BanG Dream! ガールズバンドパーティ!

# リアルタイム3Dライブを支える グラフィックとパイプラインの技術

Craft Egg　exsa　CyberAgent Group

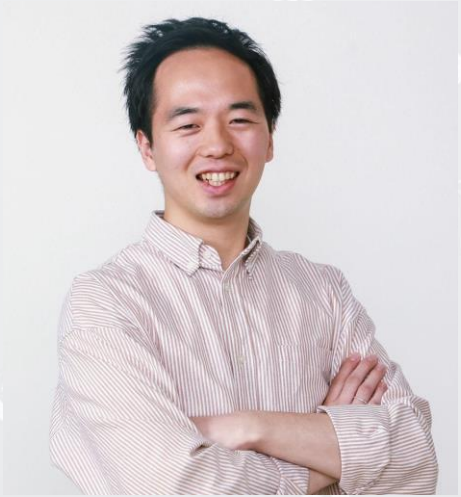
# 自己紹介

## 佐竹 雄紀

3D実装全般監修
各種レギュレーション策定
負荷＆メモリ対策

## コカー ジェローム

3DMV関連実装
各種表現、演出制御
MVシーン制作環境整備



## 江口 大喜

3DMV以外の3Dモデル制御
Timeline関連
Unity上のアセットフロー

# 「バンドリ！ ガールズバンドパーティ！」とは

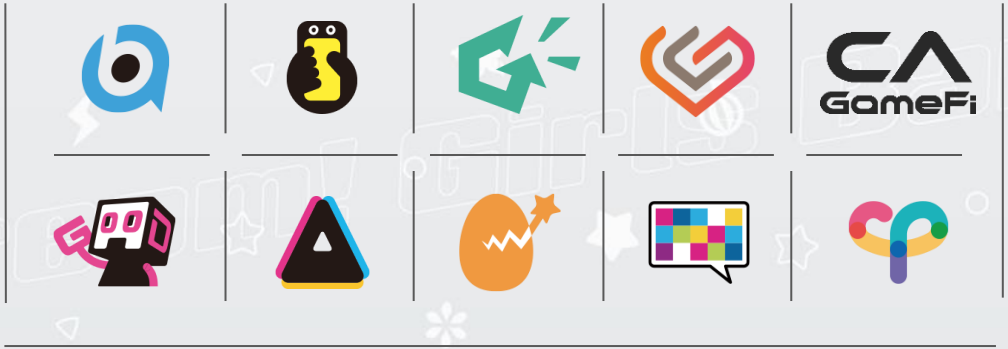
キャラクターとリアルライブがリンクする、
次世代ガールズバンドプロジェクト「BanG Dream!（バンドリ！）」
の世界観を軸に展開されるリズム＆アドベンチャーゲーム

Craft Eggが開発、2023/03にリリース6周年
今回3D開発協力＆アセット制作としてexsaが参加

# 「バンドリ！ ガールズバンドパーティ！」とは

# ゲーム・エンターテイメント事業部（SGE）について

CyberAgent




ゲーム・エンターテイメント事業部（SGE）

子会社制をとっており、
ゲーム・エンターテイメント事業に
携わる10社の子会社が
所属しています。

# アジェンダ

# 全体アジェンダ

1. 開発環境/3DMV概要
2. 3DMV：キャラクター表現
3. 3DMV：ライブ演出
4. 3DMV：パフォーマンスチューニング
5. 3DMV以外での3Dアセット活用

※　後日CEDiLにてスライド配布予定です

# 開発環境/3DMV概要

## 開発環境

Unity 2020.3.31f1, URP 10.8.1 にて開発
※ Unity 2021.3.22f1, URP 12.1.10 の対応も完了

## グラフィックス環境

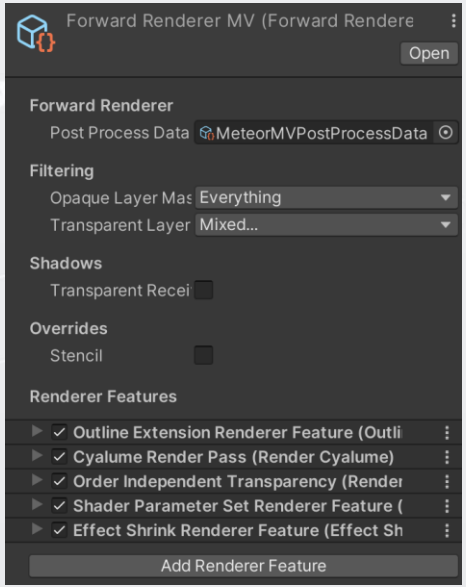
・HDR対応
・Linear色空間

→　もともとGammaだったが、今回のアップデートにて頑張って移行

# 開発環境概要

## 描画パイプライン

Forwardレンダリング
- URP標準で実装されているレンダラをそのまま使用
- Unityアップデート対応を考慮してカスタム実装は回避




## 描画フローのカスタマイズ

RendererFeature設定でできる範囲内で対応
各シーンに配置されるカメラの種類に応じたRendererDataを用意

→　3DMVカメラ用、アウトゲーム3D描画カメラ用、UI用描画用…など

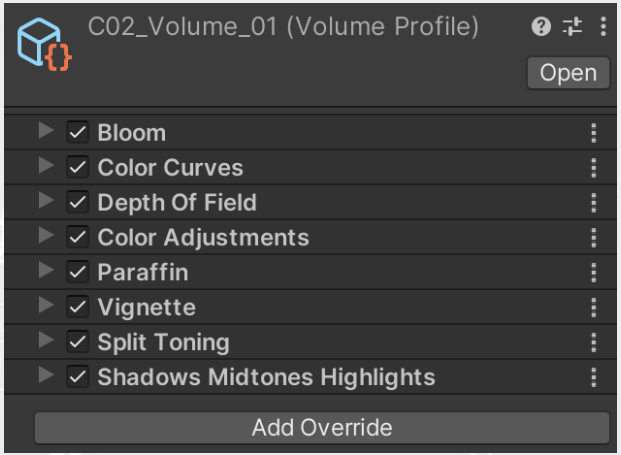
# 3DMV環境概要

## 主な描画関連要素

- キャラクター：5体
- Prop（楽器）：5つ
  - ○ 基本的に全キャラ（パート）に対応するPropが存在する
    - ■ ボーカルでもマイク＆マイクスタンドが対応Propになる
  - ○ バンドによって構成は異なる
- 背景：1つ
  - ○ MV中に異なる背景に切り替わることは想定していない
- カメラ：メインカメラ＆サブカメラ1つ
  - ○ サブカメラはステージLEDモニタに映すための描画カメラ
- ライト：キャラ、Prop、背景、それぞれ専用のライト
  - ○ ライトの種類はすべてDirectionalLight

# 3DMV環境概要

使用できるポストプロセス




- Bloom ※
- DOF（被写界深度）※
- 色調補正関連 ※
- 色収差 ※
- 周辺減光 ※
- レンズ歪み ※
- パラフレア（簡易的なグラデーションによる色味調整＆光源表現）
- 任意カラーのフェード
- UV反転（自撮りにおける端末画面の再現など、局所的に使用）

※はURP標準の機能をそのまま使用

# 3DMV環境概要

## 採用した外部アセット

- IK
  - Final IK
- クロスシミュレーション（揺れもの）
  - Magica Cloth
- エフェクト関連
  - Nova Shader
  - Volumetric Light Beam
  - ProFlares

※ 3D関連で使用しているもののみリストアップ

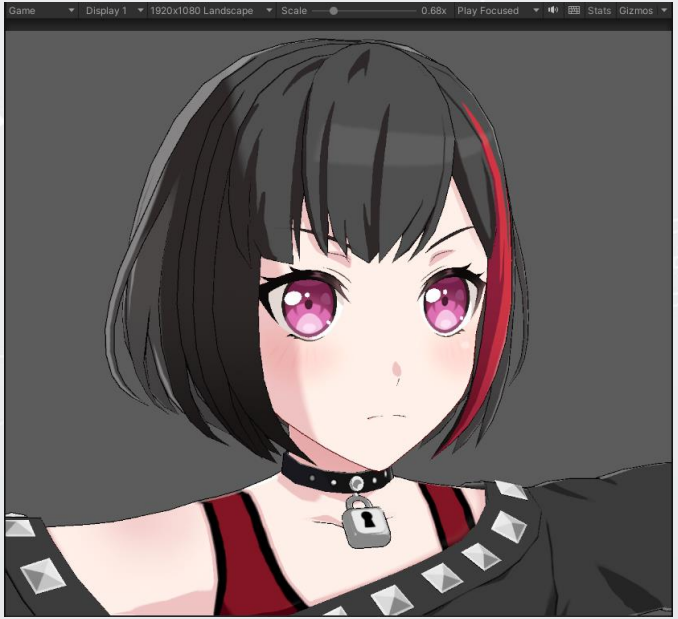
# 3DMV基本実装

## キャラクター関連

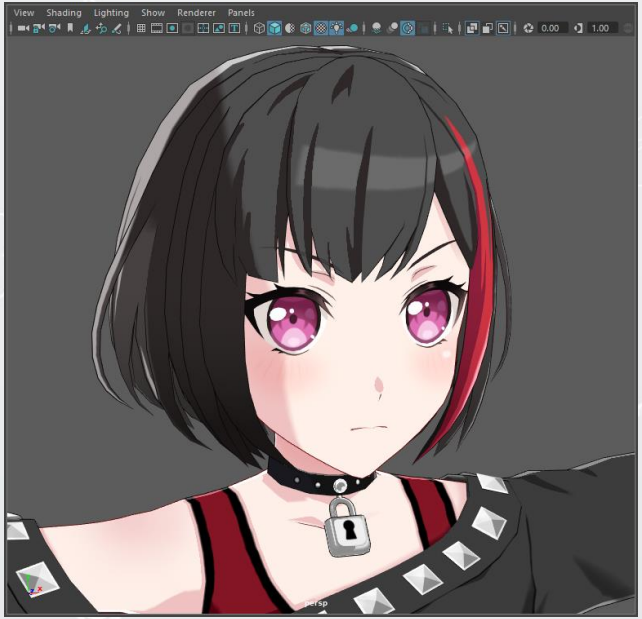
# キャラクターシェーダ

# キャラクターシェーダ：概要

## UnityシェーダとMaya DX11シェーダ同時開発




Unity




Maya

# キャラクターシェーダ：概要

制作⇒監修⇒リテイクはmayaのみで完結

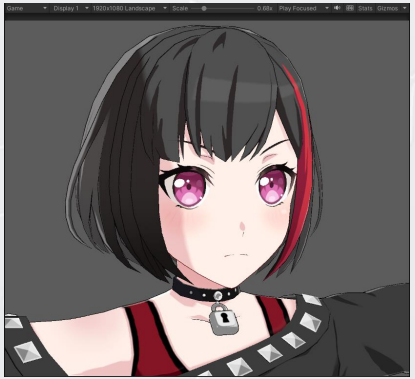
Unityデータ作成はQA開始タイミング

メリット：

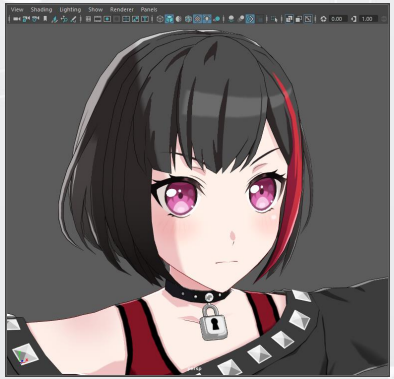
- Maya⇔Unityの往復を最小限に
- 監修フロー短縮

デメリット：

- メンテナンスコスト
- 一部の機能はmayaで再現が難しい



Unity



Maya

# キャラクターシェーダ：機能

キャラクター、背景、プロップに使われるセルシェーディング

- 基本シェーディング
- マスクによる影制御
- スペキュラー
- リムライト
- アウトライン
- 発光
- Matcapによる環境マップ
- ノーマルマップ
- 順不同半透明

