



UNREAL
ENGINE

UE4 プロファイリングツール総おさらい (グラフィクス編)

Nori Shinoyama
Senior Support Engineer @ Epic Games Japan



最適化の前にプロファイル はじめに

本日の内容

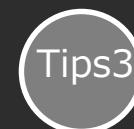
UE4のプロファイリングツールを俯瞰で眺めながら、
UE4上でどのように
プロファイルしていくのかを見ていきます

なぜプロファイル？

どうしてそんな狭く地味な範囲を？
最適化じゃないの？

なぜプロファイル? あるあるCase.1

ありがたいことに、コミュニティやライセンシ様のおかげで、ネット上に沢山の最適化Tipsが出てくるようになりました。

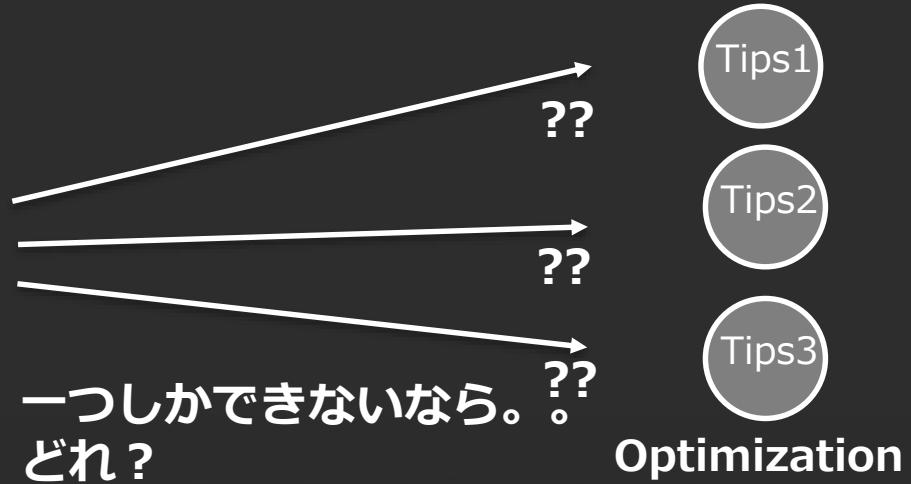


Optimization

なぜプロファイル? あるあるCase.1

でも、こんなときはどうしよう?

「今週中にこのシーン60fpsで安定させて。まだ20fpsしか出ないけど」



なぜプロファイル? あるあるCase.1

でも、こんなときはどうしよう？

「今週中にこのシーン60fpsで安定させて。まだ20fpsしか出ないけど」



Profiling

どれが一番効果的？

Tips1

Tips2

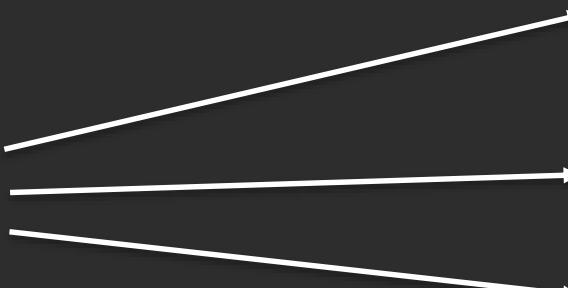
Tips3

Optimization

なぜプロファイル? あるあるCase.2



全部やりました！！



。。本当に必要？

Tips1

Tips2

Tips3

Optimization

なぜプロファイル? あるあるCase.2

半透明は重いらしいので、
エフェクトは減らしました！

すべてのテクスチャ
解像度を512に落としました！

DrawCall増えると怖いので
オブジェクトをまとめました！

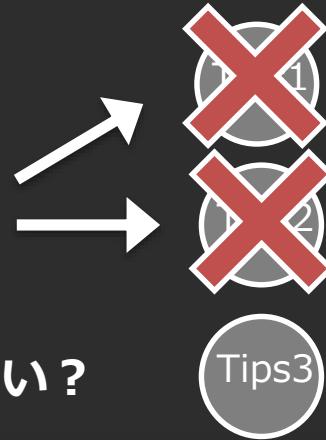
とりあえずライトはMovable
をやめて全部Stationaryに！

なぜプロファイル? あるあるCase.2



Profiling

どれはやらなくていい?



Optimization

どちらも極端な例ですが、
実際の制作現場でよくあることだと思います。

プロファイルしない最適化は危険

中身を知らずにとりあえずその設定を試したり。。
ボトルネックと違うところを最適化したり。。

最適化に時間や労力をかけても、
パフォーマンスは上がらず、
クオリティだけ下がってしまったり。。

なぜプロファイル？

最適化の前に、
開発初期から、日常的に、
プロファイルする癖を

ボトルネックの予測精度を上げて。。。

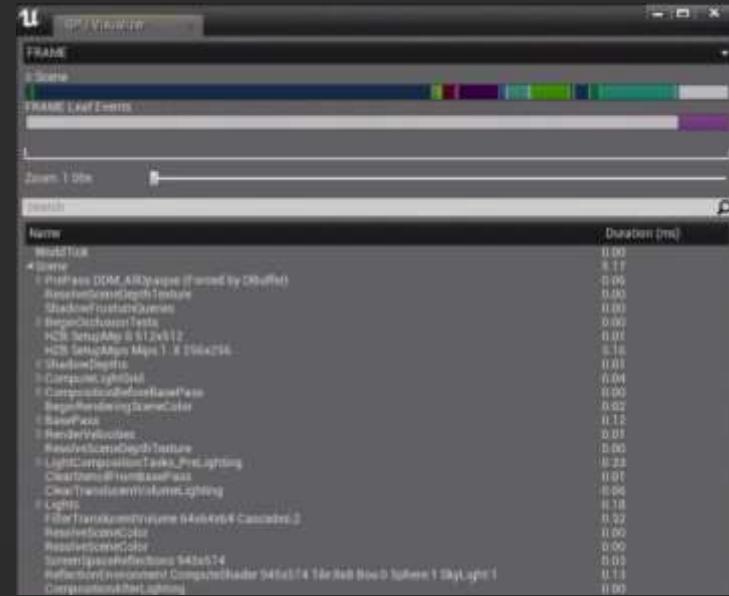
ボトルネック部分 -> 短時間で効率的な最適化を
ボトルネックじゃない部分 -> 豪沢に使いクオリティアップに

UE4の充実したプロファイリングツール



Cycle counters (flat)

	TickCount	InclusiveAvg	InclusiveMax	ExclusiveAvg	ExclusiveMax
View Visibility	0.02 ms	0.17 ms	0.01 ms	0.01 ms	
GetDynamicMeshElements	0.05 ms	0.07 ms	0.01 ms	0.01 ms	
View Relevance	0.03 ms	0.04 ms	0.00 ms	0.00 ms	
Occlusion Cull	0.02 ms	0.04 ms	0.00 ms	0.00 ms	
Init dynamic shadows	0.02 ms	0.03 ms	0.00 ms	0.00 ms	
Compute View Relevance	0.01 ms	0.02 ms	0.01 ms	0.02 ms	
Frustum Cull	0.01 ms	0.01 ms	0.00 ms	0.00 ms	



本日の内容（再掲）

UE4のプロファイリングツールを俯瞰で眺めながら、
UE4上でどのように
プロファイルしていくのかを見ていきます

本講演の対象者

UE4を触って頂いている
技術寄りのアーティストやデザイナの方がメインです

(触っていない方は、UE4はこんな機能があるのか～と眺めて頂ければ)

本講演で話すこと

- コリジョン、ネットワーク、アニメーションなどのプロファイル手法
- モバイルやVRに特化した内容
- きれいな絵作りする方法
- プラットフォーム固有の話

備考

- Twitter等での情報拡散にご協力ください。
 - #UE4CEDEC
- 本資料はすぐにネットにアップされます。
- エディタは英語版を使います。
- UE4.17をベースにします。
- 質疑応答は時間があまれば。。

本講演の目次

1. はじめに
2. 基本のプロファイリングツール 1
3. プロファイルの流れ
4. 基本のプロファイリングツール 2
5. Showcase
6. まとめ&おまけ

本講演の目次

1. はじめに
- 2. 基本のプロファイリングツール 1**
3. プロファイルの流れ
4. 基本のプロファイリングツール 2
5. Showcase
6. まとめ&おまけ

Console Command & CVars

基本のプロファイリングツール 1

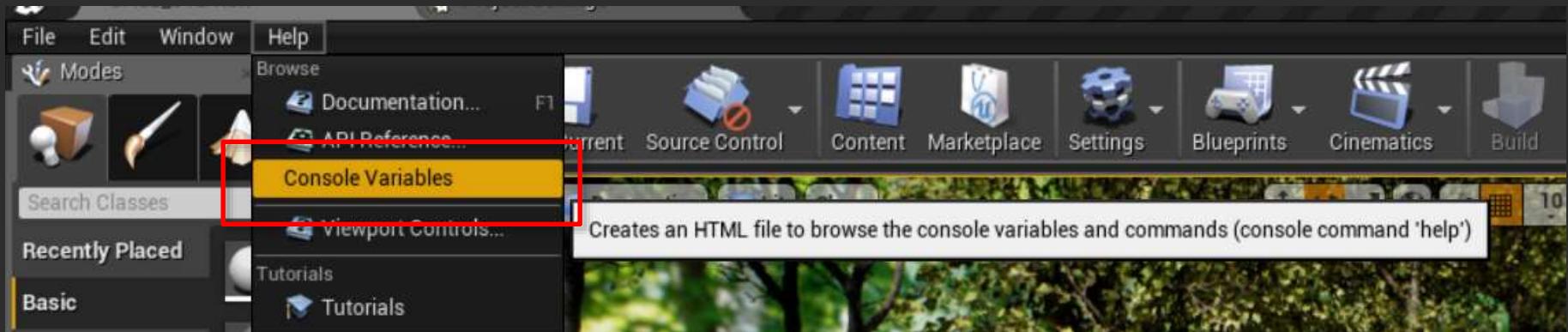
Console Command & Console Variables (CVars)

Console Command & Variables (CVars)

コマンドによって、ゲーム内の挙動を変えたり情報を出力する方法
(大文字小文字の区別はありません)



Console Command & Console Variables (CVars) 一覧表の作成



Help -> “Console Variables”
を選ぶ事によって、全コマンドのHTMLヘルプを出力

Console Command & Console Variables (CVars) 一覧表

Unreal Engine 4 Console Variables and Commands

[Search Bar]

All (1937) Renderer (700) Sound (19) Timer (16) HII (12) Network (39) OpenGL (13) Game (7) ScalabilityGroups (15) State (43) Physics (47)
ShowFlags (152) Particle FX system (19)

Type:
Console Variables
Console Commands
Exec Commands

Search in help as well

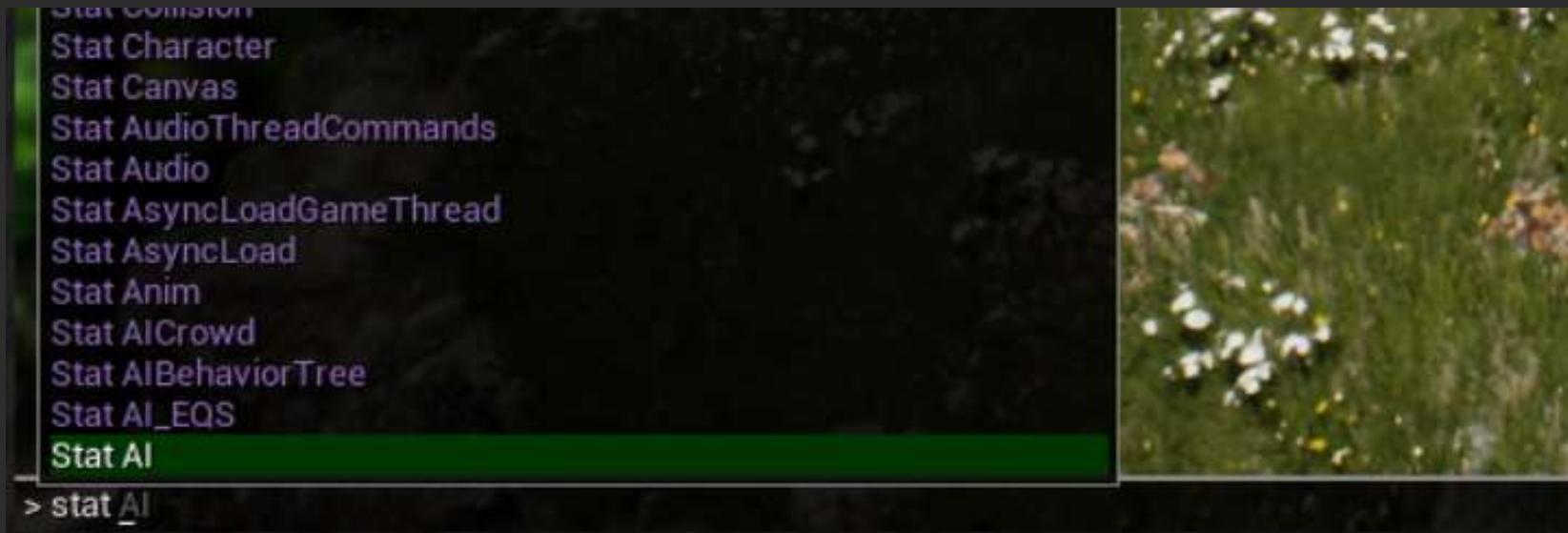
Name	Help
a_AntiNode_AimOffsetLookAt_Debug	Toggle LookAt AimOffset debug
a_AntiNode_AimOffsetLookAt_Enable	Enable/Disable LookAt AimOffset
a_AntiNode_LegIK_AveragePull	Leg IK AveragePull

.htmlファイルで、各コマンドとそのヘルプを出力してくれる



Command入力方法

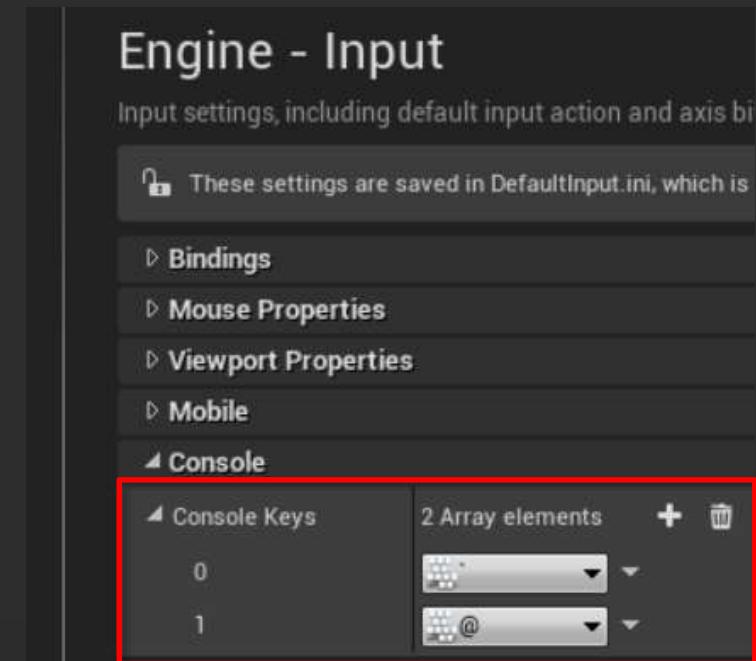
Command / CVar の送り方 1



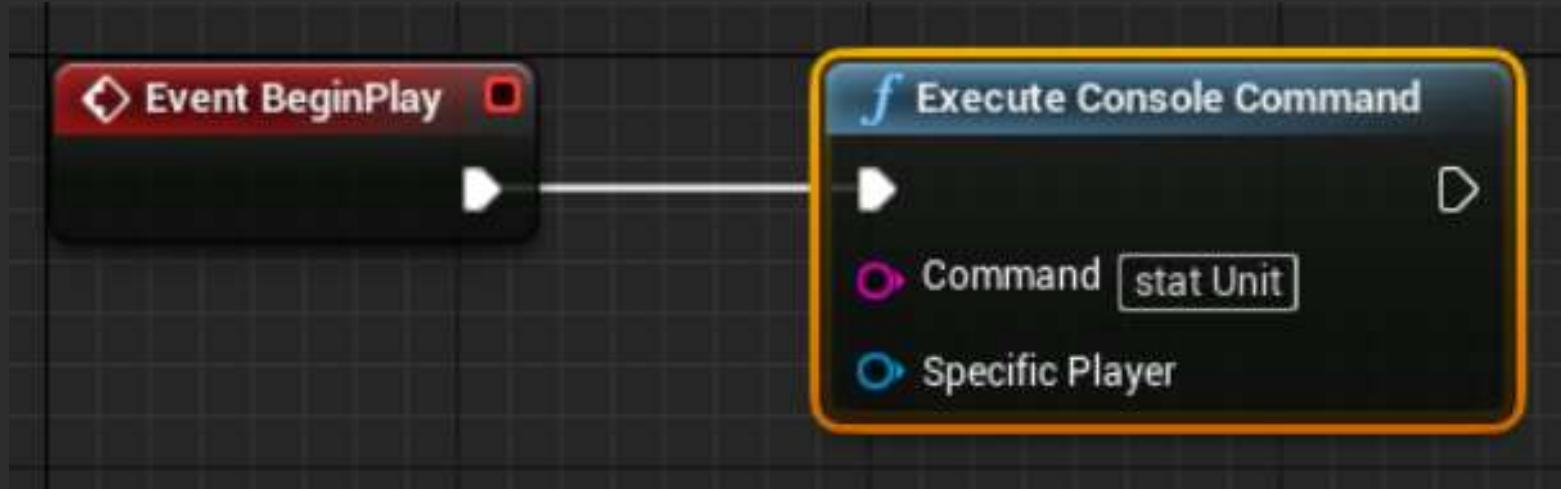
開発機にキーボードを指して、コンソールウィンドウを立ち上げる
(Editorでももちろん可能。Output Windowからも入力できる。)

Console Windowを立ち上げショートカットの指定

Project Settings/
Input
の一番下に
“Console Keys”
という設定があり、
そこで指定できます。

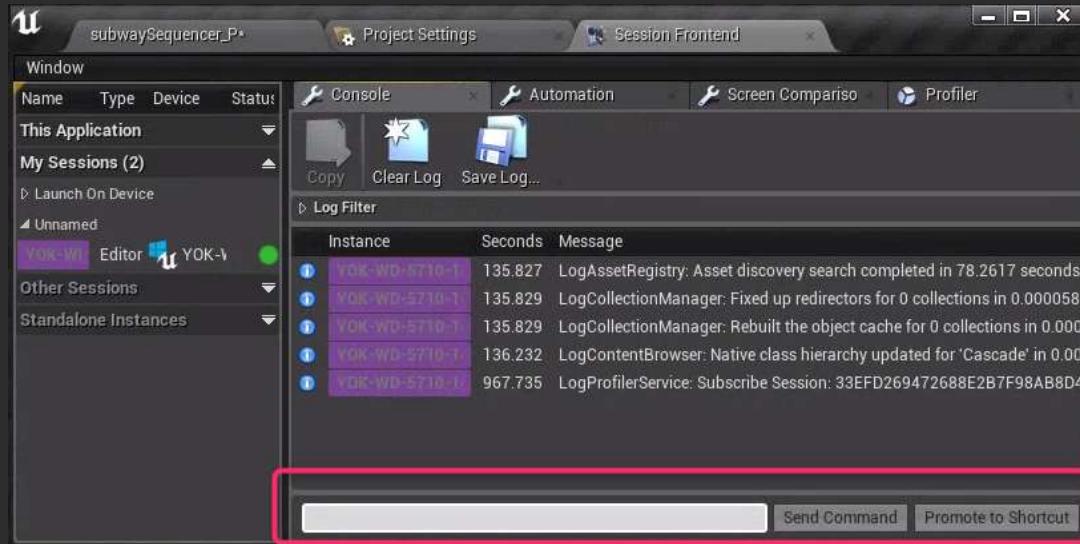


Command / CVar の送り方 2



Blueprintで送る
“Execute Console Command” ノード

Command / CVar の送り方 3



SessionFrontendを使って、PC上から実機で動いている
アプリケーションに対してコマンドを送る

Command / CVar の送り方 4

Command	\$(TargetPath)
Command Arguments	D:\Workspace\TP417\TP417.uproject -execmds="stat unit, stat fps"
Working Directory	\$(ProjectDir)
Attach	No

起動時引数で送る
-execmds="コマンドA, コマンドB"
";" で複数コマンドを区切る

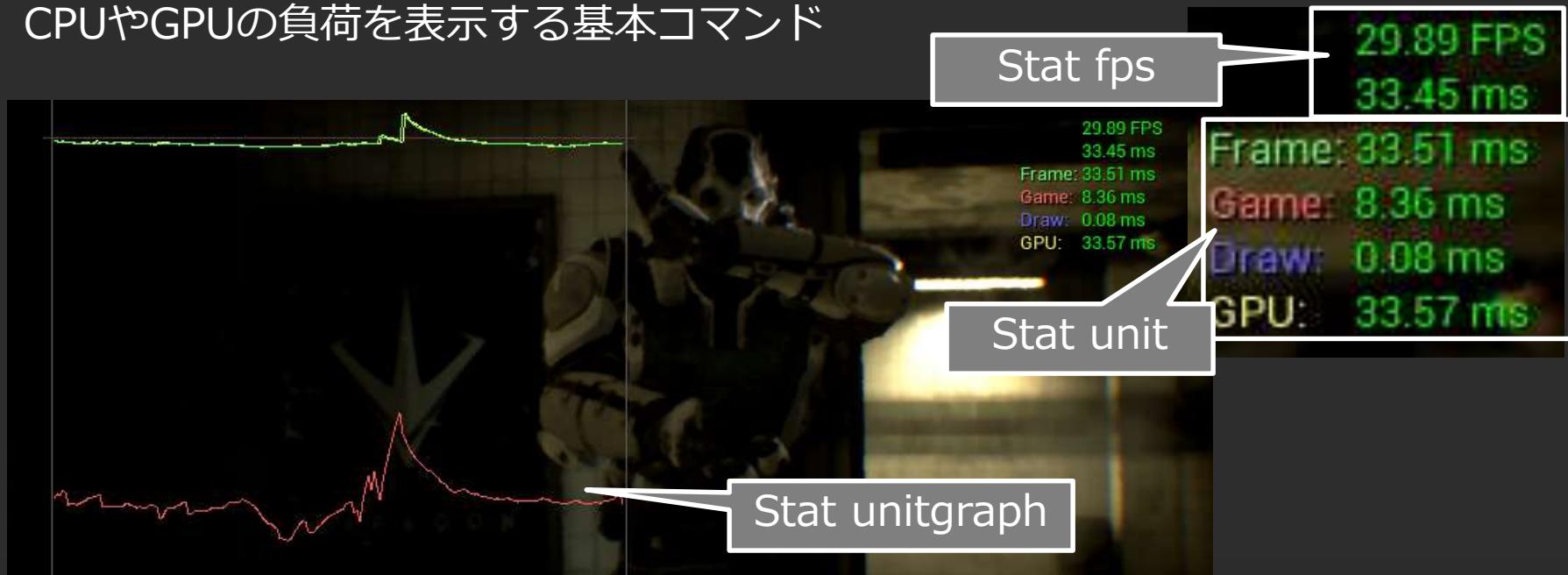


では、
プロファイルに使えるコマンドを見ていきます

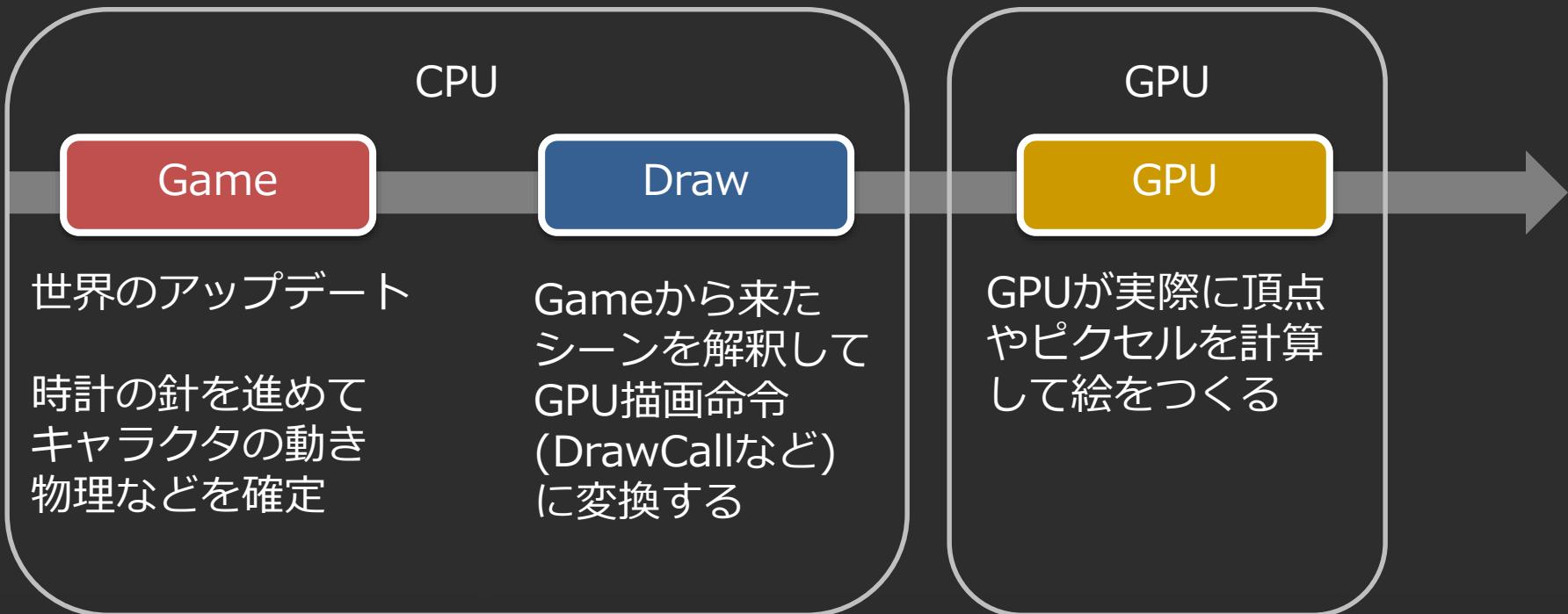
Stat fps / unit / unitgraph

Console Command **stat fps / stat unit / stat unitgraph**

CPUやGPUの負荷を表示する基本コマンド



レンダリングから見た Game / Draw / GPU



Game



Draw

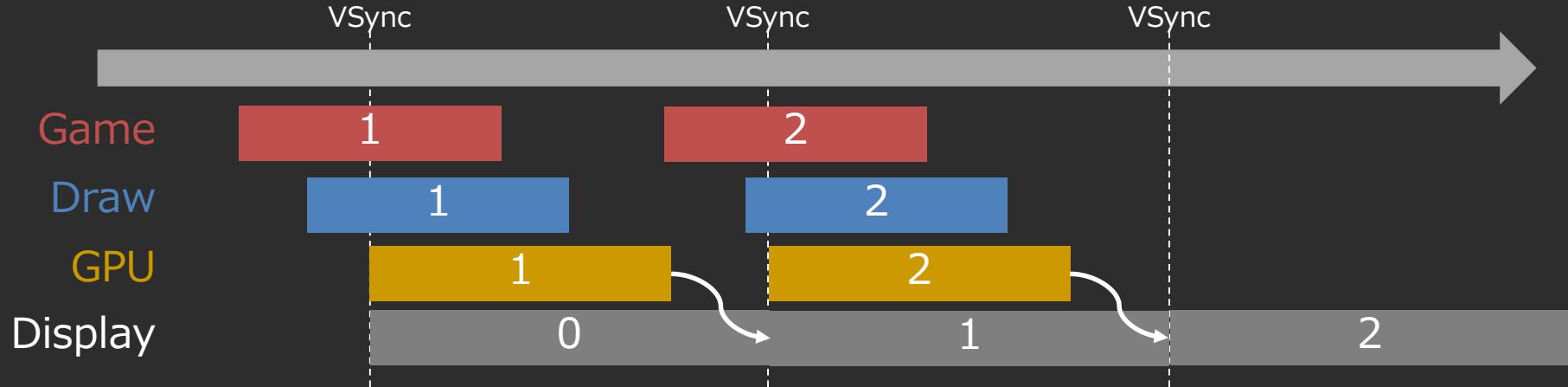


GPU



合計がそのフレームの処理時間じゃないの？？

Game / Draw / GPU の並列処理

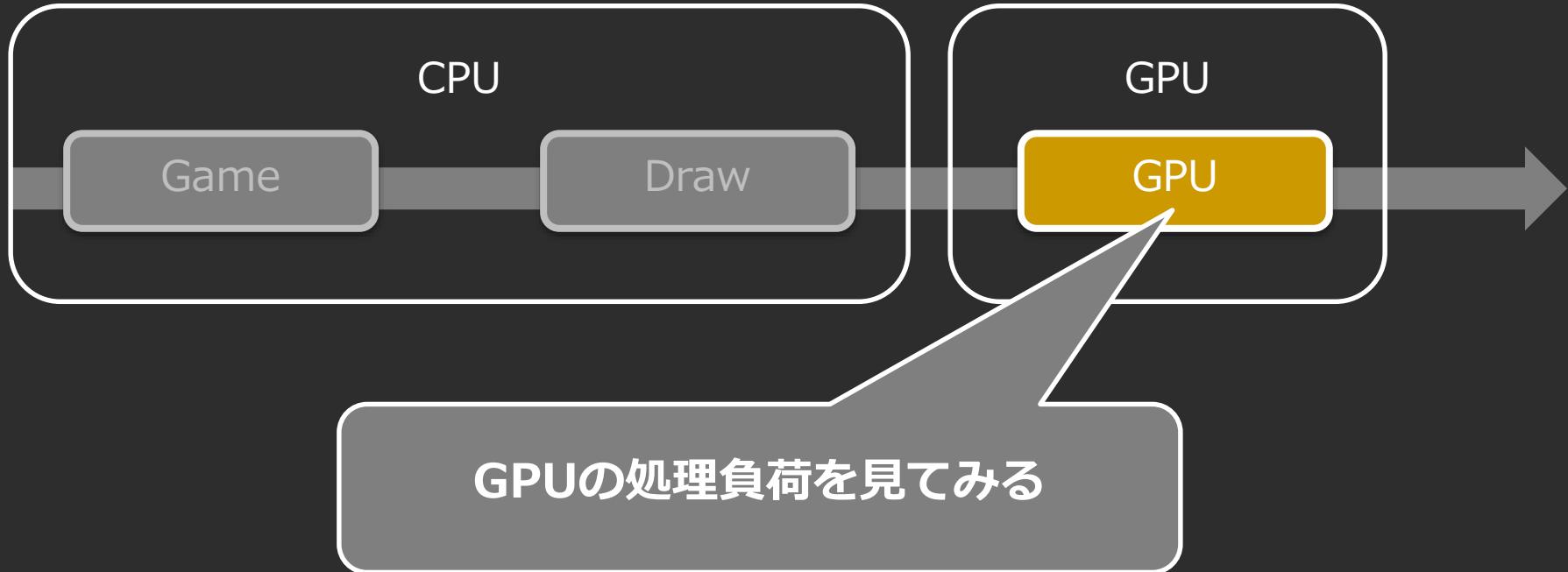


各処理が並行で進んでいるため、
Game/Draw/GPUの最大がそのフレームの処理時間になる。

この図はあくまで概要です。
実際のスレッドの流れは、プラットフォームやレンダリング手法で異なります。



本日はDrawとGPUのプロファイルがメイン

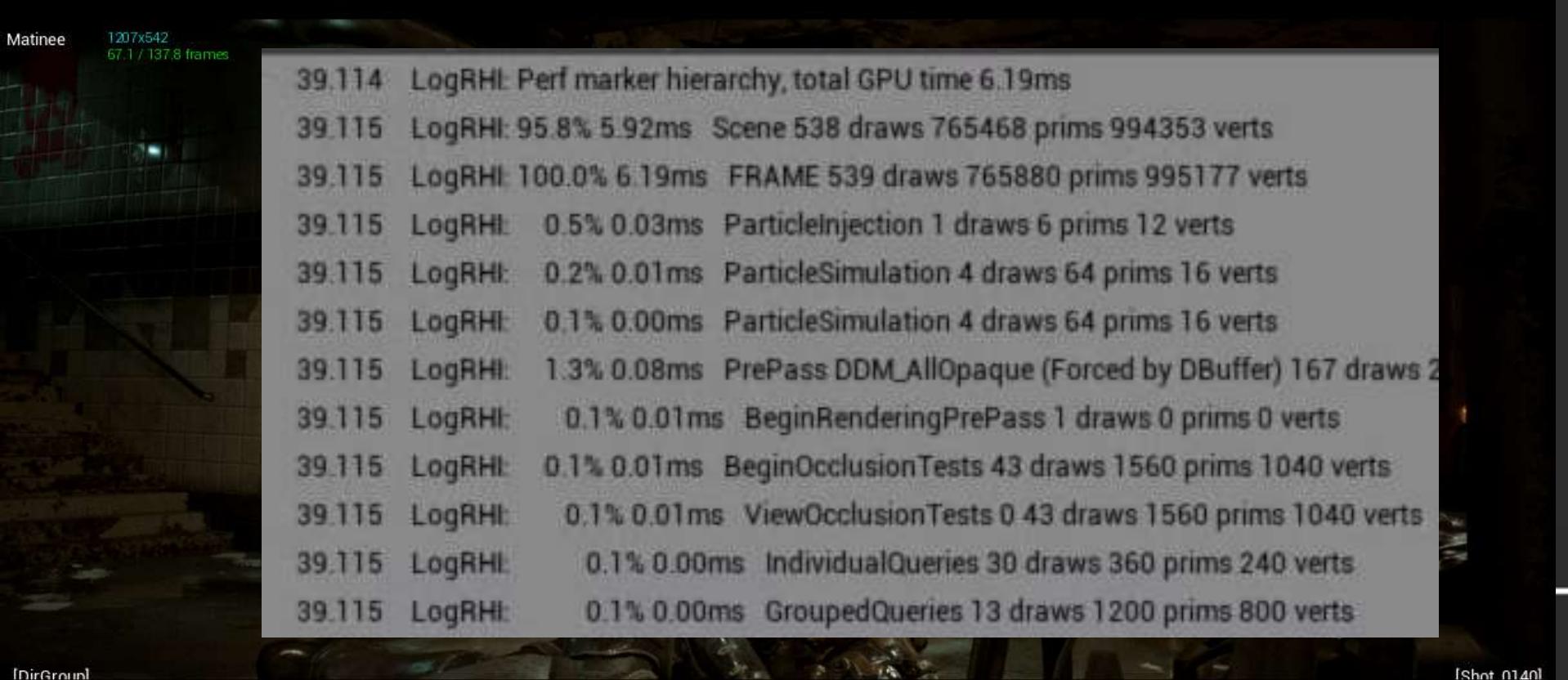


GPUの処理負荷を見てみる

GPUの処理負荷を階層的に出すConsole Command “ProfileGPU” (Default shortcut: “Ctrl + Shift + ,”)



