Nama API 参考文档

Class: PUBLIC Author: Hao Qin Last Update: 2018/3/28

本文是相芯人脸跟踪及视频特效开发包（以下简称 Nama SDK）的底层接口文档。该文档中的 Nama API 为底层 native 接口，可以直接用于 PC/iOS/Android NDK/Linux 上的开发。其中，iOS和Android平台上的开发可以利用SDK的应用层接口（Objective-C/Java），相比本文中的底层接口会更贴近平台相关的开发经验。

SDK相关的所有调用要求在同一个线程中顺序执行，不支持多线程。少数接口可以异步调用（如道具加载），会在备注中特别注明。SDK所有主线程调用的接口需要保持 OpenGL context 一致，否则会引发纹理数据异常。如果需要用到SDK的绘制功能，则主线程的所有调用需要预先初始化OpenGL环境，没有初始化或初始化不正确会导致崩溃。我们对OpenGL的环境要求为 GLES 2.0 以上。具体调用方式，可以参考各平台 demo。

底层接口根据作用逻辑归为五类：初始化、加载道具、主运行接口、销毁、功能接口、P2A相关接口。

初始化

fuSetup 函数

初始化系统环境，加载系统数据，并进行网络鉴权。必须在调用SDK其他接口前执行，否则会引发崩溃。

```
int fuSetup(float* v3data, float* ardata, void* authdata, int sz_authdata);
```

参数

v3data [in]： 内存指针，指向SDK提供的 v3.bundle 文件内容

ardata [in]： 已废弃

authdata [in]： 内存指针，指向鉴权数据的内容。如果是用包含 authpack.h 的方法在编译时提供鉴权数据，则这里可以写为 `g_auth_package` 。

sz_authdata [in]：鉴权数据的长度，以字节为单位。如果鉴权数据提供的是 authpack.h 中的 `g_auth_package`，这里可写作 `sizeof(g_auth_package)`

返回值

返回非0值代表成功，返回0代表失败。如初始化失败，可以通过 `fuGetSystemError` 获取错误代码。

备注

根据应用需求，鉴权数据也可以运行时提供（如网络下载），不过要注意证书泄露风险，防止证书被滥用。

数据长度采用了 `int`，是为了防止跨平台的数据类型问题。

加载道具包

fuCreateItemFromPackage 函数

加载道具包，使其可以在主运行接口中被执行。一个道具包可能是一个功能模块或者多个功能模块的集合，加载道具包可以在流水线中激活对应的功能模块，在同一套SDK调用逻辑中实现即插即用。

```
int fuCreateItemFromPackage(void* data, int sz);
```

参数

data [in]： 内存指针，指向所加载道具包的内容。道具包通常以 *.bundle 结尾。

sz [in]：道具包内容的数据长度，以字节为单位。

返回值

一个整数句柄，作为该道具在系统内的标识符。

备注

该接口可以和主线程异步执行。为了降低加载道具阻塞主线程，建议异步调用该接口。

fuItemSetParam 函数

修改或设定道具包中变量的值。可以支持的道具包变量名、含义、及取值范围需要参考道具的文档。

```
// 设定道具中的double类型变量
int fuItemSetParamd(int item,char* name,double value);
// 设定道具中的double数组类型变量
int fuItemSetParamdv(int item,char* name,double* value,int n);
// 设定道具中的字符串类型变量
int fuItemSetParams(int item,char* name,char* value);
```

参数

item [in]： 道具表示符，内容应为调用 `fuCreateItemFromPackage` 函数的返回值，并且道具内容没有被销毁。

name [in]：字符串指针，内容为要设定的道具变量名。

value [in]：要设定的变量值，数据类型对应不同的函数接口。

n [in]：设定变量为double数组时，数组数据的长度，以double为单位。

返回值

返回非0值代表成功，返回0代表失败。

备注

设定变量为字符串时，要求字符串以0结尾。

fuItemGetParam 函数

获取道具中变量的值。可以支持的道具包变量名、含义、及取值范围需要参考道具的文档。

```
// 获取道具中的浮点类型变量
double fuItemGetParamd(int item,char* name);
```

参数

item [in]： 道具表示符，内容应为调用 `fuCreateItemFromPackage` 函数的返回值，并且道具内容没有被销毁。

name [in]：字符串指针，内容为要获取的道具变量名。

返回值

要获取的变量的值。执行失败的情况下返回值为0。

备注

该接口可以和主线程异步执行。

```
// 获取道具中的字符串类型变量
int fuItemGetParams(int item,char* name,char* buf,int sz);
```

