

# google/mediapipe で始 めるARアプリ開発

## レギュラートーク (20分)

noppe

# 対象

- ARアプリに興味がある
- トラッキングに興味がある
- 機械学習分からなくても問題ありません。

# noppe

- 株式会社ディー・エヌ・エー
  - ソーシャルライブアプリ Pococha
- 個人開発者
  - vear  
ReplayKitでスマホから配信できる  
VTuberアプリ
- iOSDC18,19
- きつねが好き



CONCEPT

# Live Link Life

## 今この瞬間を、いつまでも

「Live」 というその時その場所でしか生まれない、一瞬一瞬の喜びと感動の一つ一つが重なり合い、そして積み重なっていく。それはライバーにとっても、リスナーにとっても、かけがえのない瞬間となり、居場所となり、思い出の1ページとして刻まれていく。

人と人が「Link」し、「Live」が「Life」へと「Link」する。

人と人、心と心、その時にしか訪れない瞬間の「つながり」や、そこから生まれるかけがえのない「よろこび」を大切にしたいという私たちの想いが「Live Link Life」というコンセプトに込められていく。



**vear - バーチャルライブ配信アプリ** 12+

VRM アバターで簡単VTuberカメラ！

Tomoya Hirano

★★★★★ 4.7 • 149件の評価

無料・App内課金が有ります

---

**iPhoneスクリーンショット**

The image displays four iPhone screenshots showcasing the vear app's virtual reality features. Each screenshot shows a different 3D VRM character in various environments:

- Screenshot 1:** A white-haired character with blue glasses and a black hoodie, wearing a pink bunny ears headband.
- Screenshot 2:** A red-haired character with bangs, wearing a dark blue patterned top.
- Screenshot 3:** A character with dark hair and bangs, wearing a black and white patterned top. This screen also shows a bottom navigation bar with filter and accessory icons, and a preview of five other filter options: origin, bit, memory, deflag, and 2.5a.
- Screenshot 4:** A blonde character with a yellow braid and a black mask over their eyes.

Each screenshot includes standard iOS camera controls at the bottom: a portrait mode icon, a shutter button, and a star icon.

# noppe

- 株式会社ディー・エヌ・エー
  - ソーシャルライブアプリ Pococha
- 個人開発者
  - vear  
ReplayKitでスマホから配信できる  
VTuberアプリ
- iOSDC18,19
- きつねが好き



