CyberAgent

CyberAgent Game & Entertainment

統合スタイライズドレンダリングシステムの開発について紹介 ～共通利用ができる基盤を目指して～

株式会社サイバーエージェント

清原 隆行 / 張 煜冰 / 重広 圭輝 / 畳 悠樹

2024.8.23

SNS OK OK

CEDEC2024
Computer Entertainment Developers Conference

UNITY-CHAN! LICENSED UNDER UNITY-CHAN LICENSE

Chapter：00

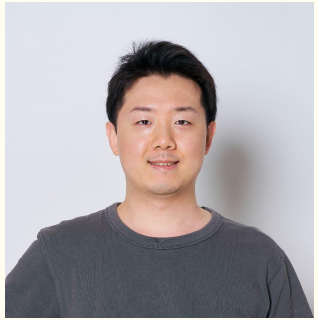
はじめに

講演者紹介

清原 隆行 Takayuki Kiyohara

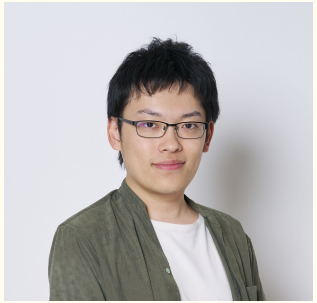
株式会社サイバーエージェント　ゲーム・エンターテイメント事業部のSGEコア技術本部（コアテク）にて、Unityのグラフィックスエンジニアとして　　従事。
３歳児の育児に奮闘しています。

張 煜冰 Zhang Yubing



中国出身で、2015年大卒後来日。5年間　にクライアントエンジニアとしていくつかのゲームの開　発に携わりました。その後サイバーエージェントに転職し、主にグラフィックス技術のR&Dや統合スタイライズドレンダリングシステムの開発に注力しています。

畳 悠樹 Tatami Yuki



新卒でコンシューマゲーム開発会社に入社。6年ほど開発に携わったのち、株式会社サイバーエージェントに中途入社しSGEコア技術本部（コアテク）に配属。
グラフィックスエンジニアとして共　通基盤の開発に従事しています。

重広 圭輝 Shigehiro Yoshiki

大学時代にCGに興味を持ち、Unityで動　作　するInkPainterなどのグラフィックスツールを開発。
2021年に株式会社アプリボットに入社。SGEコア技術本部への開発協力をしつつ、現在は新規プロジェクトでのグラフィックス機能開発を行っています。

CEDEC2024
Computer Entertainment Developers Conference

組織について

ゲーム・エンターテイメント事業部は
株式会社Cygames以外の子会社で構成されており
その**子会社群を総称してSGE**と呼んでいます。

コア技術本部は「**開発効率と品質の向上**」を
ミッションに、各社を横断した役割を担っています。
(コア技術本部を略称でコアテクと呼んでいます)

どんなセッション？

1. SIRIUSについて
2. キャラシェーダー
3. ポストプロセス
4. 最適化機能
5. プロジェクトでの導入のされ方
6. まとめ

Chapter：01

SIRIUSについて

SIRIUSとは？

URPで動作する統合スタイライズドレンダリングシステム

キャラシェーディング、アウトライン描画、ポストプロセスなどを含んだスタイライズドな表現を行える基盤を目指してSGEコア技術本部（コアテク）で開発中のシステムです。
また、コアテクだけではなく、各子会社のエンジニアとも連携して開発を進めています。
定期的に新規バージョンをリリースしており、現在はv0.10.0までリリースされています。（v1.0.0の正式リリースは採用タイトルがリリースされてからを予定）

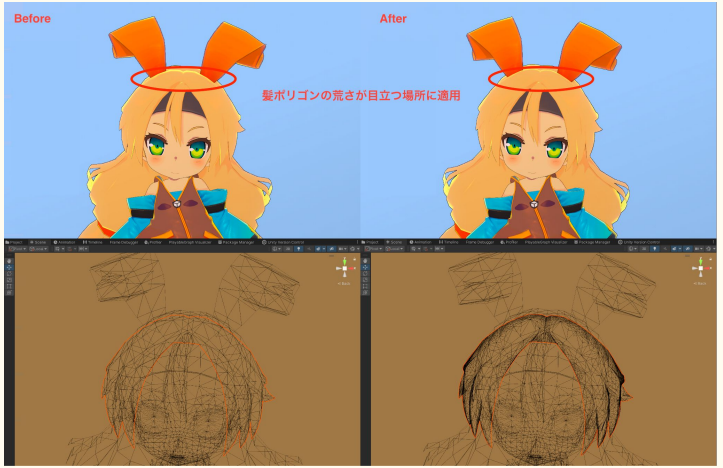
SIRIUSの構成

3つのパッケージに分かれている

- キャラシェーダー
- ポストプロセス
- 背景
    - 背景はまだ本格的に開発が進んでいない

キャラシェーダー

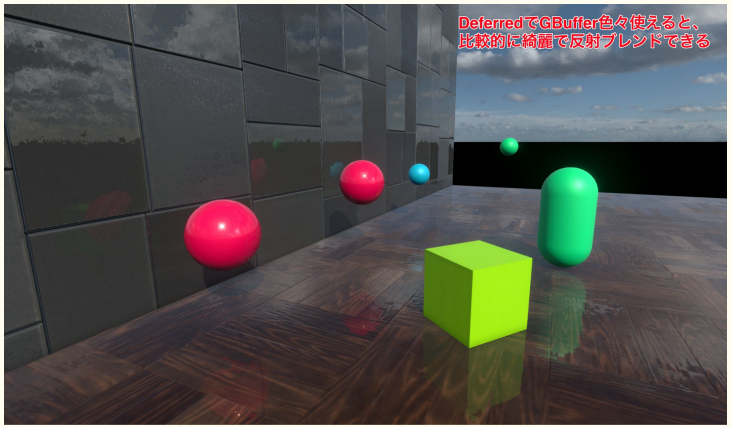
- ベースシェーディング
  - 計算オンリー、Shade Color Map、Ramp map
- スペキュラ反射
  - Standard、PBR、PBR Fast
- Mat Cap
  - Blend、Add
- リムライト
  - Classic Rim、Depth Rim
- 髪ハイライト
  - Hair_Kajiya_Kay、Front Mapping
- 視差マップ
- テッセレーション



ポストプロセス

- GTAO
- Fast Dof
- Bloom
- GodRay
- Hi-Z Screen-Space Reflections
- SSSSS
- Screen Space Volume Light
- など

部分的な利用がしやすいため、色々なプロジェクトで導入してもらっている

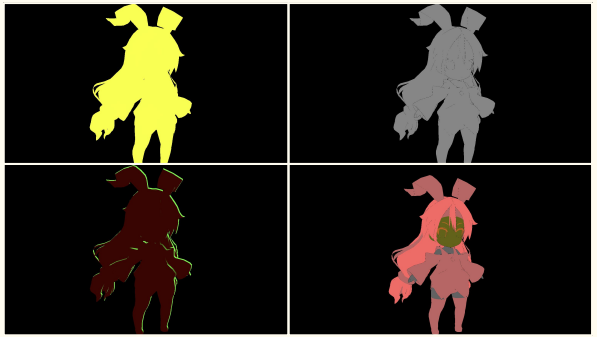


Chapter：02

キャラシェーダー

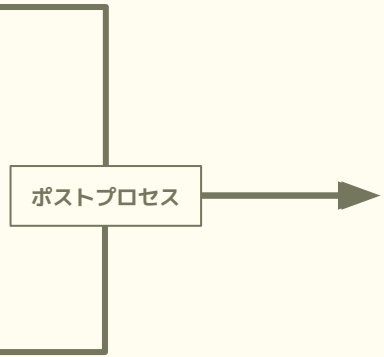
描画パイプライン

SIRIUSではレンダリングパイプラインを拡張し、MRTを用いた独自のGBuffer描画パスを実行します。
GBufferは後のレンダリングパスで実行されるキャラクター描画やポストプロセスで利用しています。



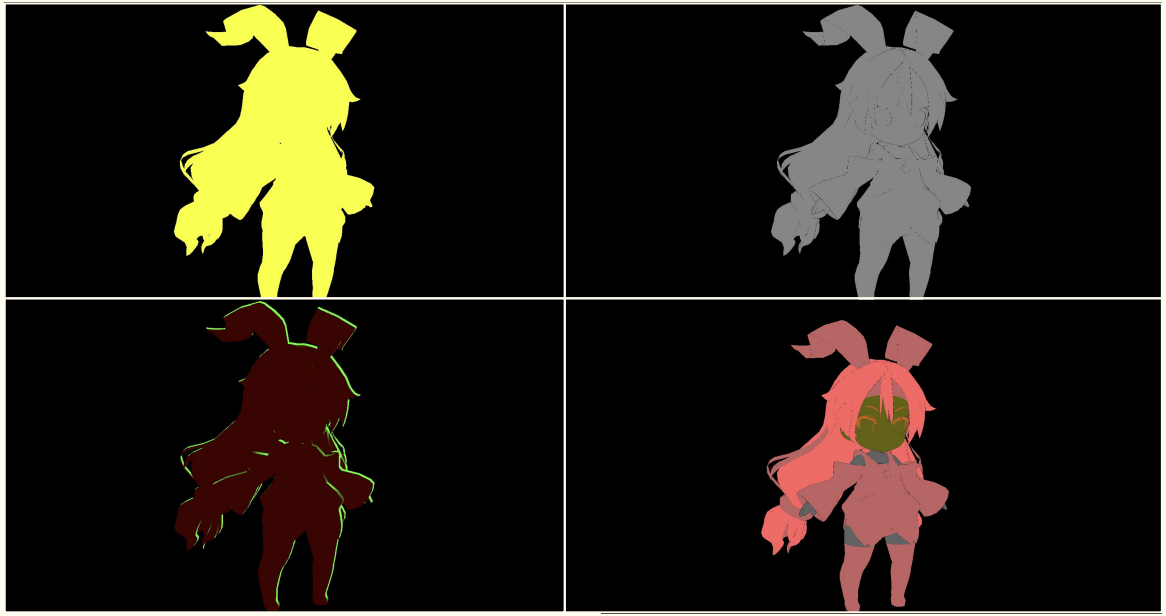
GBuffer Pass

Toon Pass



描画パイプライン

GBufferは有効化した機能に応じて最大で4枚レンダリングされます。
これらの情報は深度比較やリムライトの遮蔽判定、Bloomやアウトラインの領域計算など、様々な場面で活用されます。



シェーダー機能

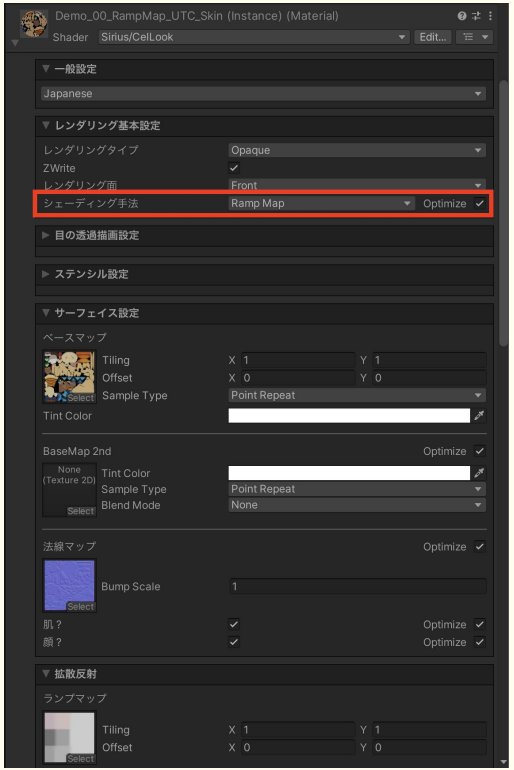
キャラシェーダーでは以下に挙げるような様々な機能が実装されています。
ここでは以下の機能の中から一部を紹介します。

- シェーディング
- スペキュラ反射
- Mat Cap
- リムライト
- 髪ハイライト
- アウトライン
- ハッチング
- 視差マップ
- テッセレーション
- etc...

