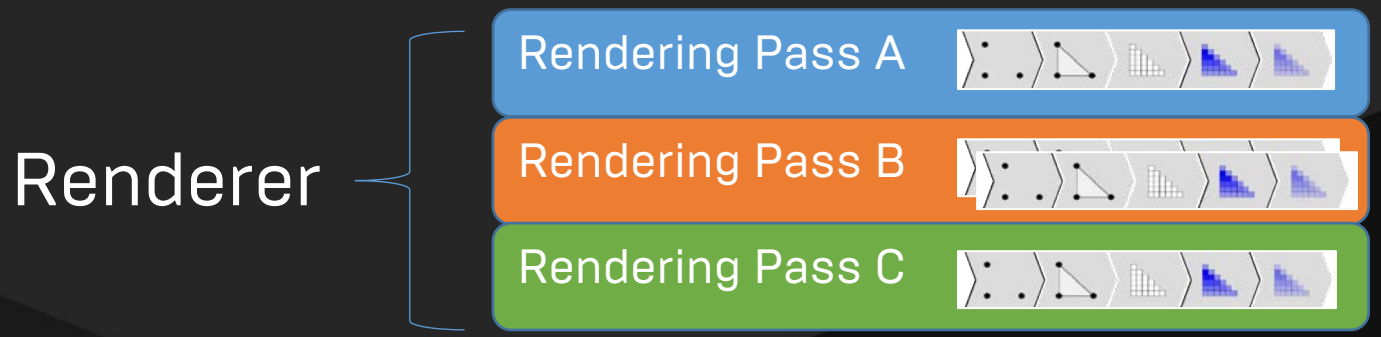
- 렌더러?
  - 렌더링 패스
  - 여러 지오메트리를 엔진 레벨에서 한데 묶어 그리는 것.
    - Ex. BasePass Pass / Light Pass / Translucency Pass
  - 즉, 하나의 패스에서 여러 DrawCall을 요청.

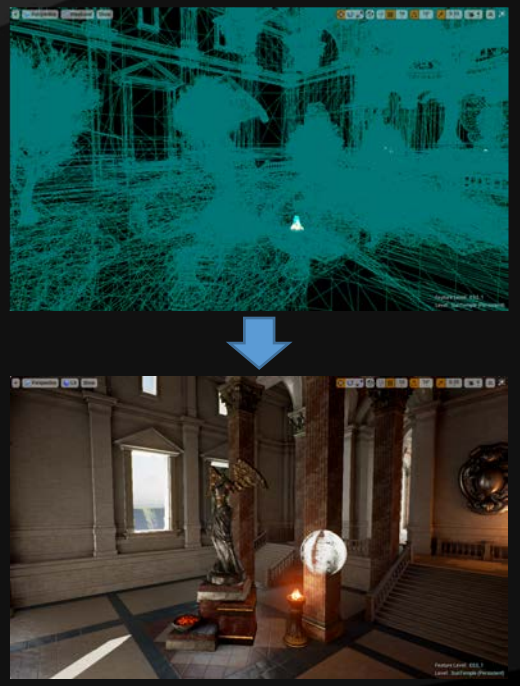


< 이미지 출처: http://vispy.org/modern-gl.html>

UE4 렌더러

- 렌더러?
  - **여러 패스**와 연산을 통해, 데이터로 이루어진 씬을 **그려내는 역할**





  - 각 Geometry을 그리는 방식에 따라 크게 두가지로 구분
    - Deferred / Forward Shading

UE4 렌더러

Deferred Shading

- For each mesh
  - Render albedo/normal/etc into G-buffer
- Render each light
  - Read infos from G-buffer
    - Accumulate lighting

Forward Shading

- For each mesh
  - For each light
    - Accumulate lighting

UE4 렌더러

Deferred Shading

- Gbuffer 사용을 기반으로 합성
- 지연된 패스에서 Shading 계산
- 다이나믹 라이팅 렌더링에 좋음
- Surface attribute 변경 불편
- 기능 비활성화에 유연

Forward Shading

- 하나의 패스에서 Shading 계산
- Translucent Surface 렌더링에 좋음
- 동적라이트 추가시 성능 부하가 크다
- Surface Attribute에 유연
- 하드웨어 MSAA를 사용할 수 있음

UE4 렌더러

Deferred Shading in UE4

- PC / Console용 (SM4이상 지원)
- Deferred Scene Renderer

Forward Shading in UE4

- Mobile용 (ES3_1 이하)
- Mobile Scene Renderer

UE4 렌더러

- Mobile Deferred Shading?
  - G-buffer를 위해 Multiple Render Target 기능이 필요
    ⇒ OpenGL ES 3.0 / Metal Specification
  - 현재 지원하지 않고 있다.
    - G-buffer사용시 많은 대역폭 요구 / 복잡한 연산 필요
    - 현 세대의 디바이스에서 게임을 플레이가능할 정도의 성능이 나오지 않음
    - 언젠가 디바이스가 더 발전을 한다면...?

- 대안 : Desktop-Class Forward Renderer (iOS only)
  - 모바일에서 높은 퀄리티 요구에 대한 대안
  - 메탈 세이더를 데스크탑 포워드 렌더러 기준으로 컴파일