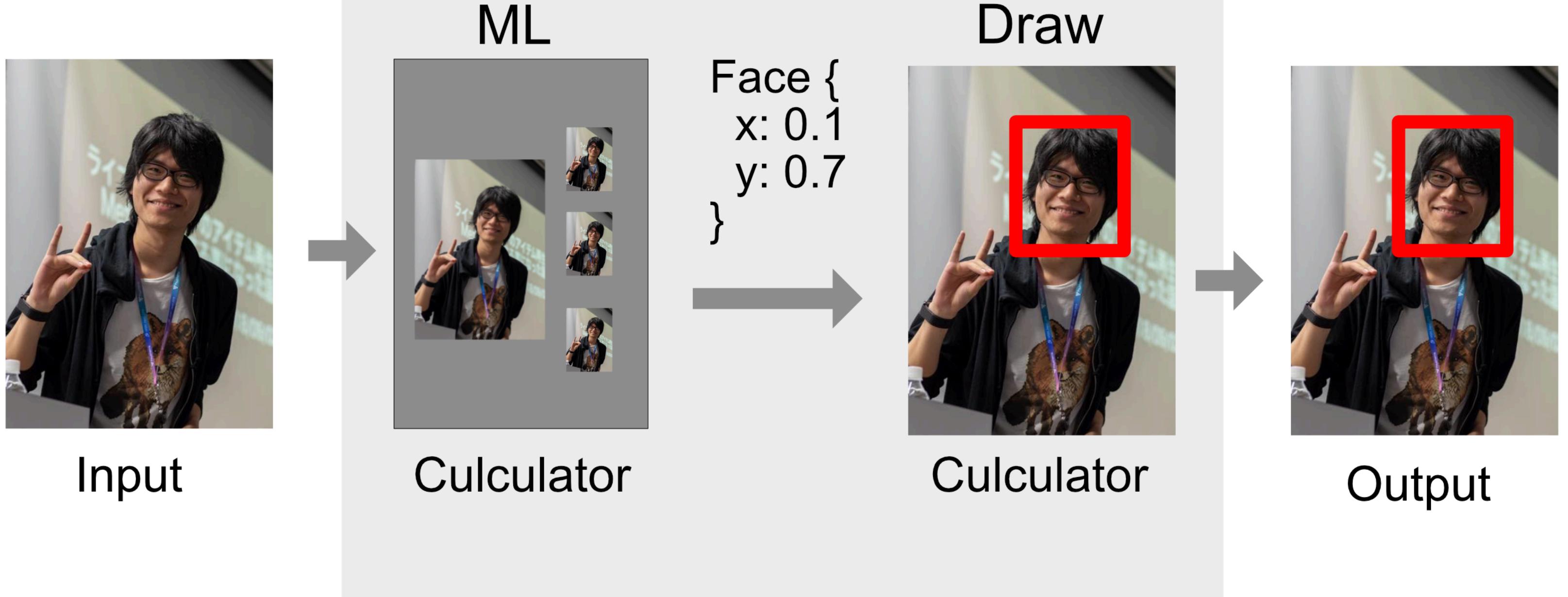
# mediapipe とは？

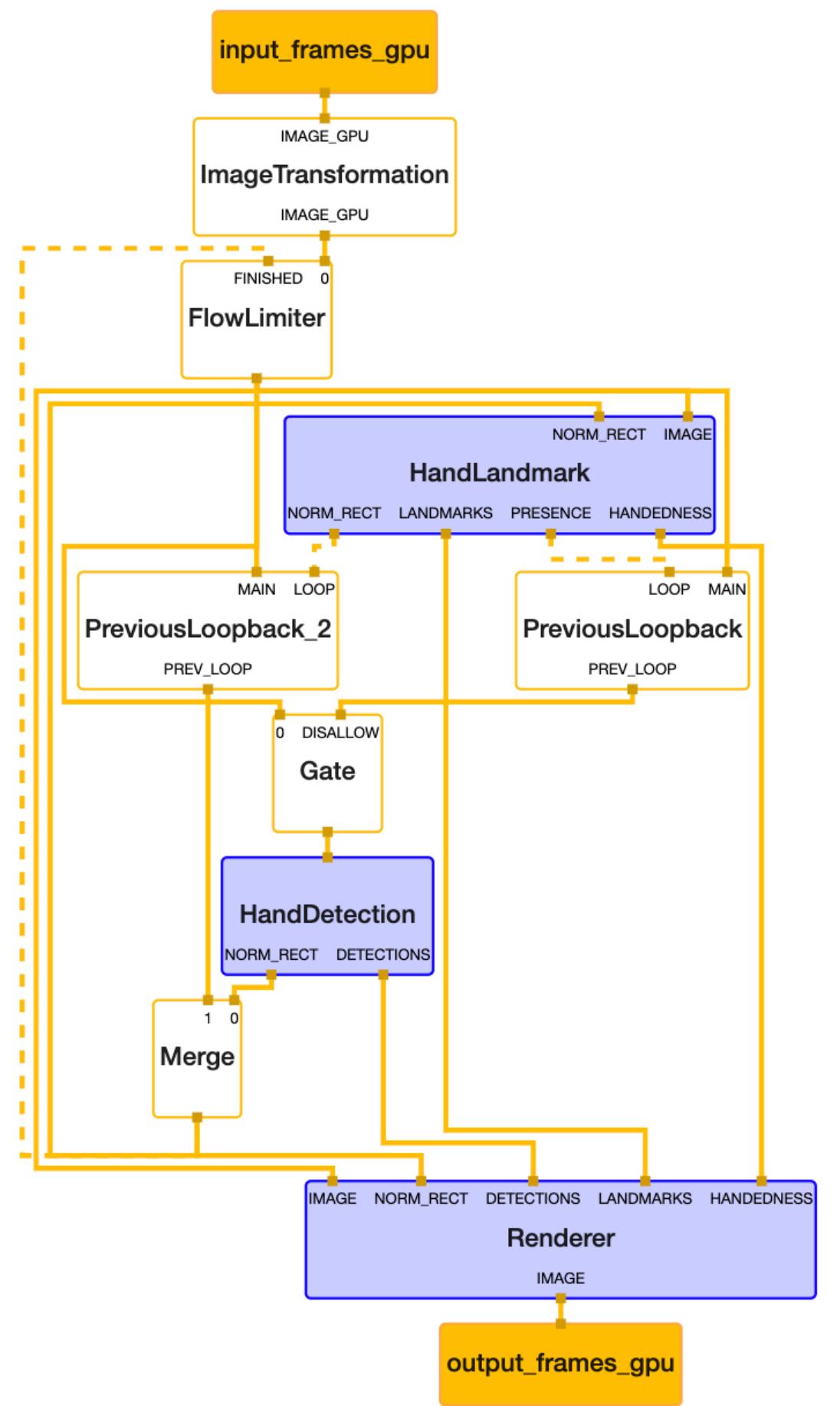
# mediapipe

[github.com/google/mediapipe](https://github.com/google/mediapipe)

- 処理(Culculator)を繋げてパイプライン (Graph) を構築するツール
- 主に画像処理のパイプライン構築に使われる
- ML処理を混ぜ込む事が出来る