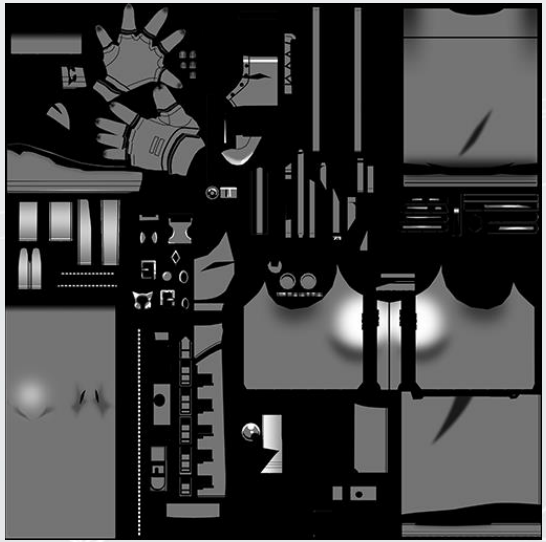
# キャラクターシェーダ：マスク構成

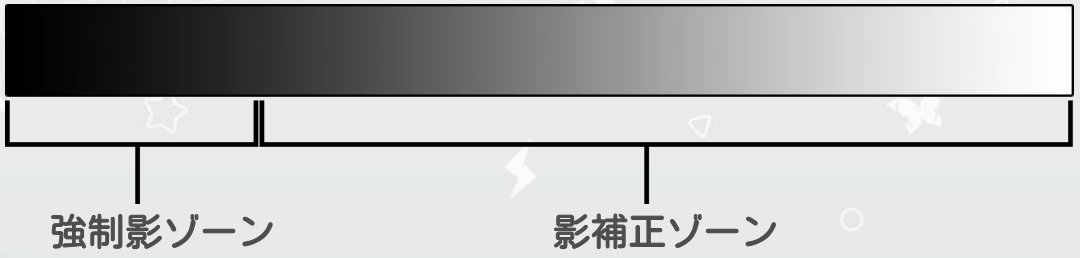
Maya: RGBテクスチャ×2＋マスク×6＝テクスチャ×8

# キャラクターシェーダ：影マスク

2つのモード：

- ノーマルモード
  - ○ 影補正ゾーン：ライト対法線の角度補正、暗くなると影が出やすい
  - ○ 強制影ゾーン：黒に近づくと影をブレンド（黒100%＝影）

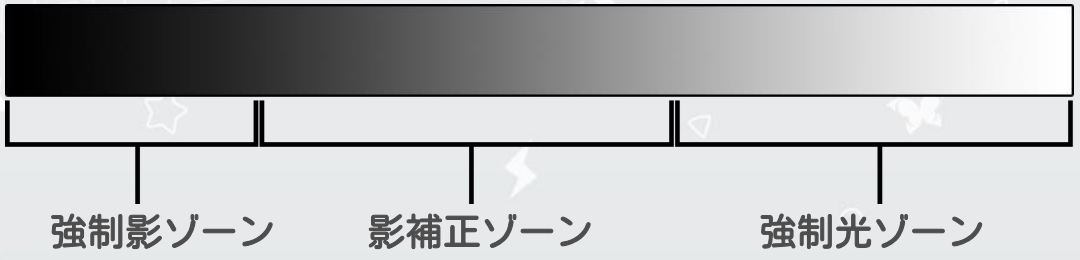




# キャラクターシェーダ：影マスク

2つのモード：

- 顔モード：
  - ○ 影制御はノーマルモードと同様
  - ○ 影エリアに強制光



# キャラクターシェーダ：眉

表現：

- 前髪の下に薄く眉を表示

描画：

- ２回描画することで実装
- １回目：通常不透明
- ２回目：アルファブレンド
- 頂点シェーダでオフセット

# 半透明について

## リアルタイム3D半透明の問題

- 重なるメッシュの描画問題
- セルフカバーリングの描画問題
- デプスによるポストプロセスの描画問題
- Linear/Gammaスペースで色が異なる、など

# Order Independent Transparency概要

順不同半透明はパフォーマンス的に不可能

近似：Weighted Blended Order Independent Transparency [McGuire and Bavoil 2013]

- ウエイトによる重なる半透明フラグメントのアベレージを計算
- ウエイトについて：
  - アルファ：透明度の影響
  - デプス：カメラに近いフラグメントのほうが影響が大きい

# Order Independent Transparency実装

MRTを使って半透明メッシュを2つのバッファに描画

- Accumulationバッファ：フラグメント蓄積
  - RGB x weight + RGB x weight + RGB x weight …
- Revealageバッファ：
  - 合計ウェイト、最終アルファ

最後にブレンドパス

lerp(source,フラグメント蓄積/合計ウェイト,最終アルファ)

DOFのためDepthのみパス

# Order Independent Transparencyデメリット

結果：

が・・・

- 重なるアルファ値が差があると結果が悪くなる
  - 特にOpaqueに近いアルファ
- 浮動小数を扱えるRGBAフルスクリーンバッファが２つ必要
  - メモリー負荷が高い
  - フィルレート
- 通常半透明と混ぜることは難しい
- MSAAで問題が発生しました

# 衣装入れ替え機能

# 衣装入れ替え機能

スケーリングシステムが必要

キャラクターモデルオーサリングについて：

- 全員155cmで作成
- スケルトン共通（マストではない）
- 胸サイズ用ブレンドシェイプ（S.M.L）
- 出力時に自動でボディ・ヘッドを分割

# スケーリングシステム概要

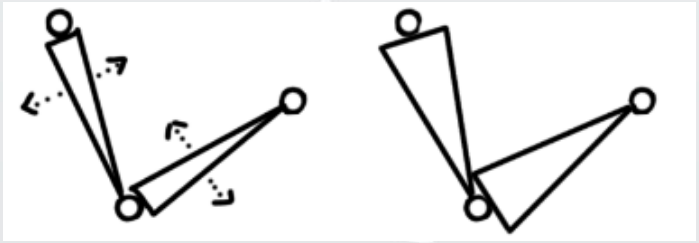
ランタイムでキャラクターロード後、スケール可能骨ごと：

- 骨のコピーする
- 子供をコピーに移動
- 元の骨もコピーに移動
- コピーを元の骨と同じ名前に変更

```
RArm
  RForeArm
    RHand
    RForeArm_Scale
```

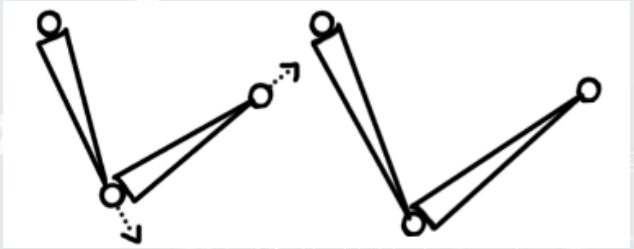
## スキニング用骨

スケールをかけても子供に反映されない



## アニメーション用骨

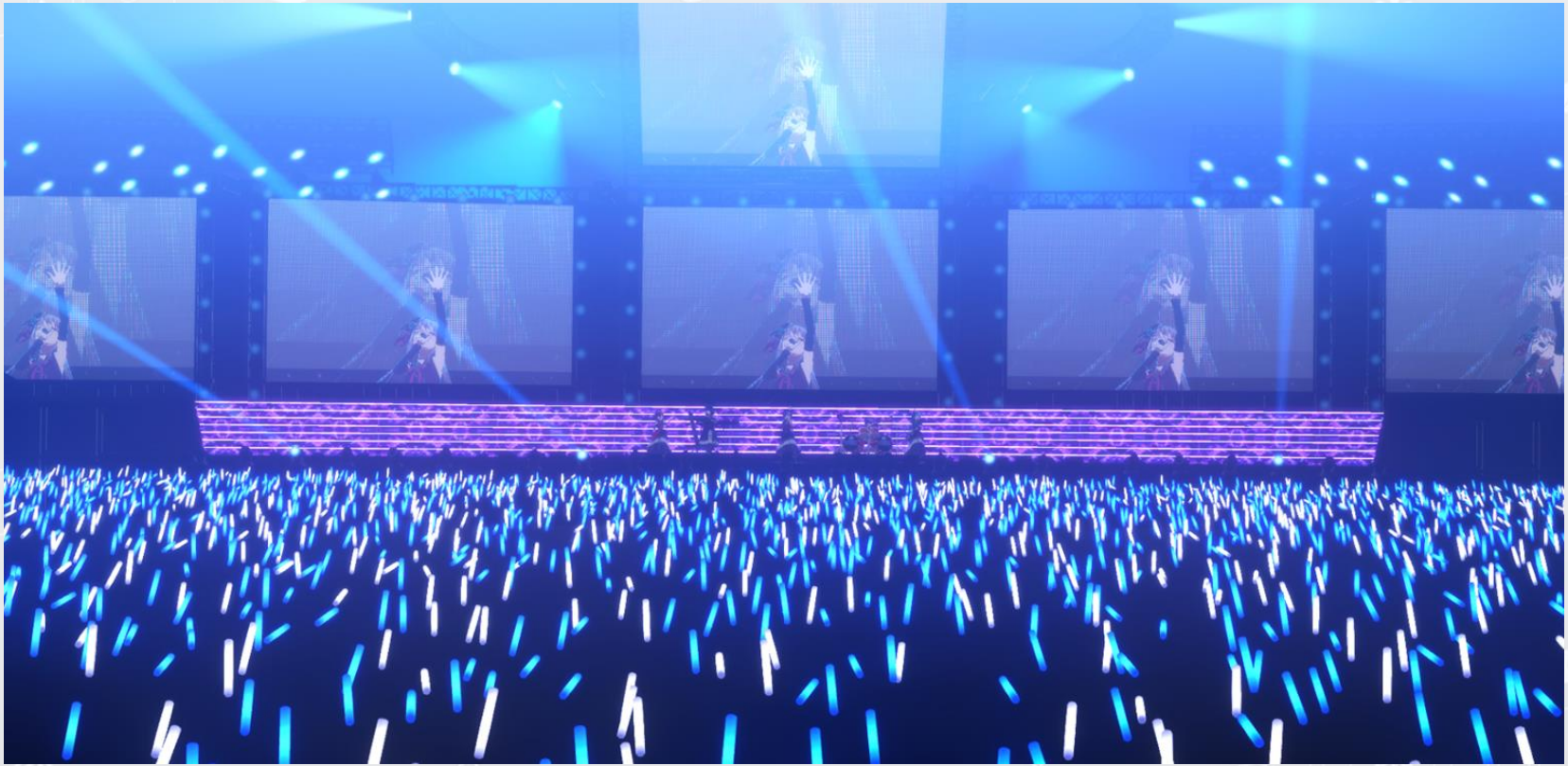
移動で長さ調整できる



# 3DMV基本実装

## ライブ演出関連

# サイリウムについて



# サイリウムについて

ニーズ：

- 軽い（パフォーマンス、メモリー）
- オーサリングしやすい

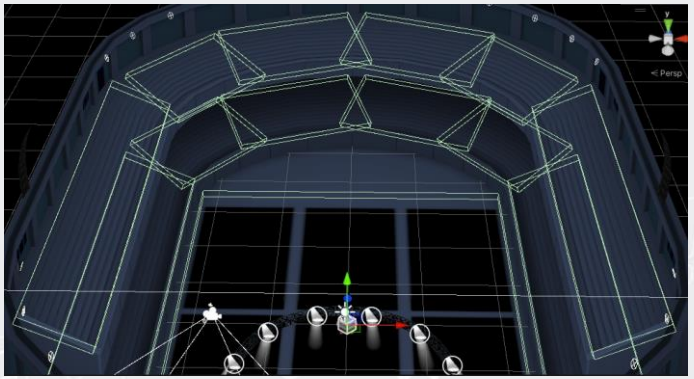
システム概要：

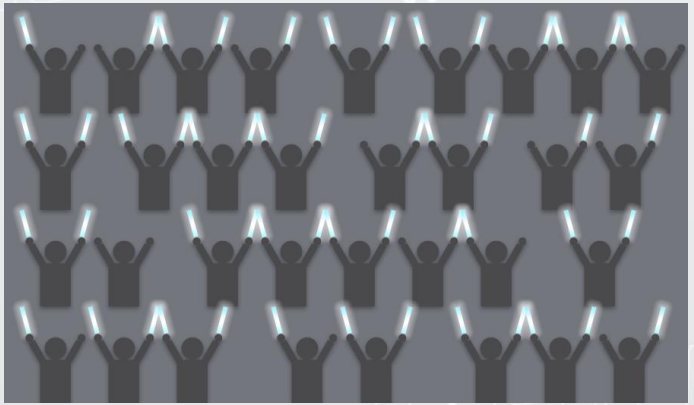
- 配置ルールによりランダム配置
- ローコストVATアニメーション
- タイムラインでシンプルな制御

