2020

Alchera Inc.

Alchera Engine Developer Guide

- Alchera Engine 2020.1.0

Android, iOS, Windows, MacOS에서
얼굴 & 손 인식 AR 애플리케이션을 제작하기 위해 Alchera Unity SDK를 이용하세요.
이 문서는 AlcheraSDK Engine을 Unity3D에서 사용하는 방법에 대하여 설명합니다.

Unity3D의 메뉴에서 Assets - Import Package - Custom Package... 를 통해
AlcheraUnityPlugin_2020.1.0.unitypackage를 Import 한 뒤,
Assets/Alchera/Example/01.Scenes/ 에 있는 데모 Scene을 확인해보세요.

Alchera

Alchera Engine Developer Guide Contents

Alchera Engine 2020.1.0

Alchera

Alchera Engine Rivision

Rivision History

<2019.12.3> AlcheraEngine_2019.7.3 - 배포.
<2019.12.16> AlcheraEngine_2020.1.0 - 배포.

Alchera API Reference

0. Getting Started

Alchera Unity SDK는 모바일에 최적화된 엔진이며, 모바일 기기의 전·후면 카메라와 Landscape Right, Landscape Left, Portrait 방향을 지원합니다.
Unity3D를 이용하여 쉽게 iOS, Android, Windows, MacOS를 지원하는 멀티플랫폼 Face & Hand Detection AR 애플리케이션을 만들 수 있습니다.
Alchera Unity SDK 개발이 처음이시라면, 먼저 예제 데모 Scene을 확인해보세요.

1. Scenes

포함된 Scenes

```
├── CaptureScene
├── Face2DFacemark
├── Hand2DSkeleton
├── ComplexScene
├── Face3DSticker
├── Face3DAnimoji
├── Face3DMask
├── FaceARKitScene
└── UI
    ├── DemoUI
    └── Splash
```

Alchera Unity SDK 개발이 처음이시라면, 데모 Scenes를 기반으로 제작하면 편리합니다.
Assets/Alchera/Example/01.Scenes/ 에 있는 데모 Scenes로 사용 실례를 볼 수 있습니다.

1.1 CaptureScene

AlcheraSDK의 네이티브 라이브러리를 사용하지 않고, 유니티 C#으로만 이루어진 Scene입니다.
카메라를 통해 받은 입력을 quad에 보여주는 기능이며, 부가적으로 Swap기능, 이미지 캡처 기능이 있습니다.
이 Scene은 AlcheraSDK 네이티브 라이브러리를 호출하지 않기 때문에 License 에러가 나지 않으며, 따라서 처음 package import 시 카메라나 다른 C# 에러를 확인하는 목적으로 실행하시는 것을 권장합니다.
이후의 예제들은 이 CaptureScene 에서 얻을 수 있는 ImageData 구조체를 이용하여 AlcheraSDK 네이티브 연산을 수행합니다.
(데모 앱에는 포함되어 있지 않습니다)

1.2 Face2DFaceMark

카메라를 통해 얻어진 ImageData를 이용하여 얼굴의 Landmark를 계산하고, 결괏값을 prefab으로 quad 위에 위치시키는 Scene입니다.
FaceService.cs 의 Need3D를 UnCheck하여 2D Landmark 추출 연산만 수행하도록 합니다.

1.3 Hand2DSkeleton

카메라를 통해 얻어진 ImageData를 이용하여 손의 정보를 계산하고, 결괏값을 prefab으로 quad 위에 위치시키는 Scene입니다.
HandService.cs 의 Need3D를 UnCheck하여 2D 연산만 수행하도록 합니다.
Landmark와 Region Box, Gesture, leftOrRight 값도 확인할 수 있습니다.
(단 현재 버전은 Hand 3D 기능을 제공하지 않습니다)

1.4 ComplexScene

Hand와 Face를 동시에 사용하는 시나리오에서 사용하는 Scene입니다. 데모의 간결함을 위해 두 연산은 선형적으로 수행되도록 C# 코드가 구성되었습니다. Face는 Need3D를 Check하여 3차원 계산도 같이 수행할 수 있습니다.

