学園アイドルマスターにおけるモバイルの性能を限界まで引き出すレンダリングパイプライン

QualiArts 渡邉俊光

自己紹介

渡邉俊光

- 2012年CyberAgent新卒入社
- 株式会社QualiArtsのテクニカルアーティスト室
- オルタナティブガールズ、IDOLY PRIDE等の開発を経て、学園アイドルマスターの描画周りの実装を担当

学園アイドルマスター

- 2024/5/16 サービス開始
- 歌とダンスが上手くなるアイドル育成シミュレーション

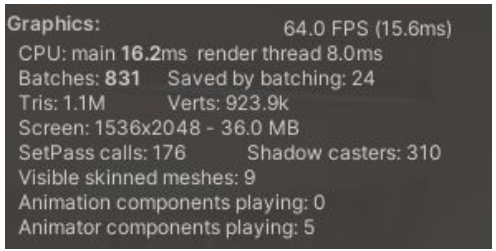
目次

- レンダリングパイプライン
- 反射表現
- ライティング
- 機種依存問題
- まとめ

レンダリングパイプライン

実行環境

- Unity2022.3.21f1 ※リリース時点
- UniversalRenderPipeline 14.0.10
- プラットフォーム
  - Android Vulkan
  - iOS Metal iOS GPU Family4
- 1キャラクター6万ポリゴン
- シーン100万ポリゴン

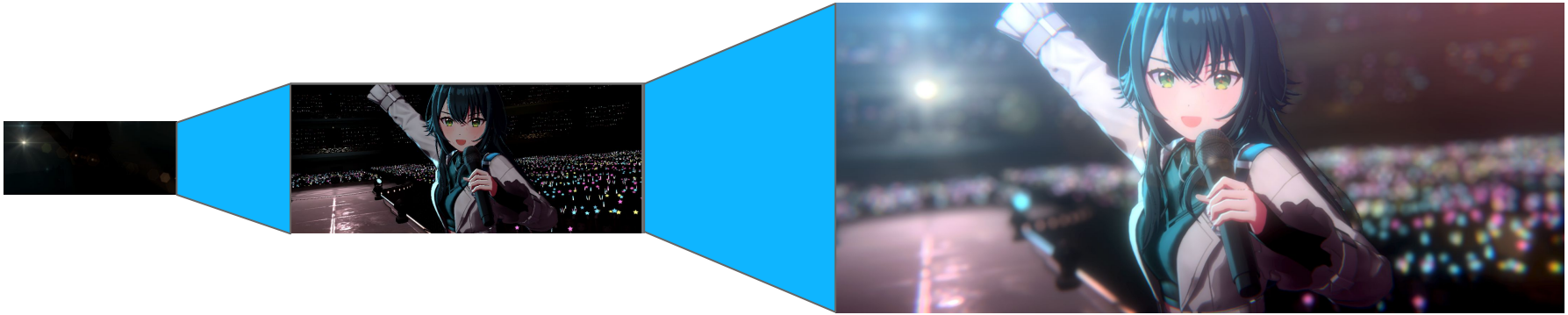


実装方針

- 現実と同じライティングを行いたいのでDeferred Lighting
- 画面の情報量をとにかく上げる
  - テクスチャサイズを節約できるように全ての機能を備えたデカール対応
  - 可能な限り直接光反射を実現する
- リアルタイムレンダリングの違和感を極力減らす
  - 高品質なアンチエイリアスの導入
  - ノイズ感の低減
  - 品質の高いポストエフェクトを実装
- 解像度は落ちるが幅広い端末で動作

描画解像度

- ポリゴン数は最低端末に合わせて制限
- 負荷の高い3Dは解像度を落としてFSRでアップスケール
- さらに負荷の高い処理は1/4で描画を行いアップスケール

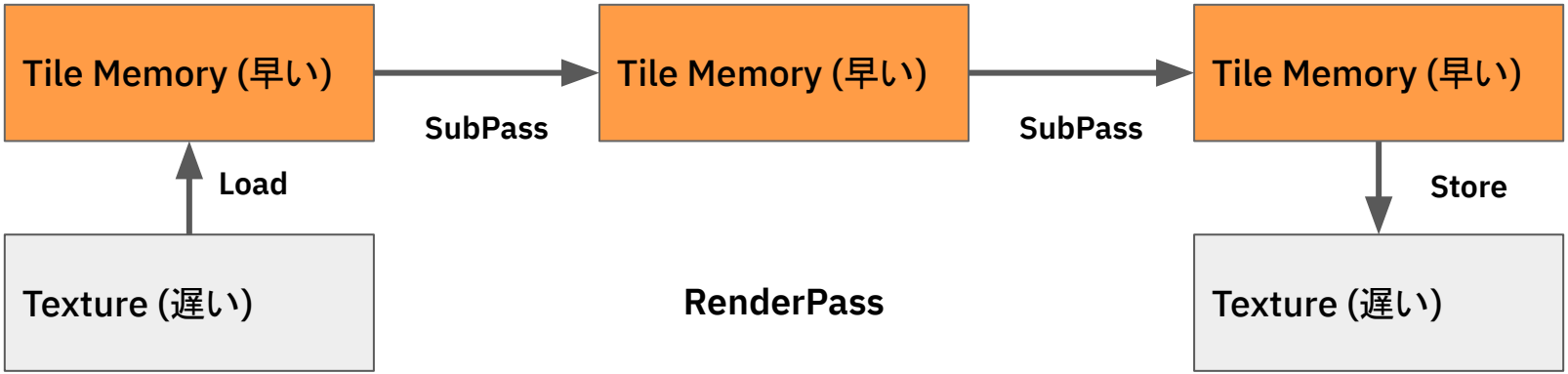


Tile-based Deferred Rendering

- 画面を分割してタイルメモリという高速な領域で実行
- 複数のパスを1つにまとめてタイルメモリのピクセルを取得可能
- タイルメモリにのみ存在するMemorylessという概念が存在
- Unityの場合
  - 結合パス:RenderPass
  - 個別パス:SubPass
  - 描画ターゲット:Attachments
  - サンプル:LOAD_FRAMEBUFFER_INPUT
- URP14にはNativeRenderPass機能があるが使用しない方がいい
- URP17はRenderGraphを用いて簡単に実装できるようになっている

Tile-based Deferred Renderingの特徴

- 隣のピクセル情報は取得できないので歪みのような表現はできない
- モバイルでは描画ターゲットの合計メモリを256bit以内に抑える
- 高速とはいえSubPass切り替え負荷はかなり高いので少ない方がいい
- 無理にRenderPassでまとめるより、分割してテクスチャサンプルする方が早い場合もある

