

# E-mote Unity SDK マニュアル

2020/03/13 有限会社エムツー

## この文書の構成

この文書は、E-mote Unity SDKのマニュアルです。

E-mote Unity SDKIには歴史的経緯から2つのSDKが存在するため、まずその2つのSDK紹介した後、基礎的セットアップを、サンプルシーンを交えながら解説します。

その後、UnityのInspectorから設定可能なプロパティを、対応するC#のAPIと共に解説します。

さらに、様々な応用的セットアップを、対応するスクリプトと共に解説します。

TYPE1 SDK、もしくはTYPE2 SDKに固有の解説がある場合は、見出しの後ろに *[TYPE1]* *[TYPE2]* と言った表記を行うことで、対応SDKを限定します。

# 目次

[この文書の構成](#)

[目次](#)

[Unityへの導入](#)

[2つの SDK](#)

[E-mote Ver3.7以前のSDKをご使用のお客様へ](#)

[更新履歴](#)

[既知の不具合](#)

[E-mote3.9.7p3](#)

[E-mote3.9.7p2](#)

[E-mote3.9.7](#)

[E-mote3.9.6](#)

[E-mote3.9.5](#)

[E-mote3.9.4](#)

[E-mote3.9.3](#)

[E-mote3.92](#)

[E-mote3.91](#)

[E-mote3.9](#)

[フォルダ構成](#)

[キャラプロファイル](#)

[基礎的セットアップ](#)

[Basic セットアップ](#)

[サンプルシーン](#)

[構成](#)

[概要](#)

[Legacy セットアップ](#)

[サンプルシーン](#)

[構成](#)

[概要](#)

[主要なプロパティとAPI](#)

[E-mote Global Settings](#)

[プロパティ](#)

[API](#)

[EmoteDeviceManager](#)

[API](#)

[API 詳細](#)

[EmoteDevice.cs](#)

[Appearance](#)

[プロパティ](#)

[API](#)

[Raw Level API](#)

[Transform](#)

[プロパティ](#)

[API](#)

[Raw Level API](#)

[Render Texture](#)

[プロパティ](#)

[API](#)

[Stereovision \[TYPE2\]](#)

[プロパティ](#)

[API](#)

[Behavior](#)

[プロパティ](#)

[API](#)

[Timeline](#)

[プロパティ](#)

[API](#)

[Raw Level API](#)

[Variables](#)

[Raw Level API](#)

[Logging](#)

[Raw Level API](#)

[Component](#)

[応用的セットアップ](#)

[E-moteロゴの表示](#)

[サンプルシーン](#)

[構成](#)

[概要](#)

[プロパティ](#)

[Generate Test](#)

[サンプルシーン](#)

[構成](#)

[概要](#)

[API](#)

[Billboard セットアップ](#)

[サンプルシーン](#)

[構成](#)

[概要](#)

[Emote Fps Camera Control](#)

[Emote Billboard Control](#)

[LipSynch セットアップ](#)

[サンプルシーン](#)

[構成](#)

[概要](#)

[Emote Lip Synch Control](#)

[Multiple セットアップ \[TYPE2\]](#)

[サンプルシーン](#)

[構成](#)

[概要](#)

[Stereovision セットアップ\[TYPE2\]](#)

[サンプルシーン](#)

[構成](#)

[概要](#)

[VRへの導入\(Unity5.6以降\) \[TYPE2\]](#)

[構成](#)

[注意点](#)

[Steam VRへの導入\(Unity5.5以前\) \[TYPE2\]](#)

[構成](#)

[VR向けテクスチャカーブの設定](#)

[構成](#)

[概要](#)

[プロパティ](#)

[既知の不具合](#)

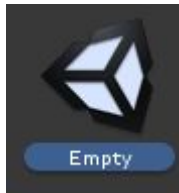
## Unityへの導入

Unity E-mote SDK は Unity5.4以降を正式なサポート対象としています。

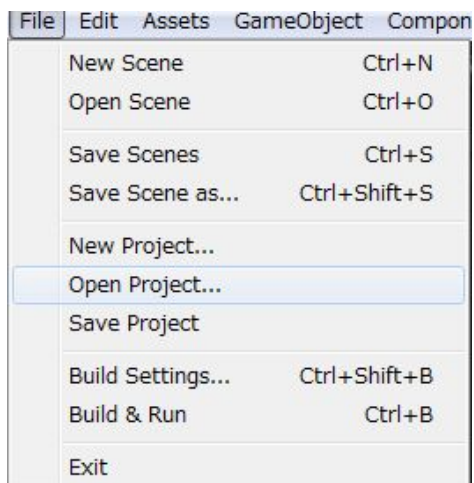
unitypackageをインポートすることで、E-moteの使用を開始出来ます。

新しいバージョンのSDKをインポートする場合は、読み込み済みのDLLが邪魔をするため、以下の手順を取って下さい。

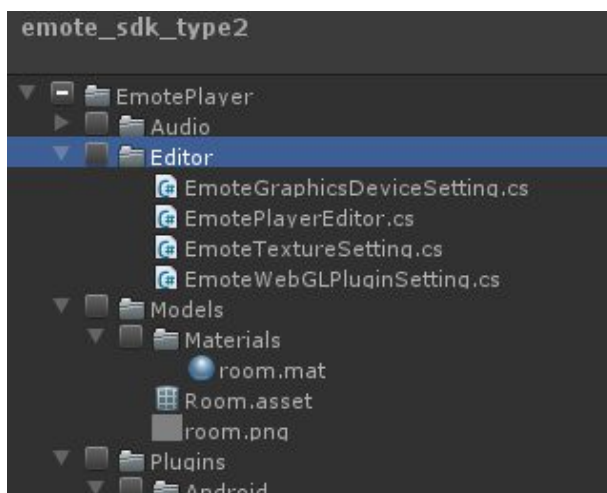
1. 空のプロジェクトを開く(無ければ適当に自分で作ってから開く)



2. プロジェクトを一旦閉じて開き直す



3. 新しいSDKのパッケージをインポートする



## 2つの SDK

E-mote SDKには歴史的経緯により、描画をNative Plugin内で行うTYPE1 SDKと描画をUnity側で行うTYPE2 SDKの2種類のSDKが存在します。

TYPE1 SDKには以下のような欠点があります。

- Unityのメジャーバージョンアップ時に描画に問題が発生しがち
- UnityのGraphics API(Metal/Vulkanなど)への対応が限られている
- VR非対応
- テクスチャをデータに内蔵するためプラットフォームごとの最適化が出来ない

今後、TYPE1 SDKはサポートを廃止する予定です。

現在 TYPE1 SDKをご使用のお客様は、お手数をおかけしますが別添えの「TYPE1からTYPE2への移行マニュアル」に従ってSDKの移行をお願いします。特にモバイル環境ではTYPE2 SDKへ移行することによって、大きなメリットが得られます。

- 最新のGraphics APIの指定が行えるようになり描画パフォーマンスを上げられる
- Multithread RenderingやGraphics Jobsのオプションにより描画パフォーマンスを上げられる

## E-mote Ver3.7以前のSDKをご使用のお客様へ

E-mote for Unity は、E-mote Ver3.8で大幅なアップデートを行いました。

これまで、構成上の制限から、E-mote プレイヤを「Unityのゲームオブジェクトとして扱う」際に様々な制約がありましたが、この度のアップデートにより、それらの制約を撤廃しました。

主に以下のような点が強化されています。

- 描画がQuadに対応したスプライト型の座標系で扱われる
- EmotePlayerをアタッチしたUnityのGameObjectを移動/回転させることでE-moteの物理演算が行われる。
- エディタで設定したキャラの身長に応じて表示が自動でUnityスケールに調整される。
- レンダリングテクスチャへの描画範囲をクリップして描画効率を最大化出来る。
- キャラの目・口・胸の各座標をUnity座標系で取得出来る。
- ボイスに対する口パクをスクリプトで自動処理出来る。

これらの新機能の恩恵を十分に受けるためには、E-mote3.8 エディタにてキャラデータに身長とバウンスの設定を行い、さらにはUnity側でのAPIの扱い方も変更する必要があります。

新機能は不要で、ドライバのみアップデートしたい場合は、[Legacy セットアップ](#)の項をご確認ください。

## 更新履歴

### 既知の不具合

- 現在、Unity TYPE2 SDKにおいて、Androidの一部の環境においてマスク処理に連動して描画の乱れが起きる現象の報告を受けております。Unityと当該端末の相性の問題と推測されますが、原因は現在調査中です。なお、問題はアルファマスクを適用している時に限定して発生するため、マスクタイプを「ステンシル」に変更することで、便宜的に問題を回避することが出来ます。現在、問題を把握している端末は、以下のとおりです。
  - HUAWEI P9 lite
  - HUAWEI BTV-W09
  - LUMIERE 503HW
  - HUAWEI P8 lite ALE-L02
  - Blade L3 g01
  - HUAWEI Y6 II
  - Ken Brown SKYLAB PLUS 10

### E-mote3.9.7p3

- TYPE2 SDKで、レンダーテクスチャへの描画時に乗算済みアルファテクスチャを生成する機能を追加しました。
- TYPE2 SDKで、半透明テクスチャ保護機能の対象パーツの周辺にアウトラインを表示して可視化する機能を追加しました。
- TYPE2 SDKで、レンダーテクスチャのクリアカラーを設定できるようにしました。
- TYPE2 SDKで、既存のモデルデータについて「適切なレンダリングテクスチャのクリアカラー」を計算する機能を追加しました。
- TYPE2 SDKで、圧縮テクスチャ適用時の微細なアーティファクトを抑えるため、シェーダーのアルファカットオフのしきい値を任意の値に設定する機能を追加しました
- TYPE2 SDKで、Legacy Alignmentでマスクのクリア範囲が適切に設定されないことがある問題を修正しました。

### E-mote3.9.7p2

- EmoteDeviceManager クラスにメインメモリのヒープサイズを「最大サイズ」と「使用サイズ」で取得するAPIを追加しました。
- TYPE1 SDKで、iOS/Androidの実行時に不要なテクスチャイメージキャッシュが行われていた不具合を修正し、メモリフットプリントを改善しました。TYPE2 SDKで、Unity2019 利用時に Map to Render Texture にチェックを入れた状態でEmotePlayerのアクティブ状態を切り替えるとエラーが発生する不具合を修正しました。
- TYPE2 SDKで、Editor動作時、マテリアルのプレビューにE-moteキャラが写り込んでしまう不具合を修正しました。

### E-mote3.9.7

- EmotePlayer.LoadData()関数でパスを指定してResourcesからE-moteモデルをロード出来なくなっていた問題を修正しました。
- E-mote Global Settings で SDKバージョンとビルド日時を確認出来るようになりました。
- TYPE2 SDKの分離テクスチャ型データで、AST( $\alpha$  分離テクスチャ)に対応しました。
- TYPE2 SDKでCamera Objectのプールにもnullチェックを入れ、Sceneチェンジ時に無効なカメラが出来上がる問題を修正しました。
- TYPE2 SDKでLegacyアライメントでレンダーテクスチャ表示した際の表示倍率が間違っていた問題を修正しました。

### E-mote3.9.6

- TYPE2 SDK で、E-mote Global Settings に「Global Translate Velocity Limit」の設定を追加しました。物理演算系に効かせるモデルの内部的移動速度の上限を設定出来ます。
- TYPE2 SDK で、RenderTargetへ描画する際に、Sorting LayerとSorting Order を指定出来るようになりました。
- TYPE2 SDK に実装漏れをしていた EmotePlayer.UnloadData() 関数を実装しました。
- TYPE2 SDK で、UnityのGCを避けるため、すべてのプラットフォームでE-mote SDK用の独自ヒープを使うように設定しました。
- TYPE2 SDK で、余計なGCを抑制するために各種のコード修正を行いました。
- TYPE2 SDK でUnity2017以降でBBM使用時にエラーが出ていた問題を修正しました。
- デバッグ目的のAPIロギング機能を実装しました。

### E-mote3.9.5

- Android SDK で ARM64bit ビルドに対応しました。
- TYPE2 SDK で、リソース管理処理を最適化し、不要なリソースがメモリ上に残らないようにしました。
- TYPE1 SDK の Androidビルドで、HAUWEI系の端末でステンシルマスクが正しく処理されない問題を修正しました。
- TYPE2 SDK で、入力のEmoteAssetを複数回使い回してLoadData()に渡すと異常動作する問題を修正しました。
- TYPE2 SDK で、内蔵テクスチャイメージのカウント用のピクセルデータ判定式が間違っていた問題を修正しました。
- TYPE2 SDK で、内蔵テクスチャイメージ展開時に、テクスチャの wrapMode を指定し忘れていた問題を修正しました。
- TYPE2 SDK で、Legacy Transform 設定時にレンダーテクスチャマッピングが設定出来ない問題を修正しました。
- TYPE1/TYPE2 SDK で、プロファイルを未設定のキャラに対して fixedScale を設定した時の挙動が不正になっていた問題を修正しました。