# Graph

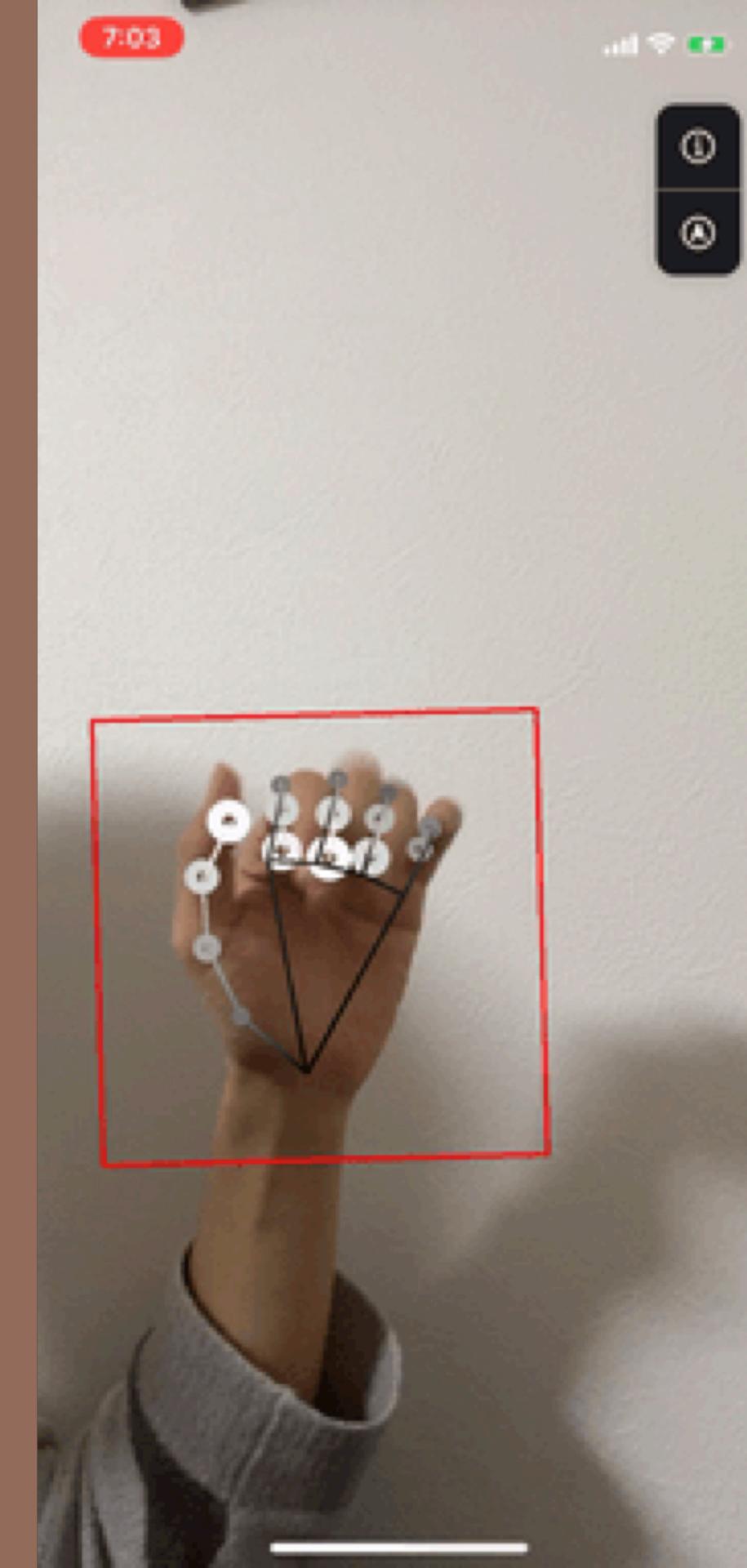




# mediapipe

Calculatorは、実装済みの物も使える

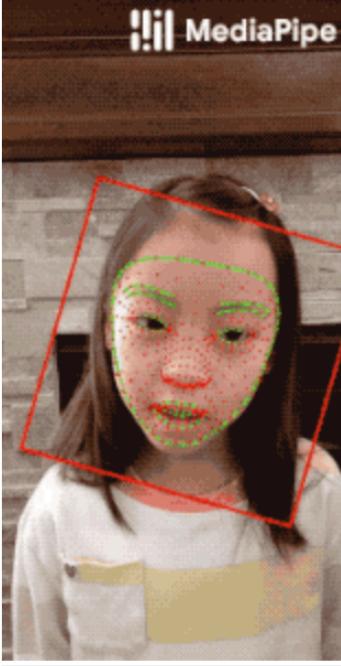
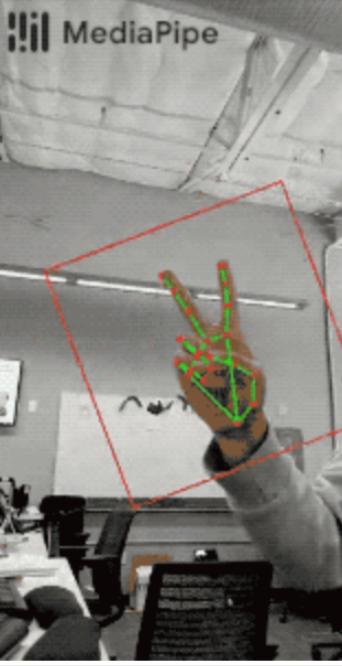
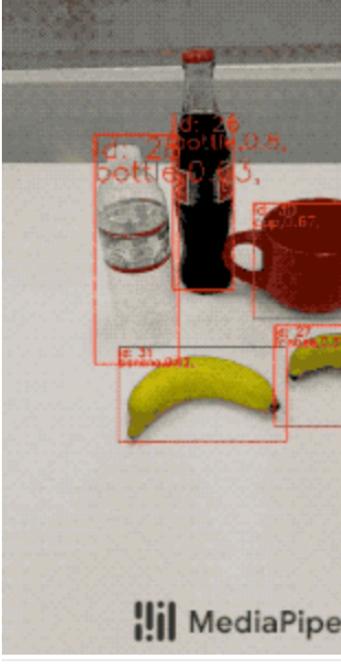
- LuminanceCalculator  
RGBを受け取り、輝度画像を出力する
  - SobelEdgesCalculator  
Sobelフィルタをかけた画像を出力する
- ...etc



# 実装済みのGraph

これらの実装はすぐに使える<sup>ios</sup>  
今日はこれを使う流れを解説

<sup>ios</sup>一部iOSでは実装がないものもある

Face Detection	Face Mesh	Hands	Hair Segmentation
			
Object Detection	Box Tracking	Objectron	KNIFT
			

# アジェンダ

- グラフのビルド
- ARKitとの連携
- できる事・できない事

# グラフのビルト

[github.com/noppefoxwolf/HandTracker](https://github.com/noppefoxwolf/HandTracker)

