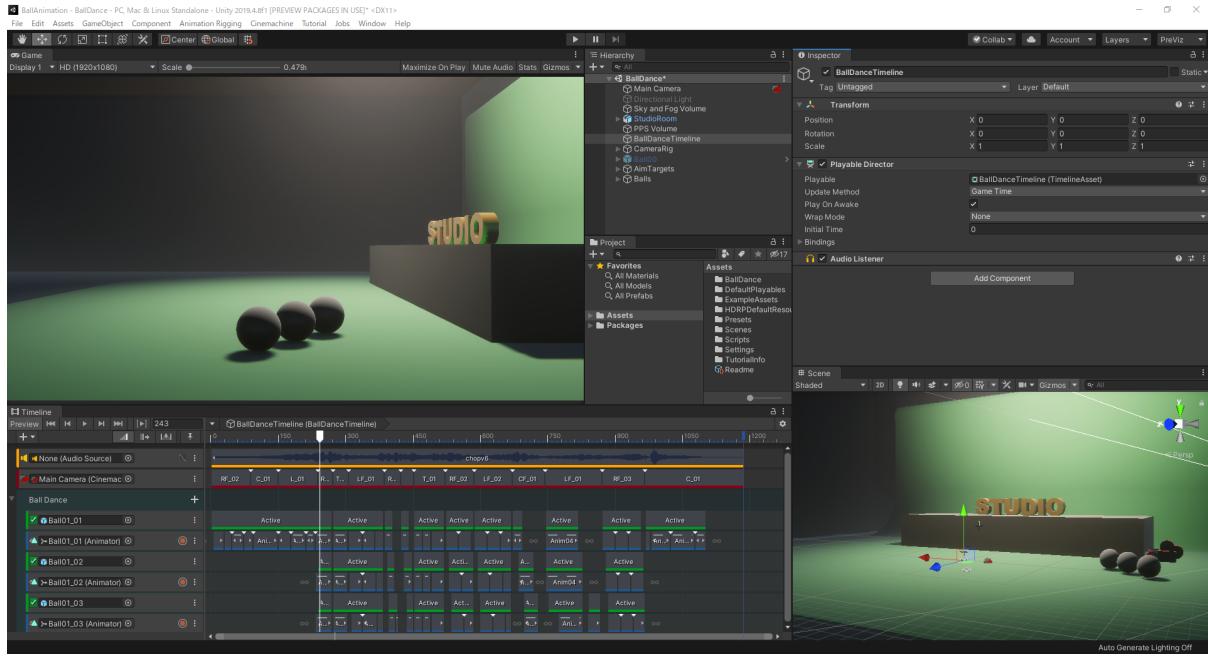


Ball Dance/PreViz・Cinemachine セットアップ

2021/04/08 N.Kobayashi/UTJ



はじめに

- 本ドキュメントは、AnimeSolution/WorkbenchProjects/DefaultWorkbenchプロジェクトに含まれる、BallDanceのセットアップ解説です。
- 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「プレビズ」作業における、Cinemachineによる作業を解説します。
- Cinemachineとは、Unityに搭載されているバーチャルカメラシステムのことです。Cinemachineを使うことで、Unity上で実際にアニメーションを再生しつつ、リアルタイム性を活かしたショットハンティングが可能になります。

Unityを使った映像編集のポイント

リアルタイム3DエンジンのUnityを使って、映像編集をするにあたりポイントとなるのは、「リアルタイムを活用して、トライ＆エラーの工程をいかに楽にするか？」にあります。トライ＆エラーの工程を効率的に繰り返すことは、映像作品のクオリティアップに繋がるからです。

Unityのリアルタイム性は、以下の4つの機能を使うことで、さらに強化されます。これらの機能は、AnimeTool Boxに最初からパッケージングされているので、すぐに使うことが可能です。

- 映像向け3DアニメーションデータをUnityにインポート / **FBX**ワークフローと **MeshSync/SceneCache**および**Alembic**ワークフロー

2. Cinemachineを使ったショットハンティング / **Framing Transposer**
3. Timeline上で複数のショットカメラからの切り替えタイミングの設計 / **Cinemachine Virtual Camera Track**
4. シーンに配置された3Dアセットを非破壊のまま、2Dレイヤー編集に持ち込む / **Visual Compositor**

特に4番目の項目に関しては、別ドキュメントの『Ball Dance/Visual Compositor セットアップ』でさらに詳しく説明します。

Ball Danceプロジェクトの場合、4番目の**Visual Compositor**による編集過程で、それまでの**HDRP**による**PBR (Physical Based Rendering : フィジカルベースドレンダリング)**的な絵作りを、リアルタイムでセルルックに変換します。セルルック変換をする際に、**Unity**用の高品質なセルレンダー-UTSの他、業界標準の**Pencil+ 4 Line for Unity**を活用する方法を紹介します。

セルルックについて

「セルルック」3DCGは、日本で人気のあるセルルック(ベタ塗りカラーとトゥーンラインの組み合わせ)風に3DCGのルックを仕上げる手法です。

プリレンダーの制作工程の場合、セルルック3DCGの多くは、直接セルルックを3Dツールから出力するのではなく、3Dツールからレンダリングした様々な画像マップを2Dコンポジッター上でレイヤー編集することで、最終的な絵作りをします。その場合、2Dコンポジッター上で3Dアセット由来のデータを再修正しようとすると、かなりの手戻りをするか、2Dコンポジッター上で各コマ単位で修正をしなければならなくなります。

工程上の手戻りが発生する一番の理由は、プリレンダーによるセルルック3DCGの制作ラインでは、3Dアセットはアセットのまま使われる訳ではなく、3Dアセットのもつ様々な情報を2D画像マップとして出力して使っている点にあります。つまりプリレンダーによるセルルック3DCGの制作工程は、後の2Dレイヤー編集工程のために3Dアセットの破壊編集を行っていると言ってもよいでしょう。

Unityを使いリアルタイムにセルルックを生成する場合、リアルタイムの特性を活かし、絵作りの面でトライアンドエラーの高速な繰り返しをすることができます。その結果、プリレンダーによるセルルック3DCGラインでは2Dコンポジッター上で行っていた調整の多くを、**Unity**上でリアルタイムに調整することが可能になります。これは、直接3Dアセットを非破壊のまま、セルルック3DCGでの絵作りに必要な調整プロセスを**Unity**上で行えることを意味しています。さらに**Anime Toolbox**に搭載されている**Visual Compositor**を使うことで、より2Dレイヤーに近い編集作業も**Unity**上で3Dアセットのままで行えるようになります。

アニメーションをインポートするワークフロー

Unityに映像用リソースからアニメーションデータをインポートするには、大きく2つの方法があります。

1. FBXワークフロー

FBXワークフローは、AutodeskのFBX形式のファイルで3DモデルやアニメーションデータをUnityにインポートするワークフローで、ゲーム開発でも標準で使われる最も一般的な方法です。

ゲーム開発で最も使われるワークフローですので、リアルタイムIKやフルボディIK、さらにはUnity上でのダイナミクスクロスやスプリングジョイントなどのリアルタイムダイナミクスシミュレーションや、物理演算Physixなどと組み合わせて、様々なインタラクションを伴うアニメーションを作成できるワークフローにもなっています。Unityのゲームエンジンとしての性能を最大限に利用したい場合には、FBXワークフローを選ぶことをお薦めします。

DCCツールからUnityへFBX経由でアニメーションデータをエクスポートする時には、全てのデータをスケルトンにベイクしてから転送します。特にDCC上でリグコントローラに打たれているIKキーなどは、一度シミュレーションベイクでFKキーとしてスケルトンにベイクされている必要があります。ベイクされたアニメーションデータは、Unity上で再度キー最適化などをサイズを圧縮することも可能です。ただし、ゲーム用途ではサイズの圧縮は有効ですが、映像用途にはオリジナルのままのほうが良いでしょう。

FBXを経由してUnityにインポートされたアニメーションデータは、Unity内部では以下の3つの形式のいずれかで扱うことになります。

1. 最も単純だが、再生パフォーマンスは良い。アニメーションデータをそのままゲームオブジェクトに割り当てる、レガシーアニメーション
2. 人型リグに限定されるが、アニメーションデータがリターゲットできるので、汎用的に使用できる。アニメーターコンポーネント経由で利用する、**Mecanim/Humanoid**
3. 自由な形状のリグに対応する代わりにアニメーションのリターゲットは難しく、リグに対して専用のアニメーションデータとなる。アニメーターコンポーネント経由で利用する、**Mecanim/Generic**

インポートしたアニメーションデータを、Timelineのトラック上でブレンドして使いたい場合、アニメーターコンポーネントが必須になりますので、Mecanim/HumanoidかMecanim/Genericのいずれかを使うことになります。

FBXを経由してアニメーションをUnityにインポートする場合、以下の制約があります。これらは映像向け用途ではしばしば使われるものですが、FBX経由で利用する場合には、使用しないでください。

- スキンドメッシュデフォーマ以外の様々なデフォーマ。特にFFD(フリーформデフォーマ)など。スキンウェイト以外の方法のデフォーマは、Unityでは使えません。
- 各ローカル軸に対して別々の値をとるスケールアニメーションを含むデータ。一般的にゲームエンジンではシアー変換が使えませんので、各ローカル軸に対して別々の値をとるスケールアニメーションが入っていると、Unity上でローカル回転

をした場合、DCCツール上とでは見た目の結果が変わってしまいます。スケール値を用いる時には、全てのローカル軸に対して同じ値を入れるようにしてください。

FBXワークフローの場合、フェイシャル表現はブレンドシェイプを使うのが主流です。その場合、フェイシャルリグはブレンドシェイプのターゲットを作成するのに使います。

2. MeshSync/SceneCacheおよびAlembicワークフロー

MeshSync/SceneCacheおよびAlembicワークフローは、DCCツール上でアニメーション変形済みのモデル頂点データをキャッシュとしてUnityにインポートする方法です。従つて、原理上DCCツール上で実現できる全ての頂点アニメーションをUnityにインポートすることが可能です。この場合、FFDを含むDCCツール上でしか使用できないようなデフォーマや、複雑なリグアニメーションの結果なども、Unity上で再現できます。またMeshSyncテクノロジーを使うことで、マテリアル情報もUnityに送りつつ、DCCツール上で調整した結果をUnity上ですぐにリアルタイムに確認することも可能となっています。

DCCツール上でのアニメーションデータの完全再現という意味では、MeshSync/SceneCacheおよびAlembicワークフローは大変に強力なソリューションですが、FBXワークフローと比べると以下のようないくつかの制約があります。

- 頂点キャッシュデータなので、Unity上で編集することができません。また、Unity上でプライマリアニメーションにあわせて、リアルタイムにセカンダリアニメーションを生成するコンポーネントを付けることもできないので、セカンダリアニメーションはDCCツール上で事前に付けることになります。
- アニメーションの修正や調整は、DCCツール上で行います。Unity上でリアルタイムにその結果を確認するためには、MeshSyncを使います。
- 頂点キャッシュデータですので、同じアニメーションデータならばFBXデータよりも大きなものになります。
- 再生できるプラットフォームは主にPCのみとなります。

3. 両者の組み合わせ

最後に、FBXワークフローとAlembicワークフローを組み合わせる方法について簡単に解説します。

映像目的での使用の場合、あるキャラクターのボディ部分のアニメーションをFBXデータで、フェイシャルや複雑なクロスシミュレーションの結果をAlembicデータで共にUnityにインポートし、Timelineのトラック上で両者をシンクロさせて再生させるという手法もあります。これは、表現が複雑でリグも重いフェイシャルアニメーションと、ボディについているアクセサリなどの揺れ物のセカンダリアニメーションの多くはUnity上で付けたい場合などに、よく使われる手法です。

Unity上では、Alembicデータは固定フレームではなく可変フレームで再生することができる、FBXでのアニメーションと同期させて動かすことも可能となっています。

複雑なフェイシャルリグを多用する映像向けには、一考する価値のある方式といえます。

『Ball Dance』で採用したアニメーションワークフロー

今回、『Ball Dance』という名前で、AnimeTool Boxを使ったサンプルプロジェクトを作つてみることにしました。イメージ的には、「軽快な音楽に合わせて、ボールが子犬のように跳ね回るショート作品」を作ることにします。作成にあたり、以下のようなポイントを押さえます。

絵コンテで決め打ちの絵作り＆アニメーション素材の作成を行うのではなく、Unity上でトライアンドエラーを繰り返しながら、作品をブラッシュアップできるものにする。

上記の目的のために、以下のような仕様でアニメーション素材を作成することにします。

1. ボールにキャラクター性のある動きを入れるために、スクウォッシュやストレッチがしやすいリグを設定する。
2. Unity上で複数のボールのモデルに、アニメーションデータを割り当て、Timeline上で楽曲に合わせつつ、ボールの演技を作成し、隨時タイミングを調整できるようにする。

仕様1のために大切なことは、ボールにつけるリグのピボット位置です。ポジションとスケールに関しては、ボールオブジェクトの接地面基準にピボットを設定し、回転に関しては、ボールオブジェクトの中心にピボットを設定すると、スクウォッシュやストレッチを設計しやすい、キャラクター的な動きができるリグができます。

仕様2のために大切なことは、特定のフレームで必ず原点(0,0,0)を通過するアニメーションを作成することです。各アニメーションデータに、必ず原点を通過するフレームを設定しておき、同時にTimeline上でそのフレームを基準にアニメーションクリップの位置や回転のオフセット、さらには原点を通過するタイミングを調整することで、複雑なアニメーション挙動の組み合わせを設計できるようになります。

ボールに割り当てるアニメーションデータは、Mayaで作成します。今回、スクウォッシュやストレッチなどの挙動は、スケールアニメーションで作りますので、Unityへのアニメーションデータのインポートは、FBXワークフローを使い、Mecanim/Generic形式でインポートすることにします。

もしFFDやラティスのようなデフォーマを使って、複雑なスクウォッシュやストレッチの挙動を作った場合には、MeshSync/SceneCacheおよびAlembicワークフローを採用するか、頂点アニメーションの部分にブレンドシェイプを活用することを検討する必要があるかもしれません。

Mecanim/Generic形式でアニメーションデータをUnityにインポートした場合、意図通りにアニメーションデータがエクスポート＆インポートできているかを確認するために、一度Unity上で再生して、Maya上と動きの違いがないか、確認することをお薦めします。

各アニメーションデータをMecanim/GenericでUnityにインポートすると、Timeline上で複数のアニメーションクリップを組み合わせたり、ブレンドすることで、複雑な挙動を作成することができるようになります。従って、Maya上で作成するアニメーションも、一連の演技のある長尺のアニメーションを一気に作るのではなく、Unity上で各アニメーションデータを組み合わせて、演技の構成がで

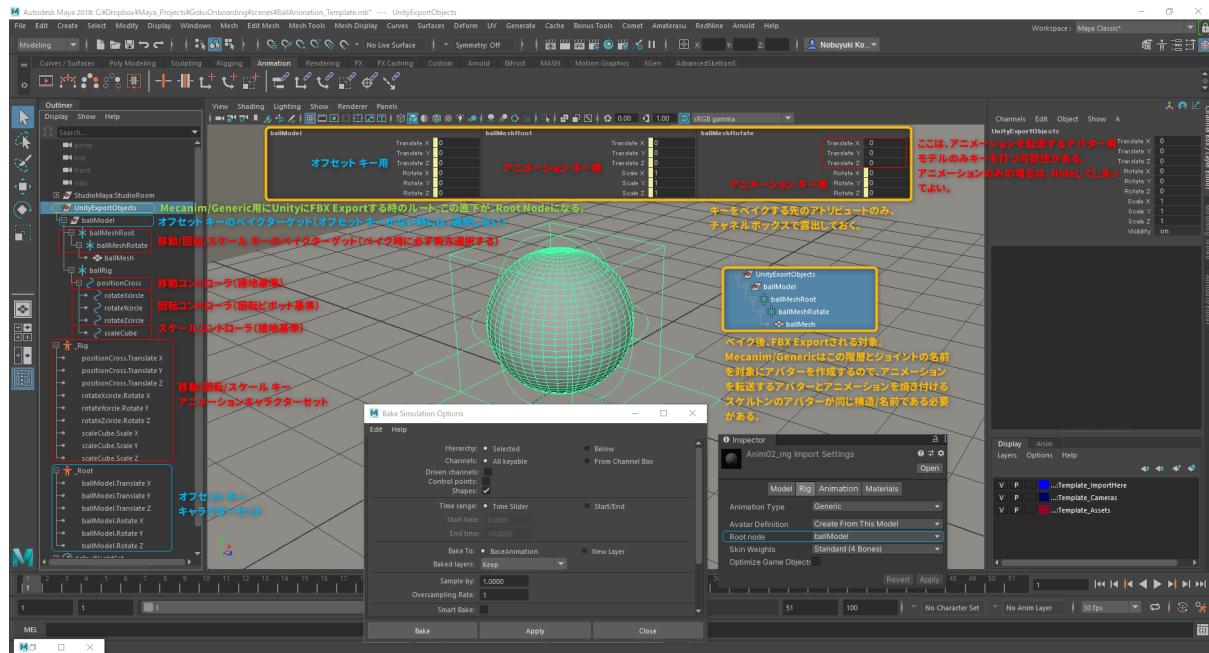
きるような汎用性のある短めのアニメーションパターンを揃える、という方針で作成することになります。