### E-mote3.9.4

- TYPE2 SDKがノベルエンジン『宴』に対応しました。
- TYPE2 SDKにTYPE1 SDKとの互換性を持たせました
  - アルファマスクに対応しました
  - TYPE1向けの「テクスチャ内蔵型データ」を読み込めるようにしました。
- TYPE2 SDKで立体視対応機能を強化しました。
  - シングルカメラでの立体視描画に対応しました(Unity5.6以降)。
  - シングルパスでの立体視描画に対応しました(Unity5.6以降)。
  - VRプレビュー向けのスクリプトとシーンを追加しました。
- TYPE2 SDKでレンダーテクスチャの最大サイズを2048から4096に拡張しました。
- TYPE2 SDKでResourcesフォルダ以外に置いたE-moteデータもInspector上でロードするデータとして指定可能にしました。
- TYPE2 SDKで、Emote Lip Synch Control コンポーネントに Auto Lip Synch プロパティを追加し、乱数に基づく自動口パクが出来るようにしました。
- TYPE2 SDKで、E-moteのタイムラインに対応したAnimator Controllerを自動生成するためのエディタスクリプトを追加しました。
- TYPE2 SDKで、E-moteのプロジェクト単位の設定を、Preferences から Project Settings へ移動させました。
- TYPE2 SDKでEmotePlayerのTransform Alignmentの初期値をLEGACYからORIGINへ変更しました。
- TYPE2 SDKで、GenerateTestのプレイヤーをスクリプトからではなくPrefabから生成するように変更しました。
- Unity TYPE2 SDKで、立体視時にカメラを傾けた際に視差の矛盾が発生する問題を修正しました。
- TYPE2 SDKで、GameObjectのTransformの z scale が0になっていると、レンダテクスチャマッピング時に正しくモデルが描画されない問題を修正しました。
- TYPE2 SDKでデータファイルに空指定をするとエラーが出る問題を修正しました。
- TYPE2 SDKでEmotePlayerにInspectorからEmoteデータでないbytesを読み込ませようとするとハングする問題を修正しました。
- TYPE2 SDKでEmotePlayerがPrefab化できない問題を修正しました。
- 差分タイムラインスロット2~6の内容がシリアライズされない問題を修正しました。
- TYPE2 SDKでマスク領域の計算を間違っていた問題を修正しました。
- TYPE2 SDKでEmoteTextureCurve.csを適用した際に、「不正なQuad」が割り当てられるタイミングがある問題を修正しました。
- TYPE2 SDKで同一のBBMが続いた際にダイナミックバッチングがオンになってメッシュが統合されてしまう問題を修正しました。

### E-mote3.9.3

- TYPE2 SDKでバッファブレンドに対応した。
  - 以下のブレンドモードに対応しました。
  - 「PS比較(暗)」「PS乗算」「PS焼き込みカラー」「PS焼き込みリニア」「PS比較(明)」「PSスクリーン」「PS覆い焼きカラー」「PS覆い焼きリニア」「PSオーバーレイ」「PSソフトラ

イト」「PSハードライト」「PS差の絶対値」「PS除外」「グレースケール」「モザイク」「ブラー」

- 対応はUnity5.6以降となります(Unity5.5以前では、上記のブレンドモードを指定しても何も描画されません)。
- レンダーターゲットへ描画している時のみ描画が有効になります。
- Unity5.6ではUnityの実装の都合でGPU負荷が高めになります。最良のパフォーマンスを得るにはUnity2017以降での対応を推奨します。

### E-mote3.92

- TYPE1 SDKで、Unity2017.2以降の環境でWebGLの描画がおかしくなってしまうバグを修正した。
- TYPE1 SDKで、DX9 で半透明テクスチャカラー保護をオンにした時に絵が化けるバグを修正した。
- TYPE1 SDKで、DX9 でバッファブレンドが正しく描画されないバグを修正した。
- TYPE1 SDKで、DX11 でバッファブレンドを使用するとハングするバグを修正した。

### E-mote3.91

- TYPE1 SDKで、Unity2017.2以降、デバイスの初期化に失敗する問題を修正した。
- TYPE1 SDKで、Unity2017.2以降、WebGLでの描画に失敗する問題を修正した。
- TYPE2 SDKで、立体視セットアップ時に face\_tears 変数の値を変更するとハングする問題を修正した。
- TYPE2 SDKで、iOSプラグインにbitcodeが含まれていないためReleaseビルドが出来ない問題を修正した。

### E-mote3.9

- パッケージの提供形態を unitypackage 形式に変更しました。
- WebGLプラットフォームに正式対応しました。
- Androidプラットフォームでx86ビルドに対応しました。
- EmotePlayerのデフォルト更新タイミングをUpdateからLastUpdateに変更した。
  - EmotePlayer.cs の更新タイミングを、Update() から LastUpdate() に変更しました。一般的には「EmotePlayerを操作するスクリプトのUpdate()」「EmotePlayerのUpdate()」と言う順序で実行される方が適切なケースが多いためです。
  - 互換性のため、以前バージョンと同様にUpdate()のタイミングで更新したい場合には、EmotePlayer.updateTiming プロパティを設定して下さい。
- アセットバンドルからのリソースの読み込みが出来るようにした。
  - EmotePlayer.LoadData() の第二引数にAssetBundleを指定することで、バンドルからリソースを読み込めるようにしました。
- 非同期にリソースの読み込みが出来るようにした。
  - EmotePlayer.LoadAssetAsync() 関数で、リソースを非同期読み込みする EmoteAssetRequest オブジェクトを取得出来るようになりました。
  - UnityEngine.AssetBundleRequest のように取り扱うことが出来ます。
- TYPE1 SDKでバッファブレンドに対応した。

- TYPE1 SDKで新規に、以下のブレンドモードの描画に対応した。
- 「PS比較(暗)」「PS乗算」「PS焼き込みカラー」「PS焼き込みリニア」「PS比較(明)」「PSスクリーン」「PS覆い焼きカラー」「PS覆い焼きリニア」「PSオーバーレイ」「PSソフトライト」「PSハードライト」「PS差の絶対値」「PS除外」「グレースケール」「モザイク」「ブラー」
- 実装の都合によりTYPE2 SDKではこれらのモードには対応しません。
- E-moteロゴを表示させるスクリプトを追加した。
  - E-moteロゴを表示する EmoteLogo.cs スクリプトを追加しました。
  - EmoteLogo.unity シーンにサンプルがあります。
  - えもふりのUnity SDKでは、最初に E-moteロゴを表示しないとE-moteキャラの表示がグレーアウトしたままになると言う制約があります。
- レンダーテクスチャ適用時に外部からメッシュとマテリアルを設定する口を追加した。
  - EmotePlayer に、レンダーテクスチャをマップするメッシュとマテリアルを外部から指定するプロパティを追加しました。
- VR向けにメッシュを湾曲させるスクリプトを追加した。
  - VR向けに、メッシュを手前に向かって丸く湾曲させるスクリプト、EmoteTextureCurve.cs を追加しました。
  - E-mote3.9βでのリリース時に、一時的にEmotePlayer.cs本体にこの機能を搭載している時期がありましたが、正式リリースではスクリプトを分離しました。
- 差分タイムラインの数を4つから6つへ増やした。
- EmotePlayer.applyTimeScale プロパティを追加した。
  - チェックを外すと、Time.deltaTimeではなくTime.unscaledDeltaTimeを参照してアニメーションを更新するようになり、Time.timeScale の影響を受けなくなる。
- Legacy Alignment時に、インスペクタからテクスチャサイズとメインカラーの設定が出来なくなっていたバグを修正
- StepUpdate時のmainColorの値が即時反映されない問題を修正。
- EmotePlayerがNon Activeな状態でLoadDataしてからActive化するとエラーが出る問題を修正。
- AndroidでHuawey端末での表示がうまくいかないバグを修正。
- TYPE2 SDKでAndroidビルド時、マルチスレッドレンダリングを選択するとハングする問題を修正した。

## フォルダ構成

E-mote for Unityは以下のようなフォルダ構成をしています。

- Assets/
  - EmotePlayer/
    - Scripts/ (必須) E-moteを制御するスクリプト。
    - Editor/ (必須) 各種エディタスクリプト。
    - Plugins/ (必須) Native Plugin。
    - Scenes/ サンプルシーン。
    - Resources/ E-moteのサンプルデータ。
    - Models/ サンプルシーンで使うモデルデータ。

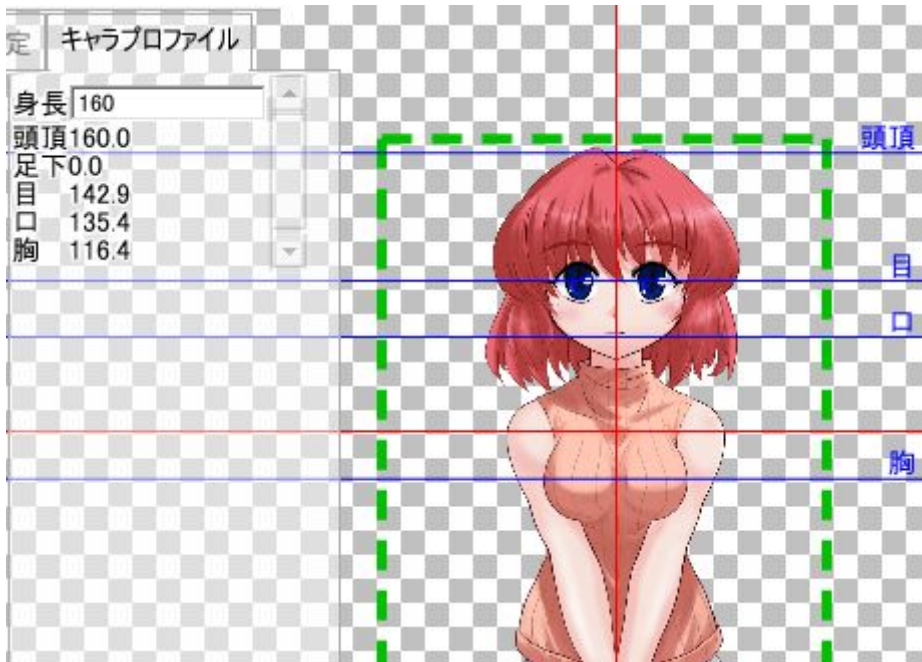
- Textures/ サンプルシーンで使うテクスチャデータ。
- Audio/ サンプルシーンで使うオーディオデータ。

## キャラプロフィール

E-moteエディタでは、キャラの表示範囲を定義する「バウンズ」と、キャラの「身長」、さらにキャラの目や口の位置を示す「マーカー」を設定することができます。

これらのメタ情報は、エクスポート時にE-moteデータに埋め込まれます。

E-mote SDKはこれらのメタ情報を元に、適切なスケーリング処理を行い、Unityの座標系にぴったり合わせたサイズでE-moteキャラを表示します(1単位を1メートルと換算)。



(緑色の枠がバウンズで、青い線がマーカーです)

「バウンズ」とは、キャラの表示範囲を定義する矩形の領域です。腕を切り替えられるモデルの場合は、一番範囲の大きい腕を基準に設定して下さい。

SDKでは、レンダーターゲットを対象にして描画する場合に、レンダーターゲットの枠内にE-moteキャラを収める基準として「バウンズ」を使用します。実際の判定ではキャラの姿勢変更を考慮に入れたマージンを加えて基準枠を設定するので、エディタ上ではピクセルぎりぎりで設定していても問題ありません。

「身長」の値と「頭頂」「足下」の2つのマーカーの設定位置から、Unity上でのE-moteキャラの表示スケールおよびポジションが決まります。そのため、この2つのマーカーの値は特に正確に設定する必要があります。

マーカーはバウンズをはみ出ても問題ないため、腰から下が存在しないモデルでも、想定される「足下」位置にマーカーを設定することで、表示位置を正確に取り扱うことができます。

「目」「口」「胸」のマーカーは完全にオプションな値で、SDKが直接これらの値を利用することはありません。アプリケーション側でUnity座標系で値を取得し様々な制御を行うために利用することができます。

