UE4 안드로이드

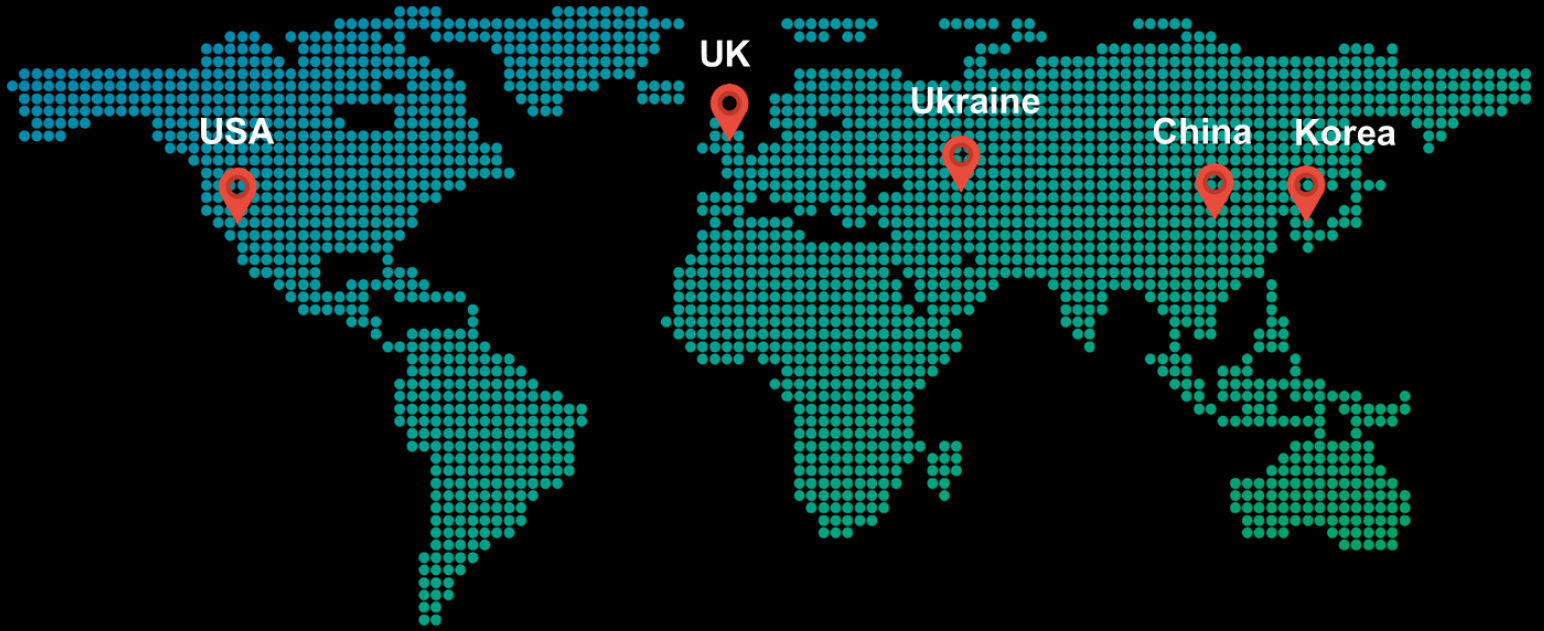
Vulkan 최적화 백서

UE4 안드로이드 불칸 최적화 백서

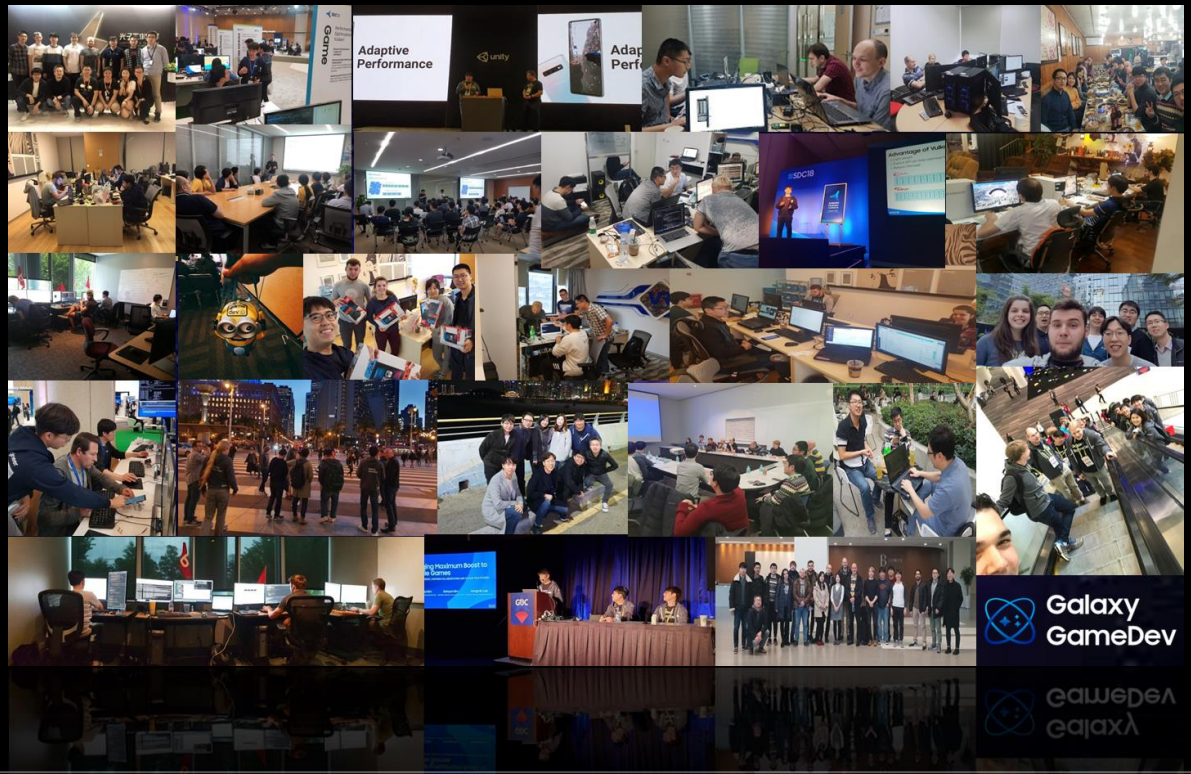
목차

- Galaxy GameDev 소개
- GPU Watch 소개
- 게임부하 바로 알기 : 전류, 발열, 성능의 상관 관계
- GPU/CPU 바운드란?
- Vulkan API
- Vulkan 최적화 사례 소개
- 위험하지만, 그래도 시도해볼만한 Vulkan 최적화 방법들

Galaxy GameDev 소개



Galaxy GameDev 소개



Partners


ACTION SQUARE
DIGITAL LEGENDS
4:33 CREATIVE LAB
netmarble
SummerTime Studio
ATYPICAL GAMES
EPIC GAMES
GAMELOFT
MOAI GAMES
NEXON
腾讯游戏 Tencent Games
Microsoft
FERAL MAKE YOUR PLAY
GAMEVIL
NAT GAMES
PEARL ABYSS
PUBG CORPORATION A BLUEHOLE COMPANY
COATSINK.
FTG.
HOUND 13
NIANTIC
THE SECRET POLICE
unity
CROTEAM
DEEP SILVER
FISH LABS
JAGEX
NCSOFT
SQUARE ENIX.
SUPER EVIL MEGACORP

GameDev Programs

- Game Engine 최적화
  - 최적화된 Game Engine 제공
- Galaxy Device 대여
  - 게임 개발 및 QA 를 위해 제공
- Workload 분석자료 제공
  - 성능과 품질의 Balancing 을 위해 제공
- 분석 툴 사용방법 공유
  - 게임 개발 환경 개선을 위해 제공
- 원격/현장 기술지원
  - 긴밀한 협업을 위해 제공

Galaxy GameDev 소개

- 2016 Epic Games, **ProtoStar** Galaxy S7 Collaboration
- 2016 NetGames, **HIT**
- 2016 Super Evil MegaCorp, **VainGlory**
- 2017 433, **HeroDC**
- 2017 Netmarble, **Lineage 2: Revolution**
- 2017 Nexon, **AxE**
- 2017 XL Games, **Archeage: Begins**
- 2017 Action Square, **Blade II**
- 2017 Hound13, **HundredSoul**
- 2017 Square Enix, **FinalFantasy XV Pocket Edition**
- 2017 Croteam, **Talos Principle**
- 2017 Roblox Corporation, **Roblox**
- 2017 Doragon Entertainment, **Danmaku Unlimited 3**
- 2017 GameLoft, **Asphalt 8**
- 2017 Cornfox&Bros, **Oceanhorn: Monster of Uncharted Seas**
- 2017 Deep Silver, **Galaxy On Fire 3: Manticore!**
- 2017 Digital Legends Entertainment, **Afterpulse**
- 2017 First Touch Games, **Score! Hero / Dream League Soccer**
- 2018 Tencent, **Honor of Kings**
- 2018 Pearl Abyss, **BlackDesert Mobile**
- 2018 Epic Games, **Fortnite Battle Royale**
- 2018 Amazon Lumberyard, **Bistro**
- 2019 Moai Games, **TRAHA**
- 2019 PUBG corporation / Tencent, **PUBG MOBILE**
- 2019 Tencent, **QQ Speed**

