



UNREAL
ENGINE

UE4で多数のキャラクターを活かすためのテクニック

Epic Games Japan / Support Engineer
Ken Kuwano



#UE4CEDEC



はじめに

- SNS上での情報の公開はOK
- 本資料は講演後できるだけ早めに公開
- 本講演の内容はUE4.20で検証
- 中辛エンジニア向け



#UE4CEDEC





イメージ



現實

ActionRPGのSampleを使って
キャラクターをたくさん出してみよう



#UE4CEDEC



PC-Test 100体

www.BANDICAM.com

AUTOPLAY



62.66 FPS
16.13 ms
Pcoll: 16.13 ms
Coll: 7.23 ms
Solv: 1.69 ms
GH: 16.13 ms
SI: 0.00 ms
System Unoptimized

9,964

NPC : 100



PC-Test 1000体



NPC : 1,000

PC-Test 1000体

AUTOPLAY



PC-Test 10000体

AUTOPLAY



NPC : 10,074

話すこと

- CPUパフォーマンスの改善方法
 - 主にGameの負荷改善について
- 多くのキャラクターを出す際のポイント
 - 懸念点や負荷になりがちな点について
- キャラクターと関連する機能の仕組み
 - 処理の流れ、設定や実装などの細かい話について

CPU



話ないこと

- GPUパフォーマンスの改善方法
- キャラクターの作り方、Engine改造について
- このスライドの50%



本題に入る前に

認識合わせと理解を深めるため
キャラクターやCPUの動作についてのおさらい



おさらい

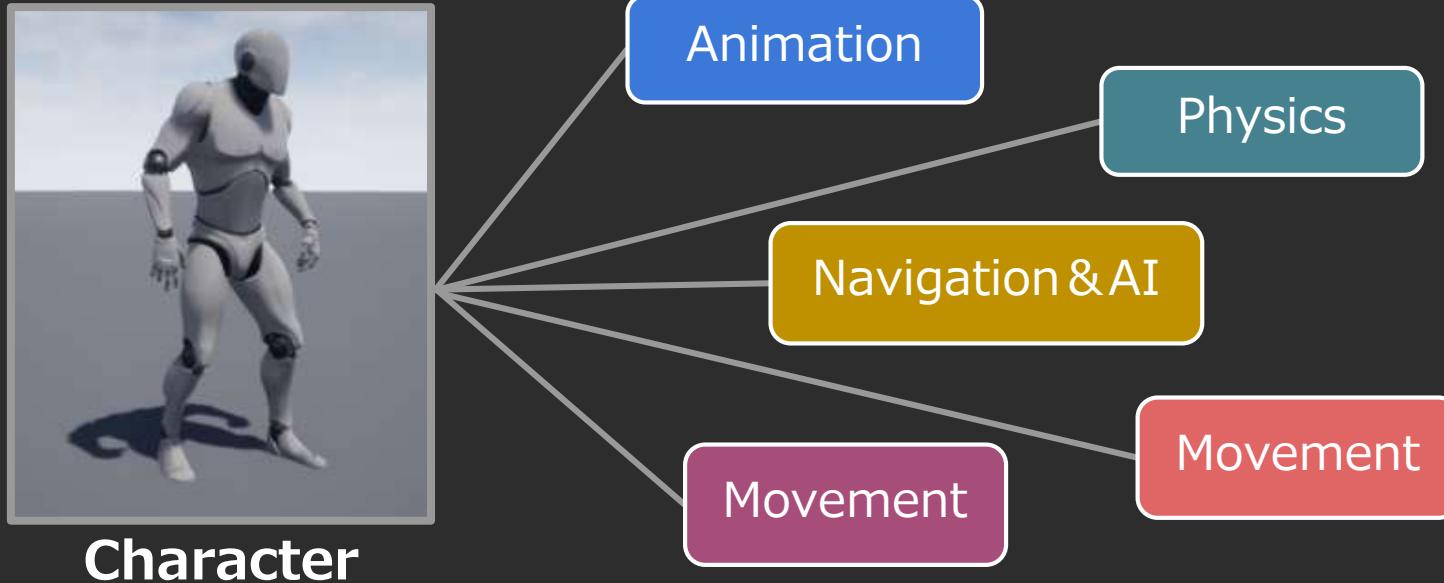
- Characterとは
- CPUの動作
- CPUのパフォーマンス計測



Characterとは



キャラクターに関連する機能



タイトルによってキャラクターが実現する内容が異なる

CPUの動作

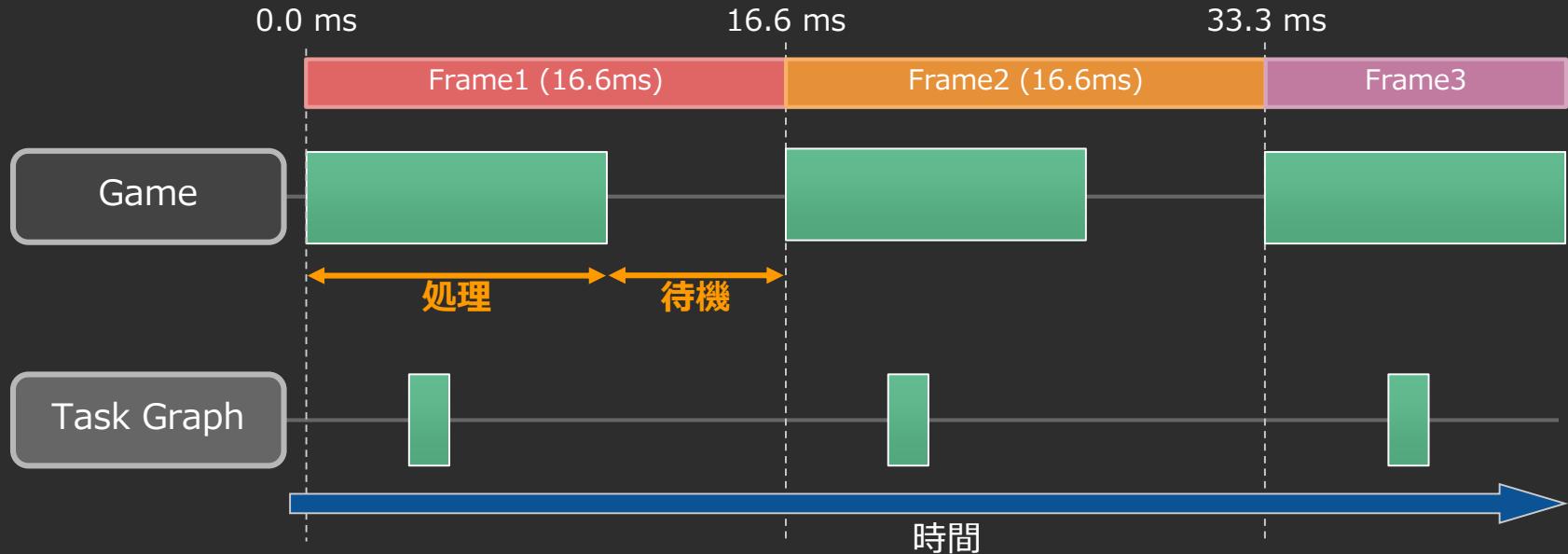
様々な機能を持つ複数のスレッドで動作

スレッド名	概要
Game	ゲームのメイン処理
Task Graph	Game の補助的役割、複数スレッド
Pool	Texture Streaming等、複数スレッド
Rendering	RHIコマンド(描画コマンド)を発行
RHI	RHIコマンドをGPUに転送

CPUで動作する代表的なスレッドと概要

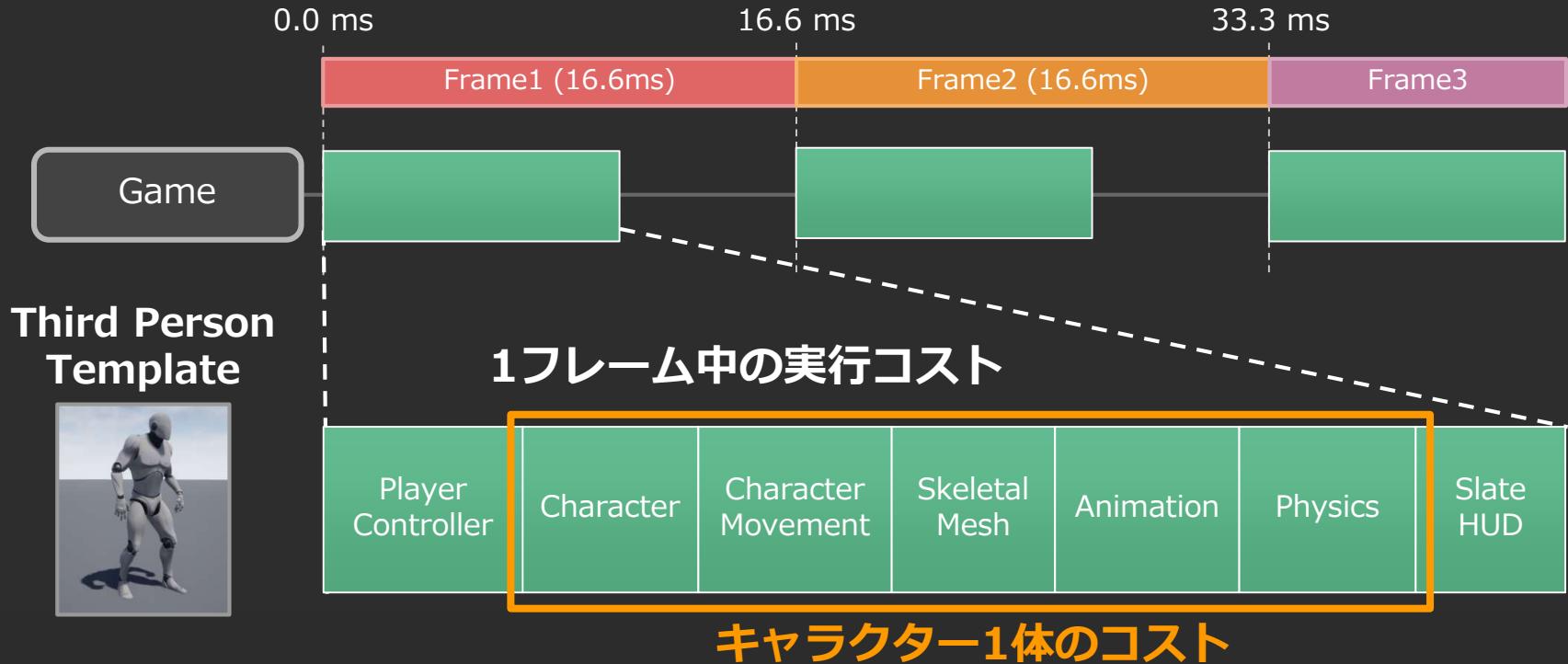


CPUの動作の流れ (60fpsのモデルケース)



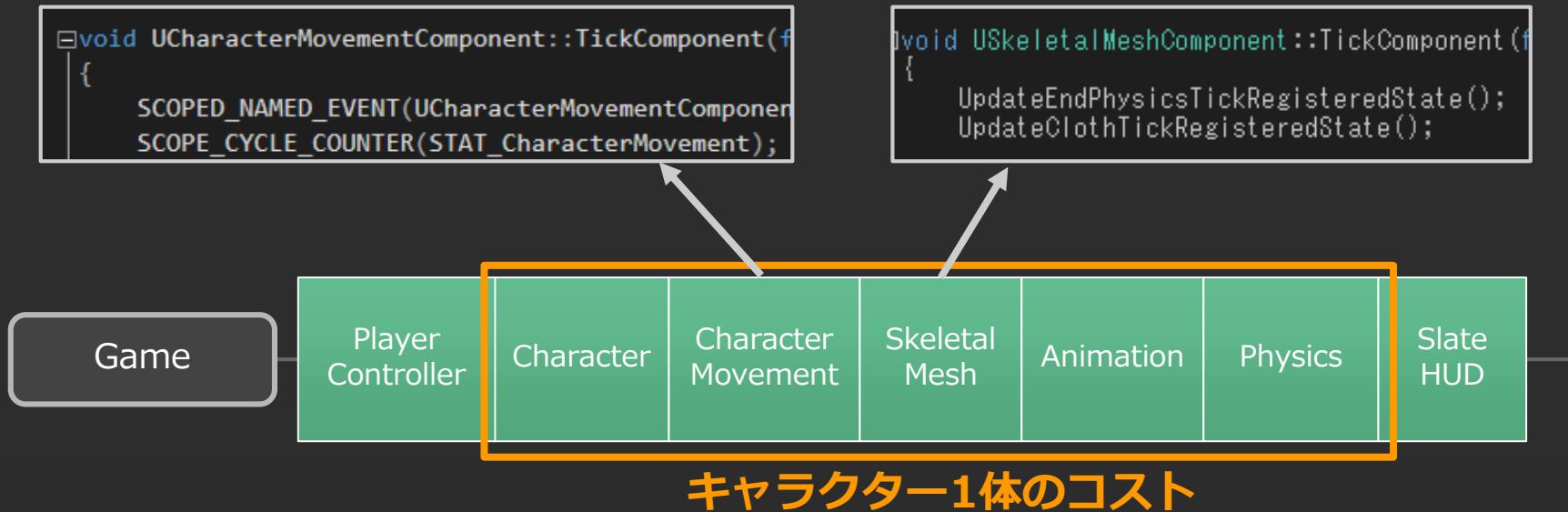
時間内に処理が収まっているので遅延が無く理想的

CPUの動作の流れ (60fpsのモデルケース：詳細)



CPUの動作の流れ (Tick)

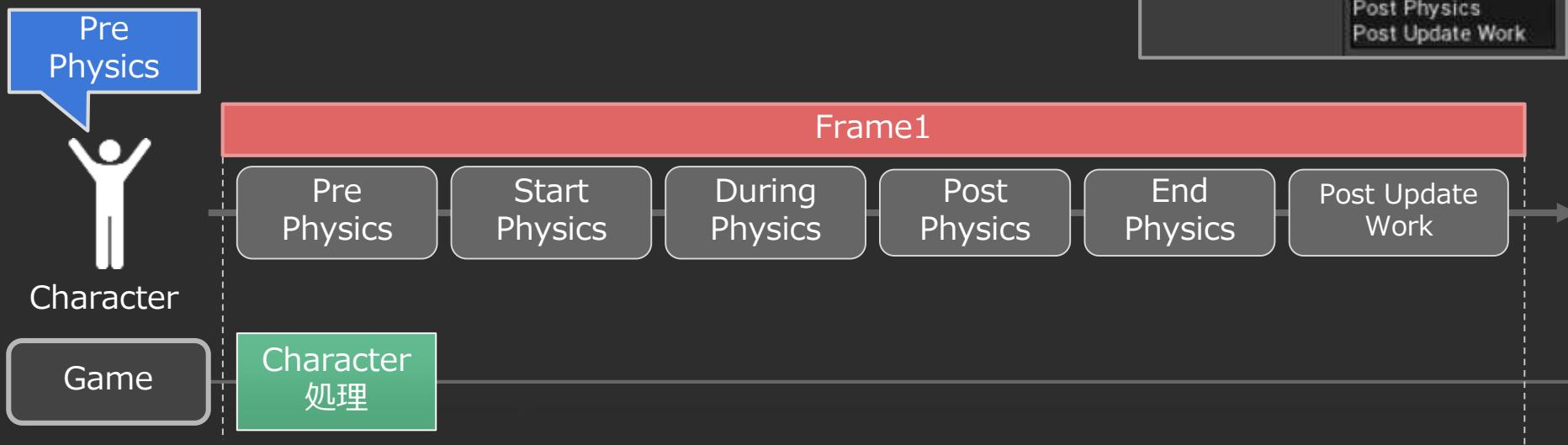
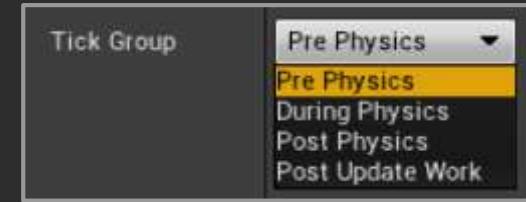
ActorやComponentが毎フレーム実行する処理



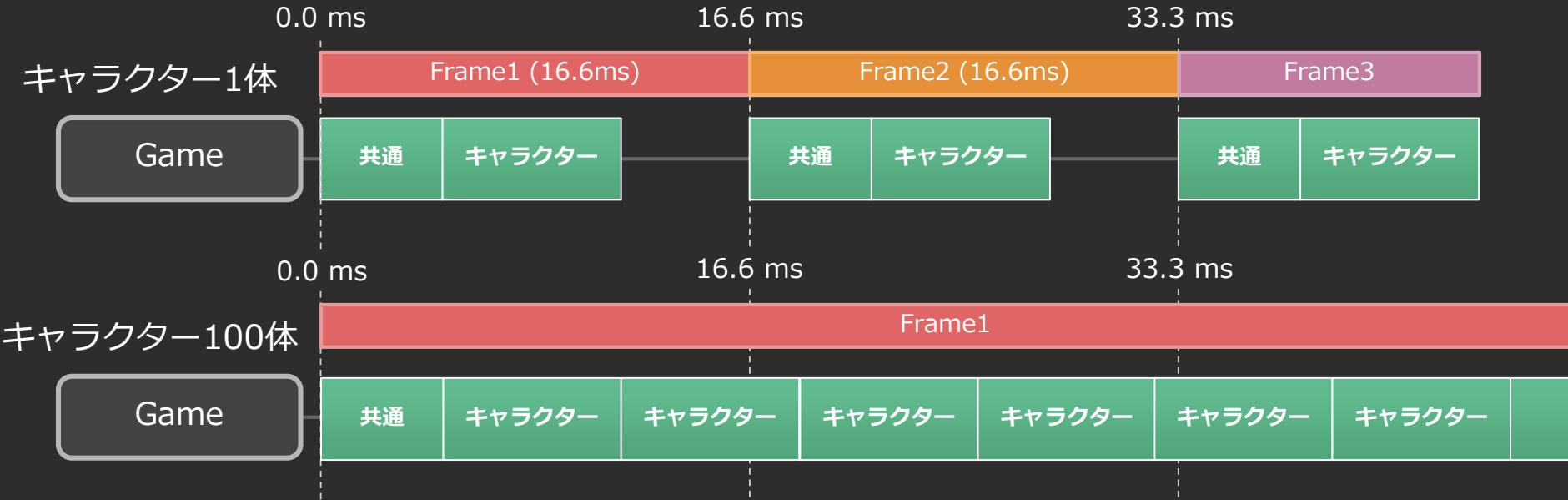
CPUの動作の流れ (Tick Group)

Tick処理を行う順番を規定するグループ

Actorなどが自身の処理タイミングを指定可能



CPUの動作の流れ (多数のキャラクターを出すケース)



多数のキャラクターを扱う場合は
キャラクター1体のコスト削減に注力する必要がある

CPUのパフォーマンス計測

- 計測方法
 - コンソールコマンド
 - プロファイリングツール (コンソールのみ)
- 計測条件
 - パッケージした**Testビルド**を使用
 - Epic Games LauncherからはTestビルドの作成不可
 - 一部のコンソールコマンドのみ使用可能
 - 出荷用ビルド(Shipping)に近い状態で計測可能



CPUパフォーマンスはTestビルドで計測しましょう



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

Twitter icon #UE4CEDEC



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	検証コードなし	Stat unit ぐらい	ほぼ同じ
Shipping		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。

Twitter icon #UE4CEDEC



昨年のセッションでも紹介していますのでご参考下さい

Twitter icon #UE4CEDEC



Testビルドでの制限を解除してデバッグする方法

Testビルドでのログ、コンソールコマンドを許可する方法が紹介されています

2018-06-15

Testビルドでも色々計測できるようにする

▶ UE4

Developmentビルドで正確なCPU負荷を図るのは難しい

UE4では、日々の開発はDevelopmentビルドで行い、最終的にリリースするパッケージはShippingビルドで作成するのが基本的なフローとなります。しかし、Developmentビルドはデータの整合性の確認であったりデータを集計していたりと、実際にリリースされるゲームでは除外されるデバッグ用コードが沢山含まれています。結果、DevelopmentビルドのCPU処理負荷はShippingビルドに比べて著しく重たく、実際にリリースされるゲームのCPU処理負荷を計測するのには適していません。※Developmentビルドでは様々なUE4のstat機能が使えます。それらを駆使して処理内容を理解したり投入されているオブジェクト数などを知ることに適しております。

TestビルドでもNamedEventsを出力する方法(4.19から)

stat dumpnameというコマンドで、各スレッドのCPU使用率を概要的に表示してくれます。これをTestビルドでもONにすることができます。

```
*#ifndef ENABLE_STATNAMEDEVENTS
#define ENABLE_STATNAMEDEVENTS 0
#endif
```

注意としてこのdefineはUE4のstat機能を有効にしてしまうのでそのままにしてしまうとDevelopmentビルドでは性能が悪くなくなってしまいます。下の様にTestで開んでからにしておいてください。

```
#ifndef ENABLE_STATNAMEDEVENTS
#if UE_BUILD_TEST
#define ENABLE_STATNAMEDEVENTS 1
#else
#define ENABLE_STATNAMEDEVENTS 0
#endif
#endif
```

<http://darakemonodarake.hatenablog.jp/entry/2018/06/15/225956>



#UE4CEDEC



出荷用ビルドに近い状態で計測可能



Third Person Templateで1000体を出した時の
DevelopmentとTestのパフォーマンスを比較

Development

Frame:	61.70 ms
Game:	55.32 ms
Draw:	61.52 ms
GPU:	45.84 ms
RHI:	16.89 ms
DynFles:	OFF

Test

Frame:	47.21 ms
Game:	38.06 ms
Draw:	40.08 ms
GPU:	40.25 ms
RHI:	12.47 ms
DynFles:	OFF

CPU

製品に近い環境で計測することで
必要以上な負荷削減を減らす



過剰な最適化を減らすこと
で
クオリティを維持

おさらい：まとめ

- Characterとは
タイトルによってキャラクターが実現する内容が異なる
- CPUの動作
多数のキャラクターを扱う場合はキャラクター1体のコスト削減に注力
- CPUのパフォーマンス計測
過剰な最適化を減らすことでコンテンツのクオリティを維持

実現する機能に併せてキャラクターを適切に最適化



アジェンダ

- Animation
- Physics
- Navigation & AI
- Movement
- Networking
- その他



#UE4CEDEC



各章の構成

Animation

タイトル



改善ポイント一覧 (Animation)

- 多数のキャラクターがアニメーションを行なう
- 移動やアニメーションに伴うBounds更新 1
- 移動やアニメーションに伴うBounds更新 2
- アニメーションのマルチスレッド処理
- ステートマシンの検索コスト削減



改善一覧



#UE4CEDEC

CharacterとAnimation



関連



おさらい : Animation (処理の流れ)



おさらい



多数のキャラクターがアニメーションを行う



- 負荷ポイント

多くのキャラクターを出すとキャラクターのアニメーション更新コストが高む



ーションの更新コストが増大

負荷ポイント

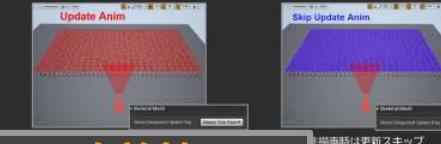


多数のキャラクターがアニメーションを行う



- 改善策

非描画キャラクターのアニメーション更新をスキップ



改善策



[フォーマット] ユースケース、改善を行うケースの概要



負荷ポイント

- 何が行われているか、負荷ポイント 等

改善点

- 改善方法、改善されることで得られる効果 等

備考

- 改善を適用した際の副作用、注意点 等

概要図



#UE4CEDEC



負荷ポイント／改善策

多数のキャラクターがアニメーションを行う

- 負荷ポイント

多くのキャラクターを出すとキャラクターのアニメーション更新コストが嵩む



多数のキャラクターを出すと...



アニメーションの更新コストが増大



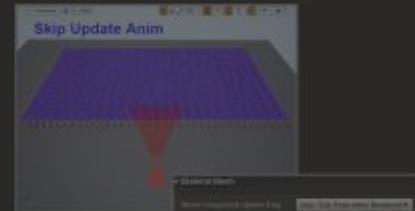
多数のキャラクターがアニメーションを行う

- 改善策

非描画キャラクターのアニメーション更新をスキップ



非描画時もアニメーション更新



非描画時は更新スキップ

Twitter icon #UE4CEDEC



Twitter icon #UE4CEDEC



評価



大きく負荷を減らすことができる
多数のキャラクターを出す場合は特に注意



それなりの負荷を減らすことができる
汎用的に使用することが可能



負荷を減らすことができる



#UE4CEDEC



誤解が無いために

- **評価**
 - 実際のパフォーマンスへの影響はコンテンツの内容に依存
- **負荷ポイント／改善点**
 - 「減らすことを検討」などと記載している部分については「多数のキャラクターを出すケース」で負荷になりがちな部分であって、使用を禁止するものではありません



アジェンダ

- **Animation**
- Physics
- Navigation & AI
- Movement
- Networking
- その他



#UE4CEDEC



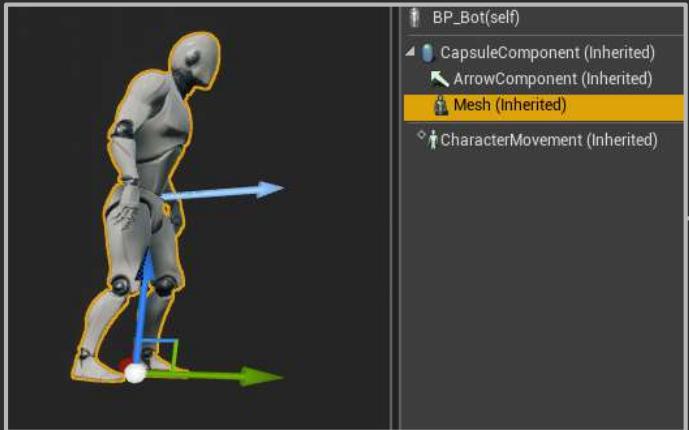
Animation



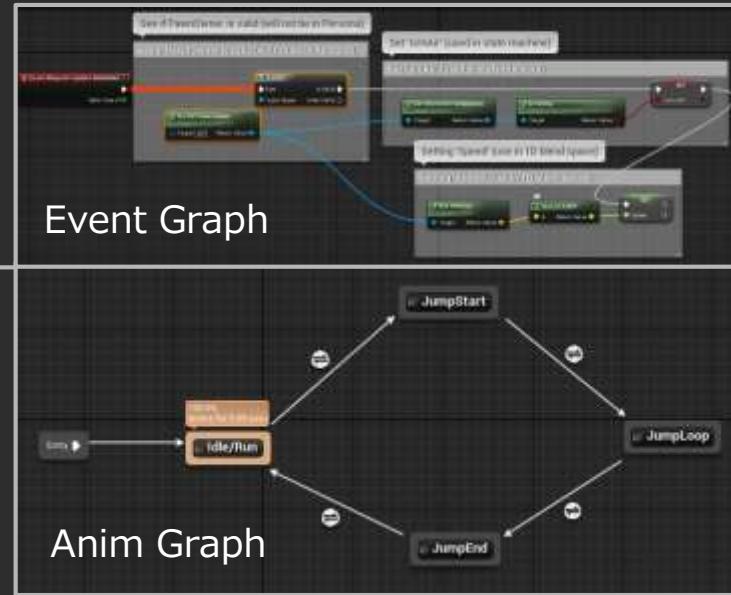
#UE4CEDEC



Character & Animation



Character



おさらい : Animation



UNREAL
ENGINE

アニメーションシステム総おさらい

Epic Games Japan / Support Engineer
Yutaro Sawada

Twitter icon #UE4CEDEC



しっかり理解して機能を使うために
総おさらい

Twitter icon #UE4 | @UNREALENGINE

6

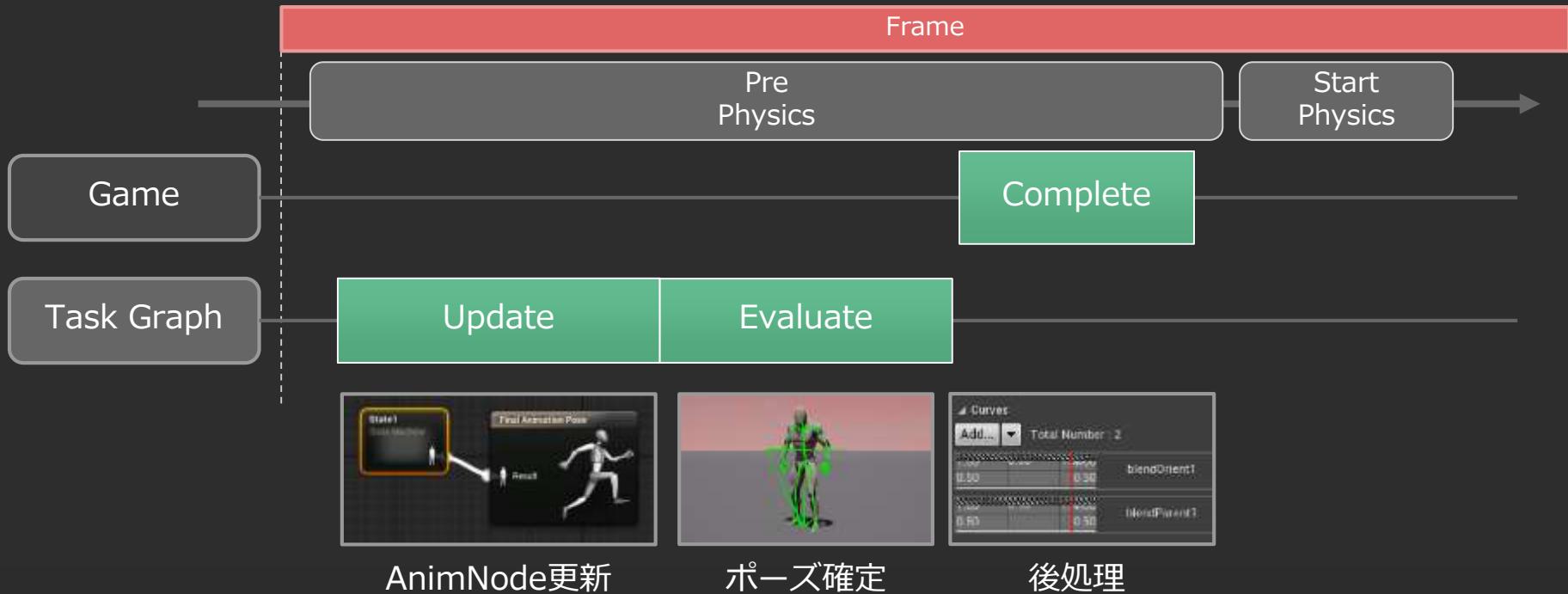
UNREALENGINE.COM The Unreal Engine logo, which consists of a stylized white 'U' character enclosed within a circle.

今年のセッションでも紹介していますのでご参考下さい

Twitter icon #UE4CEDEC



おさらい : Animation (処理の流れ)



改善ポイント一覧 (Animation)

- 多数のキャラクターがアニメーションを行う
- 移動やアニメーションに伴うBounds更新 1
- 移動やアニメーションに伴うBounds更新 2



改善ポイント一覧 (Animation)

- ・ アニメーションの実行コスト削減 1
- ・ アニメーションの実行コスト削減 2
- ・ アニメーションの間引きによる更新コスト削減
- ・ アニメーションの評価コスト削減
- ・ アニメーションが不要なメッシュ
- ・ LODによるメッシュのリダクション
- ・ ステートマシンの検索コスト削減
- ・ アニメーションのマルチスレッド処理



多数のキャラクターがアニメーションを行う



負荷ポイント

- 每フレームアニメーションの更新を行うがキャラクターが多いとアニメーションのコストが嵩む

改善点

- **非描画キャラクターのアニメーションの更新をスキップ**することでアニメーション更新コストを削減

備考

- オフスクリーンのキャラクターのアニメーション更新をスキップするので画面内にいる場合は無効となる



Skeletal Mesh

Mesh Component Update Flag

Only Tick Pose when Render



#UE4CEDEC



多数のキャラクターがアニメーションを行う

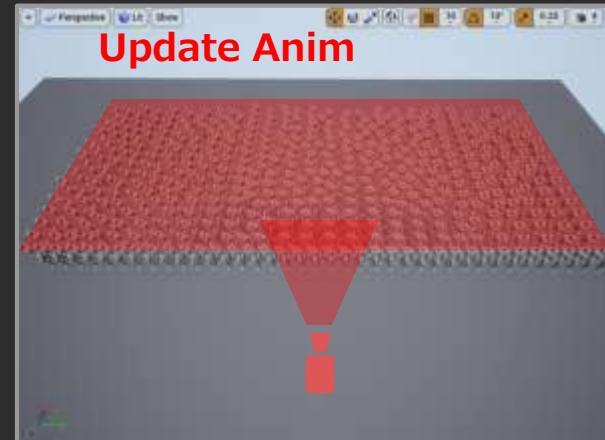


- 負荷ポイント

多くのキャラクターを出すとキャラクターのアニメーション更新コストが嵩む



多数のキャラクターを出すと...