今回使うソースコード



# Graph

- 構築したGraphはバイナリとしてビルドすることができる
- バイナリをObjCから使う実装があるので、それらを使ってフレームワークを作れる事ができる。
- 今日はデモで紹介したハンドトラッカーをフレームワークとしてビルド





Watch ▾

362

Code

Issues 193

Pull requests 6

Actions

Projects

Wiki

Security

Insights

## mediapipe / xcodeproj

- > [mediapipe/examples/android/src/java/com/google/mediapipe/apps/upperbodyposetrackinggpu/MainActivity.java](#)
- [mediapipe/examples/android/src/java/com/google/mediapipe/apps/objectdetection3d/assets/chair/model.obj.aaa](#)
- [mediapipe/examples/android/src/java/com/google/mediapipe/apps/objectdetection3d/assets/sneaker/model.obj.aaa](#)

# グラフのビルド

ビルドツールを使ってビルドを行う

- bazelbuild/bazel (ベイゼル)
  - KubernetesやTensorFlowで採用

Track E - Ryo Aoyama

**Bazelを利用したMicro Modular Architecture**

# グラフのビルド

bazelのインストールはスクリプト落としてきて実行するだけ

```
curl -L0 "https://.../bazel-3.2.0-installer-darwin-x86_64.sh"
```

```
chmod +x "bazel-3.2.0-installer-darwin-x86_64.sh"
```

```
./bazel-3.2.0-installer-darwin-x86_64.sh
```

# グラフのビルド

Xcodeでもビルドができる<sup>tulsi</sup>

- bazelbuild/tulsi

---

<sup>tulsi</sup> BuildPhaseScriptでbazelを叩くだけなので、必須ではない

# グラフのビルド

tulsiはビルドスクリプト叩くだけでインストールできる（要bazel）

```
git clone git@github.com:bazelbuild/tulsi.git  
./build_and_run.sh
```

# そのほか必要なもの

- Xcode
- 時間
- ネットワーク

# ビルドの流れ

1. フレームワークのコードを書く
2. BUILDファイルに成果物の情報や依存性などを記述する
3. bazelでbuildする
4. frameworkが出来上がる

フレームワークのコード  
を書く

# フレームワークのコードを書く

1. binarypbを読み込んでMPPGraphを作る
2. MPPGraphを開始する
3. MPPGraphに画像を送る
4. delegateで結果を受け取る

# framework



→  
send(:)

→  
send(:)

mediapipe

MPPGraph

→  
delegate

→  
delegate

Lndmark {  
x: 0.1,  
y: 0.6,  
z: 0.3  
}

# binarypbを読み込んでMPPGraphを作 る

```
+ (instancetype) init {
    NSURL* url = [NSURL ...@"hand_landmarks.binarypb"];
    NSData* data = [NSData dataWithContentsOfURL:url options:0 error:nil];
    mediapipe::CalculatorGraphConfig config;
    config.ParseFromArray(data.bytes, data.length);
    self.graph = [[MPPGraph alloc] initWithGraphConfig:config];
}
```

# MPPGraphを開始する

Graph内で利用する各種コンポーネントの初期化が行われる

```
- (void)start {  
    [graph startWithError: nil];  
}
```

# MPPGraphに画像を送る

```
- (void)sendPixelBuffer:(CVPixelBufferRef)pixelBuffer {
    [self.mediapipeGraph sendPixelBuffer:pixelBuffer
        intoStream:@"input_video"
        packetType:MPPPacketTypePixelBuffer];
}
```