# サイリウム：配置ルール

## ライブ会場を想像した配置ルール

- ボックスコライダーでサイリウムエリア指定
- 各エリア：
  - ターゲット（ステージ）に向けてＹ回転
  - Z値：席の列のようにある程度固定距離
  - X値：数人のグループ
  - 片手や両手ペア
  - 次の列で同じ対応...
  - ペアにわずかなミラー回転
- タイムラインで全体オフセット回転

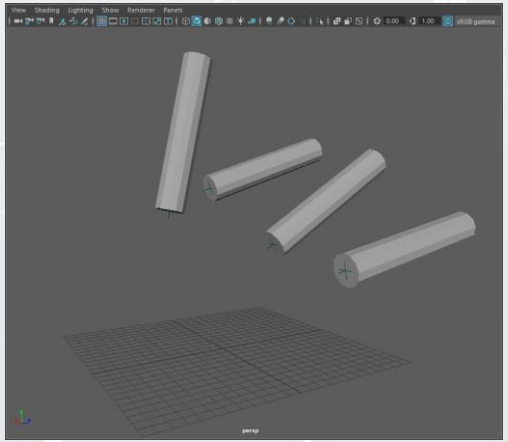


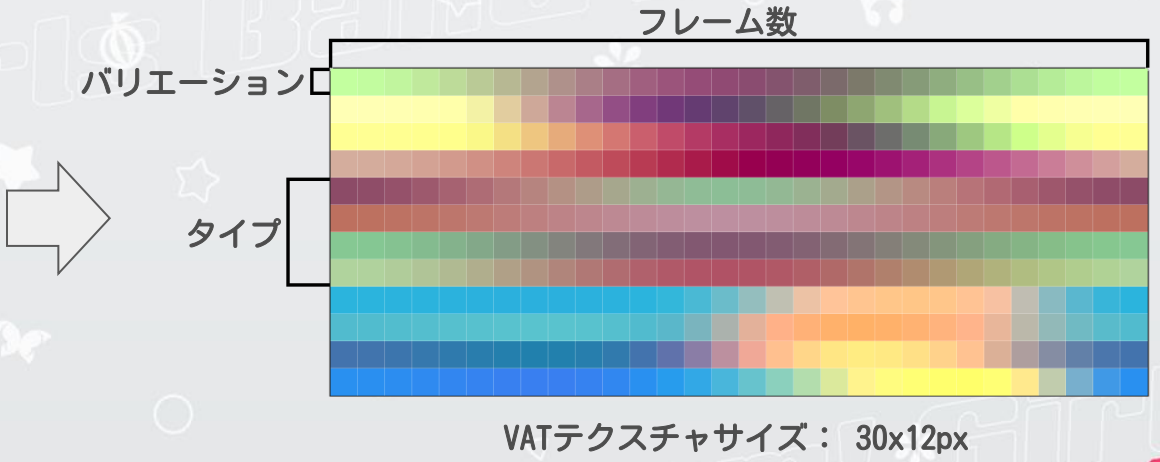


# サイリウム：アニメーション

## メモリーに優しいカスタムVAT:

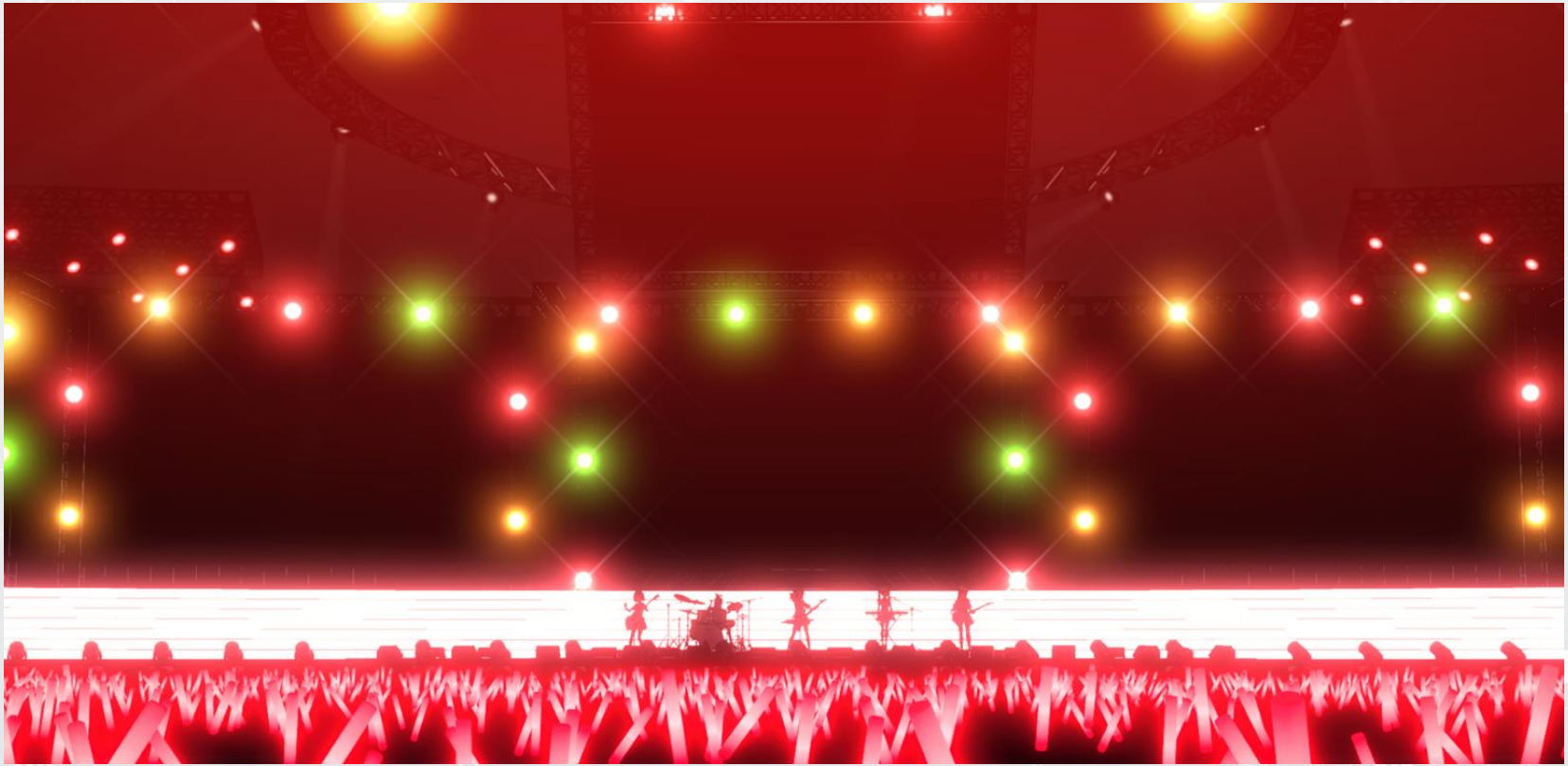
- ワールド位置とY軸回転は配置ルールで確定
- Z軸の回転は不要
- 残り：ZとYローカル位置とX軸回転
  - ○　1フレームあたりRGB1つ





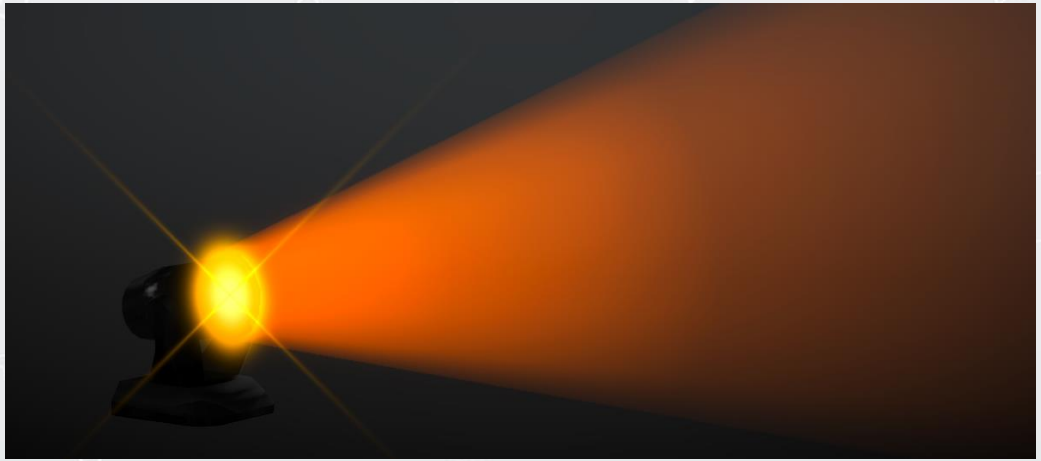

VATテクスチャサイズ： 30x12px

# ライトについて



# ライト構成

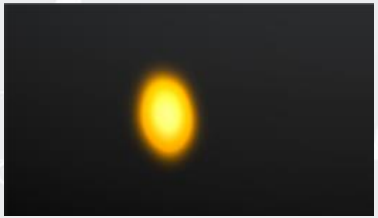
一つのスポットライト：



メッシュ

＋

ステージライト
光る部分

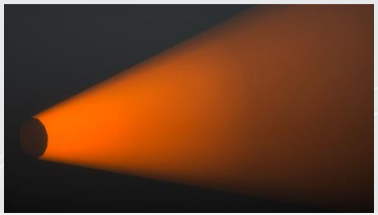


＋

レンズフレア
ProFlare

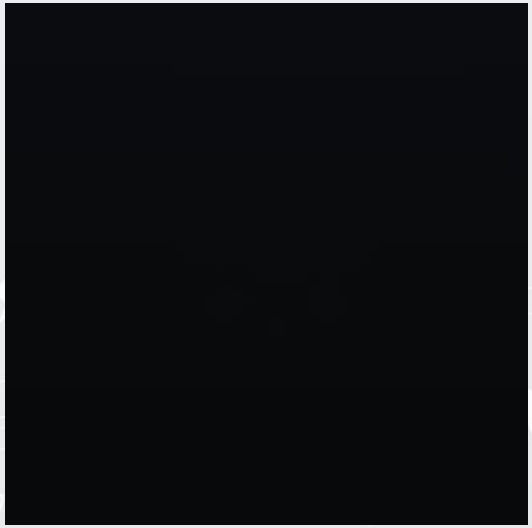
＋

ビーム
VLB

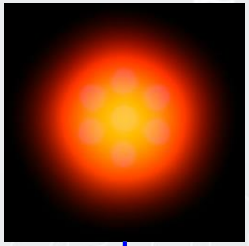


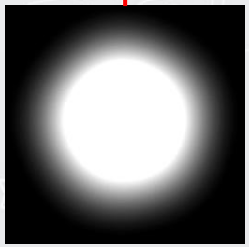
# ステージライトについて

インプット：RGBAカラー、インテンシティ、マスクテクスチャ

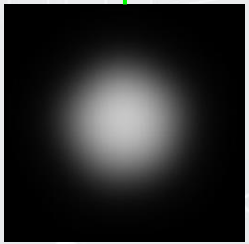


マスク構成：

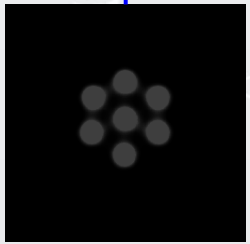




カラーマスク



Brightnessマスク



光ってない状態
ライトの残像効果

```
lerp(mask.b, (color.rgb * mask.r) + (intensity * mask.g), color.a) * color.a
```