参数

item [in]： 道具表示符，内容应为调用 `fuCreateItemFromPackage` 函数的返回值，并且道具内容没有被销毁。

name [in]：字符串指针，内容为要修改的道具变量名。

buf [out]：内存指针，指向预分配的内存空间，用于接收函数返回的字符串内容。

sz [in]：在*buf*中预分配的最大内存长度，以字节为单位。

返回值

返回为变量字符串的长度。执行失败的情况下返回值为-1。

备注

该接口可以和主线程异步执行。

主运行接口

fuRenderItems 函数

将输入的图像数据，送入SDK流水线进行处理，并输出处理之后的图像数据。该接口会执行所有道具要求、且证书许可的功能模块，包括人脸检测与跟踪、美颜、贴纸或avatar绘制等。

该接口支持两种输入输出模式，输入RGBA纹理并输出RGBA纹理，或者输入BGRA数组输出BGRA数组。其中RGBA纹理为OpenGL纹理，该模式下传入输入纹理的ID，并在函数返回值中返回输出的纹理ID。BGRA数组为内存图像缓存，数据格式为8位4通道的图像，该模式下将输入图像通过参数传入，输出图像会覆盖到同一内存空间中。

```
int fuRenderItems(int texid,int* img,int w,int h,int frame_id, int* p_items,int n_items);
```

参数

texid [in]：纹理模式下输入的OpenGL纹理ID，非纹理模式下传0。

img [in & out]：内存数组模式下输入的图像数据，处理后的图像数据也覆盖在该内存空间中。纹理模式下该参数可以传NULL。

w [in]：输入的图像宽度。

h [in]：输入的图像高度。

frame_id [in]：当前处理的帧序列号，用于控制道具中的动画逻辑。

p_items [in]：内存指针，指向需要执行的道具标识符数组。其中每个标识符应为调用 `fuCreateItemFromPackage` 函数的返回值，并且道具内容没有被销毁。

n_items [in]：*p_items*数组中的道具个数。

返回值

处理之后的输出图像的纹理ID。

备注

即使在非纹理模式下，函数仍会返回输出图像的纹理ID。虽然该模式下输入输出都是内存数组，但是绘制工作仍然是通过GPU完成的，因此该输出图像的纹理ID同样存在。输出的OpenGL纹理为SDK运行时在当前OpenGL context中创建的纹理，其ID应与输入ID不同。输出后使用该纹理时需要确保OpenGL context保持一致。

该绘制接口需要OpenGL环境，环境异常会导致崩溃。

fuRenderItemsEx 函数

将输入的图像数据，送入SDK流水线进行处理，并输出处理之后的图像数据。该接口会执行所有道具要求、且证书许可的功能模块，包括人脸检测与跟踪、美颜、贴纸或avatar绘制等。

相比 `fuRenderItems` 该接口支持更加灵活多样的输入及输出模式，详细的输入输出格式列表参加后续章节 输入输出格式列表 。

```
int fuRenderItemsEx(
    int out_format,void* out_ptr,
    int in_format,void* in_ptr,
    int w,int h,int frame_id, int* p_items,int n_items);
```

参数

out_format [in]：输出的数据格式标识符。

out_ptr [out]：内存指针，指向输出的数据内容。

in_format [in]：输入的数据格式标识符。

in_ptr [in]：内存指针，指向输入的数据内容。

w [in]：输入的图像宽度。

h [in]：输入的图像高度。

frame_id [in]：当前处理的帧序列号，用于控制道具中的动画逻辑。

p_items [in]：内存指针，指向需要执行的道具标识符数组。其中每个标识符应为调用 `fuCreateItemFromPackage` 函数的返回值，并且道具内容没有被销毁。

n_items [in]：*p_items*数组中的道具个数。

返回值

处理之后的输出图像的纹理ID。

备注

即使在非纹理模式下，函数仍会返回输出图像的纹理ID。虽然输出的图像可能是多种可选格式，但是绘制工作总是通过GPU完成，因此输出图像的纹理ID始终存在。输出的OpenGL纹理为SDK运行时在当前OpenGL context中创建的纹理，其ID应与输入ID不同。输出后使用该纹理时需要确保OpenGL context保持一致。

