

Ball Dance/Visual Compositor セットアップ

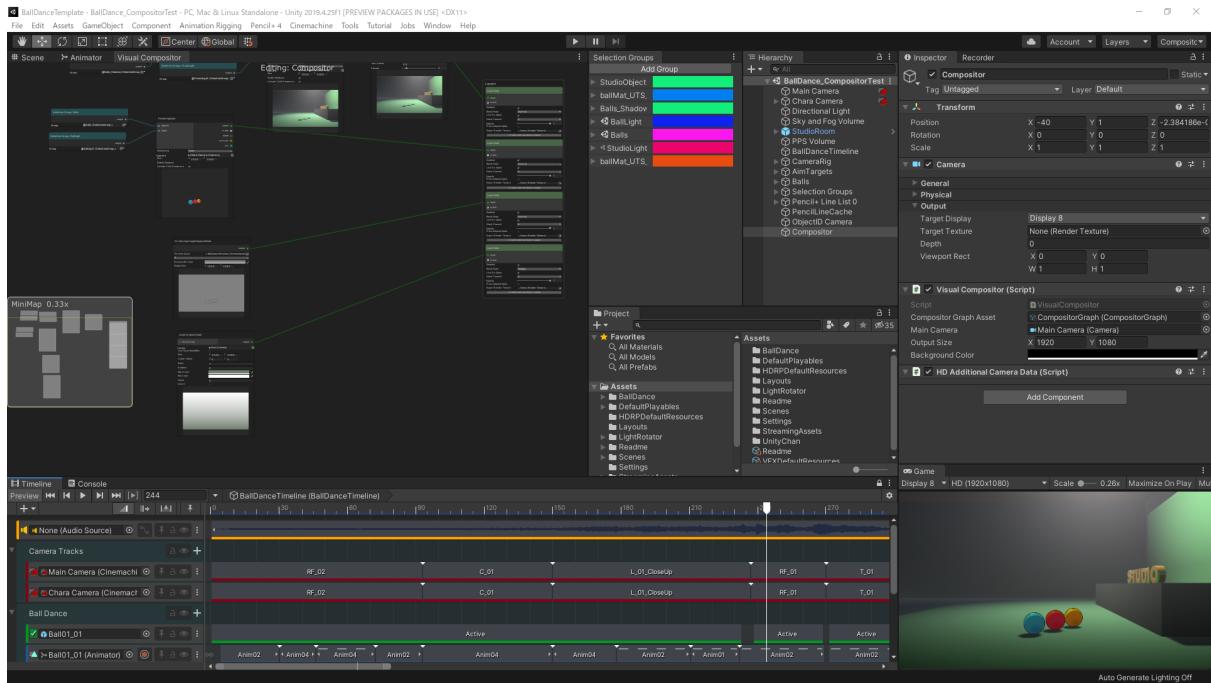
2020/09/20 N.Kobayashi/ UTJ

2020/12/14 2020_12 version用に更新

2021/04/30 2021-05-10 Release用に更新

2021/08/30 2021-09-02 Release用に更新

2022/08/10 BallDance Template-0.13.1-exp. の画面に更新



はじめに

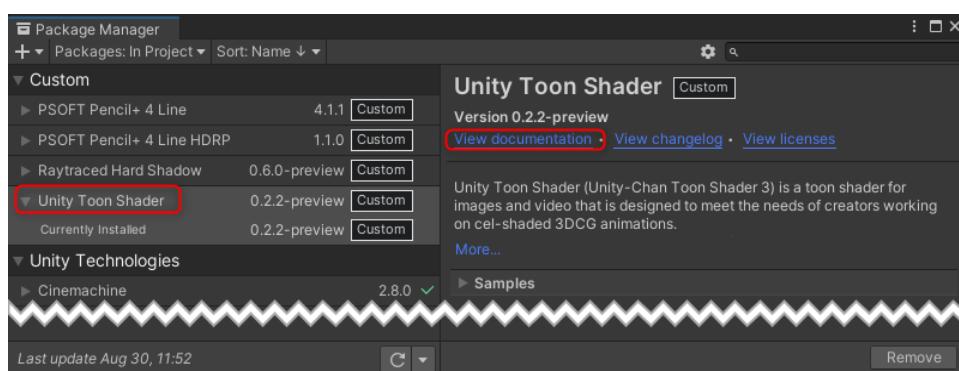
- 本ドキュメントは、BallDanceTemplate-0.xx.x-exp projectに含まれる、BallDance_CompositorTest.unityのセットアップ解説です。BallDance.unityはその元になったシーンで、Visual Compositorでの合成工程を含んでいないものです。
- 本ドキュメントでは、セルルック3DCGアニメーション制作過程での「編集撮影」作業に主に用いられるVisual Compositorの使い方を、ステップバイステップで解説します。
- 本ドキュメントの前に、「編集撮影」作業前のシーンが完成していることが前提になりますが、そちらの解説は『Ball Dance/PreViz・Cinemachine セットアップ』を参照してください。
- なお、本プロジェクトの機能を全て正しく表示するためには、Pencil+ 4 Line for Unityの有償ライセンスが必要となります。
ライセンスが有効でない場合、Pencil+ 4 Line for Unityのリアルタイム更新を有効にすると、ウォーターマークがゲームビューに現れます。
なお、Pencil+ Line Cacheのレンダリング結果のみを表示する場合には、ライセンスは不要です。

【NEW】過去バージョンとの互換性の面での注意

【2021-09-02 Release】

- 【重要】UTS/HDRPがUnity Toon Shader@0.2.2-previewになりました。ToonShader@0.2.2-previewより、UTS/HDRPをHDRP環境で使用した場合での、物理ライトのインтенシティ(明るさ)への反応が、HDRP/LITなどと同様に、明るさの物理単位「ルクス(Lux)」に対して正しく反応するようになっています。そのためのインターフェースとして、Toon Ev Adjustment Curveが追加されています。

詳しい機能説明は、Package Managerウィンドウより、Unity Toon Shaderを選択し、View documentationよりドキュメントをご確認ください。



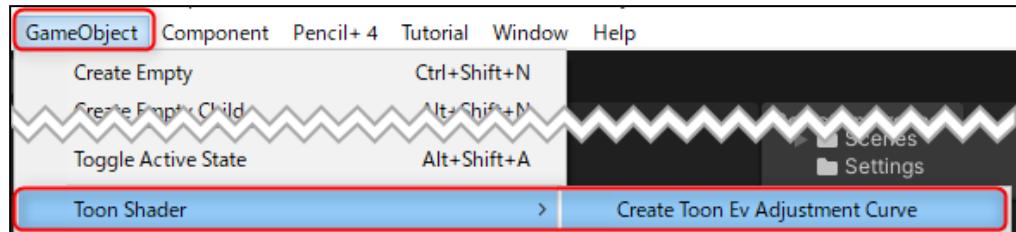
ドキュメントの冒頭に、「【NEW】HDRP Toon EV Adjustment Curve」へのリンクがありますので、そちらを利用していただくとよいでしょう。

UTS/HDRPの今回の仕様変更の結果、従来、トゥーンシェーダー用にリアルタイムディレクショナルライトのインтенシティを「1」で設定していたプロジェクトの場合、「1」のままではトゥーンシェーダーマテリアルが大変暗く表示されることになります。これは、「1」の値が従来と違い「1 Lux」として正しく評価されることになるからです。

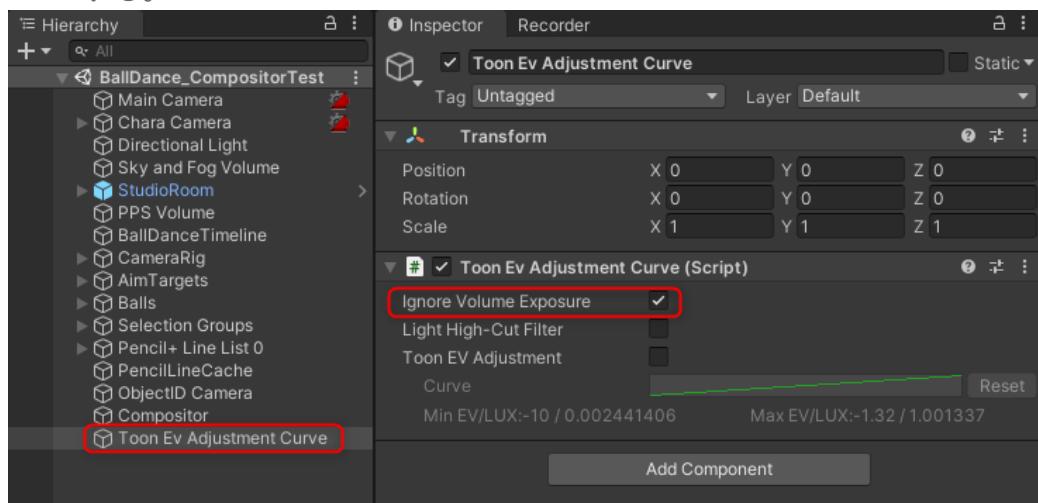
従来通りのマテリアルの明るさに合わせたり、Visual Compositor上でキャラクター用のライトとシーンマテリアル用のライトとで、インтенシティをわけているような場合には、以下の手順でシーン中にToon Ev Adjustment Curveを追加することで、元通りの表示にすることができます。

新規にシーン中にToon Ev Adjustment Curveを追加し、Ignore Volume Exposureスイッチをオンにする

1. メニューバーより、GameObject > Toon Shader > Create Toon Ev Adjustment Curveと進み、シーン中に新規のToon Ev Adjustment Curveゲームオブジェクトを配置する。



2. ヒエラルキーインデウより、新規に配置されたToon Ev Adjustment Curveゲームオブジェクトを選択し、インスペクターウィンドウより、Toon Ev Adjustment Curveコンポーネントの「Ignore Volume Exposure」スイッチをONにする。



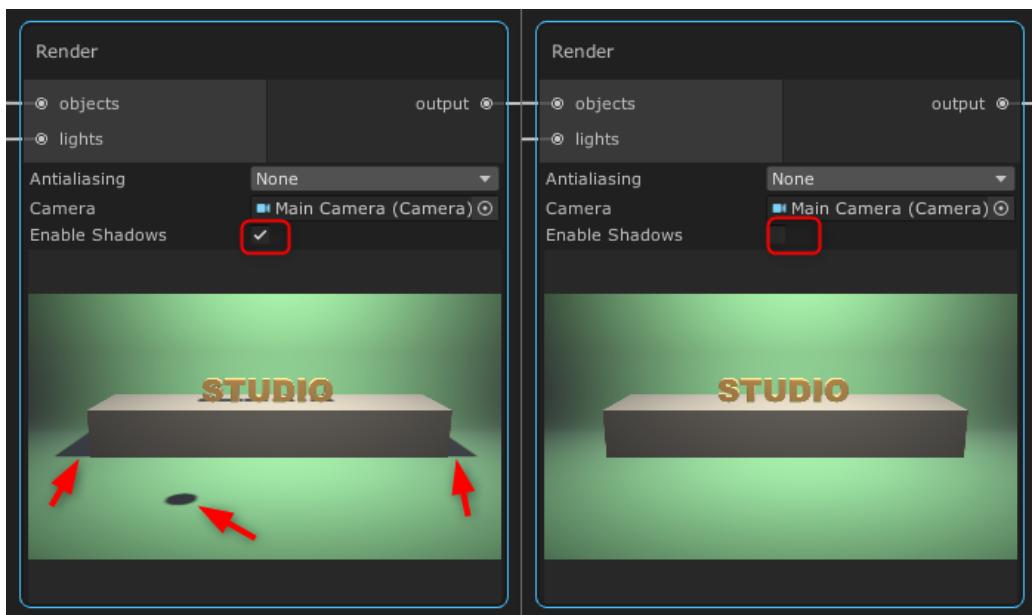
【2021-05-10 Release】

- Visual Composerの機能を提供するコンポーネントの名前が、Visual Composerに変更になりました。(従来は、Composer Component)
- Visual Composerの機能が大幅に追加されました。それに伴い、**前バージョンまでのVisual Composerノードおよびそれで作成したノードグラフに互換性がなくなっています。**
すでに前バージョンのVisual Composerでノードグラフを作成している場合は、前バージョンからAnimeToolBoxをバージョンアップする前に、ノードグラフをスクリーンショットするなどをして、再度新バージョンで同様に組み直すようにしてください。
お手数をおかけしますが、何とぞよろしくお願いします。
- Maskノードの機能が変更になりました。従来Maskノードで行っていた、背景からのα抜きの為の手順は、**Rendering**ノード側の一機能として統合されました。詳しくは該当する説明を確認してください。

- グラデーション系のノードのパラメタが変更になりました。前バージョンのVisual Compositorでノードグラフを作成した際に設定したパラメタは、新バージョンの該当するノードでは再度見直すようにしてください。
- 本ドキュメントでのVisual Compositorからのゲームビュー出力のディスプレイ番号が、従来のDisplay 3からDisplay 8に変更になりました。機能的には変わりません。

【2020_12 version】

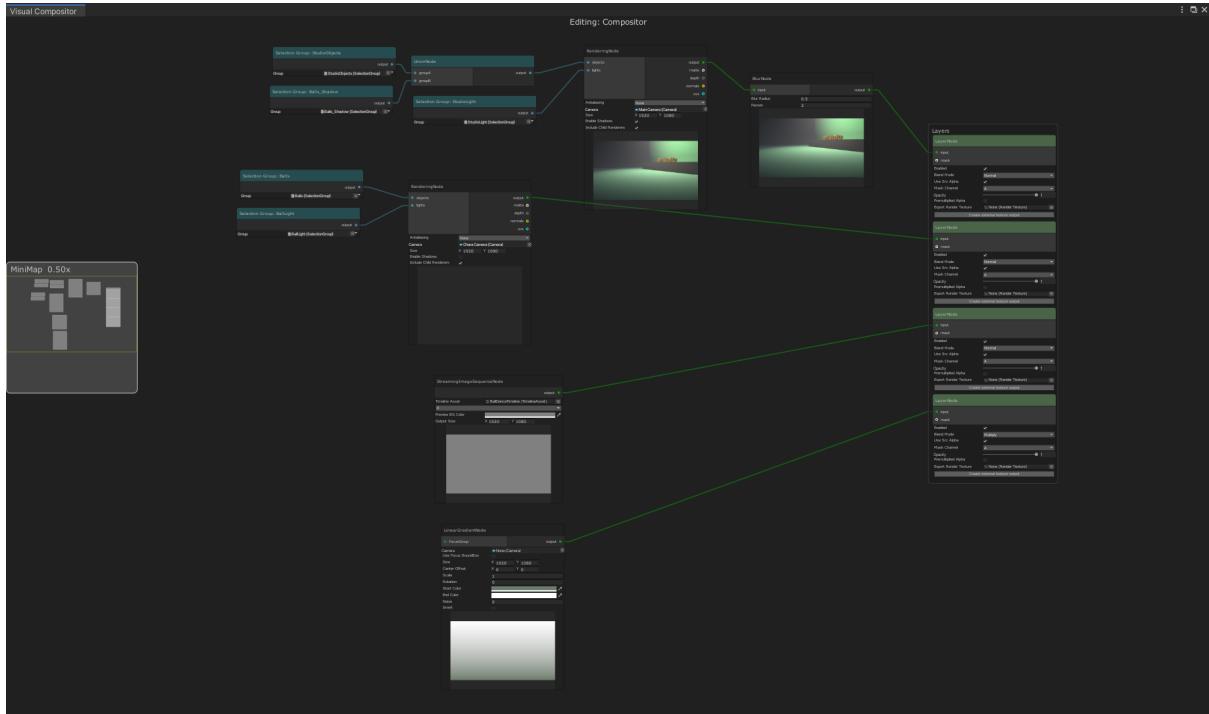
- 2020_12 versionでの大きな修正として、Renderingノードに「落影(ドロップシャドウ)をレンダリングするための**Enable Shadows**チェックボックス」が追加されました。ドロップシャドウやレシーブシャドウをレンダリングする時には、このチェックボックスを有効にしてください。(デフォルトでは無効になっています。チェックを付けることで、以前のバージョンとの挙動が同じになります。)



仕様上の注意

- 現在、Visual Compositorで加工できる最大解像度は、4200x3000ピクセルを最大の目安としてください。これはSRPの仕様上の制限ですので、今後改善される予定があります。(時期未定)

Visual Compositorとは



- Visual Compositorとは、Unity Anime Toolboxに追加された新しい機能のことです。主に、Unity上の3Dシーンセットアップを破壊することなく、2D画像レイヤー構成と各レイヤーへの編集機能を提供するものです。
Visual Compositorを利用することで、アニメ制作者は3Dモデルの撮影情報を、都度出力バッファ画像にレンダリングしてAfterEffects上で合成する必要がなくなります。その結果として、3Dシーンセットアップをリアルタイムに修正しながら、同時に2D画像イメージとしての完成型を確認しながら、作業を進めることができます。
- Visual Compositorは、従来よりあるUnityのビルトインレンダリングパイプラインに対応するだけでなく、最新のレンダリングパイプラインであるHDRPにも対応しています。本ドキュメントでは、HDRPで使用する例を解説します。
- Visual Compositorで作成した映像は、Unity Recorderを使用して、連番画像やApple ProResなどの形式で出力が可能です。出力する映像は、完成したビューティ画像だけでなく、Pencil+ラインのみの画像や、様々なマスク画像も出力することができますので、Visual Compositorで処理した映像を、従来のツール上でさらに編集、画像加工することも可能です。

Visual Compositorを使用するメリット

Visual Compositorを使って、セルルック3DCGアニメーション制作過程での「編集撮影」作業を行うメリットとして、次のようなものがあげられます。

1. レンダリング対象ごとに、ライトとカメラのセッティングを分けることができます。従来、Unityの32枚しかないLayerや、HDRPに搭載された8つのLight Layersをやりくりして設定していた、ライトリンクやカメラのカーリング設定から解放されます。
2. 特にHDRP環境のように、撮影するオブジェクトのマテリアル毎に、大きくライトのインтенシティを変えたいような場合、つまり、現実に近いリアルな明るさを持った(10,000ルクスなど、インтенシティが大変高い)ライト環境で撮影したい背景オブジェクトと、セルルックキャラクターのようにしっかりとしたカラーを出す(従来の0~1のインтенシティの範囲内に収まる)ライト環境で撮影したいキャラクターオブジェクトを、自由に組み合わせることができます。特に、トーンシェーディングやセルシェーディングの場合、カラー管理が容易になります。
3. 各々のレンダリング結果に、リアルタイムでマスクを作りながら、Visual Compositorのレイヤー機能を使用して合成することで、従来3Dモデルのままでは難しかった2D的な合成表現を簡単に設計することができます。これらには半透明などの自由な取り扱いも含まれます。
4. 作業結果を連番画像で後工程(ポストプロセス)に引き渡す場合でも、ほとんどの絵作りをVisual Compositor上で完結することができるので、AfterEffectsなどの外部編集アプリに引き渡す素材の数や種類が激減します。



一番左の図は、Visual Compositorを使用した画面です。セルルックのボールと背景がよく馴染んでいます。セルシェーダーにはUTS/HDRPを使用しています。

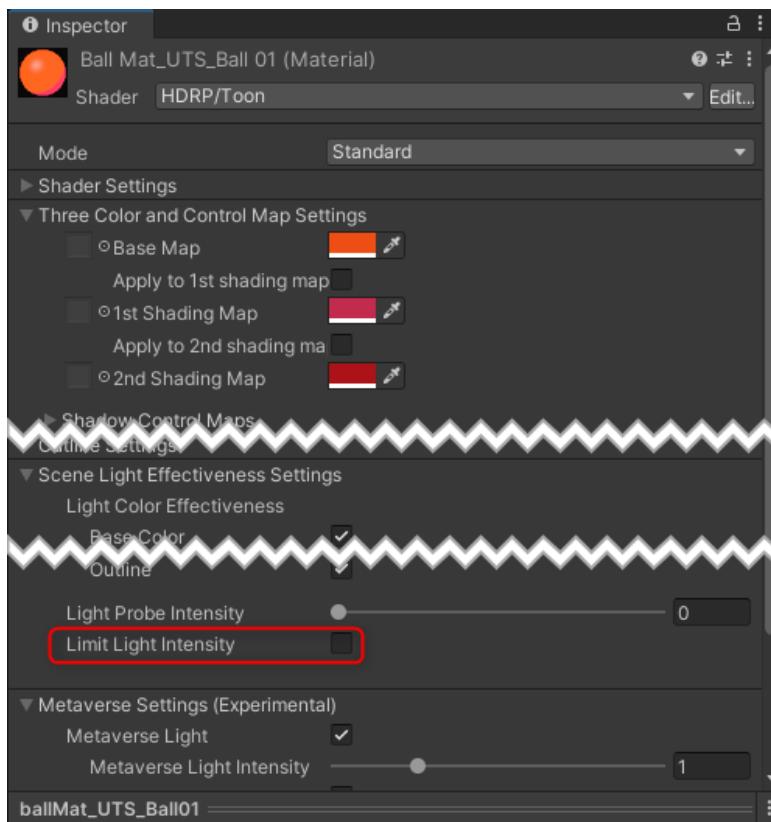
真ん中の図は、同じシーンを背景をレンダリングするHDRPのライトで照らした結果です。セルシェーダーのSDRでのカラー設計(メインライトのインтенシティが、0~1程度の範囲のもの)は、そのままではHDRPが得意としているHDRライティングの環境下(シーン全体の明るさが、10,000ルクス程度ある、現実環境での明るさと同様のもの)では、多くの場合白飛びします。

一番右の図は、真ん中の図とまったく同じ環境ですが、手前の赤いボールだけカラーが出ています。これはUTS/HDRPには、**Limit Light Intensity**(旧機能名: **SceneLights Hi-Cut Filter**)という機能があるので、赤いボールのマテリアルのみ、そのフィルタをアクティブにしているからです。UTS/HDRPのような特別なシェーダーを使うことで、HDR環境にも対応はできますが、そもそもVisual Compositorを使ってしまえば、オブジェクトを照らすライトの明るさ自体をそれぞれ変

えてしまっても問題なくなりますので、従来通り、オブジェクト単位でカラー管理をすることができるようになります。

Notes: UTS/HDRPを使う場合に、Limit Light Intensityを常に有効にしておくべきかどうかは、そのシーンでのカラー設計をどれだけ厳密にしたいかに依存します。

Limit Light Intensityは、シーン全体の明るさを見て、トーンシェーダーで設定したカラーが白飛びしないように一定の範囲内でカラーを出すように、マテリアル単位で自動調整をする機能です。代わりに常にトーンマッピングを機能させているようなものですから、カラーチャート上での数値の正確さよりは、見た目重視のカラー出力になります。この場合、シーンビュー中でもそれなりにカラーが出ますので、シーンビューでの作業はよりしやすくなるでしょう。



一方、Visual Compositorを使いながら厳密に各オブジェクト毎のライト管理をする場合には、Limit Light Intensityをオフにして、シーン毎にカラー設計をしたほうが、カラーチャート上での数値通りの正確な結果が得られます。代わりに、全てのライトが同時に有効になっているシーンビュー中ではトーンシェーダーのカラーは常に白飛びしている状態になります。(もちろんこの場合でも、シーン全体の明るさを1ルクス以下に管理していれば、白飛びは発生しません)

HDRPでの最初のセットアップ

HDRPでVisual Compositorを使用するにあたって、事前にプロジェクトに必要なセットアップを行います。合わせて、ビルトインパイプラインとの違いについて、簡単に触れておきます。

Project Settingsの設定

メニューバーより、Edit > Project Settingsと進みます。

Project Settingsウィンドウが表示されたら、以下の項目を設定します。

Quality > HDRPの設定

HDRenderingPipelineAssetを以下のように設定します。

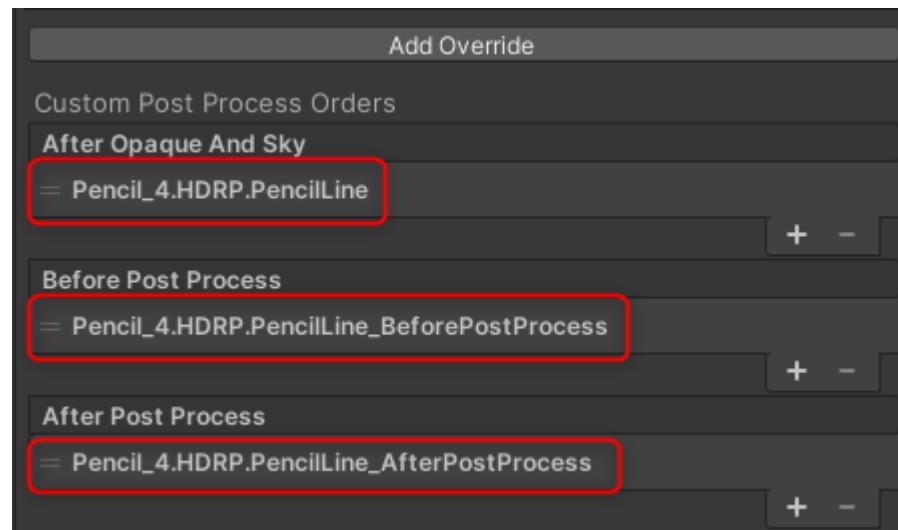
- **Rendering > Color Buffer Format**
「R16G16B16A16」に設定（必須）
- **Post-processing > Buffer Format**
「R16G16B16A16」に設定（Pencil+ 4 Line for Unityを使う場合必須）

HDRP Default Settingsの設定

HDRP Default Settingsでは、HDRPに共通の様々なセッティングを行います。ここで調整する必要があるのは、カメラの基本的な設定と、ポストプロセスの基本的な設定、Pencil+ Line for UnityのHDRP版を使う場合には、そのモジュールのポストプロセスへの追加です。