シェーディング

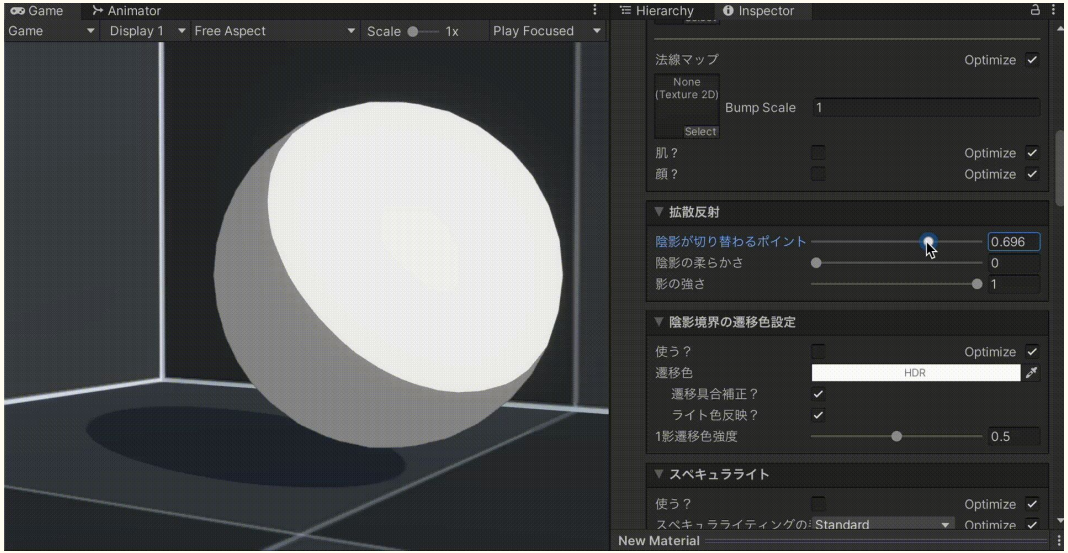
光源を考慮し、物体表面に陰影を付けて立体感や質感をだす設定です。
用途に併せて以下手法が選択可能となっています。

- Calculate Only
  - 陰影の境界や強度に関するパラメータのみを調整して、テクスチャを用いず数値計算のみで拡散反射を行います
- Ramp Map
  - 陰影の付き方を焼き付けたテクスチャ(RampMap)を使用して拡散反射を行います
- Shade Color Map
  - 陰となった際のサーフェイスの陰色を直接指定して拡散反射を行います



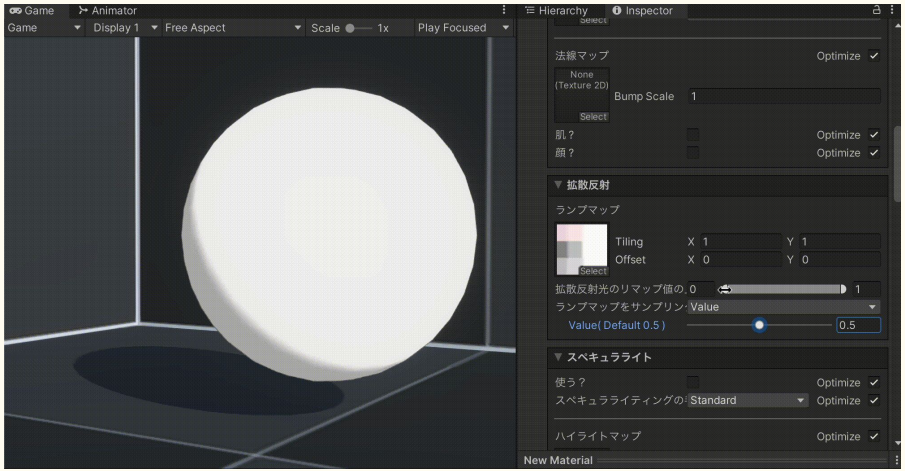
Calculate Only

- 陰影の境界や強度に関するパラメータのみを調整して、テクスチャを用いず数値計算のみで拡散反射を行います
- いわゆる1影のみのシェーディングで事足りる場合に有効です



Ramp Map

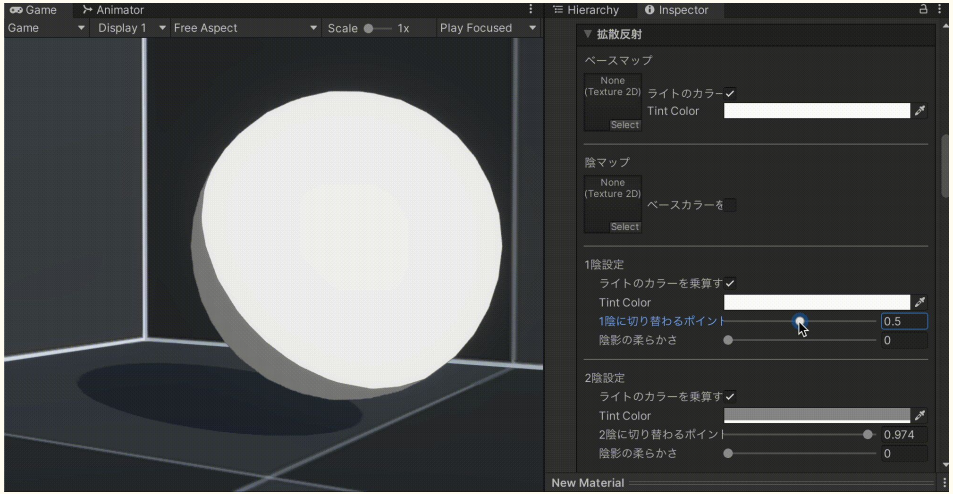
- 陰影の付き方を焼き付けたテクスチャ(RampMap)を使用して拡散反射を行います
- テクスチャの調整次第で何影でも設定可能です
- 陰のボケ具合等もテクスチャに依存します



サンプルのRampMap

Shade Color Map

- 陰となった際のサーフェイスの陰色を直接指定して拡散反射を行います
- 陰色にはカラーだけでなくテクスチャの指定が可能で、部分毎に細かく制御できます
- 陰の指定は2影まで可能です



スペキュラ反射（ハイライト）

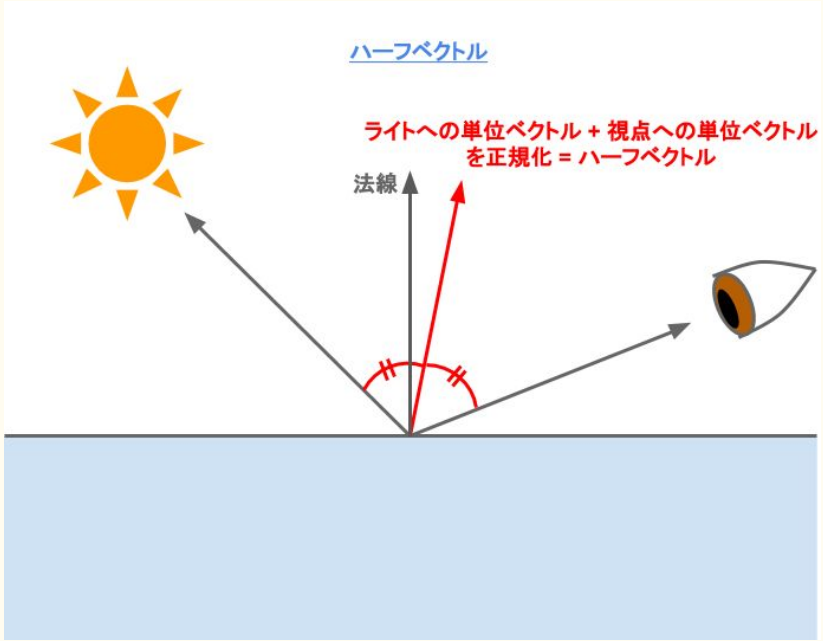
物体の光沢を表現する設定です。

用途に併せて以下手法が選択可能です

- Standard
  - 標準的なスペキュラライトの計算を行います
  - NPRな計算でハイライトの周囲に光が溢れやすい特徴があります
- PBR GGX
  - 物理ベースのスペキュラ計算で、マイクロファセット理論に基づく実装となっています
- PBR GGX FAST
  - PBR GGXの計算を簡略化して高速化したものです

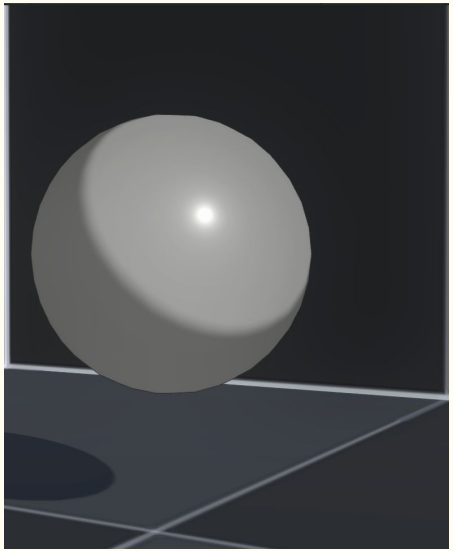
スペキュラ反射（ハイライト）

StandardではSchlickの近似式を用いた鏡面反射モデルの計算を少し改変したものを利用しています。
アルゴリズム的に周囲に光が溢れやすく、ハイライトが広がりやすいという特徴があります。



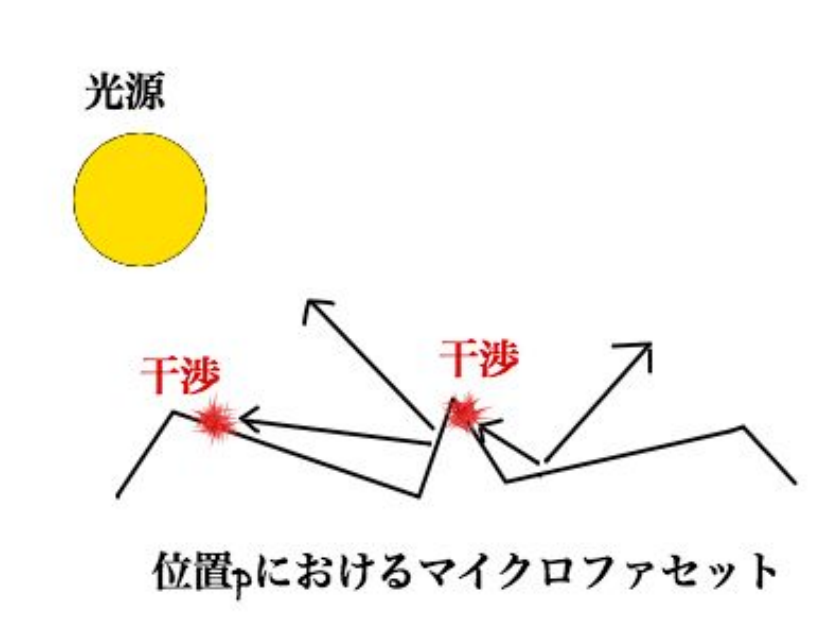
参照:https://blog.applibot.co.jp/2017/10/10/tutorial-for-unity-3d-6/

Standerd

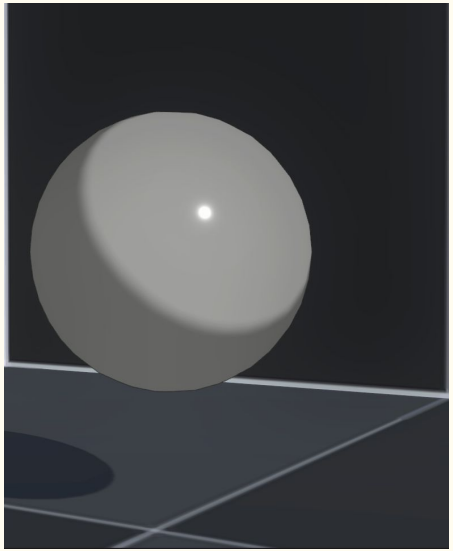


スペキュラ反射（ハイライト）

PBR GGXはマテリアルの表面の微細な凹凸や表面の粗さを考慮した光の反射をシミュレートします。
マイクロファセット理論を基にしたBRDFモデルであり、特に粗い表面や金属のような材質のレンダリングに効果的です。