该绘制接口需要OpenGL环境，环境异常会导致崩溃。

fuRenderItemsEx2 函数

将输入的图像数据，送入SDK流水线进行处理，并输出处理之后的图像数据。

相比 `fuRenderItemsEx` 该接口增加了用户对流水线的控制。通过传入流水线功能掩码参数，可以控制指定功能模块的开关，以及特定的绘制选项。通过传入道具掩码数组，可以控制每个道具在多人脸情况下对哪些人脸生效。

```
int fuRenderItemsEx2(
    int out_format,void* out_ptr,
    int in_format,void* in_ptr,
    int w,int h,int frame_id, int* p_items,int n_items,
    int func_flag, int* p_item_masks);
```

参数

out_format [in]：输出的数据格式标识符。

out_ptr [out]：内存指针，指向输出的数据内容。

in_format [in]：输入的数据格式标识符。

in_ptr [in]：内存指针，指向输入的数据内容。

w [in]：输入的图像宽度。

h [in]：输入的图像高度。

frame_id [in]：当前处理的帧序列号，用于控制道具中的动画逻辑。

p_items [in]：内存指针，指向需要执行的道具标识符数组。其中每个标识符应为调用 `fuCreateItemFromPackage` 函数的返回值，并且道具内容没有被销毁。

n_items [in]：*p_items*数组中的道具个数。

func_flag：流水线功能掩码，表示流水线启用的功能模组，以及特定的绘制选项。多个掩码通过运算符“或”进行连接。所有支持的掩码及其含义如下。

流水线功能掩码	含义
NAMA_RENDER_FEATURE_TRACK_FACE	人脸识别和跟踪功能
NAMA_RENDER_FEATURE_BEAUTIFY_IMAGE	输入图像美化功能
NAMA_RENDER_FEATURE_RENDER	人脸相关的绘制功能，如美颜、贴纸、人脸变形、滤镜等
NAMA_RENDER_FEATURE_ADDITIONAL_DETECTOR	其他非人脸的识别功能，包括背景分割、手势识别等
NAMA_RENDER_FEATURE_RENDER_ITEM	人脸相关的道具绘制，如贴纸
NAMA_RENDER_FEATURE_FULL	流水线功能全开
NAMA_RENDER_OPTION_FLIP_X	绘制选项，水平翻转
NAMA_RENDER_OPTION_FLIP_Y	绘制选项，垂直翻转

p_item_masks：道具掩码，表示在多人模式下，每个道具具体对哪几个人脸生效。该数组长度应和 *p_items* 一致，每个道具一个int类型掩码。掩码中，从int低位到高位，第i位值为1代表该道具对第i个人脸生效，值为0代表不生效。

返回值

处理之后的输出图像的纹理ID。

备注

即使在非纹理模式下，函数仍会返回输出图像的纹理ID。虽然输出的图像可能是多种可选格式，但是绘制工作总是通过GPU完成，因此输出图像的纹理ID始终存在。输出的OpenGL纹理为SDK运行时在当前OpenGL context中创建的纹理，其ID应与输入ID不同。输出后使用该纹理时需要确保OpenGL context保持一致。

该绘制接口需要OpenGL环境，环境异常会导致崩溃。

fuBeautifyImage 函数

将输入的图像数据，送入SDK流水线进行全图美化，并输出处理之后的图像数据。该接口仅执行图像层面的美化处理（包括滤镜、美肤），不执行人脸跟踪及所有人脸相关的操作（如美型）。由于功能集中，相比 `fuRenderItemsEx` 接口执行美颜道具，该接口所需计算更少，执行效率更高。

```
int fuBeautifyImage(
    int out_format,void* out_ptr,
    int in_format,void* in_ptr,
    int w,int h,int frame_id, int* p_items,int n_items);
```

参数

out_format [in]：输出的图像数据格式。

out_ptr [out]：内存指针，指向输出的图像数据。

in_format [in]：输入的图像数据格式。

in_ptr [in]：内存指针，指向输入的图像数据。

w [in]：输入的图像宽度。

h [in]：输入的图像高度。

frame_id [in]：当前处理的帧序列号，用于控制道具中的动画逻辑。

p_items [in]：内存指针，指向需要执行的道具标识符数组。其中每个标识符应为调用 `fuCreateItemFromPackage` 函数的返回值，并且道具内容没有被销毁。

n_items [in]：*p_items*数组中的道具个数。

返回值

处理之后的输出图像的纹理ID。

备注

该接口正常生效需要传入的道具中必须包含美颜道具（随SDK分发，文件名通常为 `face_beautification.bundle`）。传入道具中所有非美颜道具不会生效，也不会占用计算资源。

