

Unreal Engineで リアルタイムシミュレーションを 実現するまで



株式会社理経

石川大樹、原田和樹

自己紹介

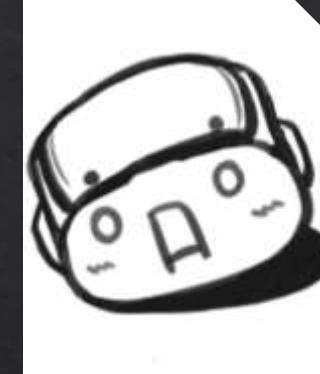
株式会社 理 経
新規事業推進室長
石川 大樹



2015年理経入社。
入社後「UE4を活用したVR開発」
に特化した新規事業部を立ち上げる。

自動車、不動産など幅広いエンター
プライズ向けVRの企画立案・開発
を担う。

株式会社 理 経
VRエンジニア
原田 和樹



2019年にVRエンジニアとして入社。
Unreal Engineと他シミュレータや
デバイスの連携をメインで担当し、
自動運転車両向け学習用環境の構築で
は中心的な役割を担う。

実績



株式会社SUBARU 様
自動運転車両開発向けシミュレーション空間



日立オートモティブシステムズ株式会社 様
自動運転技術の可視化にVRを活用

実績



消防士訓練用VR開発

東京大学バーチャルリアリティ教育研究センター

・東京理科大学・横浜市消防局と共同研究



EPIC MegaGrants 受賞



防災訓練用に羽田空港ターミナルを製作。



西日本豪雨を再現し、広島県に納入。



コンテンツ海外展開
促進事業



Oculus
ISV Program

本日のテーマ

リアルタイムシミュレーションを実現する
要素技術と事例について

<対象領域>

ADAS

HILS

AI学習

モデリング/マテリアル
Unreal Engine実装
プロファイリング

用語解説

ADAS	ドライバー補助機能の総称（自動ブレーキ、レンキープ等）
HILS	複数のシミュレーションツールを連携させ開発する手法
Carsim	世界的に有名な車両挙動シミュレーションツール

リアルタイムシミュレーションとは？

- ・シミュレーションの一部に、実コントローラやプラントを用いるシミュレーション。
- ・自動車の場合) 認識⇒判断⇒制御 の一連の流れをループさせることでシミュレーションを行う。



リアルタイムシミュレーションとは？

- ・シミュレーションの一部に、実コントローラやプラントを用いるシミュレーション。
- ・自動車の場合) 認識⇒判断⇒制御 の一連の流れをループさせることでシミュレーションを行う。



