BLUE PROTOCOLにおける アニメ表現手法について ～実装編～

バンダイナムコスタジオ　技術スタジオ　技術スタジオ付
エグゼクティブテクニカルディレクター
大井 隆義

BLUE PROTOCOL™

To save the world that is going to destroy, fight beyond the spacetime. Cooperate with friends, beat the mighty enemies, change the history. That is your mission. Now, let's run out! On a vast land, heading for a hopeful future!

blue-protocol.com

自己紹介

大井 隆義

株式会社バンダイナムコスタジオ　技術スタジオ　技術スタジオ付

エグゼクティブテクニカルディレクター

2015年株式会社バンダイナムコスタジオ入社

BLUE PRTOCOLではエグゼクティブテクニカルディレクターとして技術全般をディレクション

実務としては主にグラフィック周りを担当。

アジェンダ

- イントロダクション
- ライティング
- 輪郭線
- 影
- ポストプロセス
- まとめ

アジェンダ

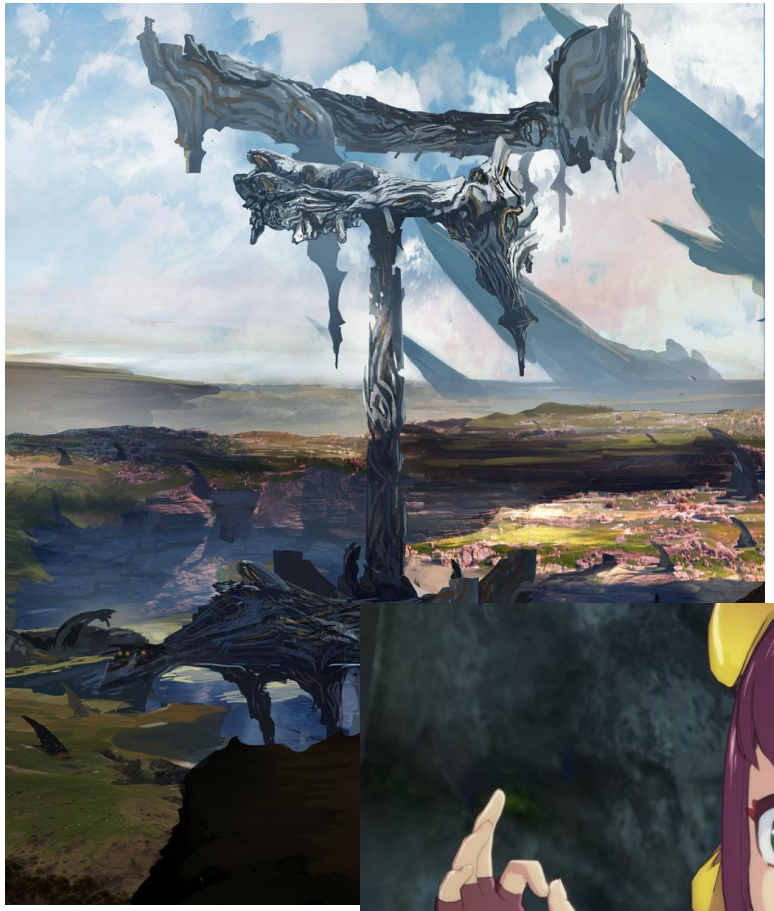
- **イントロダクション**
- ライティング
- 輪郭線
- 影
- ポストプロセス
- まとめ

BLUE PROTOCLとは

- バンダイナムコオンライン、バンダイナムコスタジオで共同開発中のオンラインアクションRPG
- Unreal Engine 4で開発
- 2020年4月にCβT実施

昨年の講演の振り返り

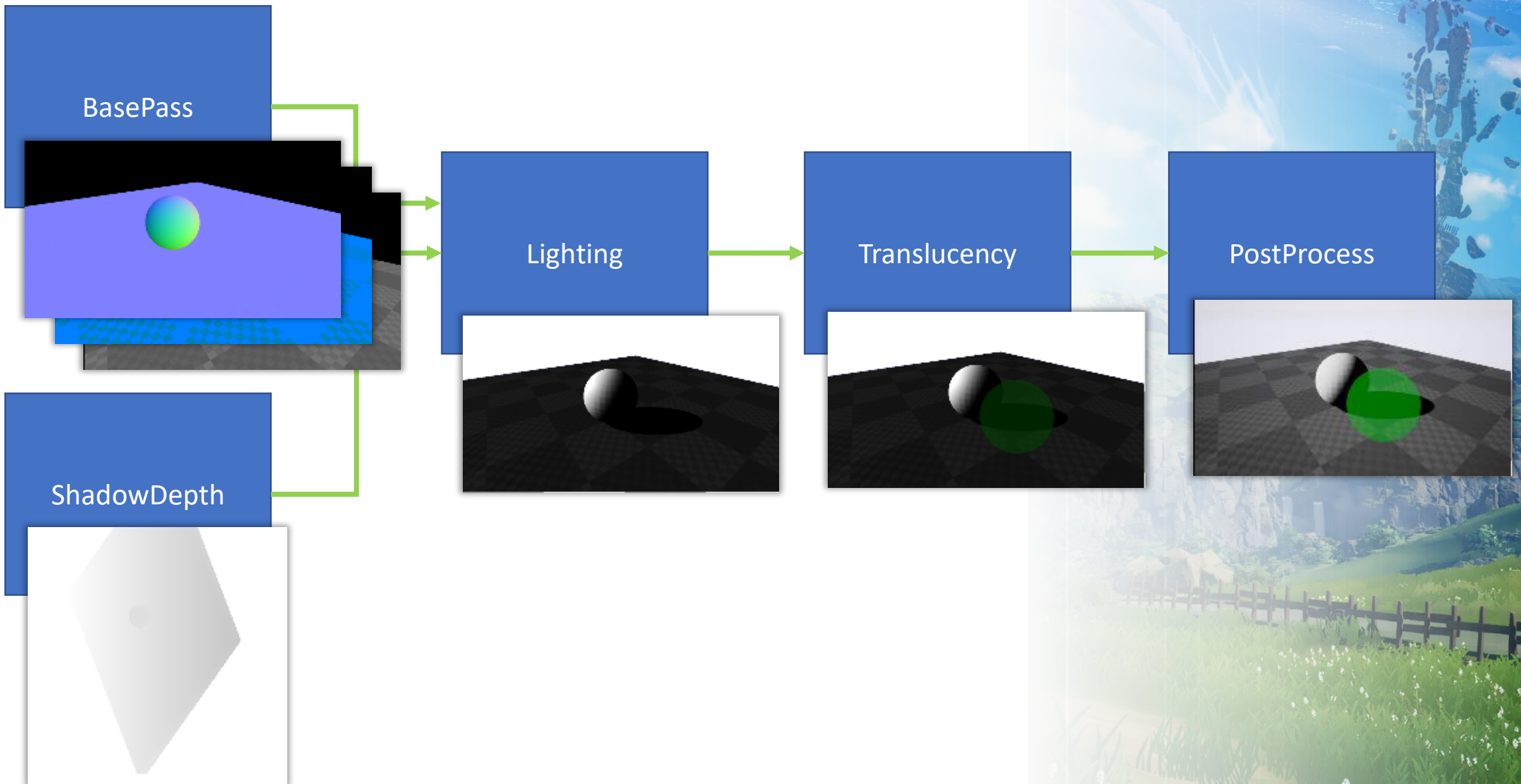
- CEDEC2020「 BLUE PROTOCOLにおけるアニメ表現手法について」



Unreal Engine 4

- デファードレンダリングによるPBR
- マテリアルはノードベースエディタでカスタマイズできる
  - BasePassやPostProcessなどに限られている