该绘制接口需要OpenGL环境，环境异常会导致崩溃。

fuTrackFace 函数

对于输入的图像数据仅执行人脸跟踪操作，其他所有图像和绘制相关操作均不执行，因此该函数没有图像输出。由于该函数不执行绘制相关操作，仅包含CPU计算，可以在没有OpenGL环境的情况下正常运行。该函数执行人脸跟踪操作后，结果产生的人脸信息通过 `fuGetFaceInfo` 接口进行获取（TODO：加链接）。

```
int fuTrackFace(int in_format,void* in_ptr,int w,int h);
```

参数

in_format [in]：输入的图像数据格式。

in_ptr [in]：内存指针，指向输入的图像数据。

w [in]：输入的图像宽度。

h [in]：输入的图像高度。

返回值

在图像中成功跟踪到的人脸数量。

备注

该函数仅支持部分输入图像格式，包括 RGBA_BUFFER，BGRA_BUFFER，NV12 BUFFER，NV21 BUFFER。

该接口不需要绘制环境，但仍需要在SDK主线程中运行。（TODO：确认这一点）

销毁

fuDestroyItem 函数

销毁一个指定道具。

```
void fuDestroyItem(int item);
```

参数

item [in]：道具标识符，该标识符应为调用 `fuCreateItemFromPackage` 函数的返回值，并且道具没有被销毁。

备注

该函数调用后，会即刻释放道具标识符，道具占用的内存无法瞬时释放，需要等 SDK 后续执行主处理接口时通过 GC 机制回收。

fuDestroyAllItems 函数

销毁系统加载的所有道具，并且会释放系统运行时占用的所有资源。

```
void fuDestroyAllItems();
```

备注

该函数会即刻释放系统所占用的资源。但不会破坏 `fuSetup` 的系统初始化信息，应用临时挂起到后台时可以调用该函数释放资源，再次激活时无需重新初始化系统。

fuOnDeviceLost 函数

特殊函数，当 OpenGL context 被外部释放/破坏时调用，用于重置系统的 GL 状态。

```
void fuOnDeviceLost();
```

备注

该函数仅在无法在原 OpenGL context 内正确清理资源的情况下调用。调用该函数时，会尝试进行资源清理和回收，所有系统占用的内存资源会被释放，但由于 context 发生变化，OpenGL 资源相关的内存可能会发生泄露。

功能接口 - 系统

fuOnCameraChange 函数

在相机数据来源发生切换时调用（例如手机前/后置摄像头切换），用于重置人脸跟踪状态。

```
void fuOnCameraChange();
```

备注

在其他人脸信息发生残留的情景下，也可以调用该函数来清除人脸信息残留。

fuIsTracking 函数

获取当前人脸跟踪状态，返回正在跟踪的人脸数量。

```
int fuIsTracking();
```

返回值

正在跟踪的人脸数量。

备注

正在跟踪的人脸数量会受到 `fuSetMaxFaces` 函数的影响，不会超过该函数设定的最大值。（TODO：加链接）

fuSetDefaultOrientation 函数

设置默认的人脸朝向。正确设置默认的人脸朝向可以显著提升人脸首次识别的速度。

```
void fuSetDefaultOrientation(int rmode);
```

参数

rmode [in]：要设置的人脸朝向，取值范围为 0-3，分别对应人脸相对于图像数据旋转0度、90度、180度、270度。

备注

一般来说，iOS的原生相机数据是竖屏的，不需要进行该设置。Android 平台的原生相机数据为横屏，需要进行该设置加速首次识别。根据经验，Android 前置摄像头一般设置参数 1，后置摄像头一般设置参数 3。部分手机存在例外，自动计算的代码可以参考 fuLiveDemo。

fuSetMaxFaces 函数

设置系统跟踪的最大人脸数。默认值为1，该值增大会降低人脸跟踪模块的性能，推荐在所有可以设计为单人脸的情况下设置为1。

```
int fuSetMaxFaces(int n);
```

参数

n [in]：要设置的最大人脸数。

返回值

设置之前的系统最大人脸数。

fuGetFaceInfo 函数

在主接口执行过人脸跟踪操作后，通过该接口获取人脸跟踪的结果信息。获取信息需要证书提供相关权限，目前人脸信息权限包括以下级别：默认、Landmark、Avatar。

```
int fuGetFaceInfo(int face_id, char* name, float* pret, int num);
```