Mecanim/Generic用のリグ

Mecanim/Generic用のリグは、DCCツール上で作成します。今回はMayaで作成しましたが、Blenderなどでも可能です。

下の図は、Mayaでボールにリグを割り当てている様子です。MayaのOutlinerウィンドウを見てください。ballModel以下のジョイントが移動/回転/スケールのそれぞれのアニメーションキーをベイクするターゲットです。一方、ballRig以下のカーブオブジェクトが移動/回転/スケールの各コントローラとなります。アニメーションは、この各コントローラに対してキーを打つことになります。

Mecanim/Generic向けにアニメーションデータを作成する場合、いくつか注意すべき点があります。



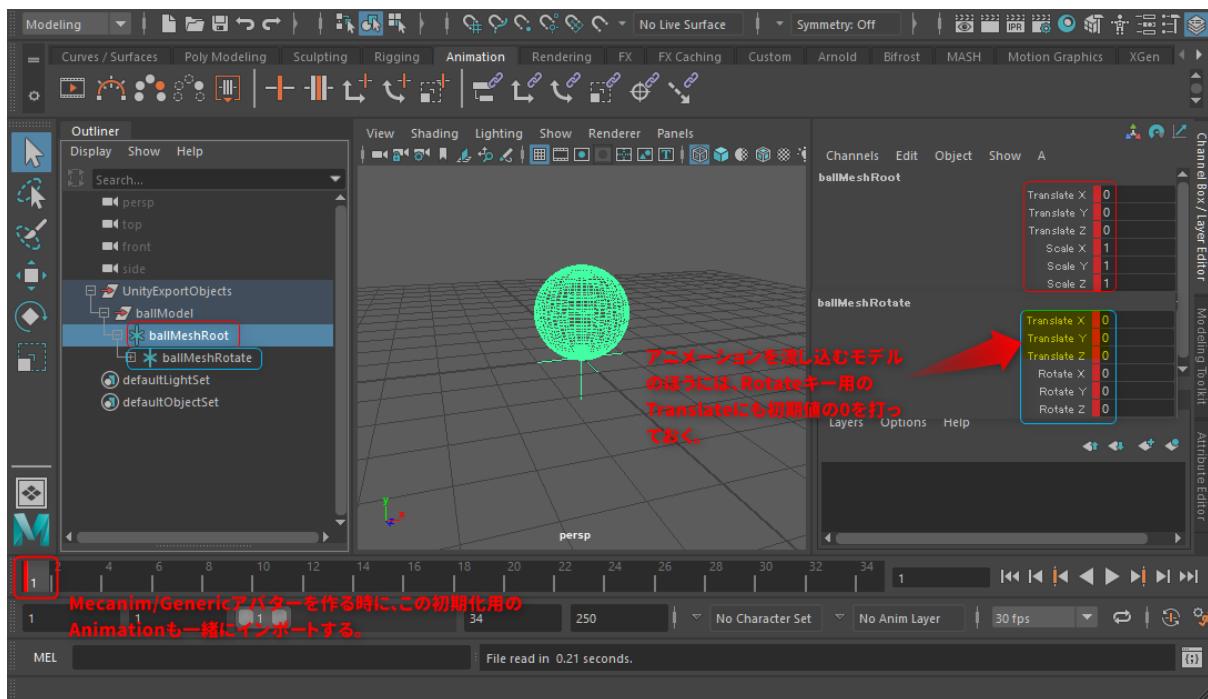
- Mecanim/GenericのRoot nodeとなるジョイントの親の位置に出力用のNULL(図の例だと、UnityExportObjects)を作り、FBXはそのNULLをルートとしてExportする。
※Unityにインポートした時、このNULLはRoot nodeの候補とはならない。
- Mecanim/Genericでアバターを作成して、アニメーションを流し込むためには、アバター作成の対象となるジョイントの階層と名前を、モデル/アバター側とアニメーションをベイクするスケルトン側で必ず合わせる。
※両者を合わせないと動かない可能性がある。
- 今回のように、移動/回転/スケールの全てにキーを打つ可能性があり、かつ回転とスケールとでピボットの位置が違っている場合には、ジョイントの階層の順番に注意する。今回の例ならば、まず回転してからスケールを適用するようにする。

※図の場合ならば、回転キーを打たれるballMeshRotateは、スケールキーを打たれるballMeshRootの下に来るようになります。

Mecanim/Generic用のモデル/アバター

Mecanim/Generic用のモデル/アバターは、リグを設定したMayaのシーンから作成します。

リグを設定したシーンに置かれているボールのメッシュ階層から、ballRig以下のコントローラを削除したものをFBXでエクスポートし、Unityにインポートすると、Mecanim/Generic用のモデル/アバターとなります。こうすることで、アニメーションデータをベイクしたスケルトンとアニメーションデータを流し込むアバターとで、ジョイント名や階層の不一致が発生するのを防ぐことができます。



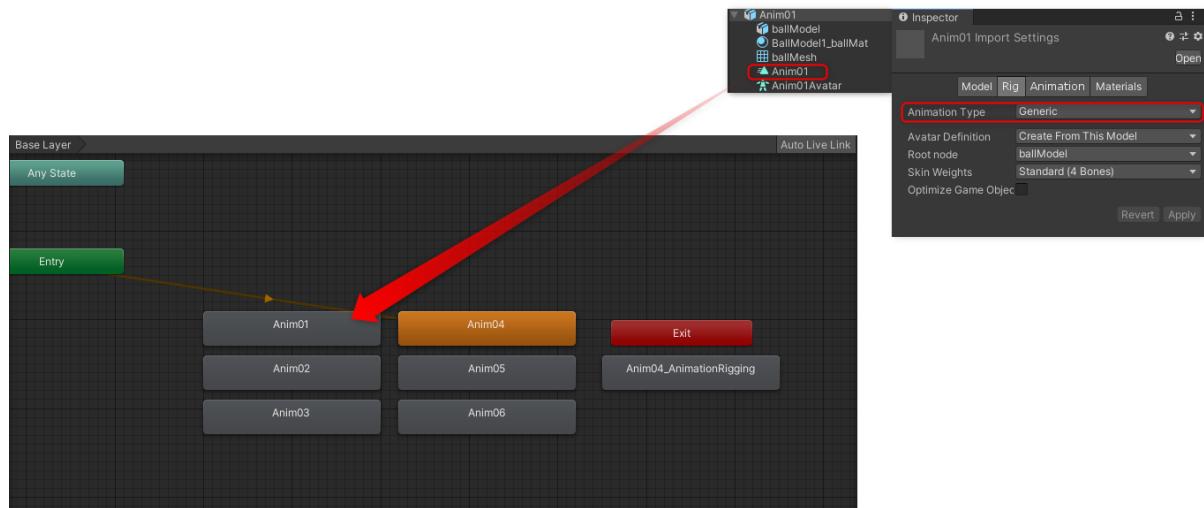
DCCツールからFBX出力する際のTips

ここでMecanim/Generic用アバターを正確に作るためのTipsとして、「**Mecanim/Generic用にモデル/アバターをFBX出力する際に、1フレーム目にスケルトンに初期値をキーとして打っておく**」ことを推奨します。これは、UnityがMecanim/Genericアバターを作成する時に利用する、初期ポーズにあたるものです。Mecanim/Humanoidの場合なら、Tスタンスがそれになります。

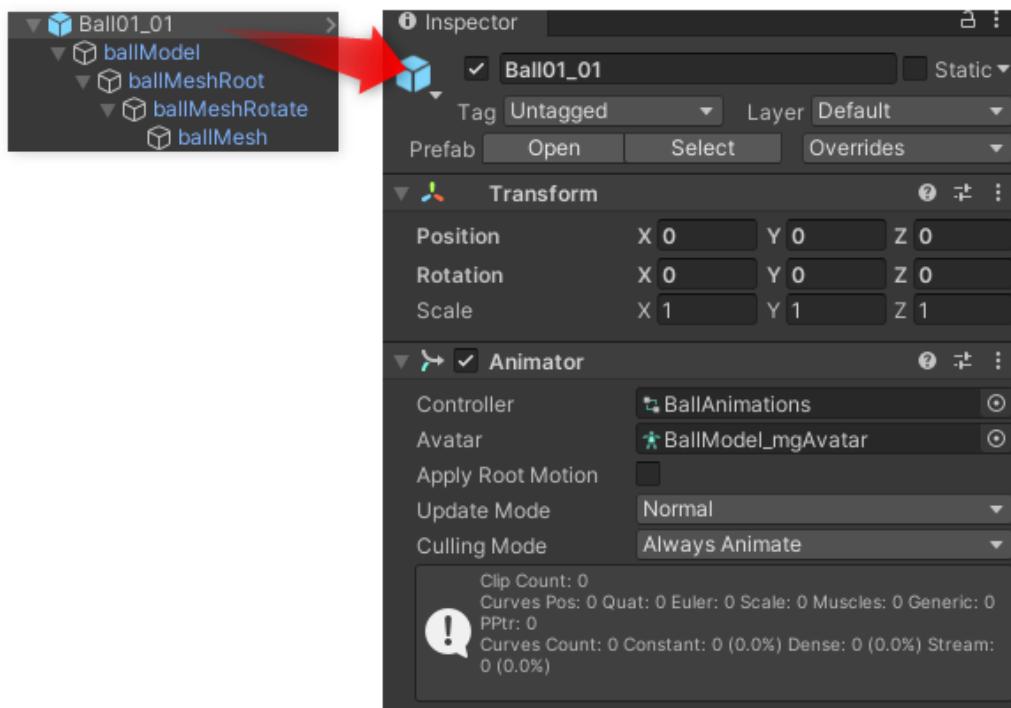
このように、アバターを生成するためにUnityにインポートするFBXファイルの1フレーム目に初期ポーズをキーとして打っておくことで、正確なアバターが生成されます。

以上のように、アニメーションを作成するリグを含むシーンと、モデル/アバターをエクスポートするためのシーンを分けておくことで、アセットの管理が容易になります。

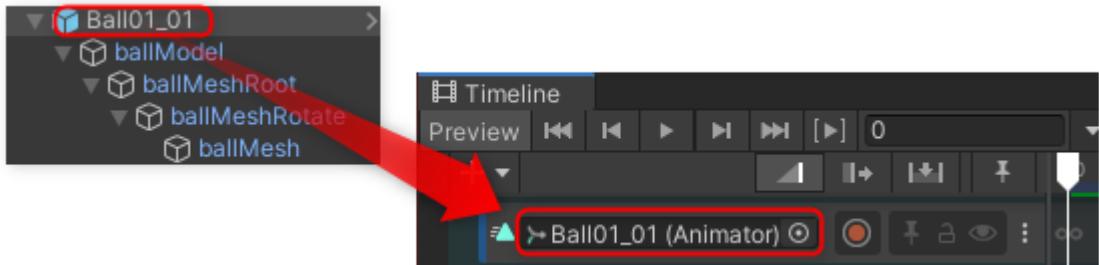
Timeline上で、Mecanim/Generic用モデルを使う



Mecanim/Generic形式でUnityにインポートしたアニメーションクリップは、Animator上で管理します。

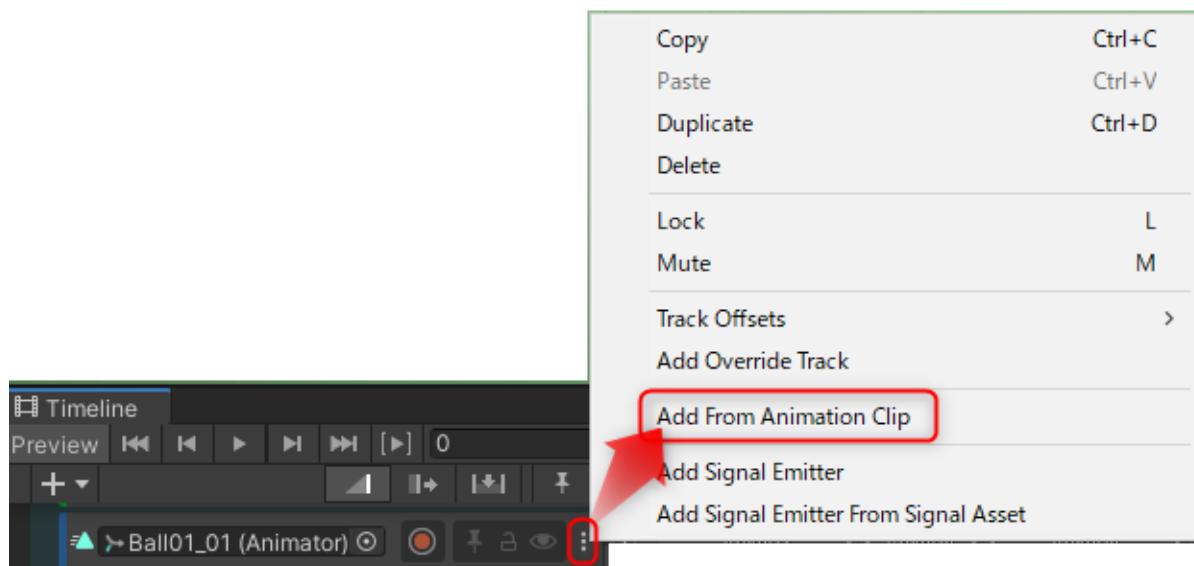


続いて、これらのアニメーションクリップを登録したAnimatorをMecanim/Generic形式のアバターを持つボールモデルに適用します。

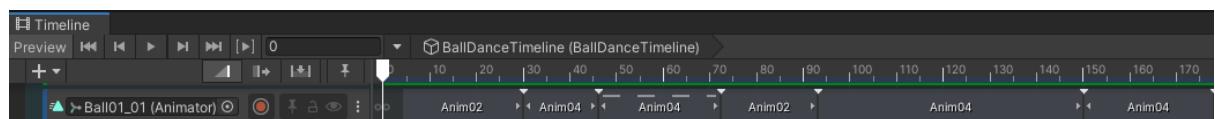


Timeline上にAnimation Trackを作成し、そこにAnimatorを適用したボールモデル(この場合、Ball01_01)をバインドします。これでTimeline上でAnimatorを適用したボールモデルを使用することができるようになります。