アニメーションの更新コストが増大



#UE4CEDEC

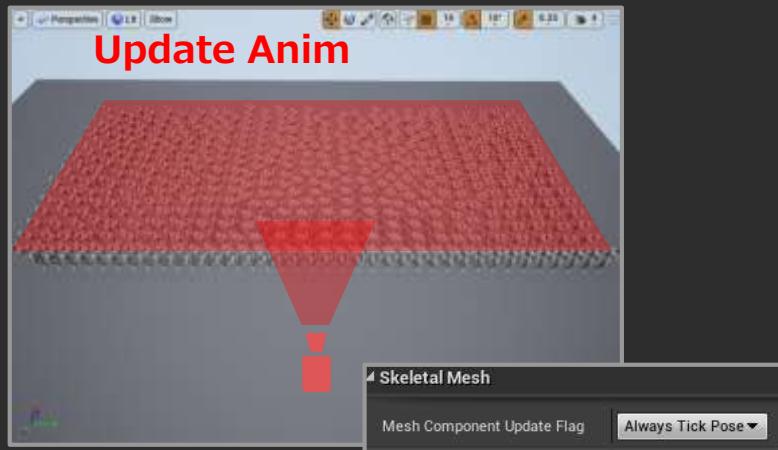


多数のキャラクターがアニメーションを行う

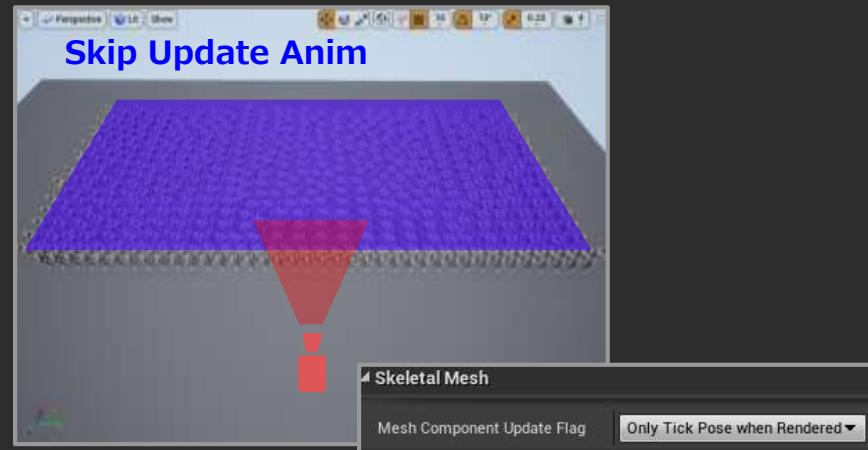


- 改善策

非描画キャラクターのアニメーション更新をスキップ



非描画時もアニメーション更新



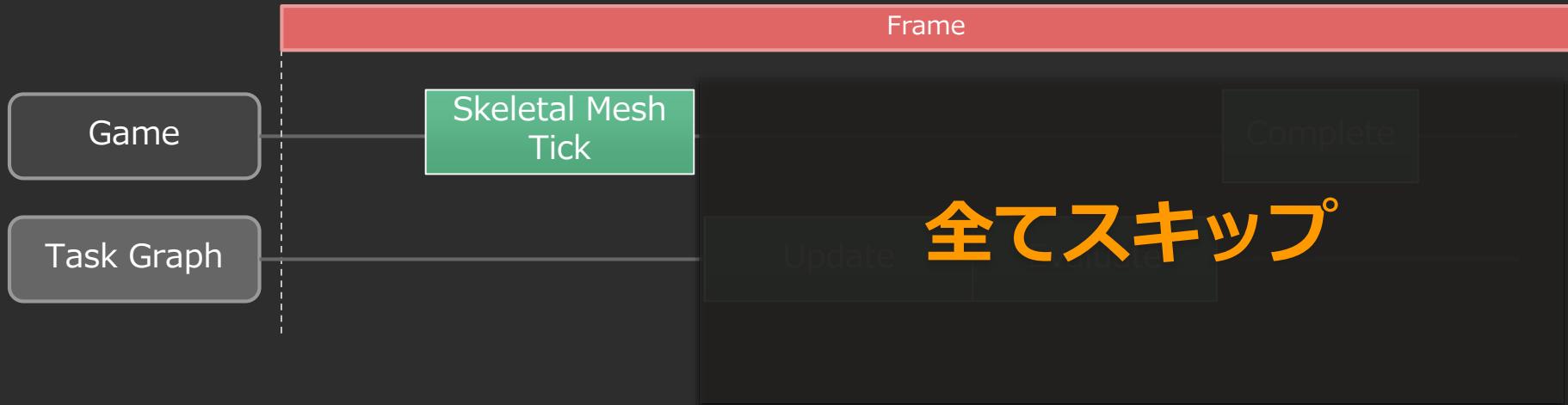
非描画時は更新スキップ



多数のキャラクターがアニメーションを行う



アニメーションのスキップにより以降の処理コストを削減



移動やアニメーションに伴うBounds更新 1



負荷ポイント

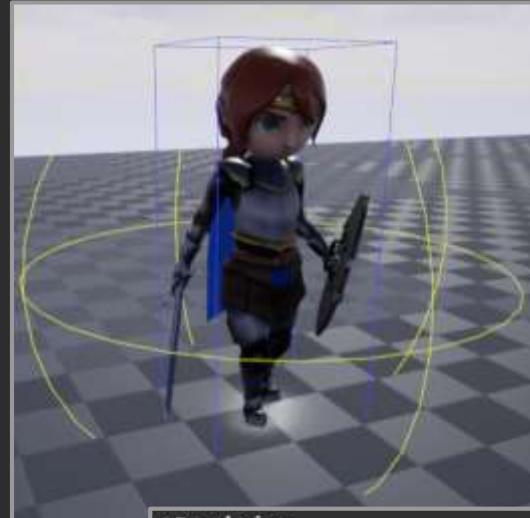
- 移動やアニメーションを行う際にキャラクターのBoundsの位置を更新するための計算コストが嵩む

改善点

- ComponentのBoundsの位置を親Componentと同じにすることでBoundsの計算コストを削減

備考

- Scene Componentの機能のため、殆どのComponentに適用することが可能
- Boundsの統合によりカリング範囲が変わるので注意**

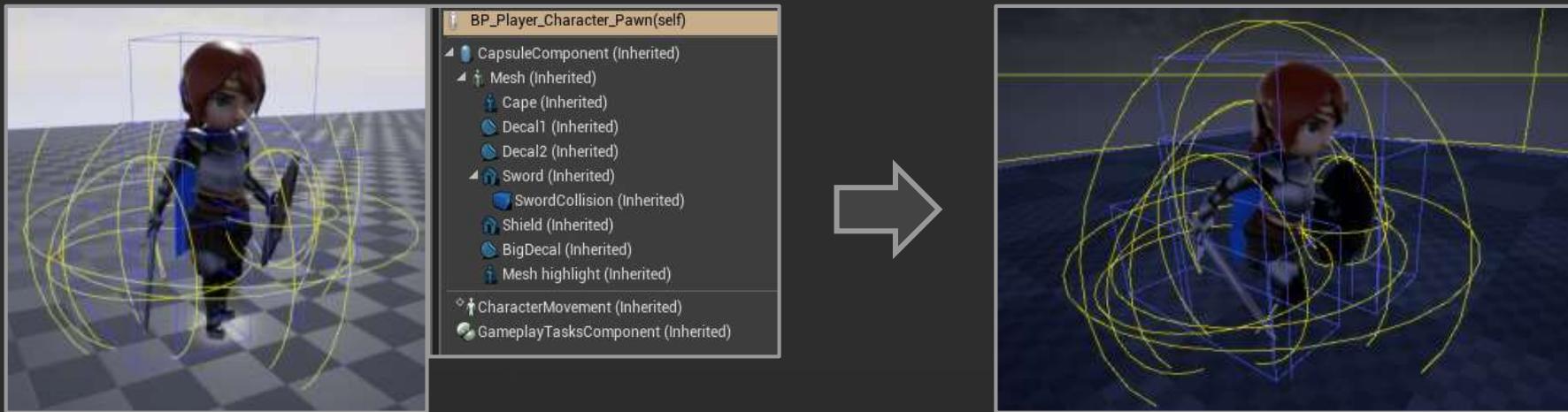


移動やアニメーションに伴うBounds更新 1



- 負荷ポイント

移動やアニメーション時にComponentのBounds計算が行われる
– Componentの数が多いほどBounds計算のコストが増加



移動やアニメーションに伴うBounds更新 1



● 改善策

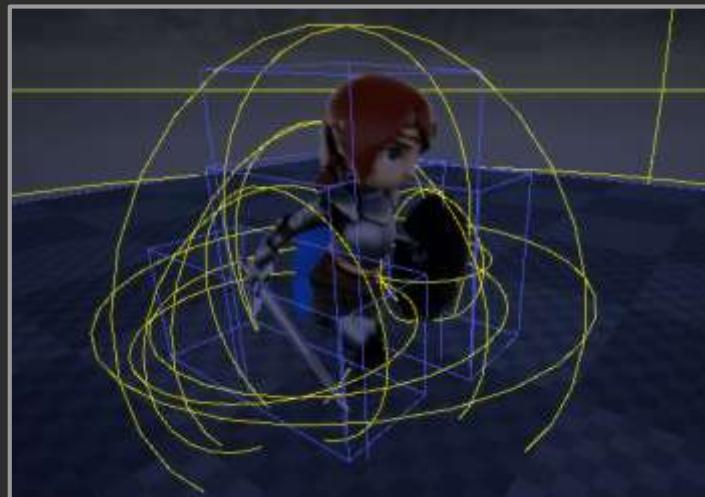
自身のBounds計算を行わず親のBoundsを使用することで計算コストを削減



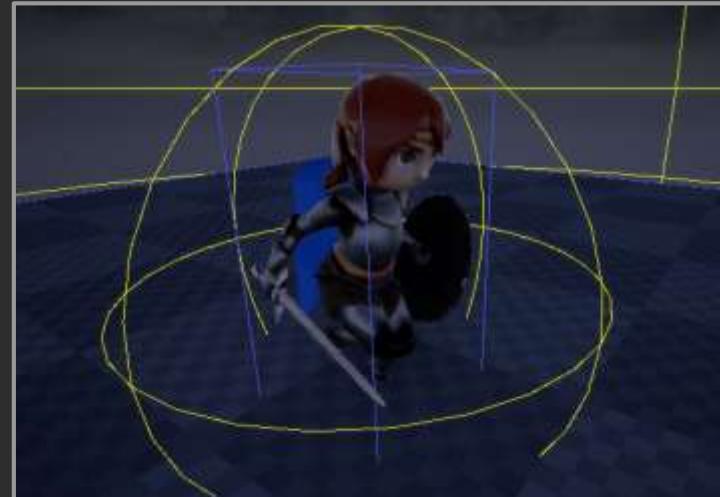
移動やアニメーションに伴うBounds更新 1



Boundsの計算を親Componentに統合することでコスト削減



自身のBoundsを計算



親のBoundsを使用



移動やアニメーションに伴うBounds更新 2



負荷ポイント

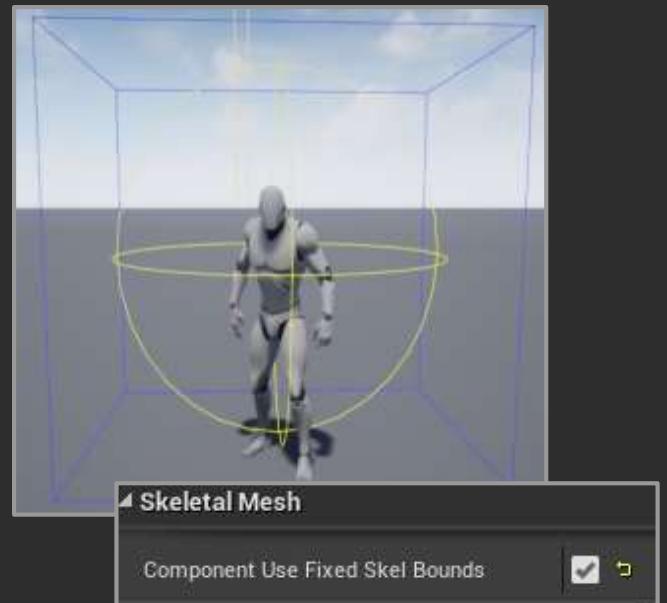
- 移動やアニメーションを行う際にキャラクターのBoundsの位置を更新するための計算コストが嵩む

改善点

- Boundsの計算にSkeletal Mesh自身が持つ固定Boundsを使用することでBoundsの計算コストを削減

備考

- Skeletal Mesh Component に適用可能
- Boundsの統合によりカリング範囲が変わるので注意**



#UE4CEDEC



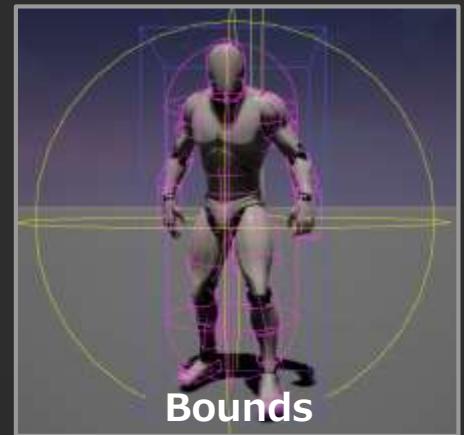
移動やアニメーションに伴うBounds更新 2



- 負荷ポイント

移動やアニメーション時にComponentのBounds計算が行われる

- Skeletal MeshはPhysics AssetからFitするBoundsを生成
- PhysicsのBody数が多いほど計算コストがかかる

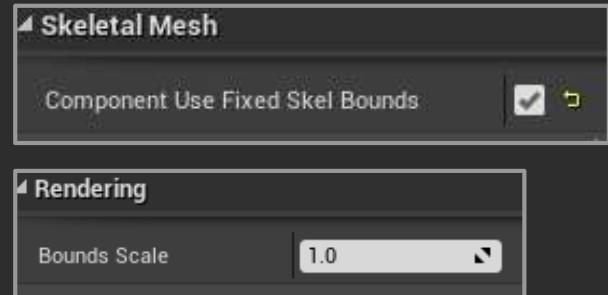
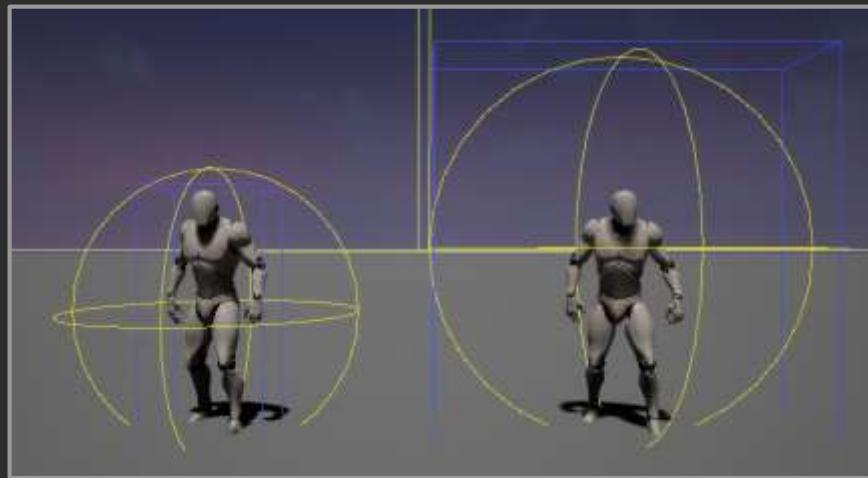


移動やアニメーションに伴うBounds更新 2



- 改善策

Skeletal Meshが持つ固定Boundsを使用することで計算コスト削減



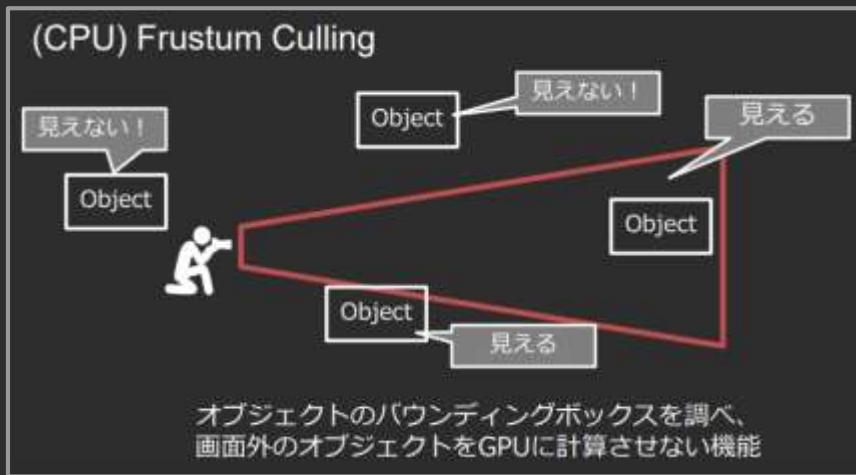
PhysicsAssetベース(左) と 固定Bounds使用(右) の比較



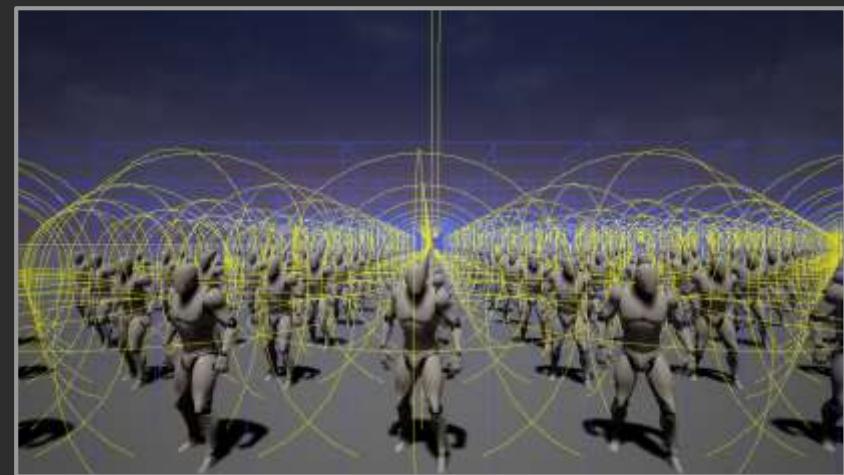
移動やアニメーションに伴うBounds更新 2



- Boundsが大きいとCullingされにくくなるのでGPU負荷に注意
 - 小さすぎても不用意にCullingされるので適切な調整を



Boundsによるカリング



Boundsが大きいとカリングされにくい



#UE4CEDEC



アニメーションのマルチスレッド処理



負荷ポイント

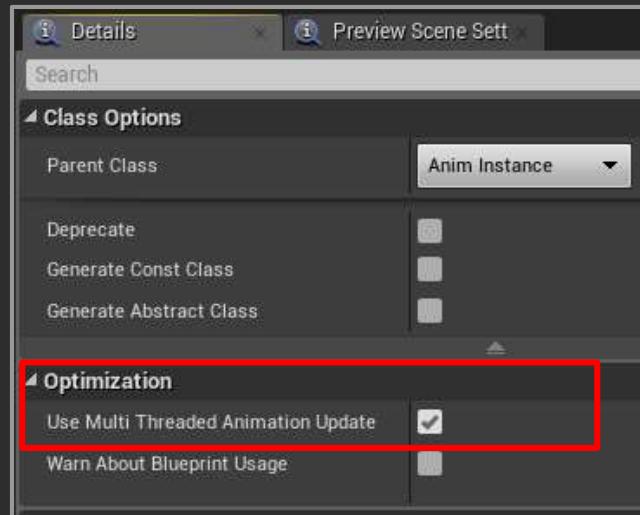
- アニメーションの更新をGame Threadで実行するとキャラクターが多いケースにおいてボトルネックになる

改善点

- TaskGraph Threadで更新を行うことでGame Threadの負荷を集中を避ける

備考

- デフォルトでは並列処理が有効になっている
- 並列処理が適用されないケースがあるので注意



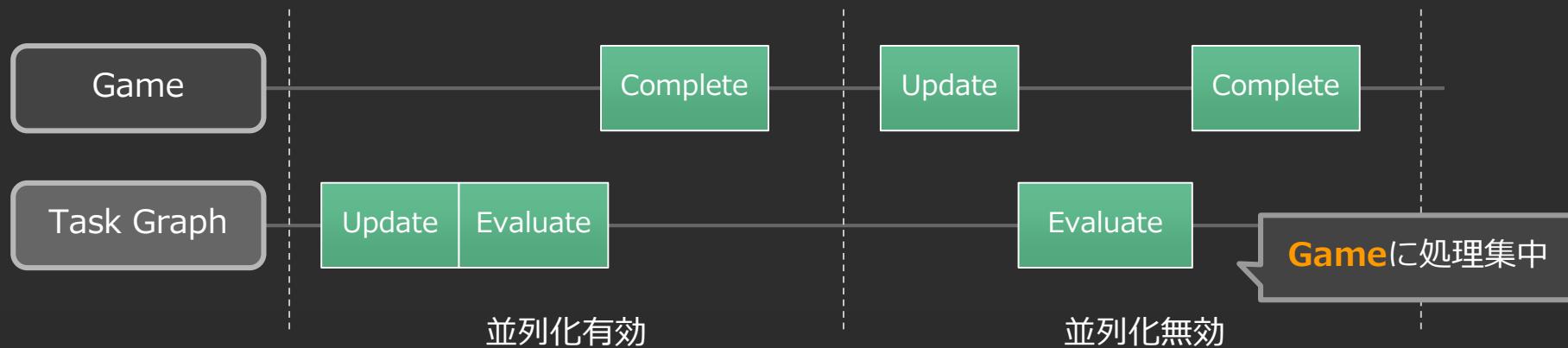
アニメーションのマルチスレッド処理



- ## ● 負荷ポイント

並列処理条件を満たさない場合はアニメーション更新Gameで実行

例：Root Motion設定で“**Root Motion from Everything**”を使用



アニメーションのマルチスレッド処理



- 改善策

並列処理条件を満たすように実装する必要がある

並列処理が可能な条件

- Project Settingsで並列更新が許可されている
- Animation Blueprintで並列更新が許可されている
- コンソールコマンドで並列更新が強制されている
- Tickで提供されるDelta Secondsが0.0以外
- ルートモーション("Root Motion from Everything")を使用していない



アニメーションのマルチスレッド処理



複数Task Graphで処理できればGameが効率的に処理可能
– 並列処理が無効になるとGameがボトルネックになる



アニメーションの実行コスト削減 1



負荷ポイント

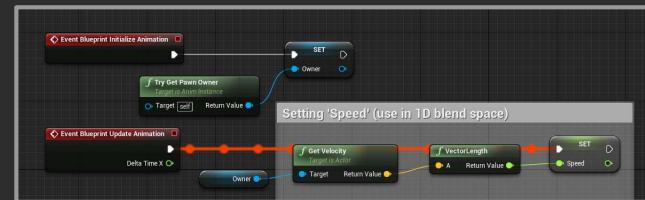
- Animation BlueprintでBlueprint Nodeを毎フレーム実行する場合、NodeやLogicが多いと実行コストがかかる

改善点

- Animation Blueprintで行う複雑な処理やアクセス頻度が高い処理はC++へ移行を検討

備考

- Native化により保守性が低下するためパフォーマンス上の負荷になる箇所からC++に移行するのが良い



```
void UAIAAnimInstance::NativeInitializeAnimation()
{
    OwnerPawn = TryGetPawnOwner();
}

void UAIAAnimInstance::NativeUpdateAnimation(float DeltaSeconds)
{
    if (OwnerPawn)
    {
        CurrentSpeed = OwnerPawn->GetVelocity().SizeSquared();
    }
}
```

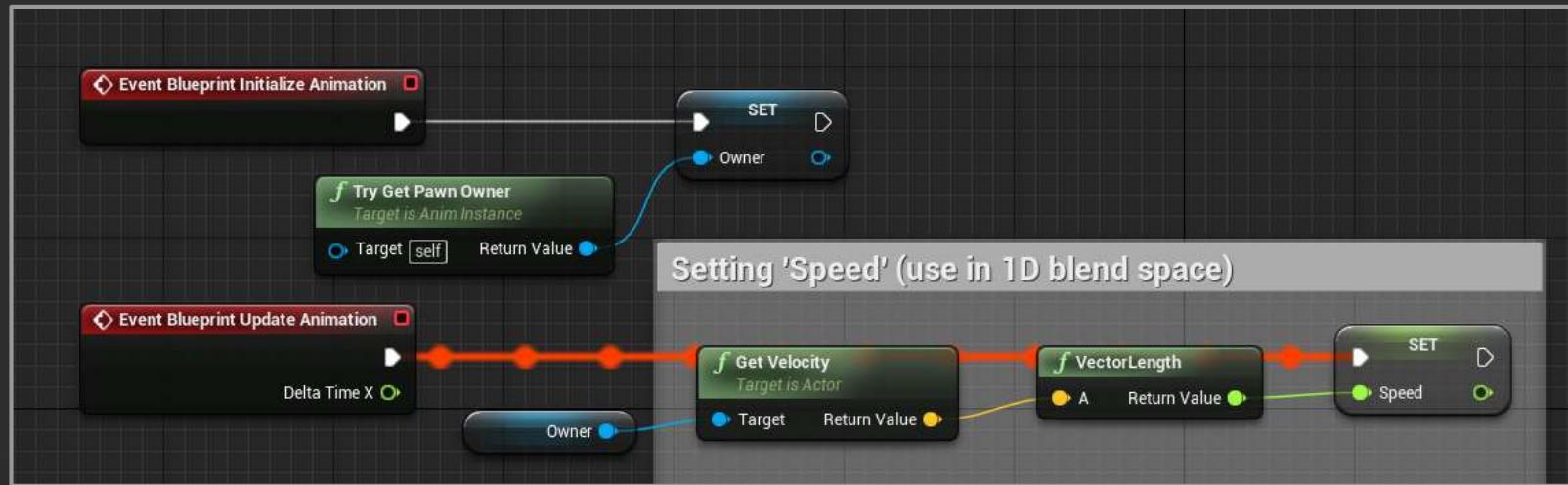


アニメーションの実行コスト削減 1



- 負荷ポイント

Blueprintでの処理はC++よりもVM(Blueprint呼び出し)コストがかかる

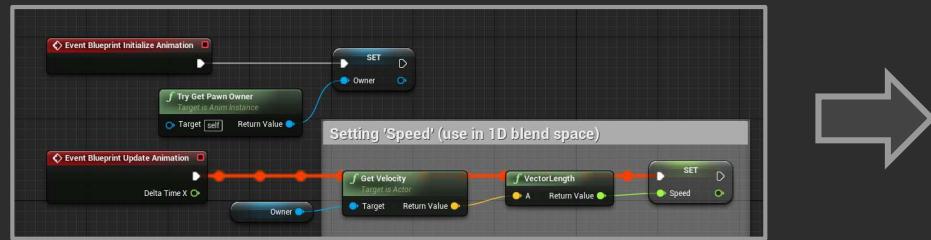


アニメーションの実行コスト削減 1



- 改善策

Blueprintでの処理はC++よりもVM(Blueprint呼び出し)コストがかかる
複雑なロジックやアクセス頻度が高い処理は出来るだけC++を利用



```
void UAIAAnimInstance::NativeInitializeAnimation()
{
    OwnerPawn = TryGetPawnOwner();
}

void UAIAAnimInstance::NativeUpdateAnimation(float DeltaSeconds)
{
    if (OwnerPawn)
    {
        CurrentSpeed = OwnerPawn->GetVelocity().SizeSquared();
    }
}
```

AnimBPをNative化することも可能だが、
Native化のコードも含むためパッケージサイズに影響



アニメーションの実行コスト削減 2



負荷ポイント

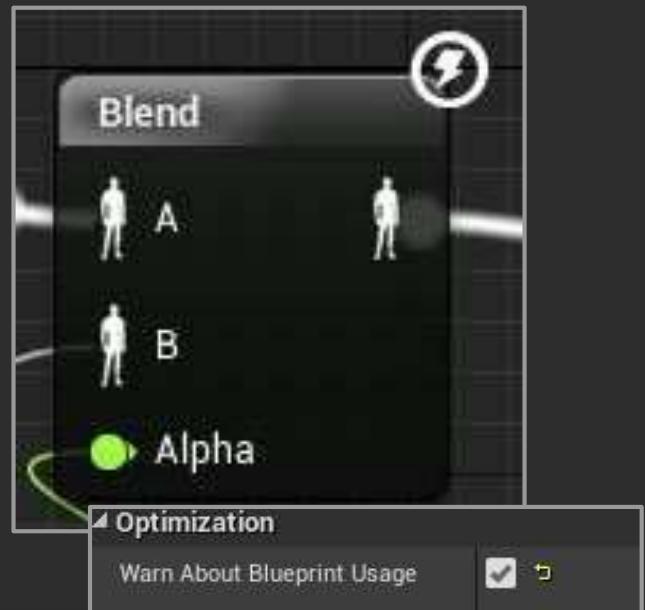
- Animation BlueprintでAnim Nodeの実行に時間がかかるとTask Graph Threadのコストに影響