参数

face_id [in]：人脸编号，表示识别到的第 x 张人脸，从0开始。

name [in]：要获取信息的名称。

pret [out]：返回数据容器，需要在函数调用前分配好内存空间。

num [in]：返回数据容器的长度，以 sizeof(float) 为单位。

返回值

返回 1 代表获取成功，信息通过 pret 返回。返回 0 代表获取失败，具体失败信息会打印在平台控制台。如果返回值为 0 且无控制台打印，说明所要求的人脸信息当前不可用。

备注

所有支持获取的信息、含义、权限要求如下：

信息名称	长度	含义	权限
face_rect	4	人脸矩形框，图像分辨率坐标，数据为 (x_min, y_min, x_max, y_max)	默认
rotation_mode	1	识别人脸相对于设备图像的旋转朝向，取值范围 0-3，分别代表旋转0度、90度、180度、270度	默认
failure_rate	1	人脸跟踪的失败率，表示人脸跟踪的质量。取值范围为 0-2，取值越低代表人脸跟踪的质量越高	默认
is_calibrating	1	表示是否SDK正在进行主动表情校准，取值为 0 或 1。	默认
focal_length	1	SDK当前三维人脸跟踪所采用的焦距数值	默认
landmarks	75x2	人脸 75 个特征点，图像分辨率坐标	Landmark
rotation	4	人脸三维旋转，数据为旋转四元数*	Landmark
translation	3	人脸三维平移，数据为 (x, y, z)	Landmark
eye_rotation	4	眼球旋转，数据为旋转四元数*	Landmark
expression	46	人脸表情系数，表情系数含义可以参考《Expression Guide》	Avatar

*注：旋转四元数转换为欧拉角可以参考 该网页。

fuGetFaceIdentifier 函数

获取正在跟踪人脸的标识符，用于在SDK外部对多人情况下的不同人脸进行区别。

```
int fuGetFaceIdentifier(int face_id);
```

参数

face_id [in]：人脸编号，表示识别到的第 x 张人脸，从0开始。

返回值

所要求的人脸标识符。

备注

跟踪失败会改变标识符，包括快速的重跟踪。

fuGetVersion 函数

返回SDK版本。

```
const char* fuGetVersion();
```

返回值

一个常量字符串指针，版本号表示如下： “主版本号_子版本号-版本校检值”

fuGetSystemError 函数

返回系统错误，该类错误一般为系统机制出现严重问题，导致系统关闭，因此需要重视。

```
const int fuGetSystemError();
```

返回值

系统错误代码。

备注

复数错误存在的情况下，会以位运算形式编码在一个代码中。如果返回代码不对应以下列表中任何一个单一代码，则说明存在复数错误，此时可以通过 `fuGetSystemErrorString` 函数解析最重要的错误信息。

系统错误代码及其含义如下：

错误代码	错误信息
1	随机种子生成失败
2	机构证书解析失败
3	鉴权服务器连接失败
4	加密连接配置失败
5	客户证书解析失败
	客户密钥解析失败
7	建立加密连接失败
8	设置鉴权服务器地址失败
9	加密连接握手失败
10	加密连接验证失败
11	请求发送失败
12	响应接收失败
13	异常鉴权响应
14	证书权限信息不完整
15	鉴权功能未初始化
16	创建鉴权线程失败
17	鉴权数据被拒绝
18	无鉴权数据
19	异常鉴权数据
20	证书过期
21	无效证书
22	系统数据解析失败
0x100	加载了非正式道具包（debug版道具）
0x200	运行平台被证书禁止

fuGetSystemErrorString 函数

解析系统错误代码，并返回可读信息。

```
const char* fuGetSystemErrorString(int code);
```

参数

code [in]：系统错误代码，一般为 `fuGetSystemError` 所返回的代码。

返回值

一个常量字符串，解释了当前错误的含义。

备注

当复数错误存在的情况下，该函数会返回当前最为重要的错误信息。

fuCheckDebugItem 函数

检查一个道具包是否为非正式道具包（debug版道具）。

```
const int fuCheckDebugItem(void* data,int sz);
```

参数

data [in]：内存指针，指向所加载道具包的内容。道具包通常以 *.bundle 结尾。

sz [in]：道具包内容的数据长度，以字节为单位。

返回值

返回值 0 代表该道具为正式道具，返回值 1 代表该道具为非正式道具（debug版道具），返回值 -1 代表该道具数据异常。

备注

如果系统加载过非正式版道具，会导致系统进入倒计时，并在倒计时结束时关闭。如果系统提示 “debug item used”，或系统在运行1分钟后停止，则需要利用该函数检查所有加载过的道具，如果有非正式道具需要进行正确的道具签名。