コンソールなどの実機では、
ウインドウは出さずに出力ウィンドウにテキストで表示できます。



Tips (Advanced)

BasePass内部で、各メッシュ単位の負荷を出さない機種もある

```
'BasePass
  ▲ Static EBasePassDrawListType=1
    DefaultTextMaterial Opaque None
  ▲ Static EBasePassDrawListType=0
    M_Sky_Panning_Clouds2 SM_Sky
    RampMaterial Ramp_StaticMesh
    RampMaterial Linear_Stair_Static
    RampMaterial LeftArm_StaticMesh
    RampMaterial RightArm_StaticMesh
    RampMaterial Bump_StaticMesh
```

Editor

BasePass	111	draws	24
Dynamic	8	draws	89
Other Children			

Console



Tips (Advanced)

実機でBasePassのメッシュ単位の処理負荷を出力する方法

- **r.ShowMaterialDrawEvents**

- ただし、RHIThreadがOffじゃないと動作しない。

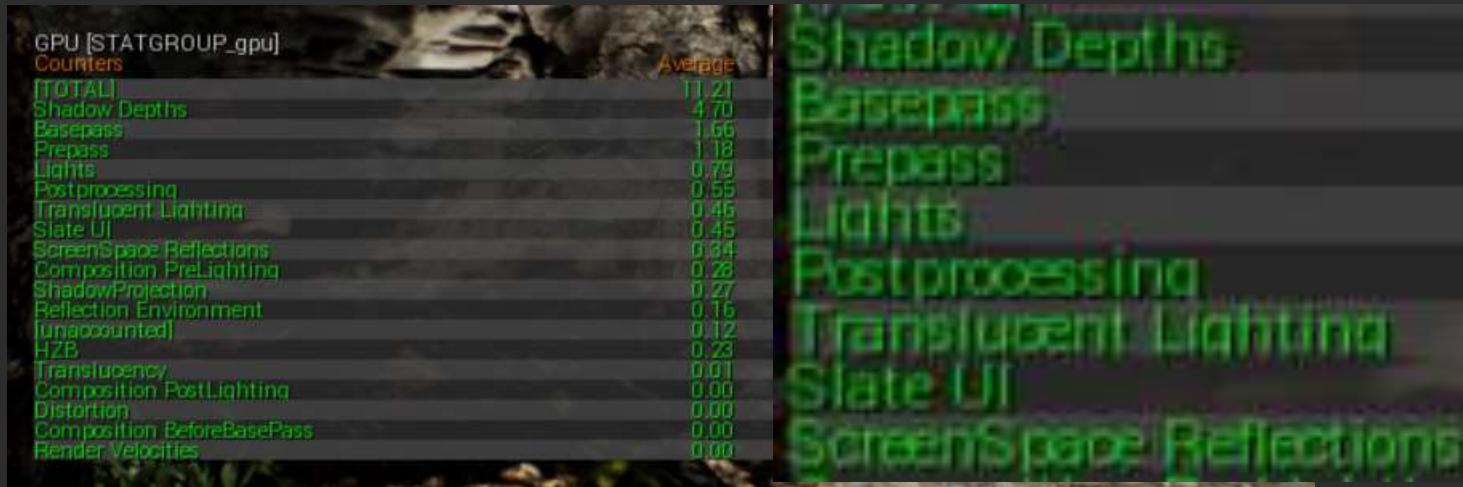
- **r.RHISetGPUCaptureOptions**

- 一括設定コマンド
 - 自動でRHIThreadをOffに

- ※DrawThreadの負荷が跳ね上がります。

- 計測するときにOnにして、それ以外ではOffに。

Console Command Stat GPU



ProfileGPUのリアルタイム表示版

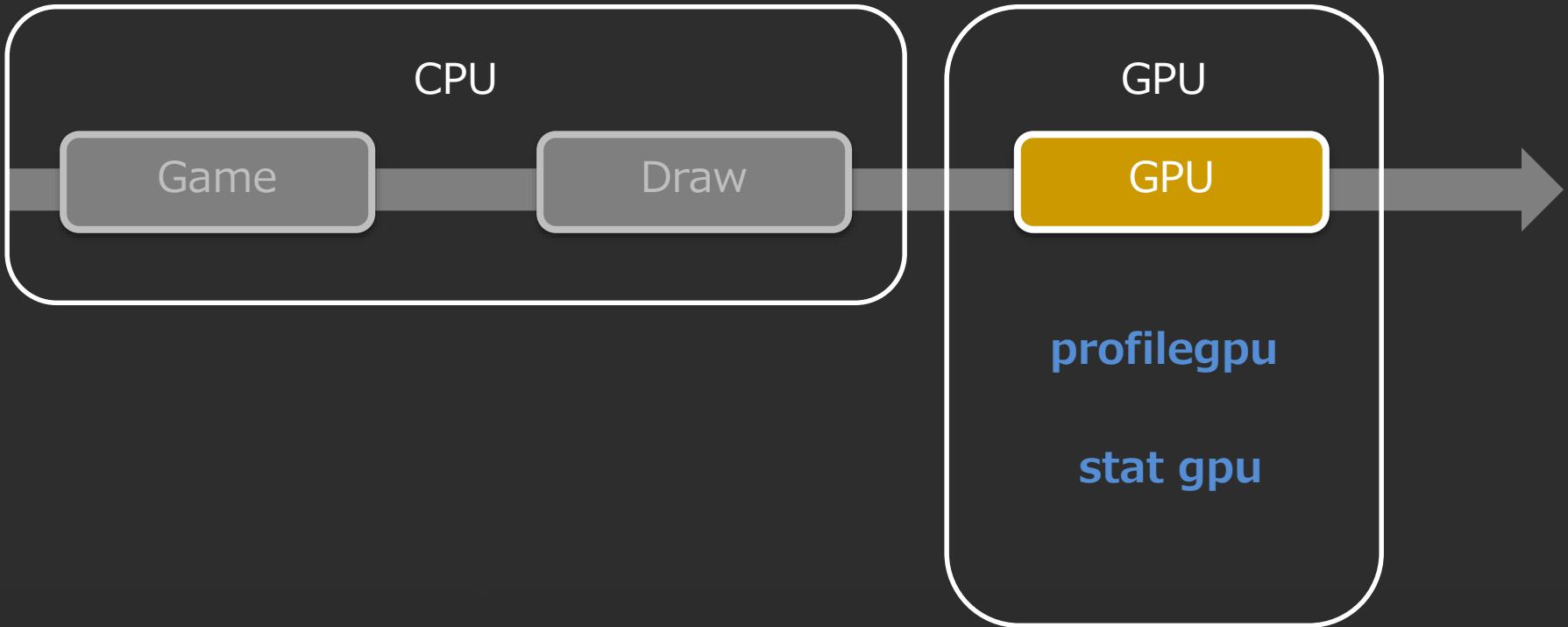
UNREAL ENGINE

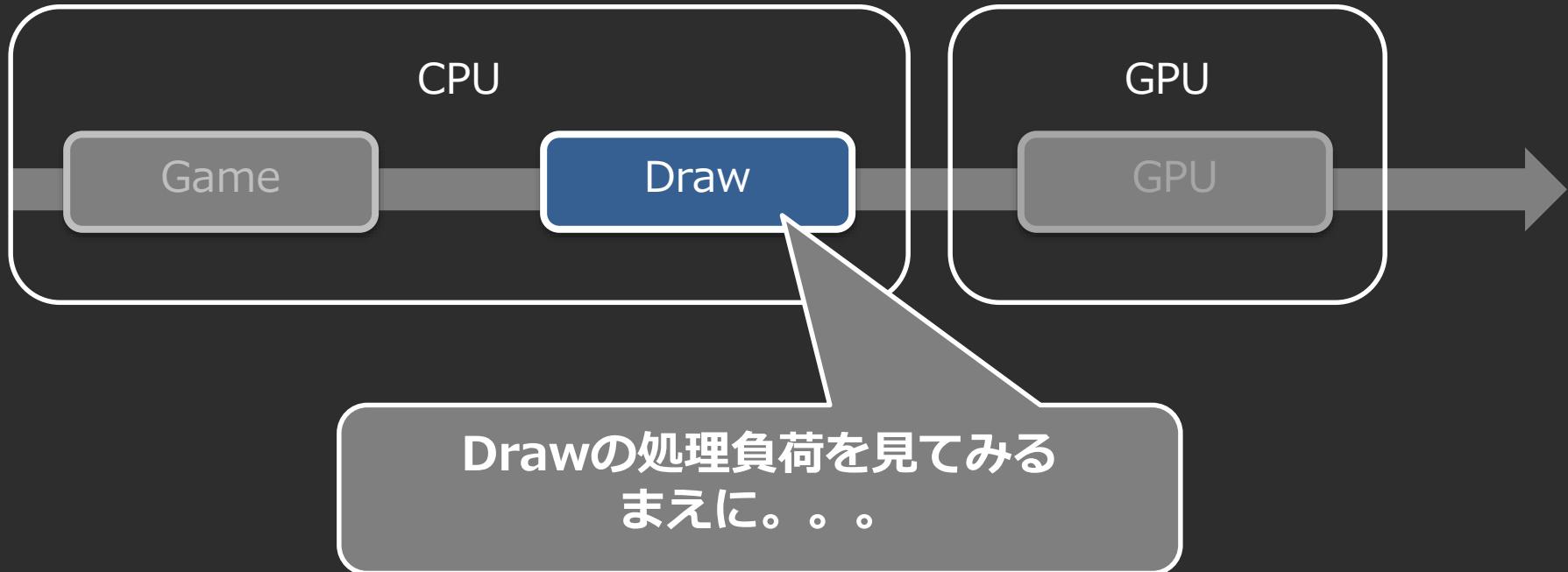
Unreal Engine 4 の
レンダリングフロー総おさらい

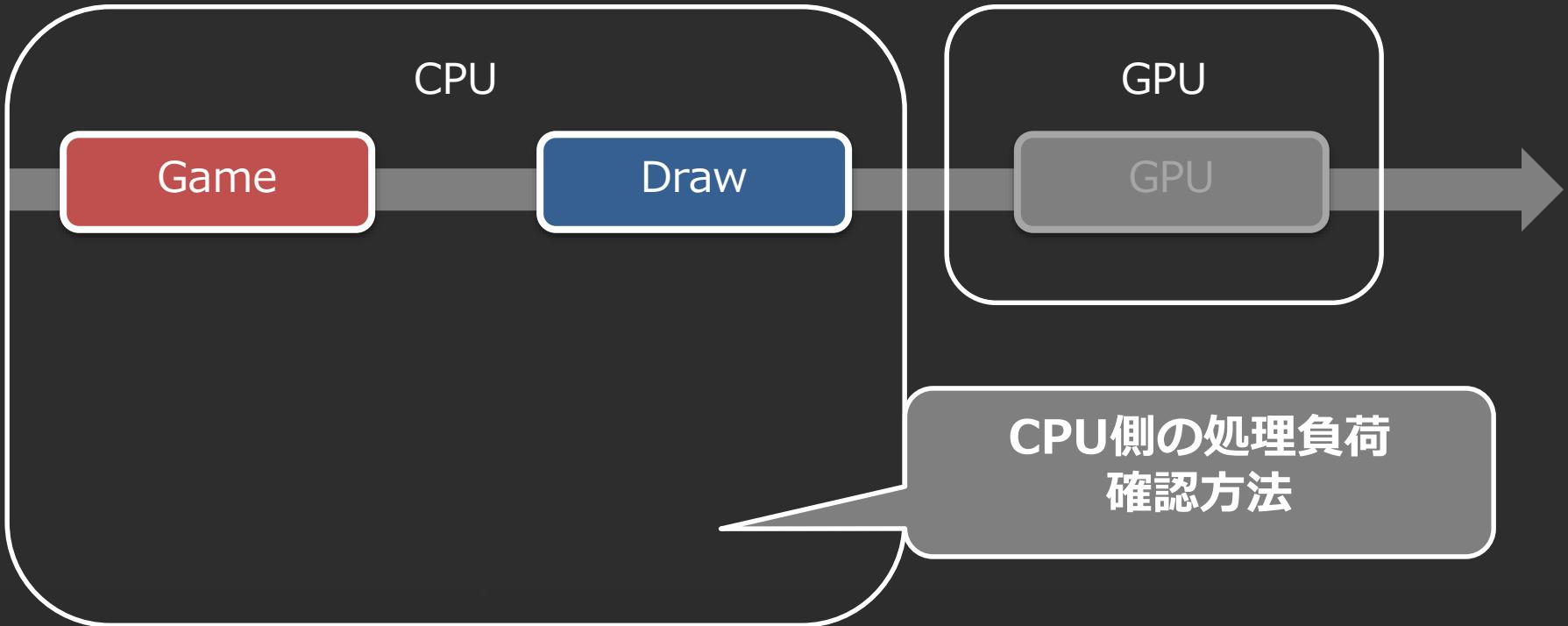
Epic Games Japan
Senior Support Engineer
篠山 範明

5

レンダリングの流れは去年のCEDECで講演させていただきました
SlideShareにUpしてあるので、良ければご参考にしてください







ConsoleCommand Stat dumpframe

1フレームのCPU処理フローの階層表示

```
Cmd: stat dumpframe root=renderthread ms=0.2
LogStats: Single Frame 122297 -----
LogStats: Culled to 0.200000ms, use -ms=0, for all data and aggregates.
LogStats: Stack -----
LogStats: 16.605ms ( 36) - Thread_3a38_0 - RenderThread - STATGROUP_Threads - STATCAT_Advanced
LogStats: 10.177ms (  7) - CPU Stall - Wait For Event - STAT_EventWait - STATGROUP_CPUStalls - STATCAT_Advanced
LogStats: 5.109ms (  1) - FDrawSceneCommand - STATGROUP_RenderThreadCommands - STATCAT_Advanced
LogStats: 5.079ms (  1) - RenderViewFamily - STAT_TotalSceneRenderingTime - STATGROUP_SceneRendering - STATCAT_Advanced
LogStats: 1.577ms (  1) - DeferredShadingSceneRenderer Lighting - STAT_FDeferredShadingSceneRenderer_Lighting - STATGROUP_SceneRendering
LogStats: 1.353ms (  1) - Lighting drawing - STAT_LightingDrawTime - STATGROUP_SceneRendering - STATCAT_Advanced
LogStats: 1.353ms (  1) - Render Lights and Shadows - STAT_LightRendering - STATGROUP_LightRendering - STATCAT_Advanced
LogStats: 0.354ms (  1) - SpotLightComponent/Game/Maps/ExampleProjectWelcome.ExampleProjectWelcome.PersistentLevel.SpotLightComponent - STAT_SpotLightComponent
LogStats: 0.246ms (  1) - Proj Shadow drawing - STAT_ProjectedShadowDrawTime - STATGROUP_SceneRendering - STATCAT_Advanced
LogStats: 0.229ms (  8) - PerObject Shadow Projections - STAT_RenderPerObjectShadowProjectionsTime - STATGROUP_ShadowProjections
LogStats: 0.215ms (  8) - Self
LogStats: 0.015ms ( 56) - OtherChildren
LogStats: 0.017ms (  3) - OtherChildren
LogStats: 0.107ms ( 10) - OtherChildren
LogStats: 0.305ms (  1) - PointLightComponent/Game/Maps/ExampleProjectWelcome.ExampleProjectWelcome.PersistentLevel.PointLightComponent
```



ConsoleCommand

Stat dumpframe

1フレームのCPU処理フローの階層表示

```
17.773ms ( 27) - Thread_12a6c_0 - RenderThread
 16.084ms (  1) - RenderViewFamily
   7.650ms (  1) - DeferredShadingSceneRenderer Lighting
   7.258ms (  1) - Lighting drawing
   2.104ms (  1) - Base pass drawing
   1.946ms (  1) - StaticDrawList drawing
   1.368ms (  1) - InitViews
```

例: Dumpされた部分のRenderThread部分



Stat dumpframe

オプションで見やすい形に調整可能

例: Drawの処理負荷を計測する方法

```
stat dumpframe -root=renderthread -depth=5 -ms=.1
```

どの処理から下を表示するか?

例:

Gamethread
Renderthread
Initviews

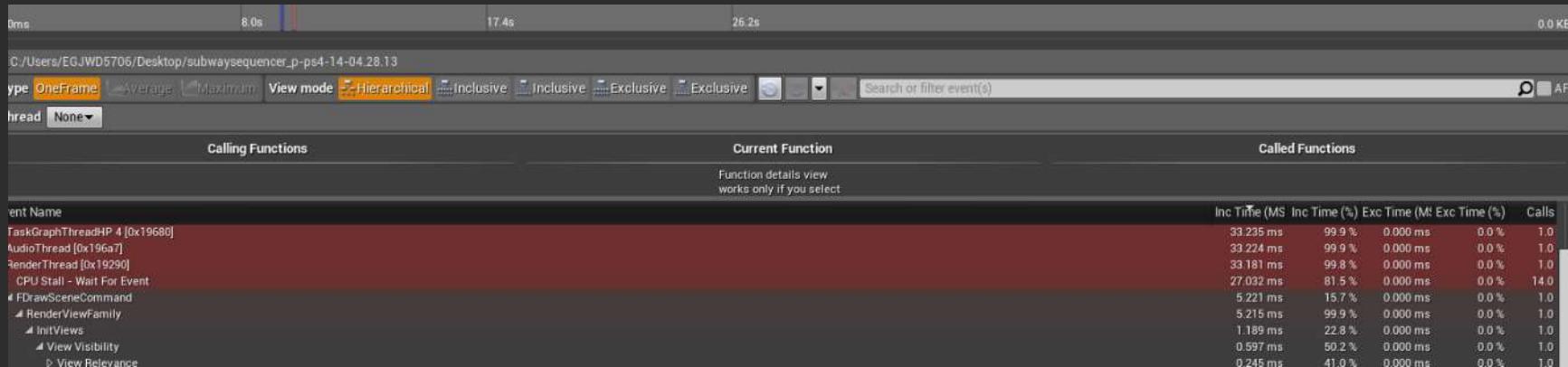
何階層深くまで表示するか

何ms以上の処理だけ記載するか



Console Command **stat startfile / stopfile**

複数フレームをキャプチャして、後々にエディタで処理負荷をチェックする



stat startfile / stopfile

使い方

```
LogStats: Wrote stats file: ../../Samples/ContentExamples/S  
LogBlueprintUserMessages: Early EndPlayMap Detection: Level '
```

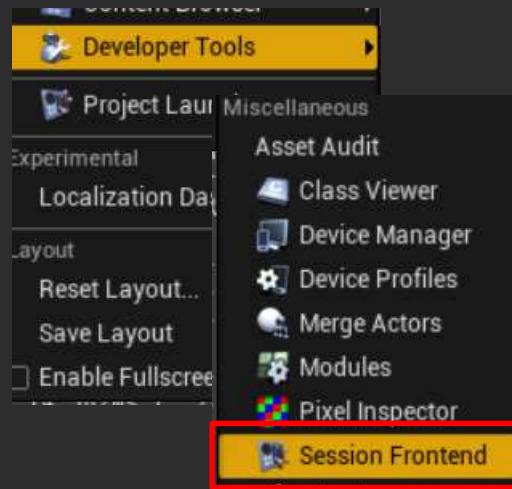
Stat stopfileと打ち込むと、プロファイルデータを出力する

書き出し先はログ(LogStats)に出る

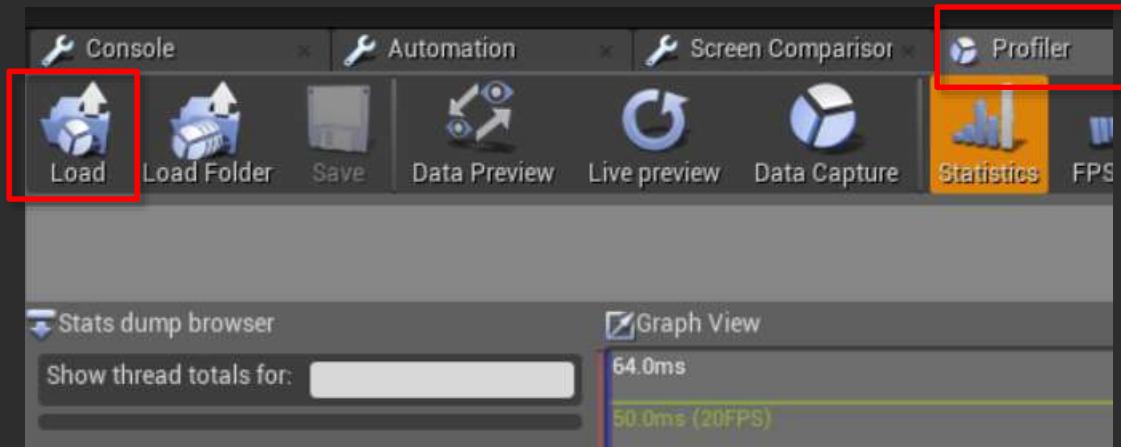


stat startfile / stopfile

使い方



Session Frontend
を立ち上げる



Profiler Tabからファイルを読み込む

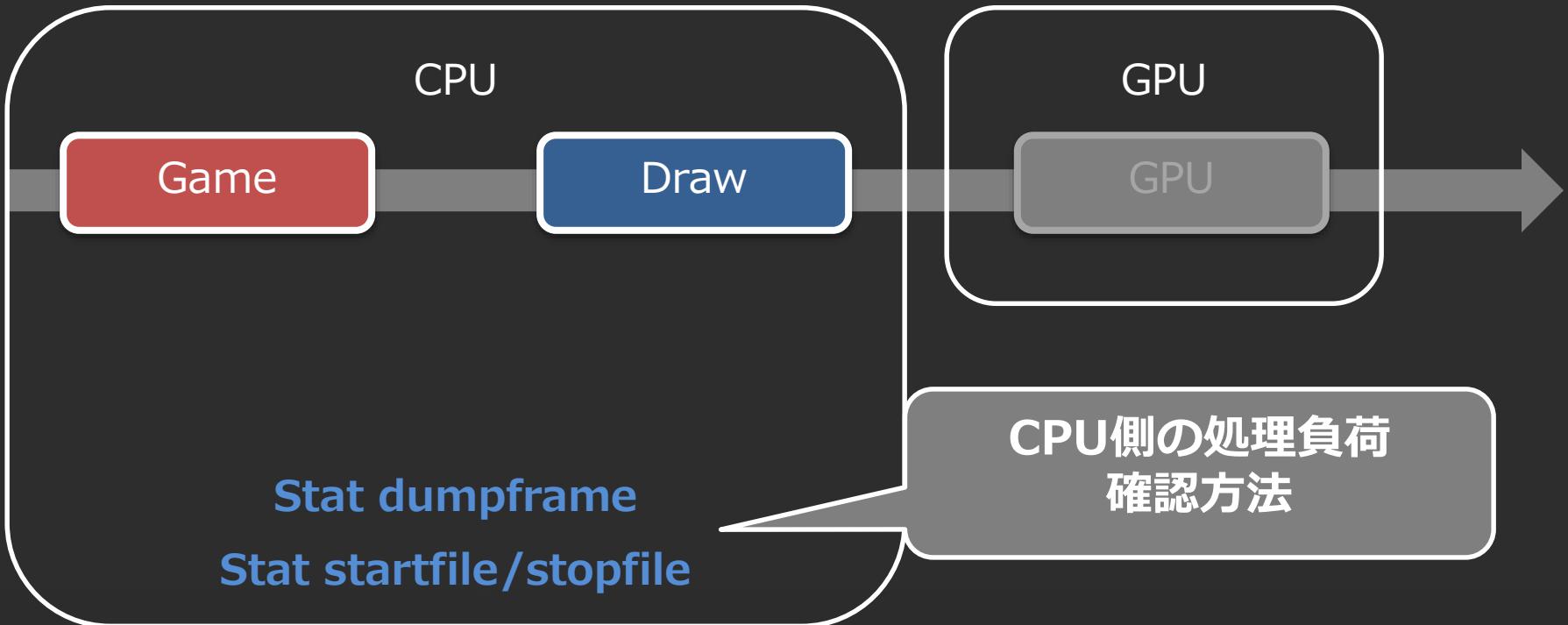
例えば、各アクターのTickがどれだけかかっているか？
などがわかるようになります。

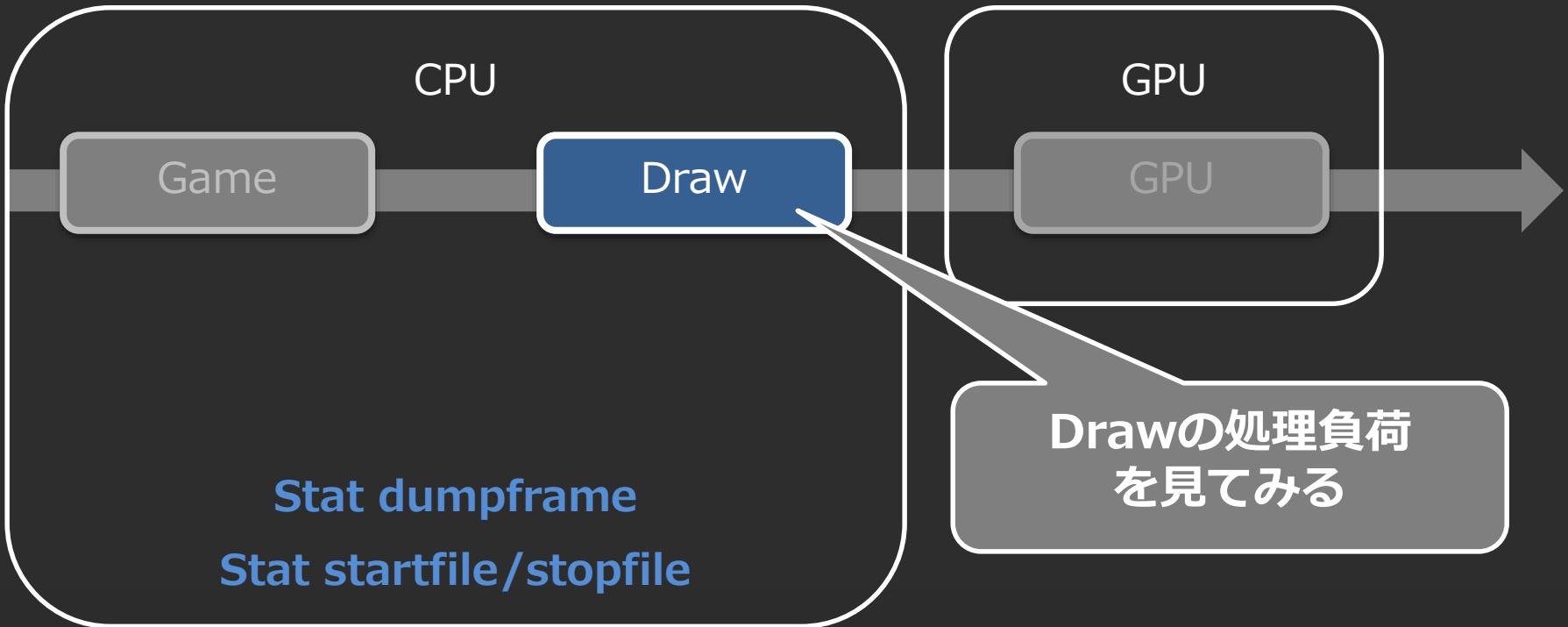
▲ GameEngine Tick	3.876 ms
▲ World Tick Time	2.475 ms
▲ Tick Time	2.201 ms
▲ TG_PrePhysics	1.370 ms
▲ ReleaseTickGroup Block	1.358 ms
▲ Game TaskGraph Tasks	1.344 ms
▲ FTickFunctionTask	1.094 ms
▷ LevelSequenceActor/Game/Maps/subwaySequencer_P.subwaySequencer_P.PersistentLevel.SubwaySequencerMASTER	0.716 ms
▷ PlayerController/Game/Maps/subwaySequencer_P.subwaySequencer_P.PersistentLevel.PlayerController	0.094 ms
▷ SkeletalMeshComponent/Game/Maps/subwaySequencer.subwaySequencer.PersistentLevel.TrooperA.SkeletalMeshComponent	0.087 ms
▷ TimelineComponent/Game/Maps/subwaySequencer.subwaySequencer.PersistentLevel.BP_Arcade_Animated_C.Tapping loop	0.055 ms

注意点:

