

# SenseME Effects iOS集成文档

## 目录

### 1.集成准备

- 1.1 导入库文件
- 1.2 添加链接库
- 1.3 关闭Bitcode
- 1.4 导入头文件

### 2.SDK授权

- 2.1 License授权
- 2.2 验证激活码

### 3.SDK各接口的使用

- 3.1 SDK句柄的初始化
- 3.2 纹理的获取
- 3.3 纹理预处理
- 3.4 帧处理流程
- 3.5 SDK句柄的释放
- 3.6 模式选择

### 4.客户自定义

- 4.1 人脸检测
- 4.2 美颜
- 4.3 贴纸
- 4.4 美妆

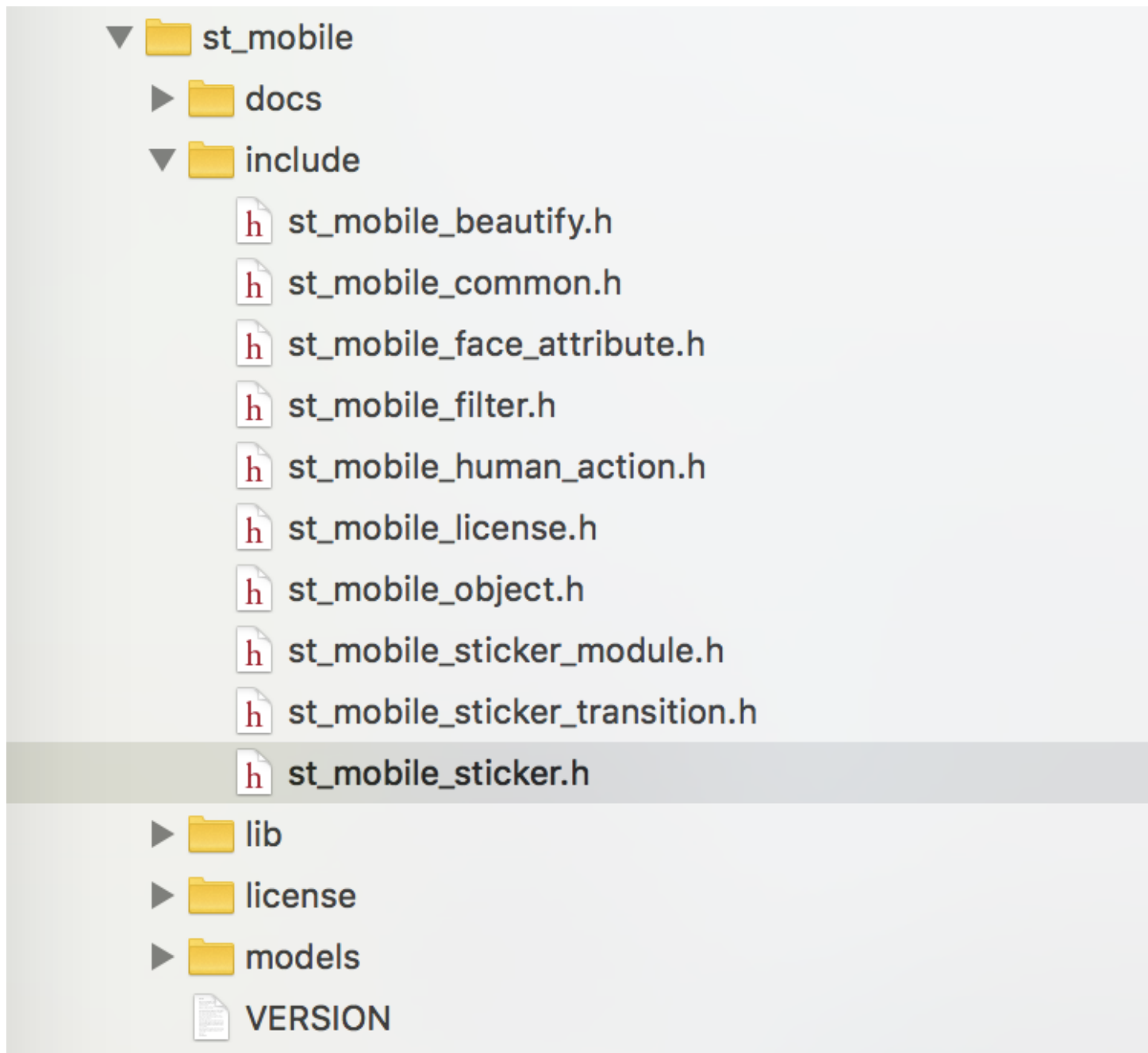
### 5.集成注意事项

### 6.检测渲染优化

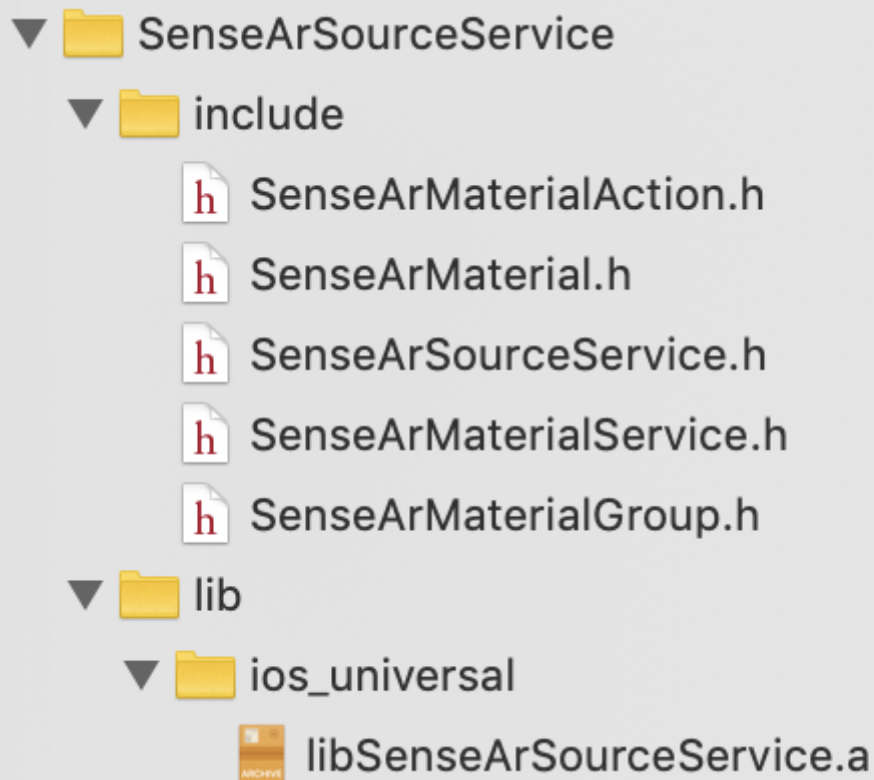
# 1 集成准备

## 1.1 导入库文件

导入 SenseME Effects iOS SDK 头文件、静态库文件(.a)、License 文件(.lic)、模型文件(.model) (如下图所示)



由于新版本Effects增加了在线鉴权以及在线素材拉取功能，因此需要导入相关静态库以及头文件，如下图所示



## 1.2 添加链接库

SenseME Effects依赖c++, 从TARGETS -> Build Settings -> Linking -> Other LinkerFlags添加-lc++

## 1.3 关闭Bitcode

SenseMe Effects不支持Bitcode, 从TARGETS -> Build Settings -> Build Options -> Enable Bitcode 设置为 NO

## 1.4 导入头文件

按需要导入所需头文件

```
#import "st_mobile_sticker.h"
#import "st_mobile_beautify.h"
#import "st_mobile_license.h"
#import "st_mobile_face_attribute.h"
#import "st_mobile_filter.h"
#import "st_mobile_object.h"
#import "st_mobile_animal.h"
#import "st_mobile_avatar.h"
#import "st_mobile_makeup.h"

#import "SenseArSourceService.h"
```

## 2 SDK授权

### 2.1 License授权

- 1.SDK需要检查算法库的使用权限，只有通过了鉴权，SDK的功能才能够正常使用。
- 2.鉴权方式分为在线鉴权以及离线鉴权，在线鉴权需要获取托管在服务器的license数据，离线鉴权使用本地license检测即可。

### 2.2 离线鉴权

- (1)读取本地license文件内容
- (2)获取本地保存的激活码
- (3)如果没有则生成一个激活码
- (4)如果有,则直接调用checkActiveCode\*检查激活码
- (5)如果检查失败,则重新生成一个activeCode
- (6)如果生成失败,则返回失败,成功则保存新的activeCode,并返回成功

```

//读取SenseME.lic文件内容
NSString *strLicensePath = [[NSBundle mainBundle] pathForResource:@"SENSEME" ofType:@"lic"];
NSData *dataLicense = [NSData dataWithContentsOfFile:strLicensePath];
NSString *strKeySHA1 = @"SENSEME";
NSString *strKeyActiveCode = @"ACTIVE_CODE";
NSUserDefaults *userDefaults = [NSUserDefaults standardUserDefaults];
NSString *strStoredSHA1 = [userDefaults objectForKey:strKeySHA1];
NSString *strLicenseSHA1 = [self getSHA1StringWithData:dataLicense];
... ..

//检查当前的激活码是否可用(这里提供两种方法)
//use file