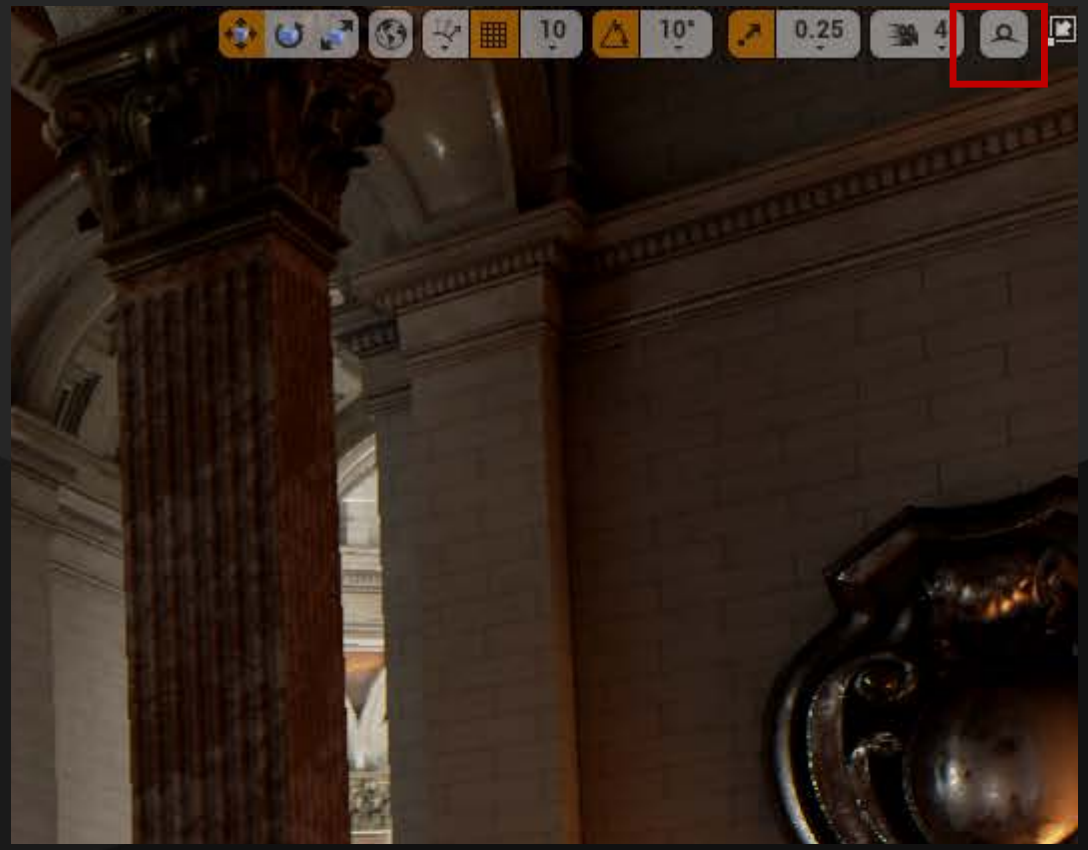
Mobile Scene Renderer

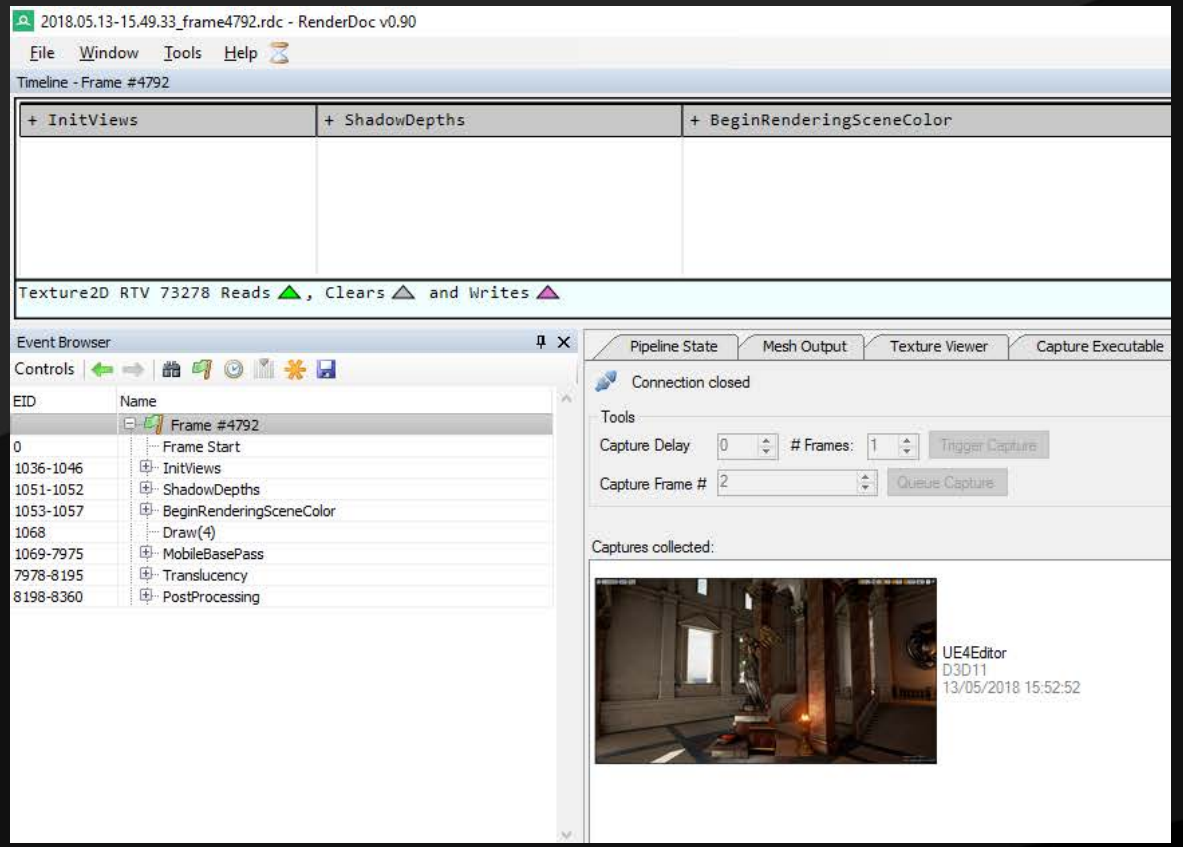
- 동작 방식 분석
  - 프로젝트 Sun Temple
    = Mobile Features 쇼케이스
  - RenderDoc 그래픽스 디버거
    - 독립 프로그램
    - UE4 플러그인 통합 제공
    - 패스별 결과 분석에 용이

