ArcSoft Face Recognition

开发指导文档

ARCSOFT FACE RECOGNITION **1**

1. 概述 **3**

1.1. 运行环境 3
1.2. 系统要求 3
1.3. 依赖库 3

2. 结构与变量 **4**

2.1. 基本类型 4
2.2. 数据结构与枚举 4
2.2.1. AFR_FSDK _FACEINPUT *4*
2.2.2. AFR_FSDK_FACEMODEL *4*
2.2.3. AFR_FSDK_Version *5*
2.2.4. AFR_FSDK_OrientCode *5*
2.2.5. 支持的颜色格式 *6*
2.2.6. 错误码 *6*

3. API 说明 **7**

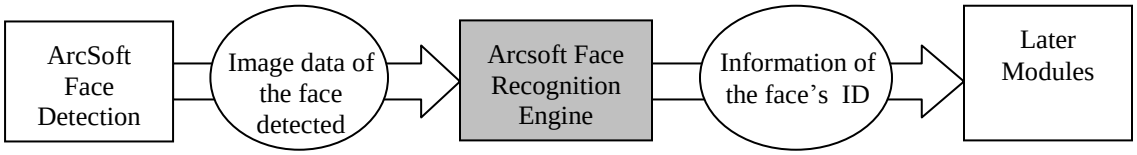
3.1. AFR_FSDK_INITIALENGINE 7
3.2. AFR_FSDK_EXTRACTFRFEATURE 7
3.3. AFR_FSDK_FACEPAIRMATCHING 8
3.4. AFR_FSDK_UNINITIALENGINE 9
3.5. AFR_FSDK_GETVERSION 9

4. 其他说明 **10**

5. 示例代码 **11**

1. 概述

虹软人脸识别引擎工作流程图：



1.1. 运行环境

- iOS Arm v4, Arm64

1.2. 系统要求

- 支持 iOS 8.x 及以上

1.3. 依赖库

- 虹软平台库

注: 请把虹软平台库的头文件以 (在 SDK 包的“platform”目录下)放入您的开发工程里面。

2. 结构与变量

2.1. 基本类型

```
typedef MInt32   AFR_FSDK_OrientCode;
```

所有基本类型在平台库中有定义。 定义规则是在 ANSIC 中的基本类型前加上字母“M”同时将类型的第一个字母改成大写。例如“long” 被定义成“MLong”。

2.2. 数据结构与枚举

2.2.1. AFR_FSDK _FACEINPUT

描述

通过人脸检测引擎获得的脸部信息

定义

```
typedef struct{
       MRECT                    rcFace;
       AFR_FSDK_OrientCode      lOrient;
} AFR_FSDK_FACEINPUT, *LPAFR_FSDK_FACEINPUT;
```

成员描述

rcFace	脸部矩形框信息
lOrient	脸部旋转角度

2.2.2. AFR_FSDK_FACEMODEL

描述

脸部特征信息

定义

```
typedef struct{
       MByte        *pbFeature;
       MInt32       lFeatureSize;
} AFR_FSDK_FACEMODEL, *LPAFR_FSDK_FACEMODEL;
```

成员描述

pbFeature	人脸特征信息

lFeatureSize	人脸特征信息长度

2.2.3. AFR_FSDK_Version

描述

SDK 版本信息

定义

```
typedef struct{
       MInt32 lCodebase;
       MInt32 lMajor;
       MInt32 lMinor;
       MInt32 lBuild;
       MInt32 lFeatureLevel;
       MPChar Version;
       MPChar BuildDate;
       MPChar CopyRight;
} AFR_FSDK_Version;
```

成员描述

lCodebase	代码库版本号
lMajor	主版本号
lMinor	次版本号
lBuild	编译版本号，递增
lFeatureLevel	人脸特征库版本号
Version	字符串形式的版本号
BuildDate	编译时间
CopyRight	版权