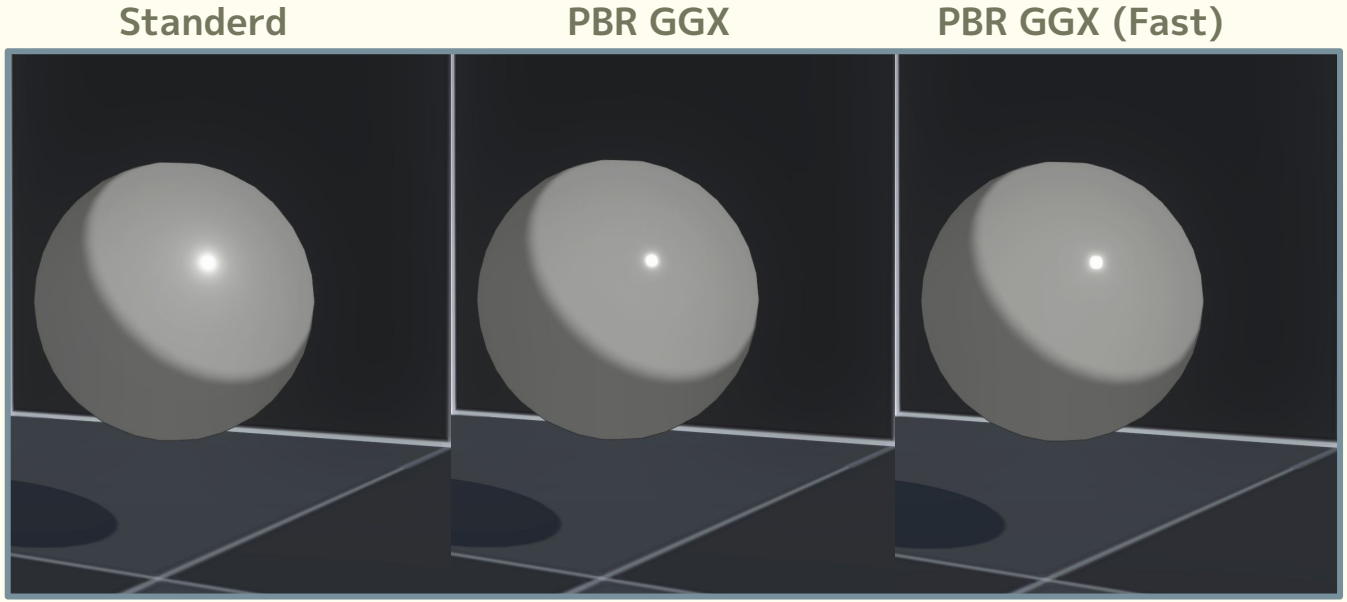
PBR GGX



スペキュラ反射（ハイライト）

以下画像は各手法のハイライトを設定したサンプルです。

光の溢れ方に差異があるので、表現に応じて使い分ける事を推奨しています。



Mat Cap

素材感と光を表現した球体図を使ってライティングをシミュレートします。

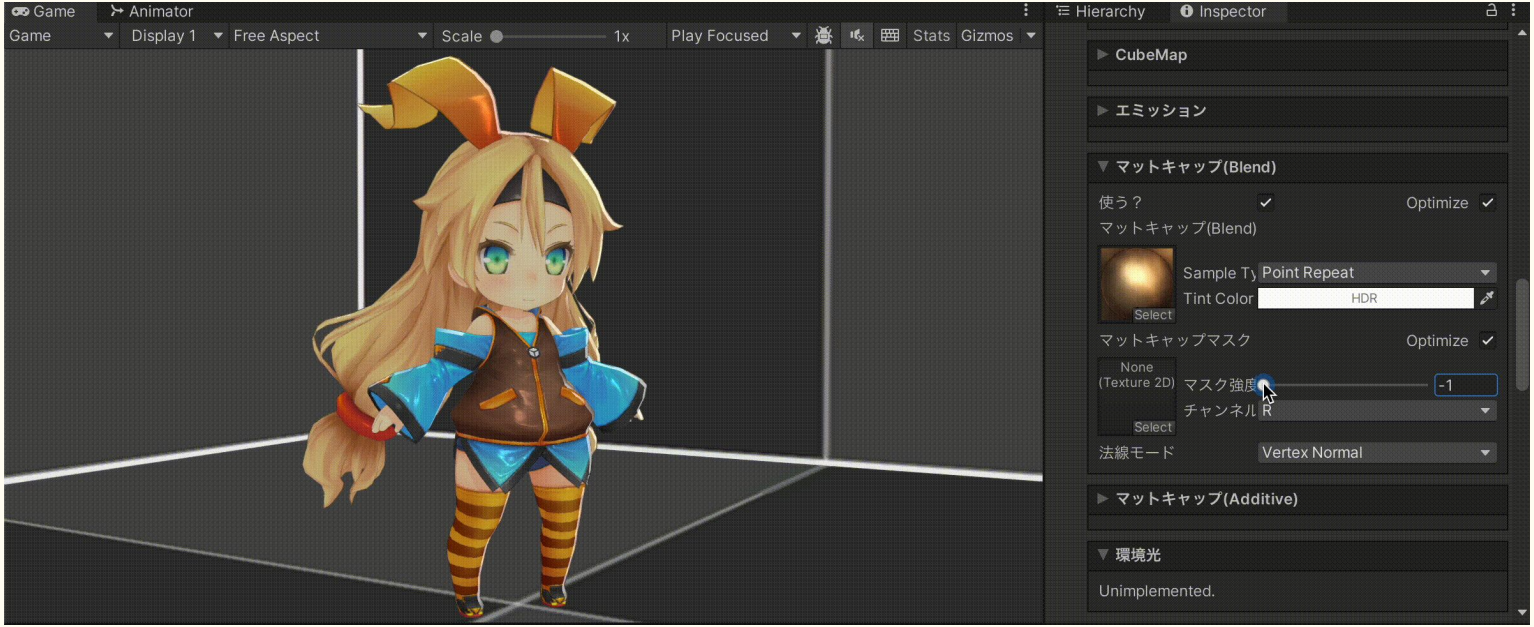
用途に併せて以下手法が選択可能です

- Blend
  - ライティング結果とMat Capの結果を補間合成します
- Additive
  - ライティング結果とMat Capの結果を加算合成します

Mat Cap

Blendモードはシェーディング処理の結果とMat Capを補間合成します。

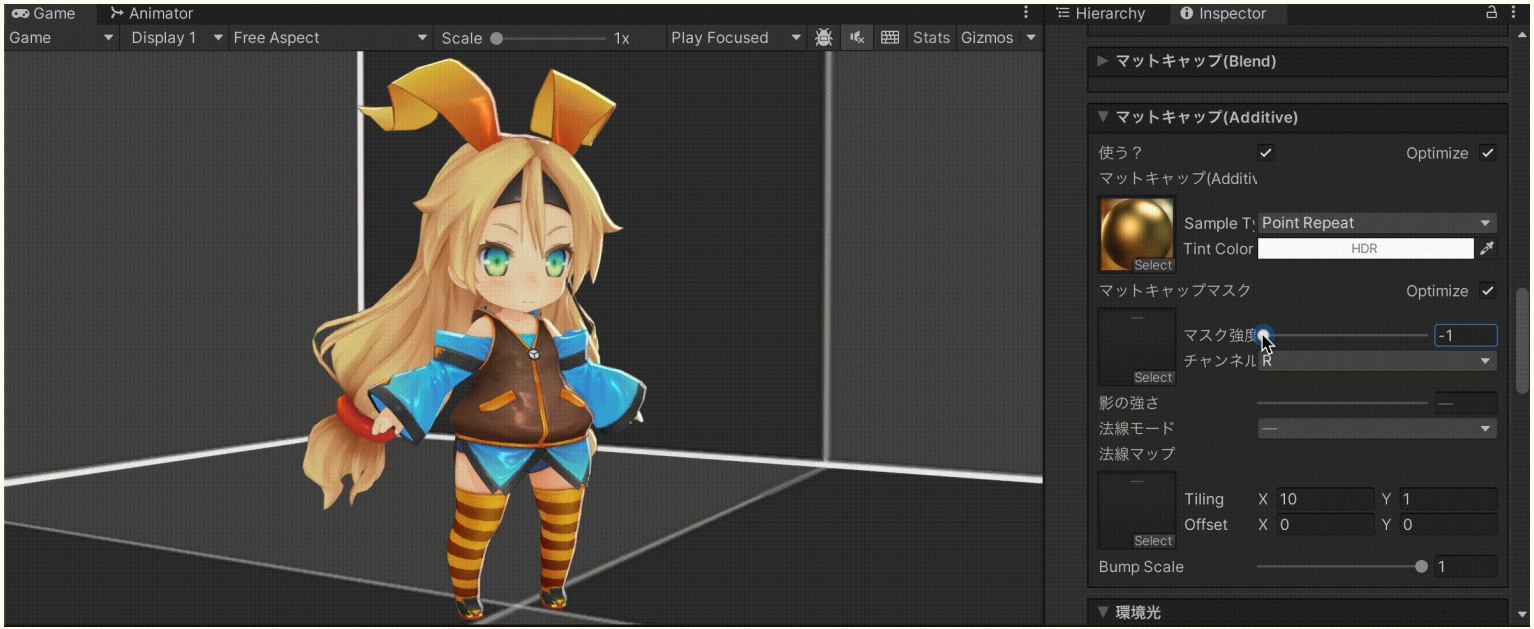
擬似的な光沢表現を行いたい場合に便利です。



Mat Cap

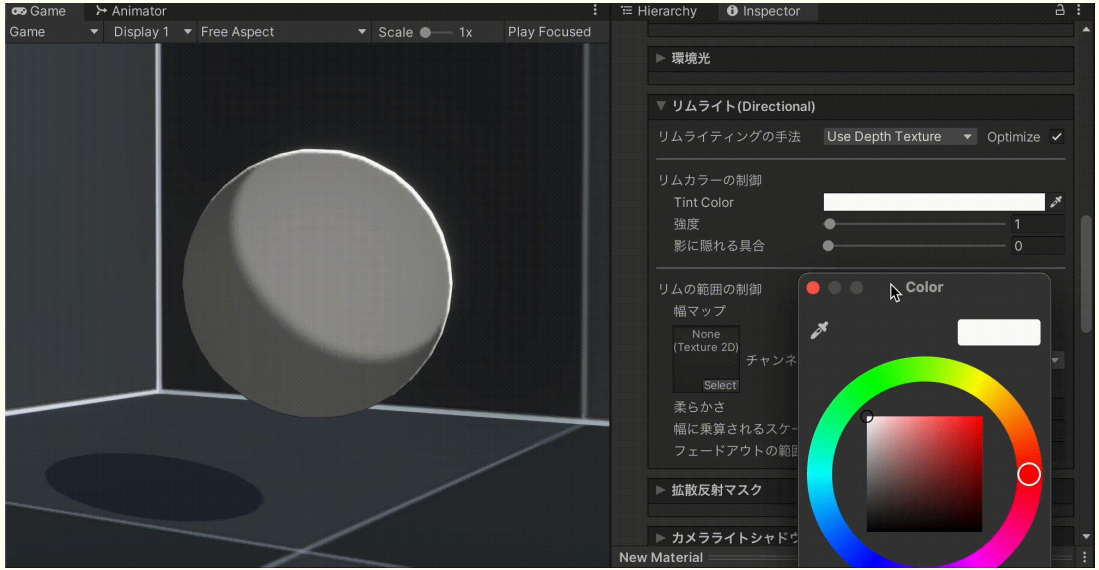
Additiveモードではライティング結果とMat Capの結果を加算合成します。

擬似的なハイライトやリムの表現として使う際などに便利です。



リムライト

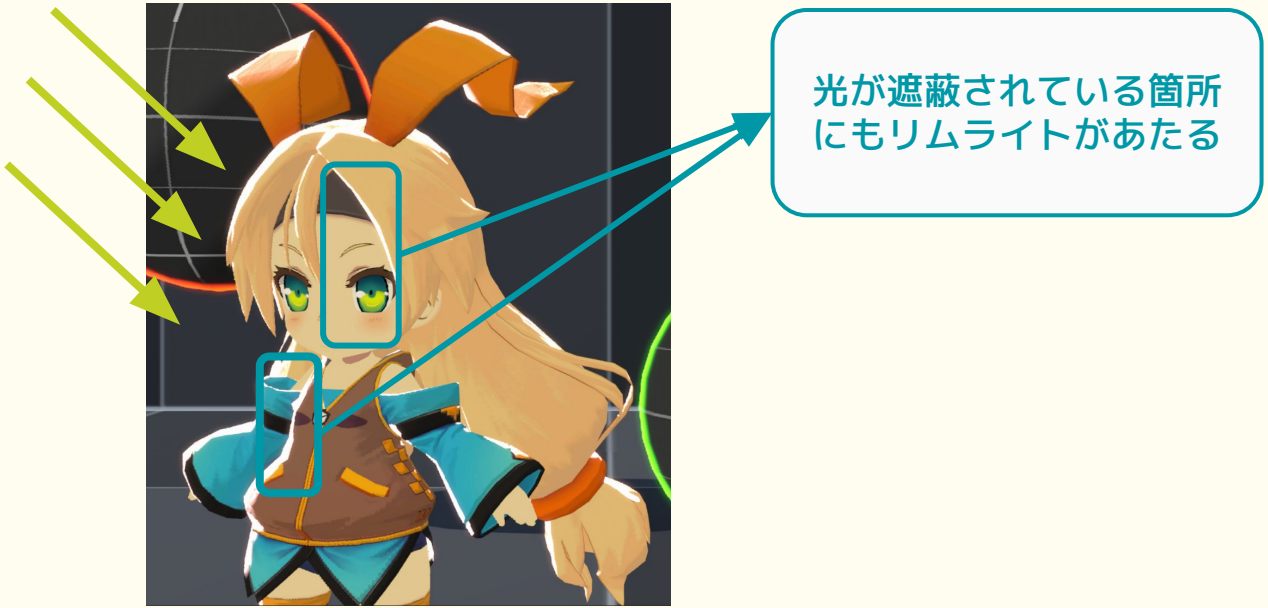
オブジェクトのエッジを照らす機能です。指向性のライト方向に影響し、カラーや強度、適用範囲などをパラメータから調整可能にしています。