1.5 Face3DSticker

3차원으로 HeadPose를 계산한 뒤 결괏값을 통해 Prefab(Sticker)을 위치시키는 Scene입니다. 여러 스티커를 교체해가면서 쓰는 시나리오를 예상해 만든 Scene이므로, 화면을 touch 하면 다음 Prefab(Sticker)으로 교체됩니다. 반드시 Need3D를 Check하여 3D계산을 추가로 수행해야 하며, LevelOf3DProcess 를 HeadPose로 하여 HeadPose 계산 외 불필요한 연산은 수행하지 않도록 하는 것이 좋습니다.

- 스티커의 경우 스티커가 얼굴에 가려지는 효과를 내기 위해 Occlusion 기능도 Unity3D로 구현되어 있습니다. Unity Layer에서 마스킹 역할을 할 Prefab의 Layer를 "StickterOcclusionMask" 으로, 마스킹 당하는 Prefab을 "StickerOpague", "StickerTransparent"중 알맞게 설정하시면 됩니다. **(Scriptable Render Pipeline 기능을 사용합니다)**
- Demo 앱으로 볼 수 있는 스티커 Prefab은 모델 License 문제로 Alchera Unity SDK에는 포함되지 않습니다.

1.6 Face3DAnimoji

3차원으로 HeadPose및 BlendShape을 계산한 뒤 결괏값을 통해 Prefab(Panda)의 위치 및 SkinnedMesh를 변경시키는 Scene입니다. 반드시 Need3D를 Check하여 3D계산을 추가로 수행해야 하며, LevelOf3DProcess 를 HeadPose_BlendShape으로 하여 HeadPose, BlendShape 외 추가적인 연산은 수행하지 않도록 하는 것이 좋습니다.

(데모 앱에는 포함되어 있지 않습니다)

1.7 Face3DMask

3차원으로 HeadPose및 얼굴 마스크의 mesh를 vertex 단위로 제공합니다. 계산한 뒤 결괏값을 통해 Prefab(Mask)의 위치 및 Mesh를 변경시키는 Scene입니다. 반드시 Need3D를 Check하여 3D계산을 추가로 수행해야 하며, LevelOf3DProcess 를 HeadPose_BlendShape_Vertex로 하여야 동작합니다.

1.8 FaceARKitScene

Apple ARKit 에서 제공하는 Unity Interface와 같은 Interface로 제작한 Scene입니다. Event 기반으로 함수를 호출할 수 있으며, ARKit Unity Interface가 익숙하시다면 이 Scene으로 시작할 수 있습니다. 기존 데모와 Interface 맞추는 코드는 C#으로 노출되어 있습니다.

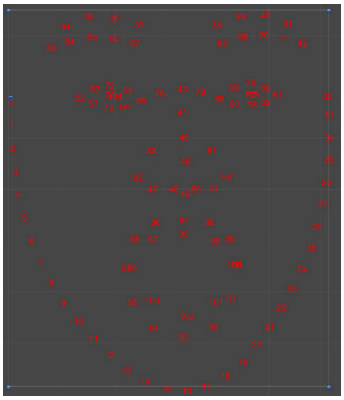
(데모 앱에는 포함되어 있지 않습니다)

1.9 UI - DemoUI, Splash

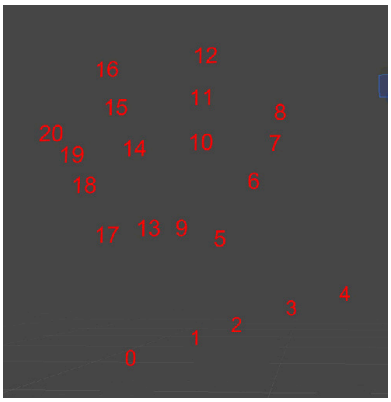
데모 앱에서 UI를 담당하는 Scene입니다. 유니티 C#으로 이루어진 코드입니다.

1.10 공통

- Face와 Hand의 Landmark 순서정보(그림1, 그림2)는 Assets/Alchera/Resources/에 가이드가 있습니다.



