언리얼 페스트 2024 서울

바로 적용해보는 언리얼 엔진 디버깅,프로파일링 툴 그리고 팁&트릭

윤준기 대리
에픽게임즈 코리아

아젠다

프레임 드랍

타이밍 인사이트

그래픽스 디버깅

Render doc/Gpu 덤프 뷰어 툴

메모리 누수

메모리 인사이트/ 메모리 릭 리포터 /정적 검증 툴

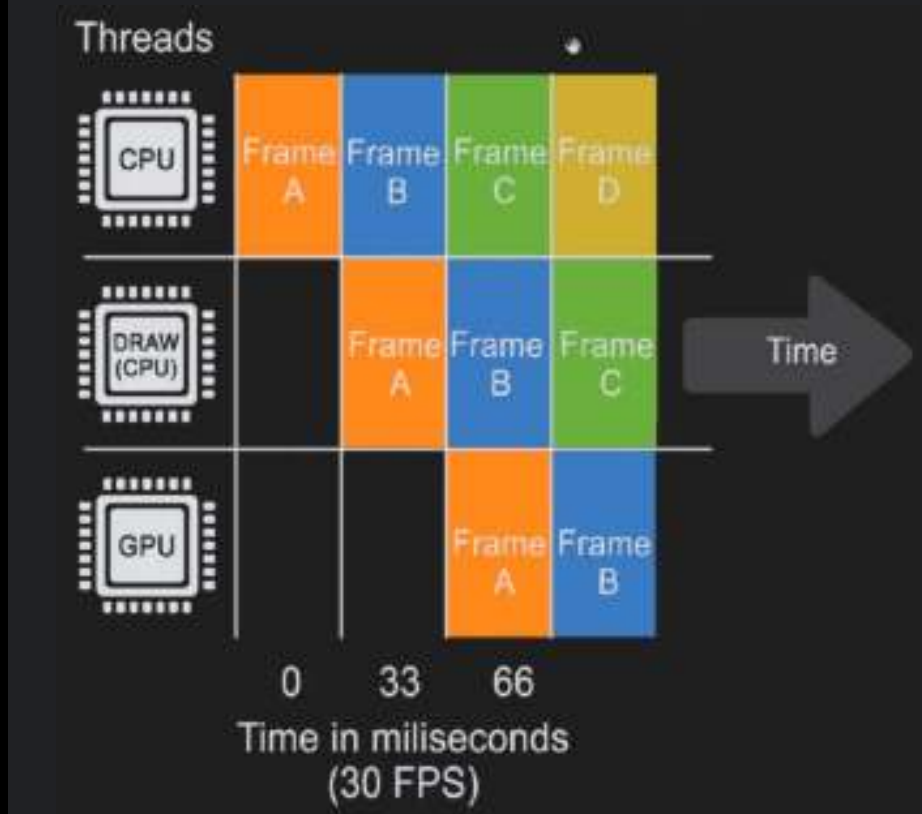
안드로이드 디버깅 팁

프레임 드랍

언리얼 엔진 흐름

30fps : 33.33ms
60fps : 16.67ms

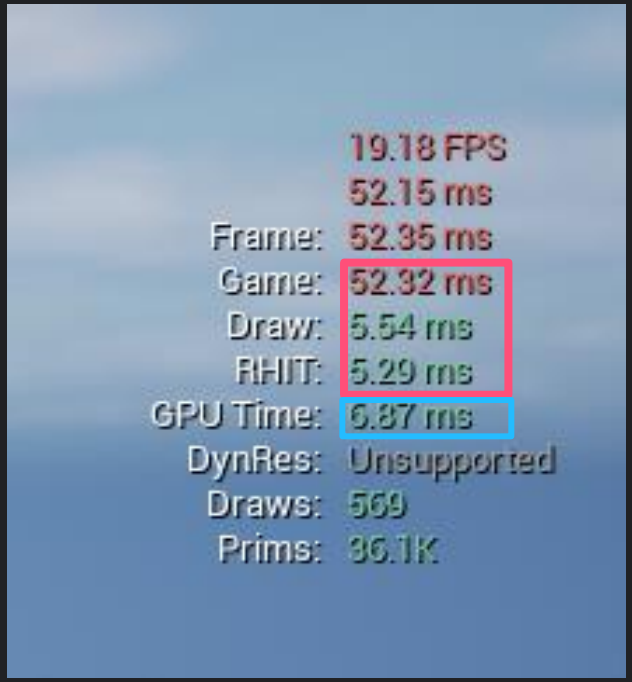
플랫폼/그래픽스 라이브러리에 따라서 RHI Thread 추가



타겟하는 FPS 보다 낮게 나온다?

CPU/GPU
정확히 어디 쪽
부하인지 파악해야 한다.

stat fps/stat unit



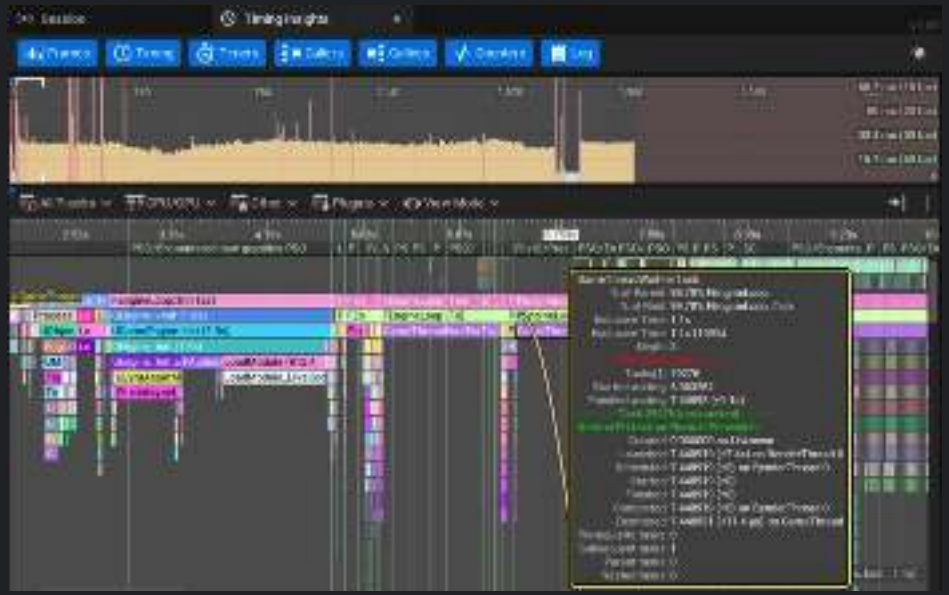
CPU 부하일 경우

어느 지점에서 병목을 일으키는지 확인하는 방법

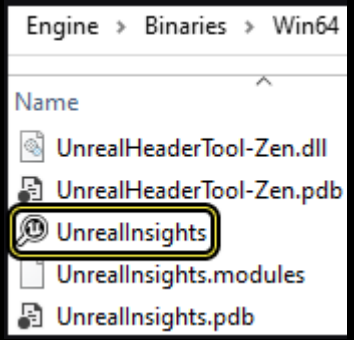
언리얼 인사이트 – 타이밍 인사이트

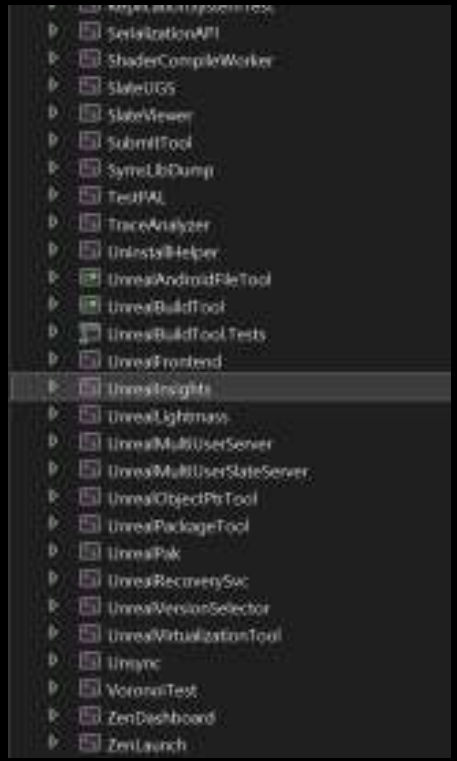
CPU/GPU 트랙을 비롯한 다양한 트랙의 프레임당 퍼포먼스 데이터를 확인할 수 있다

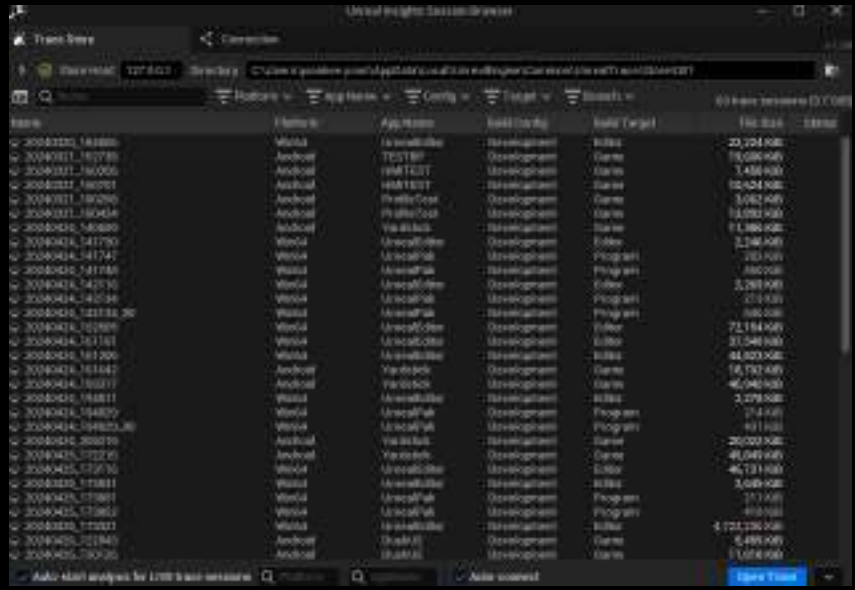
운영체제 에서 제공해 주지 않는 정보는 표시해주지 않는다.



언리얼 인사이트 플랫폼 별 연결 방법 및 사용법







언리얼 인사이트 플랫폼 연결 방법

Window

실행인자

-tracehost=127.0.0.1

Android

adb reverse tcp:1980 tcp:1980

adb shell setprop debug.ue.commandline "-tracehost=127.0.0.1\ -statnamedevents"

IOS

외부 툴을 활용해 실제 app에 접근
이후 uecommandline.txt 에 실행인자 추가

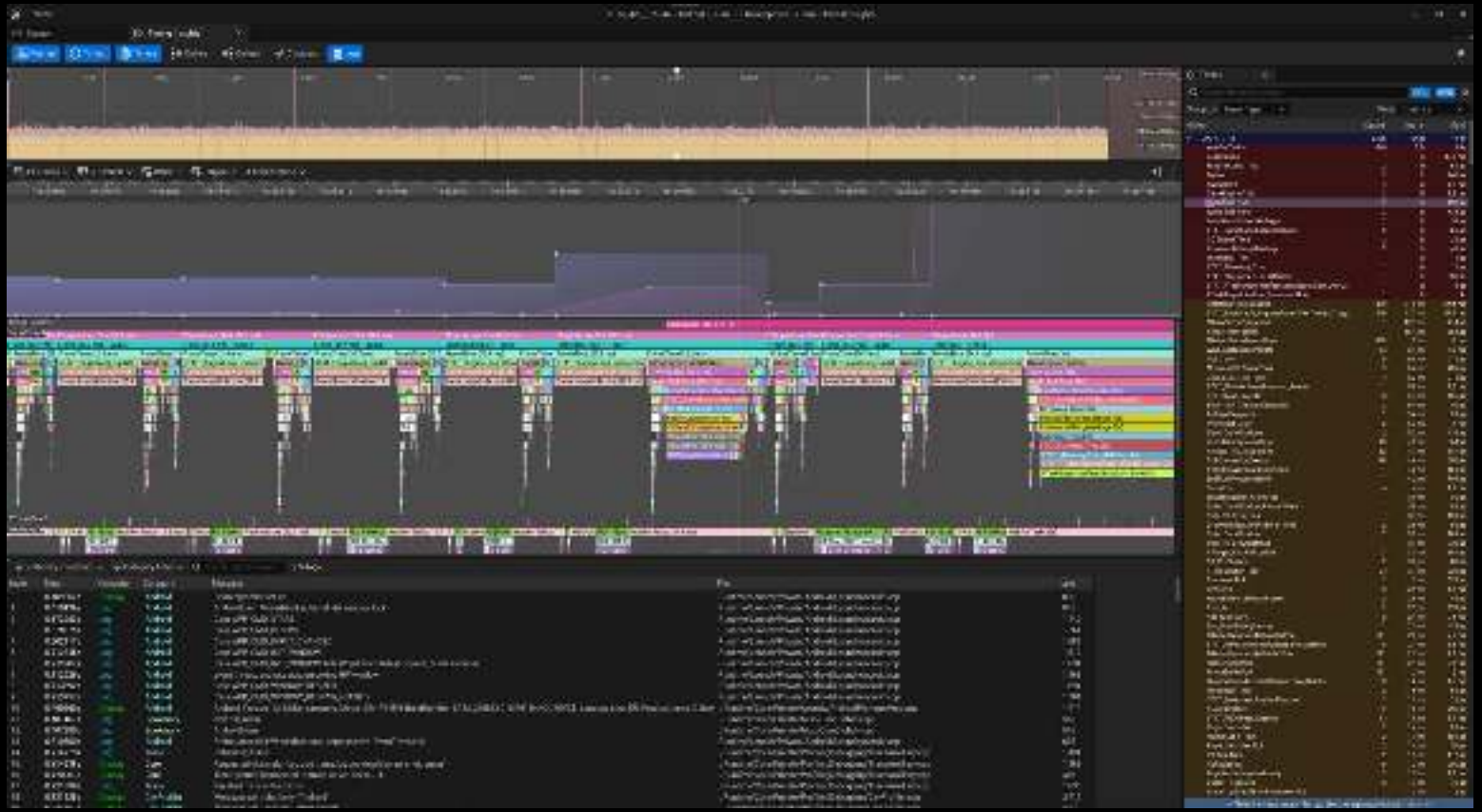
언리얼 인사이트 채널

-trace=default

```
const TCHAR* GDefaultChannels=TEXT("cpu,gpu,frame,log,bookmark,screenshot,region");
```