HDRPは、ビルトインパイプラインと違い、ポストプロセスが最初からシーン内のカメラにセットされています。そのデフォルトのセッティングを決めるのが、こちらです。もしデフォルトのポストプロセスの設定で問題がある場合には、こちらから調整が可能です。

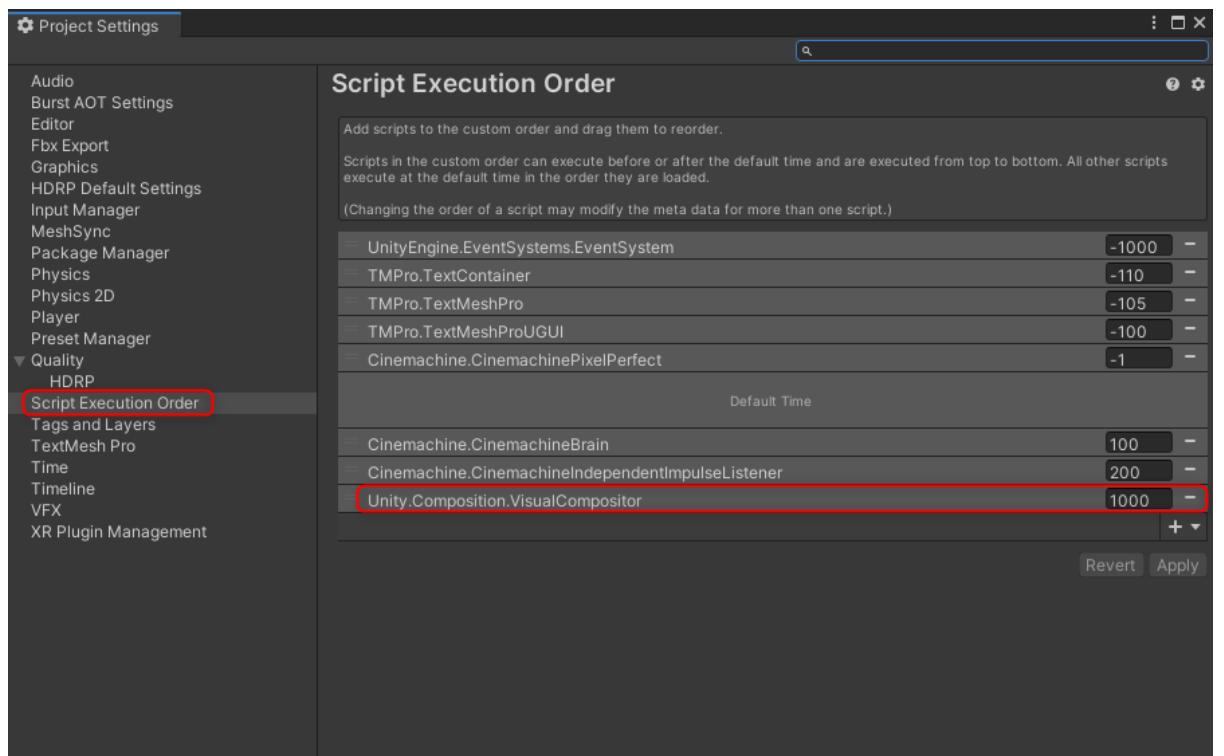
- **Frame Settings > Camera > Lighting**
「Fog」のチェックボックス（マスクノードの挙動に影響を与えます）
 - チェックの場合⇒各カメラでFogが有効になります。
マスク用カメラでは、個別にカメラ内でCustom Frame SettingsでFogのチェックを外します。
 - チェックオフの場合⇒全てのカメラでFogが無効になります。
マスク用のカメラでもFogが無効になります。（推奨）
- **Custom Post Process Orders**
HDRP版のPencil+ 4 Line for Unityのモジュールを追加します。



Pencil+ 4 Line for Unityの詳しい使い方は、製品のヘルプファイルを参照してください。

【NEW】Script Execution Orderの確認(Visual Compositor)

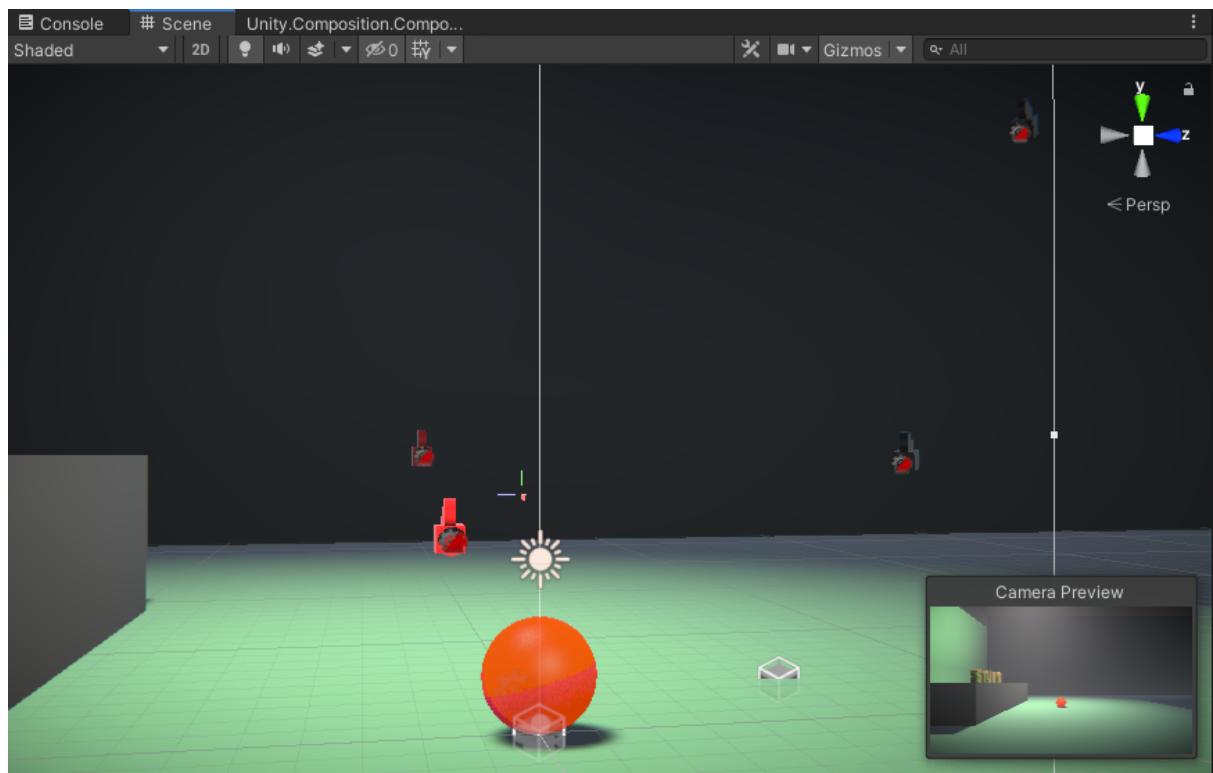
HDRPに限らず、Visual Compositorパッケージを新規にプロジェクトに追加したり、他のコンポーネントパッケージをプロジェクトに追加した際に、Visual Compositorでのフレーム更新がズレている、例えば、「Timelineの再生ヘッドのスクラブ中に1フレーム目の更新が遅れる」ように感じる場合には、Project Settings ウィンドウより、Script Execution Orderの項目を開いて、以下の部分を確認してみてください。



図のように、**Unity.Composition.VisualCompositor**は、**Script Execution Order**のリスト最下部にあることが望ましいです。これはアップデートサイクルで、Visual Compositorが真っ先に更新されることを意味しています。少なくともリストにおいて、Cinemachine関連のコンポーネントよりも下にあることを推奨いたします。

カリングマスクの設定(オプション)

例えば、カメラリグ内で設定した様々なカメラに、カメラ型のアイコンを付けておくと、シーン内のカメラ配置が一望できて便利です。しかしこれらのカメラアイコンは、ゲームビューでは表示させたくないかもしれません(レンダリング中に映り込んでしまうと困るから)....。

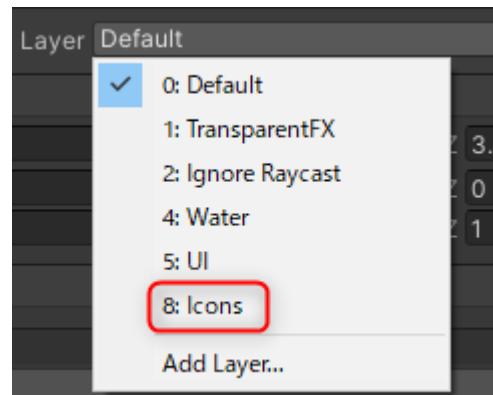


そんな時に、特定のオブジェクトをゲームビューのカメラに表示させない方法があります。それが各カメラコンポーネントに設定できる、カリングマスク(**Culling Mask**)という機能です。

カリングマスクの設定は、以下のように行います。

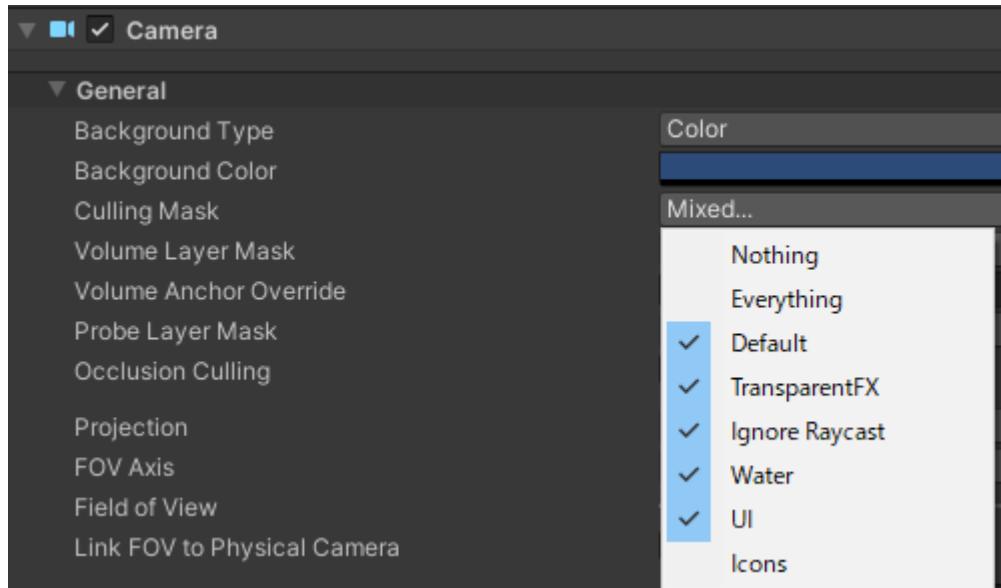
Layerを設定する

Inspectorウィンドウの上部にある、Layerドロップダウンから、Add Layerボタンを押し、空いているUser Layerのスロットに新規レイヤー名を追加します。ここでは「Icons」とします。



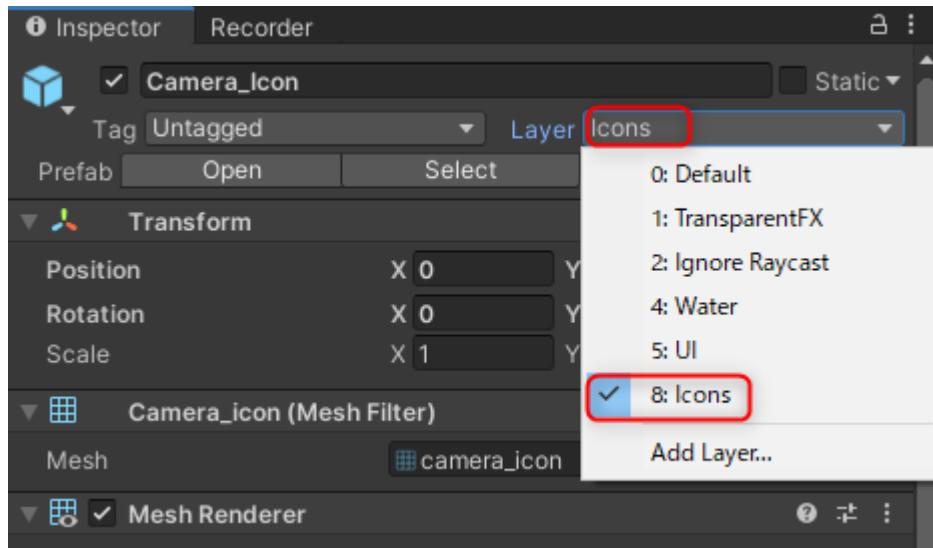
Camera > Culling Maskを設定する

CameraのCulling Maskドロップダウンを開き、Iconsをチェックアウトします。
下のような画面になります。



シーンビューでのみ表示したいオブジェクトのLayerをIconsにする

このように設定すると、これ以降、Camera_Iconはゲームビューでは表示されなくなります。



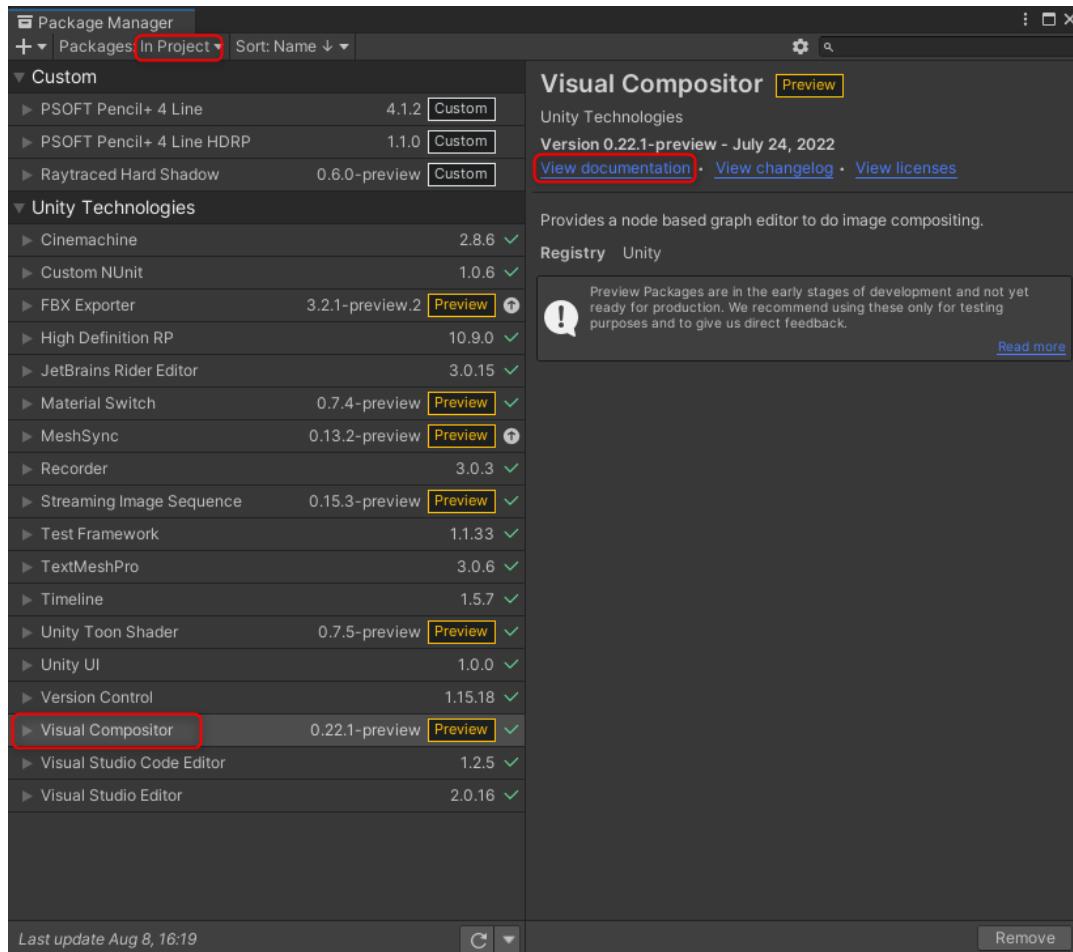
Visual Compositorの準備

以上の準備が出来たら、Visual Compositorを設定していきましょう。

ドキュメントの参照

Visual Compositorの詳しい機能を説明したドキュメントは、以下の手順で確認できます。
メニューバーより、Window > Package Managerと進み、Package Managerウィンドウが開いたら、In Projectのドロップダウンが選ばれているか確認します。
現在のプロジェクトにインポートされている全てのパッケージが表示されていますので、その中から「Visual Compositor」を探してください。

Visual Compositorを選択すると、左側のウィンドウ内に「Links」という項目があります。
View Documentationのリンクをクリックすると、ドキュメントが開きます。こちらを一読することをお勧めします。

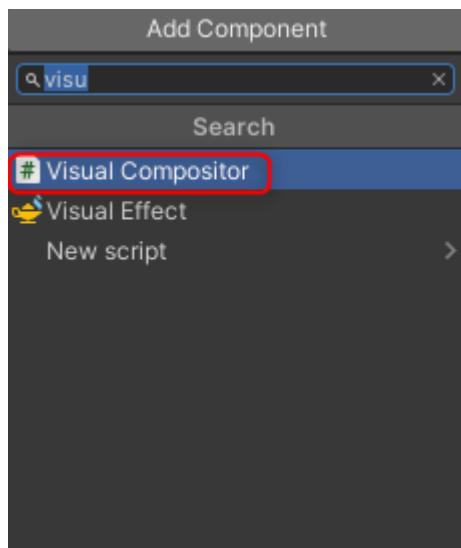


以下では、本プロジェクトで設定している機能を中心に、Visual Compositorでの作業の始め方を説明します。

シーン内にCompositorゲームオブジェクトを作成する

Visual Compositorをシーン内で利用するために、シーン内にCompositorゲームオブジェクトを作成します。以下の手順で行います。

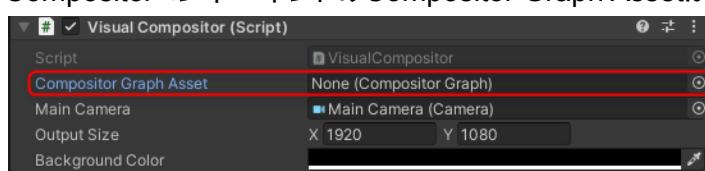
1. ヒエラルキーインディウム上で、「+」ボタンから、空のゲームオブジェクトを作成します(Create Empty)
2. 作成した空のゲームオブジェクト「GameObject」を、「Compositor」にリネームします。
3. CompositorゲームオブジェクトのインスペクターウィンドウでAdd Componentボタンから、Visual Compositorをアタッチします(検索窓でvisuと打つと簡単に見つけられます)。



4. Compositorゲームオブジェクトに、以下の2つのコンポーネントが追加されたのを確認します。
 - Camera
 - Visual Compositor

以上で、Visual Compositorが使えるようになりました。

5. 【NEW】新規にVisual Compositorオブジェクトをシーンに追加した場合、Visual CompositorコンポーネントのCompositor Graph Assetが指定されていません。

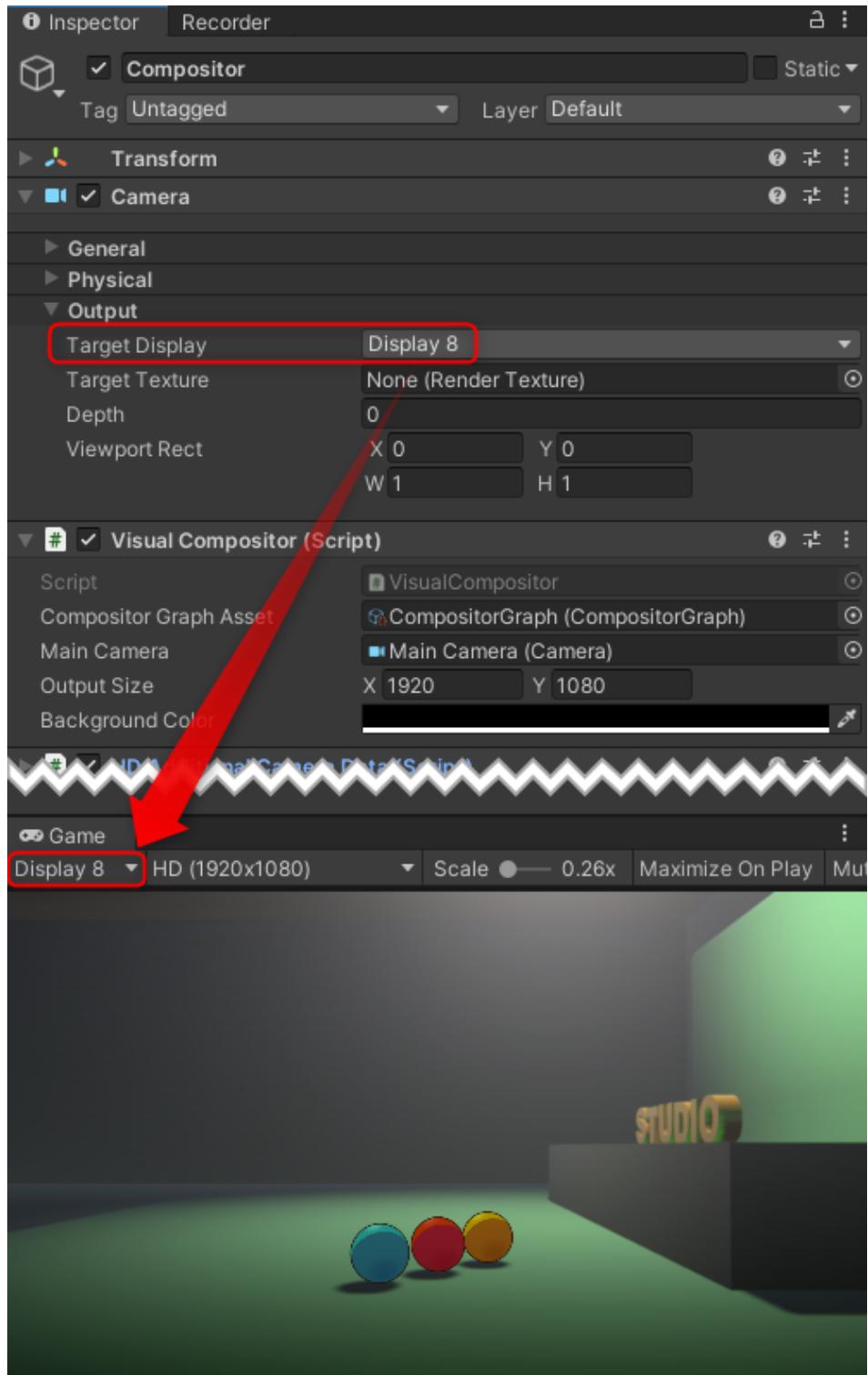


その場合、

- プロジェクトウインドウの「+」ボタンより、新規に **VisualCompositor>Compositor Graph**を作成し追加するか、
- 既存のCompositor Graphがある場合には、それを割り当てます。

※重要 Visual Compositorの合成結果確認の設定について

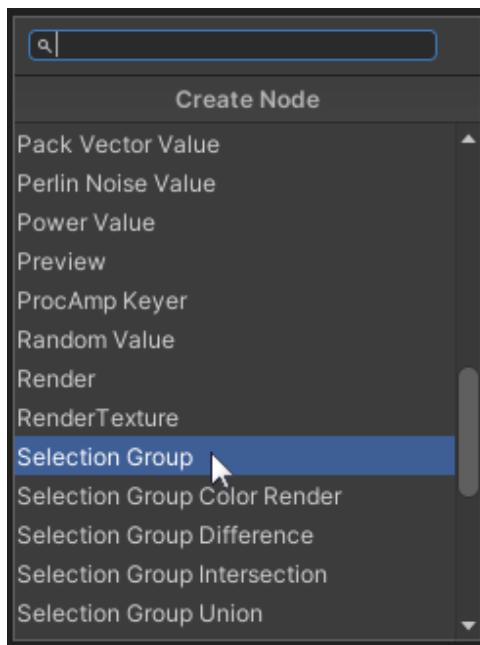
後に詳しい説明をしますが、Compositorゲームオブジェクトにアタッチされている Cameraコンポーネントをインスペクター上で確認し、Output以下にあるTarget Displayを Display 8 にします。この設定以降、ゲームビュー表示を「Display 8」にすることで、Visual Compositorの合成結果が確認できるようになります。



Visual Compositorウィンドウを開く

実際に作業を行うVisual Compositorウィンドウを以下の手順で開きます。

1. メニューバーより、Window > Rendering > Visual Compositor と辿って、Visual Compositor ウィンドウを開きます。
 2. Visual Compositor ウィンドウを、Unity の作業しやすい場所にドッキングします。
 3. Visual Compositor ウィンドウ内では、ノードベースで様々な編集合成作業ができます。
- ノードは、以下の2つの方法で呼び出すことができます。
- Visual Compositor ウィンドウ内で右クリックして、「Create Node」を選ぶ。
 - マウスカーソルをVisual Compositor ウィンドウ上において、キーボードよりスペースバーを押す。



続いて、シーン中の様々な要素を選択して、Visual Compositor ウィンドウ内で利用可能にするSelection Groupsについて説明します。

Selection Groupsでシーン内の様々な要素を選択する

Selection Groupsとは

- Visual Compositor ウィンドウ上で編集撮影工程を進めるにあたり、「どの対象物」に対して、「どのような効果」を加えるかを決める必要があります。この「どの対象物」かを、シーン内に存在する沢山のゲームオブジェクトから選択して、グループとしてまとめておく機能が、Unity Anime Toolboxに新規に追加されたSelection Groupsという機能です。本機能は、Mayaの機能で例えると、「選択セット」に当たります。使い方も選択セットによく似ています。
- Selection Groupsは、ユーザーインターフェースである「Selection Groups ウィンドウ」、ヒエラルキー上でゲームオブジェクトとして表示される「Selection Groups ゲームオブジェク