<그림1> Face Landmark



<그림2> Hand Landmark

- [Face/Hand]Service.cs의 MaxCount를 통해 한 번에 인식할 얼굴/손 의 최대 개수를 설정할 수 있습니다. 인원 제한은 없으므로, 앱의 성능과 용도를 고려하여 설정하시면 됩니다.
- 성능을 고려하여 초기에 미리 Prefab을 pooling하고, 인식하는 신체 갯수만큼 Prefab 보여주는 방식으로 구현되었습니다. 이는 Draw~.cs 코드에서 확인할 수 있습니다.
- Example/02.scripts/ 의 코드는 C# 예제로 작성된 것뿐이므로 다른 방식으로 구현해도 동작합니다.

▶ Hand에서 인식하는 Gesture 정보는 아래와 같습니다.

<그림3> Gesture (Gun)

<그림4> Gesture (Mini_heart)

<그림5> Gesture (Okay)

<그림6> Gesture (One)

<그림7> Gesture (Paper)

<그림8> Gesture (Rock)

<그림9> Gesture (Thumbs_up)

<그림10> Gesture (V)

2. Key Concepts

Alchera Unity SDK는 크게 다섯 단계를 진행합니다.

1. Initializing 2. Camera/Quad 3. Detecting 4. Rendering 5. Releasing

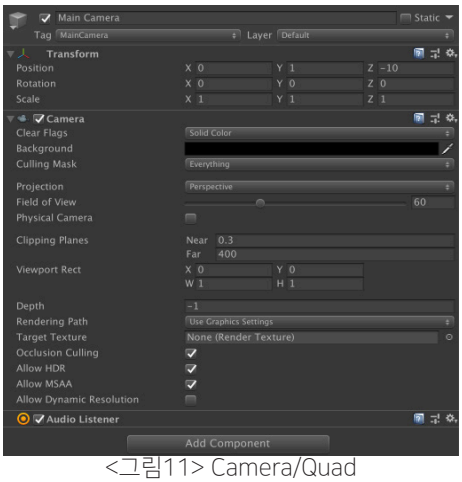
2.1 Initializing

Detection을 수행하기 위해서 먼저 Init을 호출해야 합니다. 기본적으로 2D Initializing을 수행하며, Face/HandService.cs 의 Need3D를 true로 설정했을 시 3D 라이브러리도 같이 초기화하게 됩니다.

2.2 Camera/Quad

Alchera Unity SDK는 Unity의 기본 카메라인 Main Camera를 사용합니다.
기본 카메라를 사용하기 때문에 다른 Application들과의 상호연동이 용이하며,
IOS/Android의 전·후면 카메라, Landscape Right, Landscape Left, Portrait 기기 방향 대응을 지원하고 있습니다. 이는 유니티 C#으로만 작성되어 있으므로 Native함수 호출 License가 없어도 CaptureScene 에서 확인할 수 있습니다.

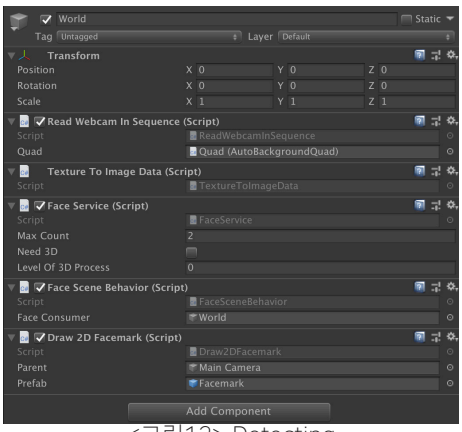
데모 예제들은 Quad에 Camera Input Texture를 입히고 Transform 값을 조절하는 방식으로 카메라 화면을 구현하며, 이 구현 방식은 AutoBackgroundQuad.cs에서 확인해 볼 수 있습니다. ReadWebcamInSequence.cs에서 카메라를 읽고, TextureToImageData.cs에서 Alchera SDK를 이용하기 위한 ImageData를 생성합니다.