改善点

- Fast Path機能を利用することで高速なパスで実行可能

備考

- Fast Pathが常時適用されるようにする
- Fast Pathの適用をチェックする警告設定がある



#UE4CEDEC

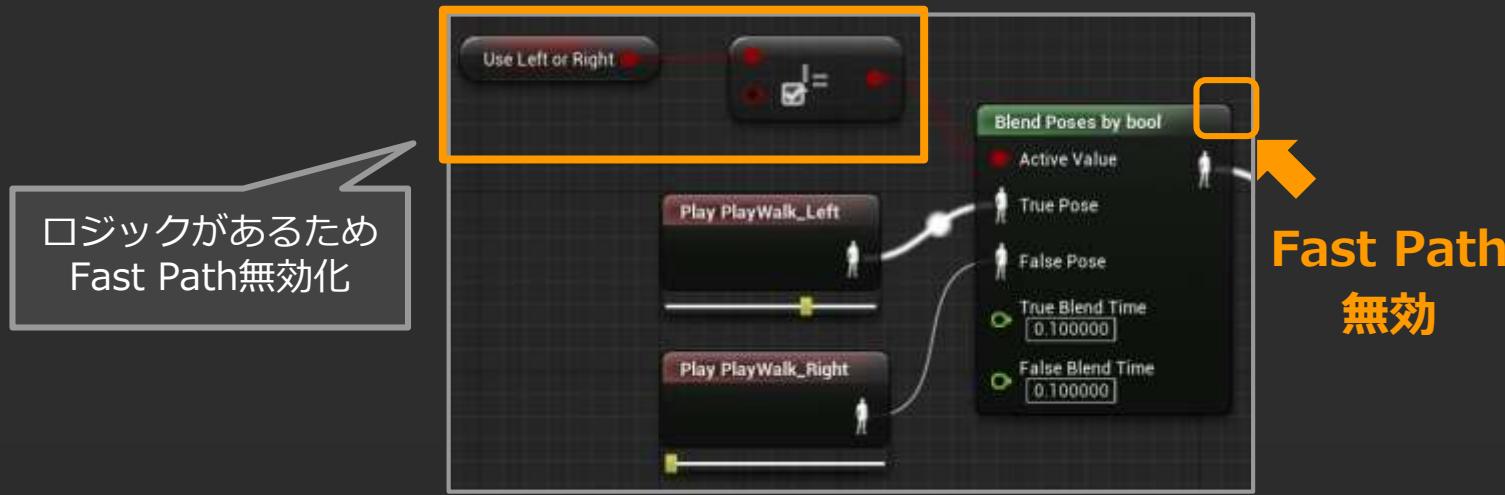


アニメーションの実行コストを削減 2



- 負荷ポイント

Anim Graphでロジックによる演算を行うとFast Pathが適用されずにTask Graph Threadの処理コストに影響

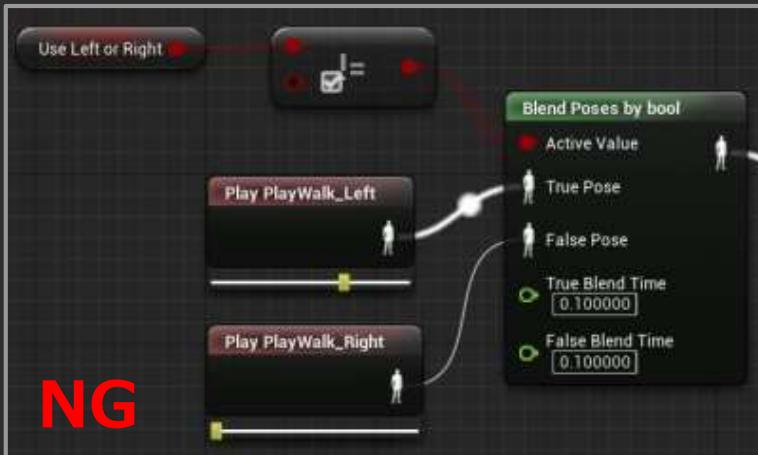


アニメーションの実行コストを削減 2

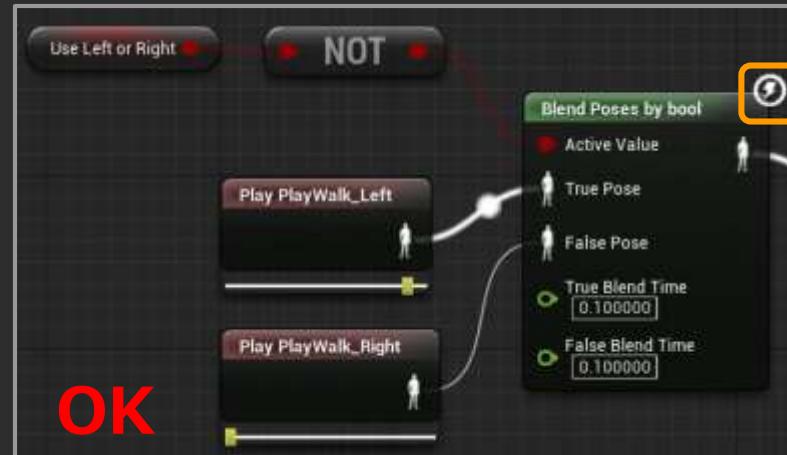


- 改善策

Blueprintでロジックを処理しないことで高速パスでの実行が可能



NG



Fast Path
有効



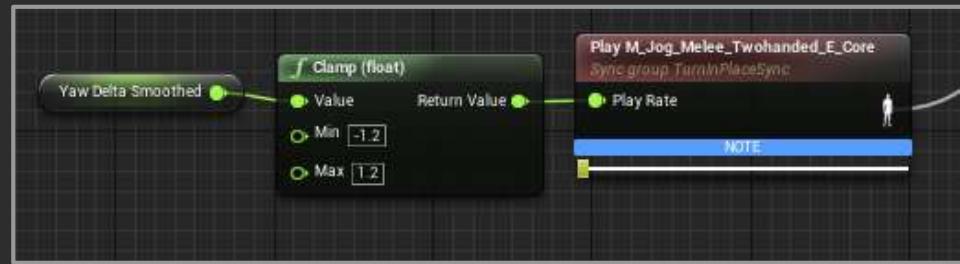
アニメーションの実行コストを削減 2



Clamp Nodeは利用ケースが多いがFast Pathを阻害してしまう
– 4.20からAnim Node内でClampを処理することが可能

- **4.20まで**

Clamp Nodeを使用することで処理
Fast Pathを維持できない



- **4.20から**

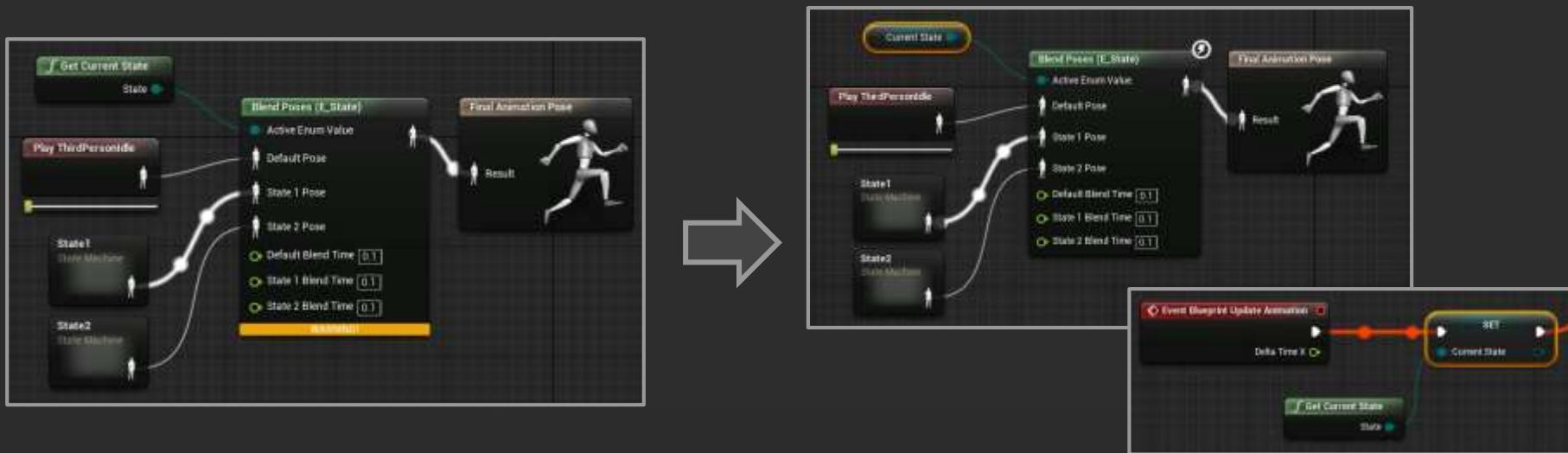
ClampをAnim Nodeで処理可能
Fast Pathを維持できる



アニメーションの実行コストを削減 2



Anim Graphでロジックを実行したい場合はEventGraphで実行する
- さらにC++に移行することでVMアクセスのコスト削減



アニメーションの間引きによる実行コスト削減



負荷ポイント

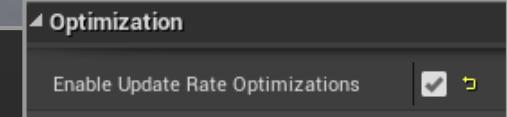
- アニメーションを毎フレーム更新することでアニメーションの更新コストが嵩む

改善点

- アニメーションの更新頻度を調整することでアニメーションの更新コストを抑制

備考

- アニメーションの更新頻度はEditorに調整項目が公開されていないのでC++で実装が必要



#UE4CEDEC



アニメーションの間引きによる実行コスト削減



- 負荷ポイント

キャラクターの位置に関係なくアニメーションの実行コストは一律
– 遠方のキャラクターは描画の重要性が低いが実行コストは同じ

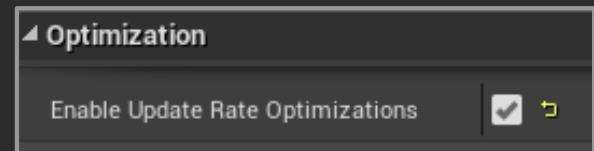


アニメーションの間引きによる実行コスト削減



- 改善策

- UROを使用してアニメーションの更新頻度を調整することでコスト削減
 - 遠く離れている場合など、アニメーションの精度が低い場合など有用



Editorに調整用パラメータは公開されていないためC++で調整が必要

- 最適な設定はプロジェクトに応じて調整

```
/** Animation Update Rate optimization parameters. */
struct FAnimUpdateRateParameters* AnimUpdateRateParams;
```



アニメーションの評価コスト削減



負荷ポイント

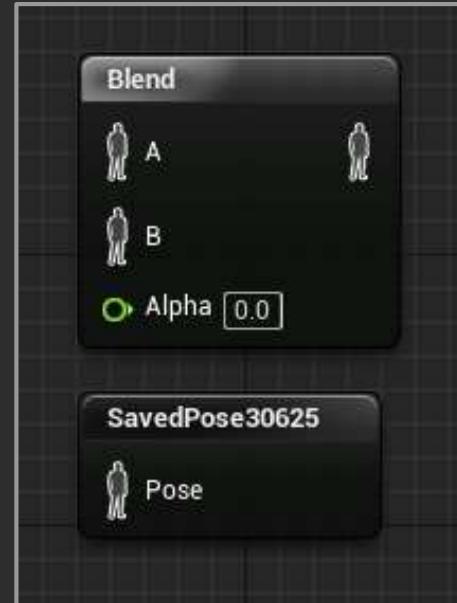
- Anim Nodeが多く接続されていることは非同期処理の実行コストが嵩む

改善点

- Anim Nodeの利用を不用意に増やしすぎないことにより TaskGraph Threadの処理コストを抑制

備考

- 処理が多いほどコストが掛かるのは当然だが、多数のキャラクターを扱う場合はTask Graphのコストも注意



アニメーションの評価コスト削減

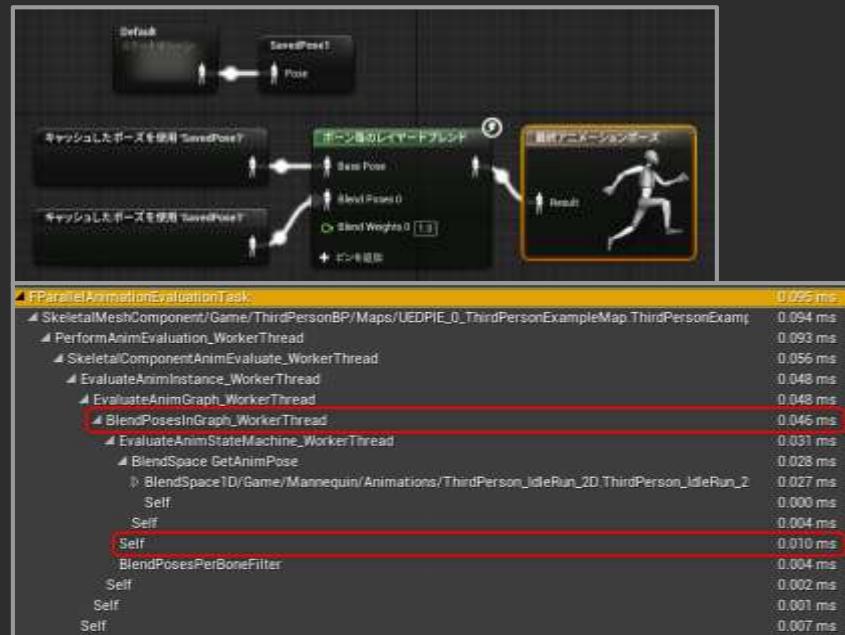
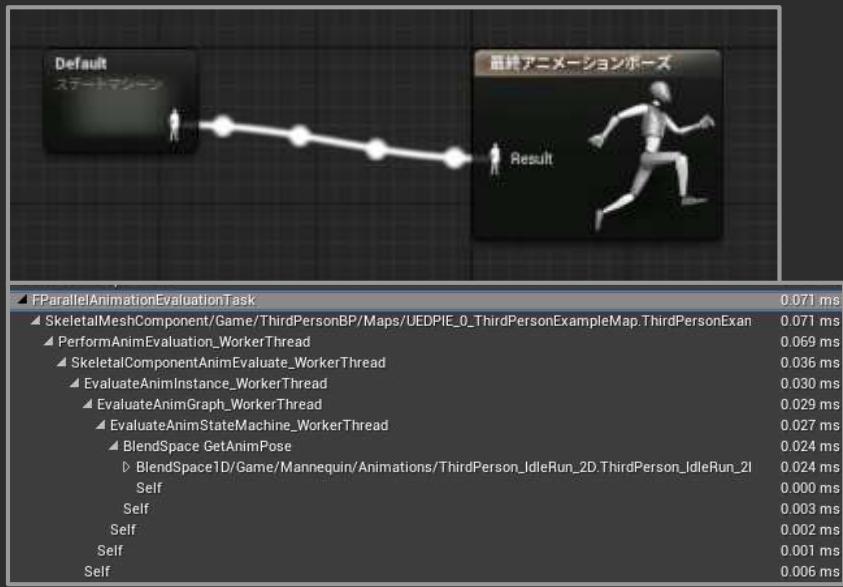


- 負荷ポイント／改善策

Anim Nodeを多く利用することは実行コストが増える多用しすぎに注意
– 非同期で処理する場合もキャラクターが多いとコストが嵩む



アニメーションの評価コスト削減



"Layer Blend per Bone"Nodeの有無による処理コストの違い



アニメーションが不要なメッシュ



負荷ポイント

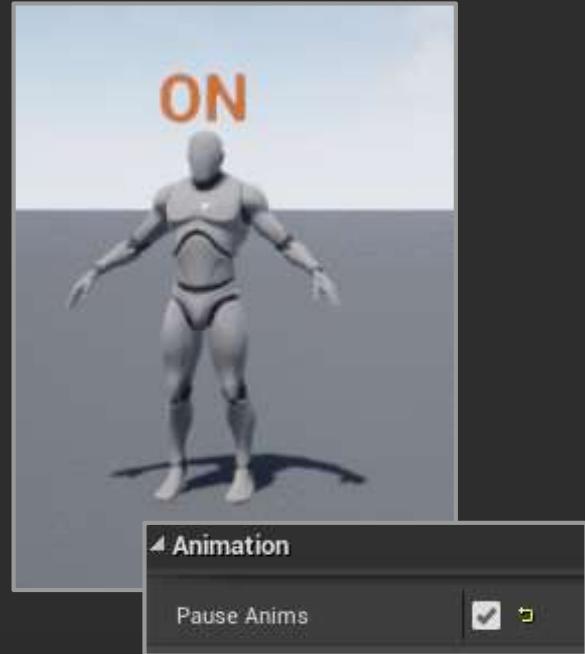
- アニメーションを必要としないキャラクターであってもアニメーションを実行することでコストが嵩む

改善点

- 明示的にスケルトンを更新しないことでアニメーション実行のフローをスキップすることでコスト削減

備考

- Skeletal Meshが持つ各種設定を用途に応じて使い分ける



#UE4CEDEC

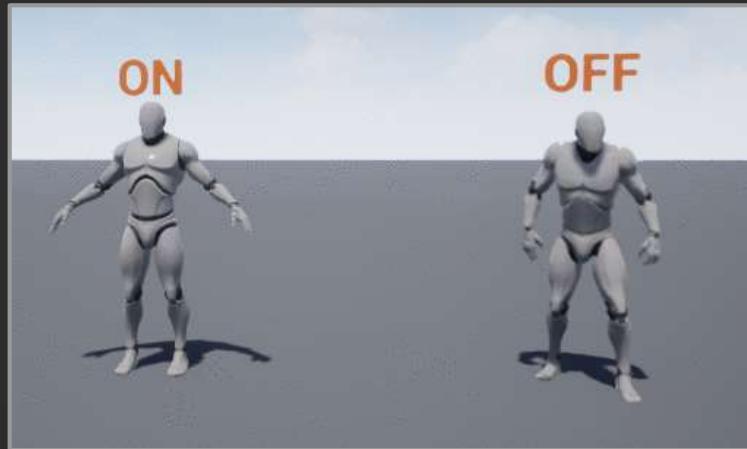


アニメーションが不要なメッシュ



- **負荷ポイント／改善策**

- アニメーションが不要なケースではアニメーションを停止させておく
 - 見た目に変化が無い場合は明示的に停止して無駄なコストを抑止



アニメーションの更新停止

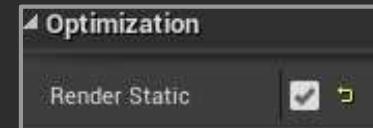
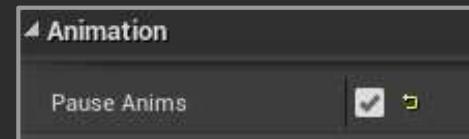


アニメーションが不要なメッシュ



Skeletal Meshの各種設定

- アニメーションの更新を停止
 - Skeletal Meshの更新負荷を削減
- アニメーションの再生を停止
 - 動的に変更が可能でTrueで更新を停止
- Static MeshのRenderingと同様の描画パスを使用
 - 動的に変更が可能でアニメーションの更新を停止



LODによるメッシュのリダクション



負荷ポイント

- キャラクターがカメラから近くにいても遠くにいてもアニメーションを行う処理コストが一定
- 遠景にいる場合は処理を間引きたい

改善点

- Skeletal MeshのLODを設定してBoneを削除することでコスト削減

備考

- 特になし

The screenshot shows the 'Bones to Remove' section of the Skeletal Mesh LOD Settings panel. It includes fields for 'Screen Size' (1.0), 'LODHysteresis' (0.02), 'Source Import Filename' (highlighted in gray), 'Allow CPUAccess' (unchecked), 'Support Uniformly Distributed' (unchecked), 'Bake Pose' (None), and a list for 'Bones to Remove' containing 'middle_01_L' with a 'Reapply removed bone' button.



LODによるメッシュのリダクション



Skeletal MeshのLODを設定してBoneを削除することが可能

The screenshot shows the Unreal Engine 4 Editor interface with the following details:

- Bones to Remove:** A list of 4 array elements containing bone names:
 - 0: middle_01_I
 - 1: middle_02_I
 - 2: middle_03_I
 - 3: pinky_01_IA button "Reapply removed bone" is also present.
- LOD 0:** Contains two sections:
 - Section 0: Highlight, Isolate
 - Section 1: Highlight, IsolateEach section has a preview image, material slot dropdown (None), clothing dropdown (None), and a checked "Cast Shadows" checkbox.
- LOD 1.0:** Contains:
 - Screen Size: 1.0
 - LOD System: 0.02
 - Source Import Parameters
 - Allow CPU Access
 - Support Dynamic Reimport
 - Static Pose
 - Bones to Remove: 1 array element (None)A "Reapply removed bone" button is also present.
- LOD Settings:** A list of 1 array element containing "None". A "Reapply removed bone" button is also present.
- Preview:** On the right, a 3D view of a white humanoid character model with black outlines and highlights, standing in a dynamic pose.



ステートマシンの検索コスト削減



負荷ポイント

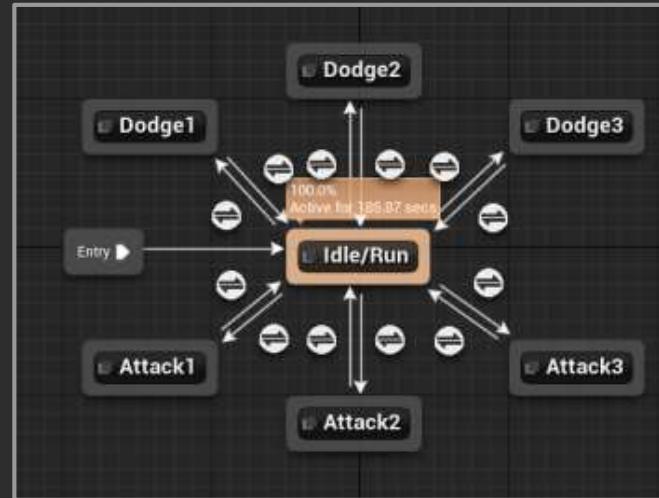
- 每フレームステートマシンで評価を行うが遷移候補が多いと検索コストが増加

改善点

- 多くの検索を行わないようにステートマシンの遷移先を細分化することで検索コスト削減

備考

- 検索候補を減らすことも重要だがステートを減らすことを見直してシンプルな実装を目指す

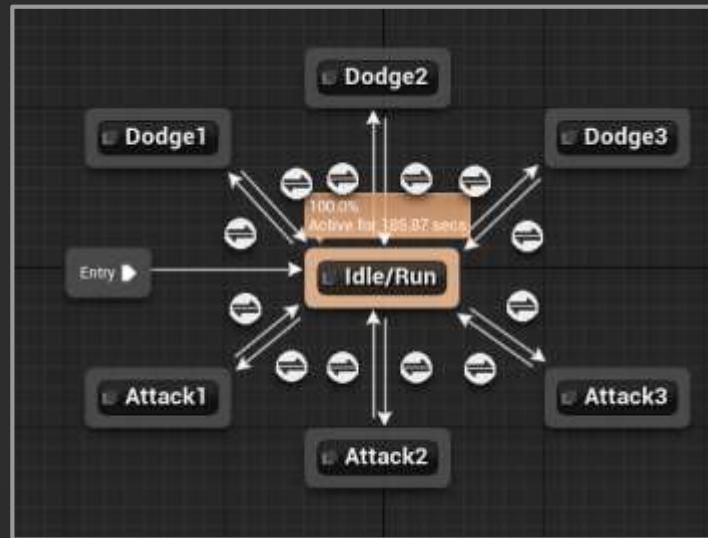


ステートマシンの検索コスト削減



- 負荷ポイント

State MachineのTransitionが多いと検索コストが増加

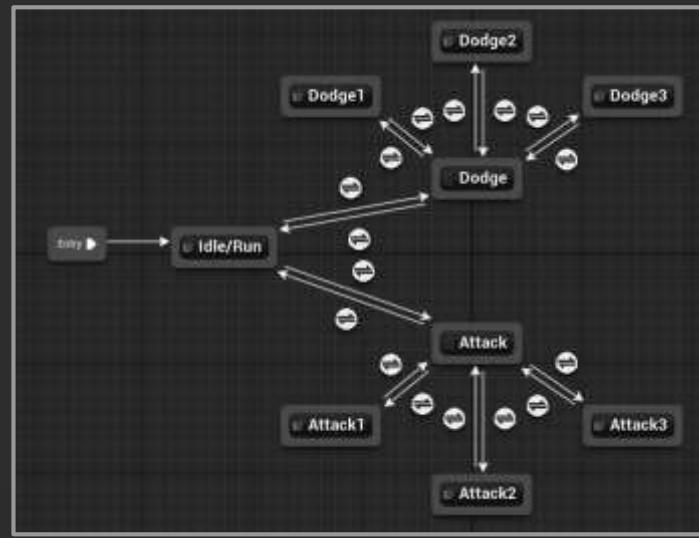


ステートマシンの検索コスト削減



- 改善策

Stateを細分化することで検索コストを下げる



アジェンダ

- Animation
- **Physics**
- Navigation & AI
- Movement
- Networking
- その他



#UE4CEDEC



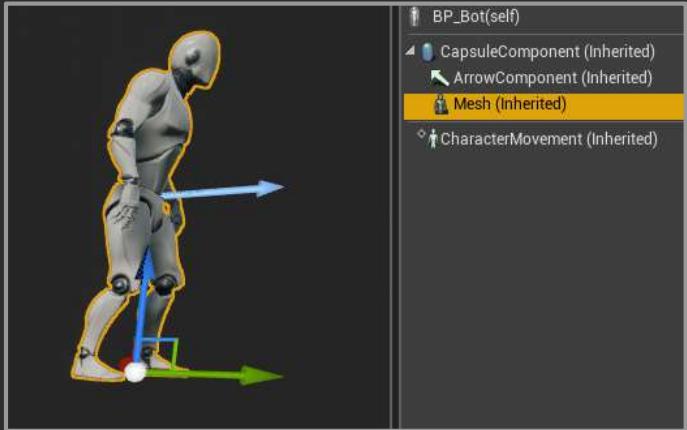
Physics



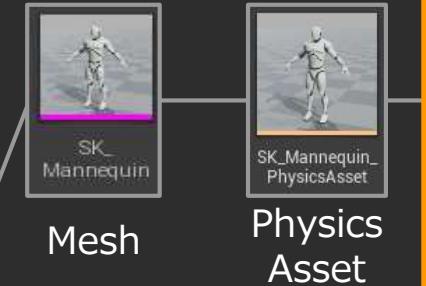
#UE4CEDEC



CharacterとPhysics



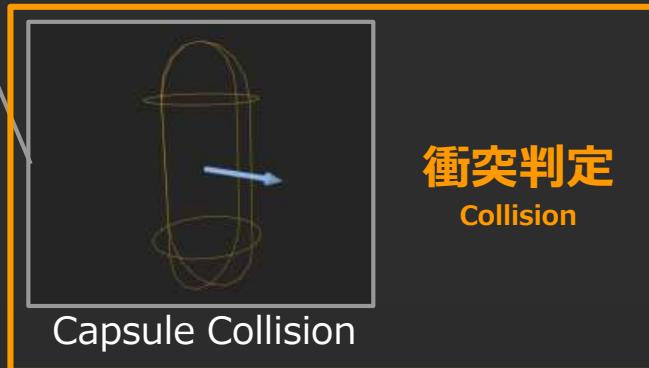
Character



Physics
Asset



剛体
Rigid Body



Capsule Collision

衝突判定
Collision

おさらい : Physics (Collision)

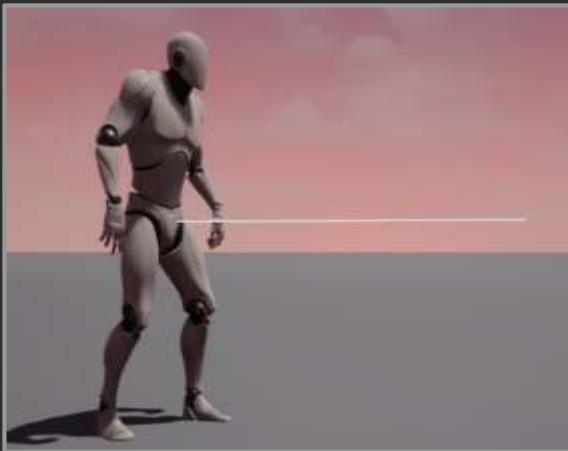
オブジェクトの物理挙動のふるまい



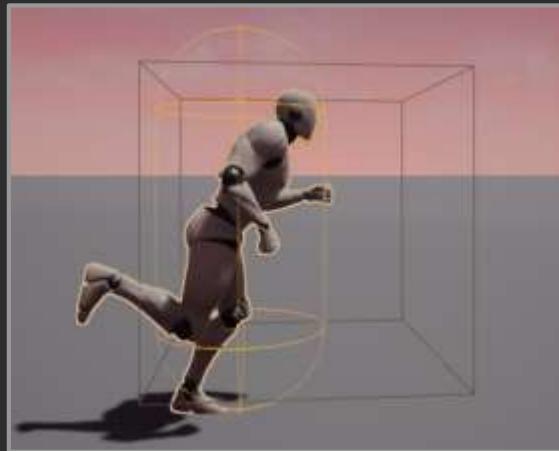
- Collision : QueryとPhysics
- Query : Raycast, Overlap, Sweepなどの**空間クエリ**
- Physics : Rigid Body, Constraintなどの**物理シミュレーション**

おさらい : Physics (Query Collision)

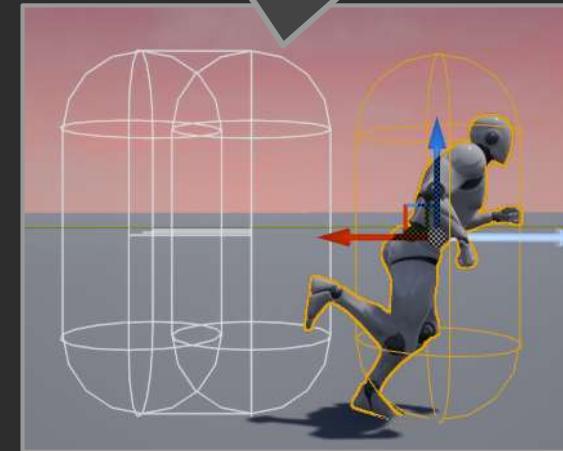
- Raycast, Overlap, Sweepなどの**空間クエリ**
– 衝突判定、当たり判定のようなイメージ



Raycast



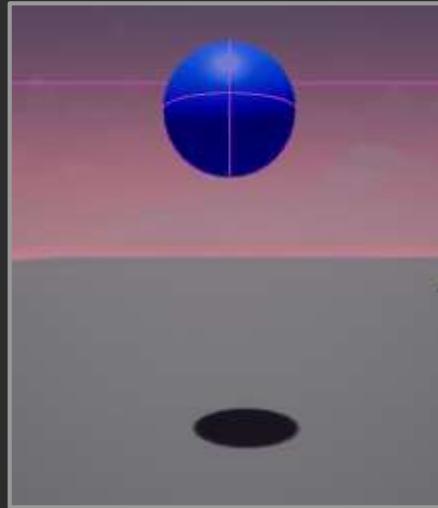
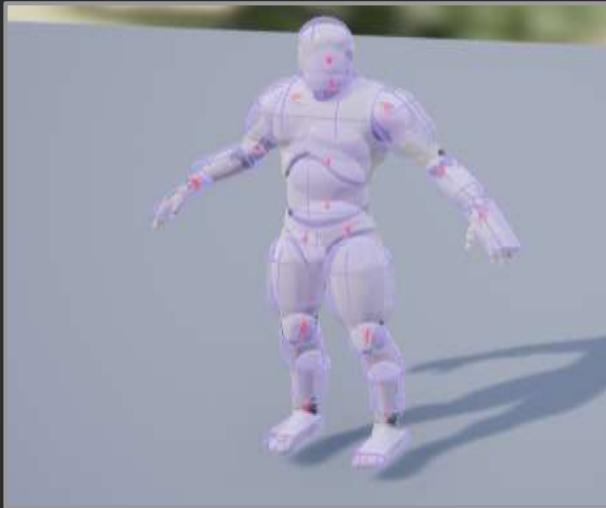
Overlap



Sweep

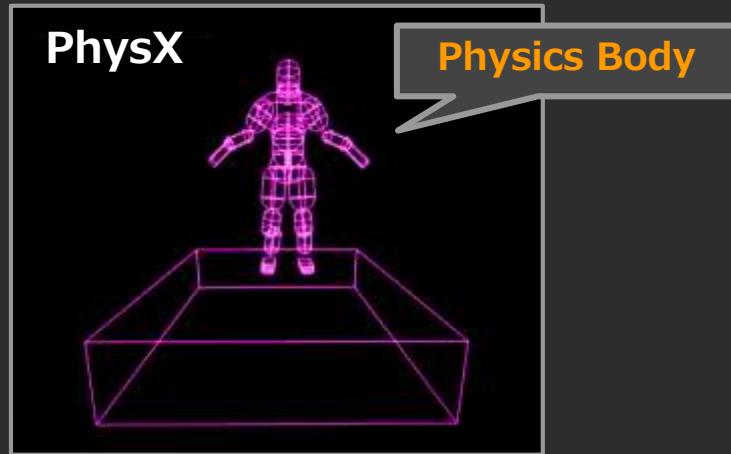
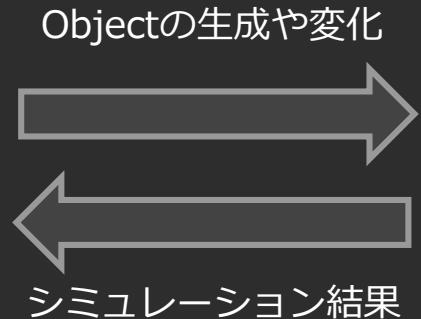
おさらい : Physics (Physics Collision)

- Rigid Body, Constraintなどの物理シミュレーション
 - 重力に従った挙動のイメージ



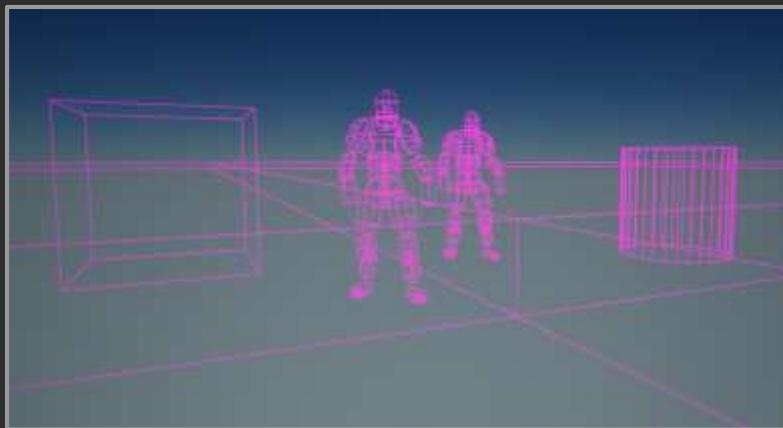
おさらい : Physics (Physics Body)