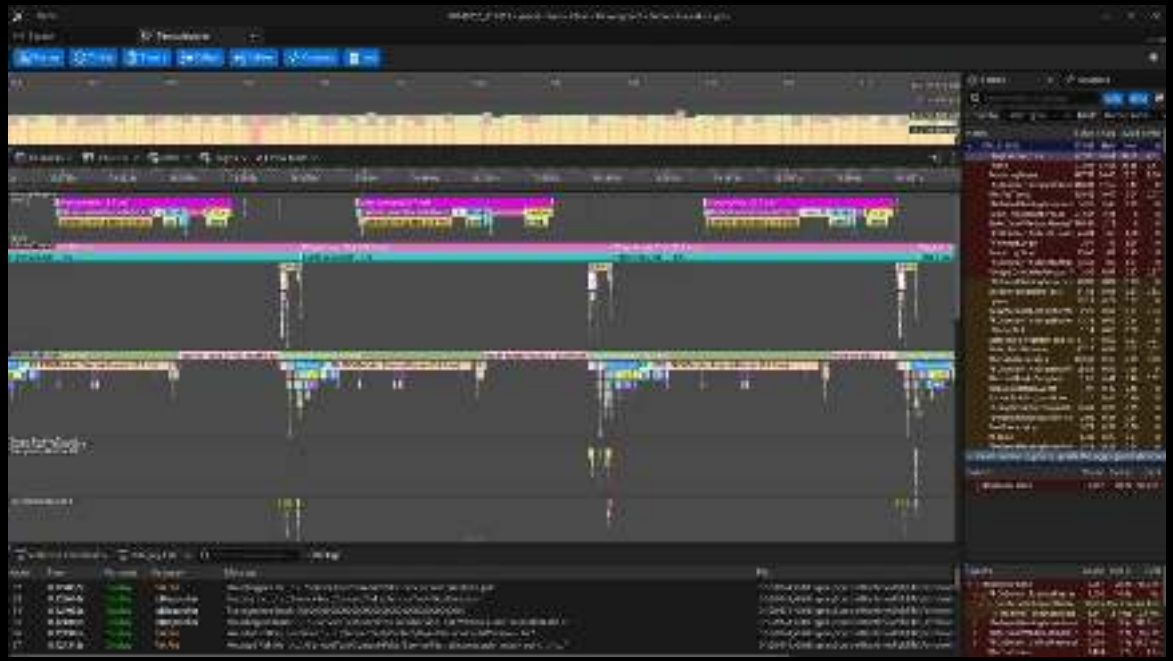
타이밍 인사이트 사용 방법 - default

CPU



타이밍 인사이트 사용 방법 - default

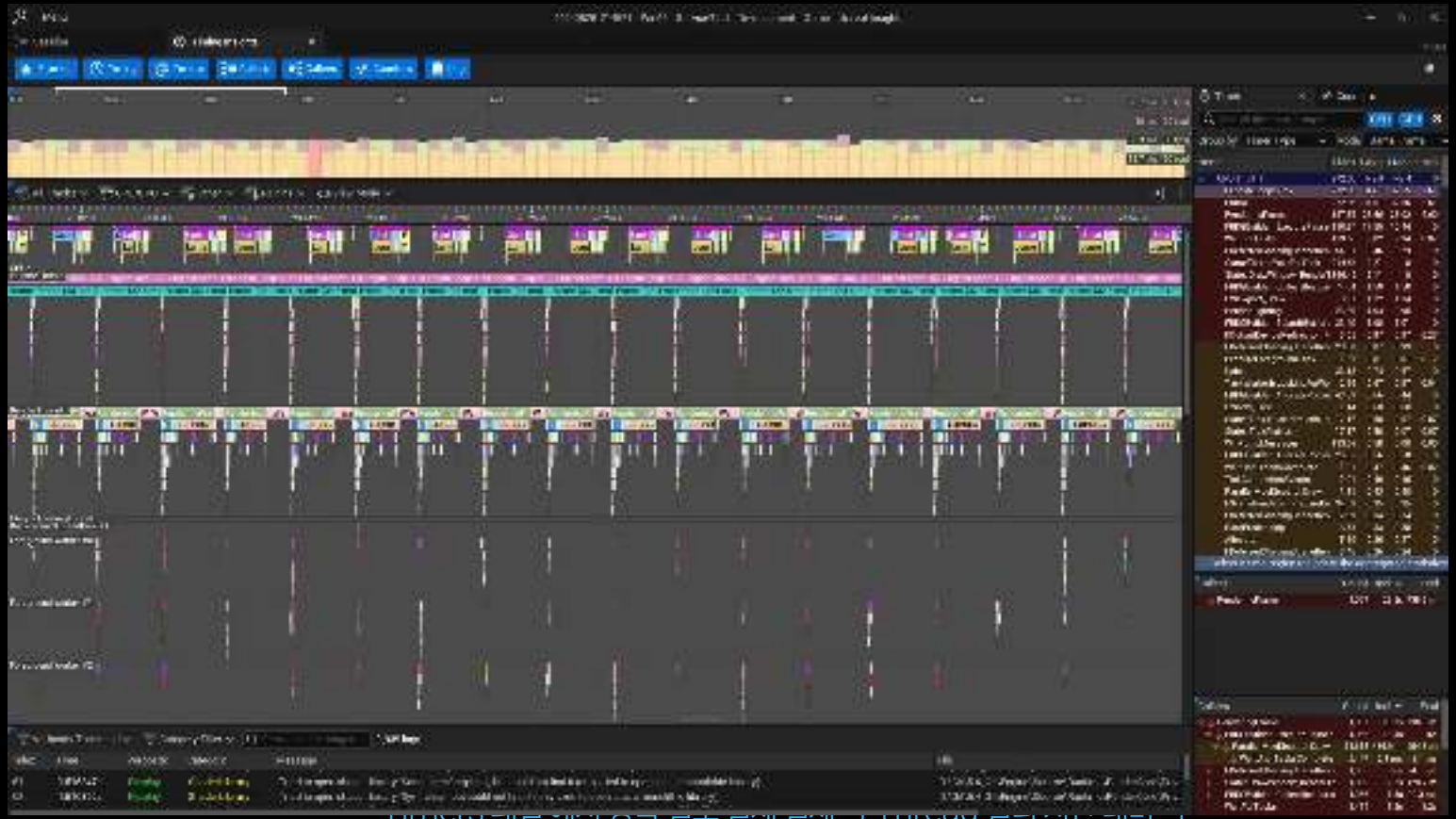
CPU



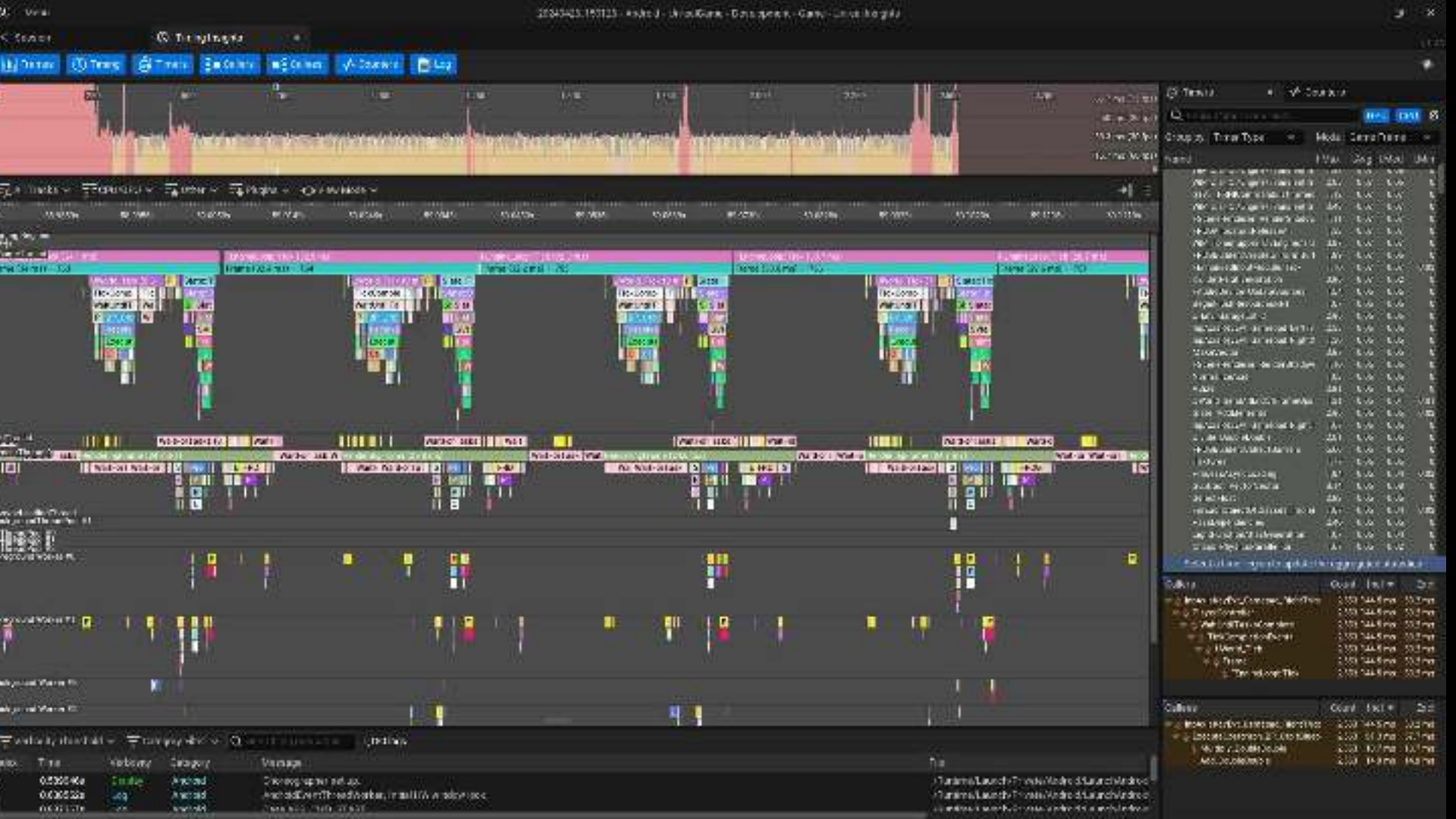
Log를 더블 클릭 시 해당 타이밍 시점으로 갈 수 있고, visual studio 로 해당 로그가 기록된 코드로 바로 이동이 가능

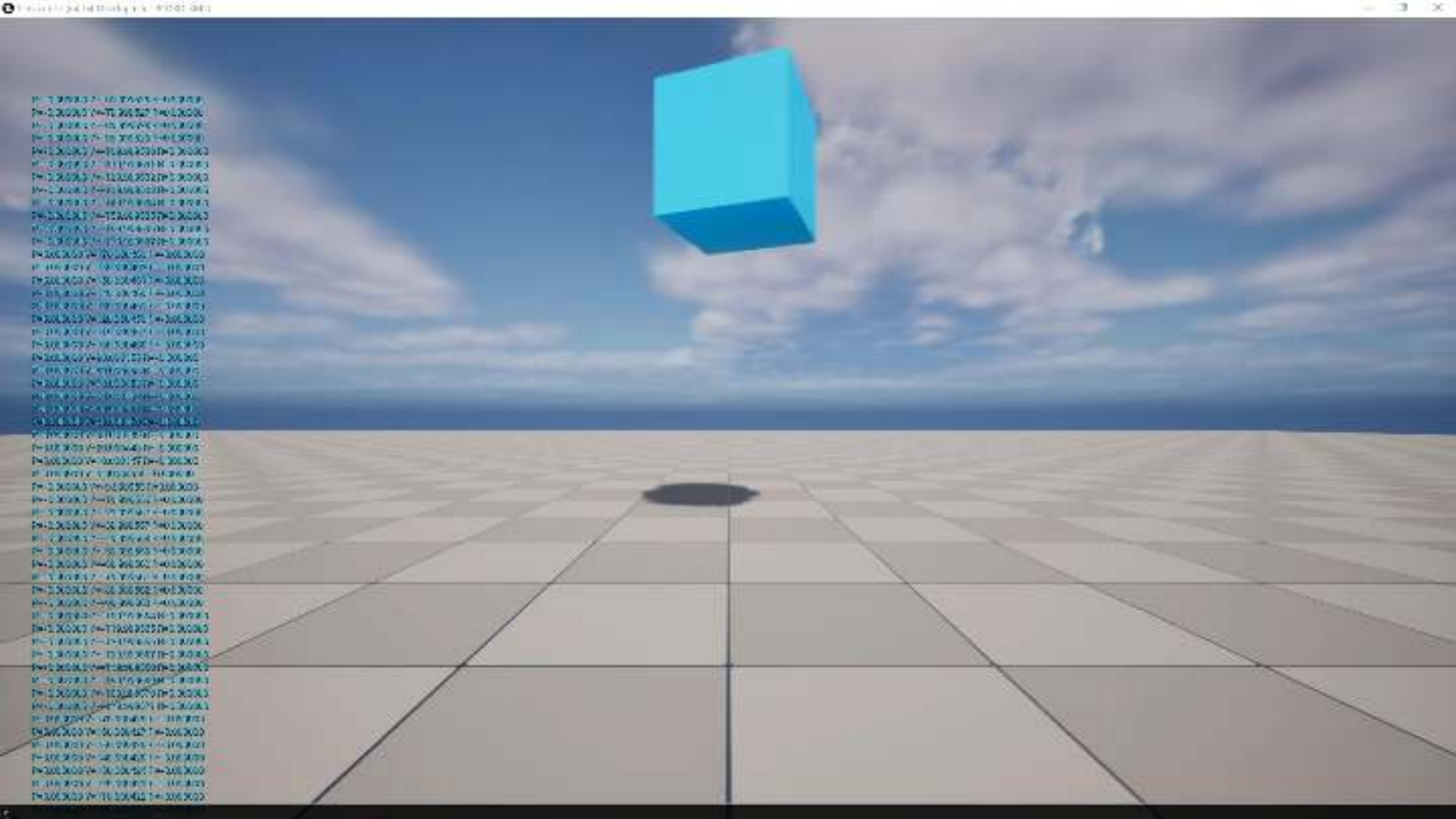
타이밍 인사이트 사용 방법 - default

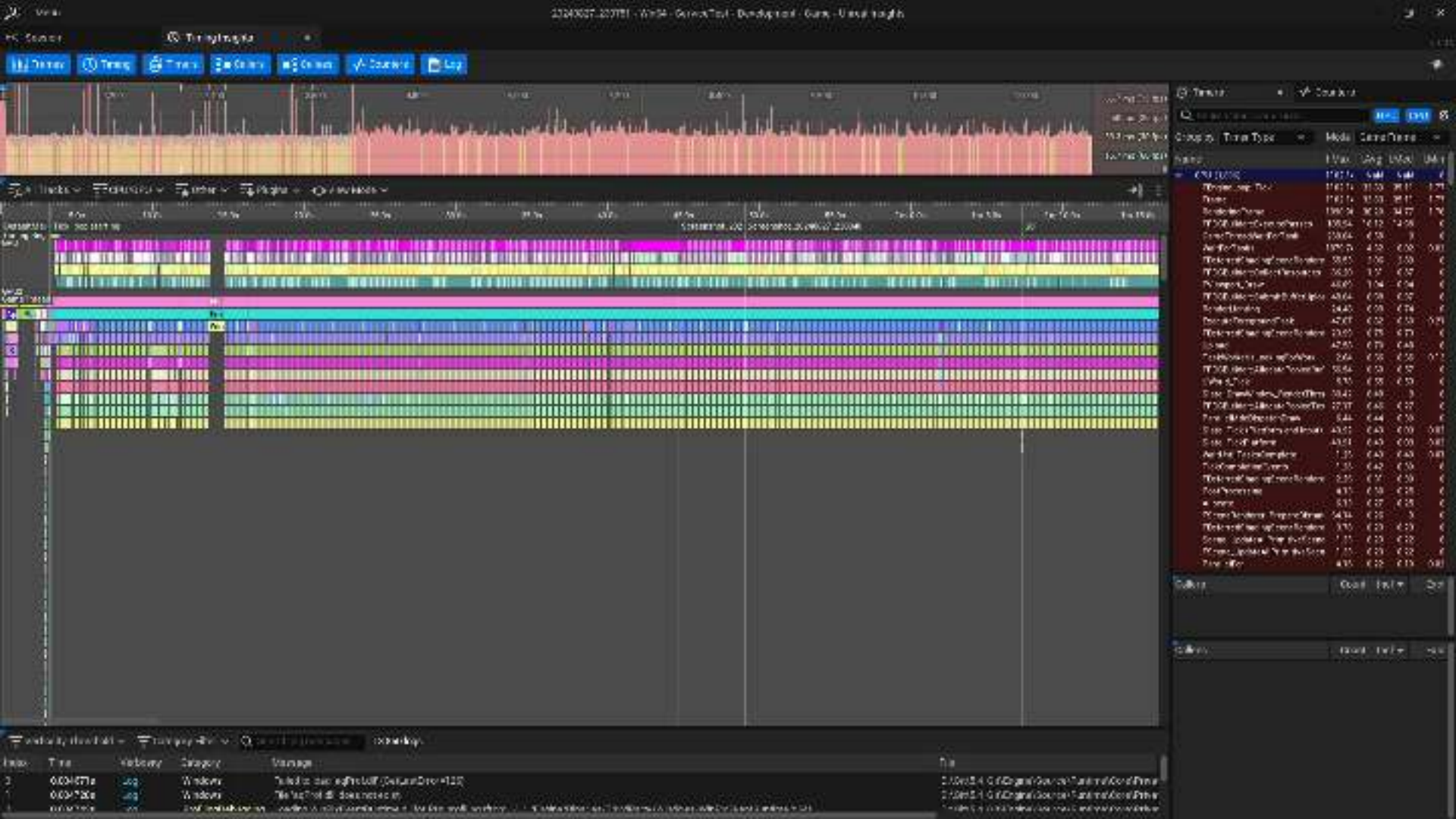
CPU



task 들이 몇 ms 걸렸는지 가시화 해서 볼 수 있음







언리얼 인사이트 채널 어디까지 있나?