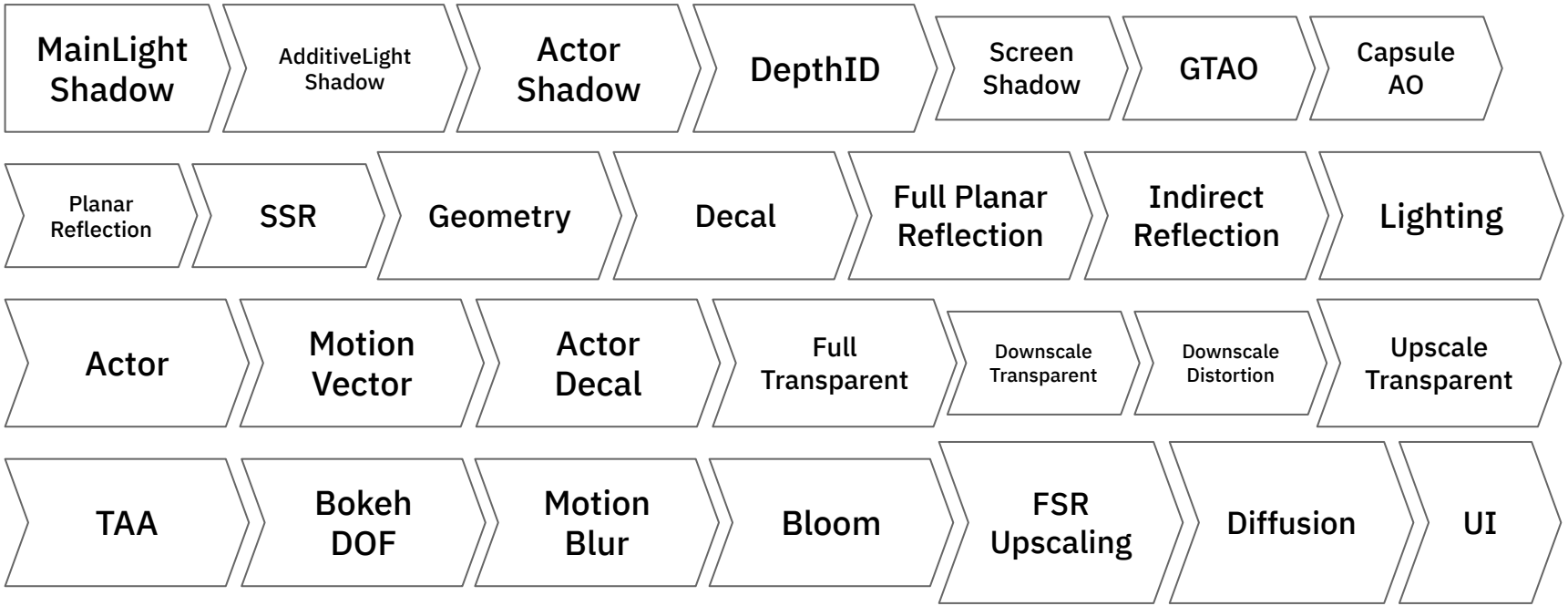


描画概要

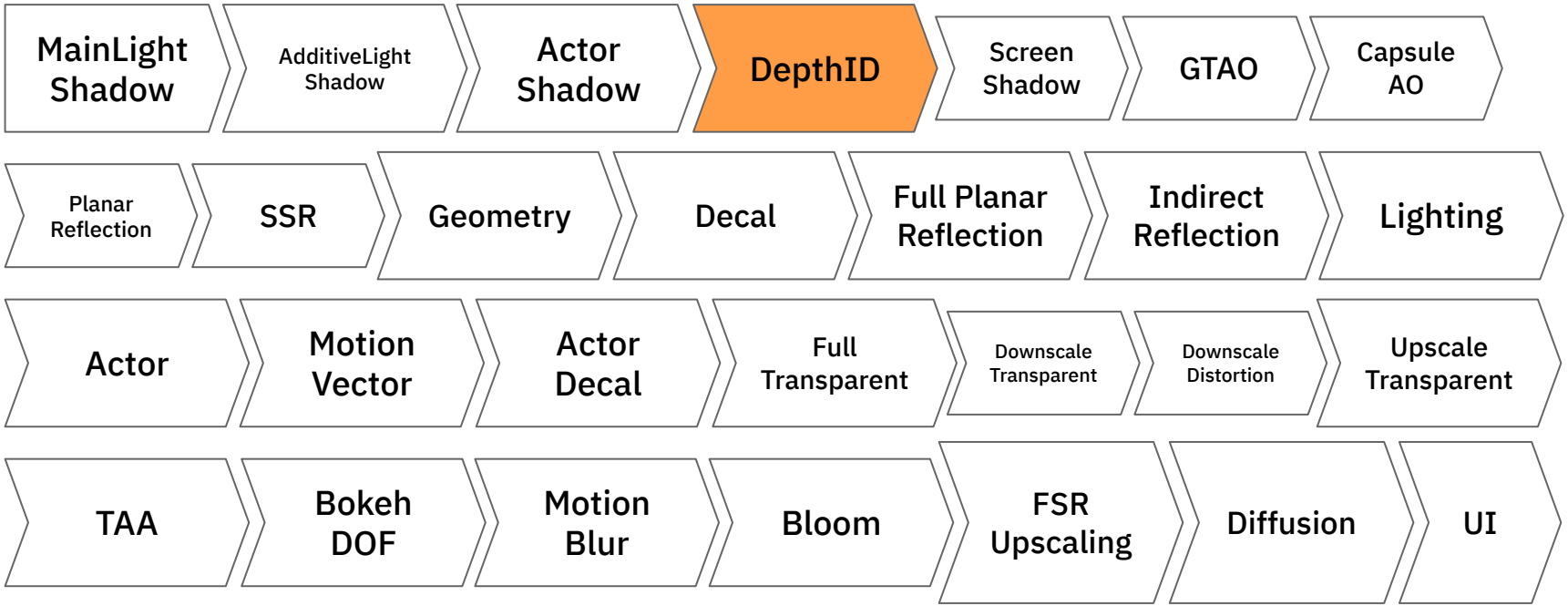
- 不透過 Deferred
- 透過 Forward+
- キャラクター Forward
- デカールライト
- GBufferデカール
- Hi-Z ScreenSpaceReflection
- PlanarReflection
- GTAO (Amplify Occlusion)
- Bokeh Depth Of Field
- Temporal Anti Alias
- FSR Upscaling

描画機能

描画機能

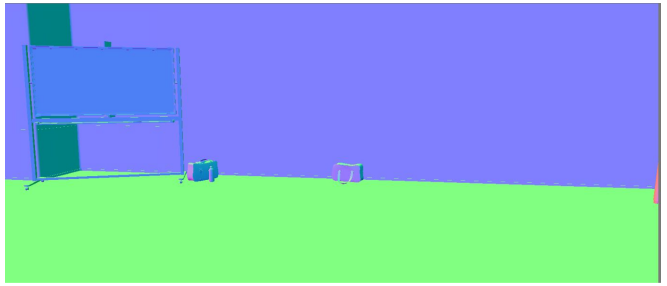


DepthID



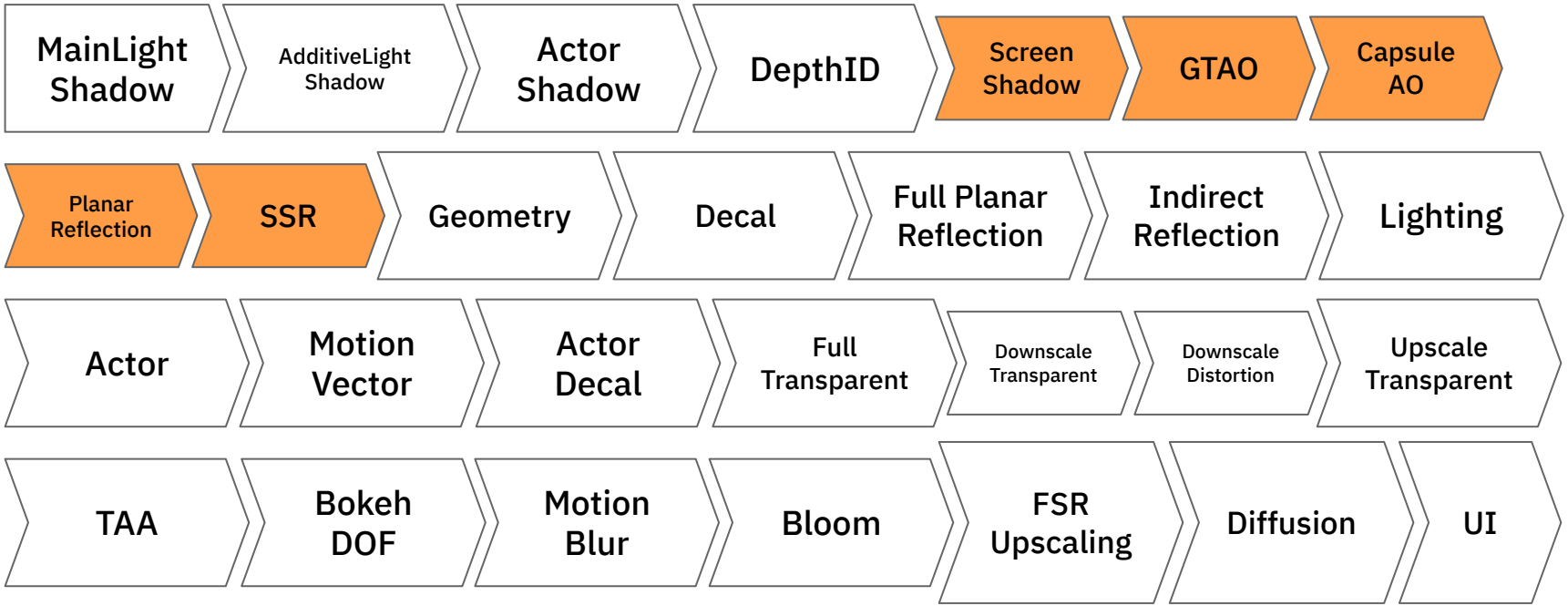
DepthID

- SSR, GTAO, デカールに使用
- ノーマルマップを含まない面法線で描画
- レイマーチのノイズ軽減
- シェーダ簡略化による負荷軽減
- SSRMask（Smoothness閾値以上）



	R	G	B	A
DepthID R8G8B8A8_UNorm	Face Normal	Face Normal	Face Normal	MaterialID (SSR Mask)

Shadow, Occlusion, Reflection

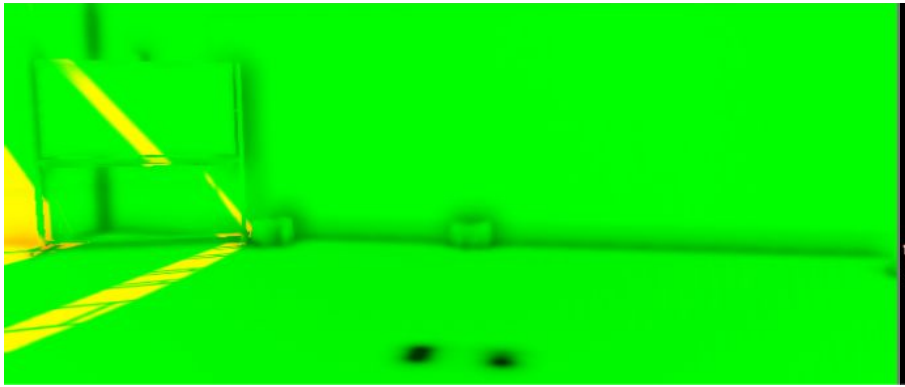


Shadow, Occlusion, Reflection

- 負荷の高いShadow, SSAOをここで解決
- 足元に特化したCapsuleAOも描画
- ScreenSpaceReflectionも同一RenderPassで実行

	R	G	B	A
Trace Buffer (Memoryless) R16G16B16A16_SFloat	Planar R Trace X	Planar G Trace Y	Planar B Trace Weight	Planar Weight
Reflection Buffer R16G16B16A16_SFloat	Reflection	Reflection	Reflection	Reflection Weight
Shadow And Occlusion R8G8_UNorm	Main Light Shadow	GTAO Capsule AO		