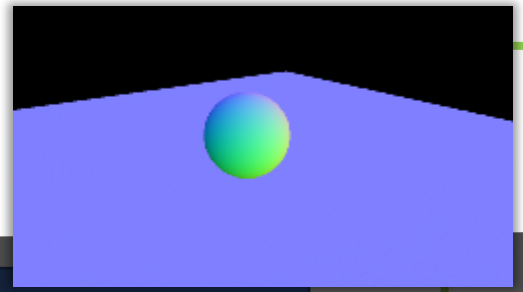
デファードレンダリング

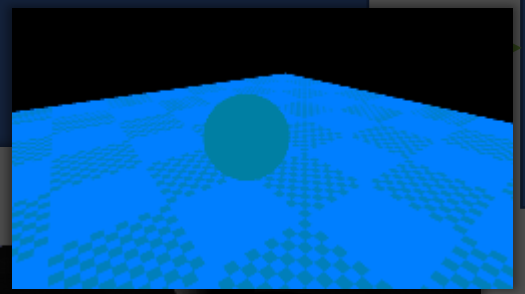


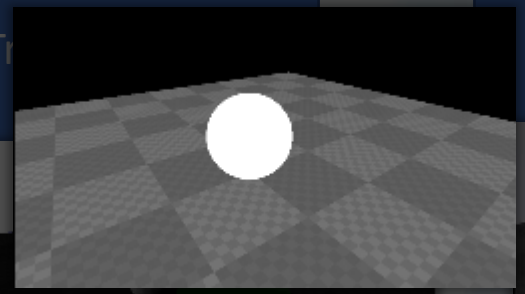
デファードレンダリング

BasePass

G-Buffer







PostProcess

etc…

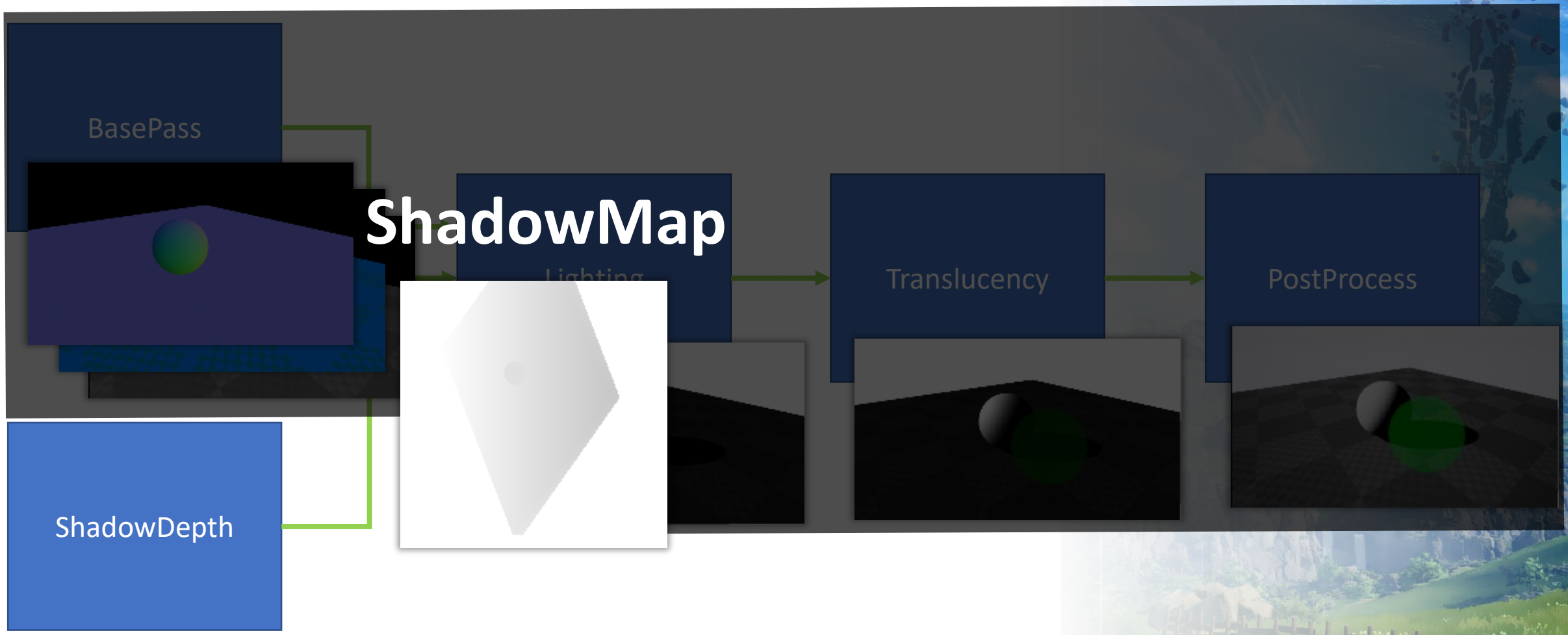
WorldNormal

Metallic
Specular
Roughness

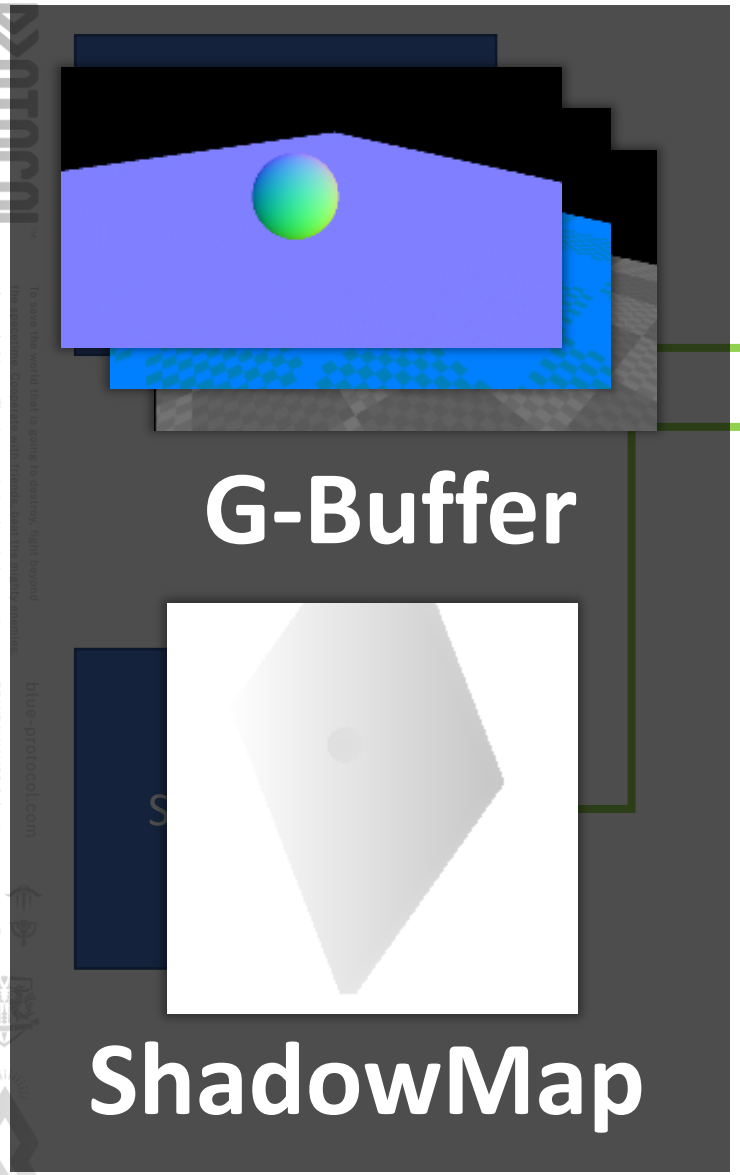
BaseColor

ShadowDepth

デファードレンダリング

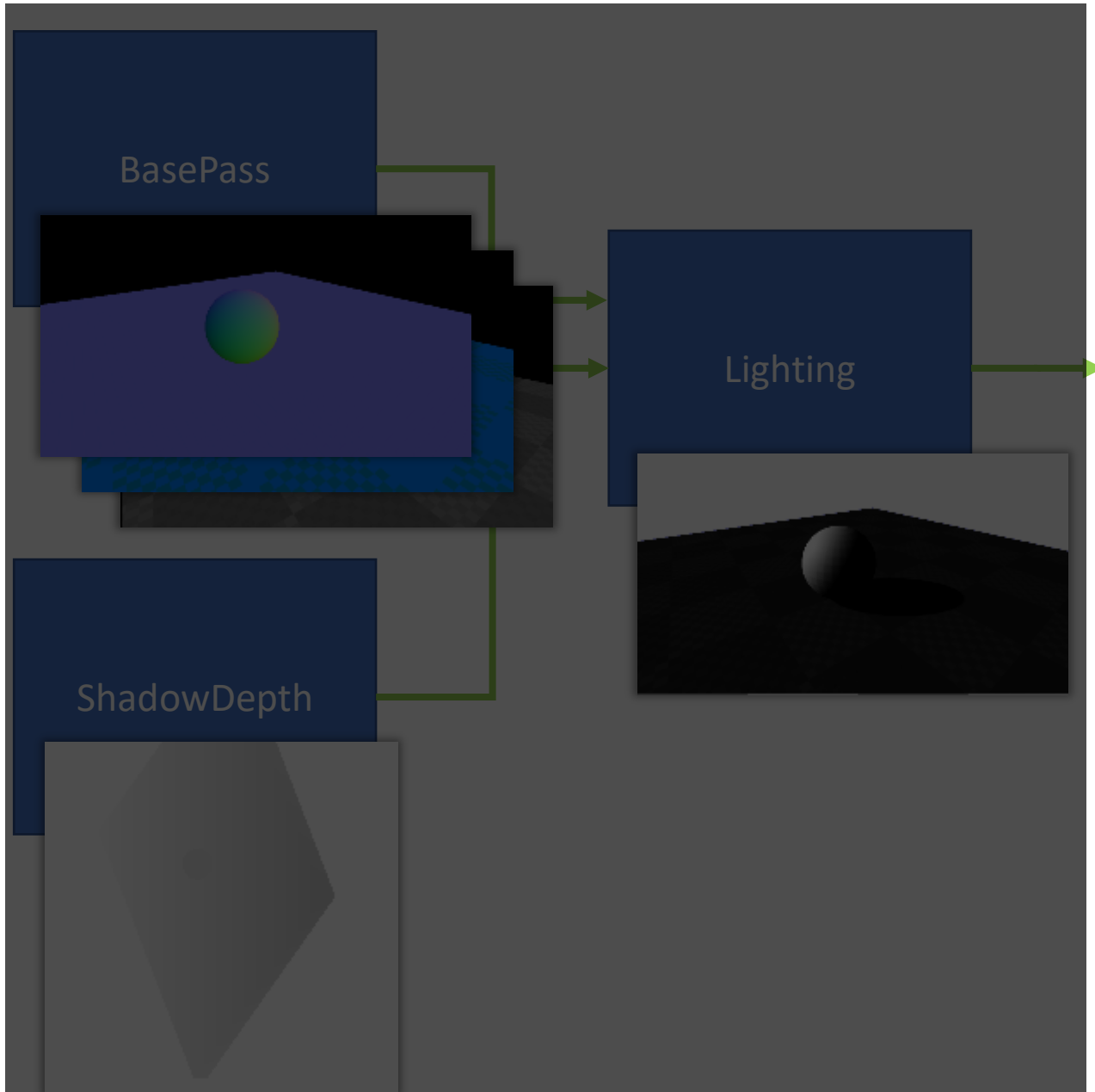


デファードレンダリング



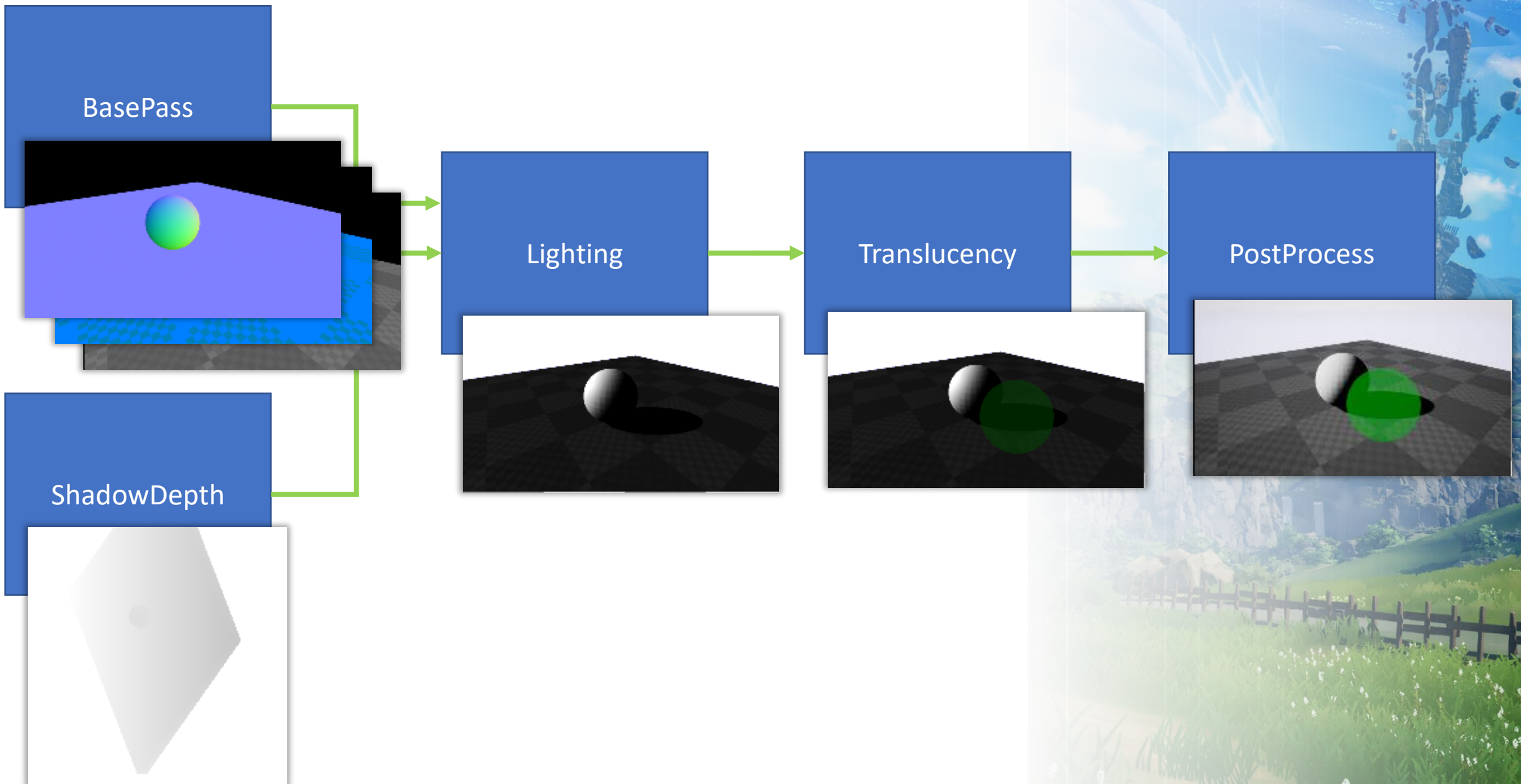
Lighting

デファードレンダリング



Translucency

デファードレンダリング



Unreal Engine 4

- 一般的なセルシェーディング手法

1. Surfaceマテリアル（BasePass）
   - ShadingModelをUnlitにして自前でライティングをする

2. PostProcessマテリアル
   - ライティング結果からランプを引いたり

Unreal Engine 4

- 一般的なセルシェーディング手法
  - メリット
    - エンジン改造をしないのでアップデートなどの影響が少ない
    - Webに参考になる情報がたくさんある
  - デメリット
    - UE4の豊富な機能を使えない
    - パフォーマンス的にもったいない
- **UE4の機能と両立できるよう最小限のエンジン改造をして行こう**
  - ShadingModel追加
  - Passの改造＆追加

アジェンダ

- イントロダクション
- **ライティング**
- 輪郭線
- 影
- ポストプロセス
- まとめ

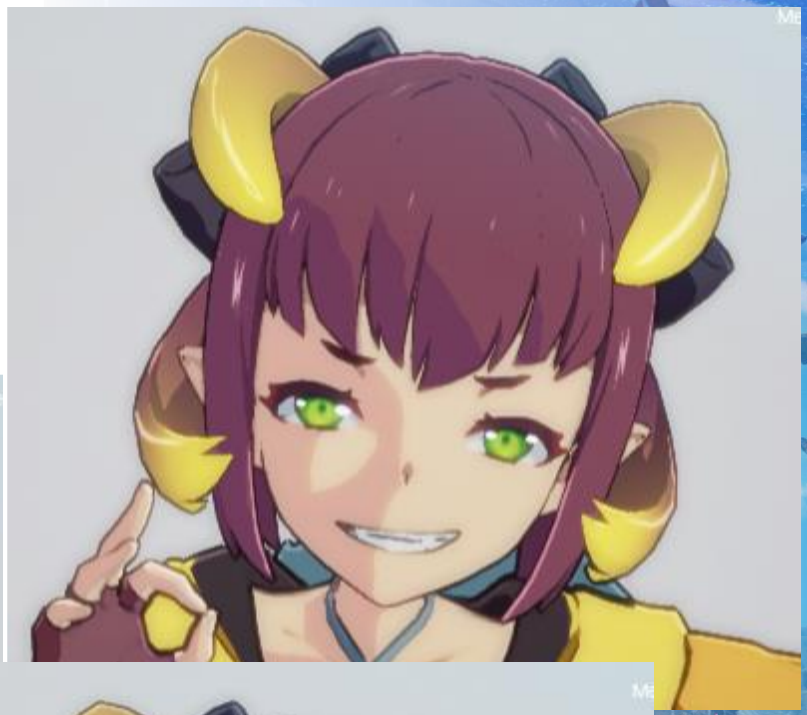
ライティング（Mesh）

- 法線が2つ
  1. DCCツールからインポートした法線
     （Import Normal）
  2. インポート時に頂点位置から求めた法線
     （Compute Normal）

- 頂点カラー
  - Red : Diffuse Offset
  - Blue : Ambient Occlusion

ライティング（Mesh）

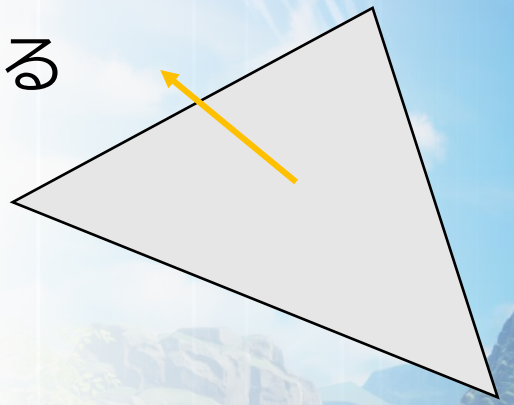
1. DCCツールからインポートした法線
   - アニメっぽい影が入るように調整



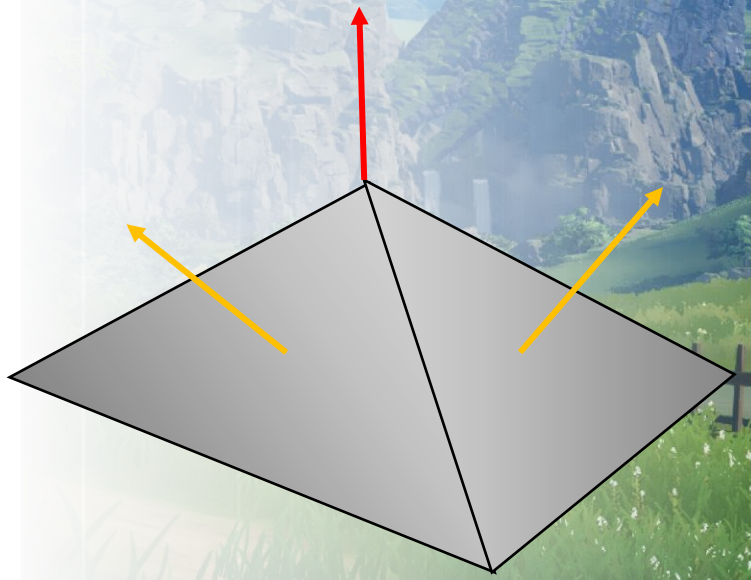
ライティング（Mesh）

2. インポート時に頂点位置から求めた法線

① 3頂点から面法線を求める

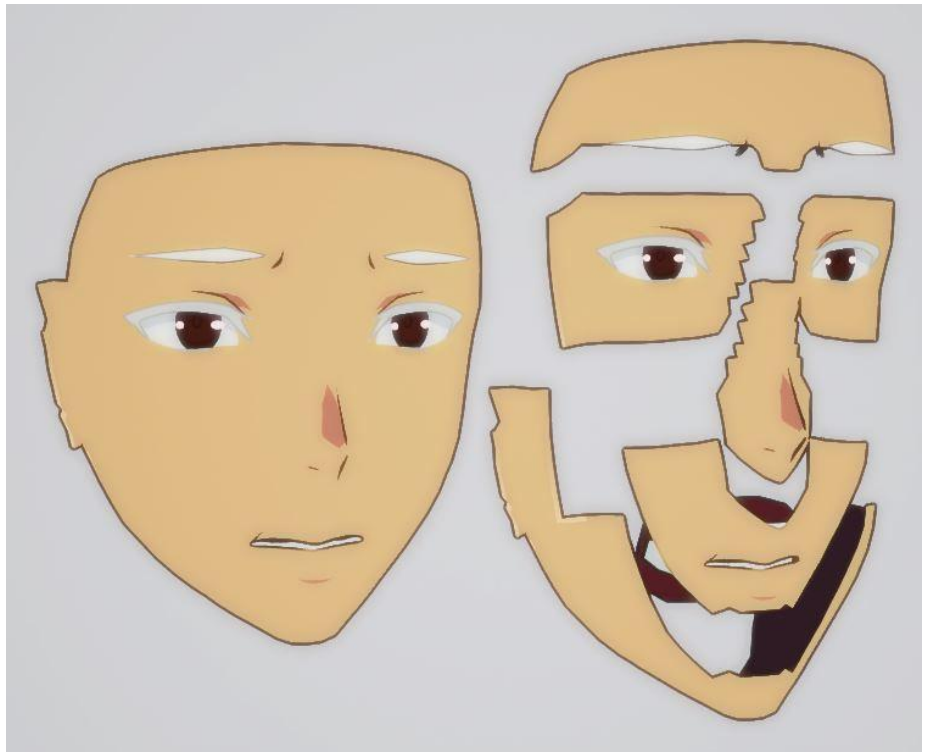


② 隣り合う三角形の面法線の平均が頂点法線



ライティング（Mesh）

2. インポート時に頂点位置から求めた法線
    - アバターはキャラクリの都合上パーツがバラバラ
    - 法線を繋げるために一括インポート



ライティング（BasePass）

BasePass

G-Buffer

ライティング（BasePass）

- BaseColor（明るいところの色）

ライティング（BasePass）

- ShadowColor（暗いところの色）

Luminance(輝度)