- UE4でのPhysicsの動作
 - 物理エンジン(**PhysX**)で物理シミュレーションを行う
 - Physicsを扱う空間(Physics Scene)に**Physics Body**が投入される

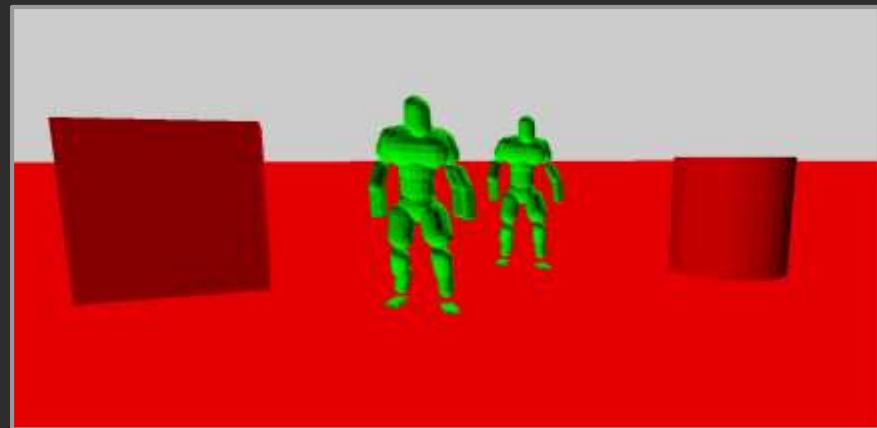


おさらい : Physics (Physics Body)

- Physics Sceneに投入されたオブジェクトを確認する方法
 - コンソールコマンド ("PXVIS SYNC COLLISION" など)
 - 専用ツール (PhysX Visual Debugger)

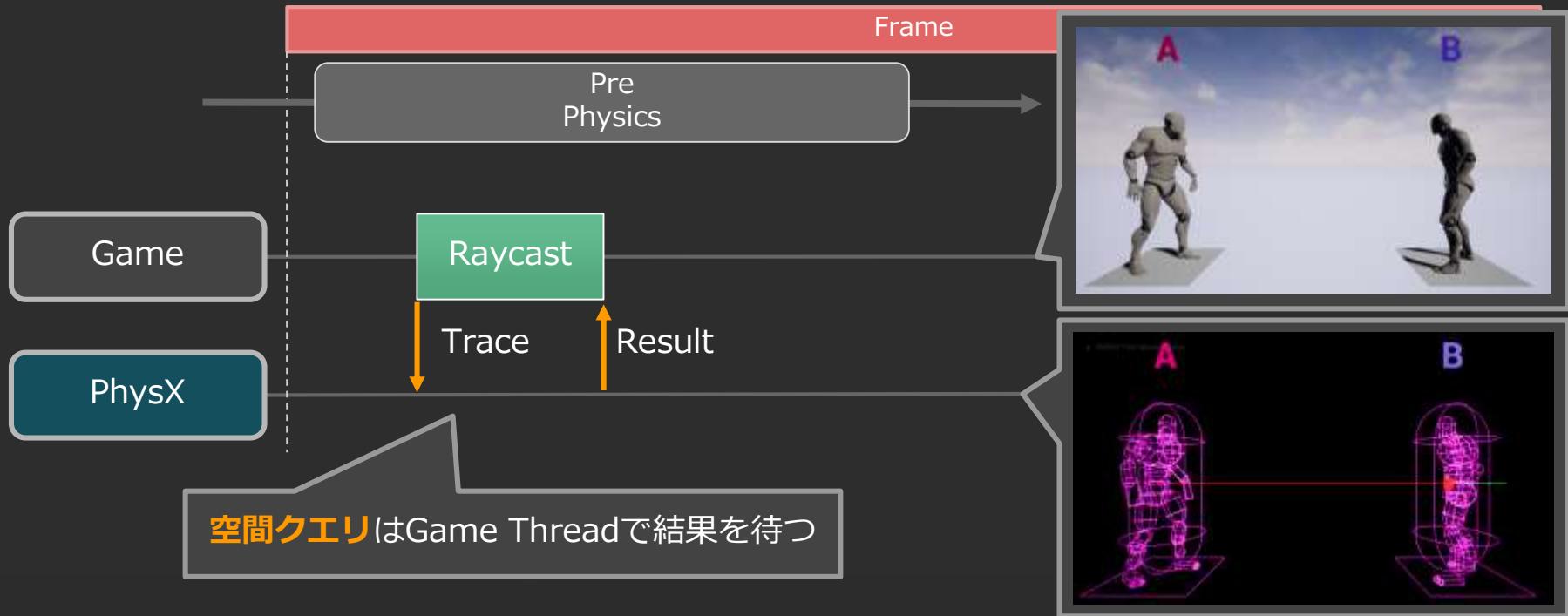


PXVIS SYNC COLLISION コマンド

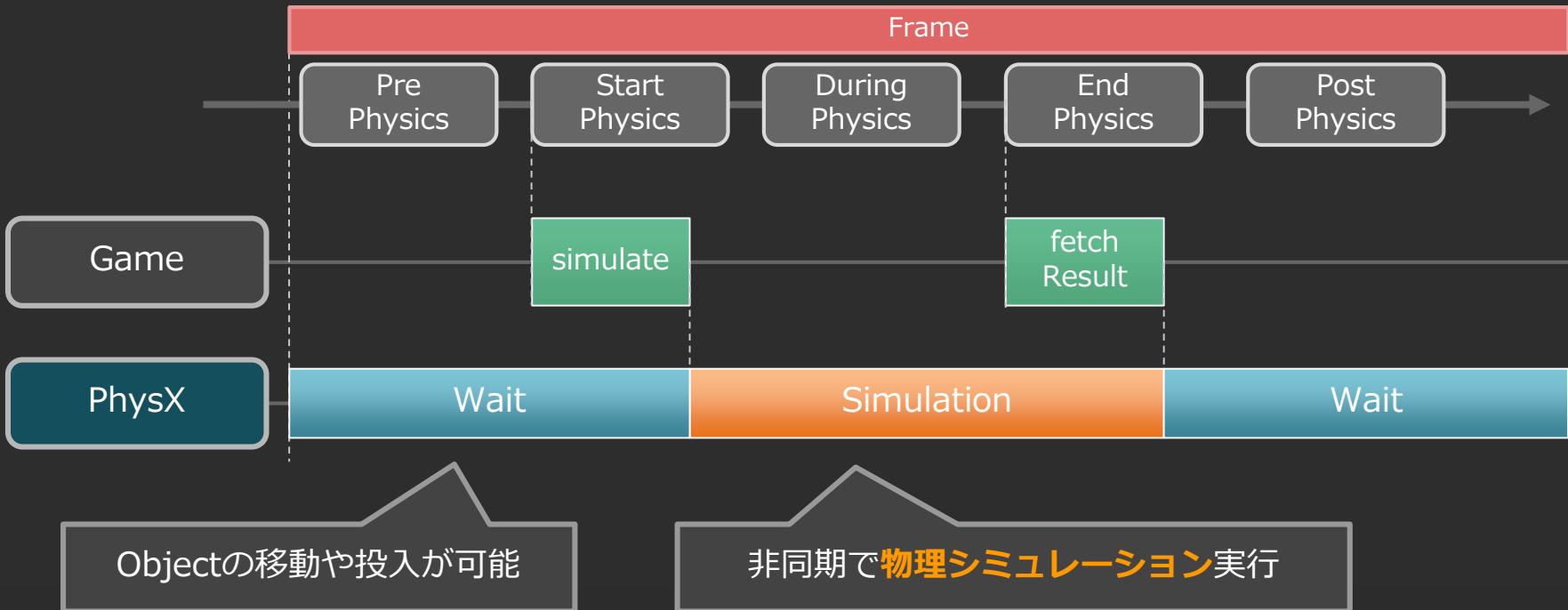


PhysX Visual Debugger

おさらい : Physics (Query処理の流れ)



おさらい : Physics (Physics処理の流れ)



改善ポイント一覧 (Physics)

- PhysicsBodyのシミュレーションが多い
- Collision設定とパフォーマンス
- アニメーションに伴うPhysicsBodyのスケール同期
- アニメーションに伴うPhysicsBodyの位置同期
- オブジェクトのオーバーラップ検出
- 移動やアニメーションに伴うBounds更新 3
- 非同期シーンを利用したRaycast



Physics Bodyのシミュレーションが多い



負荷ポイント

- 多くのPhysics Bodyを持つキャラクターが移動するケースにおいて物理シミュレーションのコストが増加

改善点

- Physics Bodyの数を出来るだけ増やさないことで、位置同期の更新やシミュレーションの実行コストを削減

備考

- Physics Bodyの数を厳密に管理して調整



#UE4CEDEC

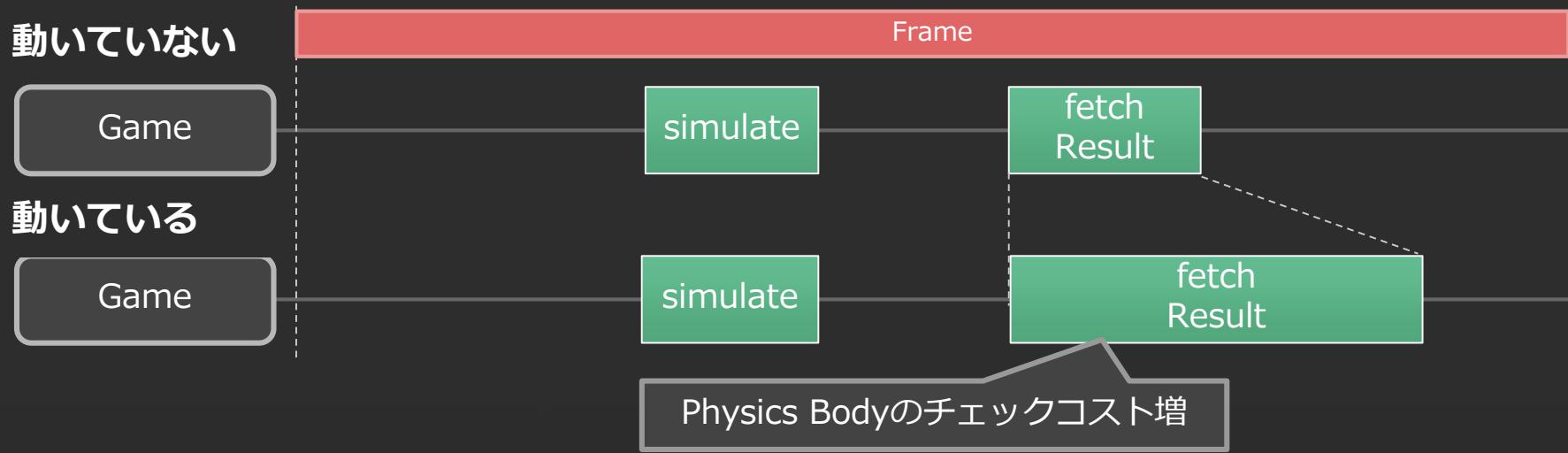


Physics Bodyのシミュレーションが多い



- 負荷ポイント

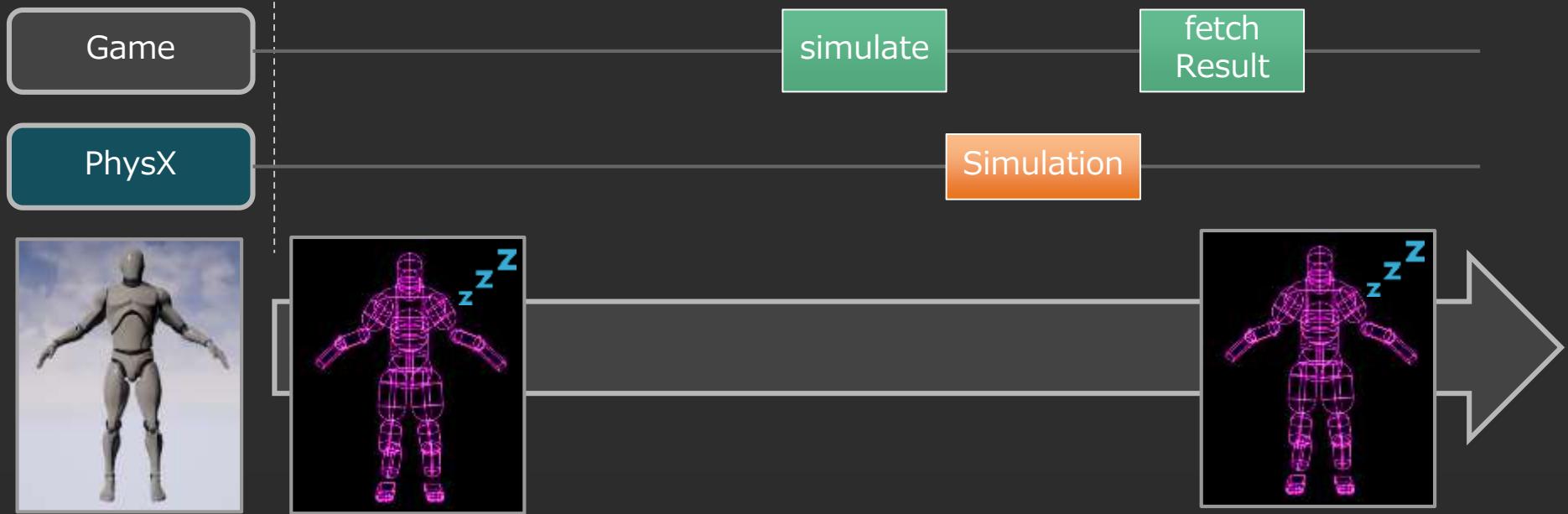
動いているPhysics Body数が多い時にPhysicsのfetchコスト増加



Physics Bodyのシミュレーションが多い



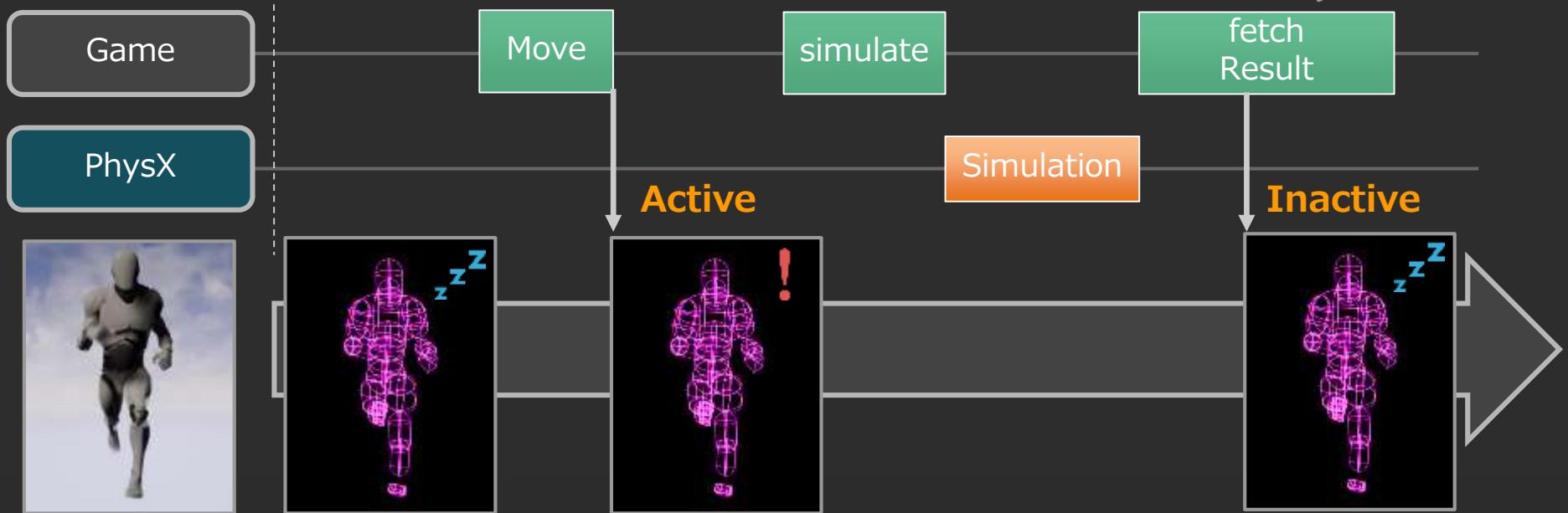
動いていない



Physics Bodyのシミュレーションが多い

ActiveなBodyを全て走査して
継続しない場合は戻す

動いている



#UE4CEDEC

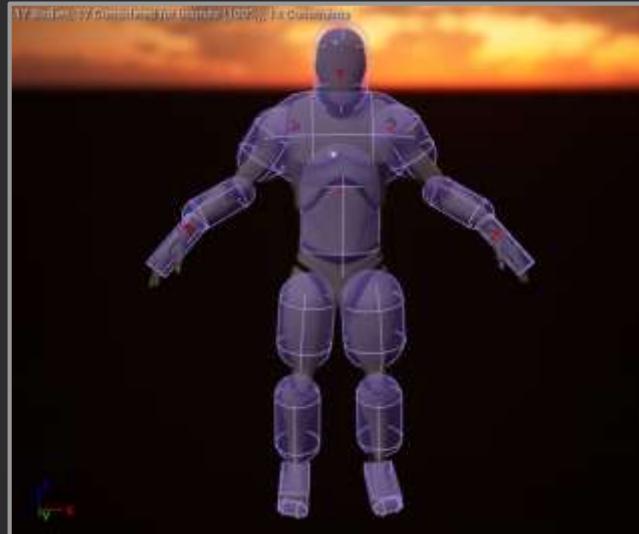
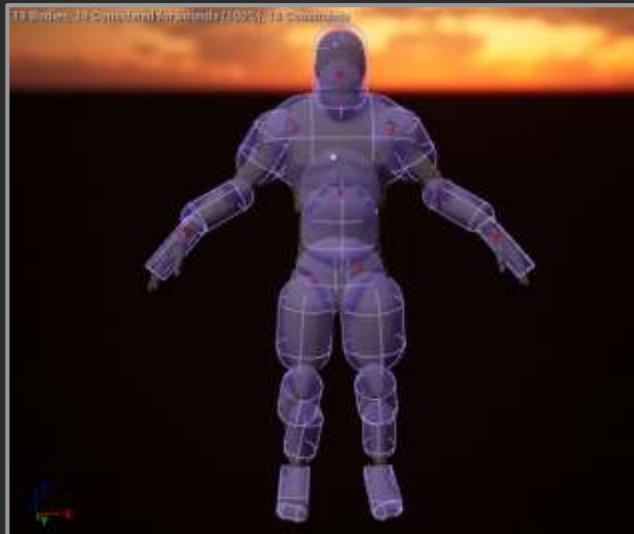


Physics Bodyのシミュレーションが多い

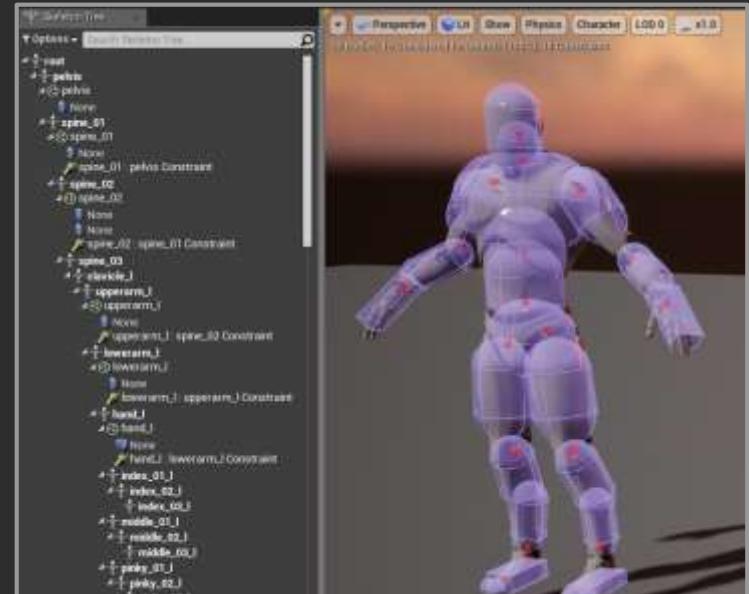


- 改善策

Physics Bodyの数を出来る限り減らすことでPhysXへの更新を減らす



Physics Bodyのシミュレーションが多い



ActiveなPhysics Body数の違いでのパフォーマンスを比較



#UE4CEDEC



Physics Bodyのシミュレーションが多い



Third Person Templateで1000体を出すと…

Physics Body:19 (neckあり)

Frame:	57.42 ms
Game:	49.02 ms
Draw:	51.35 ms
CPU:	51.20 ms
RHIT:	14.99 ms
DynRest:	OFF

Physics Body:18 (neckなし)

Frame:	56.25 ms
Game:	47.10 ms
Draw:	50.56 ms
CPU:	51.31 ms
RHIT:	14.84 ms
DynRest:	OFF

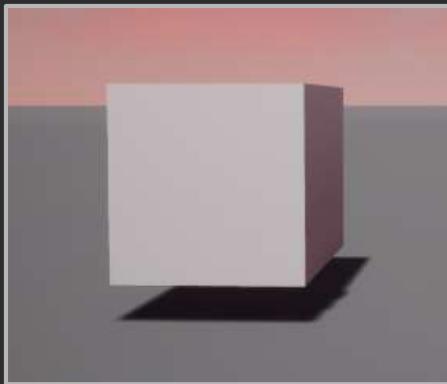
約0.9ms
削減



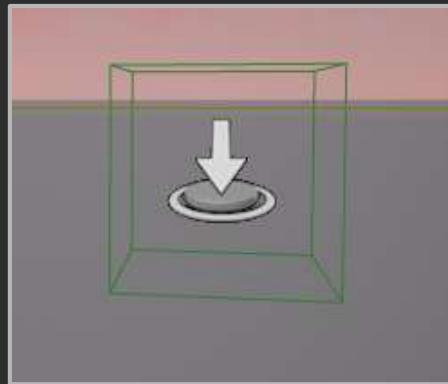
Physics Bodyのシミュレーションが多い



- 静的なオブジェクトは物理シミュレーションに影響するか?
 - レベル上に多数のオブジェクトを配置した際の定常コスト



Static Mesh
殆ど影響なし



Collision
殆ど影響なし



Destructible Mesh
影響する

Physics Bodyのシミュレーションが多い



- Static Mesh、Collision
 - 配置しただけでは定常コストに影響しない
 - Movableでも動いていない場合は影響しない
 - 回転や移動時はシミュレーションが発生
- Destructible Mesh
 - 配置しただけで定常コストに追加
 - 破壊発生のタイミングでのみ使用するなどで定常時のシミュレーションコストを削減



Collision設定とパフォーマンス



負荷ポイント

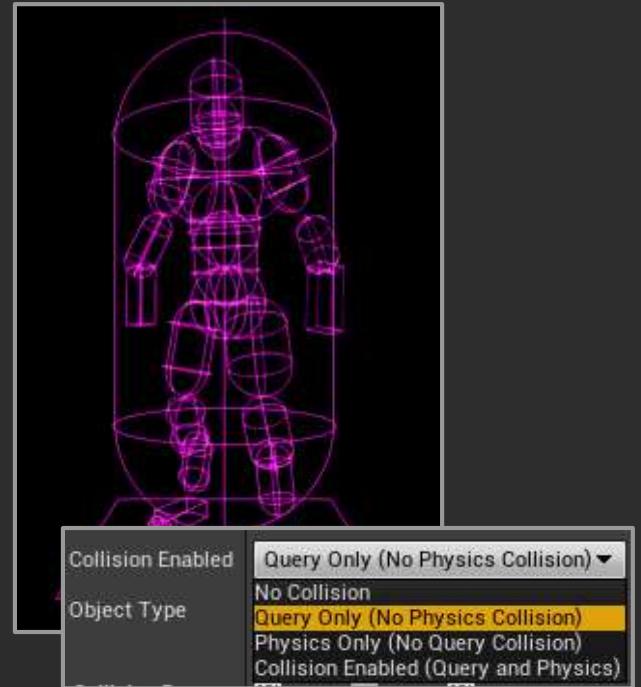
- 移動やRaycastを行う際にQuery and Physicsは空間クエリ及び物理シミュレーションを行うため、Query OnlyやPhysics Onlyと比較してコストになる

改善点

- 用途に応じてQuery Only／Physics Onlyを適切に設定することで無駄な処理コストを削減

備考

- No Collisionは物理処理のコストが一番安く、Query and Physicsは最もコストが高い



#UE4CEDEC

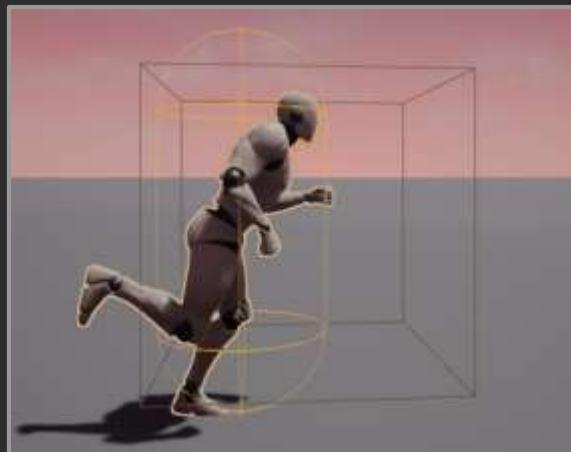


Collision設定とパフォーマンス

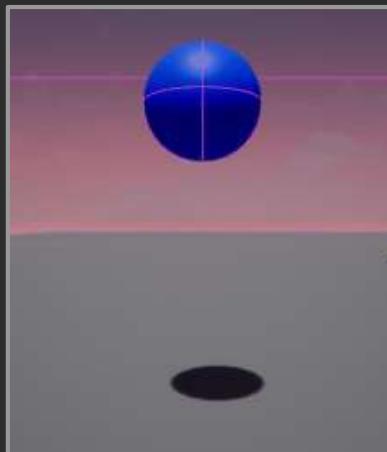


- 負荷ポイント

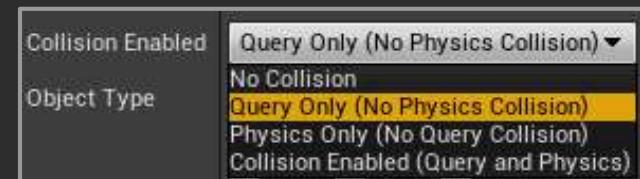
Collision設定においてQueryとPhysicsの両方を適用するとコストが嵩む



Query



Physics

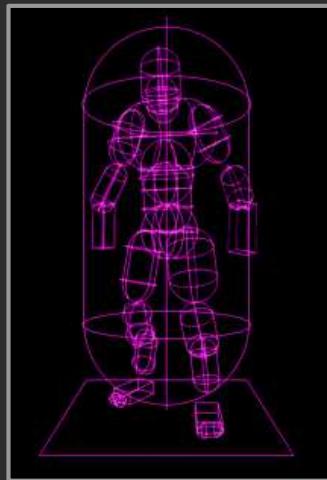


Collision設定とパフォーマンス



- 改善策

Collision設定の“Query Only”と “Physics Only”を使い分ける



Capsule

Collision Enabled (Query and Physics)
⇒ 衝突判定と物理シミュレーションが可能

Skeletal Mesh

Query Only (No Physics Collision)
⇒ 衝突判定のみ検出が可能



#UE4CEDEC

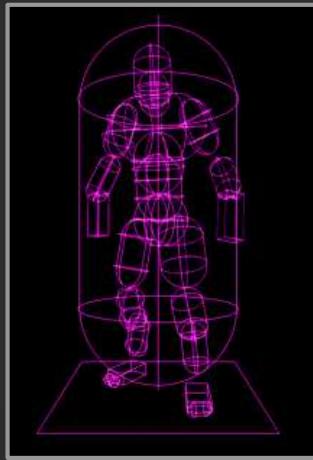


Collision設定とパフォーマンス

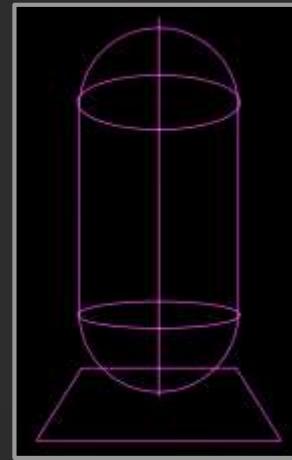
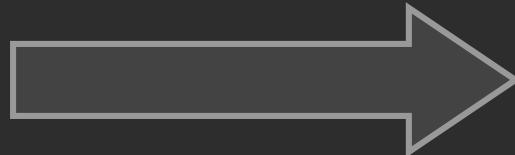


例：「キャラクターの部位判定が不要」なケースの場合…

- Skeletal Mesh は “**No Collision**” に設定
- シーンに投入されなくなるので**物理シミュレーションのコストが安くなる**



Skeletal Meshを
“**No Collision**”に変更



アニメーションに伴うPhysicsBodyの位置同期



負荷ポイント

- アニメーション再生時にPhysics Bodyを更新する際にボーンとPhysics Bodyの位置を同期する

改善点

- Physics Bodyの同期処理をスキップすることで負荷削減

備考

- 見た目とPhysics Bodyの位置が同期しなくなるので、アニメーションによる位置同期が不要な場合に有効



#UE4CEDEC

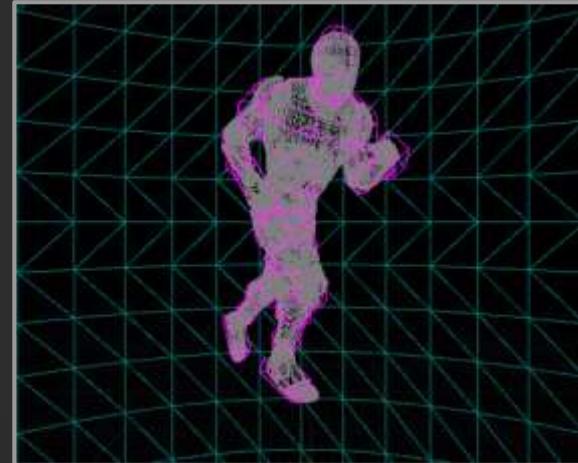


アニメーションに伴うPhysicsBodyの位置同期



- **負荷ポイント**

- アニメーション再生時にボーンに付随するPhysics Bodyの位置を更新
 - 所有するPhysics Bodyの数が多い程同期コストが嵩む



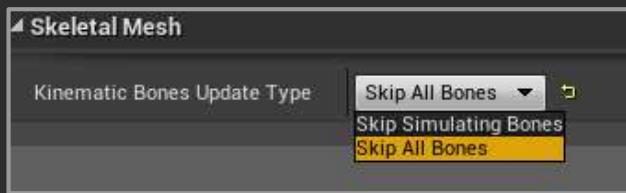
アニメーションに伴うPhysicsBodyの位置同期



- 負荷ポイント

Physics Bodyの位置同期をスキップすることで同期コストを削減

- 移動しないキャラクターなどPhysics Bodyの同期が不要な時に有効



- Skip Simulating Bones (デフォルト)

物理シミュレーションを行なうボーンのみ同期スキップ

- Skip All Bones

全てのボーンを対象に同期スキップ



アニメーションに伴うPhysicsBodyのスケール同期



負荷ポイント

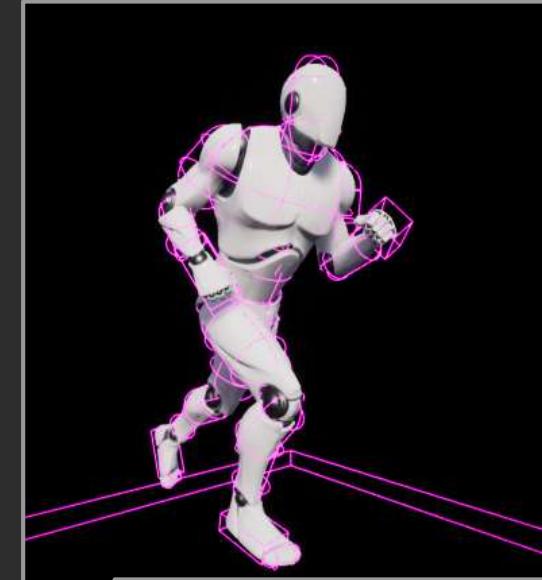
- アニメーション再生時にPhysics Bodyのスケールを更新することでボーンとPhysics Bodyのスケールを同期