GPU Watch

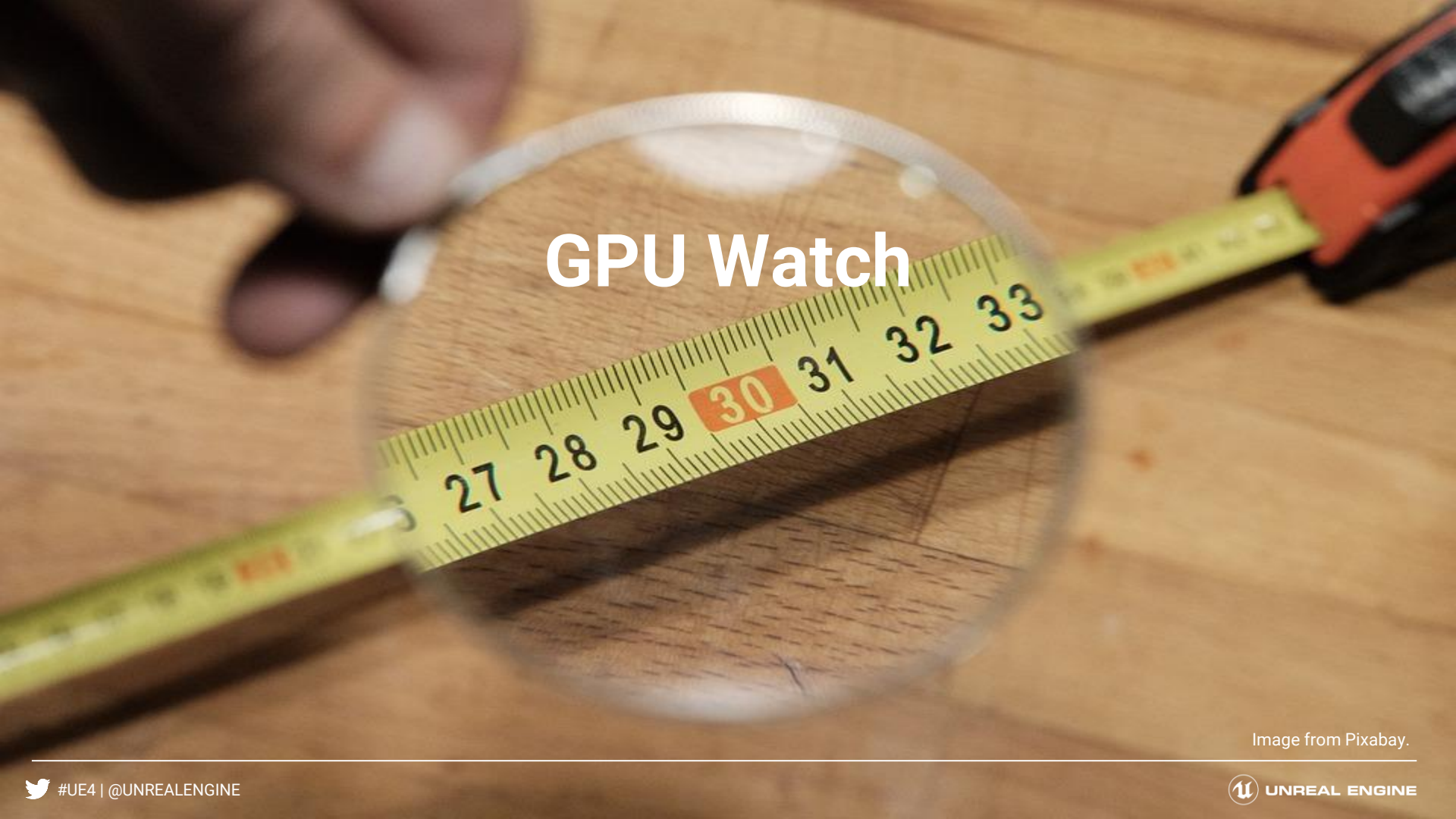


Image from Pixabay.

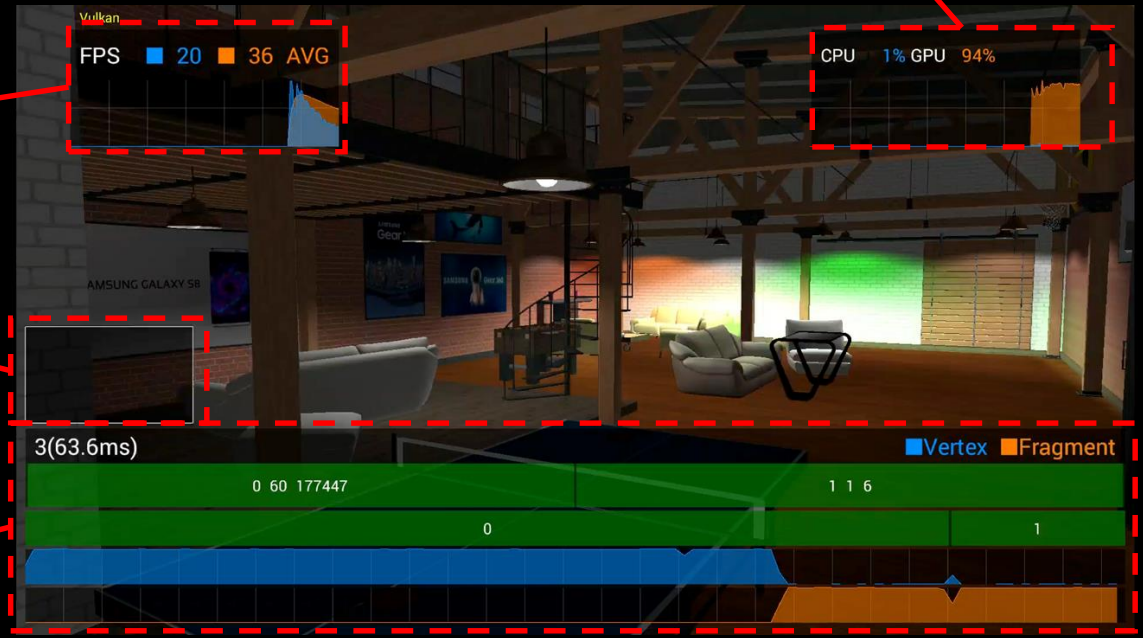
GPU Watch

CPU 사용량 (normalized)
GPU 사용량

현재 FPS /
평균 FPS

Profiling 정보의
Screenshot

GPU profiling 정보
- Renderpass
- GPU activity



GPU Watch

현재 **FPS**

평균 FPS
(최소 100 frames)

Frame drop

GPU Watch

CPU 사용량
(normalized)

GPU 사용량
(not normalized)

GPU Watch

Profiling 정보의 Screenshot

Frame 수

Frame Rendering 시간

GPU Watch

Render pass 번호

Draw call 수

Payload

GPU Watch

Render pass 번호

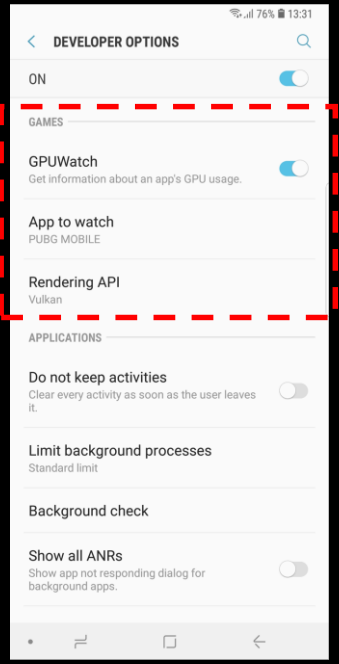
Vertex activity

Fragment activity

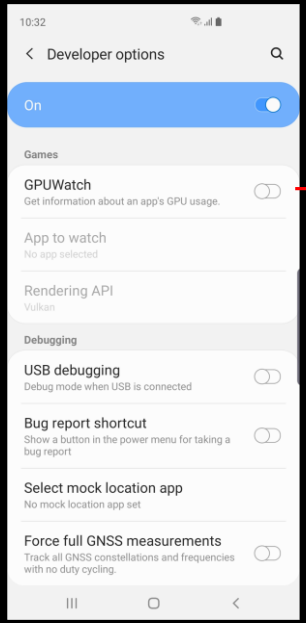
How to use GPU Watch

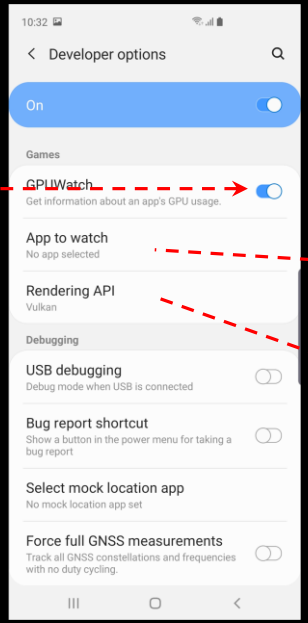
1. GPU Watch 켜기
2. Application 선택
3. Rendering API 선택
   - (Vulkan / OpenGLES)