ト」、最後にそれらをアセットとしてプロジェクトフォルダ内に保存している「SelectionGroupsアセット」の3要素で動作します。Selection Groupsは、プロジェクト単位で保存されるので、同じプロジェクトに属する全てのシーンから同じものが参照されます（ここはMayaの選択セットと違う部分ですので、注意が必要です）。

- Selection Groupsの操作は、Selection Groupsウィンドウから行います。各グループへの追加や削除も全て同じウィンドウ内から行います。詳しくは、Package ManagerウィンドウのSelection Groupsの項目から、ドキュメントを参照してください。

新規にSelection Groupsを設定する

新規にSelection Groupsをプロジェクトに設定したい場合、以下の手順で行います。

1. メニューバーより、Window > General > Selection Groupsと辿って、Selection Groupsウィンドウを開きます。
2. Selection Groupsウィンドウが開いたら、Unityの使いやすい場所にドックしておき、ウィンドウ内にグループが作成されていない場合には、「+」ドロップダウン  を押し、「Create Empty Group」を選択します。

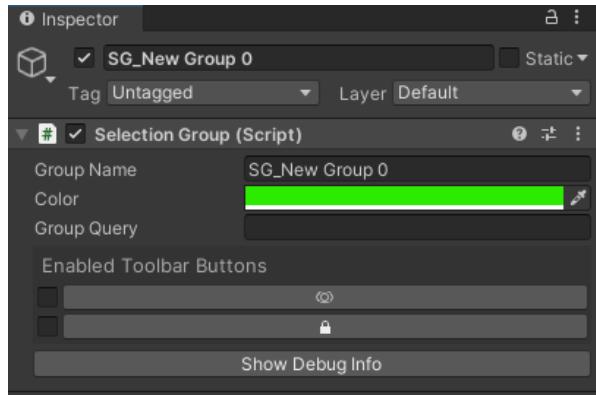


すでにSelection Groupsがプロジェクト内の別のシーンで定義されている場合は、Selection Groupsウィンドウ内はそれらも含まれた状態で開きます。各グループに含まれるゲームオブジェクトは、シーン単位でクエリーされますので、シーン内に該当する対象が存在する場合は、それらが登録されます。

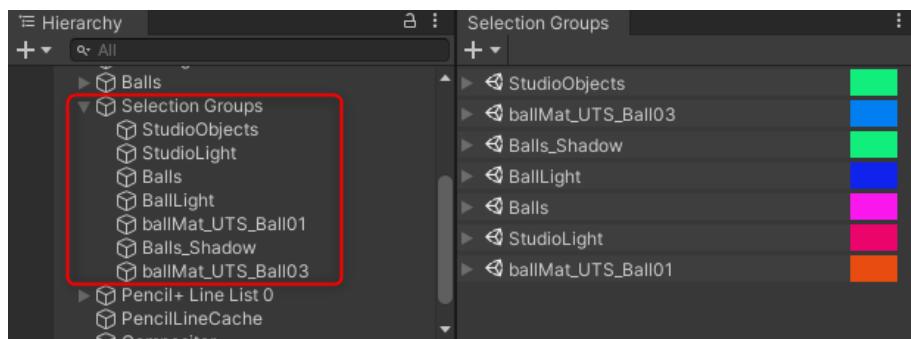
Tips: Selection Groupsはヒエラルキーと一緒に使うことが多いので、ヒエラルキーインデントの横にドッキングすると使い勝手があがります。

3. 新規にEmpty Groupを作成した場合には、Selection Groupsウィンドウより、SG_New Group 0をクリックすると、Inspectorウィンドウに、選択したSelection Groupのプロパティが表示されます。ここから、このセレクショングループに収集し

たいシーン中のゲームオブジェクトが定義できます。



4. ここで、「SG_New Group 0」の名前だけ変更し、リターンキーを押すと、該当するセレクショングループの名前が、Selection Groupsウィンドウ上でも、ヒエラルキー上でも連動して変わります。
5. ヒエラルキー上のセレクショングループは、ヒエラルキー上で他のゲームオブジェクト以下の階層にまとめておくこともできます。(下図の場合なら、Selection Groups以下にドラッグ & ドロップでまとめてみました。)



各Selection Groupの設定の仕方

各Selection Groupへゲームオブジェクトを登録するには

あるSelection Groupへゲームオブジェクトを登録するには、いくつか方法があります。簡単なものから紹介します。

- すでにSelection Groupsウィンドウの中にあるグループに、ヒエラルキー上で選択したゲームオブジェクトをD&Dして追加する
- Add Groupsボタンから新規にセレクショングループを作成し、そこに上と同様にヒエラルキー上で選択したゲームオブジェクトをD&Dして追加する
⇒これらが一番簡単ですが、参照先のゲームオブジェクトに手を加えたりすると、参照が切れてしまう場合も有り得るので注意しましょう。特に名前の書き換えに要注意です。またこの直接選択をする方法だと、シーンが変わると同じ名前のゲームオブジェクトでも抽出されなくなります、

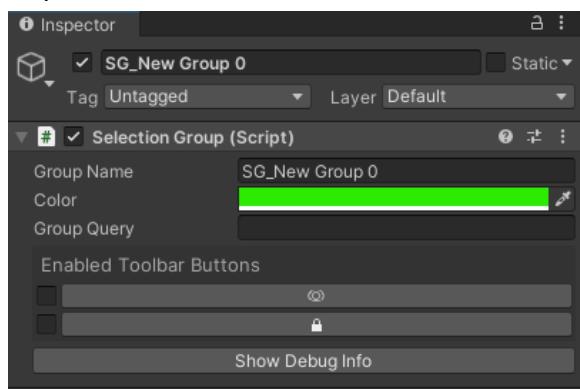
- クエリー式を利用してシーンから該当するゲームオブジェクトを抽出する(推奨)
⇒クエリー式を書く必要はありますが、汎用性もあり、別のシーンでも条件さえ合えば、そのままグループが使えるという点で、お薦めする方法です。

クエリー式の書き方

Selection Groupsで用いるクエリー式は、GoQLという独自のものですが、非常に簡単な書式となっています。詳しくは、Package Managerウィンドウより、GoQLの項目を選択し、Linksよりドキュメントを参照してください。

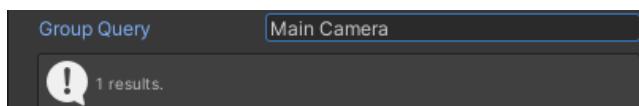
クエリー式を書く場合、以下の手順で行います。

- Selection Groupsウィンドウより、クエリー式を加えたいグループを選択し、Inspectorウィンドウにプロパティを表示させます。



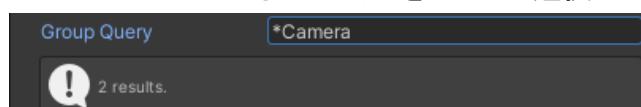
- Inspectorウィンドウ内の、Group Queryフィールド内にクエリー式を書きます。

- シーン内のMain Cameraを選択したい



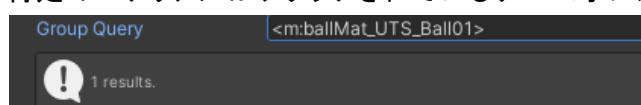
⇒名前をそのまま記入します。

- シーン内の**Main Camera**や**Chara Camera**など、名前の後半が**Camera**とついているゲームオブジェクトをまとめて選択したい



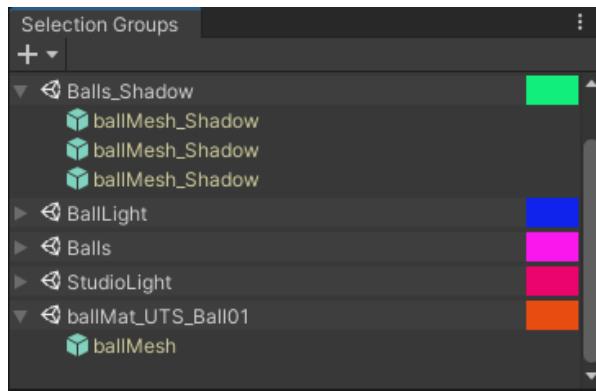
⇒ワイルドカードを使った名前選択ができます。

- 特定のマテリアルがアタッチされているゲームオブジェクトを選択したい



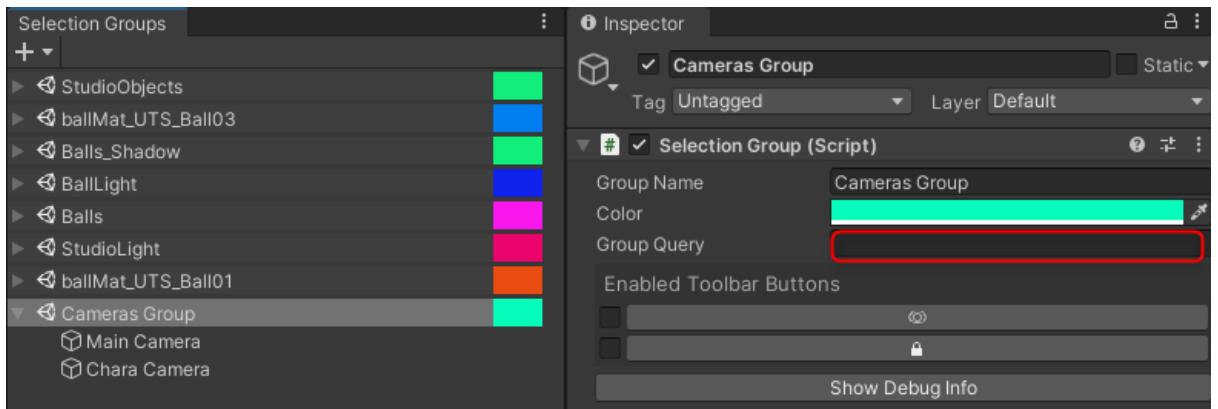
⇒タイプ指定をすることで、特定のマテリアルがアタッチされているメッシュオブジェクトが選択できます。

3. クエリー式の結果は、Selection Groupsウィンドウ内に反映されていますので、確認しておきましょう。

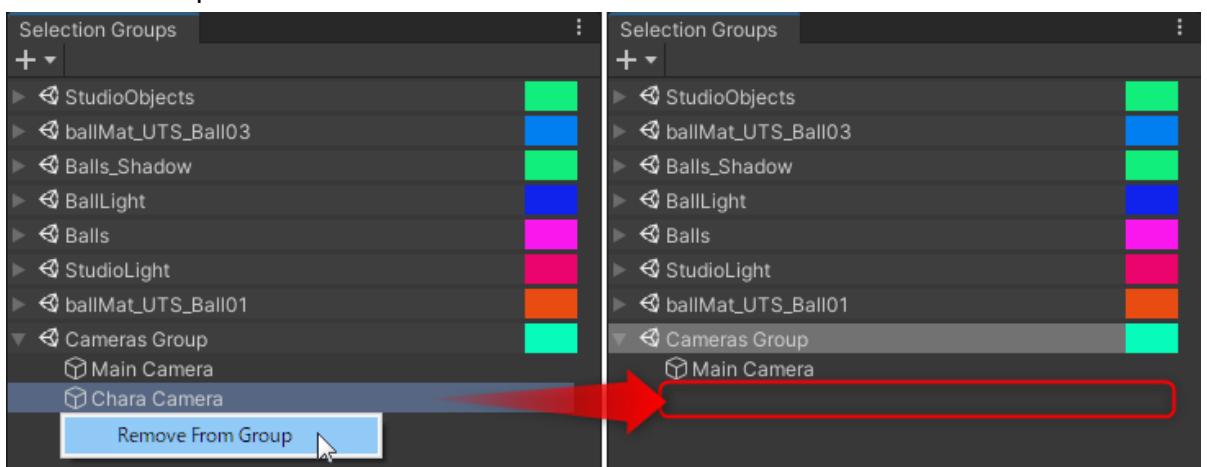


各Selection Groupから登録したゲームオブジェクトを除外する方法

1. Selection Groupsウィンドウから、修正したいSelection Groupを選択し、Inspectorウィンドウ内のGroup Queryフィールドからクエリー式を削除します。



2. 除外したい要素を選択し、右クリックします。「Remove From Group」を選択すると、Selection Groupから削除されます。



Tips: この時、元がクエリー式を指定していても、特定したゲームオブジェクトの削除後はそのクエリー式は無効になり、以降は残るゲームオブジェクトを直接選択した結果と同じになりますので、注意してください。

Visual Compositorで絵作りをする

Visual Compositorは、ノードベースの合成アプリケーションです。

機能が割り振られたさまざまなノードを線でつなぎ、流れを追って絵を完成します。

ノードは、ノード一つに一つの機能のみです。

そのため、簡単なレイヤーを重ねる、という場合でも、

- ・素材のノードA

- ・素材のノードB

- ・レイヤーノード(またはブレンドノード)

という3つのノードを組み合わせます。

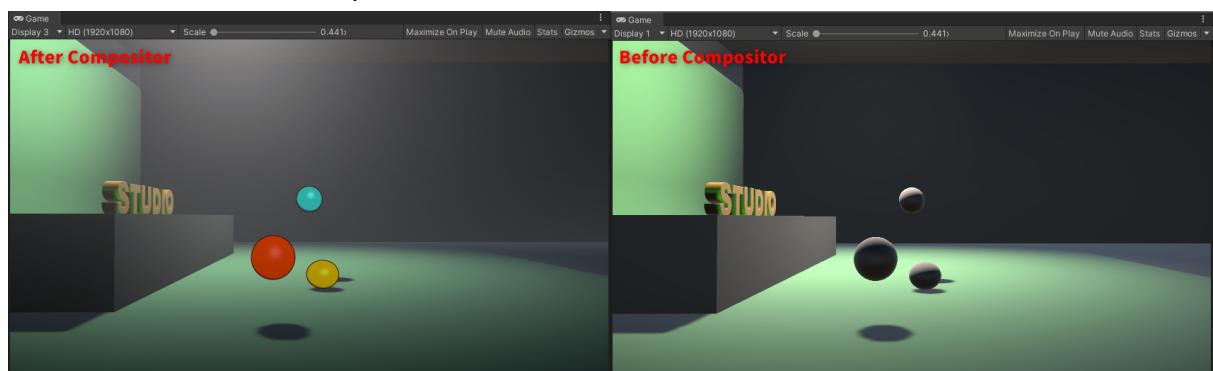
レイヤーベースと違って最初は手間がかかるように思えますが、慣れるにつれて、いつ立ち返っても判読性が高くて何をしているのかを理解しやすい、流れるように思考をとどめているノードの組み合わせが、複雑な合成作業も容易にしてくれることを実感できるでしょう。

流れるようにゴールに向かっていく様子を指してノードの組み合わせを「フロー」と呼びます。

ここからは、Visual Compositorウィンドウを使った「絵作り」について解説します。

Layerノードスタックの構成

まずは同じシーンで、Visual Compositorの使用後の画面と、使用前の画面を見てみましょう。下の図は、左がCompositor使用後のシーン、右が使用前のシーンです。



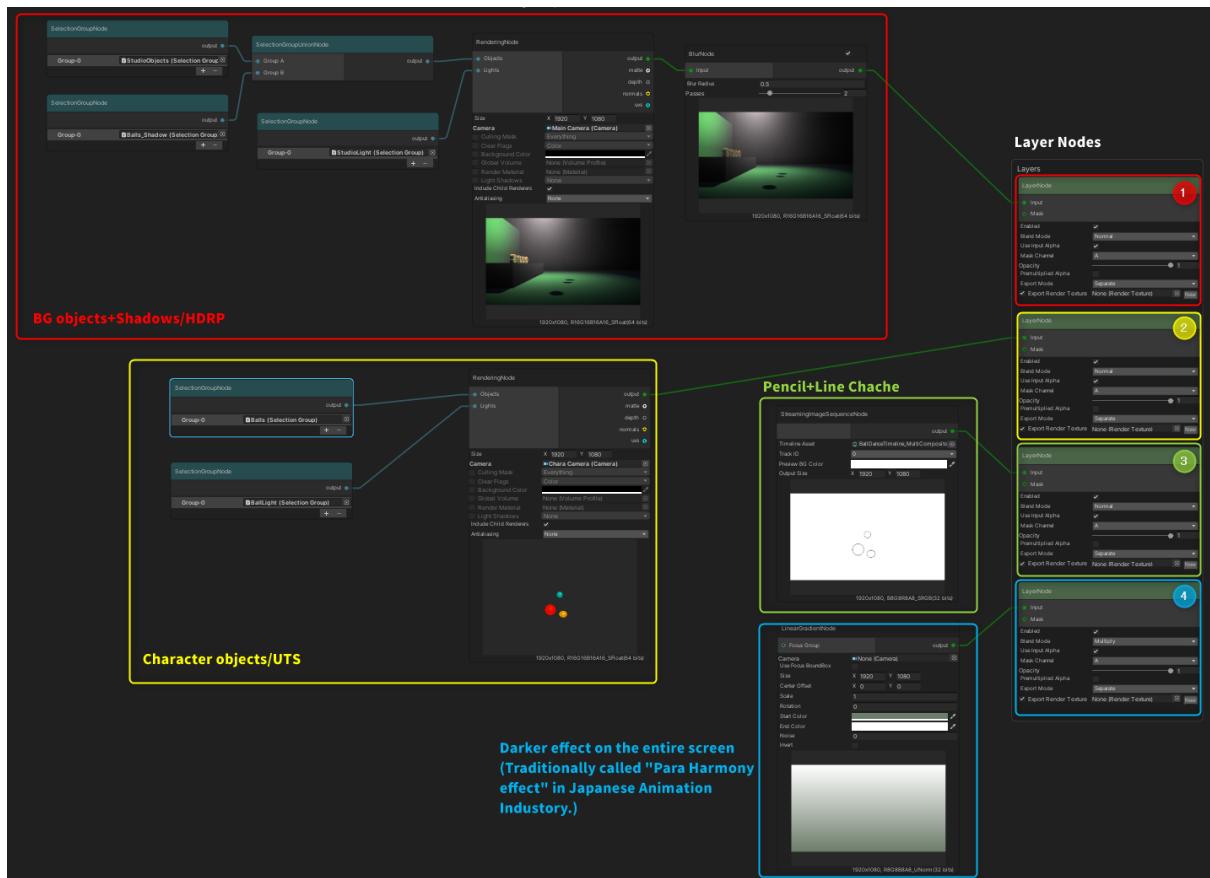
上のシーンを完成するために、Visual Compositorを使って以下のような作業を行っています。

1. BG要素: HDRP用のボリュメトリックライトを使った、背景オブジェクトのレンダリングとオブジェクトからの落ち影を作成。
2. セルルックキャラクター要素: ディレクショナルライトで照らした、UTS/HDRPマテリアルを適用したボールのレンダリング。
3. トーンライン合成:Pencil+ Line Cacheと合成してボールにトーンラインを載せる。
4. 画面全体へのパラ効果(注:画面全体を、カラーを掛けながら暗くする効果を、日本のアニメ産業では「パラ(フィン)効果」と伝統的に呼びます): 全体を馴染ませるために、画面全体に2D的に載せる。

これらのレイヤー(階層)が、 $1 \Rightarrow 2 \Rightarrow 3 \Rightarrow 4$ と順番に重ね描きされていくことで、最終的な画面ができあがります。

各々のレイヤーは、リアルタイムで必要な画像が生成されますので、従来のように AfterEffectsのような外部コンポジターで再編集するために、3Dの画像バッファを前もって出力しておく必要はありません。

以上の構成をVisual Compositorウィンドウで見ていくと、次のように対応しています。

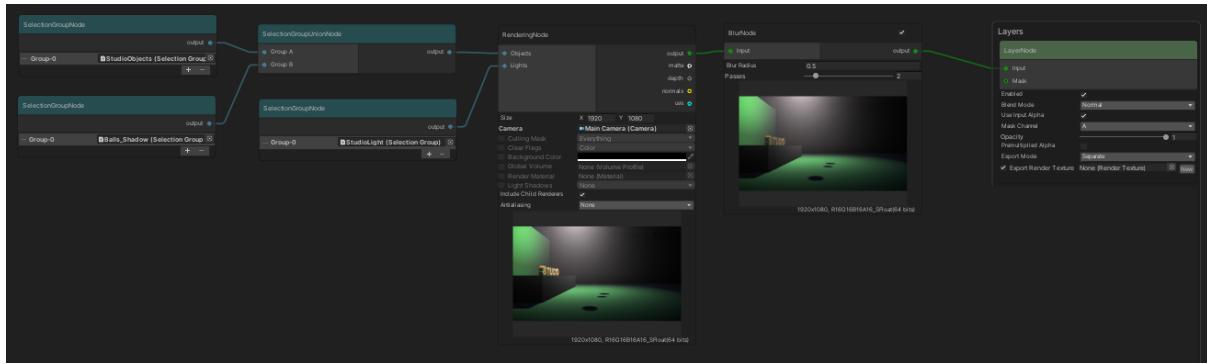


最終出力であるLayerノードのスタックは、一番下のノードが画面では一番上のレイヤーになります。水が上から下に流れていくように、1と2を足したものに3を足して、さらにその結果に4を足して、と計算処理されていきます。見た目で重ね順を操作できるPhotoshopとは考え方が違ってくることに注意してください。

続いて、各レイヤーに繋がっているノードを確認していきます。

各ノードの詳しい機能に関しては、Visual Compositorのドキュメントを参照してください。

第1レイヤー: 背景オブジェクトと影のレンダリング



まずは、基本となる第1レイヤーから見ていきましょう。

第1レイヤーは主に背景オブジェクトをレンダリングするレイヤーです。このレイヤーでは、背景にあたる一枚絵を作成していると考えるとよいでしょう。

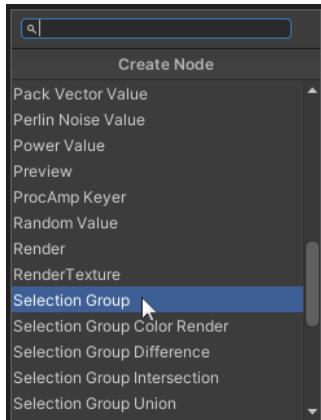
主にキャラクターをレンダリングする第2レイヤーとの大きな違いは、レンダリングに使用するライトの種類の違いです。

第1レイヤーのオブジェクトは、シーン全体の基本となるポイントライトで照らされています。ライトにはHDRPの「ボリューム効果」(大気の雰囲気を指します)を加えており、日本のアニメ風の、精緻な背景をレンダリングしています。また、同じライトが、キャラクターであるボールの落影も作ります。

新規ノードの追加

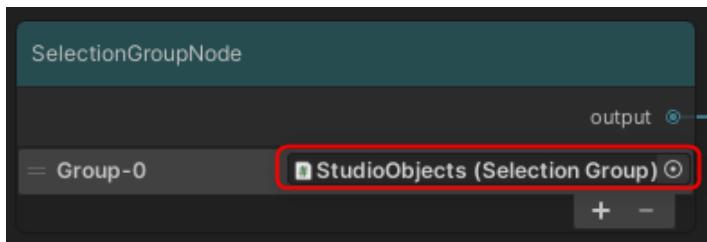
Visual Compositorウィンドウ上にマウスカーソルを置き、スペースバーを押す、もしくは右クリックからメニュー選択をすることで、Create Nodeウィンドウが開きます。

Create Nodeウィンドウから、Visual Compositorウィンドウに新規ノードを追加できます。

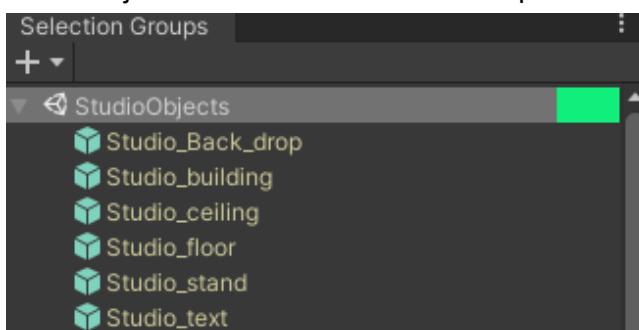


シーン内オブジェクトをVisual Compositorに入力する、Selection Groupノード

Create Nodeウィンドウから、Selection Groupノードを作成します。
Selection Groupノードは、Visual Compositorウィンドウから、Selection Groupsウィンドウ内のいずれかの要素を参照するためのノードです。
下の図の例だと、StudioObjectsというSelection Groupが参照されています。



StudioObjectsの内容は、Selection Groupsウィンドウから確認できます。

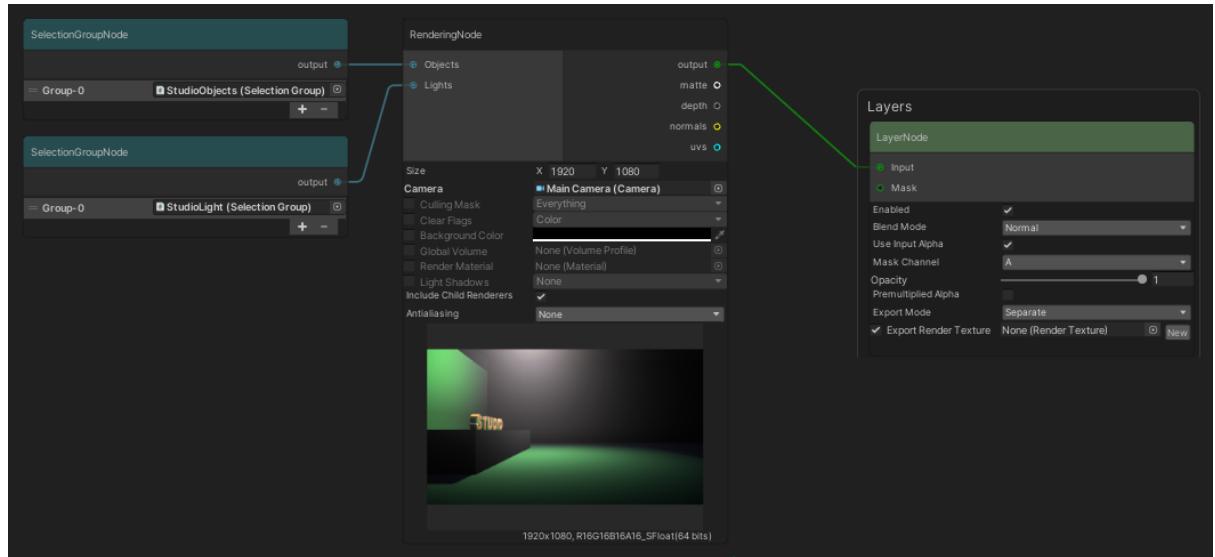


シーン内に存在するゲームオブジェクトは、Selection Groupノードを介して、Visual Compositorウィンドウ内の様々なノードに参照されることになります。