改善点

- Physics Bodyのスケール同期処理をスキップすることで同期コスト分の負荷削減

備考

- ボーンとPhysics Bodyのスケールが同期しなくなるため、ボーンのスケールが動的に変わらない場合に有効



▲ Body Setup

Skip Scale from Animation



#UE4CEDEC

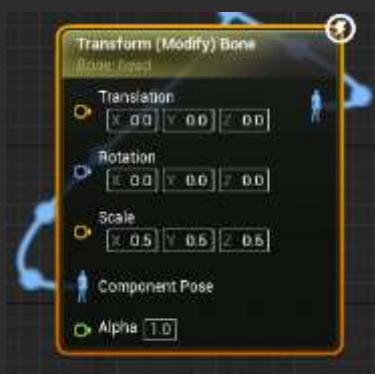


アニメーションに伴うPhysicsBodyのスケール同期

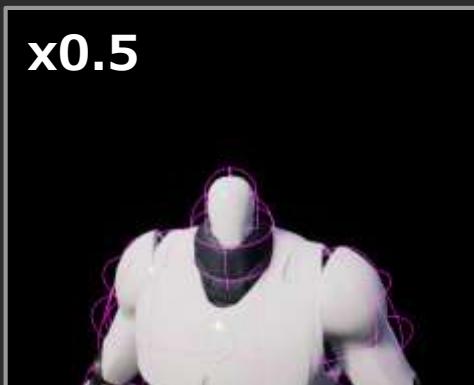


- 負荷ポイント

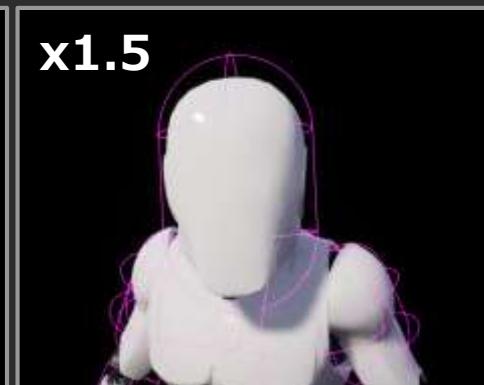
- アニメーション実行時にPhysics BodyのScaleスケールを更新
 - ボーンのスケールとPhysicsを連動させるため



HeadのScaling



Bone Scaleに応じて **Physics Bodyも同期**

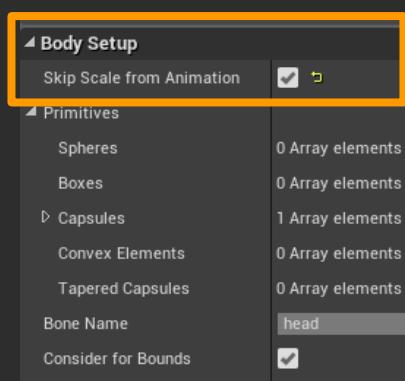


アニメーションに伴うPhysicsBodyのスケール同期

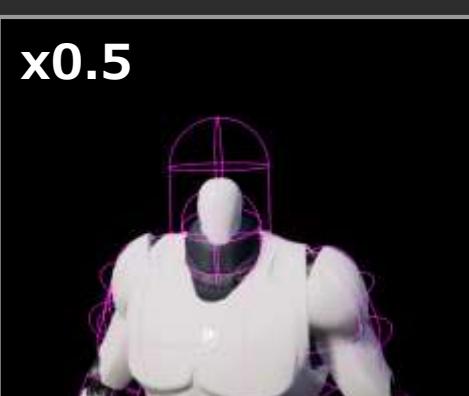


- 改善策

- アニメーション実行時のPhysics BodyのScale同期をスキップ
 - ランタイムでボーンスケールが変更しないケースなどで有効



Scale同期スキップ



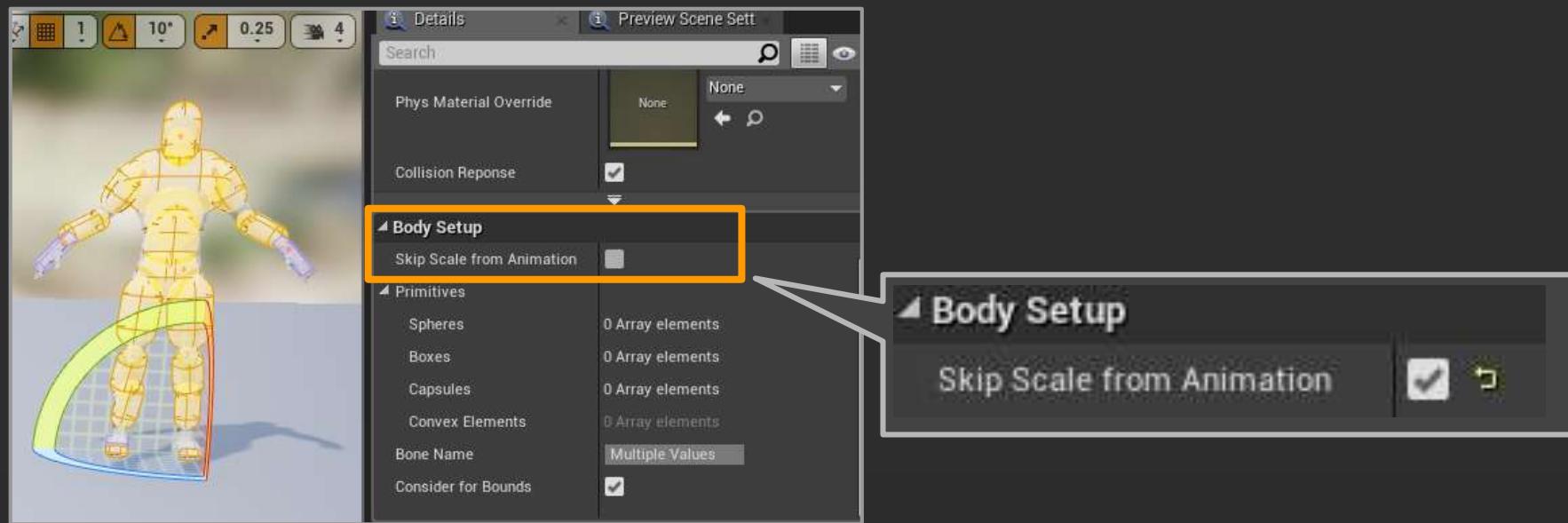
Bone Scaleと**Physics Body**の同期をSkip



アニメーションに伴うPhysicsBodyのスケール同期



Physics Editorから**Physics Body**毎に指定することが可能



移動やアニメーションに伴うBounds更新 3



負荷ポイント

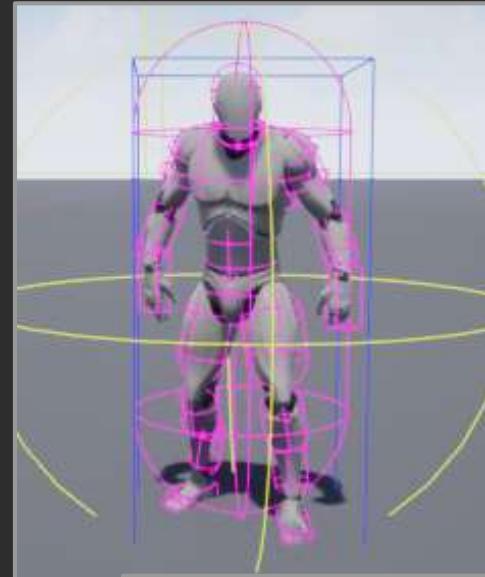
- 移動やアニメーションを行う際にキャラクターのBoundsの位置を更新するための計算コストが嵩む

改善点

- Boundsの計算に不要なPhysics Bodyを除外することによってBoundsの計算コストを削減

備考

- 特になし



移動やアニメーションに伴うBounds更新 3



- 負荷ポイント

移動やアニメーション時にComponentのBounds計算が行われる
– Skeletal MeshはPhysics AssetからFitするBoundsを生成



Skeletal Mesh

Physics Asset

Bounds

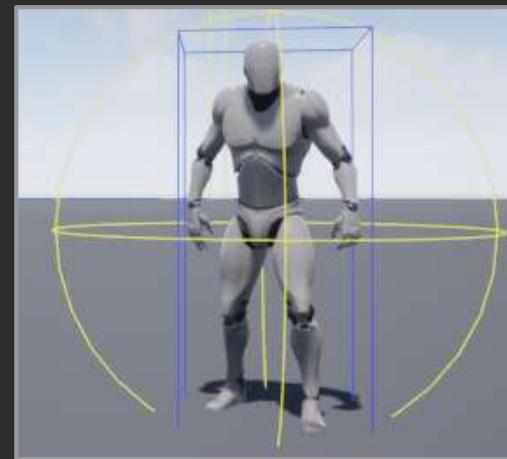
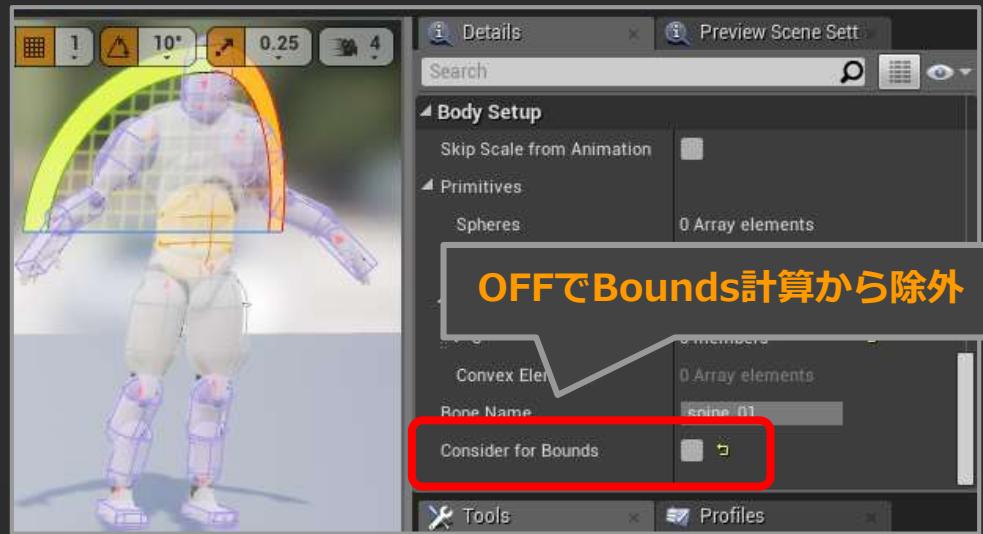


移動やアニメーションに伴うBounds更新 3



- 改善策

Boundsの計算にPhysics Bodyを指定して含まないことが可能



両手・両足・頭 だけでも成立



移動やアニメーションに伴うBounds更新 3



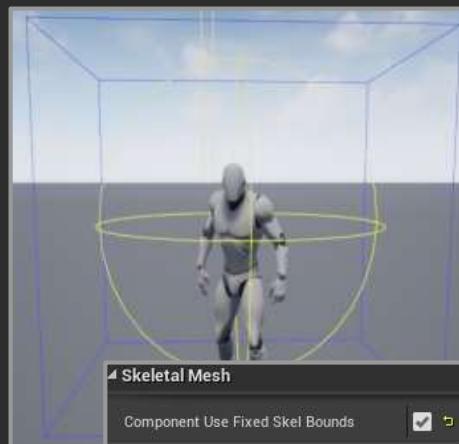
Skeletal MeshのBounds計算は次の順で優先されます

Physics Bodyが多いほど
Boundsの計算コストが嵩む

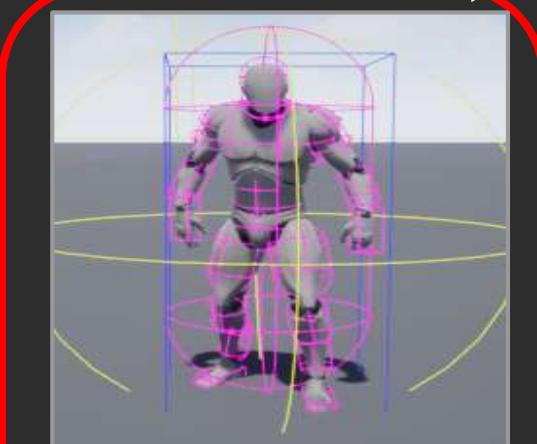
優先



親Boundsと同期



固定Bounds



オブジェクトのオーバーラップ検出



負荷ポイント

- Overlapイベントを有効にした場合、Componentの移動の度にOverlapテストを実行するのでコストが嵩む

改善点

- Overlapが不要な場合はオフにすることで、移動時の頻繁なOverlapの実行コストを削減

備考

- オフにすると当然ながらOverlapイベントを検出しないため、使用する場合はOverlapのON/OFFを制御するなど

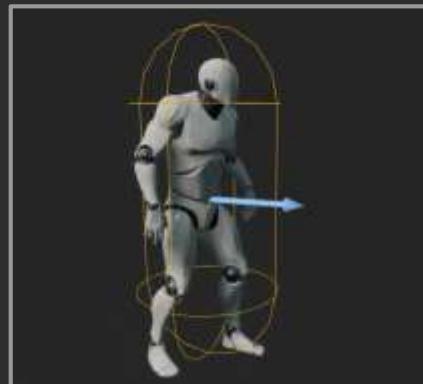


オブジェクトのオーバーラップ検出

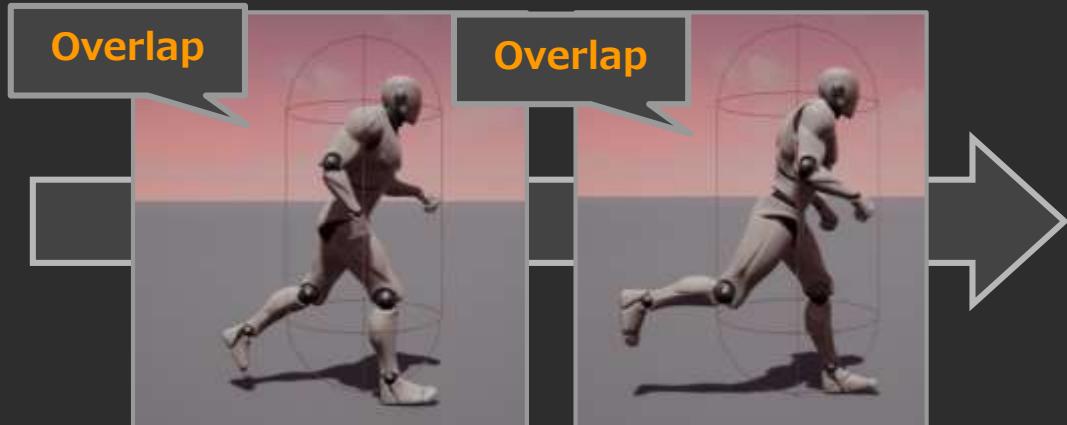


- 負荷ポイント／改善策

Overlap Overlap Eventが有効時、移動の度にOverlap検出処理を実行
– Overlap検出を抑制することで移動時のコスト削減



Capsule Collisionの
Generate Overlap Eventsが有効な場合



移動の度にOverlapテスト実行



非同期シーンを利用したRaycast



負荷ポイント

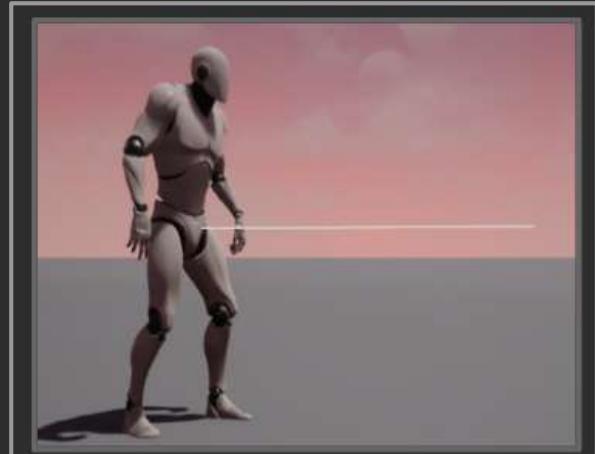
- 足音を鳴らすためにRaycastを頻繁に利用するようなケースにおいて、Raycastの多用がGameの負荷になる

改善点

- 非同期Raycastを使用することでGameの負荷を削減

備考

- 非同期Raycastは次フレーム移行で処理することから、同期して処理する必要があるケースには向いておらず、全ての処理を非同期で行えば良いとは限らない

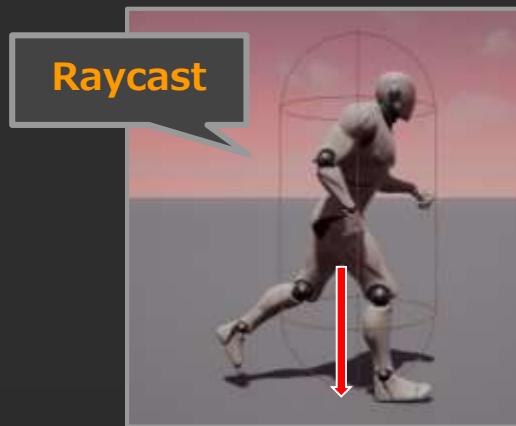


非同期シーンを利用したRaycast



- 負荷ポイント／改善策

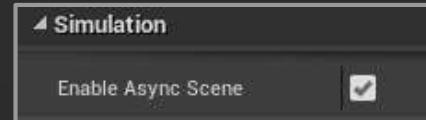
足音を鳴らすためのRaycastを頻繁に利用するとGameの負荷が嵩む
– 非同期Raycastを使用することでGameの負荷集中を避ける



- 非同期トレース用のAPIを使用

```
GetWorld()->AsyncLineTraceByChannel(EAsyncTraceType::Single,
```

- 非同期シーンの使用を許可しておく





非同期シーンを利用したRaycast

- 非同期シーン?
 - Physicsを扱う空間として3種類のPhysics Sceneが存在

- UE4でのPhysicsの動作
 - 物理エンジン(PhysX)で物理シミュレーションを行う
 - Physicsを扱う空間(Physics Scene)にPhysics Bodyが投入される



非同期シーンを利用したRaycast



Physics SceneとPhysics Body



Static MeshとSkeletal Meshの
シンプルなシーン



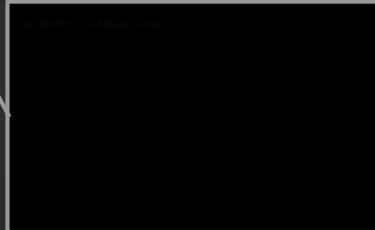
- **Sync Scene**

デフォルトで使用するシーン
Static MeshやSkeletal Meshが投入



- **Async Scene**

非同期トレースで使用するシーン
Static Mesh以外は任意で指定



- **Cloth Scene**

Cloth simulation用のシーン

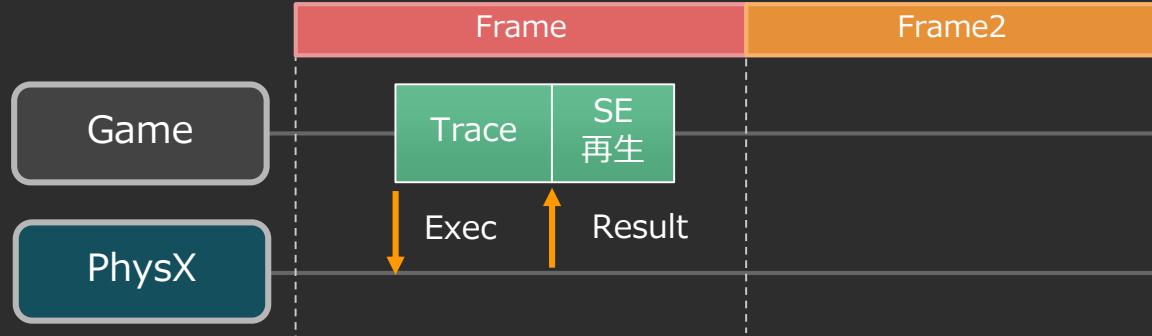


非同期シーンを利用したRaycast



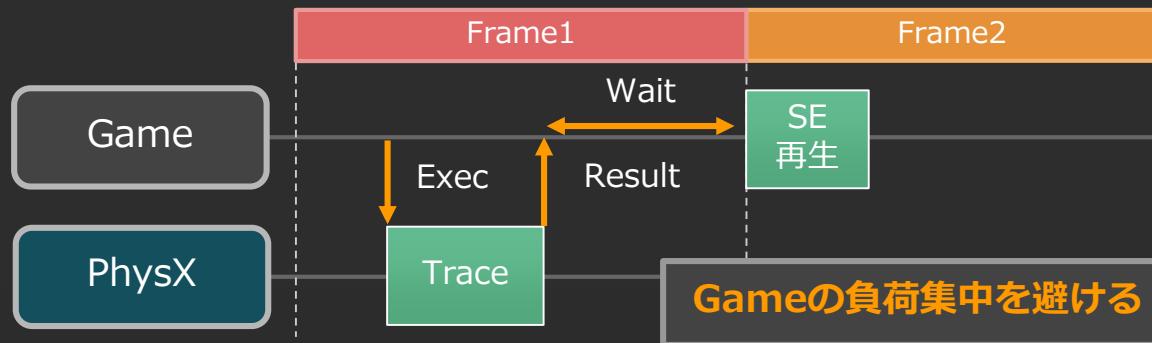
同期Raycast

- ・Game Threadでトレースを実行
- ・Game Threadで結果を待つ
- ・トレース完了後に足音を再生



非同期Raycast

- ・Game Threadで開始を通知
- ・トレースは**非同期**で実行
- ・足音は**次のフレーム以降**で再生



アジェンダ

- Animation
- Physics
- **Navigation & AI**
- Movement
- Networking
- その他



#UE4CEDEC



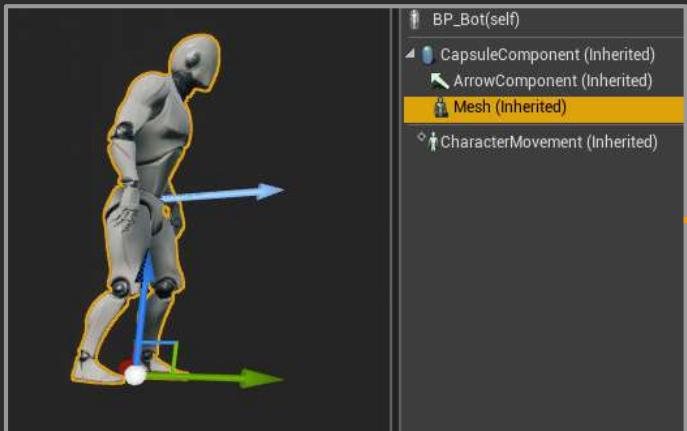
Navigation & AI



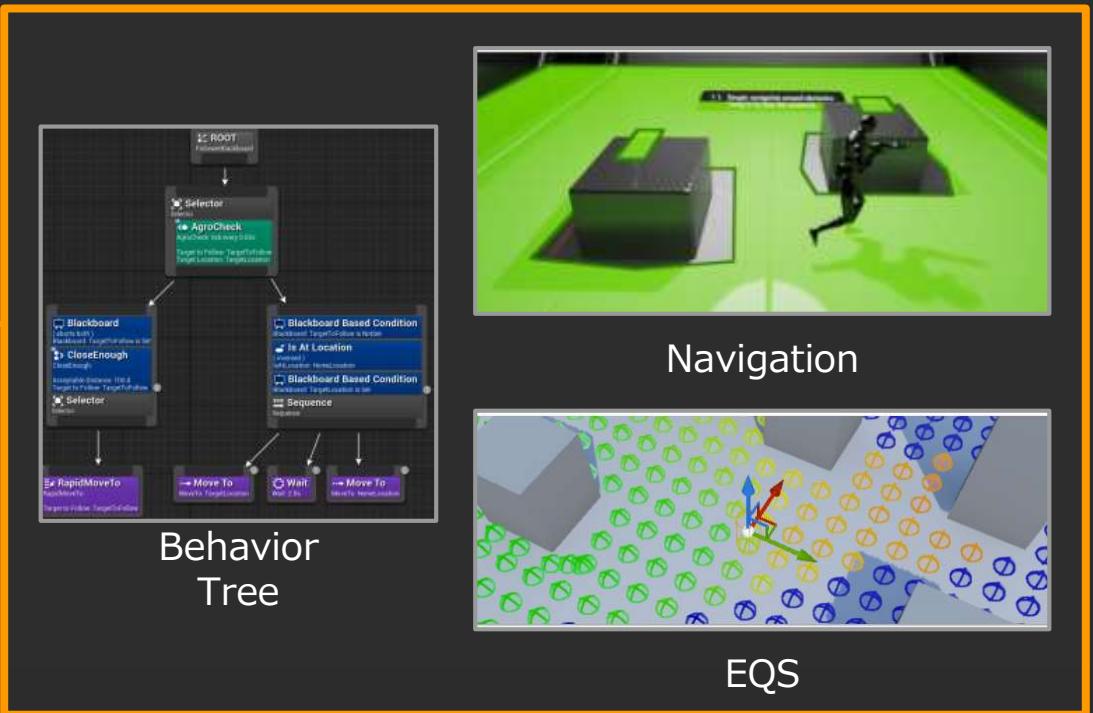
#UE4CEDEC



Character & Navigation & AI



Character

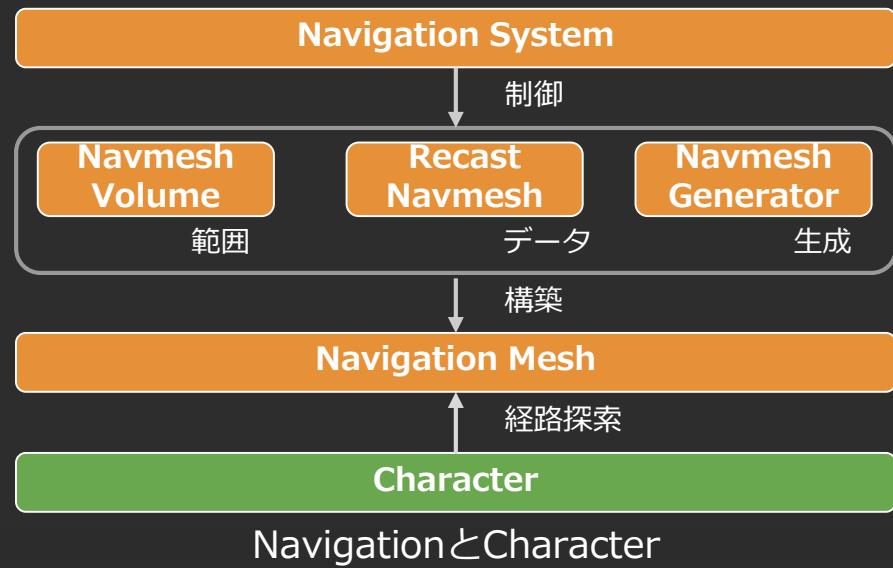
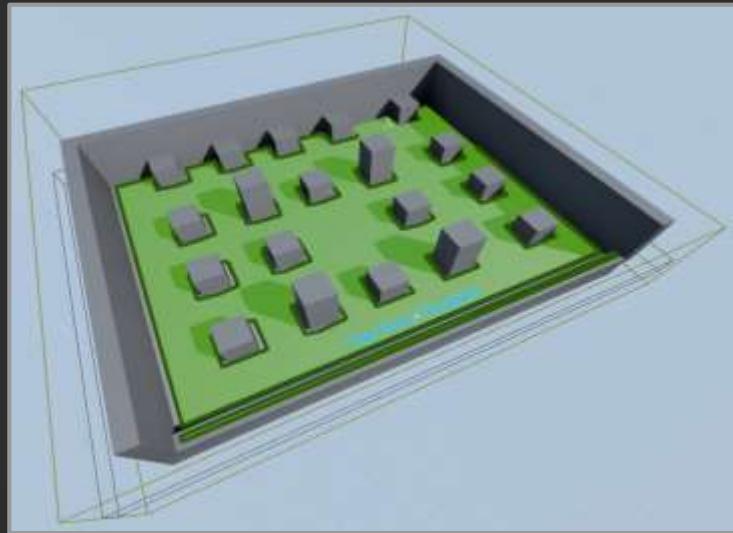


#UE4CEDEC



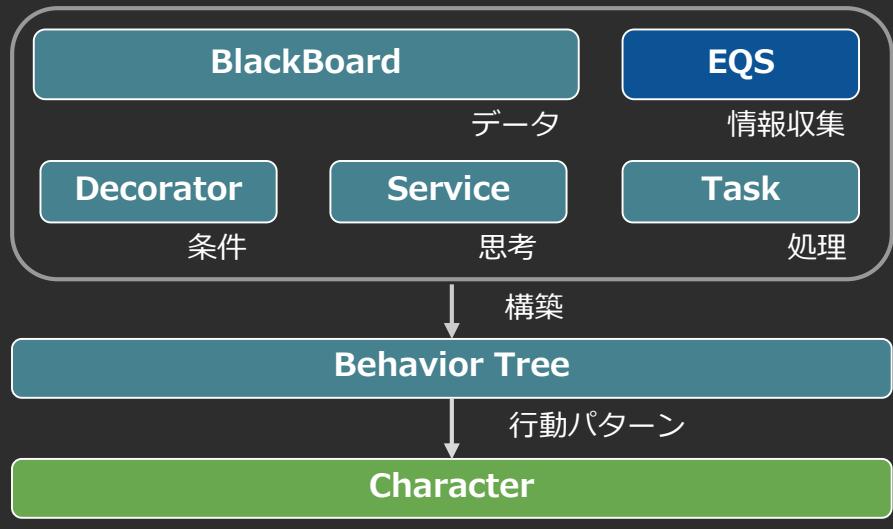
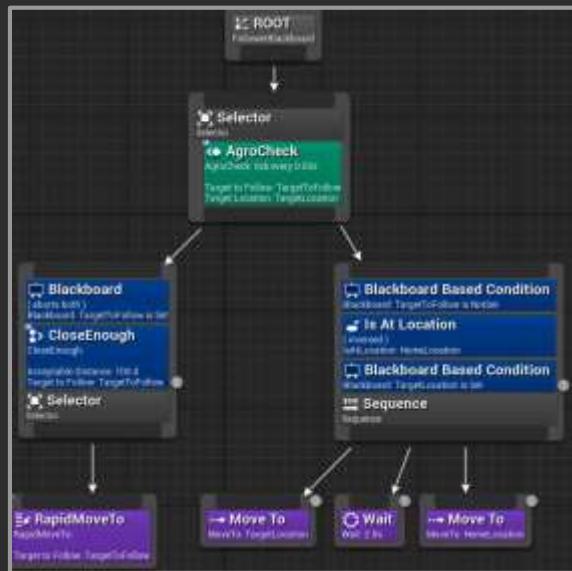
おさらい : Navigation&AI (Navigation)

Navigation Meshを使ってキャラクター移動の経路探索



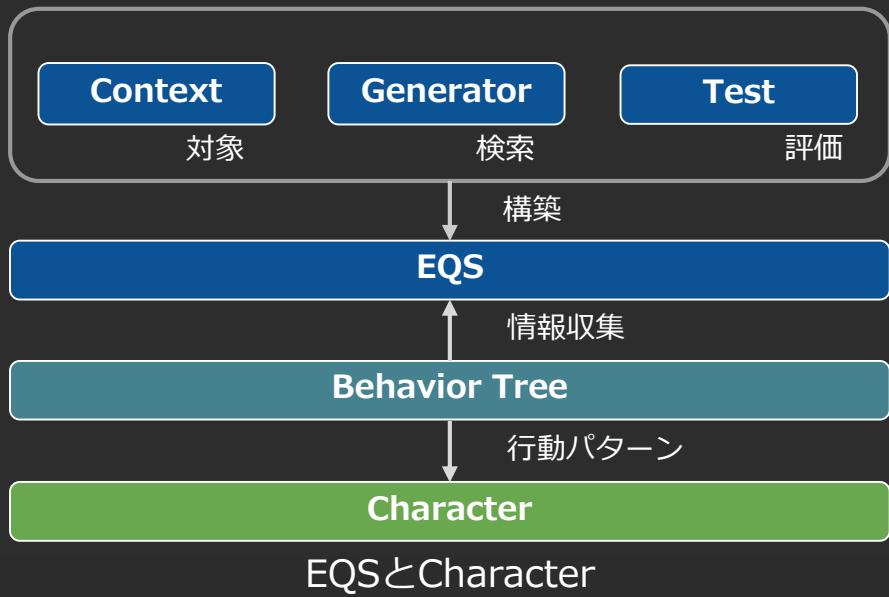
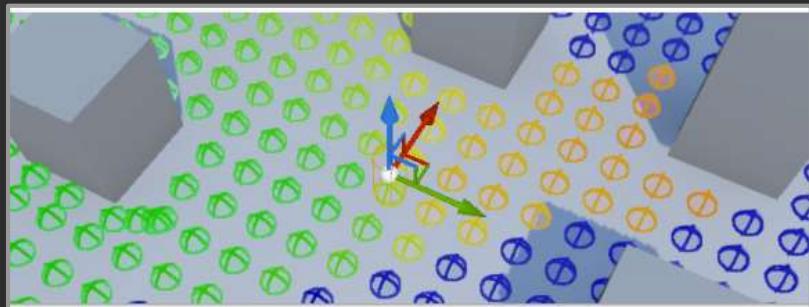
おさらい : Navigation&AI (Behavior Tree)