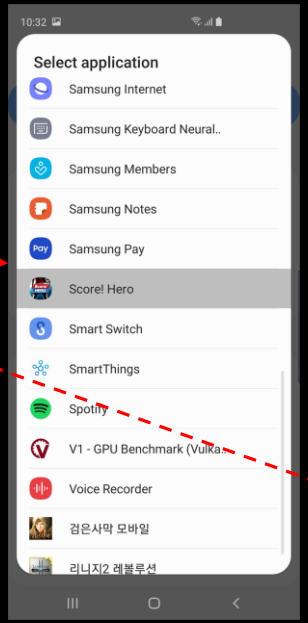
1. Application 실행

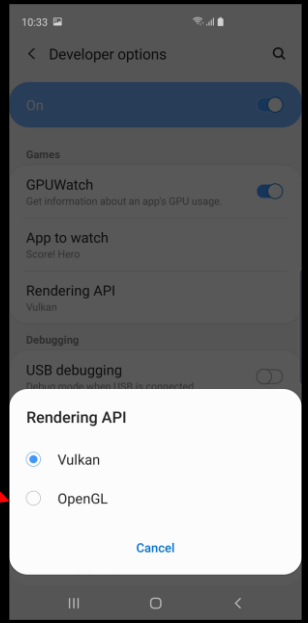


GPU Watch









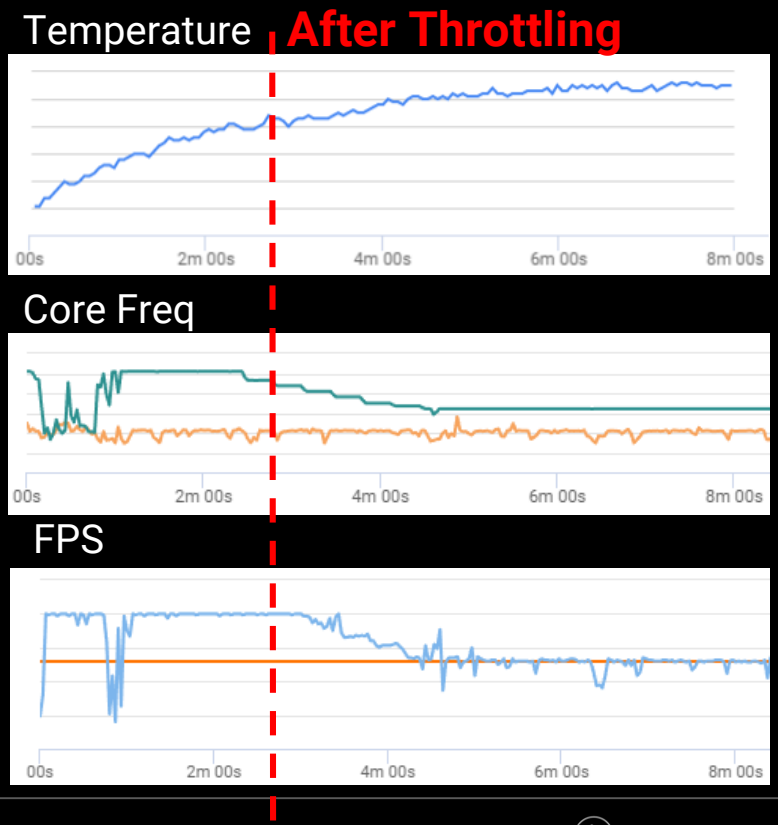
Run Game !! with GPUWatch

전류, 발열, 성능의 상관관계

전류, 발열, 성능의 상관관계

발열 문제점

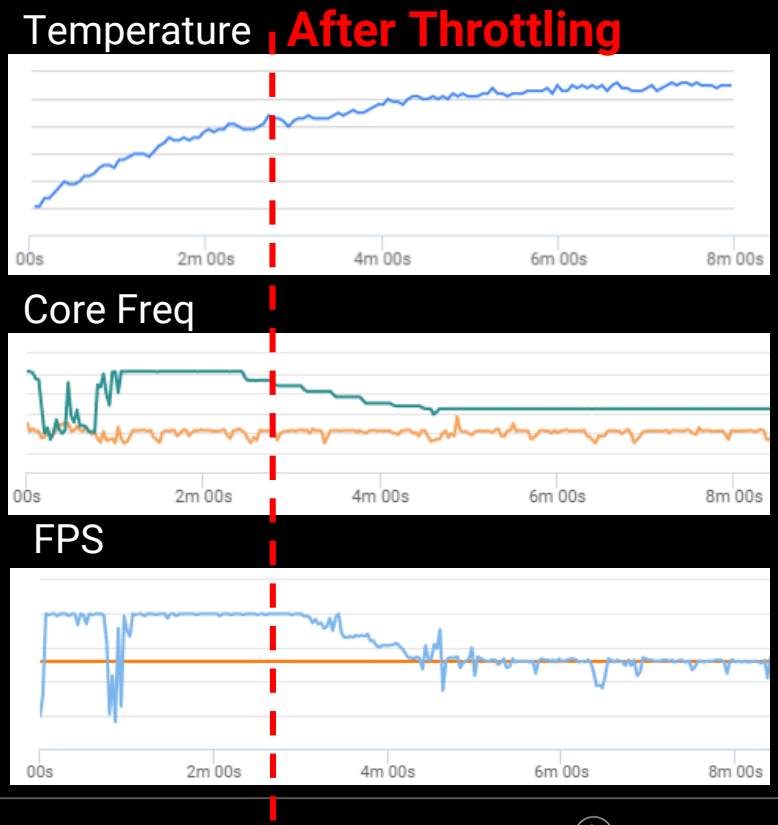
- 발열 증가
- CPU/GPU Frequency 하락
- FPS 하락

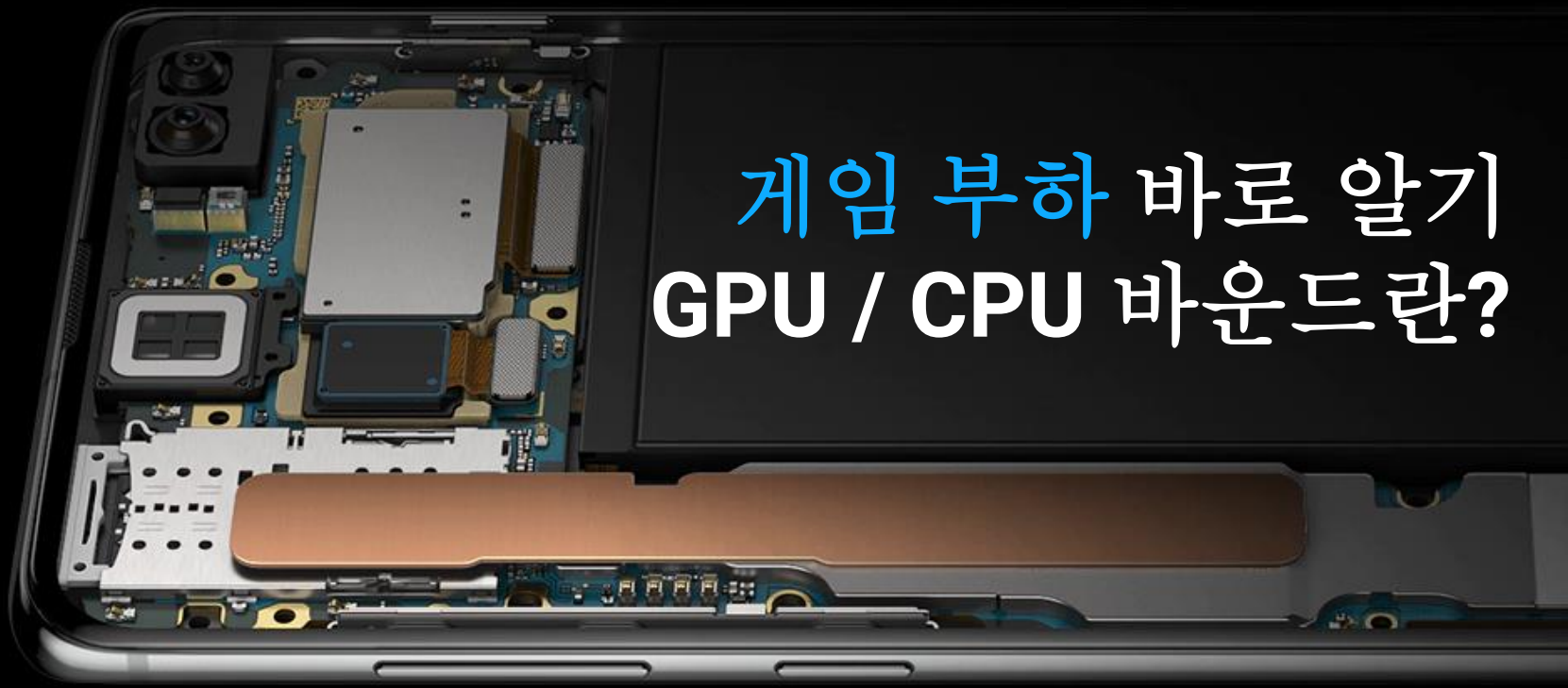


전류, 발열, 성능의 상관관계

최적화 진행

- FPS 상승
- CPU/GPU Frequency 안정
- 온도 하락

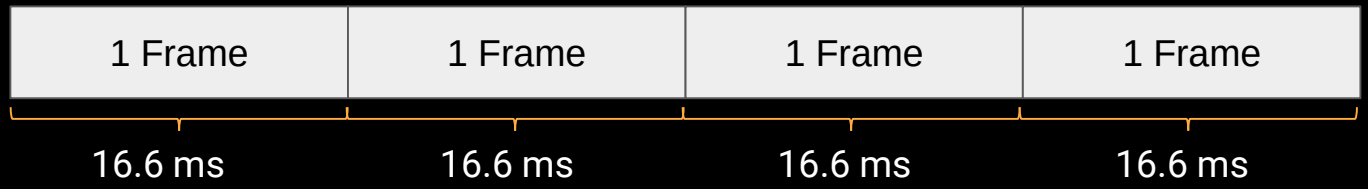




게임 부하 바로 알기
GPU / CPU 바운드란?

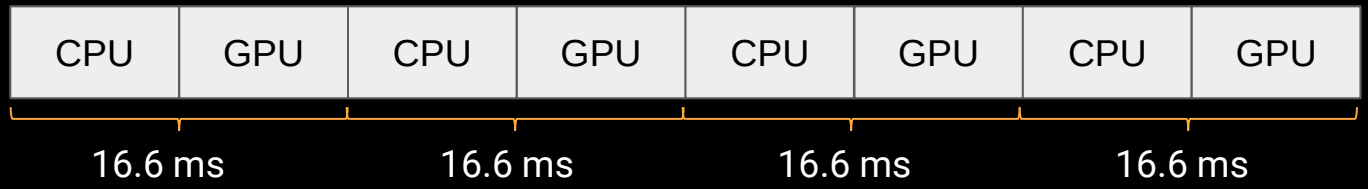
게임부하 바로 알기

60 FPS



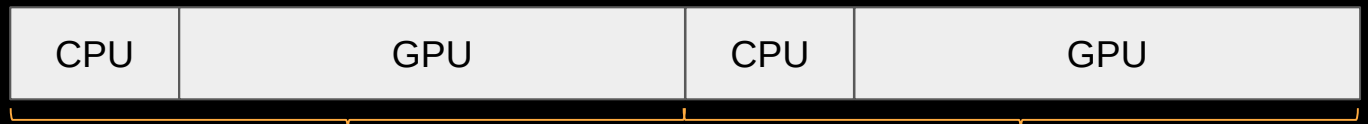
게임부하 바로 알기

60 FPS



게임부하 바로 알기

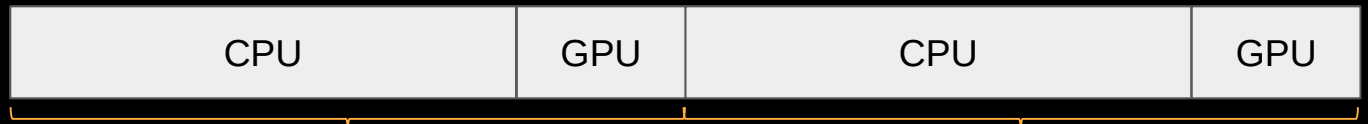
30 FPS - GPU Bound



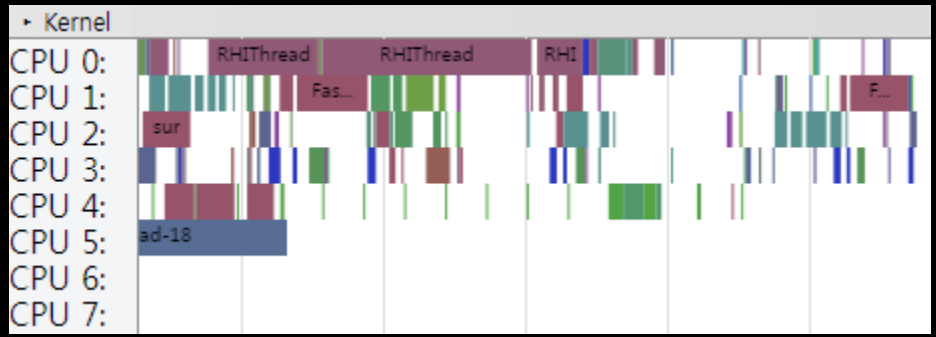
FPS	CPU	GPU
34FPS	11.78%	100.00%
34 - 35 FPS	5 - 16%	100 - 100%
39FPS	15.47%	97.93%
35 - 41 FPS	5 - 18%	91 - 100%

게임부하 바로 알기

30 FPS - CPU Bound



FPS	CPU	GPU
50FPS	14.31%	47.07%
48 - 53 FPS	7 - 22%	46 - 50%
43FPS	15.84%	51.79%
28 - 60 FPS	9 - 32%	20 - 82%



Vulkan API

Image from Pixabay.