Tips: Selection Groupとしてクエリー検索がしやすいように、グループで扱う可能性が高いゲームオブジェクトには、ワイルドカード検索をしやすいような名前を付けたり、共通のマテリアルを設定したりの工夫をしておくとよいでしょう。

Tips: Unity上で名前を変更した場合は、一度シーンをセーブしてリロードすると、**Selection Groups**に反映されます。

Visual Compositorの結果をゲームビューに表示する、Layerノード



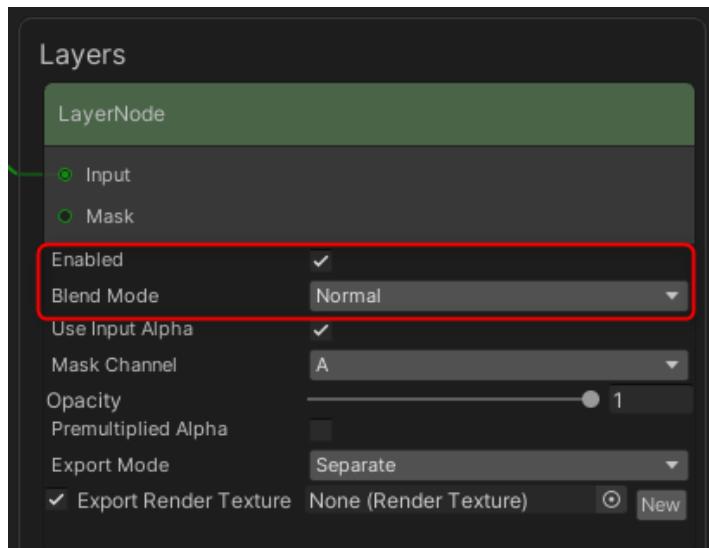
Visual Compositorウィンドウの作業結果をゲームビューに表示するためには、最終的に LayerノードのInputポートに接続する必要があります。

必要最低限のノードツリーは、上のようなSelection Groupノード>Renderingノード>Layerノードとなります。

この時、Layerノードの

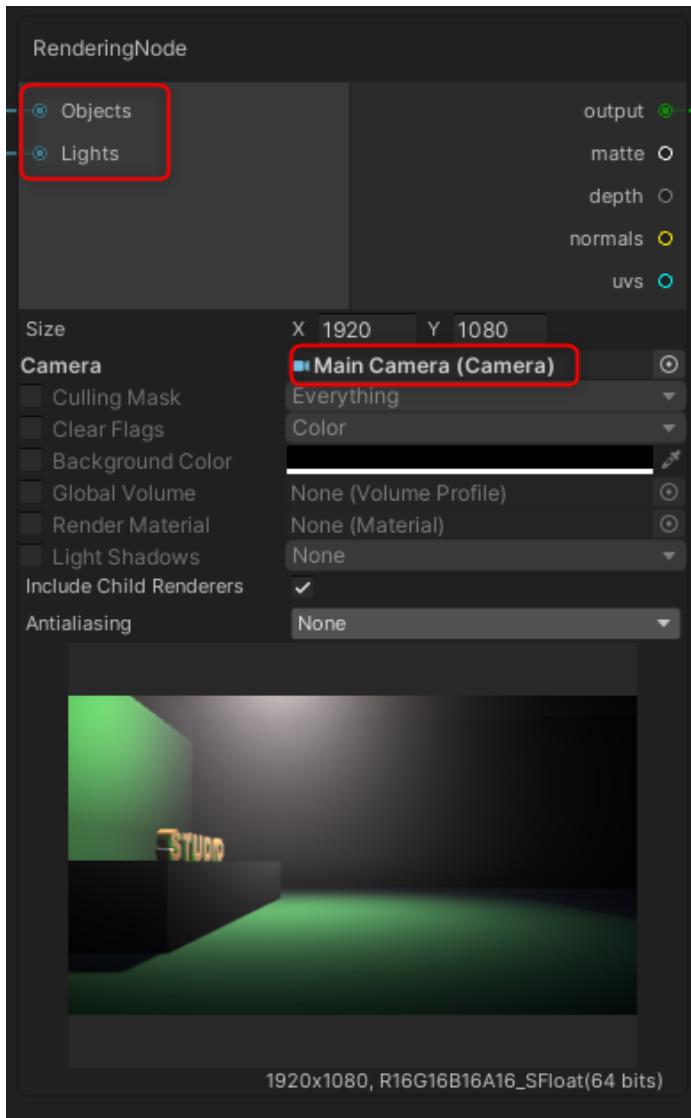
- Enabledをチェック
- Blend ModeをNormal
- 本プロジェクトの場合、ゲームビューをDisplay 8表示

にすることで、Visual Compositor上でレンダリング画像がゲームビューに表示されます。



Tips: Composerの結果をどのDisplayで表示するかは、Visual CompositorコンポーネントがアタッチされているCameraコンポーネントのTarget Displayで設定します。

レンダリング画像を生成する、Renderingノード



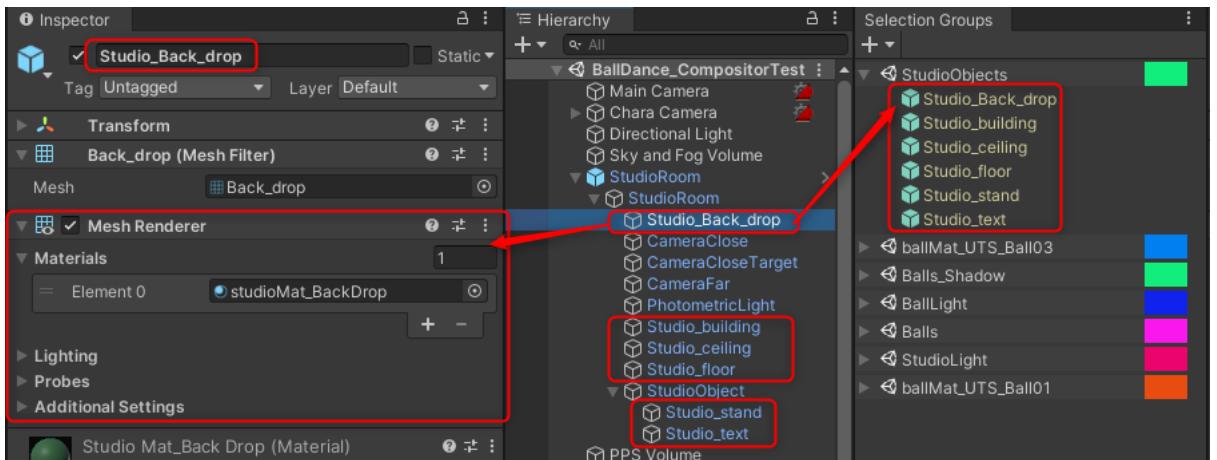
Renderingノードは、Inputポートに接続されたオブジェクトとライトのグループを、指定されたカメラでレンダリングするノードです。

各々のカメラにポストプロセスが指定されている場合には、それらも適用されます。
(オブジェクトとライトを接続しない場合、指定したカメラに写っているものすべてを描画します)。

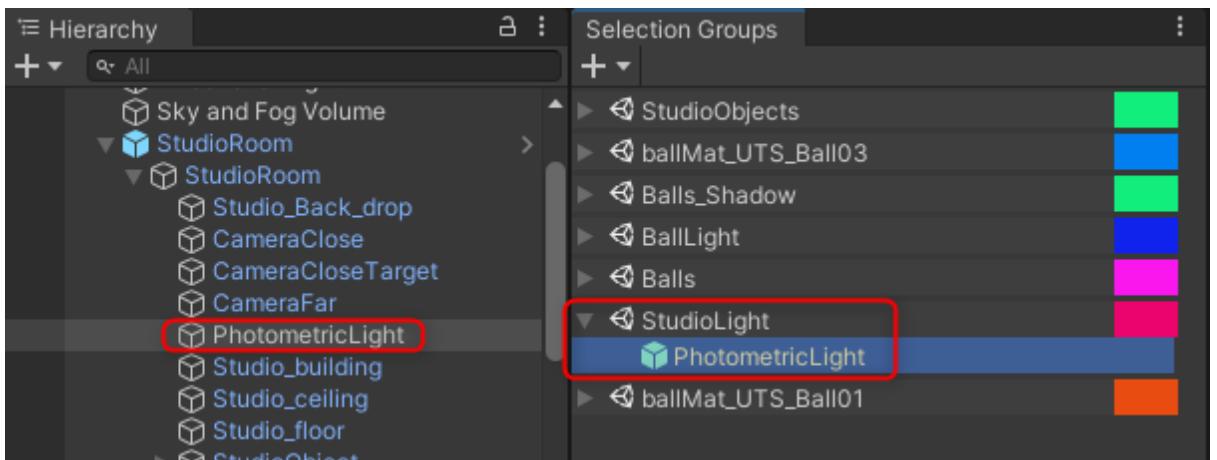
上の例では、レンダリングカメラとして、ヒエラルキーインドウのMain Cameraが使われます。Main Cameraには、CinemachineBrainコンポーネントがアタッチされていますので、TimelineのCinemachine Trackに応じて、画像がレンダリングされます。

レンダリング対象となるオブジェクトは、StudioObjectsセレクショングループで取得されるMesh Rendererを含むゲームオブジェクトが対象となります。[NEW] **Rendering**ノードで、「Include Child Renderers」がチェックされていると、セレクショングループが親階層のみを指定していても、その子階層に含まれるMesh Rendererを含む各ゲームオブジェ

クトもレンダリングされます。デフォルトはチェック状態ですので、そのまま利用するとよいでしょう。

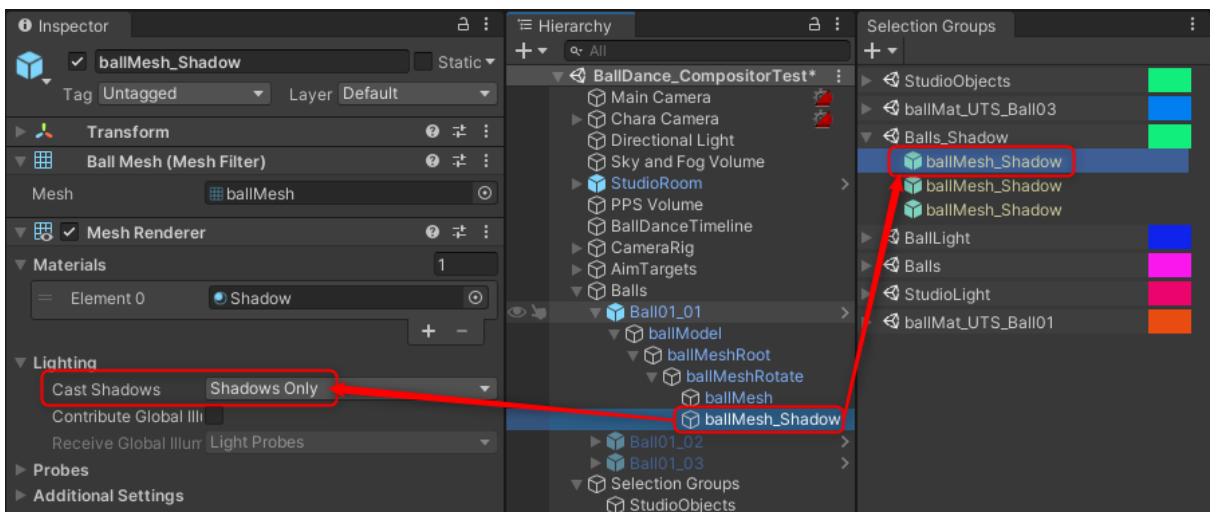


背景のレンダリングライトに使う、ボリュメトリックライトは、StudioRoom階層の中にある PhotometricLightを、StudioLightセレクショングループとして取得しています。
PhotometricLightは、6000ルクスのHDRP用ライトとなっています。



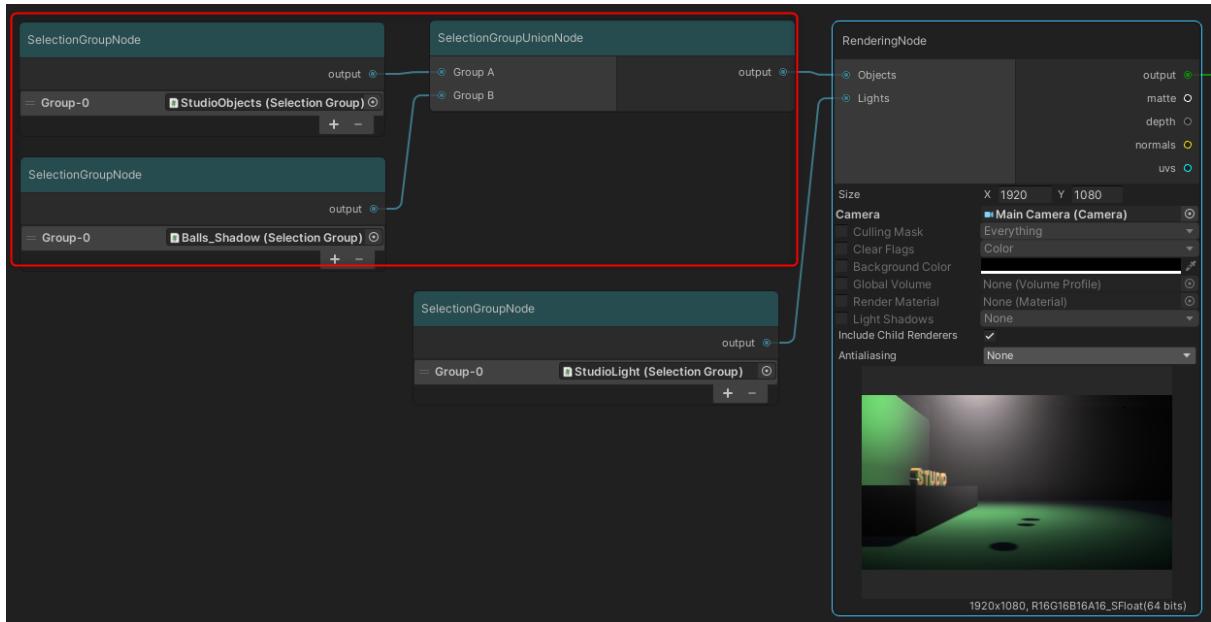
影モデルをレンダリングして、影を追加する

背景画像の上に、キャラクター(カラーボール)の影も追加します。

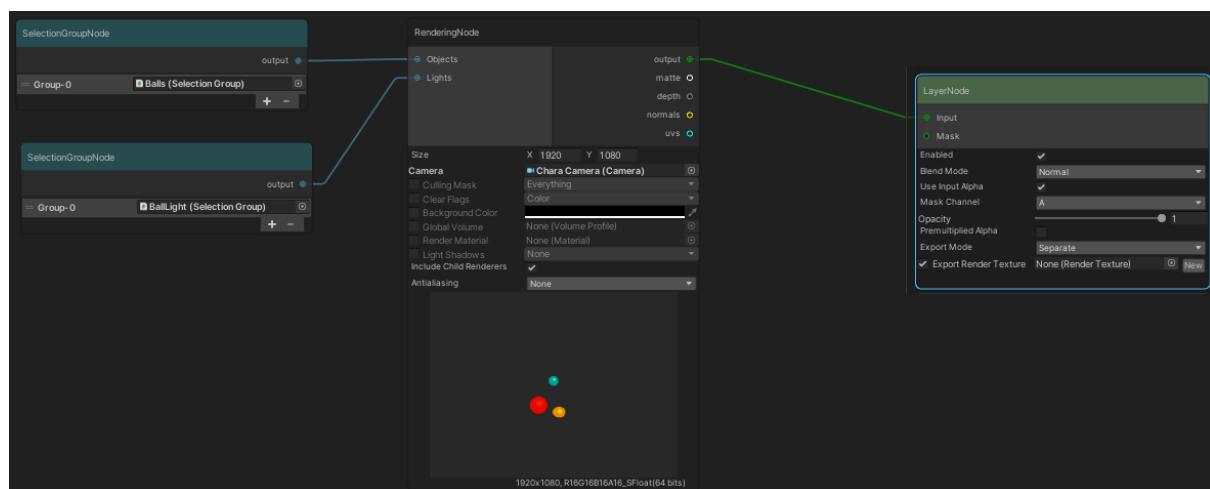


影モデルは、ボールのメッシュを含むBallMeshをデュプリケートし、ballMesh_Shadowとリネームした後で、Mesh RendererのLighting > Cast Shadowsを「Shadows Only」にします。こうすることで、影のみを落とすオブジェクトができます。

影オブジェクトのみを集めたBalls_Shadowセレクショングループを作成したら、UnionノードでStudioObjectsセレクショングループとまとめて、RenderingノードのObjectsポートに接続します。Unionノードはカメラから見えている前後関係通りにまとめられるため、groupA、groupBどちらに接続しても、重なり順が変わることはありません。重なり順を変えられるのはLayerノードでの操作のみです。



第2レイヤー：セルルックキャラクターのレンダリング



第2レイヤーでは、セルルックキャラクター(今回は3色のボール)のレンダリングを行います。セルルックキャラクターのルックデヴには、ディレクショナルライトが用いられます。ディレクショナルライトの方向で、通常色と影色の配置が決定します。セルルックキャラクターの場合、これらのカラーの塗り分けは、デザイン的な要素が加味されます。つまり、その塗り分けの方向や程度は、シーンの背景となるオブジェクト全体を照らすライトの方向と必ずしも一致してなくてもかまい

ません。むしろ各シーン中で、各キャラクターがどのように見えてほしいか、という見栄え優先で決定されるべきです。

またキャラクターのレンダリング画像は、背景画像の上に合成されますが、そのためにキャラクターの部分を切り抜くマスクが必要になります（自動的にアルファチャンネルは生成されません。一つ一つに的確に指示する必要があるのがノードベースワークフローの特徴で、それによって間違いが起きづらい堅牢な合成フローを構築できるのが利点です）。このレイヤーでは、それらのマスクの生成も行っています。

Layerノードを追加する

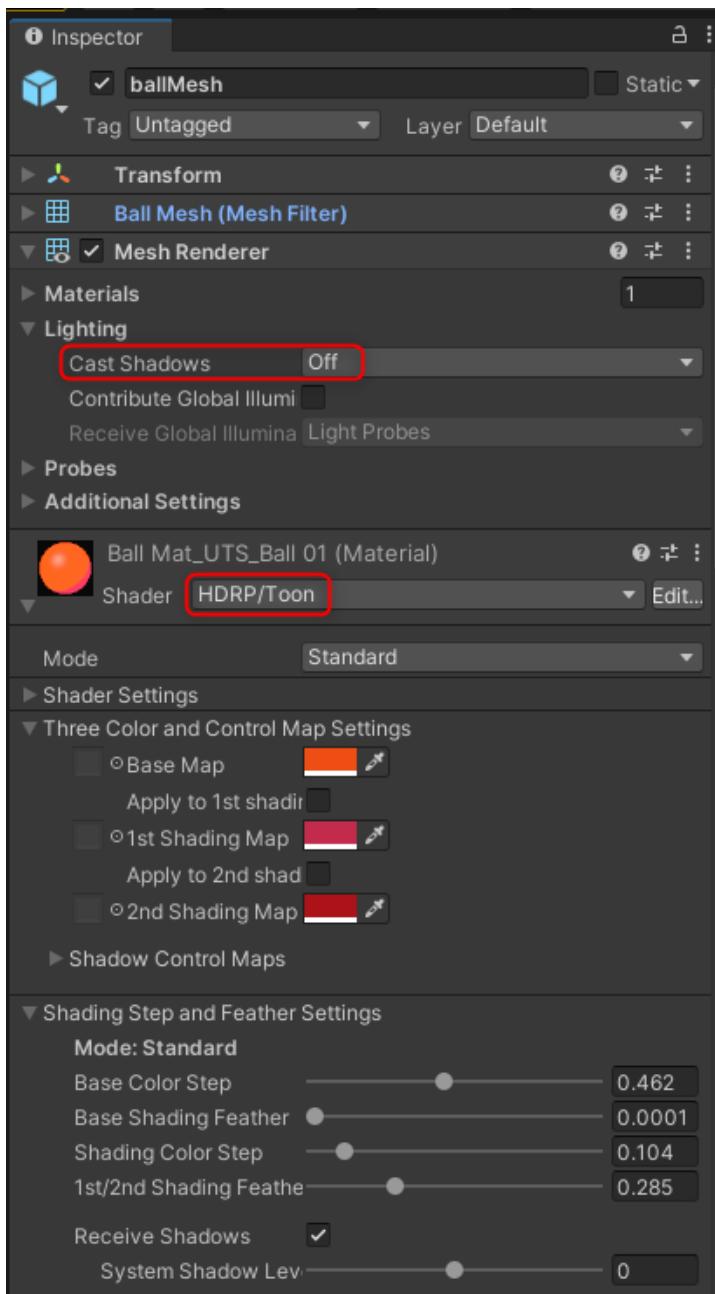
Create Nodeウィンドウから、Layerノードを作成し、Visual Compositorウィンドウ内で第1レイヤーとして使用しているLayerノードの下部にドラッグしていきます。すると、2つのLayerノードがひとつに合体してLayer Stackノードができます。これで、第1レイヤーの上に第2レイヤーが重なりました。

新規に追加したLayerノードを使うことで、背景オブジェクトのレンダリングに用いた設定とはまったく異なる「ライト設定およびカメラ設定のキャラクターレンダリング画像」を、背景画像の上に合成することができます。

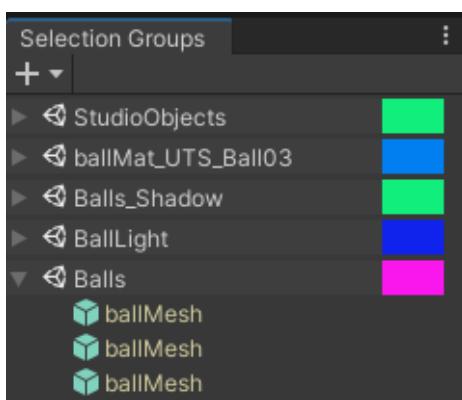
それでは、第2レイヤーとして表示するキャラクターレンダリング画像の設定をします。

UTS/HDRPマテリアルを設定する

レンダリングされるballMeshのマテリアルをUTS/HDRPに変更します。
マテリアルの**Shader**を**HDRP/Toon**に変更して、欲しいルックを作成します。
ボールが背景に落とすキャストシャドウは、背景画像側でレンダリングするため、本レイヤーでは不要になりますので、OFFにしてしまいます。



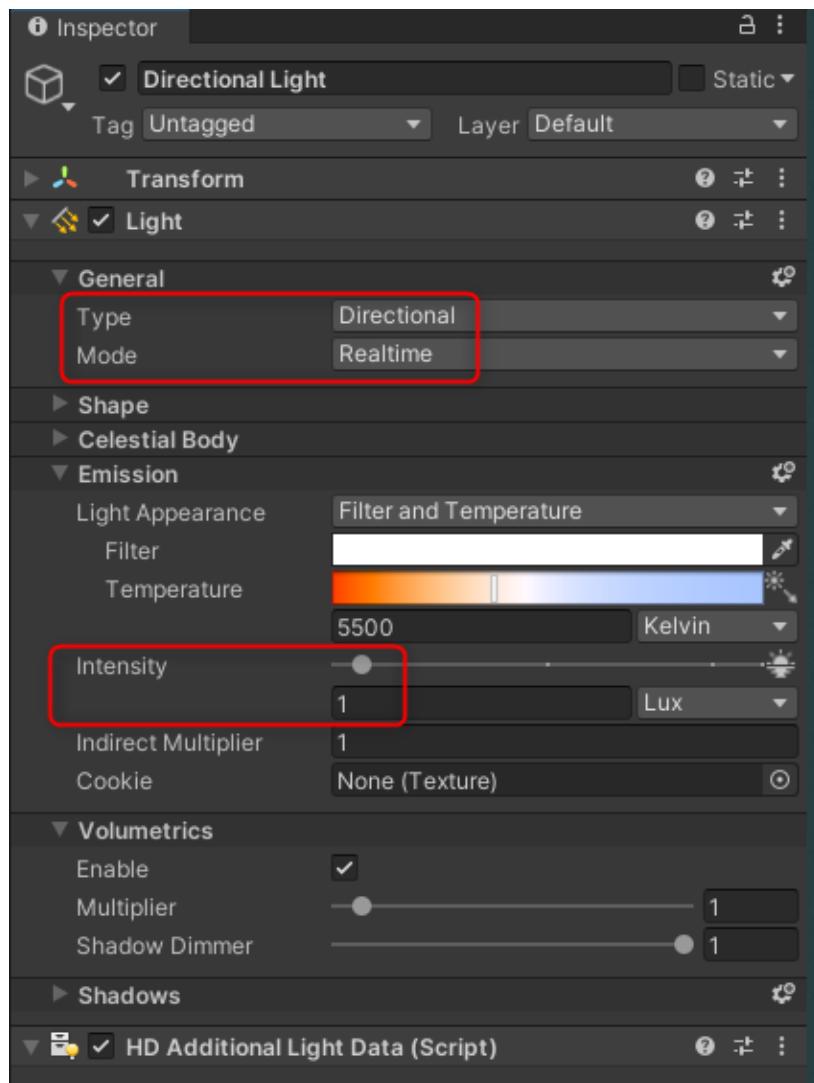
マテリアルの変更が終わったら、各ボールのballMeshを、Ballsセレクショングループとして選択できるようにしておきます。