CPU

ContextSwitch Channel

-trace=default,ContextSwitch + 관리자 권한 실행

-StatNamedEvents

CPU

적용 전

적용 후

내 코드 타이밍 인사이트에 노출하기

CPU

```
#define SCOPED_NAMED_EVENT(Name, Color)\
    FScopedNamedEventStatic ANONYMOUS_VARIABLE(NamedEvent_##Name##_)(Color, NAMED_EVENT_STR(#Name));\
    TRACE_CPUPROFILER_EVENT_SCOPE(Name);

#define SCOPED_NAMED_EVENT_FSTRING(Text, Color)\
    FScopedNamedEvent ANONYMOUS_VARIABLE(NamedEvent_)(Color, *Text);\
    TRACE_CPUPROFILER_EVENT_SCOPE_TEXT(*Text);

#define SCOPED_NAMED_EVENT_TCHAR(Text, Color)\
    FScopedNamedEvent ANONYMOUS_VARIABLE(NamedEvent_)(Color, Text);\
    TRACE_CPUPROFILER_EVENT_SCOPE_TEXT(Text);

#define SCOPED_NAMED_EVENT_TEXT(Text, Color)\
    FScopedNamedEventStatic ANONYMOUS_VARIABLE(NamedEvent_)(Color, NAMED_EVENT_STR(Text));\
    TRACE_CPUPROFILER_EVENT_SCOPE_STR(Text);

#define SCOPED_NAMED_EVENT_F(Format, Color, ...)\
    FScopedNamedEvent ANONYMOUS_VARIABLE(NamedEvent_)(Color, *FString::Printf(Format, __VA_ARGS__));\
    TRACE_CPUPROFILER_EVENT_SCOPE_TEXT(*FString::Printf(Format, __VA_ARGS__));

#define SCOPED_NAMED_EVENT_TCHAR_CONDITIONAL(Text, Color, bCondition)\
    FScopedNamedEventConditional ANONYMOUS_VARIABLE(NamedEvent_)(Color, Text, (bCondition));\
    TRACE_CPUPROFILER_EVENT_SCOPE_TEXT_CONDITIONAL(Text, (bCondition));
```

상황에 맞게 정의된 매크로를 통해서 자유롭게 사용 가능

타이밍 인사이트 내 코드에 적용하기

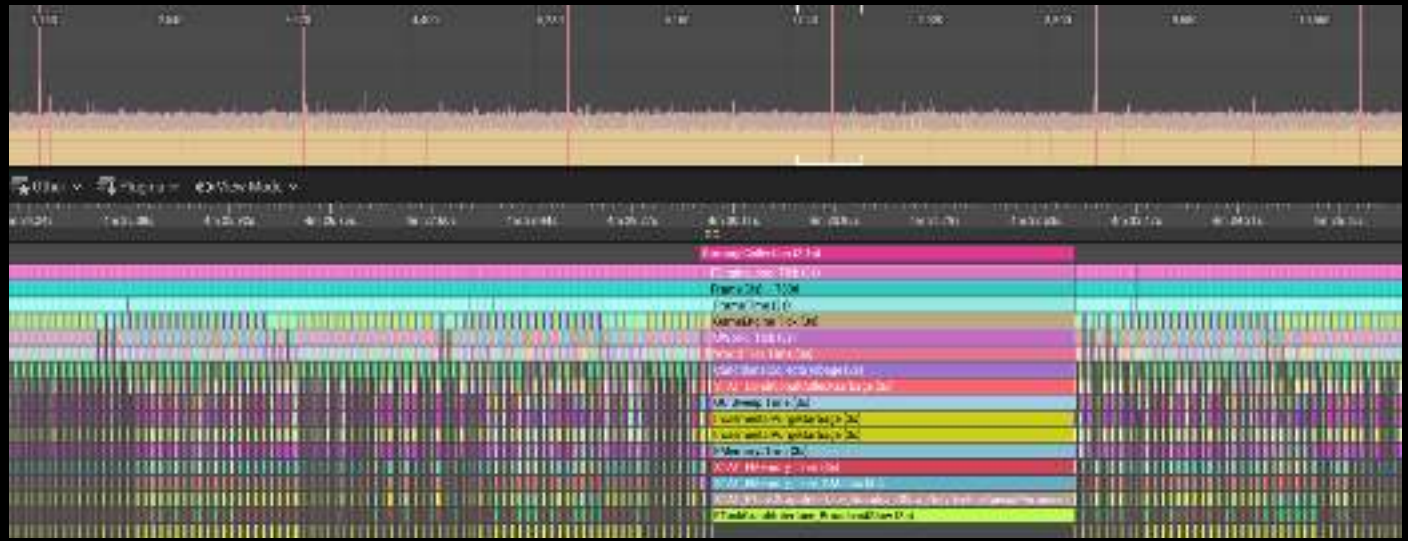
CPU

```
Void FDefaultSpectatorScreenController::RenderSpectatorScreen_RenderThread(FRHICommandListImmediate& RHICmdList,
FRHITexture2D* BackBuffer, FTexture2DRHIRef SrcTexture, FTexture2DRHIRef LayersTexture, FVector2D WindowSize)
{
    SCOPED_NAMED_EVENT_TEXT("RenderSocialScreen_RenderThread()", FColor::Magenta);

    check(IsInRenderingThread());

    StereoLayersTexture = LayersTexture;

    if (SpectatorScreenDelegate_RenderThread.IsBound())
    {
```

지역 변수 생성 시점 부터 소멸 시점 까지 Scope하는 구조

주로 나오는 병목 – 가비지 컬렉션

CPU



언리얼 엔진 GC 에 의한 히치

오브젝트가 많아 도달 가능성 분석(어떤 오브젝트의 메모리를 회수 할 지 결정) 에 의한 게임 스레드 부하

실험기능 – 점진적 가비지 컬렉션

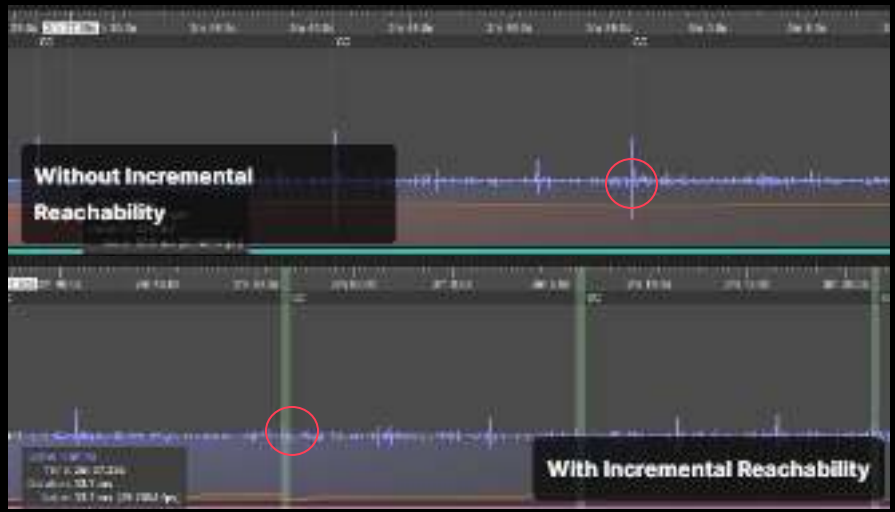
CPU

```
[ConsoleVariables]
gc.AllowIncrementalReachability=1;
gc.AllowIncrementalGather=1;
gc.IncrementalReachabilityTimeLimit=0.002;
```

어떤 오브젝트의 메모리를 회수 할 수 있을지 결정하는 과정에서 런타임에 히치가 발생 할 수 있음

해당 프로세스를 단일 프레임 내에서 완료하는게 아닌 프레임별로 제한 시간을 사용하여 여러 프레임에 걸쳐 분산시키는 기능

프로덕션 금지!



컴파일러 단 최적화 LTO

LTO 란?

Link Time Optimization

헤더 파일과 cpp가 분리되어 있는 인라인 함수에 대해서도 링크 타임에 최적화를 해주어 인라인 함수로 쓸 수 있음

```
int main
{
  IsAvailable();
}
```

```
//헤더파일
inline int square(int a)
{
 return a*a;
}
```

인라인 최적화 O

```
//헤더파일
int square(int a);

//cpp 파일
int square(int a)
{
 return a*a;
}
```

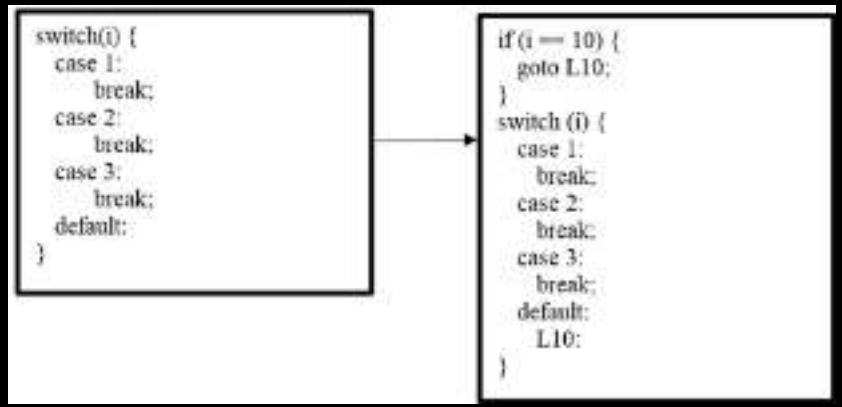
인라인 X, LTO 가능

컴파일러 단 최적화 PGO

CPU

PGO 란?

런타임에 profiledata를 뽑으면
실제로 많이 호출되는 부분을 추적하여 profiledata를 기반으로
빌드 시 컴파일러가 최적화를 해준다.



https://devblogs.microsoft.com/cppblog/profile-guided-optimization-pgo-under-the-hood/

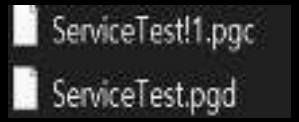
언리얼 엔진에서 PGO/LTO 적용

CPU

PGO

1. PGOProfile 추가후 빌드
빌드 시 **-PGOProfile 인자 추가 혹은** [Target.cs] bPGOProfile = true;

2. 시나리오 대로 플레이 및 ProfileData 뽑기



3. Platform/플랫폼/Build/PGO/.. ProfileData 이동 후
빌드 시 **-PGOOptimize 인자 추가 혹은** [Target.cs] bPGOOptimize = true;

LTO

[Target.cs]
bAllowLTCG = true; or -LCTG 실행인자 추가

시그니피컨스 매니저

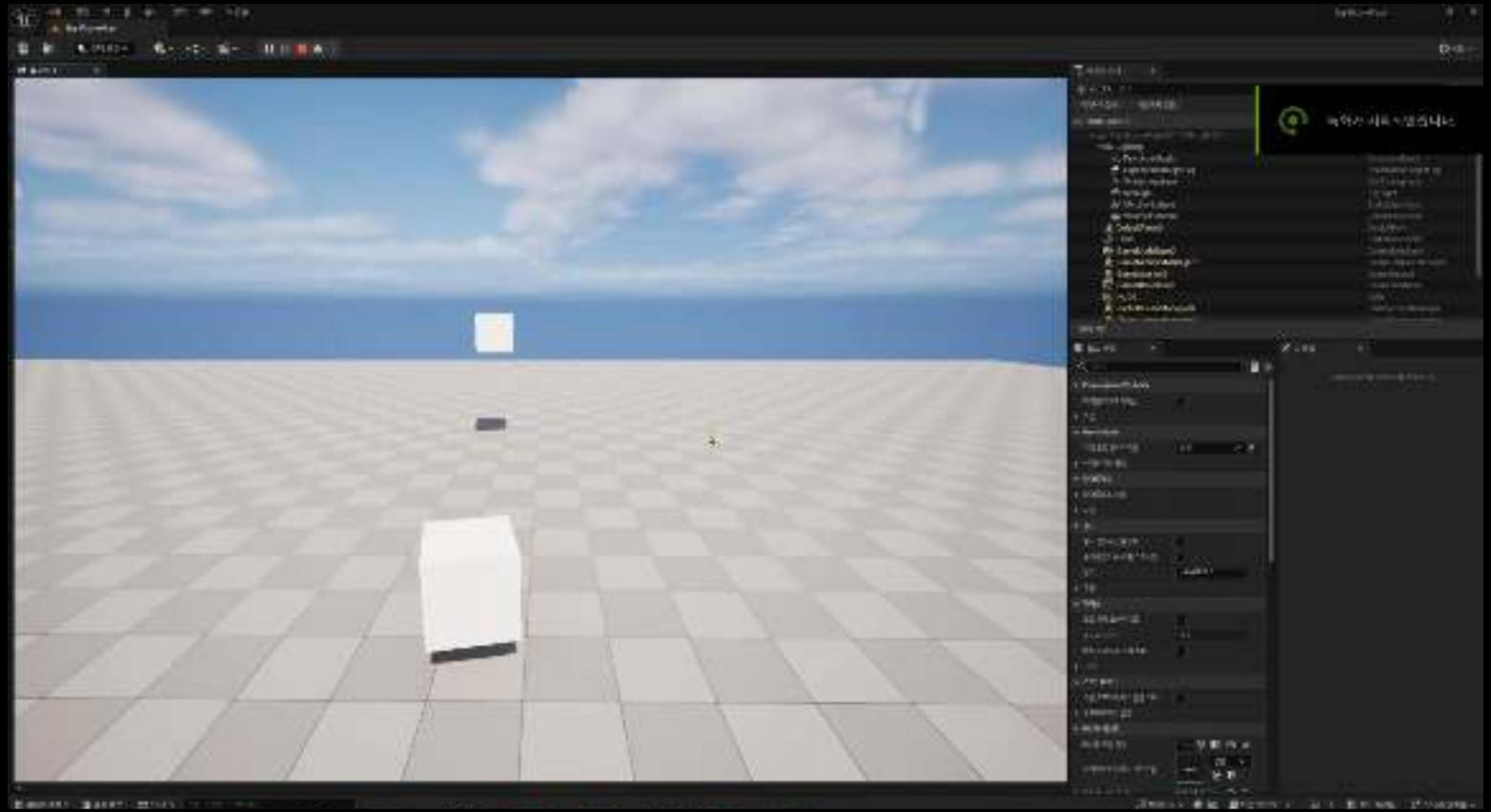
CPU

오브젝트 사이 연관성을 평가하는 인터페이스 제공

실제 평가 기능이 있는 게 아닌 사용자가 override를 통해서 정의 할 수 있는 시스템만 제공 됨

시그니피컨스 매니저 예제 카메라와의 거리에 따른 Interval 조절

CPU



시그니피컨스 매니저 사용법

CPU

```
class SIGNIFICANCEMANAGER_API USignificanceManager : public UObject
..
    typedef TFunction<float(FManagedObjectInfo*, const FTransform&)> FManagedObjectSignificanceFunction;
    typedef TFunction<void(FManagedObjectInfo*, float, float, bool)> FManagedObjectPostSignificanceFunction;
..
virtual void RegisterObject(UObject* Object, FName Tag, FManagedObjectSignificanceFunction
SignificanceFunction, EPostSignificanceType InPostSignificanceType = EPostSignificanceType::None,
FManagedObjectPostSignificanceFunction InPostSignificanceFunction = nullptr);
..
virtual void Update(TArrayView<const FTransform> Viewpoints);
```

중요도 평가 함수 등록

중요도 설정 이후 동작 함수 등록

USignificanceManager 핵심 기능

시그니피컨스 매니저 사용법

CPU

```
void ATickingActor::BeginPlay()
{
    Super::BeginPlay();

    USignificanceManager * SignificanceManager = USignificanceManager::Get<USignificanceManager>(GetWorld());

    if (nullptr != SignificanceManager)
    {
        USignificanceManager::FManagedObjectSignificanceFunction Significance =
            std::bind(&ATickingActor::SignificaneFunction, this, std::placeholders::_1, std::placeholders::_2);

        auto PostSignificance = [&](USignificanceManager::FManagedObjectInfo* ObjectInfo, float OldSignificane, float Significane, bool bFinal)
            {
                PostSignificaneFunction(ObjectInfo, OldSignificane, Significane, bFinal);
            };

        SignificanceManager->RegisterObject(this, TEXT("TickingTestActor"), Significance,USignificanceManager::EPostSignificanceType::Sequential,
PostSignificance);
    }
}
```

함수 객체 생성 및 등록

시그니피컨스 매니저 사용법

CPU

매프레임
카메라와의 거리에 따른
Significance Update를 위해 Tick override

```
void USignificaneGameViewportClient::Tick(float DeltaTime)
{
    Super::Tick(DeltaTime);

    const UWorld* _world = GetWorld();

    if (nullptr != _world)
    {
        USignificanceManager* _SignificanceManager
        = USignificanceManager::Get<USignificanceManager>(_world);

        if (nullptr != _SignificanceManager)
        {
            if (APawn* PlayerPawn = UGameplayStatics::GetPlayerPawn(_world, 0))
            {
                TArray<FTransform> TransformArray;
                TransformArray.Add(PlayerPawn->GetTransform());
                _SignificanceManager->Update(TArrayView<FTransform>(TransformArray));
            }
        }
    }
}
```

시그니피컨스 매니저 사용법

CPU

```
float ATickingActor::SignificaneFunction(USignificanceManager::FManagedObjectInfo* ObjectInfo, const FTransform& ViewPoint)
{
    if (TEXT("TickingTestActor") == ObjectInfo->GetTag())
    {
        ATickingActor* tickingActor = CastChecked<ATickingActor>(ObjectInfo->GetObject());
        const float Distance = (tickingActor->GetActorLocation() - ViewPoint.GetLocation()).Size();

        if (Distance > 1000.f)
        {
            return 0.f;
        }
        else
        {
            return 1.f;
        }
    }

    return 0.0f;
}
```