※ 다운로드: https://developer.nvidia.com/ue4-sun-temple

Mobile Scene Renderer

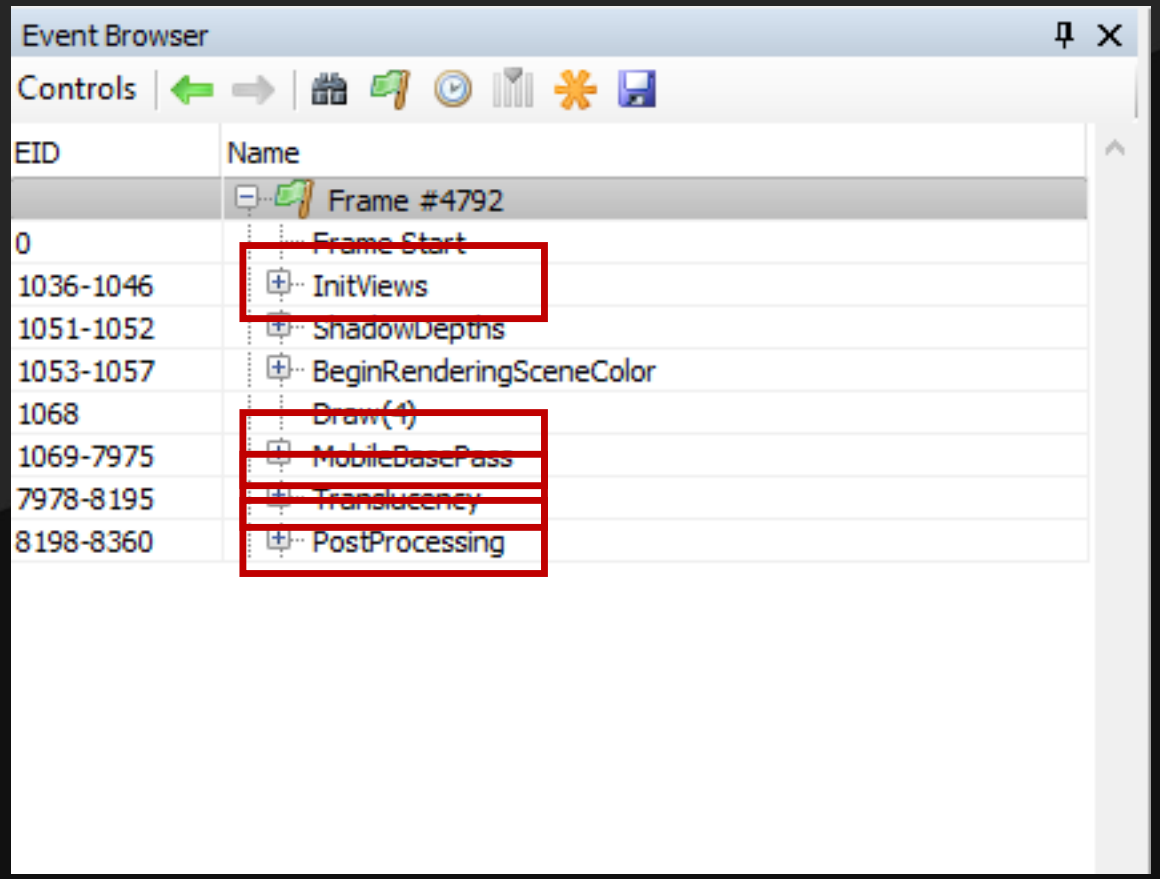
- 사용법





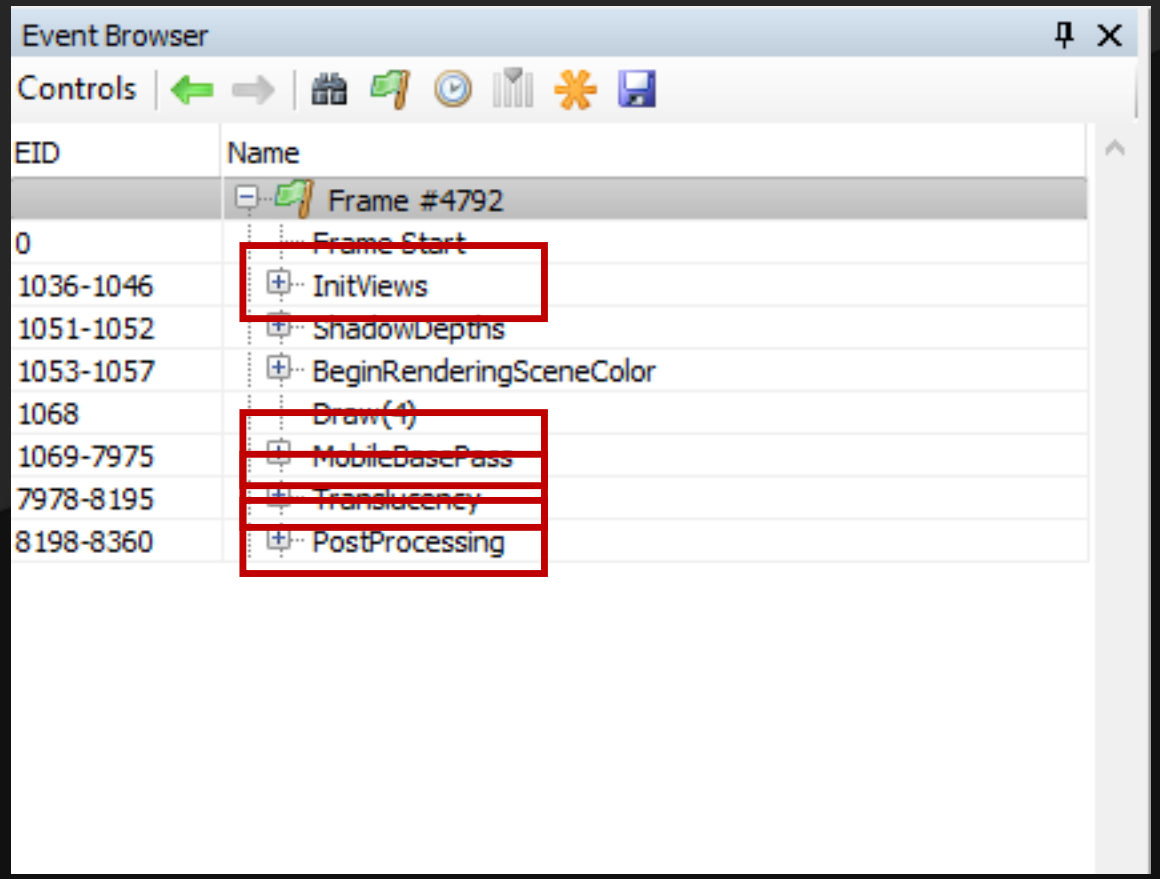
Mobile Scene Renderer

- 위 ⇒ 아래
  - 중요함수 / 렌더패스 순차 나열
  - 패스내 DrawCall 각각 확인 가능
- 중요한 부분만 간추리면
  - InitViews
  - MobileBasePass
  - Translucency
  - PostProcessing



Mobile Scene Renderer

- InitViews
  - 그려질 Geometry를 선별
- Mobile Base Pass
  - 불투명한 Geometry를 그린다
- Translucency Pass
  - 반투명한 Geometry를 그린다
- PostProcessing
  - 후처리를 한다



Mobile Scene Renderer

1. 그려질 Geometry를 선별

- FSceneRenderer::initView(...)
- 불필요한 Geometry를 그리지 않는 것은 성능을 아끼기에 좋은 방법
- 가시성 Visibility 확인 ⇒ 보이는 것만 목록에 추가
  - Triangle이 아닌 객체 단위

UE4 모바일 씬 렌더러

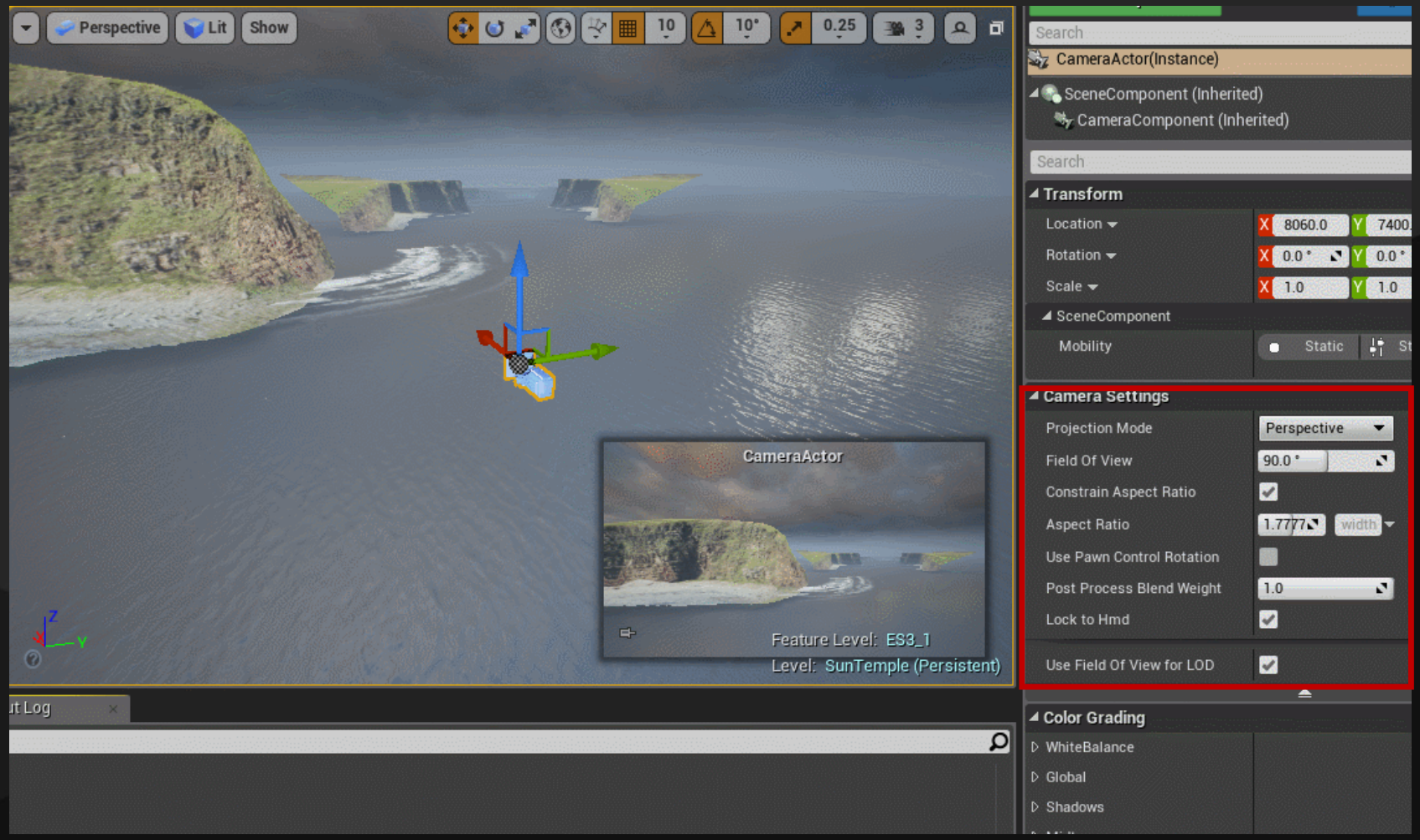
1. 그려질 Geometry를 선별

- 가시성 Visibility 확인 in Mobile
  - Distance Culling
  - Frustum Culling
  - Precomputed Visibility