キャラクター ライトを作成する

キャラクターレンダリングの専用ライトを作成します。キャラ用ライトには、セルシェーダーと最も相性のよいリアルタイムのディレクショナルライトを使用します。

HDRPのディレクショナルライトは、「明るい屋外の光源」の場合、インテンシティ(明るさ)がデフォルトで10000 Lux(ルクス)で設定されていますが、セルシェーダーにはそれでは明るすぎるので、下図のようにインテンシティは1に設定します。



Tips: もし1 Lux以上の明るさを設定したHDRP用ライトと一緒に、セルシェーダーを使う場合は、UTS/HDRPマテリアルのSceneLights Hi-Cut FilterをActivateにすると、効果的に白飛びが抑えられます。

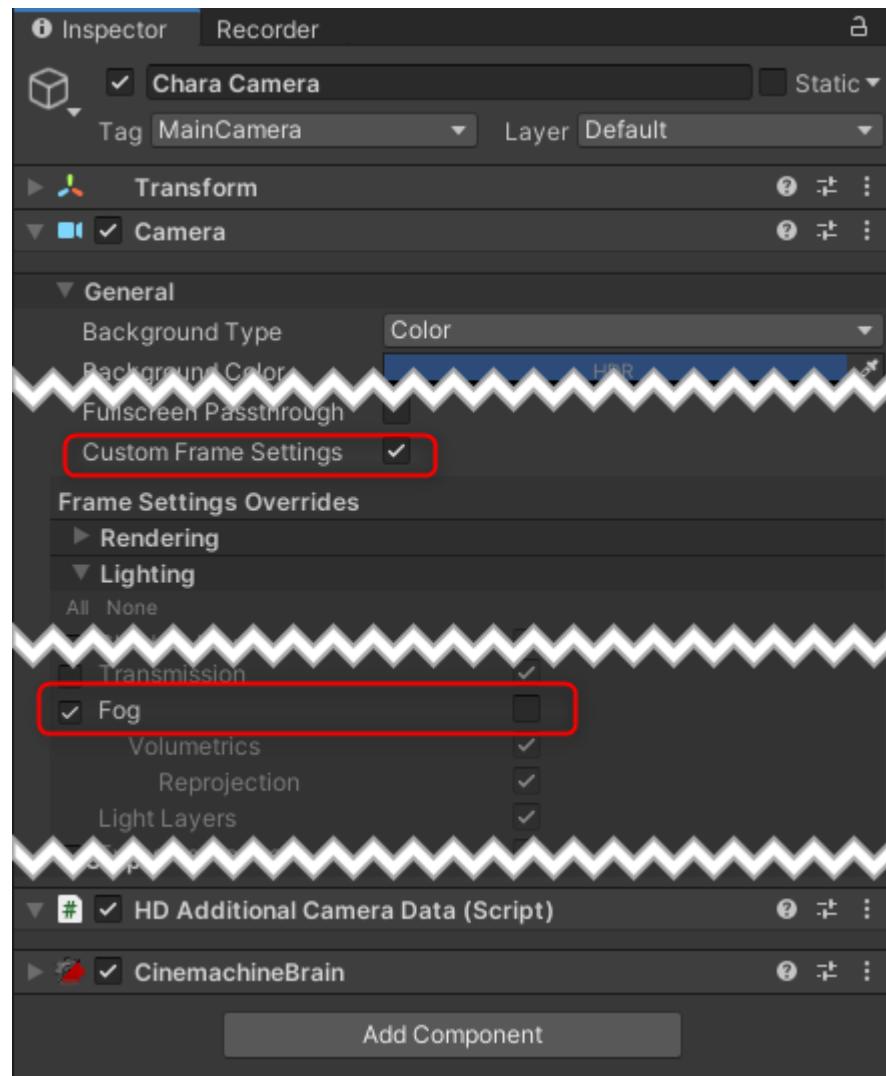
キャラ用のディレクショナルライトも、BallLightセレクショングループとして選択できるようにしておきます。



キャラクターカメラを作成する

キャラクターをレンダリングする専用カメラを、以下の手順で作成します。

- ほとんどの設定が流用できるので、ヒエラルキーインドウよりMain Cameraを複製し、Chara Cameraに名前を変更します。Main Cameraには、CinemachineBrainコンポーネントがアタッチされていますので、複製されたChara CameraにもCinemachineBrainがアタッチされています。
- Chara Cameraをインスペクターウィンドウに表示し、CameraコンポーネントのGeneralを開き、Custom Frame Settingsをチェックします。
- 「Frame Settings Overrides」という新しいメニューが追加されますので、Lightingを開き、「Fog」の冒頭のチェックをONにしてから、「Fog」の右側にあるチェックボックスをOFFにします。
この操作で**Chara Camera**のフォグが無効になります。これは後で説明をするαチャンネルを有効にしたレイヤー合成(**Use Src Alpha**)を正しく機能させるのに必須です。



キャラクターカメラをCinemachine Trackに登録する

1. Chara Cameraも、Main Cameraとまったく同じように、TimelineのCinemachine Trackで動かしたいので、Main CameraがバインドされているCinemachine Trackをデュプリケートします。
2. 複製されたCinemachine Trackには、新たにChara Cameraをバインドします。
3. Timeline上に2つのCinemachine Trackができますので、Track Groupでまとめておきます。



キャラクターのレンダリング画像をLayerノードで重ねる

Selection Groupノードを2つ追加して、それぞれに

- Balls
- BallLight

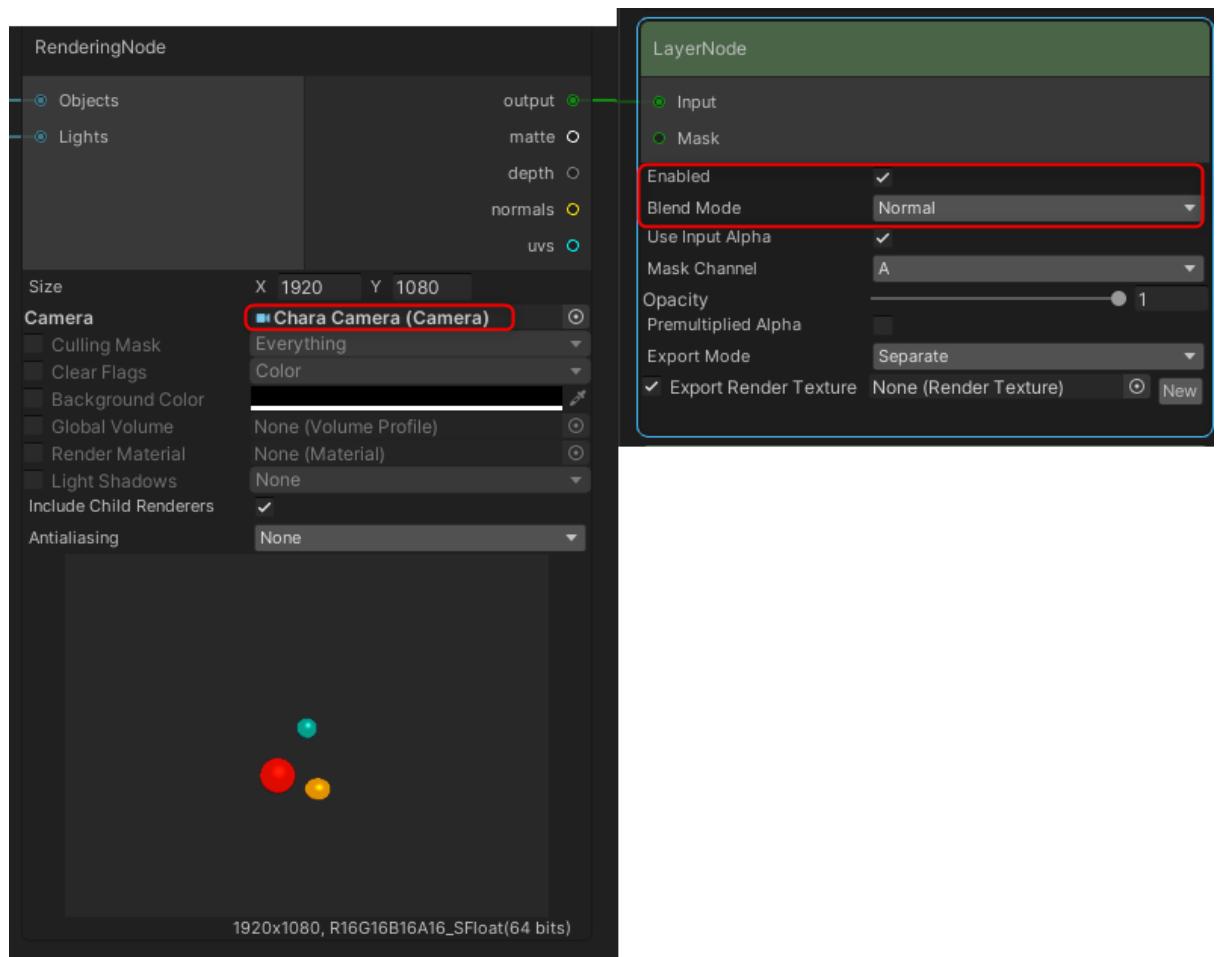
をバインドします。

つぎにキャラクターの画像をレンダリングするために、Renderingノードを追加します。

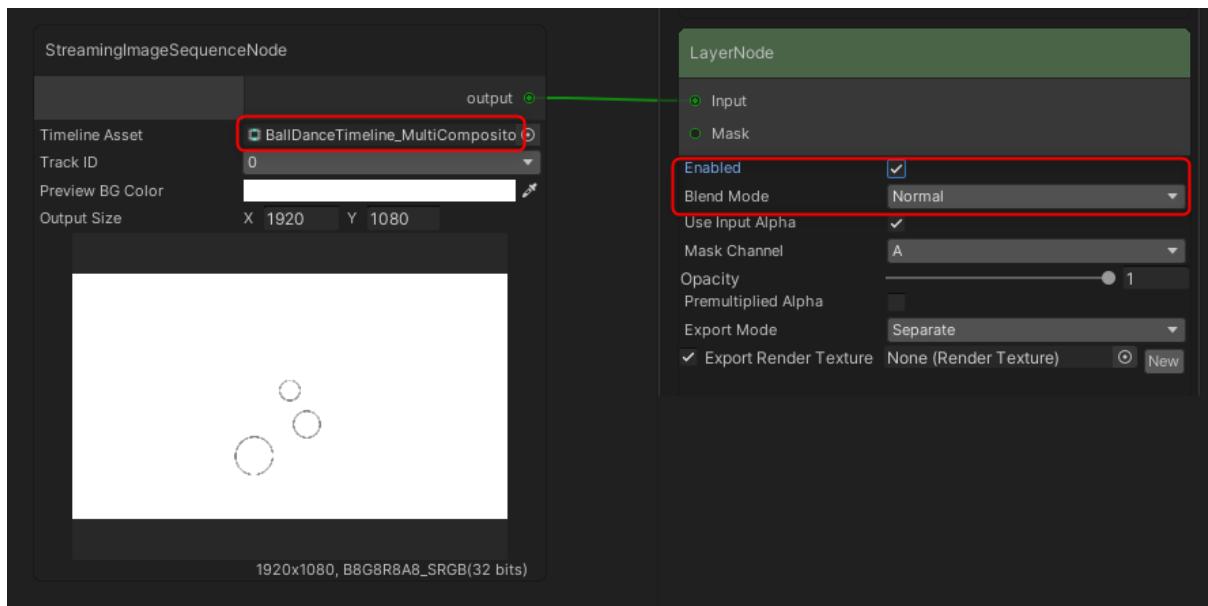
- objectsポート⇒Ballsセレクショングループを接続
- lightsポート⇒BallLightセレクショングループを接続
- Camera⇒Chara Cameraを指定

そして、**キャラクターのレンダリング結果は、αチャンネルを有効にしてレイヤー合成がしたいので、合成するLayerノードのUse Src Alphaを有効にします。**(ほとんどの場合、マスク用のチャンネルはRGBAの内、Aチャンネルで得られますので、Mask ChannelはAのままでかまいません)

最後に、Layerノードのinputポートに、Renderingノードからのoutputを接続し、Enabledをチェック、Blend ModeをNormalにします。



第3レイヤー:Pencil+ Line Cacheの合成



第3レイヤーでは、セルルックキャラクターのデザインで重要なポイントとなる、トーンラインの合成を行います。トーンラインは、Pencil+ 4 Line for Unityで事前にレンダリングした、Pencil+ Line Cacheを連番画像ソースとしています。

もちろん毎回、Pencil+ 4 Line for Unityでリアルタイムにトーンラインを作成してもよいですが、Pencil+ Line Cacheを使うことで、

- キャラクターのメッシュが細かくなつて、Pencil+ 4 Line for Unityでのリアルタイムレンダリングが難しくなつても、キャッシュを使っていれば軽快なスクラップ再生による完成画面の確認ができる。
- キャッシュされているトーンライン画像を直接手で修正することもできるので、細かい修正でつどPencil+ 4 Line for Unityの設定を修正する必要がなくなる。

などのメリットが生まれます。

Pencil+ Line Cacheは、Canvas/Imageを使ってスクリーン合成する方法もありますが、今回はVisual Compositorのノードから合成するための手順を紹介します。

Pencil+ 4 Line for Unityの準備

Pencil+ 4 Line for Unityをプロジェクトにセットアップしていきます。

詳しい手順については、別ドキュメント『Pencil+ Line Cache セットアップ』の他、製品版のヘルプを参照してください。

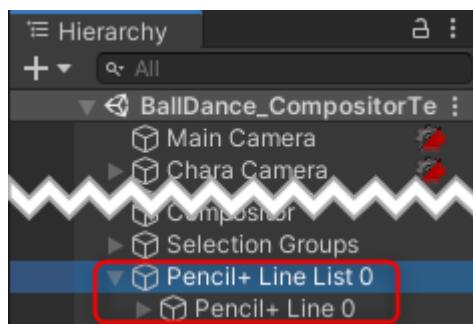
ここでは、HDRP版パッケージでのセットアップについて説明します。

Note: Pencil+ Lineを適用するFBXモデルのUnityへのインポート時に、Read/Write

EnabledのチェックをONにしましょう。これを忘れるとな、外部からインポートしたメッシュにPencil+ Lineが表示されません。

Pencil+ Line Listゲームオブジェクトをシーン中に作成する

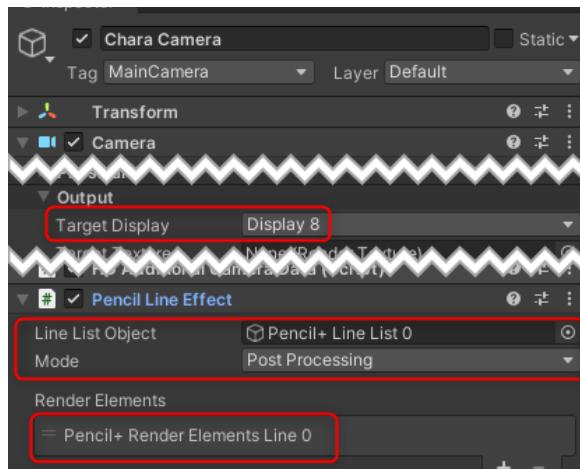
1. まず、空のゲームオブジェクトをシーン中に作成し、Line List Nodeコンポーネントをアタッチします。
2. インスペクターより、Line List Nodeコンポーネントに、新規のLine Listを追加し、ゲームオブジェクトもLine Listと同名にリネームしておきます。
3. 作成したLine Listに、Pencil+ Lineを適用したいキャラクターのマテリアルなどを登録し、ライン設定をしておきます(まだこの段階では表示できません)。



キャラクターカメラにPencil Line Effectをアタッチする

Pencil+ Lineはトゥーンラインカメラシェーダーですので、キャラクターをレンダリングするカメラにアタッチします。キャラクターはChara Cameraですので、以下の手順でPencil Line Effectコンポーネントをアタッチしていきます。

1. インスペクターより、Chara CameraのOutputを開き、Target DisplayをDisplay 8に設定します。
2. 続いて、Chara Cameraに、Add ComponentボタンからPencil Line Effectをアタッチします。
3. インスペクターより、Pencil Line EffectのLine List Objectに、上で作成したPencil+ Line Listゲームオブジェクトを登録します。



4. この時、Pencil+ Line Listが適宜設定されていれば、ゲームビューをDisplay 8に切り替えることで、トゥーンラインを確認することができます。



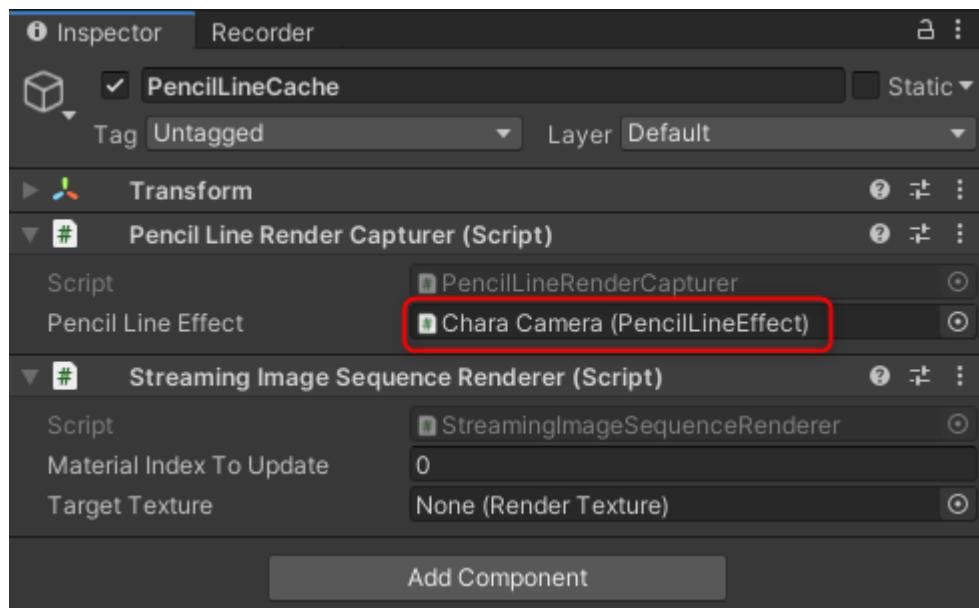
Pencil+ Line Cacheゲームオブジェクトの作成

続いて、Timelineから参照できるように、Pencil+ Line Cacheゲームオブジェクトをシーンに作成します。

1. 空のゲームオブジェクトをヒエラルキーに作成し、PencilLineCacheとリネームします。
2. 作成したゲームオブジェクトに、Pencil Line Render Captureコンポーネントをアタッチします。
3. インスペクターより、Pencil Line Render CaptureコンポーネントのPencil Line Effectに、Pencil Line Effectがアタッチされているカメラを指定します。本シーンの場合は、Chara Cameraです。
4. 続いて、同じゲームオブジェクトにStreaming Image Sequence Rendererコンポーネントをアタッチします。

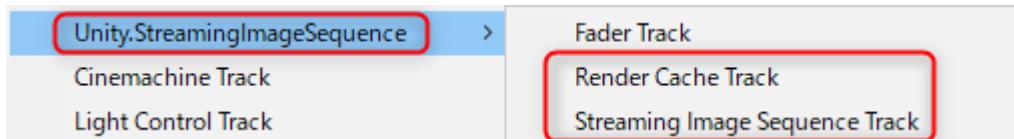
以上の結果として、PencilLineCacheゲームオブジェクトは、以下の図のような状態になります。

PencilLineCacheゲームオブジェクトは、Pencil+ Lineのキャッシュ画像を作る機能と、作成されたキャッシュ画像を再生する機能を受け持つことになります。

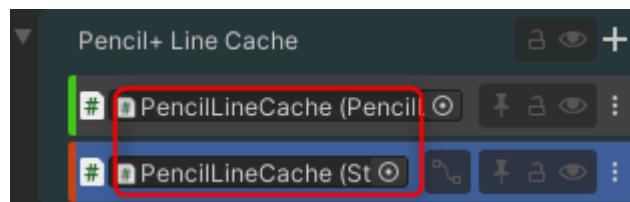


Timeline上にキャッシュデータの記録と再生トラックを作成する

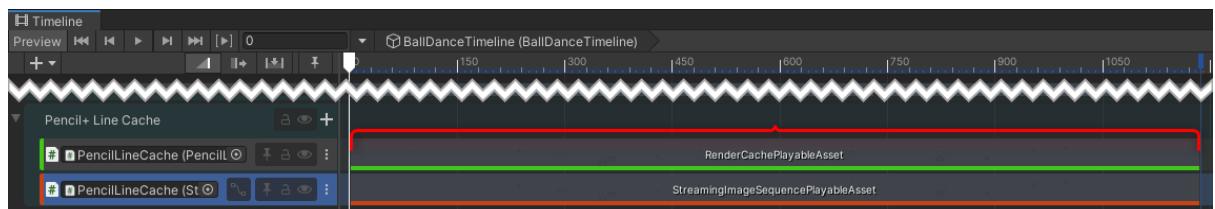
Timeline上にPencil+ Lineキャッシュデータを記録するRender Cache Trackと、キャッシュデータを再生するStreaming Image Sequence Trackを作成します。これらを追加するには、Timeline左上の「+」ボタンまたはTimelineトラック上の開いているところで右クリックして呼び出すコンテクストメニュー内のUnity.StreamingImageSequence階層内から選択します。



1. Timelineウィンドウに、Render Cache Trackを作成し、新規に作成したトラックに、Render Cache Playable Assetを追加します。
2. 同様に、Timelineウィンドウに、Streaming Image Sequence Trackを作成し、新規に作成したトラックに、Streaming Image Sequence Playable Assetを追加します。
3. 作成した両方のトラックに、共にPencilLineCacheゲームオブジェクトをバインドし、トラックグループPencil+ Line Cacheにまとめます。

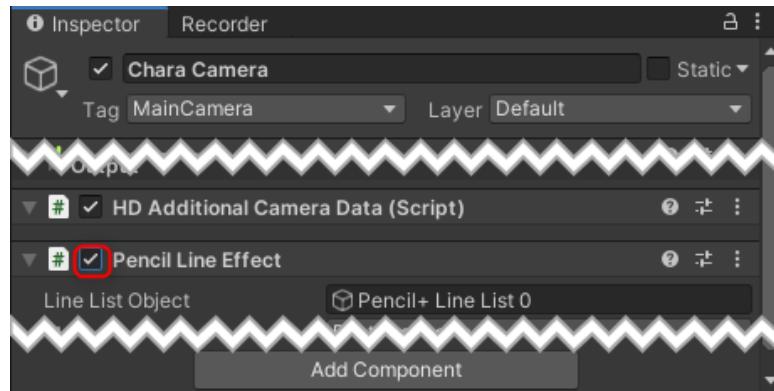


4. キャッシュを作成する区間の長さに合わせ、両方のクリップを同じ長さになるように調整します。

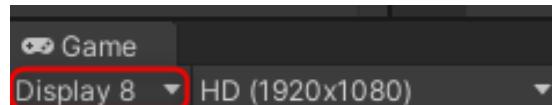


キャッシュデータを記録する

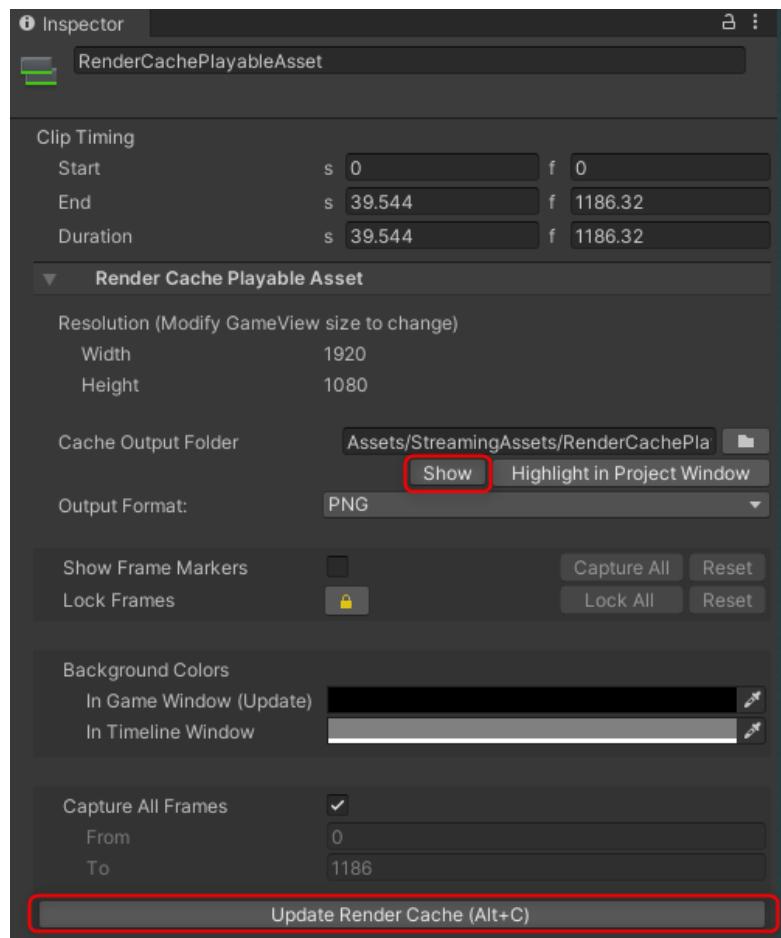
1. Chara Cameraにアタッチされている、Pencil Line Effectコンポーネントを有効にします。