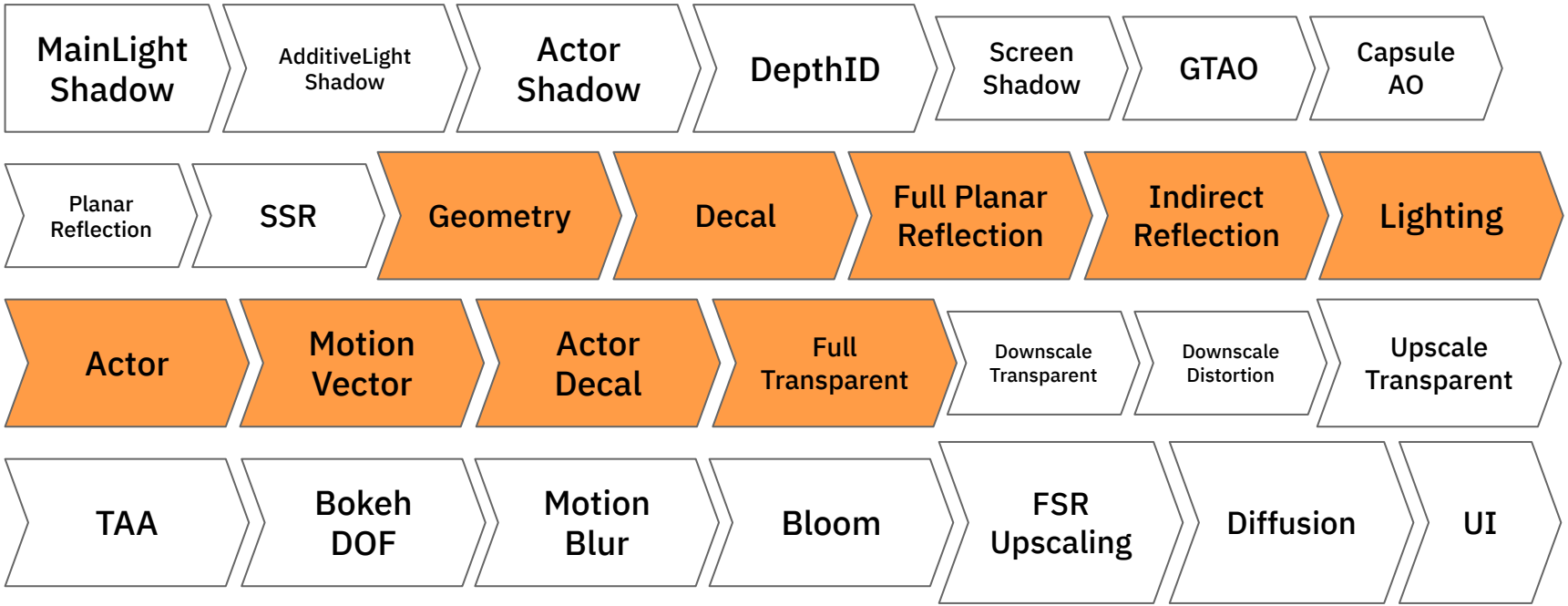
Shadow, Occlusion, Reflection



Shadow And Occlusion

Reflection

GBuffer, Lighting



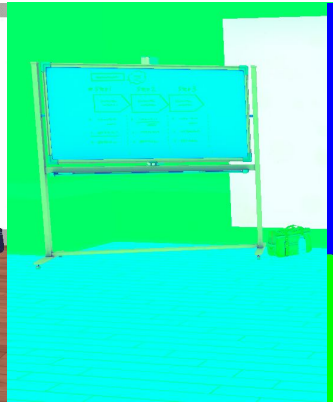
GBuffer Layout

	R	G	B	A
GBuffer 0 (Memoryless) R8G8B8A8_SRGB	Albedo	Albedo	Albedo	MainShadow + ShadowMask R
GBuffer 1 (Memoryless) R8G8B8A8_UNorm	Metallic	Occlusion	Smoothness	ShadowMaskGBA (3:3:2)
GBuffer 2 R16G16B16A16_SFloat	Normal	Normal	Normal	MaterialID
GBuffer 3 B10G11R11_UFloatPack32	Environment Emission	Environment Emission	Environment Emission	-
GBuffer 4 B10G11R11_UFloatPack32	GI	GI	GI	-

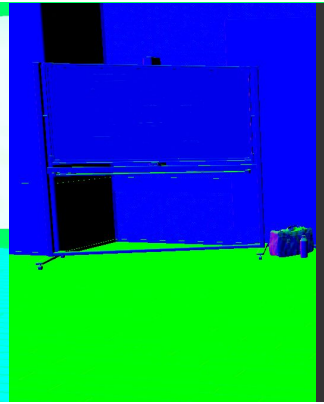
Geometry

- Albedo, Metallic, Occlusion, Smoothness他出力
- GIバッファに計算済みのライトマップとLightProbeの情報を出力
- 加算ライトのShadowMaskはディザをかけて8bitにパック

Albedo

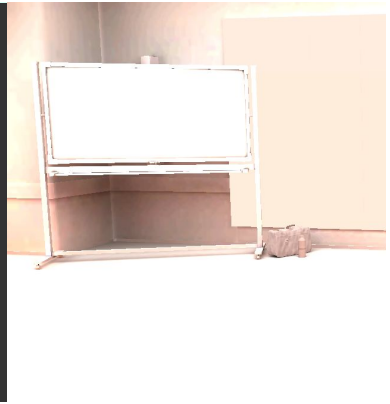


MetallicOcclusion
Smoothness



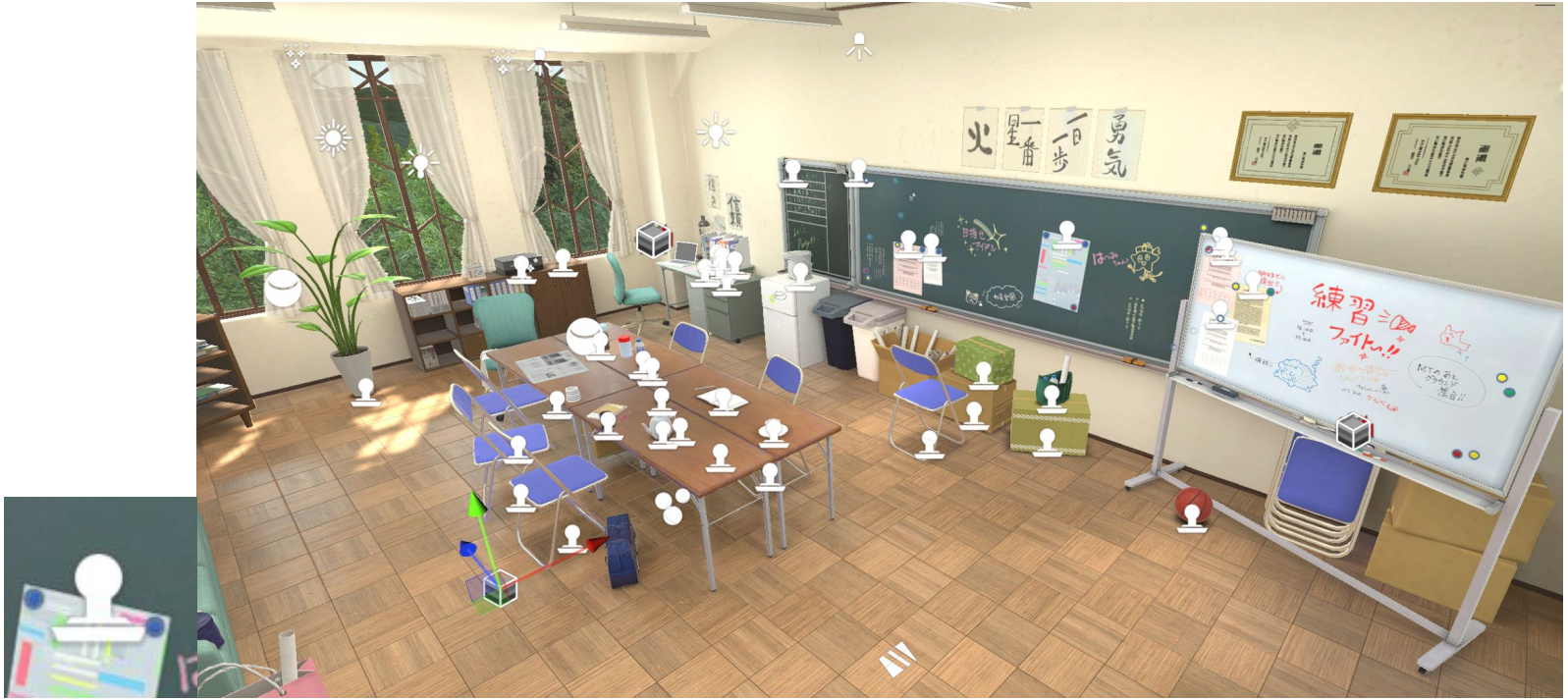
Normal

Emission



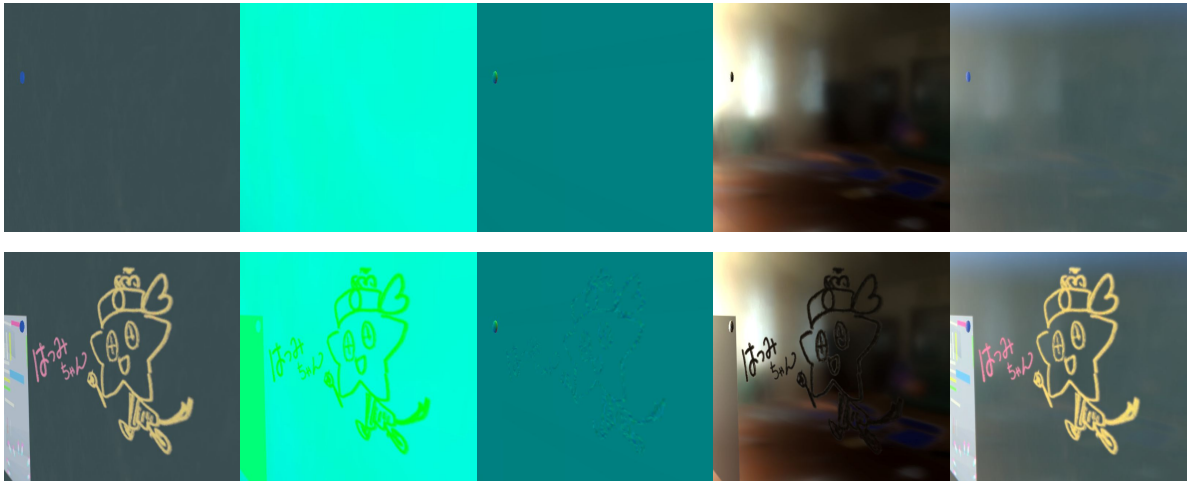
GI

Decal



Decal

- AlbedoやEmissionのみのブレンドではないので
  PBRの全ての表現を再現可能
- DBuffer方式と違いデカールがない場合追加負荷が0になる
- 下地のテクスチャを節約できるのでモバイルでも有効



Environment Emission

- キャラクターを含まない背景だけ先にライティング
- デカール適用済みのバッファを使用してReflectionを描画
- PlanarReflection、ScreenSpaceReflection、ReflectionProbeを合成
- Stencilを利用したDeferredLighting

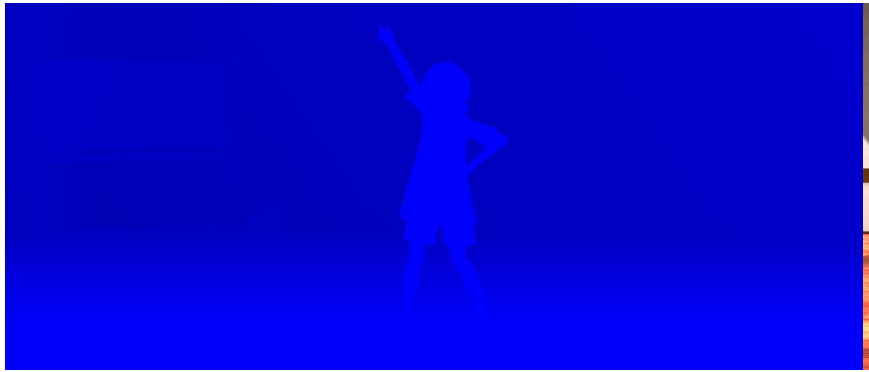
	R	G	B	A
GBuffer 3 B10G11R11_UFloatPack32	Environment Emission	Environment Emission	Environment Emission	-