エージェントの行動パターンを定義



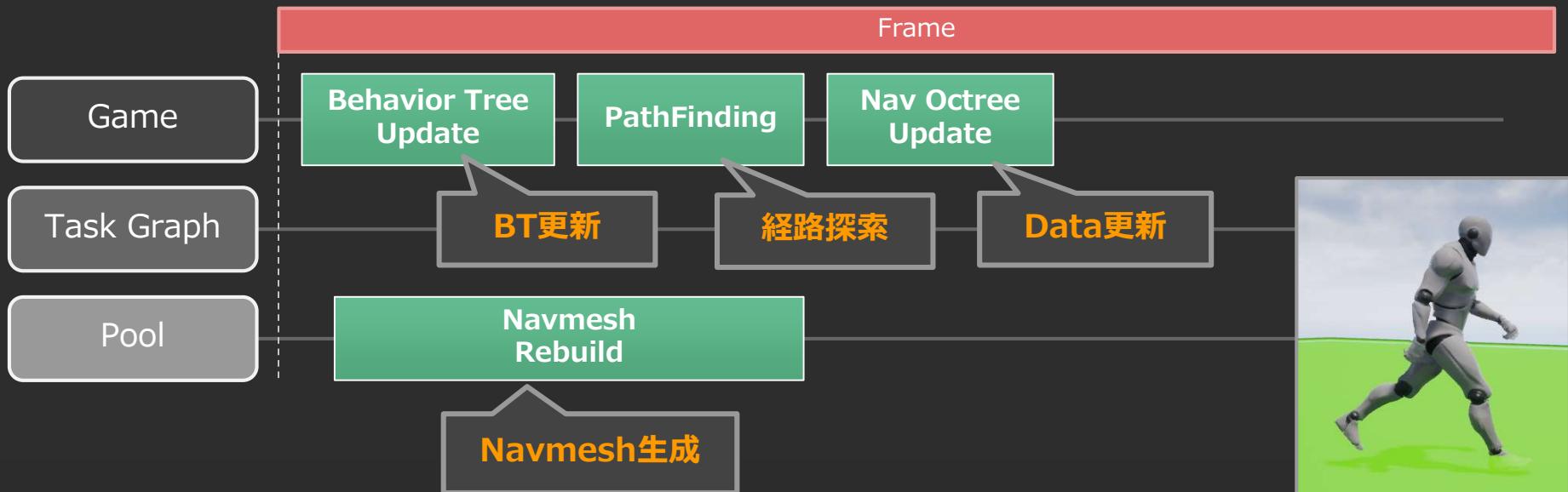
おさらい : Navigation&AI (EQS)

行動選択においてレベルから最適な情報の収集



おさらい : Navigation & AI (処理の流れ)

AIを使用したキャラクターが移動・行動する時の処理の一例



改善ポイント一覧 (Navigation & AI)

- 移動に伴うNavOctreeの更新
- EQSのQuery検索
- BehaviorTreeの実行コスト削減
- BehaviorTreeの動作制御
- AI Perceptionの制御
- AIControllerの制御



移動に伴うNavOctreeの更新



負荷ポイント

- 移動時に所有する全てのComponentの情報をNavOctreeに通知して更新するためNavOctreeの更新コストが嵩む

改善点

- 動的なNavigation Meshの変化が無い場合は、NavOctreeの更新を抑止することで移動時のコストを抑制

備考

- Navmeshを動的に再構築しない場合には更新不要



#UE4CEDEC



移動に伴うNavOctreeの更新



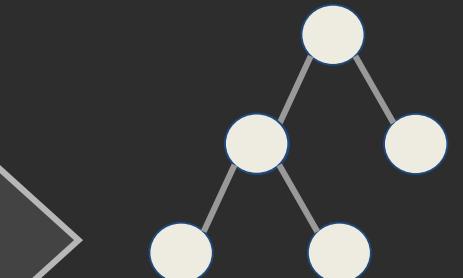
- 負荷ポイント

キャラクターが移動時にNav Octreeの情報を更新する

- 移動に伴い座標更新されたComponent数だけ更新を行う



移動に伴うComponent更新



Nav Octree

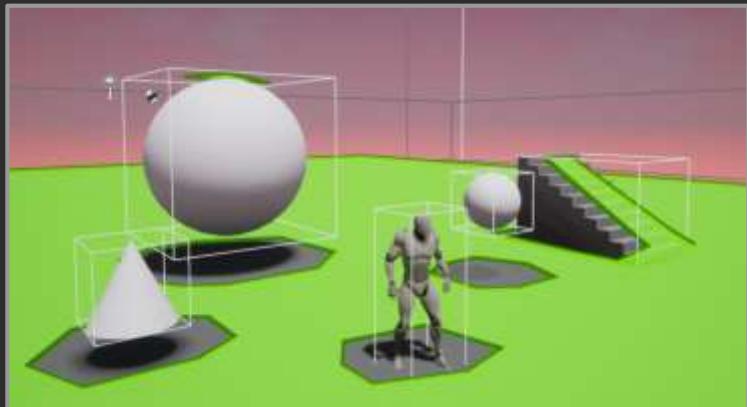
NavOctreeのComponent情報を更新



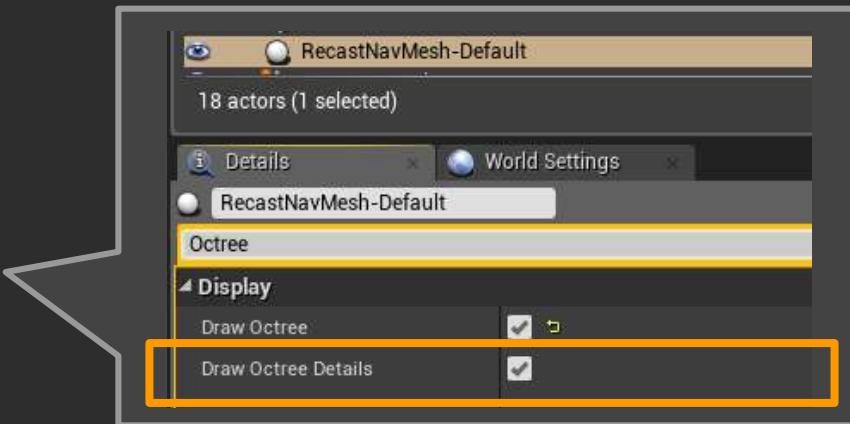
移動に伴うNavOctreeの更新



- NavOctreeとは？
 - Navigation Meshを高速に生成するためのデータ(Bounds/Collision/Voxel等)
 - 移動が発生する度にActorやComponentの情報を更新



Nav OctreeのBounds表示



Octree Details表示機能 (4.20から)



#UE4CEDEC



移動に伴うNavOctreeの更新



● 改善策

NavOctreeが持つComponent情報の更新を無効にする

- 以下のどちらかをNavSystemから実行
- UNavigationSystemV1::SetUpdateNavOctreeOnComponentChange()
- UNavigationSystemV1::ConfigureAsStatic()

```
SetUpdateNavOctreeOnComponentChange(false);
```

```
UCharaNavSystem::UCharaNavSystem()
{
    ConfigureAsStatic();
}
```

無効化した場合RuntimeでのNav Mesh更新が反映されないので注意



移動に伴うNavOctreeの更新



- NavOctreeの更新コスト
 - コンソールコマンド“**stat component**”で確認が可能
 - ComponentやActorの数を減らすことで削減



Component [STATGROUP_Component]	CallCount	InclusiveAvg	InclusiveMax	ExclusiveAvg	ExclusiveMax
Cycle counters (flat)					
UpdateComponentsToWorld	3036	5.89 ms	7.02 ms	0.26 ms	0.367 ms
UpdateChildTransforms	333	5.58 ms	6.72 ms	0.23 ms	0.27 ms
Component.PostUpdateNavData	2092	3.88 ms	5.47 ms	1.85 ms	2.95 ms
Component.UpdateBounds	3080	2.29 ms	2.89 ms	0.02 ms	0.75 ms
Component.UpdateNavData	472	0.23 ms	0.32 ms	0.08 ms	0.16 ms



EQSのQuery検索



負荷ポイント

- EQSでActorOfClassを使用した検索はWorld上の全てのActorを対象とするので検索コストがかかる

改善点

- 検索対象が限定されるケースにおいては、予め対象を絞った検索を行うことで検索コストを下げる

備考

- 特になし



#UE4CEDEC

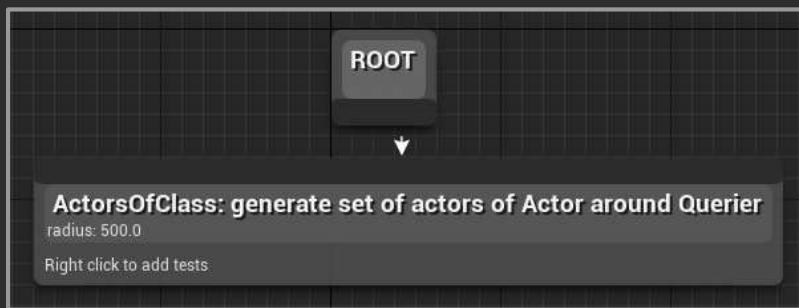


EQSのQuery検索



- 負荷ポイント

- Query検索を行う際にActor Of Classを使用すると検索コストが掛かる
 - Worldから全てのActorを収集しテストを行う



Actor Of ClassによるClass収集

```
for (TActorIterator<AActor> ItActor = TActorIterator<AActor>(World, SearchedActorClass); ItActor; ++ItActor)
{
    for (int32 ContextIndex = 0; ContextIndex < ContextLocations.Num(); ++ContextIndex)
    {
        if (FVector::DistSquared(ContextLocations[ContextIndex], ItActor->GetActorLocation()) < RadiusSq)
        {
            MatchingActors.Add(*ItActor);
            break;
        }
    }
}
```

全Actorが対象なのでStatic Mesh Actorなども含む





- 改善策

検索対象が限られている場合は収集方法を限定する

- Generatorを作成して収集対象を絞る
- 例えばPawnのみを対象とする場合はUWorld::GetPawnIterator()から取得したIteratorのリストから検索

```
■ FConstPawnIterator UWorld::GetPawnIterator() const
{
    auto Result = PawnList.CreateConstIterator();
    return (const FConstPawnIterator&)Result;
}
```



BehaviorTreeの実行コスト削減



負荷ポイント

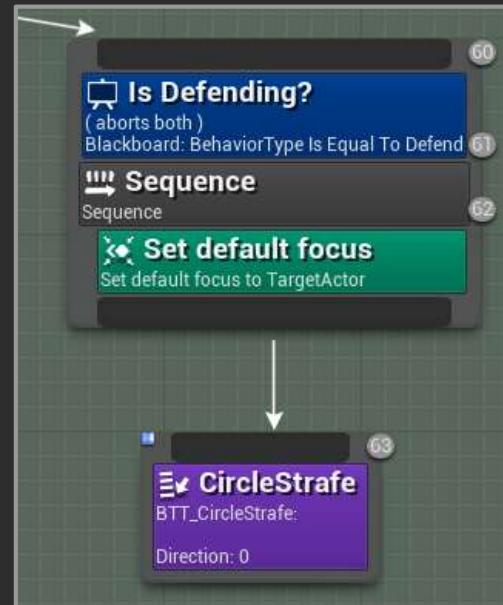
- Behavior Treeはカスタムしやすい反面、常に実行される箇所であるため呼び出しコストが嵩みやすい

改善点

- TaskやDecoratorなどの頻繁に実行されるロジックはC++側で作成したものを使用することでVMコストを削減

備考

- 保守性が低下するので調整して方針が固まった後などのタイミングで適用



BehaviorTreeの動作制御



負荷ポイント

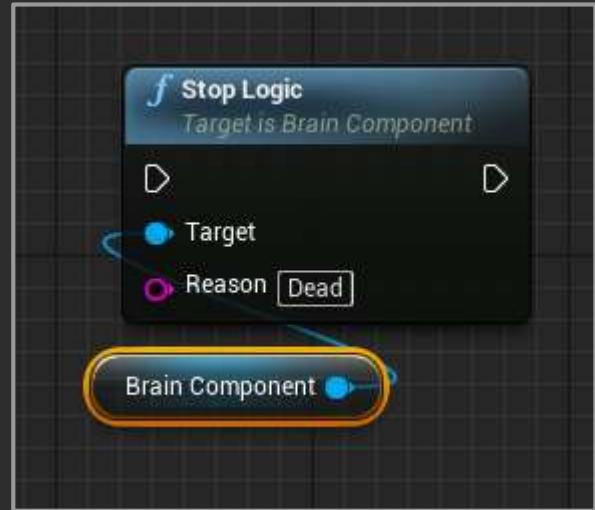
- Behavior Treeは常時稼働しているため、キャラクターが多いケースで常駐コストとして存在

改善点

- プレイヤーからの遠方のキャラクターや更新が必要無いケースなどON/OFFを制御して常駐コストにならないようにする

備考

- 特になし



#UE4CEDEC



BehaviorTreeの動作制御

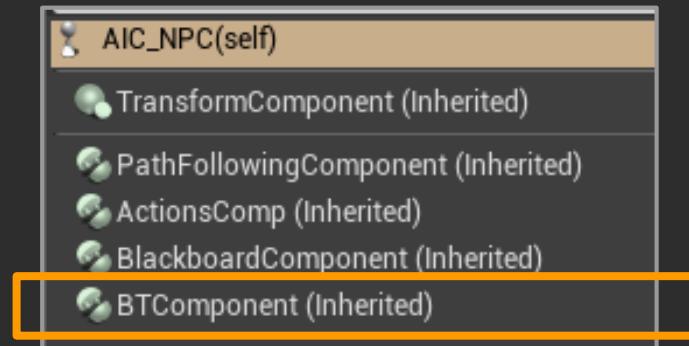


- 負荷ポイント

多数のエージェントのBehavior Treeが常駐コストとして存在
– Wait Taskで待機している状態でもBehavior Tree は稼働



BehaviorTreeを起動すると



Behavior Tree Component が生成
AIControllerからアクセス可能



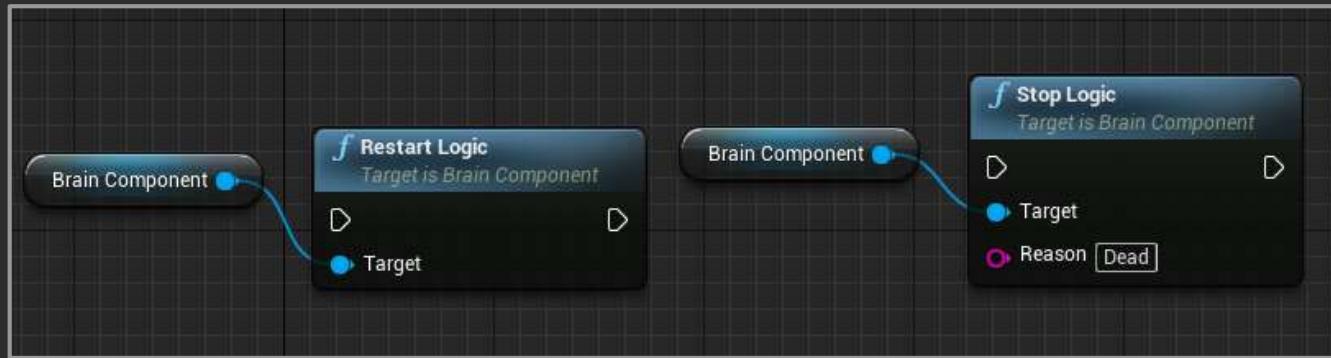
BehaviorTreeの動作制御



- 改善策

Behavior Treeを稼働状態に応じて適切にコントロールする

- 稼働状態を監視してON/OFFを切り替えることで常駐コストを抑制



AIControllerのBrain Component経由で稼働のON/OFFを制御



AI Perceptionの切り替え



負荷ポイント

- 知覚機能など常時稼働しているケースが多くキャラクターが多数出ているとコストが嵩みやすい

改善点

- プレイヤーから一定距離離れた時、ActorPoolingで稼働していない時など適切にON/OFFをコントロールする

備考

- 特になし



#UE4CEDEC



AI Perceptionの制御



- 負荷ポイント／改善策

知覚機能は稼働状態が長くキャラクターが多いと実行コストが嵩む

- エージェントの稼働状態を監視してON/OFFをコントロールする

登録 : UAIPerceptionSystem::UpdateListener

解除 : UAIPerceptionSystem::UnregisterSource



AIControllerの制御



負荷ポイント

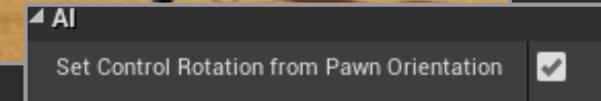
- ControllerのRotation同期機能によりAIControllerは毎フレームエージェントとRotationを同期するがComponentの移動コストとして処理コストに計上される

改善点

- AIControllerの同期が不要であればOFFにすることで回転処理(Componentの移動)コストを抑制する

備考

- 特になし



#UE4CEDEC



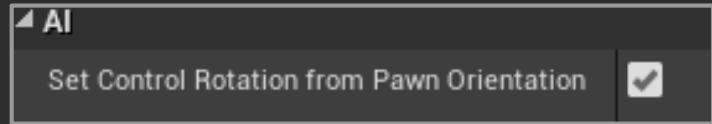
AIControllerの制御



- 負荷ポイント／改善策

AIControllerはPawnとRotation同期を行っているため移動コストとなる

- 同時自体が不要であれば下記の同期設定をOFFにすることが可能
- Rotation同期するメリットがない（他に取得する方法があるため）



この同期処理自体はAIControllerのTickで処理されるため、AIControllerのTick処理自体が不要であればTickから止めるのも可

- AIControllerのTick呼び出しコスト自体を削減



アジェンダ

- Animation
- Physics
- Navigation & AI
- **Movement**
- Networking
- その他



#UE4CEDEC



Movement

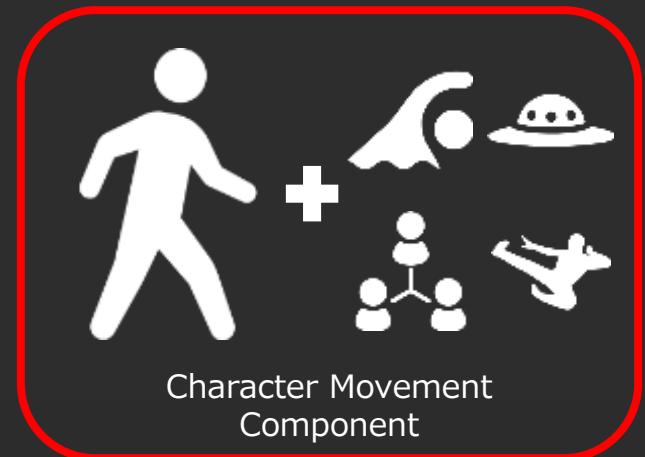
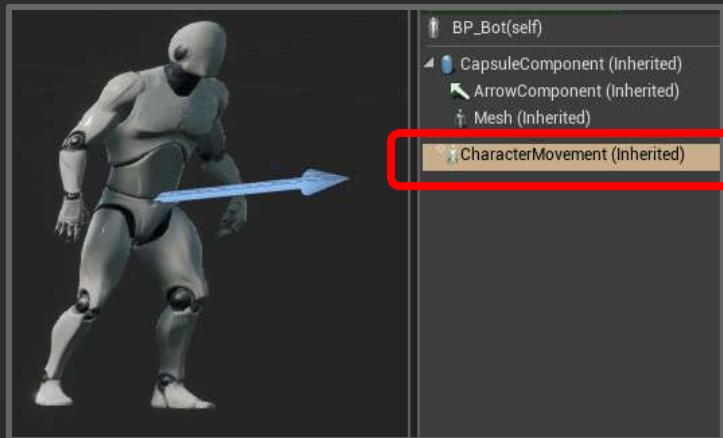


#UE4CEDEC



CharacterとMovement

- Movement = **Character Movement Component**
 - 移動の機能を提供するComponent
 - Walk以外にSwimやFlyといった移動モードやNetworkの機能を備えている



おさらい : Movement

Movement Componentから派生した様々な移動用Componentがある

[UActorComponent](#)

[UMovementComponent](#)

[UInterpToMovementComponent](#)

[UProjectileMovementComponent](#)

[URotatingMovementComponent](#)

[UNavMovementComponent](#)

[UPawnMovementComponent](#)

[UCharacterMovementComponent](#)

[UArchVisCharMovementComponent](#)

[UFloatingPawnMovement](#)

[USpectatorPawnMovement](#)

[UWheeledVehicleMovementComponent](#)

[USimpleWheeledVehicleMovementComponent](#)

[UWheeledVehicleMovementComponent4W](#)

・ 移動抽象クラス

・ 区間移動

・ 平行移動

・ 回転移動

・ 経路探索

・ Pawn移動

・ **Character**移動

・ 建築ビジュアライズ用Character移動

・ 無重力移動

・ 観戦者移動

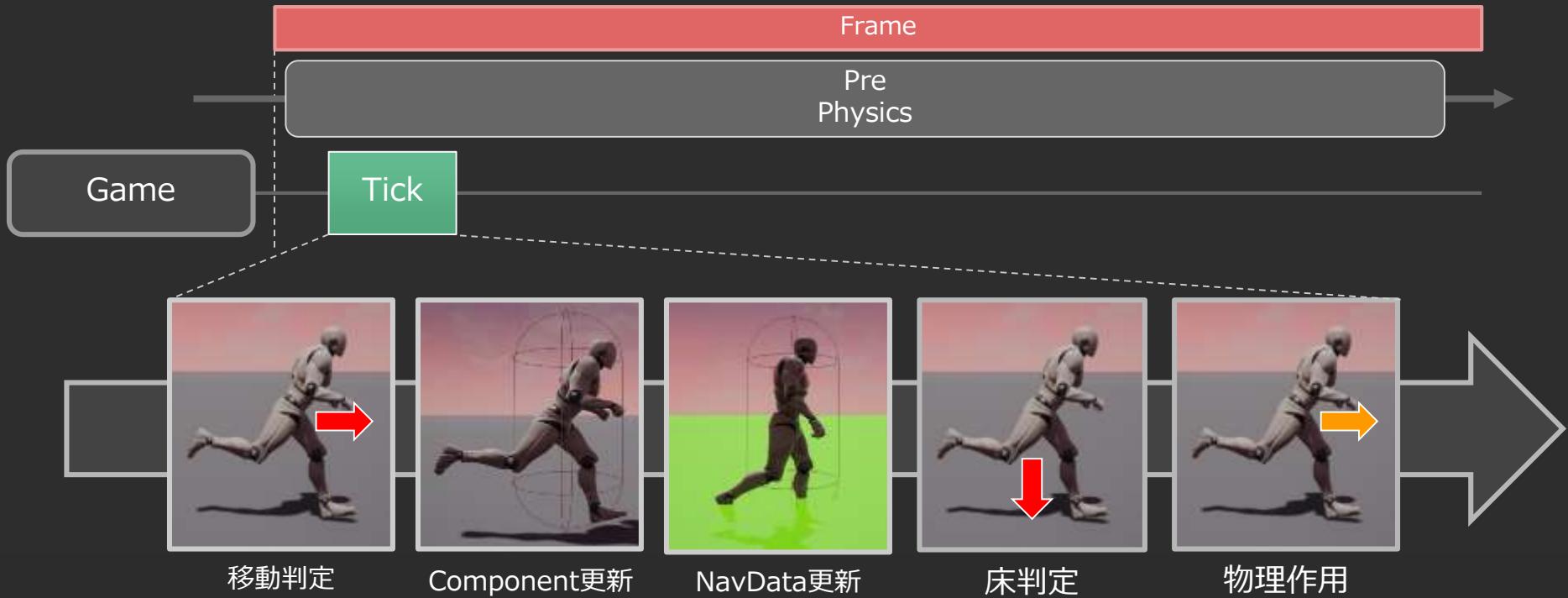
・ 車両シミュレーション移動

・ N輪駆動車移動

・ 4輪駆動車移動



おさらい : Movement (処理の流れ)



改善ポイント一覧 (Movement)

- 静止中の床判定
- 移動中の床／衝突判定
- 移動に伴うComponentの相対的移動 1
- 移動に伴うComponentの相対的移動 2
- 移動に伴う物理作用の適用
- オフスクリーン時の移動更新の抑制



静止中の床判定



負荷ポイント

- 移動していない状態で行う床判定処理においてSweepを使用するため定常コストとして追加される

改善点

- 床判定テストをスキップすることで負荷削減
- 停止しているケースが多いほど効果が大きい

備考

- 動的な床判定への対応ができなくなる



Character Movement: Walking

Always Check Floor



#UE4CEDEC

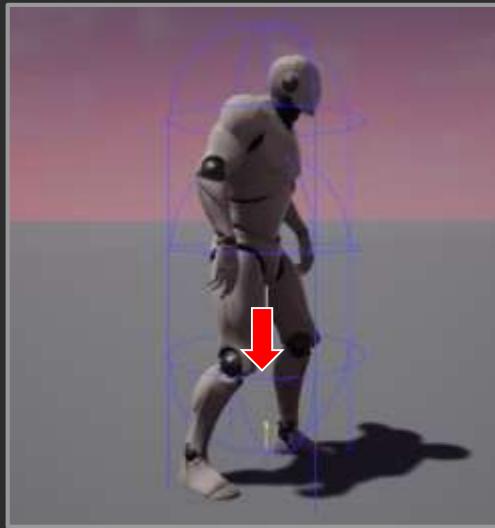


静止中の床判定



- 負荷ポイント

静止状態で**床判定処理**を毎フレーム行うため定常コストに投入される



ID	Frame	Type	Shape	Mode	Tag
354	1,000	Sweep	Capsule	Single	ComputeFloorDist
353	999	Sweep	Capsule	Single	ComputeFloorDist
352	998	Sweep	Capsule	Single	ComputeFloorDist
351	997	Sweep	Capsule	Single	ComputeFloorDist
350	996	Sweep	Capsule	Single	ComputeFloorDist
349	995	Sweep	Capsule	Single	ComputeFloorDist

Collision Analyzer



#UE4CEDEC



静止中の床判定



- 改善策

床判定の処理をスキップすることで静止時の定常コスト削減



Always Check Floor = OFFで床判定スキップ

- デフォルトはON
- 副作用あり



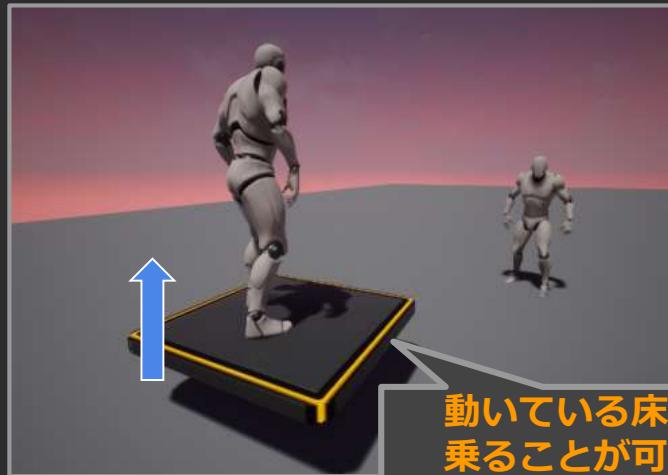
#UE4CEDEC



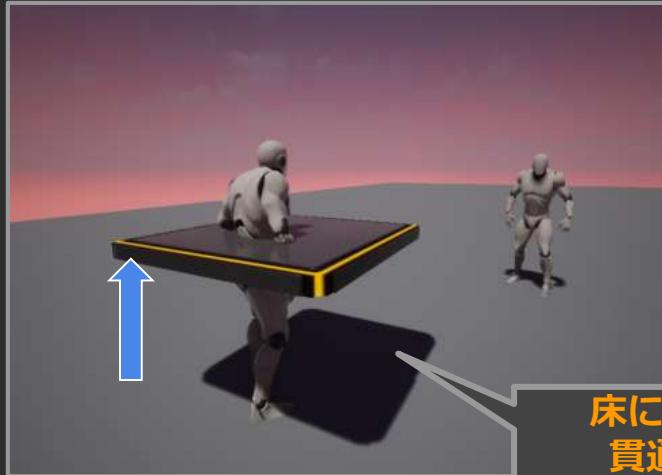
静止中の床判定



床判定処理をスキップした場合、床の変化に対応できないケースがある



床判定をスキップしない
AlwaysCheckFloor = true



床判定をスキップする
AlwaysCheckFloor = false



移動中の床／衝突判定



負荷ポイント

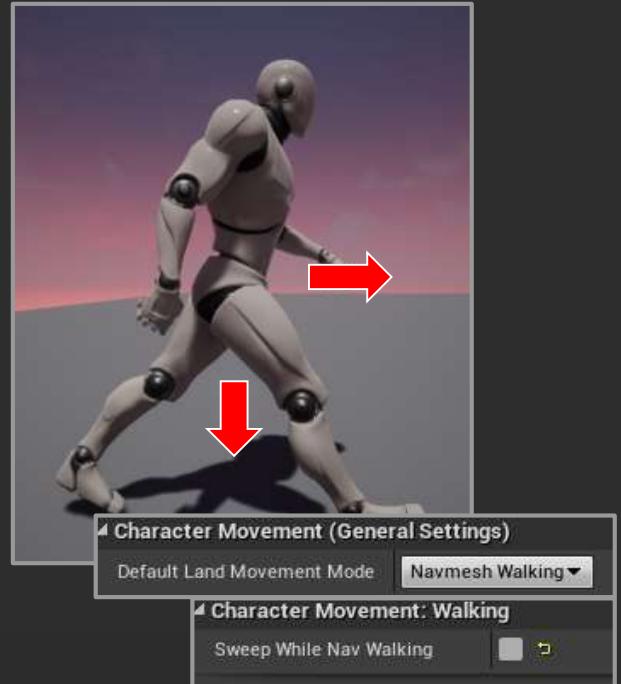
- 移動する際にSweepを用いて衝突判定、及び床判定を行うため移動時の処理コストとして計上される

改善点

- Navmeshを基準とした移動により床判定をスキップ
- Navmesh Walking時は更に移動判定をスキップ

備考

- Navmeshを使用することが前提なのでNPCのキャラクターなどに限定される

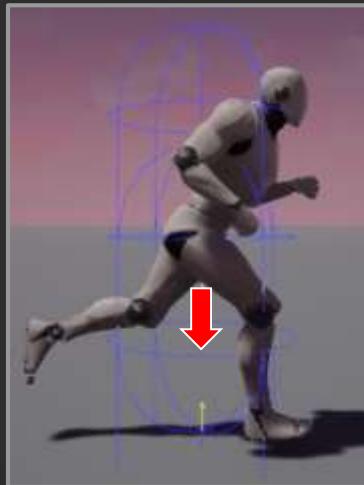
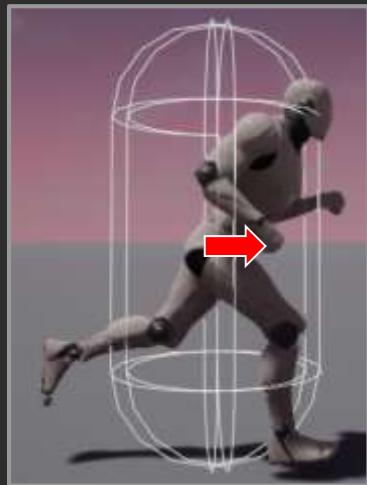