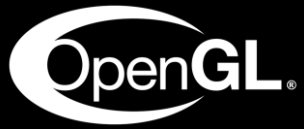
Vulkan API

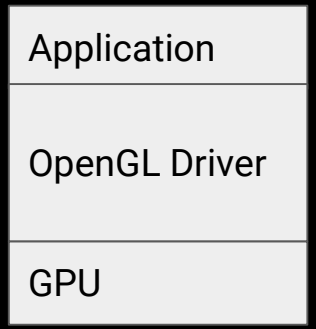
Vulkan API 장점

- 세세하게 컨트롤 가능
- CPU load 줄어들음

Vulkan API 단점

- No driver magic
- Complexity
- Long code





Fortnite Battle Royale

EPIC GAMES

- High Priority Create Thread
- Query Management
- Update DescriptorSet

Before	After
49 FPS	53 FPS

Note9 Adreno - Within Sustainable Power

PUBG MOBILE

Tencent Games

- CPU Occlusion Optimization
- RHI Thread Enable
- Bloom Optimization

Before	After
Up to 30 % Improved FPS	

Note9 Mali - Normal Play Situation

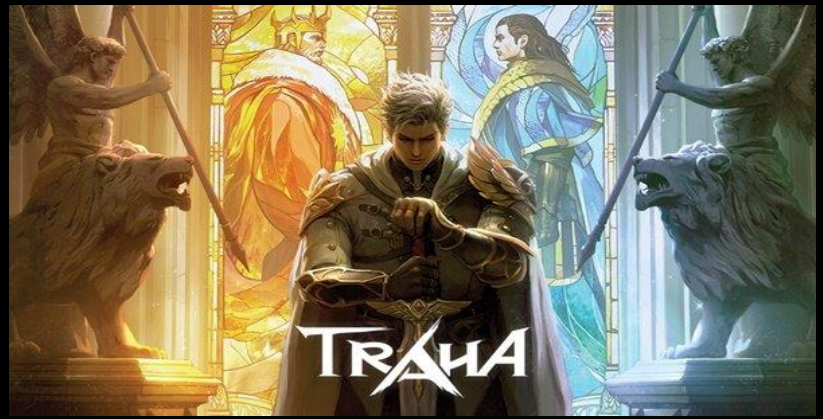
TRAHA

NEXON / MOAI GAMES

- Rendering Order
- RenderPass Load / Store
- Pending RenderPass / Clear

Before	After
35 FPS	56 FPS

S10+ Mali - Within Sustainable Power



Create Pipeline

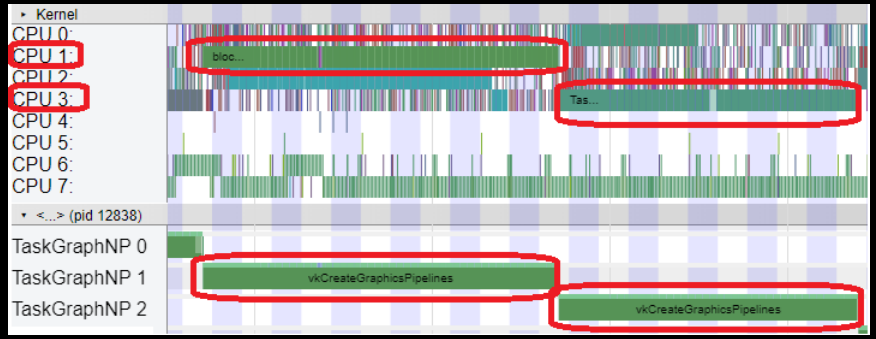
Make High Priority Thread

- 4,000 Create Graphics Pipeline
  - 194,880 ms -> 119,283 ms (In Adreno)

    61% Reduce Init Time

  - 387,120 ms -> 172,812 ms (In Mali)

    44% Reduce Init Time



Create in Small Core

Create in Big Core

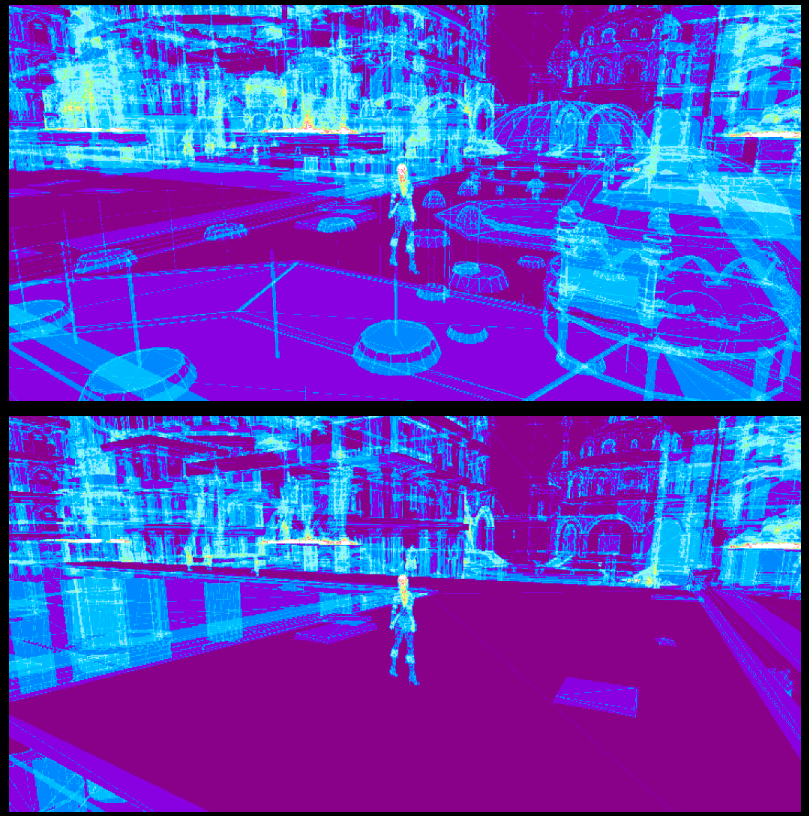
Rendering Order

Optimize logic in DrawVisibleFrontToBack

- Draw Front Object First
- Draw Background Last

App	Device	FPS	CPU	GPU
트라하	SM-G975F	56FPS	13.50%	99.00%
Version: 1.0	Samsung | OS: 9	54 - 57 FPS	6 - 15%	89 - 100%
트라하	SM-G975F	58FPS	13.82%	99.93%
Version: 1.0	Samsung | OS: 9	56 - 59 FPS	10 - 19%	99 - 100%

트라하	SM-G975F	30FPS	7.36%	62.33%
Version: 1.0	Samsung | OS: 9	29 - 30 FPS	6 - 11%	38 - 78%
트라하	SM-G975F	30FPS	7.14%	57.67%
Version: 1.0	Samsung | OS: 9	29 - 30 FPS	5 - 9%	51 - 67%



Query Management

- 다른 물체에 가려진 물체가 많은 경우 Occlusion culling 이 도움이 될 수 있음
  - 총 draw call 횟수 감소
  - CPU / GPU load 감소

- Vulkan 은 Query 들을 포함한 VkQueryPool 지원
- Vulkan 은 Occlusion Test 를 위한 VK_QUERY_TYPE_OCCLUSION 지원

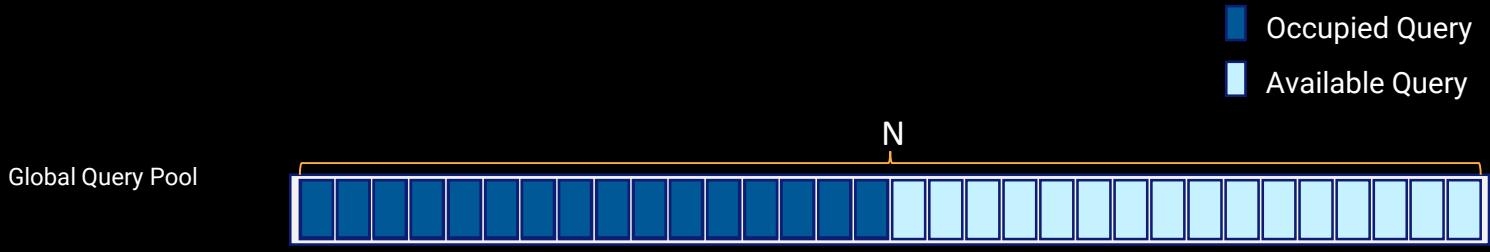
```
VkResult vkCreateQueryPool(
    VkDevice                    device,
    const VkQueryPoolCreateInfo* pCreateInfo,
    const VkAllocationCallbacks* pAllocator,
    VkQueryPool*                pQueryPool);
```

```
typedef enum VkQueryType {
    VK_QUERY_TYPE_OCCLUSION = 0,
    VK_QUERY_TYPE_PIPELINE_STATISTICS = 1,
    VK_QUERY_TYPE_TIMESTAMP = 2,
...
} VkQueryType;
```

Query Management

- Query Pool 생성
  - vkCreateQueryPool
- Query 를 이용한 Draw Call
  - vkCmdBeginQuery
  - vkCmdDraw
  - vkCmdEndQuery
- Query 결과 가져오기 및 재사용, 재설정
  - vkGetQueryPoolResets
  - vkCmdResetQueryPool

Query Management



- 일부 객체는 Query 결과를 가져오지 않음
- 이러한 것들이 단편화(fragmentation) 발생

Query Management

- 한번의 API 호출로 여러 순차 Query 처리 가능

```
VkResult vkGetQueryResults(
    VkDevice                                    device,
    VkQueryPool                                 queryPool,
    uint32_t                                    firstQuery,
    uint32_t                                    queryCount,
    size_t                                      dataSize,
    void*                                       pData,
    VkDeviceSize                                stride,
    VkQueryResultFlags  flags);
```

```
 void vkCmdResetQueryPool(
    VkCommandBuffer     commandBuffer,
    VkQueryPool                 queryPool,
    uint32_t
    uint32_t
    uint32_t
            queryCount);
```

- 여러 Query 사용시에 파편화를 막기 위한 방법
  - vkGetQueryPool 결과 및 vkCmdResetQueryPool 을 여러번 호출해야 함
- 전역 Pool 이 아닌, 각 Frame 마다 전용 Query Pool 사용

Query Management

	Global Query Pool	Dedicated Query Pool
Pseudo code	Allocate a global QueryPool RenderFrame: For i=0 to objects.size res = Get a previous query result reset a query if res is visible draw a object else query for object	RenderFrame: ResultList = Get all previous query results Reset the QueryPool Get a free QueryPool For i=0 to objects.size res = ResultList[objects.QueryIdx] if res is visible draw a object else query for object
vkGetQueryPoolResults	1 frame 에 최대 객체 수 만큼 호출	1 frame 에 1회 호출
vkCmdResetQueryPool	1 frame 에 최대 객체 수 만큼 호출	1 frame 에 1회 호출

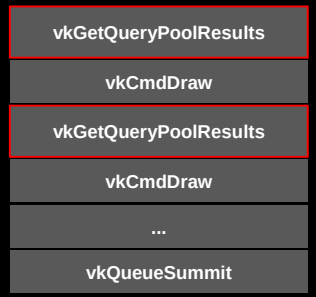
Query Management

- 한번의 API 호출로
  여러 순차 Query 처리
- 각 Frame 마다 전용 Pool 사용

30% 향상

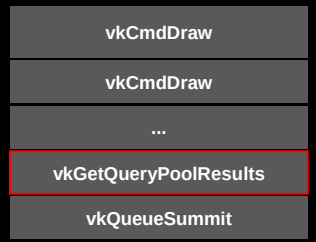
Name	Wall Duration	Occurrences
Acquire and QueuePresent	61.510 ms	1
vkGetQueryPoolResults	10.046 ms	30
vkCmdDrawIndexed	9.380 ms	403
vkCmdBindDescriptorSets	6.737 ms	412
vkCmdEndQuery	3.286 ms	300
vkCmdBeginQuery	2.821 ms	300
vkQueuePresentKHR	2.788 ms	1
vkQueueSubmit	1.603 ms	2

Global QueryPool



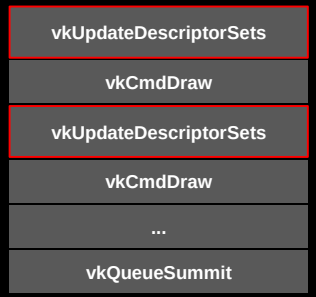
Name	Wall Duration	Occurrences
Acquire and QueuePresent	50.680 ms	1
vkCmdDrawIndexed	9.220 ms	399
vkCmdBindDescriptorSets	6.763 ms	408
vkCmdEndQuery	3.459 ms	300
vkCmdBeginQuery	2.992 ms	300
vkQueuePresentKHR	1.956 ms	1
vkQueueSubmit	1.812 ms	2
vkGetQueryPoolResults	1.010 ms	1
vkResetCommandBuffer	0.598 ms	2

Dedicated QueryPool



Update DescriptorSet

- 매 draw 마다 UpdateDescriptorSets 호출
  - Texture, Buffer, Buffer Offset, Buffer View 변경을 위해 호출