従来のシミュレーション手法

実空間



UnrealEngine4

仮想空間



リアルタイムシミュレーションの取り組み例

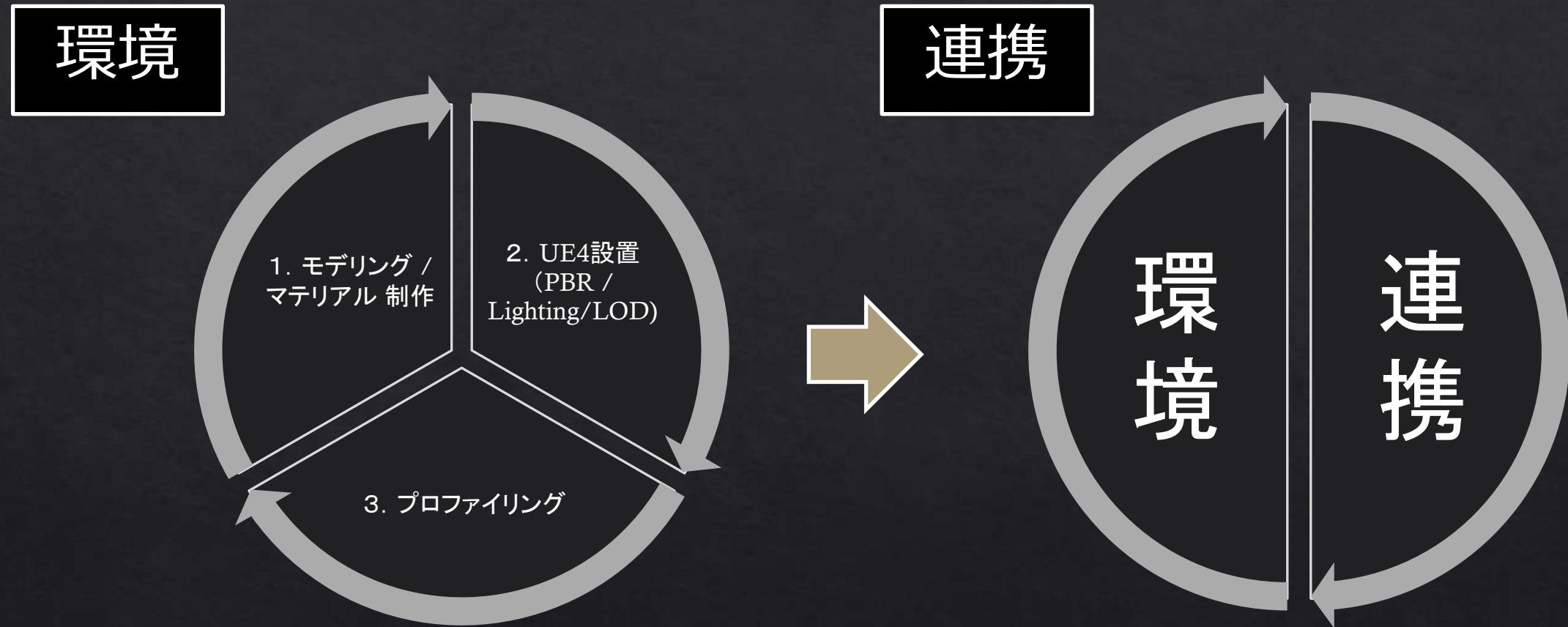
<構成例>



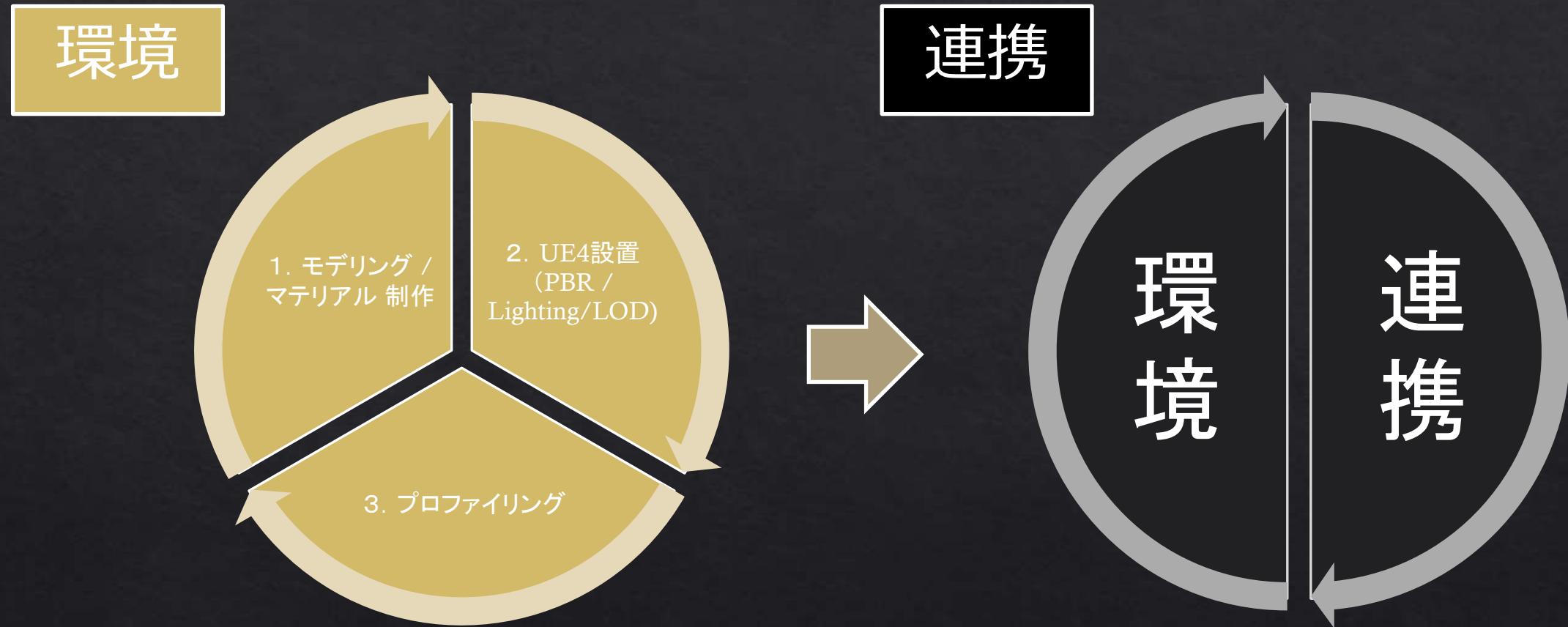


After the Rain

シミュレーションを構成する要素



シミュレーションを構成する要素



環境作り

- モデリング / マテリアル 制作
 - PBRマテリアル
 - ローポリモデル
- UE4最適化
 - LOD
 - 最適化調整
- プロファイリング

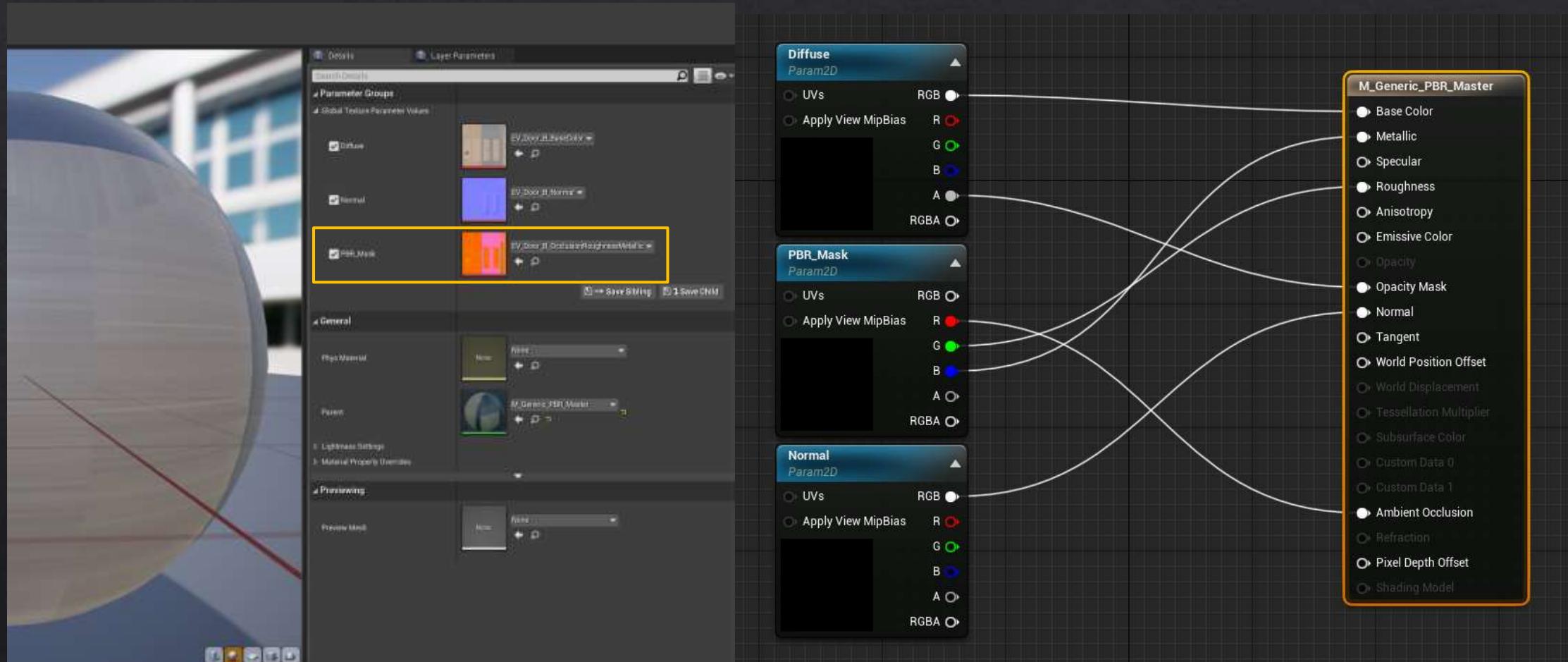
環境作り

- モデリング / マテリアル 制作
 - PBRマテリアル
 - ローポリモデル
- UE4最適化
 - LOD
 - 最適化調整
- プロファイリング

PBRマテリアル

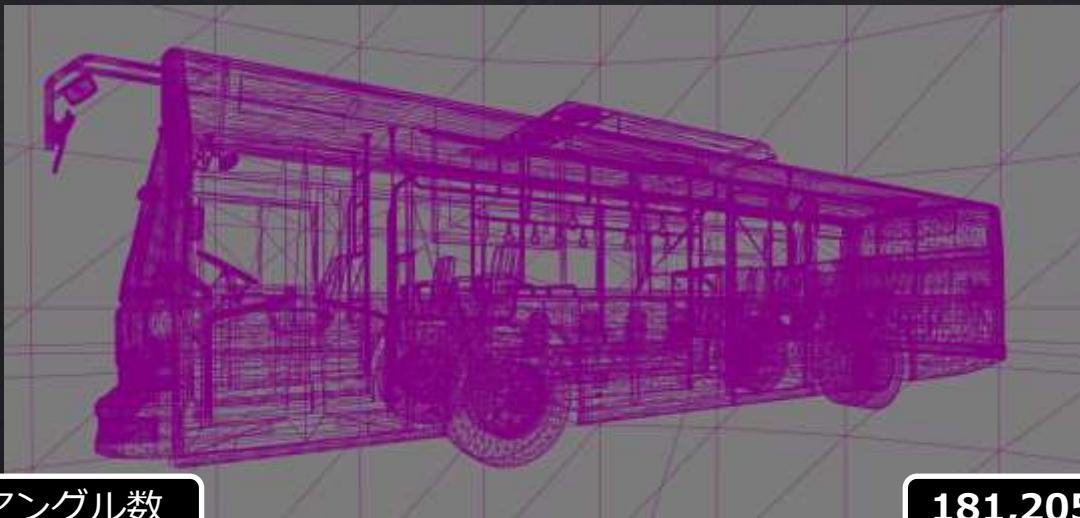
- PBRマテリアル(Physically Based Rendering)
 - 物理的に正しいマテリアル
- 2つの値(粗度:Roughness, 金属度:Metalness)で物質の質感を調整できる
- 一つのマテリアルに対し、PBRのテクスチャを入れ替えるだけ→軽量化
- PBRのテクスチャ→Substance Painterで作る
- 異なるエンジンでも同じライティングであれば使える

PBRマテリアルの設定(UE4)



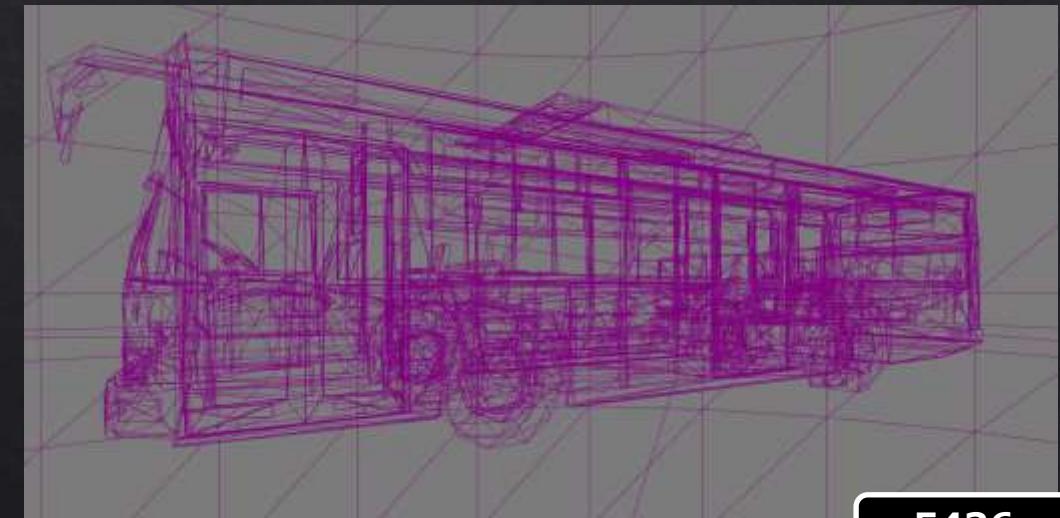
ロー・ポリモデル化

- ハイポリを作つてノーマルマップにベイク
→ロー・ポリに張り付ける



トライアングル数

181,205



5436

環境作り

- モデリング / マテリアル 制作
 - PBRマテリアル
 - ローポリモデル
- UE4最適化
 - LOD
 - 最適化調整
- プロファイリング

LODとは

- LOD - Level of Detail

→カメラからの距離に応じてメッシュの情報量を減らし、
GPUの計算負荷を軽減する手法

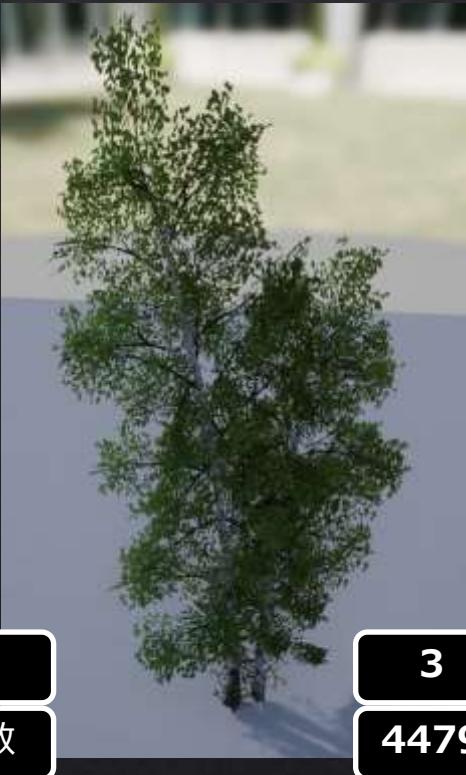
→3dsMax等3DCGソフトで編集。UE4でも調整可能

LODとは

- LODで増減するメッシュの情報例
- ポリゴン数
- マテリアル数
- マテリアルの複雑さ
- テクスチャ解像度

実際に見てみると

- カメラが近い場合



マテリアル数

3

トライアングル数

4479

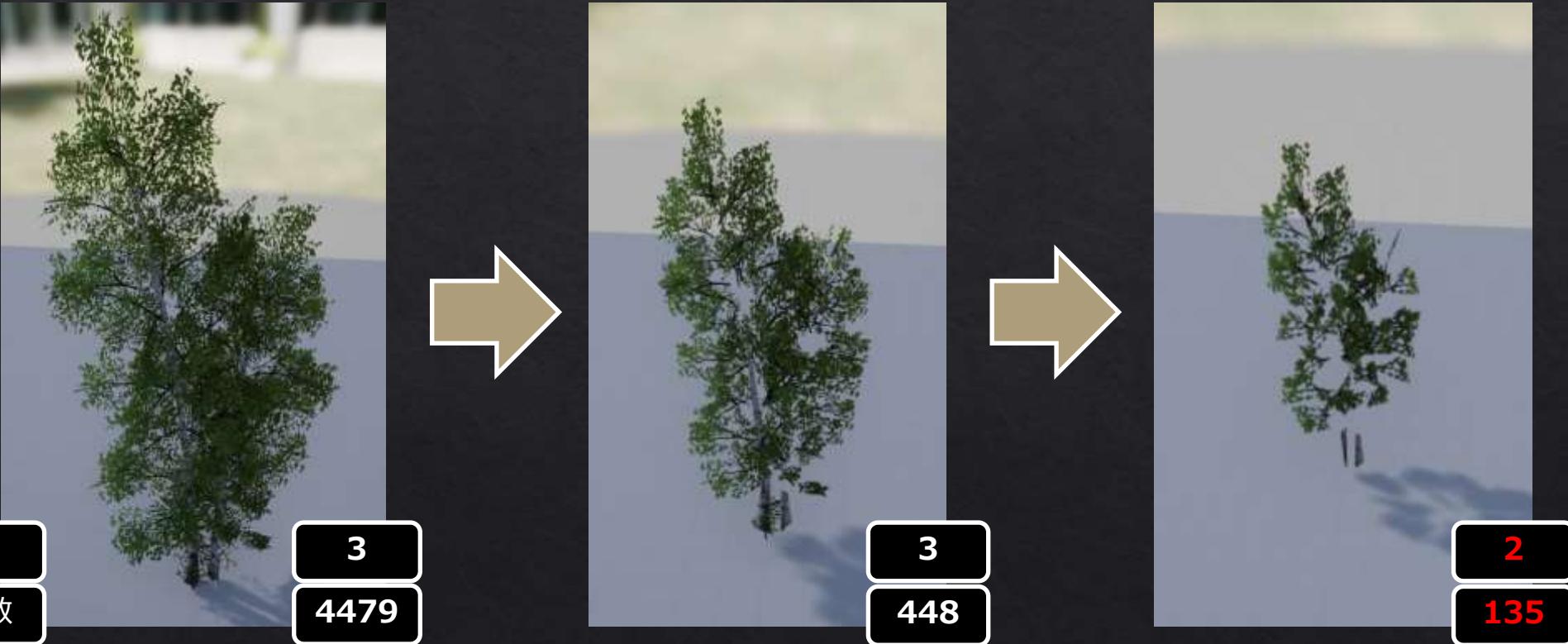
実際に見てみると

- カメラを少し離した場合（画面占有率50%以下）



実際に見てみると

- カメラをさらに離した場合（画面占有率30%以下）



LODの使い方

- リアルタイム性を重視するプロジェクトでは
主に 遠・中・近 の3段階に設定
- 複雑なメッシュの場合、形が不自然になるため
都度3DCGソフトでカスタムLODを作成

UE4最適化

レベルデザイン面

- テクスチャは2K以内(できれば1K)におさめる
 - マテリアルの複雑さは限界あり、透明or半透明のマテリアルは使わない(Translucent, Additive等)
複雑なシェーダーは使わない
- パーティクルは必要なところだけに使う
- レベルストリーミングとライトシナリオ(ベイク情報の保存)は積極的に使う

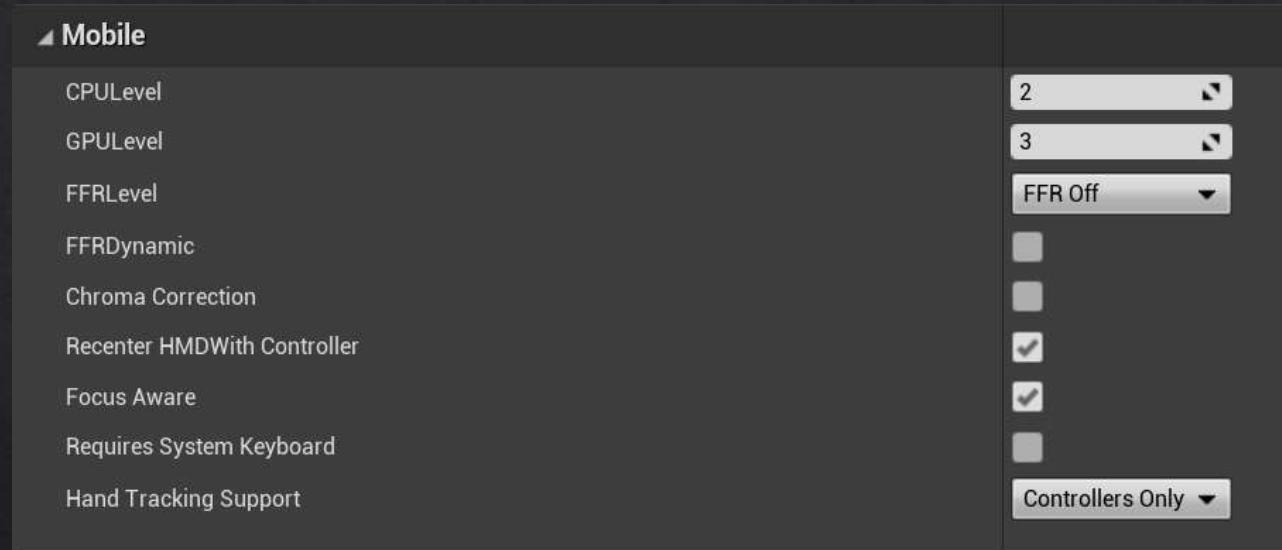
UE4最適化

内部処理面

- Pixel Density で解像度調整
- Tick(毎フレーム処理)はなるべく使わない
- 処理コストが高いもの(全アクタ取得、スポーン等)も使わない
- FFRLevel, FFRDynamicで画面外側の解像度を落としていく