Timeline上でアニメーションクリップを切り替える



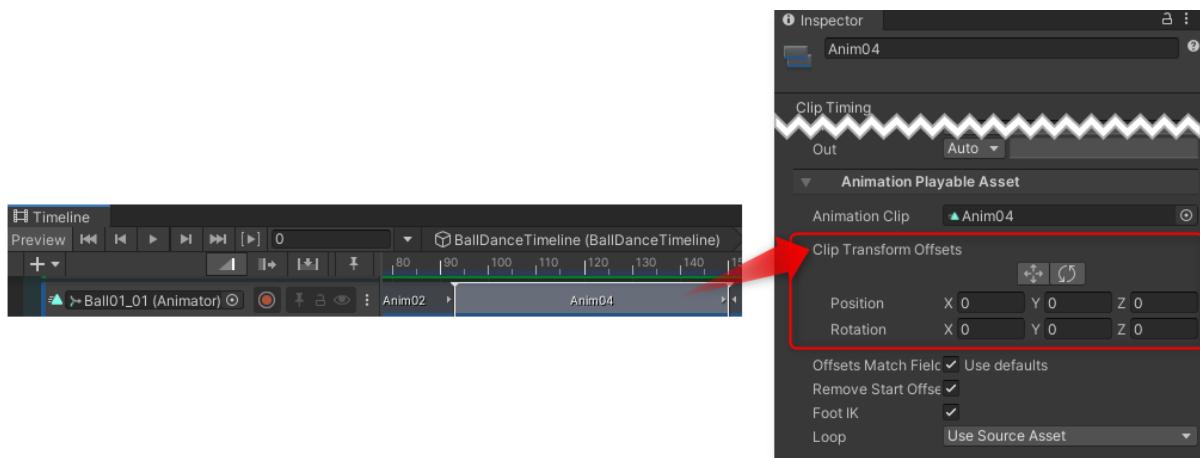
Animatorに登録されているアニメーションクリップをTimelineから呼び出すには、Animation Trackのコンテキストメニューから「Add From Animation Clip」を実行し、配置したいアニメーションクリップを選択します。



Animation Trackに配置されたアニメーションクリップは、長さを調整したり、ブレンドをすることができます。

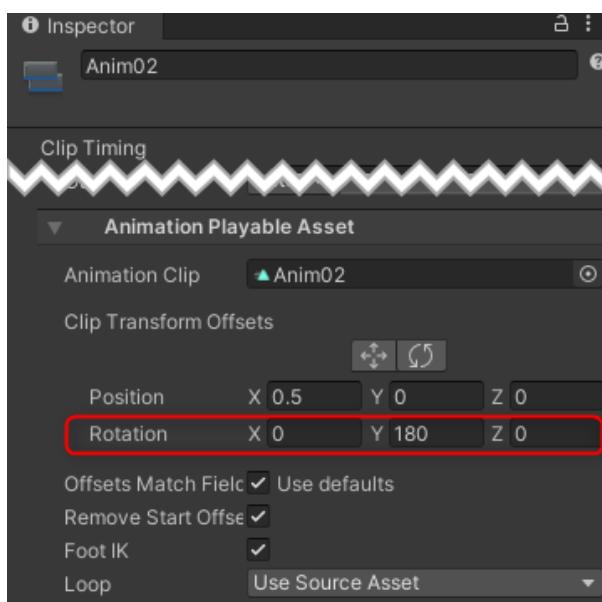
アニメーション再生位置をオフセットして、クミを演出する

アニメーションクリップのグローバルな再生位置をオフセットすることで、Animatorに登録されているアニメーションクリップをより複雑に再生することができます。再生位置のオフセットには、位置と回転が設定できます。

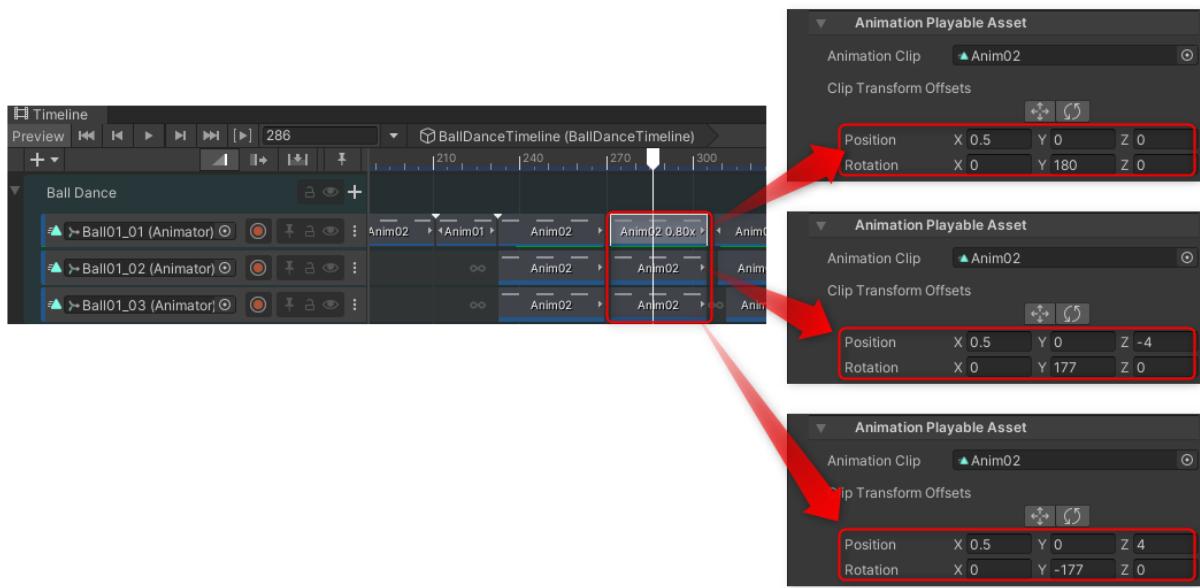


オフセットしたいアニメーションクリップをTimeline上のAnimation Trackから選択します。インスペクターウィンドウ上に、選択したアニメーションクリップのプロパティが表示されます。Animation Playable Assetsドロップダウンを展開すると、Clip Transform Offsetsという項目が現れます。このPosition(位置)とRotation(回転)を調整することで、アニメーションクリップのグローバルな再生位置がオフセットできます。

もしこのClip Transform Offsetsをいじらない場合、アニメーションクリップは作成した時に設計したように、特定のフレームで原点を必ず通ります。Clip Transform Offsetsを変化させることは、特定のフレームで通っていた「原点」の座標をオフセットする(ずらす)ことと同じです。実際に、クリップ上でオフセット値を変えてみて、どのようにアニメーションが再生されるかを確認してみると、理解が深まります。



上の例の場合、「Y軸周りに180度回転させているオフセット」が最も目立つ変化となります。アニメーションクリップAnim02は、「ステージに向かって、画面右から入ってきたボールが、画面左へと転がっていくアニメーション」なので、このY軸周りの回転オフセットの結果、「ボールの進行方向が反対」になります。



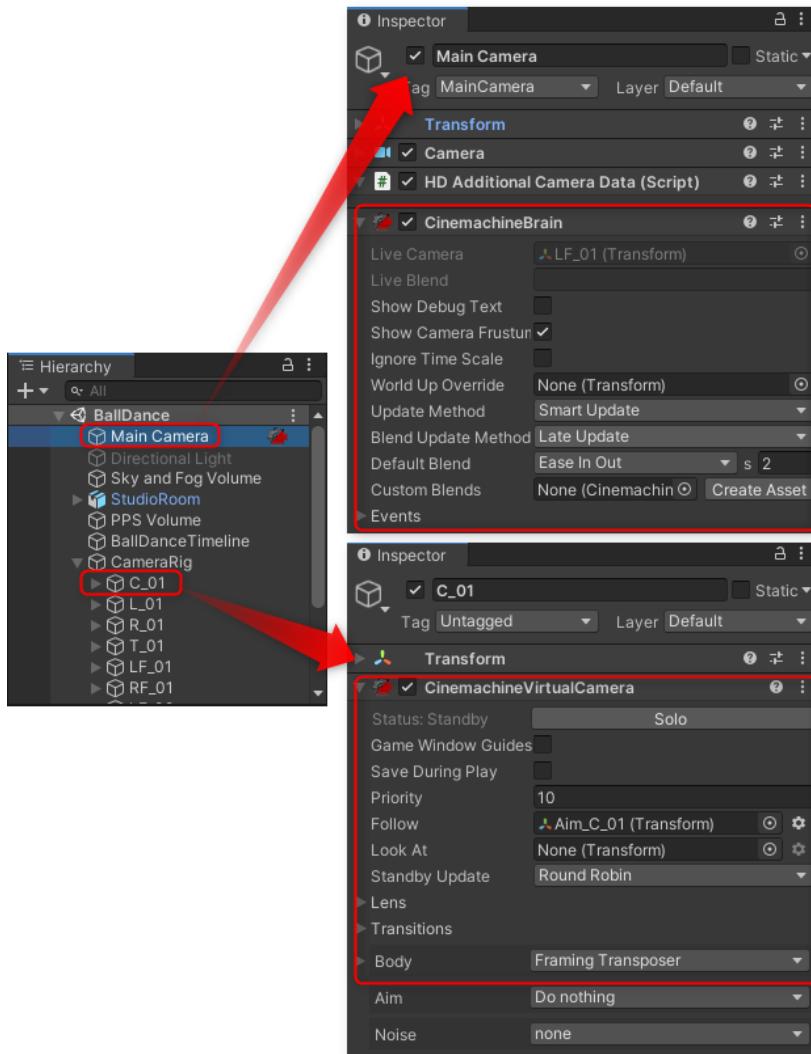
同じアニメーションクリップをTimeline上で並列に再生させても、Clip Transform Offsetsをトラック毎に変化させることで、同じタイミングでボールが移動する統一感を持たせながらも、ボール自体の軌道は異なるバリエーションを作ることが可能となります。上の例では、3つのボールが左から右に同じタイミングで転がりながら、末広がりに広がっていく挙動を設定しています。

これらを繰り返すことで、各ボールの挙動をクミとして演出することができます。

Cinemachineでショットハントをする

Cinemachineとは

冒頭で簡単に説明しましたが、Cinemachineとは、Unityに搭載されているバーチャルカメラシステムのことです。



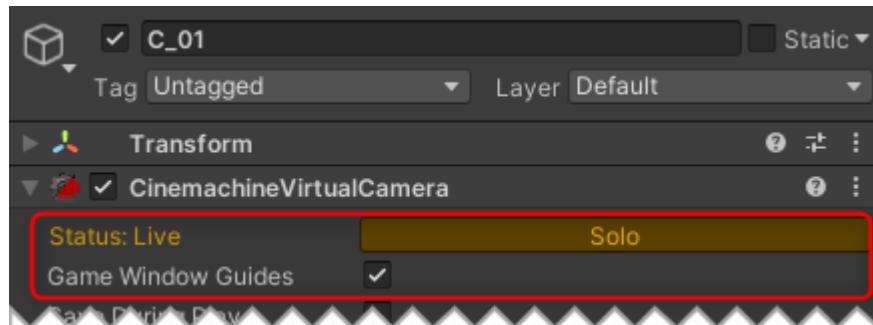
Cinemachineの基本構成は、メインカメラにアタッチされているCinemachineBrainコンポーネントと、各カメラリグを構成する「空のゲームオブジェクト（Mayaで言う、NULLオブジェクト）」にアタッチされているCinemachineVirtualCameraコンポーネントです。

CinemachineBrainがアタッチされているカメラのプロパティを、バーチャルカメラ側でオーバーライドすることで、簡単にハイパフォーマンスのカメラシステムを設計することができます。

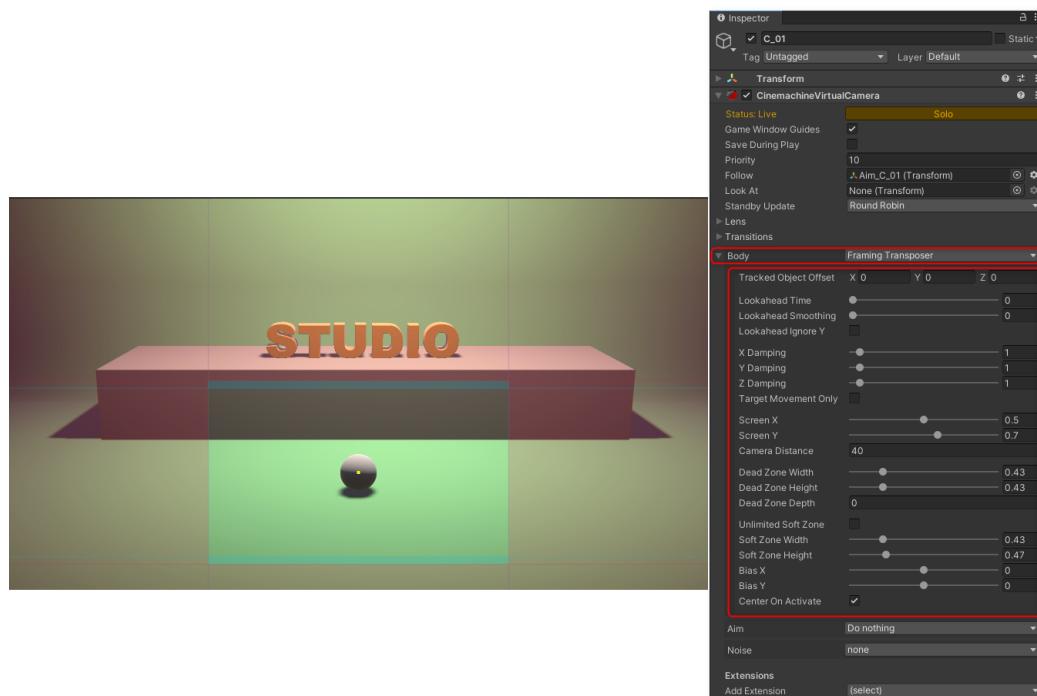
Framing Transposerを使うメリット

Cinemachine自体には、ゲームのリアルタイムカメラからカットシーンまで、様々な用途に使えるように多彩な機能が搭載されています。特に映像向けに使う場合には、その機能の中でも

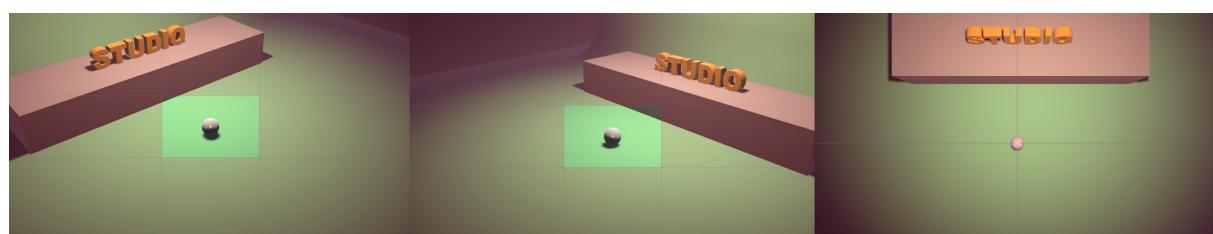
Framing Transposerという機能に注目するとよいでしょう。Framing Transposerは、CinemachineVirtualCameraコンポーネントから呼び出す機能のひとつです。インスペクターウィンドウよりCinemachineVirtualCameraコンポーネントのBodyドロップダウンから選択します。



さらにCinemachineVirtualCameraコンポーネントのStatus: LiveをSoloに、Camera Windows Guidesチェックボックスをオンにすると、下のような画面がゲームビューに表示されます。



これは、C_01というカメラリグから見た、シーンの2Dレイアウト(構図)となっています。



カメラリグを切り替えると、ゲームビューに表示されているレイアウトも変わります。

Framing Transposerを使うことで、あるカメラから常にどういう構図でシーンが見えていて欲しいかを設計することができます。もちろん設定次第で、特定の被写体と同じ構図のままフォローしたり、動く被写体とは関係なく構図をFIXすることも可能です。また、被写体が画面内を動く場合でも、奥行き方向へのカメラの微移動をカットするような設計も可能です。

特に**2D**作画を意識したアニメでは、レイアウト用紙に描かれる**2D**レイアウト設計が重要な意味を持ちます。**Framing Transposer**を活用することで、**3D**空間上に**2D**レイアウト的な構図設計を簡単に適用することができるようになるのです。

Ball Danceの作例では、こぎみよいカット切り替えを中心に複数のボールが織りなす動きを楽しむ映像を作りたいので、なるべくシーンの原点を中央に抑える、**FIX**気味のカメラリグを設計することにしますが、Framing Transposerを使いこなすことで、2Dパンなどのカメラ設計も作ることができます。それは本章の最後に説明をします。