iRet = st_mobile_check_activecode( strLicensePath.UTF8String, (const char *)[activeCodeData bytes]);
//use buffer
NSData *licenseData = [NSData dataWithContentsOfFile:strLicensePath];
iRet = st_mobile_check_activecode_from_buffer(
    [licenseData bytes],
    (int)[licenseData length],
    [activeCodeData bytes]
);

//如果检查失败, 重新生成一个, 并更新本地激活码, 同理我们提供了两种方法
// use file
iRet = st_mobile_generate_activecode(
    strLicensePath.UTF8String,
    active_code,
    &active_code_len
);
// use buffer
NSData *licenseData = [NSData dataWithContentsOfFile:strLicensePath];
iRet = st_mobile_generate_activecode_from_buffer(
    [licenseData bytes],
    (int)[licenseData length],
    active_code,
    &active_code_len
);
//更新本地已有active Code
NSData *activeCodeData = [NSData dataWithBytes:active_code length:active_code_len];
[userDefaults setObject:activeCodeData forKey:strKeyActiveCode];
[userDefaults setObject:strLicenseSHA1 forKey:strKeySHA1];
[userDefaults synchronize];

```

## 3 SDK各接口的使用

### 3.1 SDK句柄的初始化

#### 1.HumanAction的初始化

```

//HumanAction句柄初始化
//获取模型路径
NSString *strModelPath = [[NSBundle mainBundle] pathForResource:@"M_SenseME_Action" ofType:@"model"];
//创建humanAction句柄
//说明：该接口提供两种创建人体行为的句柄方式，检测视频时设置为ST_MOBILE_HUMAN_ACTION_DEFAULT_CONFIG_CREATE,检测图片
iRet = st_mobile_human_action_create(strModelPath.UTF8String, ST_MOBILE_HUMAN_ACTION_DEFAULT_CONFIG_VID);

//加载其他模型,可以调用st_mobile_human_action_add_sub_model.
NSString *strEyeCenter = [[NSBundle mainBundle] pathForResource:@"xx" ofType:@"model"];
iRet = st_mobile_human_action_add_sub_model(_hDetector, strEyeCenter.UTF8String);

//需要注意的是:avatar贴纸需要加载眼球轮廓点模型以及avatar专用模型 (avatar_core.model)

//删除模型，用户可以根据需要动态的添加或删除模型，具体使用参考sample
iRet = st_mobile_human_action_remove_model_by_config(_hDetector, ST_MOBILE_ENABLE_FACE_EXTRA_DETECT);

//设置human action参数,此处设置手势2帧检测一次，可根据需要进行设置。
//其余可以设置的参数可参考st_mobile_human_action.h头文件。
iRet = st_mobile_human_action_setparam(_hDetector, ST_HUMAN_ACTION_PARAM_HAND_PROCESS_INTERVAL, 2);