UE4最適化

- FFRLevel, FFRDynamicで画面外側の解像度を落としていく



環境作り

- モデリング / マテリアル 制作
 - PBRマテリアル
 - ローポリモデル
- UE4最適化
 - LOD
 - 最適化調整
- プロファイリング

プロファイリング作業

レベルデザイン面

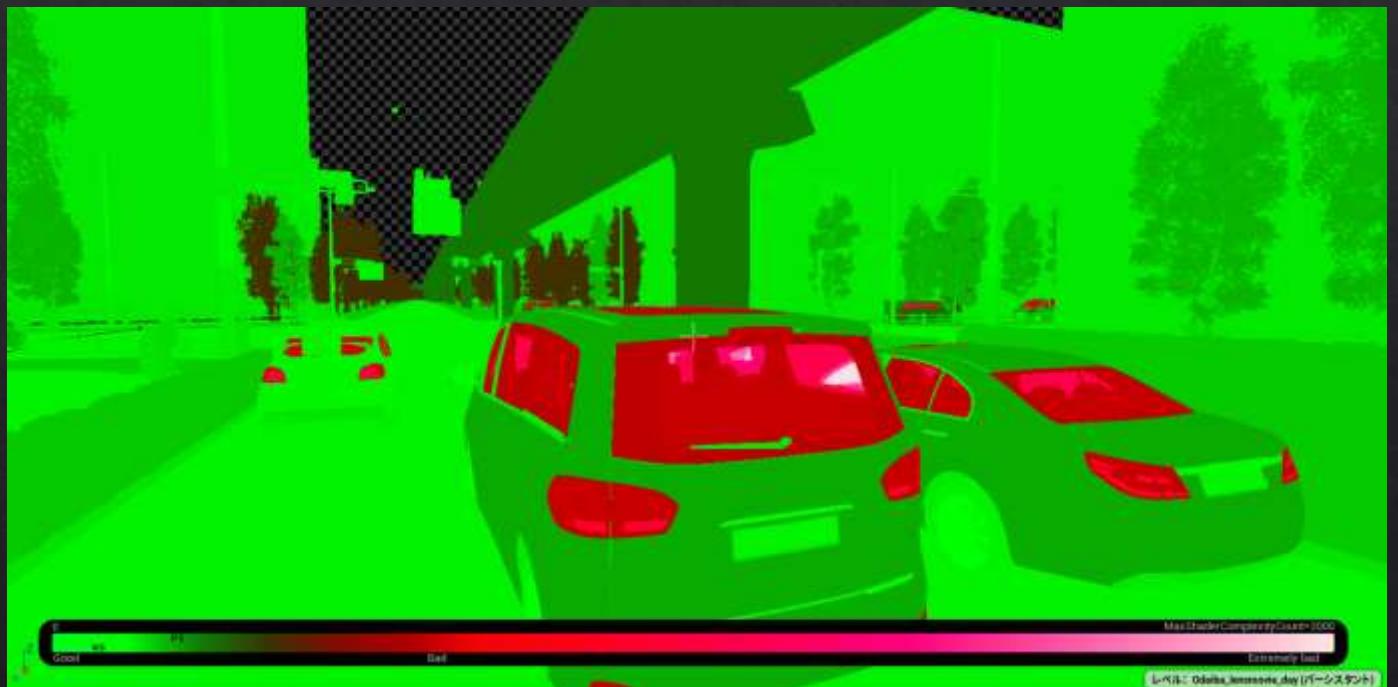
- ビューモードを使ったシーン全体の負荷チェック
 - Shader Complexityでオーバードローの確認
 - Quad OverdrawでLOD必要箇所の確認

内部処理面

- プロファイリングツールを使った内部処理チェック
 - CPU/GPUプロファイリング

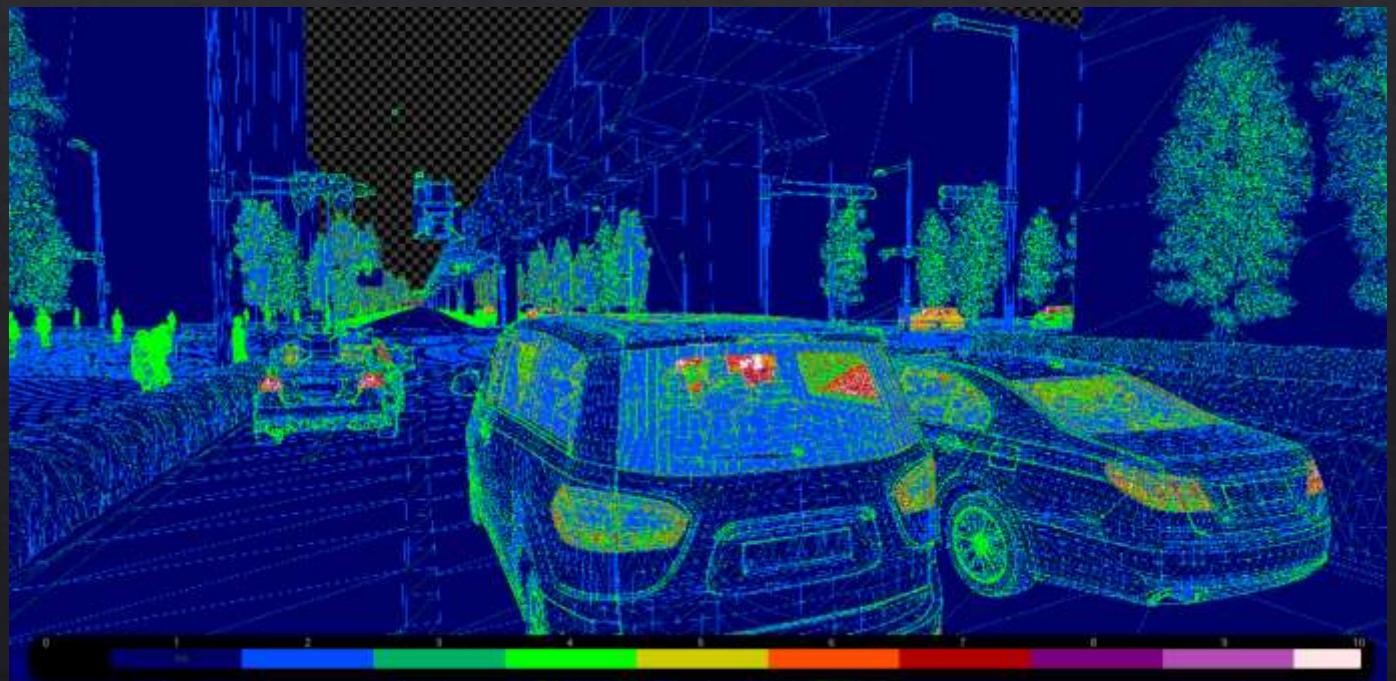
ビューモードでの負荷チェック

- Shader Complexity (シェーダー複雑度)
- ピクセル描画が重なる
オーバードローを視覚化
(主にマテリアル)
- 半透明マテリアルやパーティクルが
重なると多く発生
- 赤～白が多い場合は要調整



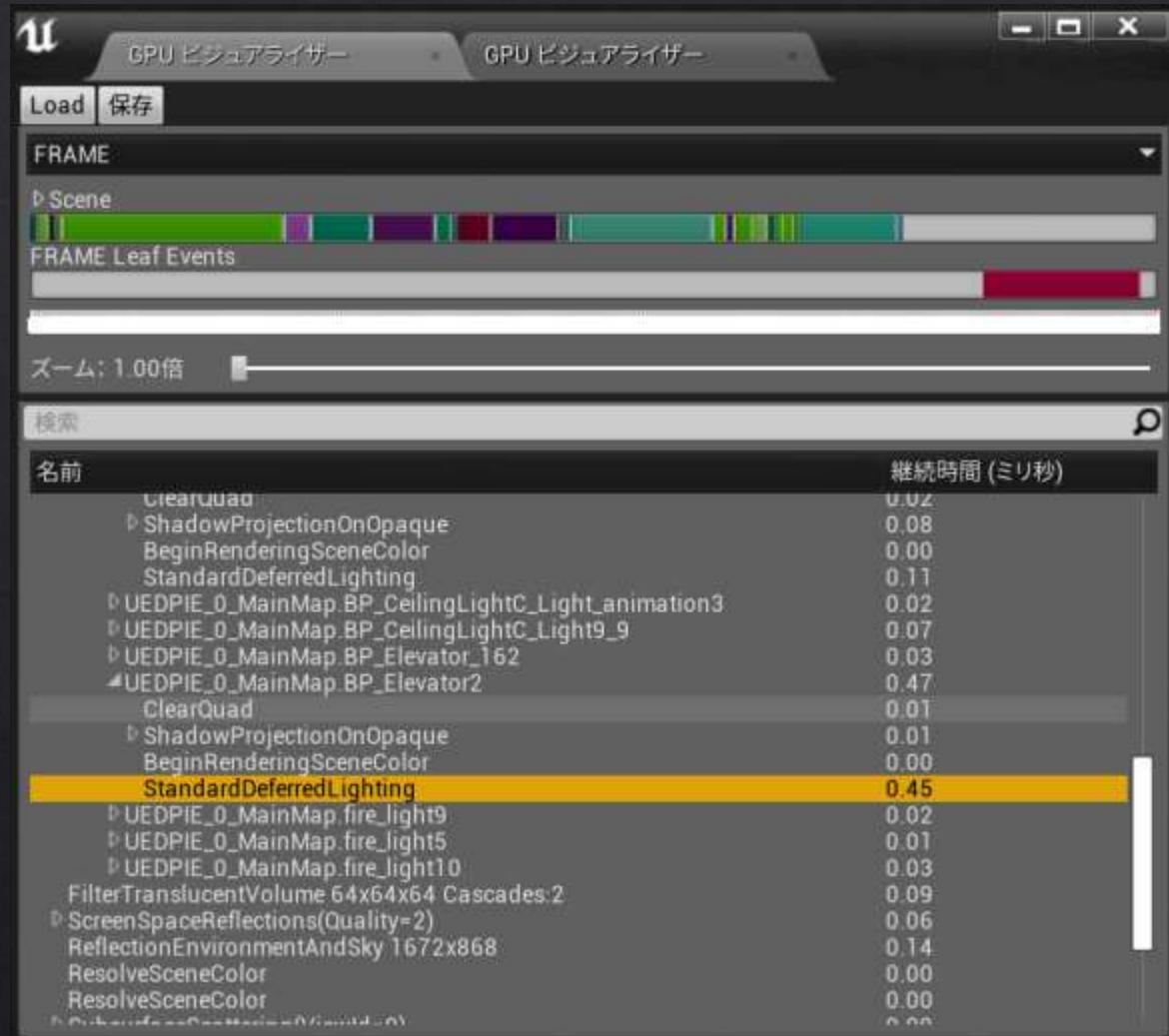
ビューモードでの負荷チェック

- Quad Overdraw (クアッドの概要)
- ポリゴンの描画が重なる部分を
視覚化
(主にメッシュ)
- メッシュが複雑なほど緑～白に
- 単体のポリゴン数が多い場合は
LODやローポリ化でポリゴン数を
削っていく