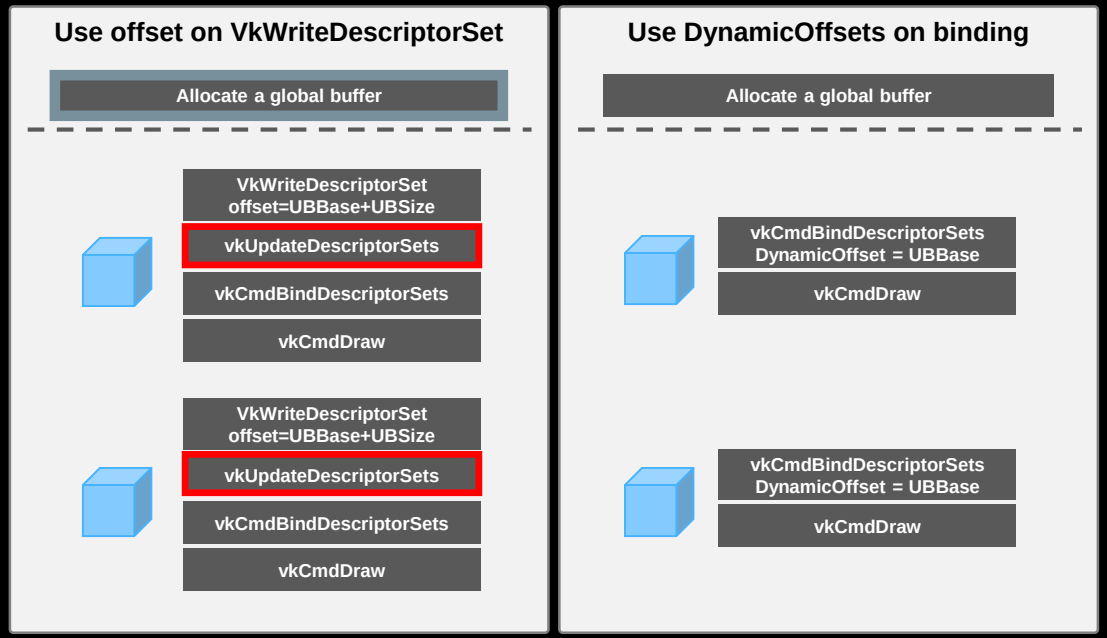

```
typedef struct VkWriteDescriptorSet {
    VkStructureType                  sType;
    const void*                      pNext;
    VkDescriptorSet                  dstSet;
    uint32_t                         dstBinding;
    uint32_t                         dstArrayElement;
    uint32_t                         descriptorCount;
    VkDescriptorType                 descriptorType;
    const VkDescriptorImageInfo*     pImageInfo;
    const VkDescriptorBufferInfo*    pBufferInfo;
    const VkBufferView*              pTexelBufferView;
} VkWriteDescriptorSet;
```

Name	Wall Duration	Occurrences
Acquire and QueuePresent	90.843 ms	1
vkUpdateDescriptorSets	25.928 ms	500
vkCmdDrawIndexed	22.503 ms	500
vkCmdBindDescriptorSets	10.820 ms	500
vkQueueSubmit	1.013 ms	1

Update DescriptorSet

Bind DescriptorSets 를 이용해서 buffer offset 이동 가능

```
void vkCmdBindDescriptorSets(
    VkCommandBuffer      commandBuffer,
    VkPipelineBindPoint pipelineBindPoint,
    VkPipelineLayout     layout,
    uint32_t
            firstSet,
    uint32_t
            descriptorSetCount,
    const VkDescriptorSet* pDescriptorSets,
    uint32_t
            dynamicOffsetCount,
    const uint32_t
            pDynamicOffsets);
```



Update DescriptorSet

- Texture, Buffer Handle 을 이용해서 DescriptorSet Caching
- Bind DescriptorSets를 이용해서 Buffer Offset 이동
- Update DescriptorSets 호출 자제

26% 향상

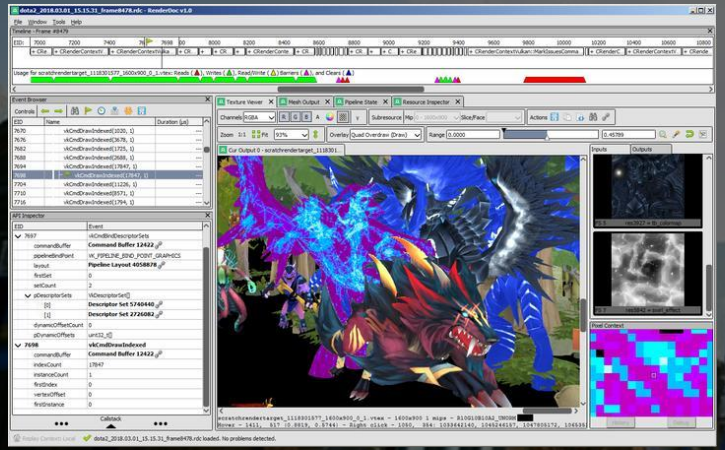
Name	Wall Duration	Occurrences
Acquire and QueuePresent	90.843 ms	1
vkUpdateDescriptorSets	25.928 ms	500
vkCmdDrawIndexed	22.503 ms	500
vkCmdBindDescriptorSets	10.820 ms	500
vkQueueSubmit	1.013 ms	1

Draw 마다 vkUpdateDescriptorSets 호출

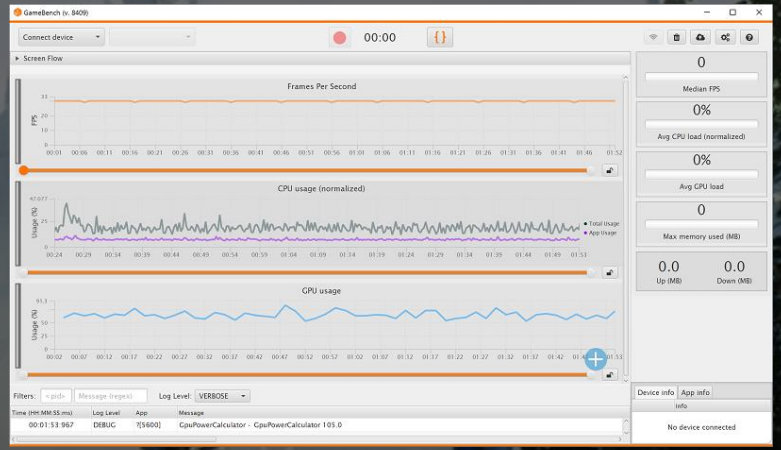
Name	Wall Duration	Occurrences
Acquire and QueuePresent	63.452 ms	1
vkCmdDrawIndexed	24.019 ms	500
vkCmdBindDescriptorSets	10.949 ms	500
vkQueuePresentKHR	2.272 ms	1

Use Caching and DynamicOffset

0. 시작하기 전에 1/3



Vulkan / GLES 구조 분석
RenderDoc
https://renderdoc.org/



성능 분석
GameBench
https://www.gamebench.net/

0. 시작하기 전에 2/3

불칸 렌더러에 삼성 특화된 구조가 존재함!

PENDING RENDERPASS
PENDING CLEAR
PENDING PIPELINE BARRIER

0. 시작하기 전에 3/3

PENDING RENDERPASS

RHI Command	Original Logic	Pending Renderpass
RHISetRenderTargets	vkCmdBeginRenderpass	Skipped
RHISetRenderTargets	vkCmdEndRenderpass / vkCmdBeginRenderpass	
RHIDrawIndexedPrimitive	vkCmdDrawIndexed	vkCmdBeginRenderpass / vkCmdDrawIndexed
RHIDrawIndexedPrimitive	vkCmdDrawIndexed	vkCmdDrawIndexed
RHIDrawIndexedPrimitive	vkCmdDrawIndexed	vkCmdDrawIndexed
RHISetRenderTargets	VkCmdEndRenderpass / vkCmdBeginRenderpass	vkCmdEndRenderpass / vkCmdBeginRenderpass
RHIDrawIndexedPrimitive	vkCmdDrawIndexed	vkCmdDrawIndexed
RHIDrawIndexedPrimitive	vkCmdDrawIndexed	vkCmdDrawIndexed
RHIDrawIndexedPrimitive	vkCmdDrawIndexed	vkCmdDrawIndexed
RHIDrawIndexedPrimitive	vkCmdDrawIndexed	vkCmdDrawIndexed
RHIDrawIndexedPrimitive	vkCmdDrawIndexed	vkCmdDrawIndexed
	vkCmdEndRenderpass	vkCmdEndRenderpass

1. HDR RT Format을 최적화해보자

기본적으로 R16G16B16**A16** Float가 사용된다 * IsMobileHDR32bpp 경우 제외

- 하지만 모든 패스가 알파 채널을 필요로 하는 것은 아님!
  - BloomSetup 및 SunMask 패스는 알파 채널 필요함
- 단 DoF 적용시는 제외, 하지만 많은 게임이 비활성화 가능한 옵션으로 지원중
- 알파 채널이 없으면 64bit > 32bit 포맷으로 사용이 가능!
  - r.Mobile.SceneColorFormat을 2로 설정 (PF_FloatR11G11B10)

R16G16B16**A16**F > R16G16B16F > R11G11B10F (B11G11R10F)

1. HDR RT Format을 최적화해보자

TRAHA, © NEXON Korea Corporation & MOAI GAMES All Right Reserved.

Device	FPS	FPS Stability	Power	CPU	GPU
SM-G970F	30FPS	100%	257mA	8.95%	77.46%
Samsung | OS: 9	29 - 30 FPS			5 - 12%	59 - 99%
SM-G970F	30FPS	100%	257mA	9.20%	78.49%
Samsung | OS: 9	29 - 30 FPS			5 - 13%	57 - 99%

1. HDR RT Format을 최적화해보자

단! 변경 하려는 패스가 알파 채널이 꼭 필요한지 확인해야함.

- 해당 패스의 Fragment (Pixel) 쉐이더를 확인
  i.e. outColor.a += Coc(InDepth), ES2_USE_DEPTHTEXTURE / ES2_USE_DOF / ES2_USE_SUN 참고

TRAHA, © NEXON Korea Corporation & MOAI GAMES All Right Reserved.