```

## 2.人脸属性接口句柄的初始化

```

//face attribute句柄初始化
//获取人脸属性模型路径
NSString *strAttriModelPath = [[NSBundle mainBundle] pathForResource:@"face_attribute" ofType:@"model"];
//创建人脸属性句柄
iRet = st_mobile_face_attribute_create(strAttriModelPath.UTF8String, &_hAttribute);

```

## 3.美颜接口句柄初始化

```
iRet = st_mobile_beautify_create(&_hBeautify);
```

```
//美颜句柄创建成功后，可以设置美颜相关参数
```

```
// 设置美白参数，范围[0,1.0]，默认值0.02，0.0不做美白
```

```
st_mobile_beautify_setparam(_hBeautify, ST_BEAUTIFY_WHITEN_STRENGTH, self.fWhitenStreng
```

```
st_mobile_beautify_setparam(_hBeautify, ST_BEAUTIFY_WHITEN_STRENGTH, self.fWhitenStreng
```

```
//美白强度(2)，[0,1.0]，默认值0.0，0.0不做美白
```

```
st_mobile_beautify_setparam(_hBeautify, ST_BEAUTIFY_WHITEN2_STRENGTH, self.fWhitenStren
```

```
// 美白强度(3)，[0,1.0]，默认值0.0，0.0不做美白
```

```
st_mobile_beautify_setparam(_hBeautify, ST_BEAUTIFY_WHITEN3_STRENGTH, self.fWhitenStren
```

```
// 设置默认红润参数，范围[0,1.0]，默认值0.36，0.0不做红润
```

```
st_mobile_beautify_setparam(_hBeautify, ST_BEAUTIFY_REDDEN_STRENGTH, self.fReddenStreng
```

```
// 磨皮模式，默认值2.0，0.0表示只对人脸磨皮，1.0表示对全图磨皮，2.0表示精细化磨皮
```

```
st_mobile_beautify_setparam(_hBeautify, ST_BEAUTIFY_SMOOTH_MODE, 2.0);
```

```
// 设置默认磨皮参数，范围[0,1.0]，默认值0.74，0.0不做磨皮
```

```
st_mobile_beautify_setparam(_hBeautify, ST_BEAUTIFY_SMOOTH_STRENGTH, self.fSmoothStreng
```

```
// 设置默认磨皮参数，范围[0,1.0]，默认值0.74，0.0不做磨皮
```

```
setBeautifyParam(_hBeautify, ST_BEAUTIFY_SMOOTH_STRENGTH, self.fSmoothStrength2);
```

```
// 设置默认大眼参数，范围[0,1.0]，默认值0.13，0.0不做大眼效果
```

```
st_mobile_beautify_setparam(_hBeautify, ST_BEAUTIFY_ENLARGE_EYE_RATIO, self.fEnlargeEye
```

```
// 设置默认瘦脸参数，范围[0,1.0]，默认值0.11，0.0不做瘦脸效果
```

```
st_mobile_beautify_setparam(_hBeautify, ST_BEAUTIFY_SHRINK_FACE_RATIO, self.fShrinkFace
```

```
// 设置小脸参数，范围[0,1.0]，默认值0.10，0.0不做小脸效果
```

```
st_mobile_beautify_setparam(_hBeautify, ST_BEAUTIFY_SHRINK_JAW_RATIO, self.fShrinkJawSt
```

```
// 窄脸比例，[0,1.0]，默认值0.0，0.0不做窄脸效果
```

```
st_mobile_beautify_setparam(_hBeautify, ST_BEAUTIFY_NARROW_FACE_RATIO, self.fShrinkJawS
```

```
///SDK新增3D美颜功能，具体参数设置如下
```

```
/// 瘦鼻比例，[0, 1.0]，默认值为0.0，0.0不做瘦鼻
```

```
st_mobile_beautify_setparam(_hBeautify, ST_BEAUTIFY_NARROW_NOSE_RATIO, self.fNarrowNose
```

```
/// 鼻子长短比例，[-1, 1]，默认值为0.0，[-1, 0]为短鼻，[0, 1]为长鼻
```

```
st_mobile_beautify_setparam(_hBeautify, ST_BEAUTIFY_LENGTH_NOSE_RATIO, self.fLongNoseSt
```

```
/// 下巴长短比例，[-1, 1]，默认值为0.0，[-1, 0]为短下巴，[0, 1]为长下巴
```

```
st_mobile_beautify_setparam(_hBeautify, ST_BEAUTIFY_LENGTH_CHIN_RATIO, self.fChinStreng
```

```
/// 嘴型比例，[-1, 1]，默认值为0.0，[-1, 0]为放大嘴巴，[0, 1]为缩小嘴巴
```

```
st_mobile_beautify_setparam(_hBeautify, ST_BEAUTIFY_SIZE_MOUTH_RATIO, self.fMouthStreng
```

```
/// 人中长短比例，[-1, 1]，默认值为0.0，[-1, 0]为长人中，[0, 1]为短人中
```

```
st_mobile_beautify_setparam(_hBeautify, ST_BEAUTIFY_LENGTH_PHILTRUM_RATIO, self.fPhiltr
```

```
/// 发际线高低比例，[-1, 1]，默认值为0.0，[-1, 0]为低发际线，[0, 1]为高发际线
```

```
st_mobile_beautify_setparam(_hBeautify, ST_BEAUTIFY_LOW_HAIRLINE_RATIO, self.fHairLineS
```

```
//设置对比度参数，范围[0,1.0]，默认0.0
```

```
iRet = st_mobile_beautify_setparam(_hBeautify, ST_BEAUTIFY_CONTRAST_STRENGTH, self.fCon
```

```
//设置饱和度参数，范围[0,1.0]，默认0.0
```

```
iRet = st_mobile_beautify_setparam(_hBeautify, ST_BEAUTIFY_SATURATION_STRENGTH, self.fS
```

```
//锐化参数
```

```
iRet = st_mobile_beautify_setparam(_hBeautify, ST_BEAUTIFY_SHARPEN_STRENGTH, self.fShar
```