거리에 따른 중요도 설정

시그니피컨스 매니저 사용법

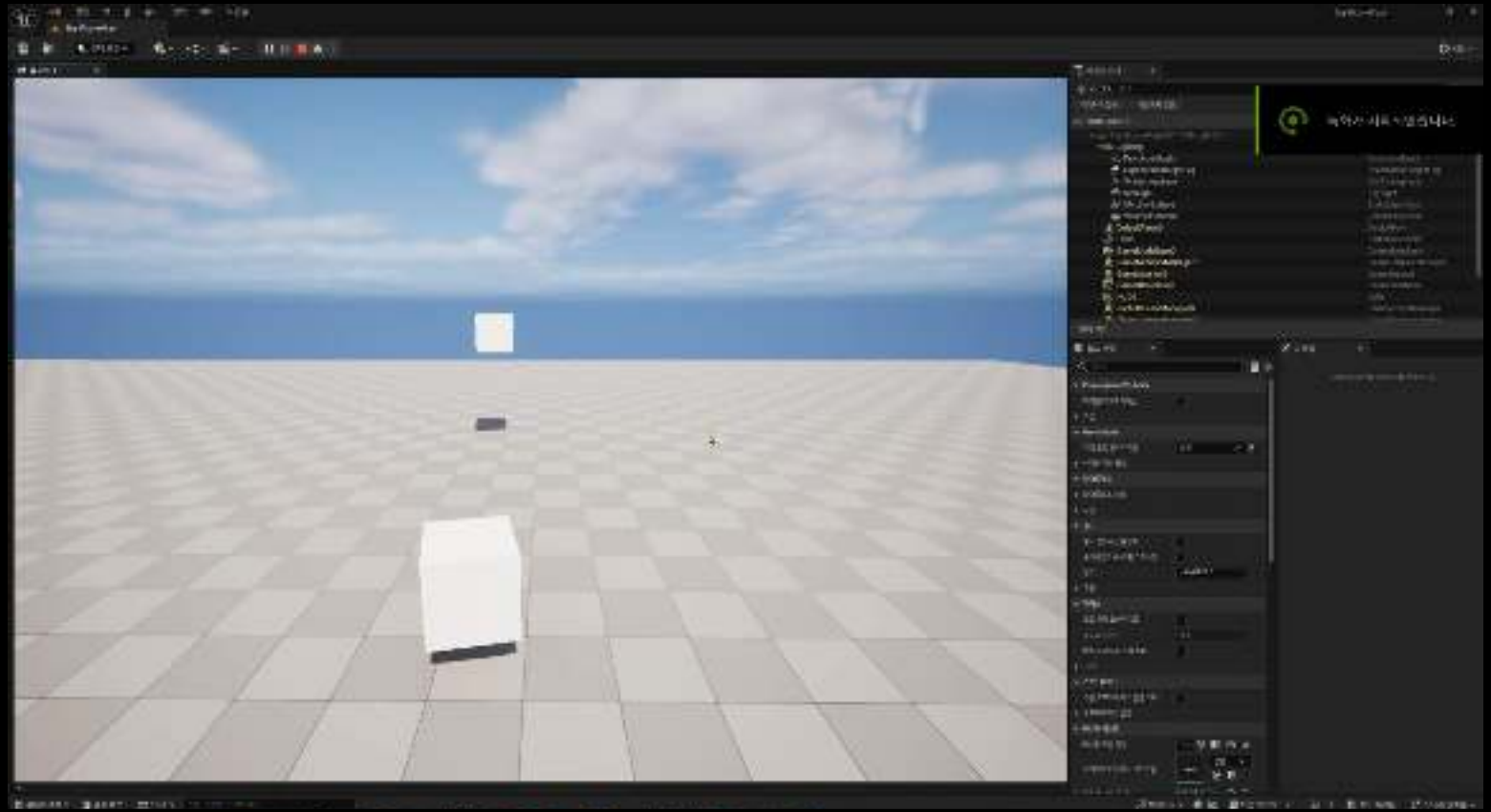
CPU

```
void ATickingActor::PostSignificaneFunction(USignificanceManager::FManagedObjectInfo* ObjectInfo, float
OldSignificane, float Significane, bool bFinal)
{

    if (TEXT("TickingTestActor") == ObjectInfo->GetTag())
    {
        ATickingActor* tickingActor = CastChecked<ATickingActor>(ObjectInfo->GetObject());

        if (Significane == 0.f)
        {
            tickingActor->SetActorTickInterval(0.5f);
        }
        else
        {
            tickingActor->SetActorTickInterval(0.f);
        }
    }
}
```

중요도에 따른 Tick interval 셋팅

시그니피컨스 매니저 예제 카메라와의 거리에 따른 Interval 조절

CPU



GPU 부하일 경우

어느 지점에서 병목을 일으키는지 확인하는 방법

타이밍 인사이트 GPU 채널

GPU 채널 추가하기

-trace=default

타이밍 인사이트를 통해서 실제로 어떤 작업이 얼마나 걸렸는지 추적이 가능하다.

Android IOS
미지원

Profile GPU

단일 프레임의 GPU 소요 시간을 간단하게 캡처 하는 방법

ProfileGPU 측정

ctrl + shift + , or **ProfileGPU command 입력**

Android IOS
미지원

다 안되면 어떻게 해야 하나요?

GPU

```
Profiling the next GPU frame
LogRHI: Perf marker hierarchy, total GPU time 9.40ms
LogRHI: 100.0% 9.40ms  FRAME 596 draws 10905 prims 14046 verts 68 dispatches
LogRHI:  0.0% 0.00ms  BufferPoolCopyOps
LogRHI:  0.0% 0.00ms  TexturePoolCopyOps
LogRHI:  0.0% 0.00ms  WorldTick
LogRHI: 90.1% 8.47ms  FRDGBuilder::Execute 119 draws 6546 prims 5652 verts 62 dispatches
LogRHI:   0.1% 0.01ms  ClearGPUMessageBuffer 1 dispatch 1 groups
LogRHI:   0.0% 0.00ms  UpdateAllPrimitiveSceneInfos 2 dispatches
LogRHI:     0.0% 0.00ms  ScatterUpload[0] (Resource: SceneCulling.Items, Offset: 0, GroupSize
LogRHI:     0.0% 0.00ms  ScatterUpload[0] (Resource: SceneCulling.ItemChunks, Offset: 0
LogRHI:   0.0% 0.00ms  ShaderPrint::UploadParameters 1 dispatch 1 groups
LogRHI:   1.4% 0.13ms  UpdateDistanceFieldAtlas 1 draw 1 prims 0 verts 2 dispatches
.
.
.
LogRHI:     0.0% 0.00ms  ExtractUniformBuffer
LogRHI:   0.0% 0.00ms  EnqueueCopy(GPUMessageManager.MessageBuffer)
LogRHI:   0.0% 0.00ms  AccessModePass[Graphics] (Textures: 15, Buffers: 5)
LogRHI:   3.5% 0.26ms  Other Children
LogRHI:  0.1% 0.01ms  FRDGBuilder::Execute 6 draws 9 prims 18 verts
LogRHI:   0.0% 0.00ms  CanvasBatchedElements 1 draw 1 prims 2 verts
LogRHI:   0.0% 0.00ms  CanvasBatchedElements 1 draw 2 prims 4 verts
LogRHI:   0.0% 0.00ms  CanvasBatchedElements 1 draw 1 prims 2 verts
LogRHI:   0.0% 0.00ms  CanvasBatchedElements 1 draw 2 prims 4 verts
LogRHI:   0.0% 0.00ms  CanvasBatchedElements 1 draw 1 prims 2 verts
LogRHI:   0.0% 0.00ms  CanvasBatchedElements 1 draw 2 prims 4 verts
LogRHI:  5.4% 0.39ms  SlateUI Title = ServiceTest - 언리얼 에디터 380 draws 0 prims 0 verts
LogRHI:  5.8% 0.42ms  SlateUI Title = <none> 7 draws 0 prims 0 verts
LogRHI:  0.0% 0.00ms  FRDGBuilder::Execute
LogRHI:   7.1% 0.52ms  Other Children
LogRHI: Total Nodes 403 Draws 540
```

플랫폼 별 GPU 성능 측정 방법 정리

Android

퀄컴 – Snapdragon Profiler
Mali/Exynos – Arm PerforamanceStudio

IOS

Mac Instruments Tool

	Android	IOS
총 GPU 시간	X	O
각 RenderPass 시간	X	X

플랫폼에 따른 GPU 프로파일링 지원

그래픽스 디버깅

- RenderDoc
- GpuDumpViewer Tool (UE 인하우스 그래픽스 디버거)

RenderDoc

싱글 프레임 캡처를 수행하고 조사할 수 있는 독립형 오픈 소스 그래픽 디버거

지원 플랫폼

Windows/Linux/Android/NintendoSwtich

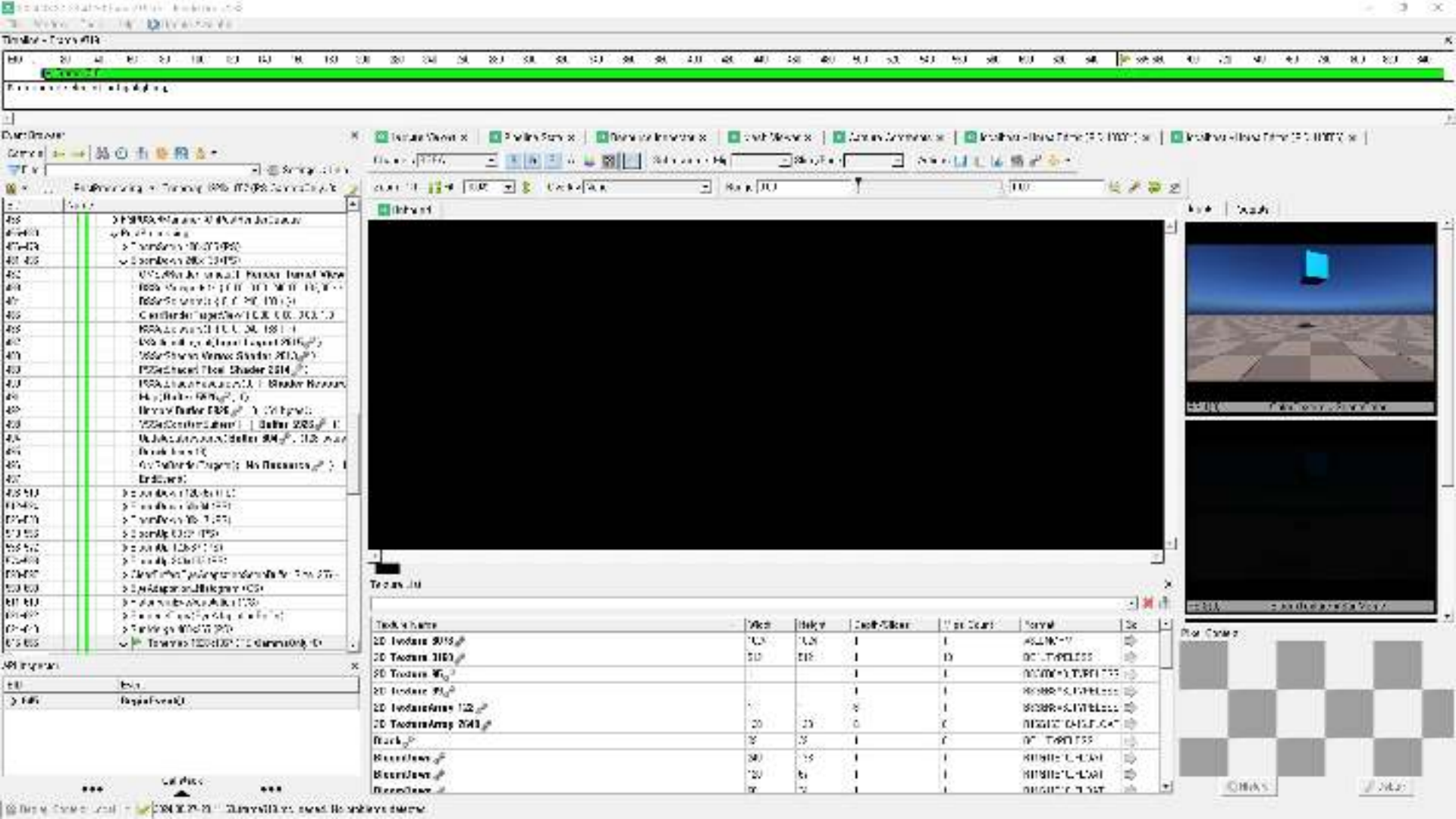
그래픽스 API

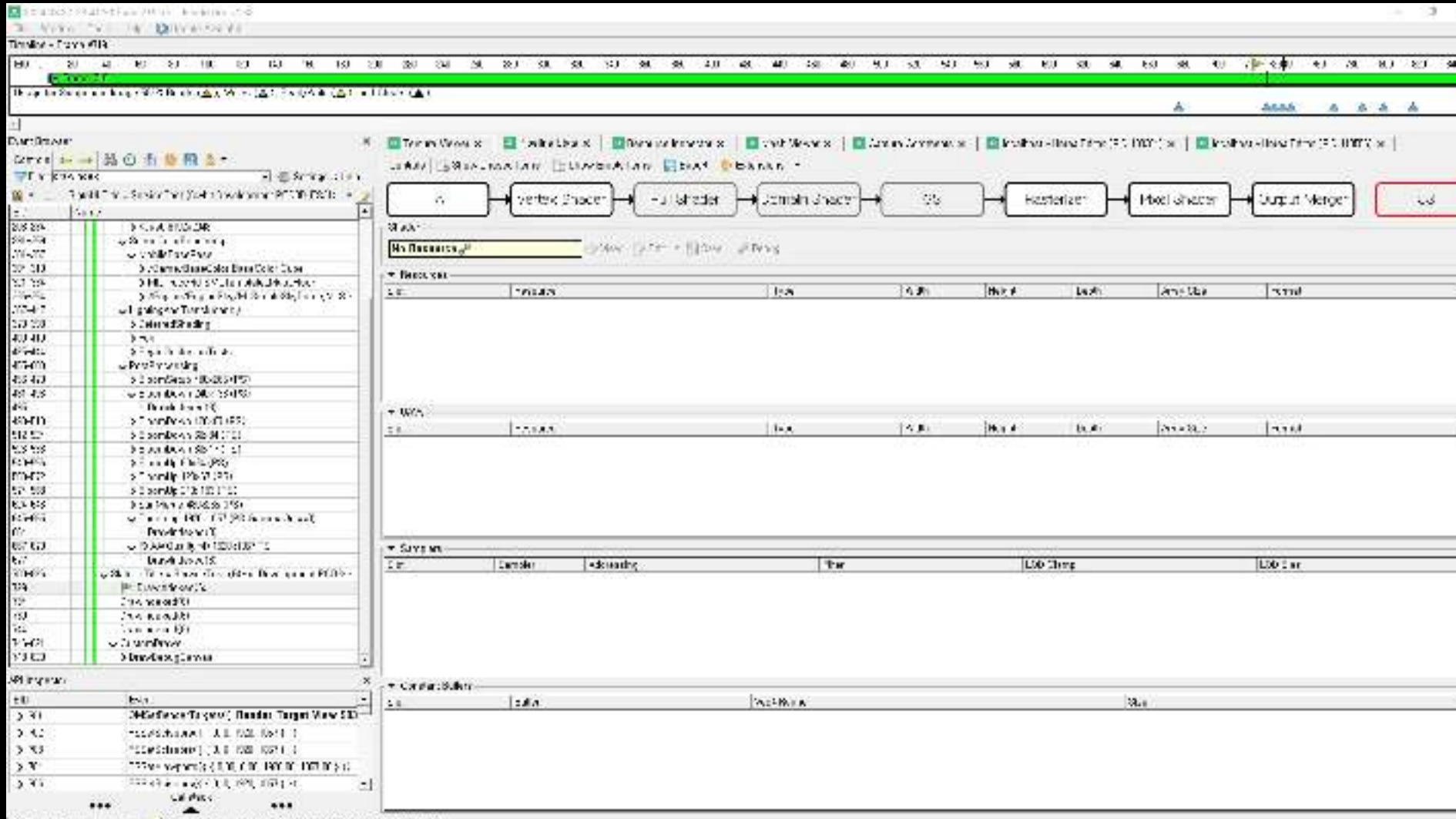
Vulkan/D3D11/D3D12/OpenGL 3.2+
/OpenGL ES 2.0 – 3.2

