Arcsoft Photo Styling SDK 开发文档

目录

1. SDK 概述

Arcsoft Photo Styling SDK 可以将输入图像转化为具有指定风格的图像。图像风格主要是指色调、笔触、线条等图像的特异性表现形式。存储某一艺术图像风格模板的文件将按需要发布给 SDK 使用者。借助 SDK 和风格模板文件，用户可以开发图像风格化等图像美化类应用。SDK 工作示意图如图 1 所示。一种艺术风格对应于一个风格模板文件，当开发者用户需要新的风格模板时，由 SDK 发行者提供新的风格模板文件供开发者用户使用。

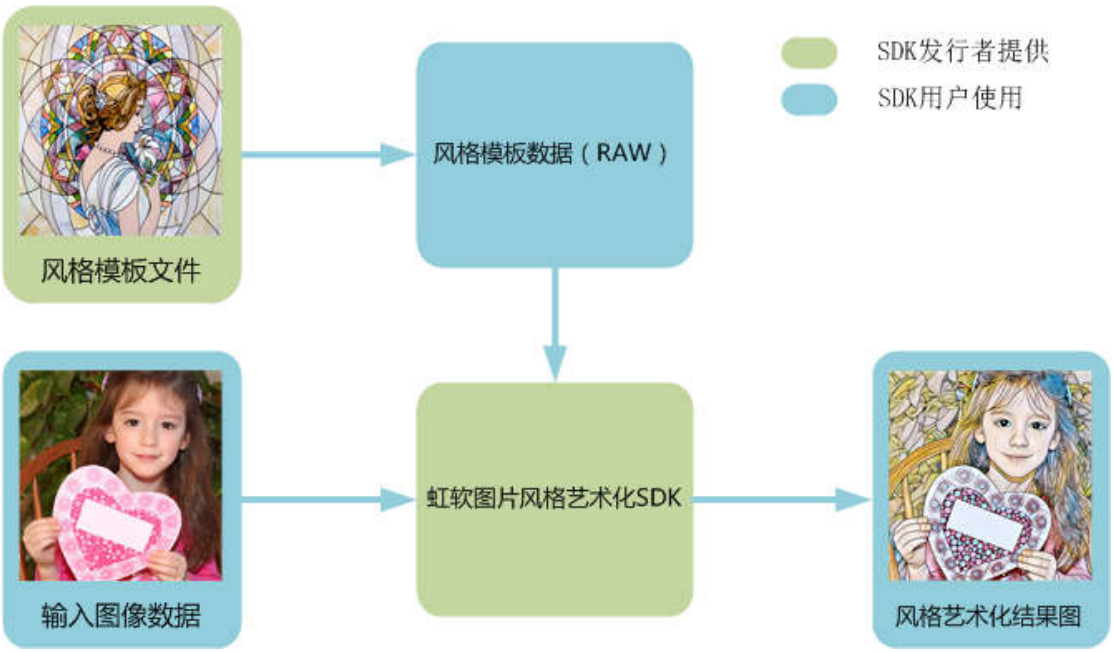

图 1

2. SDK 运行环境及相关指标

2.1 运行环境

操作系统：	Linux-x64
CPU：	Intel-Haswell 以上架构（要求支持 AVX2 指令集）
RAM：	1GB

2.2 运行指标

项目	指标
输入图像限制	支持最大像素数 2073600 支持最高 1440x1440(1:1)，1600x1200(4:3)，1920x1080(16:9) 宽高必须为 4 的整数倍
输入图像格式	BGR24 RGB24
平均耗时	2s/百万像素　(随实际机器性能可能改变)
单个风格模型文件 ROM 占用	2.1MB　　(随模型文件更新可能改变)

3. API 说明

3.1 函数

APS_FSDK_Get_Version

函数声明

```
const APS_FSDK_Version* APS_FSDK_Get_Version(APS_FSDK_ENGINE_HANDLE hAPSEngine)
```

功能描述

获取 SDK 版本信息

参数

hAPSEngine　　[输入]　引擎句柄

APS_FSDK_InitEngine

函数声明

```
MRESULT APS_FSDK_InitEngine (
    const char *        AppId,
    const char *        SDKKey,
    MHandle hMemMgr,
    APS_FSDK_DEVICE_ID APSDeviceID,
    APS_FSDK_ENGINE_HANDLE *phAPSEngine
);
```