処理順番ではありません。

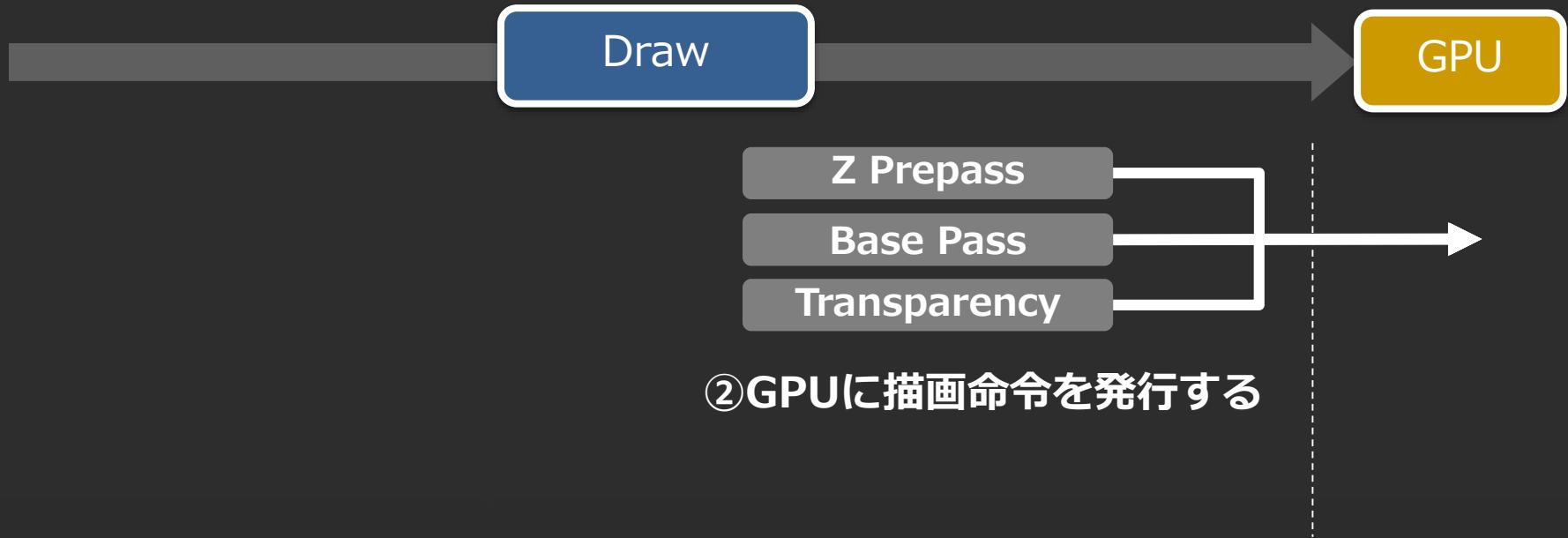
処理負荷順や名前順でソートを変えたりできます。



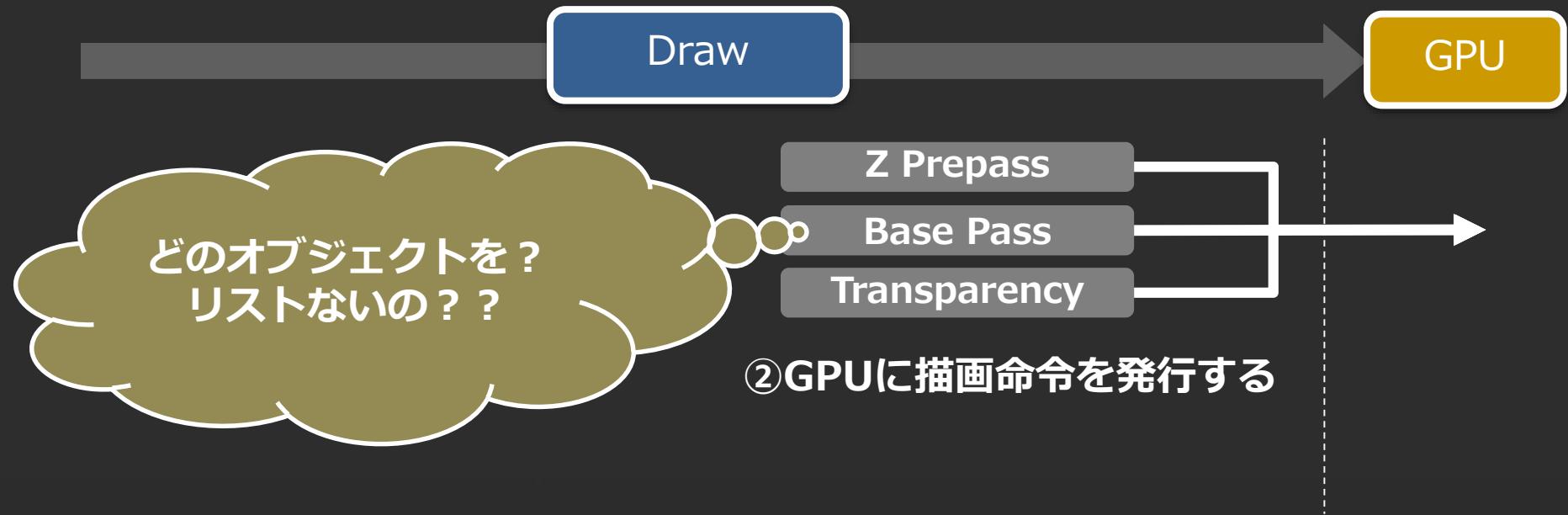


Drawの処理負荷を見てみる

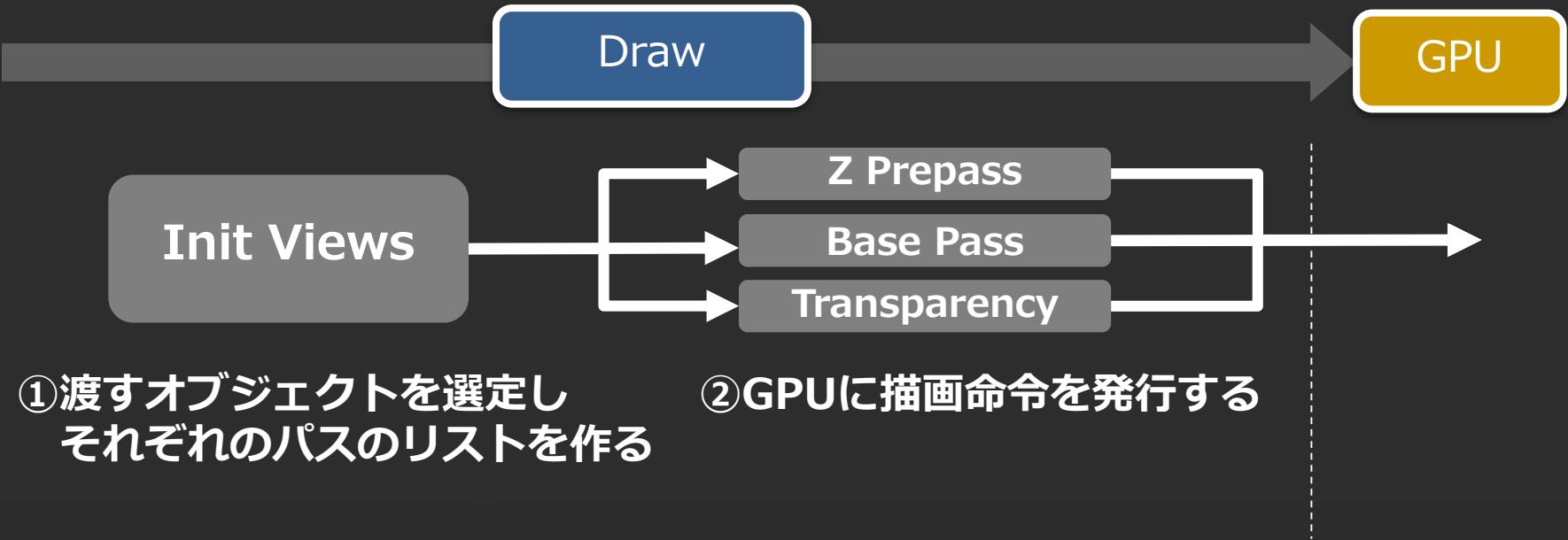
Drawの主な役割



Drawの主な役割



Drawの主な役割



InitViewsで行われるカリング

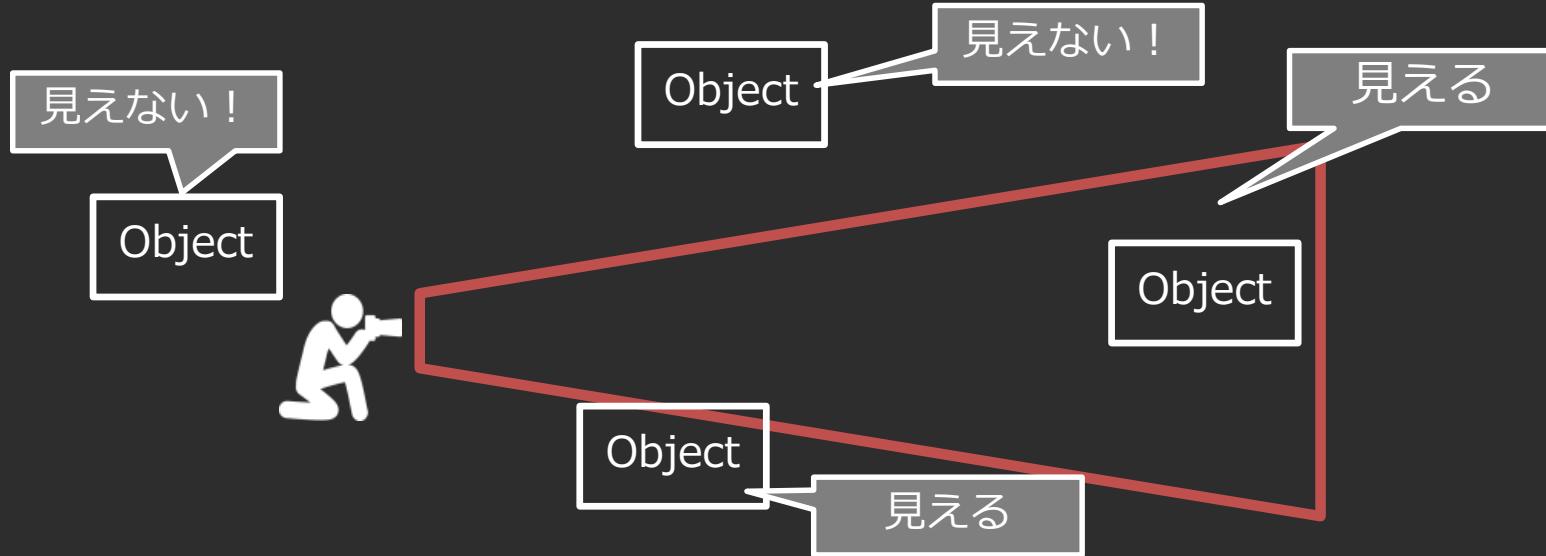
Init Views

①渡すオブジェクトを選定し
それぞれのパスのリストを作る

選定方法1: Frustum Culling

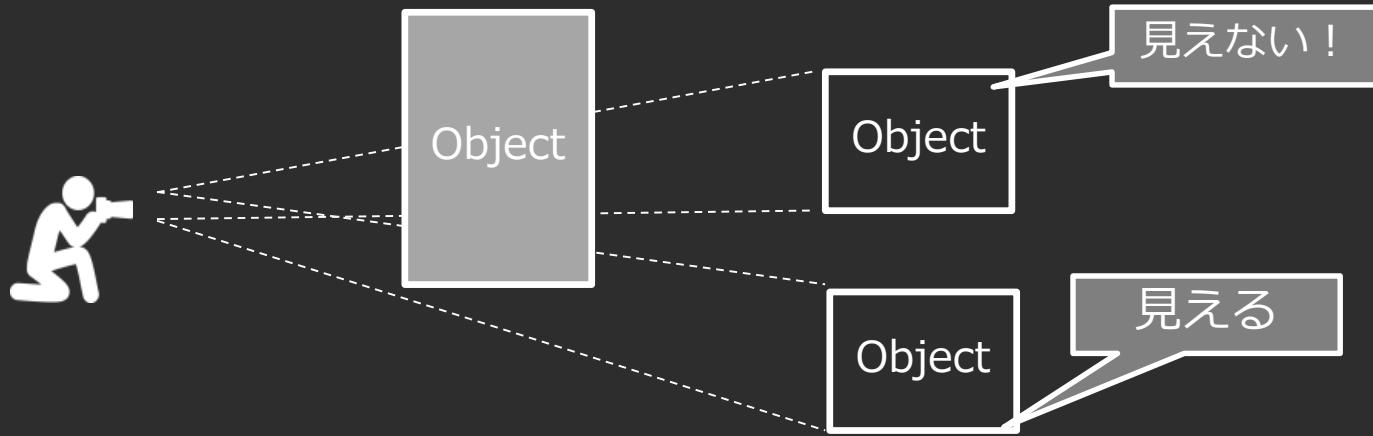
選定方法2: Occlusion Culling

(CPU) Frustum Culling

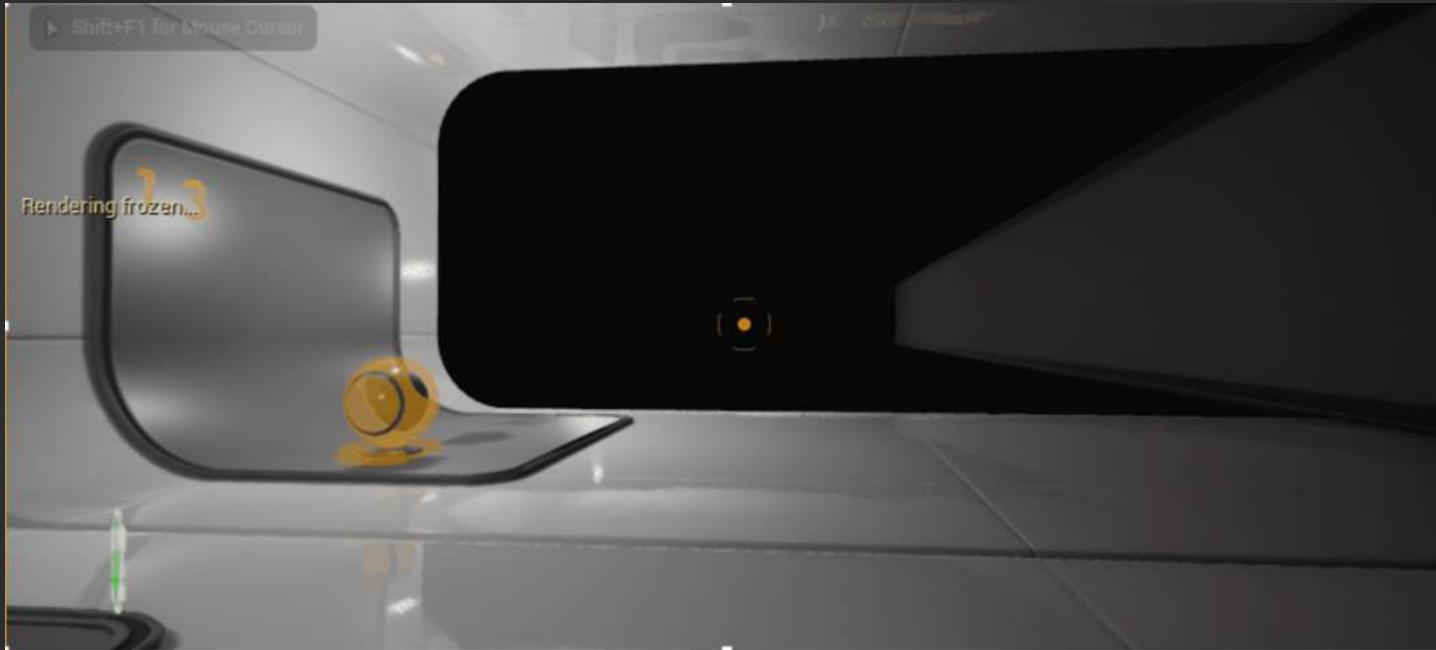


オブジェクトのバウンディングボックスを調べ、
画面外のオブジェクトをGPUに計算させない機能

Occlusion Culling

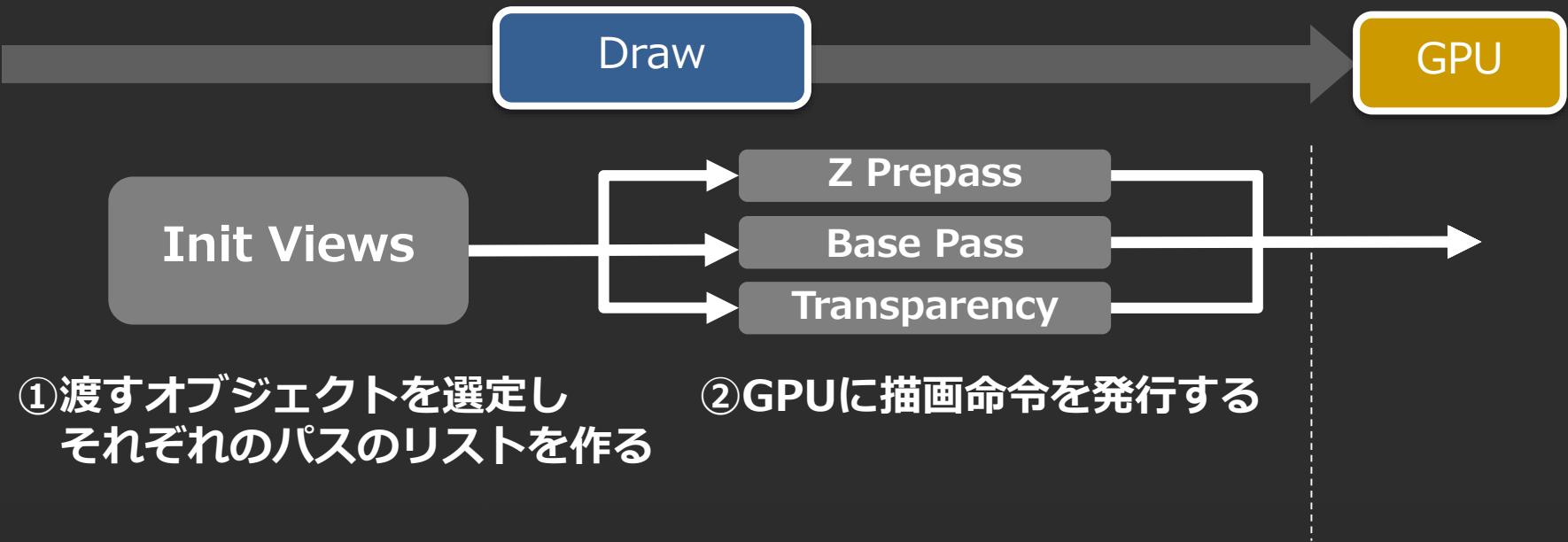


前面のオブジェクトに隠れて見えないオブジェクトを描画リストから外す
(1フレーム前のDepthを用いて行います)



FreezeRendering コマンドで、
その時点でのカメラのカーリング結果を止めることができる。

Drawの主な役割



Console Command **stat SceneRendering**

レンダリング(Draw)の全体の処理内訳をリアルタイムで表示

Category	CallCount	InclusiveAvg	InclusiveMax	ExclusiveAvg	ExclusiveMax
Scene Rendering [STATGROUP_SceneRendering]					
Cycle counters (flat)					
RenderViewFamily	1	4.51 ms	6.22 ms	0.15 ms	0.26 ms
InitViews	1	0.92 ms	1.80 ms	0.02 ms	0.03 ms
FinishRenderViewTarget	1	0.91 ms	1.09 ms	0.00 ms	0.01 ms
DeferredShadingSceneRenderer_Lighting	1	0.60 ms	0.91 ms	0.14 ms	0.22 ms
Lighting drawing	1	0.43 ms	0.69 ms	0.00 ms	0.00 ms
InitViewsPossiblyAfterPrepass	1	0.41 ms	0.88 ms	0.02 ms	0.03 ms
Dynamic shadow setup	1	0.32 ms	0.75 ms	0.12 ms	0.33 ms
Proj Shadow drawing	2	0.20 ms	0.34 ms	0.02 ms	0.04 ms
DeferredShadingSceneRenderer_AfterBasePass	1	0.21 ms	0.35 ms	0.18 ms	0.27 ms
Dynamic Primitive drawing	1	0.19 ms	0.52 ms	0.19 ms	0.52 ms

Stat scenerendeing

注意点

```
Scene Rendering [STATGROUP_SceneRendering]
Cycle counters (flat)
  RenderViewFamily
    InitViews
    FinishRenderViewTarget
    DeferredShadingSceneRenderer Lighting
```

(表示内容はフラットですが、) 処理内容は階層的です

RenderViewFamiliy

- InitViews
- BasePass drawing
 - StaticDrawList drawing
 - Dynamic Primitive drawing

Stat scenerendeing

Stat SceneRenderingで見える負荷はDrawの負荷です。
GPUの処理ではありません！

Draw が重たい場合は真っ先に参照してみてください

ConsoleCommand stat InitViews

Culling処理にかかった時間やプリミティブ数などをリアルタイムで確認

Init Views [STATGROUP_InitViews]

Cycle counters (flat)

View Visibility
PostInitViews FlushDel
GetDynamicMeshElements
View Relevance
Init dynamic shadows
Compute View Relevance
Occlusion Cull
Create WholeScene Shadow
Update Indirect Lighting Cache Prims
Static Mesh Relevance
Frustum Cull
Update Indirect Lighting Cache Finalize
Init Projected Shadow
Add View Whole Scene Shadows
Update Indirect Lighting Cache Blocks
Update Preshadow Cache
Update Fading
GatherShadowPrimitives
Decompress Occlusion
Update Indirect Lighting Cache Transitions

	CallCount	InclusiveAvg	InclusiveMax	ExclusiveAvg	ExclusiveMax
View Visibility	1	0.51 ms	0.84 ms	0.02 ms	0.03 ms
PostInitViews FlushDel	1	0.33 ms	0.42 ms	0.01 ms	0.01 ms
GetDynamicMeshElements	1	0.17 ms	0.27 ms	0.02 ms	0.05 ms
View Relevance	1	0.19 ms	0.35 ms	0.01 ms	0.01 ms
Init dynamic shadows	1	0.12 ms	0.30 ms	0.01 ms	0.01 ms
Compute View Relevance	2	0.11 ms	0.26 ms	0.11 ms	0.26 ms
Occlusion Cull	1	0.10 ms	0.21 ms	0.00 ms	0.00 ms
Create WholeScene Shadow	3	0.06 ms	0.24 ms	0.05 ms	0.17 ms
Update Indirect Lighting Cache Prims	1	0.07 ms	0.19 ms	0.07 ms	0.19 ms
Static Mesh Relevance	2	0.06 ms	0.17 ms	0.06 ms	0.17 ms
Frustum Cull	1	0.03 ms	0.09 ms	0.00 ms	0.06 ms
Update Indirect Lighting Cache Finalize	1	0.02 ms	0.09 ms	0.01 ms	0.04 ms
Init Projected Shadow	1	0.01 ms	0.01 ms	0.01 ms	0.01 ms
Add View Whole Scene Shadows	3	0.01 ms	0.01 ms	0.01 ms	0.01 ms
Update Indirect Lighting Cache Blocks	1	0.00 ms	0.07 ms	0.00 ms	0.07 ms
Update Preshadow Cache	1	0.00 ms	0.01 ms	0.00 ms	0.01 ms
Update Fading	1	0.00 ms	0.00 ms	0.00 ms	0.00 ms
GatherShadowPrimitives	1	0.00 ms	0.20 ms	0.00 ms	0.01 ms
Decompress Occlusion	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



ConsoleCommand **stat InitViews**

- InitViewsは何の処理をしている？？
 - 画面に見えないオブジェクトを省く処理
 - Frustum Culling
 - Occlusion Culling
 - レンダリングオブジェクトをソートしたり
 - シャドウ計算に寄与するオブジェクトもここで算出している

stat InitViews でよく見るところ

Init Views [STATGROUP_InitViews]	
Cycle counters (flat)	
View Visibility	
PostInitViews FlushDel	
GetDynamicMeshElements	
View Relevance	
Init dynamic shadows	
Compute View Relevance	
Occlusion Cull	
Create WholeScene Shadow	
Update Indirect Lighting Cache Prims	
Static Mesh Relevance	
Frustum Cull	
Update Indirect Lighting Cache Finalize	
Init Projected Shadow	
Add View Whole Scene Shadows	
Update Indirect Lighting Cache Blocks	

処理時間

Frustum Cull

Occlusion Cull

Processed Primitives: 投入された全オブジェクト数

Frustum Culled primitives: カリングされたオブジェクト数

Occluded Culled Primitives: オクルージョンカリングされた Primitive数

Counters	
Processed primitives	
Frustum Culled primitives	
Visible static mesh elements	
Occluded primitives	
Occlusion queries	
Visible dynamic primitives	
Indirect Lighting Cache updates	

生き残ったPrimitiveたち

Visible Static Mesh Elements

Visible Dynamic Mesh Elements

Stat initviews

Culling処理はCPU負荷をあげますが、
GPU負荷を下げるために必ず必要な処理です。

Culling負荷が高い場合、遠くのオブジェクトや目に見えないオブジェクト
を自前でVisibilityをOffにしたりなどが効果があります。

また、不要なオブジェクトがGPUに投入されていないか確認するために、
FreezeRenderingコマンドも用いて下さい。

備考:

Precomputed Visibilityについて

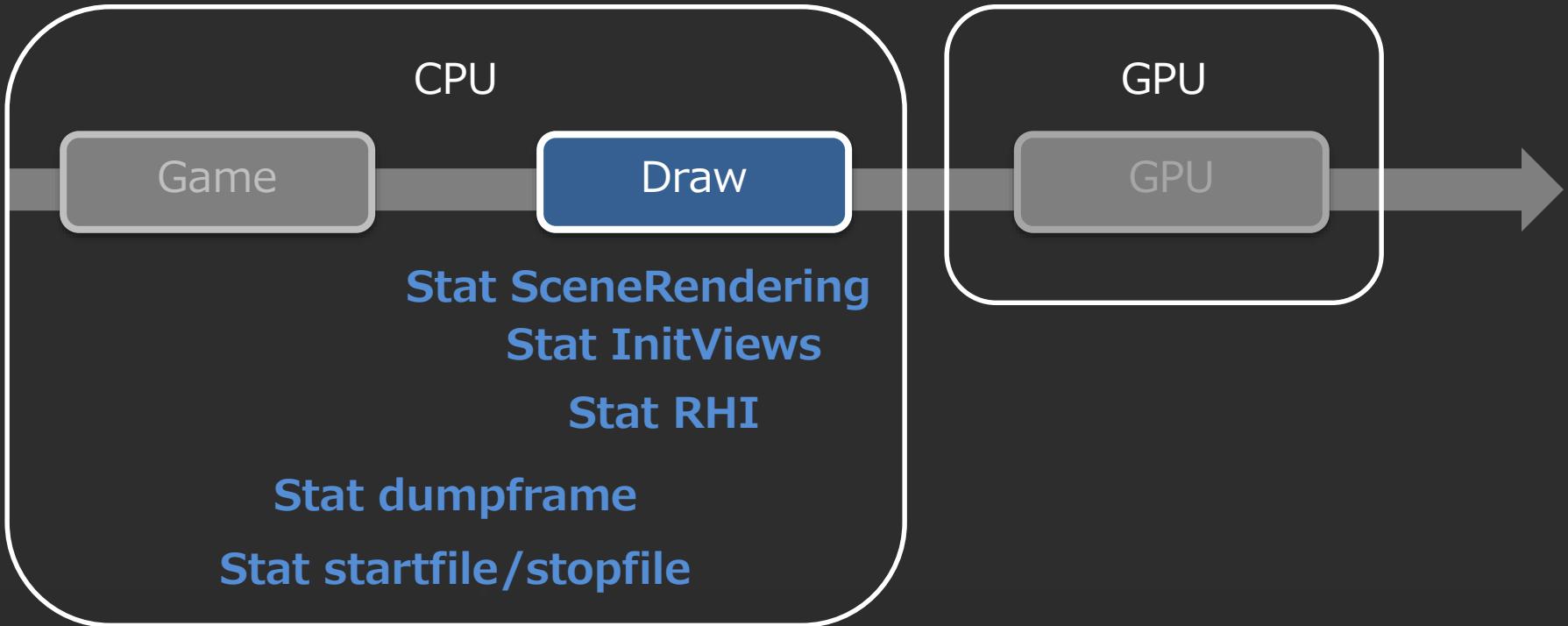
- Precomputed VisibilityはStaticなオブジェクトに対して、シーンのある視点から見えるオブジェクトを事前計算しリストを作成しておく手法です。
- メモリやデータサイズが増えますが、InitViewsの負荷削減につながりますので、静的オブジェクトが多い方は参考にしてください。
- 英語ですが、以下のスレッドに使い方がとても丁寧に説明されています。
 - <http://timhobsonue4.snappages.com/culling-precomputed-visibility-volumes>



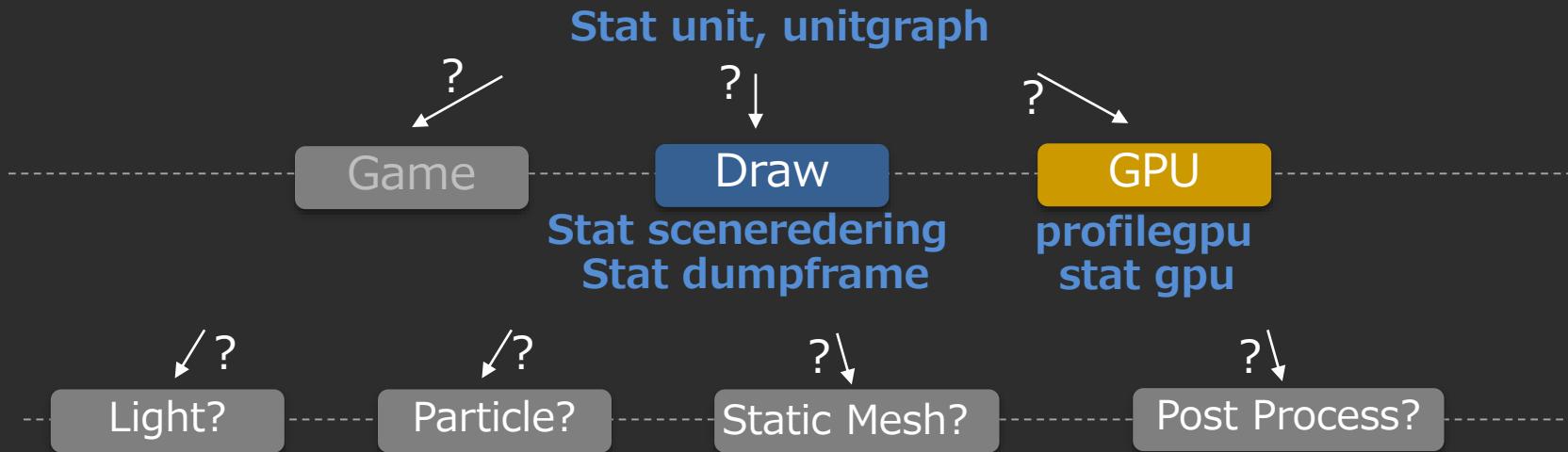
備考: 全体のDrawCall数を把握したい場合 Stat RHI



“Draw primitive Calls”でシーン内の全DrawCall数を確認



ここまででどの箇所が重たいのかだいたいわかるようになってきた



次は、なぜその処理が重たいのだろう？？

各アセットやレンダリング手法の詳細

各処理内容に特化したStat



Stat XXXXXX

100種類以上の項目がある

各処理内容専用のコマンドや変数

[680 more matches]

```
r.AmbientOcclusionMaxQuality  
r.AmbientOcclusionLevels  
r.AmbientOcclusion.FadeRadiusScale  
r.AmbientOcclusion.Compute  
r.AmbientOcclusion.AsyncComputeBudget  
r.AmbientOcclusionSphereForErosionCull
```