# スポットと影について

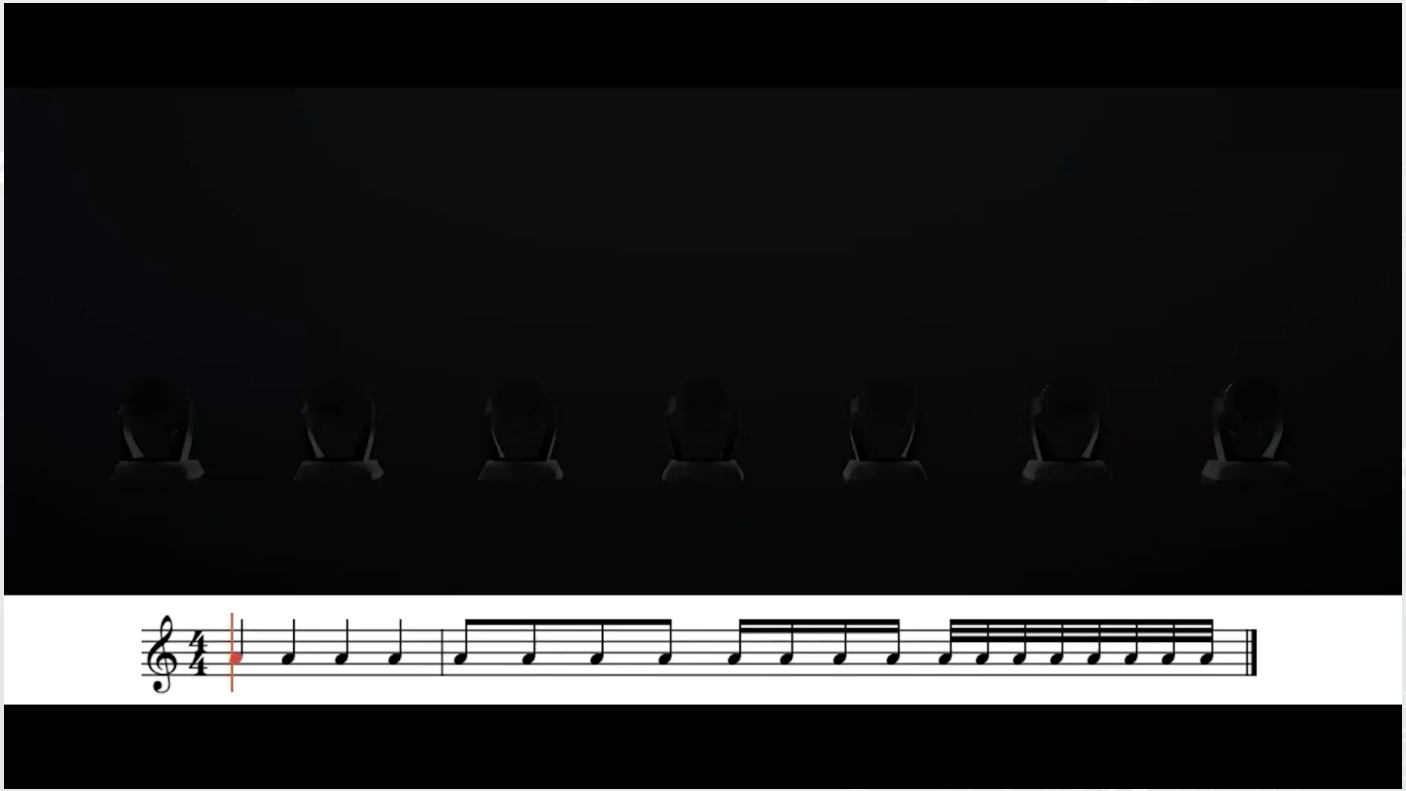
- 足元のスポットライト、クロス影はScreen Space Decalで実装
- ステンシルマスクを使用してキャラクター上の描画をスキップ
- 影の位置は両足と連動
- 足と地面の距離によってスケール

# ライトアニメーション

- ライトをグループでまとめる
- グループにアニメーションを行う
- 曲とシンクロナイズ：
  - タイムラインでシンクロマーカーを設置
  - テンポからアニメーションスピードを計算