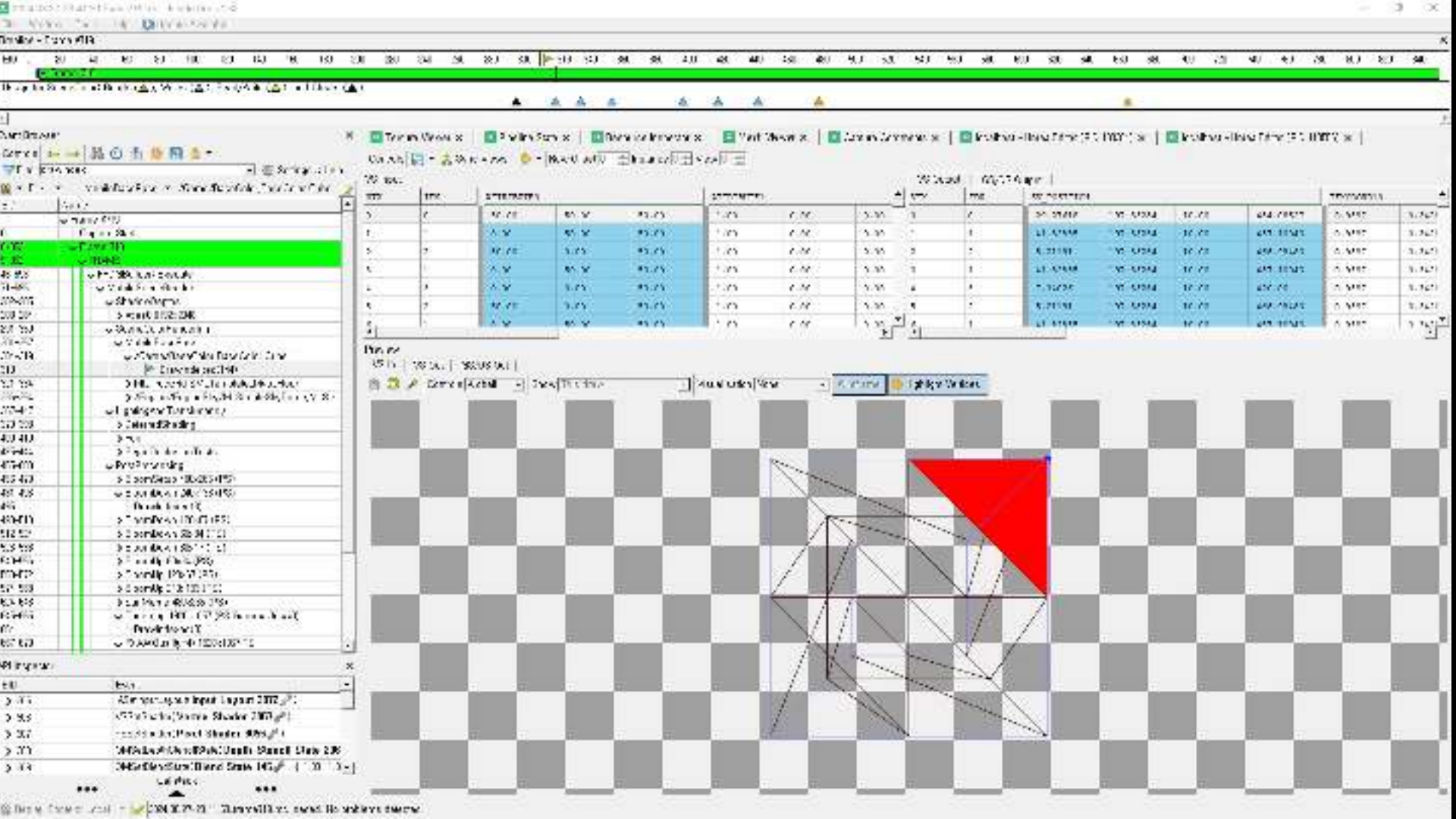
RenderDoc

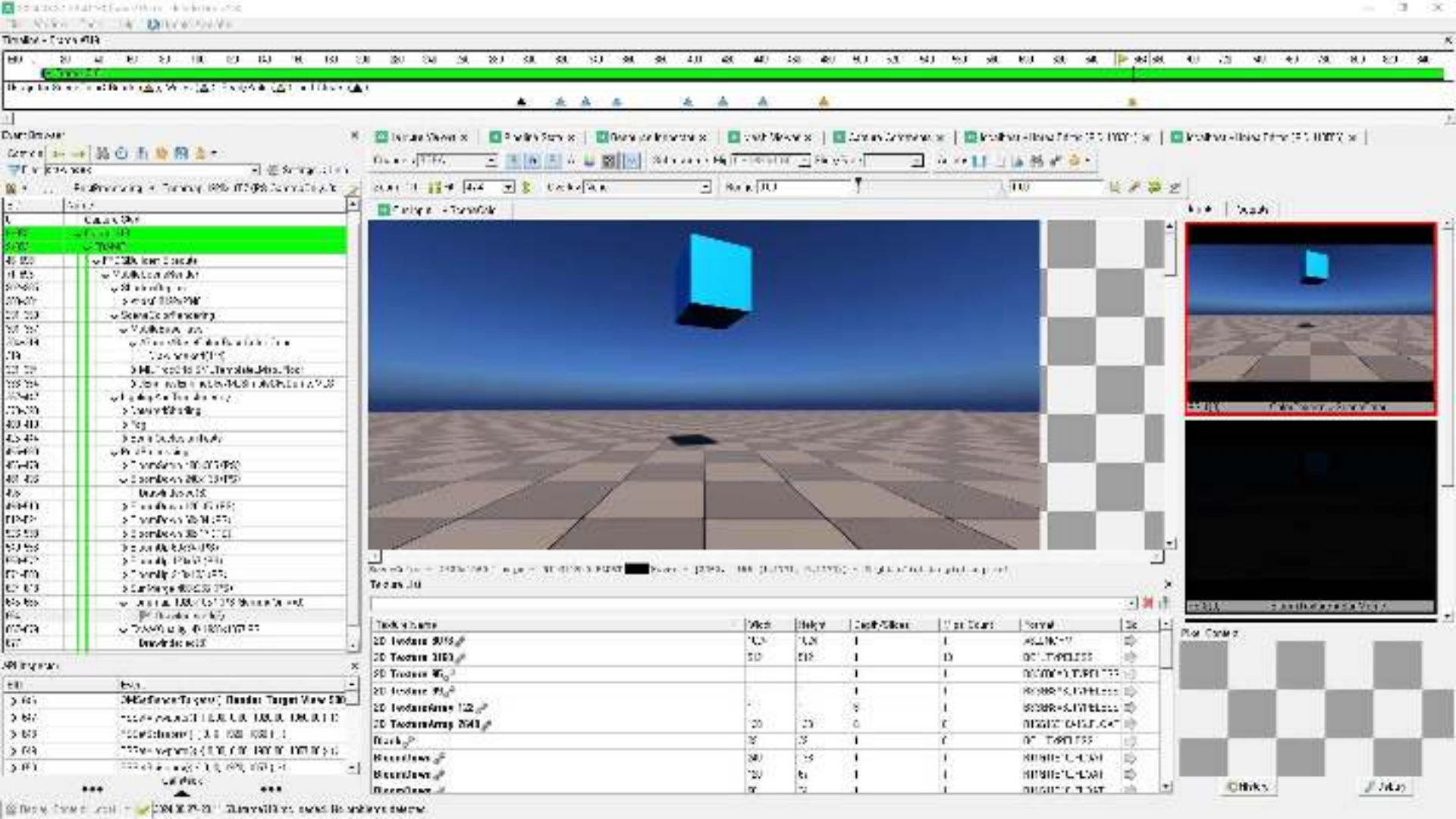
디버깅을 위한 사전 구성

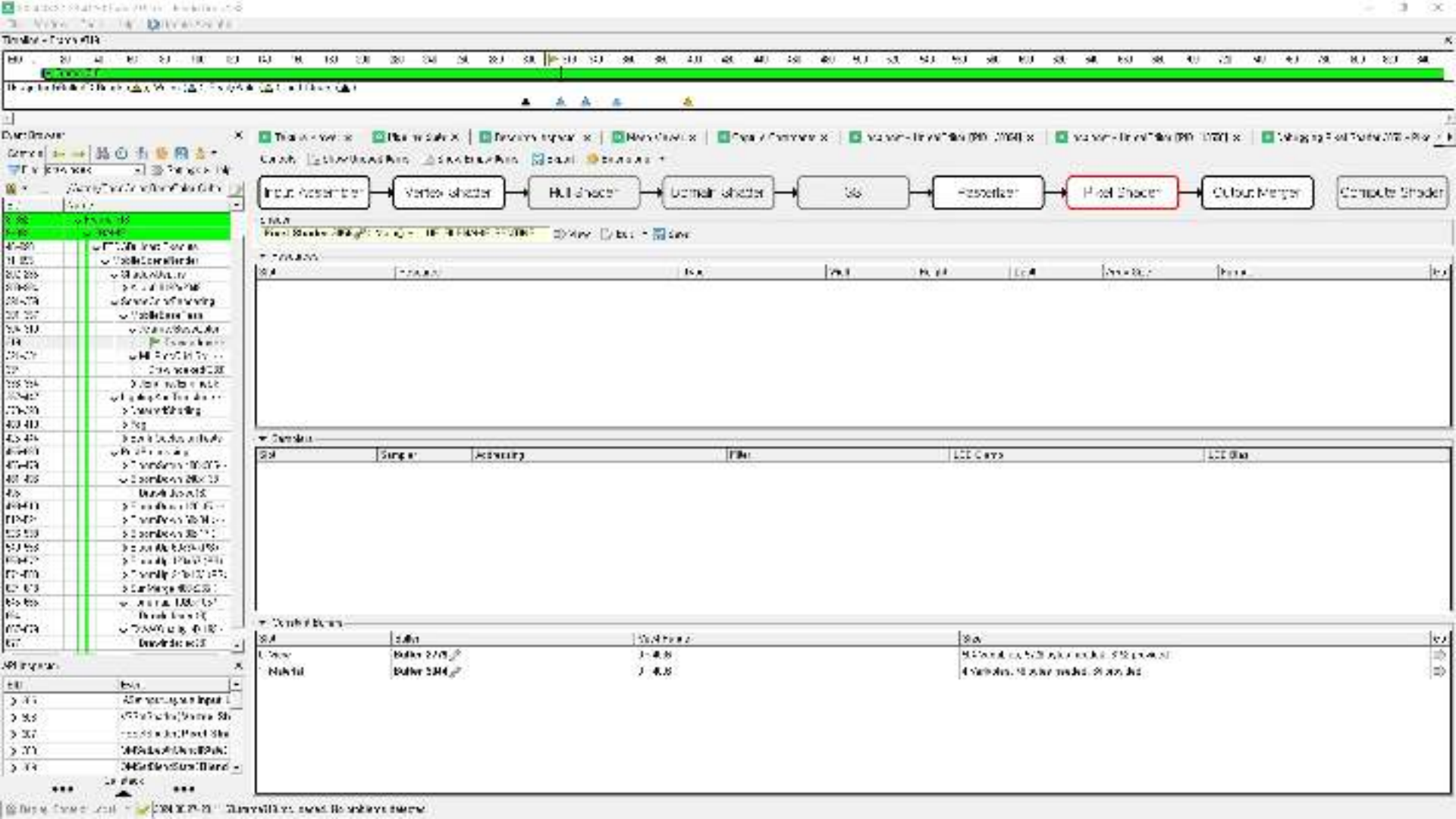
```
[Platform]Engine.ini

[ShaderCompiler]
r.Shaders.Optimize=0
r.Shaders.Symbols=1
r.Shaders.SkipCompression=1
r.ShaderDevelopmentMode=1
```