2. ゲームビューを「Display 8」に切り替えます。



3. Timelineウィンドウより、Render Cache Playable Assetクリップを選択します。
4. インスペクターウィンドウの「Update Render Cache」ボタンをクリックすると
Pencil+ Line CacheがStreamingAssetsフォルダ内に記録されます。

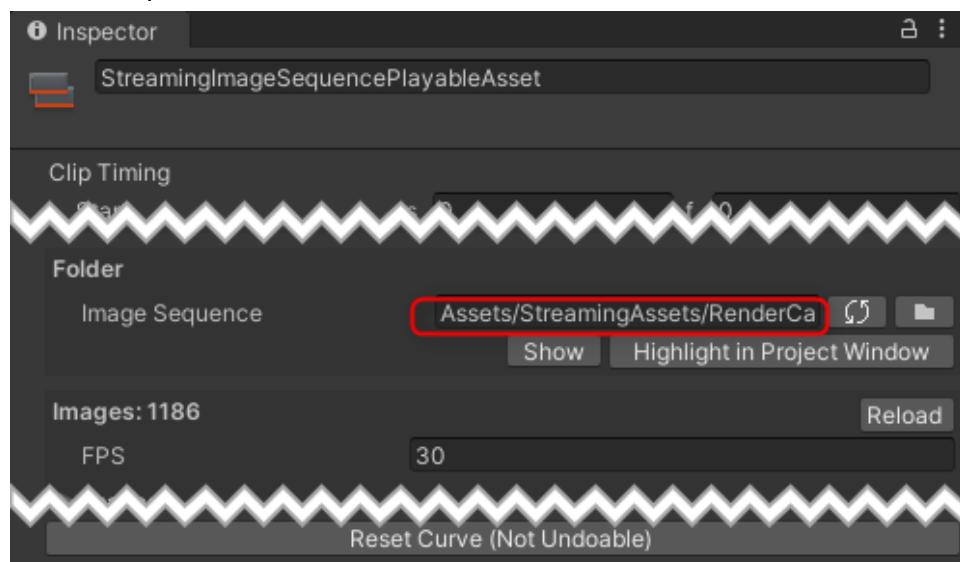


5. キャッシュデータの作成が終了したら、Chara CameraのPencil Line Effectコンポーネントを再びディスイネーブルにします。

キャッシュデータをVisual Compositorに表示する

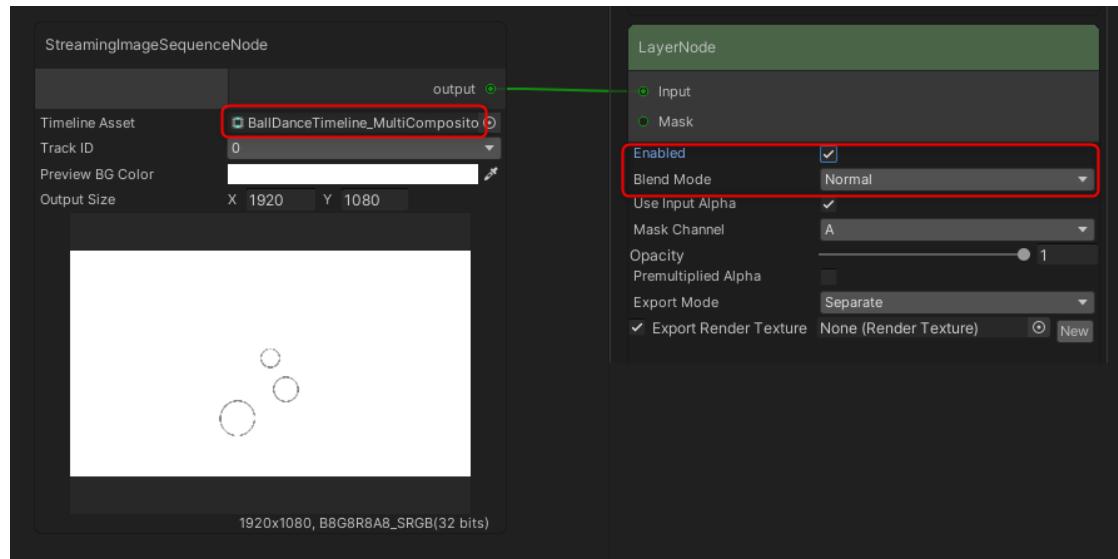
ラインキャッシュデータをVisual Compositorウィンドウ上で扱うには、Stream Image Sequence Nodeを使います。その前に、Timeline上のStreaming Image Sequenceトラックが、キャッシュデータにアクセスできるようにします。

1. Streaming Image Sequence Playable Assetを選択し、インスペクターより、Folder>Image Sequenceのターゲットとして、StreamingAssetsフォルダ内のキャッシュデータが保存されているフォルダ(RenderCachePlayableAssetのCache Output Folderのこと)を  から選択します。

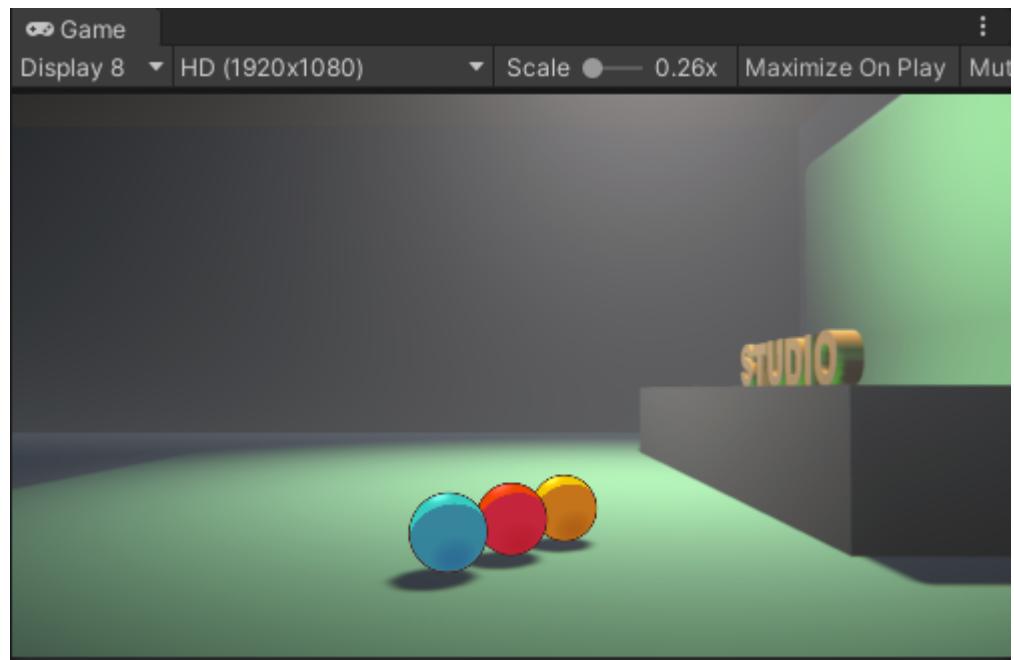


2. Visual Compositorウィンドウ上で、新規のStream Image Sequence Nodeを追加し、さらにLayer Nodeを追加します。追加した2つのノードを接続します。第2レイヤーと同じように、前回作成したLayer Stackの下部に第3レイヤーとして使用するLayerノードをドラッグしていきLayer Stackとドッキングさせます。これで、第1、第2レイヤーの結果の上に第3レイヤーが重なります。

3. Stream Image Sequence NodeのDirectorにTimeline Playable Assetを登録します。Layer NodeのEnabledをチェック、Blend ModeをNormalにします。



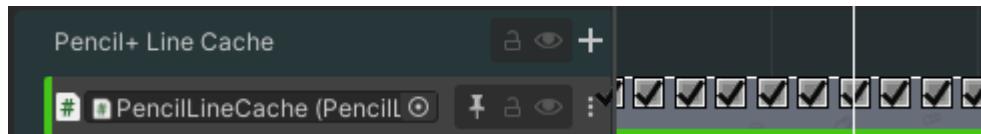
4. ゲームビューがDisplay 8表示になっていることを確認し、Timelineの再生ヘッドを左右に動かしてスクラブすると、Visual Compositorによってラインキャッシュデータが画像に合成されて再生されることが確認できます。



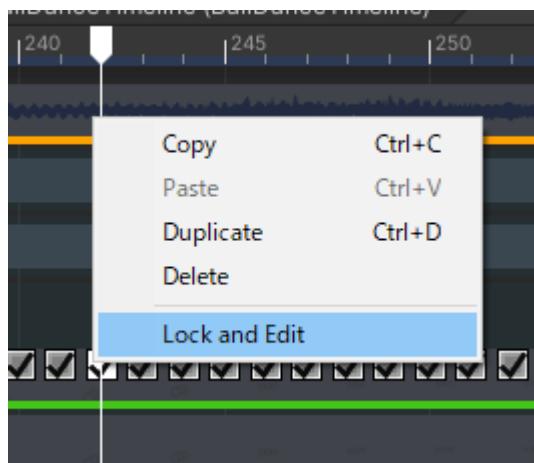
個別のキャッシュ画像データを手で修正する

最後に、連番画像としてキャッシュされているライン画像データをPhotoshopなどで修正する方法を紹介します。

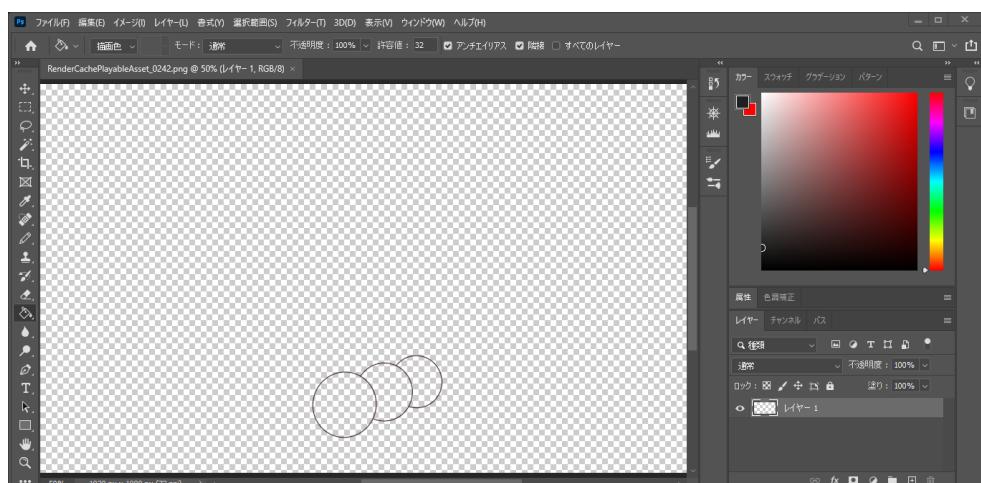
1. Timeline上で再生ヘッドをスクラブし、修正したいフレームを探します。
2. Timelineで1フレーム分のゲージが大きく十分に表示されるぐらいまでズームインします。
3. キャッシュデータを記録したRender Cache Playable Assetクリップを選択し、インスペクター上で、Show Frame Markersをチェックします。
すると、クリップ上に現在有効になっているキャッシュデータフレームがチェックボックスで表示されます。



4. 修正したいフレームを選択して、右クリックから表示されるコンテキストメニューまたはインスペクター上の「Lock and Edit」ボタンを押します。

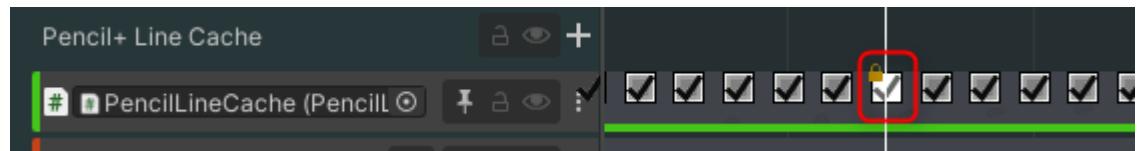


5. 指定したフレームが、Photoshopなど、Unityに登録してある画像エディタに読み込まれるので、エディタ上で修正します。

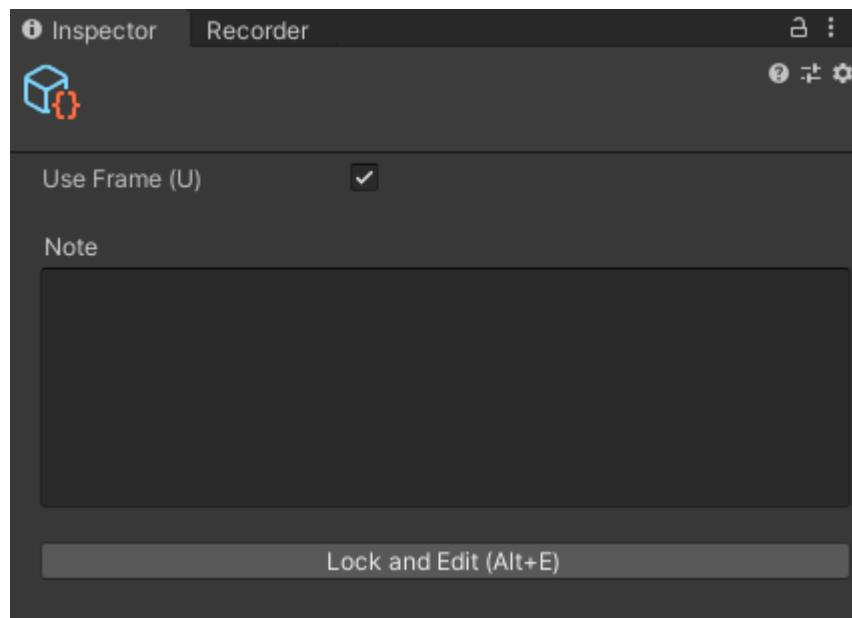


Tips: Photoshopの登録は、Edit>Preferences>External Tools>Image applicationから登録します。

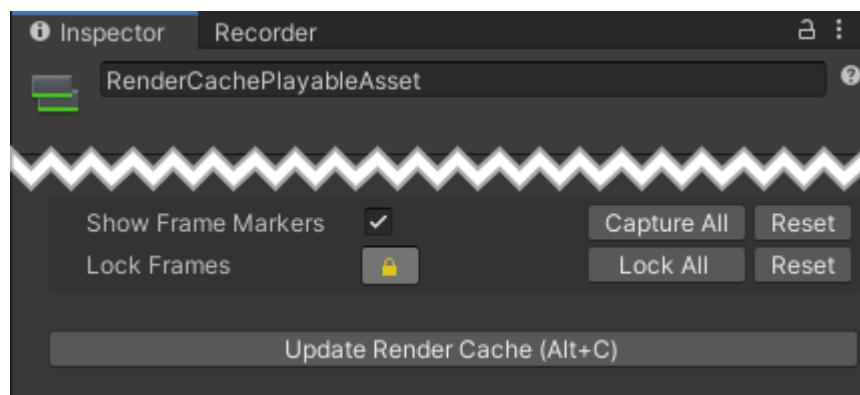
6. 修正したフレームは、ロックがかかるたった状態になっているので、キャッシュデータを更新しても保持されます。



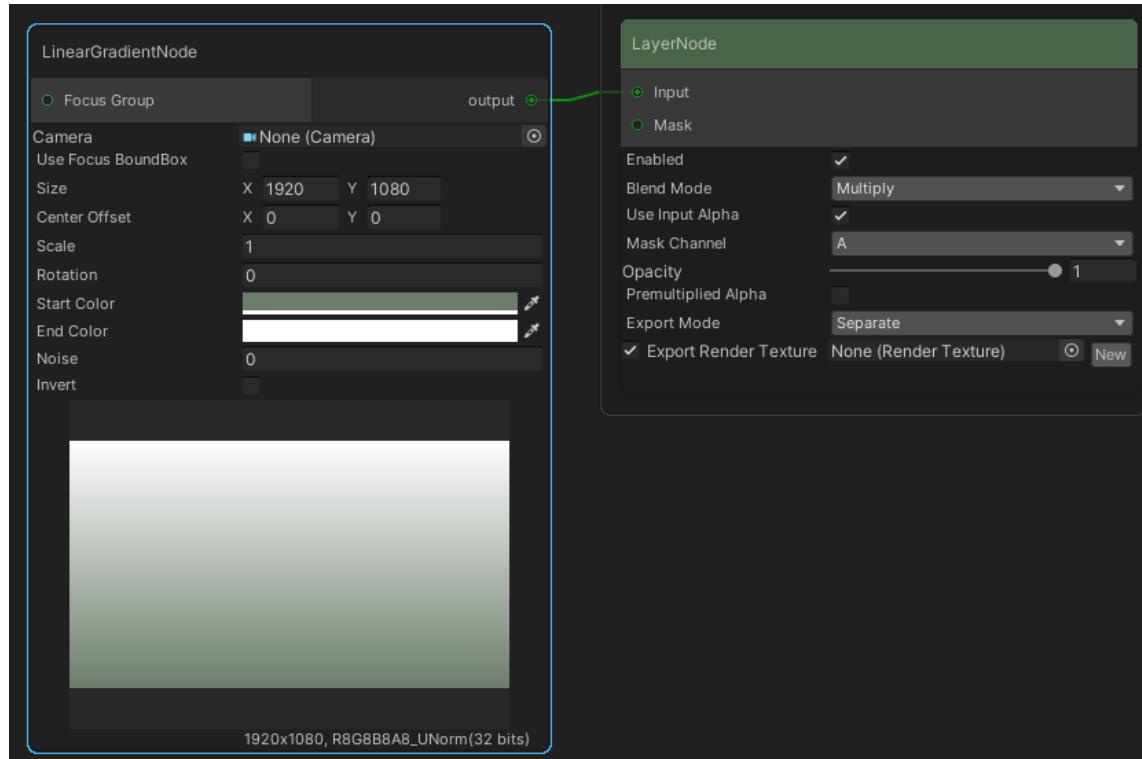
7. また修正したフレームを選択中の場合、インスペクター上のNoteにメモを残しておくことができます。



8. 再びRender Cache Playable Assetクリップを選択し、Show Frame Markersをチェックオフにすると、クリップ上からチェックボックスが消えます。
9. ロックのリセットは、Lock Framesの ボタンを押してから、Resetボタンで行います。



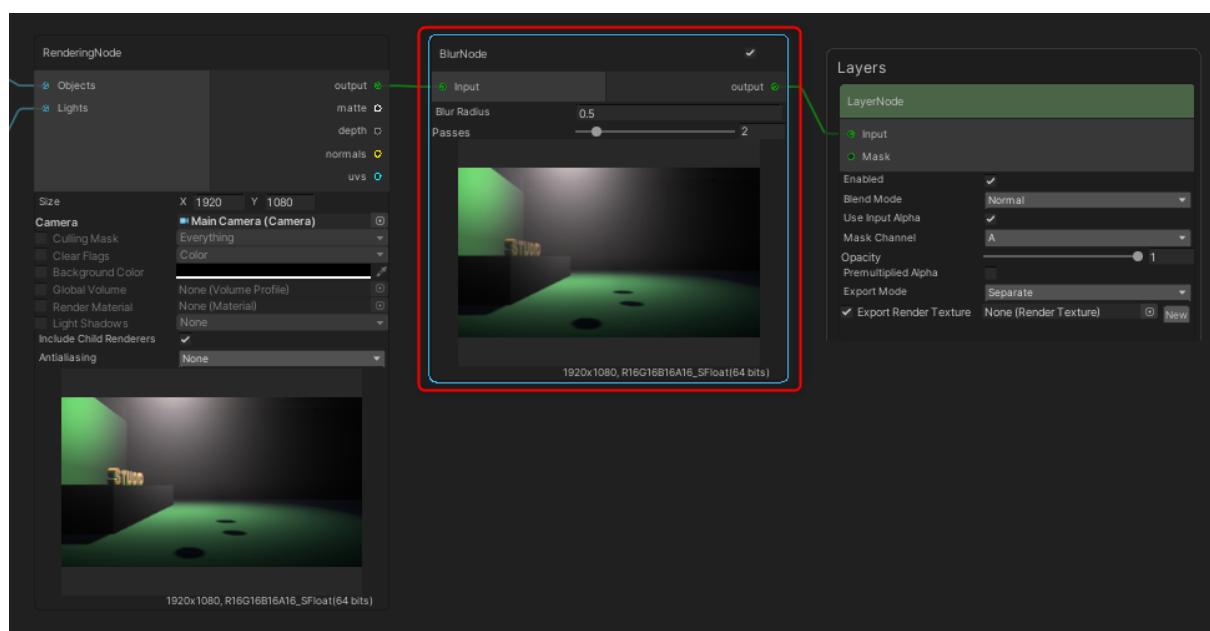
第4レイヤー:画面全体を馴染ませるパラ効果の追加



最後の第4レイヤーでは、背景画像とセルルック画像を馴染ませる処理(パラ効果の追加)を行っています。

馴染ませる処理は、撮影するカットによって手法が様々ですが、ここでは一例として、上から下へのグラデーション画像を**Linear Gradient**ノードで生成し、それを**Multiply**モードでブレンドしています。

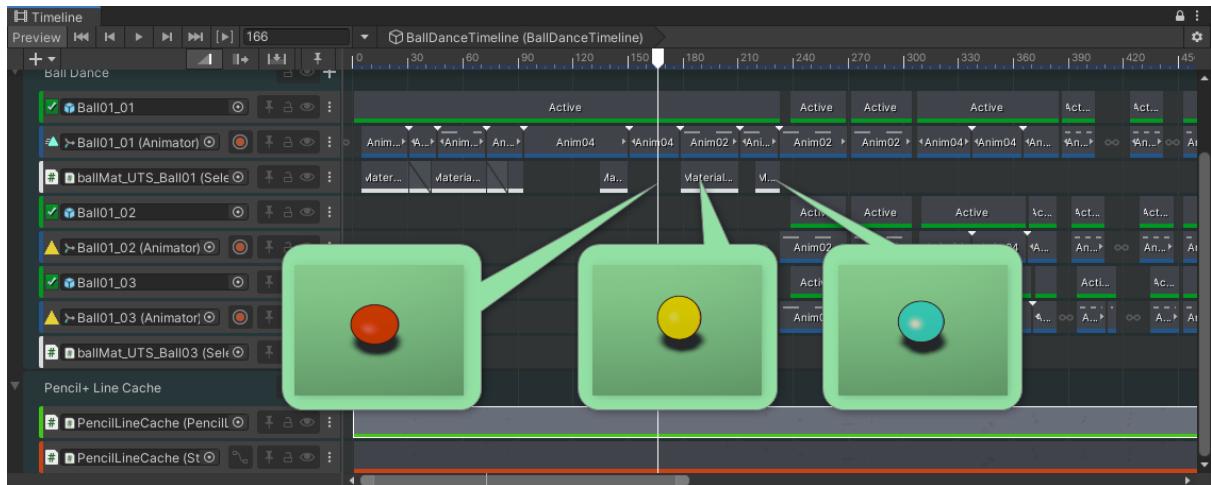
別途、フィルムグレインのノイズ画像を作成して、載せてみるのもよいでしょう。



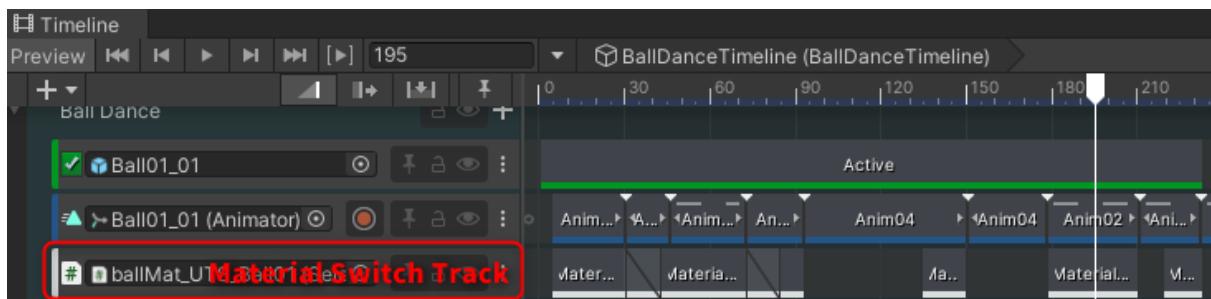
他にも、第1レイヤーの背景レンダリング画像をそのまま使うのではなく、レンダリング後にBlur

ノードを挟み、セルルックのボールよりも軽くぼかすことで、「2D背景画像」っぽく見えるように処理しています。このように、各段階で普段は2D画像加工に使われるフィルタをかけることで、3Dレンダリング画像を2D画像風に見せる工夫をすることができます。

Timelineからカラー変更をおこなうMaterial Switch Track



Timeline上で再生ヘッドを0~230フレーム程度まで動かすと、3色のボールが現れます。このシーンは、赤いボールのカラーをTimeline上で、黄色と青にスイッチしています。



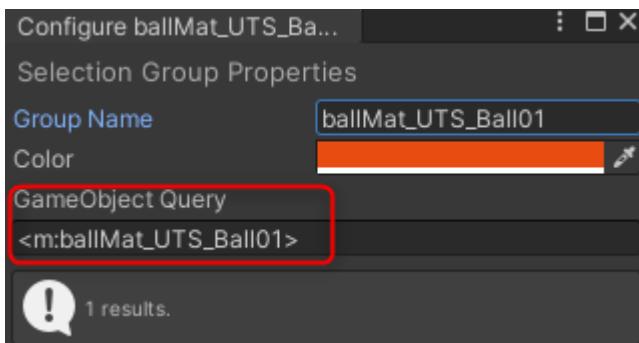
Material Switch機能は、**Material Switch Track**でマテリアルの設定をオーバーライドすることでカラー変更を実現します。