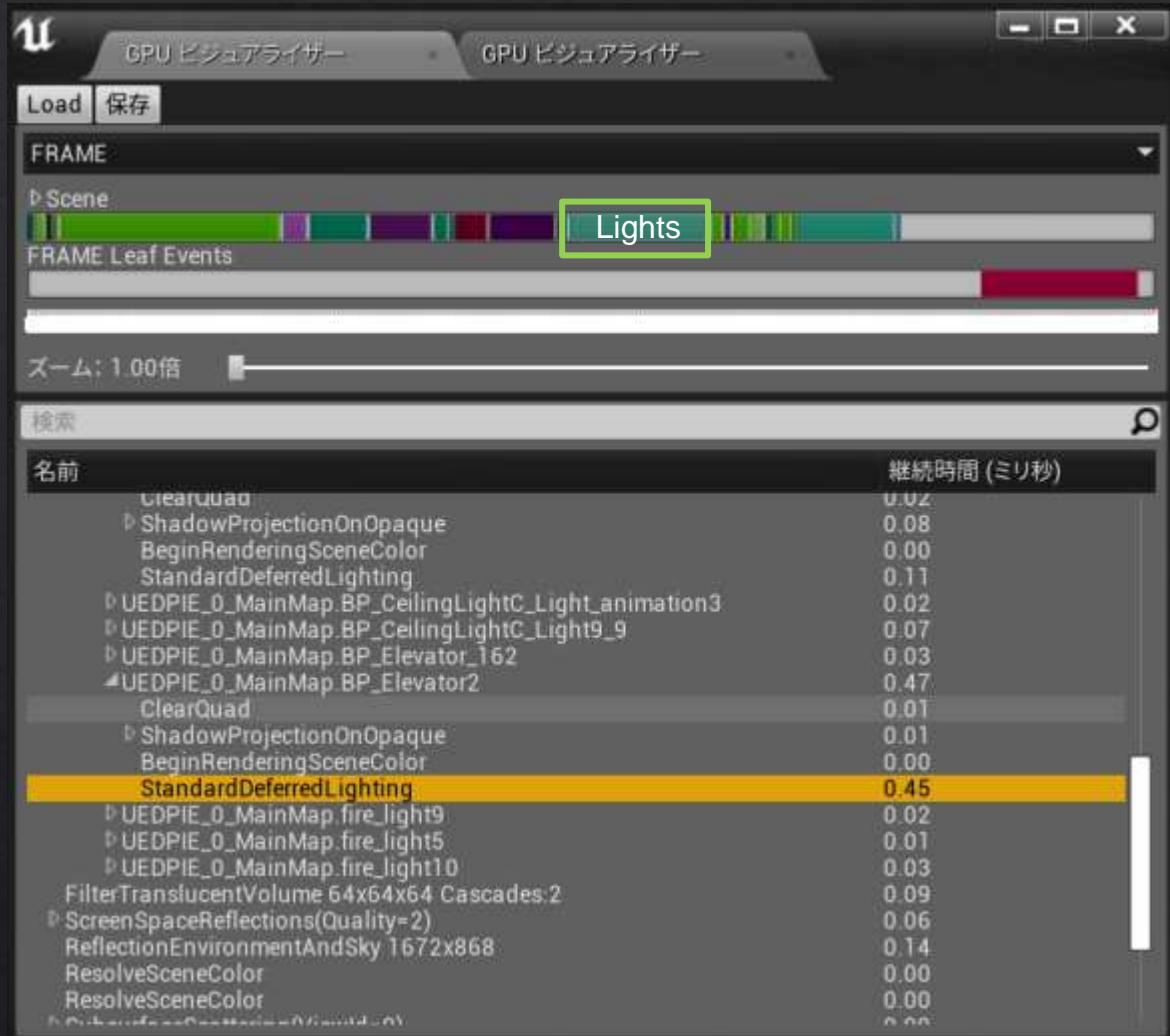
プロファイリングツール

- GPUプロファイリング
- 特定の瞬間のGPU使用の内訳を確認
- シーン内でのコストの比率を確認
- ライティングやポストプロセス等修正が必要な箇所を特定



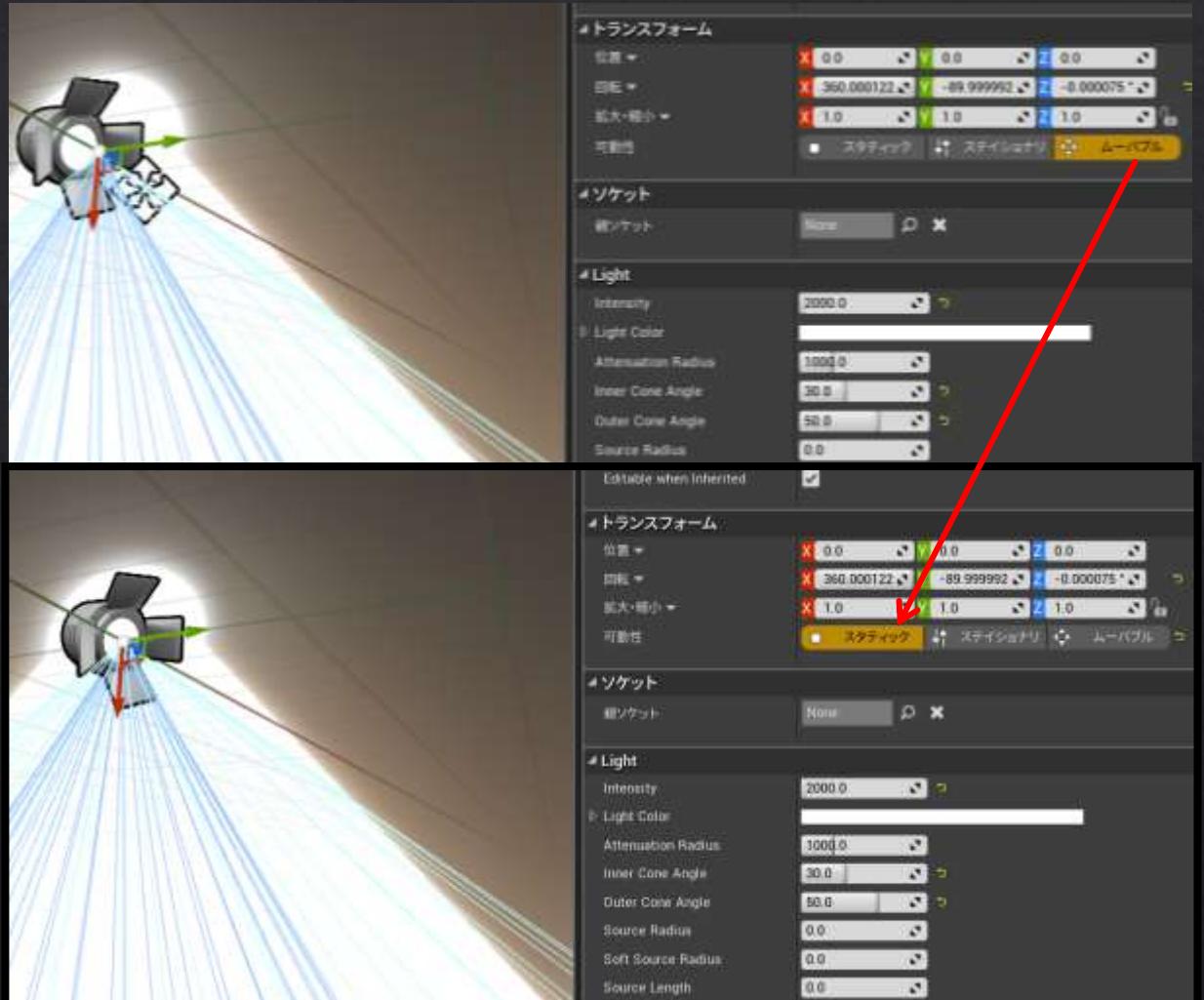
プロファイリングツール

- 例
- ライティングを示す  が大きい
- 中でも継続時間が多い
Standard Deferred Lightingに着目
(スクリーンに映っているライト)



プロファイリングツール

- 例
- ライティング部分の可動性をチェック
- ムーバブル→スタティックに変更し、
ベイクする



プロファイリングツール

調整前

名前	継続時間(ミリ秒)
ClearQuad	0.02
ShadowProjectionOnOpaque	0.08
BeginRenderingSceneColor	0.00
StandardDeferredLighting	0.11
UEDEPIE_0_MainMap_BP_CeilingLightC_Light_animation3	0.02
UEDEPIE_0_MainMap_BP_CeilingLightC_Light9_9	0.02
UEDEPIE_0_MainMap_BP_Elevator_162	0.02
UEDEPIE_0_MainMap_BP_Elevator2	0.02
ClearQuad	0.00
ShadowProjectionOnOpaque	0.00
BeginRenderingSceneColor	0.00
StandardDeferredLighting	0.45
UEDEPIE_0_MainMap_BP_Elevator2	0.02
UEDEPIE_0_MainMap_BP_Elevator2	0.01
UEDEPIE_0_MainMap_BP_Elevator2	0.03
FilterTranslucentVolume 64x64x64 Cascades:2	0.09
ScreenSpaceReflections(Quality=2)	0.06
ReflectionEnvironmentAndSky 1672x868	0.14
ResolveSceneColor:	0.00
ResolveSceneColor:	0.00
PostProcessEffectsAndFinalOutput:2	0.00

0.45

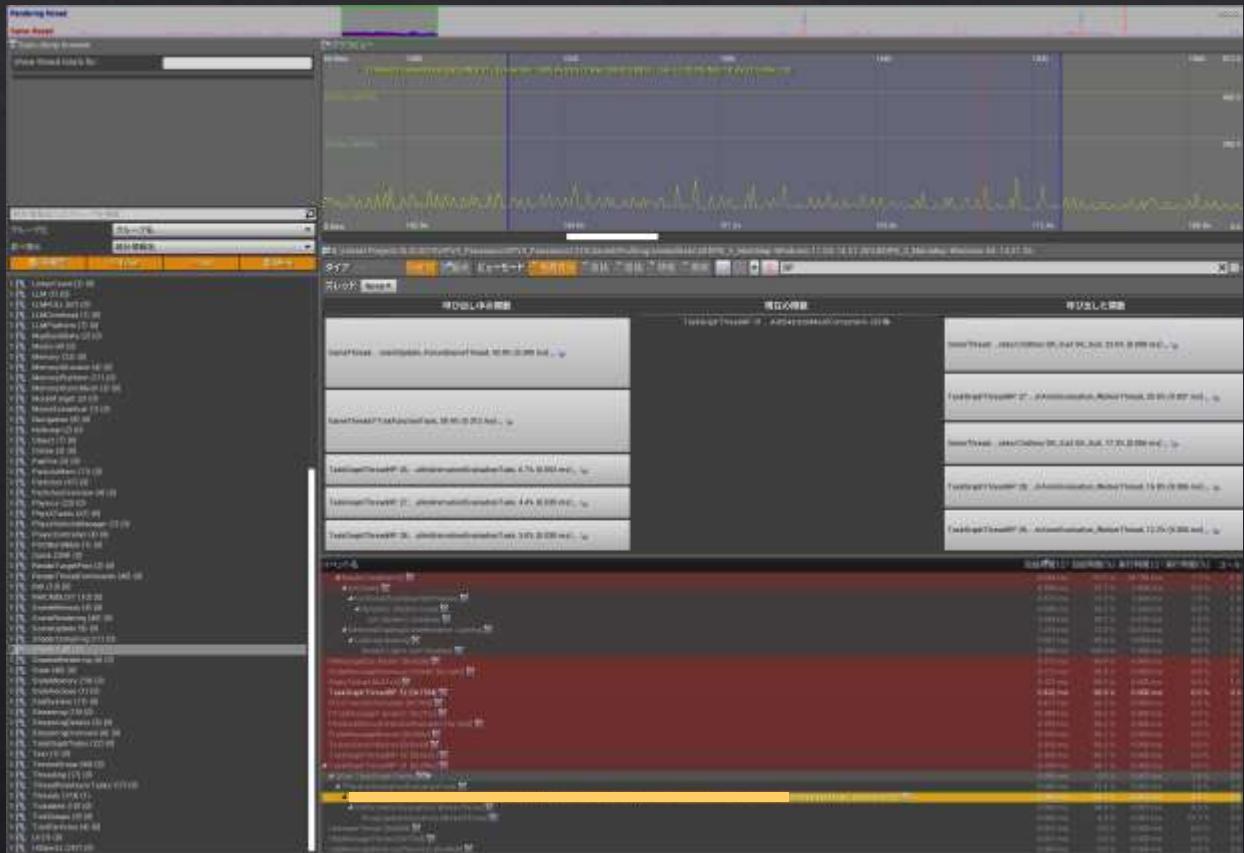
調整後

名前	継続時間(ミリ秒)
DirectLighting	0.19
NonShadowedLights	0.00
IndirectLighting	0.00
ShadowedLights	0.19
UEDEPIE_0_MainMap.fire_light4	0.01
UEDEPIE_0_MainMap.fire_light8	0.01
UEDEPIE_0_MainMap.fire_light6	0.01
UEDEPIE_0_MainMap_BP_Elevator2	0.01
ClearQuad	0.01
ShadowProjectionOnOpaque	0.00
BeginRenderingSceneColor	0.00
StandardDeferredLighting	0.01
UEDEPIE_0_MainMap_BP_Elevator2	0.03
UEDEPIE_0_MainMap_BP_Elevator2	0.02
UEDEPIE_0_MainMap_BP_Elevator2	0.04
UEDEPIE_0_MainMap_BP_Elevator2	0.02
FilterTranslucentVolume 64x64x64 Cascades:2	0.09
ScreenSpaceReflections(Quality=2)	0.07
ReflectionEnvironmentAndSky 1672x868	0.14
PostProcessEffectsAndFinalOutput:2	0.00