Depth of Field 렌더링 이슈
BloomSetup시 Depth를 담아야 할 알파 채널을
날려버렸기 때문에 발생한 문제

1. HDR RT Format을 최적화해보자

```
EPixelFormat FSceneRenderTargets::GetDesiredMobileSceneColorFormat(bool bNeedAlphaChannel/*=true*/) const
{
      EPixelFormat DefaultColorFormat = (!IsMobileHDR() || IsMobileHDR32bpp() ||
       !GSupportsRenderTargetFormat_PF_FloatRGBA) ? PF_B8G8R8A8 : PF_FloatRGBA;
            ....
            // 현재 DoF를 사용하는지 체크한다 ( 게임 설정과 연동 )
            static const auto CVarEnableDoF = IConsoleManager::Get().FindTConsoleVariableDataInt(
                                          TEXT("r.EnablePostProcessingDof"));
      int32 MobileSceneColor = CVar->GetValueOnRenderThread();
      switch (MobileSceneColor)
      {
            case 1: MobileSceneColorBufferFormat = PF_FloatRGBA; break;
            // 알파채널이 꼭 필요한 경우가 아니라면 RGB 32Bit 포맷을 사용
                        case 2: MobileSceneColorBufferFormat = bNeedAlphaChannel &&
                  CVarEnableDoF->GetValueOnAnyThread() == 1 ? PF_FloatRGBA : PF_FloatR11G11B10; break;
            case 3: MobileSceneColorBufferFormat = PF_B8G8R8A8; break;
                        default:break;
      }
      return GPixelFormats[MobileSceneColorBufferFormat].Supported ?
                  MobileSceneColorBufferFormat : DefaultColorFormat;
}
```

1. HDR RT Format을 최적화해보자

```
static EPixelFormat GetHDRPixelFormat(bool bNeedAlphaChannel = false)
{
            //우선적으로 RGB Format을 사용하려고 시도
            return FSceneRenderTargets::Get(FRHICommandListExecutor::GetImmediateCommandList()).
                  GetDesiredMobileSceneColorFormat(bNeedAlphaChannel);
}

//아래 BloomSetup, SunMask 패스는 알파 채널이 필요하다.
FPooledRenderTargetDesc FRCPassPostProcessBloomSetupES2::ComputeOutputDesc(EPassOutputId InPassOutputId)
const
{
      ....
            Ret.Format = GetHDRPixelFormat(true);
            ....
}

FPooledRenderTargetDesc FRCPassPostProcessSunMaskES2::ComputeOutputDesc(EPassOutputId InPassOutputId) const
{
            ....
            Ret.Format = GetHDRPixelFormat(true);
            ....
}
```

2. 렌더타겟을 Don’t Care로 열어보자

Renderpass는 L/S Operation이 존재, 이중 Load는 아래 3가지 Operation을 지원함

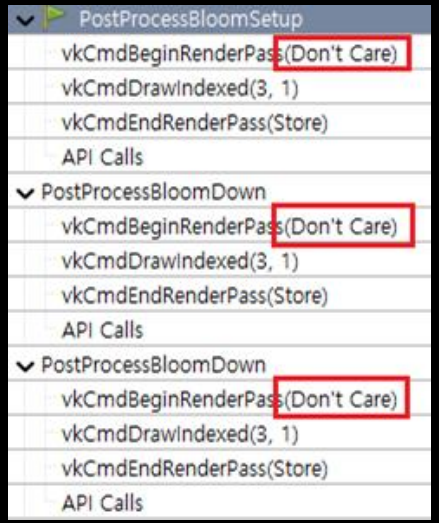
- LOAD_OP_CLEAR
  - 새로운 렌더패스를 시작할때, 바인드된 Color / Depth Stencil 영역을 클리어함
- LOAD_OP_LOAD
  - 새로운 렌더패스를 시작할때, 이전 렌더패스의 결과 (Color / Depth Stencil) 를 로드함
- LOAD_OP_DONT_CARE
  - 새로운 렌더패스를 시작할때, 아무것도 하지 않음 (실제 동작은 드라이버 구현에 맡김)

2. RT을 Don’t Care로 열어보자

대부분의 Post-Processing 패스들이 Clear / Load를 사용 중.
하지만 Bloom의 Down / Up Sample 패스등은 이전 결과를 로드 할 필요가 없음.

* 이전 패스의 결과를 텍스쳐로 가져오기 때문

FRCPassPostProcessBloomSetupES2
FRCPassPostProcessBloomDownES2
FRCPassPostProcessBloomUpES2
FRCPassPostProcessSunMaskES2
FRCPassPostProcessSunAlphaES2
FRCPassPostProcessSunBlurES2
FRCPassPostProcessSunMergeES2
FRCPassPostProcessTonemap



2. RT을 Don’t Care로 열어보자

TRAHA, © NEXON Korea Corporation & MOAI GAMES All Right Reserved.

Device	FPS	FPS Stability	Power	CPU	GPU
SM-G970F	30FPS	100%	173mA	8.69%	80.32%
Samsung | OS: 9	29 - 30 FPS			6 - 12%	55 - 100%
SM-G970F	30FPS	100%	257mA	8.95%	77.46%
Samsung | OS: 9	29 - 30 FPS			5 - 12%	59 - 99%

2. RT을 Don’t Care로 열어보자

```
void FRCPassPostProcessBloomDownES2::Process(FRenderingCompositePassContext& Context)
....
            if (DestRenderTarget.TargetableTexture->GetClearColor() == FLinearColor::Black)
            {
                        FRHIRenderTargetView View;
            bool IsVulkan = IsVulkanMobilePlatform(Context.View.GetShaderPlatform());

            // Vulkan을 사용하는 경우에만 RT를 ENoAction으로 연다
            View = FRHIRenderTargetView(DestRenderTarget.TargetableTexture, IsVulkan ?
            ERenderTargetLoadAction::ENoAction :
            ERenderTargetLoadAction::EClear);

                        FRHISetRenderTargetsInfo Info(1, &View, FRHIDepthRenderTar
                        Context.RHICmdList.SetRenderTargetsAndClear(Info);
            }
            else
....
```

```
enum class ERenderTargetLoadAction : u
{
    ENoAction,    OP_DONTCARE
    ELoad,        OP_LOAD
    EClear,       OP_CLEAR

    Num,
    NumBits = 2,
};
static_assert((uint32)ERenderTargetLoa
```

2. RT을 Don’t Care로 열어보자

Store는 아래 2가지 Operation을 지원함

* 단 사용하지 않는 렌더버퍼는 생성하지 않는것이 최선

- STORE_OP_STORE
  - 현재 렌더패스의 결과(Color / Depth Stencil)를 저장
- STORE_OP_DONT_CARE
  - 새로운 렌더패스를 저장할때, 아무것도 하지 않음 (실제 동작은 드라이버 마다 다름)

2. RT을 Don’t Care로 열어보자

Load / Store 모두 Color / Depth / Stencil 버퍼에 대한 각각의 조합이 가능.

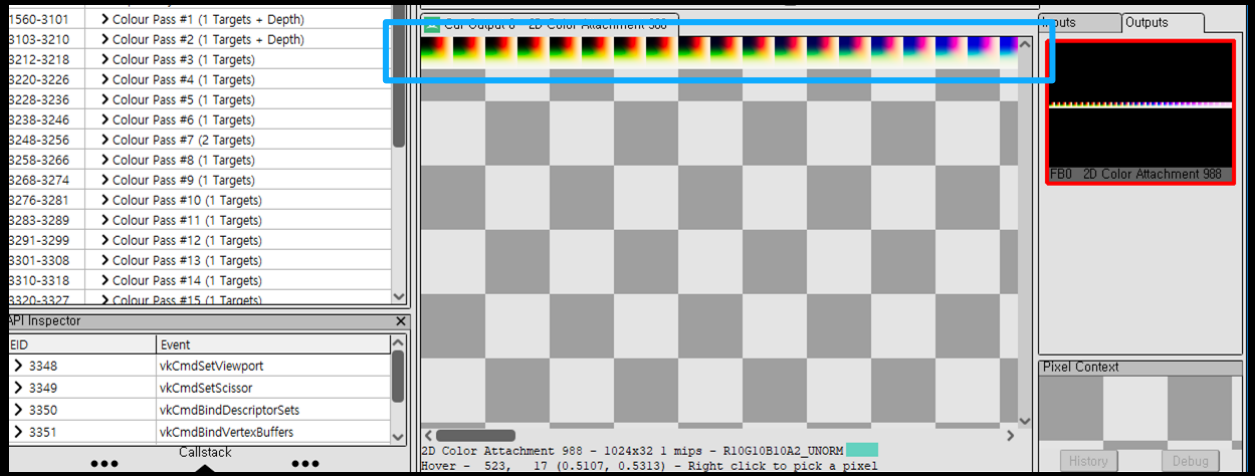
* EDepthStencilTargetActions / ERenderTargetActions 참고 (RHIResources.h)

```
//컬러버퍼에 대한 설정 Load - Clear, Store - Store
EColorTargetActions ColorLoadStoreAction = ERenderTargetActions::Clear_Store;

//뎁스 버퍼에 대한 설정 Load - DontLoad(DontCare), Store - Store
EDepthStencilTargetActions DepthLoadStoreAction = DepthStencilTargetActions::DontLoad_StoreDepthStencil;

//FRHIRenderPassInfo를 통해 명시적으로 RT를 열수 있다.
FRHIRenderPassInfo RPInfo( SceneColor,
            ColorLoadStoreAction,
            SceneContext.GetSceneDepthSurface(),
            DepthLoadStoreAction,
            FExclusiveDepthStencil::DepthWrite_StencilNop
);

RHICmdList.BeginRenderPass(RPInfo, TEXT("BasePass"));
```

3. LUT를 매번 그릴 필요는 없다

톤맵에 사용되는 LUT 텍스쳐는 대부분 매 프레임마다 그려짐

- 프레임 N vs N+1 비교시 변경되는 값이 없다면 이전 LUT를 재사용해도된다.



3. LUT를 매번 그릴 필요는 없다

TRAHA, © NEXON Korea Corporation & MOAI GAMES All Right Reserved.

Device	FPS	FPS Stability	Power	CPU	GPU
SM-G970F	30FPS	100%	192mA	9.04%	78.67%
Samsung | OS: 9	29 - 30 FPS			5 - 14%	60 - 99%
SM-G970F	30FPS	100%	257mA	8.95%	77.46%
Samsung | OS: 9	29 - 30 FPS			5 - 12%	59 - 99%

3. LUT를 매번 그릴 필요는 없다

예시) Contrast, Saturation, Gamma등 변경되는 값을 기준으로 비교하여 재사용

```
static FRCPassPostProcessTonemap* AddTonemapper(
....
            bool bIsNeedToUpdateLUT = false;

            //아래 저장되는 값들은 실제 사용되는 LUT 설정에 따라 변경되어야 함.
            static struct LUT_Setting
            {
                        FVector4 ColorContrast;
                        FVector4 ColorSaturation;
                        FVector4 SceneColorGamma;
                        UTexture* ColorGradingLUT;
                        bool CurrentMSAASetting;
                        float ScreenPercentage;
                        TArray<FFinalPostProcessSettings::FLUTBlenderEntry, TInlineAllocator<8>>
SavedContributingLUTs;
            } GSavedLUTSetting;
```

3. LUT를 매번 그릴 필요는 없다

예시) Contrast, Saturation, Gamma등 변경되는 값을 기준으로 비교하여 재사용

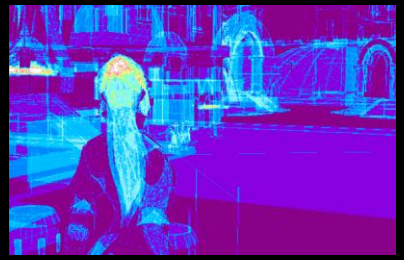
```
//Context.View.FinalPostProcessSettings 과 저장된 GSavedLUTSetting를 비교한다.
if (GSavedLUTSetting.ColorContrast != Context.View.FinalPostProcessSettings.ColorContrast ||
    GSavedLUTSetting.ColorSaturation != Context.View.FinalPostProcessSettings.ColorSaturation ||
    GSavedLUTSetting.SceneColorGamma != Context.View.FinalPostProcessSettings.ColorGamma ||
    GSavedLUTSetting.ColorGradingLUT != Context.View.FinalPostProcessSettings.ColorGradingLUT ||
    GSavedLUTSetting.ScreenPercentage != Context.View.FinalPostProcessSettings.ScreenPercentage)
{
  // 값이 변경되었으니, LUT Texture 역시 다시 그려야한다.
    bIsNeedToUpdateLUT = true;
    GSavedLUTSetting.ColorContrast = Context.View.FinalPostProcessSettings.ColorContrast;
    GSavedLUTSetting.ColorSaturation = Context.View.FinalPostProcessSettings.ColorSaturation;
    GSavedLUTSetting.SceneColorGamma = Context.View.FinalPostProcessSettings.ColorGamma;
    GSavedLUTSetting.ColorGradingLUT = View.FinalPostProcessSettings.ColorGradingLUT;
    GSavedLUTSetting.ScreenPercentage = Context.View.FinalPostProcessSettings.ScreenPercentage;
}
....
//SavedContributingLUTs 비교 생략
```