移動中の床／衝突判定



- 負荷ポイント

移動状態で**衝突と床判定処理**を行うことによるGameの負荷が嵩む



ID	Frame	Type	Shape	Mode	Tag
1,927	904	Sweep	Capsule	Multi	MoveComponent
1,926	903	Sweep	Capsule	Single	ComputeFloorDist
1,925	903	Sweep	Capsule	Multi	MoveComponent
1,924	902	Sweep	Capsule	Single	ComputeFloorDist
1,923	902	Sweep	Capsule	Multi	MoveComponent
1,922	901	Sweep	Capsule	Single	ComputeFloorDist
1,921	901	Sweep	Capsule	Multi	MoveComponent

Collision Analyzer



#UE4CEDEC

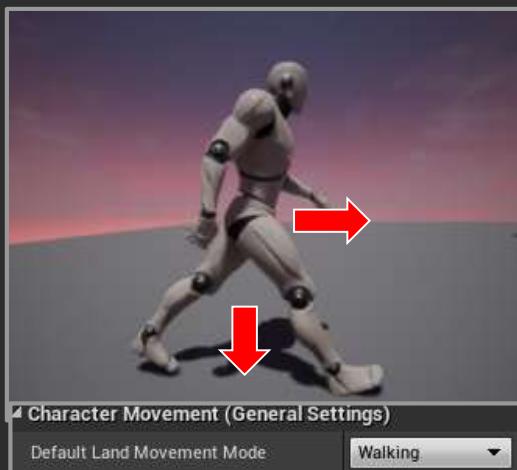


移動中の床／衝突判定



● 改善策

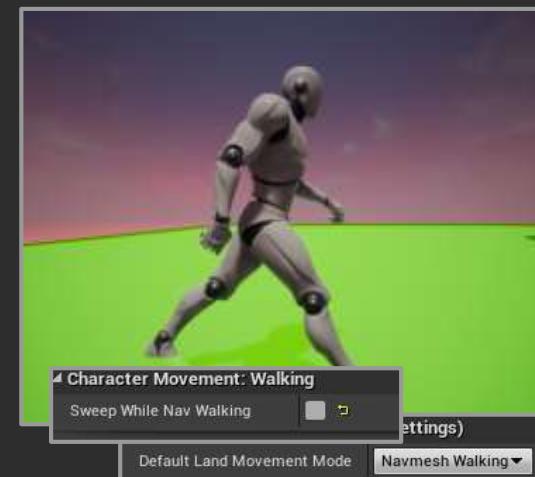
NavWalking及びSweepスキップにより移動時のコスト削減



デフォルト設定



床判定スキップ



床／衝突判定スキップ



移動中の床／衝突判定



- Characterの床判定や移動コスト
 - コンソールコマンド“**stat character**”で確認可能
 - 負荷になっている時は、床判定スキップやNavmeshWalkingを検討



Character [STATGROUP_Character]	CallCount	InclusiveAvg	InclusiveMax	ExclusiveAvg	ExclusiveMax
Cycle counters (flat)					
Char Tick	501	56.36 ms	483.56 ms	0.46 ms	1.26 ms
Char NonSimulated Time	501	55.02 ms	481.31 ms	0.18 ms	0.41 ms
Char PerformMovement	501	55.34 ms	480.64 ms	0.96 ms	2.90 ms
Char PhysWalking	501	50.02 ms	451.69 ms	5.07 ms	61.35 ms
Char FindFloor	515	14.87 ms	145.93 ms	0.50 ms	1.44 ms
Char Physics Iteration	501	0.18 ms	0.00 ms	0.18 ms	0.00 ms
Char Update Acceleration	501	0.10 ms	0.22 ms	0.10 ms	0.22 ms
Char HandleImpact	313	0.12 ms	0.32 ms	0.12 ms	0.32 ms
Char PhysFalling					
Char MoveUpdateDelegate	501	0.07 ms	0.09 ms	0.07 ms	0.09 ms
Char RootMotionSource Apply	515	0.03 ms	0.07 ms	0.03 ms	0.07 ms
Char AdjustFloorHeight	317	0.02 ms	0.06 ms	0.02 ms	0.04 ms
Char ProcessLanded					

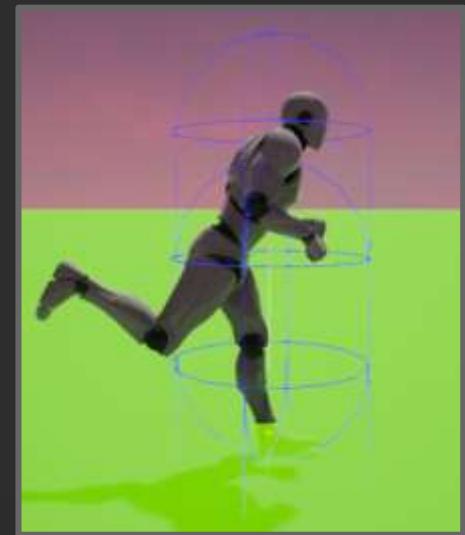


移動中の床／衝突判定



- Navmesh Walkingモードの注意点 1
 - Navigaiton Dataが無いと**Walkingモードに戻る**
 - Walkingモードに戻るとSweep判定が実行されます

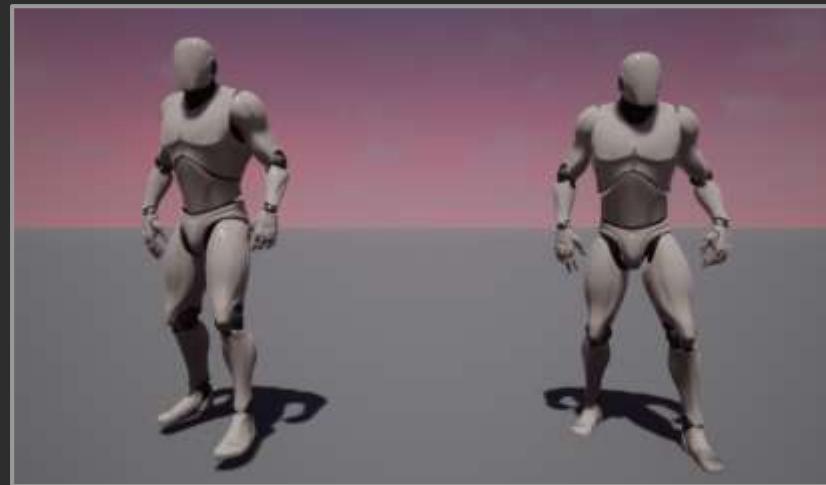
```
// If trying to use NavWalking but there is no navmesh, use walking instead.  
if (NewMovementMode == MOVE_NavWalking)  
{  
    if (GetNavData() == nullptr)  
    {  
        NewMovementMode = MOVE_Walking;  
    }  
}
```



移動中の床／衝突判定



- Navmesh Walkingモードの注意点 2
 - Navmesh上を歩行するので完全には接地していない



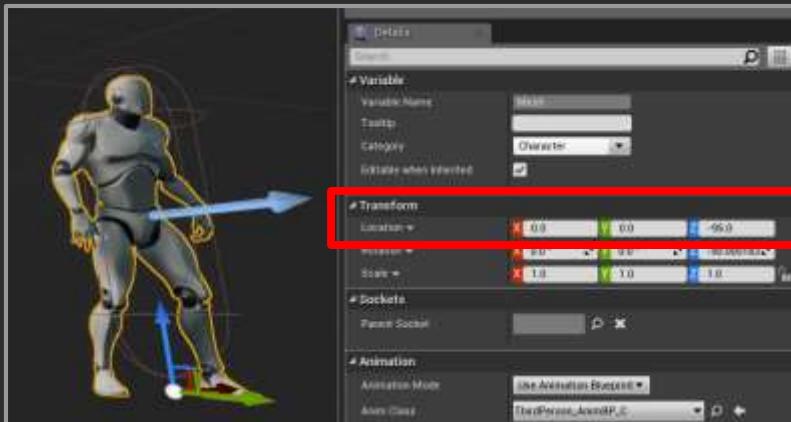
Navmesh WalkingモードとWalkingモードの比較
(左がNavmesh Walkingモード)



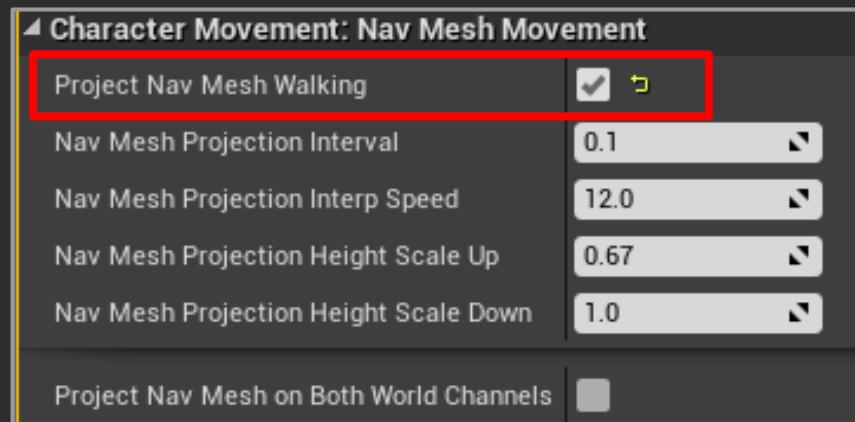
移動中の床／衝突判定



- Navmesh Walkingでの接地への対応
 - Skeletal MeshのLocationを調整
 - Navmeshの投射機能を使用 (Raycastでジオメトリを探して接地する)



Meshの高さ調整で見た目を合わせる



Raycastで設置面をチェック



#UE4CEDEC



移動に伴うComponentの相対的移動 1



負荷ポイント

- 移動やアニメーションの際に所有しているComponentの座標も併せて更新することでコストになる
- 多くのComponentを所有する場合はコストが嵩む

改善点

- Scene Component (位置情報を持つComponent) の使用を厳密に管理して無駄にComponentを増やしすぎない

備考

- 複数のComponent(Skeletal Mesh等)を統合することでComponentを減らすことも可能



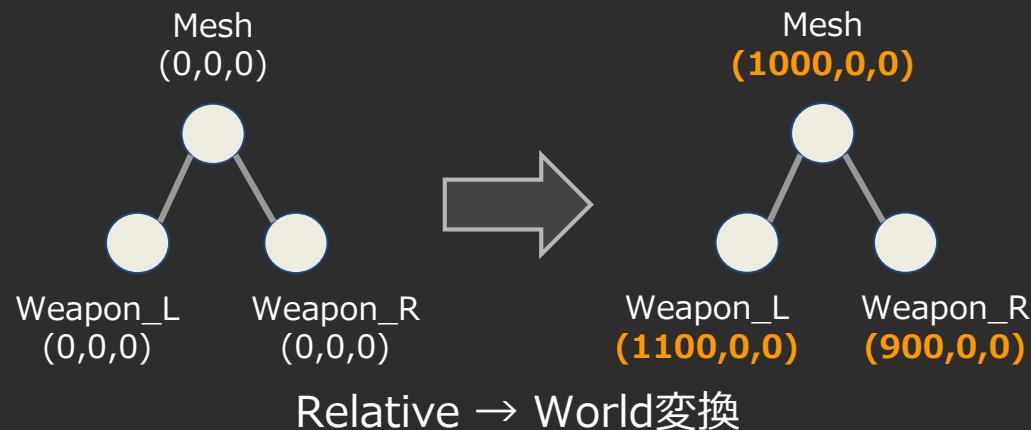
移動に伴うComponentの相対的移動 1



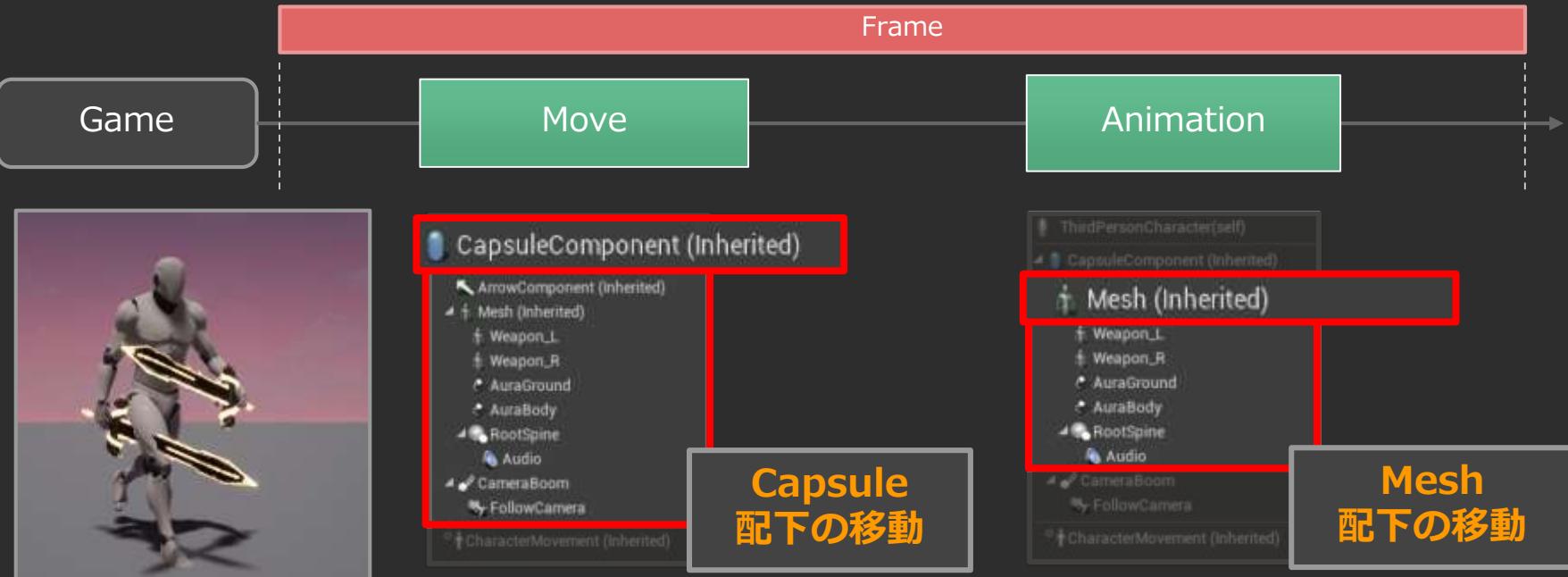
- 負荷ポイント／改善策

移動やアニメーション時にComponentの座標を更新する

- 所有するComponentを相対座標からワールド座標に変換して再配置
- Component数が多いほど再配置コストが嵩むため数を厳密に管理



移動に伴うComponentの相対的移動 1



キャラクターの移動に伴うComponentの移動



移動に伴うComponentの相対的移動 2



負荷ポイント

- 移動やアニメーションの際に所有しているComponentの座標も併せて更新することでコストになる
- 多くのComponentを所有する場合はコストが嵩む

改善点

- ComponentがActiveで無い場合はComponentを切り離すことでComponentの移動コストを削減

備考

- Audio ComponentとParticle System Component のみで使用可能な設定



#UE4CEDEC



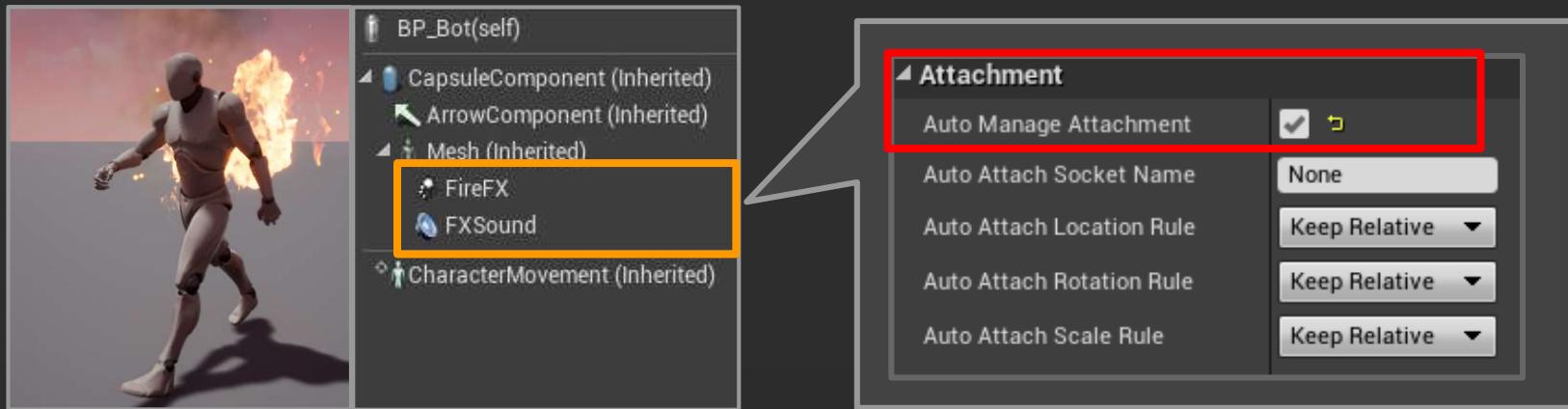
移動に伴うComponentの相対的移動 2



● 負荷ポイント／改善策

キャラクターの移動に伴うComponentの同期コストを減らしたい

- Attachしていない場合はComponentの座標変換が不要
- AudioやParticleはアクティブでない場合は自動的にDetachする



移動に伴うComponentの相対的移動 2



- Componentの変換や座標更新コスト
 - “**stat component**”で確認することができます
 - 負荷になっている場合はComponent数やDetachを適用



Component [STATGROUP_component]	CallCount	InclusiveAvg	InclusiveMax	ExclusiveAvg	ExclusiveMax
UpdateComponentToWorld	10	0.05 ms	0.09 ms	0.00 ms	0.00 ms
UpdateChildTransforms	35	0.05 ms	0.06 ms	0.00 ms	0.01 ms
Component::UpdateBounds	22	0.02 ms	0.03 ms	0.01 ms	0.02 ms
Component::PostUpdateNavData	2	0.02 ms	0.03 ms	0.01 ms	0.02 ms
RegisterComponent					
Component::CreatePhysicsState					
UnregisterComponent					
Component::OnRegister					
Component::CreateRenderState					
Component::DestroyPhysicsState					
Component::OnUnregister					
Component::DestroyRenderState					



移動に伴う物理作用の適用



負荷ポイント

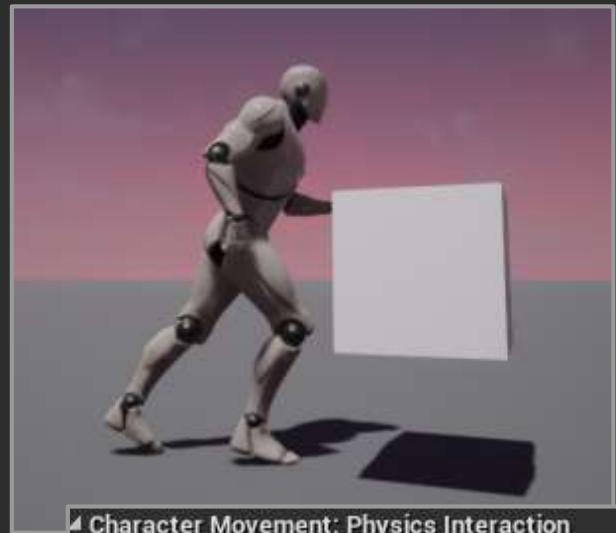
- 移動した際に物理作用(下向きの力、反発力)を発生させて周辺にインテラクションを反映

改善点

- 移動に伴う物理作用が不要な場合はオフにすることでインテラクションのコスト削減

備考

- 移動に伴う物理作用の適用は設定に従うが、単純な衝突判定についてはCapsule Collisionで行う



Character Movement: Physics Interaction

Enable Physics Interaction



#UE4CEDEC



移動に伴う物理作用の適用

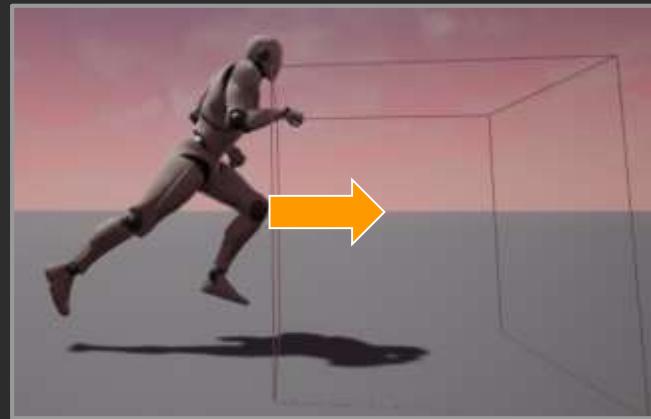
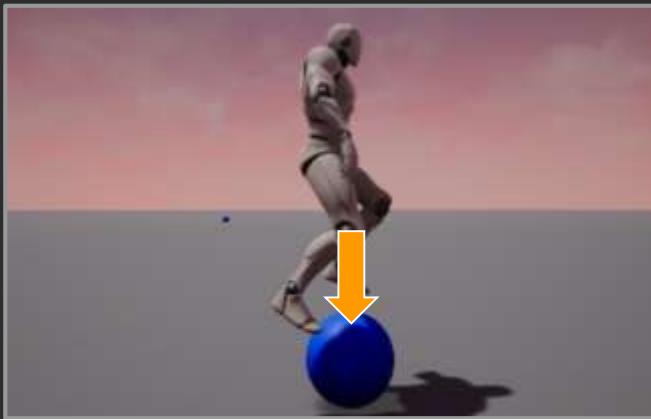


● 負荷ポイント／改善策

- ・ 移動時に下向きの力と反発力を適用
 - OFFに設定すると物理作用は適用されないがコストダウン
 - 基本的にはCapsuleの接触判定でオブジェクト同士はブロック

Character Movement: Physics Interaction

Enable Physics Interaction



#UE4CEDEC



オフスクリーン時の移動更新の抑制



負荷ポイント

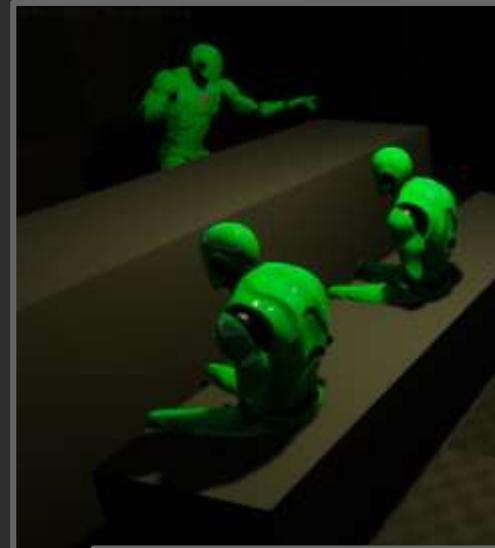
- 床判定や移動判定処理が毎フレーム実行されることで、移動しないキャラクターなどの重要度が低いキャラクターも同一コストが発生

改善点

- 描画されていないキャラクターはCharacter Movement のTick処理をスキップすることでコスト削減

備考

- Tick処理をスキップするので移動自体を行わなくなる
- 「描画されていない時に移動が不要なキャラクター」などの限られたケースで有効



#UE4CEDEC



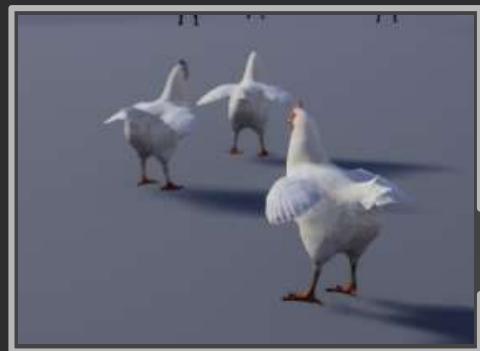
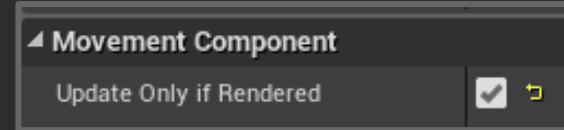
オフスクリーン時の移動更新の抑制



- 負荷ポイント／改善策

重要度が低くても同一の移動処理コストが発生

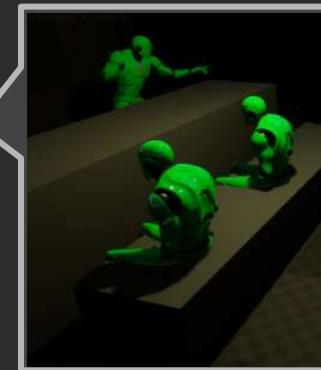
- オフスクリーン時の移動処理スキップ



にぎやかし



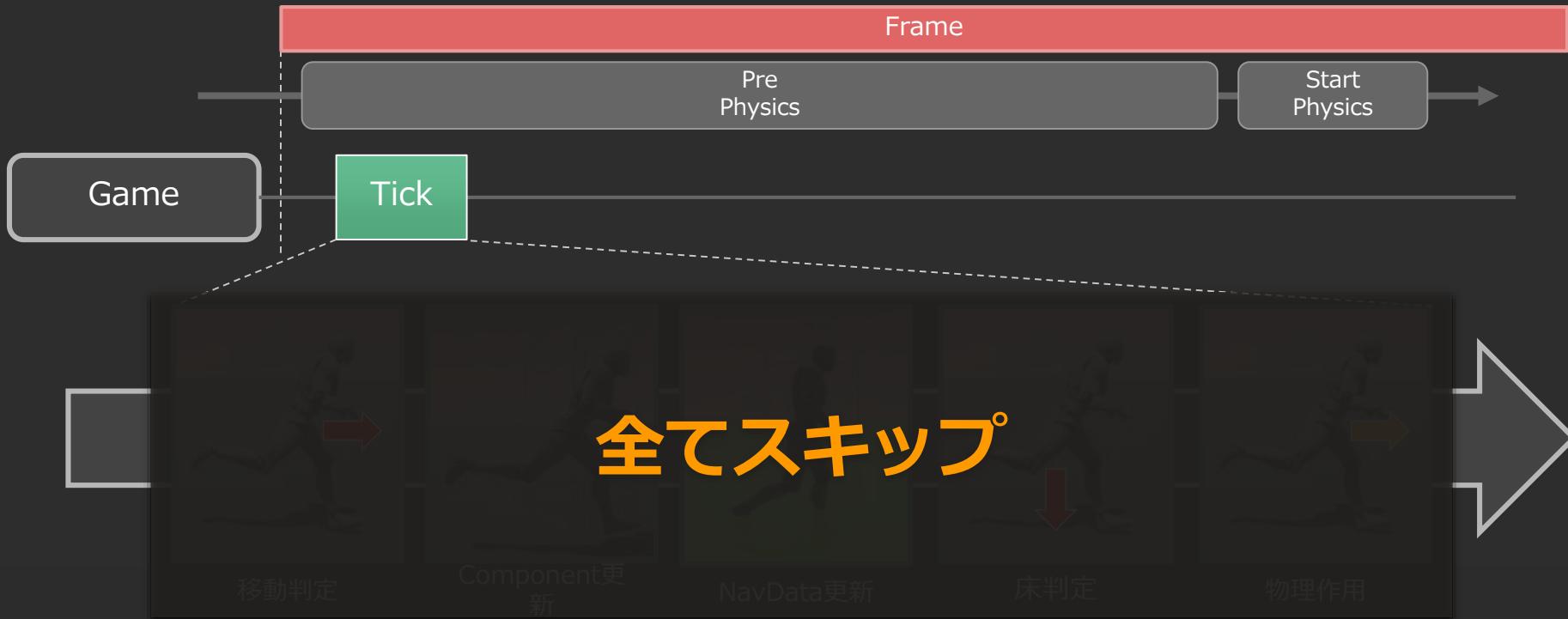
ゲーム中のカメラ視点



建物の中にいる



オフスクリーン時の移動更新の抑制



アジェンダ

- Animation
- Physics
- Navigation & AI
- Movement
- **Networking**
- その他



#UE4CEDEC



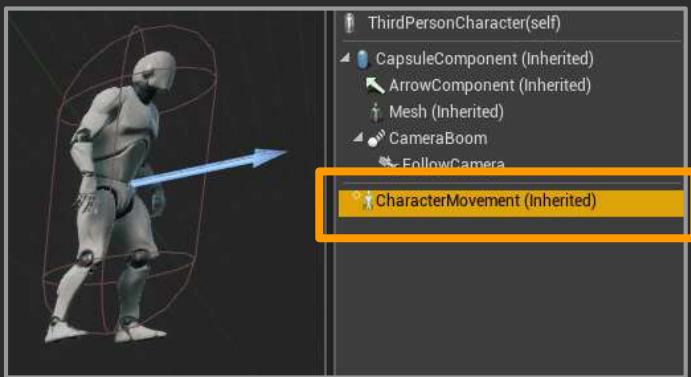
Networking



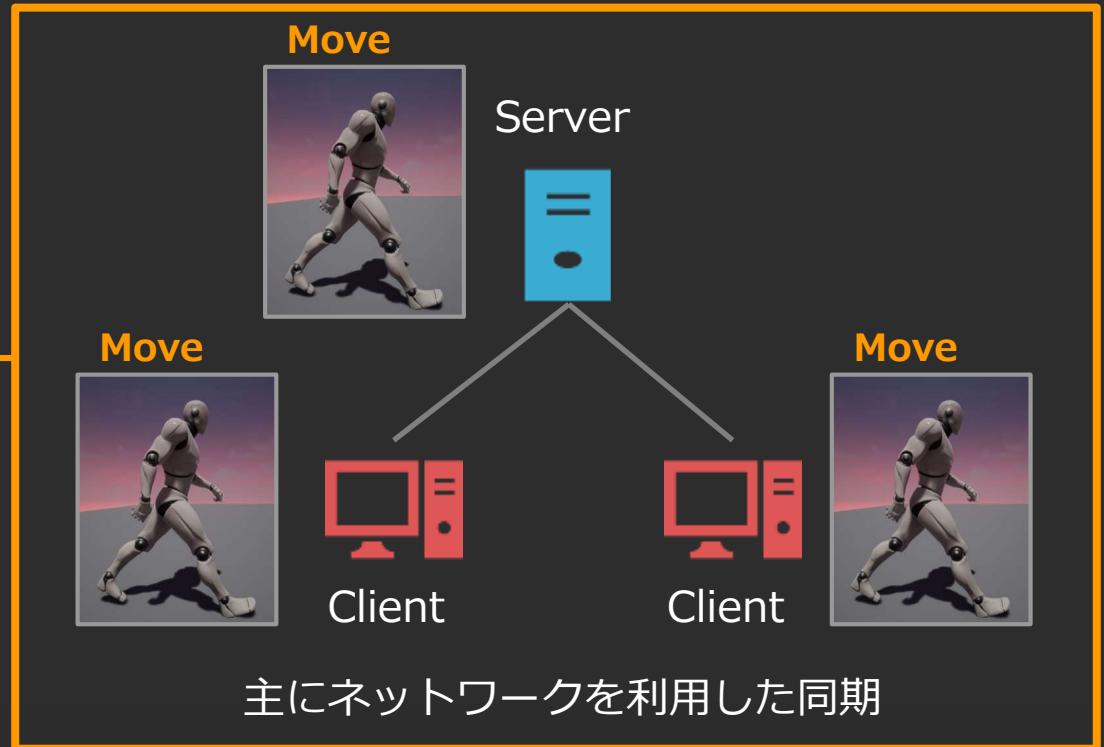
#UE4CEDEC



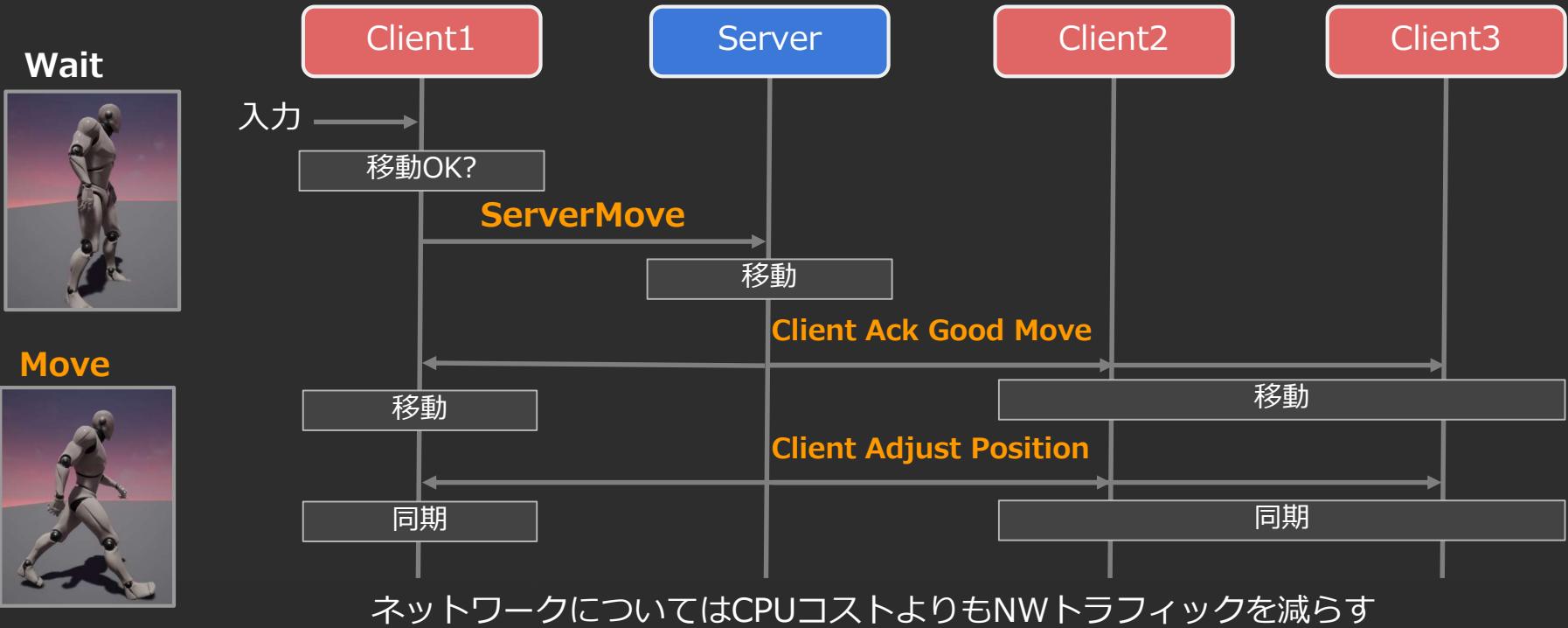
CharacterとNetworking



Character



おさらい : Networking (処理の流れ)