0.01

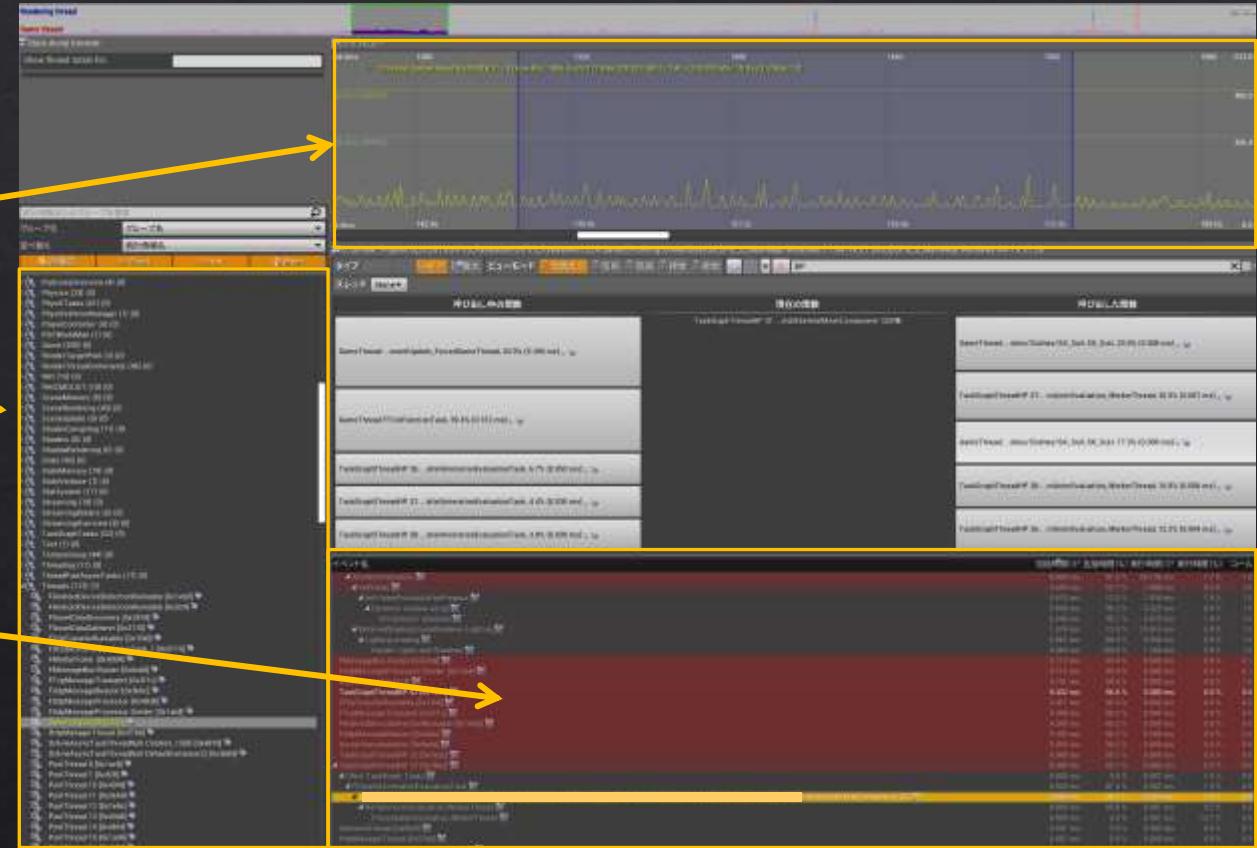
プロファイリングツール

- CPUプロファイリング
- 特定の間隔内でCPUの負荷をダンプ
→ヒッチが起きた部分を確認できる
- Tick自体の処理も可視化できるため
コストの大きいクラスや処理が見える
→最適化箇所の特定



プロファイリングツール

- プロファイラの見方
- ダンプした記録から特定した間隔
- 繼続時間グラフの表示対象一覧
- イベント一覧

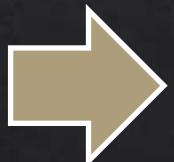


環境構成・プロファイリング後

FPS 19



FPS 90

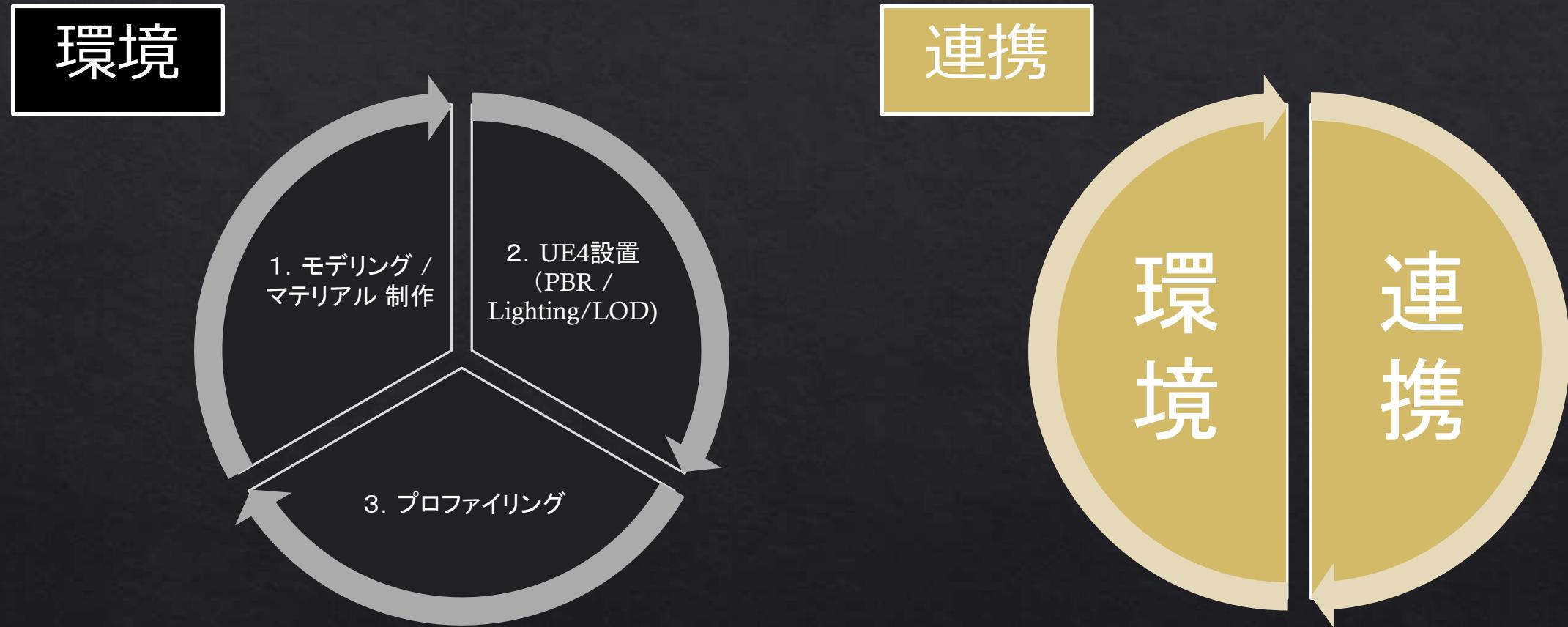


プロファイリングをしないと…

- F P Sが出ないので終盤に焦る！



シミュレーションを構成する要素



シミュレータとの連携

HILS分野

- カメラHILS連携(nDisplay, リアルとVR)
- LiDAR

画像認識分野

- アノテーション(Boxの作成, 他車両等の検出再現)

シミュレータとの連携

HILS分野

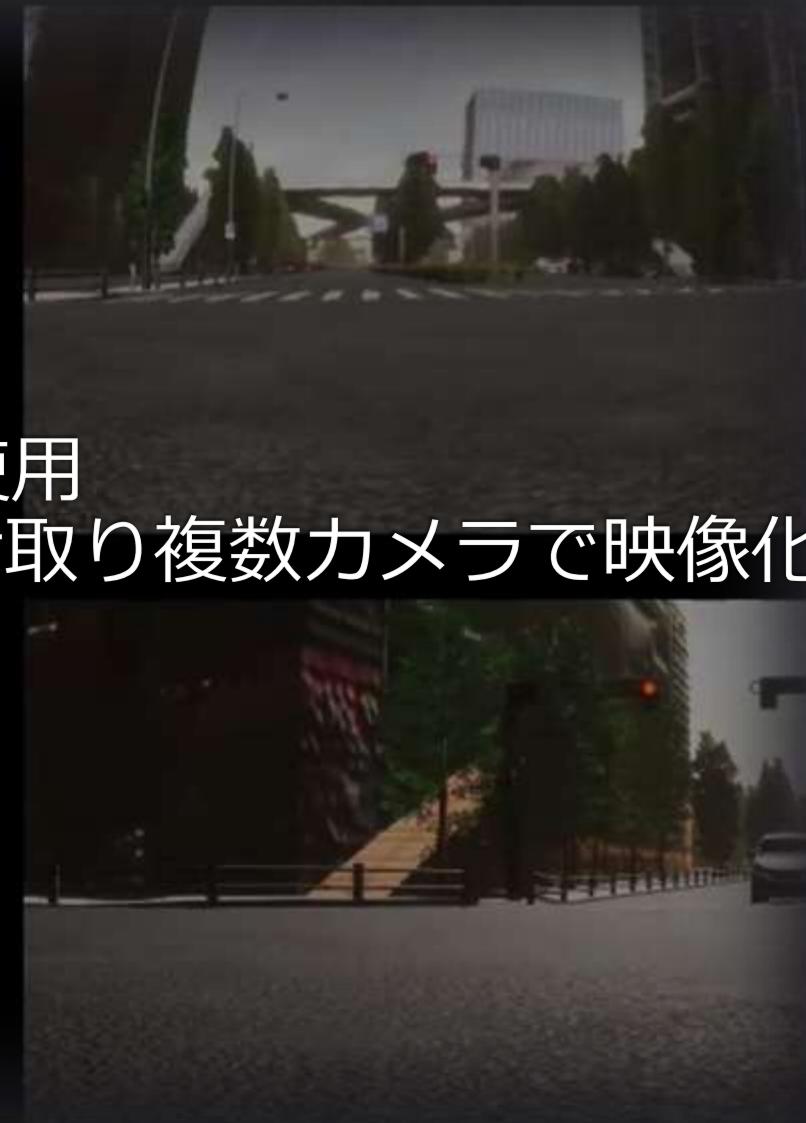
- カメラHILS連携(nDisplay, リアルとVR)
- LiDAR

画像認識分野

- アノテーション(Boxの作成, 他車両等の検出再現)

HILSとの同期

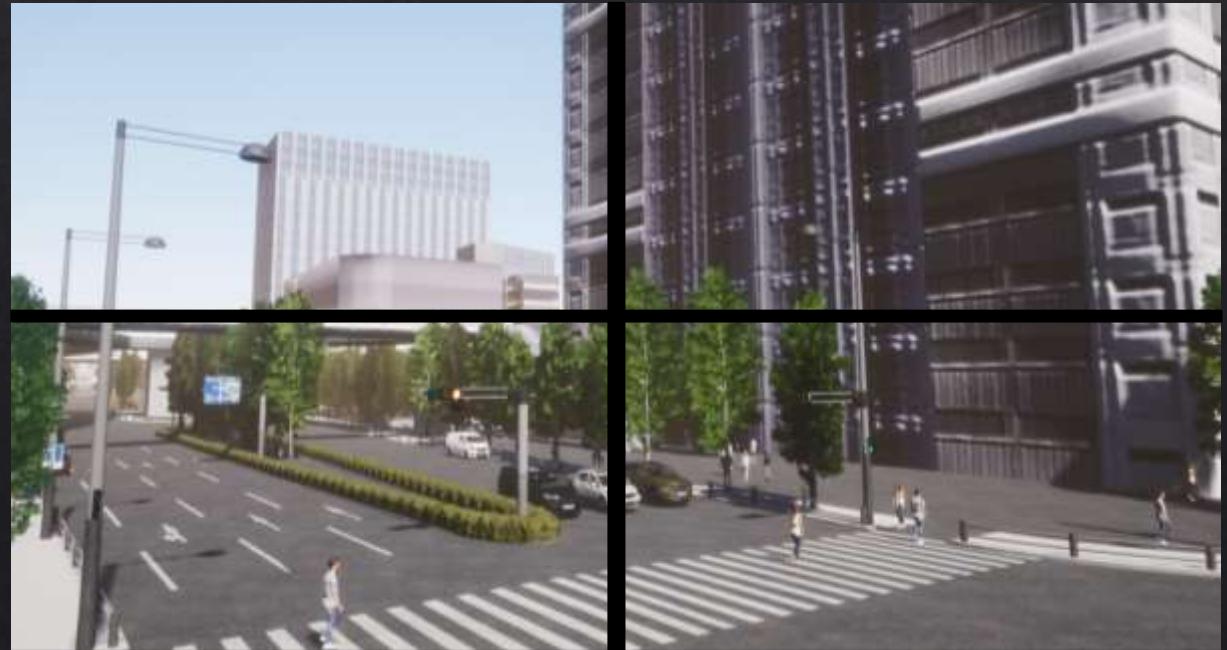
- UE4の機能であるnDisplayを使用
位置情報をリアルタイムに受け取り複数カメラで映像化