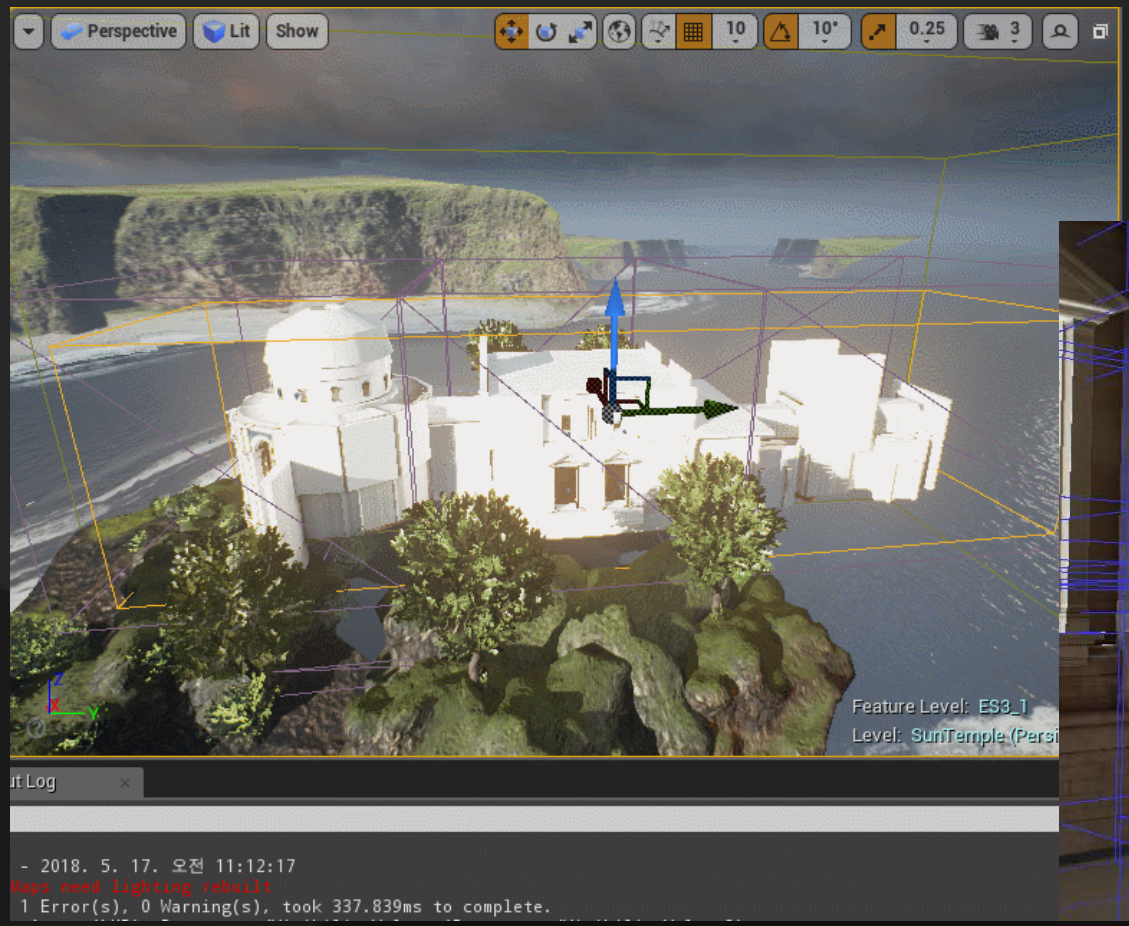
Distance Culling

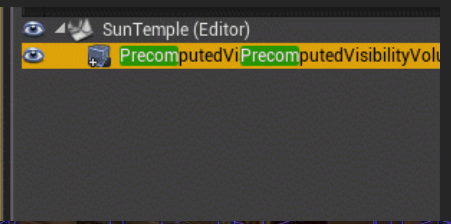
LOD	
Forced Lod Model	0
Min LOD	0
Override Min LOD	☐
Min Draw Distance	0.0
Desired Max Draw Distance	500.0
Current Max Draw Distance	500.0
Allow Cull Distance Volume	☑

Frustum Culling



Precomputed Visibility





Mobile Scene Renderer

1. 그려질 Geometry를 선별

- 가시성 Visibility 확인 in Mobile
  - Distance Culling
  - Frustum Culling
  - Precomputed Visibility
  - Occlusion Culling (4.20 지원 예정)
    - Software Occlusion culling
    - Hareware Occlusion queries
    - 이어지는 “<포트나이트>모바일: 오클루전 컬링 by Dmitriy Dyomin” 세션을 참고하세요.

※ 렌더링 파이프라인 레벨 컬링 : Clipping, Back-face culling

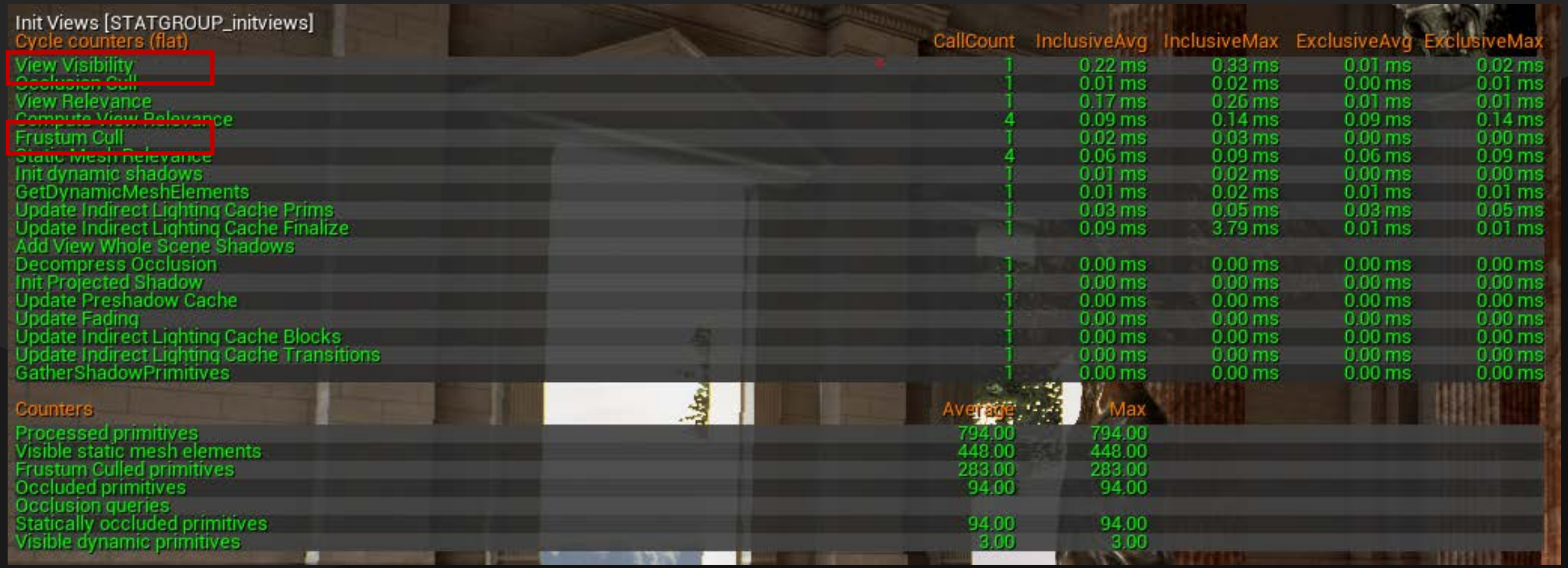
Mobile Scene Renderer

1. 그려질 Geometry를 선별

- 관련 콘솔 커맨드
  - stat initviews

  - FreezeRendering 후
    - Fly나 Ghost 활용

stat initviews



Init Views [STATGROUP_initviews]