リムライト

通常のリムライトでは物理的に光が遮蔽されている箇所にも効果が乗ってしまいます。

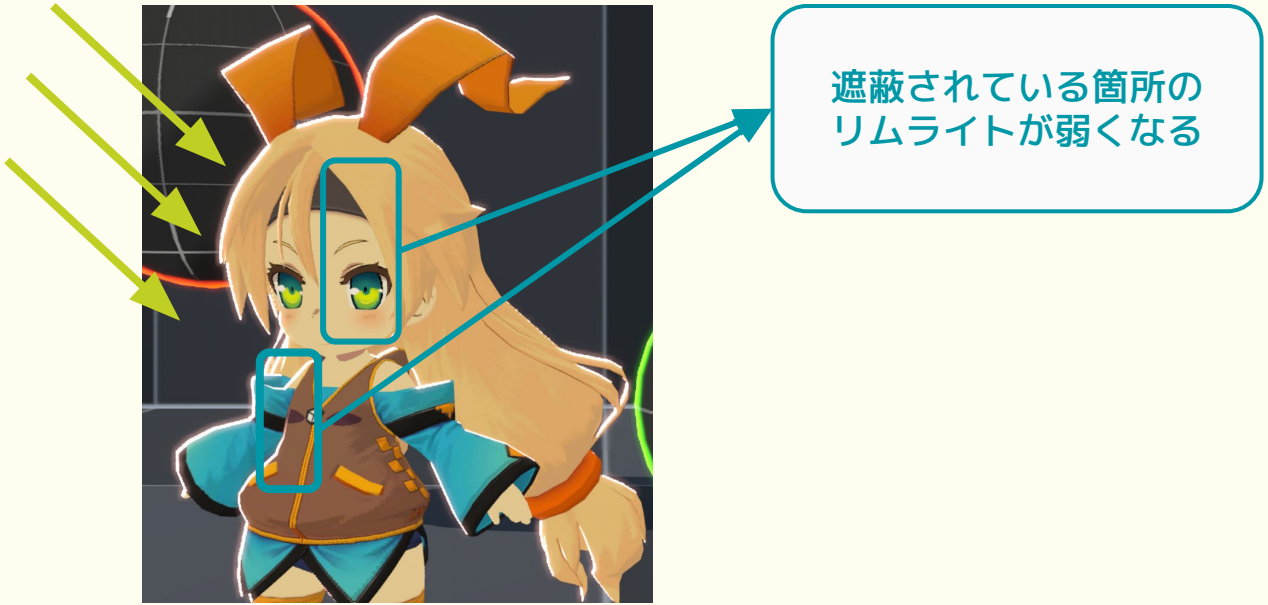
（画像では奥側から手前にライトを設定しています）



リムライト

SIRIUSのリムライトではライトやGBufferの情報を考慮し、遮蔽されている箇所への影響を抑える機能なども実装しています。

（画像では奥側から手前にライトを設定しています）



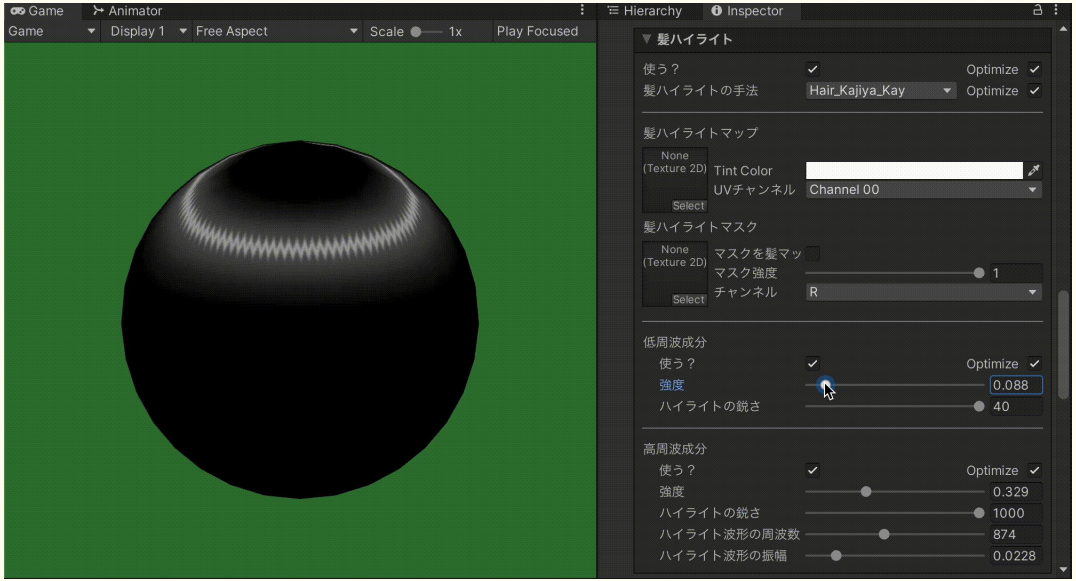
髪ハイライト

髪専用のハイライト設定です。
用途に合わせて以下の設定からハイライトの手法を選択可能です。

- Kajiya-Kay
- Front Mapping

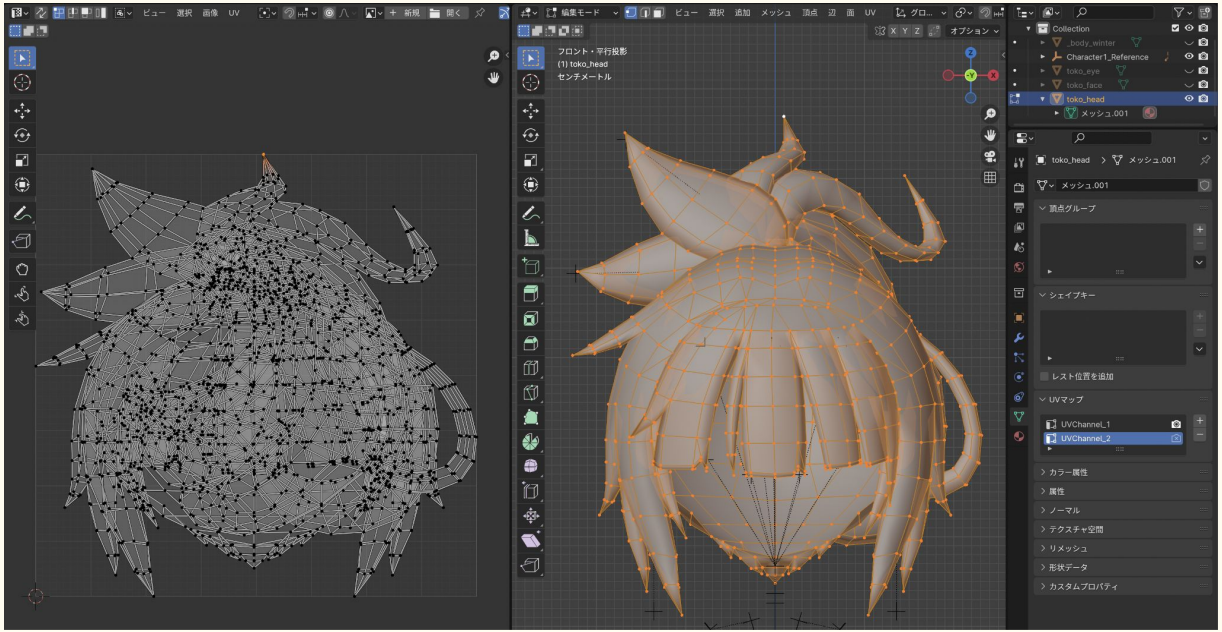
髪ハイライト

Kajiya-Kayのハイライト設定では、髪のハイライトの強い部分(高周波成分)と拡散して弱い光が漏れ出る部分(低周波成分)の2つに分けてパラメータを設定する事が可能です。



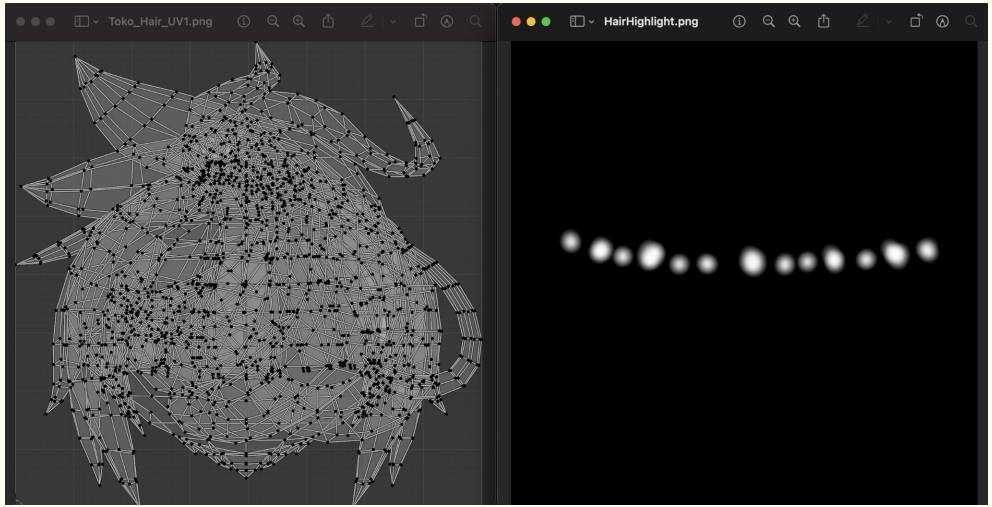
髪ハイライト

Front　Mappingの設定では、モデルの適当なUVを利用してハイライト画像を正面から転写するようなアルゴリズムでハイライトを表現します。(画像はBlenderでFront Mapping用のUV設定を行ったものです)



髪ハイライト

UVの設定に合わせて、画像右のようなハイライトテクスチャを作成して設定すると画像のような形でハイライトを表現する事ができ、ある程度形状を制御したハイライト表現を行いたい場合に便利です。

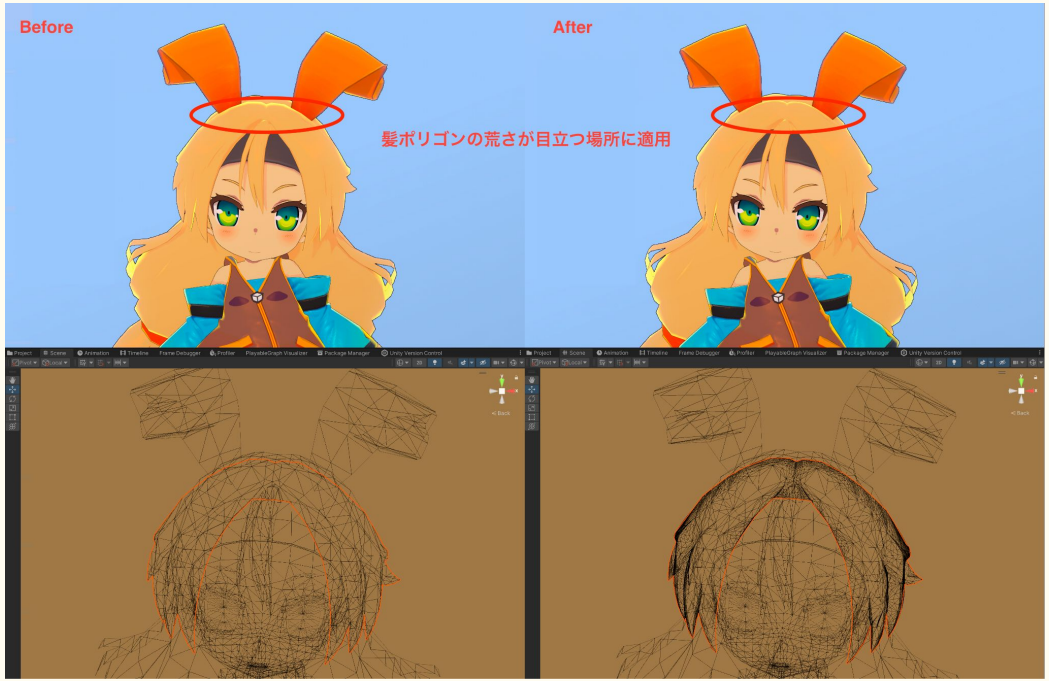


視差マップ

視差遮蔽マッピングを行うオプションです。凹凸を持たないオブジェクトのテクスチャに高さ情報を与える事でオブジェクト表面の凹凸表現を可能にします。
SIRIUSではこの計算でレイトレースに二分探索法を用いることで軽量化を行っています。

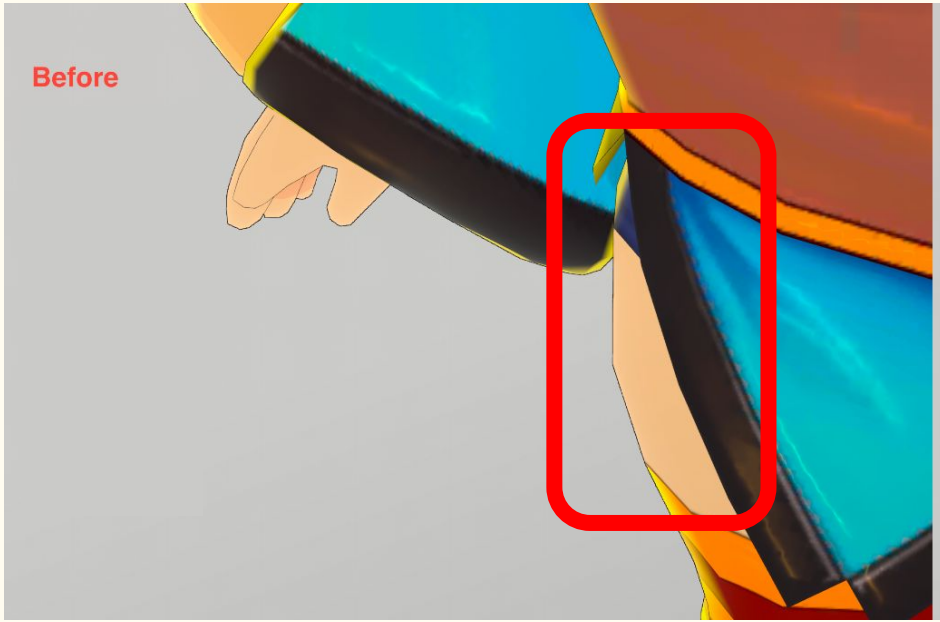
テッセレーション

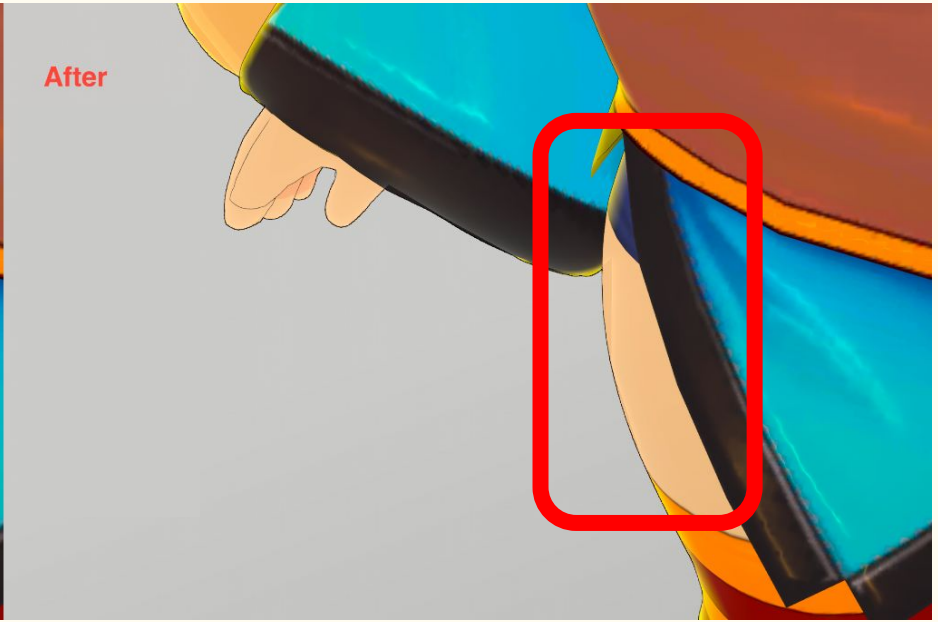
ランタイムで頂点を細分化し、モデルの形状を滑らかにする機能です。動画のレンダリングなど特殊な状況でモデルを精細に表現したい場合等での利用を推奨しています。



テッセレーション

テッセレーション適用例です。画像の赤枠の部分のように、頂点分割後のエッジを自然に膨張させる事で、ポリゴンの品質をランタイムで向上させる事が可能です。





キャラシェーダーまとめ

- SIRIUSが提供するキャラクターのレンダリングに関する一部の機能を紹介しました
- 様々な機能を搭載しており、プロダクト最初期の絵作りで様々な表現を素早くテストする事が可能です。
- 機能過多による負荷の問題は最適化機能によって解消されます
(後述する最適化機能の部分で説明します)