## 基礎的セットアップ

以下、E-moteの基礎的なセットアップを解説します。

さらに、E-mote3.7以前までのLegacyセットアップを補足的に解説します(新規セットアップのみを使う場合こちらは特に参照する必要はありません)。

### Basic セットアップ

#### サンプルシーン

Assets/EmotePlayer/Scenes/BasicSetup.unity

#### 構成

1. Cameraを作成します (TRS=(0,0,-1.5):(0,0,0):(1,1,1))
2. 空の GameObject (TRS=(0,0,0):(0,0,0):(1,1,1))を作成します。
3. Emote Player コンポーネントをアタッチします。
4. Emote Player のプロパティで、Appearance -> Data File の値を“vr\_girl”に設定します。

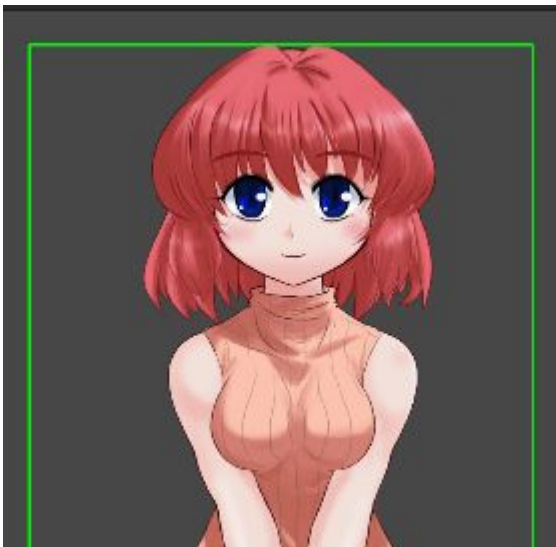
#### 概要

UnityでのE-moteの一番基本的なセットアップです。

表示のために、Emote Player 以外のコンポーネントは不要です。

E-moteキャラが、身長のちょうどセンターをGameObjectの原点とする形で、Unity座標系に表示されます。

表示方向は Quad と同等の「Zのマイナス側を向いたXY平面」となります。



## Legacy セットアップ

### サンプルシーン

Assets/EmotePlayer/Scenes/LegacySetup.unity

### 構成

E-mote3.8ではUnityプロジェクトの構造が大きく変わっています。

E-mote3.7以前に構築した環境に最新のSDKを取り込むには以下の手順でファイルを上書きして下さい。

1. Assetes/EmotePlayer/Code/ を Assets/EmotePlayer/Scripts/ にリネームする
2. Assets/Plugins/ を Assets/EmotePlayer/Plugins/ にリネームする
3. 新しい SDK のパッケージをインポートする

### 概要

EmotePlayerのTransform -> Alignment の設定を Legacy にしておくことで、回転方向やスケールなど、E-mote3.7以前と互換のあるジオメトリでの表示が行われます。

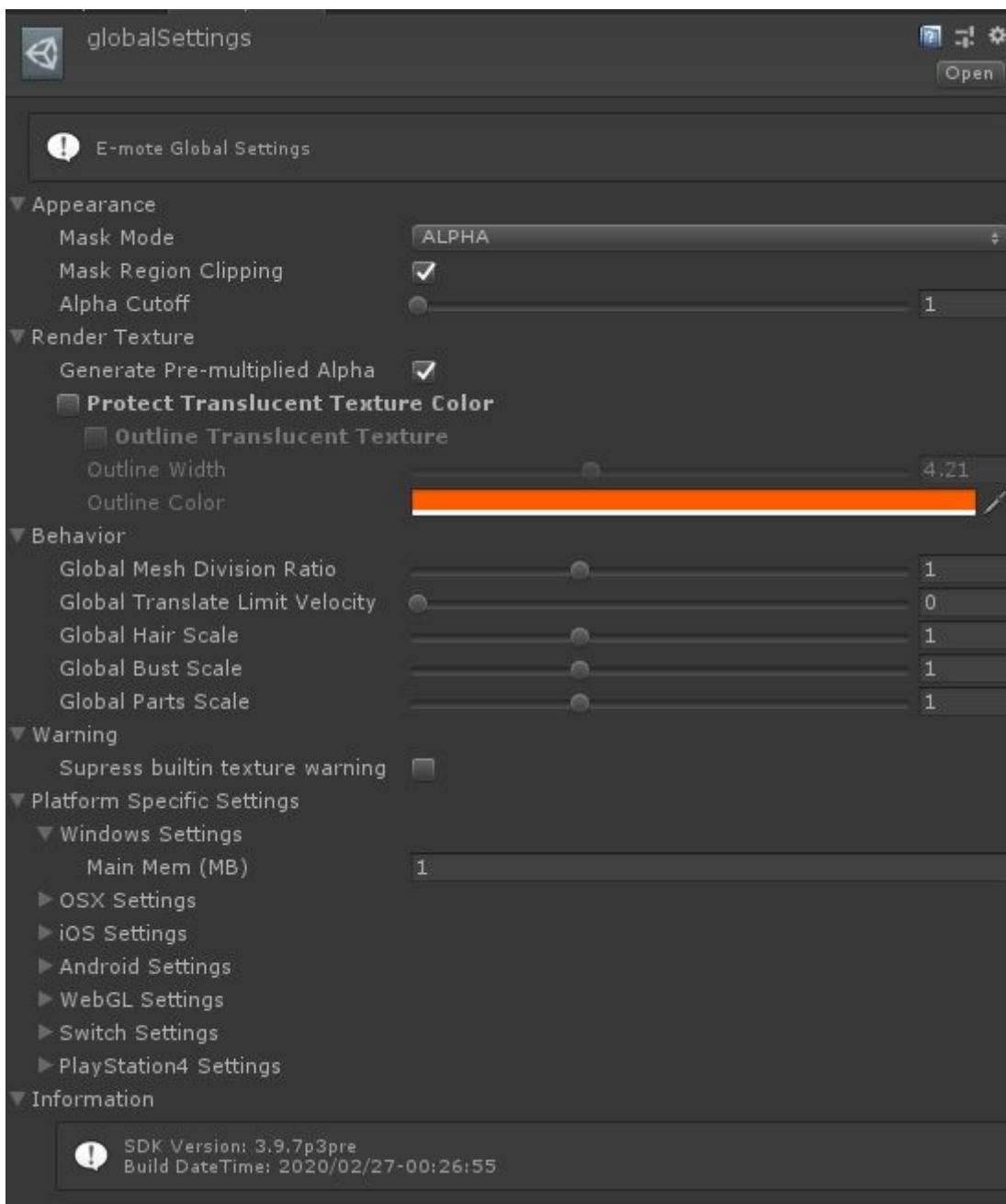
## 主要なプロパティとAPI

以下のページでは、Unity Editor で操作できる 各種 E-mote のプロパティと、それに対応したスクリプトのAPIをカテゴリごとに解説します。

また、カテゴリに関連する、詳細な制御が可能なよりローレベルなAPIが用意されている場合は、それらの解説も合わせて行います。

### E-mote Global Settings

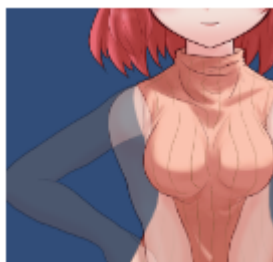
#### プロパティ



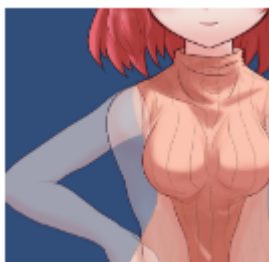
E-moteのグローバル設定を行います。

Edit/Project Settings/E-mote Global Settings を選択するか、EmotePlayerのコンテキストメニューから E-mote Global Settings を選択することで Inspector に設定画面が開きます。

- Mask Mode
  - マスクの動作モードを”STENCIL”と”ALPHA”のどちらか選択します。”STENCIL”が軽量、”ALPHA”が高品質なモードです。VRAM使用量は”ALPHA”の方が顕著に小さくなります。
- Mask Region Clipping
  - チェックを入れると、CPU時間を余分に使って計算を行いマスク領域を最小限に抑さえ、マスク処理時のGPU負荷を抑えます。特にGPUヘビーな環境での最適化に有効です。
- Alpha Cutoff [TYPE2]
  - シェーダーでテクスチャアルファが一定値未満の場合にピクセル処理をスキップするアルファカットオフの値を設定します。
  - アルファが [設定値/256] 未満の場合、処理がスキップされます。
- Generate Pre-multiplied Alpha [TYPE2]
  - チェックを入れると、レンダリングテクスチャへの描画時に、カラーにアルファの値を乗算済みの「乗算済みアルファ」テクスチャを生成するようになり、半透明テクスチャのカラーを正確に描画できるようになります。
  - 後述の「Protect Translucent Texture Color」機能とは排他的に働くので、どちらかだけ指定するようにしてください。
  - 乗算済みアルファでは、半透明テクスチャの合成時にテクスチャの色が「完全に正しく」保存されます。
  - Legacy Alignmentでシェーダーを明示的に設定しなければいけない場合は、乗算済みアルファテクスチャを適切に描画する「Emote/PMARenderer」を指定してください。
  - Materialを指定して自前のシェーダーで描画している場合、ノーマルアルファのテクスチャでは「 $\text{srcColor} * \text{srcAlpha} + \text{dstColor} * (1 - \text{srcAlpha})$ 」で計算しているところを「 $\text{srcColor} + \text{dstColor} * (1 - \text{srcAlpha})$ 」で計算するように変更したシェーダーを用意してください。
- Protect Translucent Texture Color
  - チェックを入れると、レンダリングテクスチャに対して半透明なテクスチャを描画する時に、半透明テクスチャの二度描きを行い、カラーが透明色の黒と混ざってしまう現象を抑制します。
  - 前述の「Generate Pre-multiplied Alpha」とは排他的に働くので、どちらかだけ指定するようにしてください。



OFFの場合



ONの場合

- 以下に「Generate Pre-multiplied Alpha」と「Protect Translucent Texture Color」の特性の比較を表で示します。レンダーテクスチャ描画時の半透明テクスチャのカ

ラーの乱れをおさえる目的は同じですが、総合的に見ると前者の方がメリットが大きいです。

|               | Generate Pre-multiplied Alpha                                                                          | Protect Translucent Texture Color                              |
|---------------|--------------------------------------------------------------------------------------------------------|----------------------------------------------------------------|
| 半透明カラーの描画の正確性 | 完全                                                                                                     | 一部不完全<br>半透明パーツを重ね描きした際にアーティファクトが出やすい。                         |
| 圧縮テクスチャとの相性   | 良い                                                                                                     | 悪い<br>圧縮テクスチャに特有のアルファのブレと相性が良くない。                              |
| ドローコール数       | 低い<br>通常描画とほぼ変わらない                                                                                     | 高い<br>二度描きを行う特性上ドローコールが増えがち                                    |
| ピクセルあたりの描画負荷  | 若干重い<br>ノーマルアルファと乗算アルファの相互変換が描画中に入るため若干重い。                                                             | 通常通り<br>通常描画とピクセルあたりの負荷は変わらない。が、そもそも二度描きする関係で描画するピクセル数自体が増える。  |
| 特定のブレンドモードの負荷 | 高くなる<br>「減算」「加算」「PS乗算」「PSスクリーン」の4つのブレンドモードが、一旦レンダーテクスチャの内容をキャプチャしてから描画する「バッファードブレンド」に変更され、オーバーヘッドが増える。 | 変わらない<br>「減算」「加算」「PS乗算」「PSスクリーン」の4つのブレンドモードは、通常通りのシェーダーで処理される。 |

- Outline Translucent Texture
  - チェックを入れると、半透明テクスチャ保護の対象となっているパーツの周辺にアウトラインを表示し、対象パーツを可視化します。Outline Widthで線の幅を、Outline Colorで線の色を設定できます。
- Global Mesh Division Ratio
  - 全てのE-moteキャラに影響を及ぼす、メッシュ分割率のグローバル設定です。小さくするとCPU演算が軽くなりますが、メッシュ演算の精度が下がりキャラの形が徐々に崩れます。実用上の下限は0.3あたりです。
- Global Translate Limit Velocity [TYPE2]