Cycle counters (flat)	CallCount	InclusiveAvg	InclusiveMax	ExclusiveAvg	ExclusiveMax
View Visibility	1	0.22 ms	0.33 ms	0.01 ms	0.02 ms
Occlusion Cull	1	0.01 ms	0.02 ms	0.00 ms	0.01 ms
View Relevance	1	0.17 ms	0.26 ms	0.01 ms	0.01 ms
Compute View Relevance	4	0.09 ms	0.14 ms	0.09 ms	0.14 ms
Frustum Cull	1	0.02 ms	0.03 ms	0.00 ms	0.00 ms
Static Mesh Relevance	4	0.06 ms	0.09 ms	0.06 ms	0.09 ms
Init dynamic shadows	1	0.01 ms	0.02 ms	0.00 ms	0.00 ms
GetDynamicMeshElements	1	0.01 ms	0.02 ms	0.01 ms	0.01 ms
Update Indirect Lighting Cache Prims	1	0.03 ms	0.05 ms	0.03 ms	0.05 ms
Update Indirect Lighting Cache Finalize	1	0.09 ms	3.79 ms	0.01 ms	0.01 ms
Add View Whole Scene Shadows					
Decompress Occlusion	1	0.00 ms	0.00 ms	0.00 ms	0.00 ms
Init Projected Shadow	1	0.00 ms	0.00 ms	0.00 ms	0.00 ms
Update Preshadow Cache	1	0.00 ms	0.00 ms	0.00 ms	0.00 ms
Update Fading	1	0.00 ms	0.00 ms	0.00 ms	0.00 ms
Update Indirect Lighting Cache Blocks	1	0.00 ms	0.00 ms	0.00 ms	0.00 ms
Update Indirect Lighting Cache Transitions	1	0.00 ms	0.00 ms	0.00 ms	0.00 ms
GatherShadowPrimitives	1	0.00 ms	0.00 ms	0.00 ms	0.00 ms

Counters	Average	Max
Processed primitives	794.00	794.00
Visible static mesh elements	448.00	448.00
Frustum Culled primitives	283.00	283.00
Occluded primitives	94.00	94.00
Occlusion queries		
Statically occluded primitives	94.00	94.00
Visible dynamic primitives	3.00	3.00

FreezeRendering 후 Fly / Ghost

Mobile Scene Renderer

2. 불투명한 Geometry를 그린다

- FMobileSceneRenderer::RenderMobileBasePass(...)

- 각 Geometry에 대해 “라이팅 Lighting & 그림자 Shadows”
- 제일 중요하고 제일 복잡한 패스

- “라이팅 & 그림자”는 두가지 접근이 존재
  - Dynamic / Static

Mobile Scene Renderer

2. 불투명한 Geometry를 그린다

- Dynamic Lighting & Shadows
  - 라이팅
    - 라이트는 타입(포인트, 스포트)에 따라 다른 모양의 바운드를 갖는다
    - 바운드 내에 존재하는 지오메트리는 라이팅 받는다고 판단
    - 지오메트리 셰이딩 연산시 영향을 주는 라이트에 추가
  - 그림자
    - 주로 Shadow Map 사용
    - 픽셀에서 주어진 라이트가 보이는지 보이지 않는지 판단
    - 라이트에서 씬을 바라보는 깊이 정보가 필요

Mobile Scene Renderer

2. 불투명한 Geometry를 그린다

- Dynamic Lighting & Shadows
  - 장점
    - 라이트의 색, 위치, 추가/제거를 마음대로 할 수 있다.
    - 지오메트리를 그리기 위해 추가적인 준비를 할 필요가 없다.
  - 단점
    - 그림자는 성능에 큰 영향을 준다
    - 라이트를 추가하는 것도 성능에 큰 영향을 준다

Mobile Scene Renderer

2. 불투명한 Geometry를 그린다

- Dynamic Lighting & Shadows in UE4 Mobile
  - 라이팅
    - Movable 디렉셔널 라이트 지원 (기본값: False)
    - 포인트 라이트 제한된 Movable 지원 (최대 4개)
    - 스포트 라이트 Movable 미지원
  - 그림자
    - Cascaded Shadow Map만 지원

Mobile Scene Renderer

2. 불투명한 Geometry를 그린다

- Static Lighting & Shadows
  - 라이팅 결과가 변하지 않는다면, 미리 계산해 놓고 필요할때 가져다 쓸 수 있다.
  - Real-time에 성능 세이브 + Offline에 시간을 충분히 쓸 수 있어 퀄리티도 더 좋다

  - 즉, Dynamic Lighting & Shadows보다 적은 비용으로 높은 퀄리티를 얻을 수 있다.

Mobile Scene Renderer

2. 불투명한 Geometry를 그린다

- Static Lighting & Shadows
  - 장점
    - 실시간에 엄청 빠르다
    - 복잡한 라이팅 시나리오를 계산할 수 있다.
    - 실제같은 그림자 표현이 가능하다.
  - 단점
    - 추가적인 메모리를 차지한다
    - 라이팅 빌드시 시간이 많이 든다
    - 무언가 변경이 있을 때마다 라이트 재빌드를 해야한다

UE4 Precomputed / Precalculated

- 라이트매스
  - Directional LightMap
  - ShadowMap
  - Precomputed Radiance Transfer
    - Indirect Lighting Cache
    - Volumetric Lightmap
- 리플렉션 캡쳐
  - Image Based Lighting
- 메시 디스턴스 필드
  - DF Shadows
  - DF Indirect Shadows
  - DF Ambient Occlusion

UE4 Precomputed / Precalculated

- 라이트매스
  - Directional LightMap
  - ShadowMap
  - Precomputed Radiance Transfer
    - Indirect Lighting Cache
    - ~~Volumetric Lightmap~~ (SM4)
- 리플렉션 캡쳐
  - Image Based Lighting
- 메시 디스턴스 필드
  - DF Shadows
  - ~~DF Indirect Shadows~~ (SM5)
  - ~~DF Ambient Occlusion~~ (SM5)

- 모바일에서는 위와 같은, 미리 계산된 라이팅/정보 지원

Mobile Scene Renderer

2. 불투명한 Geometry를 그린다

- Static Lighting & Shadows in UE4 Mobile
  - 라이팅
    - 모든 타입(디렉셔널, 포인트, 스포트)의 Static Mobility 라이트 지원
    - Stationary Directional Light 지원
    - Indirect 라이팅 지원
  - 그림자
    - 구워진 라이트맵 혹은 셰도우 맵 사용
    - 스태틱 메시에 대해 Distance Field Shadow 사용 가능