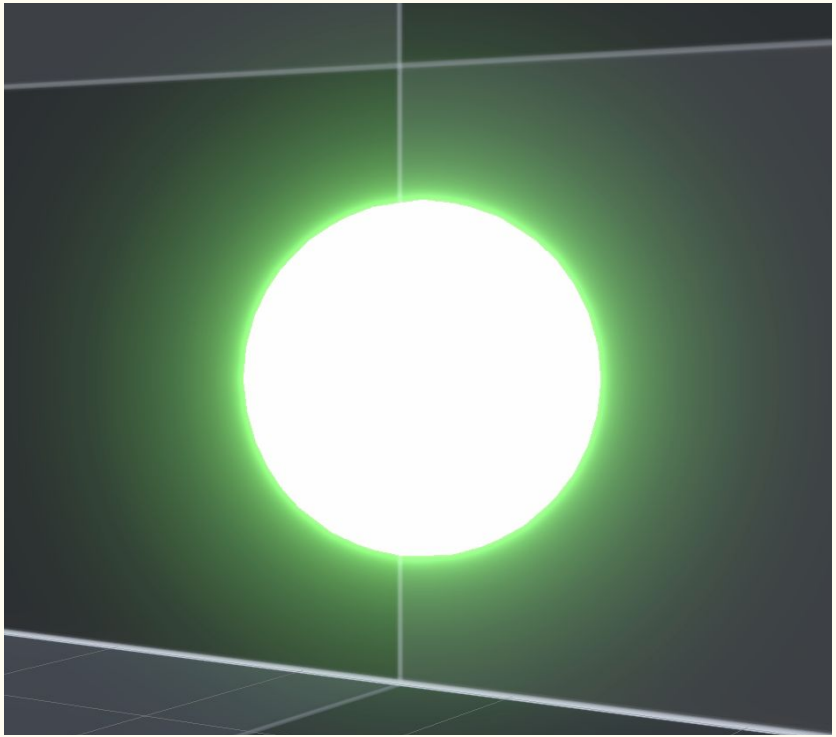
Chapter：03

ポストプロセス

ポストプロセス

- **Bloom＊**
- **Screen Space GodRay＊**
- FastDof
- Screen Space Volume Light
- Sky Diffusion
- HiZ Screen Space Reflection(HiZ SSR)
- Ground Truth Ambient Occlusion (GTAO)
- Screen Space Subsurface Scatter (SSSSS)
- など...

Bloom-概要



URP Bloomの改善

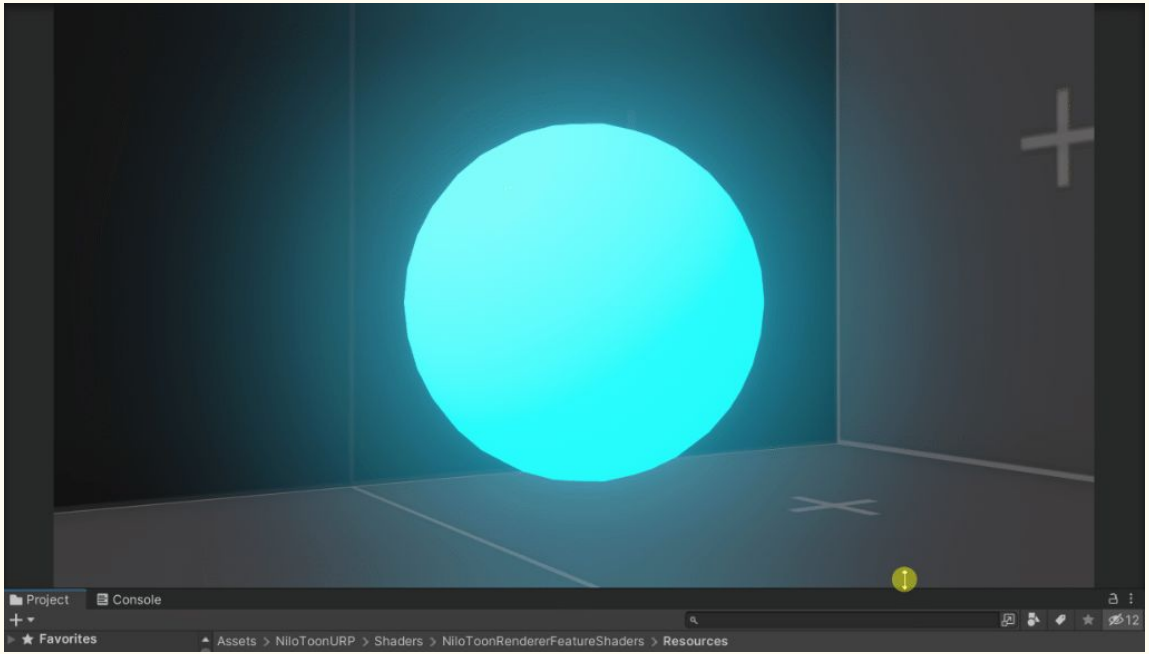
- 拡散範囲の解像度依存問題解消
- ブラー処理の負荷を軽減

機能拡張

- 区域別Bloom効果設定
- 暗部Bloom

Bloom-解像度依存問題解消

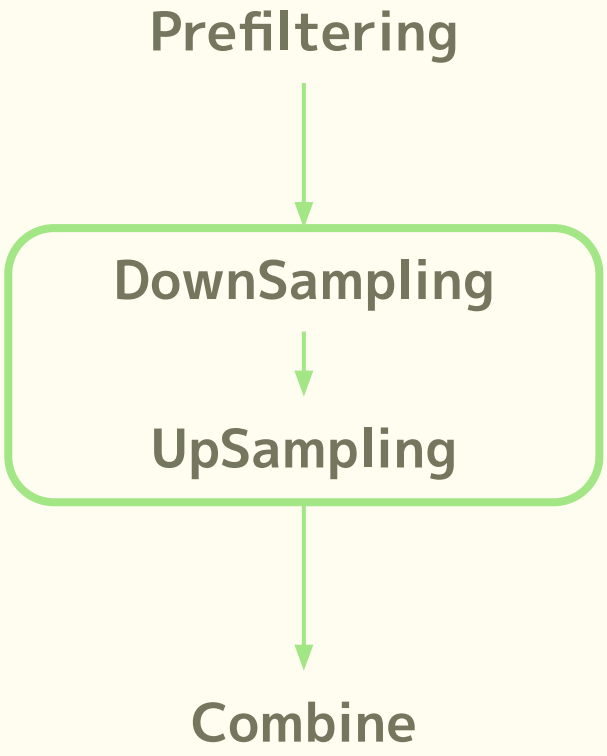
URPのBloomには拡散具合が解像度に依存してしまう問題がある



解像度が2の累乗（例：1024や512）の境目を超えて変化する際、拡散範囲が大きく変化してしまいます。

また、解像度の高さに伴って処理負荷も高くなります。

Bloom-解像度依存問題解消



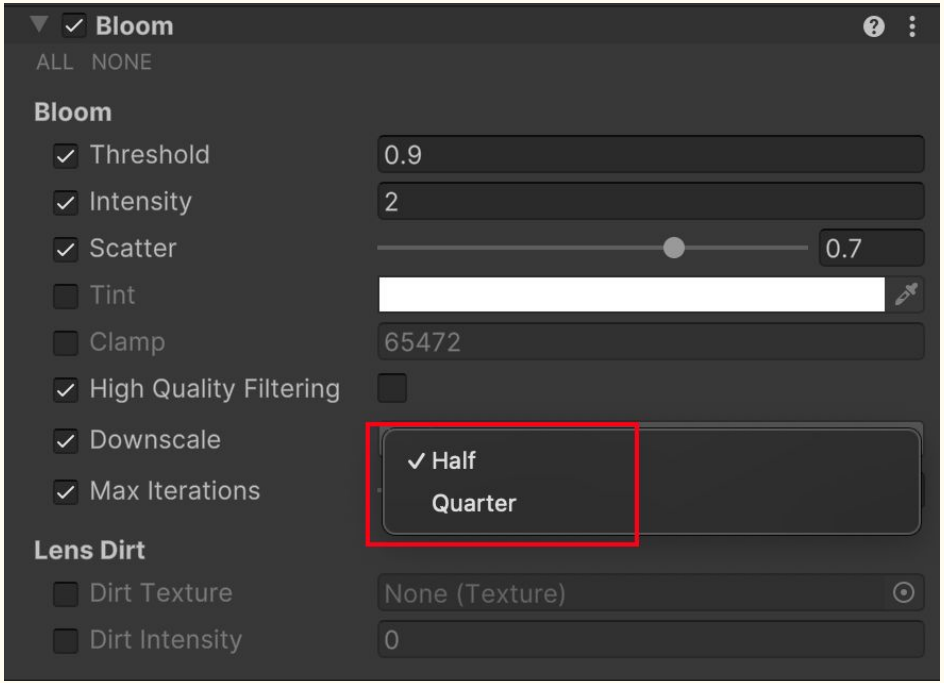
BloomBufferにCameraColorから
輝度抽出して書き込む

DualFilteringの手法で
BloomBufferにブラーをかける

BloomBufferをCameraTargetに
Combine

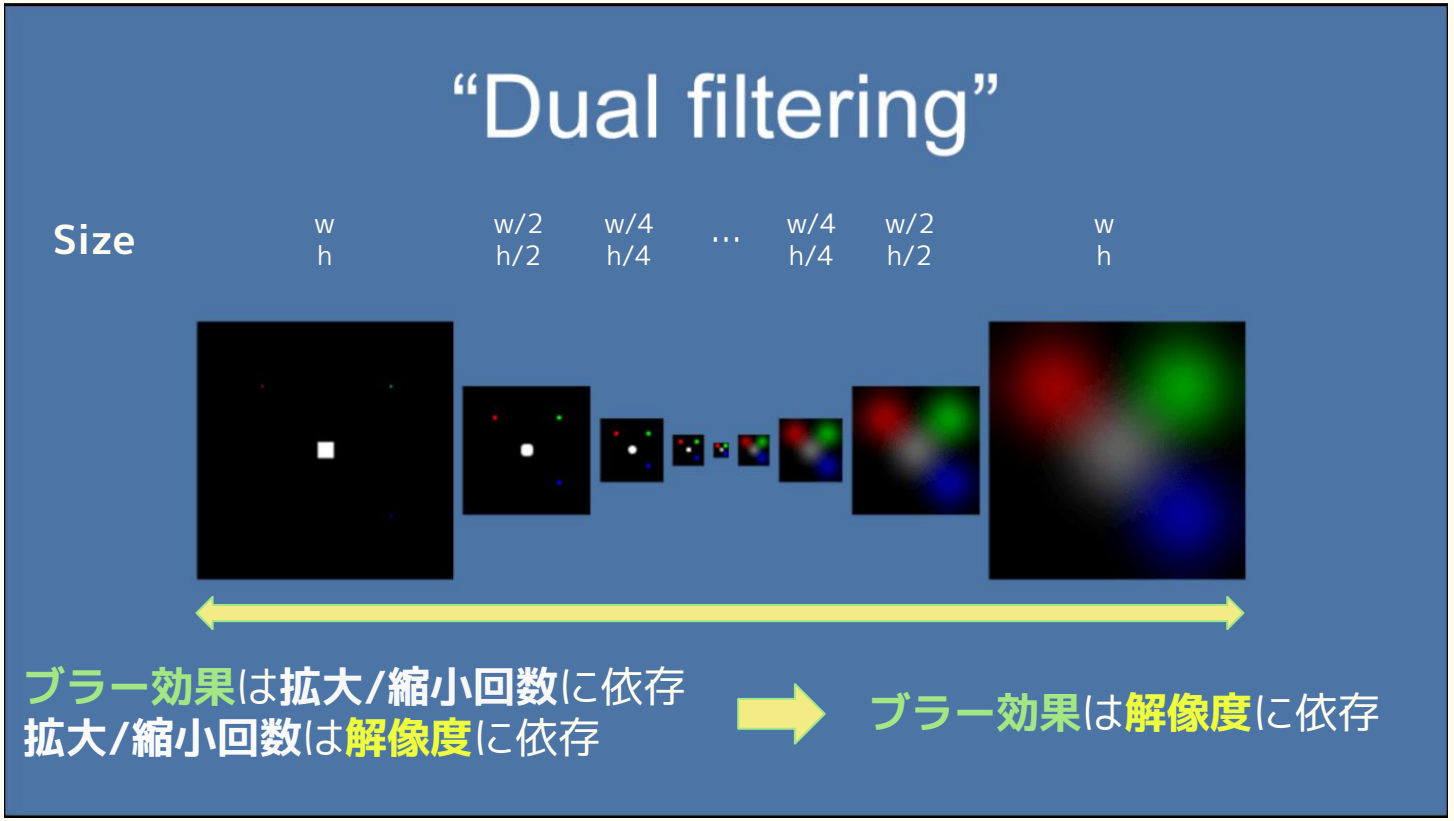
Bloom-解像度依存問題解消

URPのBloomBufferサイズ



BloomBufferのサイズはDownscaleを選ぶ形式で、カメラの描画解像度に依存してしまいます。

Bloom-解像度依存問題解消

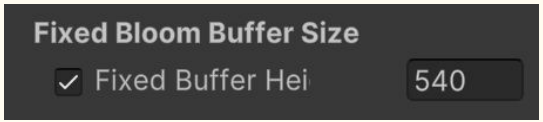


参考：SIGGRAPH 2015 - Bandwidth-efficient graphics

BloomBufferのサイズを割合ではなく、固定値にする

URP

Sirius



URPのBloom処理負荷は少し高がった

Prefiltering : Bilinear Filter (**高品質時 13tap Gaussian Filter**)

DownSampling : **9tap横方向Gaussian Filter** + **5tap縦方向Gaussian Filter**

UpSampling : Bilinear Filter (**高品質時 Bicubic Filter**)