Actor Geometry

- キャラクターをForwardで描画するパス
- Deferred処理は完了しているのでGBuffer2をMotionVector, Depth, MaterialIDバッファとして再利用
- GIも不要なのでGBuffer4を最終出力バッファとして再利用

	R	G	B	A
GBuffer 2 R16G16B16A16_SFloat	Motion Vector	Motion Vector	Depth	MaterialID
GBuffer 4 B10G11R11_UFloatPack32	Emission	Emission	Emission	-

Actor Geometry



MotionVector + Depth + MaterialID

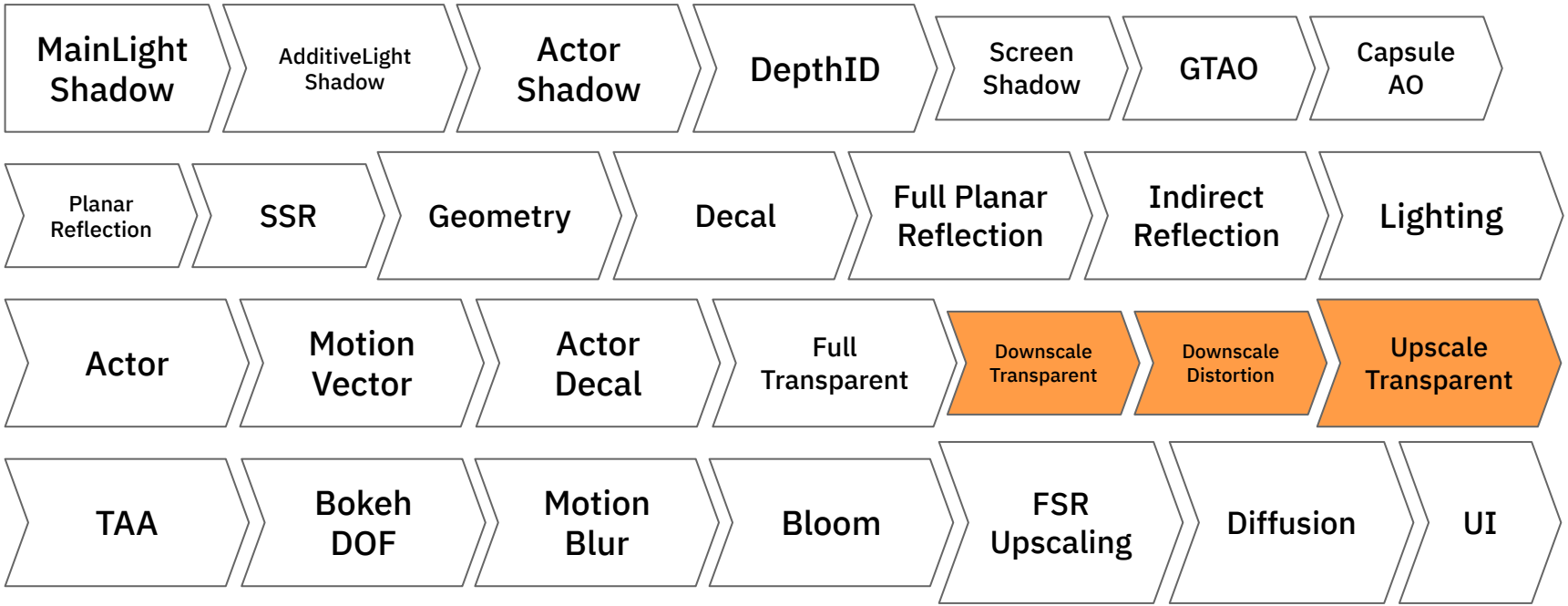
Emission

Emission

- 解像度が必要な透過メッシュ描画
- キャラクター用デカール（青ざめやチーク等）
- DepthFog、SphereFog

	R	G	B	A
GBuffer 4 B10G11R11_UFloatPack32	Emission	Emission	Emission	-

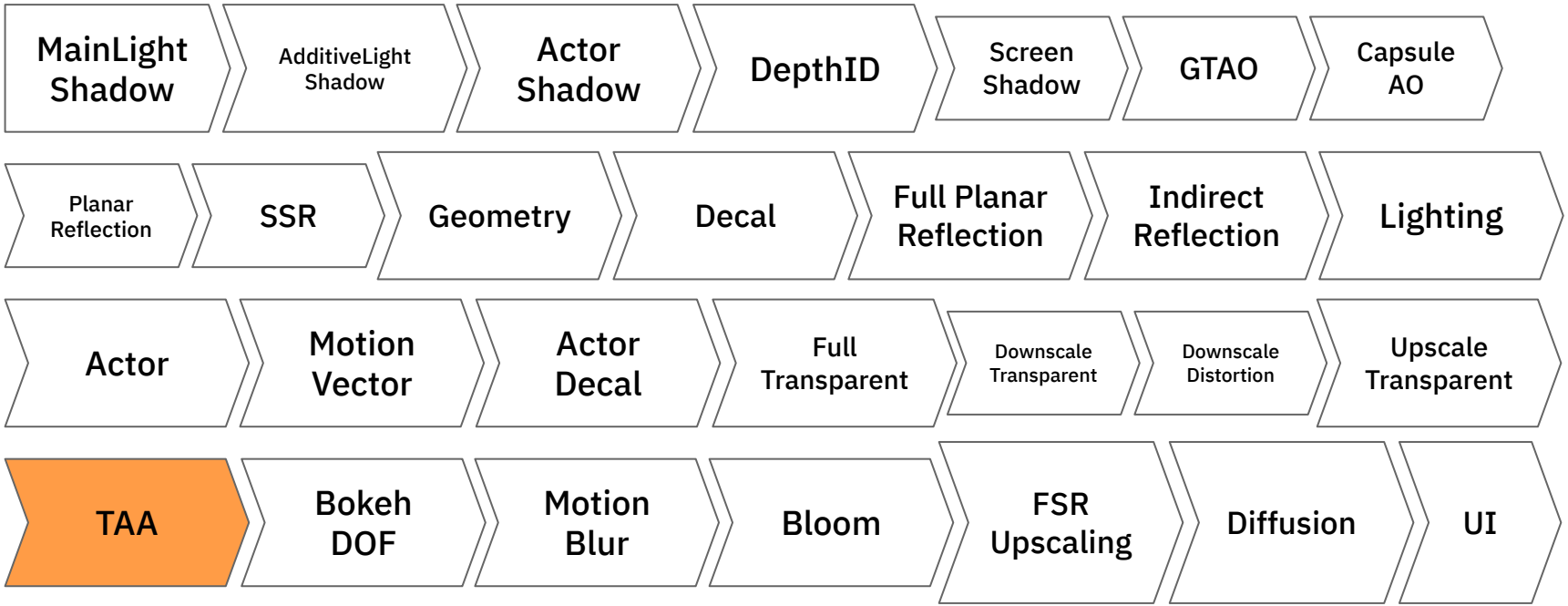
Downscale Transparent



Downscale Transparent

- 1/4の解像度で実行される透過メッシュ描画
- 高負荷なパーティクル、レンズフレア、ボリュームライト
- アップスケールして合成

TAA



Temporal Anti Aliasing (TAA)

- 過去フレームを参照してジャギーを減らすアンチエイリアス
- UnityのTAAを使用（Variance Clipping）
- 発光が溶けないようにExclude TAA対応
- 鏡用にTAAを完全にオフにするNoJitterフラグ
- GBuffer2から全ての情報を取得できるように調整

```
half4 depth4 = GATHER_BLUE_TEXTURE2D( GBuffer2, sampler_LinearClamp, uv);
half2 velocity = -SAMPLE_TEXTURE2D(_GBuffer2, sampler_LinearClamp, uv).xy;
half materialID = SAMPLE_TEXTURE2D(_GBuffer2, sampler_PointClamp, uv).w;
// 2:ExcludeTAA 4:NoJitter
if (((uint)materialID & 6u) > 0u)
{
    return ((uint)materialID >> 2 & 1u) ? half4(colorCenter, 1.0h) : nonJitterColor;
}
```

Temporal Anti Aliasing (TAA)

TAA 適用前

TAA 適用後

FXAA vs TAA

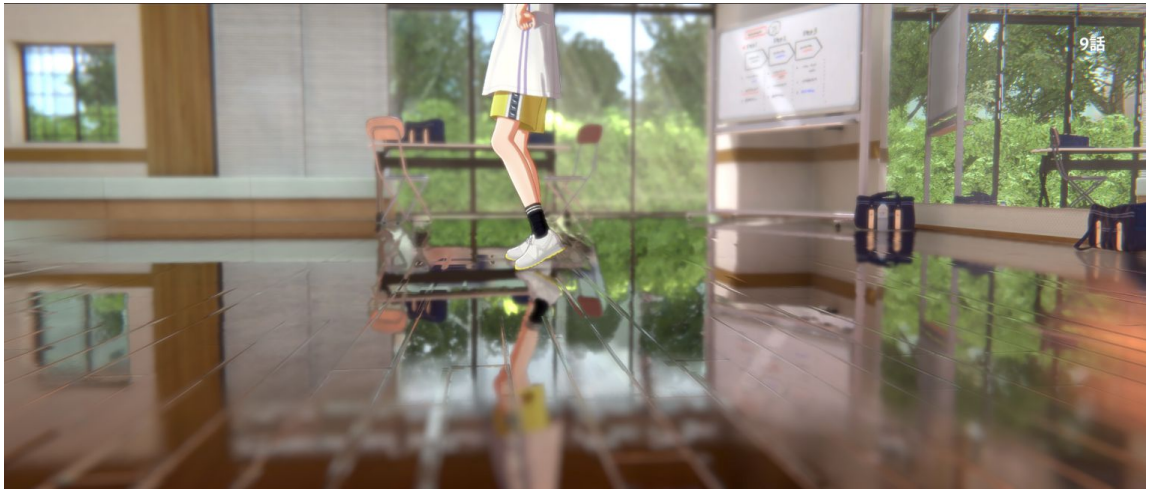
FXAA

TAA

反射表現

Screen Space Reflection (SSR)