改善ポイント一覧 (Networking)

- 移動に伴うトラフィック調整
- 不要なReplicationの通知を抑制
- Dedicated Serverでの物理シミュレーション
- Replicationの効率的な通知
- Replicationデータの効率的な設定



移動に伴うトラフィック調整



負荷ポイント

- 多数のキャラクターが存在する場合、データの同期によりReplicationでCPUコストを逼迫する

改善点

- ネットワーク設定を環境に併せて設定し、ネットワーク トラフィックを調整

備考

- 特になし



移動に伴うトラフィック調整



- 多くのキャラクターが居る場合は特にServer-Client間での同期コストが増えるため、Network設定を想定環境に併せて調整

```
[/Script/OnlineSubsystemUtils.IpNetDriver]
AllowPeerConnections=False
AllowPeerVoice=False
ConnectionTimeout=60.0
InitialConnectTimeout=60.0
TimeoutMultiplierForUnoptimizedBuilds=1
KeepAliveTime=0.2
MaxClientRate=15000
MaxInternetClientRate=10000
```

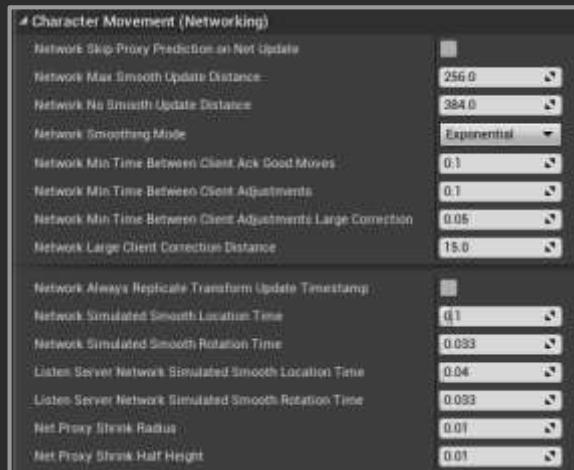
Network送受信に関するパラメータの設定 (Engine.ini)



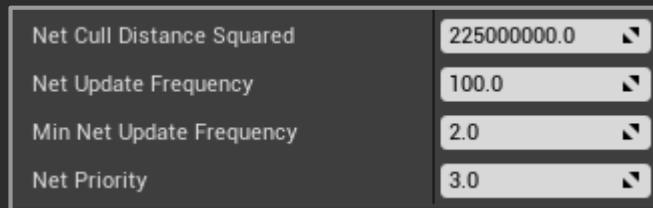
移動に伴うトラフィック調整



- 多くのキャラクターが居る場合は移動の同期コストが増えるので、移動に関するNetwork設定を調整



移動関連の調整値 (CharacterMovement)



Actor関連の調整値 (Actor)



移動に伴うトラフィック調整



以下のスライドに各種項目などの説明がありますのでご参考下さい

Multiplay Online Deepdive



編

<https://www.slideshare.net/EpicGamesJapan/ue4-multiplayer-online-deep-dive-1-byking-ue4dd>



不要なReplicationの通知を抑止



負荷ポイント

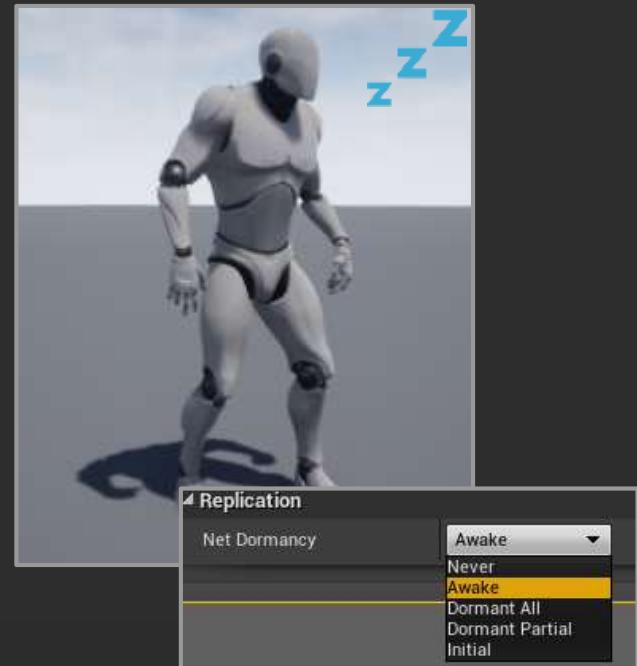
- 多数のキャラクターが存在する場合、データの同期によりReplicationでCPUコストを逼迫する

改善点

- 休止状態を適切にコントロールすることでReplicationを抑制することによりトラフィック及びコスト抑制

備考

- 特になし



#UE4CEDEC



不要なReplicationの通知を抑止



- 休止状態を適宜制御して不必要的データのやりとりを抑制
 - キャラクターの稼働状態に併せてゲーム側から動的に制御
 - 必要な情報はAActor::FulshNetUpdate()で強制的に送信



- net.UseAdaptiveNetUpdateFrequency
 - 更新頻度が低いほど頻繁に更新レートを落とし、頻繁に更新するにつれて更新レートを上げる機能



Dedicated Serverでの物理シミュレーション



負荷ポイント

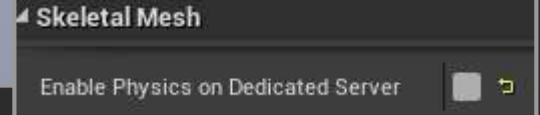
- 多くのキャラクターが存在してPhysics Bodyを持つ場合
物理シミュレーションのコストが嵩む

改善点

- Dedicated Serverでは物理シミュレーションが不要のため無効化することでサーバー上のCPUコスト削減

備考

- Dedicated Serverでのみ適用されるオプション



#UE4CEDEC



Replicationデータの効率的な通知



負荷ポイント

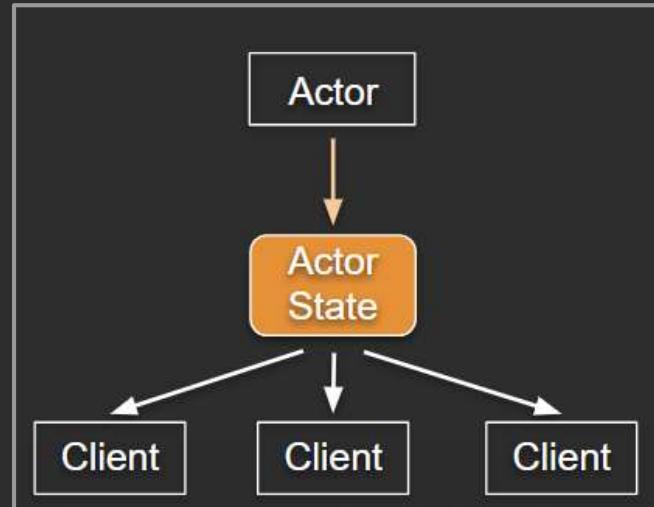
- Actorが状態を複製する際に同じ状態が多くのClientに送信されることがある

改善点

- 同一情報をキャッシュ、再利用して効率的にReplicationやRPCなどのシリアルライズデータを送信できる

備考

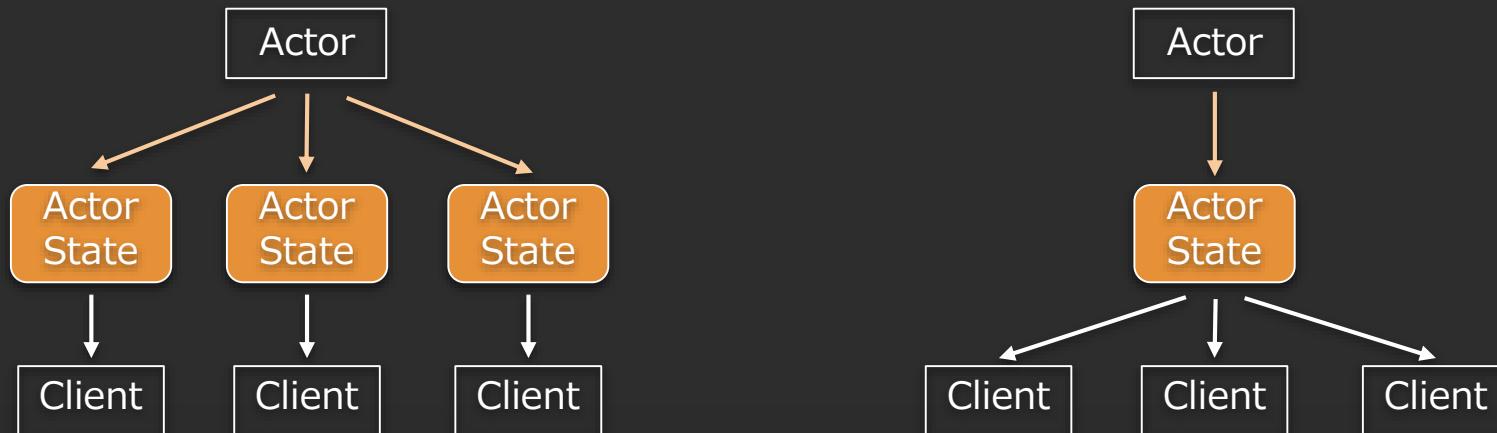
- 4.20からの機能



Replicationデータの効率的な通知



- キャッシュ／再利用が可能なシステムを利用し効率化 (4.20から)
 - “net.ShareSerializedData=1” で有効
 - ReplicationとRPCのシリアル化したデータを共有して再利用



Replicationデータの効率的な設定



負荷ポイント

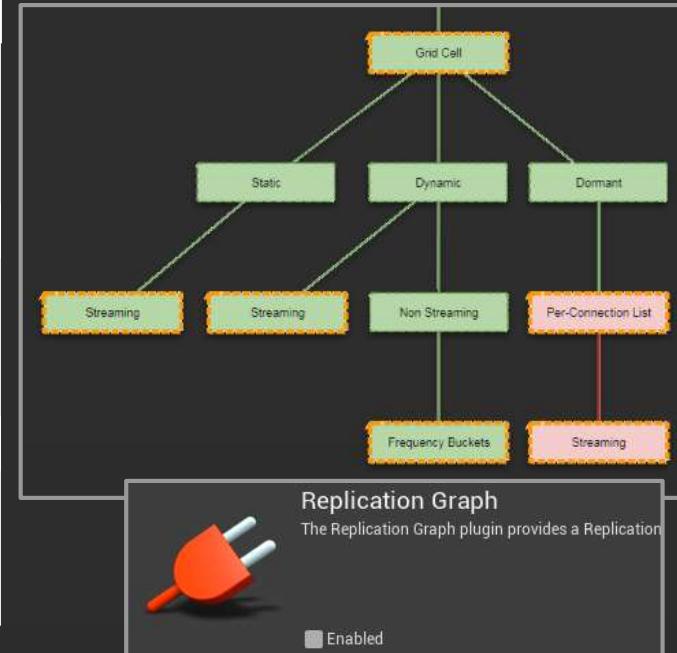
- 多数のキャラクターが存在する場合、データの同期によりReplicationでCPUコストを逼迫する

改善点

- Replication Graphを使用することでReplicationの収集や優先付けを適切に行うことが可能

備考

- 4.20からのPlugin機能 (Experimental)
- プロジェクトで独自の実装が必要



#UE4CEDEC





Replicationデータの効率的な設定



- Shooter GameにReplication Graphのサンプルがあります

```
void UShooterReplicationGraph::InitGlobalActorClassSettings()
{
    Super::InitGlobalActorClassSettings();

    // -----
    // Programatically build the rules.
    //

    auto AddInfo = [&]( UClass* Class, EClassRepNodeMapping Mapping ) { ClassRepNodePolicies.Set( Class, Mapping ); };

    AddInfo( AShooterWeapon::StaticClass(), EClassRepNodeMapping::NotRouted ); // Handled via De
    AddInfo( ALevelScriptActor::StaticClass(), EClassRepNodeMapping::NotRouted ); // Not needed
    AddInfo( APlayerState::StaticClass(), EClassRepNodeMapping::NotRouted ); // Special cased
    AddInfo( AReplicationGraphDebugActor::StaticClass(), EClassRepNodeMapping::NotRouted ); // Not needed. Re
    AddInfo( AInfo::StaticClass(), EClassRepNodeMapping::RelevantAllConnections ); // Non spatialize
    AddInfo( AShooterPickup::StaticClass(), EClassRepNodeMapping::Spatialize_Static ); // Spatialized ar

#if WITH_GAMEPLAY_DEBUGGER
    AddInfo( AGameplayDebuggerCategoryReplicator::StaticClass(), EClassRepNodeMapping::NotRouted ); // Replicated via
#endif
```



アジェンダ

- Animation
- Physics
- Navigation & AI
- Movement
- Networking
- **その他**



#UE4CEDEC



その他



#UE4CEDEC



その他一覧

- Affinity設定
- Tickコントロール
- コンソールコマンド
- Significance Manager
- Particle Pooling



#UE4CEDEC



Affinityの設定 (Console Only)

- スレッドが動作するコアを指定
 - [Platform]Affinity.hから設定

```
class [REDACTED] : public FGenericPlatformAffinity
{
public:
    // by default let all threads run whenever they can be scheduled. Game specific measurements.
    static const CORE_API uint64 GetMainGameMask()
    {
        // Keep the game thread on module 0 to avoid cache penalties
        return MAKEAFFINITYMASK(0,1,2,3);
    }

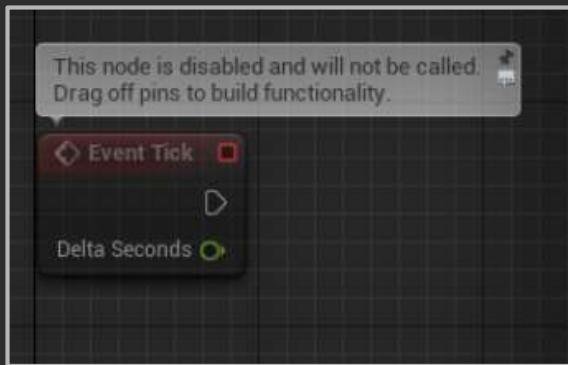
    static const CORE_API uint64 GetRenderingThreadMask()
    {
        // Keep the render thread on module 1 to avoid cache penalties
        return MAKEAFFINITYMASK(4,5);
    }

    static const CORE_API uint64 GetRTHeartBeatMask()
    {
        return SCE_KERNEL_CPUMASK_6CPU_ALL;
        //return MAKEAFFINITYMASK(5);
    }
}
```

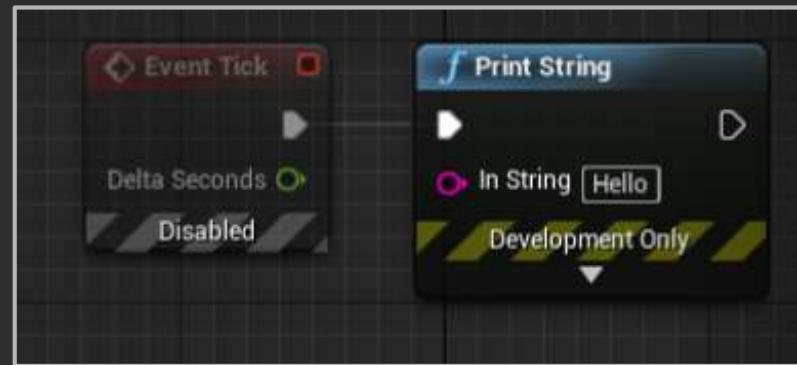
Tickを適切にコントロールする

出来るだけ非アクティブなActorはTickを止めるのが望ましい

- 次のケースではどう処理されるでしょう？



未接続のTick Node



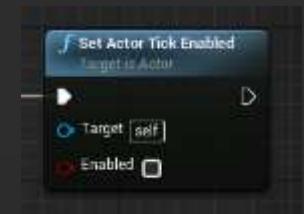
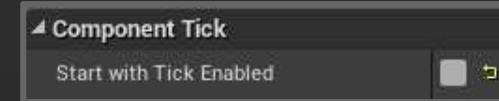
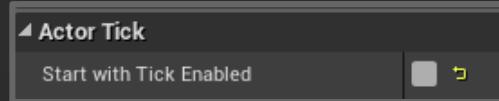
非ActiveなTick Node

Tickのコントロール

BlueprintのTickは実行されないが基底のTickが実行



そのTick処理が必要ない場合は根底から停止



コンソールコマンド (パフォーマンス関連)

dumpticks

- Level上に存在するActorやComponentのTick設定を出力
- 不要に動いているTick設定やTickGroupを見直す際に有効

```
===== Tick Functions (All) =====
FStartPhysicsTickFunction, Enabled, ActualStartTickGroup: TG_StartPhysics, Prerequisites: 0
FEndPhysicsTickFunction, Enabled, ActualStartTickGroup: TG_EndPhysics, Prerequisites: 1
    World /Game/ThirdPersonBP/Maps/UEDPIE_0_ThirdPersonExampleMap.ThirdPersonExampleMap, FStartPhysicsTickFunction
FStartAsyncSimulationFunction, Enabled, ActualStartTickGroup: TG_EndPhysics, Prerequisites: 1
    World /Game/ThirdPersonBP/Maps/UEDPIE_0_ThirdPersonExampleMap.ThirdPersonExampleMap, FEndPhysicsTickFunction
LineBatchComponent /Engine/Transient.LineBatchComponent_13[TickComponent], Enabled, ActualStartTickGroup: TG_DuringPhysics, Prerequisites: 0
LineBatchComponent /Engine/Transient.LineBatchComponent_14[TickComponent], Enabled, ActualStartTickGroup: TG_DuringPhysics, Prerequisites: 0
LineBatchComponent /Engine/Transient.LineBatchComponent_15[TickComponent], Enabled, ActualStartTickGroup: TG_DuringPhysics, Prerequisites: 0
```



コンソールコマンド (パフォーマンス関連)

stat dumphitches

- 指定時間を超えるヒッチを出力
- ヒッチを見ることで過負荷のあたりをつける際に有効

```
100.843ms ( 1) - Game TaskGraph Tasks - STAT_TaskGraph_GameTasks - STATGROUP_Threading - STATCAT_A  
69.164ms (4007) - FTickFunctionTask - STATGROUP_TaskGraphTasks - STATCAT_Advanced  
36.943ms (1000) - CharacterMovementComponent/Game/ThirdPersonBP/Maps/UEDPIE_0_ThirdPersonExample  
36.691ms (1000) - Char Movement Total - STAT_CharacterMovement - STATGROUP_Game - STATCAT_Adva  
36.610ms (1000) - Char Tick - STAT_CharacterMovementTick - STATGROUP_Character - STATCAT_Adv  
34.930ms (1000) - Char NonSimulated Time - STAT_CharacterMovementNonSimulated - STATGROUP_  
34.287ms (1000) - Char PerformMovement - STAT_CharacterMovementPerformMovement - STATGRO  
17.746ms (1000) - Char PhysNavWalking - STAT_CharPhysNavWalking - STATGROUP_Character  
15.193ms (1000) - MoveComponent(Primitive) Time - STAT_MoveComponentTime - STATGROUP  
11.219ms (1000) - GeomSweepMultiple - STAT_Collision_GeomSweepMultiple - STATGROUP  
10.176ms (1000) - SceneQueryTotal - STAT_Collision_SceneQueryTotal - STATGROUP_C  
10.094ms (1000) - GeomSweepMultiple - STAT_Collision_GeomSweepMultiple - STATG  
9.983ms (1000) - MoveComponent - STATGROUP_CollisionTags - STATCAT_Advanced  
9.927ms (1000) - Self  
0.056ms (1000) - OtherChildren
```

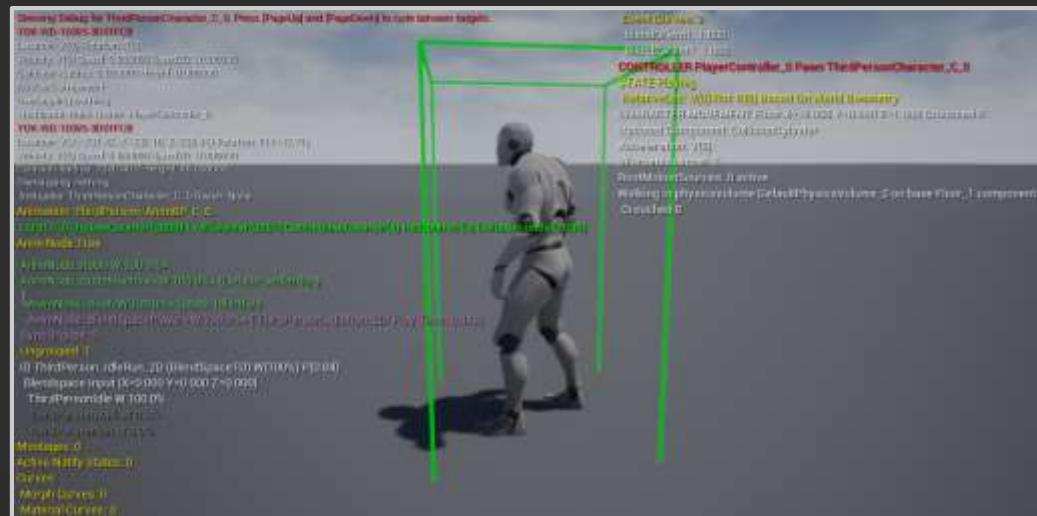


コンソールコマンド（デバッグ関連）

ShowDebug XXX

- 標準で搭載されている各機能を視覚化

ShowDebugToggleSubCategory	FULLBLENDSPACEDISPLAY	Wi
ShowDebugToggleSubCategory	CURVES	Wi
ShowDebugToggleSubCategory	3DBONES	Wi
ShowDebugToggleSubCategory		Ca
ShowDebugSelectedInfo		
ShowDebugForReticleTargetToggle		<DesiredClass>
ShowDebugForReticleTargetToggle		De
ShowDebug WEAPON		To
ShowDebug Reset		To
ShowDebug PHYSICS		To
ShowDebug None	(X=0.000 Y=0.000 Z=0.000)	To
ShowDebug NET	(X=0.000 Y=0.000 Z=0.000)	To
ShowDebug INPUT	(0.00%)	To
ShowDebug FORCEFEEDBACK		To
ShowDebug COLLISION		To
ShowDebug CAMERA		To
ShowDebug BONES		To
ShowDebug ANIMATION		To
ShowDebug AI		To
ShowDebug		De
showdebug		



コンソールコマンド (デバッグ関連)

ShowDebug Input

- キー入力情報の視覚化
- Input Componentのスタック情報

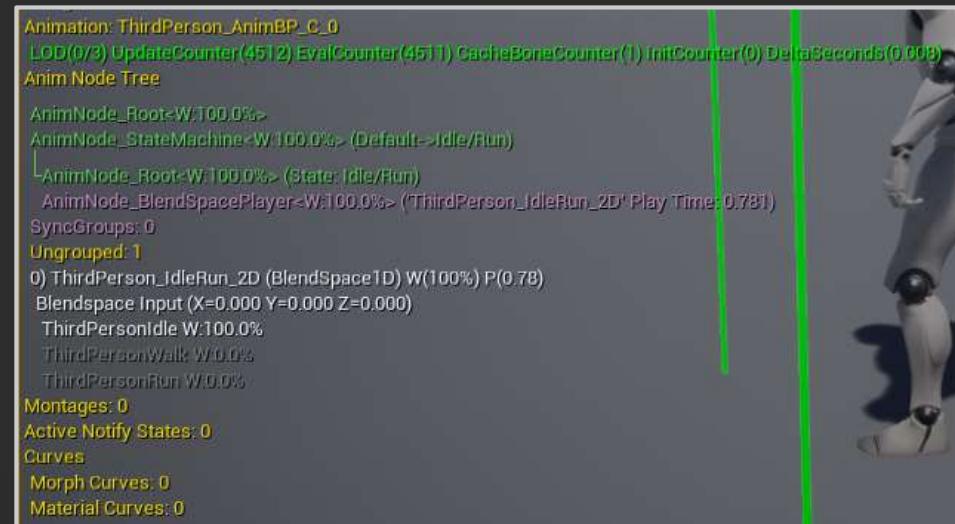
```
<<< INPUT STACK >>>
PlayerController_0 GameplayDebug_Input
PlayerController_0.PC_InputComponent0
ThirdPersonExampleMap_C_4.InputComponent0
ThirdPersonCharacter_C_0.PawnInputComponent0
INPUT PlayerInput_0
Left Mouse Button: 0.00 (raw 0.00)
Mouse X: 0.00 (raw 0.00)
Mouse Y: 0.00 (raw 0.00)
W: 0.00 (raw 0.00)
S: 0.00 (raw 0.00)
Up: 0.00 (raw 0.00)
A: 0.00 (raw 0.00)
Space Bar: 0.00 (raw 0.00)
Right Mouse Button: 0.00 (raw 0.00)
D: 0.00 (raw 0.00)
Mouse Wheel Up: 0.00 (raw 0.00)
Mouse Wheel Axis: 0.00 (raw 0.00)
Mouse Wheel Down: 0.00 (raw 0.00)
MouseSampleRate: 129.38
MouseX ZeroTime: 4.00, Smoothed: 0.00
MouseY ZeroTime: 4.06, Smoothed: 0.00
```



コンソールコマンド (デバッグ関連)

ShowDebug Animation

- アニメーション情報の視覚化



コンソールコマンド (デバッグ関連)

p.VisualizeMovement

- 加速度や移動モードを表示



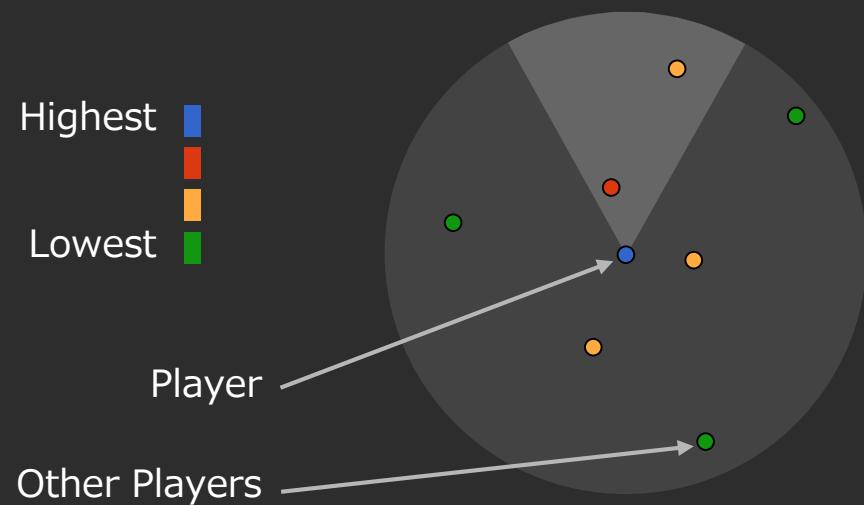
#UE4CEDEC



Significance Manager

オブジェクト同士に優先度を設けて機能のON/OFFを制御

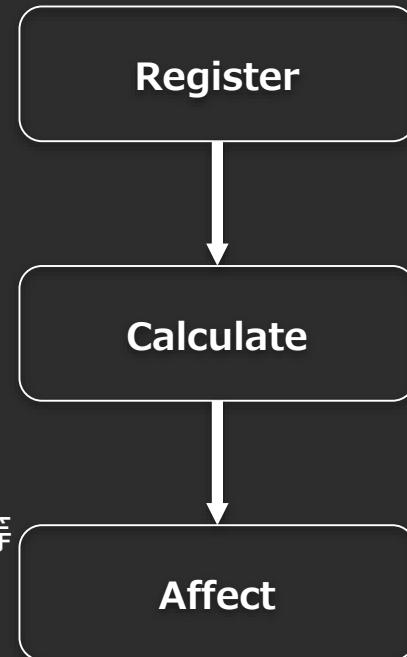
- 優先度の決め方
 - 距離、スクリーンサイズなど
- 優先度の適用対象
 - AIPawn、Particleなど
- 実装
 - 各プロジェクトで独自の実装



Significance Manager

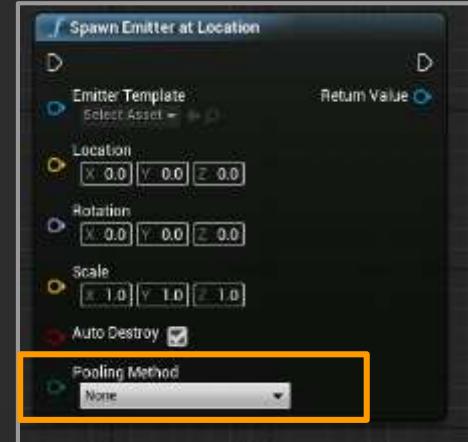
主に3ステップを実行して重要度を管理

- 対象Objectの登録
 - 例) AActor::BeginPlay()
- 重要度の計算
 - 例) SignificanceManager::Update()等
- 重要度に応じた効果適用
 - 例) SignificanceManager::UpdateSignificance()等



Particle Pooling

- Spawn Emitter関数へのPoolingオプションを追加 (4.20から)
 - 最初にSpawnしたParticle System Componentを再利用
 - SpawnしたObjectはActivate/Deactivateで再利用



まとめ



#UE4CEDEC



まとめ

- コンテンツにあわせて機能や設定のON/OFF
- 必要な機能も動的に切り替えて無駄のない処理
- 出来る限りシンプルに設計



まとめ

初期から機能を把握して設計
プロファイリングを行いボトルネックを探して最適化
開発の早期からお気軽にご相談下さい



告知



The graphic features a large, stylized white 'U' logo inside a circle, with the word 'FEST' stacked vertically next to it, and 'EAST '18' at the bottom. To the right are a QR code and the Epic Games logo. Below the main logo, the text reads 'UNREAL FEST EAST 2018' and '10/14 SUN'. At the bottom left, it says 'パシフィコ横浜 会議センター 3F' and provides the website 'http://unrealengine.jp/'. A note at the bottom right indicates '事前登録制(無料)'.

UNREAL FEST EAST 2018
10/14 SUN

パシフィコ横浜 会議センター 3F
<http://unrealengine.jp/>

事前登録制(無料)