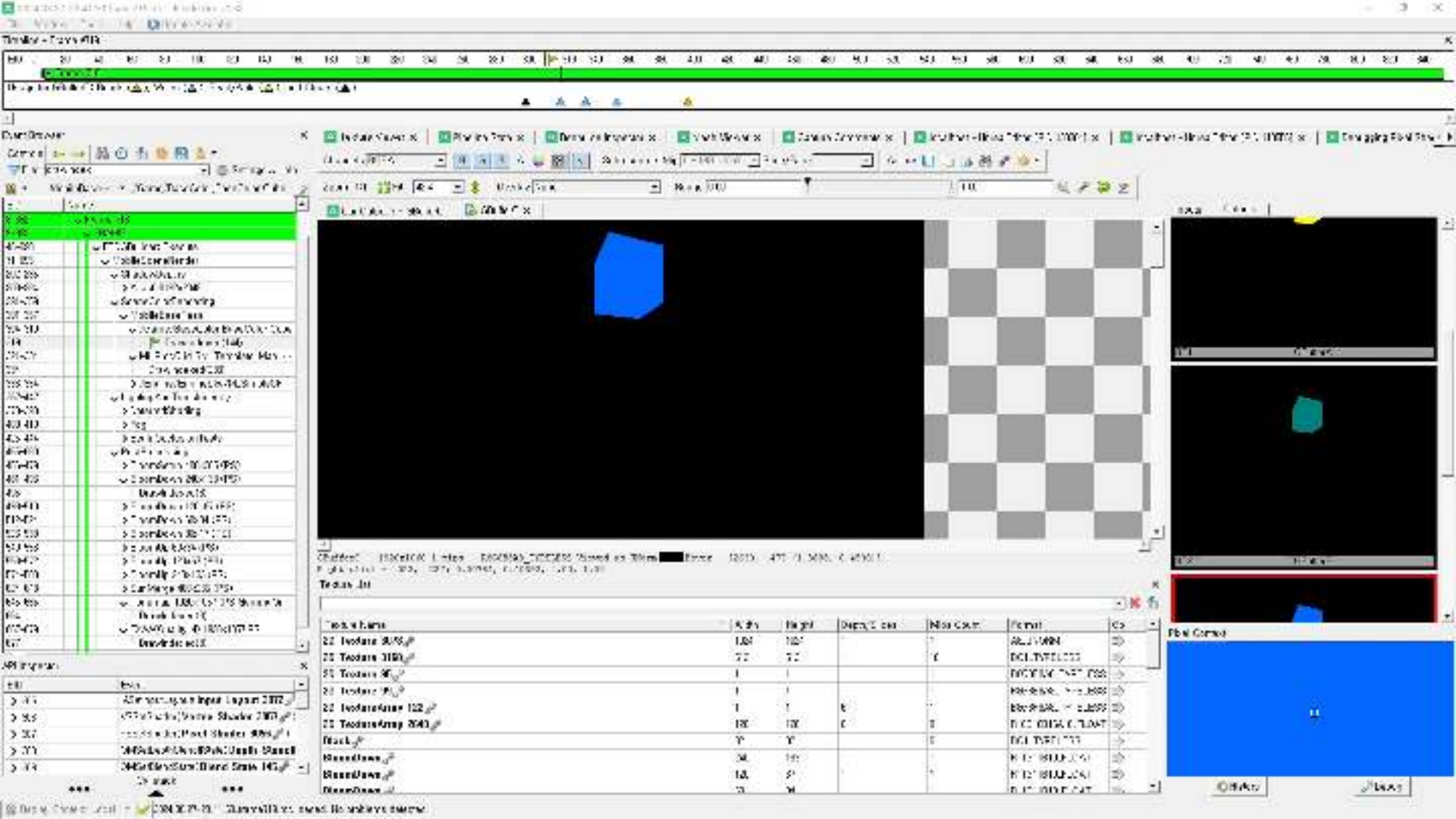


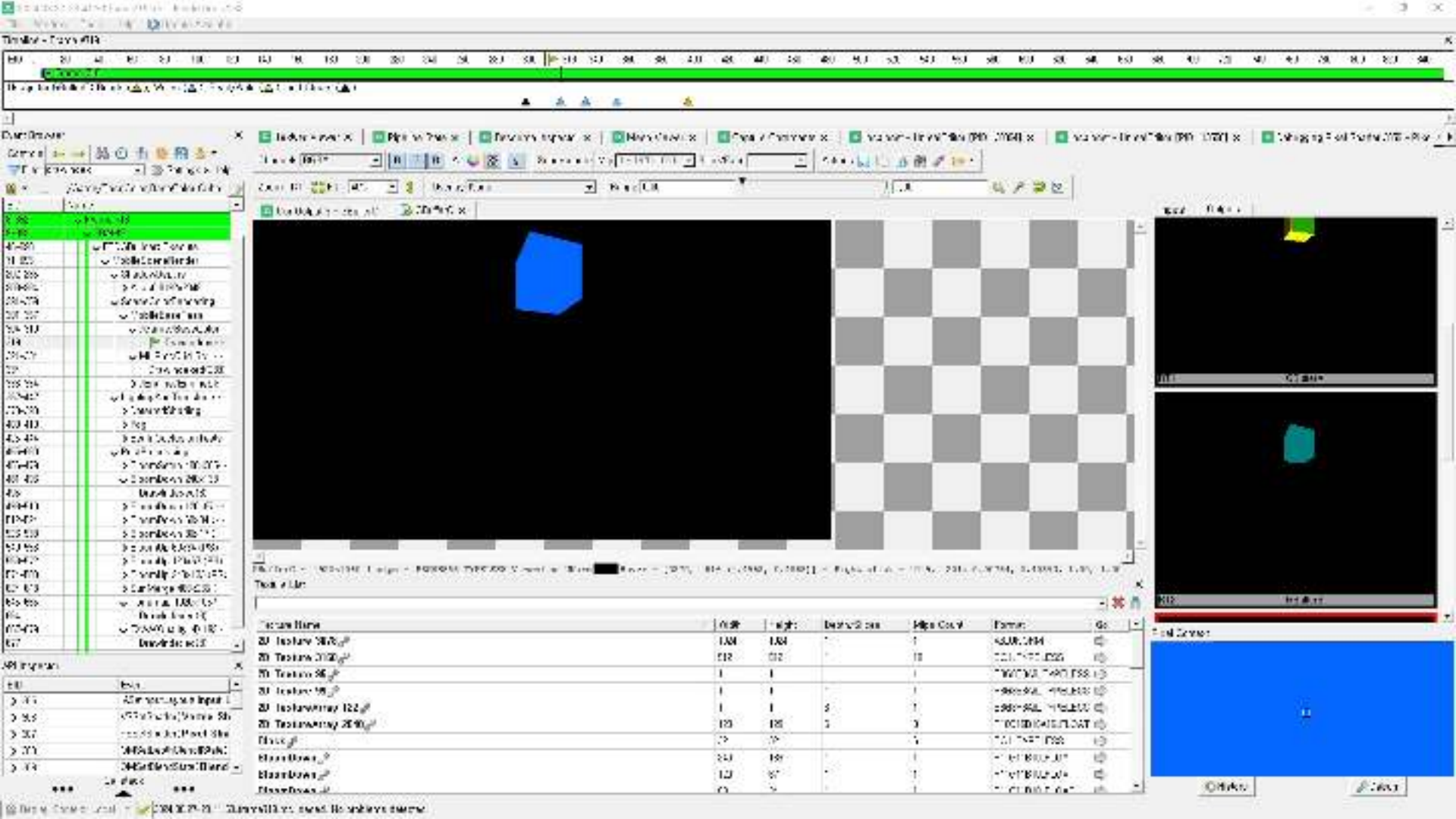

Vertex Shader
Pixel Shader
Output Merger

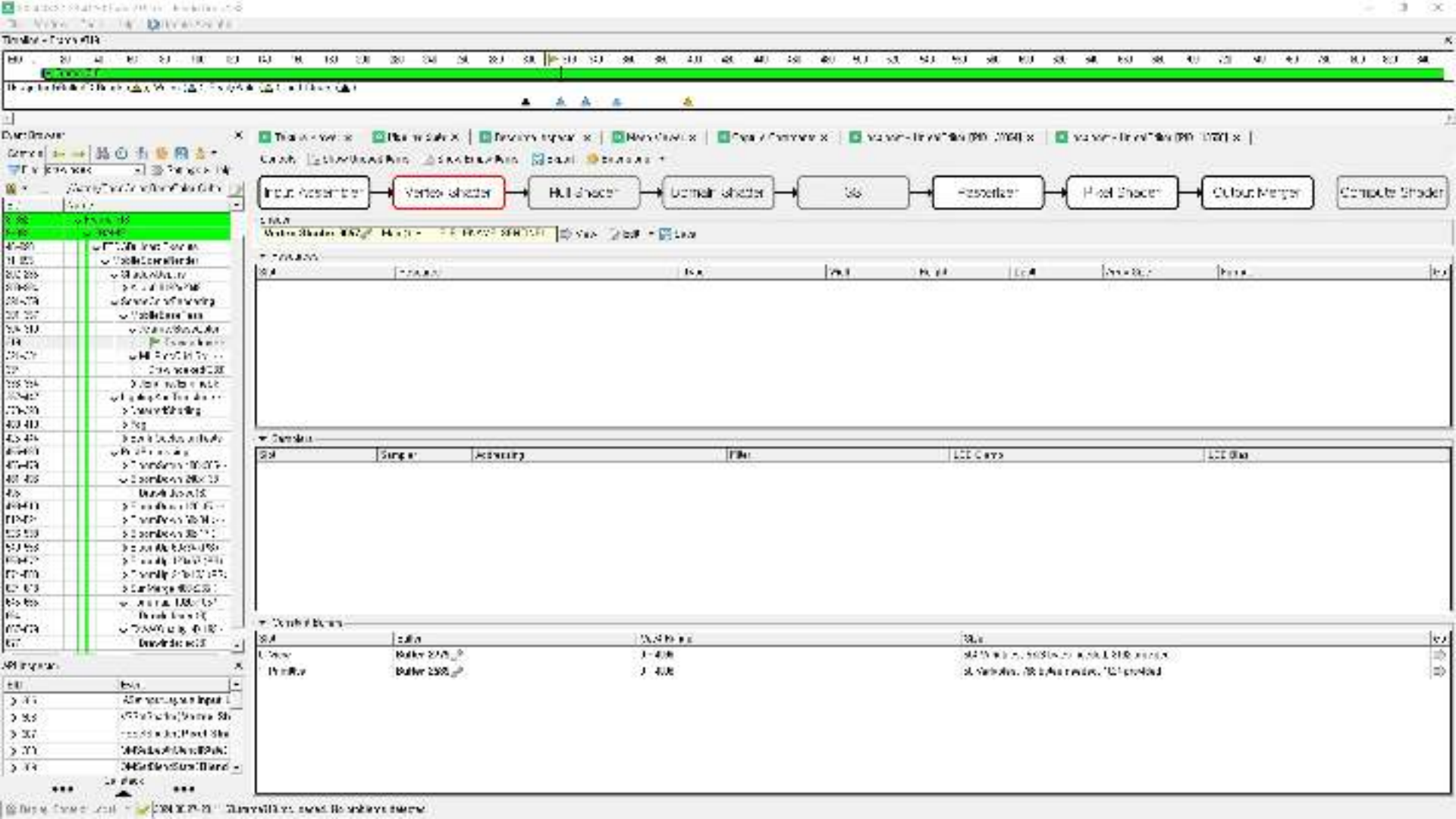





Input Assembler
Vertex Shader
Hull Shader
Domain Shader
GS
Rasterizer
Pixel Shader
Output Merger
Compute Shader






Input Assembler
Vertex Shader
Hull Shader
Domain Shader
GS
Rasterizer
Pixel Shader
Output Merger
Compute Shader

GPU Dump Viewer Tool

렌더링 문제를 조사하고 디버깅 할 수 있도록 렌더링 중간 결과물 실제 텍스처와 버퍼를 디스크에 덤프 하는 인하우스 툴

지원 플랫폼

Windows D3D11, D3D12, Vulkan
Linux Vulkan
Mac Metal, AGX
PlayStation 4 및 PlayStation 5
XboxOne, Xbox Series X/S
Nintendo Switch
iOS 및 Android

GPU Dump Viewer Tool

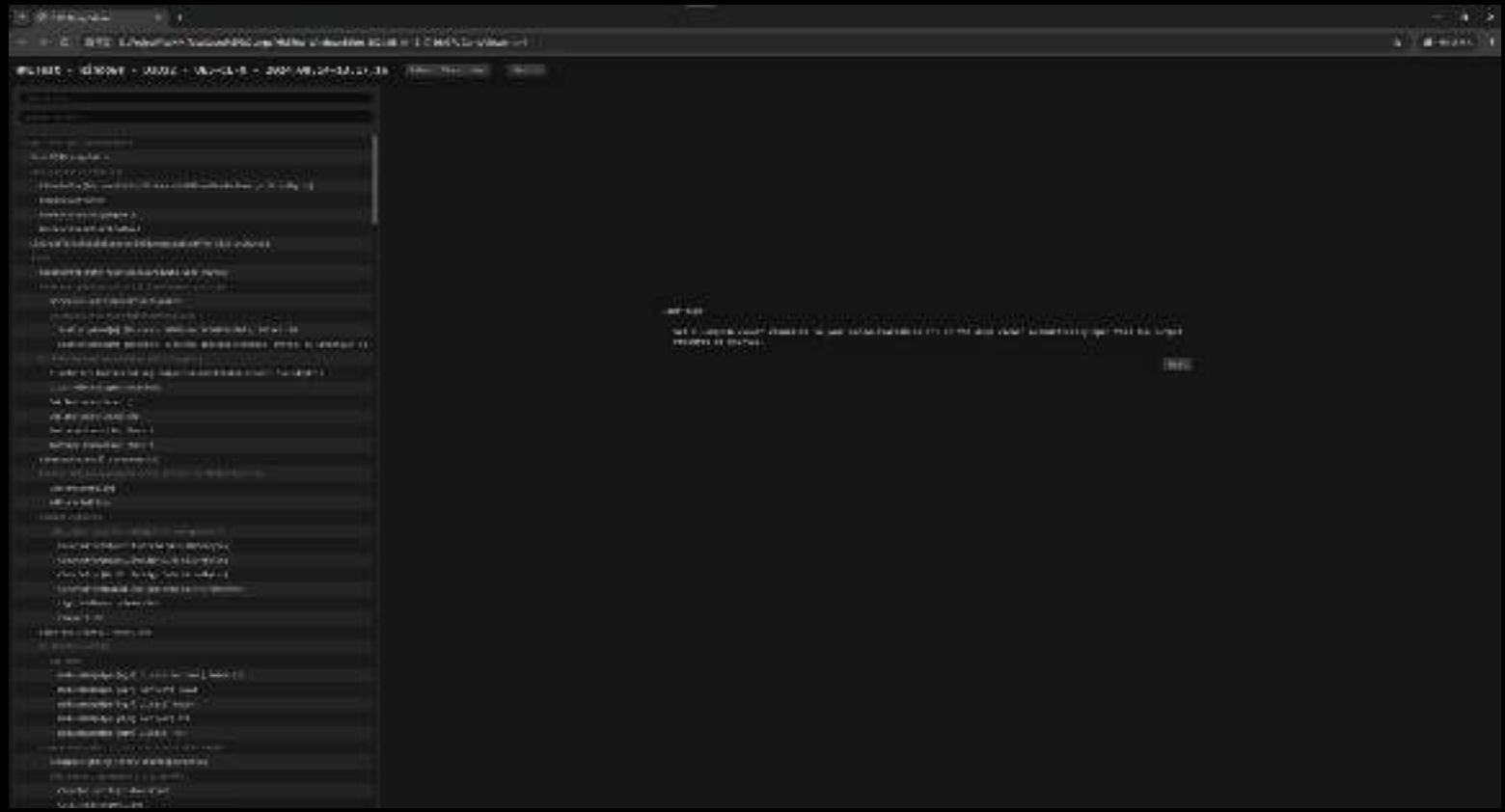
Saved/GPUDumps 위치에 생성됨

ctrl + shift + / or **DumpGPU command 입력**

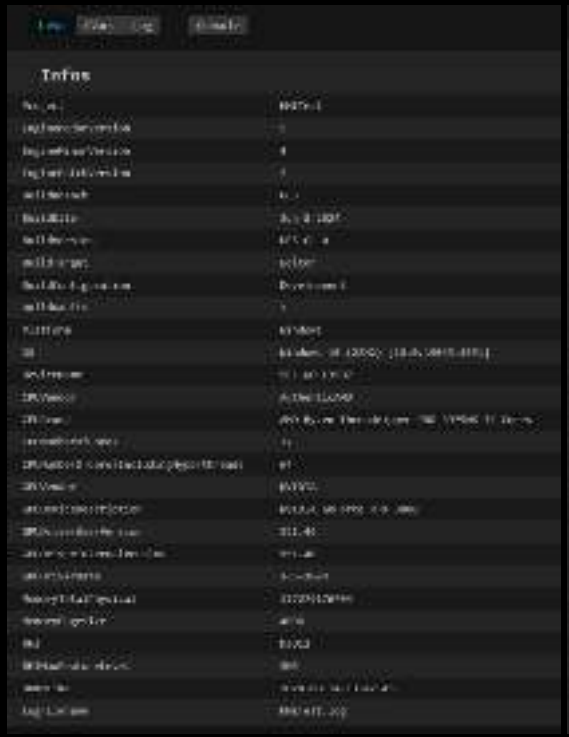
Editor builds	*<ProjectDir>/Saved/GPUDumps*
Staged builds	*<StagedDirectory>/Saved/GPUDumps*
XB1 & XSX	\\<devkit_ip>\SystemScratch\Settings\<ProjectName>\Saved\GPUDumps
PS4 & PS5	Q:\<devkit_ip>\devlog\app\<ProjectName>\<ProjectName>\saved\gpudumps
iOS	<Path to App container>/Documents/<ProjectName>/Saved/GPUDumps/
Android	/storage/emulated/0/UnrealGame/<ProjectName>/<ProjectName>/Saved/GPUDumps/
Switch	<Path SD Card>/<ProjectName>/<ProjectName>/Saved/GPUDumps

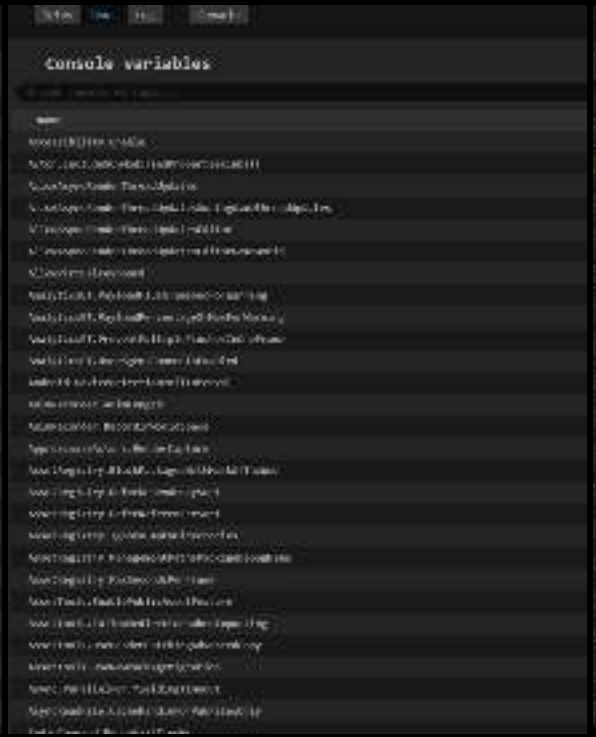
```
GlobalDefinitions.Add("ALLOW_CONSOLE_IN_SHIPPING=1");
GlobalDefinitions.Add("ALLOW_DUMPGPU_IN_SHIPPING=1");
```