# delegateで結果を受け取る

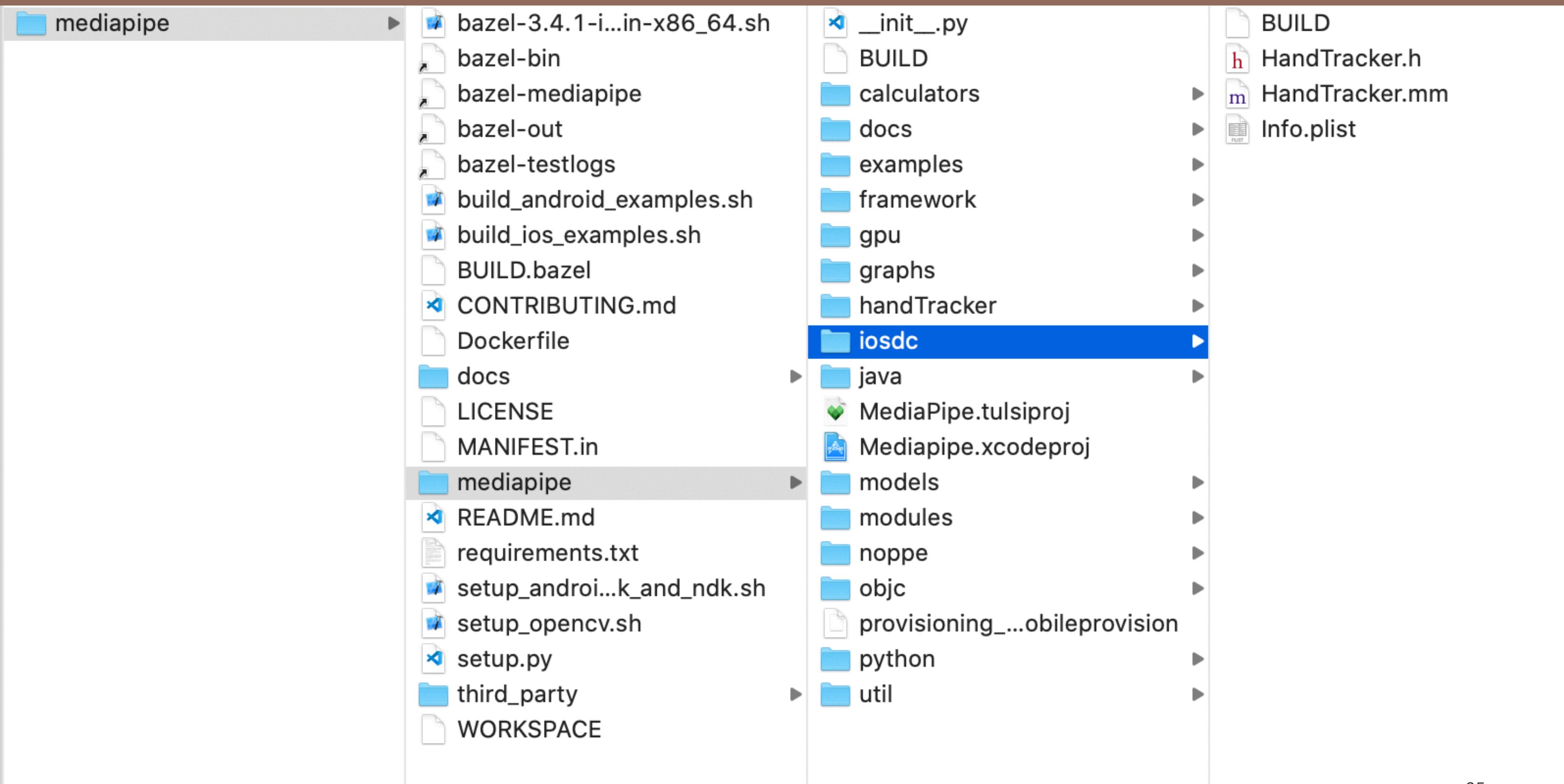
```
- (void)mediapipeGraph:(MPPGraph*)graph  
didOutputPacket:(const ::mediapipe::Packet&)packet  
fromStream:(const std::string&)streamName  
{  
    const auto& timestamp = packet.Timestamp().Value();  
  
    const auto& landmarks = packet.Get<::mediapipe::NormalizedLandmarkList>();  
  
    NSArray *landmarkObjects = [self toObject: landmarks];  
  
    [delegate didOutputLandmarks: landmarkObjects];  
}
```