- E-moteキャラを動かした場合の、「物理演算に使うキャラの移動速度の上限」を設定出来ます。単位は「m/s」です。0より大きい値を設定すると、どれだけ激しくキャラを動かしても、設定速度を上限として物理演算が行われるようになります。
- 
- Global Hair Scale
  - 全てのE-moteキャラに影響を及ぼす、髪揺れ係数のグローバル設定です。小さくすると髪が揺れなくなります。
- Global Bust Scale
  - 全てのE-moteキャラに影響を及ぼす、胸揺れ係数のグローバル設定です。小さくすると胸が揺れなくなります。
- Global Parts Scale
  - 全てのE-moteキャラに影響を及ぼす、パーツの揺れ係数のグローバル設定です。小さくするとパーツが揺れなくなります。
- Suppress builtin texture warning [TYPE2]
  - TYPE2 SDKにTYPE1用のテクスチャ内蔵型データを読み込ませるとコンソールに警告メッセージが表示されます。この項目にチェックを入れるとその警告メッセージが表示されなくなります。
- Windows/OSX/iOS/Android/WebGL/Switch/PlayStation4 Settings
  - 各プラットフォームの固有設定です。ヒープの断片化を防ぐためにあらかじめ確保するメモリブロックをMB単位で設定します。

## [API](#)

Preferencesに設定した値は、Assets/Resources/emote/EmoteGlobalSettings.asset に保存され、ビルドしたアプリに自動的に反映されます。また、アプリ起動後にAPIを通じてこれらの値を動的に変更することも出来ます。

```
class EmotePlayer {
    static public int maskMode;
    static public bool maskRegionClipping;
    static public int alphaCutoff;
    static public bool generatePremultipliedAlpha;
    static public bool protectTranslucentTextureColor;
    static public bool outlineTranslucentTexture;
    static public float outlineWidth;
    static public Color outlineColor;
    static public bool suppressBuiltinTextureImageWarning;
    static public float globalMeshDivisionRatio;
    static public float globalTranslateLimitVelocity;
    static public float globalHairScale;
    static public float globalBustScale;
    static public float globalPartsScale;
};
```



## EmoteDeviceManager

UnityのE-mote SDKでは、複数のEmotePlayerが共有するリソースをEmoteDeviceとして、一括管理しています。E-mote Global Settings で設定できるSDK固有のヒープも実体としてはEmoteDeviceに所属しています。

EmotePlayerの利用者が普段EmoteDeviceの存在を意識することはありません。EmoteDeviceはリファレンスカウント方式で管理され、EmotePlayerがひとつでも生成された時点でEmoteDeviceが生成され、EmotePlayerがひとつも存在しなくなった時点でEmoteDeviceが破棄されるというように、必要に応じて勝手に生成・破棄されるようになっています。

ですが、ADVの会話シーンのように頻繁にEmotePlayerの生成と破棄が繰り返されることが想定される場面では、頻繁なEmoteDeviceの生成・破棄がパフォーマンス上の問題につながるケースがあります。

こうした問題に対処するため、明示的に EmoteDevice をロード・アンロードできる EmoteDeviceManager クラスが存在します。

## API

```
class EmoteDeviceManager
{
    public EmoteDeviceManager();
    ~EmoteDeviceManager();

    public void Load();
    public void Unload();
    public bool loaded;
    public int activePlayerCount;
    public List<EmotePlayer> getActivePlayerList();
    public int deviceRefCount;
    public bool deviceResident;
    public int getMainMemTotalSize();
    public int getMainMemAllocatedSize();

};
```

## API 詳細

- Load()
  - 「未ロード状態」ならEmoteDeviceのリファレンスカウントを1つ増やして「ロード状態」にします。
- Unload()
  - 「ロード状態」ならEmoteDeviceのリファレンスカウントを1つ減らして「未ロード状態」にします。
- bool loaded

- 「ロード状態」かどうかを判定します。
- int deviceRefCount
  - EmoteDeviceのリファレンスカウントを取得します。
- bool deviceResident
  - EmoteDeviceが常駐状態かを判定します。(deviceRefCount > 0)と同義です。
- int activePlayerCount
  - 現在シーン上に存在するEmotePlayerの総数を返します。
- List<EmotePlayer> getActivePlayerList()
  - 現在シーン上に存在するEmotePlayerのリストを返します。
- int getMainMemTotalSize()
  - SDK固有ヒープの総サイズをバイト単位で取得します。
- int getMainMemAllocatedSize()
  - SDK固有ヒープの使用サイズをバイト単位で取得します。

### [EmoteDevice.cs](#)

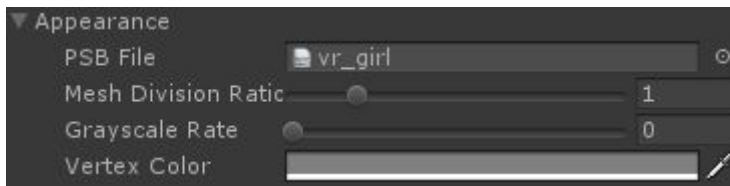
さらに、「シーン内でEmoteDeviceを常に常駐状態にする」ために、EmoteDevice.csという簡単な補助スクリプトが用意してあります。

このスクリプトは、「開始時に EmoteDeviceManager を Load()し、終了時に Unload() する」というシンプルな動作を実行します。シーン内の適当なオブジェクトにアタッチしておけば、そのシーン内では EmoteDevice がアンロードされることなく常駐し続けることが保証されます。

## Appearance

E-moteの「見た目」に関するプロパティです。

### プロパティ



- Data File
  - E-mote エディタでエクスポートしたE-moteデータを設定します。
- Mesh Division Ratio
  - キャラごとのメッシュ分割率の設定です。小さくするとCPU演算が軽くなりますが、メッシュ演算の精度が下がりキャラの形が徐々に崩れます。実用上の下限は0.3あたりです。
- Grayscale Rate
  - グレースケールの度合いです。値を大きくすると徐々に彩度が下がり、1にすると完全に灰色になります。
- Vertex Color
  - E-moteの個々のレイヤに対して適用される頂点カラーです。
  - 中性色はRGBA=(0.5f,0.5f,0.5f,1f)、RGBは0.5より小さい値を設定すると暗くなり、大きい値を設定すると明るくなります。
  - E-moteは複数のレイヤを重ねて映像を生成するので、アルファ値を設定すると以下のように透けてしまいます。



### API

```
class EmotePlayer {  
    public string DataFilePath;  
    public float meshDivisionRatio;  
    public float grayscale;  
    public Color vertexColor;  
};
```

## Raw Level API

```
class EmotePlayer {
    // E-moteデータをロードします。
    // dataにはbytesデータ、テキストアセット、パス
    // 分割エクスポートデータは、それらの配列を指定してロードします。
    // また、EmoteAssetオブジェクトを指定することも出来ます。
    // bundleを指定すると、アセットバンドルからデータをロードします。
    public void LoadData(object data, AssetBundle bundle = null);
    // E-moteデータをアンロードします
    public void UnloadData();

    // リソースの同期読み込みを行います。
    // pathObjectにはリソースのパスを文字列か、文字列の配列で指定します。
    // bundleを指定すると、アセットバンドルからデータをロードします。
    static public EmoteAsset LoadAsset(object pathObject, AssetBundle bundle = null);
};

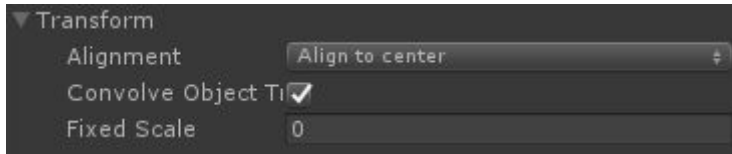
// リソースの非同期読み込みを開始します。
// pathObjectにはリソースのパスを文字列か、文字列の配列で指定します。
// bundleを指定すると、アセットバンドルからデータをロードします。
static public EmoteAssetRequest LoadAssetAsync(object pathObject, AssetBundle
bundle = null);
};

// 非同期読み込みされた EmoteAsset クラスをリクエストします。
// コルーチンとして使用可能です。
class EmoteAssetRequest : CustomYieldInstruction {
    // 読み込まれているEmoteAssetを返します。
    public EmoteAsset asset;
    // 動作が終了したか確認します
    public bool isDone;
};
```

## Transform

E-moteの「動き」に関するプロパティです。

## プロパティ



- Alignment
  - GameObjectの原点に対して、E-moteキャラのどの位置を合わせるかを設定します。
- Covolve Object Transform to Physics
  - チェックを入れると、EmotePlayerをアタッチしたGameObjectのTransformの座標移動と回転により、E-moteの物理計算が行われ、髪揺れ、胸揺れ、パーツ揺れが発生します。チェックを外すと揺れは起こりません。
- Fixed Scale
  - チェックを入れると、エディタで設定したキャラの身長設定を無視して規定のスケールでUnity座標系への座標変換を行うようになります。数値として0.01(1/100)を設定すると、Legacyアライメントとちょうど同等の倍率で表示を行います。

## API

```
class EmotePlayer {
    public enum TransformAlignment {
        LEGACY, // レガシーアライメント
        ORIGIN, // E-moteキャラの原点にアライメント
        TOP, // 頭頂部にアライメント
        BOTTOM, // 足下部にアライメント
        CENTER, // TOPとBOTTOMの中間にアライメント
        BUST, // 胸にアライメント
        MOUTH, // 口にアライメント
        EYE, // 目にアライメント
    };
    public TransformAlignment transformAlignment;
    public bool convolveObjectTransformToPhysics;
    public float fixedScale;
};
```

## Raw Level API

```
class EmotePlayer {
    // キャラの身長
    public float charaHeight;
    // キャラのマーカーの座標をワールド座標で取得します。
```

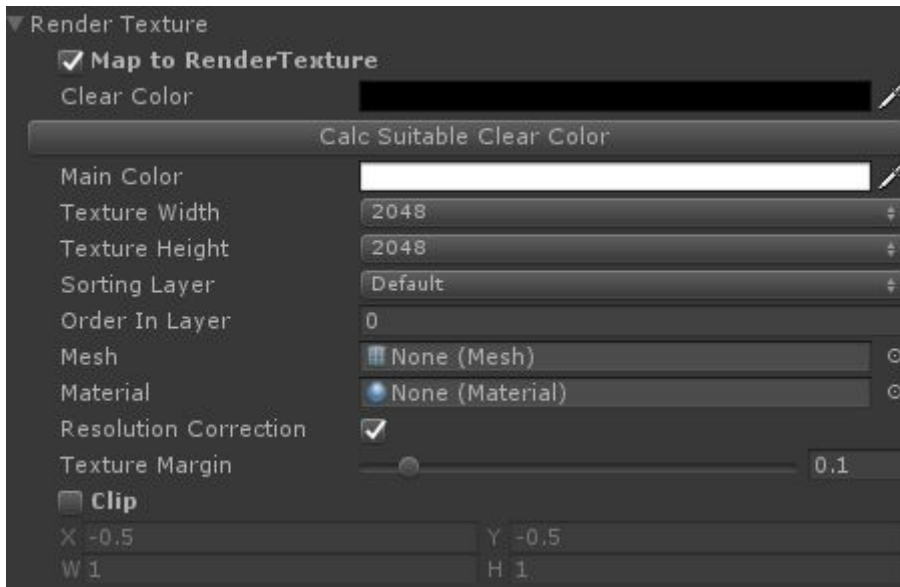
```
// key には “top”, “bottom”, “eye”, “mouth”, “bust” のいずれかを指定。  
public Vector3 GetCharaMakrer(string key);
```

```
};
```

## Render Texture

描画バッファであるレンダーテクスチャに関するプロパティです。

### プロパティ

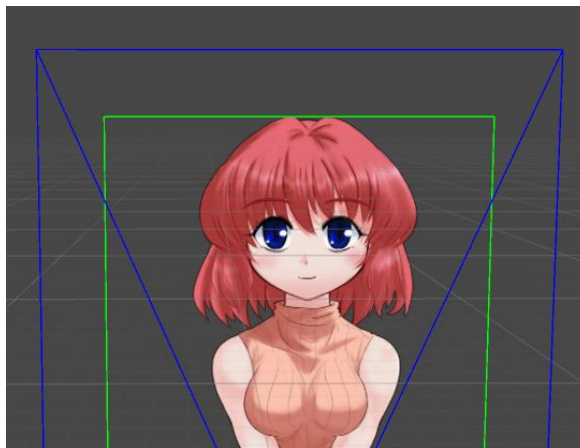


- Map to Render Texture [TYPE2]
  - TYPE2 SDKで、レンダーテクスチャにマップするかどうかを設定します。チェックを入れると、一旦レンダーテクスチャにE-moteの内容を描画してから、改めて描画を行うようになります。
  - TYPE1では常にレンダーテクスチャ経由で描画が行われます。
- Clear Color [TYPE2]
  - レンダーテクスチャのクリアカラーを設定します。
- Calc Suitable Clear Color [TYPE2]
  - モデルに設定された「適正なクリアカラー」を読み出し、クリアカラーを自動設定します。
  - 「適正なクリアカラー」はE-mote3.9.7p3以降のエディタで出力したモデルデータに自動的に付加されます。
  - モデルに「適正なクリアカラー」が設定されていない場合は、Unity上でエディタが計算するのと同じアルゴリズムでリアルタイムにその値を計算してクリアカラーに設定します。
- Main Color
  - レンダーテクスチャ全体に適用する頂点カラーです。
  - 中性色はRGBA=(1.0f, 1.0f, 1.0f, 1.0f) で、RGBに小さい値を指定すると暗くなります。