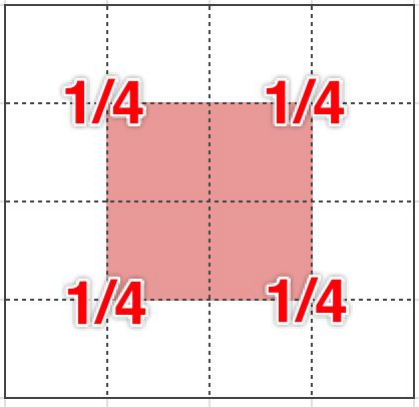
ブラーの品質はいいものの、サンプリング回数が結構多いため、負荷が高いです。
そして、解像度が高くなるほど、前述の縮小拡大処理の回数も増えるので、処理負荷がさらに増大します。

Siriusは負荷軽減のため、以下のFilterに変更しました

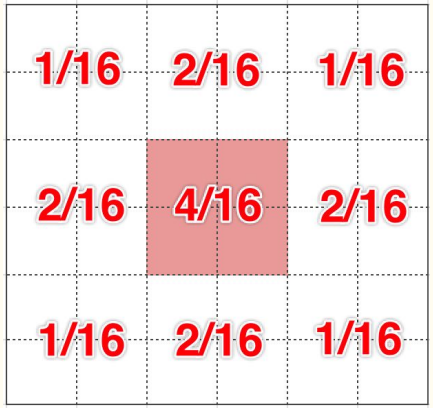
Prefiltering : Bilinear Filter (高品質時 13tap Gaussian Filter)

