VR コンテンツのつくりかた 4
ARKit Facial Tracking

youten　著

2018-10-08 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

2.1 Facial Tracking とは

2.2 ARKit とは

2.3 TrueDepth カメラと顔の位置検出

2.4 表情 BlendShape とトラッキング箇所

第 3 章

Unity で ARKit Facial Tracking で遊ぶ

3.1 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 だけ古い Unity でビルドして、アプリの開発は 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

- 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、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。

- 「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」を開きます。
- 再度「File ＞ Build Settings...」から「Add Open Scenes」を選択し、FaceBlendShape 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。

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。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。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。

https://bitbucket.org/Unity-Technologies/unity-arkit-plugin

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

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 という仕組みがありますので、まずは準備します。

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。

- 「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、起動するのを待つ。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く。
- Unity エディタ上で「UnityARKitPlugin/Examples/FaceTracking/FaceBlendShapeSloth」を開きます。
- 「Console」の Editor から「Unknown: iPhone (pid なんたら」となっているデバイスを選ぶ。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る。

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 AR 要素をサボってかんたんにする

ARKit は API 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、とても楽に扱うことができます。

3.2.1 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。

3.2.2 AR 的な移動は使わない

サンプルでは鉛直方向に追従する設定になっている箇所を、フロントカメラの向き（≒ iPhone のディスプレイに正対する向き）のみ扱うように変更します。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。

具体的には ARSession まわり初期化コードのうち、`UnityARAlignment.UnityARAlignmentCamera`を設定します。このパラメータの意味については「What does UnityARAlignment mean in Unity ARKit Plugins? [*1]」と Apple 公式 doc をあわせて読むのがわかりやすいです。

[*1] `https://stackoverflow.com/questions/50716817/`

▼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;
config.enableLightEstimation = true;
```

3.2.3　顔の Mesh は使わない

「FaceBlendshapeScene」シーンにあるような、顔表面のリアルなマスクなような Mesh を取得するまわりも扱いません。

3.2.4　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のを、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);
}
```

3.2.5 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);
}
```

3.3 VRoid でつくった 3D モデルを読み込む

ARKit Facial 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、VRM の仕様に対応するために表情 BlendShape が一通り揃った VRoid Studio が出力する VRM を利用します。

本書では VRoid Studio[*2]は v0.2.11 から出力した VRM を、UniVRM[*3]は v0.43 を利用しています。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 import 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。

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

[*3] UniVRM https://github.com/dwango/UniVRM

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入ってます。あと他のモデルで遊んでみた際のコードも入っています。アドレスや実装箇所が力技すぎるあたりをちょっとどうにかして、リポジトリは更新できたらなぁ、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）。

どちらかというと前 2 冊が少々書きすぎた感があったのですが、初代ぶりの gdgd な本となりました。まぁ言うて薄い本てそういうものじゃないですかね…？（ごめんなさい）

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 Facial Tracking

2018 年 10 月 8 日　技術書典 5 版 v0.9.0
著　者　youten