特にカラー変更に関しては、アニメ制作ワークフローで便利なように、シーン単位でのカラー設計書等の画像データを直接読み込んで変更することができます。

Material Switch Trackでも、セレクショングループを使用して変更したいマテリアルを選択して指定します。

Material Switch Trackのセットアップは、以下の手順で行います。

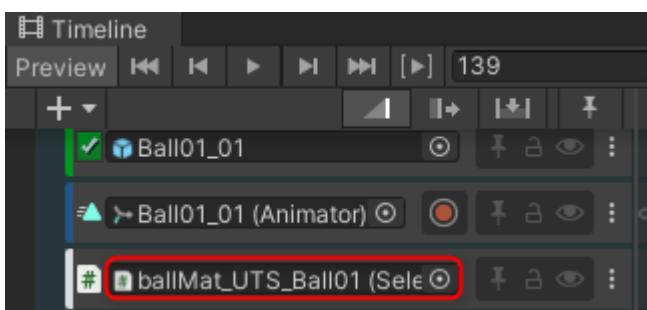
- セレクショングループで、変更したいマテリアルがアタッチされているボールのメッシュを選択します。マテリアル名でタイプ検索をするのがよいでしょう。



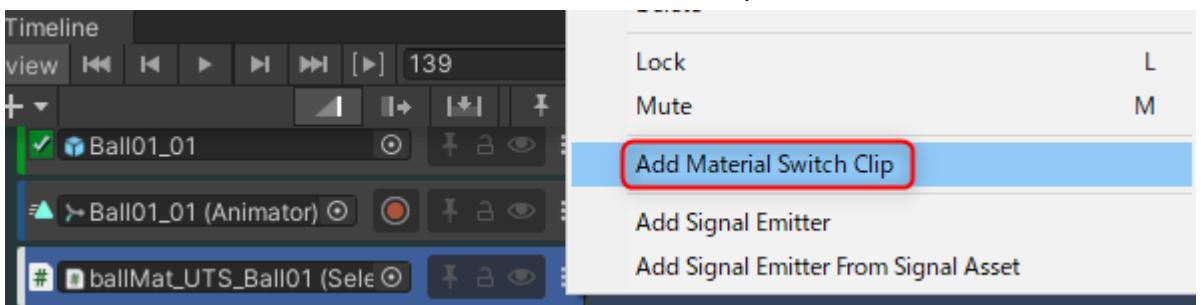
- Timelineで、左上の「+」ボタンまたはTimelineトラック上の開いているところで右クリックして呼び出すコンテキストメニュー内の「トラックの追加」から、Unity.Material Switch階層内からMaterial Switch Trackを選択し新規作成します。



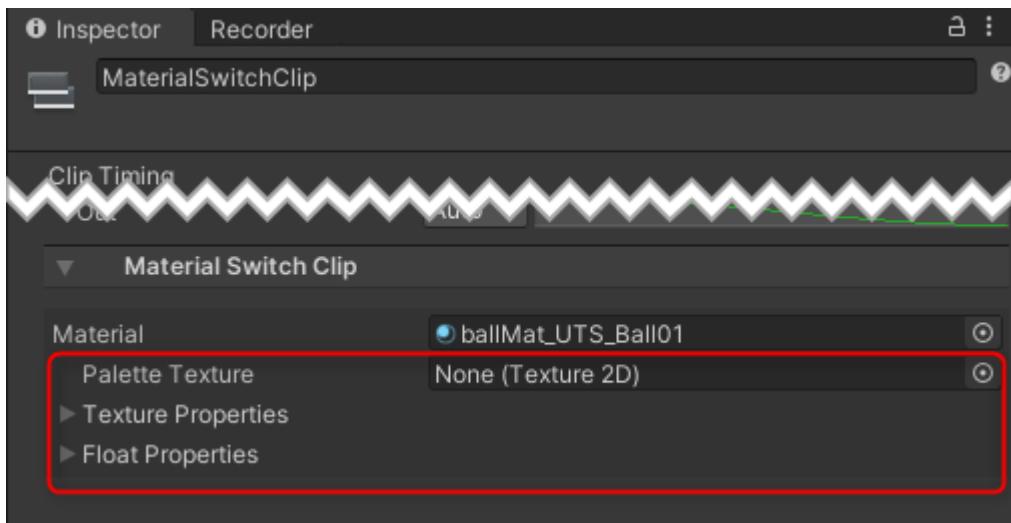
- 作成したMaterial Switch Trackに、変更したいマテリアルを含むセレクショングループをバインドします。



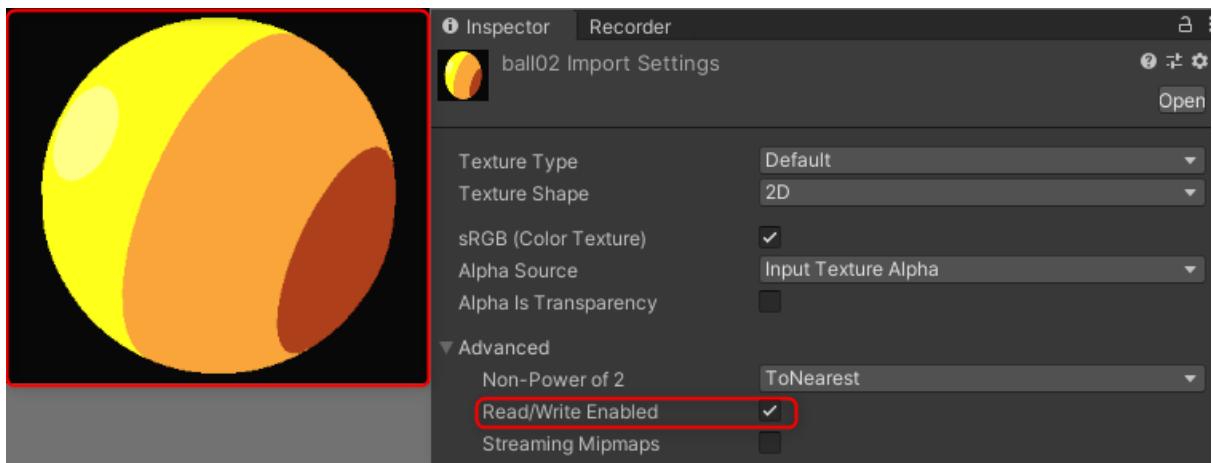
- Material Switch Track上で右クリックし、Material Switch Clipを追加します。



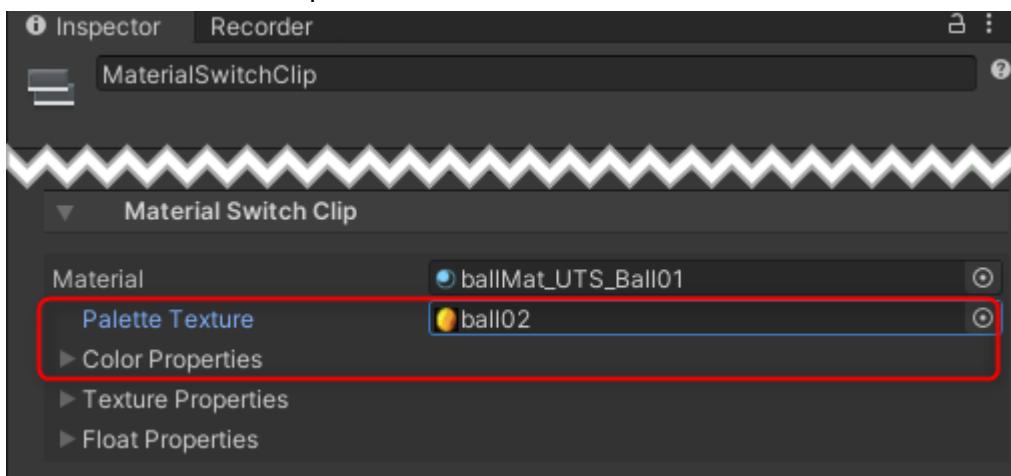
- 新規に作成したMaterial Switch Clipをインスペクターで見ると、以下のようになっています。Materialはバインドされているセレクショングループから設定されています。
下図の赤枠で囲まれている部分の設定で、元のマテリアル設定をオーバーライドすることでマテリアルが変更できる仕組みです。
ここでは、テクスチャを使って、UTS/HDRPの通常色、1影色、2影色を変更してみます。



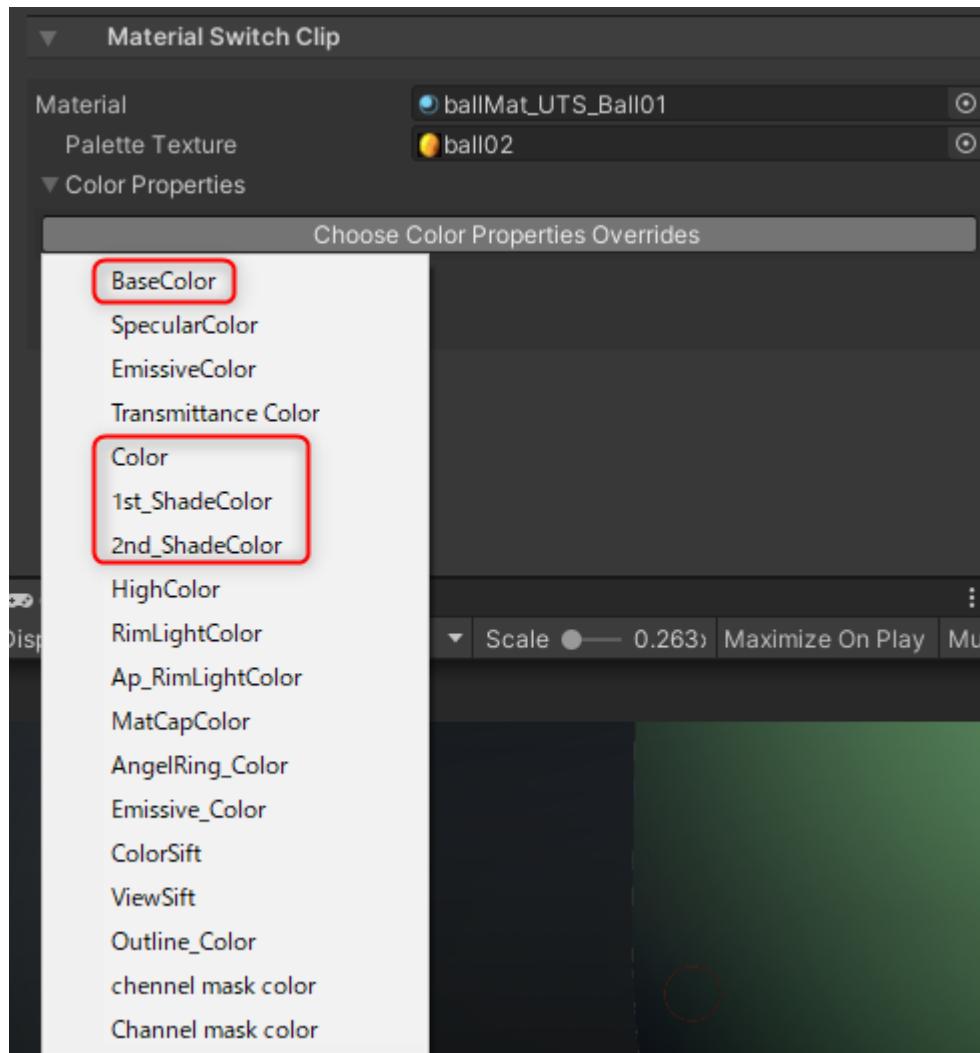
6. 赤いボールのマテリアルを黄色に変更しますので、以下のような画像を用意します。
画像のインポート設定で、「Read/Write Enabled」をチェックしておきます。



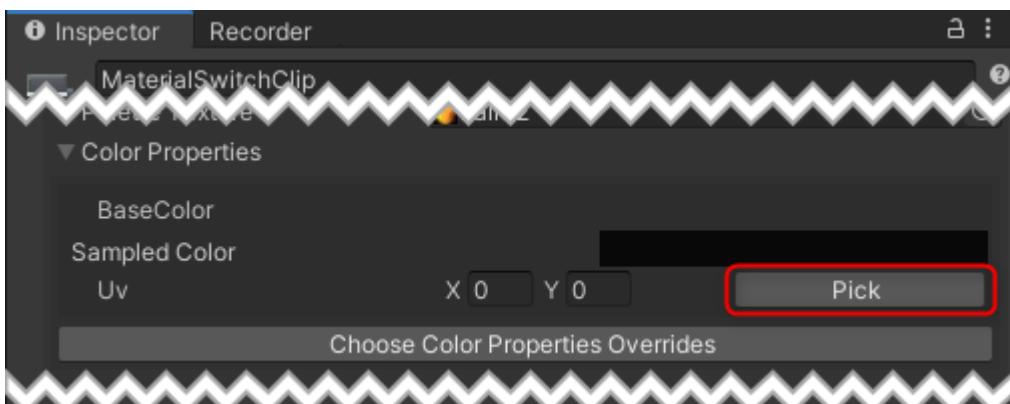
7. Material Switch ClipのPallet Textureに6でUnityにインポートした画像を指定します。
すると、新たにColor Propertiesが追加されます。



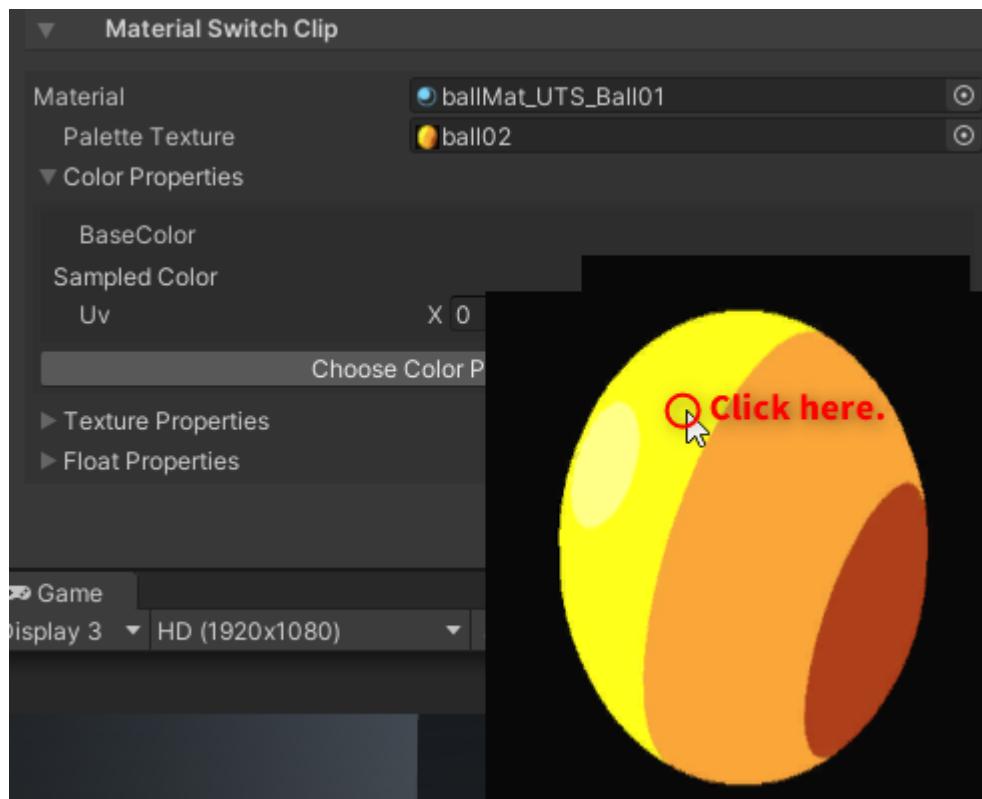
8. Color Propertiesを開くと、「Choose Color Properties Overrides」というボタンが現れますので、変更するカラーのプロパティを指定します。UTS/HDRPの場合、
- 通常色 ⇒ BaseColorとColorの両方に指定する
 - 1影色 ⇒ 1st_ShadeColorに指定する
 - 2影色 ⇒ 2nd_ShadeColorに指定する



9. まずは、BaseColorに登録してみましょう。BaseColorを選択すると、次のような画面に変わります。Pickボタンを押してください。



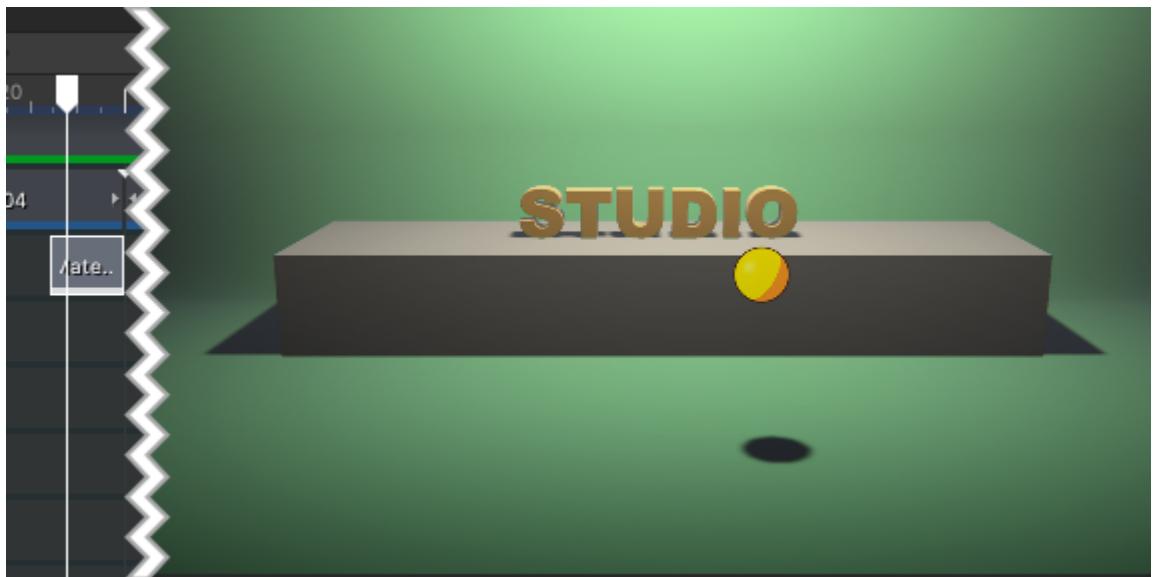
10. Pickボタンを押すと、Palette Textureで指定した画像が開きますので、通常色として指定したい部分をカーソルでクリックしてください。



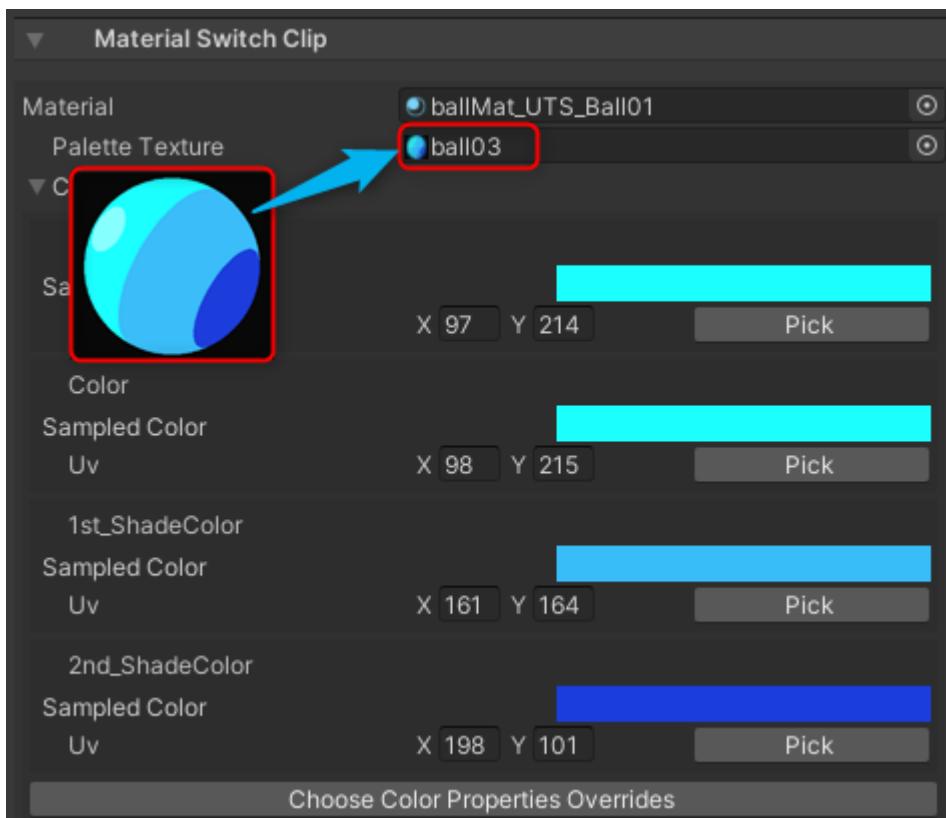
11. すると、画像が閉じて、UvのXとYに値が入ります。クリックした位置のUV値です。
同様にColor、1st_ShadeColor、2nd_ShadeColorも指定します。



12. 上のMaterial Switch Clipの位置に再生ヘッドがさしかかると、マテリアルのカラーが黄色系にスイッチされるのが確認できます。



13. 黄色に設定したMaterial Switch Clipをデュプリケートして、クリップ位置を移動します。
デュプリケートしたMaterial Switch Clipを今度は青系に変えてみます。
同様に、青系のPallet Textureを用意してテクスチャを差し替えると、UVの位置さえ共通であればカラーがすぐに差し替わります。



Note: カラーサンプル表示を更新するためには、「別のトラックを選択し、再度トラックを選択し直す」などをして、インスペクター表示を手動で切り替えする必要がありますが、内部参照はPallet Textureを切り替えた時点で切り替わりますので、ゲームビューでは更新されています。



Material Switch Trackは、上のようなカラー設定イメージ単位でのパレットチェンジの他に、マテリアルに設定される様々なマップやテクスチャのスイッチや、カラーの塗り分け領域を設定するfloat値のスイッチも、Timeline上で設定することができます。

Material Switch Trackを使うことで、キャラの基本となるマテリアル設計を変更することなしに、シーン単位で詳細にキャラクターのカラー設計をコントロールすることが可能になります。

Tips: UTS/HDRPが持っているシャドウコントロールマップも**Material Switch Track**で変更することができますので、事実上好きなようにマテリアル側から影色のコントロールがされることになります。

Unity Recorderを使い、正確なフレームレートでキャプチャをする

ゲームビューに表示されるリアルタイム画像をUnity Recorderで記録することで、コンポジット結果やTimeline上で編集した音声データを連番画像やムービー、音声ファイルにレンダリング出力することが可能となります。

ここでは、正確なフレームレートでキャプチャするためのUnity Recorderの設定と、実際にキャプチャを実行する際に、十分なウォームアップタイムを取るための操作の仕方を解説します。

Unity Recorderのドキュメント

<https://docs.unity3d.com/Packages/com.unity.recorder@2.2/manual/index.html>

Package Managerウィンドウからは、以下の設定で見ることができます。
本Projectの場合、最初からインストールされています。



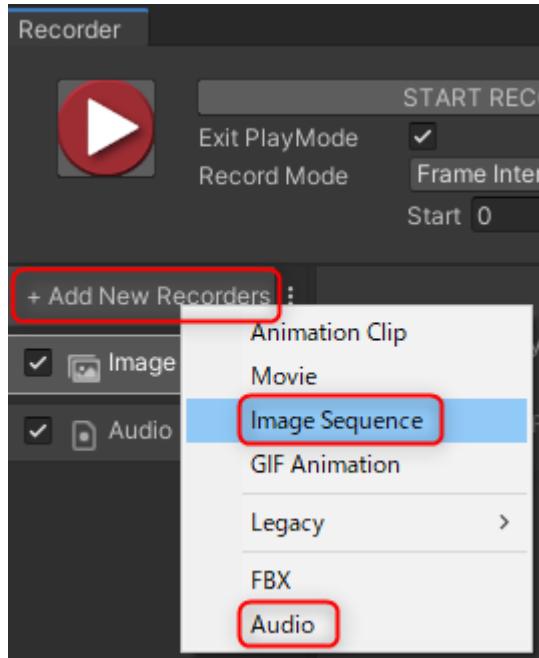
Unity Recorderの設定

Unity Recorderを操作するためには、メニューバーより、Window > General > Recorder > Recorder Windowsと辿って、Recorderウィンドウを開きます。