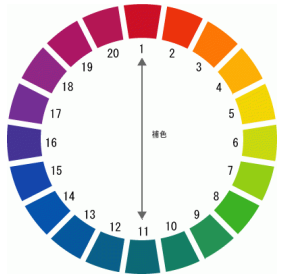
8段階

Saturation(彩度)

Value(明度)

Hue(色相)

10段階



Saturation	8 Array elements
0	15.0
1	10.0
2	10.0
3	10.0
4	10.0
5	7.5
6	5.0
7	5.0

Hue R	0.0
Hue YR	-5.0
Hue Y	-10.0
Hue GY	-10.0
Hue G	10.0
Hue BG	10.0
Hue B	8.0
Hue PB	4.0
Hue P	4.0
Hue RP	4.0

Value	8 Array elements
0	-5.0
1	-10.0
2	-15.0
3	-20.0
4	-30.0
5	-35.0
6	-35.0
7	-35.0

ライティング（BasePass）

- 髪のBaseColor
  - マテリアルで色を指定

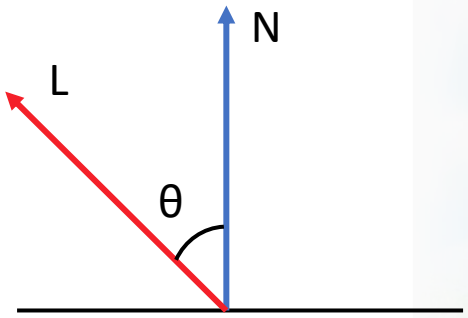
  - 二色の場合はグラデーションマップ

- 髪のShadowColor
  - マテリアルで色を指定

  - BaseColorに依存しない色を指定できる

ライティング（BasePass）

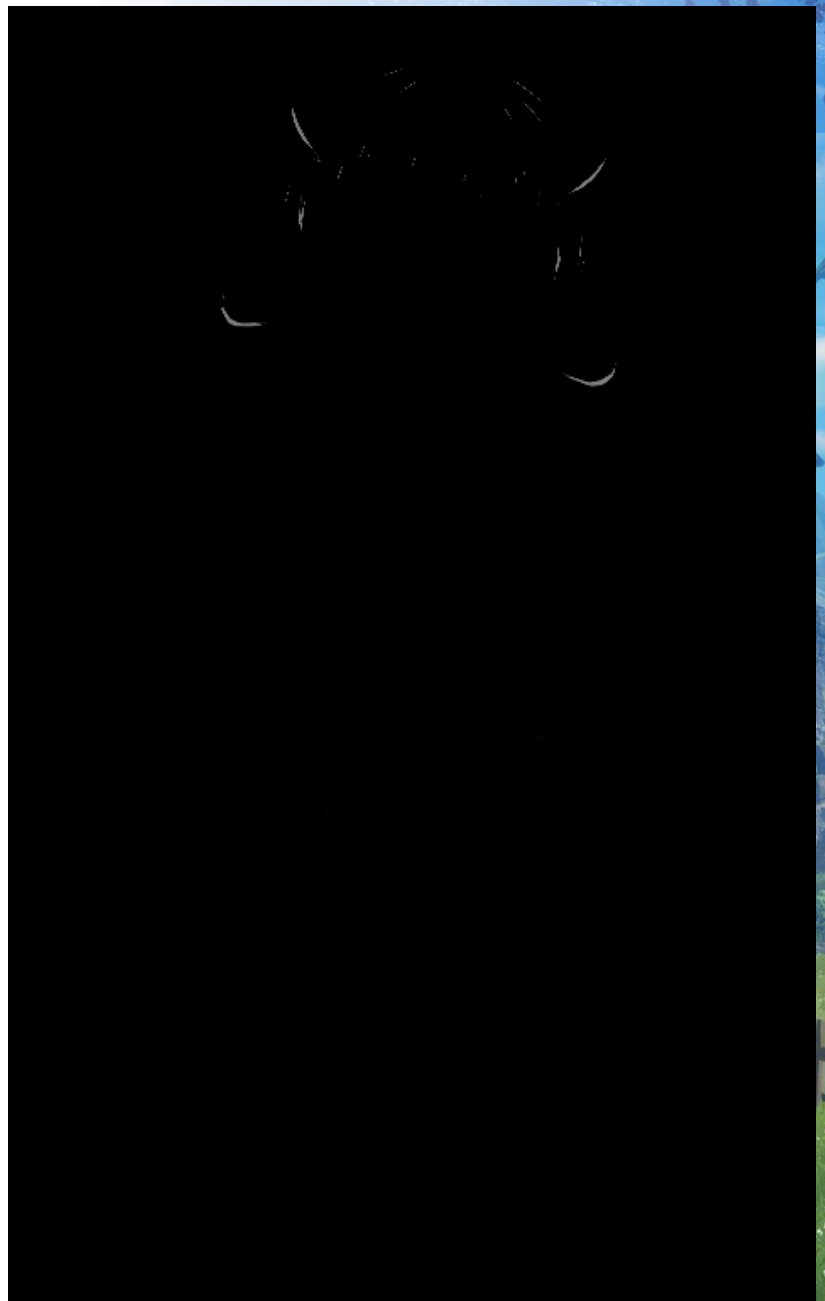
- NdotL（主光源とImport Normalの内積を2値化）
  - HalfNoL = dot(Normal, Light) * 0.5 + 0.5;
  - ToonNoL = step(Threshold, HalfNoL);



- なぜNdotLをBasePassで求めるのか？
  - G-BufferにはCompute Normalを格納したい
    形状が必要になる処理で利用するため
  - ライティングパスやポストプロセスパスで内積＆2値化を行うとG-Buffer(R10G10B10A2)の精度によりガタガタすることがある
  - 陰影をコントロールするパラメーターが多くてG-Bufferに乗りきらないため

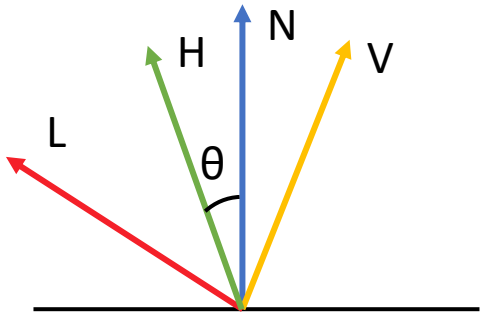
ライティング（BasePass）

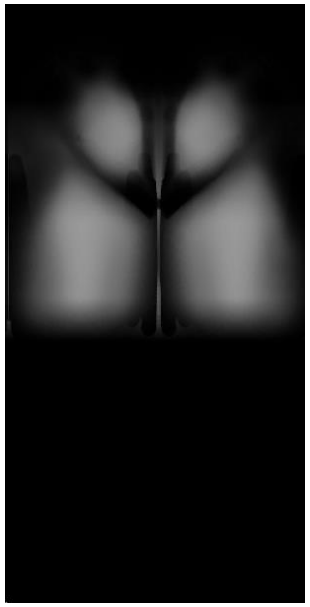
- SpecularMask（主光源のスペキュラー）



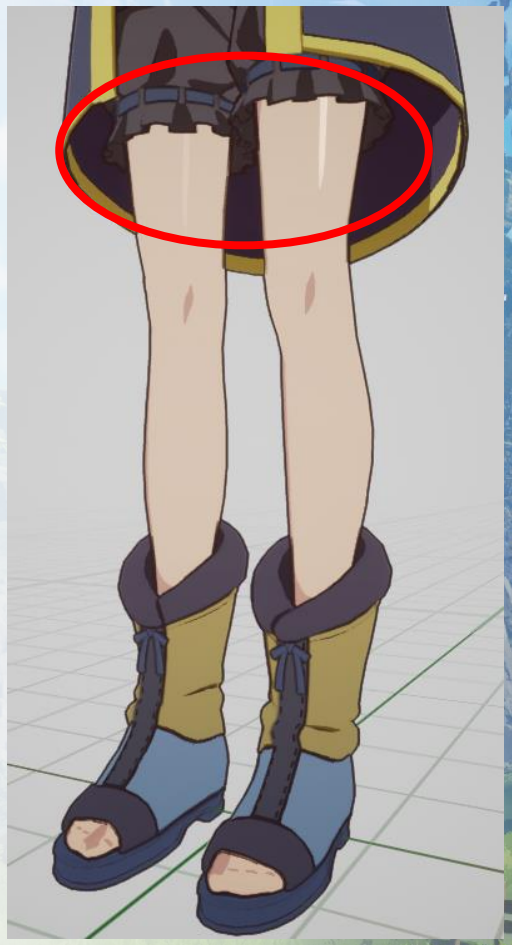
ライティング（BasePass）

- SpecularMask（主光源のスペキュラー）
  - Blinn-Phong反射モデルを2値化
    - H = normalize(Light + View);
    - NoH = dot(Normal, H);
    - ToonNoH = step(Threshold, NoH);



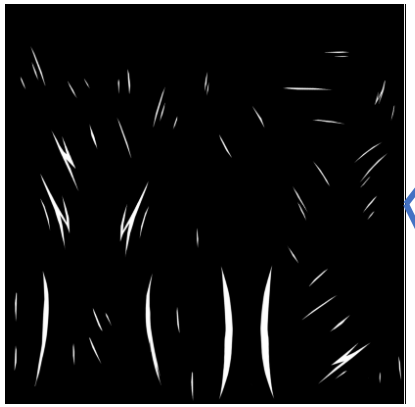


SpecularMap

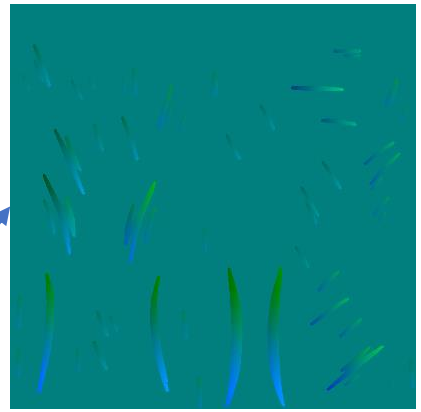


ライティング（BasePass）

- SpecularMask（主光源のスペキュラー）
  - 髪のハイライト

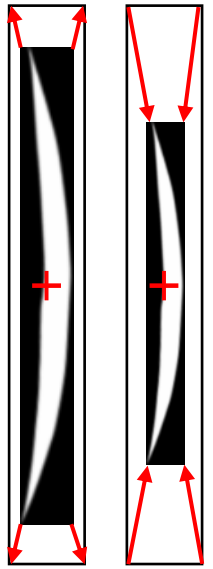


Mask（元画像）



重心からの距離XY

ID



拡大

縮小

伸縮調整用デバッグ表示

ライティング（BasePass）

- SpecularValue（スペキュラー明度）

ライティング（LightingPass）

ライティング（LightingPass）

- ディレクショナルライト
  - BasePassで出力したG-Bufferを使いライティング
  - NdotLに基づきBaseColorとShadowColorを適用
    - ShadowColorはSubsurfaceとして処理

ライティング（LightingPass）

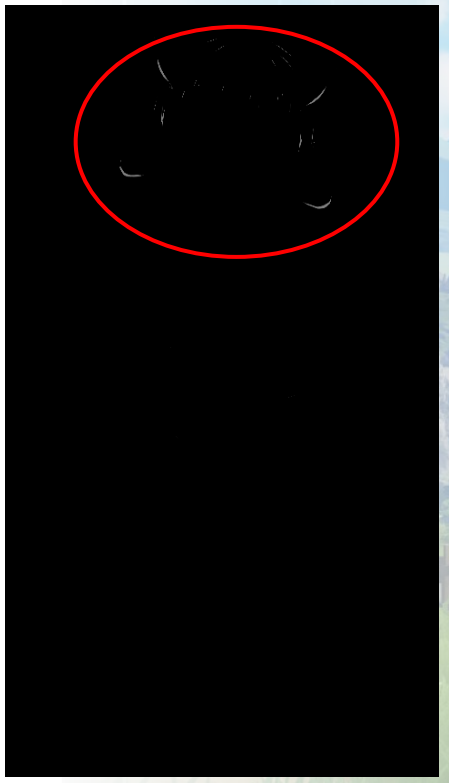
- ディレクショナルライト
  - BasePassで出力したG-Bufferを使いライティング
  - スペキュラー
    - SpecularValueをBaseColorの明度に加算してスペキュラーカラーを求める
    - SpecularMaskの個所にスペキュラーカラーを加算する