[25 more matches]

```
r.Shadow.MaxCSMResolution  
r.Shadow.FreezeCamera  
r.Shadow.ForceSingleSampleShadowingFromStationary  
r.Shadow.FilterMethod  
r.Shadow.FadeResolution  
r.Shadow.FadeExponent  
r.Shadow.FadeMethod
```

レンダリング関連の設定コマンド

r.XXXXXX

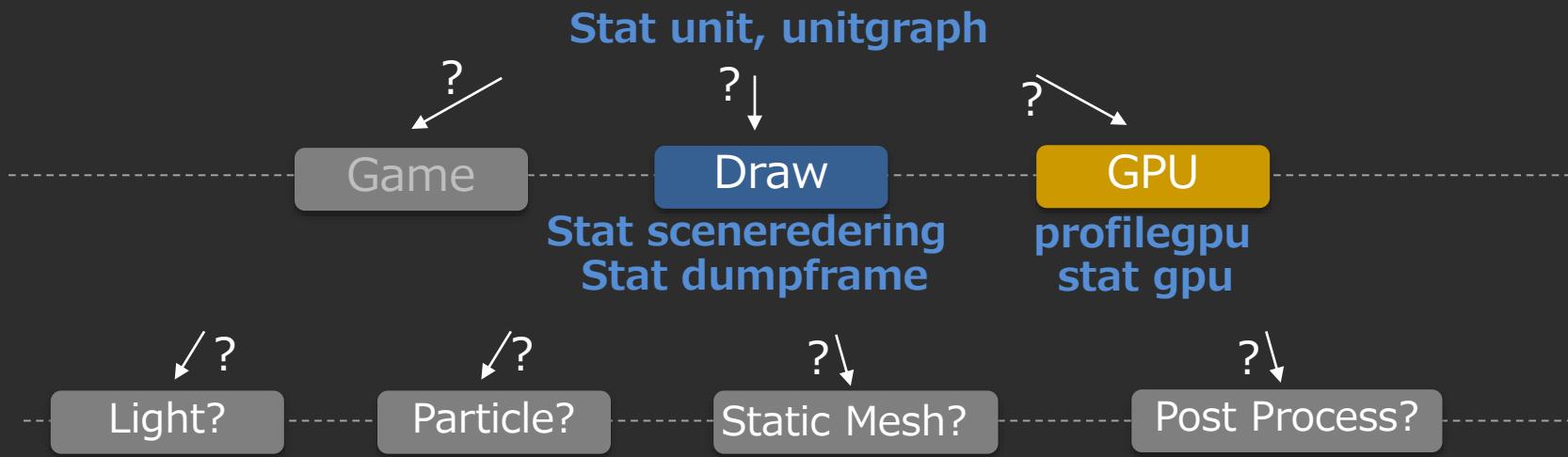
700種類以上

その中の影の設定コマンド

r.shadow.XXXXXX

50種類以上





各処理の専用コマンドで理由を調査

stat LightRendering, Foliage, Particle,

r.XXXXX,

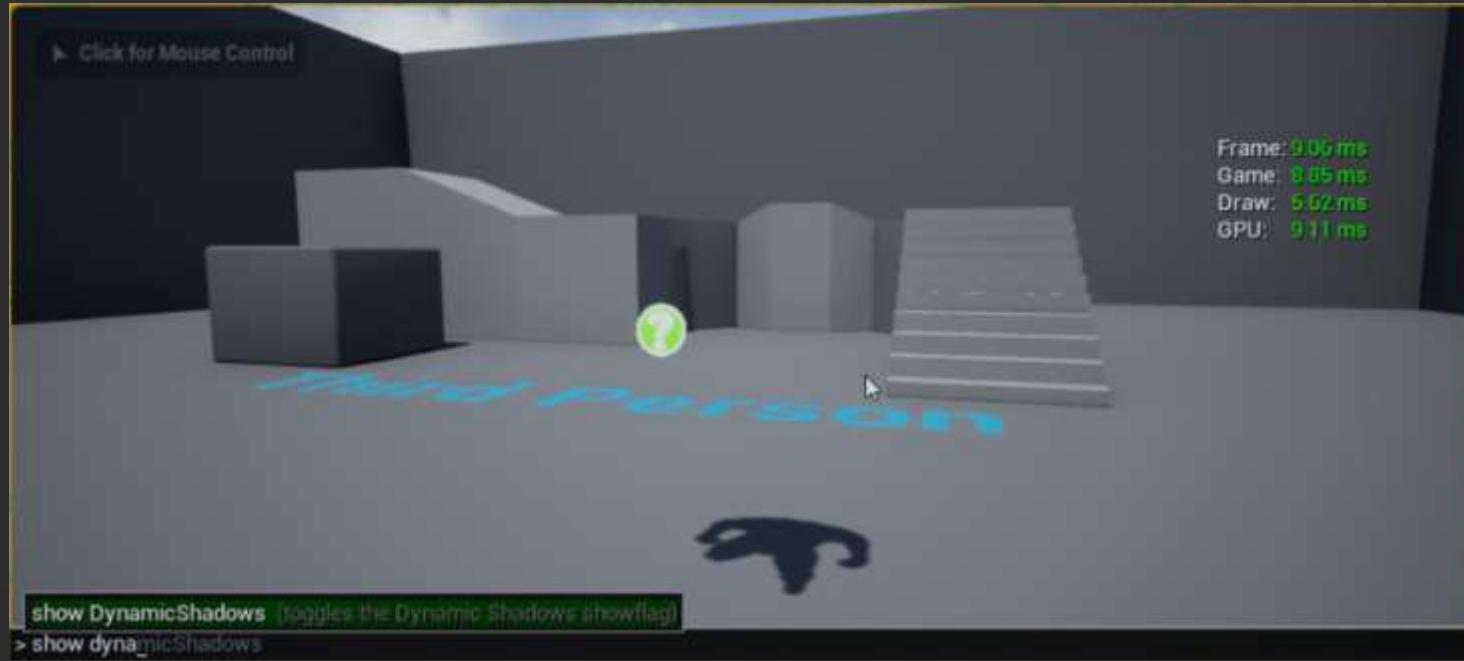
Tips

プロファイリング補助コマンド

Console Command

Show

オブジェクトや機能のOn/Off



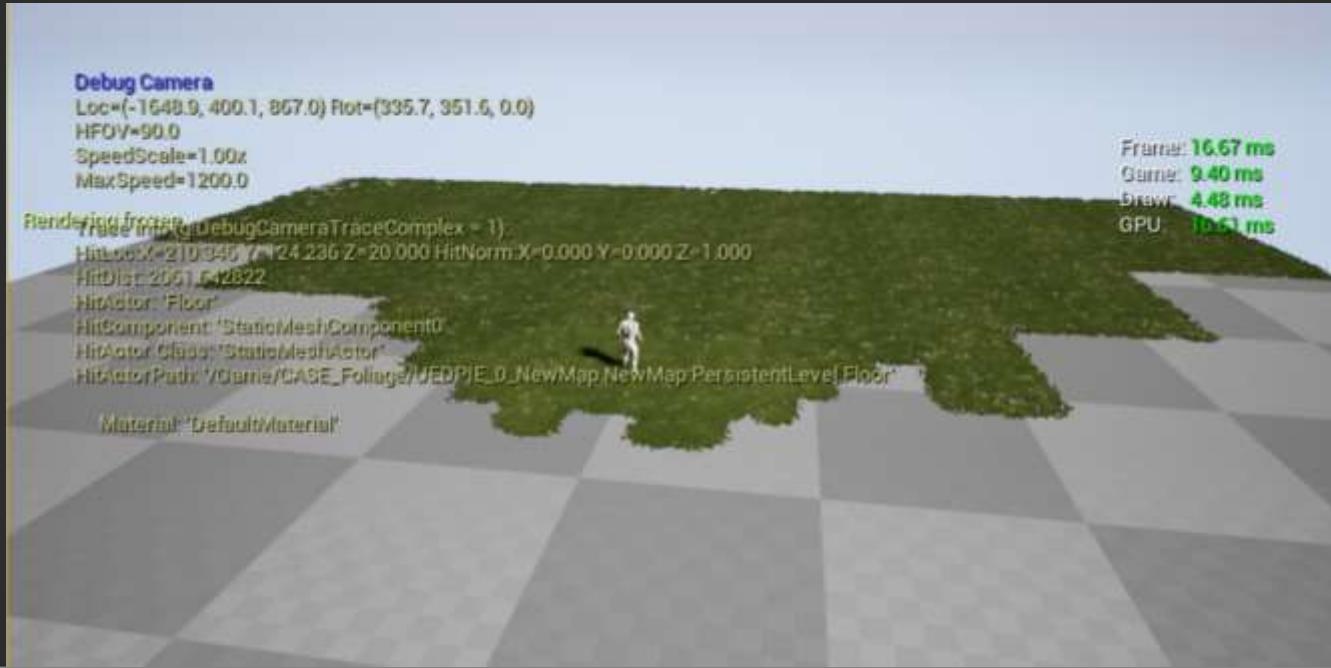
Console Command **FreezeRendering**

現時点のカメラのカーリングをフリーズする



Console Command **ToggleDebugCamera**

プレイヤーのカメラから離れ、自由に視点を変えられる



プロファイリングツール1 まとめ

グラフィクスプロファイル全体の流れ

Step.1
Game? Draw? GPU?

Stat unit / stat unit graph

Game

Draw

GPU

Step.2
どの部分が重たい？

Stat

Scenerendering
Initviews, rhi
Dumpframe, startfile

profilegpu
stat gpu

Step.3
各処理のコマンドを
使って理由を調査

Light?

Particle?

Static Mesh?

Post Process?

stat LightRendering, Foliage,,,

r.XXXXXX,
etc.

補助コマンド

show, freezeframe, toggledebugcamera, etc.

なぜ、Console Commandから？？？

実機確認できるから！

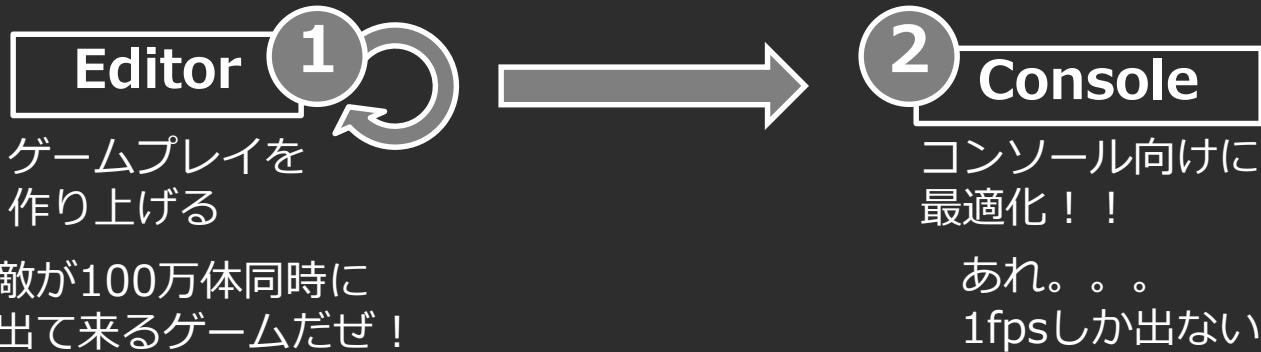
実機で確認しなきゃだめ？

本講演の目次

1. はじめに
2. 基本のプロファイリングツール 1
- 3. プロファイルの流れ**
4. 基本のプロファイリングツール 2
5. Showcase
6. まとめ&おまけ

プロファイルの流れ

✗ オススメしない運用 PCでゲームデザインしてから、コンソールで最適化



問題点

- ・ 弱: 修正するアセットが大量になる。
- ・ 強: 実機では実現不可能

対策1: 仕様を満たすために著しくクオリティを下げる

対策2: ゲームの仕様を再考する

○ オススメする運用 PCでゲームデザイン&実機プロファイリングを 並行して行う



利点

- ・ 修正アセット（手戻り）の削減
- ・ 実現可能なゲームデザイン

全体で見れば、製品開発スピードは前者よりも速くなります。(断言)

✗ 備考: (おすすめしない運用2)

コンソールと同程度のスペックのPCによる開発

残念ながら全然あてになりません。

UE4のコードが全然違う！。

SDK APIがそもそも違う

OS, ドライバがそもそも違う

ハードウェア構成が厳密には違う

同スペックのPCとコンソールの違い

各機種専用の
最適化を行っている。

プロファイルの流れ まとめ

なぜプロファイル？（再掲）

最適化の前に、開発初期から、日常的に、
実機で
プロファイルする癖を

備考 だけど 重要 各ビルドコンフィギュレーションによる設定の差

	CPU	Stat (CVars)	GPU
Development	検証コードあり	全機能使える	
Test		Stat unit ぐらい	ほぼ同じ
Shipping	検証コードなし	無効	

各ビルドコンフィギュレーションによる設定の差

CPU本来の負荷は Test ビルドで計測するようにしてください。

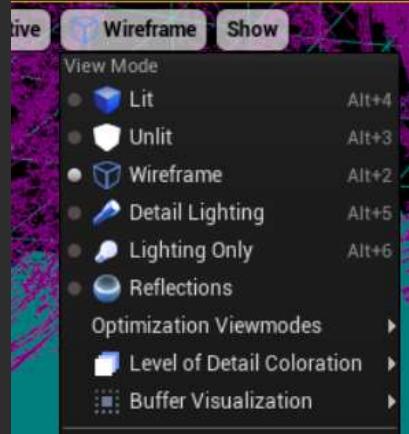
日頃 Development ビルドで計測し、重たい部分などは実際の Test ビルドで。
どれだけの差が出るかを定期的に調べて下さい。

本講演の目次

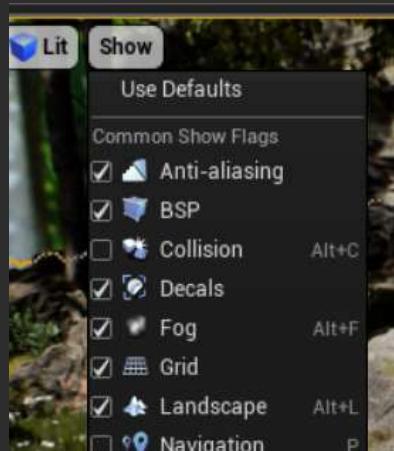
1. はじめに
2. 基本のプロファイリングツール 1
3. プロファイルの流れ
- 4. 基本のプロファイリングツール 2**
5. Showcase
6. まとめ&おまけ

Editorのデバッグ機能 基本のプロファイリングツール 2

今回紹介する3つのEditor機能



Viewmode



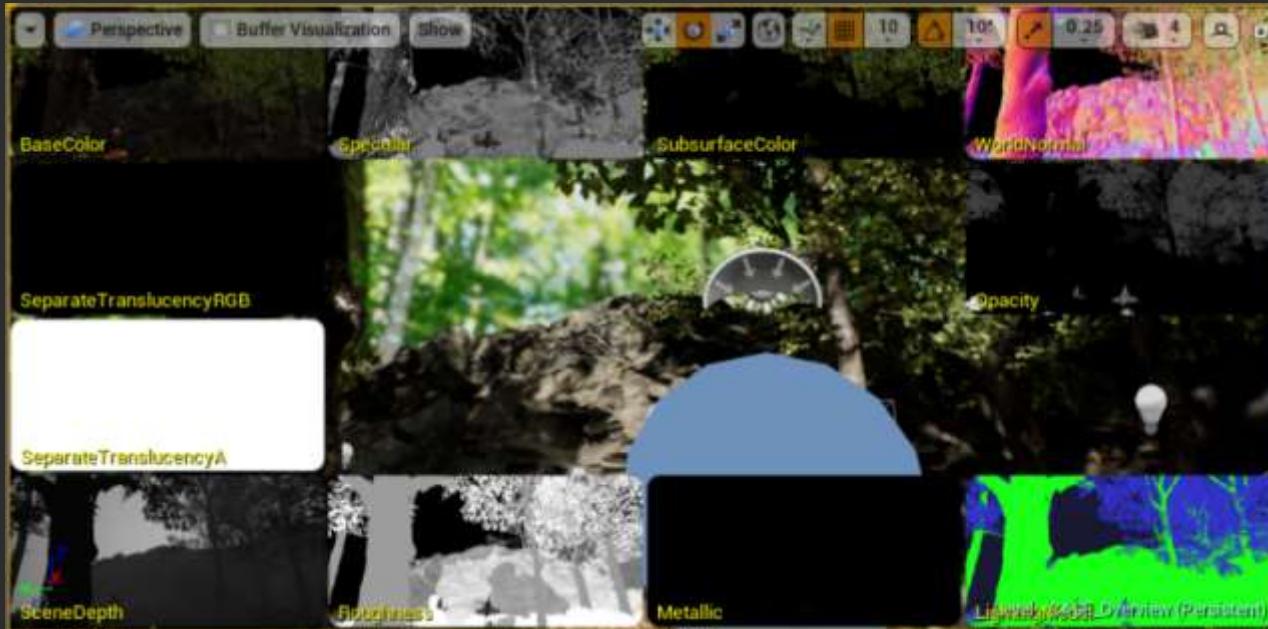
Show

Object	Actor(s)	Type	Count	Inst Se
HillTree_02	4 Actors	StaticMesh	4	16
SM_Cliff01	2 Actors	StaticMesh	2	2
SM_GroundRevealR8	8 Actors	StaticMesh	8	8
LargeVolcanicRock_2	2 Actors	StaticMesh	2	2

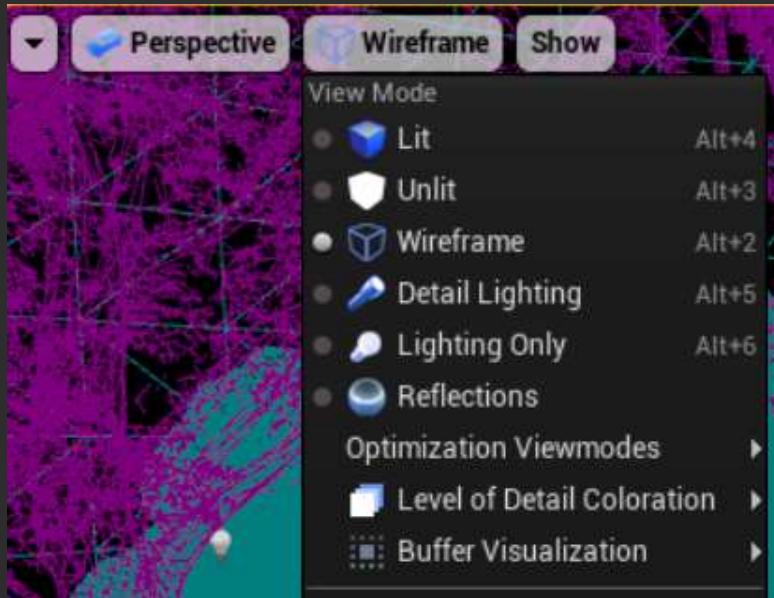
Statistics

ViewMode

中間バッファやデバッグ用表示へ切り替える機能



ViewMode

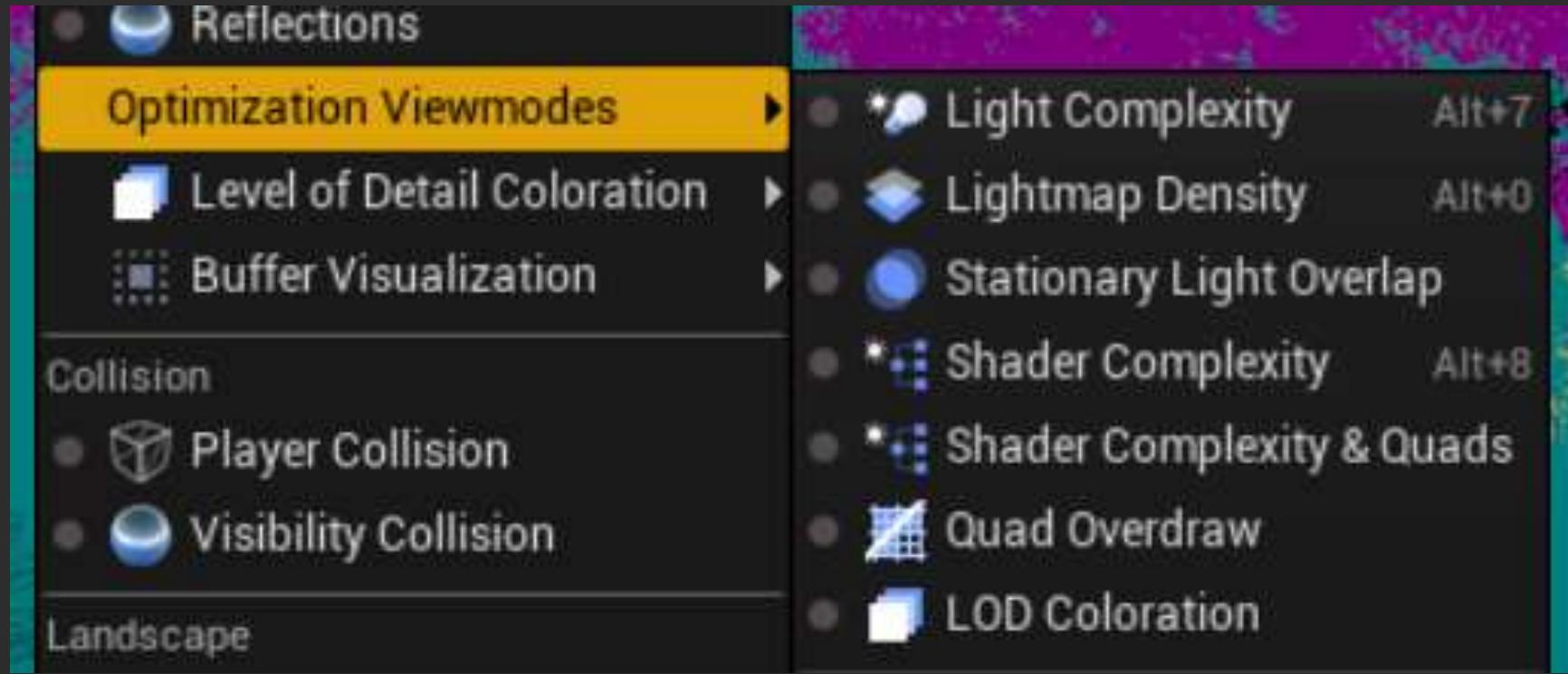


EditorのViewModeから選択可能
見れる一覧は公式ドキュメントで詳細に説明があります。

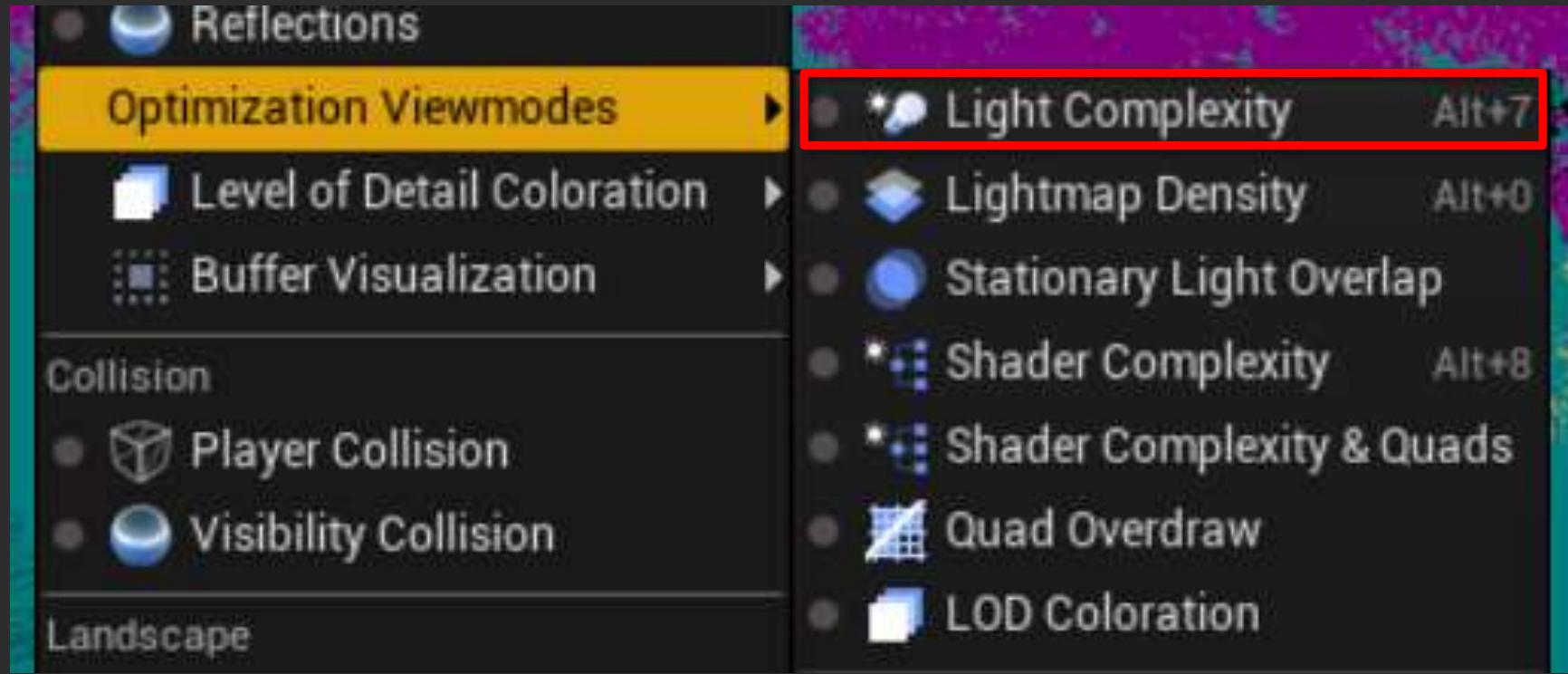
[Link: 公式ドキュメント ViewMode](#)

一部Viewmodeは
Viewmode コマンドで
実機でも見ることが可能

Optimization ViewModes



Light Complexity



Light Complexity

動的ライティングが
どれだけされている
かを視覚化

気づかず、大きな
ライティングをして
いる際などは要注意。

5以上になると紫に
なっていきます



Console Command **togglelight** 文字列

文字列を含むライトをOn/Offにすることができる。



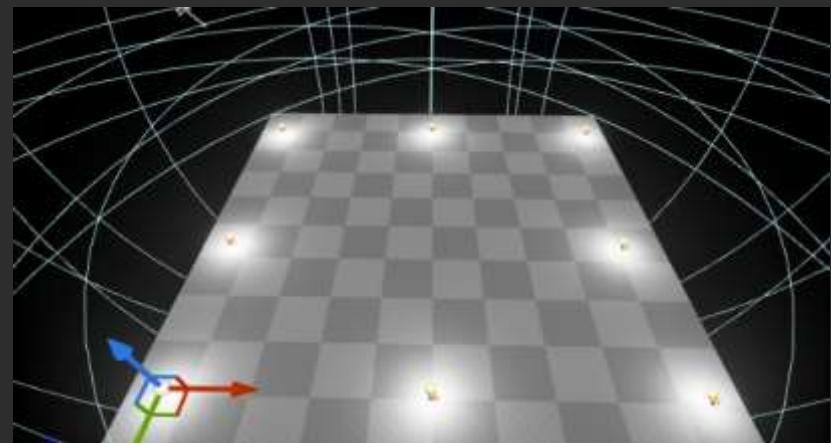
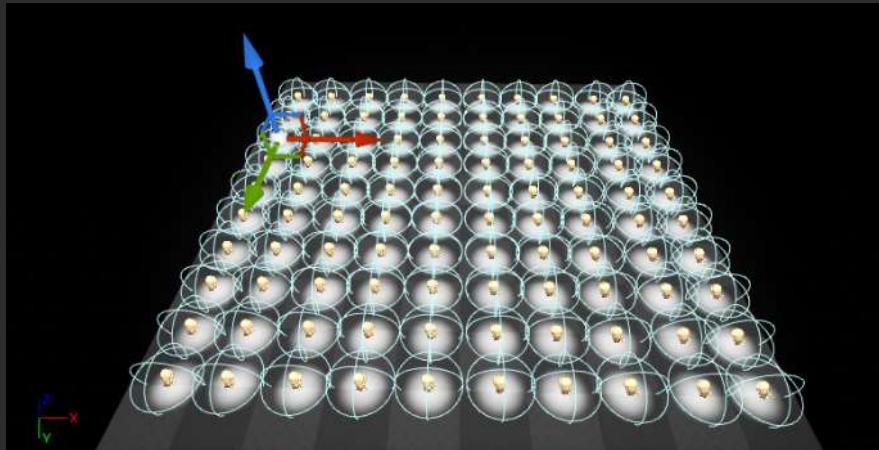
?