Mobile Scene Renderer

2. 불투명한 Geometry를 그린다

- 최고의 방법은, Dynamic과 Static을 잘 섞어 쓰는 것
  - Static 방식
    - 원거리의 약한 라이팅을 표현할 때 사용하여 성능을 아낌
    - 실시간 간접광을 얻어올 수 있는 유일한 방법
  - Dynamic 방식
    - 씬이 플레이어의 행동에 반응하는 것처럼 만들기 위한 방법
    - 정적인 라이팅/그림자에 동적인 형태를 덧입혀 현실감을 추가
  - Stationary Directional Light = Directional Lightmap + CSM

UE4 모바일 씬 렌더러

2. 불투명한 Geometry를 그린다

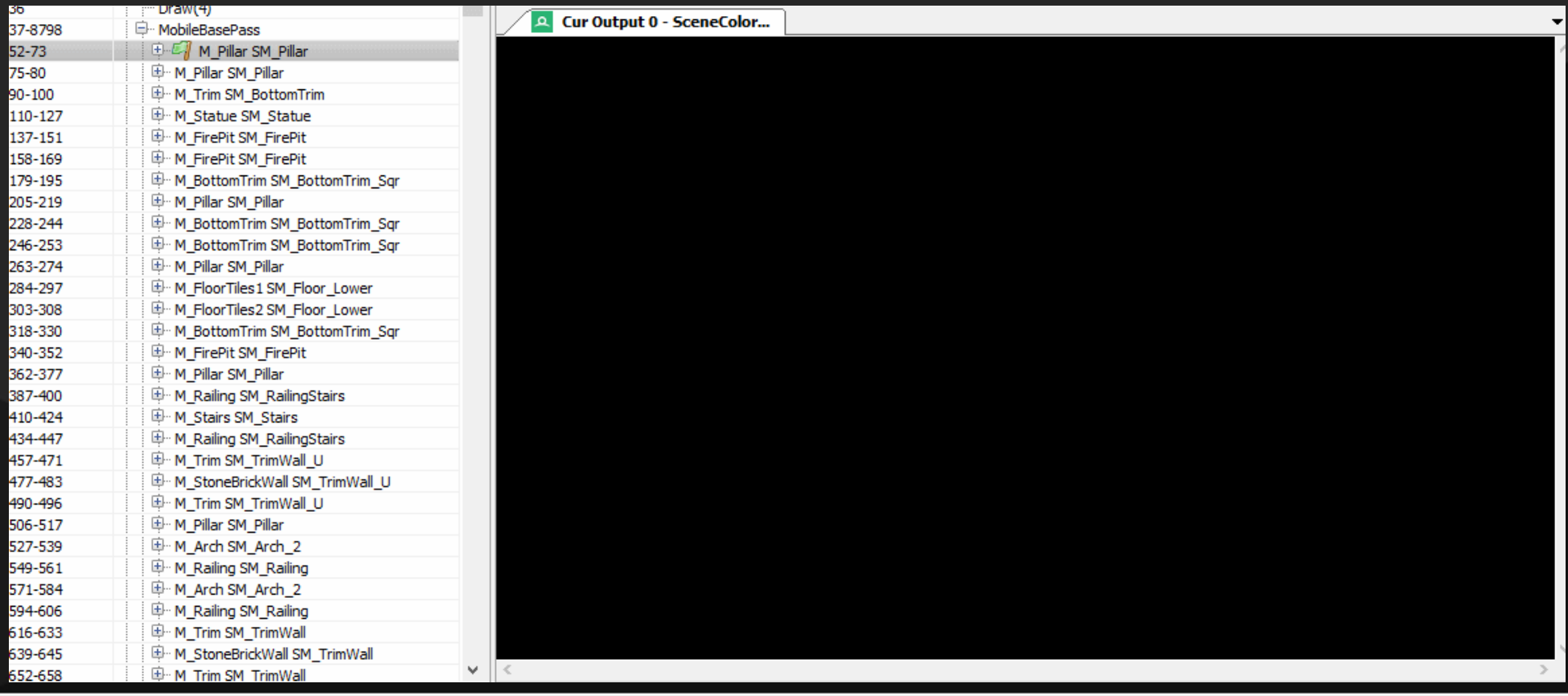
- 정리! 모바일 지원 라이팅 기능
  - Directional Lightmaps
  - Indirect Light Cache
  - Image based specular lighting

  - Linear HDR Lighting
  - Distance field shadows + analytical specular on the sun
  - Baked Shadow 혹은 Cascaded Shadow Map
  - Dynamic Mesh는 라이팅 받음.

UE4 모바일 씬 렌더러

2. 불투명한 Geometry를 그린다

※Color Range 변경



UE4 모바일 씬 렌더러

3. 반투명한 Geometry를 그린다

- FMobileSceneRenderer::RenderTranslucency(...)
- 반투명한 물체들은 뒤에 비춰보일 수 있는 배경이 필요
- Geometry를 그리는 것은 크게 다르지 않으나
- 덮어쓰기가 아닌 Alpha Blending 진행

UE4 모바일 씬 렌더러

3. 반투명한 Geometry를 그린다

8801-9137	Translucency
8802-8805	BeginRenderingSceneColor
8816-8825	M_WaveFoam SM_Waves
8827-8832	M_WaveFoam SM_Waves
8838-8843	M_WaveFoam SM_Waves2
8856-8862	M_VolumetricLightSphere Sphere
8873-8880	M_Fire SM_FireFX
8895-8909	M_Godray SM_GodRay
8913-8919	M_Godray SM_GodRay
8923-8929	M_Godray SM_GodRay
8933-8939	M_Godray SM_GodRay
8953-8958	M_VolumetricLightSphere Sphere
8969-8976	M_Fire SM_FireFX
8988-8994	M_VolumetricLightSphere Sphere
9004-9011	M_Fire SM_FireFX
9023-9029	M_VolumetricLightSphere Sphere
9039-9046	M_Fire SM_FireFX
9058-9064	M_VolumetricLightSphere Sphere
9074-9081	M_Fire SM_FireFX
9093-9099	M_VolumetricLightSphere Sphere
9109-9116	M_Fire SM_FireFX
9127-9137	M_WaveFoam Sphere

UE4 모바일 씬 렌더러

4. 후처리를 한다

- FPostProcessing::ProcessES2(...)

- 후처리 : 렌더링 과정 제일 마지막에 적용되는 비주얼 이펙트
- Off-screen 렌더타겟들 (SceneColor/SceneDepth/etc) 사용