BaseColor

+

SpecularValue

×



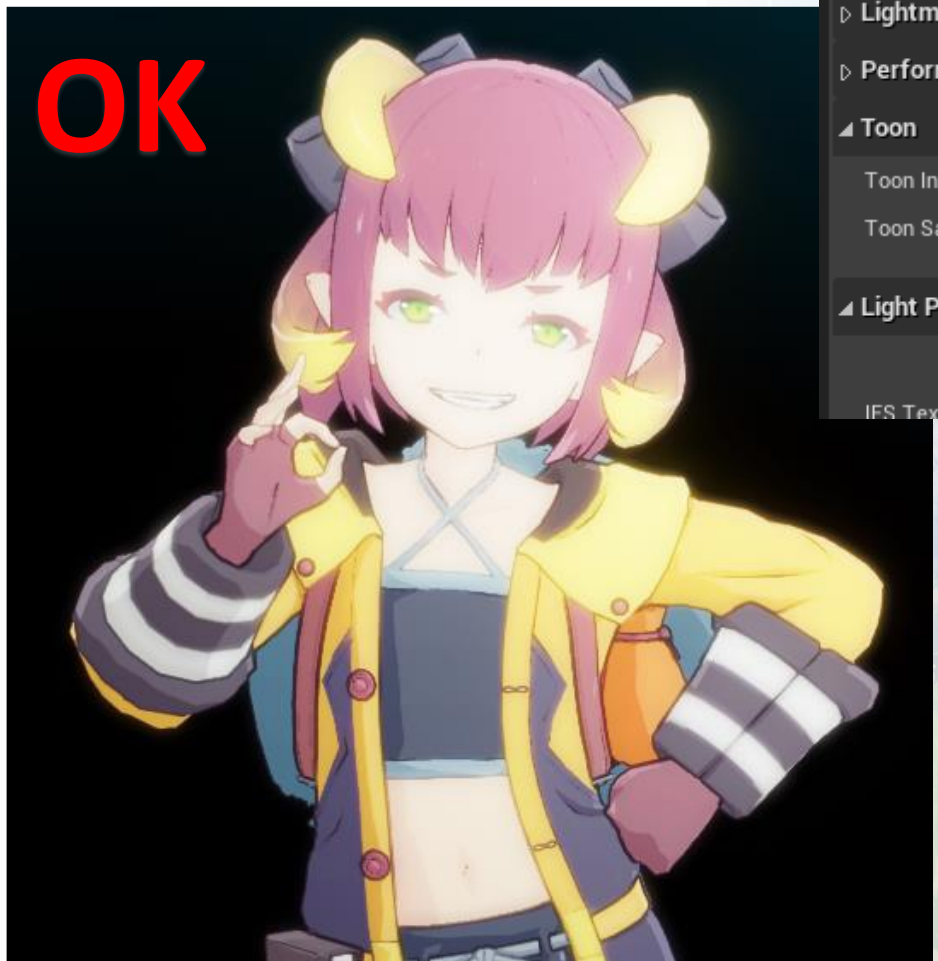
SpecularMask

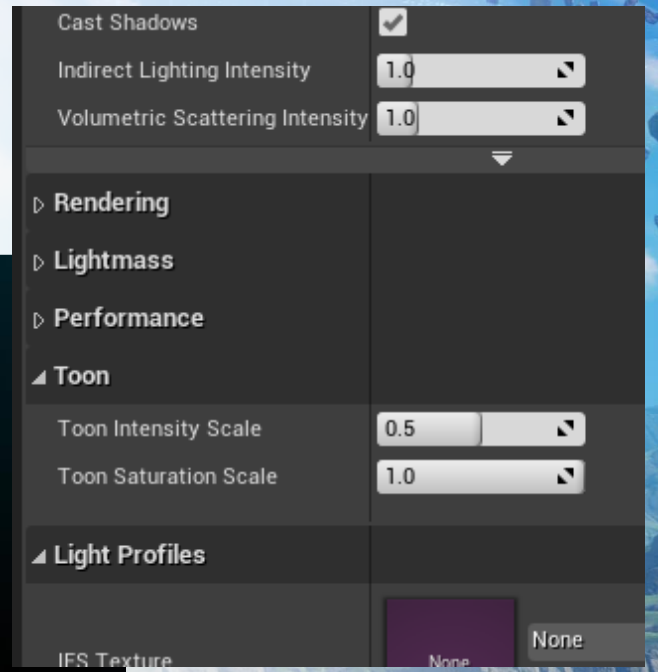
ライティング（LightingPass）

- ディレクショナルライト

ライティング（LightingPass）

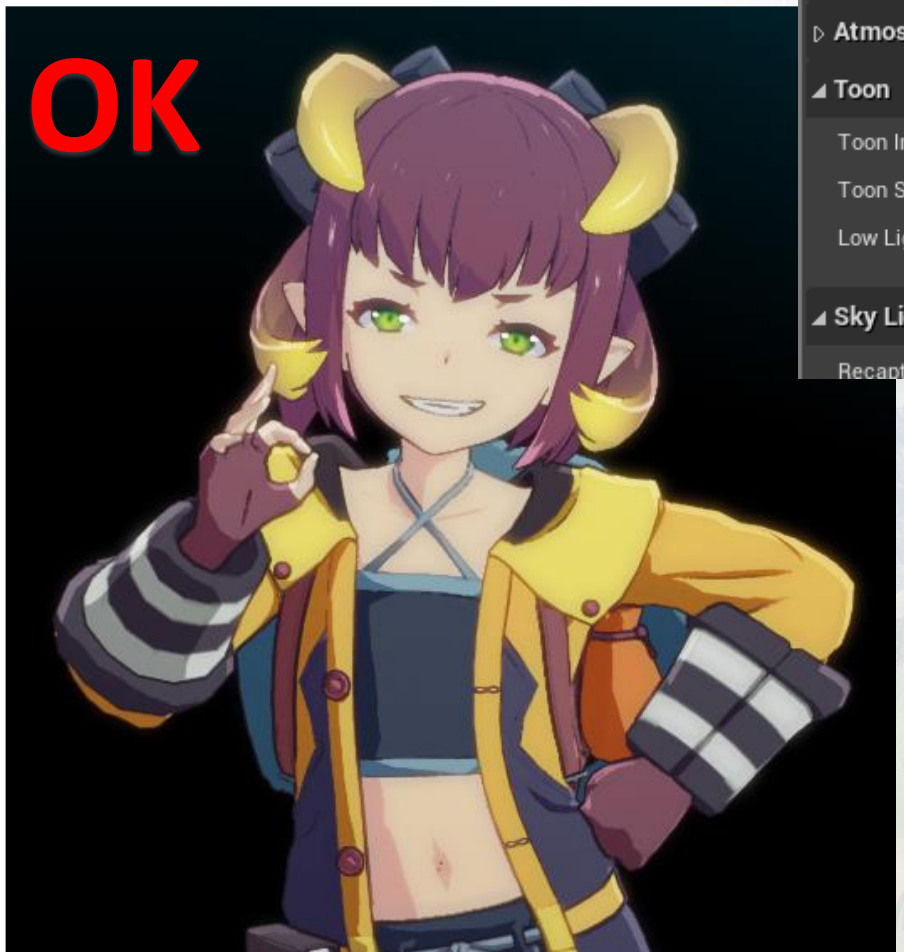
- ポイントライト
  - 法線を無視し、ライトとの距離だけで適用
  - トゥーン用にパラメーターを追加