DownSampling : **4tap Box Filter（高品質時 9tap Tent Filter）**

UpSampling : **4tap Box Filter（高品質時 9tap Tent Filter）**

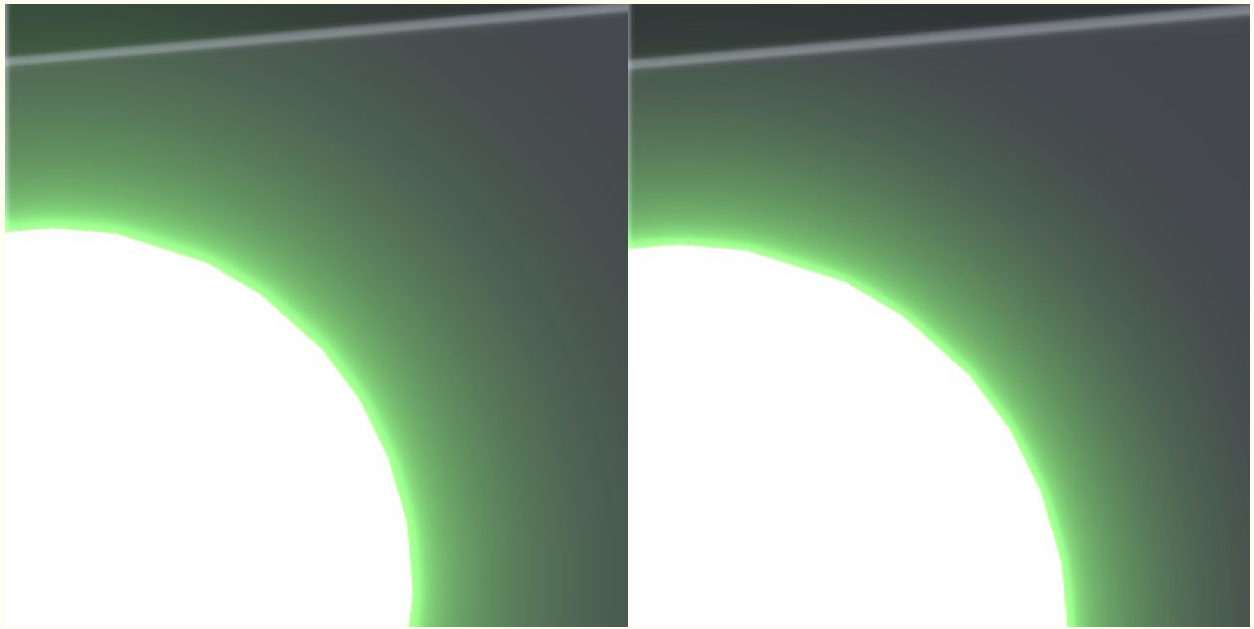


4tap Box Filter



9tap Tent Filter

Bloom効果の比較



URP　　Sirius

効果はほぼ同じ

Pixel4aとiPhone12で
性能テストした結果：

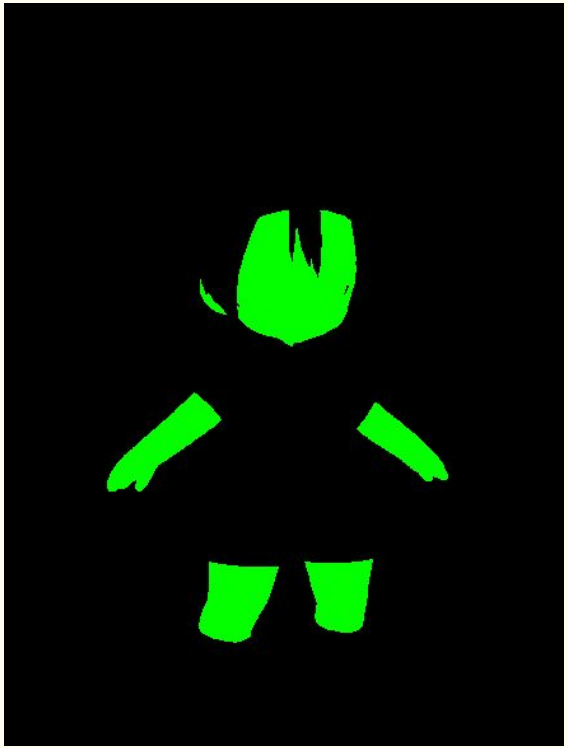
平均2FPSほど上昇

Bloom-区域別設定

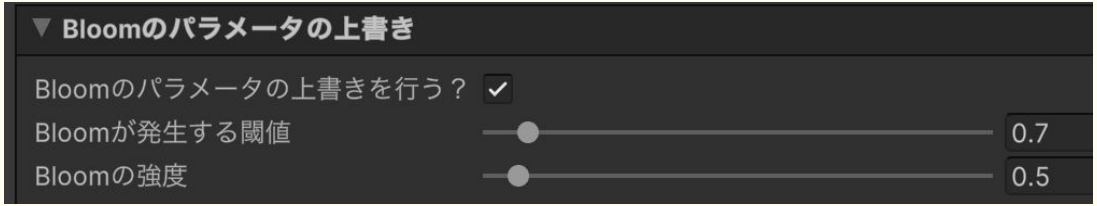
URP Bloom

Sirius Bloom

Bloom-区域別設定



Bloom Override Mask



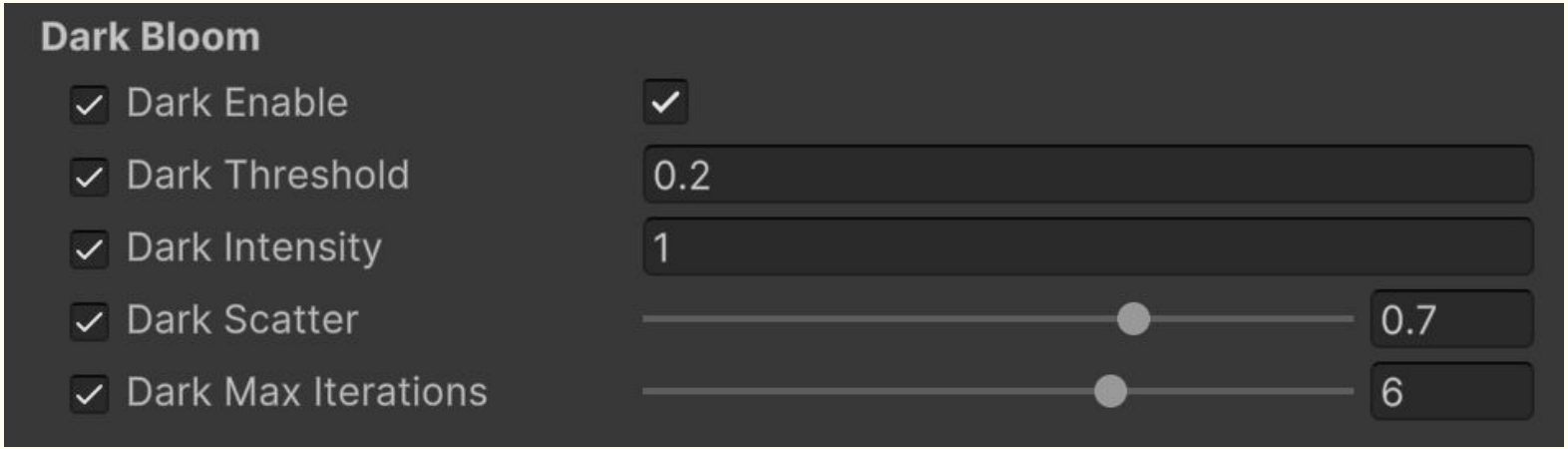
キャラマテリアルごとにBloomパラメータを設定し、GBufferに書き込む。

Bloom処理する際にGBufferの値をチェックし、有効な値が書き込まれていれば、Volumeに設定された値の代わりに使う。

Bloom-区域別設定

URP Bloom

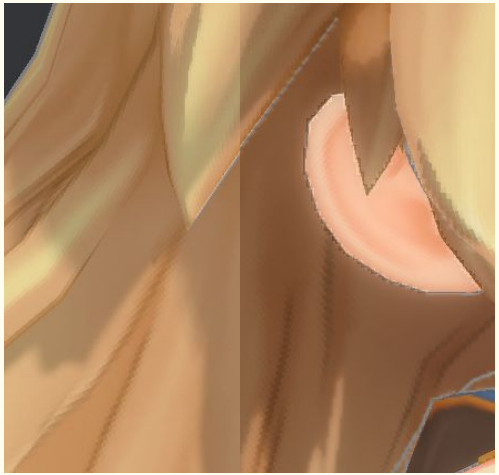
Sirius Bloom



PreFilering → Blur → Combine

（暗部抽出）

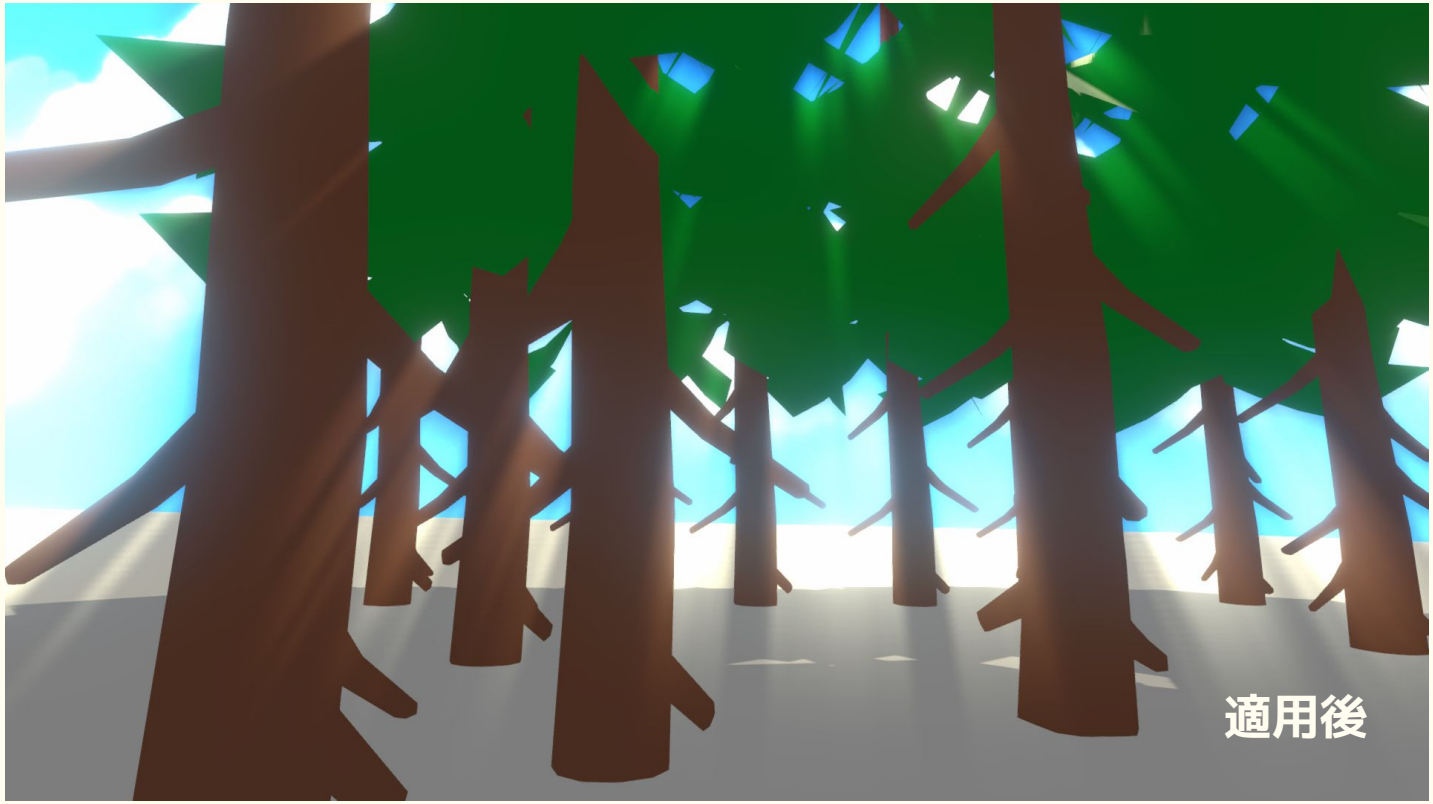
Bloom-DarkBloom



GodRay-概要

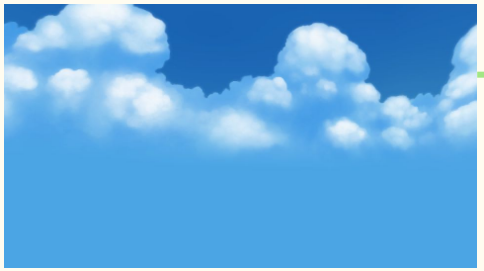
GodRay-概要

GodRay-概要



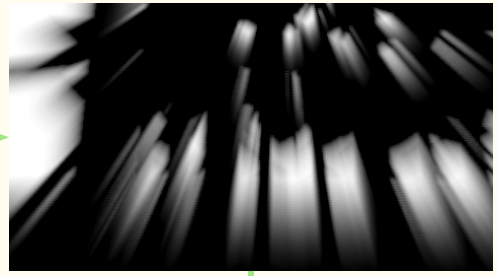
GodRay-ライト方向追従

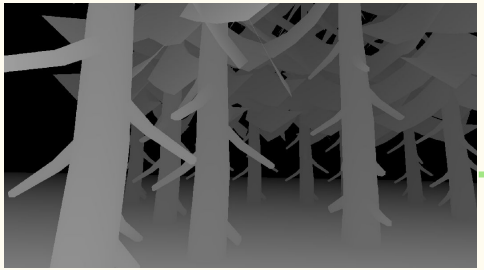
GodRay-実現



輝度抽出

ラジアル
ブラー

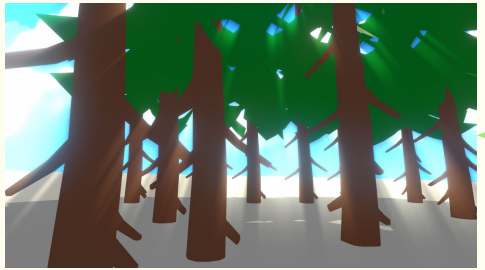




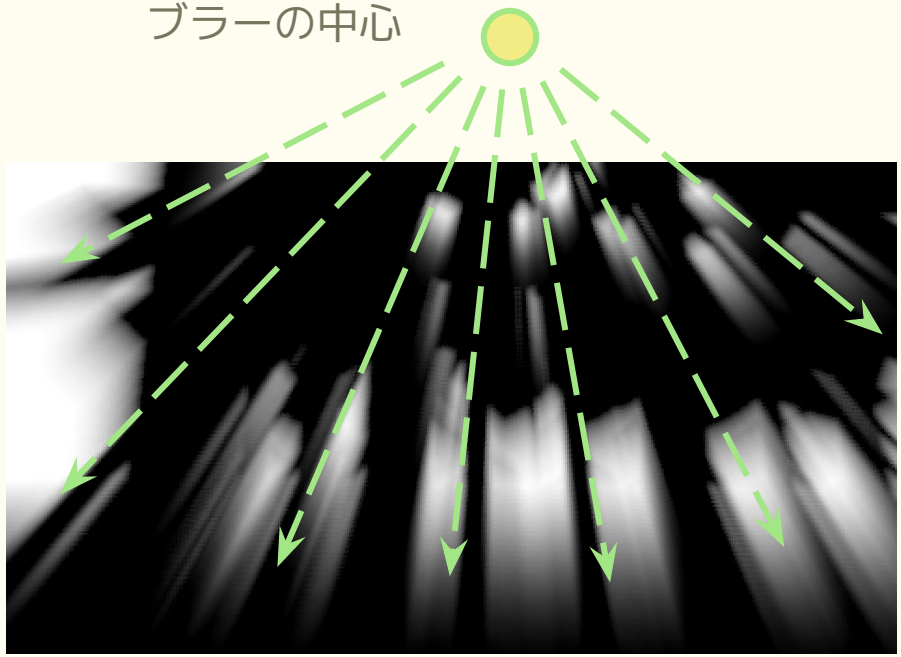
深度テスト

合成

出力



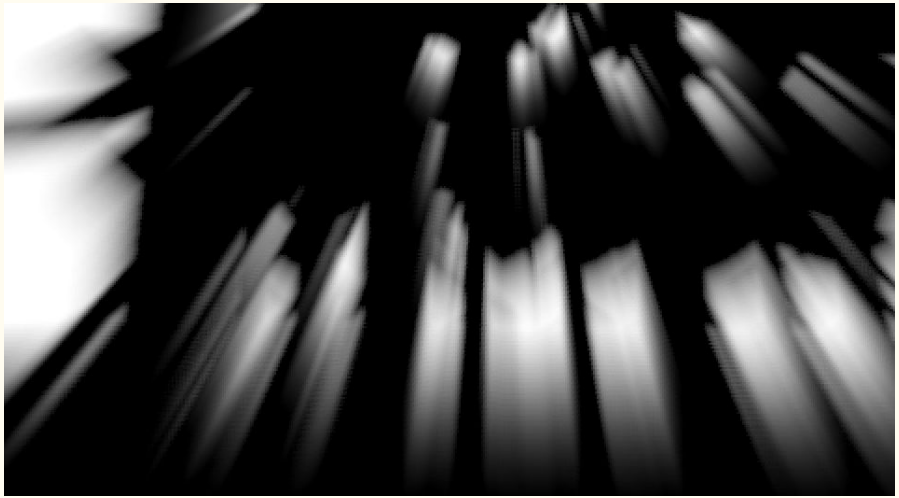
GodRay-ラジアルブラー



ラジアルブラー

GodRay-処理負荷

問題点：
　これぐらいスムーズにするために結構なサンプリング回数が必要で、
　案の定処理負荷が結構重めでした。



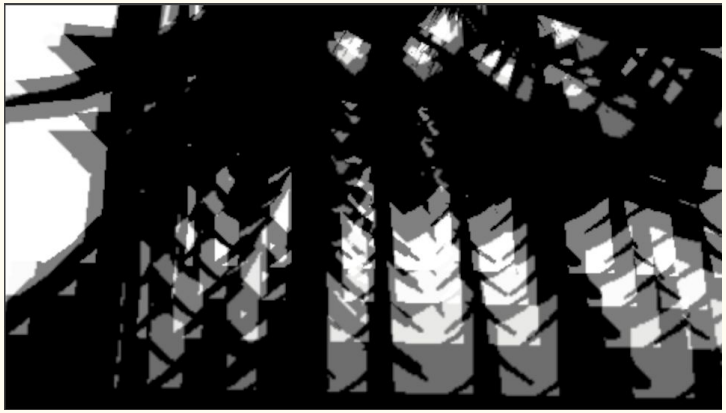
負荷軽減策：

- GodRayテクスチャを縮小
  - 解像度を1/16にする
- サンプリング処理をテンポラルにする
  - 偶数フレームと奇数フレームでサンプリングするUVを分割することで1フレームでのサンプリング回数を半減する
  - サンプリングするUVにノイズを入れ、サンプリング回数をさらに減らす
  - 最後に2フレームのブラー結果にBoxFilterをかけて合成する

テスト結果（Pixel6a、iPhone12）

- カメラターゲットにBlitするまで：4ms程度
- GodRay処理のみ：3ms程度

GodRay-テンポラルサンプリング



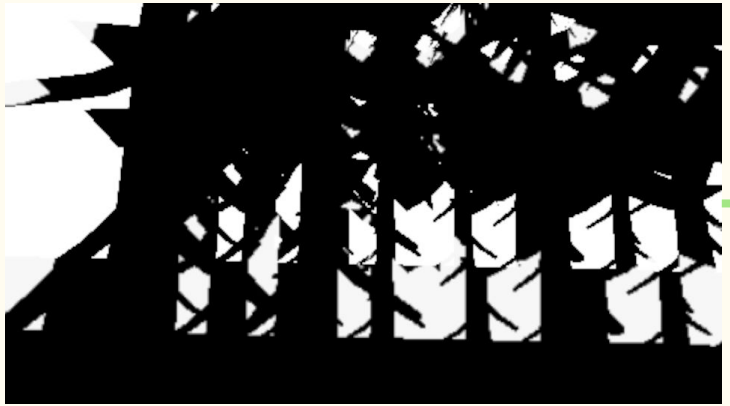
4回サンプリング

偶数フレーム

奇数フレーム

2回サンプリング

2回サンプリング



ブレンド

CEDEC2024
Computer Entertainment Developers Conference

ポストプロセスまとめ

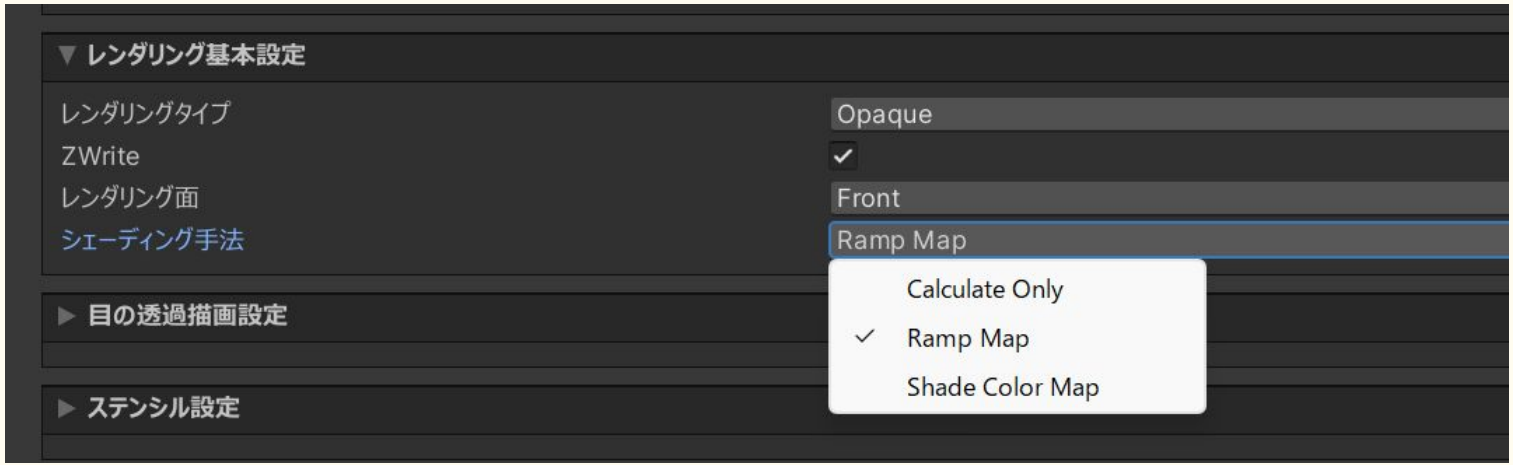
Siriusのポストプロセスは、社内各プロジェクトに導入しやすいように設計されています。
導入後は、複雑な設定や改造の必要はなく、すぐに使用することが可能になります。
さらに、柔軟な設計になっており、各プロジェクトのニーズに応じてカスタマイズすることも容易になります。
そして、将来のUnityアップデートにも備えて、Siriusはすでに新しいRenderGraphに対応しております。
今後プロジェクト側がUnity6への移行する際にも、Siriusを修正いらずに引き続き使用することができます。

