VR コンテンツのつくりかた 4
ARKit Face Tracking

youten　著

2018-10-09 版　　発行

目次

第 1 章

はじめに

就職しました（言い訳から入るスタイル）。

というわけで、「VR コンテンツのつくりかた 4」と題しまして、「iPhone X の ARKit Face Tracking で 3D モデル動かすとかんたんでおもしろい」という話を本書ではお届けします。

本書は、以下のような方をターゲットにしています。

- iPhone X の ARKit Face Tracking が気になる人
- Unity がちょっと分かって、X 系の iPhone を持っていて TrueDepth カメラで遊んでみたい人
- 最近かわいい 3D モデルをホイホイ買ったはいいものの使い道がない人

1.1 本書の賞味期限とリポジトリ公開について

なお、本書は記載内容の賞味期限が切れてしまうことへのアップデート対応、あまり紙メディアが向いていない環境等のバージョンアップに追従するため、初版以降の PDF と関連ファイル一式を全て以下のリポジトリで公開予定です。あらかじめご了承ください。

- `https://github.com/youten/howto-create-vr-contents4`
  - Re:VIEW の素材一式、出力 PDF、関連プロジェクトのソースコードを全て含む想定です。
- `https://github.com/youten/YXAvatar`
  - 本書の後半で扱う、iPhone X の ARKit Face Tracking で VRM キャラクターを動かすアバターアプリの Unity プロジェクト一式です。
  - 他 OSS をいくつか含みます、それぞれのライセンスは配下のドキュメントに

従います。

本書のうち、私 youten が著作権を有する範囲のライセンスについては、文章は CC-BY 4.0 ライセンス[*1]を適用します。ソースコードについては MIT ライセンス[*2]を適用します。

本書籍は Re:VIEW で作成されており、その設定ファイル等について、MIT ライセンスに基づき「C89 初めての Re:VIEW v2」リポジトリ[*3]で公開されているものを利用させていただいております。

[*1] https://creativecommons.org/licenses/by/4.0/legalcode.ja
[*2] https://opensource.org/licenses/MIT
[*3] https://github.com/TechBooster/C89-FirstStepReVIEW-v2

第 2 章

ARKit Facial Tracking について

▲図 2.1　iOS ARKit

ARKit Facial Tracking とは、ARKit と iPhone X を用いたフェイストラッキングする機能です。iPhone X に搭載された TrueDepth カメラを用いて顔を（カメラ画像のみを用いる類似の技術と比較して）とても正確にトラッキングします。

また、顔内の各パーツの位置を 50 種類以上のパラメータとして取得することができます。目や口の位置と開き具合といったよくあるものから、唇を突き出す `mouthFunnel`や`mouthPucker`ではキス顔が取れますし、iOS 12.0 からサポートとなった `tongueOut`ではアカンベーが取れます。個人差はあるものの、取得できる値はとても正確で扱いやすく、かんたんに 2D や 3D キャラクターに適用することができます。

Apple 公式サイトのビデオ[*1]が簡潔でわかりやすく、こちらを一度見ておくことをオススメします。親切にテキスト起こしもされています。

[*1] Face Tracking with ARKit `https://developer.apple.com/videos/play/tech-talks/601/`

2.1 Facial Tracking の詳細

空間上の顔の位置と向きを正確に取得します。位置については上下左右の他、奥行きもそれなりに取得します。向きについても、顔のパーツが隠れない Roll 方向はもちろん、Yaw や Pitch 方向もかなりがんばって追従します。

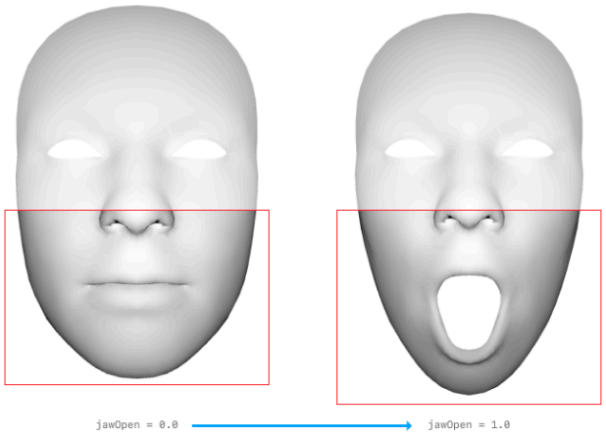
また、50 種以上の顔のパーツの位置について、0.0 から 1.0 までその強さを数値で表された BlendShape の Dictionary として取得できます。以下に取得可能な BlendShape 箇所の一覧[*2]を示します。