HILSとの同期

nDisplayとは：

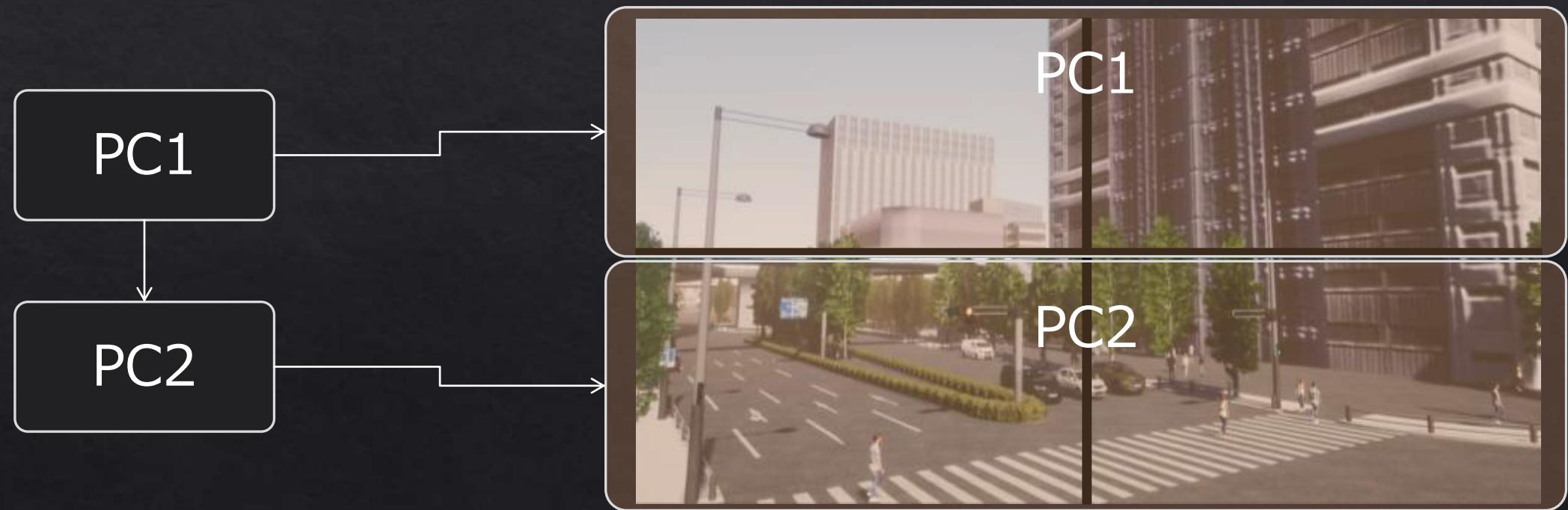
複数のPC/ディスプレイを介して1つのシーンを
同時出力する仕組み



HILSとの同期

nDisplayとは：

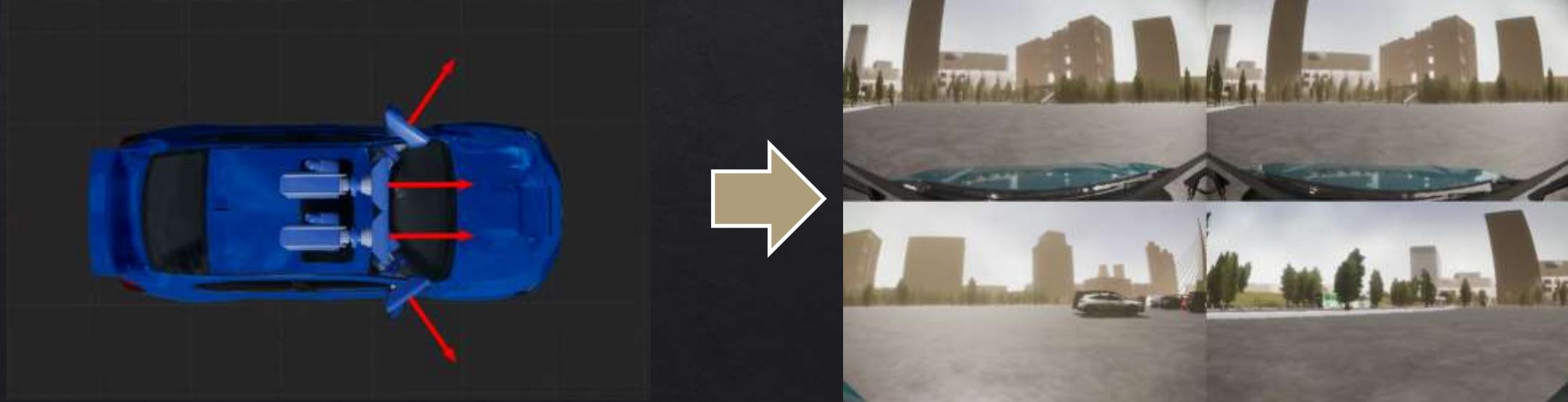
複数のPC/ディスプレイを介して1つのシーンを
同時出力する仕組み



HILSとの同期

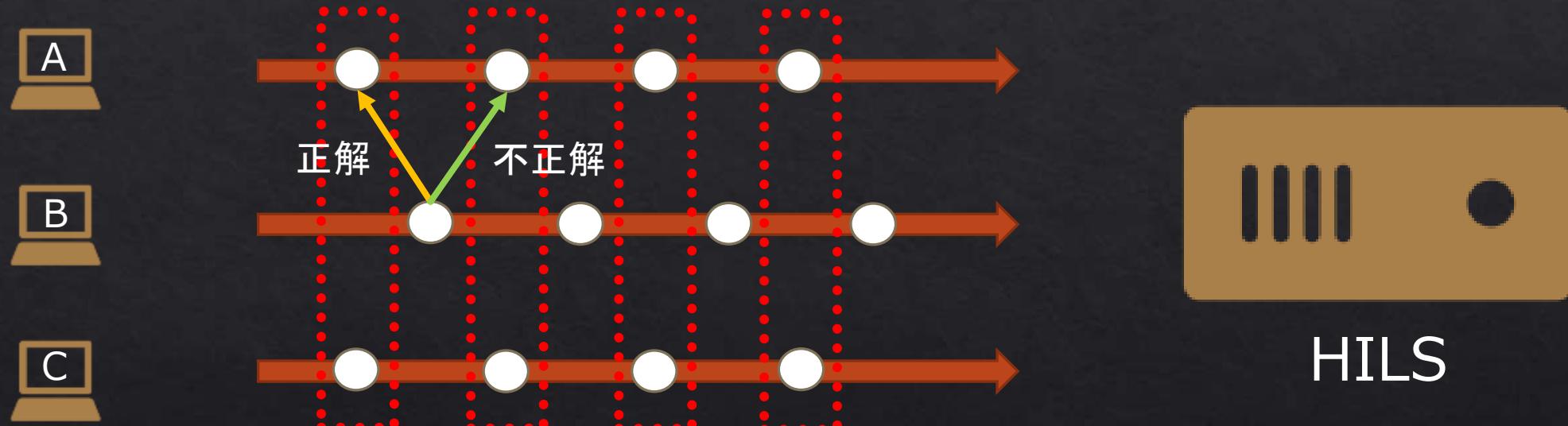
- 導入経緯

車両に搭載された複数のカメラからの映像を
複数ディスプレイに表示してHILSに入力する必要があった



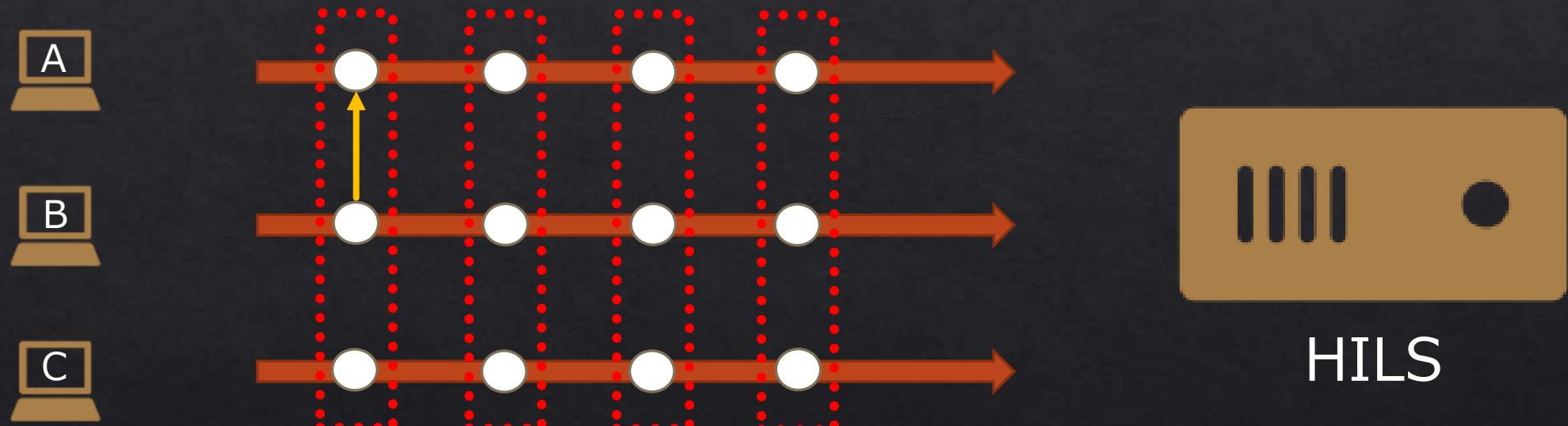
HILSとの同期

- ・ カメラ1台に対してPCが1台必要
- ・ 生成する映像は何もしないとずれる



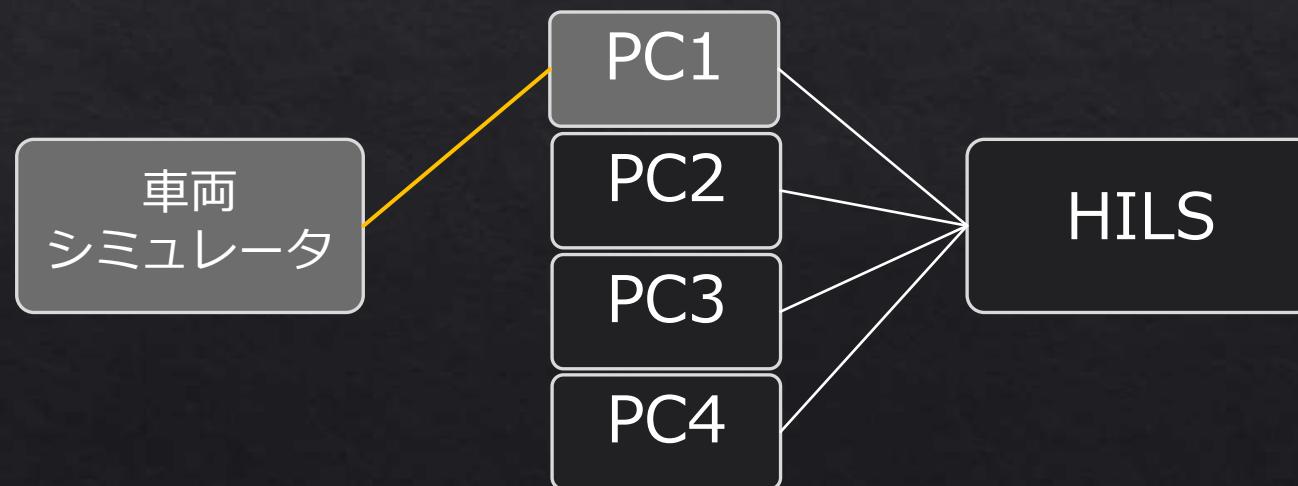
HILSとの同期

- nDisplayを使用するとフレーム同期がとれる



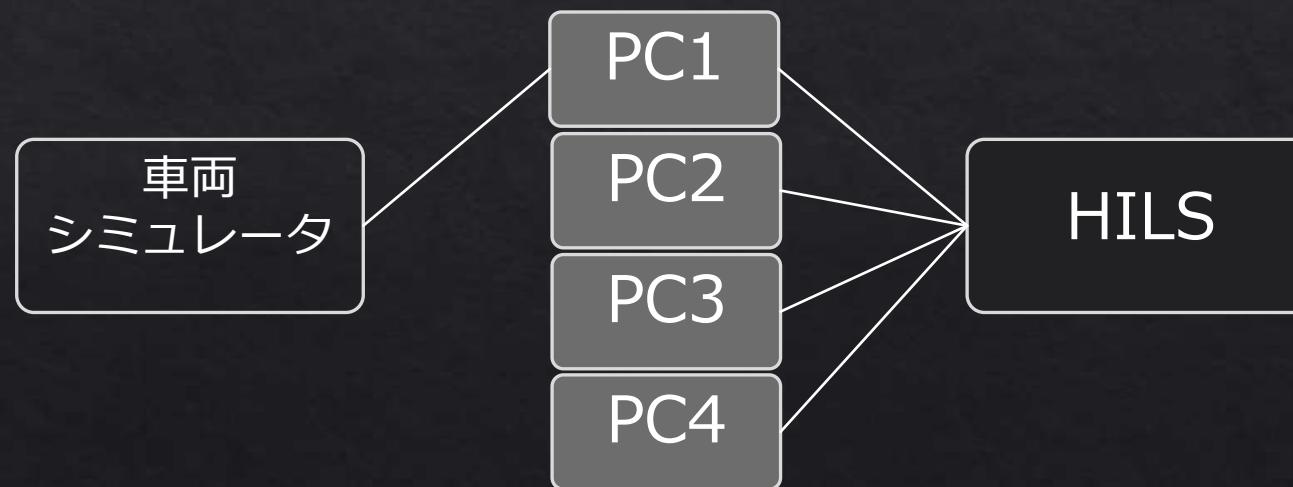
HILSとの同期

- ・位置情報を車両シミュレータからUDP通信で受け取る



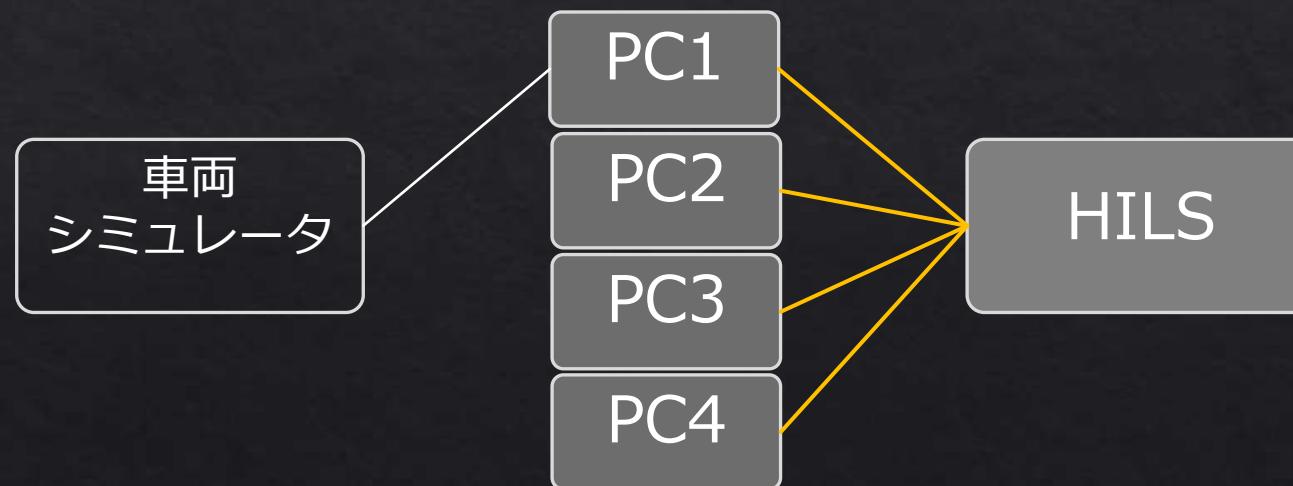
HILSとの同期

- ・位置情報をシミュレータからUDP通信で受け取る
- ・4台のPCで計4画面出力



HILSとの同期

- ・位置情報をシミュレータからUDP通信で受け取る
- ・4台のPCで計4画面出力
- ・HILSが4つの画面を入力し学習



HILSとの同期まとめ

- ディスプレイごとにフレームのズレが無いよう同期設定を行う
- nDisplay自身に処理負荷があるため通常使用よりもFPSが落ちる