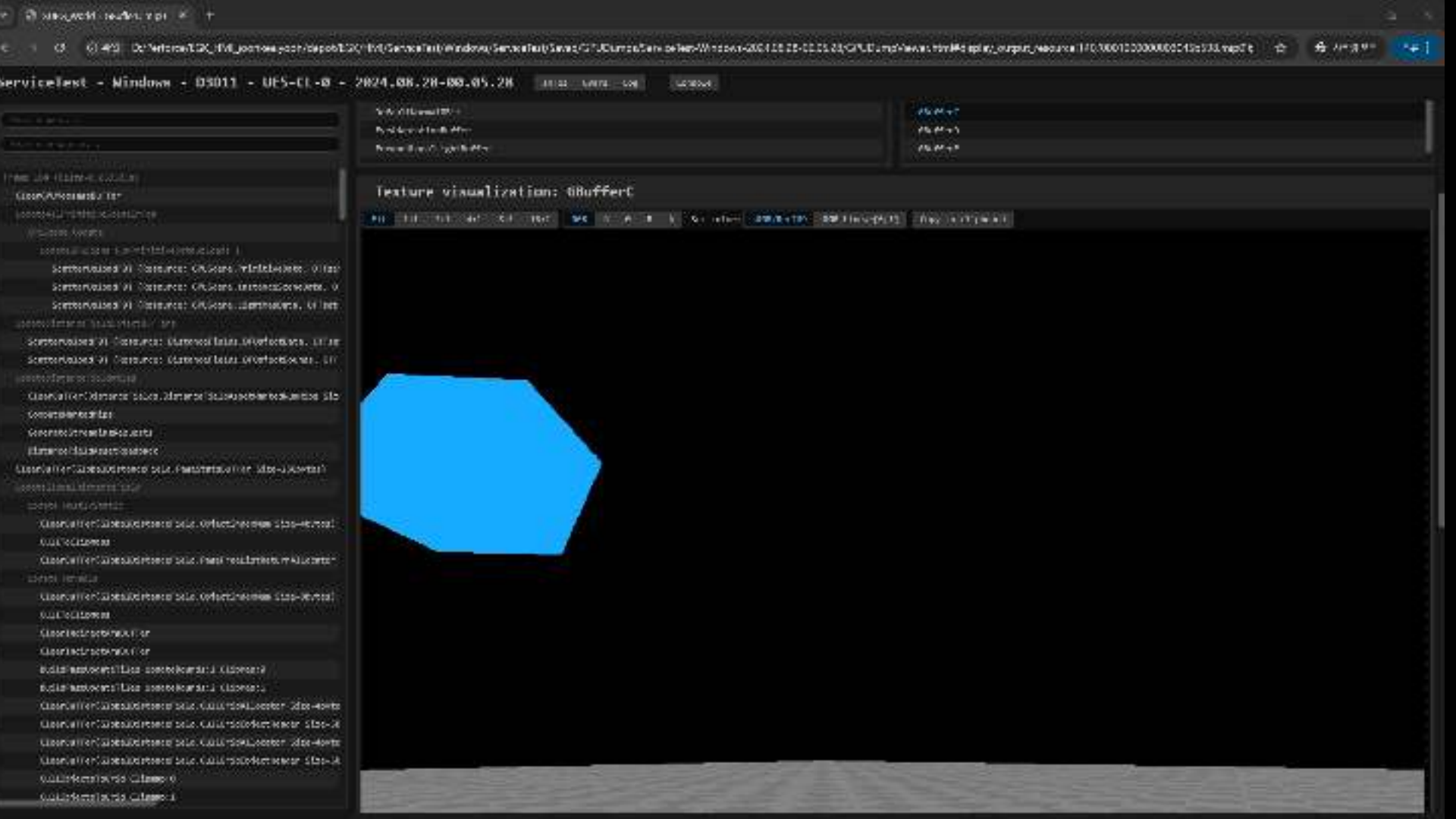
GPU Dump Viewer Tool

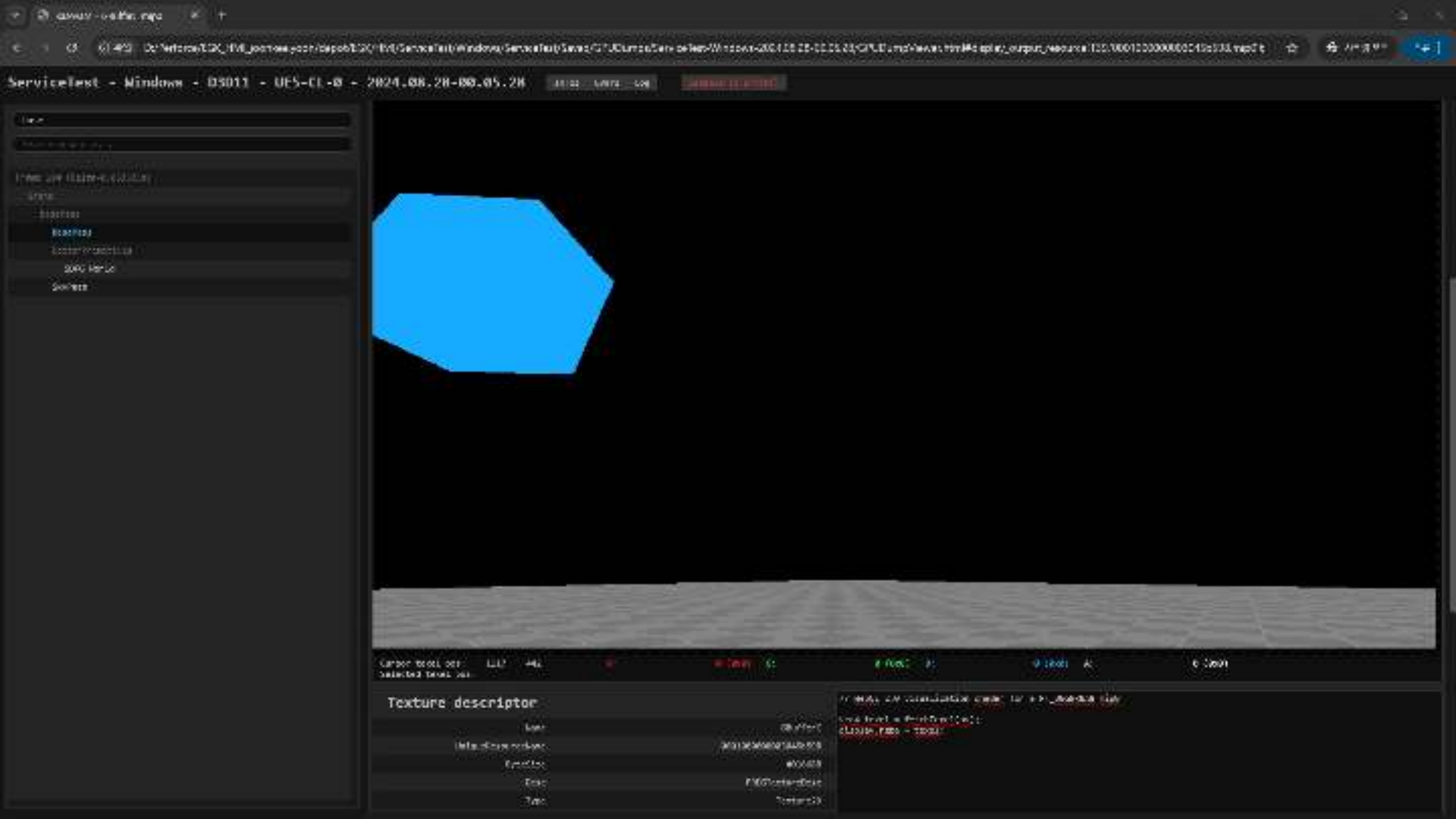


GPU Dump Viewer Tool

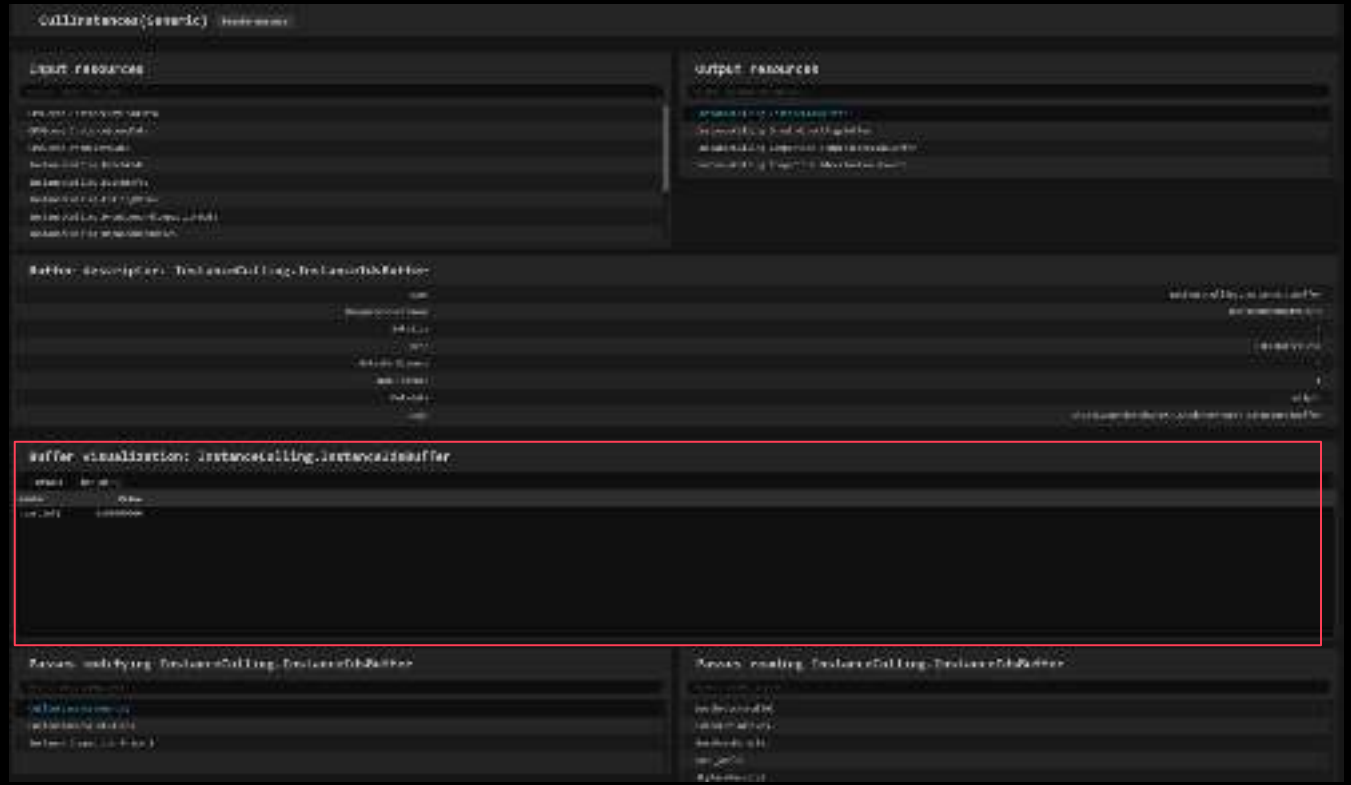





Texture visualization: GBuffer0


Texture descriptor

GPU Dump Viewer Tool



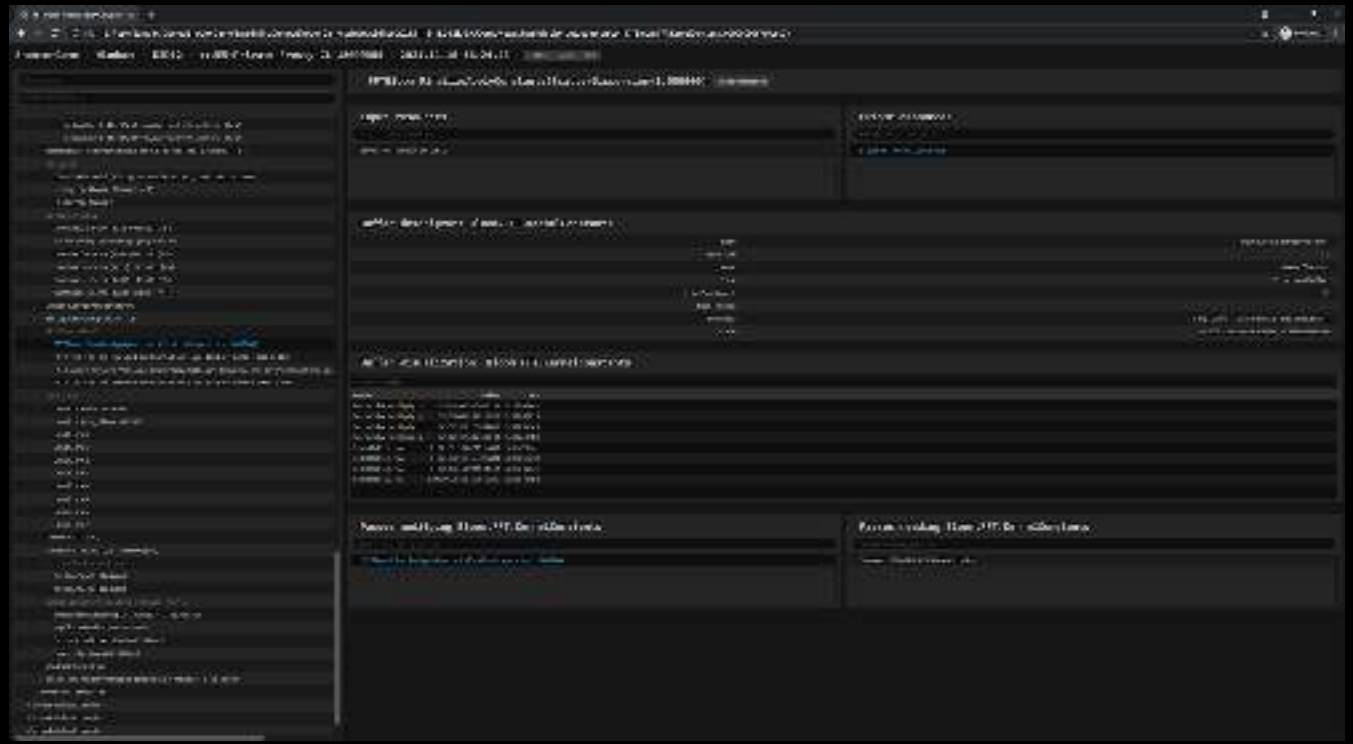
버퍼 시각화 구조체 정의하는 법

GPU Dump Viewer Tool

```
// Generate the apply constant buffer for the tone-maping pass.
{
    check(FBloomOutputs::SupportsApplyParametersBuffer(View.GetShaderPlatform()));

    BloomOutput.ApplyParameters = GraphBuilder.CreateBuffer(
        FRDGBufferDesc::CreateStructuredDesc<FBloomOutputs::FApplyInfo>(/* NumElements = */ 1),
        TEXT("Bloom.FFT.KernelConstants"));

    FBloomFinalizeApplyConstantsCS::FParameters* PassParameters = GraphBuilder.AllocParameters<FBloomFinalizeApplyConstantsCS::FParameters>();
    PassParameters->ScatterDispersionIntensity = View.FinalPostProcessSettings.BloomConvolutionScatterDispersion;
    PassParameters->KernelConstantsBuffer = GraphBuilder.CreateSRV(KernelConstantsBuffer);
    PassParameters->BloomApplyConstantsOutput = GraphBuilder.CreateUAV(BloomOutput.ApplyParameters);

    TShaderMapRef<FBloomFinalizeApplyConstantsCS> ComputeShader(View.ShaderMap);
    FComputeShaderUtils::AddPass(
        GraphBuilder,
        RDG_EVENT_NAME("FFTBloom FinalizeApplyConstants(ScatterDispersion=%f)", PassParameters->ScatterDispersionIntensity),
        Intermediates.ComputePassFlags,
        ComputeShader,
        PassParameters,
        FIntVector(1, 1, 1));
}
```

버퍼 시각화 편하게 보기

GPU Dump Viewer Tool



편안하다

RenderDoc 과 GPU Dump Viewer Tool 비교

RenderDoc 안에서
ShaderCode 나 파라미터를
수정해서 바로 결과를 확인 할 수 있다.

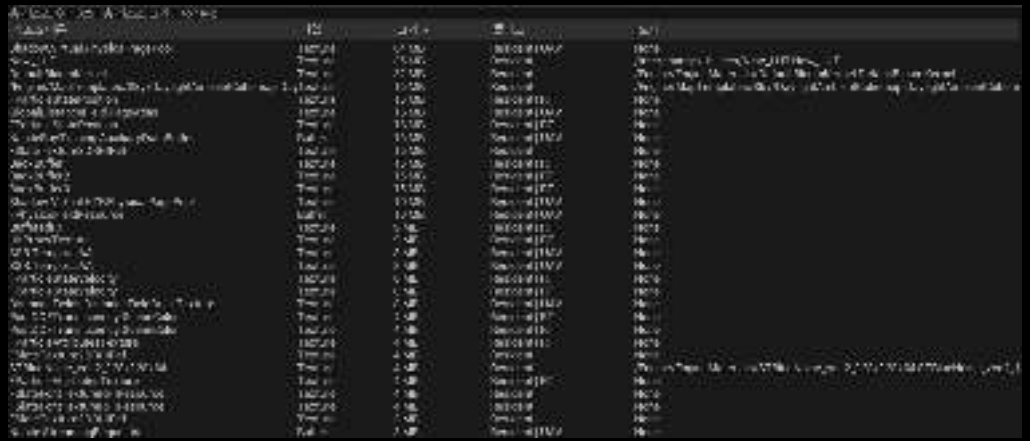
실제로 다시 재구성 하는 것이기
때문에 실제 화면과 약간 다를 수 있다.

빠르게 캡쳐가 가능하고
GPU Dump Viewer Tool
산출물인 html 를 공유 하면
어디서든 확인 할 수 있다.

그외 추천 Tool

Render Resource Viewer

실제 GPU 로 Bind 된 Memory 확인이 가능하다.



우 클릭 시 해당 에셋 으로 이동 가능

Texture/VertexBuffer/IndexBuffer
와 같은 렌더 리소스 들이 StaticMesh 나 SkeletalMesh 와 같은 어떤 Asset에서 오는 것인지 파악 할 수 있다.

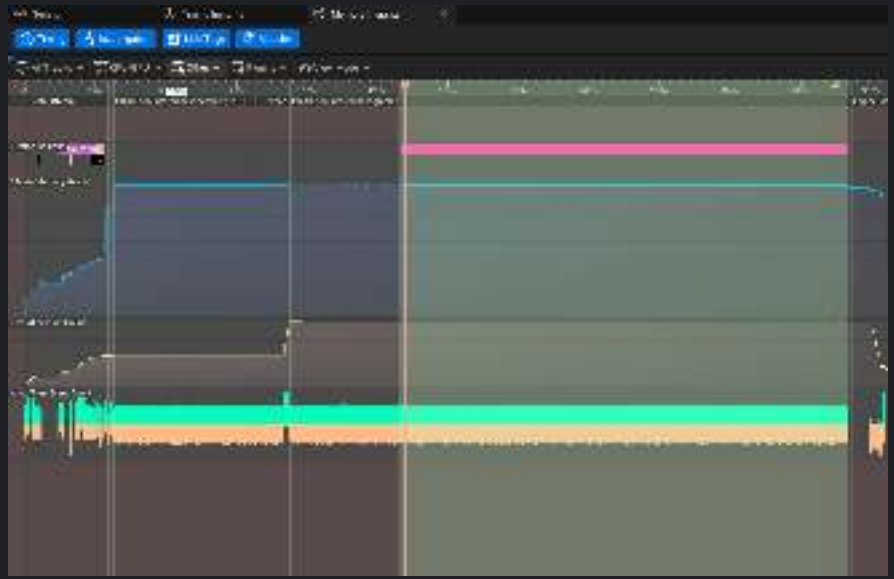
메모리 릭 발생

Memory 이슈

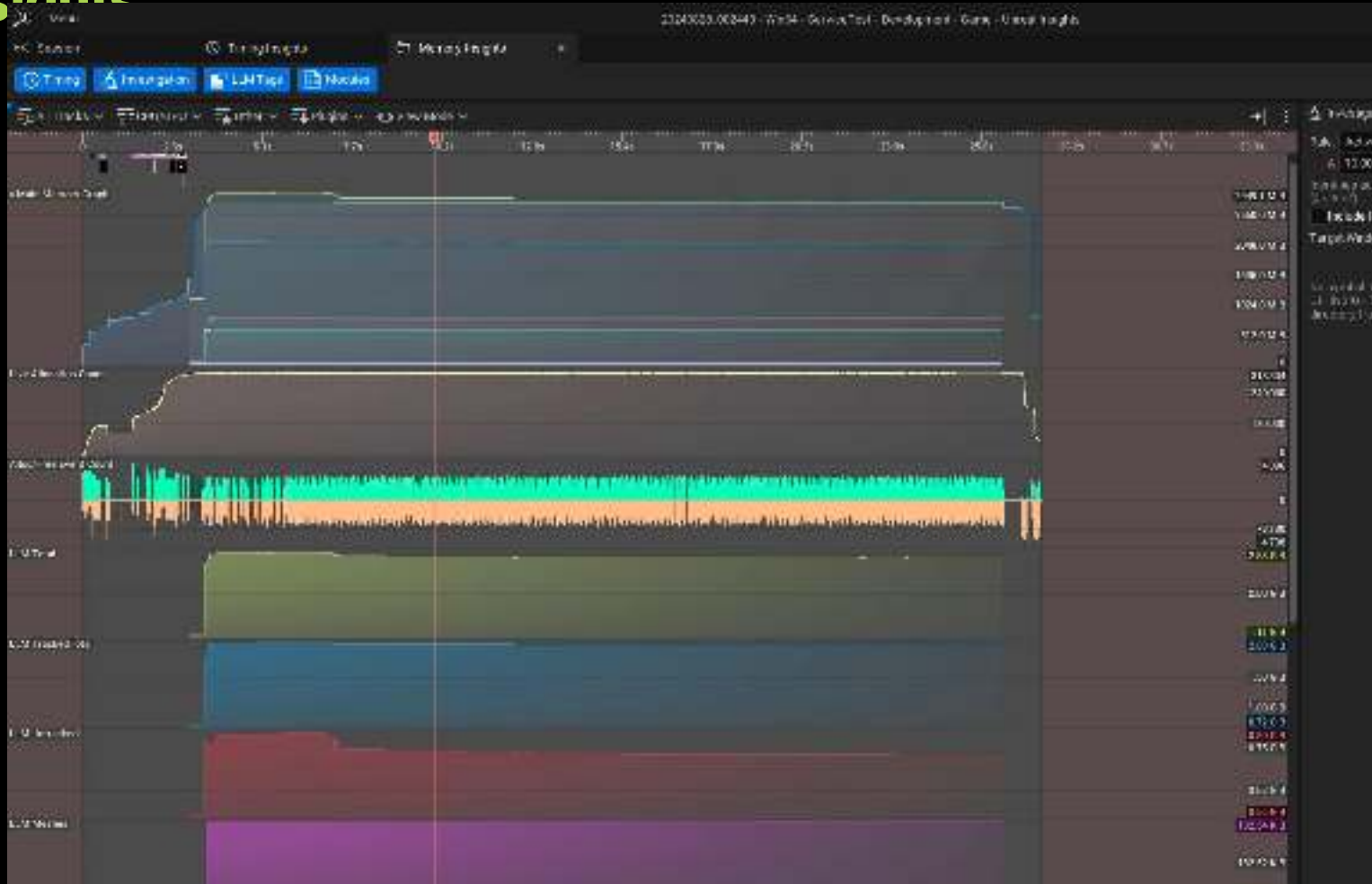
```cpp
// Called every frame
void AActorTicker::Tick(float DeltaTime)
{
    Super::Tick(DeltaTime);

    int* a = new int();

}
```