- vertexColorと違い、一旦描画を完了したレンダーテクスチャに適用する値なので、アルファ値を設定しても綺麗に半透明になります。

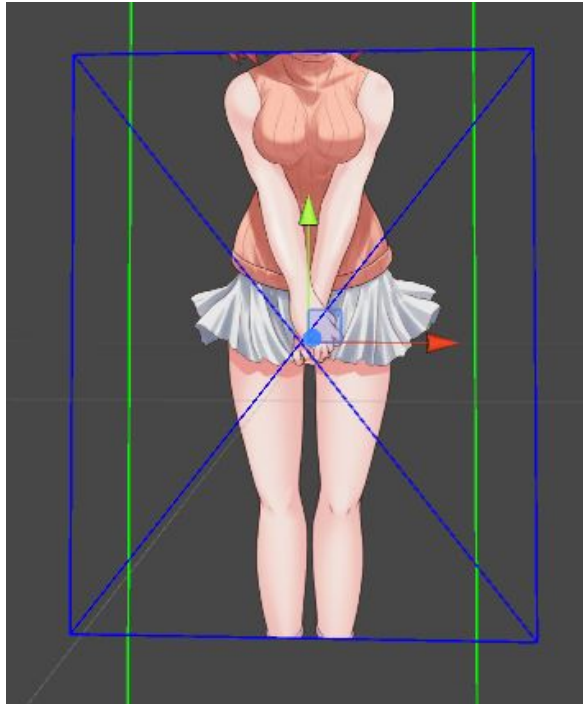


- Texture Width
  - レンダーテクスチャの横幅を指定します。
- Texture Height
  - レンダーテクスチャの縦幅を指定します。
- Sorting Layer [TYPE2]
  - レンダーテクスチャの Sorting Layer を指定します。
- Order In Layer [TYPE2]
  - レイヤー内での表示優先順位を指定します。
- Mesh
  - レンダーテクスチャを描画するメッシュを外部から指定します。
- Material
  - レンダーテクスチャを描画するマテリアルを外部から指定します。
- Resolution Collection [TYPE2]
  - レンダーテクスチャにマッピングする際にバッファブレンドのブラーやモザイクの効果量を、テクスチャ解像度に合わせて補正するかどうかを指定します。
- Texture Margin
  - レンダーテクスチャのマッピング範囲を決める際、バウンズの上下左右に加えるマージンの値をローカル座標系で指定します。
  - Sceneビューに緑色で表示されている枠がバウンズ、青色で表示されている枠がマージンを加えたレンダーテクスチャのマッピング範囲です。

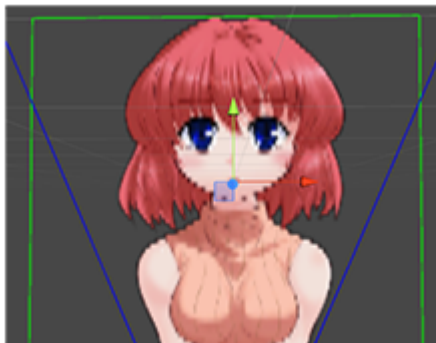


- Clip

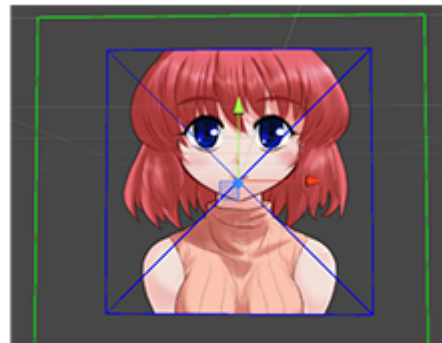
- レンダーテクスチャのマッピング範囲をクリップする矩形をローカル座標系で指定します。クリップを指定すると、レンダーテクスチャのマップ範囲はクリップと積算した領域に縮小されます。



- 画面外に表示せず切り落とす領域があることが分かっている時は、クリップを設定することでレンダーテクスチャの解像度を有効活用することが出来ます。以下に、同じ256x256のレンダーテクスチャでクリップの有無による解像度の違いが出るケースの例を示します。



**Texture Width : 256**  
**Texture Height : 256**  
**Clip: Off**



**Texture Width : 256**  
**Texture Height : 256**  
**Clip: On**

## API

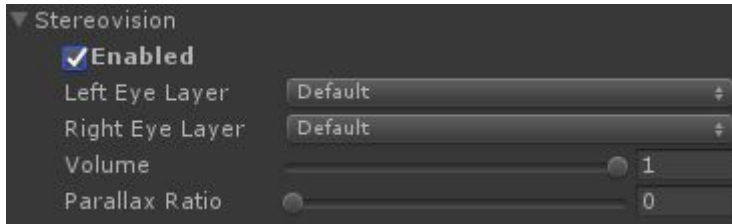
```
class EmotePlayer {  
    public bool mapToRenderTexture;  
    public Color renderTextureClearColor;  
    public Color32 suitableClearColor;
```

```
public Color mainColor;
public int texWidth;
public int texHeight;
public string renderTextureSortingLayerName;
public int renderTextureSortingOrder;
public Mesh renderTextureMesh;
public Material renderTextureMaterial;
public bool renderTextureResolutionCorrection;
public float renderTextureMargin;
public bool clipRenderTexture;
public Rect renderTextureClipRect;
public bool renderTextureCurve;
public float renderTextureCurveTopRate;
public float renderTextureCurveTopShift;
public float renderTextureCurveBottomRate;
public float renderTextureCurveBottomShift;
public float renderTextureCurveMeshCount;
public void calcSuitableRenderTextureClearColor();
};
```

## Stereovision [TYPE2]

立体視に関するプロパティです。

### プロパティ



- Enabled
  - 立体視を有効にするか無効にするかを設定します。
- Left Eye Layer
- Right Eye Layer
  - E-moteの立体視は、視差に対応した異なる2枚の画像を生成することで実現されます。右目用、左目用のカメラを個別に指定出来るように、描画対象レイヤを個別に指定可能になっています。なお、Unity5.6以降ではシングルカメラを対象とした立体視描画に対応しており、その場合レイヤを個別に指定する必要はありません。
- Volume
  - E-moteキャラに立体視をかける「度合い」を設定します。小さくすることで、徐々に立体視がかからなくなります。立体視の強度を動的に調整する必要がある場合に使います。
- Parallax Ratio
  - E-moteキャラの左右の稼働範囲の内どれだけの割合を視野角として扱うかを設定します。通常、EmoteBillboardControl がカメラとの距離に応じて自動的に値を設定するので、手動で制御する必要はありません。

### API

```
class EmotePlayer {  
    public bool stereovisionEnabled;  
    public int stereoVisionLeftEyeLayer;  
    public int stereoVisionRightEyeLayer;  
    public float stereovisionVolume;  
    public float stereovisionParallaxRatio;  
};
```

## Behavior

E-moteの「ふるまい」に関するAPIです。

## プロパティ



- Update Timing
  - E-moteのアバターを更新するタイミングを Update() と LastUpdate() から選択します。デフォルトの LastUpdate() のままにしておけば「E-moteを制御する側のスクリプトのUpdate()」より後に実行されるようになるので、通常はそれで問題ありません。
- Step Update
  - チェックを入れると「ステップ実行」を行うようになります。ステップ実行時は、アニメーションをはしより変数やタイムラインが更新されたタイミングのみ画像の更新処理を行うようになり、低速な環境でも余裕を持った動作をさせられます。
- Apply Time Scale
  - Time.timeScale の影響を受けるかどうかを設定します。チェックを外すと、Time.timeScale を無視して常に実時間でアニメーションするようになります。
- Speed
  - E-moteの再生速度を設定します。値を小さくするとスローに、大きくすると早回しになります。
- Hair Scale
  - キャラごとの髪の毛の揺れ係数の設定です。小さくすると髪が揺れなくなります。
- Bust Scale
  - キャラごとの胸の揺れ係数の設定です。小さくすると胸が揺れなくなります。
- Parts Scale
  - キャラごとのパーツの揺れ係数の設定です。小さくするとパーツが揺れなくなります。
- Wind Speed
  - 風の速度です。

- Wind Power Min
- Wind Power Max
  - 風力の最小値、最大値です。
- Skip
  - 現在進行中のタイムラインおよび変数の遷移を、「瞬間的に」終わらせます。
  - 動きを完全にリセットする際に使います。
- Pass
  - 現在進行中のタイムラインおよび変数の遷移を「なめらかに」終わらせます。
  - ADVのスキップシーンでダイアログを切り替える際などに使うと、早回しをしながらも滑らかにキャラの挙動を遷移していけます。

## API

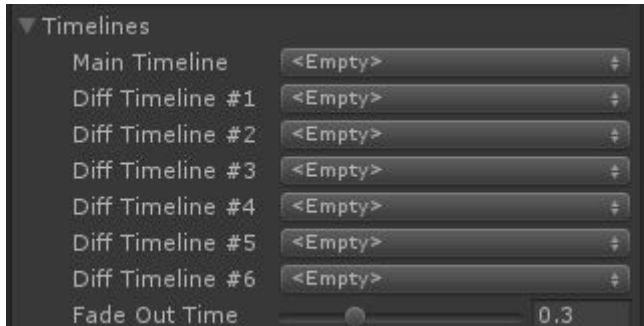
```
class EmotePlayer {
    public enum UpdateTiming {
        UPDATE,
        LATE_UPDATE
    };
    public UpdateTiming updateTiming;
    public bool stepUpdate;
    public bool applyTimeScale;
    public float speed;
    public float hairScale;
    public float bustScale;
    public float partsScale;
    public float windSpeed;
    public float windPowMin;
    public float windPowMax;
    public void Pass();
    public void Skip();
};
```

## Timeline

E-moteキャラの挙動を制御する「タイムライン」に関するプロパティです。

「タイムライン」とは、複数の変数の動きを時系列に沿って定義して、それに名前をつけた概念です。E-moteのキャラ制御は、この「タイムライン」単位の制御を基礎として行います。

## プロパティ



- Main Timeline
  - 「メインタイムライン」を指定します。
  - 「メインタイムライン」によりキャラに与えられた変化は恒久的なものとなり、「メインタイムライン」を切り替えた後もそのまま状態が残り続けます。
- Diff Timeline #1
- Diff Timeline #2
- Diff Timeline #3
- Diff Timeline #4
- Diff Timeline #5
- Diff Timeline #6
  - 「差分タイムライン」を指定します。
  - 「差分タイムライン」によりキャラに与えられた変化は一時的なもので、「差分タイムライン」を切り替えるとその影響は無くなります。
  - #1から#6まで6つのスロットが用意しており、「待機ループ」「会話向き」「仕草」など、カテゴリごとに使い分ける用途を意図しています。
- Fade Out Time
  - 差分タイムラインスロットの内容を切り替えた際、直前まで再生していた差分タイムラインは、ここで指定した時間をかけて徐々にフェードアウトします。これにより滑らかに差分タイムラインを切り替えられます。

## API

```
class EmotePlayer {  
    public string mainTimelineLabel;  
    public string diffTimelineSlot1;  
    public string diffTimelineSlot2;  
    public string diffTimelineSlot3;  
    public string diffTimelineSlot4;  
    public string diffTimelineSlot5;  
}
```

```

        public string diffTimelineSlot6;
        public float diffTimelineFadeOutTime
    };