▼表 2.1　BlendShape Location 一覧

eyeBlinkLeft	eyeLookDownLeft	eyeLookInLeft	eyeLookOutLeft
eyeLookUpLeft	eyeSquintLeft	eyeWideLeft	eyeBlinkRight
eyeLookDownRight	eyeLookInRight	eyeLookOutRight	eyeLookUpRight
eyeSquintRight	eyeWideRight		
jawForward	jawLeft	jawRight	jawOpen
mouthClose	mouthFunnel	mouthPucker	mouthLeft
mouthRight	mouthSmileLeft	mouthSmileRight	mouthFrownLeft
mouthFrownRight	mouthDimpleLeft	mouthDimpleRight	mouthStretchLeft
mouthStretchRight	mouthRollLower	mouthRollUpper	mouthShrugLower
mouthShrugUpper	mouthPressLeft	mouthPressRight	mouthLowerDownLeft
mouthLowerDownRight	mouthUpperUpLeft	mouthUpperUpRight	
browDownLeft	browDownRight	browInnerUp	browOuterUpLeft
browOuterUpRight	cheekPuff	cheekSquintLeft	cheekSquintRight
noseSneerLeft	noseSneerRight	tongueOut	

具体的に `jawOpen`について見てみましょう。

[*2] ARFaceAnchor.BlendShapeLocation　`https://developer.apple.com/documentation/arkit/arfaceanchor/blendshapelocation`



▲図 2.2　BlendShapeLocation jawOpen

一番あごを閉じた状態で 0.0、一番開いた状態で 1.0 が取得できます。このように、全ての BlendShape について白マスクの画像が載っていますのでドキュメントのみでどのパラメータが顔のどの箇所を担当しているのか、分かるようになっています。

2.2　ARKit としてのトラッキングとモード

ARKit の一機能であるため、「iPhone をバーチャル空間を覗き込む窓として」カメラが捉えた顔のトラッキング結果を返す動きをします。これが少しややこしいので、気をつけましょう。

`ARConfiguration.WorldAlignment`[*3]設定によって、現実世界の情報にどれだけ追従して 3 軸を返すか、という違いがあります。

- `gravity`：鉛直方向を Y 軸として、ディスプレイ正面の向きを Z 軸、同じくディスプレイの左右の向きを X 軸とするモード
- `gravityAndHeading`：鉛直方向を Y 軸として、南北を Z 軸、東西を X 軸とするモード
- `camera`：重力や地磁気方向を考慮せず、ディスプレイ正面の向きを Z 軸、左右の向きを X 軸、上下の向きを Y 軸とするモード

[*3] `https://developer.apple.com/documentation/arkit/arsessionconfiguration/worldalignment`

現実世界を拡張する AR 的な使い方をしたい際には gravityや gravityAndHeading を、そうではない際には cameraのモードを設定しましょう。

2.3 TrueDepth カメラと顔の位置検出

TrueDepth カメラを用いていると言われていますが、点滅している（携帯のカメラプレビューなどで確認できます）赤外線ライト部を隠してもトラッキングが継続できたりと謎が多く、詳細な動作については明かされていません。

トラッキングを動かしつつ、Xcode で CPU 負荷を観測していると、「顔の位置をトラッキングするモード」と「顔の位置がトラッキングできた上で、顔の各パーツのトラッキングするモード」の 2 段階を切り替えながら動作しているように見えます。