3. LUT를 매번 그릴 필요는 없다

예시) Contrast, Saturation, Gamma등 변경되는 값을 기준으로 비교하여 재사용

```
if (GSavedLUTSetting.SavedContributingLUTs.Num() !=
Context.View.FinalPostProcessSettings.ContributingLUTs.Num() ||
     Context.View.FinalPostProcessSettings.ContributingLUTs.Num() == 1)
{
      bIsNeedToUpdateLUT = true;
            GSavedLUTSetting.SavedContributingLUTs = View.FinalPostProcessSettings.ContributingLUTs;
}

bIsNeedToUpdateLUT = IsVulkanMobilePlatform(View.GetShaderPlatform()) ? bIsNeedToUpdateLUT : true;
FRenderingCompositeOutputRef TonemapperCombinedLUTOutputRef;

//업데이트가 필요한 경우에만 LUT 패스를 진행
if (StereoPass != eSSP_RIGHT_EYE  && bIsNeedToUpdateLUT)
{
      FRenderingCompositePass* CombinedLUT = Context.Graph.RegisterPass(new(FMemStack::Get())
      FRCPassPostProcessCombineLUTs(View.GetShaderPlatform(), View.State == nullptr, bIsComputePass));
                        TonemapperCombinedLUTOutputRef = FRenderingCompositeOutputRef(CombinedLUT);
}
```

4. Rendering 순서에는 정답이 없다.



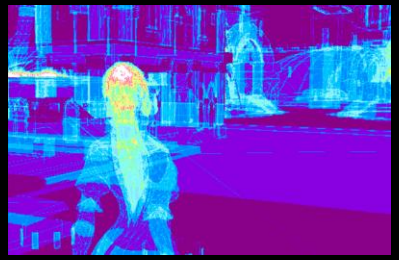
바꾼 렌더링 순서:

DynamicData (Opaque)

DynamicData (Masked)

Static (EBasePass_Default)

Static (EBasePass_Masked)



원본 렌더링 순서:

Static (EBasePass_Default)

DynamicData (Opaque)

Static (EBasePass_Masked)

DynamicData (Masked)

App	Device	FPS	FPS Stability	Power	CPU	GPU
Traha	SM-G970F	59FPS	100%	427mA	12.91%	80.23%
Version: 1.0	Samsung | OS: 9	47 - 60 FPS			9 - 18%	49 - 98%
Traha	SM-G970F	59FPS	99%	359mA	13.19%	85.69%
Version: 1.0	Samsung | OS: 9	46 - 60 FPS			8 - 18%	71 - 99%

4. Rendering 순서에는 정답이 없다.

```
void FMobileSceneRenderer::RenderMobileBasePass(FRHICommandListImmediate& RHICmdList, const
TArrayView<const FViewInfo*> PassViews)
...

// 1. 다이나믹 데이터들을 먼저 그린다 ( 보통 캐릭터들 )
// render dynamic opaque primitives (or all if Wireframe)
const bool bWireframe = !!ViewFamily.EngineShowFlags.Wireframe;
RenderMobileBasePassDynamicData(RHICmdList, View, DrawRenderState, BLEND_Opaque ...);

// render dynamic masked primitives(or none if Wireframe)
if (!bWireframe)
{
            RenderMobileBasePassDynamicData(RHICmdList, View, DrawRenderState, BLEND_Masked ...);
}

...
// 이제 스테이틱 데이터들을 그리자
```

4. Rendering 순서에는 정답이 없다.

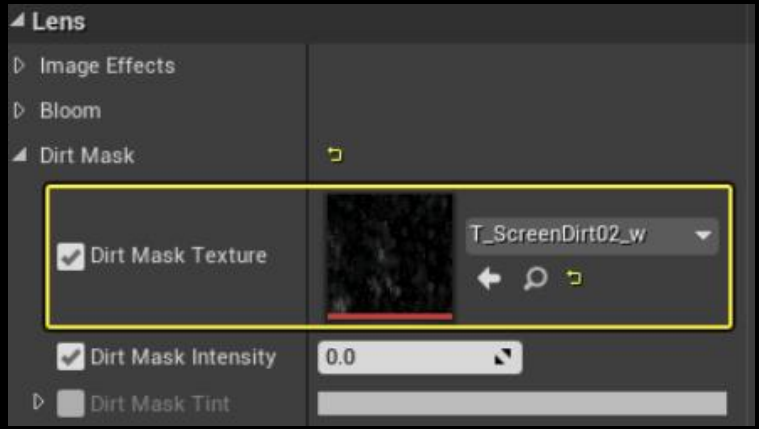
```
// Render the base pass static data
if (SortMode == EBasePassSort::SortPerMesh)
{
      DrawVisibleFrontToBack(RHICmdList, Scene, EBasePass_Default...);
}
else
{
            DrawVisible(RHICmdList, Scene, EBasePass_Default...);
}
...
// Issue static draw list masked draw calls last, as PVR wants it
if (SortMode == EBasePassSort::SortPerMesh)
{
      DrawVisibleFrontToBack(RHICmdList, Scene, EBasePass_Masked...);
}
else
{
            DrawVisible(RHICmdList, Scene, EBasePass_Masked...);
}
```

5. Bloom: DirtMask 최적화

Post-Processing의 Bloom DirtMask는 보통 고해상도 이미지가 사용된다.

- 하지만 Bloom의 rgb x DirtMaskColor에 의해 결과물이 보여짐
- 즉 Bloom Intensity ≒ 0.0 인 곳에서는 고해상도 Mask Texture가 불필요.

```
TonemapCommonPS(
...
float3 BloomDirtMaskColor =
Texture2DSample(BloomDirtMask.Mask,
BloomDirtMask.MaskSampler,
DirtViewportUV).rgb * BloomDirtMask.Tint.rgb;
...
LinearColor += CombinedBloom.rgb *
(ColorScale1.rgb + BloomDirtMaskColor);
```



5. Bloom: DirtMask 최적화

Bloom이 활성화 되어 있는 경우에만 DirtMask를 적용

```
void FPostProcessTonemapShaderParameters::Set(
{
....
      if (BloomDirtMaskParam.IsBound())
            {
                        FBloomDirtMaskParameters BloomDirtMaskParams;
                        FLinearColor Col = Settings.BloomDirtMaskTint * Settings.BloomDirtMaskIntensity;
                        BloomDirtMaskParams.Tint = FVector4(Col.R, Col.G, Col.B, 0.f /*unused*/);
                        BloomDirtMaskParams.Mask = GSystemTextures.BlackDummy-
>GetRenderTargetItem().TargetableTexture;
                        const bool IsBloomEnabled = IsVulkanMobilePlatform(ViewFamily.GetShaderPlatform()) ?
                                    Context.View.FinalPostProcessSettings.BloomIntensity > 0 : true;
                        //Bloom이 실제로 보일때에만 DirtTexture를 설정한다. ( 기본은 BlackDummy 사용 )
                        if (Settings.BloomDirtMask && Settings.BloomDirtMask->Resource && IsBloomEnabled)
            {
                                    BloomDirtMaskParams.Mask = Settings.BloomDirtMask->Resource->TextureRHI;
                        }
....
```

6. Decal Pass 최적화

실제 Decal 렌더링이 필요한 경우에만 Pass를 열자.

```
void FMobileSceneRenderer::RenderDecals(FRHICommandListImmediate& RHICmdList)
{
....
#if 1
            bool NeedToStartRenderpass = true;
#else
      // 데칼 유무와 관계없이 밖에서 렌더패스를 열게 되어 있음
            FSceneRenderTargets& SceneContext = FSceneRenderTargets::Get(RHICmdList);
            SceneContext.BeginRenderingSceneColor(RHICmdList,
            ESimpleRenderTargetMode::EExistingColorAndDepth, FExclusiveDepthStencil::DepthRead_StencilRead);
            RHICmdList.ApplyCachedRenderTargets(GraphicsPSOInit);
#endif
....
```

6. Decal Pass 최적화

실제 Decal 렌더링이 필요한 경우에만 Pass를 열자.

```
if (SortedDecals.Num())
{
      // 데칼이 있는 경우에만 렌더패스를 열자
      if (NeedToStartRenderpass)
            {
                  EShaderPlatform ShaderPlatform = ViewFamily.GetShaderPlatform();
                  bool bIsNeedToBeginRenderpass = RHICmdList.IsOutsideRenderPass();
            if (ShaderPlatform != SP_METAL && bIsNeedToBeginRenderpass)
                  {
                        FSceneRenderTargets& SceneContext =
FSceneRenderTargets::Get(RHICmdList);

                  SceneContext.BeginRenderingSceneColor(RHICmdList,ESimpleRenderTargetMode::EExisting
                ColorAndDepth, FExclusiveDepthStencil::DepthRead_StencilRead);
                  }
                  RHICmdList.ApplyCachedRenderTargets(GraphicsPSOInit);
                  NeedToStartRenderpass = false;
            }
....
```

7. 저사양 Bloom도 고려해볼만하다

단 Small Bloom은 너무 퀄리티가 떨어짐. Down / Up 을 적절하게 조절해보자.

Down Sample (**1 / 6**)
Down Sample (**1 / 18**)
Up Sample (**1 / 18**)
Up Sample (**1 / 4**)

Customized Bloom

Original Bloom

Down Sample (**View / 4**)
Down Sample (**View / 8**)
Down Sample (**View / 16**)
Down Sample (**View / 32**)
Up Sample (**View / 32**)
Up Sample (**View / 16**)
Up Sample (**View / 8**)

7. 저사양 Bloom도 고려해볼만하다

단 Small Bloom은 너무 퀄리티가 떨어짐. Down / Up 을 적절하게 조절해보자.

```
float DownScale = 0.66f * 6.0f;
{
            FRenderingCompositePass * Pass = Context.Graph.RegisterPass(new(FMemStack::Get())
            FRCPassPostProcessBloomDownES2(PrePostSourceViewportSize / 6, DownScale));
            Pass->SetInput(ePId_Input0, PostProcessBloomSetup);
            PostProcessDownsample2 = FRenderingCompositeOutputRef(Pass);
}

{
            FRenderingCompositePass * Pass = Context.Graph.RegisterPass(new(FMemStack::Get())
            FRCPassPostProcessBloomDownES2(PrePostSourceViewportSize / 18, DownScale));
            Pass->SetInput(ePId_Input0, PostProcessDownsample2);
            PostProcessDownsample3 = FRenderingCompositeOutputRef(Pass);
}
```