道具签名流程请联系技术支持。

功能接口 - 效果

fuSetExpressionCalibration 函数

设置人脸表情校准功能。该功能的目的是使表情识别模块可以更加适应不同人的人脸特征，以实现更加准确可控的表情跟踪效果。

该功能分为两种模式，主动校准 和 被动校准。

- 主动校准：该种模式下系统会进行快速集中的表情校准，一般为初次识别到人脸之后的2-3秒钟。在该段时间内，需要用户尽量保持无表情状态，该过程结束后再开始使用。该过程的开始和结束可以通过 `fuGetFaceInfo` 接口获取参数 `is_calibrating`。

- 被动校准：该种模式下会在整个用户使用过程中逐渐进行表情校准，用户对该过程没有明显感觉。该种校准的强度比主动校准较弱。

默认状态为开启被动校准。

```
void fuSetExpressionCalibration(int mode);
```

参数

mode [in]：表情校准模式，0为关闭表情校准，1为主动校准，2为被动校准。

备注

当利用主处理接口处理静态图片时，由于需要针对同一数据重复调用，需要将表情校准功能关闭。

fuLoadAnimModel 函数

加载表情动画模型，并启用表情优化功能。

表情优化功能可以使实时跟踪后得到的表情更加自然生动，但会引入一定表情延迟。

```
int fuLoadAnimModel(void* dat, int dat_sz);
```

参数

dat [in]：内存指针，指向动画模型文件内容。该文件随SDK分发，文件名为 anim_model.bundle。

dat_sz [in]：动画模型文件长度，以字节为单位。

返回值

返回值 1 代表加载成功，并启用表情优化功能。返回值 0 代表失败。

fuLoadExtendedARData 函数

加载高精度人脸重建数据。使用高精度人脸重建功能所必须的数据加载。

```
/**
\brief Load extended AR data, which is required for high quality AR items
\param data - the pointer to the extended AR data
\param sz - the data size, we use plain int to avoid cross-language compilation issues
\return zero for failure, non-zero for success
*/
int fuLoadExtendedARData(void* data,int sz);
```

参数

dat [in]：内存指针，指向高精度人脸重建数据文件内容。该文件随SDK分发，文件名为 ardata_ex.bundle。

dat_sz [in]：高精度人脸重建数据文件长度，以字节为单位。

返回值

返回值 1 代表成功加载。返回值 0 为失败。

fuSetStrictTracking 函数

启用更加严格的跟踪质量检测。

该功能启用后，当面部重要五官出现被遮挡、出框等情况，以及跟踪质量较差时，会判断为跟踪失败，避免系统在跟踪质量较低时出现异常跟踪数据。

```
void fuSetStrictTracking(int mode);
```

参数

mode [in]：0为禁用，1为启用，默认为禁用状态。

fuSetFocalLengthScale 函数

修改系统焦距（效果等价于focal length, 或FOV），影响三维跟踪、AR效果的透视效果。

参数为一个比例系数，焦距变大会带来更小的透视畸变。

```
/**
\brief Scale the rendering perspectivity (focal length, or FOV)
    Larger scale means less projection distortion
    This scale should better be tuned offline, and set it only once in runtime
\param scale - default is 1.f, keep perspectivity invariant
    <= 0.f would be treated as illegal input and discard
*/
void fuSetFocalLengthScale(float scale);
```