また、前述の公式ビデオでは 1 秒間に 60 回トラッキングしているとのことですが、実際にコールバックを数えてみると、概ね秒間 50 回のようです。

第 3 章

Unity で ARKit Face Tracking で遊ぶ

それでは、概要について抑えたところで、実際に Unity で 3D モデルを動かして遊んでみましょう。

3.1 sloth アライグマサンプル

▲図 3.1 Unity ARKit Plugin フェイシャルトラッキング sloth アライグマサンプル

Unity ARKit Plugin に含まれる sloth、アライグマのサンプルが一番目的に近いので、まずそちらを動かします。

Unity のバージョンですが、実は少しややこしい話があります。後述の ARKitRemote というデバッグ用アプリが新しめの Unity だと動かないため、LTS の 2017.4 系最新が無難そうです。本書では 2017.4.7f1 という（中途半端に）古めの Unity で動作確認をしています。

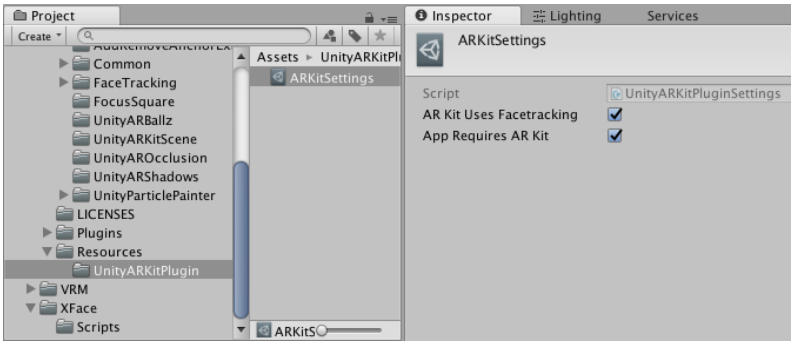
ARKit Face Tracking そのものは 2017.4 系〜2018.2 系でチマチマ動かしていますが設定等や使い勝手に特に差はない認識です。.NET 4.x 対応の新しい C#で書きたい人は、ARKitRemote だけを古い Unity でビルドしてインストールしたのち、アプリの開発には 2018.2 系を使うのをオススメします。

3.1.1 プロジェクト作成と Unity ARKit Plugin

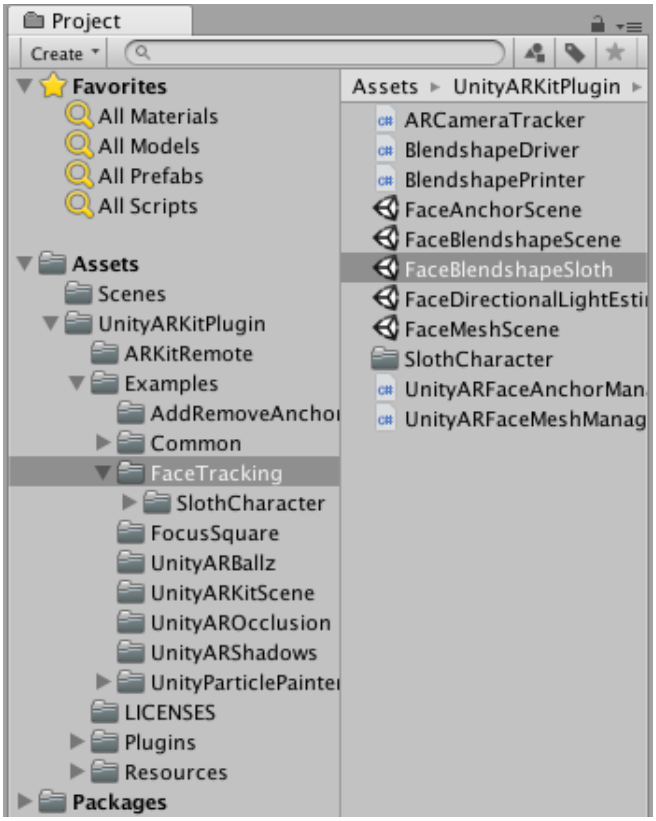
▲図 3.2　Unity ARKit Plugin

- iPhone X と OS バージョンに対応する Xcode がインストールされた Mac で Unity を起動します。
- 新規に Unity 3D プロジェクトを作成します。
- 「File ＞ Build Settings...」から iOS に「Switch Platform」します。
- Asset Store で「Unity ARKit Plugin」をダウンロードし（図 3.2）、import します。
- 「UnityARKitPlugin ＞ Resources ＞ UnityARKitPlugin ＞ ARKitSettings」を選択し、「AR Kit Uses Facetracking」と「App Requires AR Kit」のチェックを ON にします（図 3.3）。
- 「Edit ＞ Project Settings ＞ Player」から、「Other Settings」を開き、「Target minimum iOS Version」を ARKit Face Tracking にあわせて 11.0 にします。