```

### Raw Level API

一般的な使い方については上記の通常APIで足りるでしょう。ローレベルAPIでより高度な制御を行いたい場合、プラットフォームごとのE-mote Native SDKのリファレンスマニュアルに詳細な解説がありますので、そちらを参照して下さい。

```

class EmotePlayer {
    public enum TimelinePlayFlags : int {
        TIMELINE_PLAY_PARALLEL = 1 << 0,
        TIMELINE_PLAY_DIFFERENCE = 1 << 1
    };
    public string[] mainTimelineLabels;
    public string[] diffTimelineLabels;
    public bool IsLoopTimeline(string label);
    public float GetTimelineTotalSeconds(string label);
    public void PlayTimeline(string label, int flags = 0);
    public void StopTimeline(string label = "");
    public bool IsTimelinePlaying(string label = "");
    public void SetTimelineBlendRatio(string label, float value, float frameCount = 0,
float easing = 0, bool stopWhenBlendDone = false);
    public float GetTimelineBlendRatio(string label);
    public void FadeInTimeline(string label, float frameCount, float easing = 0);
    public void FadeOutTimeline(string label, float frameCount, float easing = 0);
    public struct TimelineInfo {
        public string label;
        public int flags;
    };
    public TimelineInfo[] GetPlayingTimelineInfoList()
};

```

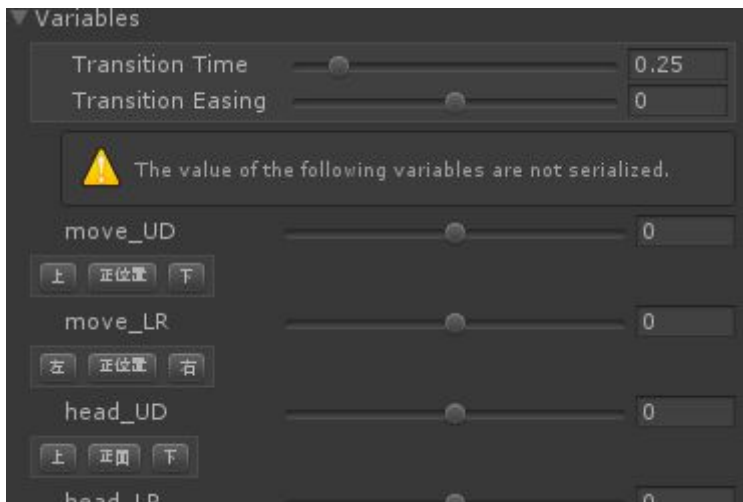
## Variables

E-moteキャラの挙動を制御する「変数」に関するプロパティです。

「変数」は「目の左右向き」「口の開閉」など、E-moteの挙動を制御する最小単位概念です。

キャラの演技をつける用途では主に前述の「タイムライン」を使う事が多く、変数を直接制御するケースは少ないです。

マウスカーソルに目線を追従させるなど、演技とは別にアプリからキャラの挙動を動的に制御したい場合は、変数差分設定機能で実現することが出来ます。



Inspectorの注意書きにある通り、このプロパティで設定した変数の値はどこにも保存されず、アプリをビルドしても反映されません。単純に、挙動の確認のために、定義されている全ての変数をレビューすることが出来るようになっています。

- Transition Time
  - 変数が新しい値へ遷移する時間を指定します。
- Transition Easing
  - 変数を新しい値へ遷移する際のイーasingを指定します。

## Raw Level API

```
class EmotePlayer {
    public class VariableFrame {
        public string label;
        public float value;
    };
    public class Variable {
        public List<VariableFrame> frameList;
        public string label;
        public float minValue;
        public float maxValue;
    };
}
```

```
};

public List<Variable> variableList
// 変数の値を変更する
public void SetVariable(string label, float value, float sec = 0, float easing = 0);
// 変数の値を取得する
public float GetVariable(string label);
// 変数の値に差分を加える。
// module名で異なる文字列を与えることで、
// 1つの変数に同時に複数の異なる変位を加えることが出来る。
public void SetVariableDiff(string module, string label, float value, float sec = 0,
float easing = 0);
// 変数の値の差分を取得する
public float GetVariableDiff(string module, string label);

};
```

## Logging

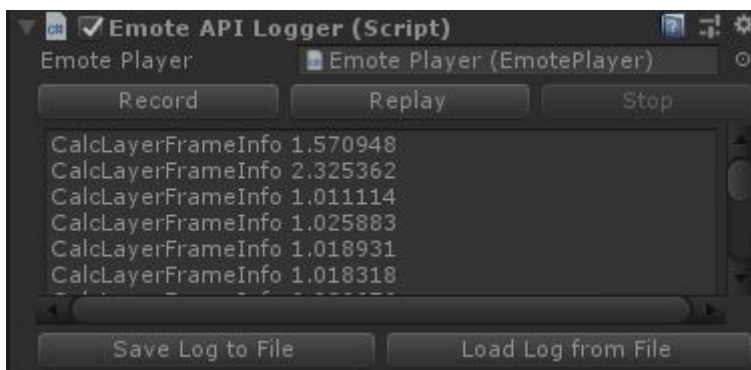
E-mote SDKは、デバッグ目的でAPIコールのログを記録・再生する機能を持っています。

## Raw Level API

```
class EmotePlayer {  
    // ログの記録を開始する  
    public void StartRecordAPILog();  
    // ログの記録を終了する  
    public void StopRecordAPILog();  
    // ログの記録中か判定する  
    public bool recordingAPILog;  
    // ログの再生を開始する  
    public void StartReplayAPILog();  
    // ログの再生を終了する  
    public void StopReplayAPILog();  
    // ログの再生中か判定する  
    public bool replayingAPILog;  
    // ログをクリアする  
    public void ClearAPILog();  
    // ログを文字列として取得・設定する  
    public string apiLog;  
};
```

## Component

補助コンポーネントである EmoteAPILogger.cs を EmotePlayer オブジェクトにアタッチすることで、Raw Level APIを直接叩かなくても、Inspector より簡単にAPIログを記録・再生することが出来ます。



## 応用的セットアップ

このセクションでは、様々な応用的なセットアップを紹介します。

また、それに伴い、いくつかの有用な補助スクリプトについて解説を行います。

### E-moteロゴの表示

#### サンプルシーン

Assets/EmotePlayer/Scenes/EmoteLogo.unity

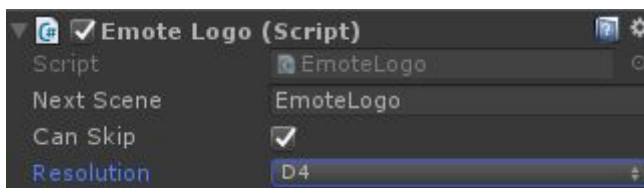
#### 構成

1. EmoteLogo.csと同等のセットアップを行う。

#### 概要

E-moteロゴの表示を行います。

#### プロパティ



- nextScene
  - ログの再生後に遷移するシーンを指定します。
- canSkip
  - プレイヤーの入力によりスキップ可能かどうかを設定します。
- resolution
  - ログのおおよその解像度をD2, D4, D5 から選択します。アプリケーションの実行解像度に合わせて選んでください。

## Generate Test

### サンプルシーン

Assets/EmotePlayer/Scenes/GenerateTest.unity

### 構成

1. Cameraを作成します (TRS=(0,0,-1.5):(0,0,0):(1,1,1))
2. Emote Generate Test コンポーネントをCameraにアタッチします

### 概要



E-moteキャラをスクリプトから動的に生成するサンプルです。

また、E-moteが占有する各種デバイスリソースの確保タイミングを管理する EmoteDeviceManager の動作テストにもなっています。

E-moteは、その表示のために専用のヒープやVRAM領域、シェーダーなど様々なデバイスリソースを必要とします。

これらのデバイスリソースは通常「ひとつでもEmotePlayerがアクティブになった」瞬間に確保され、「最後のEmotePlayerが無効になった」瞬間に破棄されます。

ですが、デバイスリソースの確保・破棄はそれなりに重い処理であるため、EmotePlayer の生成とは独立して任意のタイミングでデバイスリソースの生成タイミングを制御する仕組みが用意されています。

EmoteDeviceManager クラスのオブジェクトを作成して、Load()関数、Unload()関数でそれぞれデバイスリソースのロード、アンロードを任意のタイミングに実行出来ます。

さらにより簡便な方法として、Emote Device コンポーネントをシーン中の任意のオブジェクトにタッチしておくことで、シーンのロードと同時にE-moteのデバイスリソースが確保され、シーンのアンロードと同時にデバイスリソースが破棄されるようになります。

### API

```
class EmoteDeviceManager {  
    public void Load();  
    public void Unload();  
    public bool loaded;  
};
```

## Billboard セットアップ

### サンプルシーン

Assets/EmotePlayer/Scenes/BillboardSetup.unity

### 構成

1. 空のGameObject (TRS=(0,0,0):(0,0,0):(1,1,1))を生成します。
2. GameObjectに部屋モデル、Models/room をアタッチします。
3. Camera (TRS=(0,0,-1):(0,0,0):(1,1,1))を生成します。
4. Cameraに Emote Fps Camera Control コンポーネントをアタッチします
5. Clipping Planesの near を0.1に変更します。
6. 空のGameObject (TRS=(0,0,0):(0,0,0):(1,1,1))を生成します。
7. GameObjectに Emote Player コンポーネントをアタッチします。
8. Transform -> Alignment に Align to bottom を指定します。
9. Data File に vr\_girl を指定します
10. Emote Billboard Control コンポーネントをアタッチします。
11. Emote Fps Control コンポーネントの Target Player に EmotePlayer を指定します。
12. Emote Billboard Control の Target Camera に Camera を指定します。

### 概要

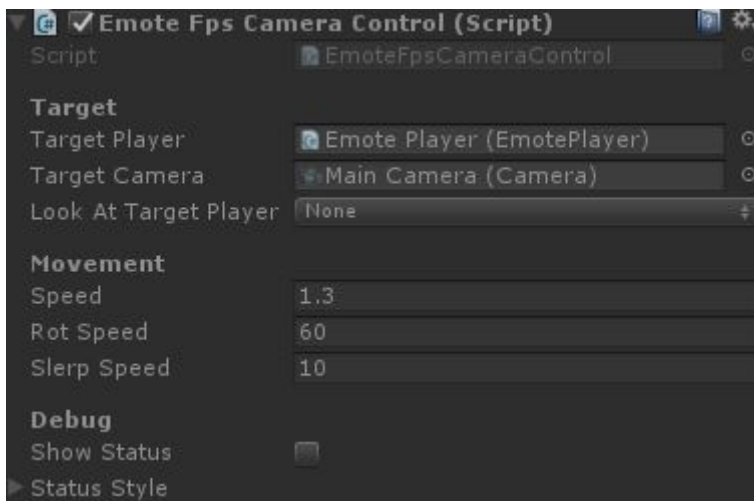


E-moteキャラを三次元に空間に配置し、常にカメラへ相対するビルボードとして制御するサンプルです。

Camera にアタッチした Emote Fps Camera Control コンポーネントは、デバッグ用にカメラのFPS 風キー操作を可能にするスクリプトです。

Emote Player の GameObject にアタッチした Emote Billboard Control は、E-moteキャラを常にカメラに正対させ、ビルボードとして制御します。E-mote3.8のVR仕様テンプレートで作られたキャラは、「まず視線を向け、次に首を動かし、最後に身体を向ける」と言う自然な動作でカメラに追従するようになります。

### Emote Fps Camera Control

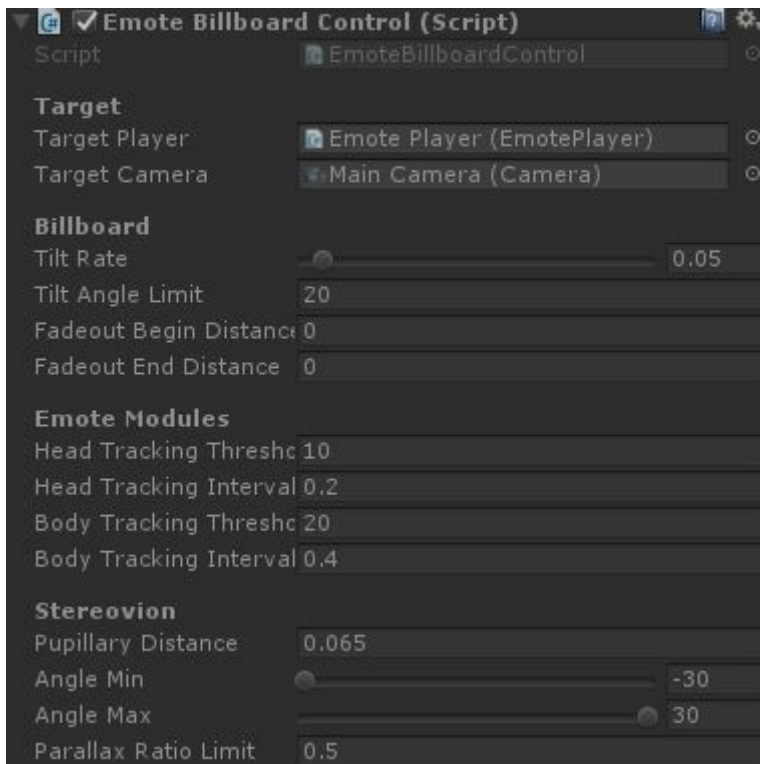


- Target Player
  - ターゲットとなる EmotePlayer を指定します。
- Target Camera
  - ターゲットとなる Camera を指定します。
- Look At Target Player
  - キャラの身体の特定のポイントを注視するように設定します。
- Speed
  - 移動速度を設定します。
- Rot Speed
  - 旋回速度を設定します。
- Slerp Speed
  - 移動や旋回で慣性が残る度合いを指定します。
- Show Status
  - チェックを入れると、変数の現在値が表示されます。