Deferred Lightingは
影がなければ動的ライト沢山おいても軽い？

範囲の小さい
Movable Lightを100個

VS

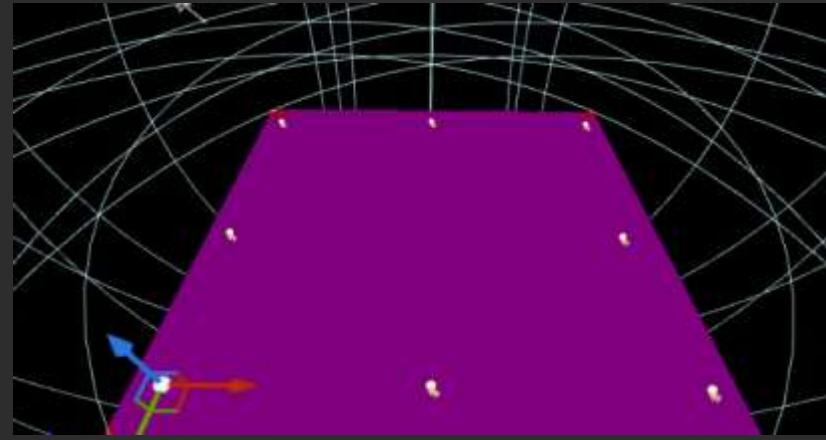
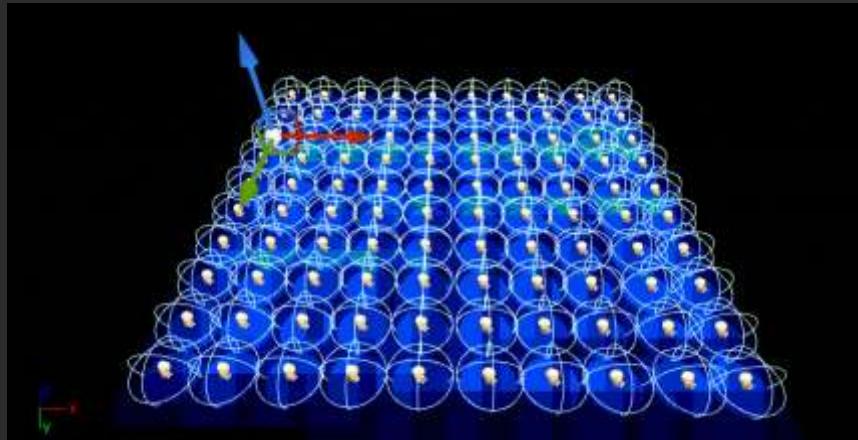
範囲の大きい
Movable Lightを8個



範囲の小さい
Movable Lightを100個

VS

範囲の大きい
Movable Lightを8個

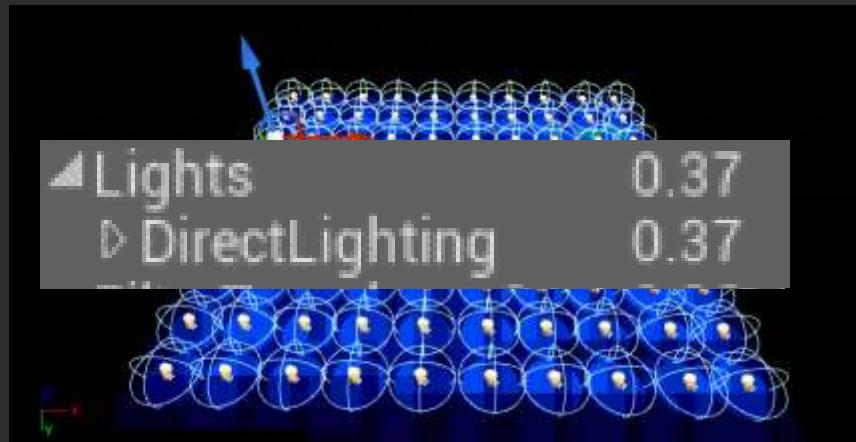


Light Complexityでの比較

範囲の小さい
Movable Lightを100個

VS

範囲の大きい
Movable Lightを8個



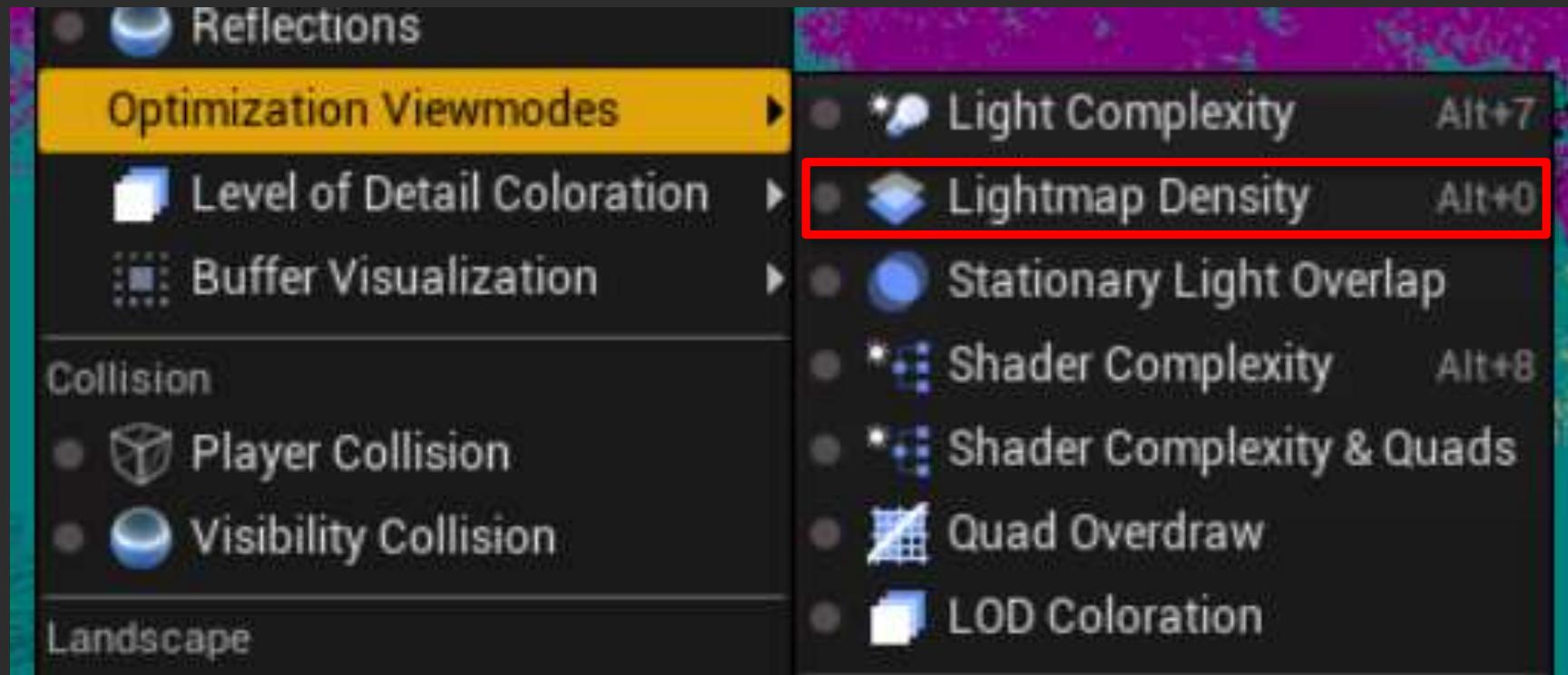
処理負荷確認方法
ProfileGPU / stat GPUのLightsにGPU負荷がでます

Q: Deferred Lightingならライトを沢山おいていいのか？

ライトを沢山を”置けます”が、
ライトが**触れる**ピクセルが増えれば増えるほど処理負荷は増えます。

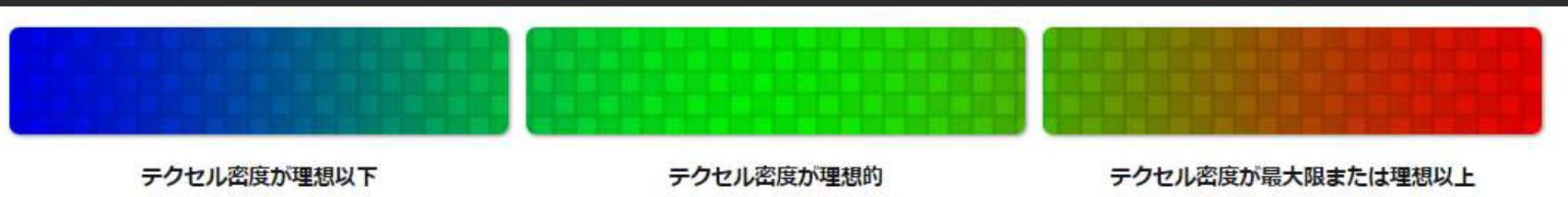
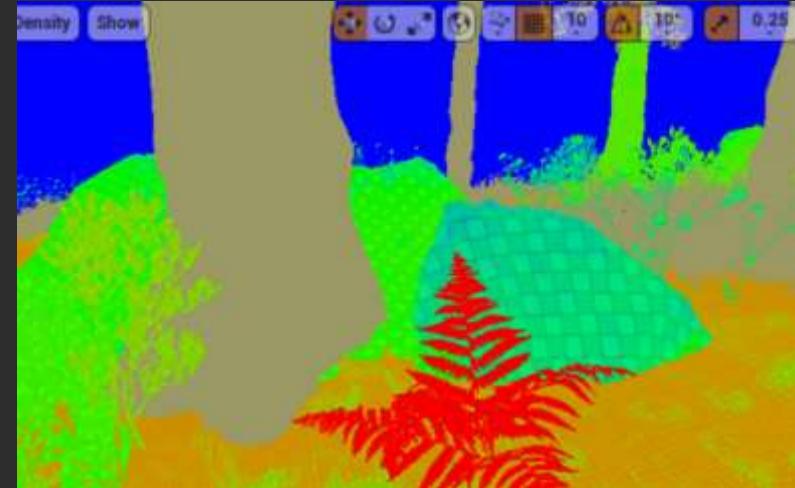
簡易的に、Light Complexityという機能があるので、
そちらで高負荷なライティングの場所を探りましょう。

Lightmap Density



Lightmap Density

Lightmap 密度を視覚的に表示



Lightmap Density

ライトマップの密度は
処理負荷にも影響はしますが
それよりも
メモリ(テクスチャストリーミング)
ロード時間に影響します

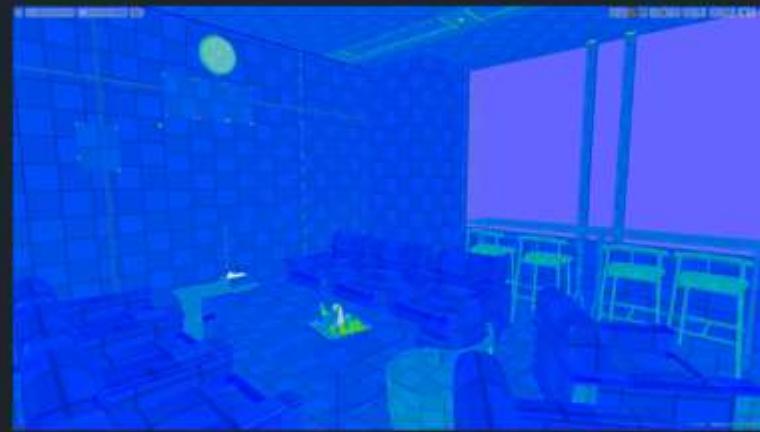
事例があるので紹介させてください



大規模CSゲーム における ライトマス運用

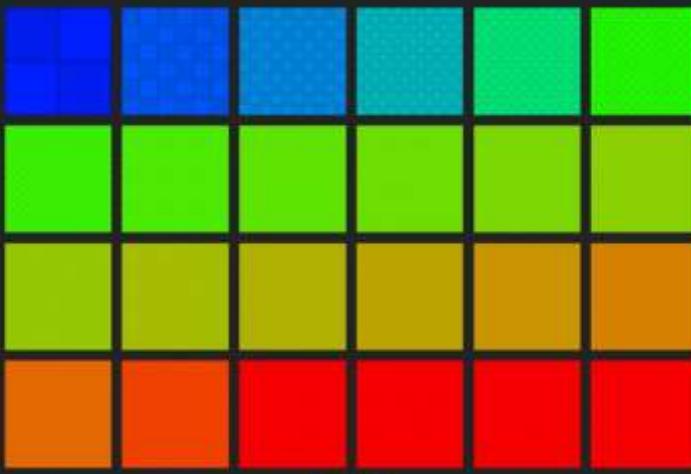
© 2016 SQUARE ENIX CO., LTD. All Rights Reserved.





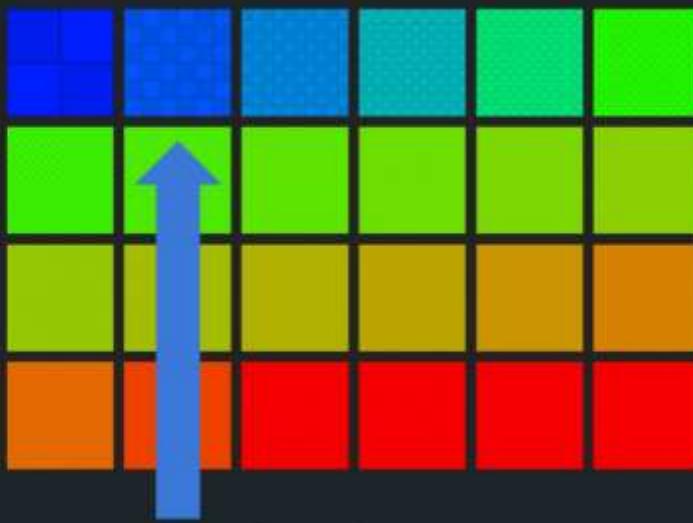
ライトマップ密度は
基本的に「青」に制限

© 2016 SQUARE ENIX CO., LTD. All Rights Reserved.



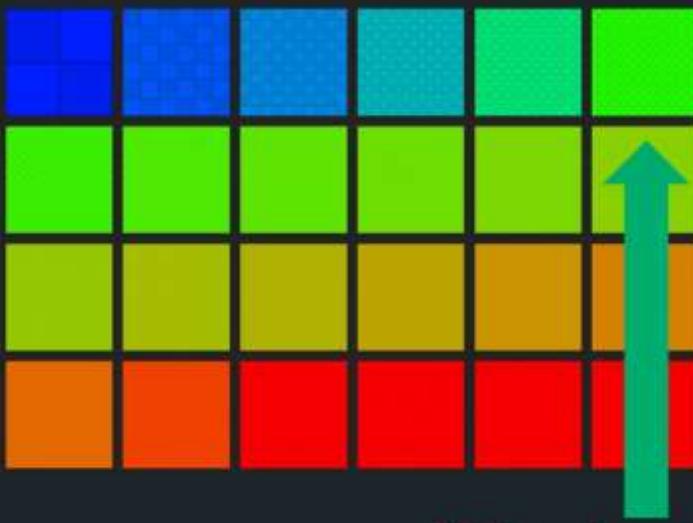
ライトマップ密度 「青」 ってどれくらい?

© 2016 SQUARE ENIX CO., LTD. All Rights Reserved.



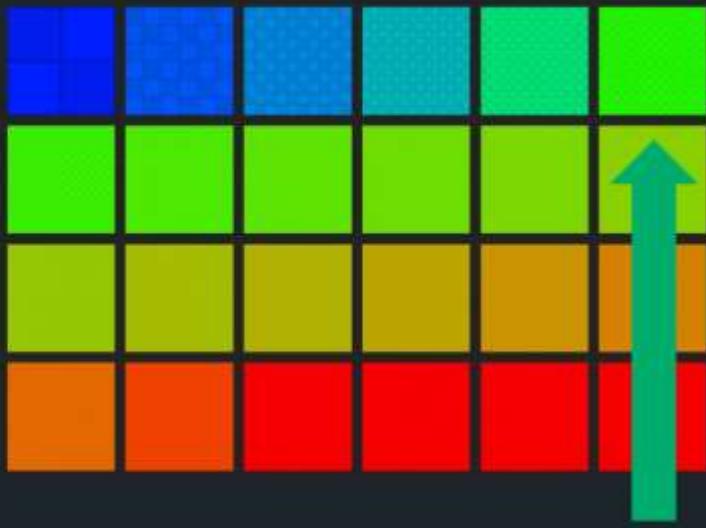
1m ÷ 5Pixel

© 2016 SQUARE ENIX CO., LTD. All Rights Reserved.



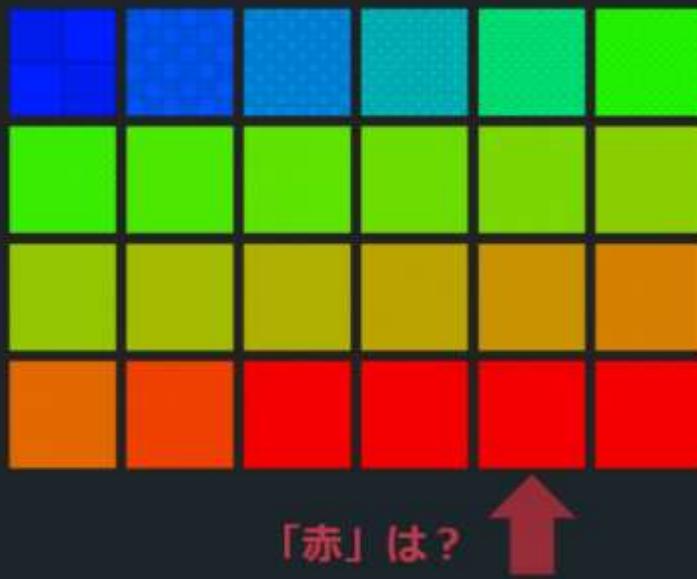
どうしてもって言う時は
「縁」も使用可能です

© 2016 SQUARE ENIX CO., LTD. All Rights Reserved.



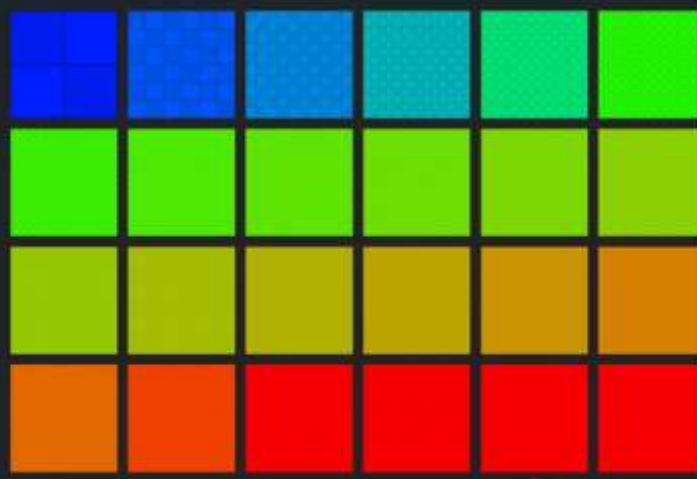
1m ≈ 20Pixel

© 2016 SQUARE ENIX CO., LTD. All Rights Reserved.



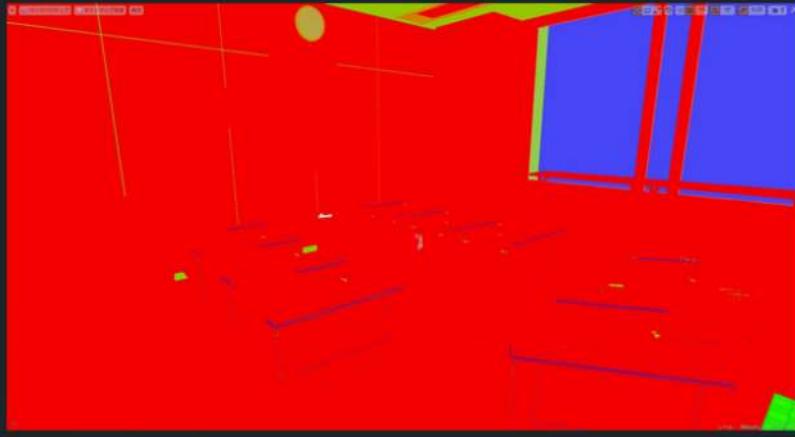
「赤」は?

© 2016 SQUARE ENIX CO., LTD. All Rights Reserved.



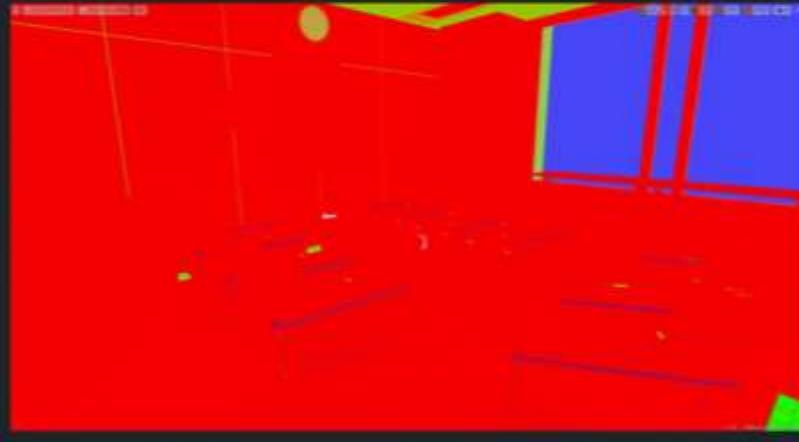
不可能です
1m ÷ 80Pixel

© 2016 SQUARE ENIX CO., LTD. All Rights Reserved.



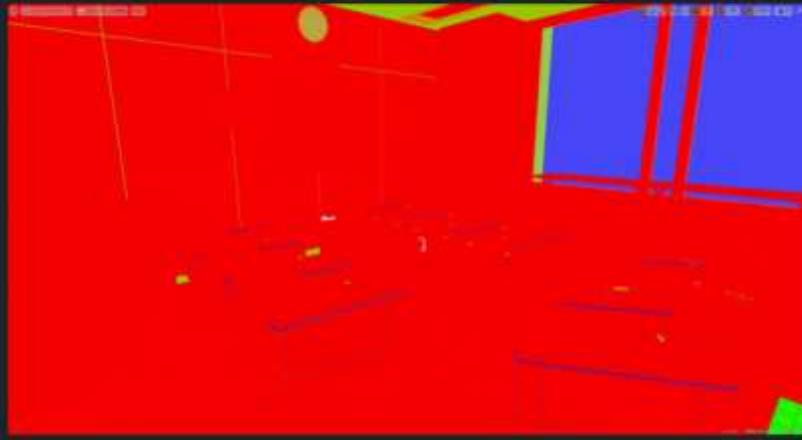
起きた事例

© 2016 SQUARE ENIX CO., LTD. All Rights Reserved.



設定ミスで真っ赤の状態で
アップされたマップ

© 2016 SQUARE ENIX CO., LTD. All Rights Reserved.



ライトマップだけで**4GB**

© 2016 SQUARE ENIX CO., LTD. All Rights Reserved.



テクスチャプールに入りきらず
エラーで見れない

© 2016 SQUARE ENIX CO., LTD. All Rights Reserved.

さらには重すぎて
サーバーアクセスエラー

© 2016 SQUARE ENIX CO., LTD. All Rights Reserved.



怒られる

© 2016 SQUARE ENIX CO., LTD. All Rights Reserved.



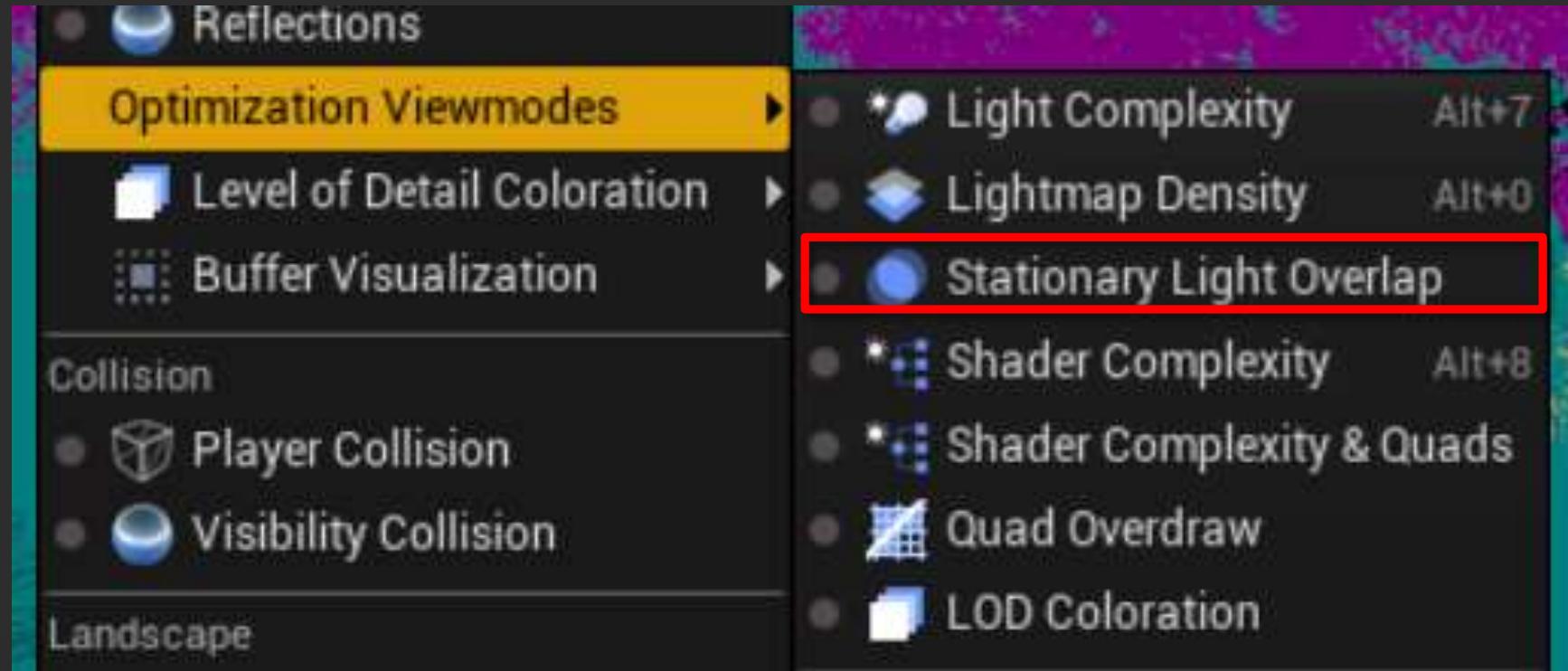
大規模CSゲーム
———— における ———
ライトマス運用

© 2016 SQUARE ENIX CO., LTD. All Rights Reserved.

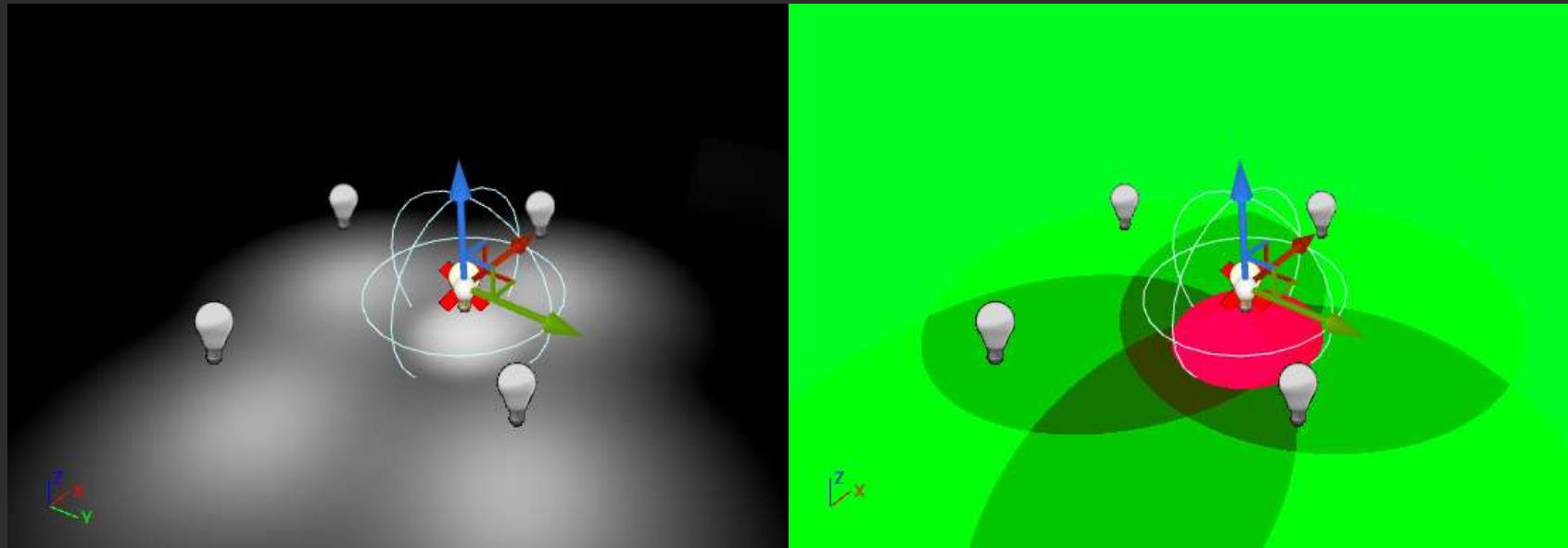
続きを読む



Stationary Light Overlap

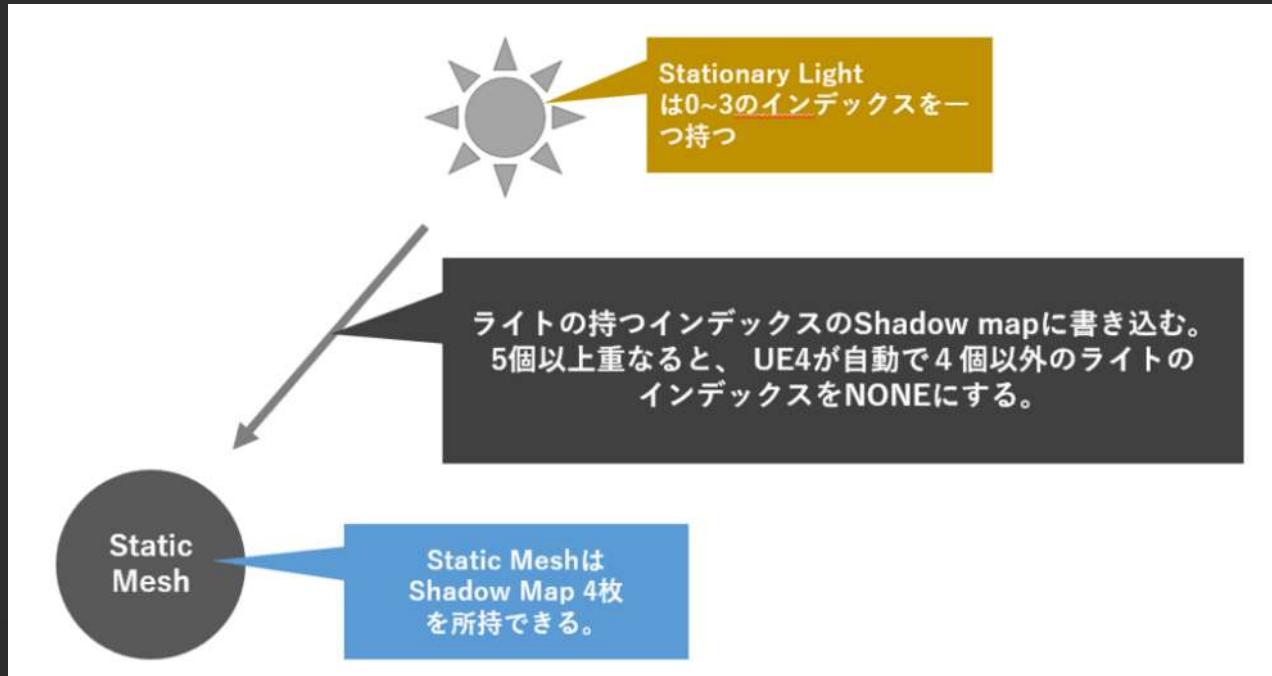


Stationary Lightのメインの制約



影付きStationary Lightを5つ以上重ねると、
5つ目以降のライトが”自動的に”MovableなLightになる。

Stationary Lightのメインの制約



2015-12-16

Stationary Light の影について

▶ UE4

この記事はUnreal Engine 4 (UE4) 其の弐 Advent Calendar 2015の16日の記事です。昨日はじゅる (@xxJulexx) | Twitterさんの映像クリエータ視点な使い方とか。でした。そもそもコンポジットをオフラインですることを前提に、BasePassと一部のポストエフェクトをそのまま出力できる機能があれば、UE4の映像制作がもっと捲りそうですね。

今回はライティングの細かい話をしようと思います。なんとなく設定している人が多い、ステーショナリーライトの仕組みです。カタカナと英語が乱立しますが、気が向いたら直させてください。