Memory Insights

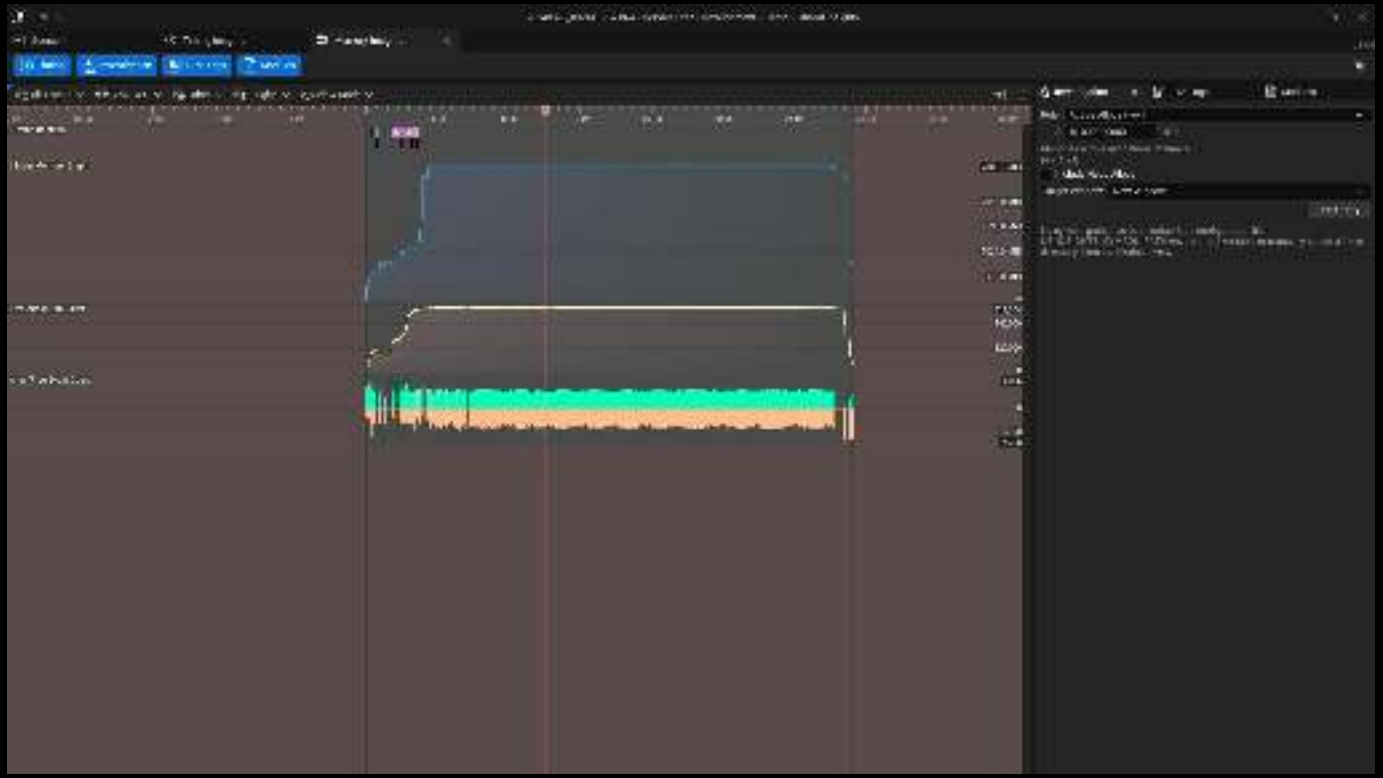
메모리 블록의 연결 된 콜스택 메모리 할당 및 해제에 대한 정보와 사용량의 증감, 단기/장기 할당을 구분하고 메모리 누수를 찾을 수 있는 쿼리 시스템을 제공



Memory Insights



Memory Insights



-trace=default,memory
Rule: memory Leak A-B 구간 에 메모리가 할당된 이후 C구간 이후에 해제 된 메모리 추적

메모리릭 리포터

```
GlobalDefinitions.Add("MALLOC_LEAKDETECTION=1")
```

[command line]
mallocleak.start
mallocleak.report
mallocleak.stop

Saved/Profiling/MemReports/…
.memreport

```
LogLeakDetector: No leaks found
LogUObjectHash: Compacting FUObjectHashTables data took  X.XXms
LogEngine: MemReportDeferred: saving
to ../../../MyProject/Saved/Profiling/MemReports/NewMap-WindowsNoEditor-
XX.XX-XX.XX.XX/XXX_NewMap.memreport
```

누수가 없을 경우

```
AllocSize: 1776 KB, Num: 37, FirstFrame 1572, LastFrame 1890,
KnownDeleter: 1, KnownTrimmer: 0, Alloc Rate 0.00B/frame
0x00007ff6546ae3db
MyProject.exe!FWindowsPlatformStackWalk::CaptureStackBac
kTrace()0x00007ffc9d2ace51 ntdll.dll!UnknownFunction []
```

누수가 있을 경우

언리얼 빌드 툴을 통한 정적 검증

RunUBT.bat [ProjectName] Android Development –StaticAnalyzer=Clang –project=[.uproject 경로] –Module=[모듈이름] –DisableUnity

모듈 단위로 돌려 볼 수 있기 때문에 자기가 짠 코드는 자주자주 확인하면 실수를 방지 할 수 있다.

Unity Build(통합 빌드) 사용시 제대로 분석 실패 할 수 있기 때문에 –DisableUnity를 적용

언리얼 빌드 툴을 통한 정적 검증

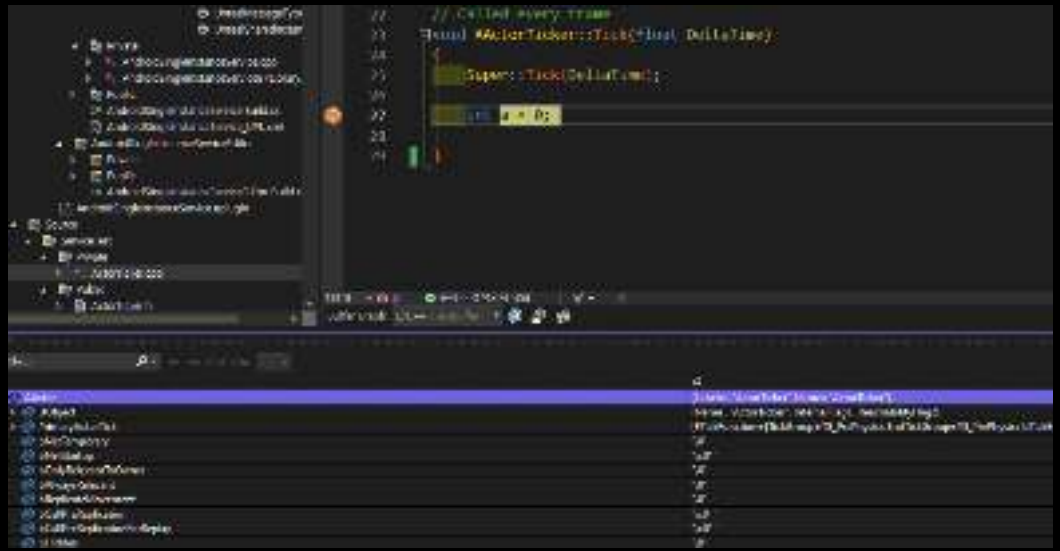
```
D:\Git\5.4\Engine\Build\BatchFiles>RunUBT.bat UnrealFestSeoul Android Development -StaticAnalyzer=Clang -project="D:\ProjectFile\Unre
alFestSeoul\UnrealFestSeoul.uproject" -Module=UnrealFestSeoul -DisableUnity
Using bundled DotNet SDK version: 6.0.302
Log file: D:\Git\5.4\Engine\Programs\UnrealBuildTool\Log.txt
Using 'git status' to determine working set for adaptive non-unity build (D:\Git\5.4).
Waiting for 'git status' command to complete
Building UnrealFestSeoul...
[Adaptive Build] Excluded from Launch unity file: LaunchAndroid.cpp
Determining max actions to execute in parallel (32 physical cores, 64 logical cores)
  Executing up to 32 processes, one per physical core
Using Parallel executor to run 3 action(s)
------ Building 3 action(s) started ------
[1/3] Compile [arm64] SharedPCH.Engine.Project.ValApi.Cpp20.h
[2/3] Analyze [arm64] LeakActor.cpp
D:/ProjectFile/UnrealFestSeoul/Source/UnrealFestSeoul/Private/LeakActor.cpp(21,1): warning: Potential leak of memory pointed to by 'p
IntTest' [cplusplus.NewDeleteLeaks]
}
^

D:/ProjectFile/UnrealFestSeoul/Source/UnrealFestSeoul/Private/LeakActor.cpp(19,18): note: Memory is allocated
        int* pIntTest = new int;
                        ^~~~~~
D:/ProjectFile/UnrealFestSeoul/Source/UnrealFestSeoul/Private/LeakActor.cpp(21,1): note: Potential leak of memory pointed to by 'pInt
Test'
}
^

1 warning generated.
[3/3] Analyze [arm64] UnrealFestSeoul.cpp
Total time in Parallel executor: 34.27 seconds
Total execution time: 39.53 seconds
```

모듈 단위로 돌려 볼 수 있기 때문에 빠르게 빌드 할 수 있어서
자기가 짠 코드는 자주자주 확인하면 실수를 방지 할 수 있다.

안드로이드 디버깅 팁

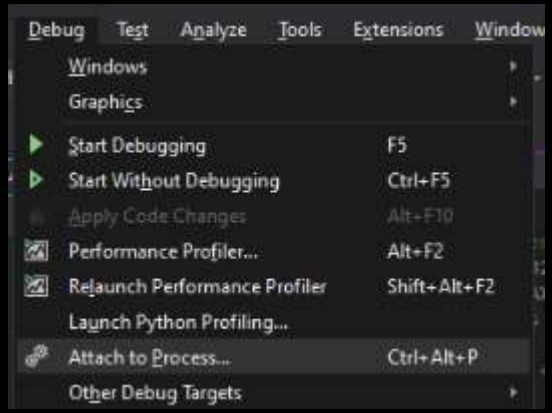
AGDE 안드로이드 디버깅

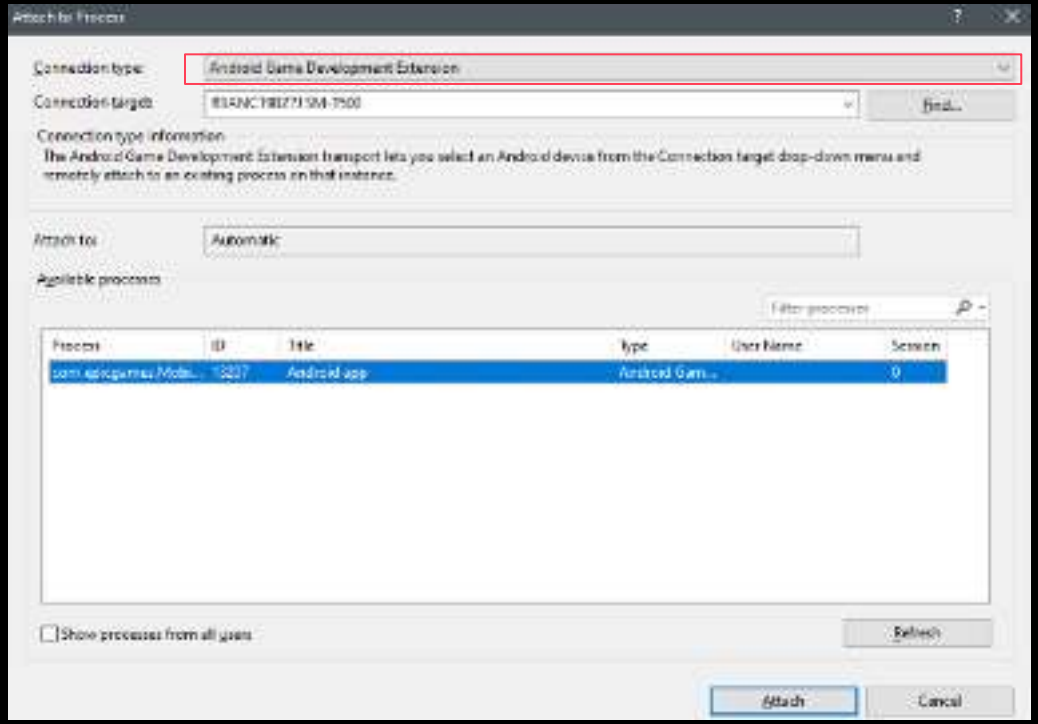
Visual studio
안드로이드 디버깅 (현재 java 는 디버깅 불가)



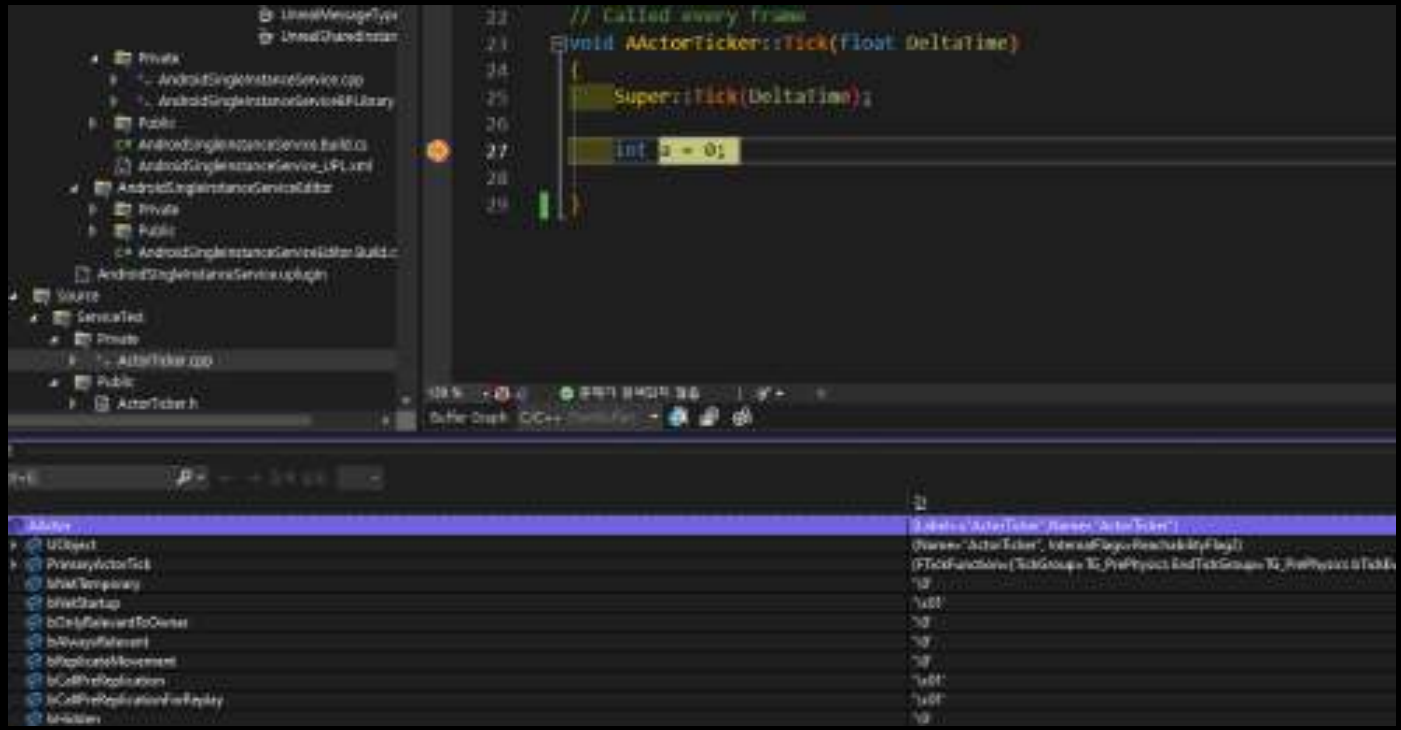
VisualStudio 플러그인을 통해서 가능
AGDE version : 23.1.82 이상
JDK version : OpenJDK 17.0.6 2023-01-17

AGDE 안드로이드 디버깅





AGDE


// Called every frame
void AActorTicker::Tick(float DeltaTime)
{
Super::Tick(DeltaTime);
int i = 0;
}


감사합니다.
— 에픽게임즈 코리아