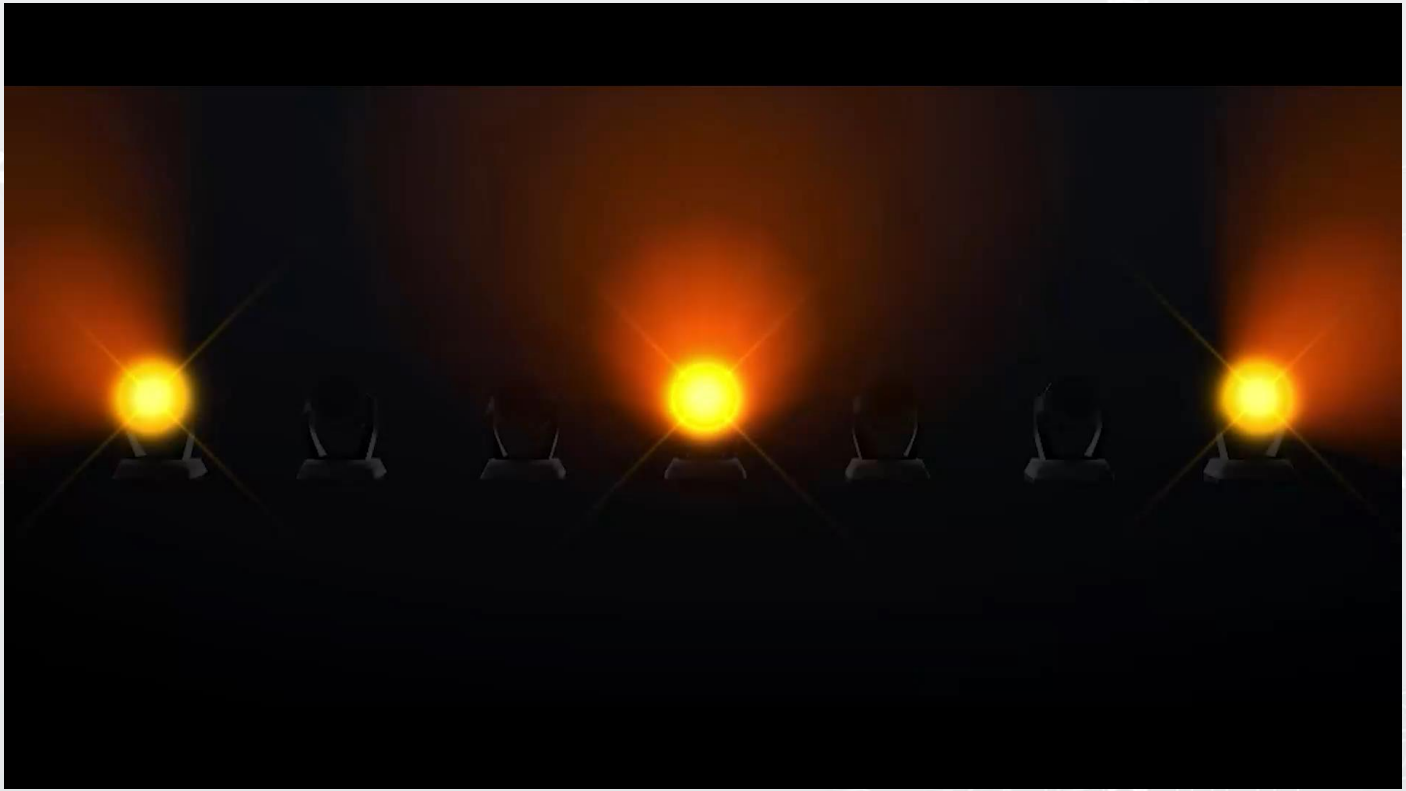
# ライトアニメーション:設定方法

①テンポ：ライトパルス間隔と長さを譜面の小節の中のビートのように管理



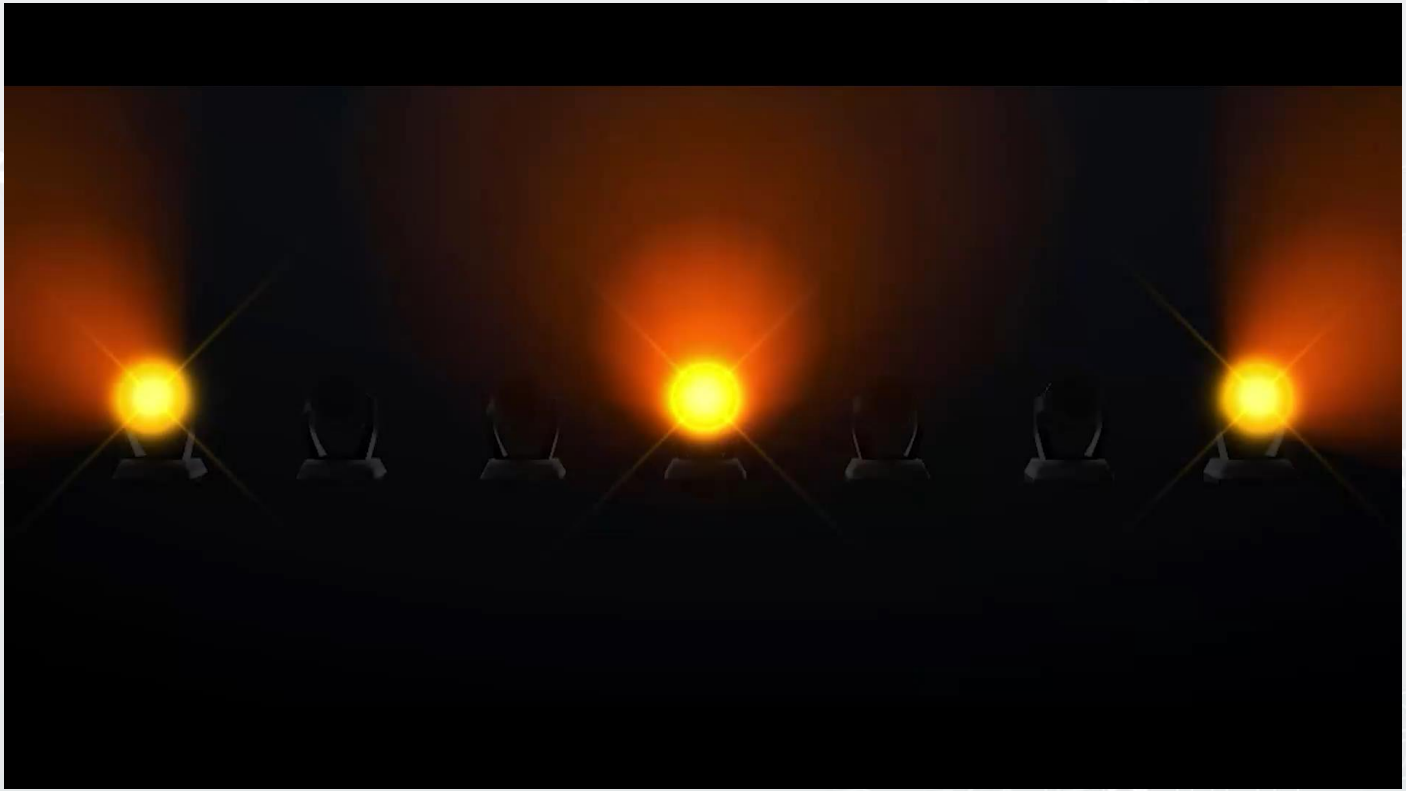
# ライトアニメーション:設定方法

②パターン：パルスごと何個のライトが光るかを設定



# ライトアニメーション:設定方法

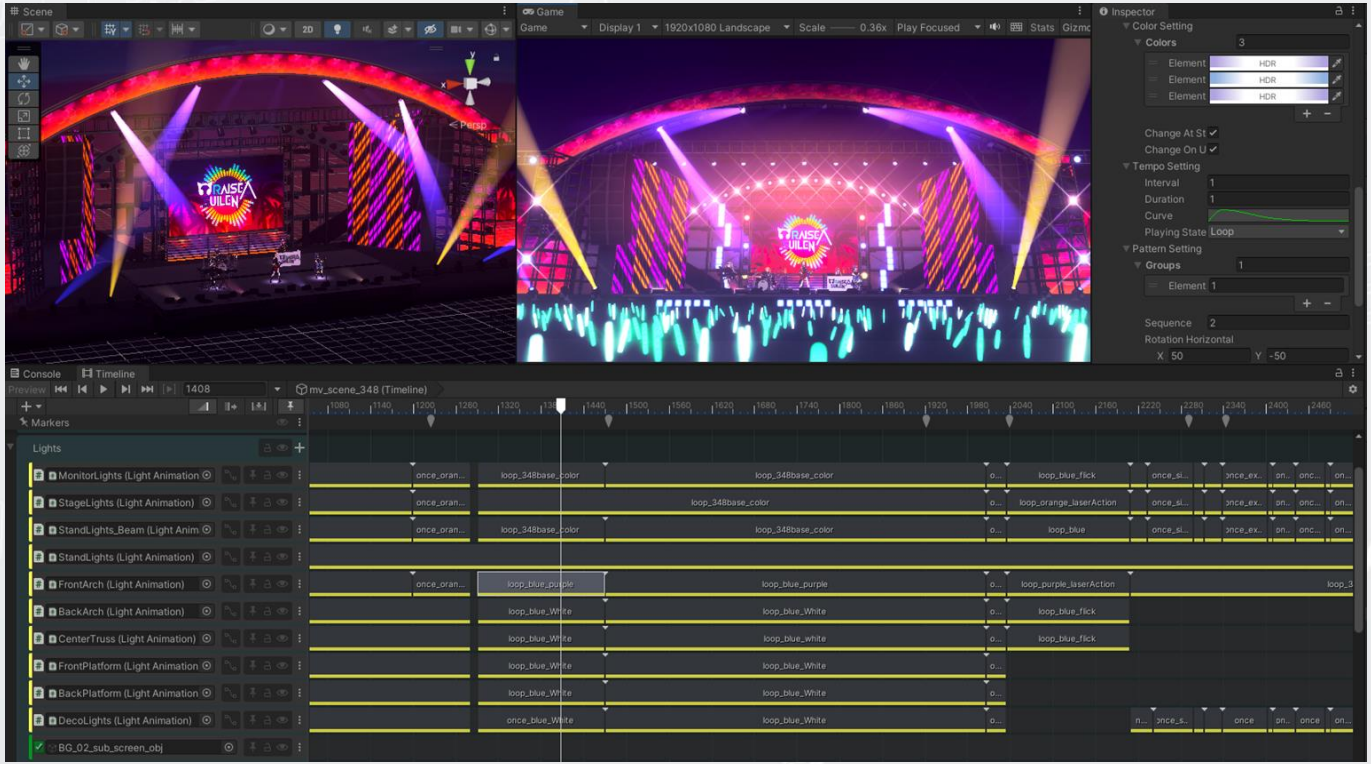
③カラー：ライトやグループごとのカラーパターンを設定



# ライトアニメーション:設定方法

④回転を設定

# ライトアニメーションの制御



# バンド構成変更について

# バンド構成変更について

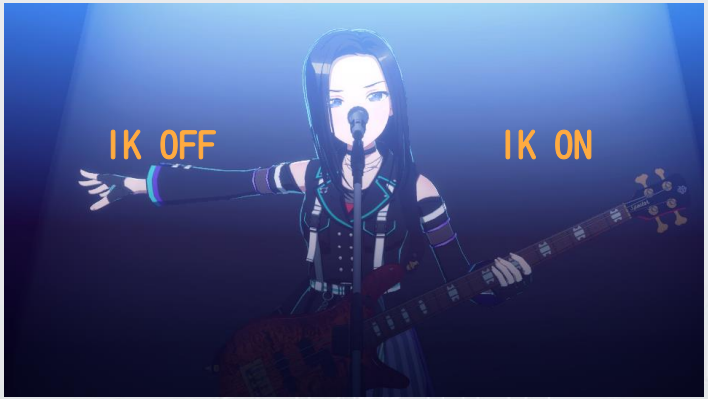
どのキャラクターでも、バンドのどのポジションでも設置可能

注意点：

- 身長差分あってもどの楽器でもプレイ可能
  - ○ シンバルが届かない
  - ○ 指が鍵盤やフレットとずれている
- 身長差分によるカメラレイアウト崩れ
  - ○ 高身長キャラクターの頭が画面外
- ミッシェルは大丈夫か？ 

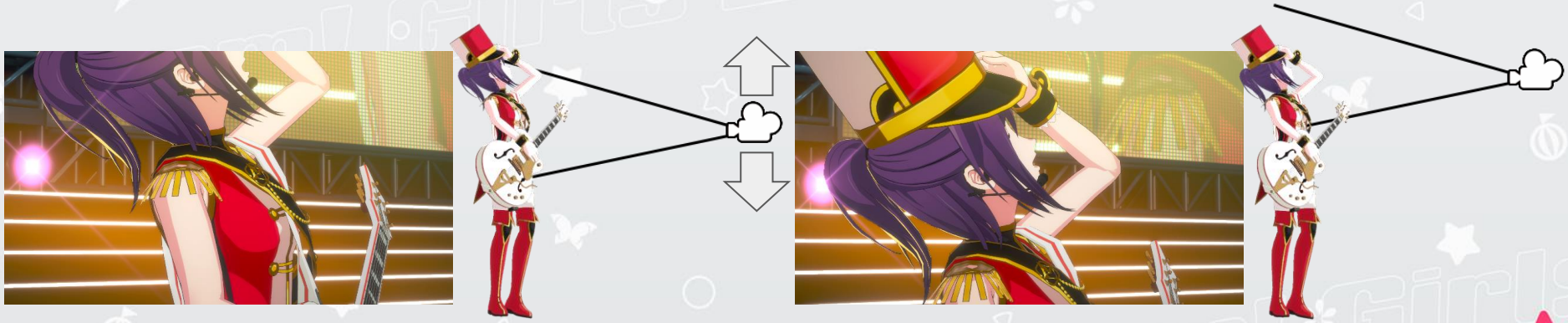
# バンド構成：楽器

- 手の位置と回転を楽器アニメーションにベイク
- ランタイムでIKを使ってベイクした位置に移動
- 場合によって、楽器にスケールをかける
- カスタムトラックでキャラごとの右手左手のIK具合を調整
- IKオーバーライドで特殊なインタラクション対応
  - ○ 手でスタンドマイクを持つ
  - ○ キャラクター同士で手合わせ

# バンド構成：カメラレイアウト

- 身長差分によるカメラの高さ補正
- カスタムトラックでメインフォーカスのキャラクターと補正率調整
- フォーカスキャラクター切り替え時にアニメーション可能
- 高身長、低身長、ミックスパターンで全カットを確認