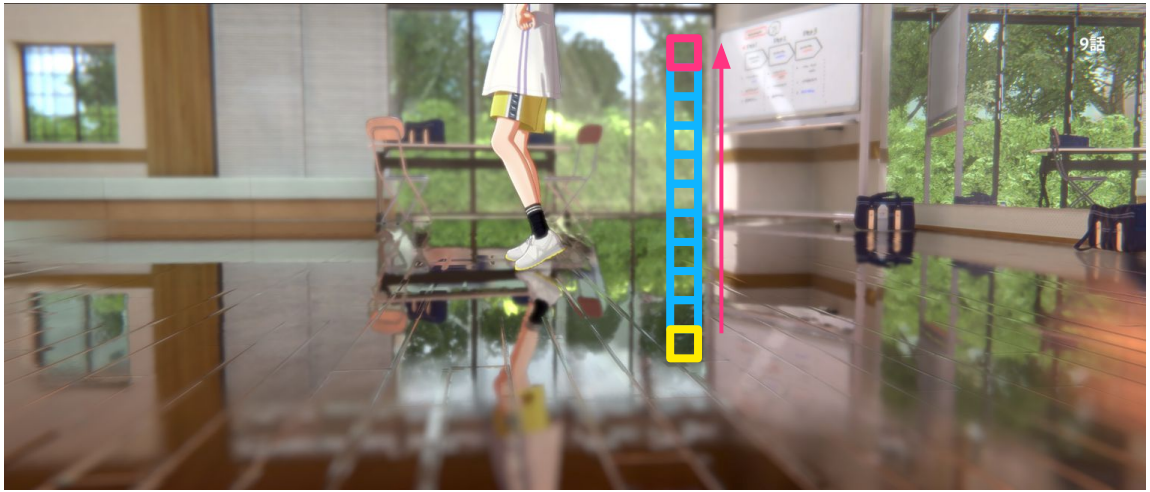
- 画面の情報量を増やす手段として一番力を入れて実装を行なった
- 1フレーム前の画面内に存在していればその色を参照して書き込み
- ライブのようなリアルタイムに変化する環境でより効果的



※わかりやすく床を鏡面に変更しています

SSRの簡単な仕組み

- NormalDepthを参照してワールド座標とワールド反射ベクトルを計算
- ワールド反射ベクトル方向にforループを使ってDepthの衝突判定
- 衝突したらそのUVの1フレーム前の色を取得



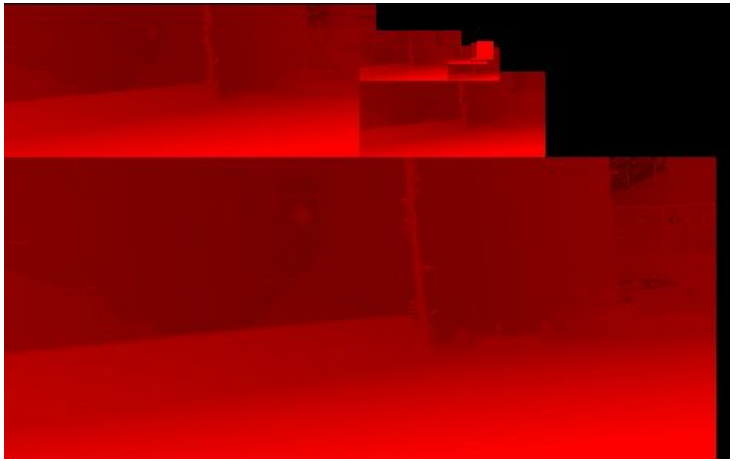
※わかりやすく床を鏡面に変更しています

SSRの長所と短所

- 長所
  - 画面内のライティング結果をリアルタイムに反映可能
  - シーンの複雑度に依存せず一定の負荷で実行
- 短所
  - 解像度が高い場合より多くDepthをチェックする必要がある
  - 1フレーム前の画面内に存在しないピクセルはサンプルできない
  - 別オブジェクトに隠れているピクセルはサンプルできない
  - シーンの法線が複雑な場合サンプル位置がずれて隙間が発生

高解像度トレース

- Hi-Z Depth
- Hi-Zを利用することで解像度に依存しにくくなる
- UnityのHDRPから逆輸入
- ComputeShaderのRandomWriteを利用