# BUILDファイル書く

# BUILDファイルとは

bazelでビルドするファイルや依存関係を記述したファイル

```
objc_library(  
    name = "HandTrackerLibrary",  
    hdrs = ["HandTracker.h"],  
    srcs = ["HandTracker.mm"],  
    data = [  
        "hand_tracking:hand_tracking_mobile_gpu_binary_graph",  
        ...  
    ],  
    deps = [  
        "//mediapipe/objc:mediapipe_framework_ios",  
        "//mediapipe/objc:mediapipe_input_sources_ios",  
        "//mediapipe/graphs/hand_tracking:mobile_calculators",  
        "//mediapipe/framework/formats:landmark_cc_proto",  
    ],  
),  
)
```



# bazelでビルドする

```
$ bazel build mediapipe/iosdc:HandTracker
```

# frameworkが出来上がる

githubに置いておきました

<https://github.com/noppefoxwolf/HandTracker>

# ARKitとの連携

# ARKitとの連携

ARKitの仕様上出来ない事をmediapipeで補うことで、表現力を解放できる

- AR空間上のオブジェクトを手で操作する
- 検出したオブジェクトの横にキャラクターを配置する
- 顔の骨格を補正しながら髪色を変える<sup>hair</sup>

---

<sup>hair</sup> Hair Segmentationは現状iOS非対応ですが

# ARKitとの連携アプリ開発

こんなものを作ります



# ARKit連携の流れ

1. ARFrameからcaptureImageを取り出す
2. PixelFormatを変換
3. Trackerへ送る
4. Delegateで結果を受け取る
5. RealityKitのボタンを上下・カウントを表示



# ARFrameからcaptureImageを取り出す

- captureImage = カメラからの映像

```
arSession.delegate = self  
...  
// ARSessionDelegate  
func session(_ session: ARSession, didUpdate frame: ARFrame) {  
    let captureImage: CVPixelBuffer = frame.capturedImage  
}
```

# PixelFormatを変換

画像データがどのように格納されているかを表している

- ARKitでは**YCbCr**形式
- mediapipeは**BGRA**のみを受け取れる  
→変換の必要がある

# kCVPixelFormatType\_32BGRA

1ピクセル32bitで表現するフォーマット

B:FF G:00 R:00 A:FF → 赤

# kCVPixelFormatType\_420YpCbCr8BiPl anarFullRange

- 2枚のY(8bit)+CbCr(16bit)の組み合わせで1フレームを表現するフォーマット
- 軽量なのでビデオデータで使われることが多い

# YCbCrからBGRAへの変換

- Accelerate
  - vImage
- Metal
  - MSL

GPUを使う分Metalの方が高速

# YCbCrからBGRAへの変換

次の計算式で変換ができる

$$R = Y + 1.402 \times Cr$$

$$G = Y - 0.344136 \times Cb - 0.714136 \times Cr$$

$$B = Y + 1.772 \times Cb$$

$$A = 1$$

# BlueDress

YCbCrからBGRAに高速コンバートするライブラリ

<https://github.com/noppefoxwolf/BlueDress>

# Trackerへ送る

```
let captureImage = frame.capturedImage  
let bgraCaptureImage = converter.convertToBGRA(captureImage)  
handTracker.send(bgraCaptureImage)
```