//SDK美体功能，该功能可设置参数如下：

//设置美体参数，范围 $[0, +\infty)$ ，默认1.0，1.0不做美体

iRet = st\_mobile\_beautify\_setparam(\_hBeautify, ST\_BEAUTIFY\_BODY\_WHOLE\_RATIO, self.fBeau

//设置美头参数，范围 $[0, +\infty)$ ，默认1.0，1.0不做美头

iRet = st\_mobile\_beautify\_setparam(\_hBeautify, ST\_BEAUTIFY\_BODY\_HEAD\_RATIO, self.fBeaut

//设置美肩参数，范围 $[0, +\infty)$ ，默认1.0，1.0不做美肩

iRet = st\_mobile\_beautify\_setparam(\_hBeautify, ST\_BEAUTIFY\_BODY\_SHOULDER\_RATIO, self.fB

//设置美腰参数，范围 $[0, +\infty)$ ，默认1.0，1.0不做美腰

iRet = st\_mobile\_beautify\_setparam(\_hBeautify, ST\_BEAUTIFY\_BODY\_WAIST\_RATIO, self.fBeau

//设置美臀参数，范围 $[0, +\infty)$ ，默认1.0，1.0不做美臀

iRet = st\_mobile\_beautify\_setparam(\_hBeautify, ST\_BEAUTIFY\_BODY\_HIP\_RATIO, self.fBeauti

//设置美腿参数，范围 $[0, +\infty)$ ，默认1.0，1.0不做美腿

iRet = st\_mobile\_beautify\_setparam(\_hBeautify, ST\_BEAUTIFY\_BODY\_LEG\_RATIO, self.fBeauti

//新增小头比例， $[0, 1.0]$ ，默认值0.0，0.0不做小头效果

iRet = st\_mobile\_beautify\_setparam(\_hBeautify, ST\_BEAUTIFY\_THINNER\_HEAD\_RATIO, self.fBe

#### 4.贴纸接口句柄的初始化



```

//初始化贴纸handle
iRet = st_mobile_sticker_create(&_hSticker);

//是否等待素材加载完成，用于希望等待模型加载完毕再渲染的场景，比如单帧或较短视频的3D绘制等
//在handle初始化之后调用，sample在处理图片时调用了此方法。
iRet = st_mobile_sticker_set_param_bool(_hSticker, -1, ST_STICKER_PARAM_WAIT_MATERIAL_LOAD);

//如需使用音乐贴纸，需要设置回调
st_mobile_sticker_set_param_ptr(_hSticker, -1, ST_STICKER_PARAM_SOUND_LOAD_FUNC_PTR, load_sound);
st_mobile_sticker_set_param_ptr(_hSticker, -1, ST_STICKER_PARAM_SOUND_PLAY_FUNC_PTR, play_sound);
st_mobile_sticker_set_param_ptr(_hSticker, -1, ST_STICKER_PARAM_SOUND_STOP_FUNC_PTR, stop_sound);

//加载音乐
void load_sound(void* sound, const char* sound_name, int length) {
    if ([messageManager.delegate respondsToSelector:@selector(loadSound:name:)]) {
        NSData *soundData = [NSData dataWithBytes:sound length:length];
        NSString *strName = [NSString stringWithUTF8String:sound_name];
        [messageManager.delegate loadSound:soundData name:strName];
    }
}

//播放音乐
void play_sound(const char* sound_name, int loop) {
    if ([messageManager.delegate respondsToSelector:@selector(playSound:loop:)]) {
        NSString *strName = [NSString stringWithUTF8String:sound_name];
        [messageManager.delegate playSound:strName loop:loop];
    }
}

//停止播放
void stop_sound(const char* sound_name) {
    if ([messageManager.delegate respondsToSelector:@selector(stopSound:)]) {
        NSString *strName = [NSString stringWithUTF8String:sound_name];
        [messageManager.delegate stopSound:strName];
    }
}

```

## 5.滤镜接口句柄的初始化

```

//初始化滤镜handle
iRet = st_mobile_gl_filter_create(&_hFilter);

```

## 6.通用物体接口句柄初始化

```

//初始化贴纸handle
st_result_t iRet = st_mobile_object_tracker_create(&_hTracker);

```

## 7.avatar句柄初始化

```

NSString *strAvatarModelPath = [[NSBundle mainBundle] pathForResource:@"M_SenseME_Avatar_Core" ofType:
iRet = st_mobile_avatar_create(&_avatarHandle, strAvatarModelPath.UTF8String);
if (iRet != ST_OK) {
    NSLog(@"st mobile avatar create failed: %d", iRet);
} else {
    self.avatarConfig = st_mobile_avatar_get_detect_config(_avatarHandle);
}

```

SDK句柄的初始化，到这里介绍完毕，更多细节可以在官方提供Demo中一探究竟。  
 注意：因为在处理美颜、滤镜的时候SDK需要在统一上下文环境，设置方法如下：

```

if ([EAGLContext currentContext] != self.glContext) {
[EAGLContext setCurrentContext:self.glContext];
}

```

注意：要保证OpenGL上下文环境相同，否则会有错误。

## 3.2 纹理的获取

### 1.使用STCamera

STCamera:是商汤科技对iOS系统相机的封装，可以方便的实现分辨率、输出格式等的设置，更多信息请看STCamera。  
`self.stCamera = [[STCamera alloc] initWithDevicePosition:AVCaptureDevicePositionFront//i`  
`sessionPreset:self.currentSessionPreset` //分辨率  
`fps:30` //帧率  
`needYuvOutput:NO];` //输出数据格式  
`self.stCamera.delegate = self;` //设置相机数据代理