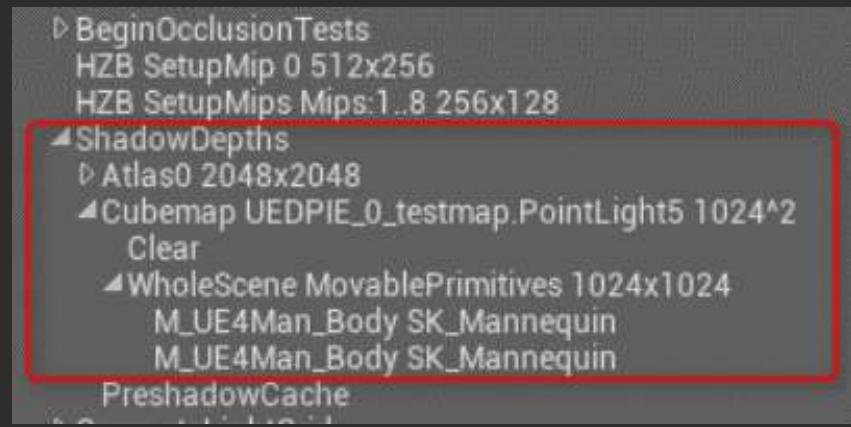
UE4のライティングは様々な組み合わせに応じて内部実装が変わります

Stationary Lightの影については、上記ブログが参考になります



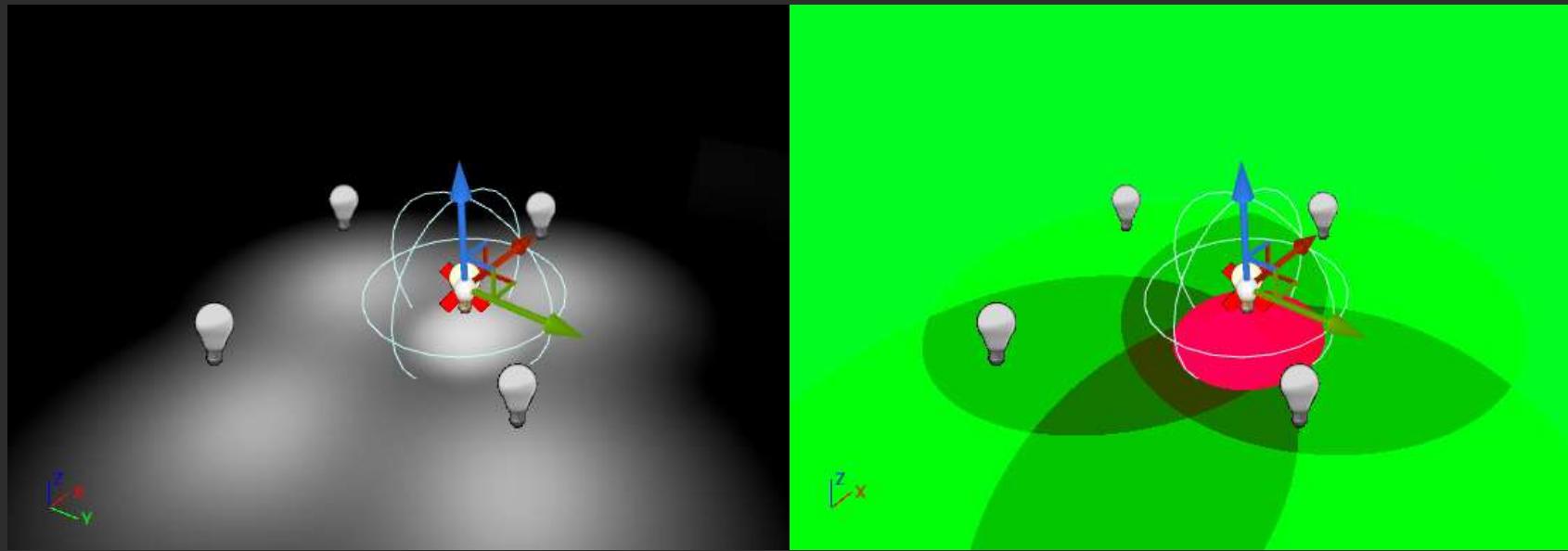
オーバラップしたStationary Lightのランタイム負荷

- Movable Lightとして動的シャドウを生成することになるので。。
 - Draw, GPUともに負荷がかかります。



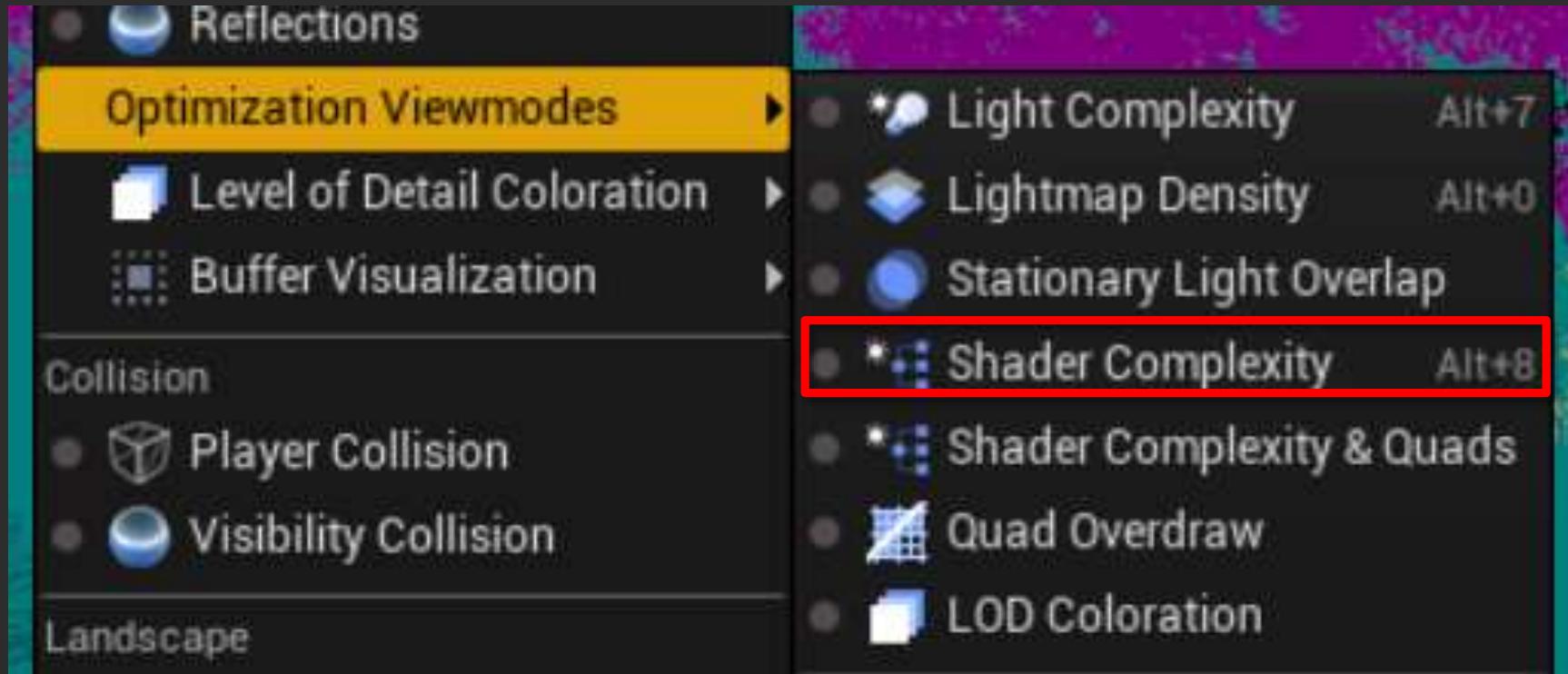
```
↳ BeginOcclusionTests
H2B SetupMip 0 512x256
H2B SetupMips Mips:1..8 256x128
◀ ShadowDepths
  ▷ Atlas0 2048x2048
  ▷ Cubemap UEDPIE_0_testmap.PointLight5 1024^2
    Clear
  ▷ WholeScene MovablePrimitives 1024x1024
    M_UE4Man_Body SK_Mannequin
    M_UE4Man_Body SK_Mannequin
  PreshadowCache
  ▷ PreshadowCache
```

Stationary Lightのメインの制約



特に良いことは何もないでの、
原則禁止が良いかと思います

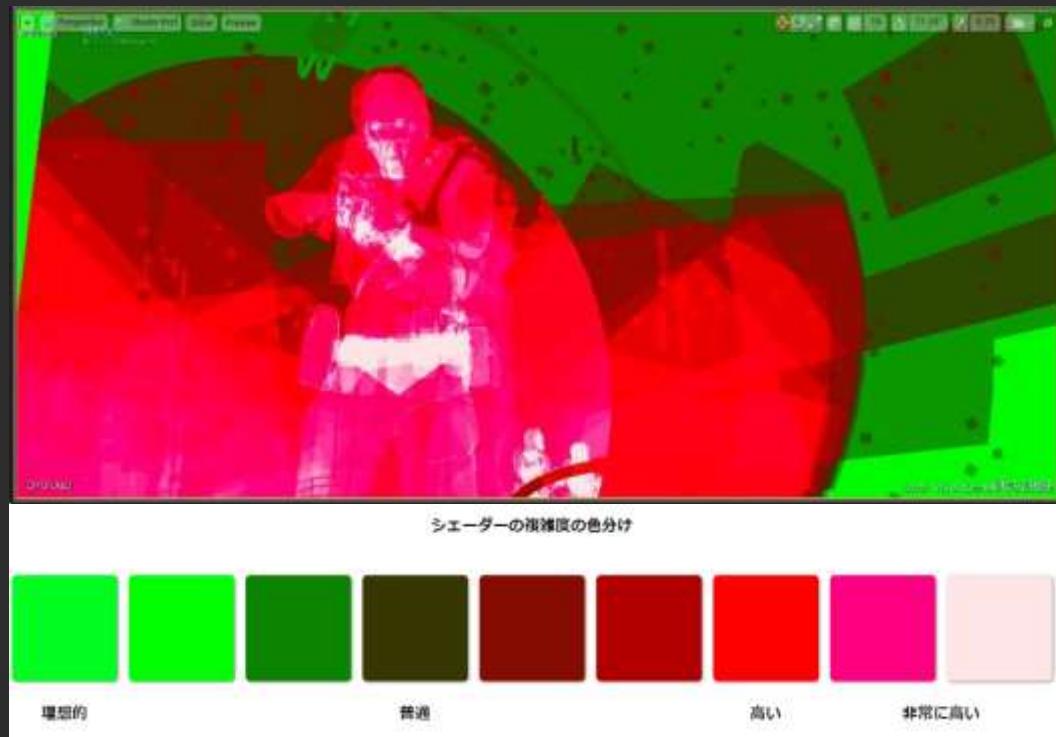
Shader Complexity



Shader Complexity

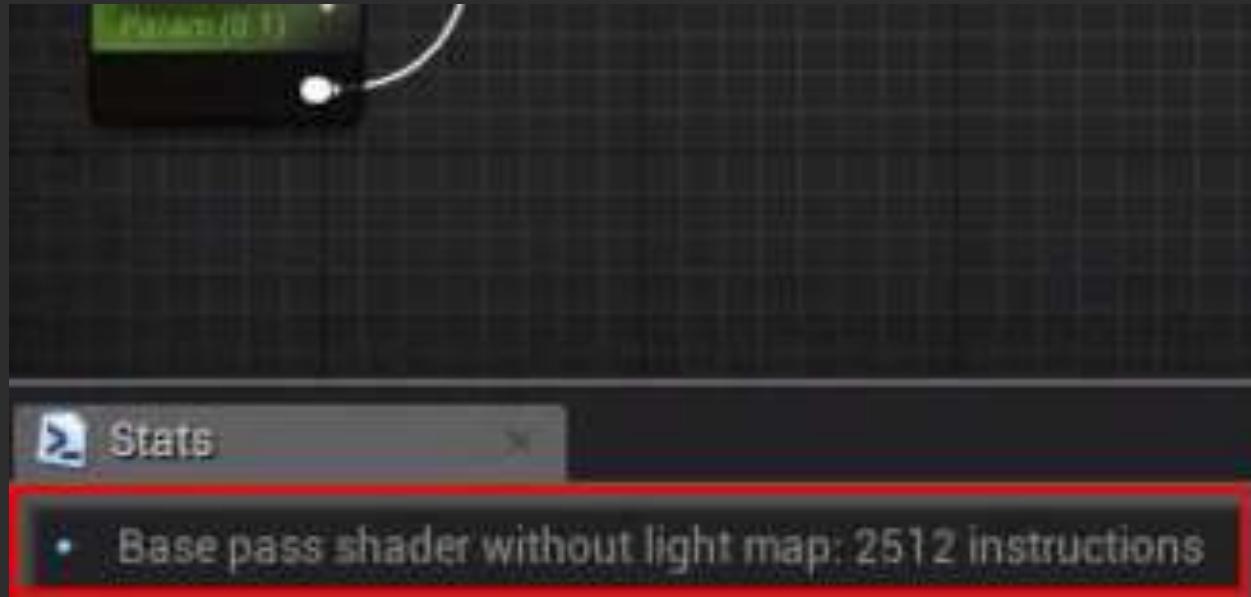
各ピクセルのシェーダの命令数の合算を表示する

(赤い部分の殆どは、
半透明描画の重ね合わせ)



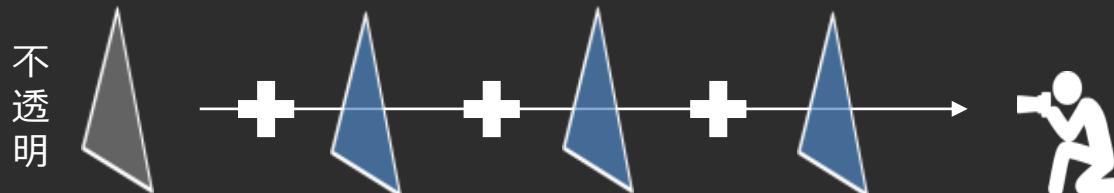
Shaderの命令数の見方

Material Editorの**Stat Window**

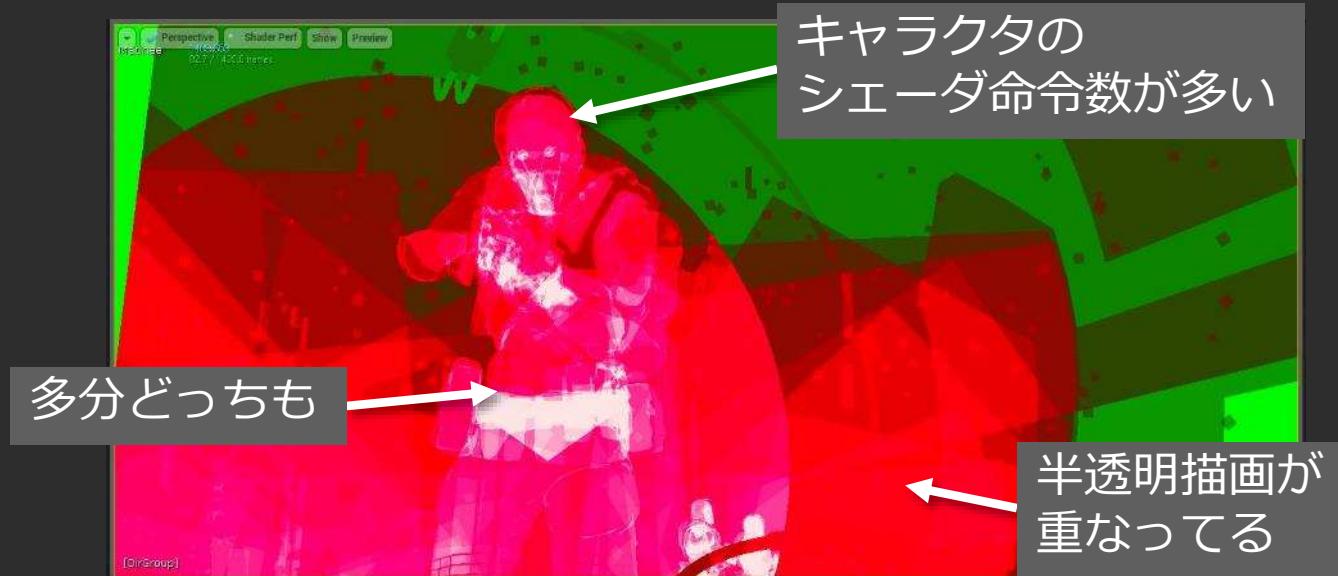


Stat WindowにPC版での大まかなシェーダ命令数を出してくれます

Shader Complexity の仕組み



Shader Complexity の見方





Shader Complexityが赤いのは
直さなきや駄目？

注意点

シェーダの命令数 ≠ シェーダの重たさ
(Cycle数と言おうか。。)

(大雑把に見れば=でもいいけど。。。)

命令数と処理負荷の違い:1

1命令の重たさの違い

パスワード変更しといて

GDCでアテンドしてきて

どちらも同じ1行(命令)だけれども、
タスク量(Cycle数)が違う。

命令数と処理負荷の違い:2

命令の仕方の違い

命令数: **50**

ビールで乾杯;

もう一杯;

もう一杯;

もう一杯;

...

もう一杯;

50行

命令数: **1**

50杯ビール飲む

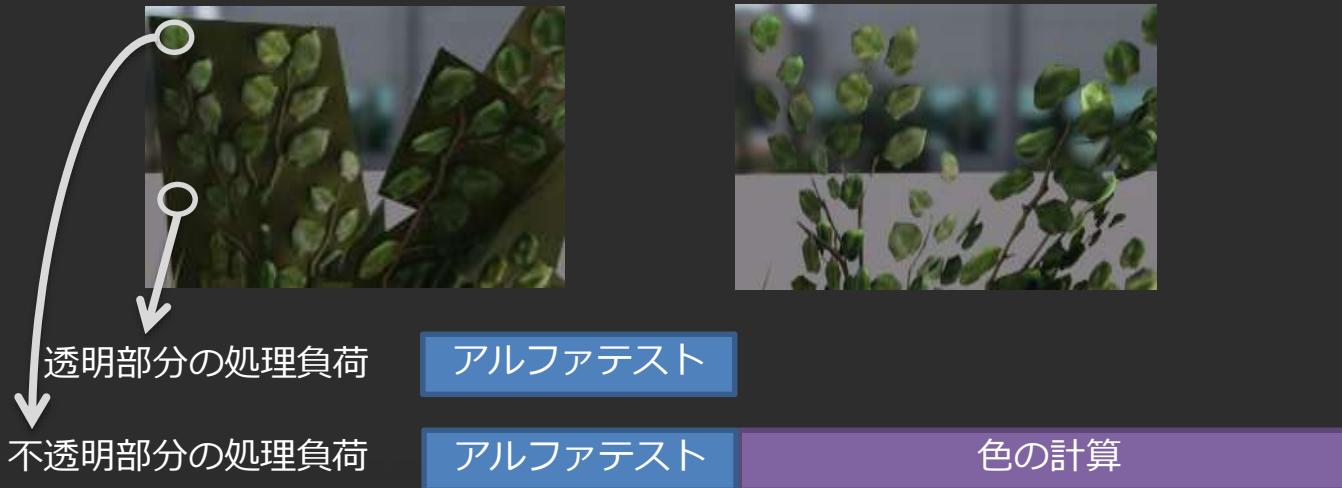
1行

※あくまでイメージです。コンパイラは賢く処理するし、実際にForLoopが1命令になるわけではありません。

命令数と処理負荷の違い:3

同じシェーダでも処理負荷がランタイムで変わる

コンパイラは賢い。アルファテストのあるシェーダを
アルファテスト -> 色と計算するようにしたりする。(Early Exit)



シェーダの命令数とシェーダの重たさの違う例1

悪意をもってマテリアルを作ると。。。スフィアに貼り付けただけで、

39命令のシェーダでも、

20msの処理負荷に



シェーダの命令数とシェーダの重たさの違う例2

悪意をもってマテリアルを作ると。。。スフィアに貼り付けただけで、

2533命令のシェーダでも、

0.04msの処理負荷で済む。



2533 instructions

0.04

注意点

シェーダの命令数 ≠ シェーダの重たさ
(Cycle数と言おうか。。)

(大雑把に見れば=でもいいけど。。。)