功能描述

初始化引擎

参数

AppId　　[输入]　用户申请 SDK 时获取的 app id

SDKKey	[输入]	用户申请 SDK 时获取的 key
hMemMgr	[输入]	内存管理器句柄
APSDeviceID	[输入]	程序运行设备，-1 代表 CPU，0,1,2,…代表指定序号的 GPU
phAPSEngine	[输出]	指向引擎句柄的指针

APS_FSDK_StyleTransfer

函数声明

```
MRESULT APS_FSDK_StyleTransfer (
    APS_FSDK_ENGINE_HANDLE      hAPSEngine,
    MByte*                      pBModel,
    MLong                       lModelSize,
    LPASVLOFFSCREEN             lpImgIn,
    LPASVLOFFSCREEN             lpImgOut
);
```

功能描述

根据指定风格模型进行图片风格艺术化，保存转换输入图像风格后的输出图像数据。

参数

hAPSEngine	[输入]	引擎句柄
pBModel	[输入]	风格模型数据指针
lModelSize	[输入]	风格模型数据大小
lpImgIn	[输入]	待进行风格学习图像
lpImgOut	[输出]	风格学习结果图像

APS_FSDK_UninitEngine

函数声明

```
MVoid APS_FSDK_UninitEngine (
    APS_FSDK_ENGINE_HANDLE   hAPSEngine
);
```

功能描述

销毁引擎

参数

hAPSEngine	[输入]	引擎句柄

3.2 类型定义

APS_FSDK_DEVICE_ID

描述

程序运行设备，-1 代表 CPU，0,1,2,等代表指定序号的 GPU。当前仅支持 -1 (CPU 设备)

定义

typedef MInt32 APS_FSDK_DEVICE_ID;

APS_FSDK_ENGINE_HANDLE

描述

风格学习引擎句柄

定义

typedef MHandle APS_FSDK_ENGINE_HANDLE;

APS_FSDK_Version

描述

SDK 版本信息

定义

```
typedef struct
{
    MLong lCodebase;
    MLong lMajor;
    MLong lMinor
    MLong lBuild;
    MTChar* Version;
    MTChar* BuildDate;
    MTChar* CopyRight;
} APS_FSDK_Version;
```

成员变量

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
Version	字符串形式的版本号
BuildDate	编译日期
CopyRight	Copyright 信息

3.3 返回码定义

定义	值	说明
MOK	0x00000	成功
MERR_UNKNOWN	0x00001	其他内部错误
MERR_NO_MEMORY	0x00004	内存不足
MERR_FSDK_INVALID_APP_ID	0x07001	非法的 APPID

MERR_FSDK_INVALID_SDK_ID	0x07002	非法的 SDKKEY
MERR_FSDK_INVALID_ID_PAIR	0x07003	SDKKEY 不是于当前 APPID 名下的
MERR_FSDK_MISMATCH_ID_AND_SDK	0x07004	SDKKEY 不是当前 SDK 所支持的
MERR_FSDK_LICENCE_EXPIRED	0x07006	SDK 过期
MERR_FSDK_APS_ENGINE_HANDLE	0x11001	引擎句柄非法
MERR_FSDK_APS_MEMMGR_HANDLE	0x11002	内存管理器句柄非法
MERR_FSDK_APS_DEVICEID_INVALID	0x11003	DEVICEID 非法
MERR_FSDK_APS_DEVICEID_UNSUPPORTED	0x11004	DEVICEID 不支持
MERR_FSDK_APS_MODEL_HANDLE	0x11005	模板数据指针非法
MERR_FSDK_APS_MODEL_SIZE	0x11006	模板长度非法
MERR_FSDK_APS_IMAGE_HANDLE	0x11007	图像结构体指针非法
MERR_FSDK_APS_IMAGE_FORMAT_UNSUPPORTED	0x11008	图像格式不支持
MERR_FSDK_APS_IMAGE_PARAM	0x11009	图像 Width,Height,Pitch 错误
MERR_FSDK_APS_IMAGE_SIZE	0x1100A	图像尺寸大小超出支持范围

4. 示例代码

!!!! 注意,使用时请替换申请的 APPID SDKKEY，并设置好文件路径!!!!