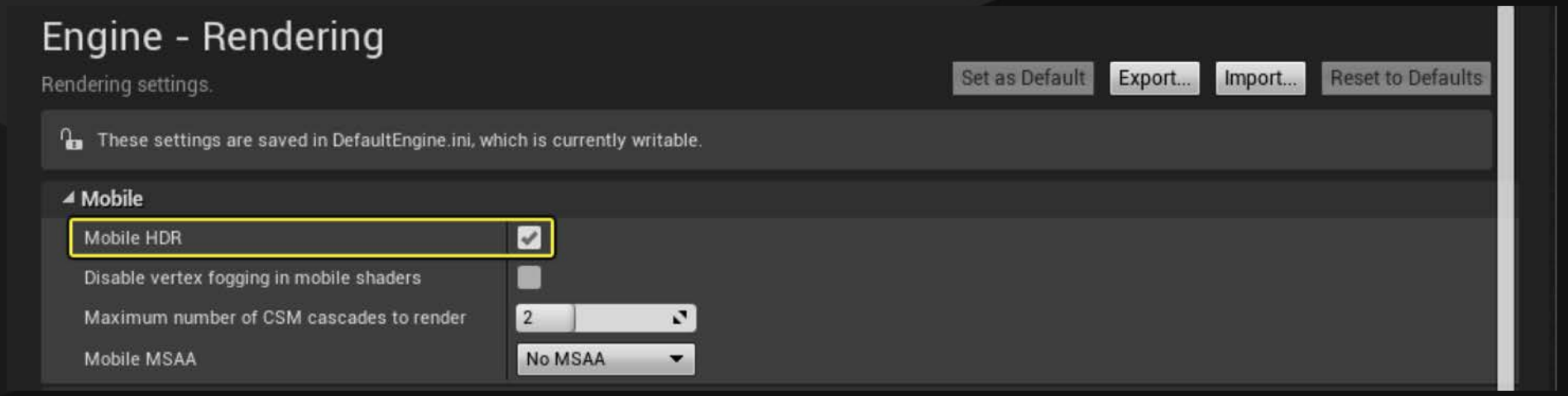
- 픽셀셰이더 바운드

- 모바일과 다른 플랫폼의 기능/퀄리티 차이가 가장 많이 나는 부분

UE4 모바일 씬 렌더러

4. 후처리를 한다

- UE4의 모바일 포스트 프로세싱
  - 퀄리티를 조금 희생하더라도, 성능을 확보하도록 구현
  - 필수 : “Mobile HDR” True



UE4 포스트 프로세싱

In Deferred Renderer

- ColorGrading
  - WhiteBalance
  - Global
  - Shadows
  - Midtones
  - Highlights
  - Misc
- Film
- MobileTonemapper

- Lens
  - Chromatic Aberration
  - Bloom
  - Dirt Mask
  - Camera
  - Exposure
  - Lens Flares
  - Image Effects
  - Depth of Field

- Rendering Features
  - Post Process Material
  - Ambient Cubemap
  - Ambient Occlusion
  - Global Illunmination
  - Motion Blur
  - Light Propagation Volume
  - Screen Space Reflection
  - Misc

UE4 모바일 포스트 프로세싱

r.MobileHDR 1 & r.Mobile.TonemapperFilm 1

- ColorGrading
  - WhiteBalance
  - Global
  - Shadows
  - Midtones
  - Highlights
  - Misc
- Film
- MobileTonemapper

- Lens
  - Chromatic Aberration
  - Bloom (Limited)
  - Dirt Mask
  - Camera (Limited)
  - Exposure (Limitied)
  - Lens Flares
  - Image Effects
  - Depth of Field (Limited)

- Rendering Features
  - Post Process Material
  - Ambient Cubemap
  - Ambient Occlusion
  - Global Illunmination
  - Motion Blur
  - Light Propagation Volume
  - Screen Space Reflection
  - Misc

UE4 모바일 포스트 프로세싱

r.MobileHDR 1 & r.Mobile.TonemapperFilm 0

- ColorGrading
  - WhiteBalance
  - Global
  - Shadows
  - Midtones
  - Highlights
  - Misc
- Film
- MobileTonemapper

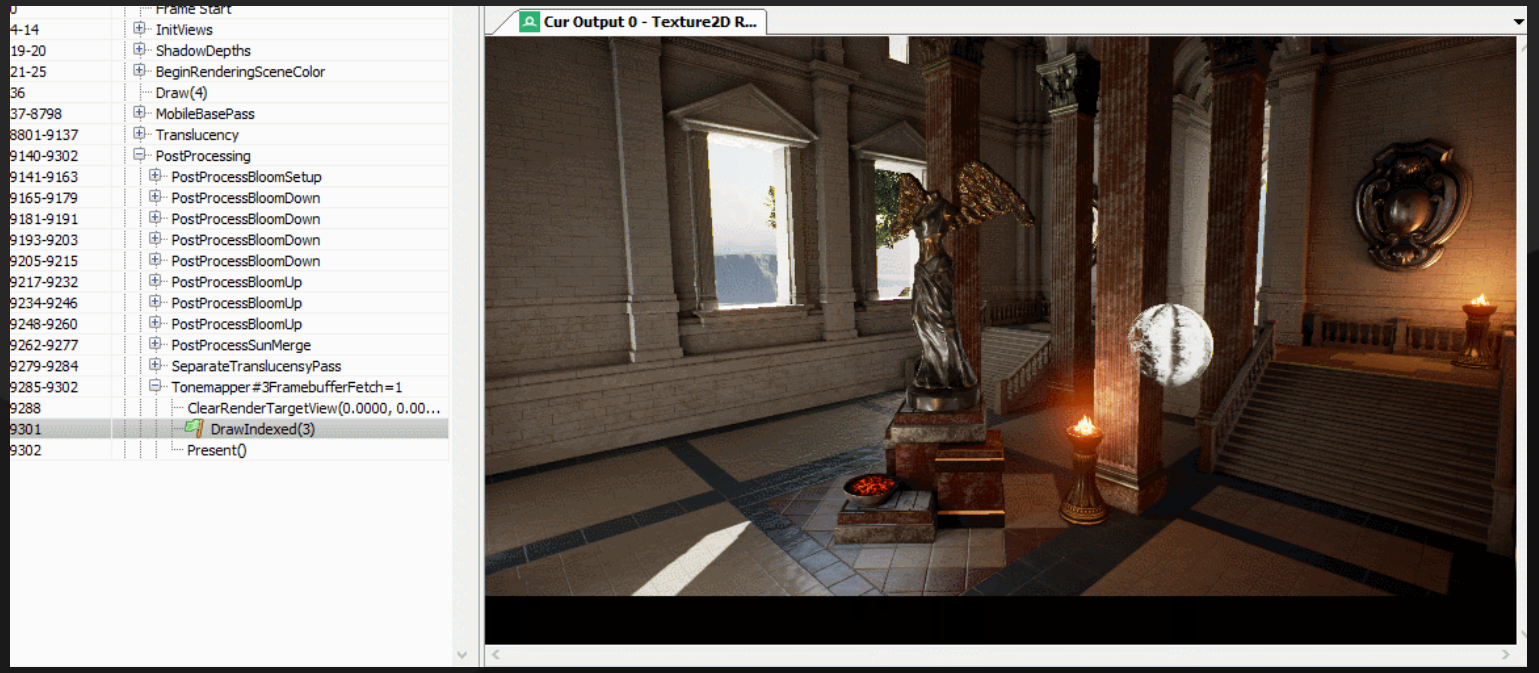
- Lens
  - Chromatic Aberration
  - Bloom (Limited)
  - Dirt Mask
  - Camera (Limited)
  - Exposure (Limitied)
  - Lens Flares
  - Image Effects (Limited)
  - Depth of Field (Limited)