参数

scale [in]：焦距调整的比例系数，1.0为默认值。建议取值范围为 0.1 ~ 2.0。

备注

系数小于等于0为无效输入。

输入输出格式列表

RGBA 数组

RGBA 格式的图像内存数组。

数据格式标识符

FU_FORMAT_RGBA_BUFFER

数据内容

连续内存空间，长度为 `w*h*4`。数组元素为 `int`，按 RGBA 方式表示颜色信息。

输入输出支持

可输入 / 可输出

备注

由于平台上的内存对齐要求，图像内存空间的实际宽度可能不等于图像的语义宽度。在主运行接口传入图像宽度时，应传入内存实际宽度。

BGRA 数组

BGRA 格式的图像内存数组。

数据格式标识符

FU_FORMAT_BGRA_BUFFER

数据内容

连续内存空间，长度为 `w*h*4`。数组元素为 `int`，按 BGRA 方式表示颜色信息。

输入输出支持

可输入 / 可输出

备注

由于平台上的内存对齐要求，图像内存空间的实际宽度可能不等于图像的语义宽度。在主运行接口传入图像宽度时，应传入内存实际宽度。

该格式为原生 iOS 的相机数据格式之一。

RGBA 纹理

RGBA 格式的 OpenGL 纹理。

数据格式标识符

FU_FORMAT_RGBA_TEXTURE

数据内容

一个 `GLuint`，表示 OpenGL 纹理 ID。

输入输出支持

可输入 / 可输出

RGBA OES 纹理

RGBA 格式的 OpenGL external OES 纹理。

数据格式标识符

FU_FORMAT_RGBA_TEXTURE_EXTERNAL_OES

数据内容

一个 `GLuint`，表示 OpenGL external OES 纹理 ID。

输入输出支持

仅输入

备注

该格式为原生安卓相机数据格式之一。

NV21 数组

NV21 格式的图像内存数组。

数据格式标识符

FU_FORMAT_NV21_BUFFER

数据内容

连续内存，前一段是 Y 数据，长度为 `w*h`，后一段是 UV 数据，长度为 `2*((w+1)>>1)`（分辨率是Y的一半，但包含UV两个通道）。两段数据在内存中连续存放。

输入输出支持

可输入 / 可输出

备注

该格式要求UV数据交错存放（如：UVUVUVUV），如UV数据分开存放（UUUUVVVV），请用I420数组格式（TODO：加链接）。

该格式为原生安卓相机数据格式之一。

NV12 数组

NV12 格式的图像内存数组。

数据格式标识符

FU_FORMAT_NV12_BUFFER

数据内容

结构体 `TNV12Buffer`，其定义如下。

```
typedef struct{
    void* p_Y;
    void* p_CbCr;
    int stride_Y;
    int stride_CbCr;
}TNV12Buffer;
```

参数

p_Y：指向 Y 数据的指针。

p_CbCr：指向 UV 数据的指针。

stride_Y：Y 数据每行的字节长度。

stride_CbCr：UV 数据每行的字节长度。

输入输出支持

可输入 / 可输出

备注

该格式与 NV21 数组格式非常类似，只是 UV 数据中 U 和 V 的交错排布相反。不过该格式支持 Y 数据和 UV 数据分别存放，不再要求数据整体连续。

该格式为原生iOS相机数据格式之一。

I420 数组

I420 格式的图像内存数组。

数据格式标识符

FU_FORMAT_I420_BUFFER

数据内容

连续内存，第一段是 Y 数据，长度为 `w*h`，第二段是 U 数据，长度为 `((w+1)>>1)`，第三段是 V 数据，长度为 `((w+1)>>1)`（后两个通道分辨率是Y的一半）。三段数据在内存中连续存放。

输入输出支持

可输入 / 可输出

备注

该格式和 NV21 数组基本一致，区别在于 U 和 V 数据分别连续存放。

iOS 双输入

针对iOS原生相机数据的双输入格式。双输入分别指GPU数据输入——OpenGL 纹理，以及CPU内存数据输入——BGRA数组或NV12数组。

相比仅提供内存数组或纹理的单数据输入，该输入模式可以减少一次CPU-GPU间数据传输，可以轻微优化性能（iphone平台上约为2ms）。

数据格式标识符

FU_FORMAT_INTERNAL_IOS_DUAL_INPUT

数据内容

结构体 `TIOSDualInput`，其定义如下。

```
typedef struct{
    int format;
    void* p_BGRA;
    int stride_BGRA;
    void* p_Y;
    void* p_CbCr;
    int stride_Y;
    int stride_CbCr;
    int tex_handle;
}TIOSDualInput;
```

参数

format：指示内存数组的数据格式，仅支持 BGRA 和 NV12 两种，分别对应常量 `FU_IDM_FORMAT_BGRA` 和 `FU_IDM_FORMAT_NV12`。

p_BGRA：内存数据为 BGRA 格式时，指向内存数据的指针。

stride_BGRA：内存数据为 BGRA 格式时，每行图像数据的字节宽度。

p_Y：内存数据为 NV12 格式时，指向 Y 通道内存数据的指针。

p_CbCr：内存数据为 NV12 格式时，指向 UV 通道内存数据的指针。

stride_Y：内存数据为 NV12 格式时，Y 通道内存数据的字节宽度。

stride_CbCr：内存数据为 NV12 格式时，UV 通道内存数据的字节宽度。

tex_handle：GPU数据的输入，默认为 RGBA 格式的 OpenGL 纹理 ID。

输入输出支持

仅输入

备注

由于在iOS平台上性能优化不大，因此推荐仅在环境中已有 GPU 数据希望复用，或者有意自定义 GPU 数据输入的情况下使用该接口。

Android 双输入

针对 Android 原生相机数据的双输入格式。双输入分别指GPU数据输入——RGBA / NV21 / I420 格式的 OpenGL 纹理，以及CPU内存数据输入—— NV21/ RGBA / I420 格式的图像内存数组。