7. 저사양 Bloom도 고려해볼만하다

```
float UpScale = 0.66f * 2.0f;
{ //실제로 사용되는 BloomTint 컬러를 고려해야한다.
            FVector4 TintA = FVector4(Settings.Bloom3Tint.R, Settings.Bloom3Tint.G, Settings.Bloom3Tint.B,
0.0f);
            TintA *= View.FinalPostProcessSettings.BloomIntensity;
            FVector4 TintB = FVector4(1.0f, 1.0f, 1.0f, 0.0f);
            FRenderingCompositePass * Pass = Context.Graph.RegisterPass(new(FMemStack::Get())
FRCPassPostProcessBloomUpES2(PrePostSourceViewportSize / 18, FVector2D(UpScale, UpScale), TintA, TintB));
            Pass->SetInput(ePId_Input0, PostProcessDownsample2);
            Pass->SetInput(ePId_Input1, PostProcessDownsample3);
            PostProcessUpsample3 = FRenderingCompositeOutputRef(Pass);
}

{ //실제로 사용되는 BloomTint 컬러를 고려해야한다.
            FVector4 TintA = FVector4(Settings.Bloom2Tint.R, Settings.Bloom2Tint.G, Settings.Bloom2Tint.B,
0.0f);
            TintA *= View.FinalPostProcessSettings.BloomIntensity;
            TintA *= 0.5;
            FVector4 TintB = FVector4(1.0f, 1.0f, 1.0f, 0.0f);
            FRenderingCompositePass * Pass = Context.Graph.RegisterPass(new(FMemStack::Get())
FRCPassPostProcessBloomUpES2(PrePostSourceViewportSize / 4, FVector2D(UpScale, UpScale), TintA, TintB));
            Pass->SetInput(ePId_Input0, PostProcessBloomSetup);
            Pass->SetInput(ePId_Input1, PostProcessUpsample3);
            PostProcessUpsample2 = FRenderingCompositeOutputRef(Pass);
}
```

8. Early-Z를 활용하자

불필요한 discard를 줄이면 Early-Z 를 통한 성능 향상을 얻을 수 있다.

- 단 아래 3가지가 사용되면 Late-Z로 처리된다.
  - Fragment Shader에서 Depth를 변경하는 모든 처리 (DepthBias, gl_FragDepth..)
  - Alpha to Coverage (MSAA)
  - **discard()** in shader

Rasterizer State

Fill Mode:	Solid	Cull Mode:	Front	Front CCW:	✖	Conservative Raster:	Disabled
Depth Bias:	0.00	Depth Bias Clamp:	0.00	Slope-Scaled Bias:	0.00	Overestimate Size:	0.00
Depth Clamp:	✖	Rasterizer Discard:	✖	Line Width:	1.00	Multiview:	Disabled

Multisample State

Sample Count:	1	Sample Shading:	✖
Min Sample Shading:	0.00	Sample Mask:	FFFFFFFF
Alpha to 1:	✖	Alpha to Coverage:	✖

8. Early-Z를 활용하자

대부분의 경우는 Shader내의 Discard가 문제

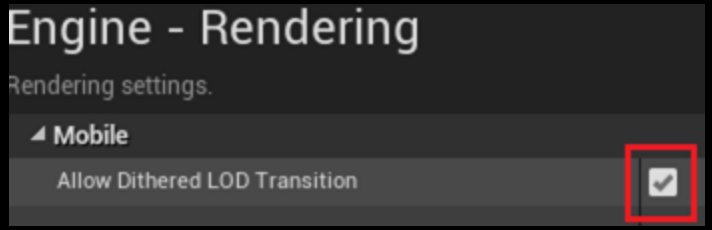
```
void main()
{
    highp float f0 = _15.pu_h[31].x;
    highp vec4 v1 = gl_FragCoord;
    v1.w = 1.0 / gl_FragCoord.w;
    if (in_OUTCLIPDIST < 0.0)
    {
        discard;
    }
```

UE4의 경우 r.AllowGlobalClipPlane가 설정되어 있으면 모든 Draw에 Discard가 적용되어 버린다. 꼭 필요한 경우에만 사용되도록 주의!

MobileBasePassCommon.ush 내부의 USE_PS_CLIP_PLANE / PROJECT_ALLOW_GLOBAL_CLIP_PLANE 참고

8. Early-Z를 활용하자

Instanced Mesh LOD도 Option에 따라 Discard를 사용
USE_DITHERED_LOD_TRANSITION_FOR_INSTANCED / USE_DITHERED_LOD_TRANSITION 참고



Project Setting > Mobile 에서 토글 가능

```
static void
ModifyCompilationEnvironment(EShaderPlatform
Platform, const FMaterial* Material,
FShaderCompilerEnvironment& OutEnvironment)
...
// On mobile dithered LOD transition has to be
explicitly
//enabled in material and project settings

          OutEnvironment.SetDefine(TEXT("USE_DITHERE
D_LOD_TRANSITION_FOR_INSTANCED"), 0);
/*Material->IsDitheredLODTransition() &&
ALLOW_DITHERED_LOD_FOR_INSTANCED_STATIC_MESHES);*/
```

8. Early-Z를 활용하자

경우에 따라 성능 향상을 얻을 수 있다.

TRAHA, © NEXON Korea Corporation & MOAI GAMES All Right Reserved.

FPS	FPS Stability	Power	CPU	GPU
35FPS 33 - 35 FPS	100%	988mA	12.96% 8 - 18%	100.00% 100 - 100%
54FPS 52 - 56 FPS	100%	423mA	15.97% 8 - 21%	99.97% 98 - 100%

S8+ 보통 퀄리티 세팅 - 위, 아래 (Discard 제거시)

60FPS 59 - 60 FPS	100%	224mA	15.00% 8 - 19%	59.00% 53 - 69%
59FPS 58 - 60 FPS	100%	160mA	13.41% 2 - 22%	46.41% 36 - 53%

S8+ 낮음 퀄리티 세팅 - 위, 아래 (Discard 제거시)

9. VB, IB는 매번 바인드 할 필요가 없다.

다음 Draw가 같은 버퍼를 사용한다면 다시 바인드하지 않아도 된다.

519-1954	Colour Pass #1 (1 Targets)
519	vkCmdBeginRenderPass(C=Clear, D=Clear, S=Don't Care)
525	vkCmdDrawIndexed(378, 1)
527	vkCmdDrawIndexed(378, 1)
529	vkCmdDrawIndexed(378, 1)

519-1954	Colour Pass #1 (1 Targets)
519	vkCmdBeginRenderPass(C=Clear, D
525	vkCmdDrawIndexed(378, 1)
527	vkCmdDrawIndexed(378, 1)
529	vkCmdDrawIndexed(378, 1)

EID	Event
520	vkCmdSetViewport
521	vkCmdSetScissor
522	vkCmdBindDescriptorSets
523	vkCmdBindVertexBuffers
524	vkCmdBindIndexBuffer
525	vkCmdDrawIndexed

EID	Event
526	vkCmdBindDescriptorSets
527	vkCmdDrawIndexed

9. VB, IB는 매번 바인드할 필요가 없다.

단, 다음의 조건의 경우에만!

App	Device	FPS
Traha Version: 1.0	SM-G970F Samsung | OS: 9	53FPS 42 - 58 FPS
Traha Version: 1.0	SM-G970F Samsung | OS: 9	55FPS 41 - 59 FPS

Index Buffer

- 같은 IndexBuffer를 사용
- 같은 Offset을 사용
- 같은 CommandBuffer 안에 사용됨
- 같은 Pipeline을 사용
- 같은 Frame에 사용

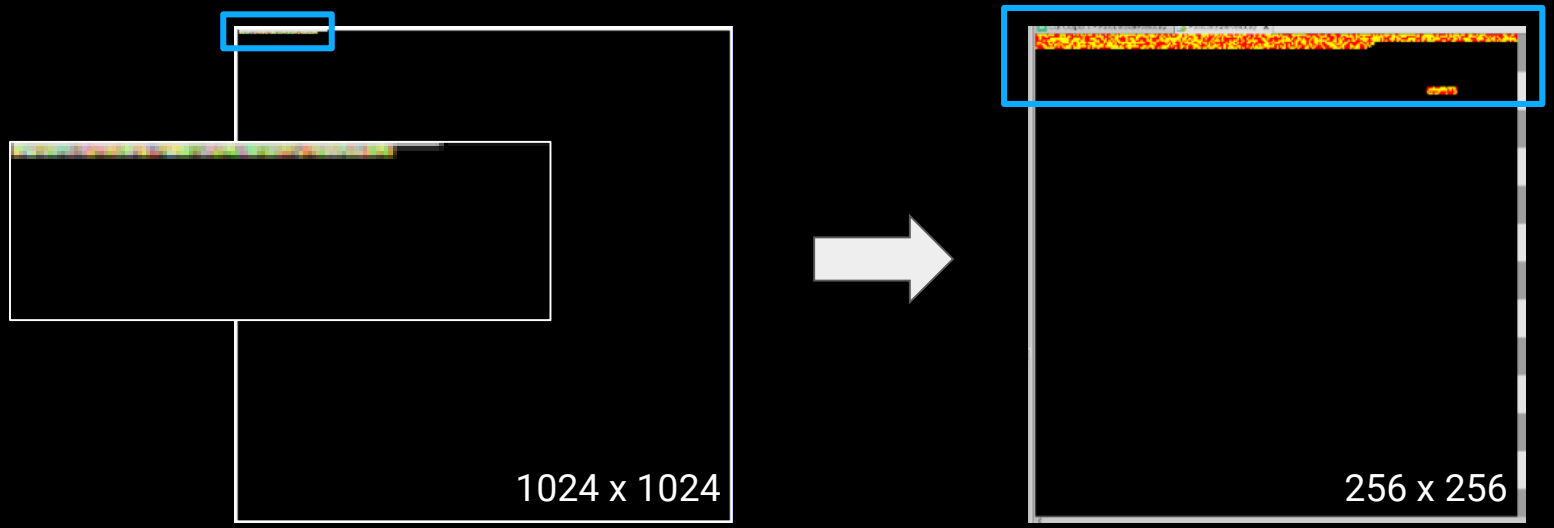
Vertex Buffer

- 같은 VertexBuffer를 사용
- 같은 Binding Index를 사용
- 같은 Offset을 사용
- 같은 CommandBuffer 안에 사용됨
- 같은 Pipeline을 사용
- 같은 Frame에 사용

10. GPU Particle RT 사이즈 줄이기

UE4는 GPU Particle에 1024 x 1024 Simulation RT를 사용

- 실제 컨텐츠에 맞게 줄여도 충분하다!



10. GPU Particle RT 사이즈 줄이기

```
....
/** The texture size allocated for GPU simulation. */
#define USE_SMALL_PARTICLE_TEXTURE_SIZE PLATFORM_ANDROID
#if USE_SMALL_PARTICLE_TEXTURE_SIZE
int32 GParticleSimulationTextureSizeX = 256;
int32 GParticleSimulationTextureSizeY = 256;
#else
int32 GParticleSimulationTextureSizeX = 1024;
int32 GParticleSimulationTextureSizeY = 1024;
#endif

static FAutoConsoleVariableRef CVarParticleSimulationSizeX(
            TEXT("fx.GPUSimulationTextureSizeX"),
            GParticleSimulationTextureSizeX,
....
```

fx.GPUSimulationTextureSizeX / fx.GPUSimulationTextureSizeY


불칸 VULCAN
감사합니다!
Q & A
Contact : gamedev@samsung.com
https://developer.samsung.com/game
TRAHA, © NEXON Korea Corporation & MOAI GAMES All Right Reserved.
#UE4 | @UNREALENGINE
TRAHA
UNREAL ENGINE