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <assert.h>

#include <chrono>

#include "arcsoft_fsdk_photo_styling.h"

//#define APPID         "your appid"
//#define SDKKEY        "your sdkkey"

#define STYLE_MODEL_PATH "CHINESEART-001.tmpl"
#define INPUT_IMAGE_PATH "your_input_image.bmp"
#define OUTPUT_IMAGE_PATH "your_output_image.bmp"
#define IMAGE_WIDTH    (1920)
#define IMAGE_HEIGHT    (1080)

#define WORKBUF_SIZE (256 * 1024 * 1024)

#pragma pack(push,1)
typedef     struct    tag_BMP46_Header {
```

```
    uint16_t bfType;
    int32_t bfSize;
    uint16_t bfReserved1;
    uint16_t bfReserved2;
    int32_t bfOffBits;
    int32_t biSize;
    int32_t biWidth;
    int32_t biHeight;
    int16_t biPlanes;
    int16_t biBitCount;
    int32_t biCompression;
    int32_t biSizeImage;
    int32_t biXPelsPerMeter;
    int32_t biYPelsPerMeter;
    int32_t biClrUsed;
    int32_t biClrImportant;
} BMP46_Header;
#pragma pack(pop)

int fu_ReadBMPFile(const char* path, uint8_t **raw_data, size_t* pSize) {
    assert(sizeof(BMP46_Header) == 54);

    int res = 0;
    FILE *fp = 0;
    uint8_t *data_file = 0;
    size_t size = 0;

    fp = fopen(path, "rb");
    if (fp == nullptr) {
        res = -1;
        goto exit;
    }
    fseek(fp, 0, SEEK_END);
    size = ftell(fp);
    fseek(fp, sizeof(BMP46_Header), SEEK_SET);

    data_file = (uint8_t *)malloc(size - sizeof(BMP46_Header));
    if (data_file == nullptr) {
        res = -2;
        goto exit;
    }
    if ((size - sizeof(BMP46_Header)) !=
        fread(data_file, sizeof(uint8_t), size - sizeof(BMP46_Header), fp)) {
        res = -3;
```

```
            goto exit;
        }

        *raw_data = data_file;
        data_file = nullptr;
exit:
        if (fp != nullptr) {
            fclose(fp);
        }
        if (data_file != nullptr) {
            free(data_file);
        }
        if (nullptr != pSize) {
            *pSize = size - sizeof(BMP46_Header);
        }
        return res;
}


int fu_SaveBMPFile(const char *path, uint8_t *raw_data, int width, int height) {
        FILE* fp = fopen(path, "wb+");
        if (fp != nullptr) {
            BMP46_Header header = { 0 };
            header.bfType = 'MB';
            header.bfSize = width*height * 3 + sizeof(BMP46_Header);
            header.bfOffBits = sizeof(BMP46_Header);
            header.biSize = 40;
            header.biWidth = width;
            header.biHeight = height;
            header.biPlanes = 1;
            header.biBitCount = 24;
            header.biSizeImage = width*height * 3;
            header.biXPelsPerMeter = 0xEC4;
            header.biYPelsPerMeter = 0xEC4;

            fwrite(&header, sizeof(BMP46_Header), 1, fp);
            fwrite(raw_data, width*height * 3, 1, fp);
            fclose(fp);
            return 0;
        }
        return -1;
}
```

```
int fu_ReadFile(const char* path, uint8_t **raw_data, size_t* pSize) {
	int res = 0;
	FILE *fp = 0;
	uint8_t *data_file = 0;
	size_t size = 0;

	fp = fopen(path, "rb");
	if (fp == nullptr) {
		res = -1;
		goto exit;
	}

	fseek(fp, 0, SEEK_END);
	size = ftell(fp);
	fseek(fp, 0, SEEK_SET);

	data_file = (uint8_t *)malloc(sizeof(uint8_t)* size);
	if (data_file == nullptr) {
		res = -2;
		goto exit;
	}

	if (size != fread(data_file, sizeof(uint8_t), size, fp)) {
		res = -3;
		goto exit;
	}