新規カメラポジションをカメラリグに追加する

Cinemachine Virtual Cameraを使って、新規カメラポジションをカメラリグに追加してみましょう。ヒエラルキー上で、カメラリグの位置にCameraRig、シーン全体の原点にはAimTargetsという名前の空のゲームオブジェクト(Mayaで言うヌルオブジェクト)が配置されています。以下の手順で新規カメラポジションをCameraRigに追加します。

1. ヒエラルキー上でCameraRigを選択し、コンテキストメニューからCreate Emptyを選択する。CameraRig直下に新規にできた空のGameObjectをリネームし、C_02とする。
2. ヒエラルキー上でC_02を選択し、シーンビュー上でビューを操作する。
おおよそ「新規にこんなカメラ位置から見たい」というような位置まで、ビューを操作して移動する。
3. ヒエラルキー上でC_02が選択されたままなどを確認した上で、メニューバーより、GameObject > Align With Viewを実行し、C_02のTransformを現在、ビューを見ているカメラの位置と回転に移動する。
注:これはビュー操作位置にカメラ位置を合わせる方法ですが、ライトの位置をビュー操作位置に合わせたい場合などにも使えます。
4. C_02を選択したまま、インスペクターウィンドウのAdd Componentボタンを押し、Cinemachine Virtual Cameraコンポーネントをアタッチする。
5. インスペクターよりCinemachine Virtual CameraコンポーネントのBodyをFraming Transposerに変更する。Framing Transposerには、Followする対象が必要なので、固定カメラの場合、注視点ポイントとしてAimTargetsをFollowに割り当てる。
6. Status: LiveをSoloに、Game Window Guidesをオンにする。ゲームビューがC_02にアタッチされているバーチャルカメラに切り替わる。
7. Bodyとして指定されているFraming Transposerのプロパティより、Camera Distanceを調整し、被写体とカメラとの間の距離を設定する。原点を注視点としている場合、カメラの距

離を調整すればおおよそ最初に設計したシーンビューと同じようなレイアウトが得られる。

注:完全にシーンビューに合わせる場合には、Cinemachine Virtual CameraのField of Viewの値もシーンカメラに合わせる。

8. あとは、Screen XやScreen Yなどのパラメーターを調整することで、カメラのフレーミングを上下左右に調整できる。

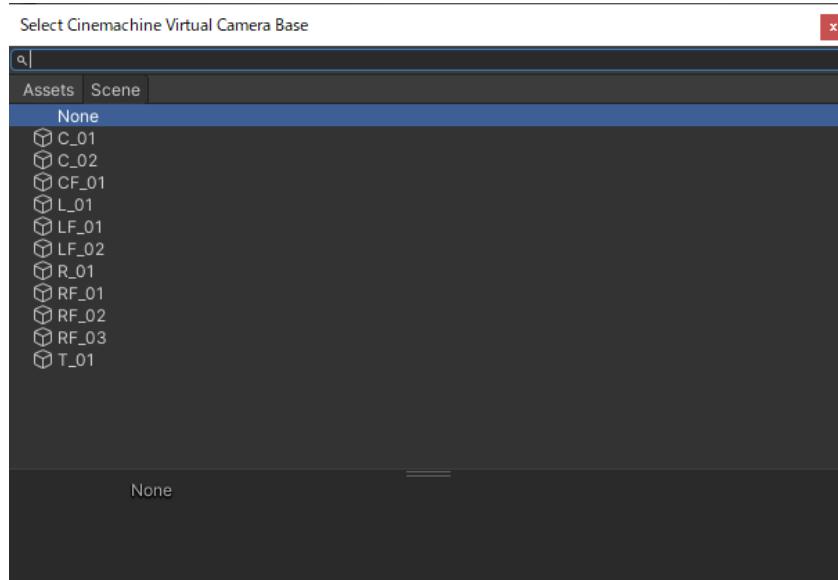
注:フレーミングの調整中は、Dead Zone Width/Height、Soft Zone Width/Heightは初期値のままで触らないか、全て0にしてしまうほうがよいでしょう。これらのプロパティは注視点が動く場合に、カメラのフレーミングが動き始める範囲を設定するのに用います。

TimelineにCinemachineバーチャルカメラを配置する

Timeline上でCinemachineバーチャルカメラを利用するには簡単です。

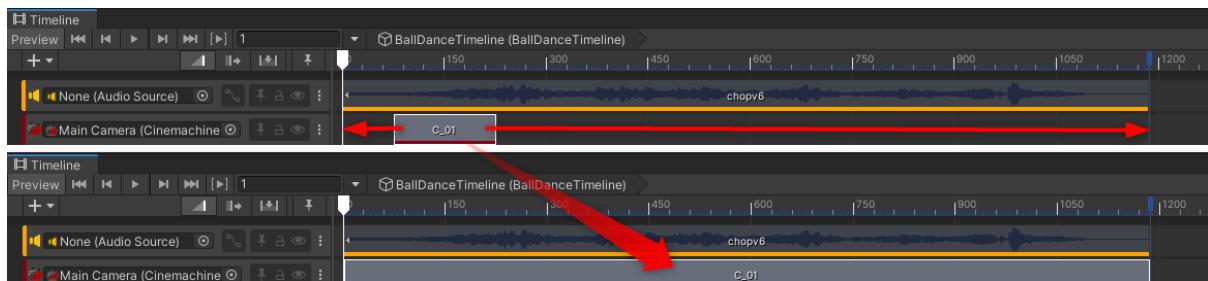
Timelineウィンドウの「+」ボタンを押して、Cinemachine Trackを追加します。Cinemachine TrackにはCinemachineBrainコンポーネントがアタッチされている、シーン内のカメラが選択できますので、対象したいカメラを選択します。

追加したCinemachine Trackを選択し、右クリックから呼び出されるコンテキストメニューより、「Add From Cinemachine Virtual Camera Base」を選択すると、現在シーン上で選択できるCinemachine Virtual Cameraの一覧ウィンドウがでてきますので、その中から使いたいカメラポイントを選択します。



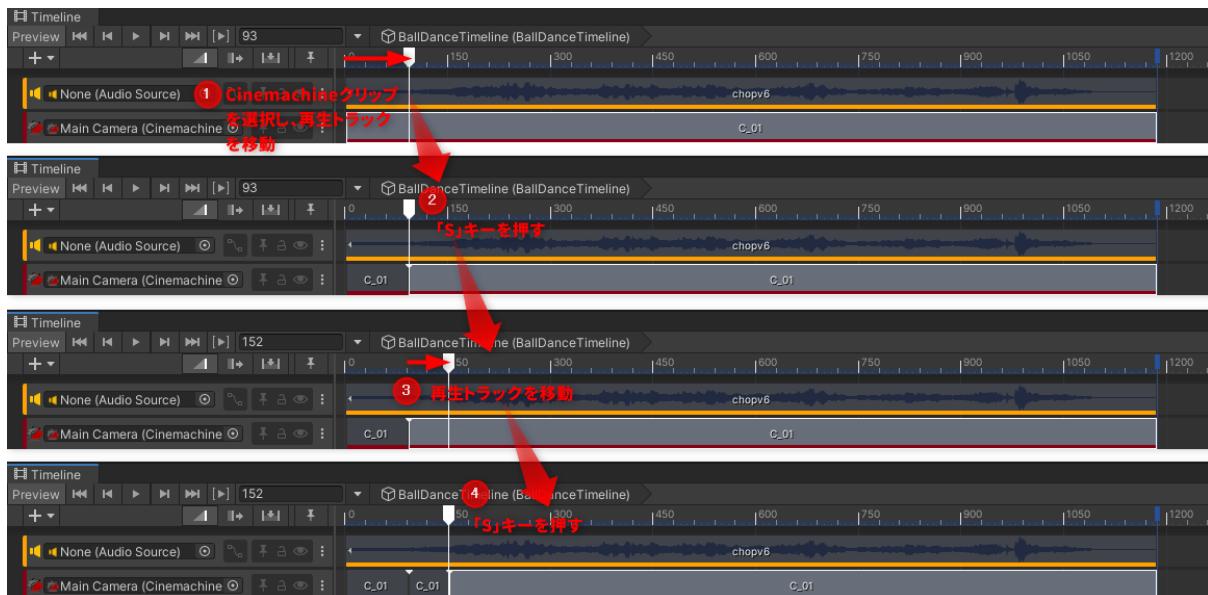
以上により、クリップの範囲内では選択したカメラポイントがショットカメラとして利用出来るようになります。

ここでは例としてC_01(ステージを真っ正面から捉えるカメラポイント)を選び、クリップの範囲をプロジェクトの有効範囲内にまで広げておきます。



Cinemachine Trackにカメラ切り替えのタイミングを記録する

さて、『Ball Dance』の作例のように、すでに決まった尺を持つ音響データのようなものがすでに存在するプロジェクトの場合、その音響データをTimeline上のトラックに配置してしまい、それを聞きながらカメラを切り替えるタイミングを設計してしまうのがよいでしょう。このような手法は、ミュージッククリップを作るような場合だけでなく、プレスコでの音響データがすでにある場合などに活用できるものです。



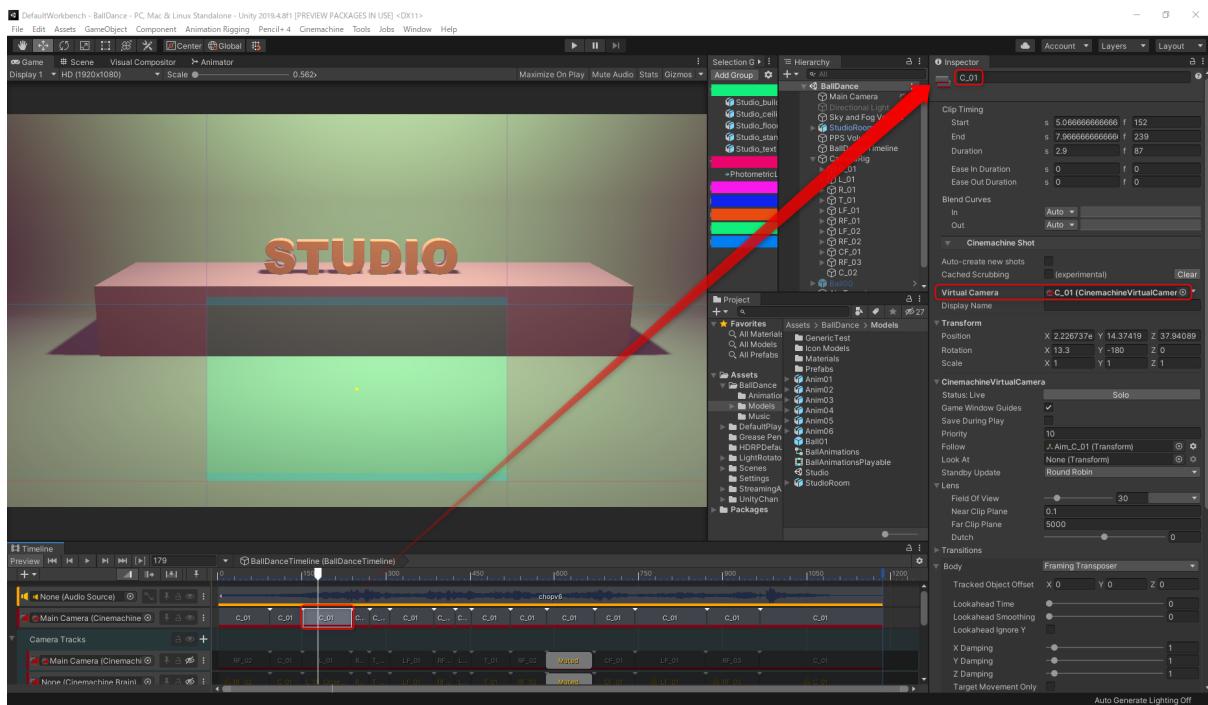
Cinemachineクリップを選択し、Timelineの再生トラックを動かしながら、音響データをスクラップ再生します。再生トラックがカメラを切り替えたい位置にやってきたら、「S」キーでクリップにカット位置を入れることができます。

カット位置は後でいくらでも微調整できますので、まずは音を聞きながら、「ここでカメラを切り替えると気持ちいいな」と思う場所に、ザクザクと切れ目を入れてしまうのがコツです。

Timelineに割り付けられたカメラを切り替える

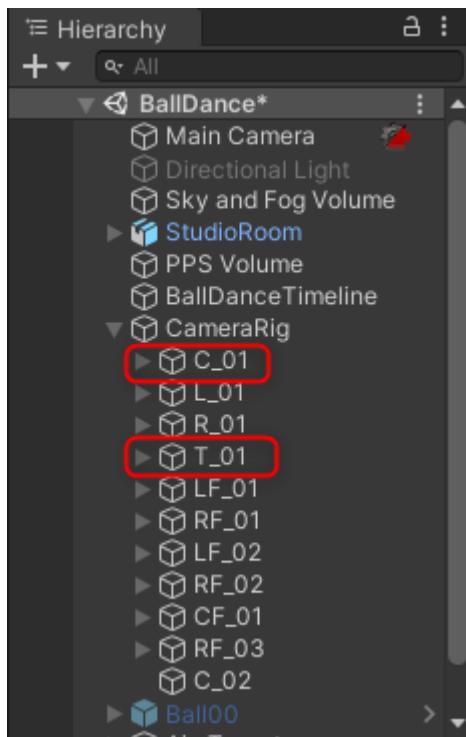
次は実際に、割り付けられたカメラを切り替えてみましょう。

Timeline上で再生トラックを移動し、あるカメラクリップを元から割り当てられているカメラから切り替えてみます。

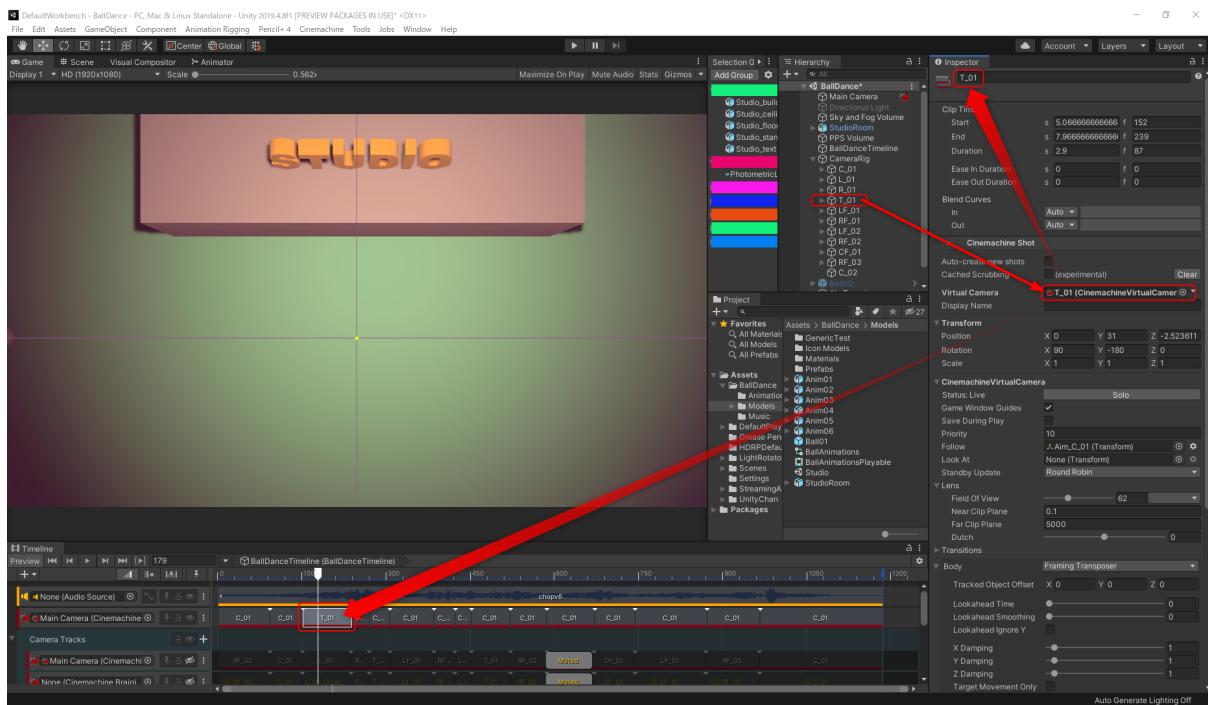


切り替えたいカメラクリップ位置に再生トラックが来たら、そのクリップを選択してインスペクター ウィンドウを見てください。インスペクター ウィンドウには、選択中のカメラクリップのプロパティが表示されています。