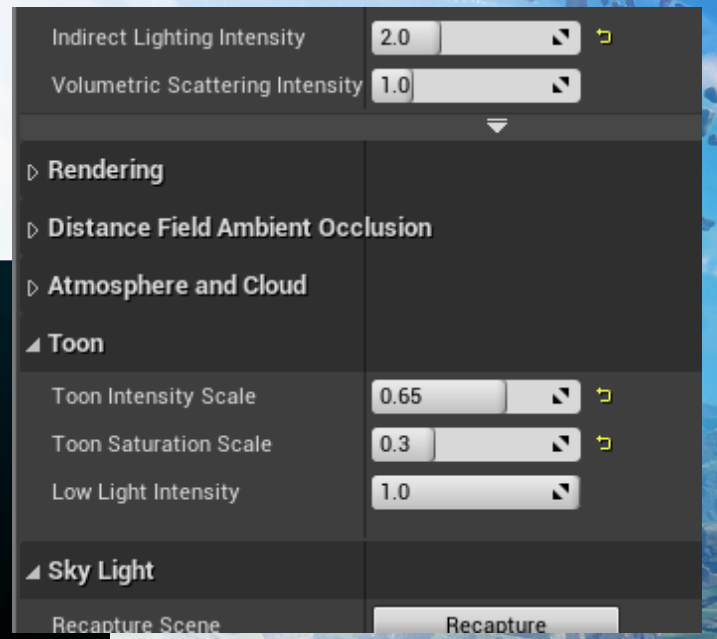




ライティング（LightingPass）

- スカイライト&間接光
  - 法線をすべて真上（0.0, 0.0, 1.0）として処理
  - トゥーン用にパラメーターを追加





リムライト

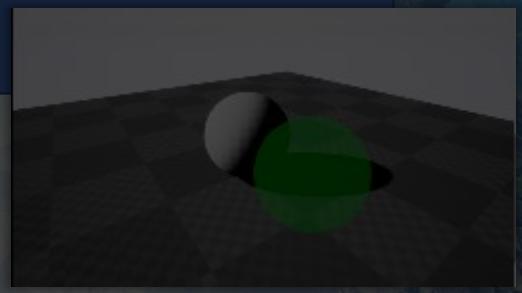
- 太さを均一にするためにポストプロセスのような手法で行っています

リムライト（BasePass）

BasePass

G-Buffer

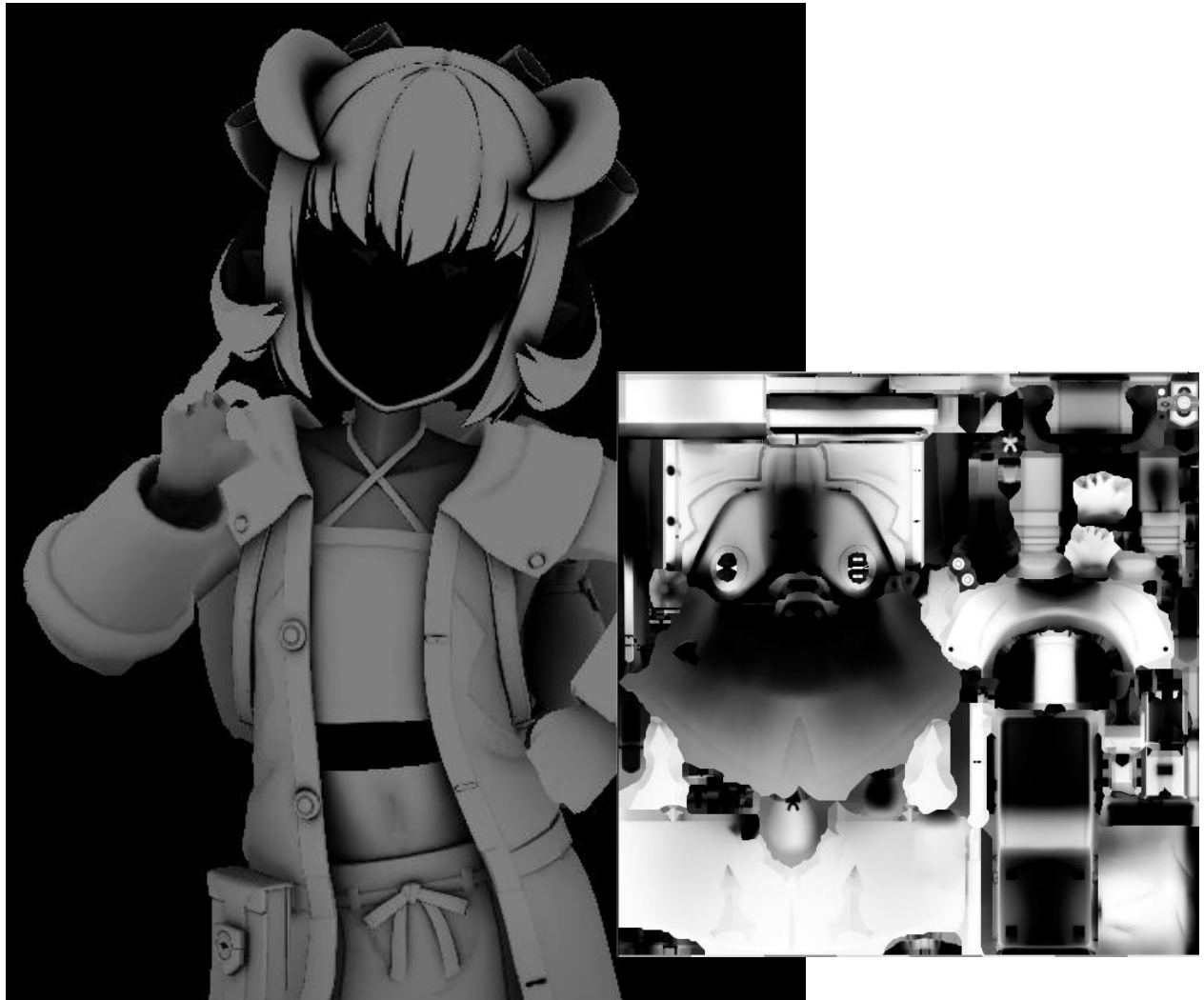
PostProcess



ShadowDepth

リムライト（BasePass）

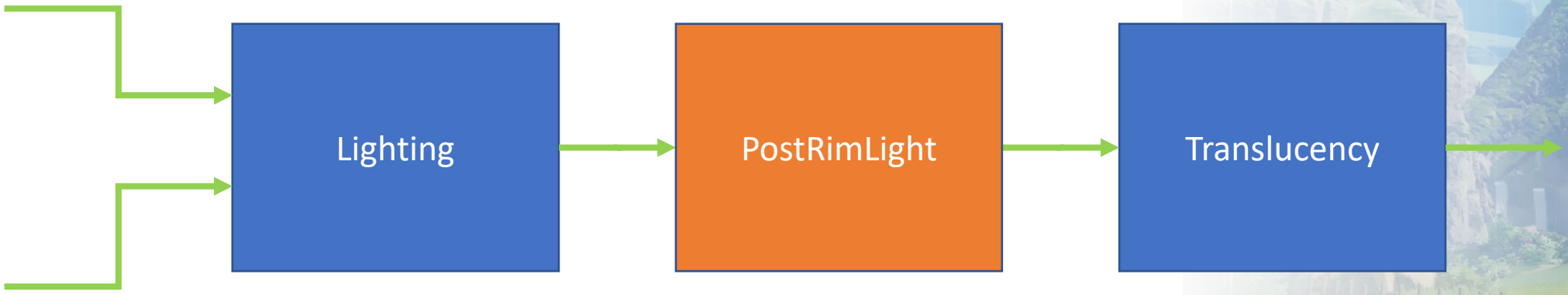
RimLightMask（テクスチャ＆パラメーター）

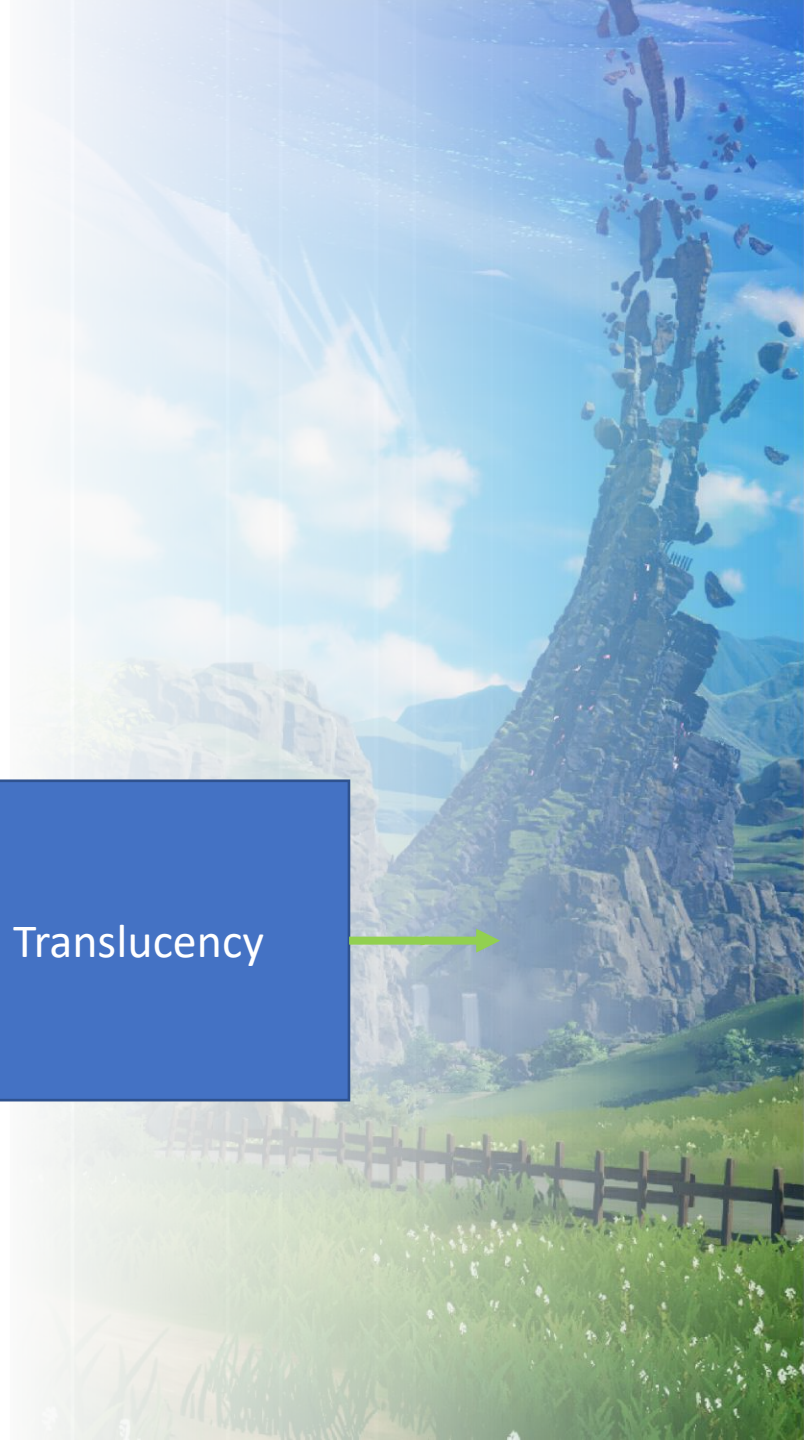


RimLightWidth（パラメーター）

リムライト（PostRimLight）

- パスの追加
  - Lightingパスの次に追加
  - ライティングと同様に扱う





リムライト（PostRimLight）

- デプスからSobelフィルターによる輪郭抽出

Depth

RimLightWidth

×

×

X方向エッジ抽出

-1	**0**	**1**
-2	0	2
-1	0	1

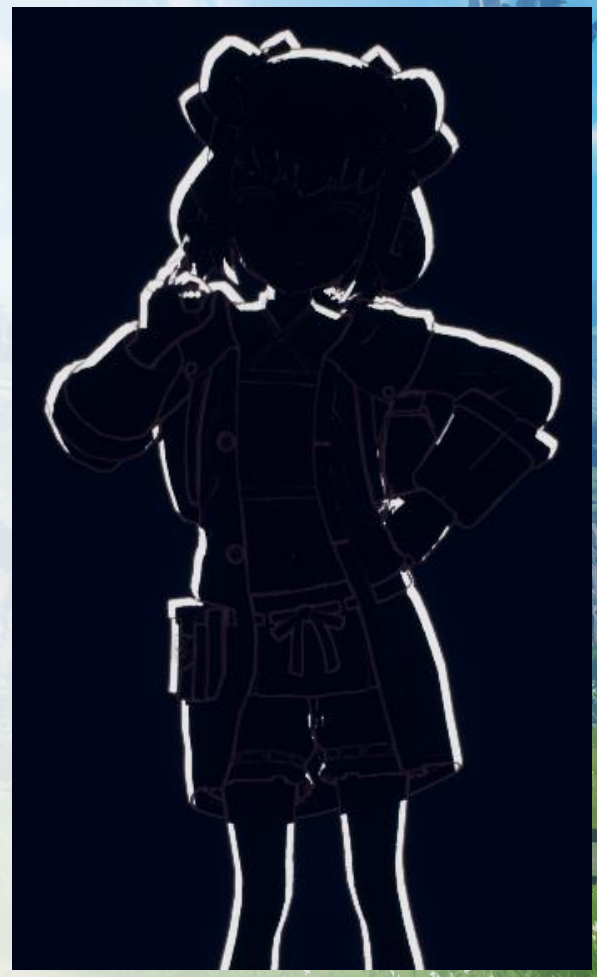
+

Y方向エッジ抽出

-1	**-2**	**-1**
0	0	0
1	2	1

=

抽出結果



リムライト（PostRimLight）

- RimLightMaskで出てほしくないところをマスク
- スペキュラーカラーを加算

抽出結果

×

RimLightMask

アジェンダ

- イントロダクション
- ライティング
- **輪郭線**
- 影
- ポストプロセス
- まとめ

輪郭線（Mesh）

- 頂点カラー1（Green）
  - デプスシフト

輪郭線（Mesh）

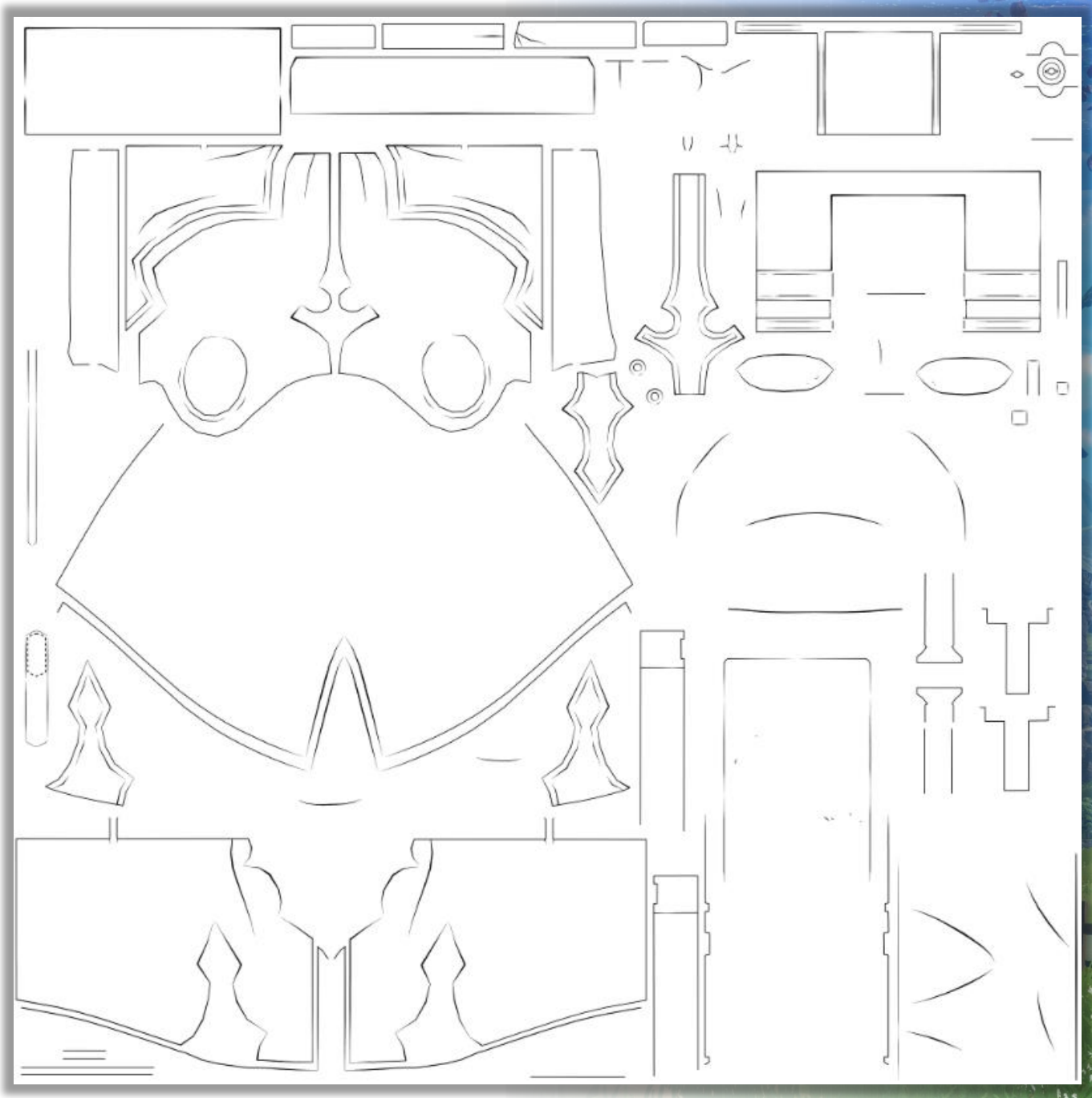
- 頂点カラー2（Red、Green）
  - インポート時にTexCoord1へ格納