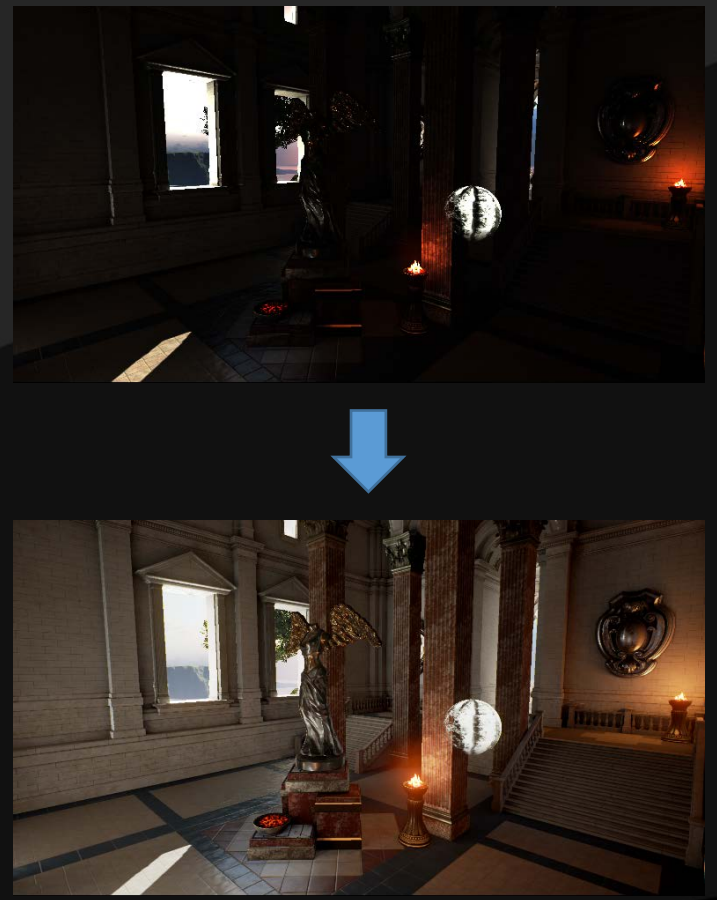
- Rendering Features
  - Post Process Material
  - Ambient Cubemap
  - Ambient Occlusion
  - Global Illunmination
  - Motion Blur
  - Light Propagation Volume
  - Screen Space Reflection
  - Misc

참조 : https://docs.unrealengine.com/en-us/Platforms/Mobile/PostProcessEffects

UE4 모바일 씬 렌더러

4. 후처리를 한다





Tips

UE4 모바일 프리뷰 / Tips / 모바일 최적화

UE4 모바일 프리뷰

- 모바일 플랫폼 개발을 돕기위한 편의 기능
  - 모바일 Feature 시뮬레이션

  - 모바일 그래픽스 렌더링 API : OpenGL ES / Metal / Vulkan
  - PC는 Metal / OpenGL ES 지원하지 않음

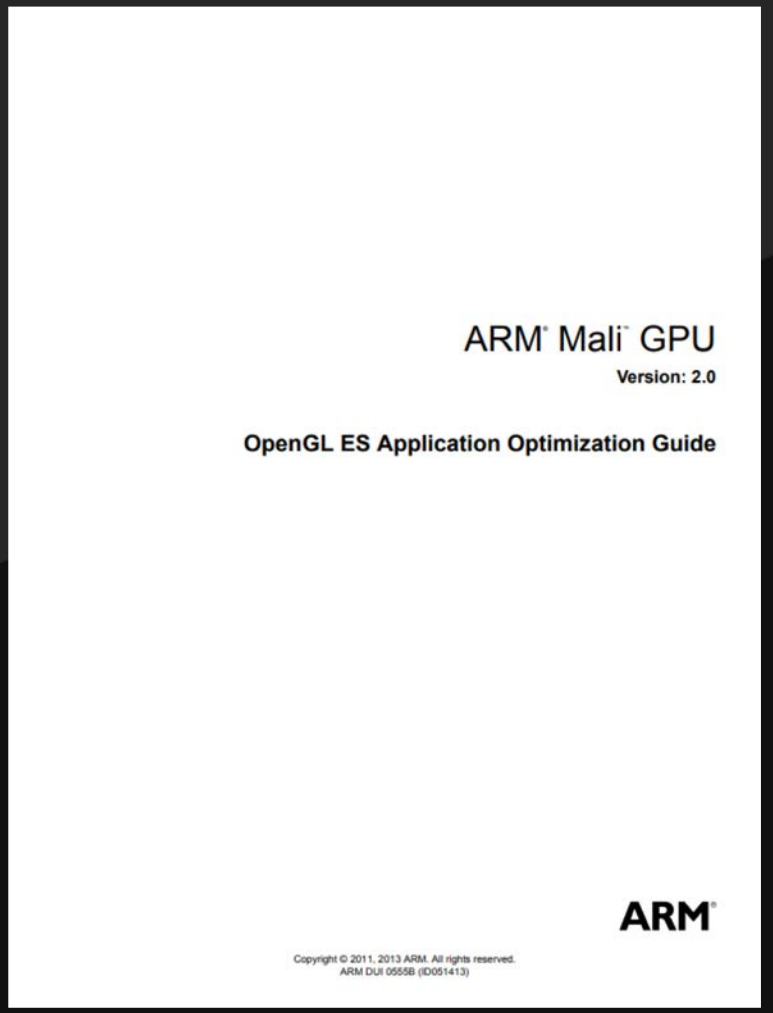
- 어떻게?
  - D3D11 / OpenGL의 일부 Feature만 사용 ⇒ 시뮬레이션
  - 실제 디바이스에서와 동일한 셰이딩 퀄리티를 보장하지 않음.

모바일 렌더링 Tips

- 모바일 GPU에서 Read back은 엄청 느리다
  - GPU메모리 위의 FrambeBuffer에 대한 “ReadPixel”발생시
  - 모든 타일이 완료된 후에 원하는 Pixel 정보를 돌려준다
    - 즉, 그래픽스 파이프라인이 멈추게 되어 성능이 크게 떨어진다
- 드라이버 레벨 최적화
  - Ex. Opaque가 한번도 그려지지 않은 타일에 대해 깊이버퍼 초기화 생략
    - 깊이 정보를 가져다 쓰는 연출에서 문제를 일으키기도..
- 성능 쓰로틀링
  - Silver Bullet은 없다.
  - CPU / GPU를 자주 쉬게 해주어야 한다.

모바일 최적화

- Application Processor 최적화
- API Level 최적화
- Vertex Processing 최적화
- Fragment Processing 최적화
- Bandwidth 최적화
- Misc. 최적화



https://developer.arm.com/docs/dui0555/b/miscellaneous-optimizations

UE4 모바일 최적화

- Draw call 최소화
  - 드로우 콜 발생시 어떤 지오메트리를 그리냐에 무관하게
  - 항상 “메모리 할당, 데이터 복사 등” 프로세스 진행
  - 최대한 메시가 배칭되도록 컨텐츠를 구성하세요

- RenderState 변경 최소화
  - 그래픽스 파이프라인은 상태 머신 / 상태 변경 하나하나가 오버헤드
  - 이전 상태가 남아서 다음 지오메트리 그릴때 영향을 주는 형태
  - UE4는 패스별 지오메트리 특성별로 그룹화 시킴.. 렌더러 수정시 유의

- 성능을 올리려면 그래픽스 파이프라인이 쉬지 않도록

UE4 모바일 최적화

- Vertex Processing
  - LOD를 적극적으로 사용하세요
  - 세세한 지오메트리는 노멀맵으로 대체
  - 객체 단위 컬링을 대충하고, Triangle Clipping에 기댄다면 그래픽스 파이프라인에게 큰 부담

- Pixel Processing
  - Overdraw를 최대한 줄이세요.
  - 성능 버짓에 따라 가용할만한 복잡도의 Pixel Shader 사용
  - 전체 해상도를 줄여 연산할 픽셀 수 조절 : r.ScreenPercentage {%}

UE4 모바일 최적화

- 메모리 대역폭
  - 데스크탑 GPU 대비 많이 부족한 스펙 : 모바일 플랫폼의 주된 병목
  - 대역폭은 공유 자원
    - CPU 등에서 크고 빈번한 데이터 접근시, GPU가 사용할 대역폭이 부족
    - GPU에서 너무 많이 메모리 대역폭 사용시, CPU가 제 성능을 못 냅니다
  - 메모리 대역폭 절약 = 배터리 소모 최소화
  - 캐시의 활용은 성능 확보 및 배터리 절약에도 도움이 됩니다.
  - 텍스쳐 밉맵 / 텍스쳐 압축 / 가능한 낮은 Bit depth / LOD

감사합니다.

QnA