- 「UnityARKitPlugin/Examples/FaceTracking/FaceBlendShapeSloth」シーンを開きます（図 3.4）。
- 再度「File ＞ Build Settings...」から「Add Open Scenes」を選択し、FaceBlendShapeSloth シーンを追加後、「Build And Run」を選択します。
- 初回ビルド時には Xcode プロジェクトの生成先のパスを聞く、フォルダ選択ダイアログが出てきます。"Xcode"等、適当なフォルダ名を入力しましょう。



▲図 3.3　ARKitSettings



▲図 3.4　FaceBlendShapeSlot シーン

Unity でビルド成功後、Xcode で引き継いだビルドがうまく行けば iPhone X 上で自分の顔をトラッキングするアライグマが表示されます（図 3.1）。初回は Xcode 上で Signing エラーが出ます。有効な Apple ID を Xcode に設定しておけば、無料で Personal Team の署名ができるはずなので、そのあたりは適当に調べてみてください。

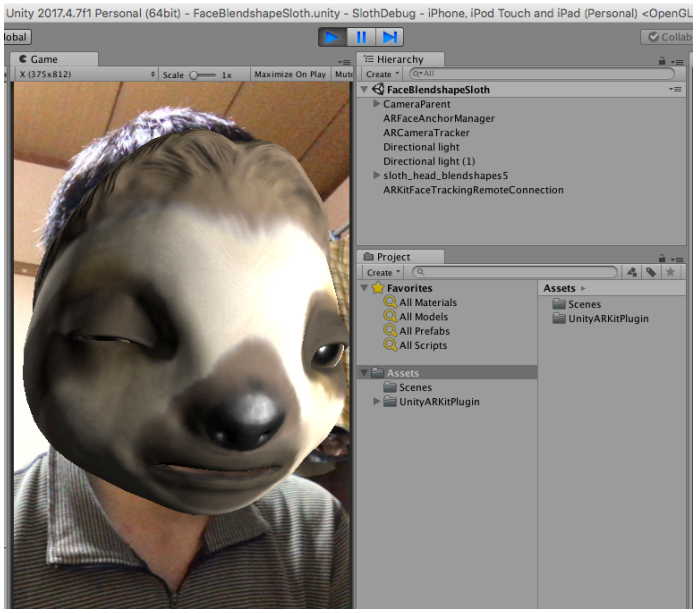
また、Unity ARKit Plugin は BitBucket で OSS として公開されています[*1]。ARKit の最新の機能を使う際にはこちらを使う必要があるタイミングもありますが、本件においては Asset Store からダウンロードできるバージョンで問題ありません。

Unity ARKit Plugin は全体としては MIT ライセンスで、SlothCharacter 配下のみ UCL（Unity Companion License）ライセンスと記載されています。後者はざっくりで言うと「Unity として組み込んで際に限っては特別の制限はないよ」というライセンスに

[*1] https://bitbucket.org/Unity-Technologies/unity-arkit-plugin

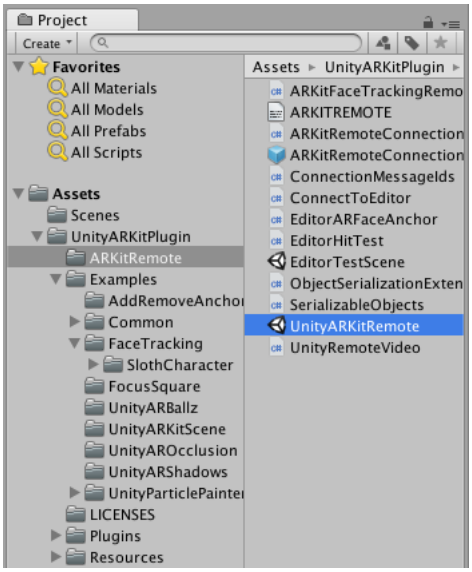
読めます。Unity ARKit Plugin 配下の通常ロジックのソースコードは MIT ということで、サンプルに付属するコードの改造・再配布は問題がなさそうです。