Chapter：04

キャラシェーダーの最適化機能

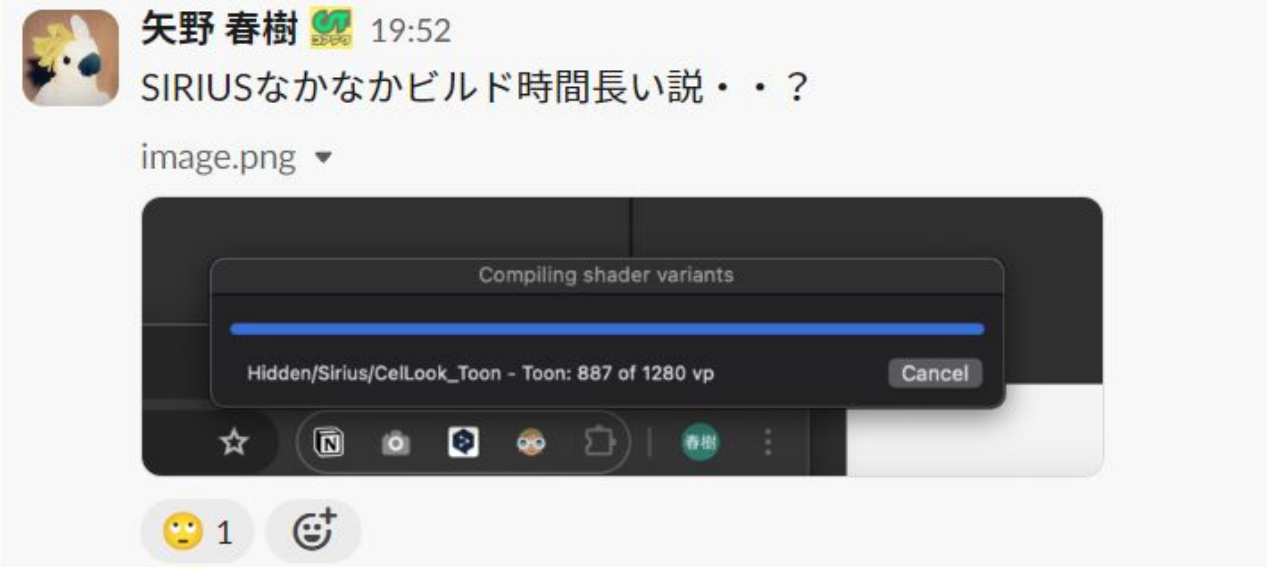
シェーダーバリアントほぼ0

- 機能切り替えをシェーダーバリアントではなく、マテリアルパラメータの分岐で行っているため、シェーダーバリアントはほぼ0



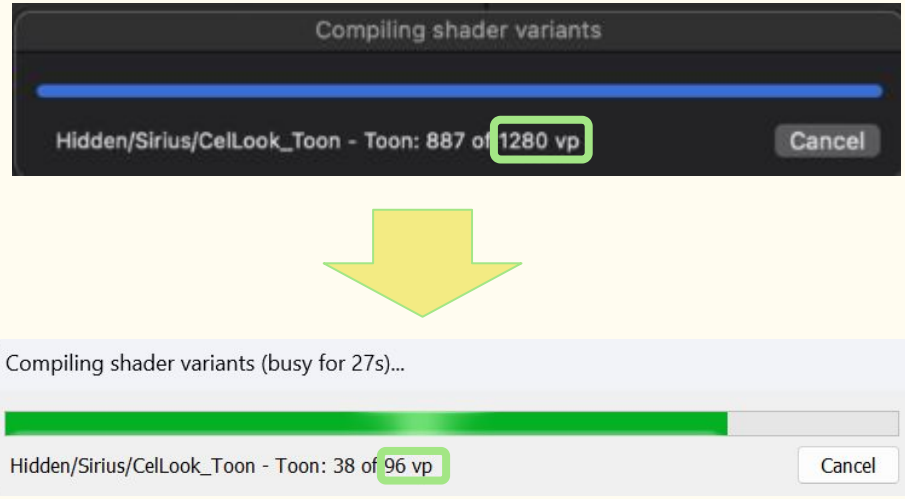
ビルド時間の問題

- v0.8.0まではそこそこシェーダーバリアントを使っていたため、実機ビルド時のコンパイル時間が結構かかっていた



ビルド時間の問題

- v0.9.0からシェーダーバリアントを大幅に削減してビルド時間が大幅に削減
  - Look Dev中の実機確認の効率が大幅に向上



シェーダーのバリエーションの爆発が起きない

- シェーダーバリアントがほぼ0になっているため、シェーダーのバリエーションの爆発が起きない
  - メモリ使用量を削減できる

パフォーマンスに問題が起きないのか？

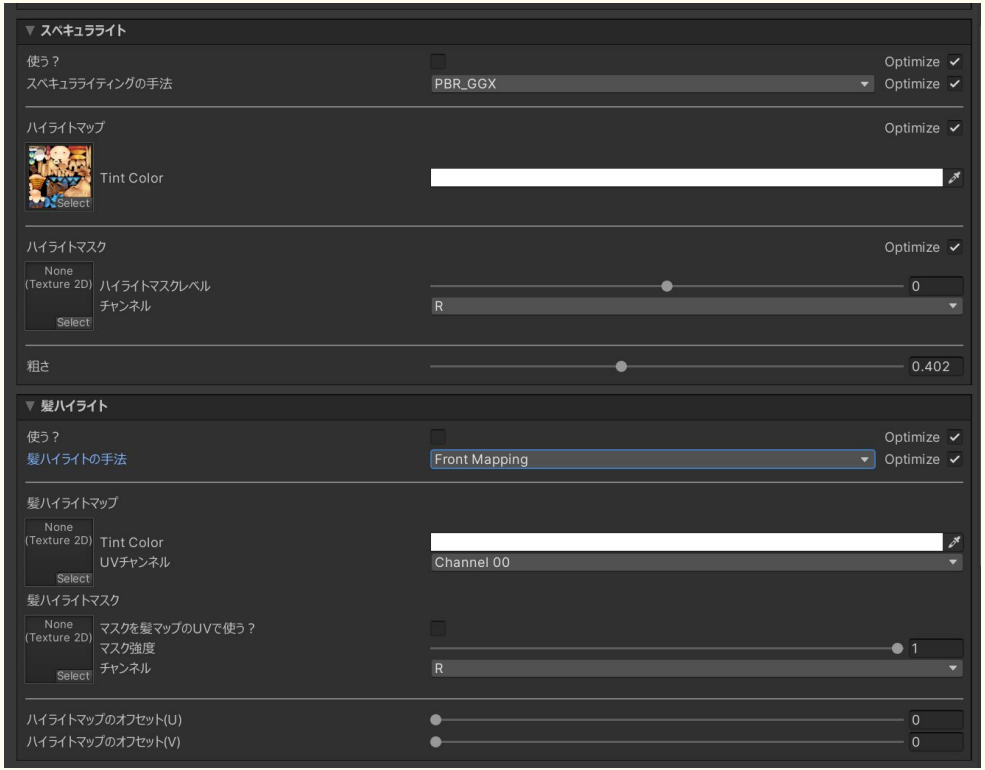
- SIMTアーキテクチャなら一つのドローコールで同じ分岐パスを通るなら大きな問題が起きないはずだが・・・
  - 一部端末で問題が起きた
    - シェーダーコードの肥大化によるレジスタ数が不足している？
    - if文のブロックが残っていることによって、コンパイラの最適化が最大限生かせていない
      - → シェーダーコードが大きくなると問題が顕著化

独自のコードストリップ機能

- オリジナルのシェーダーをベースに最適化されたシェーダーを出力できる
  - 出力されたシェーダーをカスタマイズ可能
- 最適化される機能を選択することができる
  - デフォルトは全て最適化対象

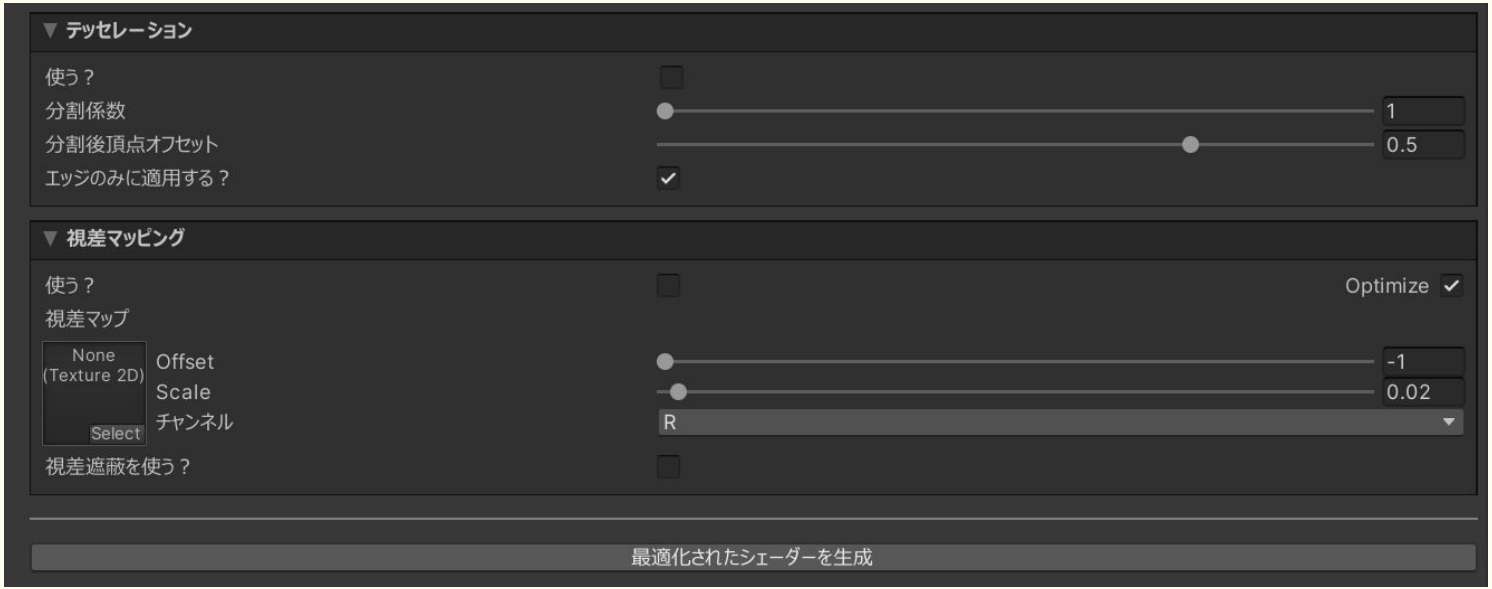
独自のコードストリップ機能

Optimizeチェックボックスで最適化対象を選択できる



独自のコードストリップ機能

「最適化されたシェーダーを生成」ボタンを押すと不要なコードを除去したシェーダーが生成される



どんなコードが出力されている？

オリジナル

```
/**
 * \brief スペキュラライトの計算
 * \param surfaceData  サーフェイスデータ
 * \param lightingData  ライティングデータ
 * \param light ライト
 */
half3 CalculateSpecularLightColor(ToonSurfaceData surfaceData, ToonLightingData lightingData, Light light)
{
    half3 specularColor = half3(0, 0, 0);

    SIRIUS_ifdef_with_dynamic_branch(_UseSpecularLight > 0.5, _CT_USE_SPECULAR_LIGHT)
    {
        // Standard
        SIRIUS_ifdef_with_dynamic_branch(_SpecularLightMethod == CTSpecularLightMethod_Standard, _CT_SPECULAR_METHOD_STANDARD)
        {
            specularColor = CalculateSpecularLightColor_Standard(surfaceData, lightingData, light);
        }
        SIRIUS_endif

        // PBR GGX
        SIRIUS_ifdef_with_dynamic_branch(_SpecularLightMethod == CTSpecularLightMethod_PBR_GGX, _CT_SPECULAR_METHOD_PBR_GGX)
        {
            specularColor = CalculateSpecularLightColor_Pbr_Ggx(surfaceData, lightingData, light);
        }
        SIRIUS_endif


        // PBR GGX Fast
        SIRIUS_ifdef_with_dynamic_branch(_SpecularLightMethod == CTSpecularLightMethod_PBR_GGX_FAST, _CT_SPECULAR_METHOD_PBR_GGX_FAST)
        {
            specularColor = CalculateSpecularLightColor_Pbr_Ggx(surfaceData, lightingData, light);
        }
        SIRIUS_endif
    }
    SIRIUS_endif
    return specularColor;
}
#endif
```

最適化されたシェーダー

```
/**
 * \brief スペキュラライトの計算
 * \param surfaceData  サーフェイスデータ
 * \param lightingData  ライティングデータ
 * \param light ライト
 */
half3 CalculateSpecularLightColor(ToonSurfaceData surfaceData, ToonLightingData lightingData, Light light)
{
    half3 specularColor = half3(0, 0, 0);

    {
        // Standard

        // PBR GGX

        // PBR GGX Fast

        {
            specularColor = CalculateSpecularLightColor_Pbr_Ggx(surfaceData, lightingData, light);
        }
    }
    return specularColor;
}
#endif
```

どれくらい変わるのか？

オリジナル

ScriptableRenderer.Execute:	57.19395
Sirius.SetCharacterShadingPara	0.000312
DrawOpaqueObjects	44.7974
Sirius.RenderGBuffer	4.475625
Sirius.RenderTransmissivePass	0.000417

最適化されたシェーダー

ScriptableRenderer.Execute:	27.19969
Sirius.SetCharacterShadingPara	0.000365
DrawOpaqueObjects	14.44812
Sirius.RenderGBuffer	4.616354
Sirius.RenderTransmissivePass	0.000417

想定している利用のされ方

- オリジナルのシェーダーはLook Dev用
  - 多機能ということは、最終的に不要なコードが多いということにつながるので、あくまでLook Devを想定
    - Look Devの効率化のために実機ビルド時のコンパイル時間優先
- Look Devで作られたシェーダーから最適化された各種シェーダーを生成
  - スキン用、衣服用、瞳用、髪の毛用など
    - 個々の機能は十分に最適化されているので、非現実的なパフォーマンスの処理はない
      - 後々の手戻りを減らせる

Chapter：05

プロジェクトでの導入のされ方

現在の導入状況

- **キャラシェーダ**
  - 新規開発中のプロジェクトで1つ
    - 拡張を行わずオリジナルそのままでの採用
  - 一部機能のみの利用は他プロジェクトでも採用実績あり
- **ポストエフェクト**
  - 多数の開発中プロジェクトで採用

キャラシェーダ

- **Look検証期間を3ヶ月程度削減**
  - 技術資産のない状態から早期に最終仕様の決定および量産へ
    - 表現検証
      - 導入フローが整備されており速やかに検証へ着手
      - 機能が揃っており高速なイテレーション
    - 負荷検証
      - リリースに耐えられる水準にチューニング
- **スケジュールや技術資産が不足しているプロジェクトで大きな恩恵**

ポストエフェクト

- **多くのプロジェクトで採用**
  - 各プロジェクトで要求される機能はほとんど同じ
  - 凝縮された機能単位で利用でき、導入や検証後の破棄が簡単
  - 高品質で高いパフォーマンス

共通基盤を導入してもらうための取り組み

- 子会社の目線では社外ツール
- 導入を避けられる要因
  - ニーズへのミスマッチ
  - 学習コストの高さ
  - パフォーマンスの懸念
  - サポートとメンテナンスへの不安

共通基盤を導入してもらうための取り組み

- **ニーズへのミスマッチ**
  - プロジェクト毎に固有のニーズを持ち、柔軟性に欠けると採用しづらい
  - 対策
    - 子　　会　　社　　にヒアリングを　　行　　うだけでなく、
      専任のTAと協力することで解像度高く事業部内の需要を抽出
    - 機能追加を続けることでニーズをカバー
    - インターフェースを整備して利便性を確保

共通基盤を導入してもらうための取り組み

- **学習コストの高さ**
  - 新しいフレームワークや機能を導入する際には学習コストが発生
  - 対策
    - ドキュメントを整備
    - 各社内広報手段でのアップデート告知や社内勉強会での解説
    - 相談があればサポートも

共通基盤を導入してもらうための取り組み

- **パフォーマンスの懸念**
  - スマートフォンではパフォーマンス　　要　　求　　がシビアであり、求める表現を実現するだけでは不十分
  - 対策
    - コアテク側でチューニング済み
      - 高　度 なグラフィックス 技　術 を 持 たないプロジェクトでも製品レベルの機能を利用可能

共通基盤を導入してもらうための取り組み

- **サポートとメンテナンスの問題**
  - バグやプロジェクトで利用するバージョンへの対応コストが不透明
  - 対策
    - 導　　入　　しているプロジェクトからフィードバックをもらい、コアテク側でアップデートとサポートを実行
    - Unity6のRender Graphにも対応済み

Chapter：05

まとめ

まとめ

- Look Devの開発効率の向上を目指して多機能なキャラシェーダーを開発中
- 部分的な導入をしやすいように設計された軽量で品質の高いポストプロセスを開発中
- 独自のコードストリップ機能によって、パフォーマンスも保証
- 近々正式採用されたタイトルがリリースされる予定

ご清聴ありがとうございました。