### 2.从STCamera获取纹理

```

//从STCameraDelegate中获取帧数据转换为纹理
- (void)captureOutput:(AVCaptureOutput *)captureOutput didOutputSampleBuffer:(CMSampleBuffer *)sampleBuffer {
//获取每一帧图像信息
CVPixelBufferRef pixelBuffer = (CVPixelBufferRef)CMSampleBufferGetImageBuffer(sampleBuffer);
//锁定一帧数据
CVPixelBufferLockBaseAddress(pixelBuffer, 0);
//获得视频数据地址
unsigned char* pBGRAImageIn = (unsigned char *)CVPixelBufferGetBaseAddress(pixelBuffer);
//获取视频数据宽高
int iBytesPerRow = (int)CVPixelBufferGetBytesPerRow(pixelBuffer);
int iWidth = (int)CVPixelBufferGetWidth(pixelBuffer);
int iHeight = (int)CVPixelBufferGetHeight(pixelBuffer);
//关联pixelBuffer和texture
CVReturn cvRet = CVOpenGLTextureCacheCreateTextureFromImage(kCFAAllocatorDefault,
_cvTextureCache, //纹理缓存
pixelBuffer, //输出视频数据buffer
NULL,
GL_TEXTURE_2D, //2D纹理
GL_RGBA, //颜色格式
self.imageWidth, //图像宽度
self.imageHeight, //图像高度
GL_BGRA, //iOS format
GL_UNSIGNED_BYTE,
0,
&_cvTextureOrigin //输出纹理
);

if (!_cvTextureOrigin || kCVReturnSuccess != cvRet) {

NSLog(@"CVOpenGLTextureCacheCreateTextureFromImage %d" , cvRet);

return NO;
}
//获取纹理
_textureOriginInput = CVOpenGLTextureGetName(_cvTextureOrigin);
//绑定纹理
glBindTexture(GL_TEXTURE_2D , _textureOriginInput);
//纹理过滤函数，图像从纹理图像空间映射到帧缓冲图像空间(映射需要重新构造纹理图像,这样就会造成应用到多边形.
//GL_TEXTURE_2D:表示处理2D纹理
//GL_TEXTURE_MIN_FILTER:缩小过滤
//GL_TEXTURE_MAG_FILTER:放大过滤
//GL_TEXTURE_WRAP_S: S方向上的贴图模式，纹理坐标st，对应物理坐标xy
//GL_TEXTURE_WRAP_T: T方向上的贴图模式
//GL_CLAMP_TO_EDGE:纹理坐标的范围是[0,1],如果某个方向上的纹理坐标小于0，那么取0；如果大于1，则取1
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR); //对2D纹理进行缩小过滤，返回
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR); //对2D纹理进行放大过滤，返回
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE); //对2D纹理在S方向上进行
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE); //对2D纹理在T方向上进行
glBindTexture(GL_TEXTURE_2D, 0); //绑定一个默认纹理，之前绑定的纹理失效

```

```
};
```

### 3.3 纹理预处理

```
//初始化纹理
- (void)initResultTexture
{
//创建美颜纹理
[self setupTextureWithPixelBuffer:&_cvBeautifyBuffer w:self.imageWidth h:self.imageHeight]
//创建贴纸纹理
[self setupTextureWithPixelBuffer:&_cvStickerBuffer w:self.imageWidth h:self.imageHeight]
//创建滤镜纹理
[self setupTextureWithPixelBuffer:&_cvFilterBuffer w:self.imageWidth h:self.imageHeight]
}

- (BOOL)setupTextureWithPixelBuffer:(CVPixelBufferRef *)pixelBufferOut w:(int)iWidth h:(int)iHeight
{
//创建一个数组
CFDictionaryRef empty = CFDictionaryCreate(kCFAllocatorDefault, NULL, NULL, 0, &kCFTypedef, NULL, 0);
//创建一个动态数组
CFMutableDictionaryRef attrs = CFDictionaryCreateMutable(kCFAllocatorDefault, 1, &kCFTypedef, NULL, 0);
//设置Value
CFDictionarySetValue(attrs, kCVPixelBufferIOSurfacePropertiesKey, empty);
//创建pixelBuffer
CVReturn cvRet = CVPixelBufferCreate(kCFAllocatorDefault, iWidth, iHeight, kCVPixelFormatType_32ARGB, pixelBufferOut);
if(kCVReturnSuccess != cvRet){
NSLog(@"CVPixelBufferCreate %d", cvRet);
}
//关联buffer和texture
cvRet = CVOpenGLESTextureCacheCreateTextureFromImage(kCFAllocatorDefault, _cvTextureCache, pixelBufferOut, 0, kCGLTextureOptions_0, kCGLPixelFormatType_32ARGB, &glTexture);
if(kCVReturnSuccess != cvRet){
NSLog(@"CVOpenGLESTextureCacheCreateTextureFromImage %d", cvRet);
return NO;
}
//释放资源
CFRelease(attrs);
CFRelease(empty);
//获得纹理指针
*glTexture = CVOpenGLESTextureGetName(*cvTexture);
//绑定纹理
glBindTexture(CVOpenGLESTextureGetTarget(*cvTexture), *glTexture);
//设置纹理属性
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
glBindTexture(GL_TEXTURE_2D, 0);
}
```

## 3.4 帧处理流程

### 1.HumanAction接口使用

```
//检测人脸关键点，脸部动作、表情，手势，前后背景分割，肢体及肢体动作，需要通过cofig进行配置
//只有配置了对应的config，检测结果中才会有相应数据
//检测结果的结构定义在st_mobile_human_action.h中，具体使用参考sample
iRet = st_mobile_human_action_detect(_hDetector, //人脸检测句柄
pBGRAImageIn, //输出数据地址
ST_PIX_FMT_BGRA8888, //图像格式
iWidth, //宽
iHeight, //高
iBytesPerRow, //iWidth * 4(4通道bgra)
stMobileRotate, //旋转角度
iConfig, //动作检测
&detectResult //检测结果
);
```