3.1.2 Unity エディタ上でリモートデバッグができる ARKitRemote



▲図 3.5　ARKitRemote で sloth アライグマサンプルを Unity エディタ上で確認

iPhone X 上でデバッグ専用のアプリを動かして、Unity エディタ上でトラッキング結果を受け取る ARKitRemote というツール（仕組み）があります。手順がそれなりに複雑で、接続相性や安定性の観点でもあまりよい子とは言えないものですが、iPhone 実機ビルドを経ずに確認ができるというメリットのために、トライすべきです。



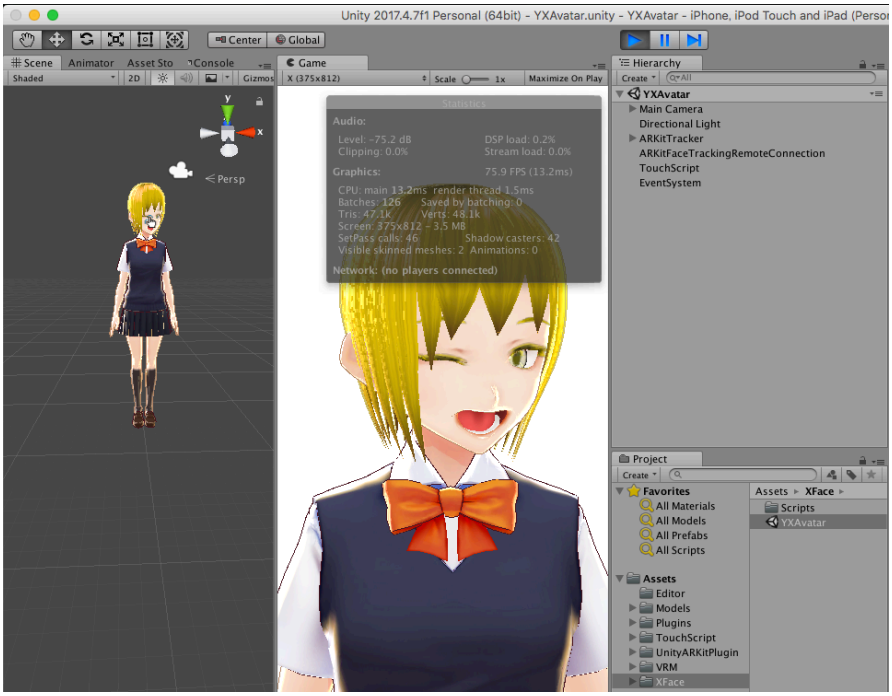
▲図 3.6　ARKitRemote シーンの格納パス

ところがこの ARKitRemote、Unity2018.2 あたりから Native まわりの動きが変わったのか、少し古めの Unity でビルドする必要がありますのでご注意ください。2017.4.x または 2018.1.x なら大丈夫だと思われます。

- 「File ＞ Build Settings...」から「UnityARKitPlugin/ARKitRemote/ARKitRemote」のみチェックを ON にします。
- 「Development Build」のチェックを ON にします。
- 別アプリとしてビルドするために、「PlayerSettings」の「Product Name」を"ARKitRemote"に、「Other Settings」の「Bundle Identifier」を"com.unity.arkitremote"あたりの別名に変えておきます。
- 「File ＞ Build and Run」を選択し、Xcode ビルドまで成功後、起動するのを待ちます。今後、動きが怪しいことになった際には ARKitRemote アプリのタスクキル＋再起動が一番復帰率が高いことを覚えておいてください。
- Unity エディタ上で「UnityARKitPlugin/Examples/FaceTracking/FaceBlendShapeSloth」シーンを開き、再生します。
- 「Console」の Editor から「Unknown: iPhone (pid なんたら」となっているデバイスを選びます。Lightning ケーブルで接続された iPhone X が正常に認識されていると表示されます。