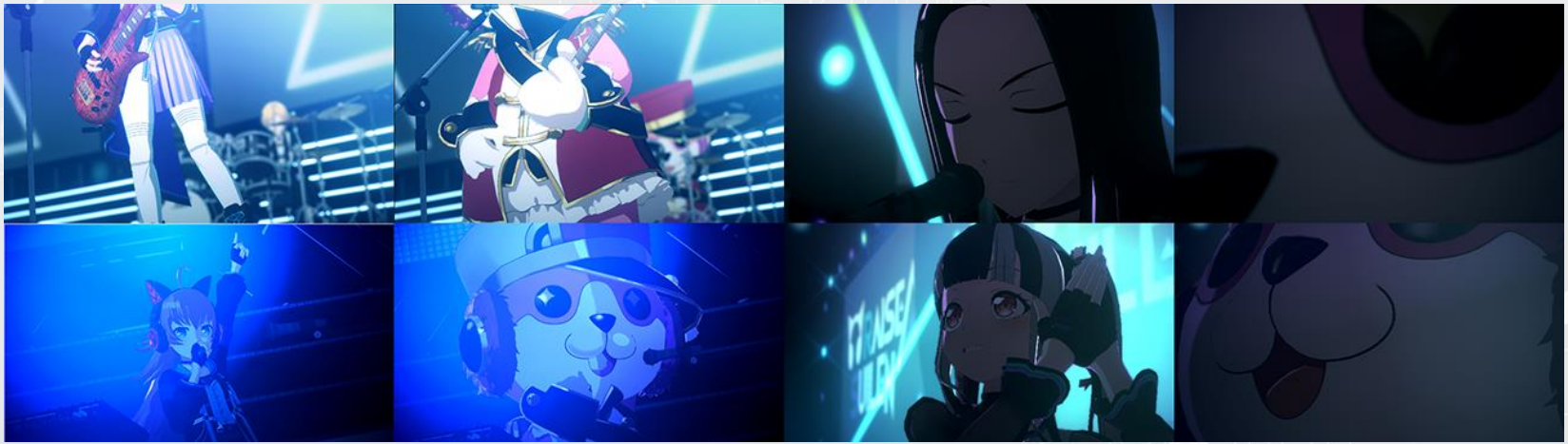


# ミッシェル問題

体型が大幅に違う着ぐるみキャラクターミッシェル

そのままだと様々な問題がある：

- ギターが体の中
- 手が頭の中
- クローズアップカットで何も見えない



# ミッシェル問題：ギター・ベースの対応

キャラクター側でギターの位置アニメーションで２つのダミーを利用

Hip骨以下、腰当たりの位置

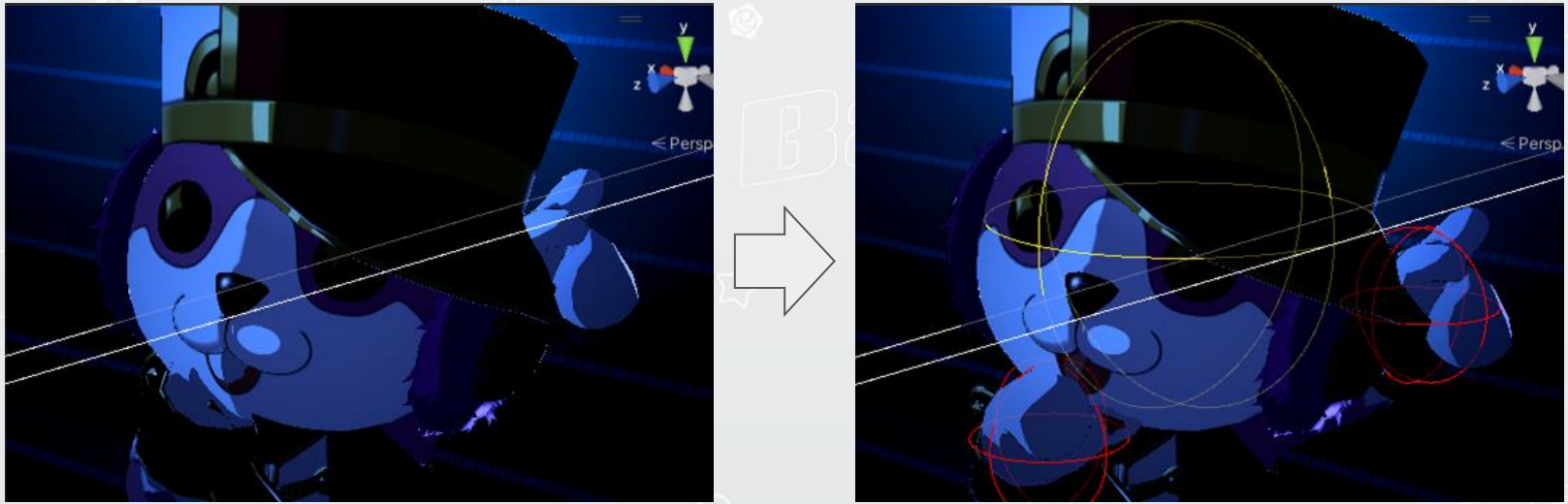
直接ギターをアタッチするダミー

⇒間にアニメーションがないアジャスト用ダミーを追加

# ミッシェル問題：手と頭の貫通の対応

頭に特殊コリジョンを実装

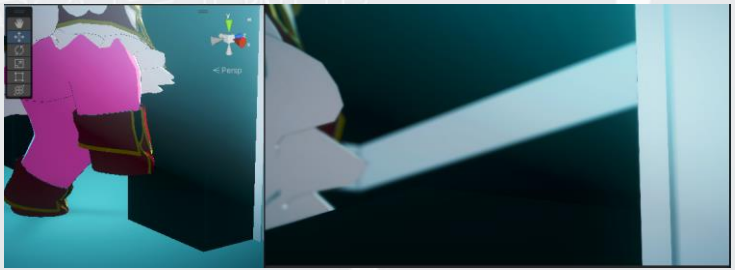
アニメーション更新、IK更新の後に実行

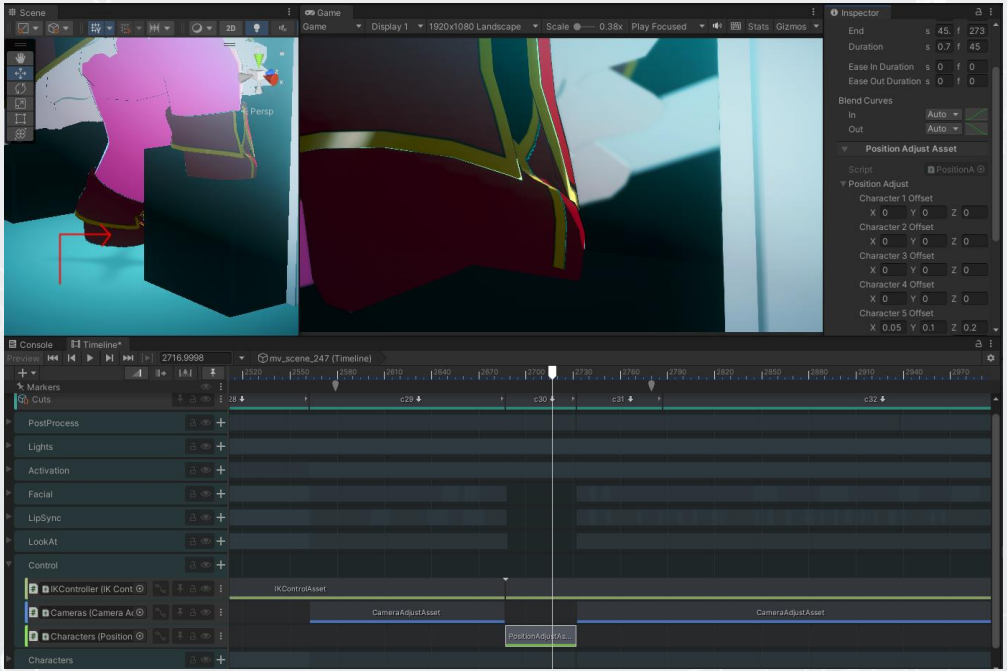


# ミッシェル問題：カメラレイアウト対応

カスタムトラックで位置オフセット

スケール調整も可能





# パフォーマンスチューニング

# パフォーマンスチューニング：アジェンダ

- フレーム分割レンダリング
- エフェクト縮小バッファ
- 品質設定について

# パフォーマンスチューニング
## フレーム分割レンダリング

# 本タイトルにおける負荷対策

ゲーム内容が「リズムゲーム」

→　フレームレートが重要（60fps）

→　60fpsを維持しつつリズムゲームの背景として3DMVを流す

動作保証目標端末で60fpsを維持するのは厳しい

→　正攻法の負荷対策だけではおそらく無理

→　3DMVのフレームレートのみ下げる

→　フレーム分割レンダリング

# フレーム分割レンダリング：導入

2フレーム使って30fpsで3Dシーンを更新する形に

- 分割するにあたって一番自然
- 3フレーム以上に分割すると、処理の分け方が難しい
- 3DMVディレクターの意見
  - ○ 60fpsだと滑らかすぎて少しアニメ感が薄れる
  - ○ 「バンドリ！のアニメを再現する」
    ということを考えるとむしろ30fpsにするべき
  - ○ 品質設定周りでもこの意見が反映されています

# フレーム分割レンダリング：導入

時間軸 ——▶

# フレーム分割レンダリング：概要

1フレーム目：メインカメラのジオメトリ描画（不透明、半透明）
→　ジオメトリフェーズ

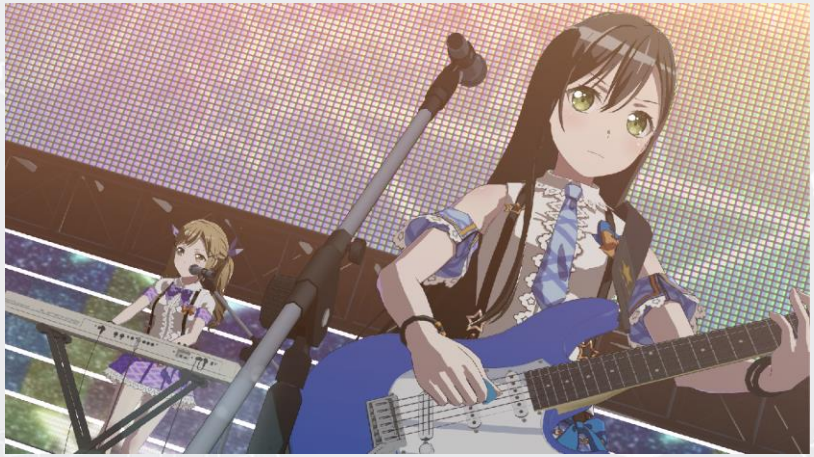
2フレーム目：メインカメラのポストプロセス描画＆サブカメラ全体描画
→　ポストプロセスフェーズ

※　描画負荷に応じた動的な処理分配はできていない
→　2フレーム目に引き継ぐ描画情報が不定になるため難しいと判断
→　メインカメラ描画の処理分割としては最も単純
→　ただしネックになりがちなGPU負荷対策としては効果大

# フレーム分割レンダリング：全体像

- 各描画フェーズ用のカメラを用意する
  - ○ 毎フレーム、対象カメラの有効無効を切り替えて描画フローを制御
- 3Dシーンを画面表示するためのテクスチャ（3Dシーンテクスチャ）は、ジオメトリフェーズでは更新しない
  - ○ ポストプロセスフェーズに渡すための一時テクスチャを更新するだけ
  - ○ 更新されなかった3Dシーンテクスチャは前のフレームの状態のまま表示される
- ポストプロセスフェーズでは、ジオメトリフェーズで使用された描画パラメータを用いて描画する
  - ○ 描画パラメータ＝描画処理に使用される全パラメータ
    - ■ カメラやポスプロなどの各種設定
  - ○ ここが一致していないと分割レンダリングにならない

# フレーム分割レンダリング：描画経過画像



ジオメトリフェーズ

ポストプロセスフェーズ

# フレーム分割レンダリング：留意点

実装時に色々試行錯誤した点

- カメラパラメータのコピータイミング
- 描画フェーズに応じたポストプロセスパラメータ更新制御
- ポストプロセスフェーズにおける
  ジオメトリ描画時の深度バッファ引き継ぎ

# フレーム分割レンダリング：留意点

## カメラパラメータのコピータイミング

- UnityのLifeCycle関連メソッドはタイミング的にどれも合わない
  - ○ Timelineのカメラアニメクリップのメインカメラ適用後にコピーする必要がある
  - ○ LateUpdateやEndOfFrameなどもダメ
- 描画フェーズに応じたコピー処理はLifeCycle上で行えない
  - ○ 「ジオメトリ描画用カメラの描画直前にコピー」
    ということができればズレは発生しないはず
- RenderPipelineManager.beginCameraRenderingを使用して解決
  - ○ URPではOnPreRender等は呼び出されないので注意
  - ○ 引数に入ってくるカメラ情報を元に、ジオメトリ描画用カメラの場合のみコピー

# フレーム分割レンダリング：留意点

## 描画フェーズに応じたポストプロセスパラメータ更新制御

- 実際にポストプロセス描画パスに渡るのはVolume設定

- VolumeはTimelineカスタムトラックにてパラメータ制御している
  - ○ ポストプロセスフェーズではVolumeを更新しないようにする
  - ○ これだけだとカットチェンジ時に対応できない
    - ■ カットチェンジにおいてVolumeProfileが切り替わっているため

- 一つのTimelineClipに現Clipと直前Clipの2つのVolumeProfileを持たせる
  - ○ Behaviour処理開始（Clip開始）がポストプロセスフェーズだった場合、直前ClipのVolumeProfileを使う
  - ○ 次のフレームで正常な状態（現ClipのVolumeProfileが有効）になるように

# フレーム分割レンダリング：留意点

## ポストプロセスフェーズにおけるジオメトリ描画時の深度バッファ引き継ぎ

- ジオメトリフェーズの_CameraDepthTextureはそのまま使えない
  - ○ MSAA無効時、ポスプロ有効＆DOF有効だとCopyDepthPassが勝手に走る
    - ■ 何も描画されていない深度バッファが_CameraDepthTextureにコピーされる
  - ○ MSAA有効時はCopyDepthが無効
    - ■ 代わりにDepthPrepass（DepthOnlyPass）が有効
    - ■ この描画パスは不透明描画前に実行
    - ■ この描画パスのOnCameraSetupにて_CameraDepthTextureがクリアされる
- よって、ジオメトリフェーズの深度バッファを別の一時バッファに退避、ポスプロフェーズの特定タイミングにて_CameraDepthTextureに戻す
  - ○ 戻すタイミングはポストプロセス描画パス実行直前

# パフォーマンスチューニング
## エフェクト縮小バッファ

# ワーストケースにおける負荷対策

フレーム分割レンダリングにおいて、
ジオメトリフェーズの負荷の方がボトルネックになることが多かった

→　特にレンズフレアやライトビームなどの半透明エフェクトが多い場面

→　画面の半分以上を覆う形のエフェクトが配置されているケースも
　→　ポストプロセスで対応できない形の色調整のため

→　ジオメトリフェーズ内でこの辺に何か対策が必要

→　エフェクト縮小バッファ

# エフェクト縮小バッファ

該当ワーストケースに手っ取り早く対応したい

→ 縮小バッファ上に描画面積の大きいエフェクト群をまとめて描画、
その縮小バッファをシーンに描き戻すような描画フローを用意

→ 該当エフェクトがすべて加算の描画物だったため、
一般的な縮小バッファ対応ではなく簡易版での対応としている

→ 通常アルファブレンドエフェクトも含めて対応させる場合
縮小バッファ描画用の専用エフェクトシェーダと
縮小バッファをシーンに描き戻すための特殊ブレンドシェーダが必要

# エフェクト縮小バッファ：適用前画面

# エフェクト縮小バッファ：対象エフェクト

# エフェクト縮小バッファ：対象エフェクト

# エフェクト縮小バッファ：対象エフェクト

# エフェクト縮小バッファ：品質比較（縮小バッファ無し）

# エフェクト縮小バッファ：品質比較（スケール0.5）

# エフェクト縮小バッファ：品質比較（スケール0.3）

# エフェクト縮小バッファ：まとめ

- **描画面積の大きい対象描画物が増えるほど負荷削減効果大**
  - ○ 画面全体の1/4サイズのメッシュが10枚ある場合、
    縮小スケール0.5なら25%程度の描画ピクセル数削減になる
  - ○ もっと多ければより大きい効果が見込める