Cinemachine Shotの中のVirtual Cameraというプロパティの中に、現在選択されているカメラリグの名前が表示されています。上の図の場合にはC_01です。



次に、クリップのカメラをステージ直上にあるカメラT_01に切りかえましょう。ヒエラルキー ウィンドウのCameraRig階層を開くとT_01がありますので、T_01をD&Dの要領で、C_01のインスペクター ウィンドウ内のVirtual Cameraスロットに適用します。



T_01をVirtual Cameraスロットに割り当てると、インペクターウィンドウの名前も変わり、クリップに表示されているカメラ名も変わります。もしその時、再生トラックがリネームされたクリップ上にまだあれば、ゲームビューに映る景色も、T_01のものに変わるでしょう。

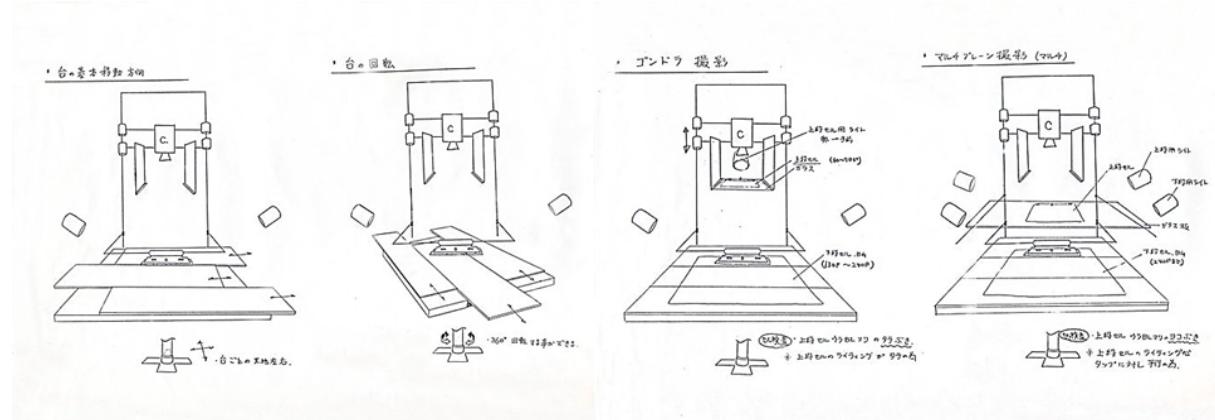
同様の操作を繰り返すことで、各クリップにアタッチされているカメラをTimeline上で変更しながら、ショットハントができます。

Framing Transposerを使いこなす

この章では、Cinemachine:Framing Transposerの機能について、さらに詳しく見ていくことにします。

CinemachineはUnityに搭載されている優秀なバーチャルカメラシステムで、様々な機能を持っています。中でも**Framing Transposer**は、3Dシーンにおけるカメラ設計を2Dレイアウトの感覚でおこなうことができるシステムです。Framing Transposerを上手に利用することで、3D空間内に撮影台的な動きをするカメラリギングを設計できます。

「撮影台」的な動きをする3Dカメラで、2Dレイアウトを考慮したアングルを設計する



<https://togetter.com/li/1376576>

撮影台は、一番下に背景美術を置き、その上にセルをいくつか重ねながら置き、最後に一番上からカメラで撮影するものです。

一番下の台は正確に左右に引くことが可能です。またカメラが吊り下げられているクレーンも、水平面で上下左右に動かせるだけでなく、垂直軸で回転することもできます。これにゴンドラを足したり、マルチプレーンと呼ばれる多段の仕組みを追加することで、被写体深度のある撮影も可能です。これらの機械式の撮影台の上で、日本の作画アニメーションのカメラワークは長年の間、作られてきました。

特に、「撮影台」は、その構造上、背景を奥行き方向(Z軸方向)に動かせません。背景を奥行き方向に動かすためには、「それ専用の動画を作成する(背景動画)」か、もしくは「背景を複数のパーツ(ブック素材)に分けて、マルチプレーンカメラで撮影する」しかありません。従って、アニメ作品のほとんどのカットでは、キャラクターが奥行き方向に移動することはあっても、それに合わせて背景も奥行き方向に移動することは、極めてまれです。

アニメーション制作がデジタルセルに移行すると、撮影台はAfterEffectsなどの2Dコンポジッターに置き換えられるようになりました。それは2Dコンポジッターのもつレイヤー単位の編集機能が、撮影台でのカメラワークを上手にシミュレートできたからです。

一方、通常の3D空間に置かれたカメラは奥行き方向に移動することができるので、背景もキャラクターの動きに伴って、ごく当たり前のように奥行き方向にも動いてしまいます。しかもそれが狙ったショットならよいのですが、普通にキャラをフォローしている最中に、この奥行き方向への細かい移動が、必要以上に起こっているとしたら問題です。それらの予期せぬ動きは、いくら2D風に見えるようにルックを調整しているにしても、そのルックに不要な3D感を追加してしまう原因にもなります。

セルルック3DCGでは、これらの3Dっぽい動きを無くすために、一度各素材をレンダリングし、2DコンポジッターであるAfter Effectsの上で、動きを付け直します。それは、3Dアセットで構成されるシーンを一度破壊して、2Dコンポジット作業に持ち込むということです。

Framing Transposerは、Unityのシーン上に、撮影台風の2Dレイアウトカメラを設計することができます。Framing Transposerを使うことで、3Dアセットを非破壊のまま、2Dレイアウトを考慮したアングルで見ることができます。



上は、キャラクターがスクリーン上で左右に移動しても、カメラがあまり追隨しないように設定したFraming Composerのレイアウトです。

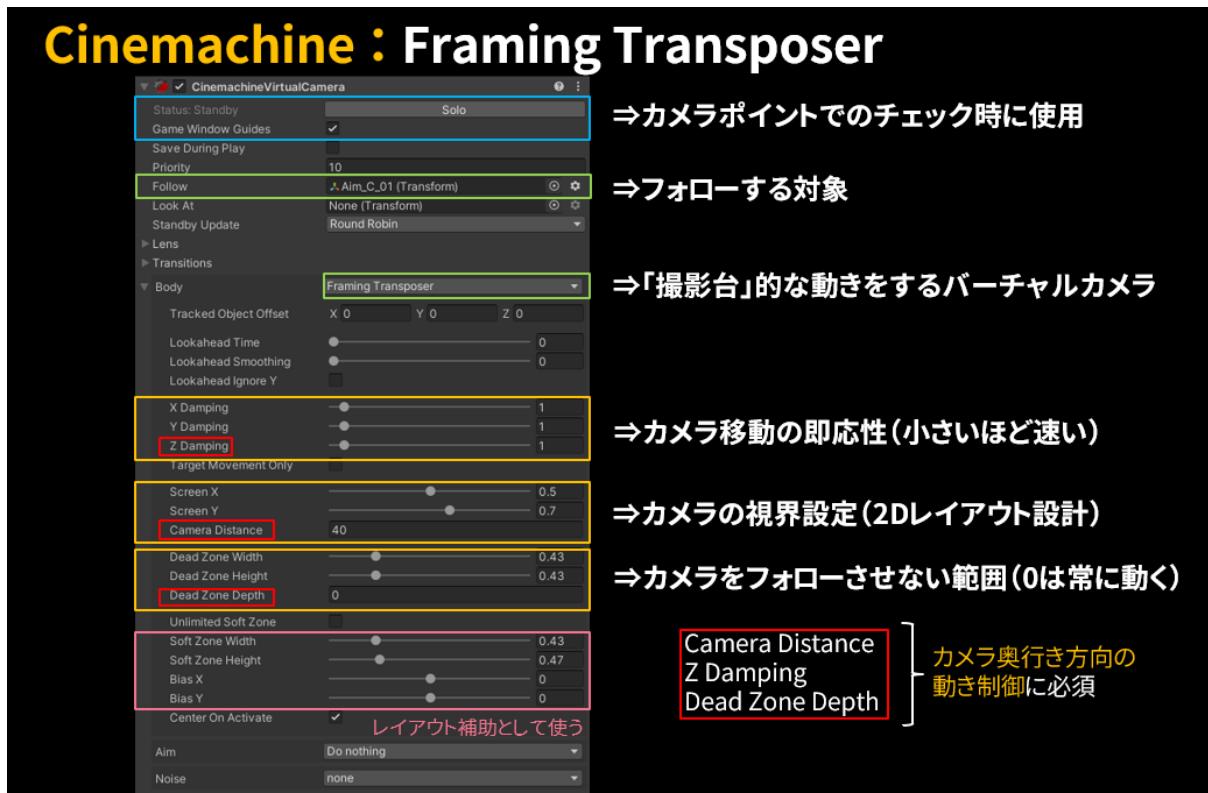
キャラがかがんだり、大きくジャンプすると、カメラはキャラを追いかけるように上下にTU/TDLします。それ以外のキャラの移動に関しては、カメラはFIX気味にキャラを捉え続けます。



次は、常にキャラを画面中央よりも少し右寄りにキープしつつ、キャラの動きをフォローし続けるように設定したFraming Composerのレイアウトです。キャラが動いても、常に画面の左に十分な空間(アイスペース)をあけておくレイアウトが維持されます。

Framing Transposerのプロパティ

Cinemachine Virtual Cameraコンポーネントより、Framing Transposerのプロパティを見ると、沢山の設定項目があります。こう沢山プロパティがあるといかにも設定が大変そうですが、以下の図のように整理すると機能が理解しやすくなります。



まず青で囲まれた部分ですが、これらはバーチャルカメラからのレイアウトのチェック時に使用するものです。Soloボタンを押すとGameビューが現在のバーチャルカメラからのビューに切り替わります。Game Window Guidesをチェックすると、ゲームビュー上にDead ZoneやSoft Zoneを示すガイドがオーバーレイ表示されます。

次に緑で囲まれた部分ですが、これらはFraming Transposerを呼び出すために最低限必要な部分です。Followに関しては、画面内に特にフォローする対象を設けない場合には、設定しなくても大丈夫ですが、Camera Distanceを設定する時に不便なので、そのような場合には原点に「空のゲームオブジェクト(Empty GameObject)」を設定し、その座標をフォローするようにするとよいでしょう。Bodyは必ず「Framing Transposer」を指定します。

その次の黄色で囲まれた3つのブロックですが、これらのブロックは、Framing Composerが設定されているバーチャルカメラから見たビュー内のレイアウトを調整するための項目になっています。その中でもさらに赤で囲まれたプロパティは、Z軸方向(奥行き方向)の調整項目で、特に重要なのがバーチャルカメラと被写体との距離を設定するCamera Distanceです。Framing Composerを設定する時には、一番最初にこの距離を設定することになります。

すでに簡単に解説はしていますが、Framing Composerでまずざっくりとカメラからの見え方を設定をする時には、Camera Distanceの他にScreen XとScreen Yを調整します。

次にその下の黄色ブロックに含まれる各Dead Zoneの調整項目は、カメラが今フォローしている被写体が動いた場合、カメラがフォローを開始しはじめる領域を示しています。この値を0にすると、フォローする被写体の動きに対して常に動いているカメラになりますが、そのようなカメラは特別な場合を除き、あまり設定をしないものです。

最後に市場上の黄色ブロックに含まれる各Dampingの設定項目は、カメラ移動の即応性を設定するものです。この値が小さいほど素早く反応します。

最後にピンクで囲まれた部分は、フォロー開始範囲(Dead Zone)の微調整に使う項目です。Soft Zoneは、Dead Zoneのマージンになっており、被写体がSoft Zoneの範囲内にある時、カメラが移動してDead Zoneの範囲内に収めようとします。一方BiasはSoft Zoneの位置を中央から上下左右に移動します。これらを調整することで、カメラのレイアウトに対して動的にも補助ができます。

奥行き方向のカメラの動きを抑制する

Framing Transposerのフォロー対象が、動かないポイントなら問題ありませんが、キャラのHipジョイントなどに付けてキャラと一緒にフォローさせている場合、キャラが奥行き方向に移動したとしても、カメラはある程度そのまま、所定の位置に留めておきたい場合がよくあります。つまり奥行き方向への少しの移動ならカメラはそのまま動かないが、キャラがさらに移動する場合には、それに応じてカメラも一緒に付いていくような設定です。

カメラとフォロー対象の間の距離を設定するのがCamara Distanceですが、**Dead Zone Depth**は、その距離がどれほど変化したらカメラをフォロー対象に対して奥行き方向に追随させるかの値です。つまりこの値を大きく取ることで、フォロー対象が奥行き方向に移動しても、カメラも一緒に追随しないように設定することができます。ひとつの目安として、**Camera Distance**の値よりも**Dead Zone Depth**を大きくとることで、奥行き方向への意図外のカメラの動きを抑止できます。

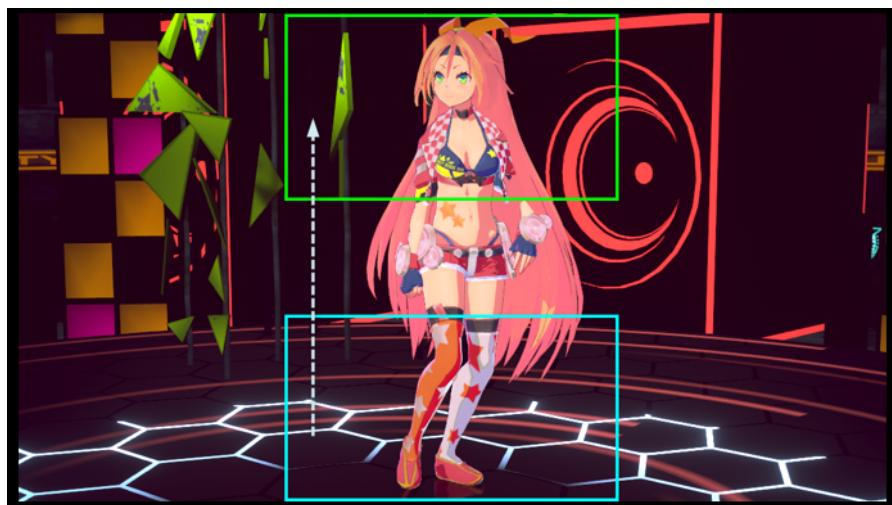
Z Dampingは、もしカメラが追随して動く場合、その反応の速さを設定します。これも値が小さいほど素早く反応します。