結局のところ、この ARKitRemote デバッグアプリと Unity エディタは IP 通信をしているだけのため、無線経由でも動くのですが、どうもカメラ映像を送信するところが負荷が高いらしく、Lightning ケーブル直結を強く推奨します。また、実は iTunes をインストールした Windows 上の Unity エディタでも動きます。

3.2 3D キャラクターアバターアプリ YXAvatar とリポジトリ



▲図 3.7　YXAvatar が ARKitRemote 経由で動作している様子

続けて、ARKit Face Tracking の機能をつかって、3D キャラクターを動かすアバターアプリをつくってみましょう。

完成品の Unity プロジェクトのソースコード一式を以下のリポジトリで公開しております。

- `https://github.com/youten/YXAvatar`

UnityARKitPlugin, UniVRM, TouchScript, VRoid から出力した VRM の 3D モデルと、ライセンス上再配布できないものがなさそうだったため、全てリポジトリに含めました。clone 後、すぐ動かすことが（おそらく）できます。

3.3 AR 要素をサボってかんたんにする

YXAvatar で実際に組み込まれている機能について、順番に説明していきます。

ARKit Face Tracking API 群としては色々なことができるのですが、3D モデルに適用して遊ぶ際には「顔の向き・目・口」のみを扱うようにだけ絞ると、実装をかんたんにすることができます。

3.3.1 現実世界のカメラ画像は扱わない

`UnityARVideo`等のカメラまわりは使わないので適用しません。Unity 通常のカメラの正面にモデルを配置して、近距離でも歪まないように FoV を 30 あたりの少し低めに設定します。

3.3.2 AR 的な移動は扱わない

前述した `ARConfiguration.WorldAlignment`の話ですが、サンプルでは重力の向きに追従する設定になっている箇所を、iPhone のカメラ（ディスプレイ）の向きに追従する設定に変更します。これにより、現実世界の動きにカメラのプロジェクションをあわせる話を考えなくて済むようになります。

ARSession まわりの初期化コードのうち、`UnityARAlignment.UnityARAlignmentGravity`になっているところを、`UnityARAlignment.UnityARAlignmentCamera`に変更します。このあたりがよくわからない際には、「What does UnityARAlignment mean in Unity ARKit Plugins? [*2]」と Apple 公式 doc をあわせて読むと理解が進むかもしれません。

▼リスト 3.1　ARKitTracker.cs ARKit 初期化処理

```
m_session = UnityARSessionNativeInterface.GetARSessionNativeInterface();

Application.targetFrameRate = 60;
ARKitFaceTrackingConfiguration config = new ARKitFaceTrackingConfiguration();
// AR ではなくて Avatar システムとかだと非 Gravity が楽
config.alignment = UnityARAlignment.UnityARAlignmentCamera;
```

[*2] https://stackoverflow.com/questions/50716817/

```
config.enableLightEstimation = true;
```

3.3.3 顔の Mesh は使わない

「FaceBlendshapeScene」シーンにあるような、認識した顔表面の Mesh を取得してマスクのように表示できるあたりは扱いません。

3.3.4 顔の位置と角度はモデルの Position と頭の Rotation に適用

ARKit のトラッキング結果として取得できる顔の位置と角度は、それぞれモデルの Position と頭の Rotation に適用します。iPhone の正面に立って、キャラクターが鏡として動くように適当に符号や向きの入れ替えを行います。

▼リスト 3.2　AvatarTracker.cs 顔の位置と角度のトラッキング部

```
var p = new Vector3(
    _pose[ARKitPose.Index.PosY],
    -_pose[ARKitPose.Index.PosX],
    _pose[ARKitPose.Index.PosZ] * 1.5f);
var r = Quaternion.Euler(
    -_pose[ARKitPose.Index.RotY],
    -_pose[ARKitPose.Index.RotX],
    -_pose[ARKitPose.Index.RotZ] + 90.0f);

_poser.Update(p, r);
```

以下のコードでは、角度を特定ボーンのみに適用すると首がかなりグキッといってしまうところを、頭から首、背骨の方へ 4 ボーンの指定があった際には分散するようにしています。