太さ

内部輪郭用ID

輪郭線（Mesh）

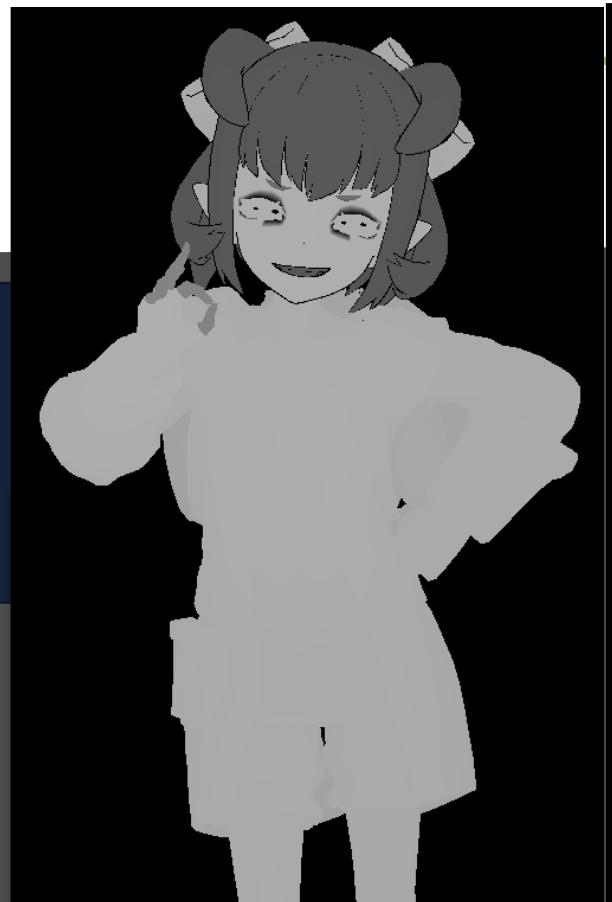
- テクスチャ描きこみライン



輪郭線（BasePass）

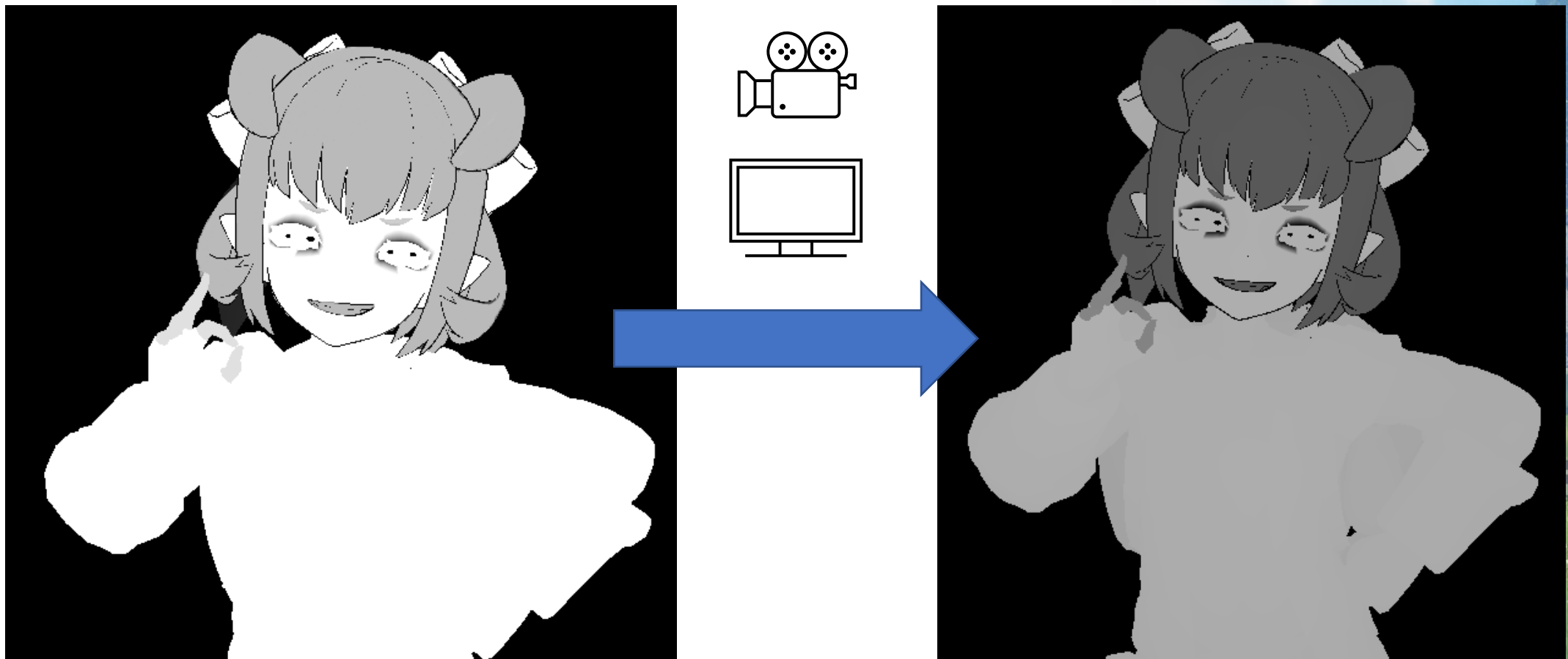
BasePass

G-Buffer



輪郭線（BasePass）

- Red：輪郭線の太さ
  - TexCoord1.xからの入力
  - カメラからの距離による調整
  - アスペクト比による調整

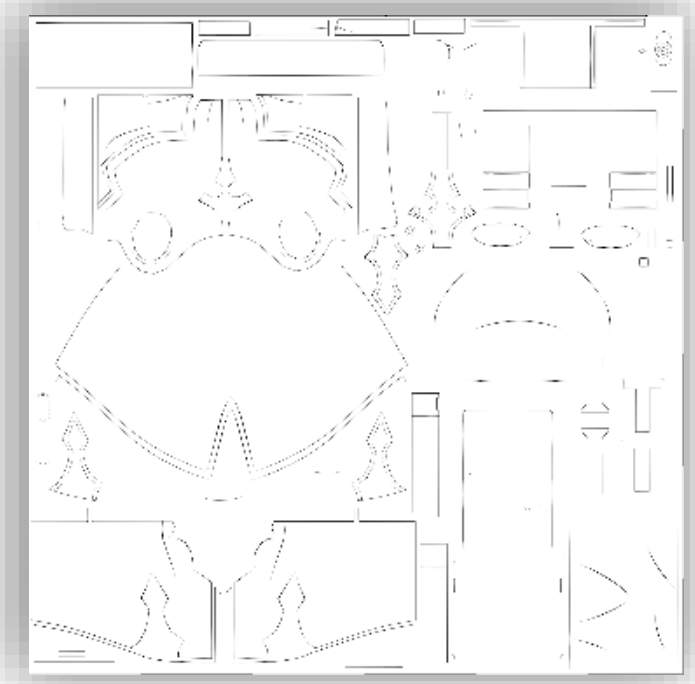


輪郭線（BasePass）

- Green：内部輪郭用ID
  - TexCoord1.yからの入力

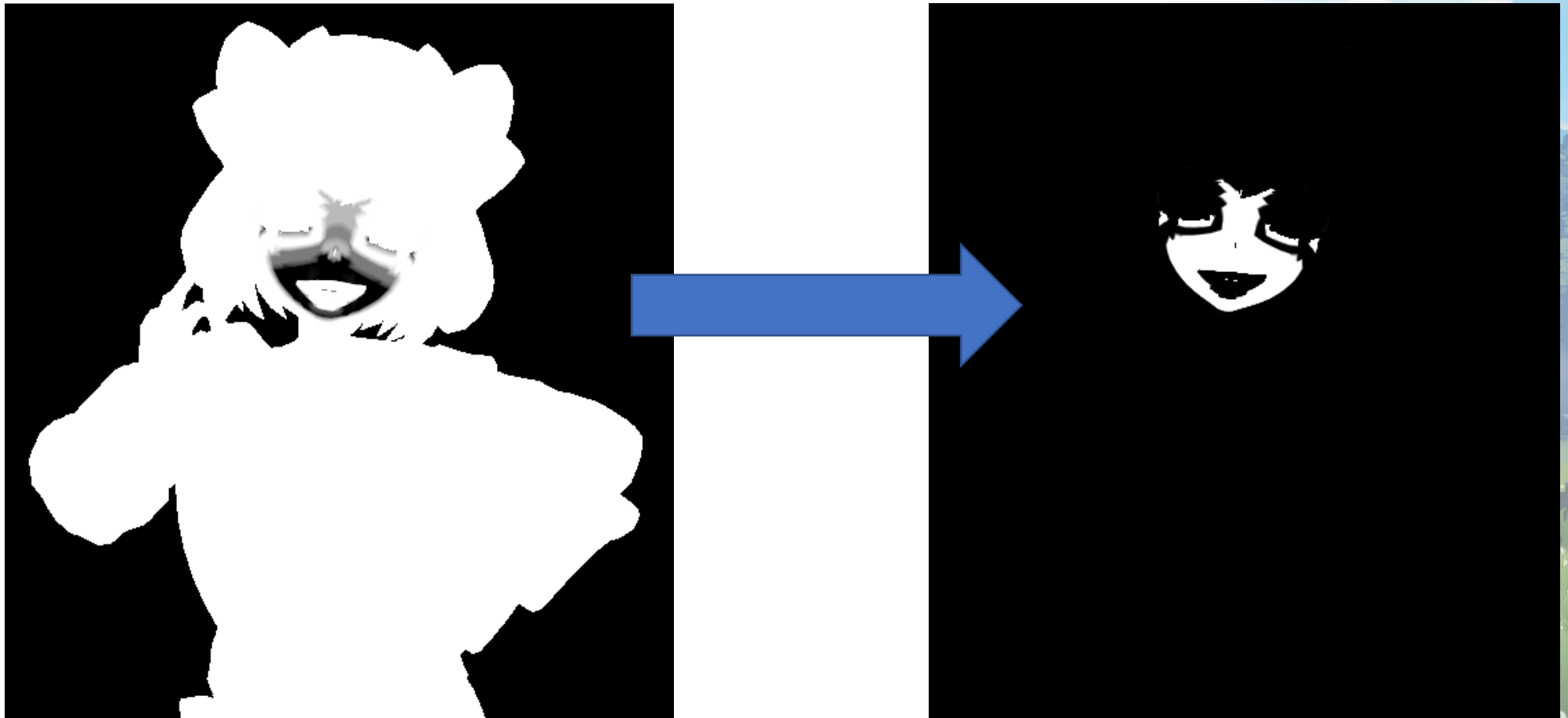
輪郭線（BasePass）

- Blue：描きこみ
  - テクスチャ
  - 輪郭モデル（オーバーハング対策）



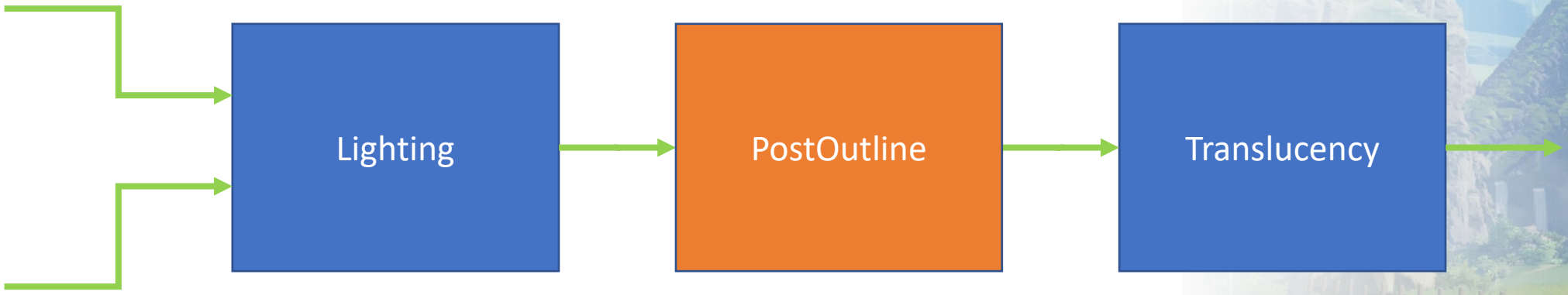
輪郭線（BasePass）

- Alpha：デプスシフト
  - VertexColor.gから入力
  - デプスによる輪郭抽出にて利用
  - 反転＆パラメーターによる調整



輪郭線（PostOutlinePass）

- パスの追加
  - LightingとTranslucencyの間に追加
  - 半透明の処理にて屈折などが行われるため（水面とか）
  - 半透明には輪郭線が描けない



輪郭線（PostOutlinePass）

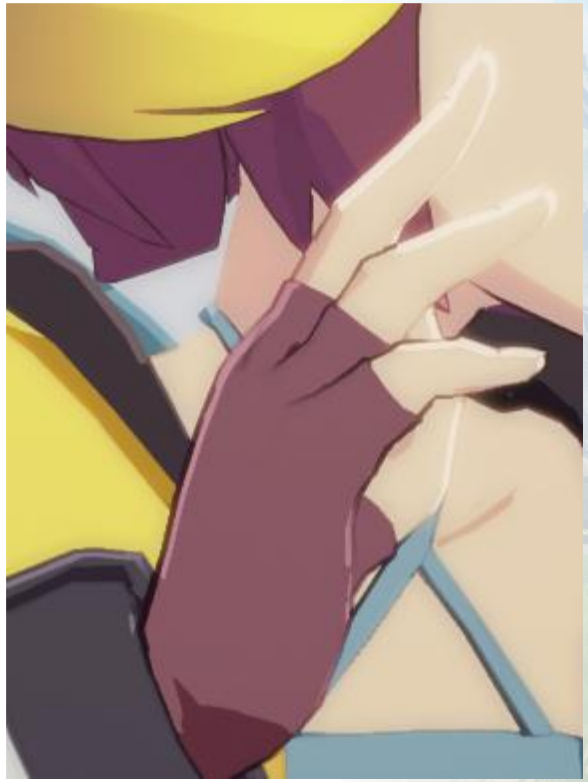
- デプスによる判定
  - Sobelフィルターにて求める
  - デプスを比較して大きい値のほうにのみ輪郭を描く
    - モデルの外側に描かれるイメージ
  - デプスシフトにて線を描かれにくくする

輪郭線（PostOutlinePass）

- IDによる判定
  - Sobelフィルターにて求める
  - 値を比較して大きいほうに描かれる

輪郭線（PostOutlinePass）

- 法線による判定
  - 内積で求める
  - デプスの差が少ない
  - IDを分けることができない



輪郭線（PostOutlinePass）

- 描き込み
  - G-Bufferに出力された箇所

輪郭線（PostOutlinePass）

- 完成

輪郭線OFF

抽出結果

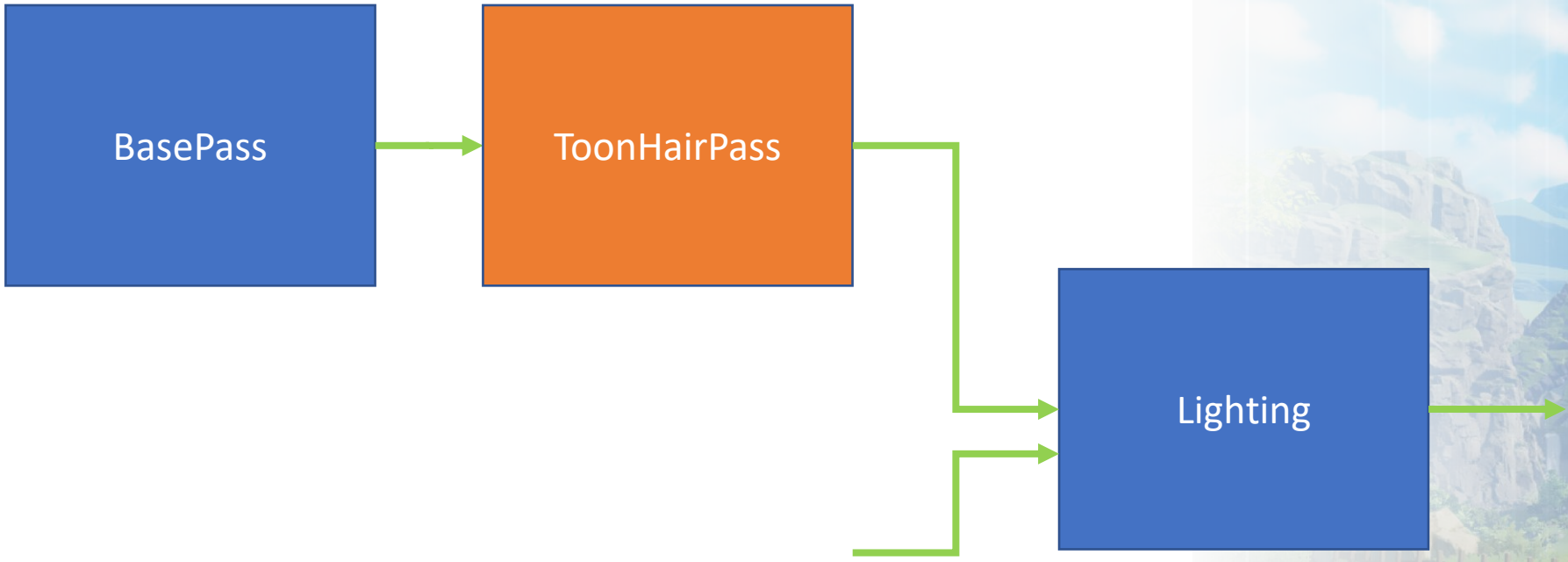
輪郭線ON

眉輪郭線

- アニメでよくある眉の輪郭線が髪の上に描かれるやつ

眉輪郭線（ToonHairPass）

- 眉の輪郭線を透過させるため髪の描画をBasePassと分ける



輪郭線（BasePass）

- BasePassでまず髪以外を描画する
- 眉をマスクしておく
  - マテリアルが分かれている

髪以外のBaseColor

髪以外の輪郭線ID

眉部分のマスク

輪郭線（ ToonHairPass ）

- 眉でマスクされている箇所は、内部輪郭用IDを更新しない
- これでポストによって輪郭線が引かれる

眉輪郭線

- アニメでよくある眉の輪郭線が髪の上に描かれるやつ

輪郭線（アンチエイリアス）

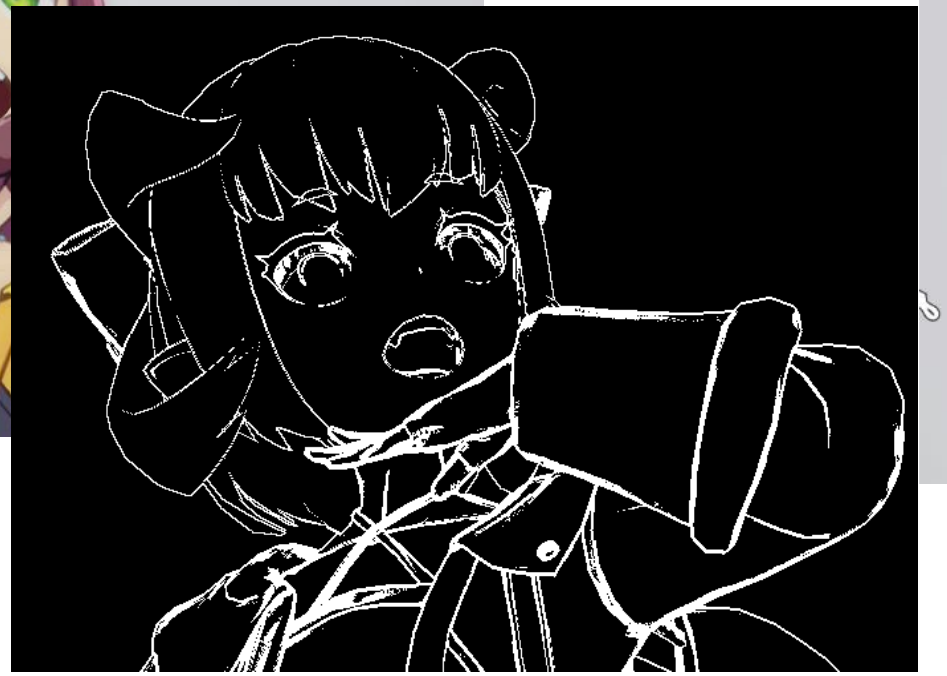
- TemporalAAによって輪郭線が溶ける

輪郭線（アンチエイリアス）

- TemporalAAによって輪郭線が溶ける

輪郭線（アンチエイリアス）

- ResponsiveAAを利用
  - 輪郭線をステンシルに描きこみ、溶けるのを軽減させる

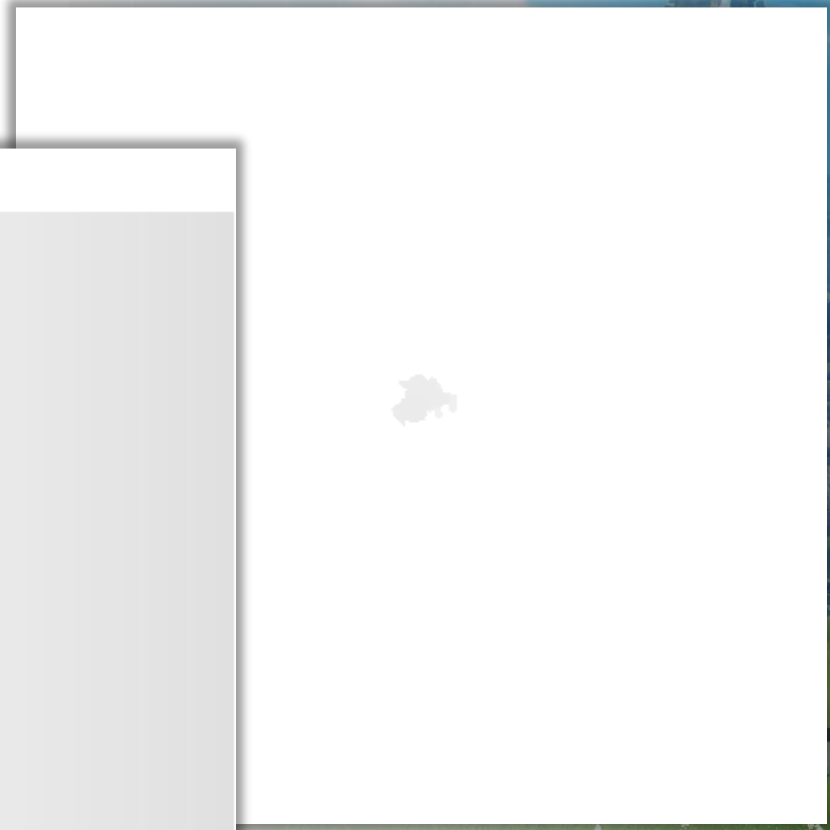


アジェンダ

- イントロダクション
- ライティング
- 輪郭線
- **影**
- ポストプロセス
- まとめ

影（ShadowMap）

- トゥーンはシャドウマップによるセルフシャドウは受けない
  - シャドウマップの解像度によって余り綺麗に落ちない
  - トゥーンマテリアルか否かでシャドウマップを分ける
  - 背景の影などは受ける