様々な作画アニメ風のカメラワークを設計する

動画サンプルは以下のリンク先にあります。

<https://www.youtube.com/watch?v=BAUFzqjs3eI>

1. アニメ風の2DPANを設計する



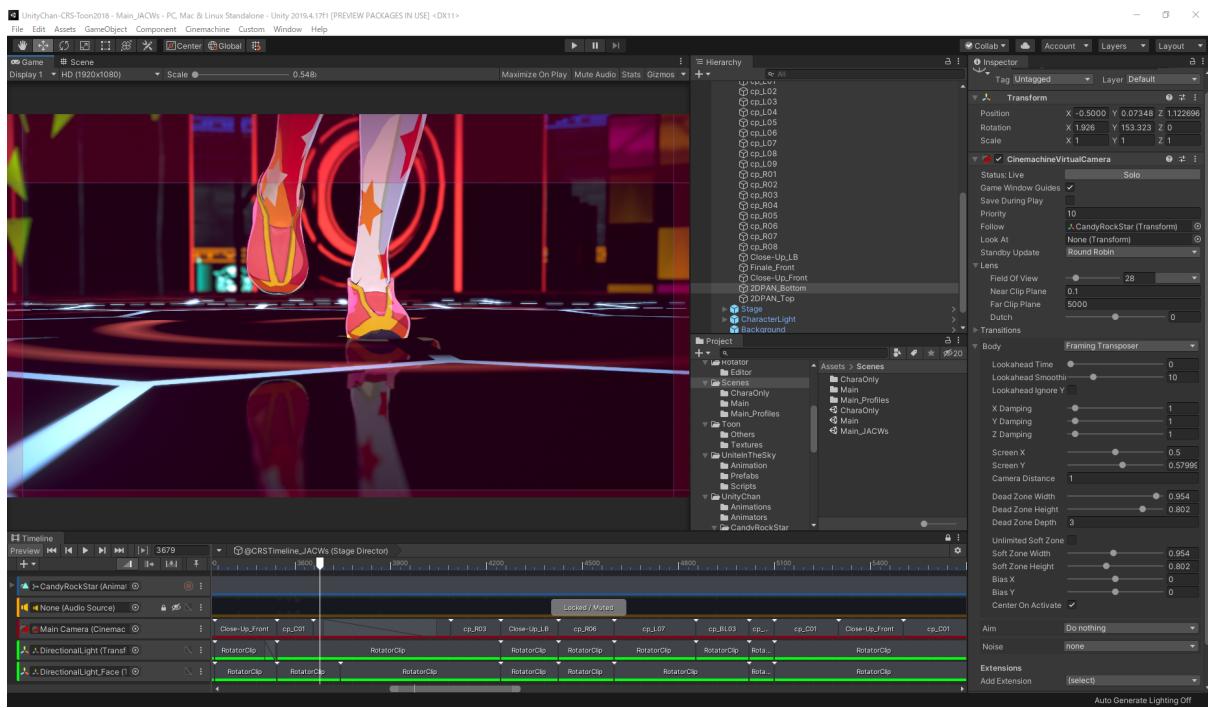
キャラの足元からバストアップまで、カメラが上に向かってパンしていくような**2DPAN**は、アニメではよく使われるカメラワークです。(※一般的にはティルトアップ、もっと正確に言えばクレーンアップですが、アニメではしばしば**2DPAN**と呼ばれますので、そのように記述します。)

2DPANは、絵コンテではしばしば上の図のように指定されますが、**Framing Transposer**を使えばこの図の通りに設計することができます。

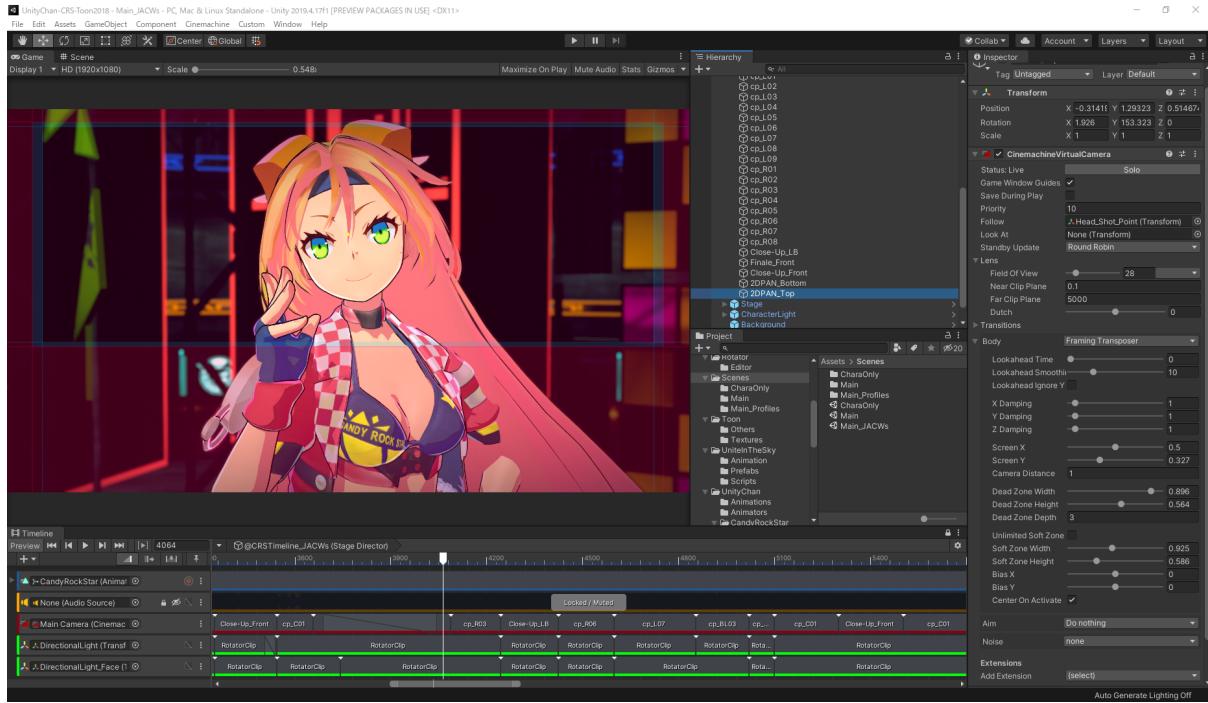
2DPANを設計する場合、「キャラ足元のクローズアップのカメラポイント(2DPAN_Bottom)」と「キャラのフェイスアップのカメラポイント(2DPAN_Top)」を設定します。

以下の図は、2DPAN_Bottomの設定の例です。まず足元のアップをシーンビューで表示し、そこにカメラポイント用の空のゲームオブジェクトを設定することからはじめるとよいでしょう。

後でそのカメラポイントよりも1メートルほど上に2DPAN_Topを設定しますので、Cinemachine Virtual Cameraコンポーネントを付ける前に、2DPAN_Bottomをデュプリケートして、2DPAN_Topとリネームしておきます。Cinemachine Virtual ComponentよりFraming Transposerを有効にしてしまうと、カメラポイントの座標がインスペクター上で直接動かせなくなるので、コンポーネントを付ける前に複製しておくのがコツです。



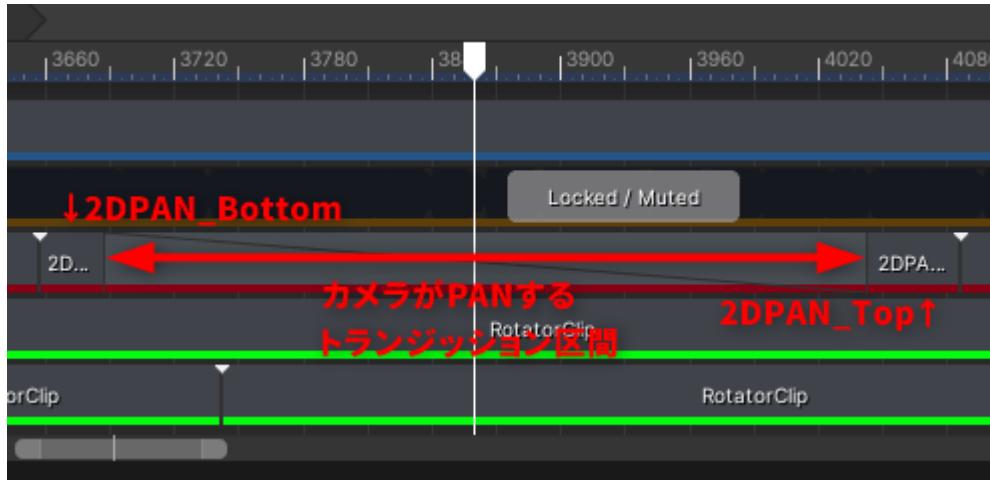
2DPAN_Bottomは、Framing Transposerで、キャラの足元にある座標(CandyRockStar)を常にフォローするようにしておくと、キャラがどんなに移動しても常に足元のアップを追いかけて続けるカメラが設計できます。またカメラとの距離を、FOV28のレンズで1メートル程度の近い距離に設定していますので、Dead Zone Depthを3とすることで、2DPANを設計したいアクションの範囲では、カメラを奥行き方向に動かさないと同時に、被写体深度エフェクトを安定させます。カメラから見たレイアウトの最終調整は、主にScreen Yで行います。キャラの足元と床のミラーエフェクトとの対比が面白いショットですので、画面上でその配分が上手く映るように高さを設計します。Dead Zoneの設計からもわかると思いますが、キャラが左右に動いても、極力カメラは左右に振れないような設計になっています。



続いて2DPAN_Topの設定です。

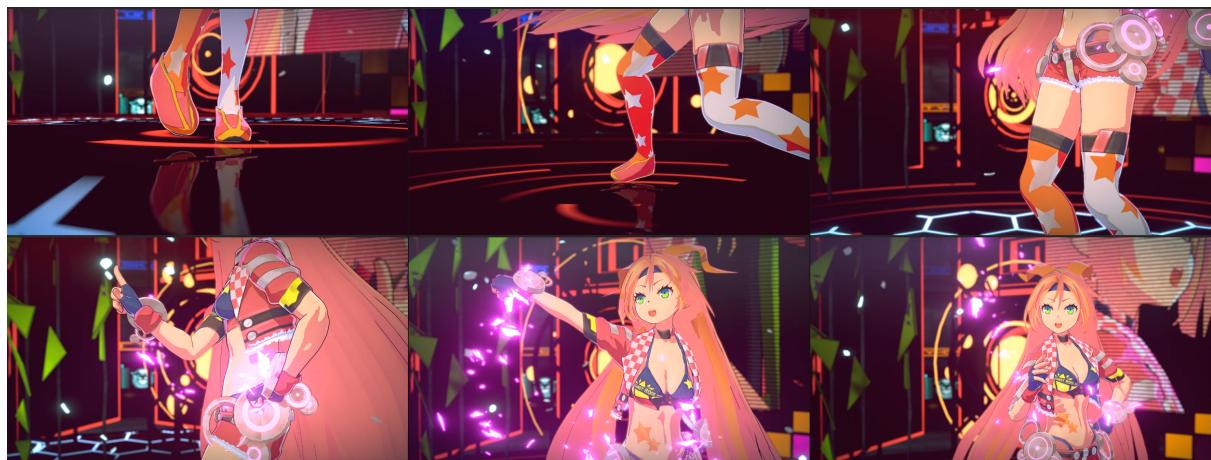
2DPAN_Topは、常にキャラの顔正面の座標(Head_Shot_Point)をカメラがフォローし続ける設定となっています。こちらもカメラの距離やレンズのFOVなどの基本的な設定は2DPAN_Bottomと同じにすることで、見え方が急に変わらないようにしています。

キャラの表情をなるべく画面の上のほうで写したいので、Screen Yを調整して、Dead Zoneを画面の上の方にずらしています。このカメラポイントでも必要以上にカメラを左右に振らないことと、奥行き方向に移動させないことが大切なので、各Dead Zoneをそのように設計します。



最後にCinemachine Track上にふたつのカメラポイントのクリップを並べ、2つのクリップをトランジションさせれば完成です。

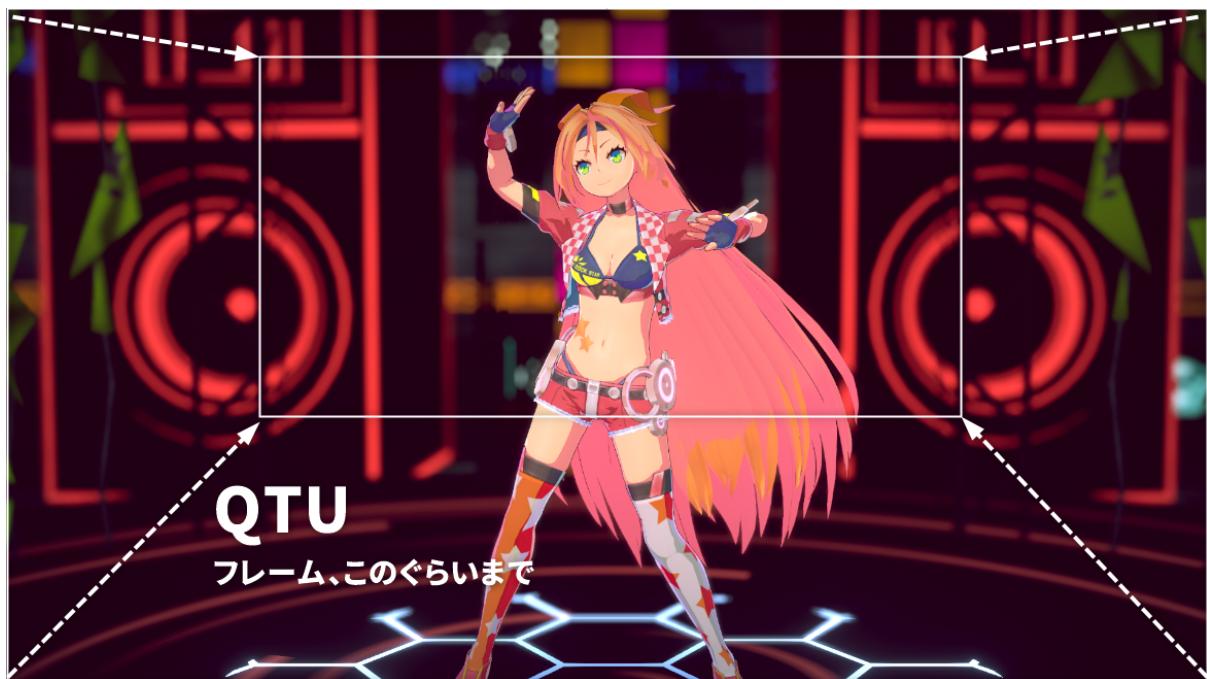
カメラがパンするスピードの調整は、このトランジション区間の調整で行うことができます。



最終的に出来上がる画面は、キャラを2DPANで押さえつつ、背景オブジェクトは2D作画背景っぽく映っています。(スピーカーの置かれているシーンの大判縦長背景の前に、ミラー効果が合成されている床のブック背景があり、それらをスライドで撮影しているような映り方です) 不必要なカメラの動きをコントロールしているためです。