2.2.4. AFR_FSDK_OrientCode

描述

基于逆时针的脸部方向枚举值。

定义

```
enum _AFR_FSDK_OrientCode{
       AFR_FSDK_FOC_0            = 0x1,
       AFR_FSDK_FOC_90           = 0x2,
       AFR_FSDK_FOC_270          = 0x3,
```

```
        AFR_FSDK_FOC_180            = 0x4,
        AFR_FSDK_FOC_30             = 0x5,
        AFR_FSDK_FOC_60             = 0x6,
        AFR_FSDK_FOC_120            = 0x7,
        AFR_FSDK_FOC_150            = 0x8,
        AFR_FSDK_FOC_210            = 0x9,
        AFR_FSDK_FOC_240            = 0xa,
        AFR_FSDK_FOC_300            = 0xb,
        AFR_FSDK_FOC_330            = 0xc
};
```

成员描述

AFR_FSDK_FOC_0	0 度
AFR_FSDK_FOC_90	90 度
AFR_FSDK_FOC_270	270 度
AFR_FSDK_FOC_180	180 度
AFR_FSDK_FOC_30	30 度
AFR_FSDK_FOC_60	60 度
AFR_FSDK_FOC_120	120 度
AFR_FSDK_FOC_150	150 度
AFR_FSDK_FOC_210	210 度
AFR_FSDK_FOC_240	240 度
AFR_FSDK_FOC_300	300 度
AFR_FSDK_FOC_330	330 度

2.2.5. 支持的颜色格式

描述

颜色格式及其对齐规则

定义

ASVL_PAF_NV12	8-bit Y 层，之后是 8-bit 的 2x2 采样的 U 层和 V 层
ASVL_PAF_RGB24_B8G8R8	每个像素 8-bit B，8-bit R，8-bit R

2.2.6. 错误码

具体的错误码定义可以参考平台库中 merror.h 文件

3. API 说明

3.1. AFR_FSDK_InitialEngine

原型

```
MRESULT AFR_FSDK_InitialEngine(
       MPChar        AppId,
       MPChar        SDKKey,
       MByte*        pMem,
       MLong         lMemSize,
       MHandle      *phEngine
);
```

描述

初始化人脸识别引擎。

参数

AppId	[in]	用户申请 SDK 时获取的 App Id
SDKKey	[in]	用户申请 SDK 时获取的 SDK Key
pMem	[in]	分配给引擎使用的内存地址
lMemSize	[in]	分配给引擎使用的内存大小
phEngine	[out]	引擎句柄

返回值

成功返回 MOK，否则返回失败 code。失败 codes 如下所列:

MERR_INVALID_PARAM	参数输入非法
MERR_NO_MEMORY	内存不足

3.2. AFR_FSDK_ExtractFRFeature

原型

```
MRESULT AFR_FSDK_ExtractFRFeature (
       MHandle             hEngine,
       LPASVLOFFSCREEN     pInputImage,
       LPAFR_FACEINPUT     pFaceRes,
       LPAFR_FACEMODEL     pFaceModels
);
```

描述

提取脸部特征。

参数

hEngine	[in]	引擎句柄
pInputImage	[in]	输入的图像数据
pFaceRes	[in]	已检测到的脸部信息
pFaceModels	[out]	提取到的脸部特征信息

返回值

成功返回 MOK，否则返回失败 code。失败 codes 如下所列:

MERR_INVALID_PARAM	参数输入非法
MERR_NO_MEMORY	内存不足

3.3. AFR_FSDK_FacePairMatching

原型

```
MRESULT AFR_FSDK_FacePairMatching(
       MHandle              hEngine,
       AFR_FACEMODEL        *reffeature,
       AFR_FACEMODEL        *probefeature,
       MFloat               *pfSimilScore
);
```

描述

脸部特征比较。

参数

hEngine	[in]	引擎句柄
reffeature	[in]	已有脸部特征信息
probefeature	[in]	被比较的脸部特征信息
pfSimilScore	[out]	脸部特征相似程度数值

返回值

成功返回 MOK，否则返回失败 code。失败 codes 如下所列:

MERR_INVALID_PARAM	参数输入非法
MERR_NO_MEMORY	内存不足

3.4. AFR_FSDK_UninitialEngine

原型

```
MRESULT AFR_FSDK_UninitialEngine(
      MHandle          hEngine
);
```

描述

销毁引擎，释放相应资源。

参数

hEngine	[in]	引擎句柄

返回值

成功返回 MOK，否则返回失败 code。失败 codes 如下所列:

MERR_INVALID_PARAM	参数输入非法

3.5. AFR_FSDK_GetVersion

原型

```
const AFR_FSDK_Version *AFR_FSDK_GetVersion(
      MHandle          hEngine
);
```

描述

获取 SDK 版本信息

参数

hEngine	[in]	引擎句柄

4. 其他说明

此版本为免费开放的标准版本(为保证最优体验，建议注册人脸数小于 1000)，若有定制升级需求，请联系我们。

5. 示例代码

对于人脸识别, 分成两个步骤，首先需要注册一个人脸，然后再使用另外人脸做识别操作。下面的示例代码就是按照这两个步骤来的：

```
#include "ammem.h"
#include "merror.h"
#import <arcsoft_fsdk_face_recognition/arcsoft_fsdk_face_recognition.h>

#include <stdlib.h>
#include <string.h>


#define ARC_APP_ID                      "YOUR OWN APP ID"
#define ARC_FR_SDK_KEY                  "YOUR OWN SDK KEY"
#define ARC_FR_MEM_SIZE                 1024*1024*40

#define ARC_FR_MAX_FEATURE_COUNT_PER_PERSON     5
#define ARC_FR_SCORE_THRESHOLD                  (0.56f)

typedef struct
{
    MInt32 lFeatureSize;
    MByte *pbFeature;
}AFR_FSDK_FACEDATA, *LPAFR_FSDK_FACEDATA;

typedef struct
{
    MLong lPersonID;
    MChar szPersonName[128];
    AFR_FSDK_FACEDATA* pFaceFeatureArray;
    MInt32             nFeatureCount;
}AFR_FSDK_PERSON, *LPAFR_FSDK_PERSON;

//人脸检测实现
MRESULT detectFace(LPASVLOFFSCREEN pOff, AFR_FSDK_FACEINPUT*
pFaceInput)
{
    //使用人脸检测引擎检测人脸信息
    return MOK;
}

MRESULT addPersonToDB(LPAFR_FSDK_PERSON person)
{
    return  MOK;
}

MRESULT loadPersonsFromDB(LPAFR_FSDK_PERSON* persons, MLong*
```

```
pPersonCount)
{
    return MOK;
}

//人脸注册实现
MRESULT doRegister()
{

    MVoid* pMemBuffer = MMemAlloc(MNull, ARC_FR_MEM_SIZE);
    MHandle hEngine = MNull;

    //初始化人脸识别引擎
    MRESULT mr = AFR_FSDK_InitialEngine((MPChar)ARC_APP_ID,
(MPChar)ARC_FR_SDK_KEY, (MByte*)pMemBuffer, ARC_FR_MEM_SIZE, &hEngine);
    if (MOK != mr) {
        //错误码检查
    }


    do {
        ASVLOFFSCREEN offScreenIn = {0};
        offScreenIn.u32PixelArrayFormat = ASVL_PAF_NV12;
        offScreenIn.i32Width = 1280;
        offScreenIn.i32Height = 720;
        offScreenIn.pi32Pitch[0] = offScreenIn.i32Width;
        offScreenIn.pi32Pitch[1] = offScreenIn.i32Width;
        offScreenIn.ppu8Plane[0] = MNull;
        offScreenIn.ppu8Plane[1] = MNull;

        AFR_FSDK_FACEINPUT faceInput = {0};
        mr = detectFace(&offScreenIn, &faceInput);
        if (MOK != mr)
        {
            continue;
        }

        AFR_FSDK_FACEMODEL faceModel = {0};

        //提取人脸特性信息
        mr = AFR_FSDK_ExtractFRFeature(hEngine, &offScreenIn, &faceInput,
&faceModel);
        if (MOK != mr)
        {
            continue;
        }

        AFR_FSDK_FACEDATA featureData = {0};
        featureData.lFeatureSize = faceModel.lFeatureSize;
```

```
        featureData.pbFeature = (MByte*)MMemAlloc(MNull, featureData.lFeatureSize);
        MMemCpy(featureData.pbFeature, faceModel.pbFeature,
featureData.lFeatureSize);

        AFR_FSDK_PERSON person = {0};
        person.lPersonID = 330;
        strncpy(person.szPersonName , "Jack", strlen("Jack")); // input name