SSRのトレース失敗問題

- 画面にないピクセルは取得できない
- 遮蔽物がある場合ループ回数が低いと到達できない
- キャラクターのような複雑な形状が挟まると周囲の失敗確率が上がる

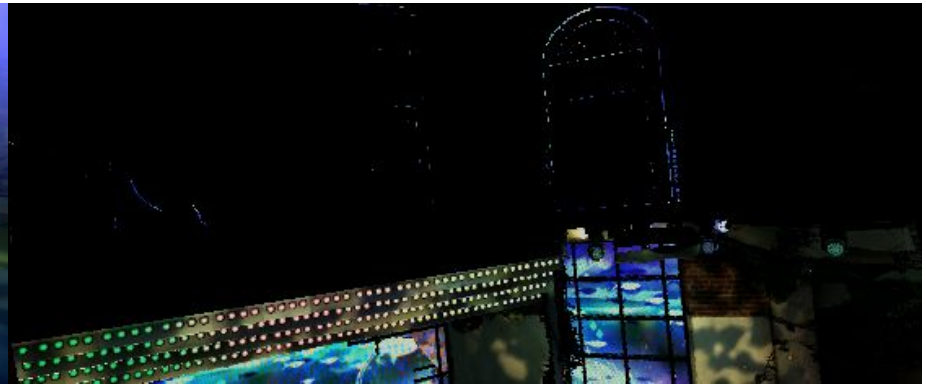
キャラクターを除外する

- 問題になりやすいのはキャラクター
- SSRで使用するテクスチャにキャラクターを含めない（パワー）

SSR改善結果

- 床面の黒い領域を解決できた
- キャラクターが反射しない

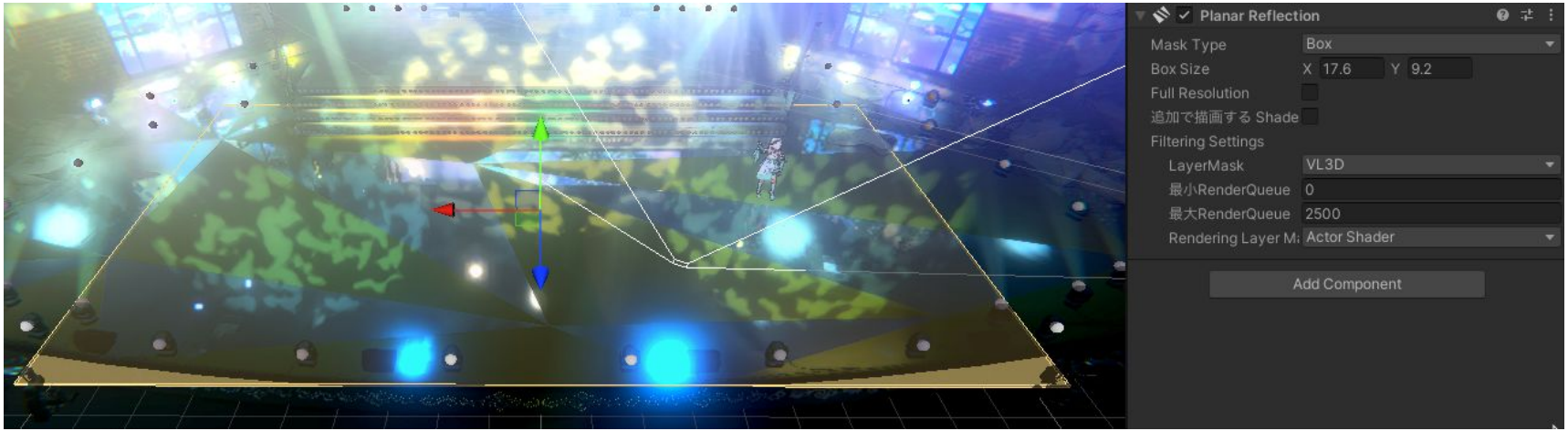
SSR改善結果



Reflection

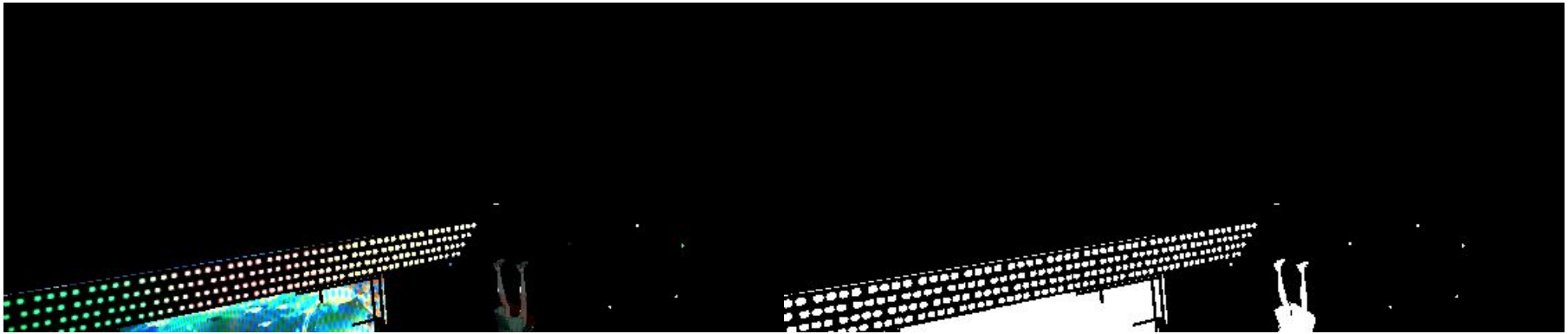
PlanarReflectionとSSRの統合

- 昔ながらの平面反射実装
- 負荷の高い処理を省いたForwardでキャラクターを反転描画
- SSRのトレースと合成



PlanarReflection描画

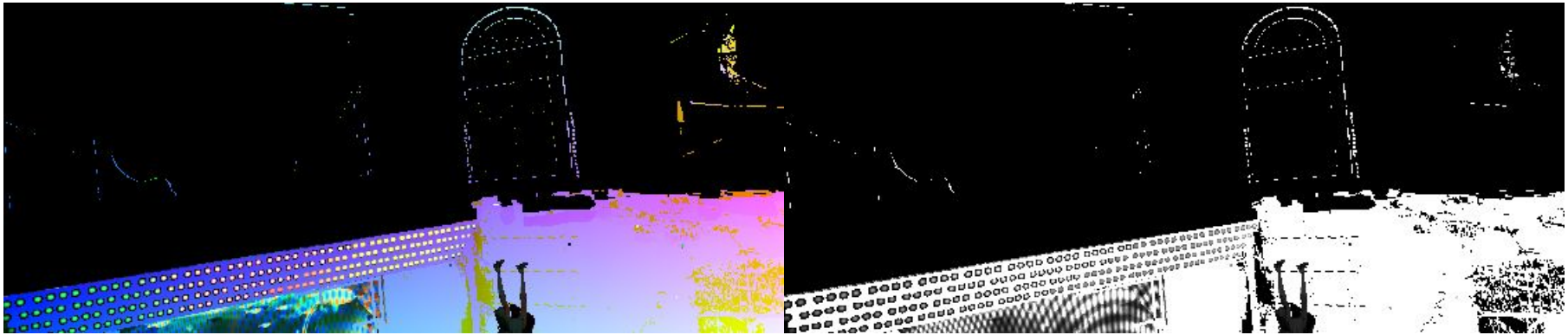
- キャラクターと一部発光メッシュ描画
- 発光メッシュはモニターなどが画面内になくても映り込むように
- アルファチャンネルは**PlanarReflection**の描画領域マスクとして出力



RGB: PlanarReflectionColor　　A: PlanarReflectionMask

SSR Trace

- SSRのレイ計算を実行
- PlanarReflection描画済みの領域やSmoothnessが低い箇所は除外して高速化



RG: TraceUV

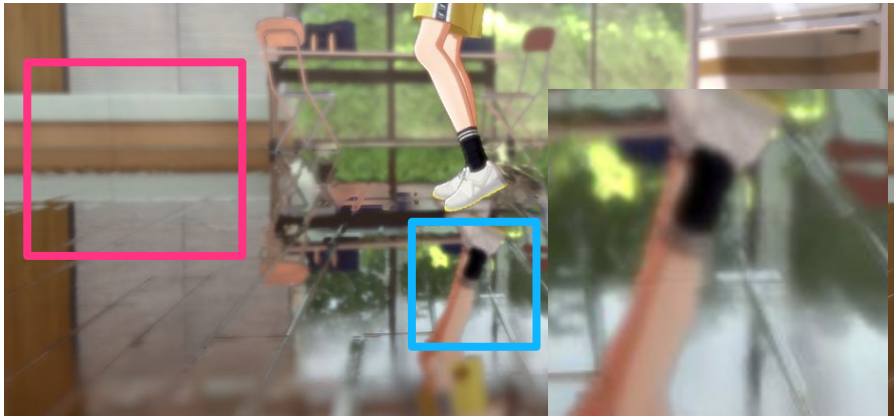
B: SSRWeight

Reflection解決

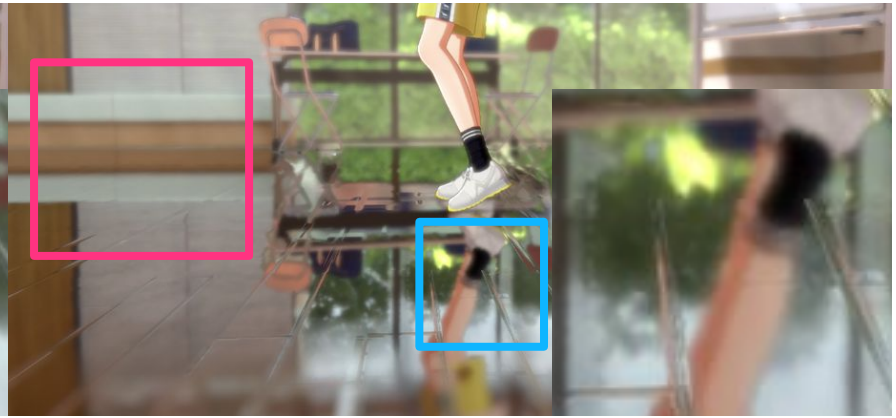
Reflection解決

```
half4 ResolveReflectionFragment(Varyings input) : SV_Target
{
    half4 trace = LOAD_FRAMEBUFFER_INPUT(0, input.positionCS.xy);

    // Planar Reflection
    if (trace.a > 0.0h) return trace;

    if (trace.z < 0.001h) discard;

    // 省略

    half4 output = ResolveSSR(data, _LitColorTexture, sampler_LinearClamp);
    return output;
}
```

法線テクスチャによる SSRの変化



ノーマルマップ法線



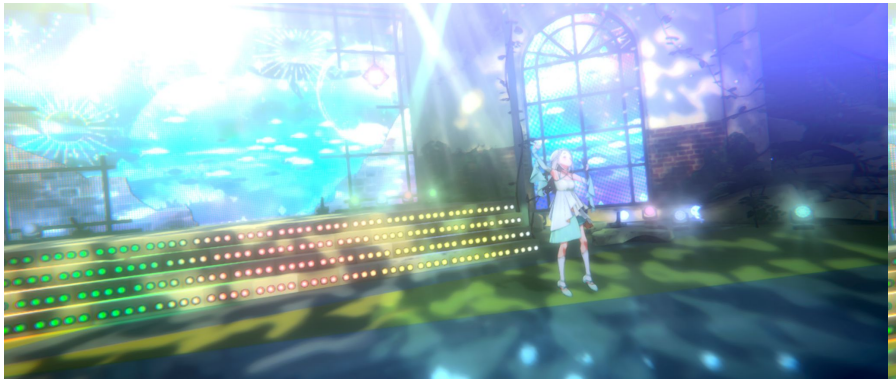
フェース法線

Reflectionの歪み処理

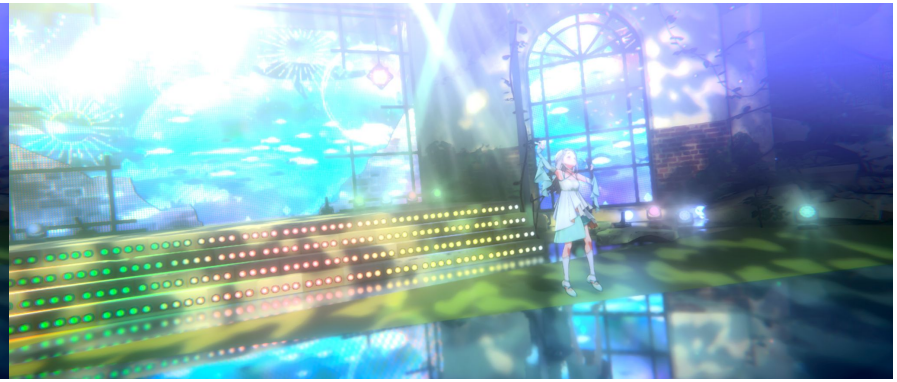
- 歪みを高解像度で処理することでReflectionの引き伸ばし感を低減
- フェース法線とノーマルマップ法線の差分から歪み方向を算出

```
half3 faceNormalWS = SampleSceneNormals(normalizedScreenSpaceUV);
half3 faceNormalVS = mul(faceNormalWS, (half3x3)UNITY_MATRIX_I_V);
half3 normalVS = mul(normalWS, (half3x3)UNITY_MATRIX_I_V);

// メッシュの法線とBumpMap法線のずれの分だけ擬似的にオフセット
float2 offset = (normalVS - faceNormalVS).xz * float2(0.1, -0.1);
```