Framing Transposerを使うことで、このような2Dアニメの良さを活かした見せ方が可能となります。

2.QTU(Quick Truck Up/ポン寄り)を設計する



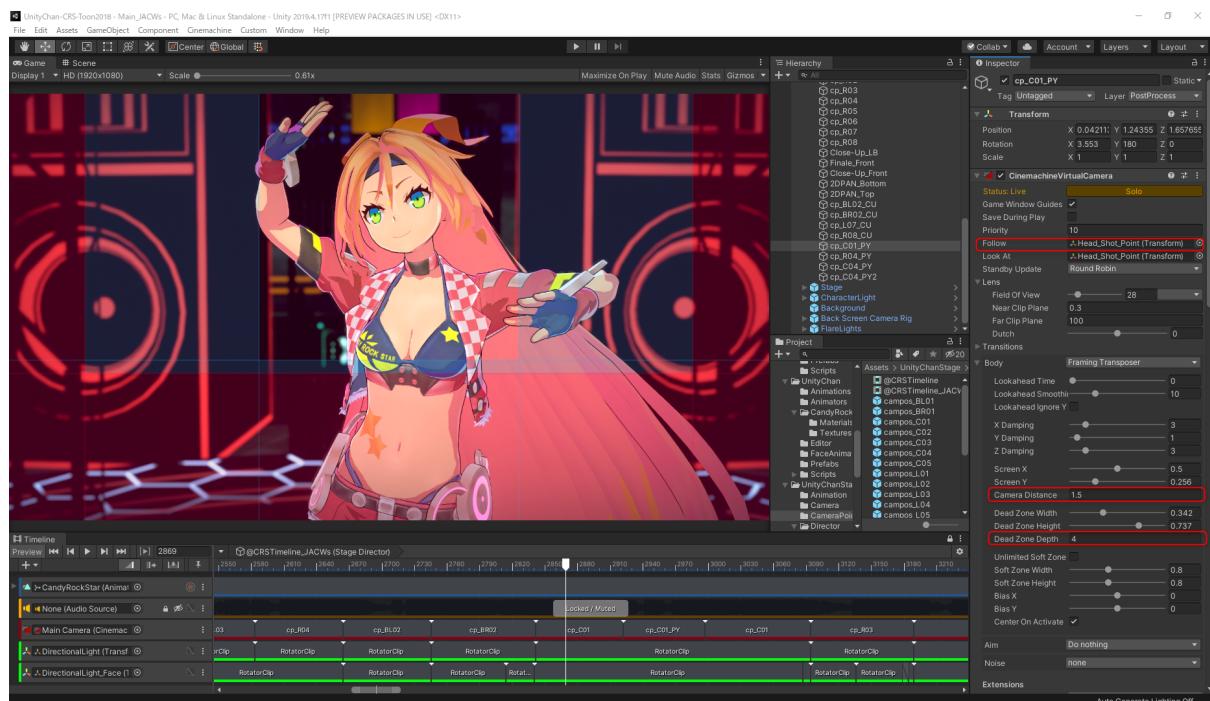
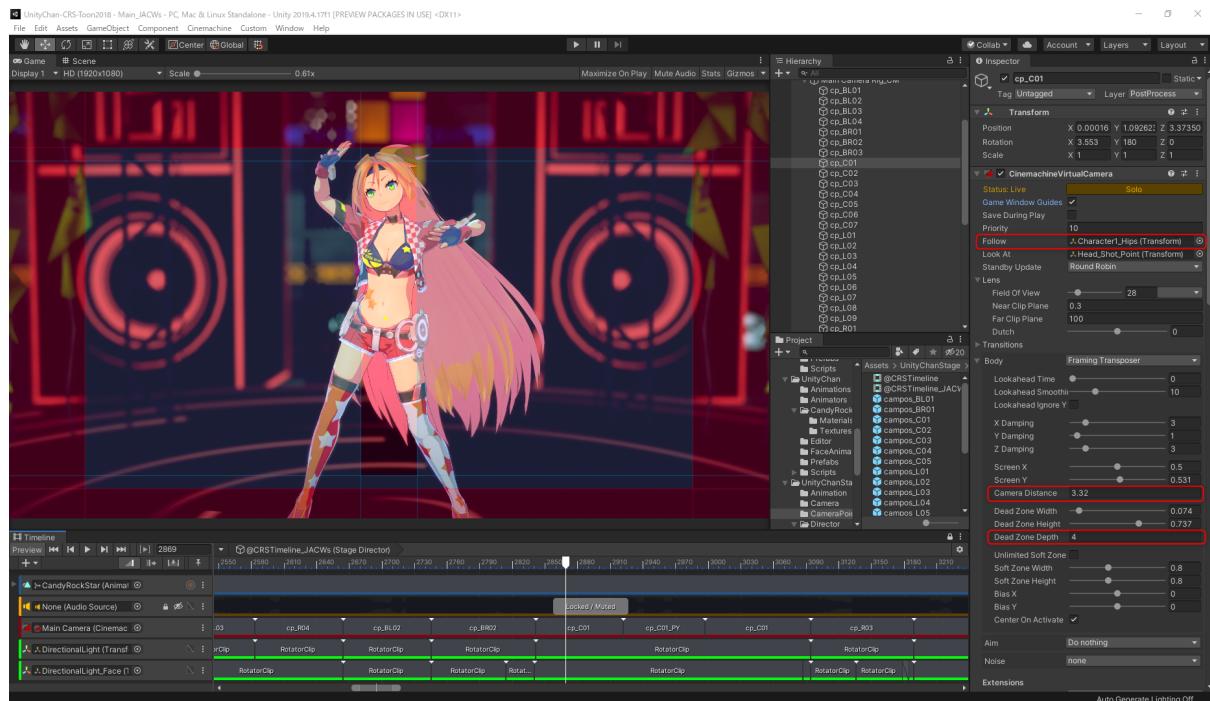
上の図のように、「同一レイアウト(ポジション)からのクイックトラックアップ(QTU)」というのも、作画アニメのカメラワークではしばしば見かけます。QTUとは、素早いカメラの寄りのことを指しています。

QTUの結果、キャラは大写しになりますが、ズームとは違ってキャラの見た目のパース感は変化しません。背景オブジェクトの見え方は、カメラのレンズFOVによって変わりますが、被写体となっているキャラの見栄えとしては2D的な拡大になっています。



このようなQTU的なカメラを設計するためには、元のカメラポイントをデュプリケートした後で、新規にできたカメラポイントにアタッチされているCinemachie Virtual Cameraコンポーネントの**Camera Distance**をより小さな値にします。この時、**Lens**の**FOV**設定は一切変化させないのがポイントとなります。

また**QTU**の前と後とで、**Dead Zone Depth**に十分に大きめの値を確保することで、カメラの奥行き方向への不要な移動を抑止します。



QTU後のカメラポイントでは、しばしば注視点が変わります。上の例でしたら、キャラの顔を中心
にレイアウトを組むことになります。その場合、Followする座標を「顔」(Head_Shot_Point)に変
更しておくとよいでしょう。

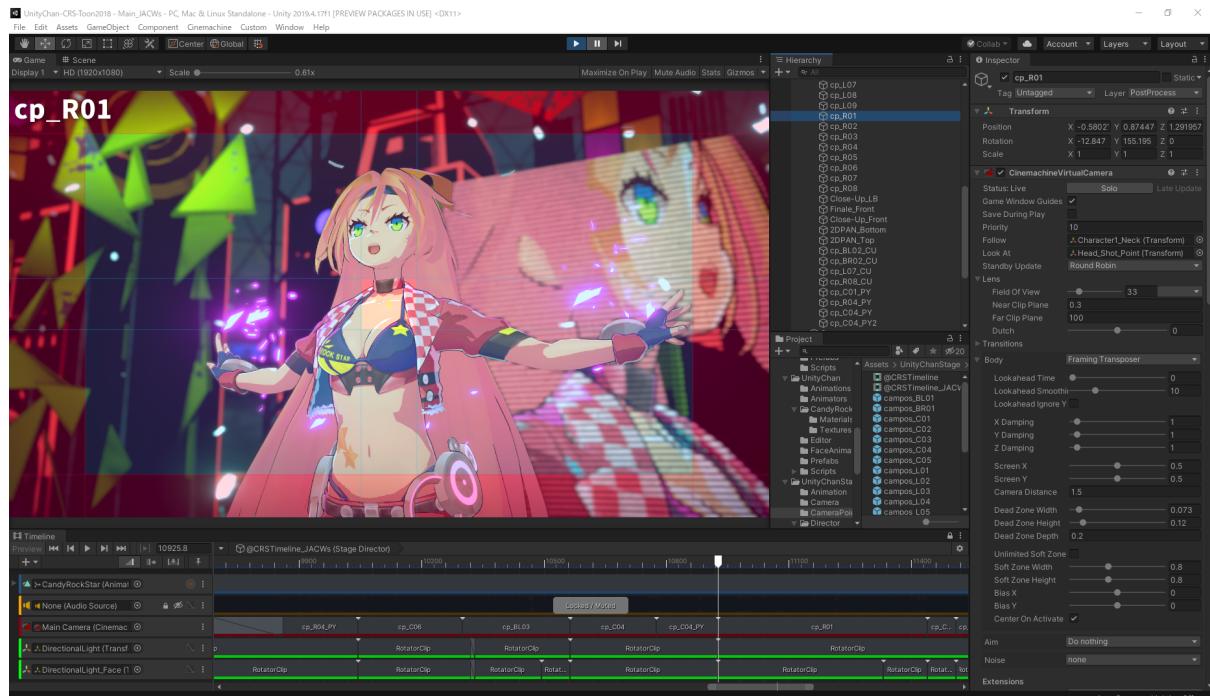
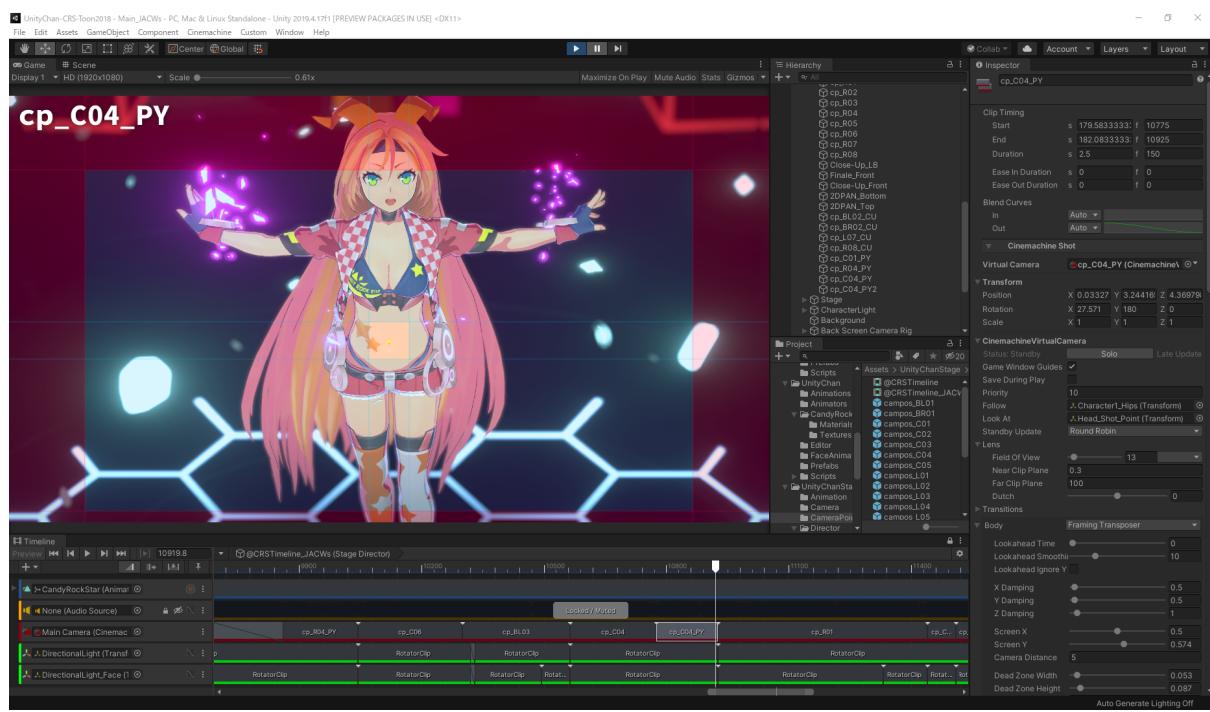
2つのカメラポイントを、カメラトラックの上で短い時間でトランジッションさせれば、
「QTU」となります。

トランジッションなしで2つのカメラを切り替えると、「ポン寄り」となります。小気味よくポンと寄るカ
メラワークも、アニメでしばしば使われるものです。

3. アクション繋ぎ

最後に、一連の演技を効果的に見せるための「アクション繋ぎ」について説明します。

アクション繋ぎとは、一連のアクション(モーション)の途中でカットを切り替える技法です。



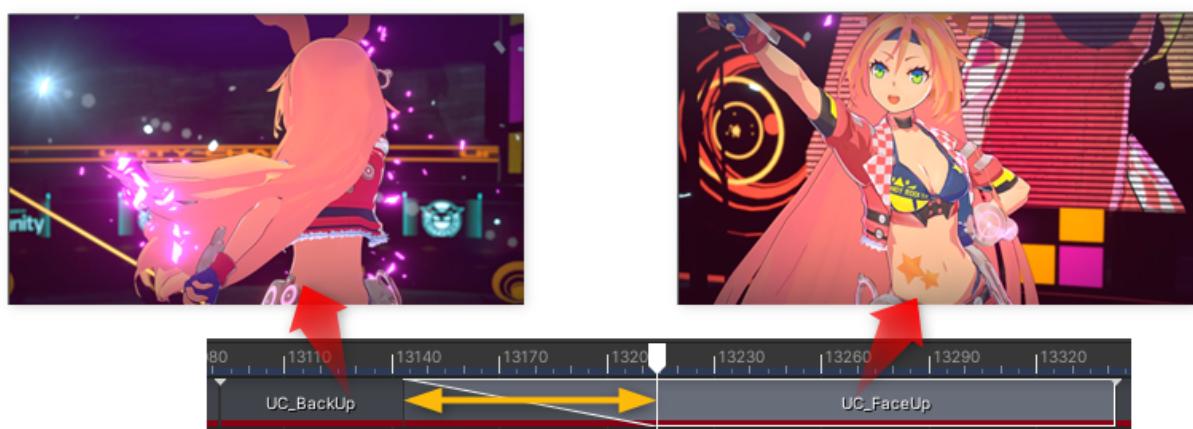
上では、「ユニティちゃんが唱いながら、ゆっくりと両手を降ろしていくアクション」の途中でトランジションなしでカメラを切り替えていました。アクション繋ぎを入れやすいポイントは、一連の続くアクションの中で、視覚記憶に残りやすい動きやポーズ、エフェクトの入っているところです。キャラのサイズも同じぐらいに映っているカメラどおしたと、さらに効果的です。

アクション繋ぎを効果的に入れることで、一連のアクションが流れるようにカット間を繋げることが可能です。

4. ブレンドカーブで見え方を調整する、高速な回り込み

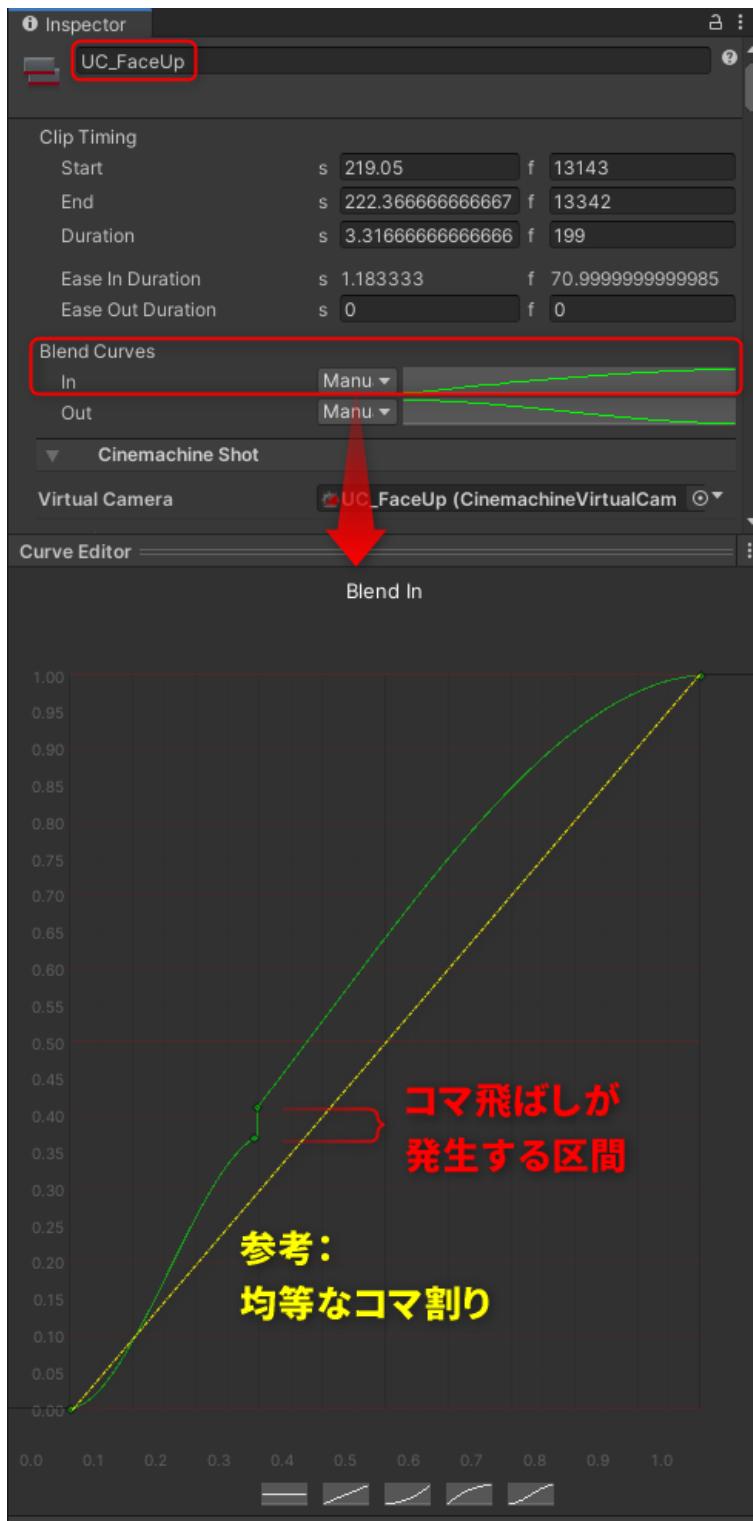


上の図は、左上のキャラクターの背後からのショットから、右下のキャラのバストアップまで、約1.2秒程度で高速に回り込むカメラワークです。このようなカメラワークも、Framing Transposerで設計することができます。