        person.nFeatureCount = 1;
        person.pFaceFeatureArray = (AFR_FSDK_FACEDATA*)MMemAlloc(MNull,
sizeof(AFR_FSDK_FACEDATA) * person.nFeatureCount);
        person.pFaceFeatureArray[0] = featureData;

        addPersonToDB(&person);

      } while (MFalse);

    //对人脸识别引擎做销毁
    mr = AFR_FSDK_UninitialEngine(hEngine);

    if(pMemBuffer != MNull)
    {
        MMemFree(MNull,pMemBuffer);
        pMemBuffer = MNull;
    }

    return mr;
}

//人脸识别实现
MRESULT doRecognition()
{

    MVoid* pMemBuffer = MMemAlloc(MNull, ARC_FR_MEM_SIZE);
    MHandle hEngine = MNull;

    //初始化人脸识别引擎
    MRESULT mr = AFR_FSDK_InitialEngine((MPChar)ARC_APP_ID,
(MPChar)ARC_FR_SDK_KEY, (MByte*)pMemBuffer, ARC_FR_MEM_SIZE, &hEngine);
    if (MOK != mr) {

    }


    do {
        ASVLOFFSCREEN offScreenIn = {0};
        offScreenIn.u32PixelArrayFormat = ASVL_PAF_NV12;
        offScreenIn.i32Width = 1280;
        offScreenIn.i32Height = 720;
        offScreenIn.pi32Pitch[0] = offScreenIn.i32Width;
```

```
        offScreenIn.pi32Pitch[1] = offScreenIn.i32Width;
        offScreenIn.ppu8Plane[0] = MNull;
        offScreenIn.ppu8Plane[1] = MNull;

        AFR_FSDK_FACEINPUT faceInput = {0};
        mr = detectFace(&offScreenIn, &faceInput);
        if (MOK != mr)
        {
            continue;
        }

        AFR_FSDK_FACEMODEL faceModel = {0};

        //提取人脸特性信息
        mr = AFR_FSDK_ExtractFRFeature(hEngine, &offScreenIn, &faceInput,
&faceModel);
        if (MOK != mr)
        {
            continue;
        }

        AFR_FSDK_FACEMODEL probeFaceModel = faceModel;

        AFR_FSDK_PERSON personRecognized = {0};
        MFloat fMaxScore = 0;

        LPAFR_FSDK_PERSON persons = MNull;
        MLong lPersonCount = 0;
        loadPersonsFromDB(&persons, &lPersonCount);
        for (int i = 0; i < lPersonCount; ++i)
        {
            AFR_FSDK_PERSON person = persons[i];
            for (int j = 0; j < person.nFeatureCount; ++j)
            {
                AFR_FSDK_FACEMODEL refFaceModel = {0};
                refFaceModel.lFeatureSize = person.pFaceFeatureArray[j].lFeatureSize;
                refFaceModel.pbFeature = person.pFaceFeatureArray[j].pbFeature;

                MFloat fScore = 0.0;

                //人脸特性信息比对，返回两个人脸特性的相似度
                MRESULT mr = AFR_FSDK_FacePairMatching(hEngine, &refFaceModel,
&probeFaceModel, &fScore);
                if (mr == MOK && fScore > fMaxScore) {
                    fMaxScore = fScore;

                    personRecognized = person;
                }
            }
        }
```

```
        if (fMaxScore > ARC_FR_SCORE_THRESHOLD) {

        }

    } while (MFalse);

    //对人脸识别引擎做销毁
    mr = AFR_FSDK_UninitialEngine(hEngine);
    if(pMemBuffer != MNull)
    {
        MMemFree(MNull,pMemBuffer);
        pMemBuffer = MNull;
    }

    return mr;
}
```