- **対象描画物が少ない場合、逆に負荷が増える**
  - ○ 縮小バッファを適用する描画フローは固定の描画コストが掛かる
  - ○ ただし、対象が少ない＝おおよそワーストケースではない、なので
    負荷が増えてもあまり問題にならない
  - ○ 効果がない場面では縮小バッファを無効にできるような描画フロー整備ができればベスト

- **元の描画解像度が小さい場合は負荷削減効果よりも
  品質劣化が目立ちやすいので注意**

# パフォーマンスチューニング

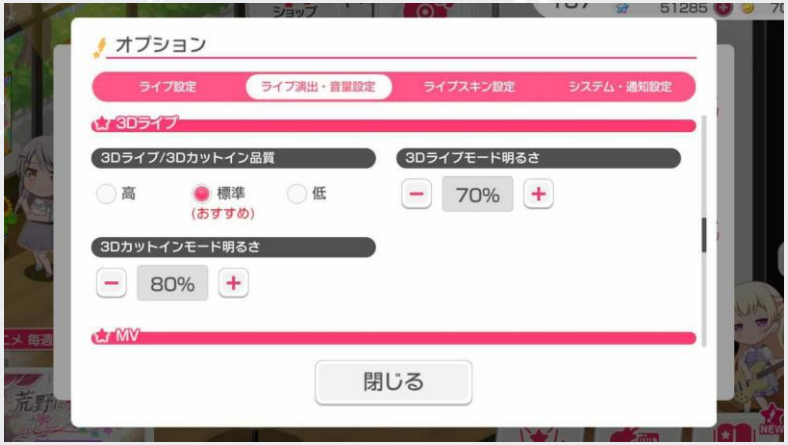
## 品質設定について

# 品質設定について

ゲームオプション「3Dライブ品質」項目

→　高・標準・低　の3種

→　実は再生モードによって
同じ品質でも微妙に設定が異なる




【再生モード】

- インゲームモード：リズムゲーム背景として3DMVを再生する
- 観賞モード：観賞用に3DMVを単純再生する

# 品質設定：モード別調整方針

【インゲームモード】

- 各品質の動作目標端末において、フレームレート維持が最優先
- 一定以上のカクツキは許容しない
- 品質によっては負荷の高い一部演出のカットも許容する

【観賞モード】

- 動作目標端末で多少フレームレートが落ちても許容、画面品質重視
- 各種演出効果は品質関係なくカット厳禁

# 品質設定：詳細項目

- 3Dシーンの描画解像度（描画スケール）
  - ○ 最も負荷＆品質に影響のある項目
  - ○ モード毎、品質毎にしっかり個別調整する必要がある
- 縮小バッファスケール値
  - ○ 微調整用
  - ○ 主に高品質における負荷＆品質調整のための項目
- 分割レンダリング設定
  - ○ 全モード、全品質で有効化
  - ○ 当初、高品質では3DMVも60fpsで再生できるようにする前提で準備していた
    最終的に30fpsで再生するのがベストということで、常時有効となっている

# 品質設定：詳細項目

## ● ポストプロセス設定

- ○ 主にインゲームモードの低品質における処理落ち対策
- ○ ポスプロ処理の中でも最も重いDOFを強制的に無効にできる

## ● サブカメラ設定

- ○ 上記同様
- ○ サブカメラのレンダリングを強制的に無効にできる
- ○ 無効になった場合、ステージ上LEDモニタにはサブカメラの描画テクスチャは映らない
  - ■ サブカメラを映す設定になっていた場合、プリレンダムービーが映る

# 3DMV以外での3Dアセット活用

# 3Dカットインとは

- リズムゲームを行う際、ノーツの後ろにUIとして映し出される演出の中で3Dモデルのキャラが演出を行うもの
  - ○ 全画面に映る二人で演出を行うカットイン
  - ○ 全画面に映る一人で演出を行うカットイン
  - ○ 画面の両端にUIが出て掛け合い演出を行うカットイン

- リズムゲームが行われる前画面で3Dカットインをonにすると表示される


2人演出カットイン


掛け合い演出のカットイン

# 2人演出カットインでのHandIK

- ハイタッチ、肩タッチ、グータッチなどで使用
- 2人演出は他のキャラが同演出を行うのでHandIKの設定は不可欠
- IKターゲット先
  - 相手の部位・・・5演出
  - Emptyオブジェクトを生成し空気中の特定箇所に手を近づける・・・5演出
- Emptyオブジェクトの位置はTimelineに情報として入っており、ランタイムに生成（カットインはシーンに情報を置けないため）

# IKオフ：手を近づける

# IKオン：手を近づける

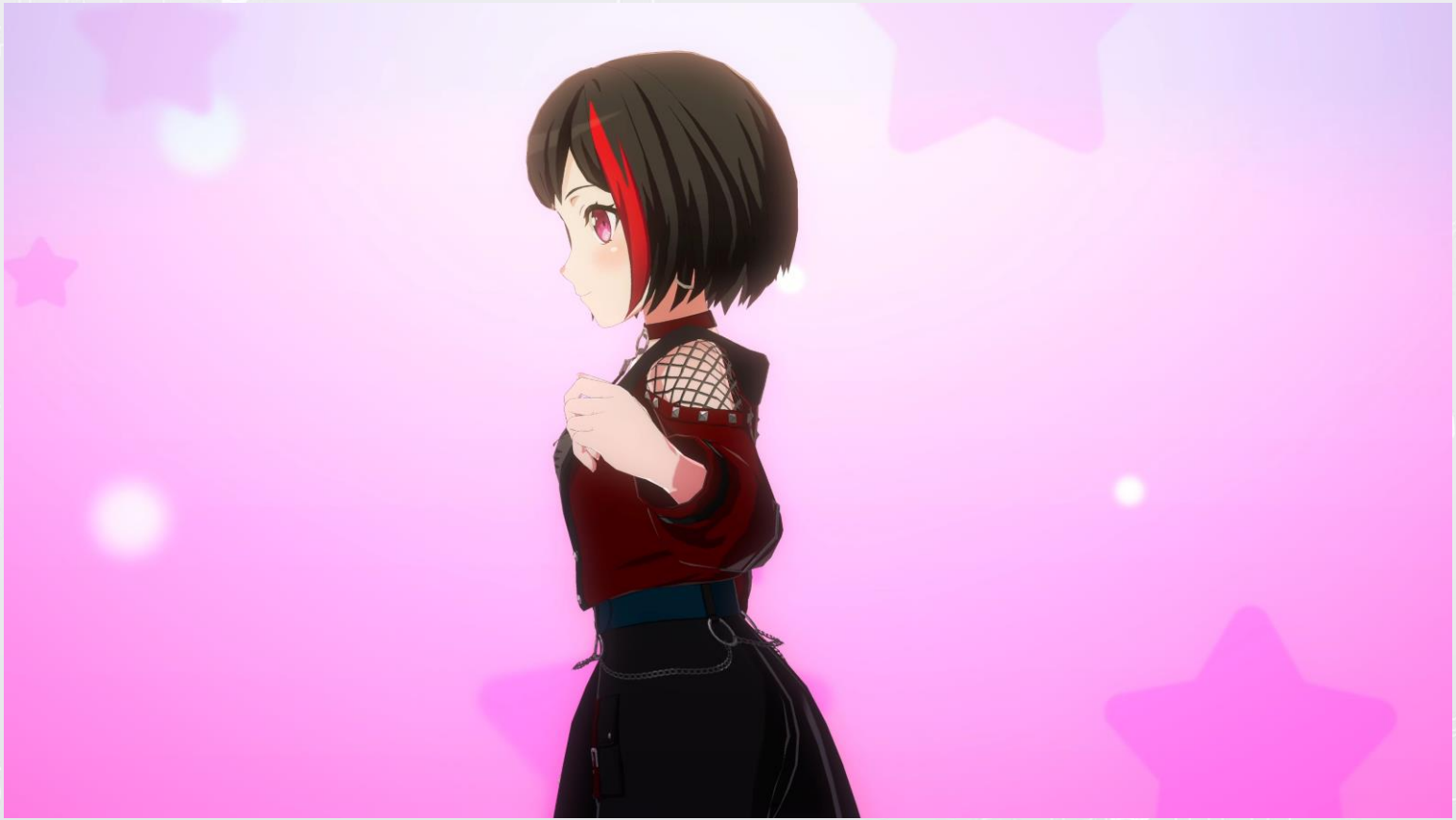
# IKオフ：ハイタッチ、グータッチ

# IKオン：ハイタッチ、グータッチ

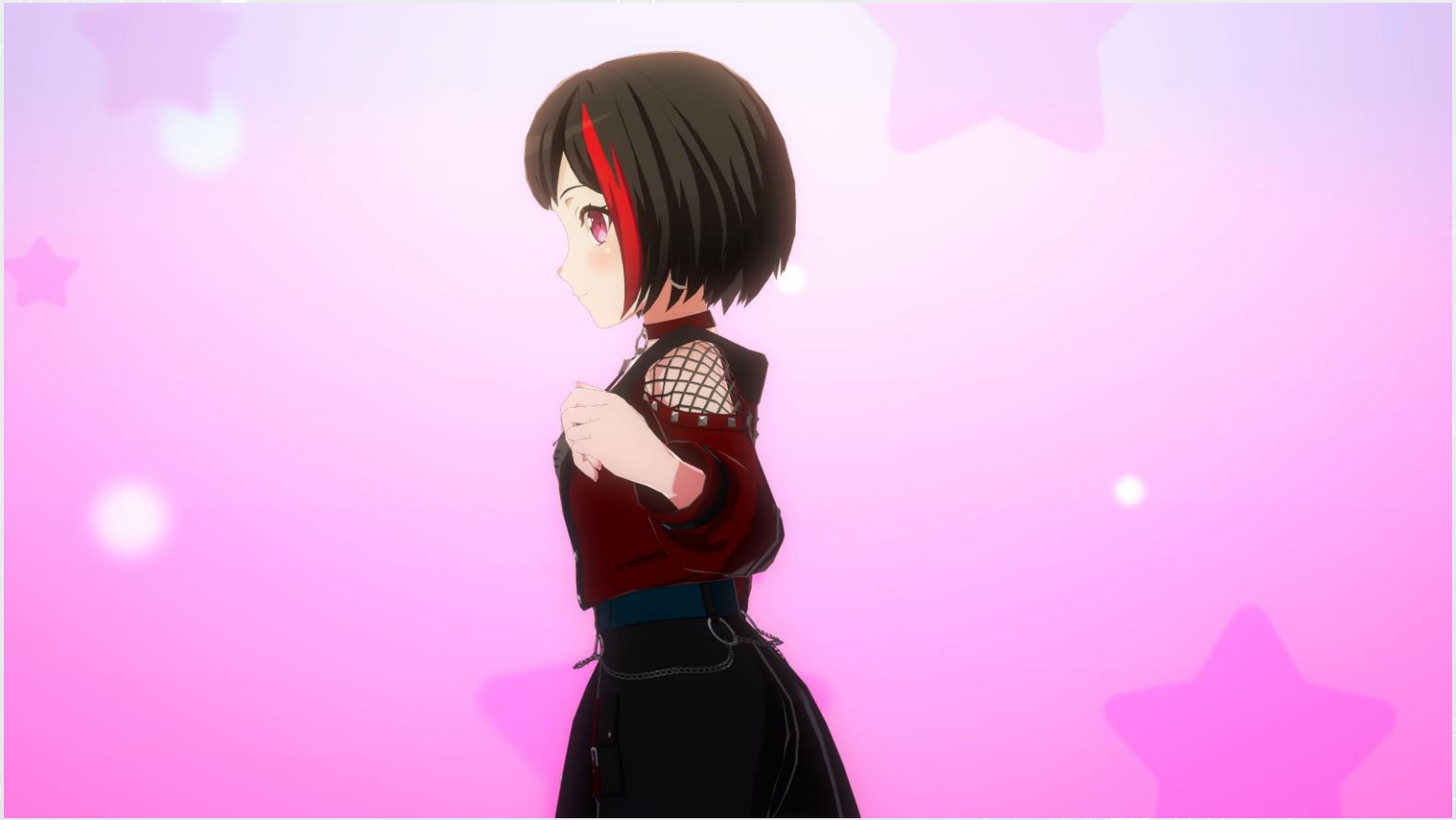
# 3DカットインでのFacial

- フェイシャルはMayaでアニメーション制作を行いアニメーションとして出力
  - 3DMVはUnityで表情をつけている
  - カットインではキャラごとの表情をMayaで調整したいという要望があった
  - 歯のアニメーションはカット
  - 目のハイライトはMVと同じく、スクリプト制御に任せた
  - 目玉のアニメーションはRotationのみだけアニメーションさせる
  - 目のIKは特に二人カットインで積極的に使用した