<그림11> Camera/Quad

2.3 Detecting

[Face/Hand]Service.cs가 ImageData를 input으로 Detection을 호출하며
각각 FaceData, HandData 구조체로 결괏값을 받아오는 단계입니다.
~SceneBehavior.cs는 데모 앱의 주요 흐름 과정을 수행시켜주는 스크립트입니다.



<그림12> Detecting

2.4 Rendering

Draw~.cs 에서 Detection 후 보여줄 Prefab을 관리합니다. 시작 단계에서 Prefab을 미리 pooling 하여 사용합니다.
Prefab안의 ~Prefab.cs에서 Detection이 수행된 결괏값(FaceData, HandData)을 이용하여 연산 및 렌더링을 수행합니다.

2.5 Releasing

앱이 종료되거나 기능을 사용하지 않을 때, 메모리를 해제하는 과정입니다.

3. Unity Example Scripts

3.1 Assets/Alchera/SDK/

Interface 들과 Alchera 함수를 사용하는 [Face/Hand]Service.cs 가 있습니다.

[Face/Hand]Service.cs는 서로 유사한 형태로 구성되어 있습니다. Translator struct를 이용하여 [Face/Hand]Data및 parameter 가 있는 메모리를 미리 할당하고, Fetch를 통해 필요 시에 Detection 해서 나온 결과물에 후 처리를 한 뒤, 의미 있는 데이터가 담긴 메모리 만큼만 return 합니다.

3.2 Assets/Alchera/Example/02.Scripts

위 인터페이스들을 구현해 실질적인 예제를 담고 있는 스크립트입니다.

SceneBehaviorWorks/

기본적인 앱의 흐름을 제어하는 스크립트입니다. WebCamTexture로부터 ImageData를 생성하고, Detector가 있을 시 Detection을 수행하며, 해당 데이터들을 소비(Consume)합니다. Update() 함수에 넣지 않고 Start() for문 에 넣은 이유는 기기별 카메라 성능에 따라 업데이트마다 Camera texture가 업데이트 되지 않을 수 있음을 고려해서 Detection 사이클을 따로 관리하기 위함입니다.

TextureWorks/

ReadWebcam.cs : 카메라에 대한 전반적인 정보를 관리하는 스크립트입니다. WebCamTexture, 기기 orientation, front/rear, 플랫폼 등을 관리합니다.
ReadWebcamInSequence.cs : 카메라로부터 업데이트가 될 때마다 Task<Texture>업데이트를 하며, 이를 제어하는 스크립트입니다.
TextureToImageData : WebCamTexture를 받아 Alchera SDK에 필요한 형식인 ImageData를 return 하는 스크립입니다.
SaveLastTexture.cs : 텍스쳐를 이미지로 저장하기 위한 스크립트입니다.

DrawWorks/

Fetch 된 [Face/Hand]Data를 소비(Consume)하는 스크립트입니다. Prefab을 미리 생성하여 pooling하고, Detection된 신체 개수에 맞는 Prefab만 함수를 호출하는 용도입니다. 이 폴더는 용도에 맞게 복제해서 사용하길 권장합니다.

PrefabWorks/

DrawWork에 의해 관리되는 Prefab이 실질적으로 [Face/Hand]Data를 이용하여 기능을 수행하거나 렌더링을 하는 스크립트입니다. Prefab을 생성할 때 최상단에 EmptyObject를 만들고 그곳에 이 스크립트를 Component로 추가하여 이용합니다. 이 폴더는 용도에 맞게 복제해서 사용하길 권장합니다.

AutomatedWorks/

카메라 텍스쳐를 렌더링하는 quad가 카메라의 FOV, Screen width/Height, Device rotation, Platform, front/rear camera 등에 따라 자동으로 크기 및 위치가 변하도록 하는 스크립트입니다. quad 오브젝트에 Component로 추가되어 있습니다.

UIWorks/

데모 앱 UI를 담당하는 스크립트입니다.

DemoScript/

예제에 간단한 효과를 추가하기 위한 스크립트입니다.