▼リスト 3.3　Poser.cs Position と Rotation の適用部

```
_targetRoot.position = pos;

if (_targetHeads.Length == 1)
{
    _targetHeads[0].localRotation = _defaultHeadsRot[0] * rot;
}
else if (_targetHeads.Length == 4)
{
    _targetHeads[0].localRotation =
        _defaultHeadsRot[0] * Quaternion.Lerp(_defaultHeadsRot[0], rot, 0.5f);
    _targetHeads[1].localRotation =
        _defaultHeadsRot[1] * Quaternion.Lerp(_defaultHeadsRot[1], rot, 0.35f);
    _targetHeads[2].localRotation =
```

```
            _defaultHeadsRot[2] * Quaternion.Lerp(_defaultHeadsRot[2], rot, 0.1f);
        _targetHeads[3].localRotation =
            _defaultHeadsRot[3] * Quaternion.Lerp(_defaultHeadsRot[3], rot, 0.05f);
    }
```

3.3.5 50 種以上とれる表情 BlendShape は一部だけ使う

前章で紹介した通り、表情 BlendShape として取得できるパラメータは非常に多いのですが、一部だけ適用しましょう。ミニマムでは「口を開く jawOpen」「左右の目のまばたき eyeBlink_*」と、「左右の目の上下左右 eyeLook_*」をマッピングするぐらいでなんとかなります。

以下のコードでは加えて、「笑顔度」の mouthSmileを適用して、少しニンマリできるようにしています。

▼リスト 3.4　AvatarTracker.cs BlendShape トラッキング適用部

```
if (_targetModel == Model.VRoid)
{
    // jawOpen
    _targetBlendShape.SetBlendShapeWeight(VRoidMouthJoy,
        _jawOpenCurve.Evaluate(_pose[ARKitPose.Index.JawOpen]) * 100.0f);
    // eyeWide not supported
    // eyeBlink
    _targetBlendShape.SetBlendShapeWeight(VRoidLeftBlinkJoy,
        _eyeBlinkCurve.Evaluate(_pose[ARKitPose.Index.EyeBlinkLeft]) * 100.0f);
    _targetBlendShape.SetBlendShapeWeight(VRoidRightBlinkJoy,
        _eyeBlinkCurve.Evaluate(_pose[ARKitPose.Index.EyeBlinkRight]) * 100.0f);
    // mouthSmile
    _targetBlendShape.SetBlendShapeWeight(VRoidMouthFun, mouthSmile * 100.0f);
}

// hidarime
if (_leftEyeRot)
{
    float lookInLeft = _pose[ARKitPose.Index.EyeLookInLeft];
    float lookOutLeft = _pose[ARKitPose.Index.EyeLookOutLeft];
    float lookUpLeft = _pose[ARKitPose.Index.EyeLookUpLeft];
    float lookDownLeft = _pose[ARKitPose.Index.EyeLookDownLeft];
    _leftEyeRot.transform.localEulerAngles = new Vector3(
        (lookDownLeft - lookUpLeft) * 10.0f,
        (lookInLeft - lookOutLeft) * 15.0f,
        0);
}

// migime
if (_rightEyeRot)
{
    float lookInRight = _pose[ARKitPose.Index.EyeLookInRight];
    float lookOutRight = _pose[ARKitPose.Index.EyeLookOutRight];
    float lookUpRight = _pose[ARKitPose.Index.EyeLookUpRight];
```

```
        float lookDownRight = _pose[ARKitPose.Index.EyeLookDownRight];
        _rightEyeRot.transform.localEulerAngles = new Vector3(
            (lookDownRight - lookUpRight) * 10.0f,
            (lookOutRight - lookInRight) * 15.0f,
            0);
    }
```