# 目のIK:オフ



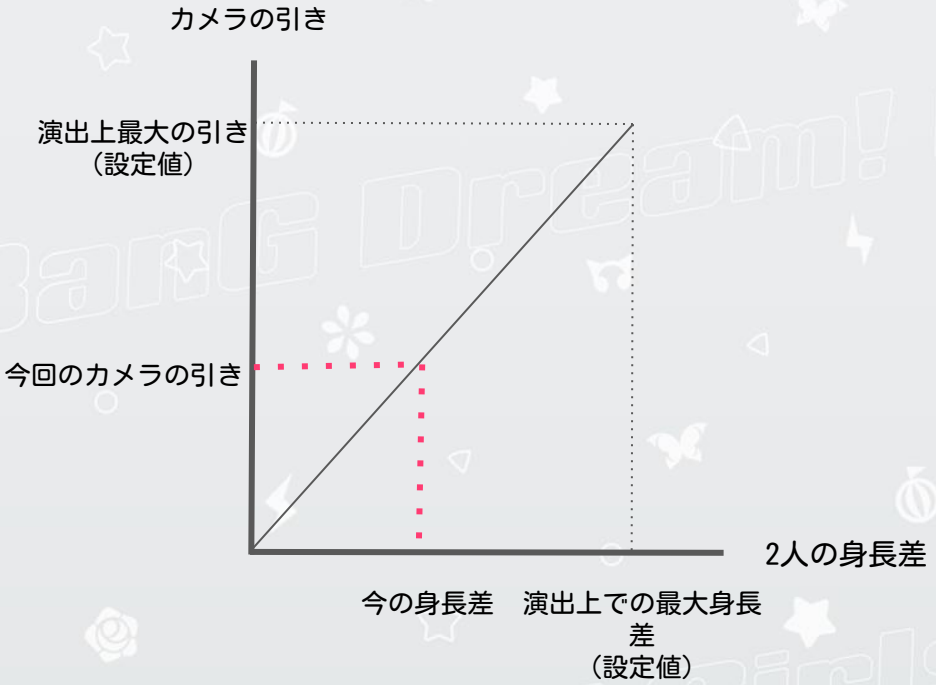
# 目のIK:オン



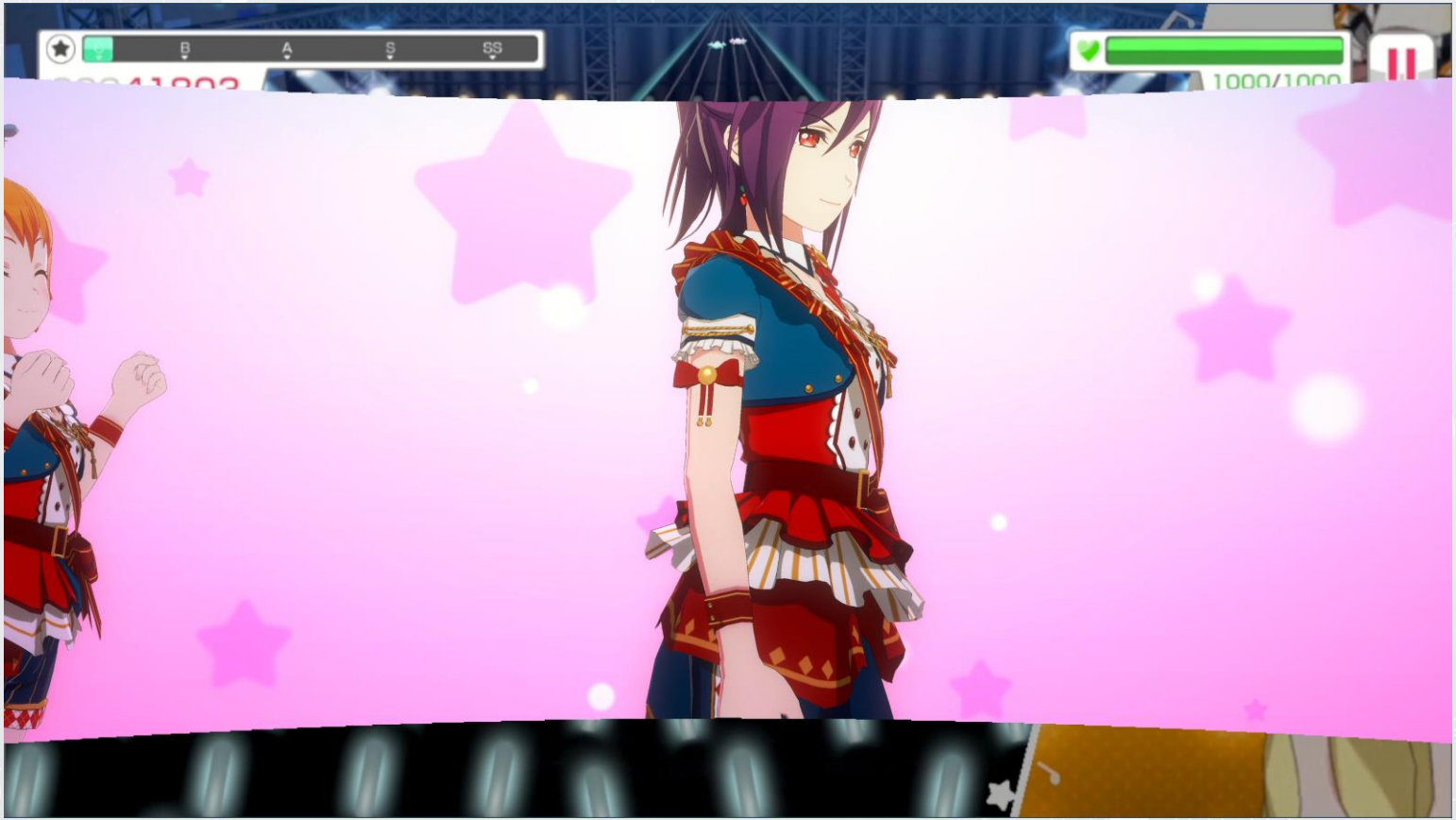
# 2人演出カットインでのカメラ補正

- 高さとカメラを引く補正が実装
  - 画角調整ではなくPosition自体を引くことに
    - 曲がって見える懸念
    - 後ろにオブジェクトがない
  - 高さの補正
    - カメラをどちらに合わせるかの設定
    - そのキャラの標準身長(155cm)との差でy軸補正
  - 引きの補正
    - 演出ごとに最大身長差における引き量を決める
    - 演出の際の二人の身長差で線形補間した値を実際の引きとする
    - ex）最大身長差が20で最大の引きが10。再生中の身長差が5の時

      カメラの引き = 5 * (10/20) = 2.5



# カメラ補正OFF



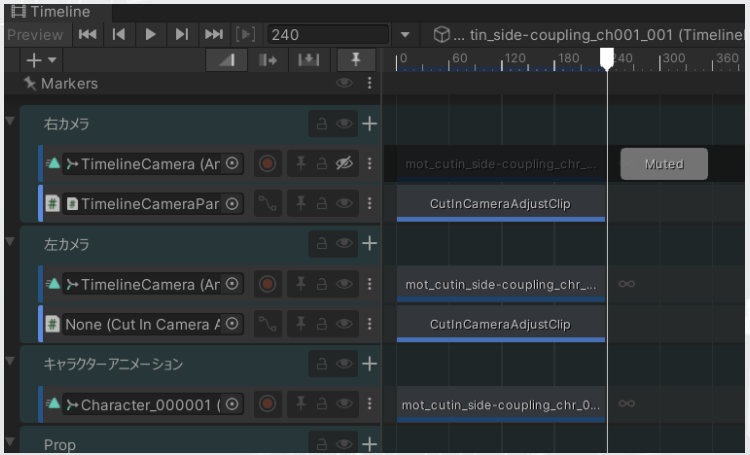
# カメラ補正ON

# 掛け合いカットインでの再生機構の工夫

## 掛け合いカットイン

- 掛け合いカットインのカメラワークは右カットインと左カットインで左右反転ではなく、個別のアニメーションを用意した
- Timelineとしては両方のカメラアニメーションを入れておき、ランタイムに片方ミュートすることで実現


左UIが左カットイン、右UIが右カットイン

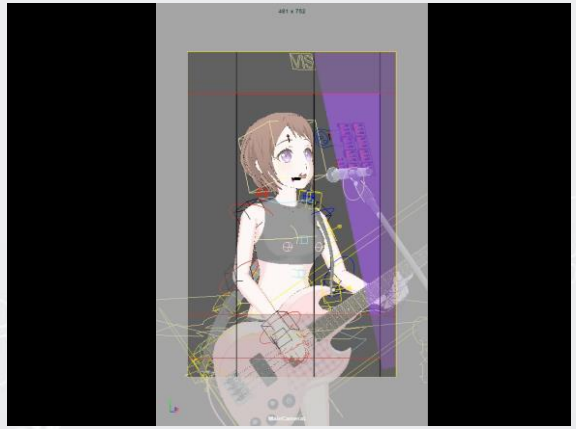


# 掛け合いカットインでの再生機構の工夫：右カットイン

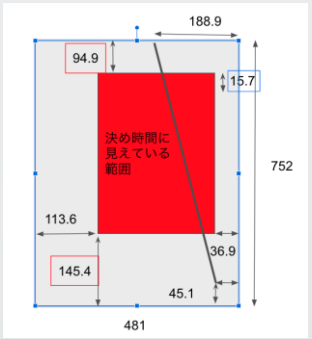
# 掛け合いカットインでの再生機構の工夫：左カットイン

# 掛け合いカットインの制作環境

- UIマスクのよる演出の見切れが目立った
- 実機確認後の修正だと工数を圧迫
- 演出UIがアニメーションするため、制作が難しい



- MayaでUIマスク部分がわかるように
- Mayaでマスクを被せて映像制作
- 掛け合いカットインは「決め時間」を中心に制作




マスク

# アセットレギュレーションの自動検知

- ゲーム内の3D関連アセットは膨大で人作業によるチェックは不可能
  - ○ Exporter、Importer、そしてレギュレーション自動チェックという3本柱でヒューマンエラーを検知する仕組みを構築

- レギュレーションチェックはOSSのAssetRegulationManagerを使用
  - ○ Jenkins上で毎朝4時にレギュレーションのチェックが行われる
  - ○ チェックに違反したアセットのリストがSlackで通知される
  - ○ リストを渡されただけでは修正まで及ばないこともあるため、機能ごとに担当のメンバーを決めてメンションつけることで改善を図った

# アセットレギュレーションの自動検知項目

- キャラ、楽器モデル
  - 身長や胸のサイズは仕様通りか
  - 揺れもの設定やスクリプトの参照が外れていないか
  - 骨の命名規則など
  - 所定のスクリプトがアタッチされているか
- Material、TextureやVideoなどの各種アセット
  - 解像度やサイズ
  - 各種設定が間違っていないか(ToggleのONなど)
  - 演出Timelineの尺や、使用するモーションがあっているかなど
- 全般アセットでアセットパスやAB名が正しいか
- 計63項目

# 質疑応答

ありがとうございました！！