ARKitInterfaceWorks/

호출 SDK를 한 번 더 wrapping 하여 Apple ARKit Unity Demo와 유사하게 맞춘 인터페이스를 이용하여 사용하는 예제 스크립트입니다. Apple ARKit Unity Demo에 익숙하신 분들은 쉽게 사용할 수 있으며, 아니라면 다른 예제 Scene 사용을 권장합니다.

4. Unity Settings

4.1 공통

▶ Unity 2019.2 에서 데모 Scene 제작. 데모 제작에 Universal Render Pipeline (LWRP)를 사용하였기 때문에, 이하 버전은 AlcheraSDK 함수는 동작하지만, Sharder나 Material이 깨지는 등 유니티 에러가 생길 수 있습니다.

▶ Tested External Camera : Logitech C920 webcam (1280x720)

▶ Edit - Project Settings - Graphics

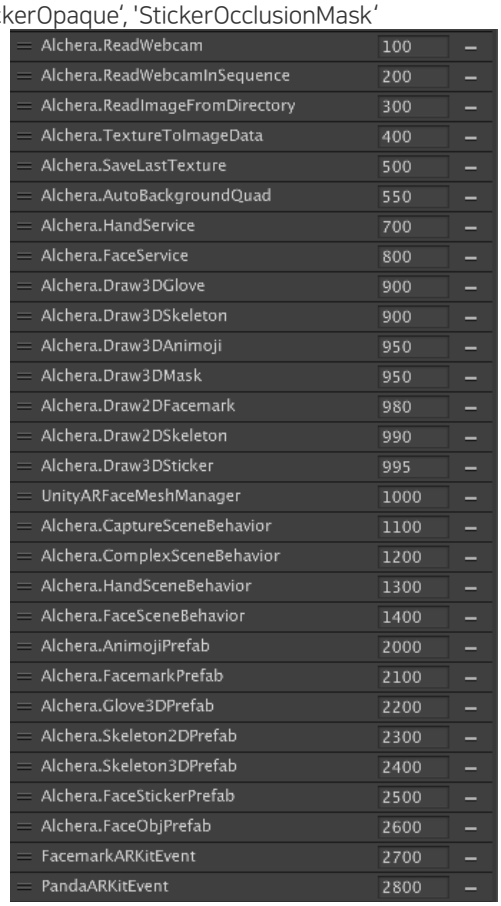
- Scriptable Render Pipeline Settings : LightweightAsset
- Tags and Layers : Tags : 'WebCamQuad' , Layers: 'StickerTransparent', 'StickerOpaque', 'StickerOcclusionMask'
- Script Execution Order (<그림13> 참고)

▶ Edit - Project Settings - Player - Resolution and Presentation

- Allowed Orientations for Auto Rotation - Portrait Upside Down : UnCheck

▶ Edit - Project Settings - Player - Other Settings

- Allow 'unsafe' Code : Check



<그림13> Script Execution

4.2 Windows

지원 Platform : x64

▶ Edit - Project Settings - Player - Other Settings

- Api Compatibiliity Level* : .NET 4.x

4.3 MacOS

지원 Platform : x86_64

4.4 Android

지원 Platform : armeabi-v7a, arm64-v8a

▶ Edit - Project Settings - Player - Other Settings

- Graphic APIs : OpenGLES2

  (2019.2버전에서 Vulkan, OpenGLES3는 유니티 내부 버그가 있어 예제구동은 되지 않습니다. Unity3D 문서에 수정 중으로 되어있습니다)[1]
- Write Permission : External (SDCard)
- Minimun API Level : 21. 구글 정책으로 Android 8.0 'Oreo' (API level 26) 권장
- Scripting Backend : IL2CPP
- Target Architectures : ARMv7a, ARM64

4.5 iOS

지원 Platform : armv7, armv7s, arm64

▶ Edit - Project Settings - Player - Other Settings

- Camera Usage Description : "카메라 권한설명 추가"
- Target Minimun iOS Version : 10.3

5. License

Alchera Inc. 에 연락 부탁드립니다.