キーバインドは以下の通りになります。

- W/S/A/D      前/後/左/右
- Q/E            左回転/右回転
- ↑/↓            上昇/下降
- ←/→          視点下/視点上
- L                注視点変更

## Emote Billboard Control



- Target Player
  - ターゲットとなる EmotePlayer を指定します。
- Target Camera
  - ターゲットとなる Camera を指定します。
- Tilt Rate
- Tilt Angle Limit
  - カメラがキャラに極端に近付いた時に、ビルボードを奥側に倒して板っぽさを軽減する制御をする、その度合いと限度角を指定します。
- Fadeout Begin Distance
- Fadeout End Distance
  - カメラがキャラに近付き過ぎた時にキャラがくるくる回転してしまう様子を見えなくするための、フェードアウト制御範囲を設定します。Begin Distance まで近付くとフェードアウトを開始し、End Distanceでフェードアウトを完了します。レンダーターゲットが有効な場合はmainColor、無効な場合はvertexColorのalpha値でフェードの制御を行います。
- Head Tracking Threshold
- Head Tracking Interval
- Body Tracking Threshold
- Body Tracking Interval
  - 頭と身体のそれぞれについて、トラッキングを開始する境界となる角度と、トラッキングを開始してから完了するまでの時間を設定します。
- Pupillary Distance

- 左目と右目の間の視差を設定します。立体視が有効な場合は、キャラとカメラ間の距離、角度とこの値を元に、EmotePlayer.stereovisionParallaxRatio が自動設定されます。
- Angle Min
- Angle Max
  - E-moteキャラの左右設定変数に設定されている見た目上の角度のレンジを指定します。
- Parallax Ratio Limit
  - カメラがキャラに近付いてもこれ以上は上げないという視差率の最大値を設定します。

## LipSynch セットアップ

### サンプルシーン

Assets/EmotePlayer/Scenes/LipSynchSetup.unity

### 構成

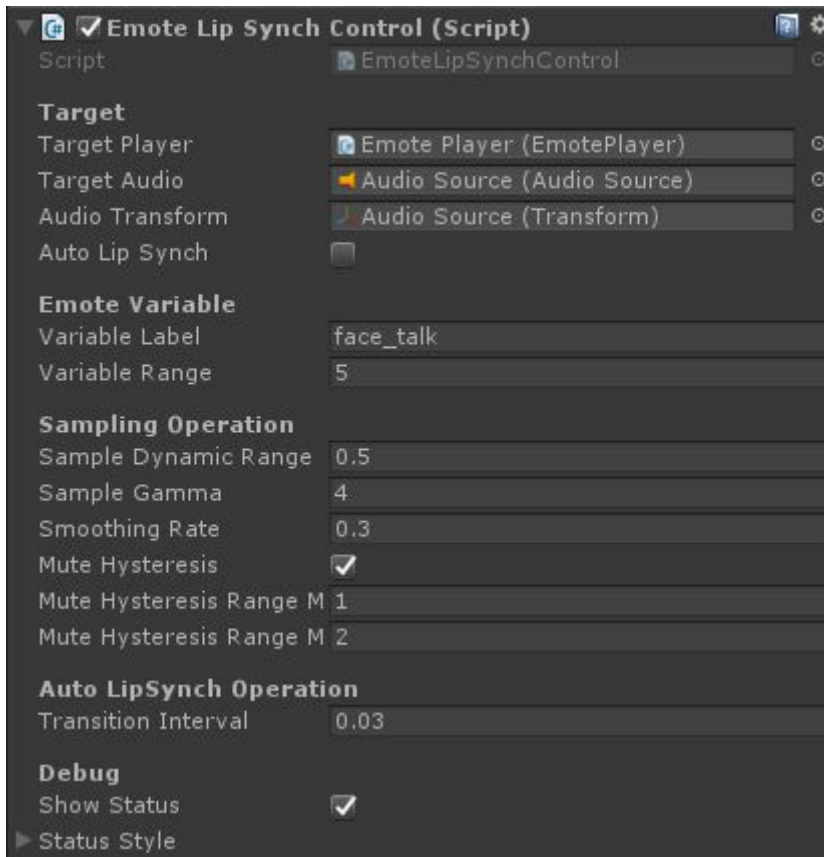
1. まず、[Billboard セットアップ](#)と同等の設定を行う。
2. Emote Player の PSB File を vr\_boy に変更する
3. Emote Player の GameObject の子として新しい Game Object を作成し、Audio Sourceをアタッチする。
4. Audio Source の Spatial Blend を 3Dに振り切る
5. Emote Lip Synch Control コンポーネントをアタッチする
6. Targe Player に親のEmote Player、Target Audio と Target Transformに自分自身のGame Object を指定する。
7. Show Status にチェックを入れる

### 概要



Audio Sourceの値を動的にキャプチャして、E-moteに自動で口パクをさせるスクリプトです。

## Emote Lip Synch Control



- Target Player
  - ターゲットとする Emote Player を指定します
- Target Audio
  - ターゲットとする Audio Source を指定します。
- Audio Transform
  - Audio Source をアタッチした GameObjectのTransformを指定すると、そのオブジェクトはキャラマーカ “mouth” の位置に自動的に追従して、3D的に正しい口の位置で発声するようになります。
- Auto Lip Synch
  - チェックを入れると、ボイスの再生と無関係にランダムに口パクをするようになります。
- Variable Label
- Variable Range
  - E-mote の口パクに対応した変数の名前とレンジを指定します。
- Sample Dynamic Range
  - サンプルした音声のダイナミックレンジを指定します。
- Sample Gamma
  - サンプルした音声を補正するガンマ係数を指定します。
- Smoothing Rate
  - サンプルを直前のフレームの値と合成してスムーシングをかける係数を指定します。
- Mute Hysteresis

- Mute Hysterisis Range Min
- Mute Hysterisis Range Max
  - チャタリングを防ぐため、口の「開き」「閉じ」を判定する境界領域を設置するかどうかを指定します。
- Transition Interval
  - Auto Lip Synch 時に、口が次の値まで遷移する時間を指定します。小さくすれば口パクが速くなり、大きくすれば口パクが緩やかになります。
- Show Status
  - チェックを入れると、音声の再生ボタンがGUIに表示されます。

## Multiple セットアップ [TYPE2]

### サンプルシーン

Assets/EmotePlayer/Scenes/MultipleSetup.unity

### 構成

1. まず、[Billboard セットアップ](#)と同等の設定を行う。
2. Emote Player のゲームオブジェクトを複製する
3. 片方の PSB File を vr\_boy に変更する
4. Position を調整し、0.4メートルほど間隔を離す
5. Cameraに Emote Depth Sorter コンポーネントをアタッチする

### 概要



TYPE2 SDKでは、EmotePlayer の OnRenderObject イベントが呼ばれたタイミングでキャラの描画を発行しています。そのため、デフォルト状態ではカメラに対してキャラの前後関係を正しく処理出来ません。(Map to Render Texture にチェックを入れている場合は、通常のQuadを使って描画を行うのでこうした問題は起きません)。

これに対処するには、描画を実行するカメラに Emote Depth Sorter コンポーネントをアタッチして下さい。カメラと EmoteのGameObject の距離に応じて、自動的にソートが行われ、正しい順序でキャラが描画されるようになります。

OnRenderObject() イベントは「カメラの通常のレンダリングが全て終わった」後で呼ばれます。  
TYPE2 の描画は「そのカメラの通常描画が全て終わった後に、半透明ポリゴンと同等の扱い(Z値を  
更新しない)」で実行されます。

E-moteオブジェクトのさらに上にUIなどを表示したい場合は、E-moteのカメラよりもDepth設定の  
大きい追加のカメラを作成し、E-moteの描画が終わった「後」にそれらの描画を行うように構成して  
下さい。

## Stereovision セットアップ[TYPE2]

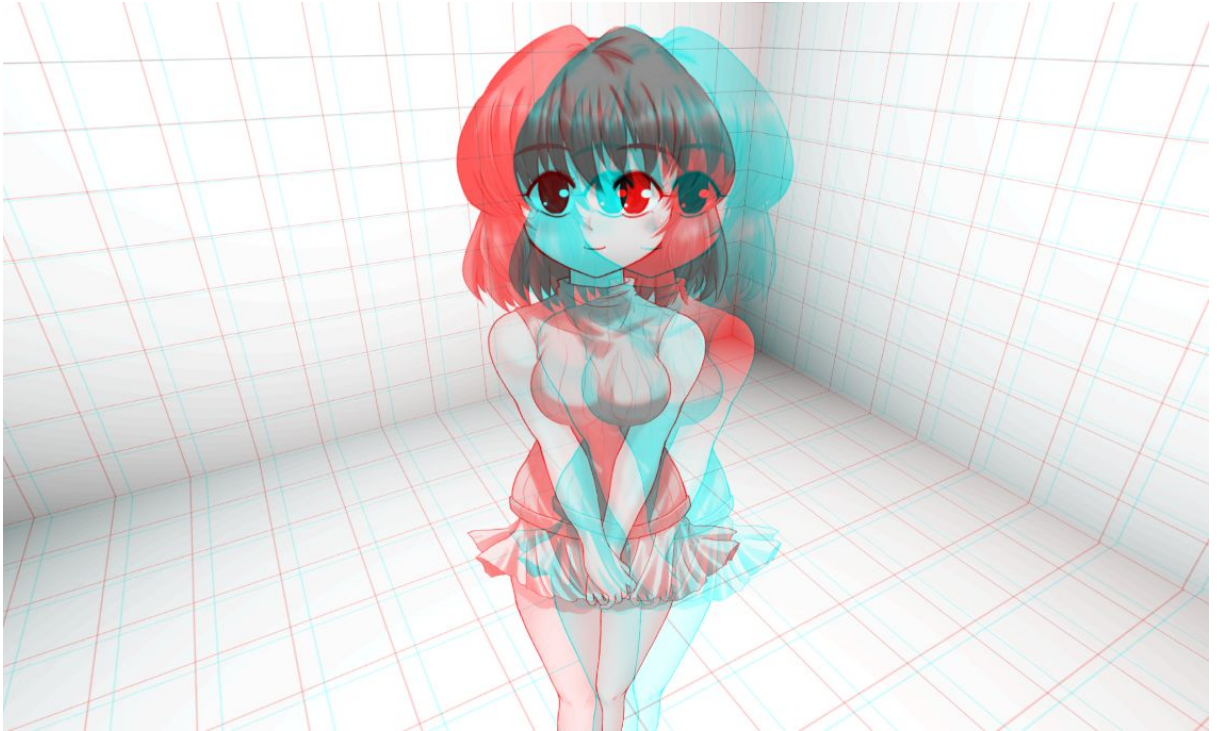
### サンプルシーン

Assets/EmotePlayer/Scenes/StereovisionSetup.unity

### 構成

1. まず、[Billboard セットアップ](#)と同等の設定を行う。
2. Layer 設定を変更し、“User Layer 8”を“Left Eye Layer”、“User Layer 9”を“Right Eye Layer”と名付ける。
3. Emote Player の Stereovision → Enable にチェックを入れる。Left Eye Layer と Right Eye Layer に先ほど名付けたレイヤをそれぞれ設定する。
4. -----  
-----ここまでが、立体視用に右目と左目に別々の画像を作り出すための構成で、これ移行は、生成した画像を異なるカメラでレンダリングバッファに描き出し、合成して赤青立体視用に合成した描画を行わせるための構成です。  
-----
5. Layer 設定を変更し、“User Layer 10”を“Anaglyph Composite”と名付ける。
6. レンダリングテクスチャ Left Eye Texture と Right Eye Texture を任意のサイズで作成する。
7. Main Camera の Culling Mask を nothing に変更して何も移さないようにする。
8. Main Camera の下に新たに2つのカメラ Left Eye Camera, Right Eye Camera を追加する。Left Eye Cameraは左に -0.0325, Right Eye Camera は右に0.0325 ずらし、視差をつける。
9. Left Eye Camera の near clip を0.1に変更。Culling Mask を Default と Left Eye Layer のみに変更。Target Texture に Left Eye Textureを設定する。
10. Right Eye Cameraも同様に設定する。
11. 新規に赤青合成用のカメラ Anaglyph Composite Camera を作成する
12. Anaglyph Composite CameraのCulling Mask を Anaglyph Composite のみに変更する。
13. Anaglyph Composite Cameraに2つのGameObject を追加し、それぞれLeft Eye Quad と Right Eye Quad と名付ける
14. Left Eye Quad にQuadをアタッチ。レイヤをAnaglyph Composite に変更。シェーダー Emote/AnaglyphComposite をアタッチ。Color を (1,0,0) に設定。Texture に Left Eye Texture を設定。
15. Right Eye Quad も同様に設定。Color は (0,1,1)
16. Anaglyph Composite CameraにEmote Anaglyph Composite コンポーネントをアタッチする。Left Eye Camera, Right Eye Camera, Left Eye Quad, Right Eye Quad の各項目を適切に設定する。