キャプチャする対象を追加する

Recorderウィンドウの「+ Add New Recorders」をクリックすると、キャプチャする対象をメニューで選択できます。

ここでは、連番画像データ(Image Sequence)と、音声データ(Audio)を追加することにします。



共通の設定

まず、共通となるレコーディング設定をします。以下の図を参考にしてください。

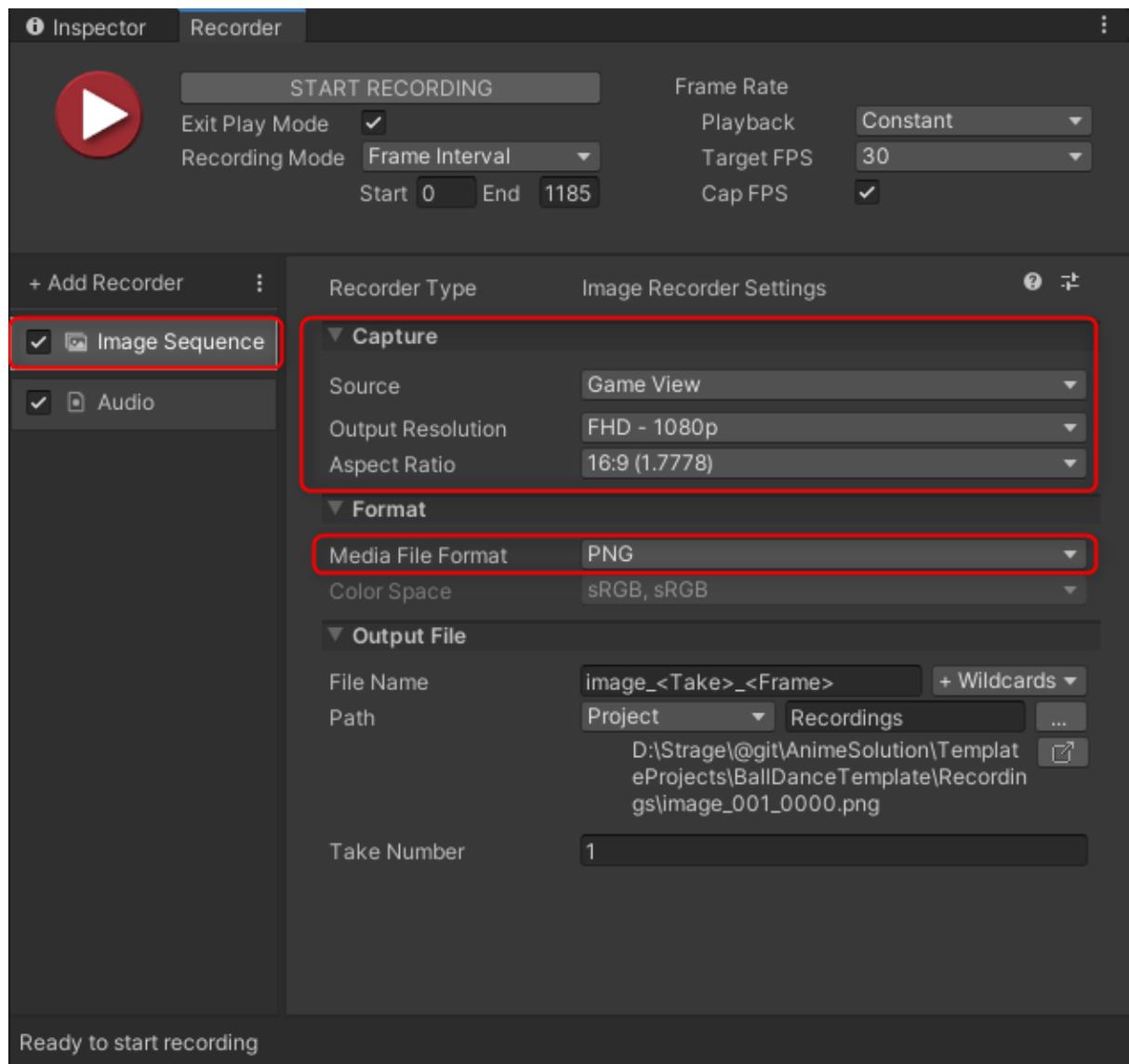


- Recording Mode ⇒ 「Frame Interval」にして、StartとEndにそれぞれTimelineの始点と終点のフレーム数を入力します。
- Frame Rate ⇒ Playbackは「Constant」に、TargetはTimelineに設定されているFrame Rateにあわせます。今回の場合は、「30」と入力します。(※ 24fpsでレコーディングをする場合には、24と入力します。)

Tips: Timelineのフレームレートは、Timelineウィンドウのギアアイコンから、**Frame Rate > 指定されたフレーム** で調べることができます。

Image Sequenceの設定

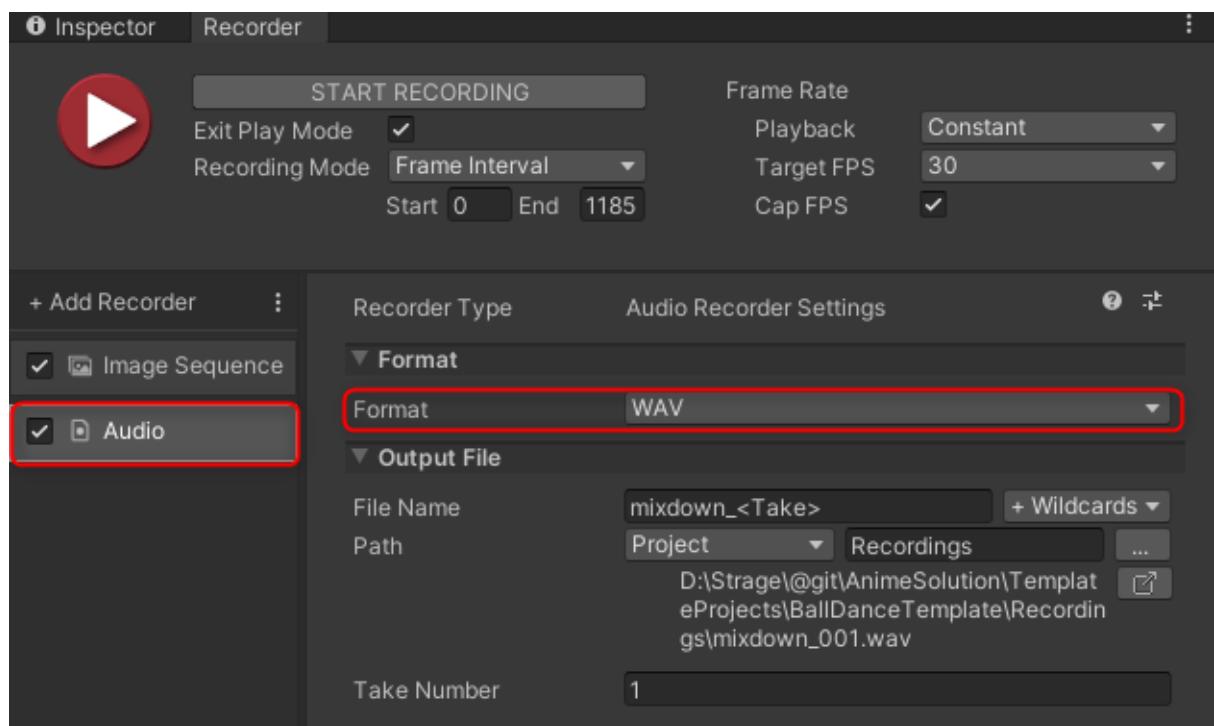
Image Sequence(連番画像)は以下のように設定しておきます。



- Capture ⇒ 「Game View」。
本プロジェクトの場合、Visual Compositorの出力先ゲームビューが「Display 8」になっているか、確認を忘れないでください。
- Output ResolutionやAspect Ratio ⇒ 最終出力の指定サイズに合わせます。
- Format ⇒ SDR画像の場合、「PNG」で。HDR画像の場合、「EXR」にします。

Audioの設定

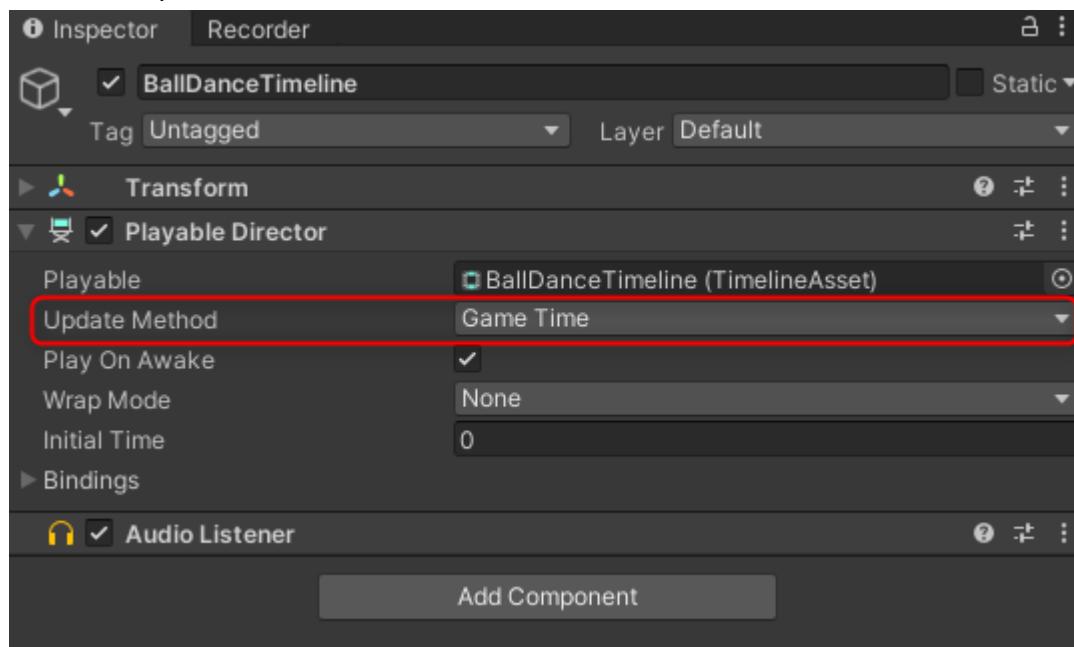
Audioは以下のように設定しておきます。



- Format ⇒ 「WAV」。

TimelineのPlayable Directorの設定確認

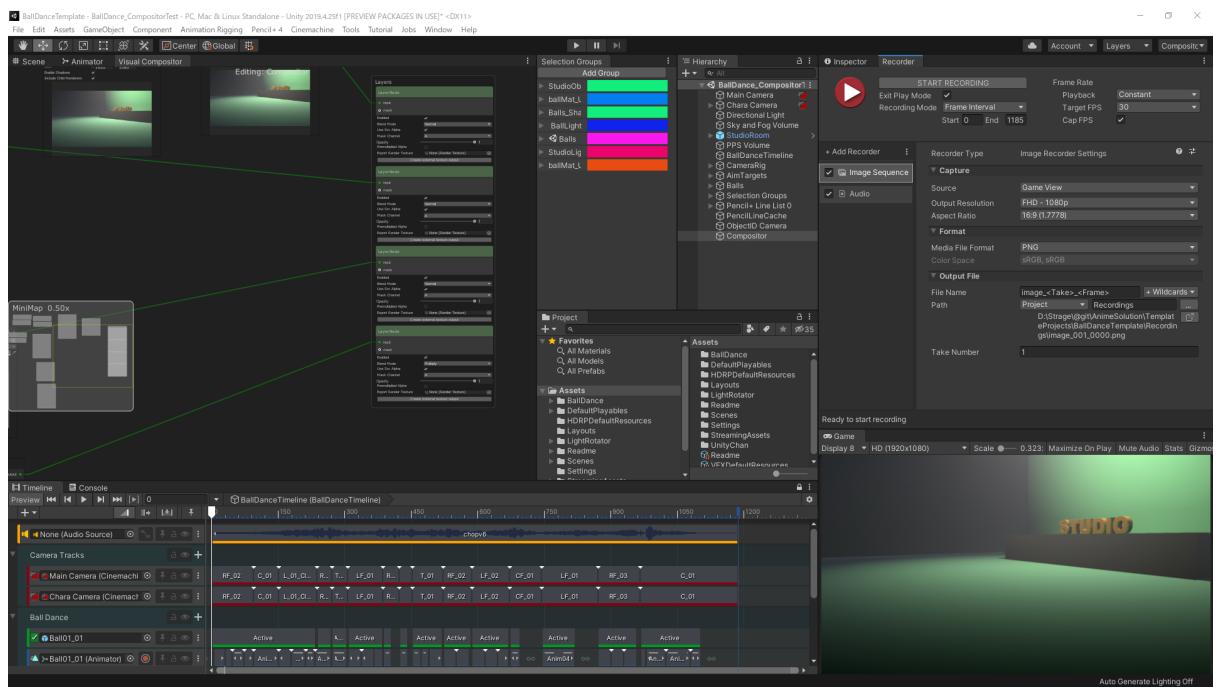
ヒエラルキーインドウよりキャプチャをするTimelineを選択し、インスペクター内Playable DirectorのUpdate Methodが「Game Time」になっていることを確認します。



ウォームアップを考慮したキャプチャの手順

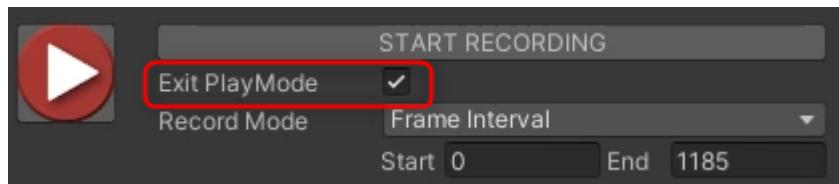
Unityは元々ゲームエンジンですので、必要に応じたアセットをメモリにロードします。ゲーム中では、マシンスペックや処理の重さに応じてフレームレートを変化させることで、ユーザーのゲームプレイ体験を損なわない設計になっていますが、映像向けに一定の更新フレームで画面をキャプチャするような場合には、事前にメモリに全てのアセットをロードしてしまう操作のほうが、不要なミスを防げます。

ここでは、映像用に固定フレームレートで画像をキャプチャする際に、Timeline上でオーディオトラックとの同期も含めたタイミング設計が正確に反映するように、十分なウォームアップタイムを取るUnity Recorderの操作法を説明します。



Unity RecorderとCompositor、ゲームビューなど、キャプチャ状況を確認するのに必要なビューを開き、以下の手順で操作します。

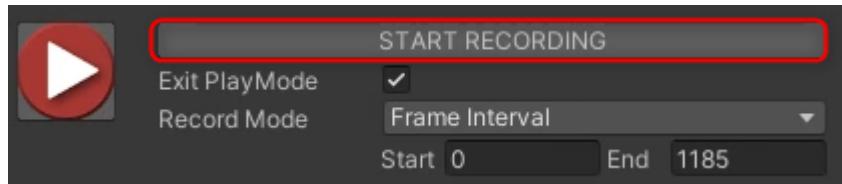
1. Recorderウィンドウの「Exit PlayMode」がチェックオンであることを確認します。



2. Unityエディタの「Pause」ボタンを押します。エディタの状態がミュートになります。

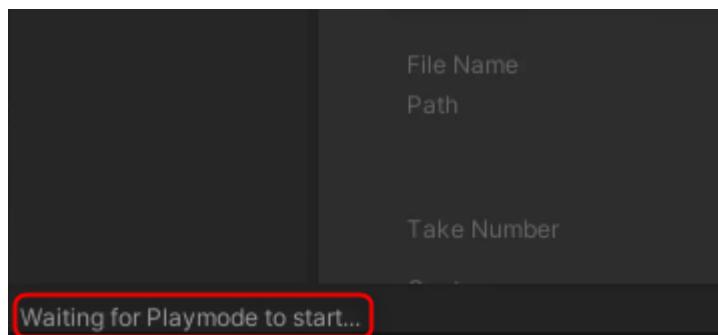


- Recorderウィンドウの「Start Recording」ボタンを押します。



⇒エディタのポーズボタンが押されているので、キャプチャは一時停止状態になります。

Recorderウィンドウ右下のメッセージ表示に、「Waiting for Playmode to start...」と表示されます。



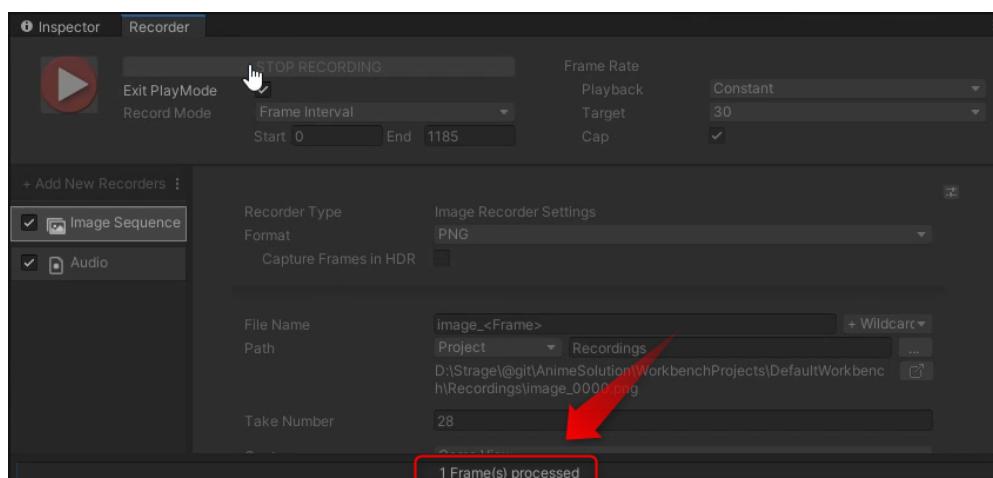
- 一時停止中にVisual Compositorウィンドウもしくはゲームビューが更新されます。

(多くの場合、プレビューが一瞬ブラックアウトして、一番最初のフレームに切り替わります。)

この間に必要なアセットがロードされます。

長くても1~数秒程度の、この時間をウォームアップタイムと呼びますが、ウォームアップタイムの間に、Unityが音声系アセットなどサイズが大きめのアセットを全てメモリ上にロードしてしまいます。

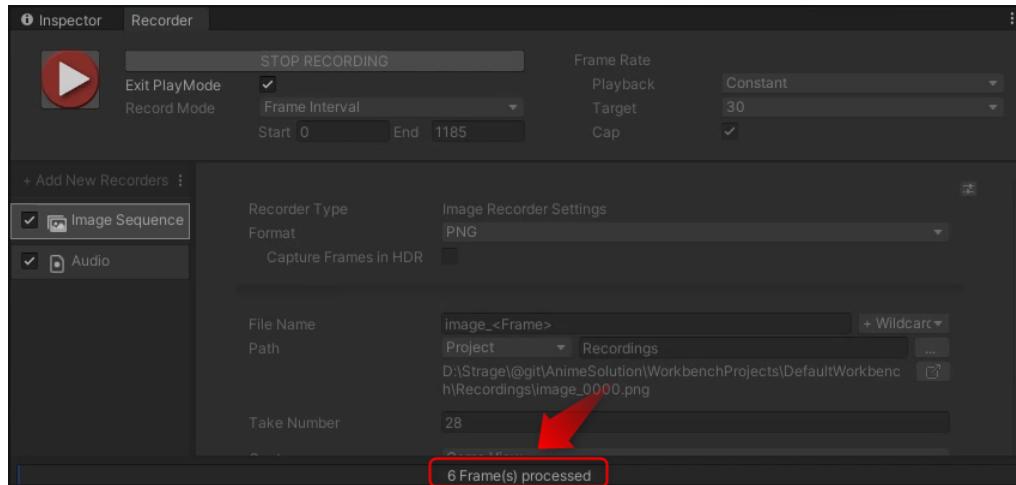
ウォームアップタイムが終了すると、Recorderウィンドウの下部に「1Frame(s) processed」というメッセージ表示が出て、1フレーム目の収録が終わったことが表示されますので、確認するとよいでしょう。



5. 再びUnityエディタの「Pause」ボタンを押して、Recorderの一時停止状態を解除します。レコーディングが開始されます。



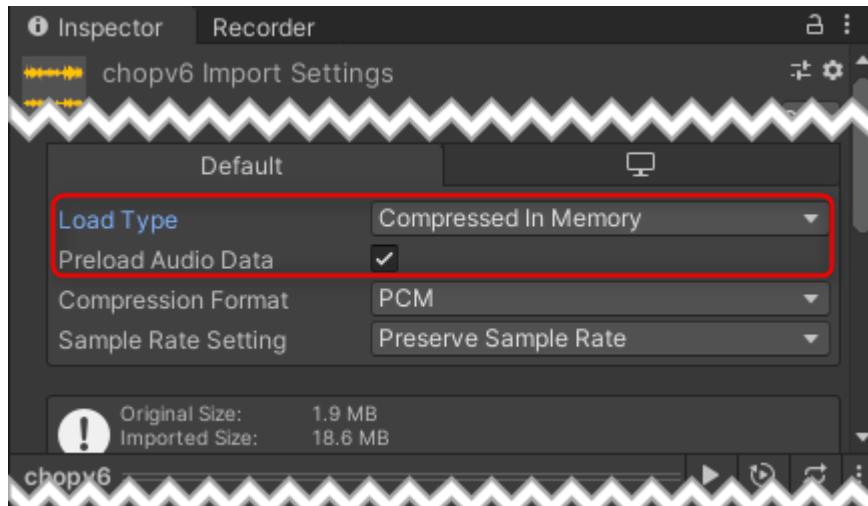
Recorderウィンドウの「Frame(s) processed」メッセージ表示が進行していきます。



キャプチャが無事開始されても、Timeline再生ヘッドの位置がしばらくの間、初期位置から更新されないこともあります、すぐに動き始めますので気にしないで構いません。

6. 「Exit PlayMode」がオンになっていれば、キャプチャのレコーディングが終了次第、プレイモードも停止します。
以上でキャプチャは終了です。

Tips: 音声アセットを以上の手順でメモリにロードするためには、Timeline上のオーディオクリップに配置する音声アセットは、以下ののような設定にしておくことを推奨します。

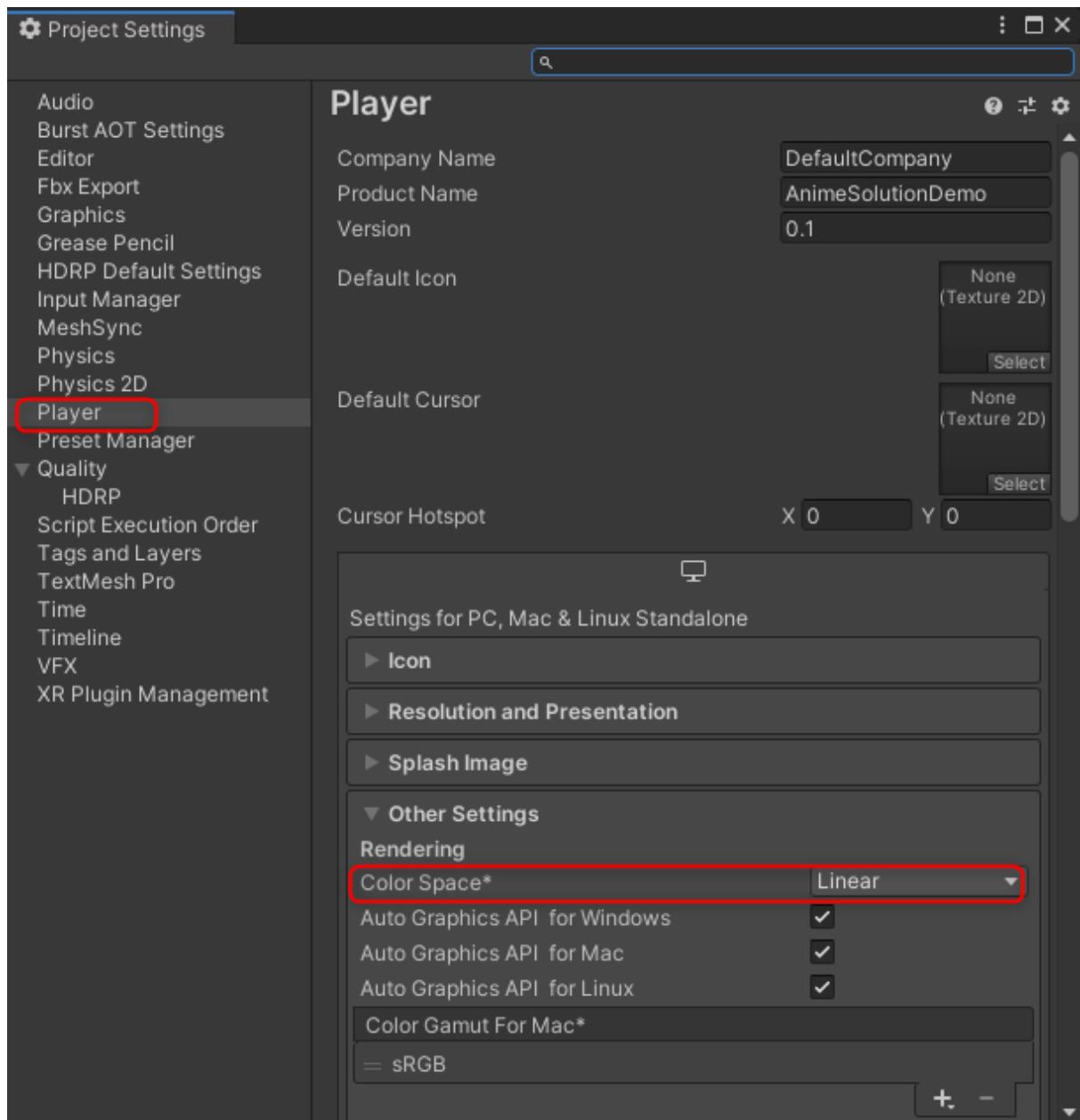


参考: プロジェクトのカラースペースについて

本プロジェクトのように、HDRPでプロジェクトをスタートした場合、カラースペースは最初から「リニア設定」になっていますので、特に気にすることはありませんが、Unityの従来のビュートインパインのようなレガシーパイプラインで本ドキュメントで解説をしたような映像編集を行う場合、プロジェクトのデフォルト設定のカラースペースを必ず確認するようしてください。
多くの場合、映像編集は、ガンマスペースよりもリニアスペースで編集するほうが良い結果が得られます。

カラースペースの確認は、メニューバーよりEditor > Project Settingsを辿って、Project Settings ウィンドウを開き、Player項目の中に入ります。

- HDRPの場合(Unity 2019.4.x)



- 従来のビルドインパイプラインの場合(Unity 2019.4.x)