旋转角度含义：拿一张照片对照着手机进行旋转，看看旋转到哪个方向能检测到人脸，此角就是人脸检测的设置角度，详见enum：

```
typedef enum {
ST_CLOCKWISE_ROTATE_0 = 0,    ///< 图像不需要旋转,图像中的人脸为正脸
ST_CLOCKWISE_ROTATE_90 = 1,   ///< 图像需要顺时针旋转90度,使图像中的人脸为正
ST_CLOCKWISE_ROTATE_180 = 2,  ///< 图像需要顺时针旋转180度,使图像中的人脸为正
ST_CLOCKWISE_ROTATE_270 = 3   ///< 图像需要顺时针旋转270度,使图像中的人脸为正
} st_rotate_type;
```

### 2.expression接口使用

该接口需要使用human action detect的结果因此，需要在detect之后调用，使用方法如下：

```
//expression检测结果，每个元素代表的表情定义在ST_MOBILE_EXPRESSION枚举中
bool expressionResult[128] = {0};
st_result_t iRet = st_mobile_get_expression(&detectResult, stMobileRotate, N0, expressionResult);

//设置expression阈值,推荐使用默认阈值,无需手动设置
iRet = st_mobile_set_expression_threshold(ST_MOBILE_EXPRESSION_HEAD_NORMAL, 0.5);
```

3.人脸属性接口的使用。FaceAttribute接口的输入参数依赖于HumanAction参数的输出，也就是说运行人脸属性之前需要先做HumanAction：

```
iRet = st_mobile_face_attribute_detect(_hAttribute,
pBGRAImageIn,
ST_PIX_FMT_BGRA8888,
iWidth,
iHeight,
iBytesPerRow,
_pFacesDetection,
1, // 这里仅取一张脸也就是第一张脸的属性作为演示
&pAttrArray);
```

4.美颜接口的使用。同样的，美颜的大眼瘦脸等功能也依赖于HumanAction：

```
//美颜处理
iRet = st_mobile_beautify_process_texture(_hBeautify,
                                          _textureOriginInput,
                                          iWidth,
                                          iHeight,
                                          &detectResult,
                                          _textureBeautifyOutput,
                                          &processedResult);
```

其中，detectResult是human action检测结果，processedResult是经过美颜（大眼瘦脸等）之后的脸部关键点信息（processedResult需上层手动分配，并且需要和detectResult相同，也可直接传入detectResult）。

美颜参数设置：

```
//设置美颜参数
iRet = st_mobile_beautify_setparam(_hBeautify, ST_BEAUTIFY_REDDEN_STRENGTH, 0.36f); //红润强度，[0,1.0]，
```

5.贴纸接口的使用。贴纸主要包括：2D贴纸，3D贴纸，脸部变形贴纸、前后背景贴纸、手势贴纸、音乐贴纸、美妆贴纸、粒子动画贴纸、avatar贴纸、天空盒贴纸，贴纸功能的实现需要人脸信息，需先做HumanAction。

渲染贴纸

```
iRet = st_mobile_sticker_process_texture(_hSticker, //贴纸句柄
                                          _textureOriginInput, //输入纹理
                                          iWidth, //宽
                                          iHeight, //高
                                          stMobileRotate, //旋转角度
                                          ST_CLOCKWISE_ROTATE_0, //前景贴纸旋转角度
                                          false, //是否镜像
                                          &detectResult, //human action检测结果
                                          &inputEvent, //一些硬件参数和自定义事件
                                          _textureStickerOutput //处理之后的图像数据);
```

注意：

- 1.该接口增加了前景贴纸旋转角度参数，根据需要进行使用。
- 2.如需使用天空盒贴纸，需要传入相机四元数，如果不使用天空盒inputEvent传入NULL即可。

```

st_mobile_input_params_t inputEvent;
memset(&inputEvent, 0, sizeof(st_mobile_input_params_t));
int type = ST_INPUT_PARAM_NONE;
iRet = st_mobile_sticker_get_needed_input_params(_hSticker, &type);
if (CHECK_FLAG(type, ST_INPUT_PARAM_CAMERA_QUATERNION)) {

    CMDeviceMotion *motion = self.motionManager.deviceMotion;
    inputEvent.camera_quaternion[0] = motion.attitude.quaternion.x;
    inputEvent.camera_quaternion[1] = motion.attitude.quaternion.y;
    inputEvent.camera_quaternion[2] = motion.attitude.quaternion.z;
    inputEvent.camera_quaternion[3] = motion.attitude.quaternion.w;
    if (self.stCamera.devicePosition == AVCaptureDevicePositionBack) {
        inputEvent.is_front_camera = false;
    } else {
        inputEvent.is_front_camera = true;
    }
} else {
    inputEvent.camera_quaternion[0] = 0;
    inputEvent.camera_quaternion[1] = 0;
    inputEvent.camera_quaternion[2] = 0;
    inputEvent.camera_quaternion[3] = 1;
}

```

具体使用方法参考sample。

切换贴纸：

```

st_result_t iRet = st_mobile_sticker_change_package(_hSticker, stickerPath, NULL);

```

重新播放素材

```

if (_hSticker) {
    st_mobile_sticker_replay_package(_hSticker, packageId);
}

```

获取贴纸触发动作：(需要在st\_mobile\_sticker\_change\_package之后调用才可以获取,具体的action定义在头文件中)

```

st_result_t iRet = st_mobile_sticker_get_trigger_action(_hSticker, &action);