1 Vulkan-UnityVideoplayer 미지원 여부 (https://docs.unity3d.com/ScriptReference/Video.VideoPlayer.html?_ga=2.161231871.331366313.1574922129-10526039.1572238404)

6. Alchera Classes

6.1 FaceLib

입력된 사진을 기반으로 2차원 얼굴을 찾고 얼굴의 landmark(눈, 코, 입 등 특징점 정보)를 찾습니다.
Detect 함수 수행 시 Detection 또는 Tracking이 수행됩니다. Detection은 전체 화면에 대하여 얼굴을 인식하고, Tracking은 이전 프레임의 얼굴 정보를 기반으로 얼굴 위치를 추적합니다. Detection의 수행 시간이 Tracking보다 크므로, 일반적으로는 Tracking을 주로 수행하고 그사이에 간헐적으로 Detection을 수행합니다.

Init(ref FaceLib.Context context)

FaceLib을 초기화합니다.

context.detectableSize : 일식하는 얼굴의 최소 픽셀 크기를 지정한다. 숫자가 작을수록 작은 얼굴도 검출하며, 너무 낮을 시 인식 속도가 늦어진다. (기본 128)

context.logStateMode : 라이브러리 동작을 확인하기 위한 디버그용 변수. 0으로 설정한다.

context.maxCount : Detect/track하는 최대 얼굴 개수를 정의한다. 이미지에 존재하는 얼굴 개수가 maxCount보다 클 경우 크기가 큰 얼굴부터 찾는다.

context.InitPath : 얼굴인식을 위한 deep learning network의 위치를 지정한다.

Detect(ref FaceLib.Context context, ref ImageData image, FaceData* facePtr)

2차원 이미지(ImageData image)를 입력으로 받아 해당 이미지에서 얼굴을 인식하여 인식한 얼굴의 FaceData 값을 저장합니다.

Unity의 Texture를 ImageData로 전환하는 함수가 예제의 TextureToImageData.cs에 정의되어 있어 사용할 수 있다.

image.WebcamWidth : webcam의 width.

image.WebcamnHeight : webcam의 height.

image.DetectionWidth : 인식범위의 width.

image.DetectionHeight : 인식범위의 height.

image.Degree : Image의 회전, 스마트폰 카메라의 회전에 대응한다.

image.Data : Pixel 데이터 메모리를 가리키는 포인터.

image.OffsetX, image.OffsetY : 입력 이미지를 crop 하여 일부분만 사용할 경우에 설정한다. 전체 이미지를 사용할 경우에는 해당 값을 설정하지 않는다.

facePtr : FaceData를 context.maxCount개 가지고 있는 array.

facePtr[i].ID : i번째 face의 ID (output).

facePtr[i].Landmark : i번째 face의 2차원 랜드마크(output)

해당 함수는 detect한 얼굴의 개수를 반환한다. (output).

FaceData의 다른 변수들은 3차원 fitting에 관련되어 있어 FaceLib에서는 사용하지 않는다.

Release(ref FaceLib.Context context)

FaceLib에 할당된 리소스를 release합니다.

6.2 Face3DLib

FaceLib이 검출한 2차원 얼굴 및 Landmark를 기반으로 3차원 얼굴과 AR 이모지를 위한 표정을 검출합니다.

Init(ref Face3DLib.Context context3D)

Face3DLib을 초기화합니다.

context3D : 3차원 fitting을 위한 struct space

context3D.levelOf3DProcess : enum 타입으로 되어있으며 3D 계산을 어느정도까지 계산하는지 설정한다.

levelOf3DProcess가 0일 경우, 얼굴의 위치와 방향만 계산(target.Headpose). (안경 등의 단순 3d 스티커 부착 시 이 정보만 있으면 된다).

levelOf3DProcess가 1일 경우 target.Headpose에 더해 표정도 계산(target.GetAnimation). (애니모지 등의 용도로 사용).

levelOf3DProcess가 2일 경우 위의 두 값에 더해 target.Vertices도 계산. (보다 복잡한 효과 얼굴 표면에 입히기 위해 사용).