	*raw_data = data_file;
	data_file = nullptr;
exit:
	if (fp != nullptr) {
		fclose(fp);
	}

	if (data_file != nullptr) {
		free(data_file);
	}

	if (nullptr != pSize) {
		*pSize = size;
	}

	return res;
}
```

```
int main(int argc, char* argv[]) {

    void *pWorkMem = malloc(WORKBUF_SIZE);
    void *hMgr = MMemMgrCreate(pWorkMem, WORKBUF_SIZE);

    uint8_t *pRawModel = nullptr;
    size_t modelength;
    fu_ReadFile(STYLE_MODEL_PATH, (uint8_t**)&pRawModel, &modelength);
    if (!pRawModel) {
        fprintf(stderr, "fail to fu_ReadFile(%s): %s\r\n",
                STYLE_MODEL_PATH, strerror(errno));
        free(pWorkMem);
        exit(0);
    }

    auto ts_start = std::chrono::high_resolution_clock::now();
    void *hEngine = nullptr;
    int32_t device = -1;
    int ret = APS_FSDK_InitEngine(APPID, SDKKEY, hMgr, device, &hEngine);
    if (ret != 0) {
        fprintf(stderr, "fail to call FreeSDK_APS_InitEngine(): 0x%x\r\n", ret);
        free(pRawModel);
        free(pWorkMem);
        exit(0);
    }

    const APS_FSDK_Version*pVersionInfo = APS_FSDK_Get_Version(hEngine);
    printf("%ld %ld %ld %ld\r\n", pVersionInfo->lCodebase,
        pVersionInfo->lMajor, pVersionInfo->lMinor, pVersionInfo->lBuild);
    printf("%s\r\n", pVersionInfo->Version);
    printf("%s\r\n", pVersionInfo->BuildDate);
    printf("%s\r\n", pVersionInfo->CopyRight);

    ASVLOFFSCREEN inputImg = { 0 };
    inputImg.u32PixelArrayFormat = ASVL_PAF_RGB24_B8G8R8;
    inputImg.i32Width = IMAGE_WIDTH;
    inputImg.i32Height = IMAGE_HEIGHT;
    inputImg.pi32Pitch[0] = inputImg.i32Width * 3;
    inputImg.ppu8Plane[0] = nullptr;
    fu_ReadBMPFile(INPUT_IMAGE_PATH, (uint8_t**)&inputImg.ppu8Plane[0], nullptr);
    if (!inputImg.ppu8Plane[0]) {
        fprintf(stderr, "fail to fu_ReadFile(%s): %s\r\n", INPUT_IMAGE_PATH, strerror(errno));
```

```
        APS_FSDK_UninitEngine(hEngine);
        free(pRawModel);
        free(pWorkMem);
        exit(0);
    }

    ASVLOFFSCREEN outputImg = { 0 };
    outputImg.u32PixelArrayFormat = ASVL_PAF_RGB24_B8G8R8;
    outputImg.i32Width = IMAGE_WIDTH;
    outputImg.i32Height = IMAGE_HEIGHT;
    outputImg.pi32Pitch[0] = outputImg.i32Width * 3;
    outputImg.ppu8Plane[0] = (uint8_t*)malloc(outputImg.pi32Pitch[0] * outputImg.i32Height);

    ret = APS_FSDK_StyleTransfer(hEngine, pRawModel, modelength, &inputImg, &outputImg);
    if (ret != 0) {
        fprintf(stderr, "fail to call FreeSDK_APS_StyleTransfer(): 0x%x\r\n", ret);
        APS_FSDK_UninitEngine(hEngine);
        free(inputImg.ppu8Plane[0]);
        free(outputImg.ppu8Plane[0]);
        free(pRawModel);
        free(pWorkMem);
        exit(0);
    }

    APS_FSDK_UninitEngine(hEngine);
    auto ts_end = std::chrono::high_resolution_clock::now();
    printf("cost %ld ms\r\n",
        std::chrono::duration_cast<std::chrono::milliseconds>(ts_end - ts_start).count());
    fu_SaveBMPFile(OUTPUT_IMAGE_PATH, (uint8_t*)outputImg.ppu8Plane[0],
                    outputImg.i32Width, outputImg.i32Height);

    free(inputImg.ppu8Plane[0]);
    free(outputImg.ppu8Plane[0]);
    free(pRawModel);
    free(pWorkMem);
    return 0;
}
```