```

6.滤镜接口的使用：

```

//设置滤镜风格, 必须在OpenGL线程中调用
st_result_t iRet = st_mobile_gl_filter_set_style(_hFilter, [model.strPath UTF8String]);
//设置滤镜特效强度, 范围[0,1], 设置为0无滤镜效果, 设置为1效果最强
iRet = st_mobile_gl_filter_set_param(_hFilter, ST_GL_FILTER_STRENGTH, self.fFilterStrength);
//滤镜处理函数
iRet = st_mobile_gl_filter_process_texture(_hFilter, textureResult, iWidth, iHeight, _textureFilterOutp

```

## 7.通用物体接口的使用

```

//设置跟踪区域, 只需设置一次, reset之后需要重新设置
st_result_t iRet =
st_mobile_object_tracker_set_target(
st_handle_t handle,           //已初始化的通用物体跟踪句柄
const unsigned char *image,   //用于检测的图像
st_pixel_format pixel_format, //用于检测的图像数据的像素格式, 内部会统一转换成灰度图
int image_width,              //用于检测的图像的宽度 (以像素为单位)
int image_height,             //用于检测的图像的高度 (以像素为单位)
int image_stride,             //用于检测的图像的跨度 (以像素为单位), 即每行的字节数
st_rect_t* target_rect       //输入指定目标的矩形框, 目前只能跟踪2^n的正方形
);

//对连续视频帧中的目标进行实时快速跟踪
st_result_t iRet =
st_mobile_object_tracker_track(
st_handle_t handle,           //已初始化的实时通用物体跟踪句柄
const unsigned char *image,   //用于检测的图像数据, 同上
st_pixel_format pixel_format, //用于检测的图像数据的像素格式, 同上
int image_width,              //用于检测的图像的宽度, 同上
int image_height,             //用于检测的图像的高度, 同上
int image_stride,             //用于检测的图像的跨度, 同上
st_rect_t *result_rect        //输出实际跟踪的矩形框的新位置
float *result_score            //置信度, 根据需要设置(0,1), 用来判断是否追踪失败。
);

//重置通用物体跟踪句柄
st_mobile_object_tracker_reset(
st_handle_t handle //通用物体跟踪句柄
);

```

## 3.5 SDK句柄的释放

### 1.GL资源必须在GL线程释放



```

- (void)releaseResources
{
    if ([EAGLContext currentContext] != self.glContext) {
        [EAGLContext setCurrentContext:self.glContext];
    }
    //释放贴纸句柄
    if (_hSticker) {
        st_mobile_sticker_destroy(_hSticker);
        _hSticker = NULL;
    }
    //释放美颜句柄
    if (_hBeautify) {
        st_mobile_beautify_destroy(_hBeautify);
        _hBeautify = NULL;
    }
    //释放人脸检测句柄
    if (_hDetector) {
        st_mobile_human_action_destroy(_hDetector);
        _hDetector = NULL;
    }
    //释放人脸属性句柄
    if (_hAttribute) {
        st_mobile_face_attribute_destroy(_hAttribute);
        _hAttribute = NULL;
    }
    //释放滤镜句柄
    if (_hFilter) {
        st_mobile_gl_filter_destroy(_hFilter);
        _hFilter = NULL;
    }
    //释放通用物体句柄
    if (_hTracker) {
        st_mobile_object_tracker_destroy(_hTracker);
        _hTracker = NULL;
    }
    //释放avatar句柄
    if (_avatarHandle) {
        st_mobile_avatar_destroy(_avatarHandle);
        _hSticker = NULL;
    }
    //释放检测输出人脸信息数组
    if (_pFacesDetection) {
        free(_pFacesDetection);
        _pFacesDetection = NULL;
    }
    //释放美颜输出人脸信息数组
    if (_pFacesBeautify) {
        free(_pFacesBeautify);
        _pFacesBeautify = NULL;
    }
    //释放纹理资源

```

```

[self releaseResultTexture];
//释放纹理缓存
if (_cvTextureCache) {
CFRelease(_cvTextureCache);
_cvTextureCache = NULL;
}

[EAGLContext setCurrentContext:nil];
}

```

小结：第3部分对SDK各个接口的使用作了基本的介绍，结合Demo的展示，对SDK的集成可以更快的上手，在这里要强调几点：

- 1.人脸属性检测、美颜、贴纸的都要基于humanAction，因此应该将humanAction的检测放在其他三者之前；
- 2.美颜、贴纸、滤镜都是使用OpenGL做渲染，因此要保处在同一个EAGLContext中，具体实现可见：

```

if ([EAGLContext currentContext] != self.glContext) {
[EAGLContext setCurrentContext:self.glContext];
}

```

## 3.6 模式选择

设置性能/效果优先级倾向，引擎内部会根据设置调整渲染策略

```

typedef enum {
    ST_PREFER_EFFECT,           //效果优先
    ST_PREFER_PERFORMANCE,     //性能优先
    ST_PREFER_AUTO_TUNE,
    ST_PREFER_NOTHING,
} st_performance_hint_t;

st_mobile_sticker_set_performance_hint(_hSticker, st_performance_hint_t);
st_mobile_makeup_set_performance_hint(_hBmpHandle, st_performance_hint_t);

```

## 4 客户自定义

要想使美颜功能和贴纸功能能够正常使用，SDK要求首先开启人脸检测功能，方法如下所示：

### 4.1 人脸检测

```

//开启人脸检测
iRet = st_mobile_human_action_detect(_hDetector, //人脸检测句柄
pBGRAImageIn, //输出数据地址
ST_PIX_FMT_BGRA8888, //图像格式
iWidth, //宽
iHeight, //高
iBytesPerRow, //iWidth * 4(4通道bgra)
stMobileRotate, //旋转角度(见3.4.1 HumanAction接口使用中旋转角度的解释)
iConfig, //动作检测, 用户可以通过修改该参数来检测特定的动作, 具体的config在st_mobile_human_action中定义
&detectResult //检测结果
);