## 概要



ポピュラーな赤青メガネで見られるアナグリフレンドリングとして立体視可能な映像を出力するセットアップです。

VR用のE-moteモデルの調整などを、手軽に行う目的で利用することができます。

Gキーを押すと、「カラーアナグリフ」と「モノクロアナグリフ」を切り替えることができます。前者ではある程度色情報も確認出来ますが、より疲れやすくなっています。

アナグリフの閲覧は目と脳みそへの負担が大きいため、長時間続けないようにお気をつけください。

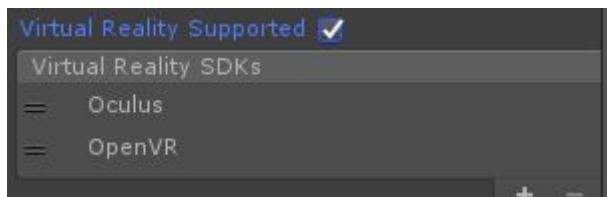
## VRへの導入(Unity5.6以降) [TYPE2]

VRの構成例として、[Billboard セットアップ](#)をベースに、VR環境にE-mote VRを導入するステップを解説します。

Unity5.6以降では、Unityの単一カメラによる立体視描画に対応しているため、非常に手軽にVR環境へE-mote VRを導入することが出来ます。

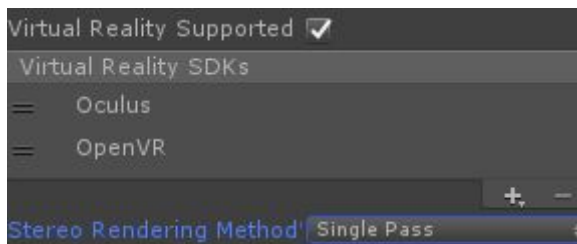
### 構成

1. まず、[Billboard セットアップ](#)と同等の設定を行う。
2. Emote Player の Stereovision -> Enable にチェックを入れる。
3. Player SettingでVirtual Reality Supported にチェックを入れる



### 注意点

- Unity2017.2以降の環境で「シングルパスステレオレンダリング」を使用する場合は、必ず EmotePlayerの「Map to RenderTexture」にチェックを入れて、レンダーテクスチャに対してレンダリングをするように指定してください。そうしないと立体視が有効になりません。



- シングルパスステレオレンダリングを利用する場合、単一のシェーダーで、シェーダーパラメータを参照して右目用と左目用のテクスチャのサンプリングを切り替えています。外部マテリアルを設定する場合、「\_MainTex」で左目、「\_MainTex2」で右目のテクスチャを参照し、unity\_StereoEyeIndex の値を見てサンプリングテクスチャを切り替えるシェーダーを記述してやってください。詳細は EmotePlayer/Scripts/Resources/EmoteStereovisionSprite.shader の実装を参考にしてください。

## Steam VRへの導入(Unity5.5以前) [TYPE2]

VRの構成例として、[Billboard セットアップ](#)をベースに、Steam VRにE-mote VRを導入するステップを解説します。

なお、この構成方法はUnity5.5以前に対するオプションで、Unity5.6以降では前述の方法でよりシンプルなセットアップが可能です。

### 構成

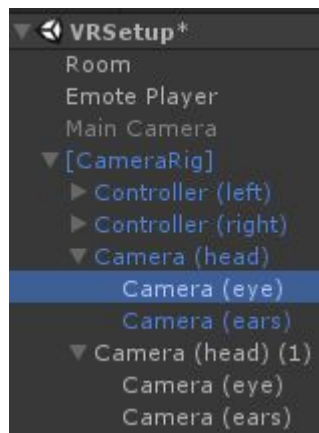
4. まず、[Billboard セットアップ](#)と同等の設定を行う。
5. Layer 設定を変更し、“User Layer 8”を“Left Eye Layer”、“User Layer 9”を“Right Eye Layer”と名付ける。
6. Emote Player の Stereovision → Enable にチェックを入れる。Left Eye Layer と Right Eye Layer に先ほど名付けたレイヤをそれぞれ設定する。
7. AssetStore からSteamVR Pluginをインポートする
8. 既存のMain Cameraを無効化する



9. SteamVR/Prefabsより、[CameraRig] をインスタンスエイトする
10. Emote Player の Emote Billboard Control コンポーネントの Target Camera を、Camera (eye) に付け替える。
11. Camera (head) を Duplicate する。



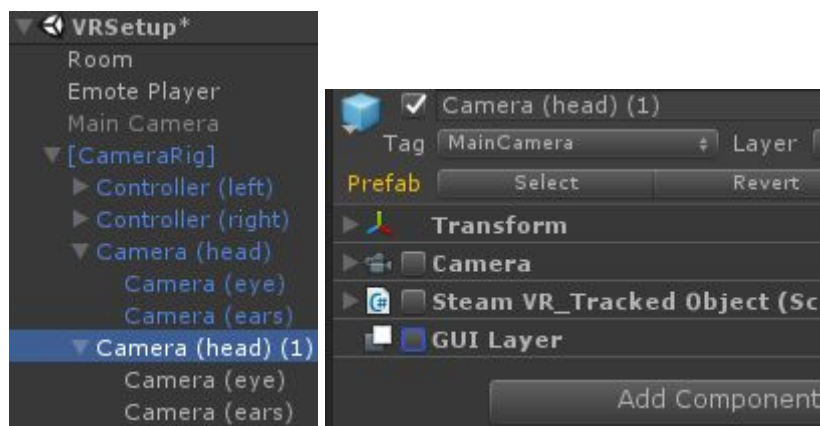
12. 複製元の Camera (eye) の、Culling Mask から Right Eye Layer を外し、Target Eye を Left に設定する。



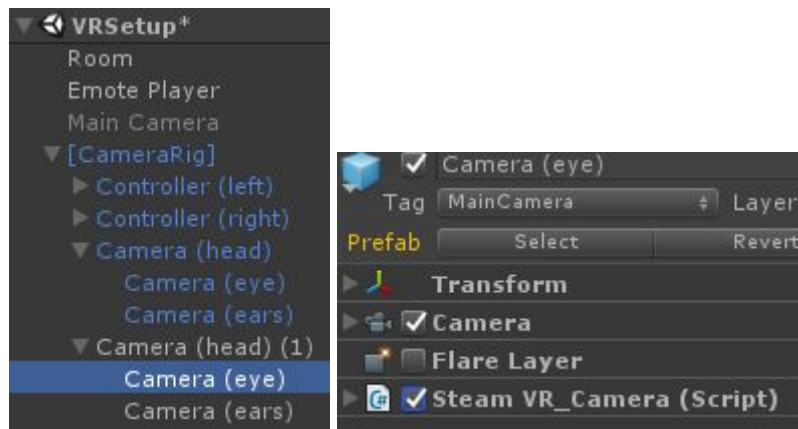
13. 複製先の Camera (eye) の、Culling Mask から Left Eye Layer を外し、Target Eye を Right に設定する。



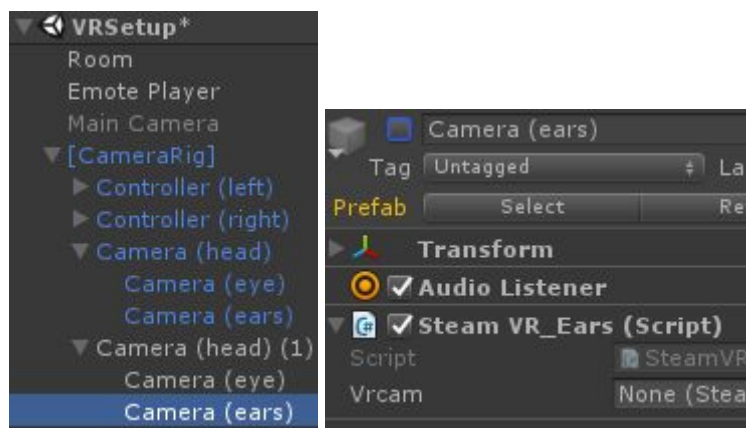
14. 複製先の Camera (head) の Camera, Steam\_VR\_Tracked Object, GUI Layer を無効化する。



15. 複製先の Camera (eye) の Flare Layer を無効化する。



16. 複製先の Camera (ear) を GameObjectごと無効化する。



## VR向けテクスチャカーブの設定

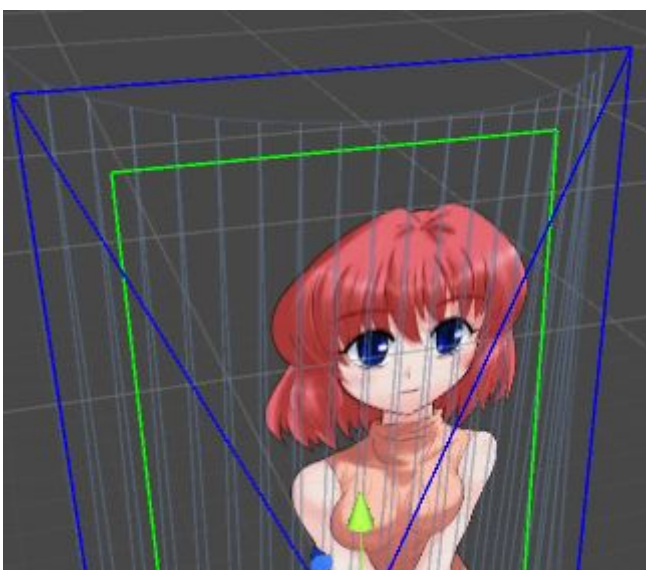
### 構成

1. EmotePlayerに、EmoteTextureCurve.csをアタッチします。

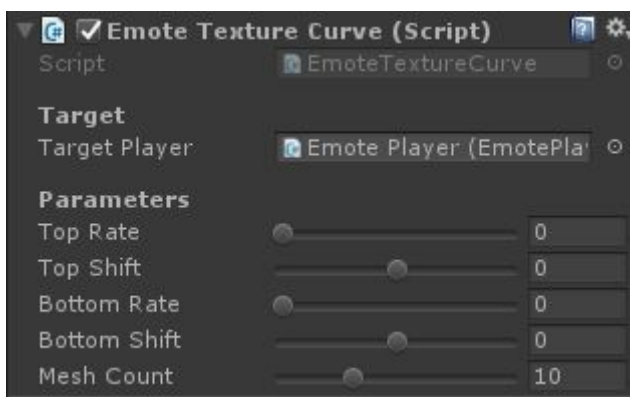
### 概要

VRはビルボードを表示した際の平面感を軽減するには、メッシュをあらかじめ円筒上に湾曲させておく対策が有効です。

EmoteTextureCurve スクリプトではそうしたVR向けのメッシュを生成できます。



### プロパティ



- Target Player
  - ターゲットとなるEmotePlayerを指定します。
- Top Rate
  - 頭頂部側の湾曲度合いを0から1の範囲で設定します。

- Top Shift
  - 頭頂部側を「奥に引っ込めるか」「手前に出すか」を-1から1の範囲で設定します。
- Bottom Rate
  - 足下部側の湾曲度合いを0から1の範囲で設定します。
- Bottom Shift
  - 足下部側を「奥に引っ込めるか」「手前に出すか」を-1から1の範囲で設定します。
- Mesh Count
  - 横方向のメッシュの分割数を設定します。

## 既知の不具合

以下の不具合は近日中にリリースするバージョンでの修正を予定しています。

- Emote Player のプロパティへの操作にUNDO/REDOが効かない