Q. Shader Complexityで赤いのは直さなきや駄目？

このように、
処理負荷は命令数だけでは厳密にはわかりません。

一番怖いのは、
効果の少ない最適化をしてしまうこと

Shader Complexityが真っ赤だからといって、
必死に命令数を減らしても効果がないかもしれません。

ダメな例：
テクスチャの参照回数がネックなのに、
キャラの重要な強調表現の計算量を減らす

Shader Complexityのおすすめの使い方 1 キャラクタなどの不透明オブジェクト

赤や白の場合。。。。

マテリアルに問題ないかエンジニアに
一度ご相談

プロファイル例:
キャラクタを単純なマテリアルに差し替えて
差分計測



Shader Complexityのおすすめの使い方 2 半透明やMaskedマテリアル

赤や白の場合。。。

透明部分を小さくできないか検討

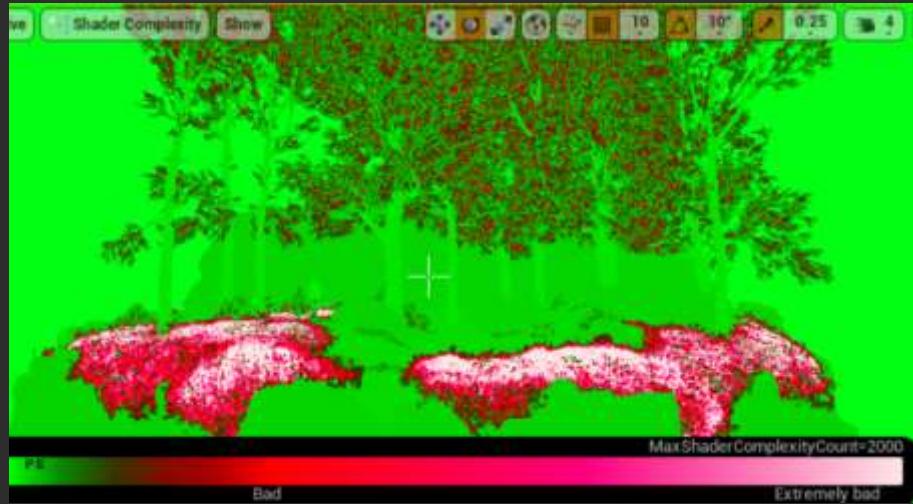
プロファイル例

Show Particles

などを使ってエフェクトをOn/Offにしてみる

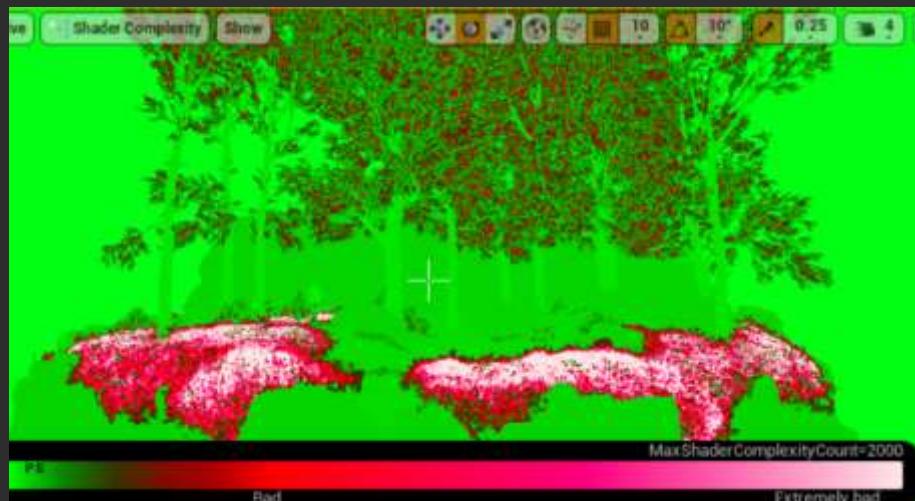


ということで、個人的には**Shader Complexity**は透明物の重なり具合にのみ使っています。。。

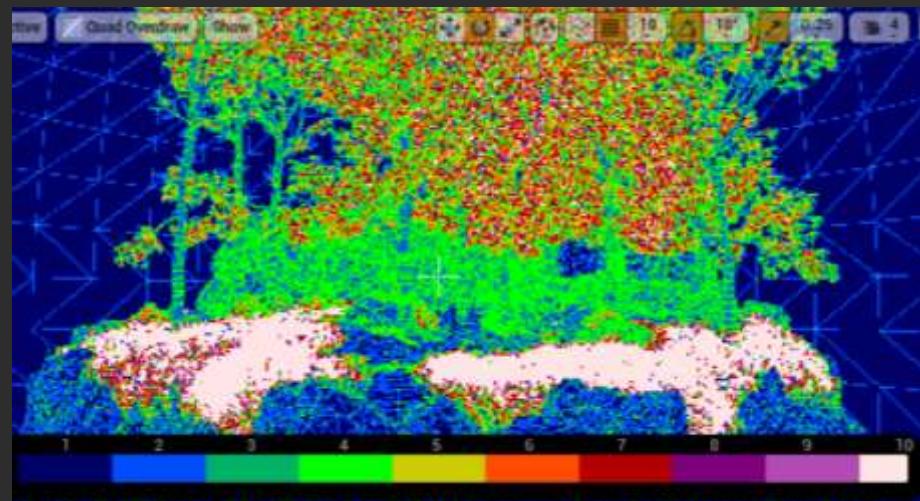


Shader Complexity View

それって。。。Quad Overdrawと同じじゃ。。

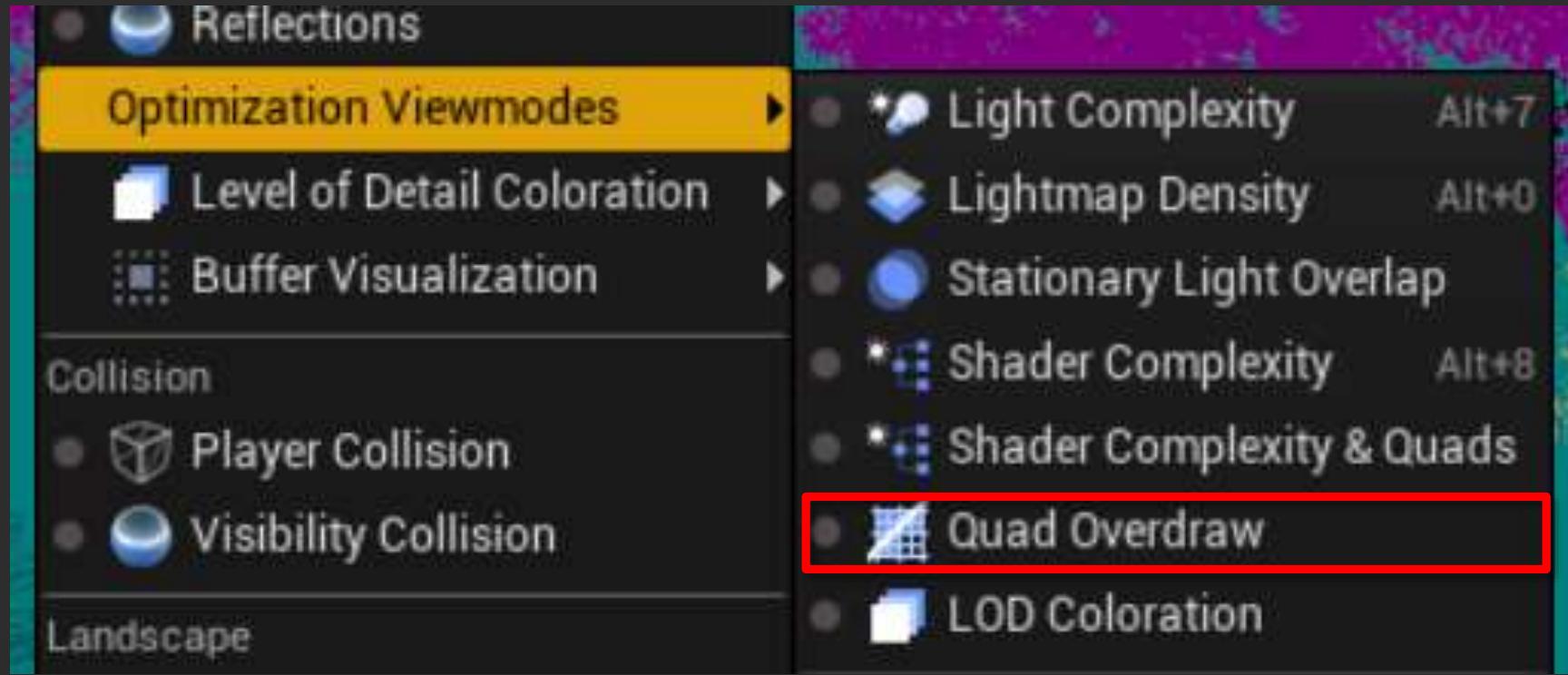


Shader Complexity View



Quad Over Draw

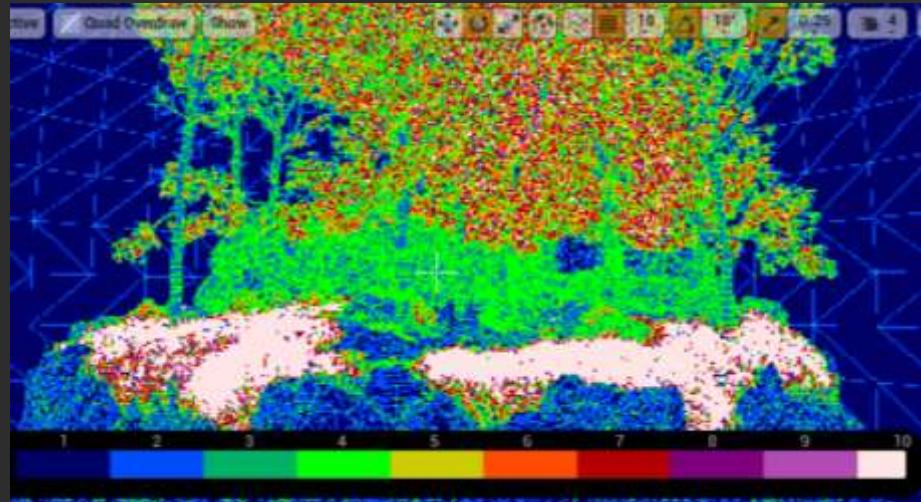
Quad Overdraw



Quad Overdraw

同一Quadで
何回描画計算が走ったかを表示

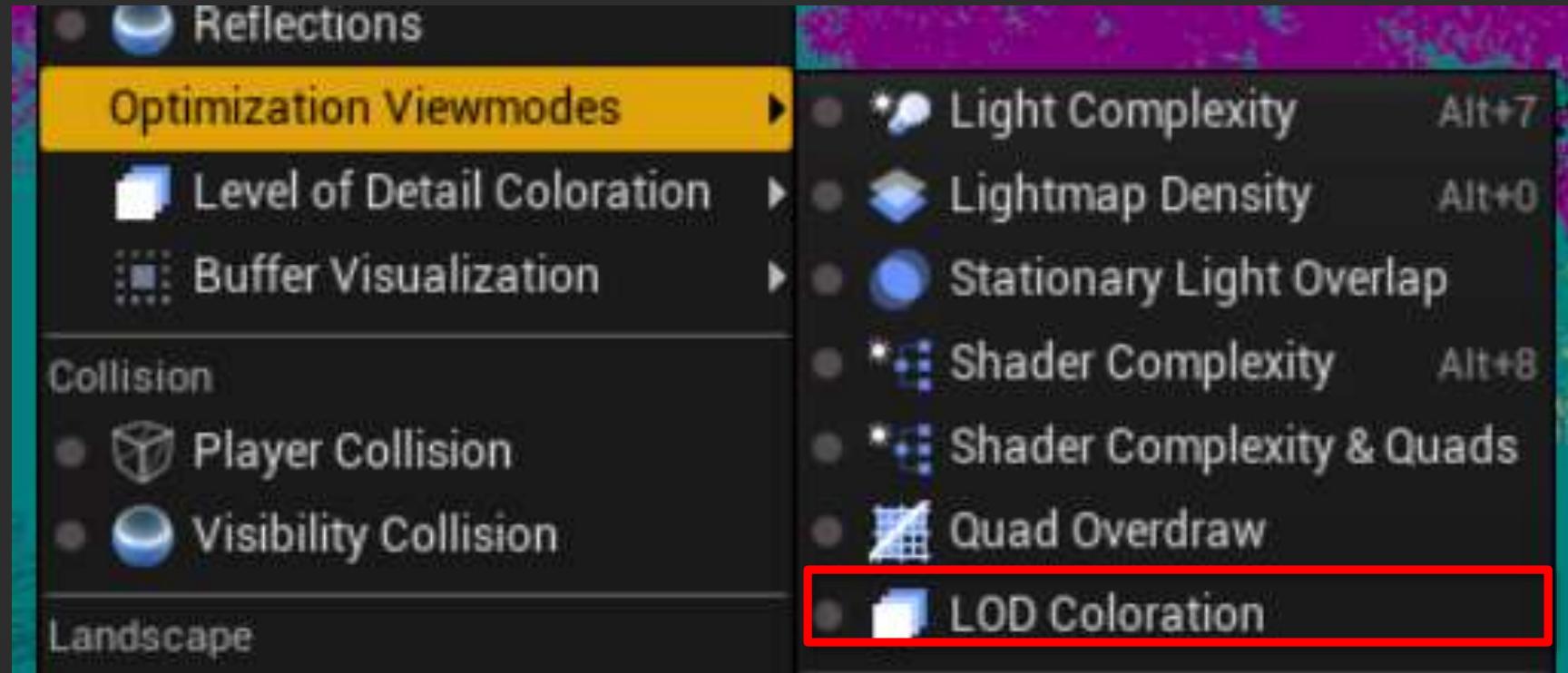
Quad = $2 * 2$ Pixel
GPUはQuad単位で描画している。



Quad Over Draw

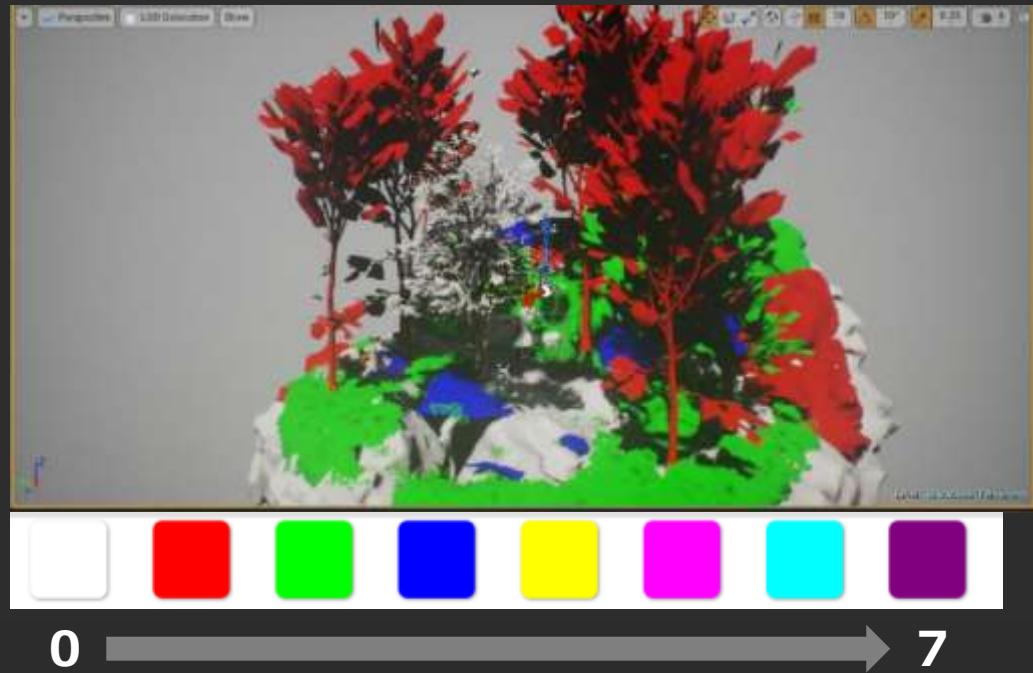


LOD Coloration



LOD Coloration

どのLODモデルを表示しているかを色で表示



LODの処理負荷確認方法

LODを使わない場合、GPU描画負荷の様々な箇所で負荷としてできます。

処理負荷確認方法

BasePass、Prepass, Shadowなどの処理が重たい際に、そのオブジェクトがどれくらいの処理負荷になっているか確認する。

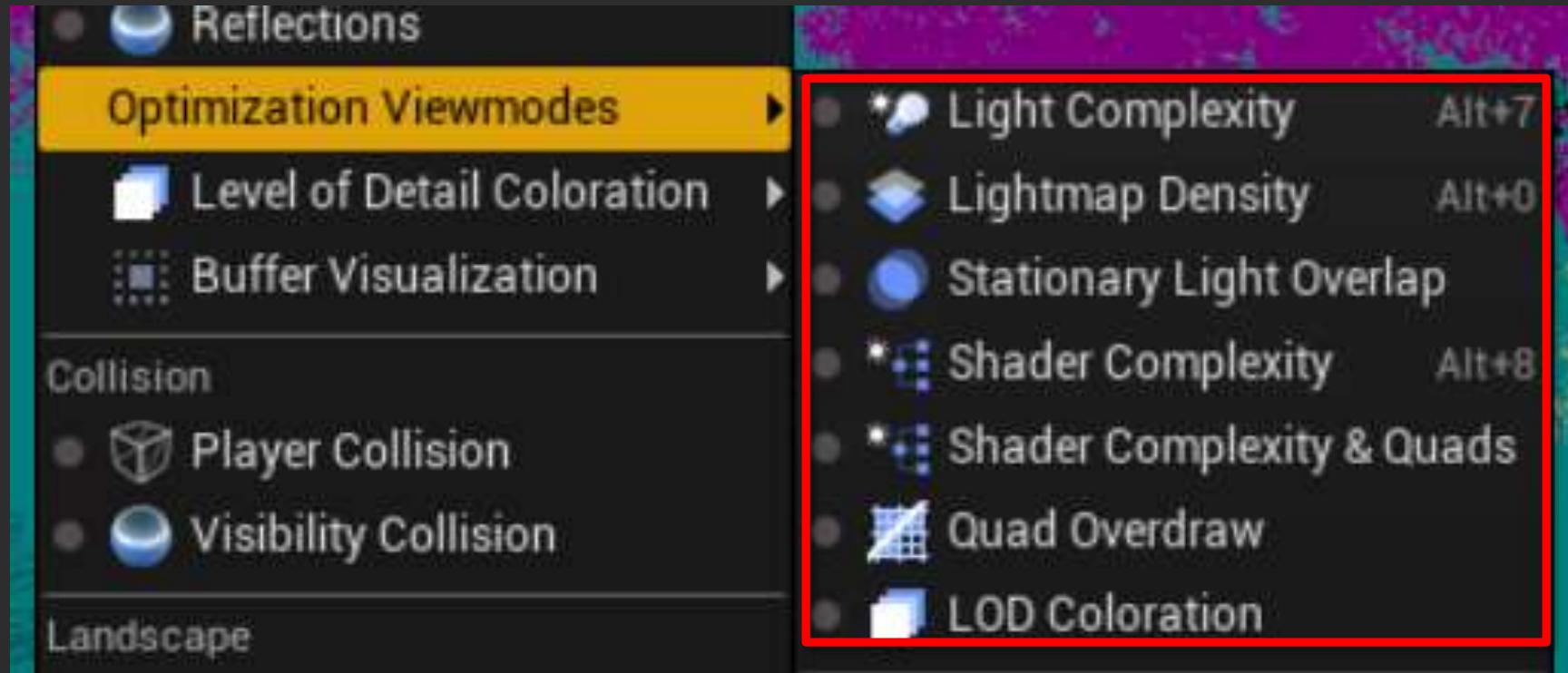
r.staticmeshLODDistanceScaleでおおざつぱにLODの範囲を全体で変えて、どれくらいの処理軽減になるか見積もれる。

▲ BasePass	1.93
▲ Static EBasePassDrawListType=1	1.00
M_DeadLeaves SM_DeadLeaves	0.01
M_DeadLeaves SM_DeadLeaves	0.00
M_DeadLeaves SM_DeadLeaves	0.00
M_DeadLeaves SM_DeadLeaves	0.00
M_DeadLeaves SM_DeadLeaves	0.01
M_DeadLeaves SM_DeadLeaves	0.00
M_DeadLeaves SM_DeadLeaves	0.01
HillTree_01_Leaves_Mat HillTree_02	0.04
HillTree_01_Leaves_Mat HillTree_02	0.17
HillTree_01_Leaves_Mat HillTree_02	0.17
HillTree_01_Leaves_Mat HillTree_02	0.07
M_Fern_01 SM_Fern_01	0.01
HillTree_01_Leaves_Mat2 HillTree_Tall_02	0.06

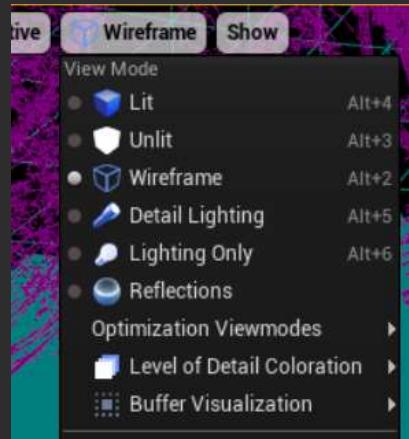
Console Command **r.staticmeshLODDistanceScale**

コマンドでLODのスケールを一括で変更
(大雑把に、LODを強くしたらどれだけ処理負荷が下がるかの見積もりとして)

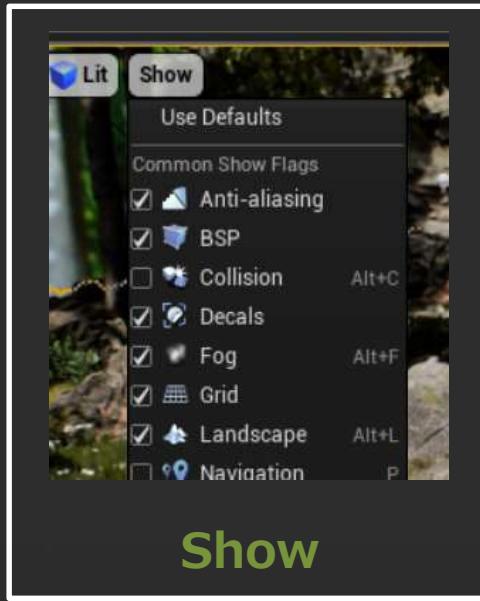
Optimization ViewModes を見てきました。。。.



今回紹介する3つのEditor機能



Viewmode



Show

A screenshot of the Unreal Engine Content Browser showing the 'Statistics' tab. The table lists objects, their counts, and types. The data is as follows:

Object	Actor(s)	Type	Count	Inst Se
HillTree_02	4 Actors	Static Mesh	4	16
SM_Cliff01	2 Actors	Static Mesh	2	2
SM_GroundRevealR8	8 Actors	Static Mesh	8	8
LargeVolcanicRock_2	2 Actors	Static Mesh	2	2

Statistics

Tips: 実機でViewModeの変更

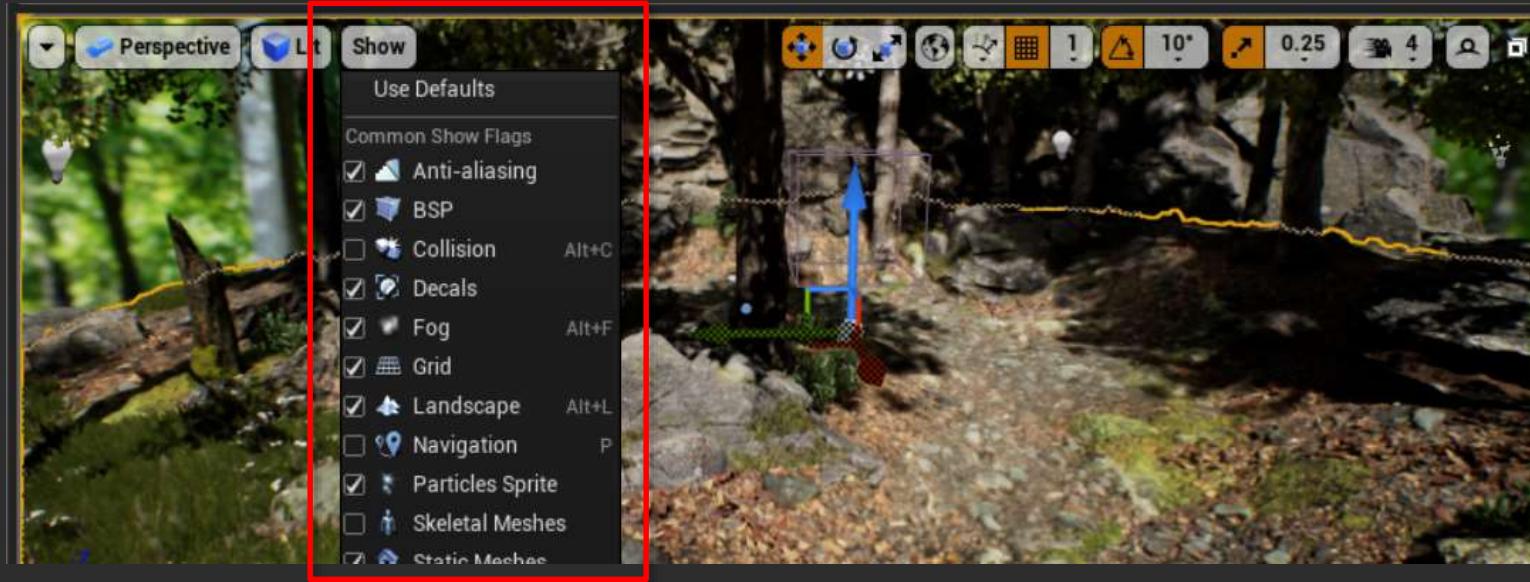
Console Command: **ViewMode XXXX**

- コンソールが対応していないViewMode
 - Unlit
 - StationaryLightOverlap
 - Lit_DetailLighting
 - ReflectionOverride
- **r.ForceDebugViewModes**を1にすると。。
 - ShaderComplexityも可能

ConsoleCommand ViewMode “HOGEHOGE” 一覧

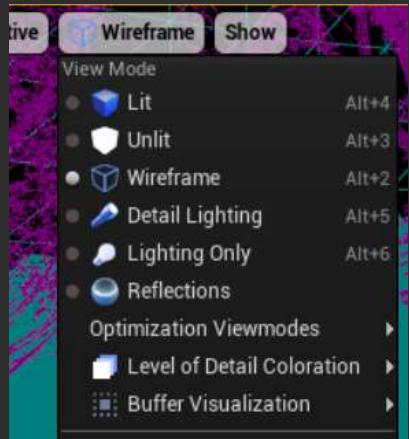
- case VMI_BrushWireframe:return TEXT("BrushWireframe");
- case VMI_Wireframe:return TEXT("Wireframe");
- case VMI_Unlit:return TEXT("Unlit");
- case VMI_Lit:return TEXT("Lit");
- case VMI_Lit_DetailLighting:return TEXT("Lit_DetailLighting");
- case VMI_LightingOnly:return TEXT("LightingOnly");
- case VMI_LightComplexity:return TEXT("LightComplexity");
- case VMI_ShaderComplexity:return TEXT("ShaderComplexity");
- case VMI_QuadOverdraw:return TEXT("QuadOverdraw");
- case VMI_ShaderComplexityWithQuadOverdraw: return TEXT("ShaderComplexityWithQuadOverdraw");
- case VMI_PrimitiveDistanceAccuracy:return TEXT("PrimitiveDistanceAccuracy");
- case VMI_MeshUVDensityAccuracy:return TEXT("MeshUVDensityAccuracy");
- case VMI_MaterialTextureScaleAccuracy: return TEXT("MaterialTexturecaleAccuracy");
- case VMI_RequiredTextureResolution: return TEXT("RequiredTextureResolution");
- case VMI_StationaryLightOverlap:return TEXT("StationaryLightOverlap");
- case VMI_LightmapDensity:return TEXT("LightmapDensity");
- case VMI_LitLightmapDensity:return TEXT("LitLightmapDensity");
- case VMI_ReflectionOverride:return TEXT("ReflectionOverride");
- case VMI_VisualizeBuffer:return TEXT("VisualizeBuffer");
- case VMI_CollisionPawn:return TEXT("CollisionPawn");
- case VMI_CollisionVisibility:return TEXT("CollisionVis");
- case VMI_LODColoration:return TEXT("LODColoration");
- case VMI_HLODColoration:return TEXT("HLODColoration");

Show

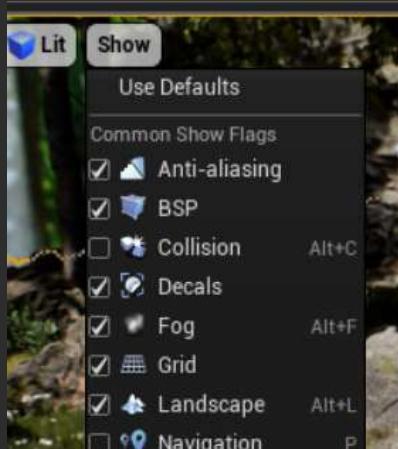


Console Command **Show** のエディタ設定版

今回紹介する3つのEditor機能



Viewmode



Show

The Content Browser window shows a statistics table with object counts and instance counts.

Object	Actor(s)	Type	Count	Inst Se
HillTree_02	4 Actors	Static Mesh	4	16
SM_Cliff01	2 Actors	Static Mesh	2	2
SM_GroundRevealR8	8 Actors	Static Mesh	8	8
LargeVolcanicRock_2	2 Actors	Static Mesh	2	2

Statistics

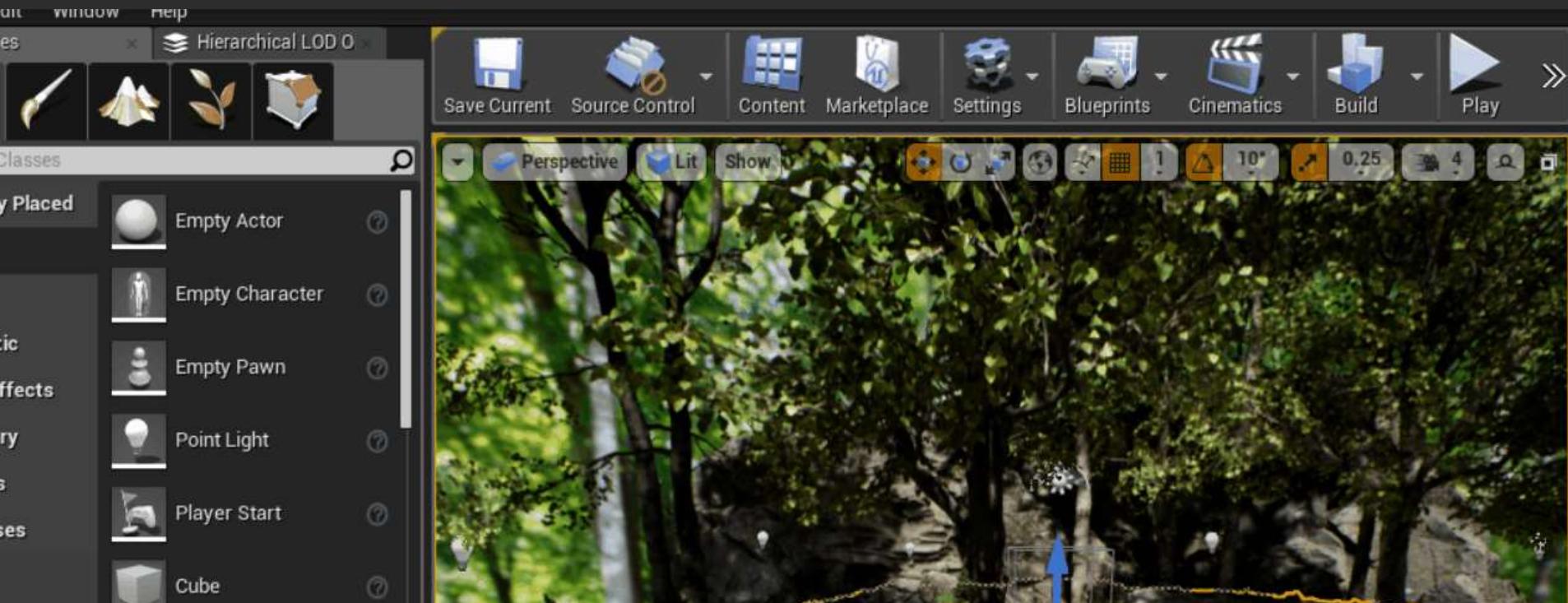
Statistics

The screenshot shows the Unreal Engine 4 interface with the Statistics mode selected in the Modes dropdown. The Statistics panel displays a table of scene assets with their respective mesh and texture counts.

Object	Actor(s)	Type	Count	Inst Secti	Tris	Sum Size	VC	Ins	Av	Av	Su	C
HillTree_02	4 Actors	StaticMesh	4	16	49,495	197,914,893.7	2520	KEO	4	0	0	0
SM_Cliff01	2 Actors	StaticMesh	2	2	29,154	58,301,1,383.80	0	KEO	0	0	0	0
SM_GroundRevealR	8 Actors	StaticMesh	8	8	10,589	84,719,31.370	0	KEO	0	0	0	0
LargeVolcanicRock_2	2 Actors	StaticMesh	2	2	16,257	32,518,27.950	0	KEO	0	0	0	0

シーンにおいてあるメッシュやテクスチャの情報を表で出してくれる

Statistics



Statistics

Primitive Stats

シーンのメッシュの統計情報を表示

何個シーンにある？

そのメッシュの
ポリゴン数は？

そのメッシュの
シーン内の大きさは？

The screenshot shows the Statistics panel in the Unreal Engine interface. The 'Primitive Stats' tab is selected, indicated by a red box. The table lists various objects along with their counts, sizes, and bounding box dimensions.

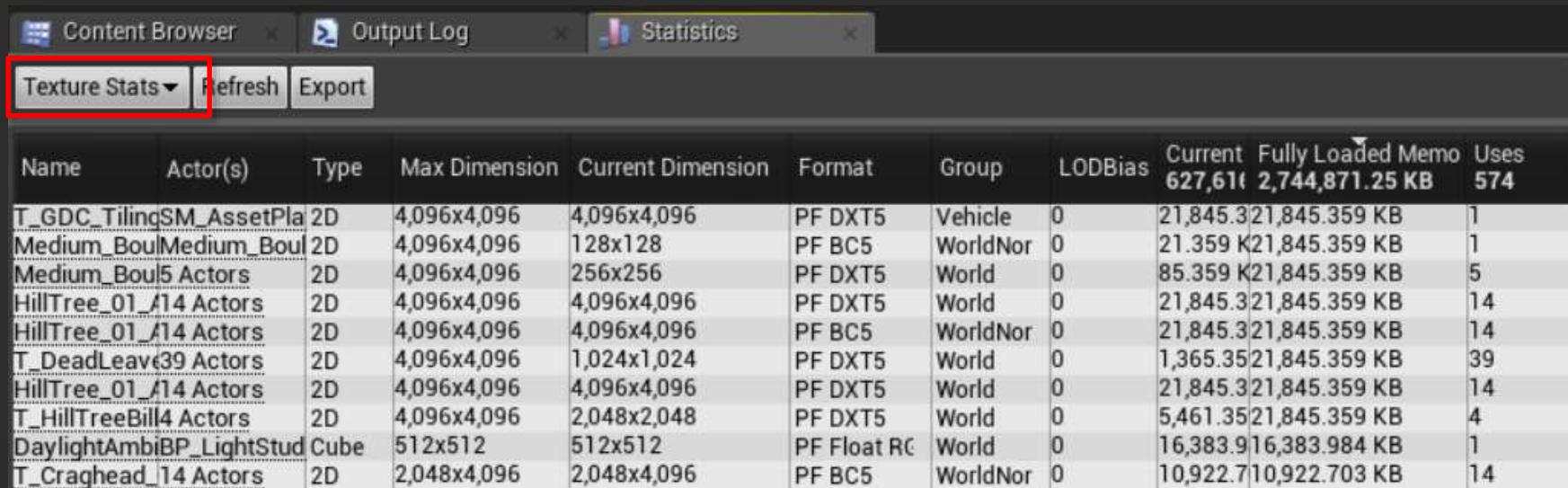
Object	Actor(s)	Type	Count	Inst	Setti	Tris	Sum	Size	VC	Ins	Av	Av	Su	Co	LM	Res	Min R	Max R	Avg R
HillTree_02	4 Actors	StaticM	4	5		49,495	197,914	893.7	2520	KEO	4	0	0	0	KB	8,192	444.272	580.648	2,138.491
SM_Cliff01	2 Actors	StaticM	2	2		29,154	56,301	1,383.80	KEO	KEO	0	0	0	5.318	KB	64	754.203	755.624	1,509.827
SM_GroundRevealR8	8 Actors	StaticM	8	8		10,589	84,719	31.370	KEO	KEO	0	0	0	85.117	KB	512	391.389	527.833	3,483.146
LargeVolcanicRock_2	Actors	StaticM	2	2		16,257	33,514	827.950	KEO	KEO	0	0	0	85.119	KB	256	242.531	431.661	674.192
BogMyrtleBush_02	2 Actors	StaticM	2	2		6,124	12,241	796.260	KEO	KEO	0	0	0	21.279	KB	128	63.333	63.333	126.666
SM_MountainRock_(25	Actors	StaticM	25	25		15,320	383,076	3.780	KEO	KEO	0	0	0	1,063.989	KI	3,200	356.128	1,874.213	21,212.75
SM_MountainRock_4	Actors	StaticM	4	4		9,088	24,357	745.110	KEO	KEO	0	0	0	170.238	KB	512	247.987	607.753	1,490.693



Statistics

Texture Stats

シーン内に読み込まれているテクスチャの統計情報を表示



The screenshot shows the 'Statistics' tab in the Unreal Engine interface. The 'Texture Stats' section is selected, indicated by a red box around the 'Texture Stats' dropdown menu. Below it, there are buttons for 'Refresh' and 'Export'. The main table lists various textures with their details and memory usage.

Name	Actor(s)	Type	Max Dimension	Current Dimension	Format	Group	LODBias	Current	Fully Loaded	Memo	Uses
T_GDC_TilingSM_AssetPla		2D	4,096x4,096	4,096x4,096	PF DXT5	Vehicle	0	21,845.3	21,845.3	59 KB	1
Medium_BoulMedium_Boul		2D	4,096x4,096	128x128	PF BC5	WorldNor	0	21,359 K	21,845.3	59 KB	1
Medium_Boul5 Actors		2D	4,096x4,096	256x256	PF DXT5	World	0	85.359 K	21,845.3	59 KB	5
HillTree_01_A14 Actors		2D	4,096x4,096	4,096x4,096	PF DXT5	World	0	21,845.3	21,845.3	59 KB	14
HillTree_01_A14 Actors		2D	4,096x4,096	4,096x4,096	PF BC5	WorldNor	0	21,845.3	21,845.3	59 KB	14
T_DeadLeaves39 Actors		2D	4,096x4,096	1,024x1,024	PF DXT5	World	0	1,365.35	21,845.3	59 KB	39
HillTree_01_A14 Actors		2D	4,096x4,096	4,096x4,096	PF DXT5	World	0	21,845.3	21,845.3	59 KB	14
T_HillTreeBill4 Actors		2D	4,096x4,096	2,048x2,048	PF DXT5	World	0	5,461.35	21,845.3	59 KB	4
DaylightAmbiBP_LightStud	Cube	512x512	512x512	PF Float RC	World	0	16,383.9	16,383.9	84 KB	1	
T_Craghead_14 Actors		2D	2,048x4,096	2,048x4,096	PF BC5	WorldNor	0	10,922.7	10,922.7	03 KB	14

Example

Statistics (Primitive Stats) の半径を見て
不要に小さなオブジェクトがないかを探す

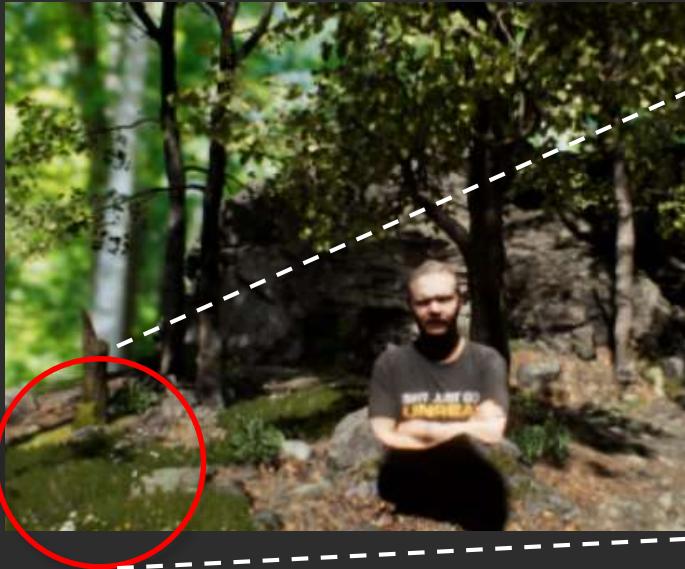
Robがいるこのシーン、
Robのメッシュを**Primitive Stats**で見てみる



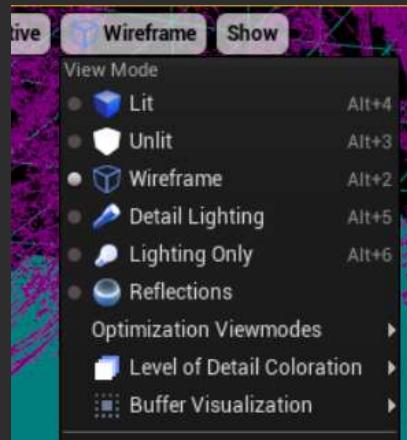
Res	Min R	Max R	A
6.	1,00	0.008	1
704	9.054	113.178	2
10,246	96.768	1,086.609	9
8,192	444.272	580.648	2
KB64	754.203	755.624	1
7,451	281.280	527.833	2

あきらかに小さいRobがいる。

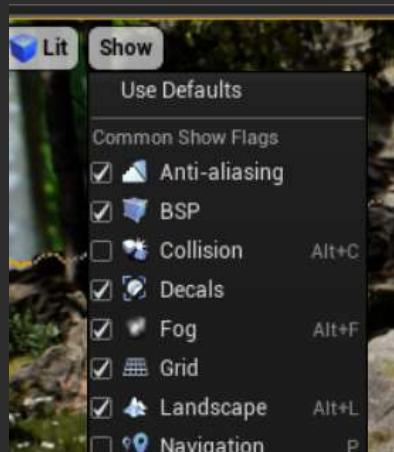
草むらに小さなロブが沢山いた



3つのEditor機能を紹介してきました



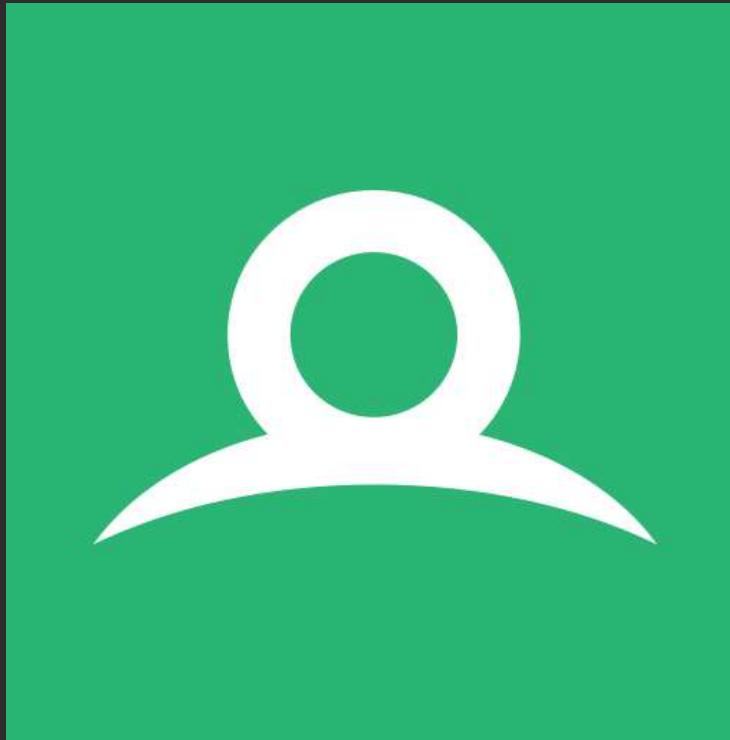
Viewmode



Show

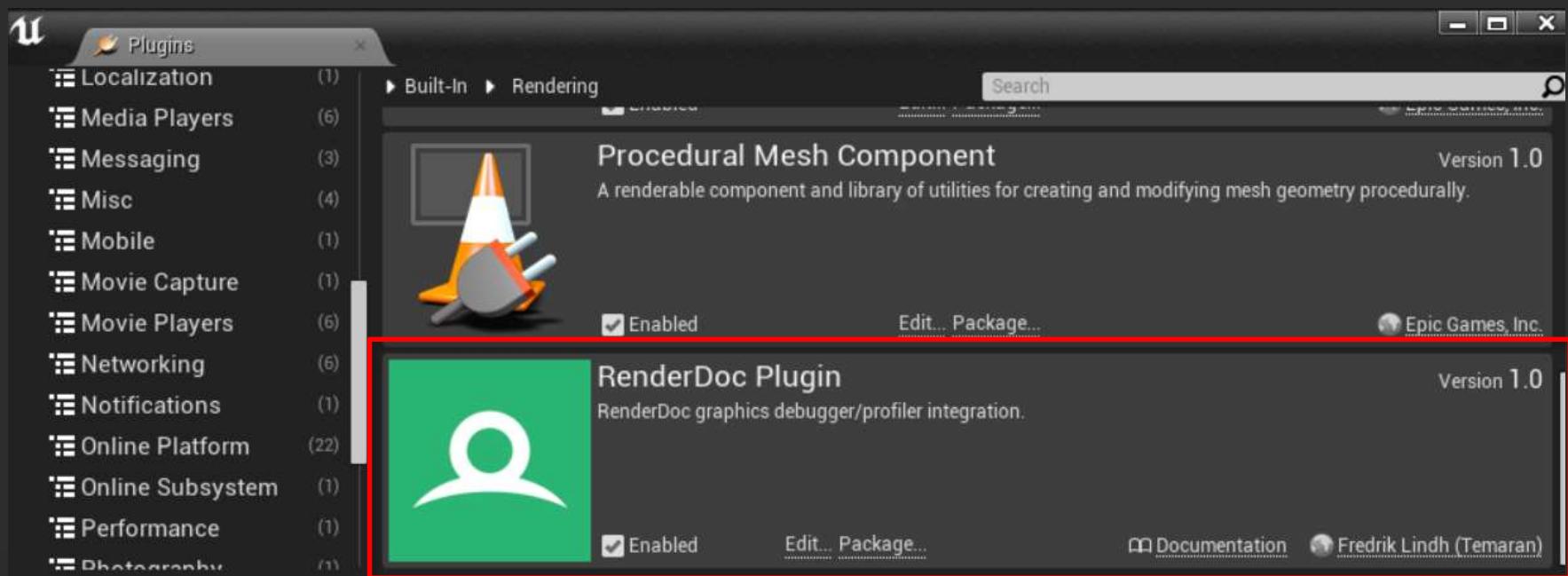
Object	Actor(s)	Type	Count	Inst Se
HillTree_02	4 Actors	StaticM	4	16
SM_Cliff01	2 Actors	StaticM	2	2
SM_GroundRevealR8	8 Actors	StaticM	8	8
LargeVolcanicRock_2	2 Actors	StaticM	2	2