Set(ref Face3DLib.Context context3D, int width, int height, float fieldOfView)

3차원 fitting에 필요한 데이터를 받아 param에 세팅합니다.

context3D : 3차원 fitting을 위한 struct space

width : 화면의 넓이.

height : 화면의 높이.

FieldOfView : 화면 렌더링을 위해 사용되는 카메라의 field of view.

Process(ref Face3DLib.Context context3D, ref FaceData face)

FaceLib이 검출한 2차원 얼굴 및 Landmark를 입력받아, 3차원 얼굴 위치 및 Blendshape 표정을 출력합니다.

context3D : 3차원 fitting을 위한 struct space

face는 FaceLib이 검출한 2차원 얼굴을 가리키는 포인터.

face.Landmark : face의 2차원 랜드마크.

face.HeadPose : 2차원 랜드마크에 대응하는 3차원 얼굴의 위치 및 rotation (output).

face.GetAnimation : 2차원 랜드마크에 대응하는 3차원 얼굴의 표정 (output).

face.Vertices : 정합된 3차원 얼굴 메쉬의 정점 (output).

6.3 HandLib

입력된 사진을 기반으로 손의 ROI 영역 및 2D Skeleton을 찾습니다.
Detect 함수 수행 시 입력된 이미지에 대하여 전체 영역에서 손 검출이 수행되며, 이후 내부적으로 검출된 손에 tracking이 이루어집니다.
또한 이미지별로 각각 검출된 손의 2D Skeleton 위치를 추정하며, 해당 결과는 handPtr을 통해 반환됩니다.

Init(HandLib.Context context)

FaceLib을 초기화합니다.

context.maxCount : detect/track 하는 최대 손의 개수를 정의. 이미지에 존재하는 손 개수가 maxCount보다 클 경우 크기가 큰 손부터 찾는다.

context.minHandSize : 손을 인식하기 위한 최소 크기. 60이 기본

context.InitPath : 얼굴인식을 위한 deep learning network의 위치를 지정한다.

Detect(HandLib.Context context, ImageData image, HandData* handPtr)

2차원 이미지(ImageData image)를 입력으로 받아 해당 이미지에서 손을 인식하여 인식한 손의 HandData 값을 저장합니다.

Unity의 Texture를 ImageData로 전환하는 함수가 예제의 TextureToImageData.cs에 정의되어 있어 사용할 수 있다.

image.WebcamWidth : webcam의 width.

image.WebcamnHeight : webcam의 height.

image.DetectionWidth : 인식범위의 width.

image.DetectionHeight : 인식범위의 height.

image.Degree : Image의 회전, 스마트폰 카메라의 회전에 대응한다.

image.Data : Pixel 데이터 메모리를 가리키는 포인터.

image.OffsetX, image.OffsetY : 입력 이미지를 crop 하여 일부분만 사용할 경우에 설정한다. 전체 이미지를 사용할 경우에는 해당 값을 설정하지 않는다.

handPtr : handData를 context.maxCount개 만큼 가지고 있는 array.

handPtr[i].NumPoints : hand의 Skeleton Points 개수 (output).

handPtr[i].ID : i번째 hand의 ID (output).

handPtr[i].Posture : i번째 hand의 제스쳐 타입 (output).

- v, gun, rock, paper, mini_heart, one, thumbs_up,okay, unknown 지원합니다.

handPtr[i].LeftOrRight : i번째 hand의 왼손/오른손 구분 (output).

handPtr[i].Box : i번째 hand의 BoxRigion (output).

handPtr[i].Center : 손의 가운데 위치의 스크린 좌표

handPtr[i].Points : i번째 hand의 Skeleton Points (output) .

해당 함수는 detect한 손의 개수를 반환한다(output).

HandLib다른 변수들은 3차원 fitting에 관련되어 있어 HandLib에서는 사용하지 않는다.

Release(HandLib.Context context)

HandLib에 할당된 리소스를 release 합니다.

Still have Questions?

이 문서를 읽고 문의사항이 있다면 sg.ju@alcherainc.com 으로 메일 보내주시면 감사하겠습니다.

Alchera

Make your AI dreams a reality

www.alcherainc.com