比較



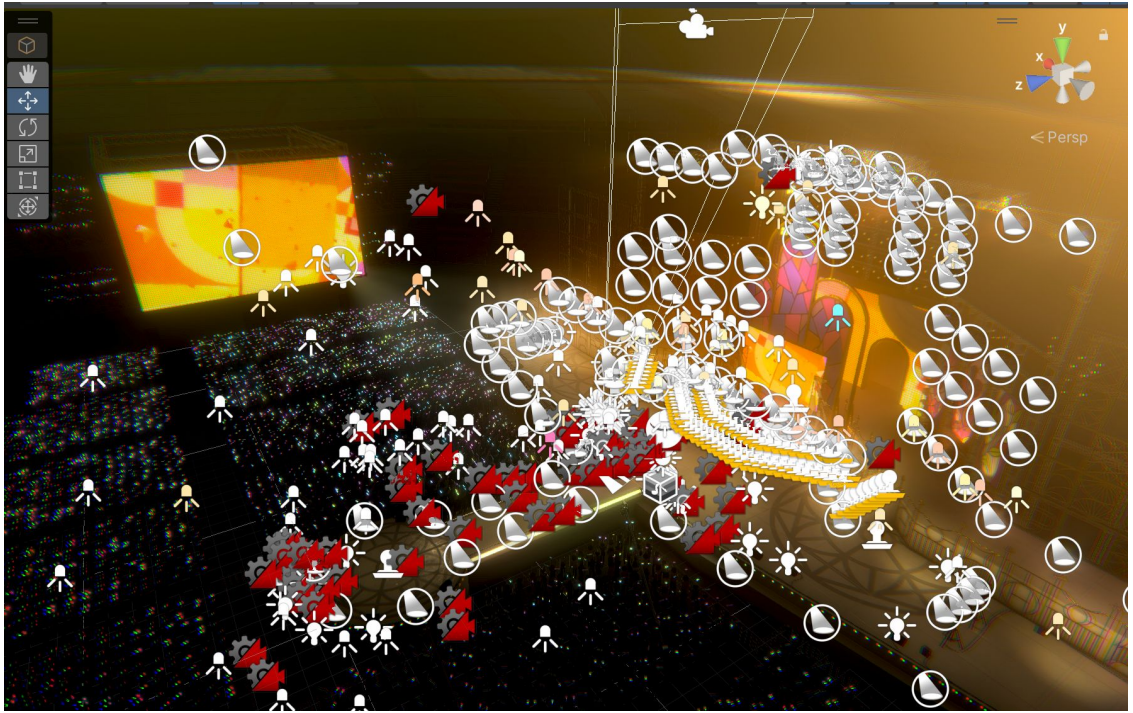
Reflection Probe

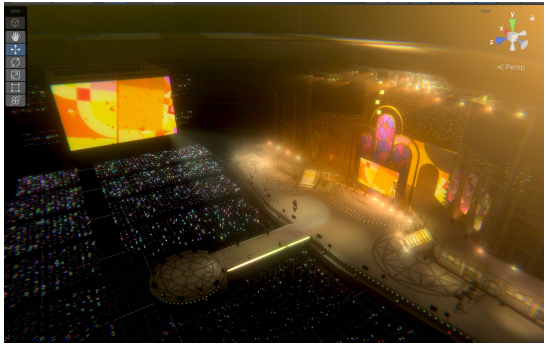


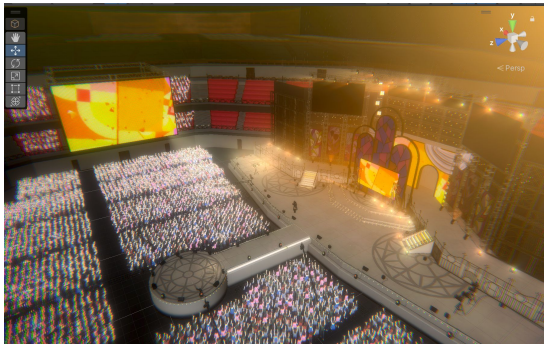
SSR + Planar Reflection +
Reflection Probe

ライティング

SceneView

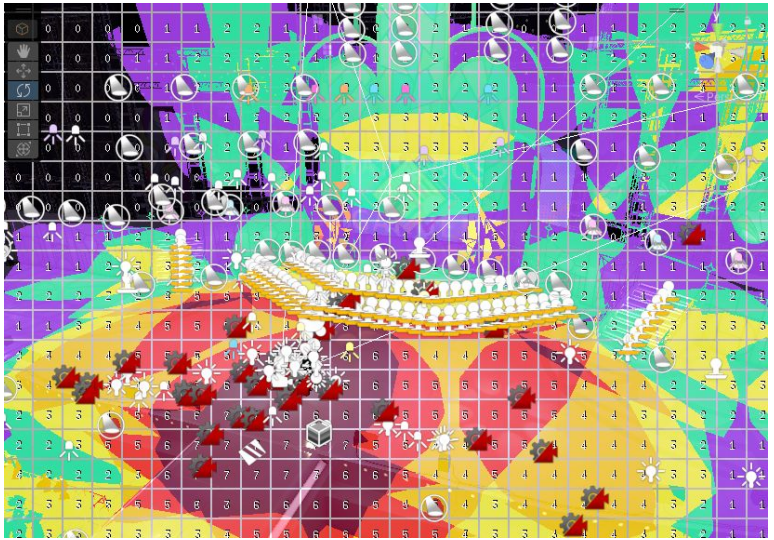






Unityライト配置ルール

- シーンには280個ほど存在
- 画面内のUnityライトは32個まで
  - LightCookieManagerが32個以上で描画バグが発生するので注意
- ライトが重なる数は4個程度を目標
  - デバッグで個数を確認できるように