→可能な限りコンテンツの負荷を減らす必要

リアルとVR

人間の目 ⇔ AIの目

リアルとVR

人間の目 ⇔ AIの目

認識方法に違い

- 画像認識ではコントラスト比で物体を認識する手法もあるため
実映像のコントラストに寄せる必要がある
→ヒストグラムに着目した

リアルとVR

- AIが認識しない例：信号の色
※人間の認識とAIの認識は異なる
→AIのカメラ画像に合わせマテリアル・テクスチャ調整
- 実画像に寄せるため、LUTを使用して彩度やコントラストを調整
→実映像のヒストグラムに近づける

リアルとVR

- 信号の色調整

加工前



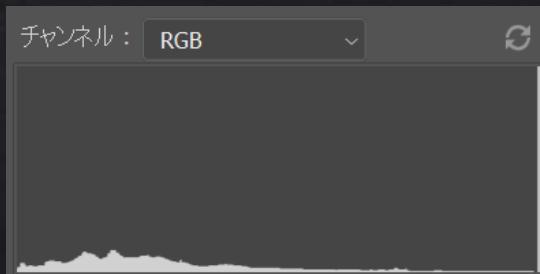
加工後



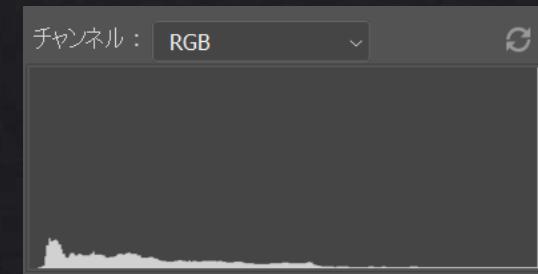
リアルとVR

- ・車両の色調整

加工前



加工後



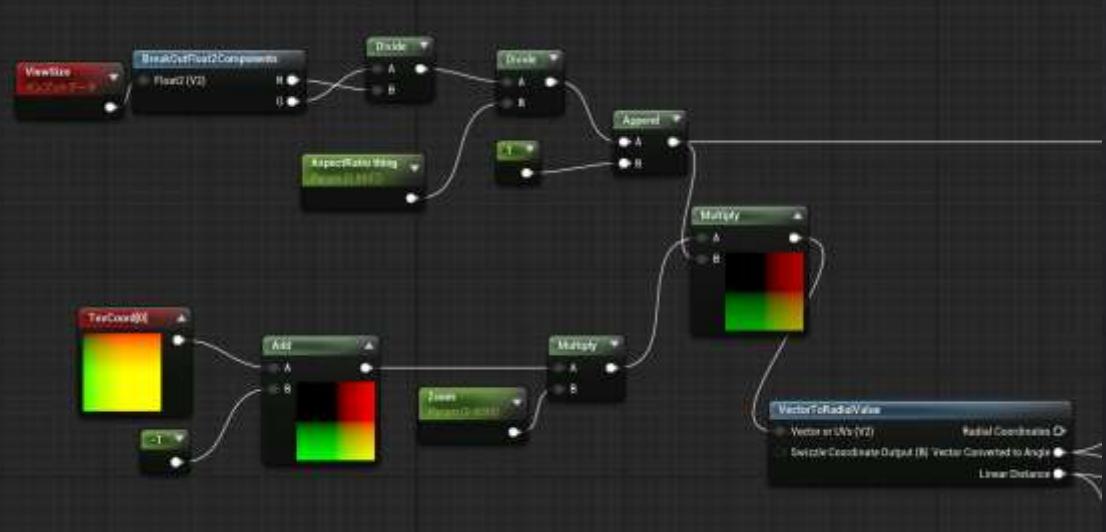
空間を切り取る⇒レンズの再現



レンズ再現

- レンズを実現するために何をしたか
→ PostProcessMaterialを作った

ViewSizeからビュー取得
+
UVを作成
↓
UVを極座標に変換し出力



レンズ再現

個々のレンズ歪みをCineCameraで再現

レンズ歪み低い



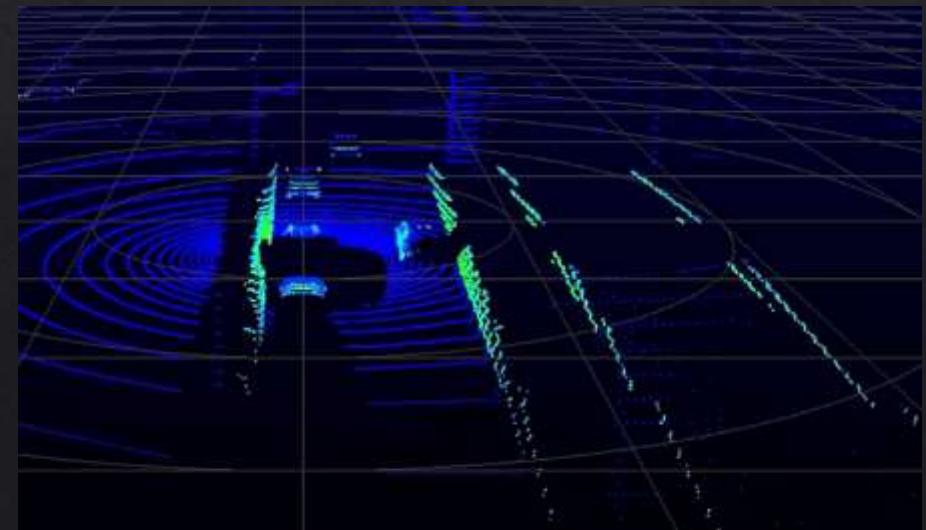
レンズ歪み高い

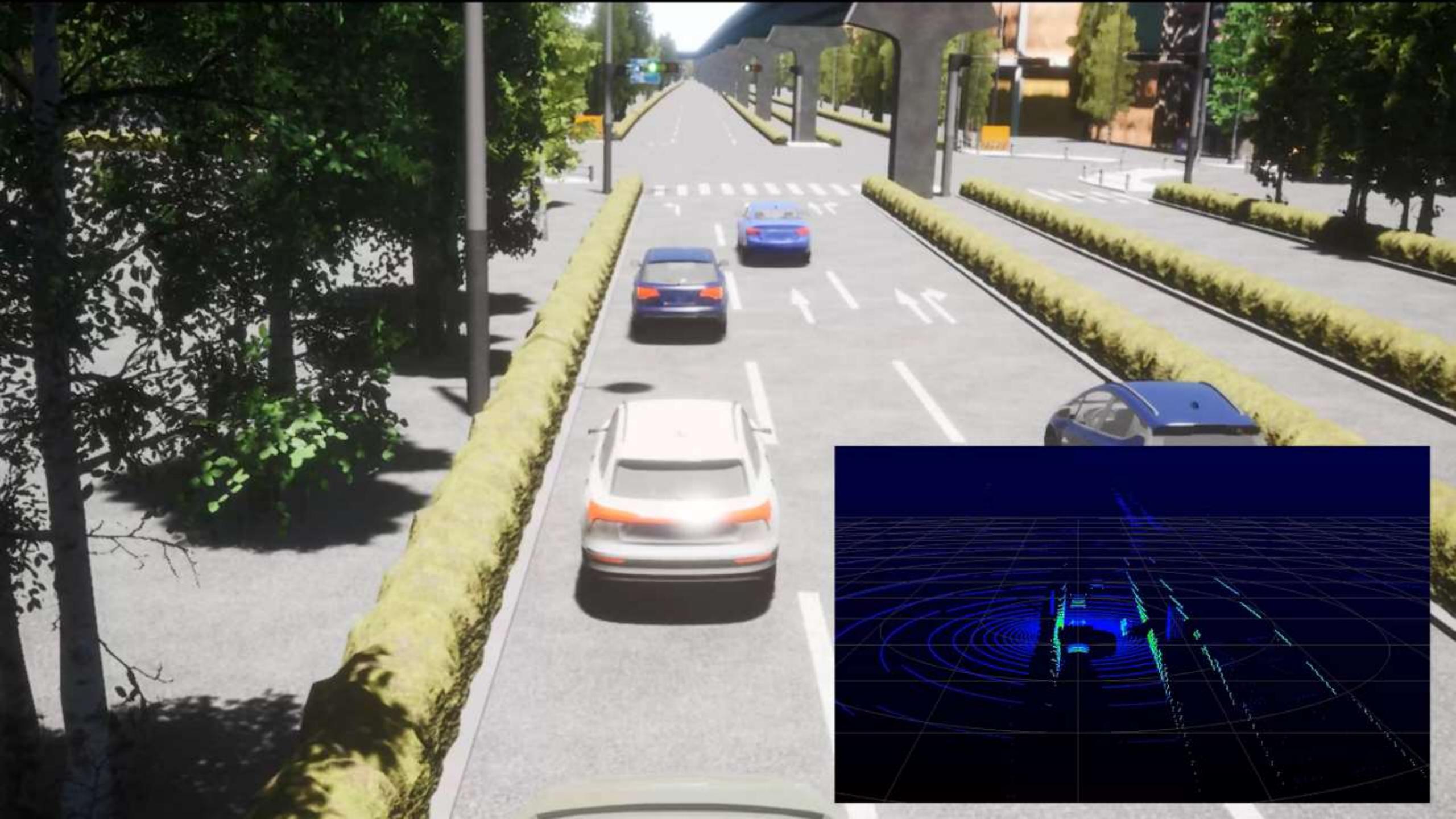


お台場

LiDARシミュレーション

- LiDAR - light detection and ranging(光での検知と測距)
光を照射し反射して戻るまでの時間から点群を生成
- VR空間でLiDARの点群生成をシミュレート
- 各オブジェクトの物質的性質も再現