Statistics



RenderDoc

UE4.16から標準プラグインに



RenderDocで何ができる？

GPUのレンダリング工程を
まるはだかにできる！

※Windows限定

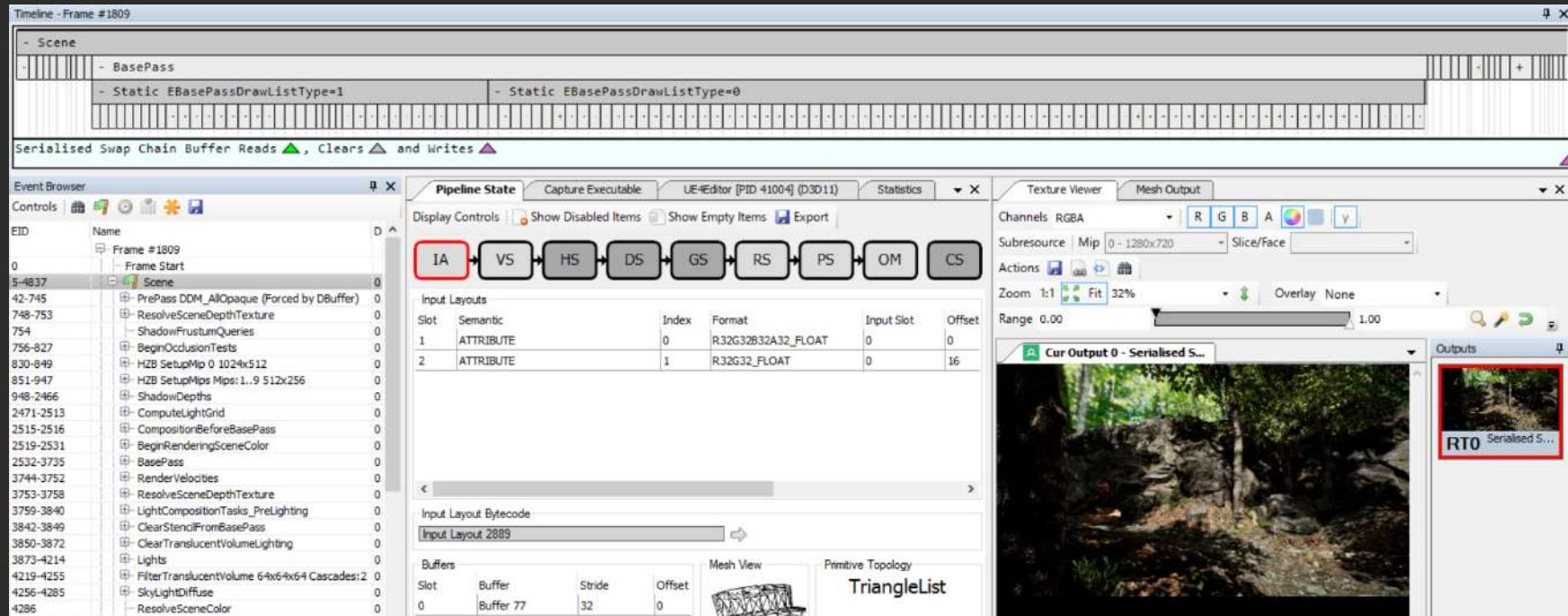
EditorからのCapture
ボタンひとつでCaptureを取ることができます



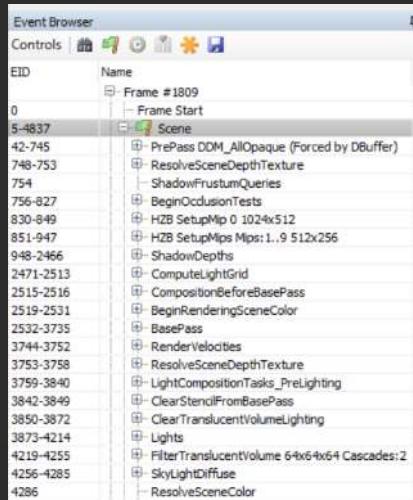
Runtimeからのキャプチャ
renderdoc.CaptureFrameでCaptureしてくれます。



RenderDoc の画面



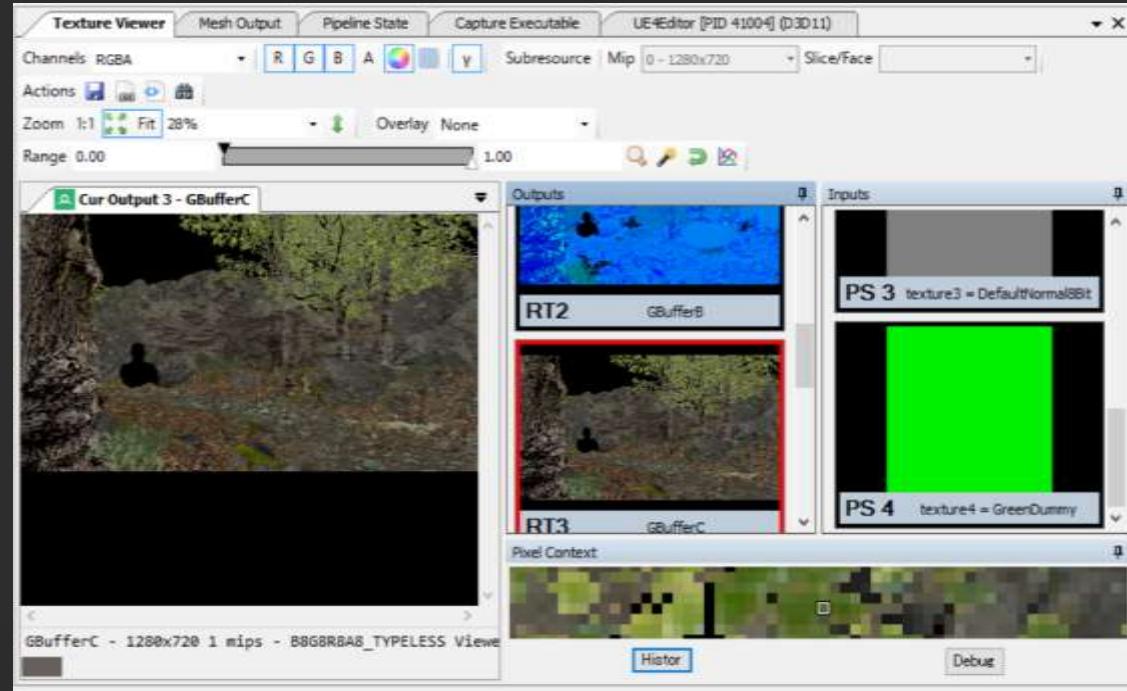
Event Browser



- 呼び出された命令を階層表示してくれる。
- このBrowserでDrawCallを選ぶ
- この後説明するエディタで、選んだDrawCallの詳細がわかる

Texture Viewer

- Input
 - そのDrawCallが使用したテクスチャ一覧が
- Output
 - そのDrawCallで出力されたRenderTargetの一覧



Texture Viewer

選択したDrawCallまで描画してくれるので、Event ViewerでDrawCallを選択していくことによって、描画の流れを理解することができる。

EID: 3536



次の
DrawCall

EID: 3559

3535-3536	└ HillTree_01_Branches_2_Mat	HillTree_01_Branches_2_Mat	8
3536	└ DrawIndexed(9)	DrawIndexed(9)	8
3549-3559	└ M_Rob Rob-Clean_Smoothed	M_Rob Rob-Clean_Smoothed	3
3559	└ DrawIndexed(613458)	DrawIndexed(613458)	3
3571-3572	└ Untextured Rob-Clean_Smoothed	Untextured Rob-Clean_Smoothed	3
3583-3591	└ HillTree_01_Branches_2_Mat	HillTree_01_Branches_2_Mat	1
3593-3597	└ HillTree_01_Branches_2_Mat	HillTree_01_Branches_2_Mat	2
3599-3603	└ HillTree_01_Branches_2_Mat	HillTree_01_Branches_2_Mat	8
3605-3609	└ HillTree_01_Branches_2_Mat	HillTree_01_Branches_2_Mat	2
3612-3720	└ Dynamic	Dynamic	2

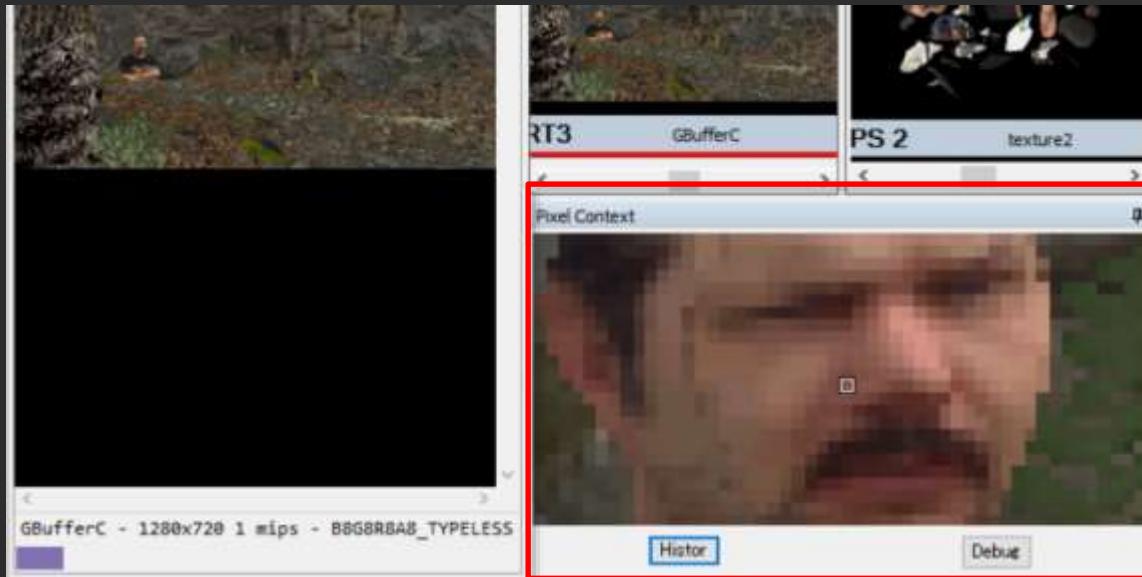


3549-3559	└ M_Rob Rob-Clean_Smoothed	M_Rob Rob-Clean_Smoothed	3
3559	└ DrawIndexed(613458)	DrawIndexed(613458)	3
3571-3572	└ Untextured Rob-Clean_Smoothed	Untextured Rob-Clean_Smoothed	3
3583-3591	└ HillTree_01_Branches_2_Mat	HillTree_01_Branches_2_Mat	1
3593-3597	└ HillTree_01_Branches_2_Mat	HillTree_01_Branches_2_Mat	2
3599-3603	└ HillTree_01_Branches_2_Mat	HillTree_01_Branches_2_Mat	8
3605-3609	└ HillTree_01_Branches_2_Mat	HillTree_01_Branches_2_Mat	2
3612-3720	└ Dynamic	Dynamic	2



Texture Viewer

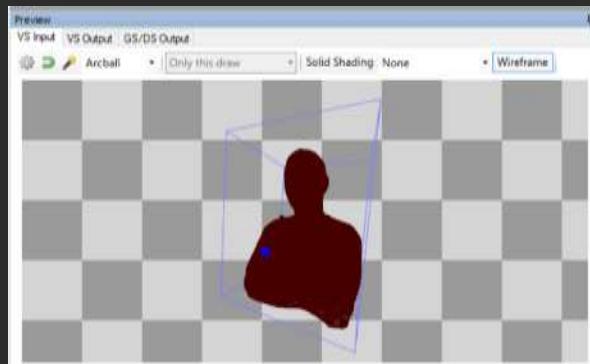
選んだRenderTargetのピクセルの描画履歴を追うことが可能。
RenderTargetのみたいあたりに、右ボタンクリックで、Pixel Contextのズーム画面に。
更に正確には PixelContext画面で矢印キーで、ピクセルを選択



	Tex Before	Tex After
> Scene ...		
> Static EBaseF		
> M_Photoscar	R: 0.00 G: 0.00 B: 0.00 A: 0.00	R: 0.00 G: 0.00 B: 0.00 A: 0.00
+ EID 3224	D: 0.01057 S: 0x00	D: 0.01057 S: 0x00
DrawIndexed(4)		
Depth test fail		
1 Fragment...	D: 0.01057 S: 0x00	D: 0.01057 S: 0x00
> Scene ...		
> Static EBaseF		
> M_Photoscar	R: 0.00 G: 0.00 B: 0.00 A: 0.00	R: 0.00 G: 0.00 B: 0.00 A: 0.00
+ EID 3266	D: 0.01057 S: 0x00	D: 0.01057 S: 0x00
DrawIndexed(4)		
Depth test fail		
2 Fragment...	D: 0.01057 S: 0x00	D: 0.01057 S: 0x00
> Scene ...		
> Static EBaseF		
> M_Photoscar	R: 0.00 G: 0.00 B: 0.00 A: 0.00	R: 0.00 G: 0.00 B: 0.00 A: 0.00
+ EID 3298	D: 0.01057 S: 0x00	D: 0.01057 S: 0x00
DrawIndexed(6)		
Depth test fail		
1 Fragment...	D: 0.01057 S: 0x00	D: 0.01057 S: 0x00
> Scene ...		
> Static EBaseF		
> M_Rob Rob-C	R: 0.00 G: 0.00 B: 0.00 A: 0.00	R: 0.5098 G: 0.32549 B: 0.28235 A: 0.25098
+ EID 3559	D: 0.01057 S: 0x00	D: 0.01057 S: 0x00
DrawIndexed(6)		
1 Fragment...	D: 0.01057 S: 0x00	D: 0.01057 S: 0x00

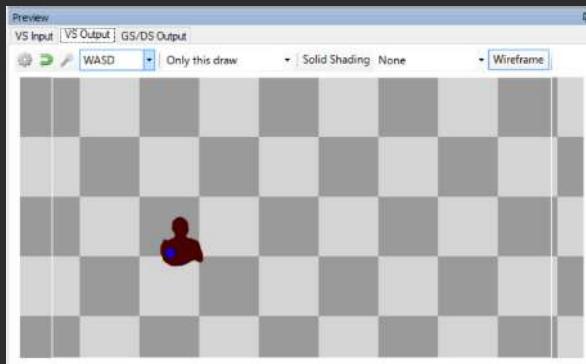
Mesh Output

そのDrawCallで呼び出された
メッシュを描画してくれる



VS Input

そのメッシュが、画像のどのあたりにレンダリング
されるのかも描画してくれる



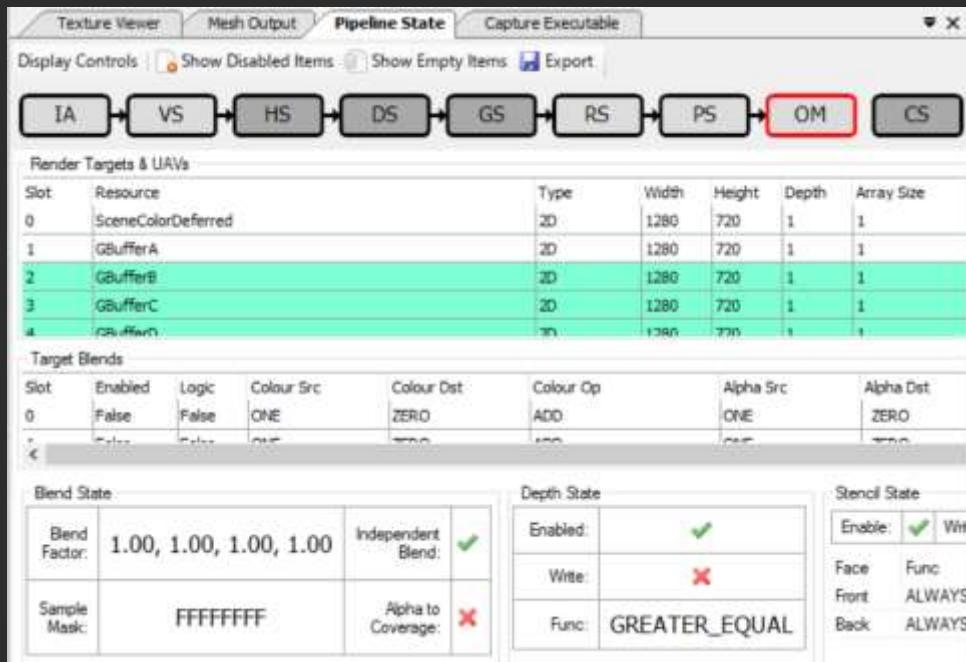
VS Output



Texture
Viewerと比較

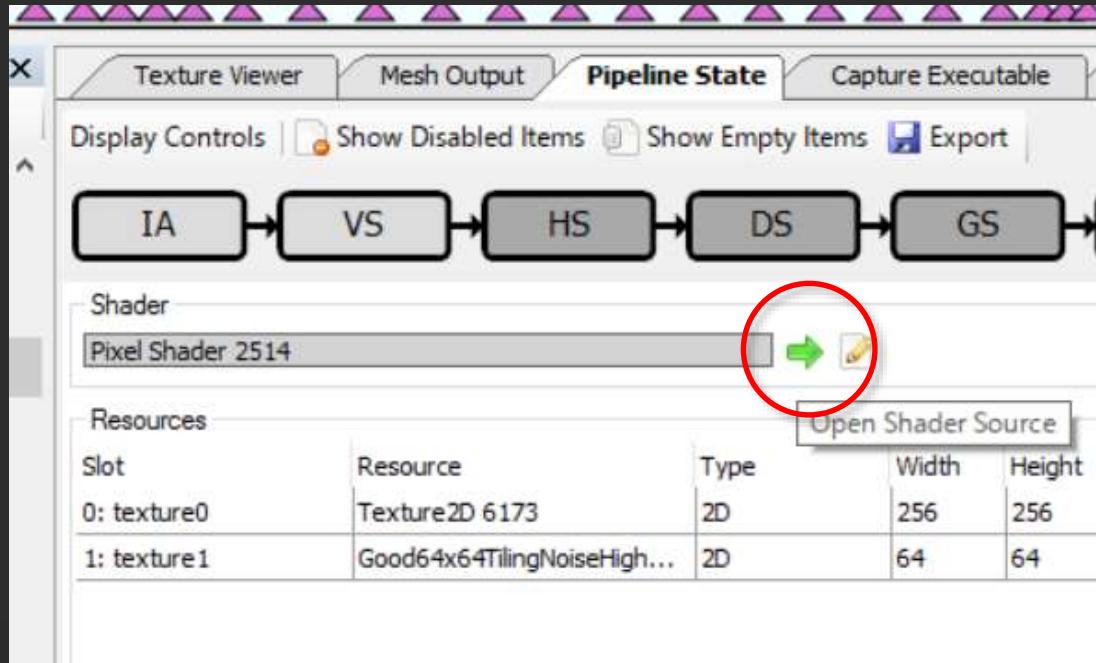
Pipeline State

- そのDrawCallの各種設定などがわかる
- 例えば、
 - Depthの設定
 - Rasterizerの設定



Pipeline State

Shaderも見ることがきる。



Disassembly

```
36: 20: ishl r0.z, r0.z, l(1)
37: 21: iadd r0.y, r0.z, r0.y
38: 22: udiv null, r0.y, r0.y, l(5)
39: 23: utof r0.y, r0.y
40: 24: add r0.x, r0.x, r0.y
41: 25: add r0.y, v2.w, -cb1_v10.x
42: 26: div_sat r0.y, r0.y, cb1_v10.y
43: 27: sample_b(texture2d)float,float,float,
44: 28: mad r0.y, r0.y, -r0.z, r0.z
45: 29: mad r0.x, r0.x, l(0.166667), r0.y
46: 30: add r0.x, r0.x, l(-0.833300)
47: 31: lt r0.x, r0.x, l(0)
48: 32: discard_nz r0.x
49: 33: mov o0.xyzw, l(0, 0, 0, 0)
50:
51: 34: ret
```



Tips

RenderDocでソースを見る方法

- r.Shaders.KeepDebugInfo=1
をConsoleVariables.iniに記述
- “-d3ddebug”オプションをエ
ディタ起動
- シェーダコンパイルが再度すべ
てに走るがRenderDocでソ
ースが見れるように

```
3512
3513
3514 float4 BloomThreshold;
3515
3516
3517 void MainPS(
3518     noperspective float4 UVAndScreenPos : TEXCOORD0,
3519     nointerpolation float InExposureScale : TEXCOORD1,
3520     out float4 OutColor : SV_Target0)
3521 {
3522     float2 UV = UVAndScreenPos.xy;
3523
3524     float4 SceneColor = Texture2DSample(PostprocessInput0, PostprocessInput0Sampler,
3525
3526
3527     SceneColor.rgb = min(float3(256 * 256, 256 * 256, 256 * 256), SceneColor.rgb);
3528
3529     float3 LinearColor = SceneColor.rgb;
3530 }
```

RenderDocまとめ

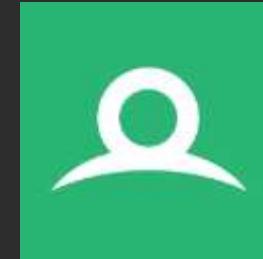
1フレームの描画をキャプチャし、

- どんな順に
- どんなオブジェクトが
- どんなテクスチャ/シェーダで。。。

レンダリングしているかがわかる便利ツール

※Windows限定

RenderDoc



本講演の目次

1. はじめに
2. 基本のプロファイリングツール 1
3. プロファイルの流れ
4. 基本のプロファイリングツール 2
- 5. Showcase**
6. まとめ&おまけ

Showcase



Foliageのプロファイリングをデモしてみます
(資料は別途公開します)

本講演の目次

1. はじめに
2. 基本のプロファイリングツール 1
3. プロファイルの流れ
4. 基本のプロファイリングツール 2
5. Showcase
6. まとめ&おまけ

まとめのまえに。。

おまけ

各PlatformのProfiler

各コンソールにもRenderDocみたいななのないの？

PC

Console



各コンソールにもRenderDocみたいななのないの？

PC

Console



PS4	Razor
XBOX	Pix

などなど
各プラットフォームが用意してくれてます

Q. プラットフォームのプロファイラを
使ったほうがいいの？

A.

(テクニカルよりの方ならば)

絶対使った方が良いです

チートです

プラットフォーム専用プロファイルのできること

- RenderDocと同様の機能 (フレームのCapture&Replayシステム)
- ハードウェアレベルのTrace機能

ハードウェアレベルのTrace機能でわからること



その描画が重たい理由が一発でわかります

Wavefront occupancy

Event timing

Pipeline state

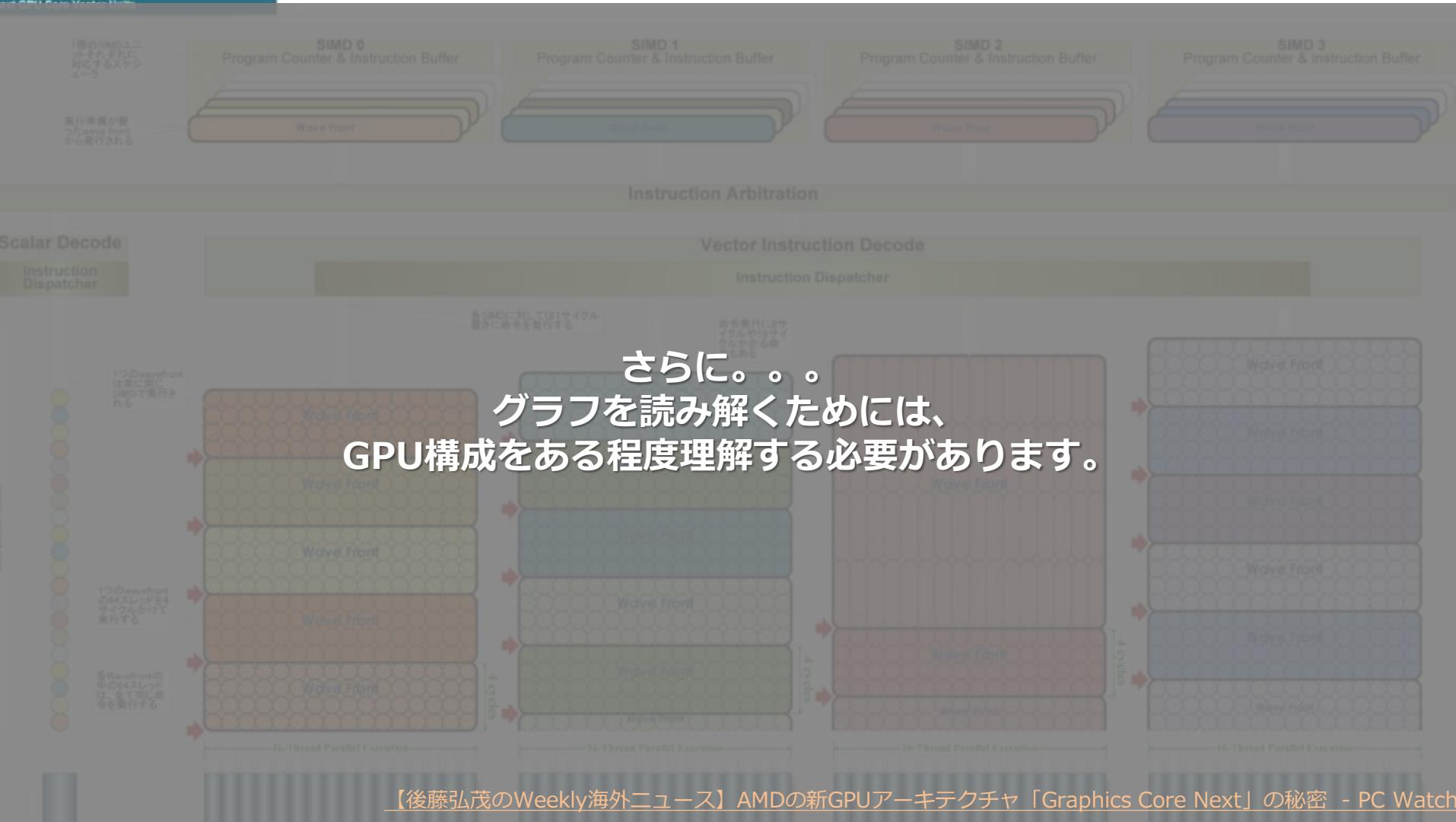
Color by API shader stage

GCN shader stages

Zoom to selection

Reset zoom





自習するのは大変そう。。。。

プラットフォーム別のUE4勉強会を開催します！

UE4ライセンシの方々には日程が決まり次第別途ご連絡いたします
勿論ですが、各プラットフォームのNDAライセンスをお持ちの方限定です



10月末、11月頭
東京大阪二箇所で開催



11月末or12月上旬
東京開催

まとめ

本日の内容

UE4のプロファイリングツールを俯瞰で眺めながら、
UE4上でどのように
プロファイルしていくのかを見てきました

グラフィクスプロファイル全体の流れ

Step.1
Game? Draw? GPU?

Stat unit / stat unit graph

Game

Draw

GPU

Step.2
どの部分が重たい？

**Stat scenerendering
Stat dumpframe...**

**profilegpu
stat gpu**

Step.3
各処理のコマンドを
使って理由を調査

Light?

Particle?

Static Mesh?

Post Process?

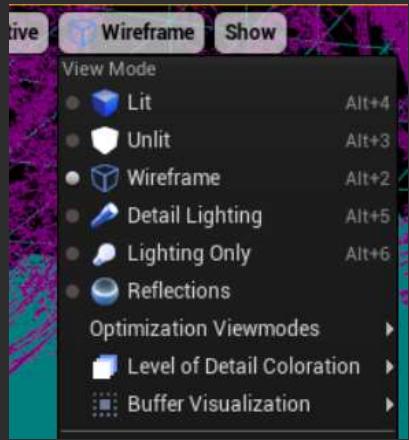
stat LightRendering, Foliage,,,

**r.XXXXXX,
etc.**

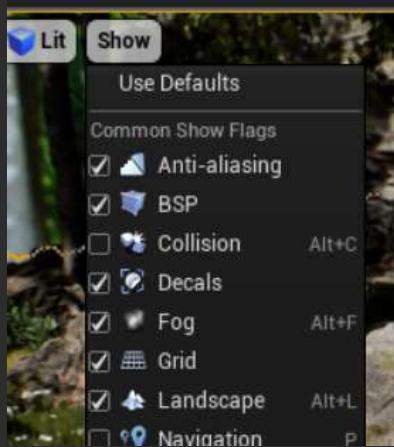
補助コマンド

show, freezeframe, toggledebugcamera, etc.

3つのEditor機能やプラグイン



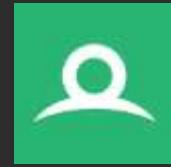
Viewmode



Show

Object	Actor(s)	Type
HillTree_02	4 Actors	Static Mesh
SM_Cliff01	2 Actors	Static Mesh
SM_GroundRevealR	8 Actors	Static Mesh
LargeVolcanicRock_2	2 Actors	Static Mesh

Statistics



RenderDoc

なぜプロファイル？

プロファイルしない最適化は危険

最適化の前に、開発初期から、日常的に、
実機で、
プロファイルする癖を

UNREAL FEST EAST 2017

パシフィコ横浜 会議センター 3F

2017/10/08(日)

本当に、本当にありがとうございました！
次回は10月8日のこの場所で！