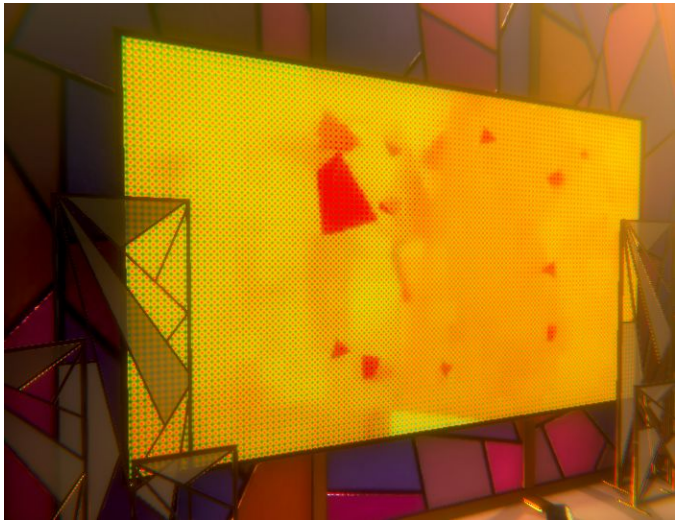


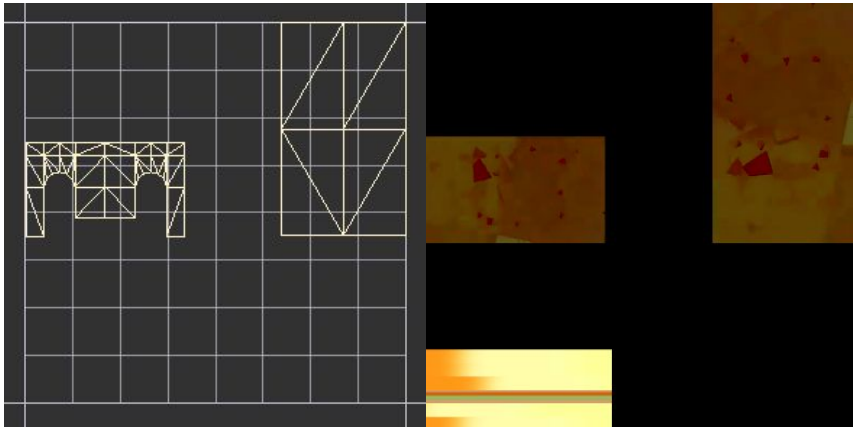
Monitorシステム

- 背景に配置されているモニターや発光メッシュの色を制御
- 専用のカメラでUIを撮影してテクスチャに出力
- HDRなので発光量もUI側で制御

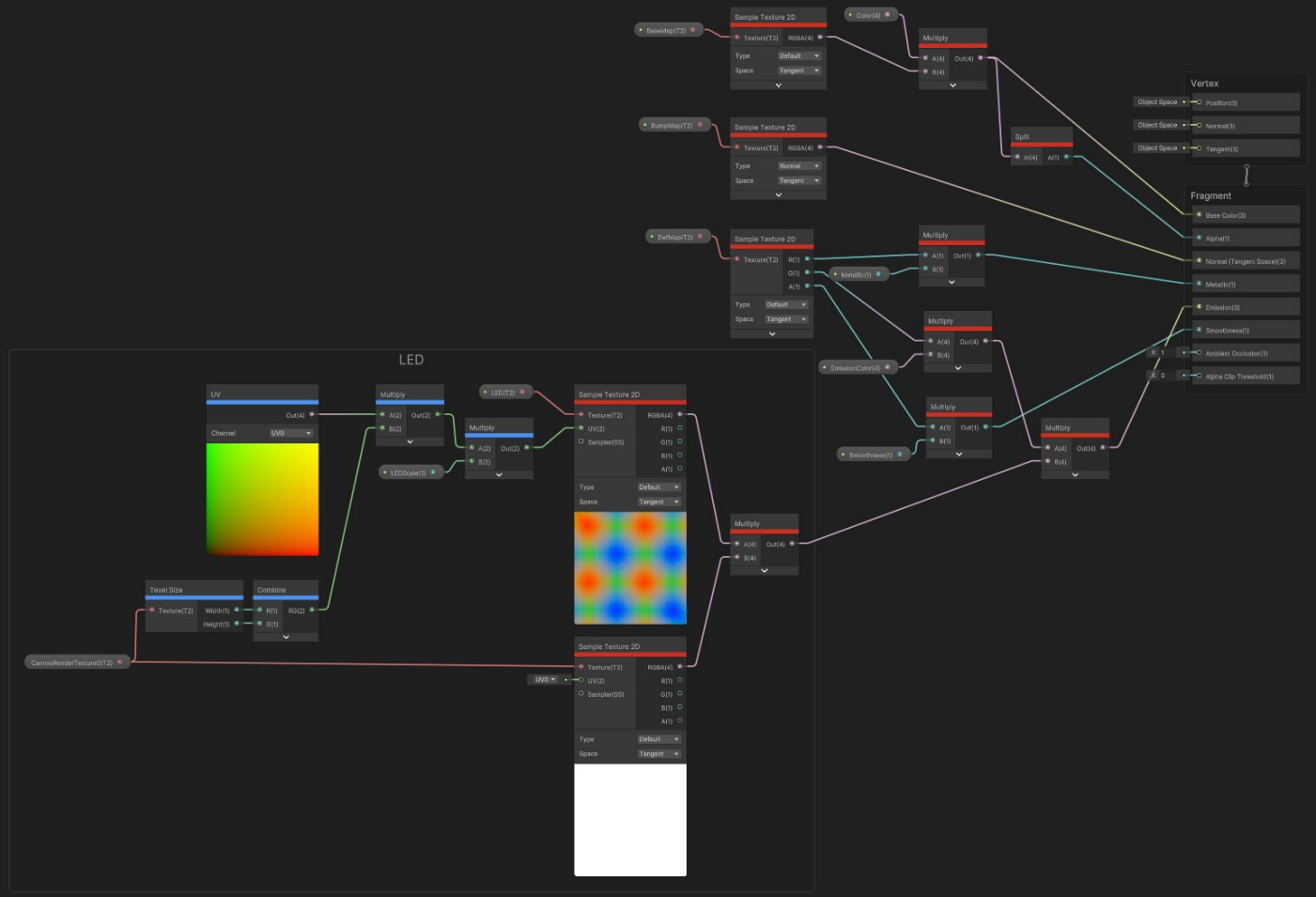
Emissionメッシュ

- Monitorテクスチャをサンプルして発光するメッシュ
- UVでサンプル箇所を指定



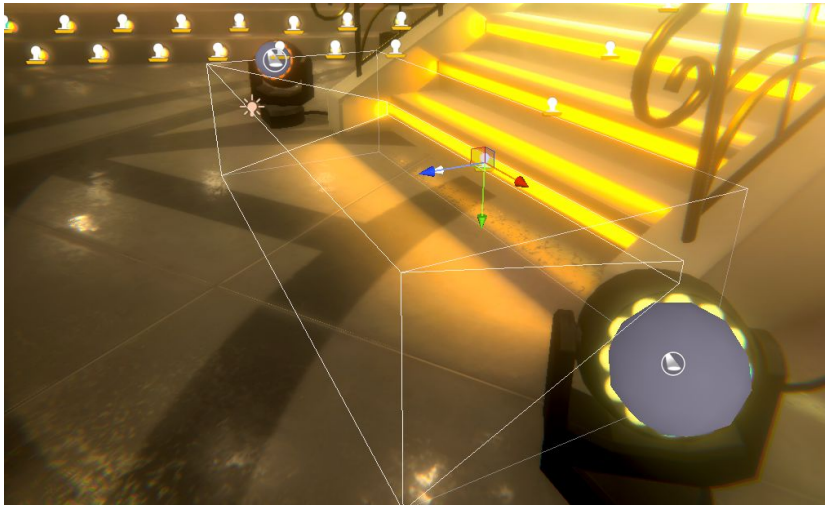


UV Layout



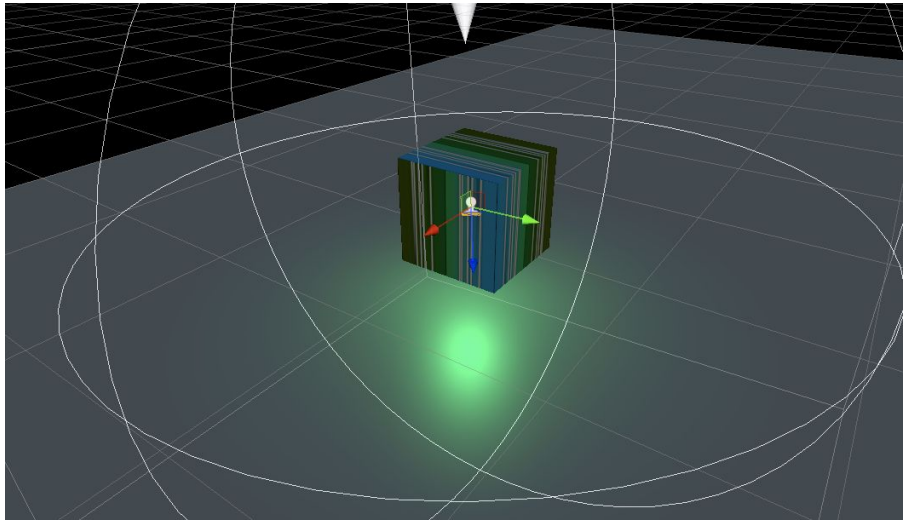
デカールライト

- Monitorテクスチャを使用して
  ライティングを行う
- Emissionメッシュとの連動が簡単
  - 初期配置のUV設定は面倒
- 計算処理はUnityLightと同等なので
  無駄を削ったライトのイメージ
- Instancingで一括描画
  （このシーンは110個程度）



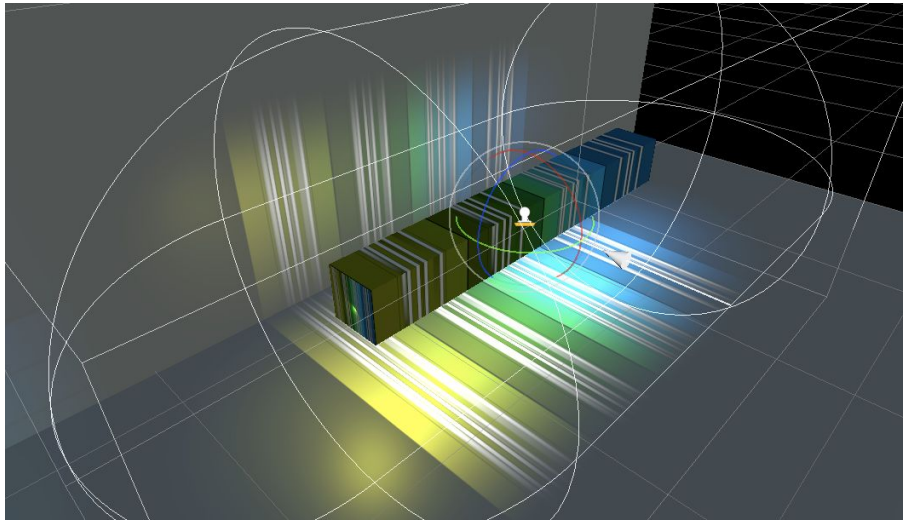
デカールライト (Point)

- Unityのポイントライトと同等
- Monitorテクスチャの1点をサンプル



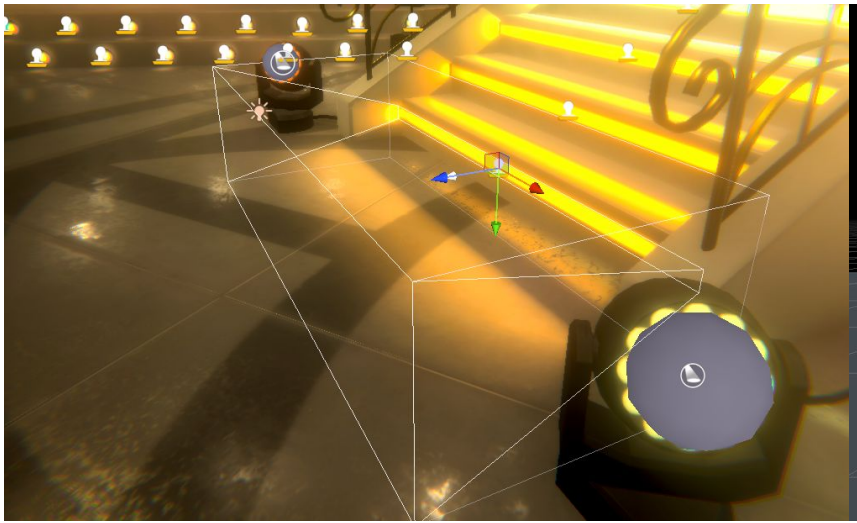
デカールライト (Capsule)

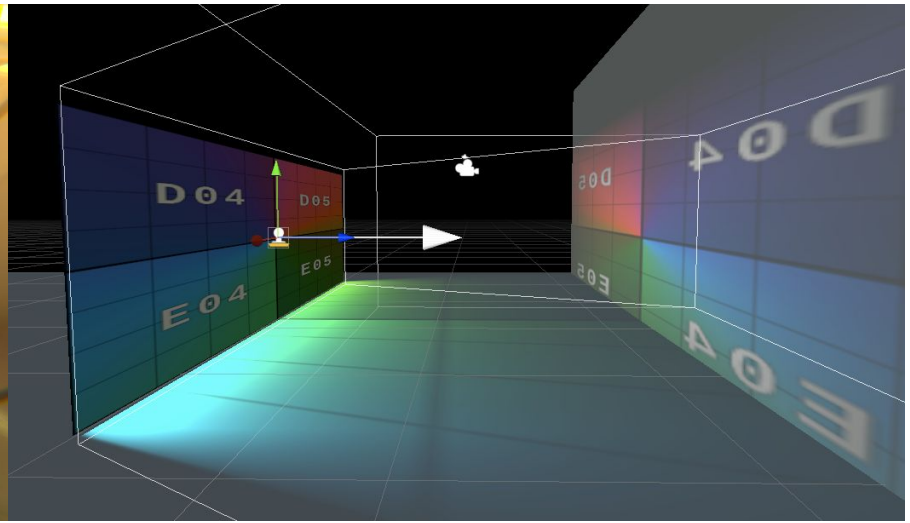
- カプセル形状のライト、線分から放射状に光を飛ばす
- Monitorテクスチャを線としてサンプル



デカールライト(Area)

- 面形状のライト、面から台形状に光を飛ばす
- Monitorテクスチャを面としてサンプル





機種依存問題

機種依存問題 iOS

- iOS全般
  - ReflectionProbe Atlasの拡張時にClear失敗する場合がある
  - ShaderVariantCollectionが大量のメモリを使用する（オフスクリーンレンダリングに）
  - 同一GraphicsBufferにSetDataを複数回呼ぶと都度別バッファが作成される
- A14 Bionic（iPhone12）
  - bitの範囲外チェックを厳密にする必要あり
  - UnityのForward+で多くのバグ発生（基本的に範囲外アクセス）
  - 画面回転時にクラッシュ（DPI Factorが特定の値のみ）
- A9, A10（iPhone7）
  - タイルメモリが最大256bit（32bitHDRは64bit扱いなので注意）

機種依存問題 Android

- Android全般
  - 一部halfの乗算で精度不足が発生して値が壊れる
  - 最大MultipleRenderTargetが4枚のためDeferred不可（2016年頃のGPU）
- Pixel6a, 7a
  - x==0の判定ができない x < 0.004に置き換え
  - Android14のバグでビットシフトが正常に動作しない（3月パッチで修正）
- Pixel8a
  - OSバグで負荷が高くなるとフレームが飛ぶ（6月パッチで修正）
- XperiaAceII
  - Vulkanドライバにバグがあり Unity側で起動拒否

まとめ

まとめ

- 次世代を見据えた新しいレンダリングパイプラインを構築できた
- ライティングとデカールにこだわることでモバイルでも情報量の増加につながる
- デカール方式のライトは様々な応用が可能
- 反射は物理的に正しくなくてもアニメ調の場合高品質に見える
- 問題が発生しやすい端末に注意
  - Snapdragon端末は常に安定
  - PixelはPixel6以降の全端末常にチェックした方がいい
  - iOSはiPhone12世代だけ挙動が違うので常にチェックした方がいい

ご清聴ありがとうございました

おまけ

- キャラクターをForwardにした理由
  - 最初はDeferredで実装していたがGBufferが足りなくなった
  - Forward+で実装したが、負荷が高すぎて実用的ではなかった
  - Forwardでさらにライト数を制限することで高速化に繋がった
- FSRについて
  - https://gpuopen.com/fidelityfx-superresolution/
  - ここに載っているUltra Quality, Quality, Balanced, Performanceに合わせてスケール調整を行なうと効果的だった
  - 公式で書かれているようにFSRは十分なAAをかけないと効果的ではなく、FXAAに掛けるとほぼ効果がないが、高品質なTAAであれば非常に高い効果がある
  - ノイズやジャギーを残すとそれも強調されるのでとにかく柔らかい画を作ることが求められる