シミュレータとの連携

HILS分野

- カメラHILS連携(nDisplay, リアルとVR)
- LiDAR

画像認識分野

- アノテーション(Boxの作成, 他車両等の検出再現)

アノテーション

- ・ アノテーションとは

→ AIに画像認識を学習させるために使われるタグ付け
(一般的に認識対象物を囲うバウンディングボックスを使用)

- ・ UE4で学習用映像と教師データを生成

アノテーション

- 認識⇒判断⇒制御 の **認識** に該当
- AIが正しく認識・学習するためには正解データが必要
→正解データとVR空間で学習させる



アノテーション

- アノテーションBoxの作成

→Box用の赤枠テクスチャを用意し、
マテリアルを作成

→人物を囲うようにPlaneを配置、
Tickでカメラに対し常にYawを90度
回転させ平面に表示



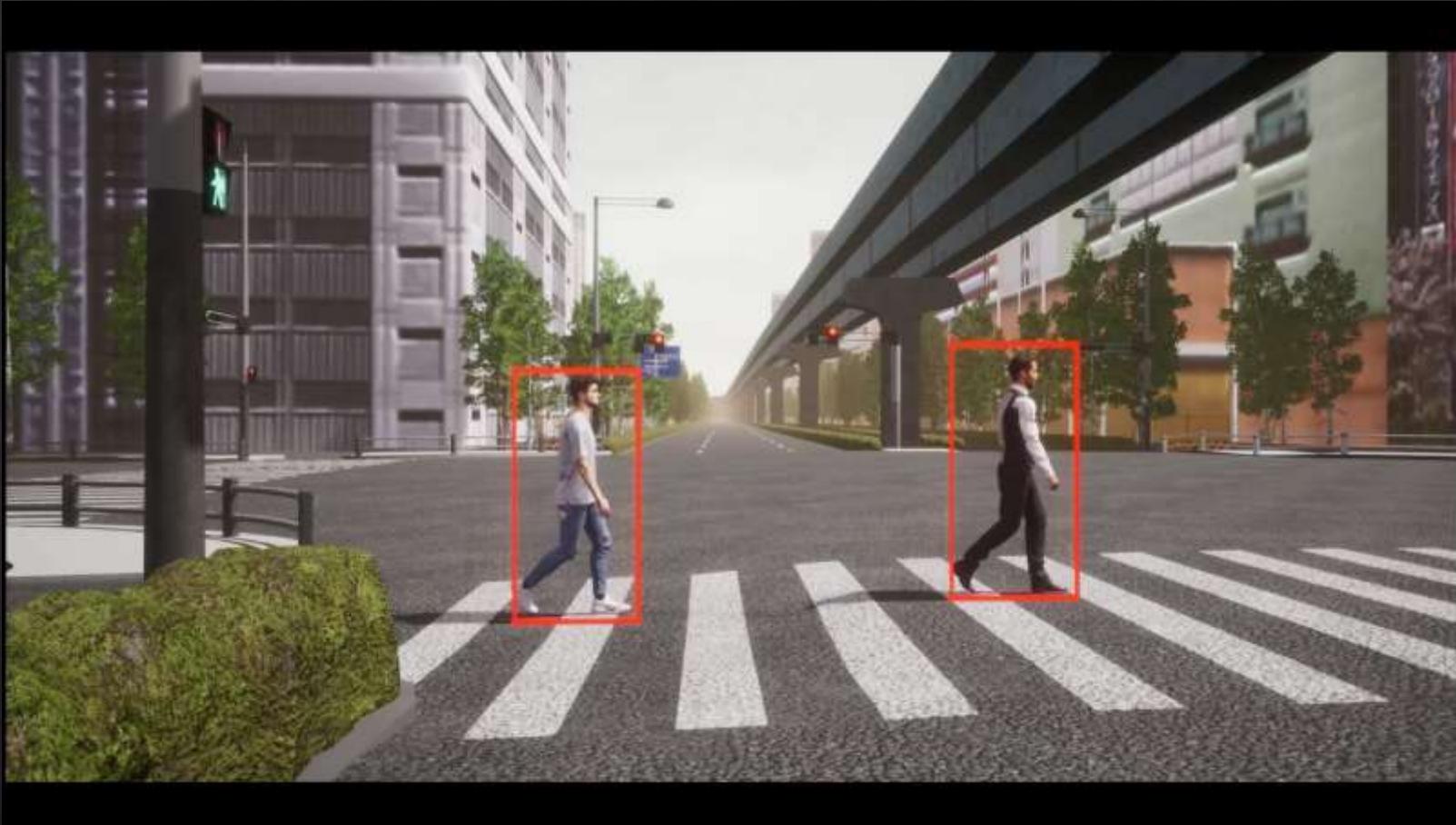
アノテーション

- 他車両の検出再現
 - 自車両からの距離
 - 進行方向
 - スクリーンショット

→これらの情報を
リアルタイム出力



アノテーション

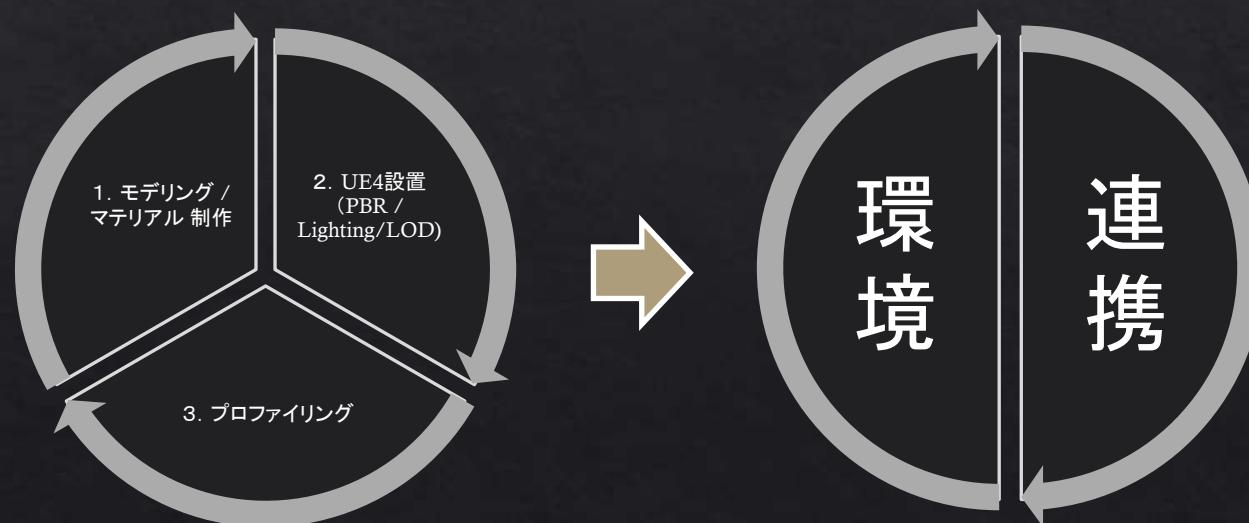


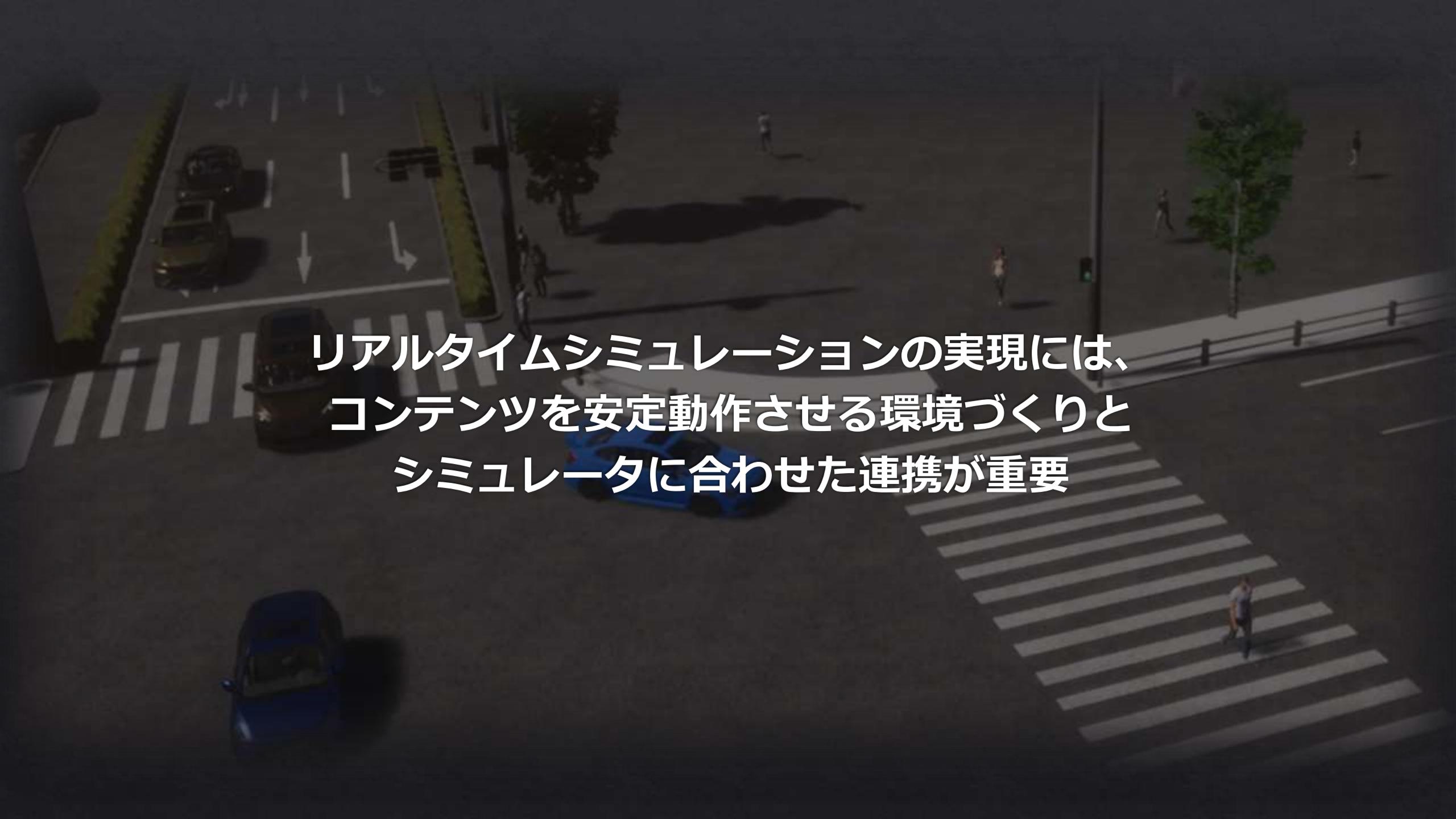
アノテーションまとめ

- Box枠や各種データを画像認識AIが必要とする形式で送信
- マップやシナリオに合わせ即座に物体認識の正解データを生成
- Boxのない映像をシミュレータに学習させ正解データと比較
→AIの学習に実際に活用できている

まとめ

- 環境→FPSを出すために日常的に軽量化を意識して作る
- 連携→シミュレータとのやり取り方法、同期するための仕組み作り





リアルタイムシミュレーションの実現には、
コンテンツを安定動作させる環境づくりと
シミュレータに合わせた連携が重要

Unreal Engineで リアルタイムシミュレーションを 実現するまで



株式会社理経

石川大樹、原田和樹