# Delegateで結果を受け取る

関節の位置はX,Y,Zで取れます…が、世界座標系では無く

X,Y: Screen座標系

Z: 手首からの奥行き

しか取れません。残念！

3



4



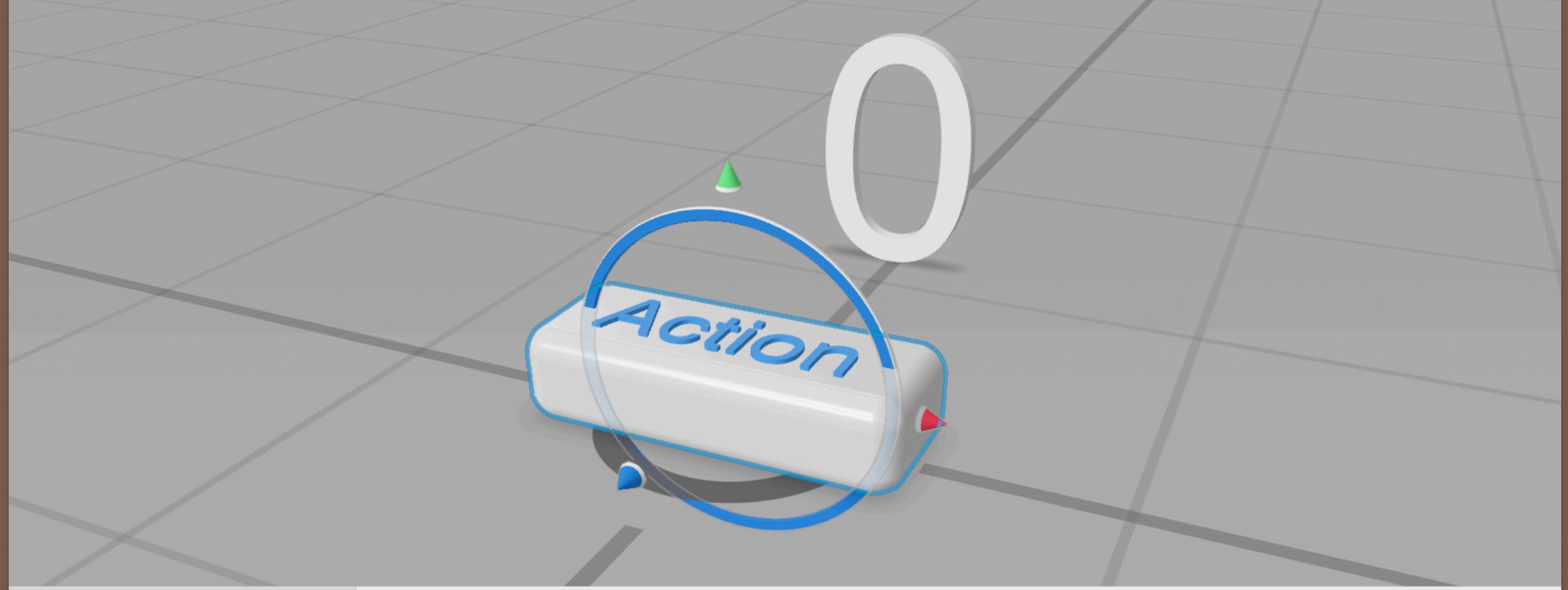
# RealityKitのボタンを上下・カウントを表示

今回は人差し指の延長線上にEntityがあれば、`isPress`をtrueにする

```
func handTracker(_ handTracker: HandTracker!, didOutputLandmarks landmarks: [Landmark]!) {  
    let indexFinderPosition = landmarks[8]  
    let screenLocation: CGPoint = ... (indexFinderPosition)  
    self.isPress = arView  
        .entities(at: screenLocation)  
        .contains(where: { $0.id == self.buttonEntity.id })  
}
```

# RealityKitのボタンを上下・カウントを表示

```
private func press() {  
    self.scene.notifications.press.post()  
    count += 1  
    changeText("\(count)")  
}
```



ビヘイビア +

● Press

● UnPress

トリガ

通知

識別子  
press

Appのコードから上の通知を送信すると、アクションシーケンスが開始します(Xcode)。

アクションシーケンス + | ☰ ▶

移動/回転/拡大（絶対）

イージングタイプ  
イーズイン

位置  
X: 0 cm Y: -2 cm Z: 0 cm

回転

▼ アクセシビリティ



できる事できない事

# 気になるところ

- HandTrackerはカメラからの距離が取れるわけではない事に注意  
→Irisでは奥行きが取れる
- 各関節の回転も取れない  
→自分で計算して自然に見せる工夫が必要
- フレームワークサイズがまあまでかい  
→100MBくらい

# いいところ

- 実装済みのグラフが非常に質が高い
- OSS
- マルチプラットフォーム

# やってみたいこと

- HairSegmentationなどをiOSに対応していないのは何故なのか調べたい
- NSObjectを経由しなければ高速化が望めるのでは
- ランドマークだけを出力するグラフを作ってみたい

ご清聴ありがとうございました