```

## 4.2 美颜

需要先开启人脸检测功能

开启美颜功能

方法一（仅输出texture）：

```

iRet = st_mobile_beautify_process_texture(_hBeautify, //美颜句柄
                                           _textureOriginInput, //输入纹理
                                           iWidth, //宽
                                           iHeight, //高
                                           &detectResult, //human action结果
                                           _textureBeautifyOutput, //美颜输出纹理
                                           &processedResult //美颜过后的detectResult
);

```

当要将美颜后的数据用于推流或录制小视频时，可以读取在纹理预处理阶段创建的\_cvBeafigyBuffer中的数据。

方法二（既输出texture又输出buffer）：

```

st_mobile_beautify_process_and_output_texture(
    st_handle_t handle, //已初始化的美颜句柄
    unsigned int textureid_src, //待处理的纹理id
    int image_width, int image_height, //输入纹理的宽度和高度
    const st_mobile_human_action_t* p_human_action_t, //人脸信息
    unsigned int textureid_dst, //输出纹理
    unsigned char *img_out, st_pixel_format fmt_out, //输出buffer, 输出数据格式
    st_mobile_human_action_t* p_huaman_out //美颜之后人脸信息
);

```

PS：如果是推流的应用场景

- 1.使用读取在纹理预处理阶段创建的\_cvBeafigyBuffer中的数据，详细内容可参考说明文档中3.3纹理预处理中buffer的创建方法；
- 2.可用方法二输出buffer的形式来处理，但是考虑到glReadPixels的性能问题，建议不要使用该方法；

## 4.3 贴纸

需要先开启人脸检测功能

开启贴纸功能

方法一:

```
iRet = st_mobile_sticker_process_texture(_hSticker ,//贴纸句柄
_textureOriginInput,//输入纹理
iWidth,//宽
iHeight,//高
stMobileRotate,//旋转角度
ST_CLOCKWISE_ROTATE_0,//前景贴纸旋转角度，根据需要设置
false,//是否镜像
&detectResult,//human action检测结果
-1, //用户自定义事件
_textureStickerOutput//经过贴纸处理之后的图像数据
);
```

当要将添加了贴纸后的数据用于推流或录制小视频时，可以读取在纹理预处理阶段创建的\_cvStickerBuffer中的数据。

方法二:

```
st_mobile_sticker_process_and_output_texture(
st_handle_t handle,//已初始化的贴纸句柄
unsigned int textureid_src,//输入纹理
int image_width, //宽
int image_height,//高
st_rotate_type rotate,//旋转角度
st_rotate_type frontStickerRotate,//前景贴纸旋转角度，按需设置
bool need_mirror,//是否镜像
p_st_mobile_human_action_t human_action,//human action检测结果
int custom_event,//用户自定义事件
unsigned int textureid_dst,//经过贴纸处理之后的图像数据
unsigned char* img_out, //输出buffer
st_pixel_format fmt_out//输出图像数据格式
);
```

输出buffer中的数据是添加了贴纸之后的数据，可以用于推流或者小视频录制，但是考虑到glReadPixels的性能问题，建议不要使用该方

## 4.4 美妆

需要先开启人脸检测功能

开启美妆功能

方法一：

```
iRet = st_mobile_makeup_create(  
st_handle_t handle, //已初始化的美妆句柄  
);
```

方法二：

```
st_mobile_makeup_add_makeup_for_type(  
st_handle_t handle, //已初始化的美妆句柄  
st_makeup_type makeup_type, //制定素材包所属的type，定义如st_makeup_type  
const char* zip_path, //素材包文件路径  
int* package_id, //成功返回ST_OK，失败返回其他错误码，错误码定义在st_mobile_common.h中，如ST_E_FAIL等  
)
```

方法三：

```
st_mobile_makeup_process_texture(  
st_handle_t handle, //已初始化的美妆句柄  
unsigned int texture_src, //输入纹理  
int image_width, //宽  
int image_height, //高  
st_rotate_type rotate, //旋转角度  
st_mobile_human_action_t* human_action, //human action检测结果  
unsigned int texture_dst //经过贴纸处理之后的图像数据  
);
```

输出buffer中的数据是添加了美妆之后的数据，可以用于推流或者小视频录制，但是考虑到glReadPixels的性能问题，建议不要使用该方

小结：第4部分给出了单独开启美颜功能和单独开启贴纸功能的方法，客户可以根据上述思路，自由的组合功能，得到自己想要的效果。

## 5 集成注意事项

### 禁止在后台进行OpenGL的相关操作

## 6 检测渲染优化

sample使用检测渲染并行策略，即使用一个线程检测，另一个线程渲染。对于iOS来说我们直接在相机回调的queue中进行human action检测，检测完成以后在另一个同步的queue中渲染，具体实现如下（其中省略了部分代码，完整代码请参考sample）：

- (void)captureOutput:(AVCaptureOutput \*)captureOutput didOutputSampleBuffer:(CMSampleBufferRef)sampleB

```
//缓存两帧
if (self.iBufferedCount >= 2) {
    return;
}
//human action检测
st_result_t iRet = st_mobile_human_action_detect(_hDetector,

pBGRAIr
ST_PIX_
iWidth,
iHeight
iBytesP
stMobil
self.iC
&detect

self.iBufferedCount ++;
CFRetain(pixelBuffer);
//拷贝human action检测结果, 用于另一线程渲染
__block st_mobile_human_action_t newDetectResult;
memset(&newDetectResult, 0, sizeof(st_mobile_human_action_t));
copyHumanAction(&detectResult, &newDetectResult);

//渲染线程
dispatch_async(self.renderQueue, ^{

    //美颜, 贴纸等操作

    //释放拷贝的human action结果
    freeHumanAction(&newDetectResult);
    //结果渲染
    [self.glPreview renderTexture:textureResult];

    CFRelease(pixelBuffer);
    self.iBufferedCount --;
});
}
```

其中, copyHumanAction、freeHumanAction两个方法可参考sample。