影（ShadowMap）

- 背景用シャドウマップ

影（ShadowMap）

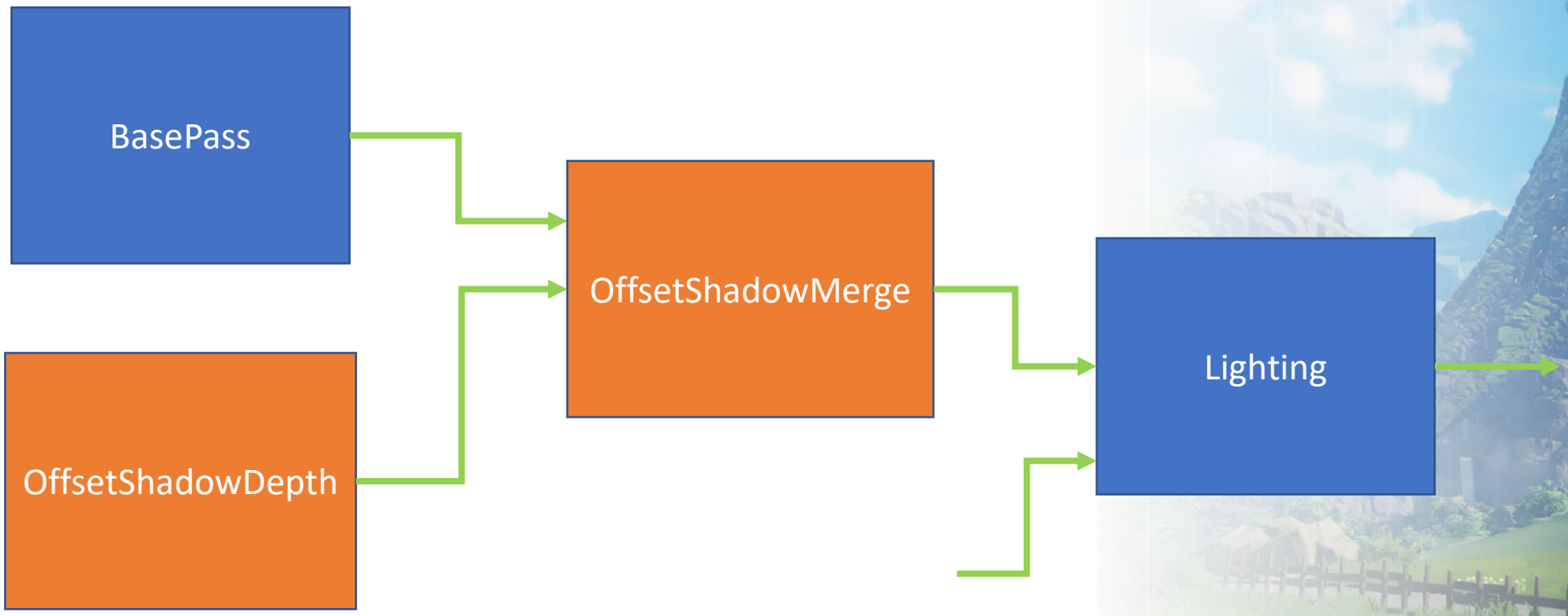
- キャラクター用シャドウマップ

オフセットシャドウ

- セルフシャドウを受けない代わりに情報量を増やすため
- スクリーンスペースでのオフセットで表現

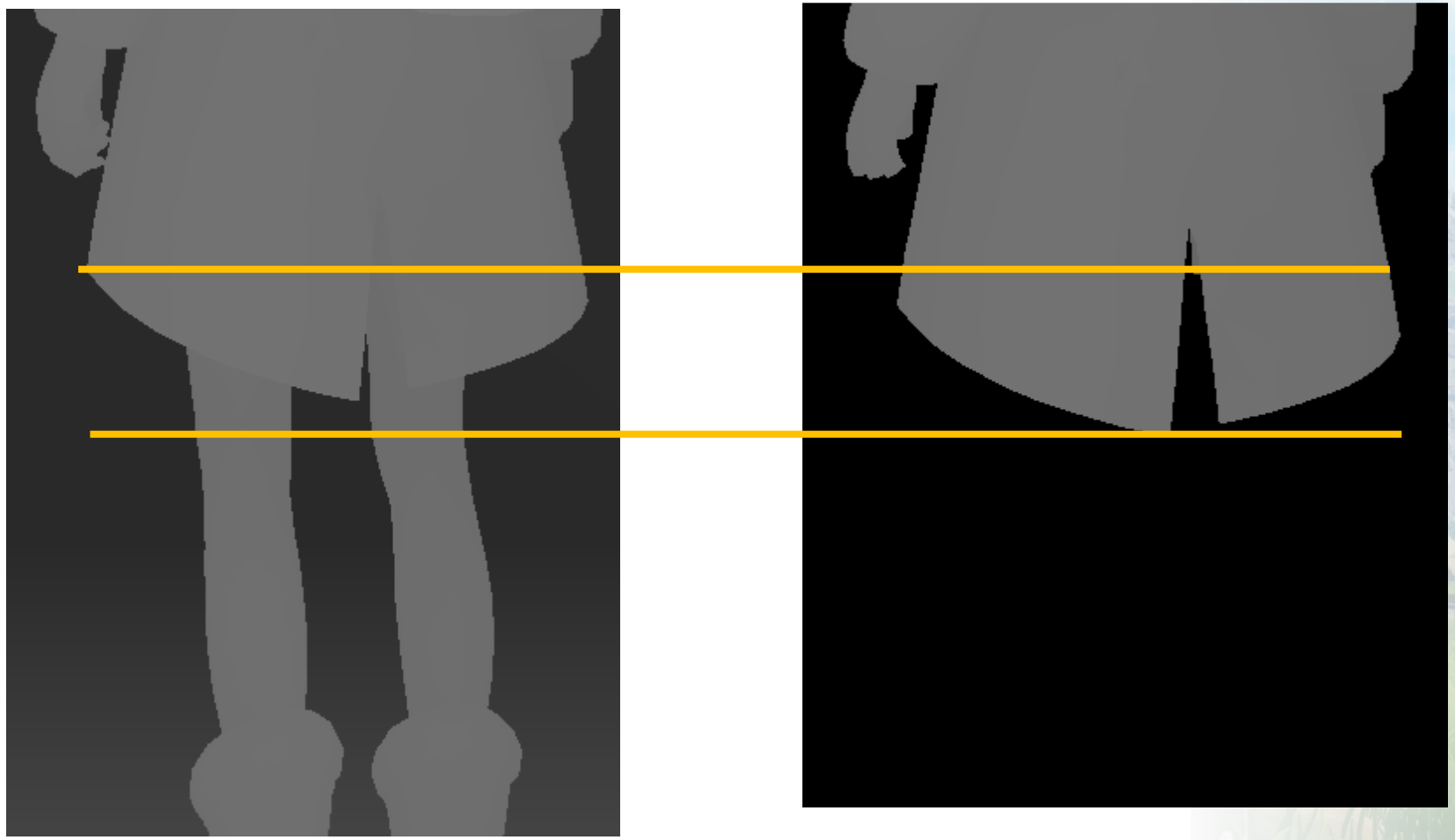
オフセットシャドウ（OffsetShadowPass）

- 影を落とす対処のデプスを描画するパスと、G-Bufferに合成するパスを追加



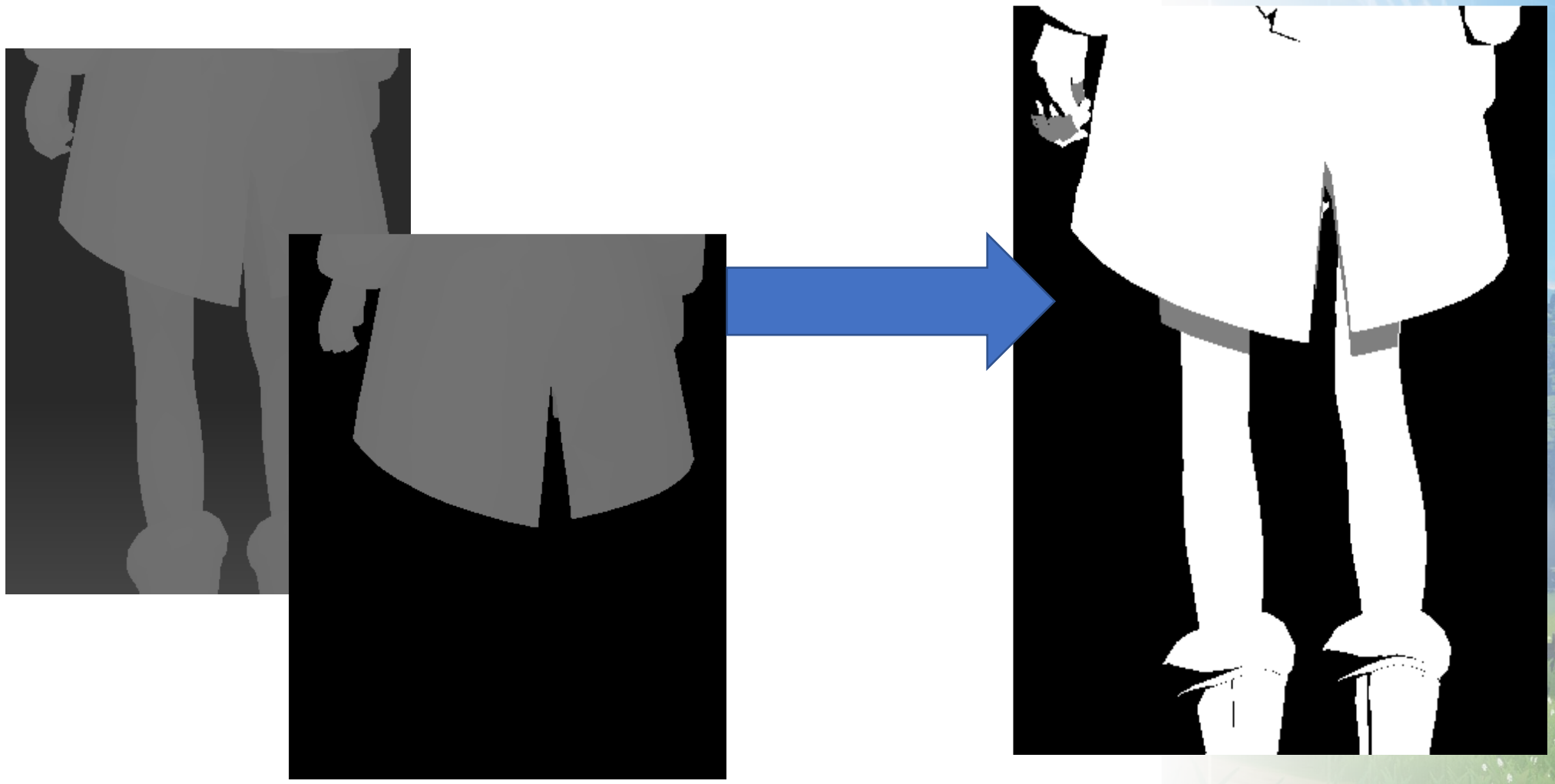
オフセットシャドウ（OffsetShadowDepth）

- 影を落とす対処をデプス描画
  - スクリーンスペースでずらして描画
    - 画面の端でも途切れない



オフセットシャドウ（OffsetShadowMerge）

- デプスを比較してG-BufferのNdotLへ描きこむ



アジェンダ

- イントロダクション
- ライティング
- 輪郭線
- 影
- **ポストプロセス**
- まとめ

ポストプロセス

- ディフュージョン
  - アニメなどでもよく使われるフィルター
  - ブルームでも同じような効果は出せるが別に用意

アジェンダ

- イントロダクション
- ライティング
- 輪郭線
- 影
- ポストプロセス
- **まとめ**

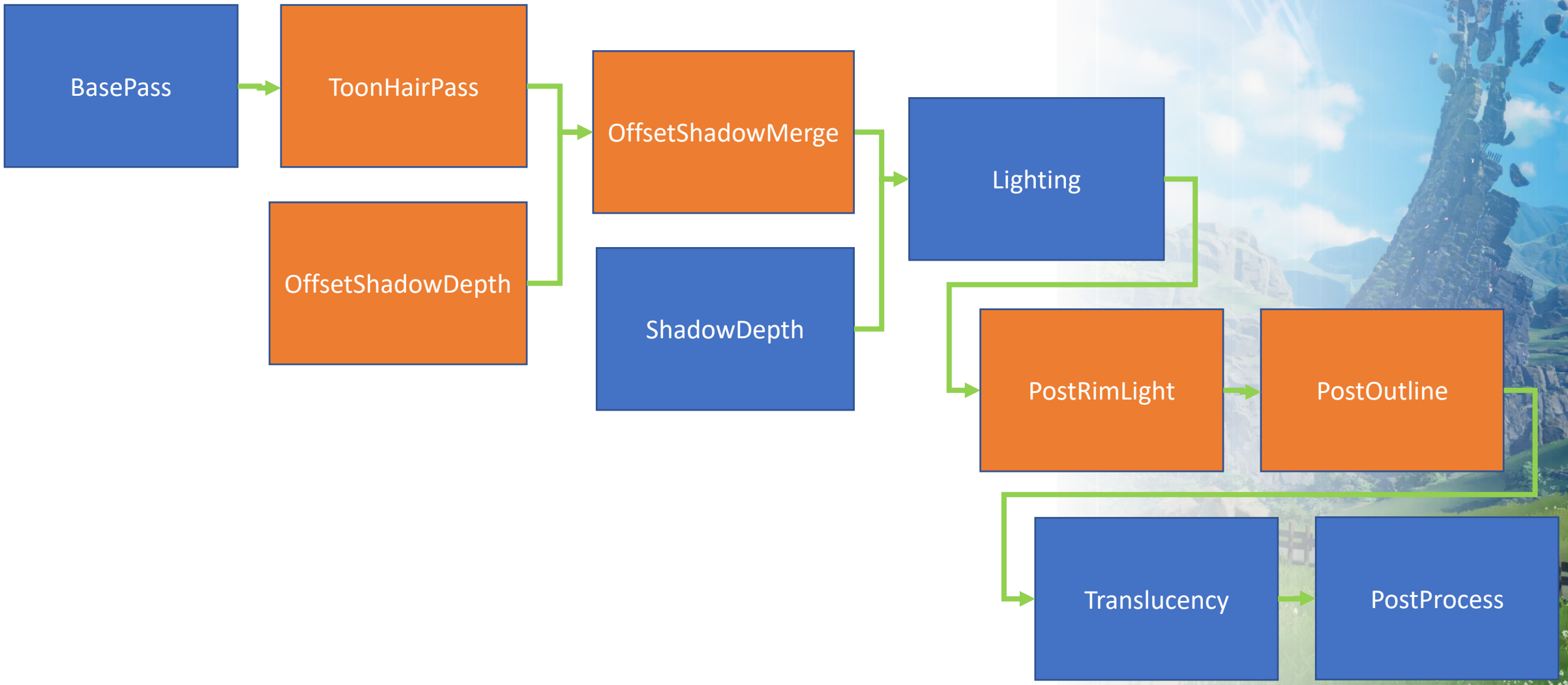
まとめ

- G-Bufferレイアウト

	Red	Green	Blue	Alpha
GBufferA	ComputeNomal.x	ComputeNomal.y	ComputeNomal.z	PerObjectGBufferData
GBufferB	ToonHairMask OffsetShadowMask	SpecularMask	SpecularValue	ShadingModelID SelectiveOutputMask
GBufferC	BaseColor.r	BaseColor.g	BaseColor.b	AmbientOcclusion
GBufferD	ShadowColor.r	ShadowColor.g	ShadowColor.b	NdotL
GBufferE	-	RimLightMask	DiffuseOffset	RimLightWidth
GBufferF	OutlineWidth	OutlineID	OutlinePaint	OutlineZShift