このカメラワークでポイントとなるのは、2つのカメラポイントを繋ぐトランジション部分の設定です。上の図で黄色の矢印の区間で示されている、トランジション自体の長さは変化させずに、その間に表示される絵を変えることができます。これは、上の図でいえば、トランジット先である **UC_FaceUp** カメラポイントの **Blend Curves** プロパティの内、**In** に当たるカーブを調整することで、トランジション中に表示されるフレームを詰めて、特定のアングルをより多く見せることができます。

タイムラインより、**UC_FaceUp** のトラックを選択し、インスペクターに表示すると、トランジションのブレンドカーブを調整できるようになります。この場合は、**In** にあたるブレンドカーブを **Manual** に変更することで、インスペクターの下部に表示される **Curve Editor** より **Brend In** を微調整します。

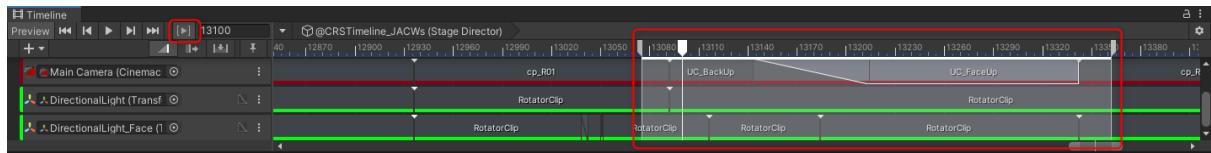


通常のBlend Inのカーブには、イースイン/アウトを意識した、柔らかいS字カーブが描かれていますが、マニュアルモードに切り替えることで、それを直接手で操作できます。カーブ上で右クリックメニューを呼び出すと、新規にキーを追加する他、キー近傍でのカーブのタンジェントが操作できますので、例えば上のような形にしてみます。

すると、2つのカメラポイントへのトランジション中に描画される画面が変わることが確認できます。

上図のカーブエディタ画面には、参考用に均等にコマを割った場合のカーブを黄色の破線で入れてあります。これを参考にすると、緑色のブレンドカーブは、均等割りよりも多めに

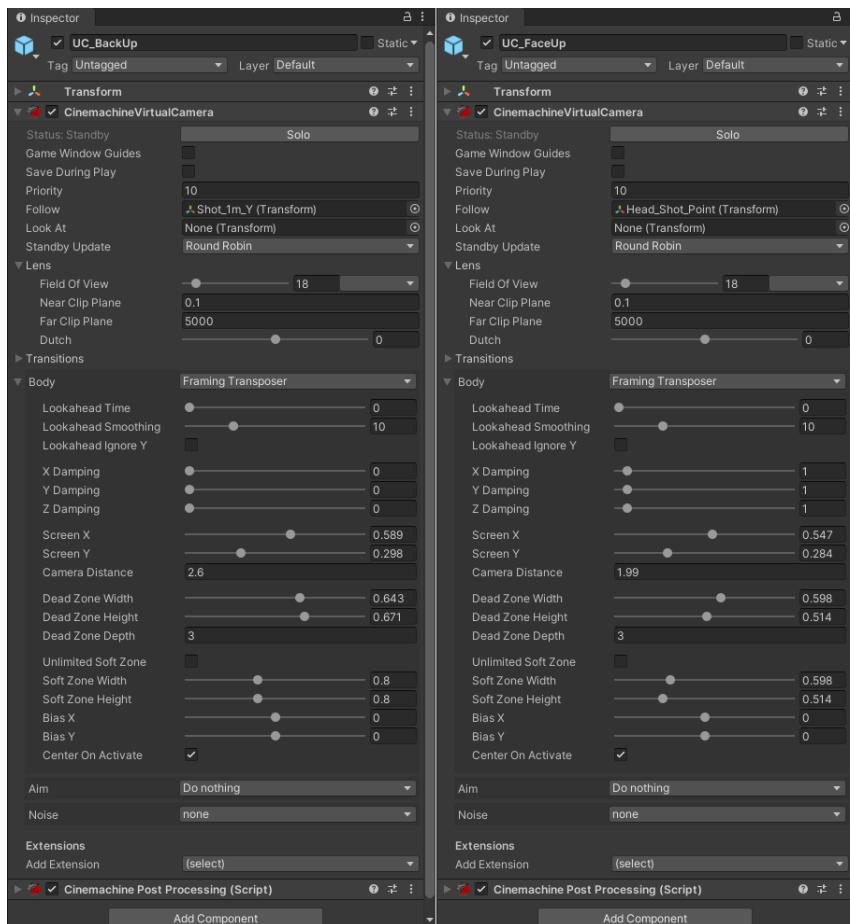
UC_FaceUpへと変化していく画面を表示しつつ、中間ちょっと手前ぐらいのタイミングで、コマ飛ばしを入れることで、キャラがカメラから逃げてしまうコマを排除していることがわかります。このような作業をする際には、区間の連続再生をしながら、ブレンドカーブを調整するのが便利です。



もちろんCinemachineの機能の一つにドリートラックがありますので、それを使って同様のカメラワークを作ることも可能ですが、2つのカメラポイントでの2Dレイアウトをしっかりと設計した上で、その2点間のトランジッション時間を決定し、同時にその間のフレーム変化をカーブエディタで調整する方式のほうが、より絵コンテと原画中割のツメ指定による作画アニメ風のカメラワークに近い感覚でカメラワークが設計できます。

実際に、ブレンドカーブを手でいじってみて、どのようにカメラワークが変わるのがかを体験してみるのが、理解への早道です。

なおトランジッションする2点間のカメラポイントの設定に関しましても、従来どおり、カメラのFOVを必要以上に変化させなかつたり、**Camera Distance**と**Dead Zone Depth**の関係を適宜とつてやることで、トランジッション中でもより安定感のあるフレーミングを確保することができます。



アニメ風の3Dキャラモデルを見栄え良くみせるためのFOV

ここでは、アニメ風の3Dモデルを見栄え良くみるためのカメラレンズのFOVについて説明します。これらの設定は、各カメラポイントにアタッチされている、Cinemachine Virtual Cameraコンポーネント側のLensプロパティのField Of Viewで行います。UnityでのFOVは、垂直画角での値ですので、設定時には注意してください。



基本となるカメラのFOVは、20~38ぐらいの範囲。28、33、35辺りは使い勝手がよいです。
カメラとの距離は割と近めです。

垂直画角でのFOVが20~38の範囲は、35mmフル(センサーサイズ36mm×24mm)の対角画角での換算だと、35~64ぐらいです。FOV=35~64というと、カメラ的には「概ね標準レンズ」ぐらいです。

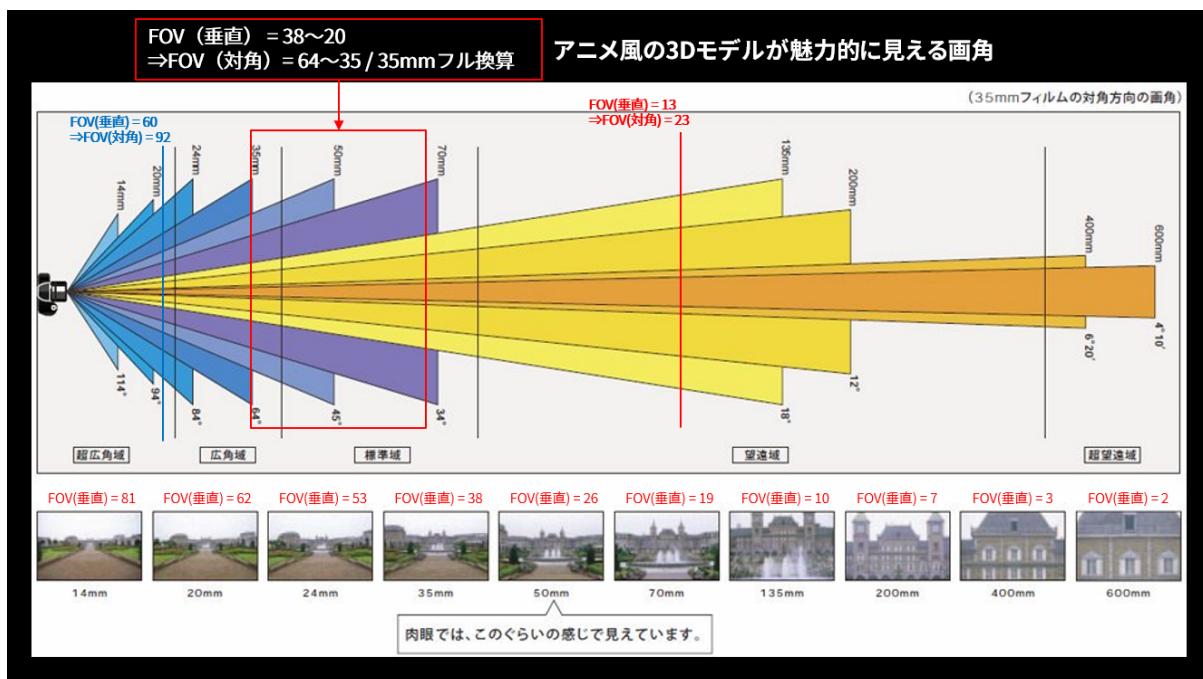
パースを見せつつも、2D作画に近い感じにモデルを撮したい場合には、標準レンズ～望遠レンズを選ぶようにすると、アニメ風3Dモデルが「絵」的基準で魅力的に見えやすいです。

一方、UnityデフォルトのカメラのFOV(垂直)= 60 は、標準的なレンズからみれば超広角にあたるので、フレーミングはしやすですが、アニメ風3Dモデルを扱うためには、パースがキツいといえるでしょう。

もちろん、アニメ風3Dモデルが一番見栄えよく見えるのは、それをモデリングしている最中に主にデザイナーが確認に使っているカメラ画角ですので、その情報を事前に確認しておくのも大切なことです。

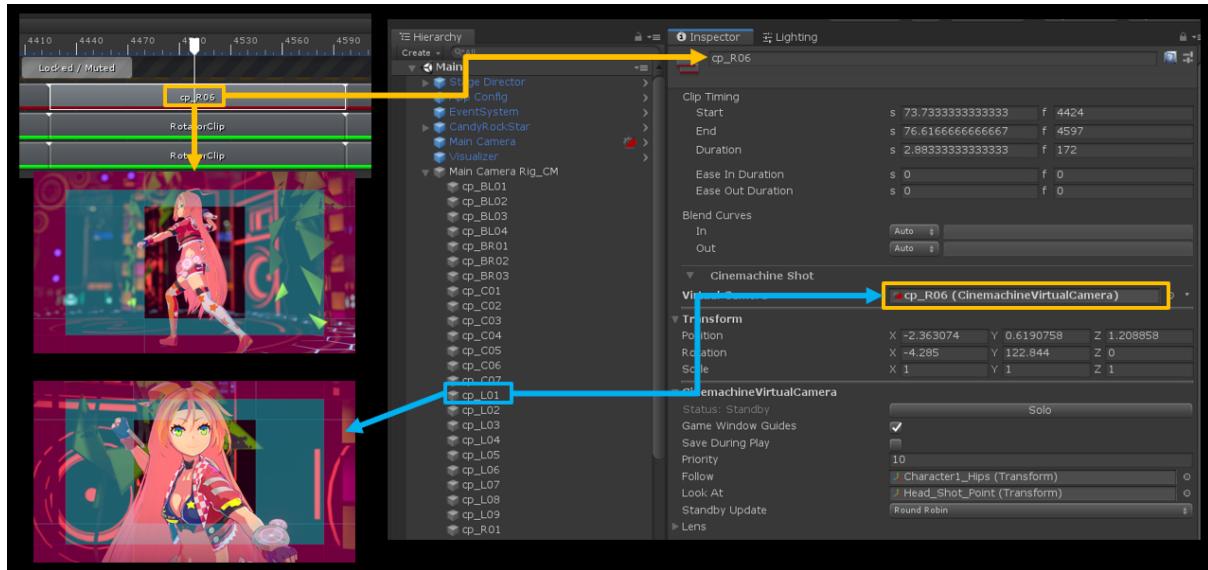


俯瞰のレイアウトは元々バースがかかりやすいものです。
従ってこのようなレイアウトには、FOVを望遠にとりつつ、距離をとります。
そうすることで、3Dキャラモデルに必要以上にバースがかかるのを防ぎます。



https://www.drone-press.jp/drone-basic-information/drone-introduction/drone_camera/

ショットハンティングのためのTips



すでに説明したように、Timelineのカメラトラックから、各クリップが指定しているカメラポイントを交換するのは簡単です。

カメラポイントを差し替える、「クリップの幅」が示している時間や、そのクリップが有効になるタイミングは変わらないので、その時点で見栄えのするカメラポイントにどんどん差し替えていけばよい(ショットハンティング)でしょう。

クリップを切り替えるタイミングは、ミュージックデモならば音楽とアニメーションを見ながら、小気味よく刻んで行きます。もちろんコンテがあれば、それに従います。



その準備のために、**Framing Transposer**を使って各カメラポイントに対して、2D的な見栄えを重視したレイアウト設計をしておくとよいでしょう。これらのレイアウトは、監督、原画アニメータ、作画デザイナーの観点から、良い悪いを判断します。もしくはそれらの人達に、各カットを確認してもらうチェック工程を入れることで、各カメラポイントのレイアウト精度をあげることができます。

もちろん絵コンテなどがあったら、それを参考にしてカメラポイントとレイアウトを設計してもよいです。

Cinemachine Virtual Cameraに、そのカメラポイントにおける代表的な動き込みのレイアウトを設定しておくことで、キャラクターにアニメーションを流し込むと、キャラに従って動く注視点を狙つて各カメラがレイアウトを保持しながらフォローし続けます。

これらの機能を組み合わせると、Unity上でキャラモデルにリアルタイムに流し込んだモーションキャプチャデータを、必要な複数のカメラポイントから確認しつつ、演技指導をするようなことも可能になります。

Cinemachineの日本語ドキュメント

<https://docs.unity3d.com/ja/Packages/com.unity.cinemachine@2.6/manual/index.html>

Cinemachineは、Framing Transposerの他にも様々な機能を持っています。一度、日本語ドキュメントを確認していただくのがよいでしょう。

Cinemachineで2D作画アニメ風のカメラワークを設計する場合のまとめ

- 各カメラポイントでは、3Dモデルキャラクターが魅力的に見える、FOVをキープする。
- 2D作画アニメのカメラワークでは、元々撮影台の制約があるので、奥行き方向に背景オブジェクトが動いていくカットは必要以上には作らない習慣がある。むしろ3Dカメラでは簡単にできてしまうカメラワークだが、逆にそれが3D臭さの原因になることを注意する。
- Framing Transposerの機能を活かして、意図以外の不要なカメラの移動やブレを極力抑えるのと同時に、2Dレイアウトとして絵的な見栄えが優れている画面設計を考慮しながら、各カメラポイントを設計する。
- 2つのカメラポイントを繋ぐ場合に、極端にキャラの見え方(パース)が変わらないように注意すると、それだけで3D臭さを十分に防ぐことができる。