3.4 VRoid でつくった 3D モデルを読み込む

ARKit Face Tracking のトラッキング結果をどう扱うか決めたあとは、最終的に 3D モデルに適用します。

ここでは、表情 BlendShape が一通り揃った VRoid Studio が出力する VRM が便利なのでお借りしました。

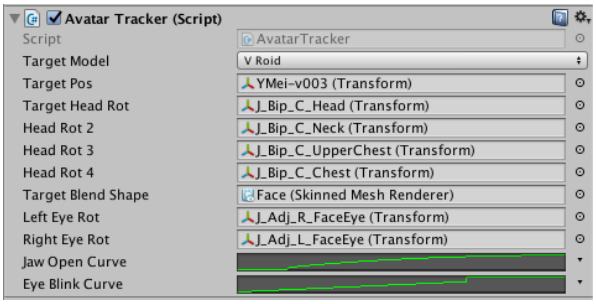
本書では VRoid Studio[*3]は v0.2.11 から出力した VRM を、UniVRM[*4]v0.43 で読み込んでいます。これらのバージョンについて、Unity エディタ上で import する際には手作業で修正できるので問題ないのですが、normal map のランタイムロードまわりが決着がついていない印象です。前述の Unity や ARKit Plugin がそれなりに古くとも問題ないのに対して、こちらの 2 つはリリース時期をあわせたバージョンで、なるべく新しいものを使うべきだと思われます。

また、せっかく VRM モデルを使うのであれば、表情など、VRM 仕様のインタフェースを使うべきではあるのですが、顔の表情については Joy, Angry, Sorrow, Fun など、コンテクスチュアルなインタフェースになっているため、少々申し訳なく感じつつも BlendShape や目ボーンの角度を直接操作しています。

[*3] VRoid Studio `https://vroid.pixiv.net/`

[*4] UniVRM `https://github.com/dwango/UniVRM`

3.5 制御対象のマッピングと Inspector 設定



▲図 3.8　AvatarTracker Inspector

ここまで説明してきた処理の対象となる Transform、BlendShape 等を AvatarTracker の Inspector から設定できるようにしています（図 3.7）。現状 1 体しか含めていませんが、複数のキャラクターを切り替えたりする際にはここや ARKitTracker の Avatar List を編集してください。

- Target Model：BlendShape 処理等を分岐するための識別子です。
- Target Pos：平行移動の対象となる Transform を指定します。
- Target Head Rot：角度適用の対象となる Transform を指定します。頭ボーンを想定しています。
- Head Rot 2,3,4：頭以降、角度を分散しながら伝播させていく Transform を指定します。頭から首、背骨、腰に向かって順番に設定することを想定しています。
- Target Blend Shape：BlendShape 適用先の Skinned Mesh Renderer です。表情モーフが複数の Mesh に分かれてるようなモデルではがんばって改造する必要があります。
- Left/Right Eye Rot：目の向きを角度で制御する際には、左右の目のボーンを指定します。BlendShape 制御の際には不要です。
- Jaw Open/Eye Blink Curve：唇の「ほんのちょっと開きプルプル」や目の「ギリギリ閉じきってないプルプル」をフィルタしています。Animation Curve の UI を Animation 以外に使えるのを最近知りました。便利です。

あとがき

本当はモーションデータをネットワーク上にぶん投げて、VR で繋ぐところまでやりたかったのですがそこまでは間に合いませんでした！

リポジトリにはちょっとだけ片鱗が含まれています。一応動いたりはしたのですが、IP アドレスまわりなど力技すぎるあたりをなんとかして、更新できたらな、と思っています（せっかくライセンス上問題ない形で最後まで作れるので）。

また、他の 3D モデルで遊んでみた際のコードも入っています。BlendShape で表情モーフがそれなりに実装されたモデルであればだいたい繋ぎこむことができますので、ぜひ好きなキャラクターでチャレンジしてみてください。

3D キャラクターのモーションをスマホ単体でつくりだすことができるという、夢のある時代になりました。きっとすぐに全身トラッキングの時代も来ますよね。

「VR コンテンツのつくりかた」シリーズとして、初代ぶりの gdgd な本となりました。悪しからず。どちらかというと前 2 冊が少々書きすぎたというのが正しくて、まぁ言うて薄い本てそういうものじゃないですかね…？

Twitter: @youten_redo
Web: ReDo -Refrigerator Door- れいぞうこのドア http://greety.sakura.ne.jp/redo/

VR コンテンツのつくりかた 4 ARKit Face Tracking

2018 年 10 月 8 日　技術書典 5 版 v0.9.0
2018 年 10 月 9 日　GitHub 公開版 v0.9.1
著　者　youten