まとめ

- パス


BasePass
ToonHairPass
OffsetShadowDepth
OffsetShadowMerge
ShadowDepth
Lighting
PostRimLight
PostOutline
Translucency
PostProcess

まとめ

- まだ開発中なので変更される可能性はかなり高いです！
- 資料をまとめてみると気になる所が出てきた…
- アニメ表現のためには技術というよりは細かいこだわりが大事
- UE4のエンジンソースを直接いじっても割と何とかなる
  - エンジンのアップデートはしんどい
- レイトレースなど新しい技術で何かできないか？

ご清聴ありがとうございました

1年半運用してわかった！
BLUE PROTOCOLにおける
ビルドパイプラインのクラウド化の壁

11:20～　第1会場

Q & A

時間が余ってしまったので・・・

BLUE PROTOCOLにおける背景の最適化…から1つ

バンダイナムコスタジオ　技術スタジオ　技術スタジオ付
エグゼクティブテクニカルディレクター
大井 隆義

交易都市アステルリーズ

- 開発当初から増改築を繰り返す
- アクター数：約20,000
- メッシュ数：約2,700
- 木はFoliage
- 草はGrass

交易都市アステルリーズ

- 描画スレッドのCPU負荷が深刻
  - カリング
  - Drawコール
- GPU負荷もそれなりに高い
- メモリも結構きつい

DirectX12対応

- Drawコマンドを並列処理できるようになる
  - 描画スレッドのCPU負荷の改善
- かなり改善したもののもう一歩
- DirectX11も無視できない

アクターのマージは？

- 手動でやるのはかなり大変
- アセットが増えるので、メモリを圧迫する
- マージした後にもとに戻せない
  - 今後も増改築が予想される

インスタンス化は？

- カリングやLODが効かなくなる
  - ある程度分割してやるには大変

- インスタンス化した後に元に戻せない
  - 今後も増改築が予想される

HLODは？

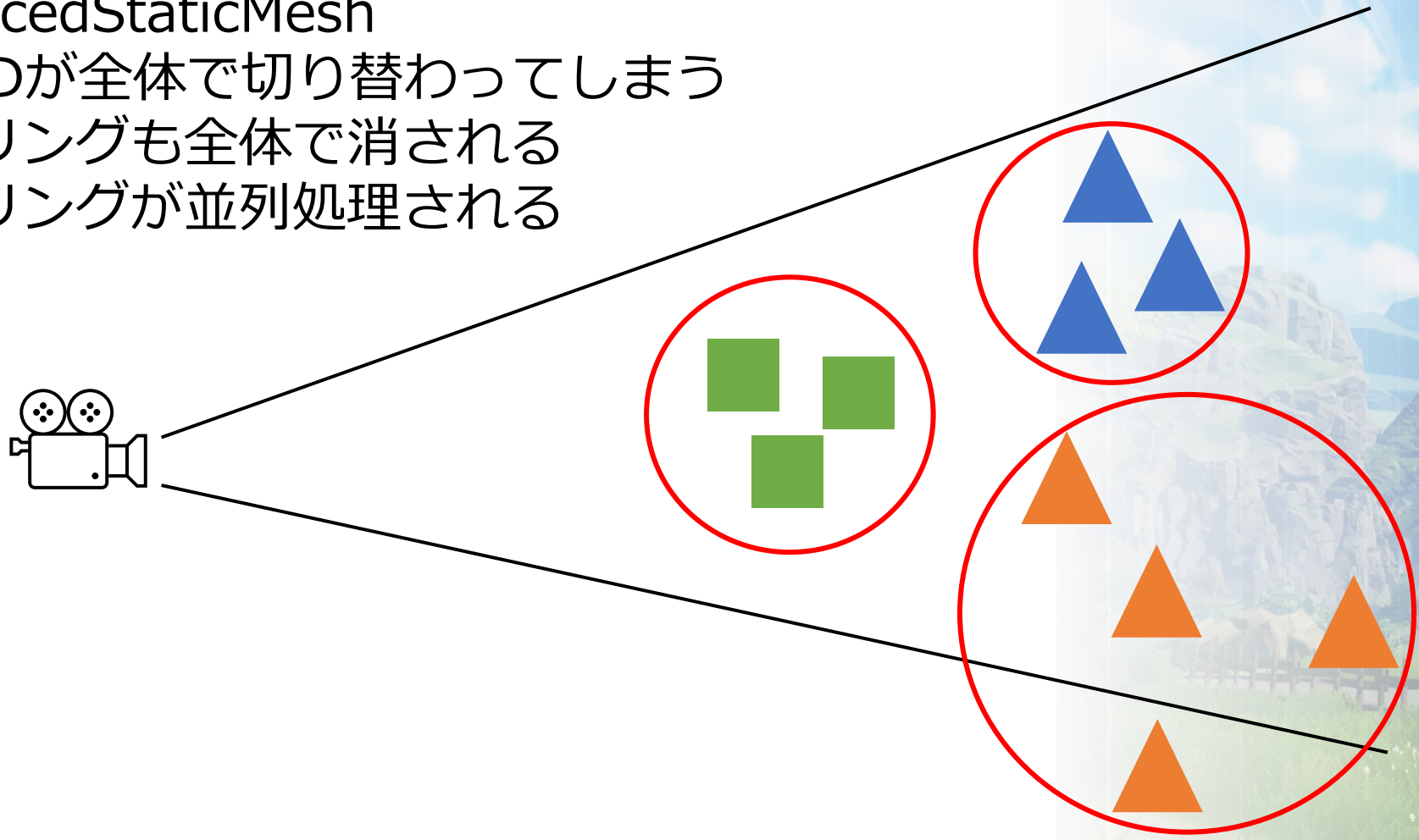
- Hierarchical Level of Detail
- 設定が複雑
- 処理時間が長い
- アセットが増えるので、メモリを圧迫する

インスタンス化ツール

- インスタンス化するツールを作成することに
- ルールベース
- 逆変換も同時に作成

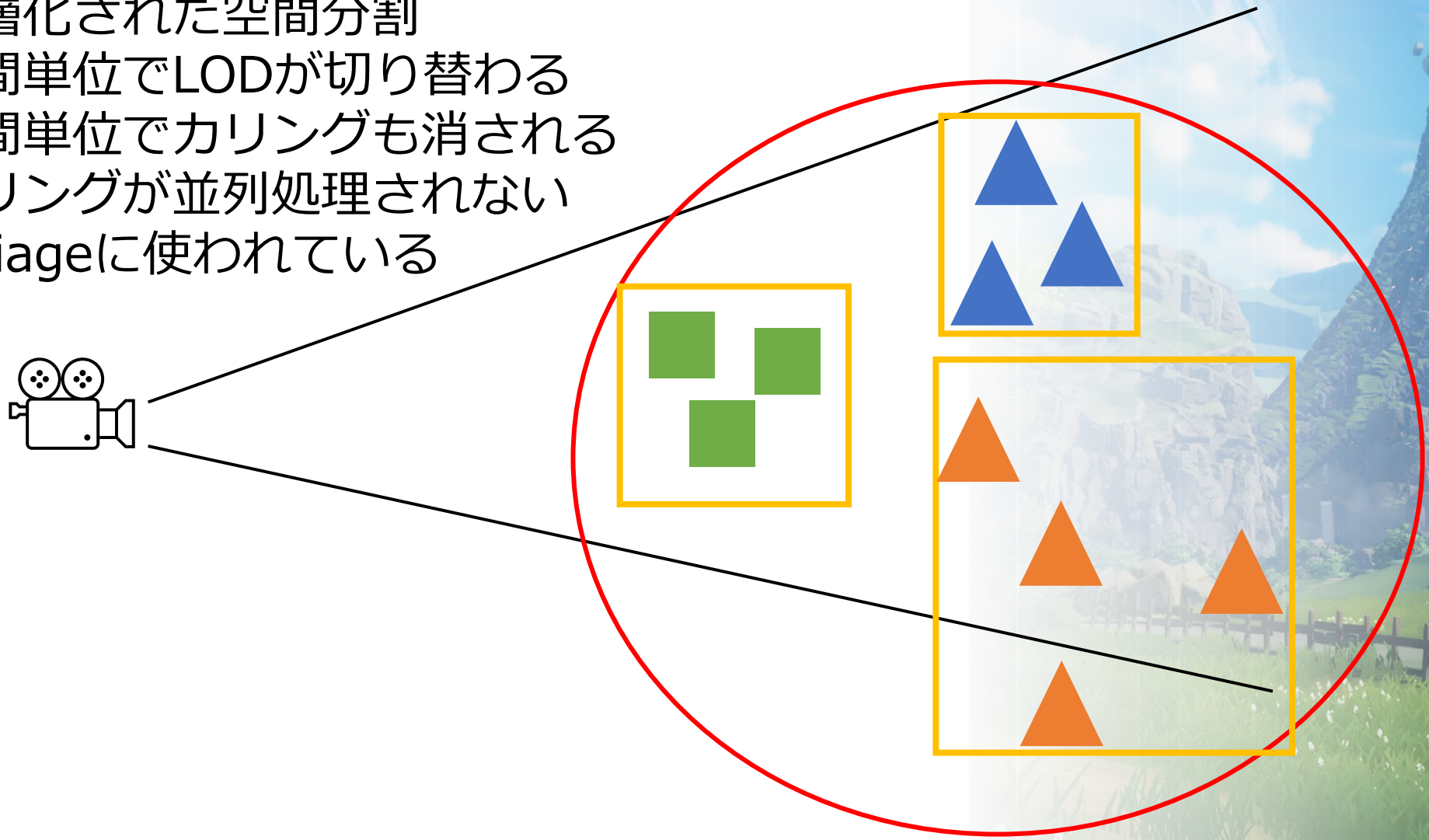
インスタンスの種類

- InstancedStaticMesh
  - LODが全体で切り替わってしまう
  - カリングも全体で消される
  - カリングが並列処理される



インスタンスの種類

- HierarchicalInstancedStaticMesh
  - 階層化された空間分割
  - 空間単位でLODが切り替わる
  - 空間単位でカリングも消される
  - カリングが並列処理されない
  - Foliageに使われている



インスタンス化ツール

①インスタンス化できるものを集める
- 同一レベル
- 同一パラメーター
  - メッシュ（StaticMesh）
  - マテリアル
  - カリング距離
  - 影フラグ
  - ライトマップ解像度
- インスタンス化できないものを省く
  - マイナススケール
  - 半透明
  - 頂点カラーペイント

インスタンス化ツール

② HierarchicalInstancedStaticMesh判定

- 数が多いもの
  - デフォルト128個

- ライトマップ解像度が収まる範囲

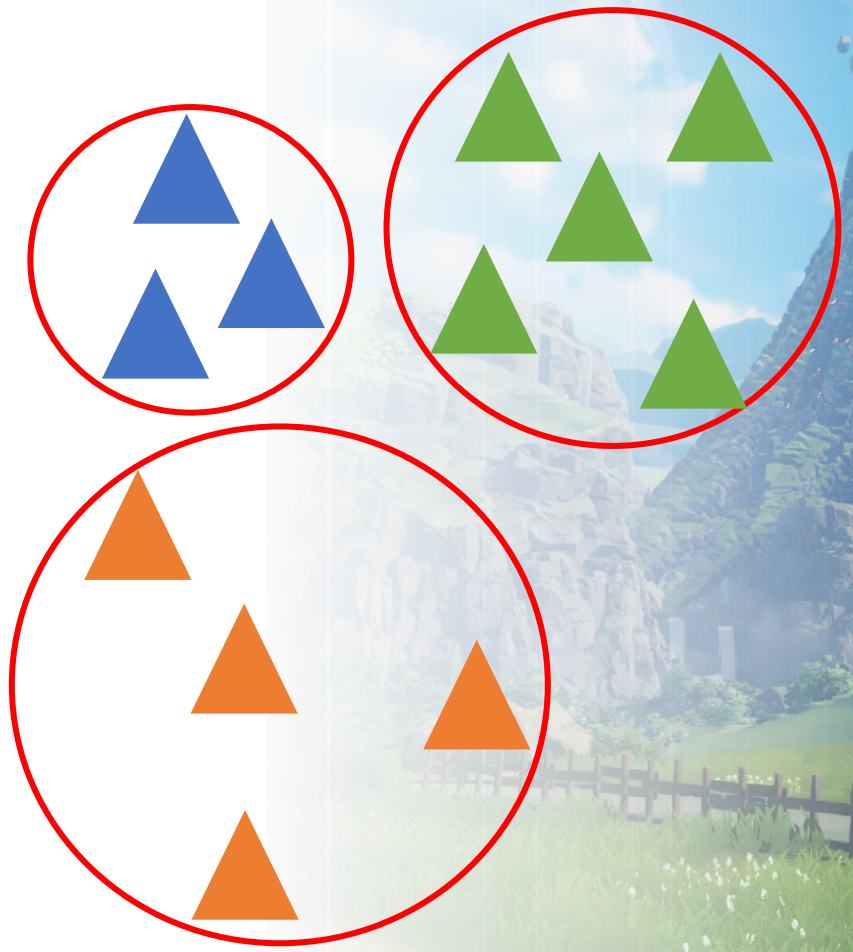
解像度	インスタンス数
64	128
32	512
16	2048
8	8192
4	32768

インスタンス化ツール

③残りは複数のInstancedStaticMeshに割り振る

- 座標でクラスタリング
  - k-means法

- 一定の範囲に収まったらまとめる
  - デフォルト半径2,000



インスタンス化ツール

- パラメーターのコピー
  - コリジョン設定
  - 影フラグ（Cast Shadow）
  - マテリアル
  - カリング距離
  - ライトマップ解像度

- メッシュ単体（StaticMeshActor）に戻せるように対応

交易都市アステルリーズ

- アクター数：約8,000（半分以下に）
- 60FPS出るようになった

Q＆A その2