相比仅提供内存数组或纹理的单数据输入，该输入模式可以减少一次 CPU-GPU 间数据传输，在 Android 平台上可以显著优化性能，因此**推荐尽可能使用该接口。**

数据格式标识符

FU_FORMAT_ANDROID_DUAL_MODE

数据内容

结构体 `TAndroidDualMode`，其定义如下。

```
typedef struct{
    void* p_NV21;
    int tex;
    int flags;
}TAndroidDualMode;
```

参数

p_NV21：指向内存图像数据的指针。

tex：OpenGL 纹理 ID。

flags：扩展功能标识符，所有支持的标识符及其功能如下。多个标识符通过"或"运算符连接。

扩展功能标识符	含义
FU_ADM_FLAG_EXTERNAL_OES_TEXTURE	传入的纹理为OpenGL external OES 纹理
FU_ADM_FLAG_ENABLE_READBACK	开启后将处理结果写回覆盖到传入的内存图像数据
FU_ADM_FLAG_NV21_TEXTURE	传入的纹理为 NV21 数据格式
FU_ADM_FLAG_I420_TEXTURE	传入的纹理为 I420 数据格式
FU_ADM_FLAG_I420_BUFFER	传入的内存图像数据为 I420 数据格式
FU_ADM_FALG_RGBA_BUFFER	传入的内存图像数据为 RGBA 数据格式

输入输出支持

仅输入

当前 FBO

指调用主处理接口时当前绑定的 OpenGL FBO。主处理接口可以直接将处理结果绘制到该 FBO 上。

数据格式标识符

FU_FORMAT_GL_CURRENT_FRAMEBUFFER

数据内容

无，数据指针直接传 NULL。

输入输出支持

仅输出

备注

需要在传入 FBO 前完成 FBO 的创建，包括颜色纹理的绑定，该 FBO 需通过 OpenGL 完备性检查。

如果有 3D 绘制内容，需要该 FBO 具备深度缓冲。

指定 FBO

可以将外部已经准备好的 OpenGL FBO 传入，不一定在调用主处理接口时作为当前绑定的 FBO。 主处理接口可以直接将处理结果绘制到该 FBO 上。

数据格式标识符

FU_FORMAT_GL_SPECIFIED_FRAMEBUFFER

数据内容

结构体 `TSPECFBO`，其定义如下。

```
typedef struct{
    int fbo;
    int tex;
}TSPECFBO;
```

参数

fbo：指定的 FBO ID。

tex：该 FBO 上绑定的颜色纹理 ID。

输入输出支持

仅输出

备注

需要在传入 FBO 前完成 FBO 的创建，包括颜色纹理的绑定，该 FBO 需通过 OpenGL 完备性检查。

如果有 3D 绘制内容，需要传入 FBO 具备深度缓冲。

Avatar 驱动信息

特殊的输入数据，不是图像数据，而是人脸驱动信息，用于驱动avatar模型。人脸驱动信息可以在主处理接口执行后获取，也可以外部输入，比如avatar动画录制的信息，或者用户交互产生的信息等。

数据格式标识符

FU_FORMAT_AVATAR_INFO

数据内容

结构体 `TAvatarInfo`，其定义如下。

```
typedef struct{
    float* p_translation;
    float* p_rotation;
    float* p_expression;
    float* rotation_mode;
    float* pupil_pos;
    int is_valid;
}TAvatarInfo;
```

参数

p_translation：指向内存数据的指针，数据为3个float，表示人脸在相机空间的平移。其中，x/y 的单位为图像分辨率，z 是相机空间中人脸的深度。

p_rotation：指向内存数据的指针，数据为4个float，表示人头的三位旋转。旋转表示方式为四元数，需要经过换算转化成欧拉角旋转。

p_expression：指向内存数据的指针，数据为46个float，表示人脸的表情系数。表情系数的含义请参考《Expression Guide》。

rotation_mode：一个int，取值范围为 0-3，表示人脸相对于图像数据的旋转，分别代表旋转0度、90度、180度、270度。

pupil_pos：指向内存数据的指针，数据为2个float，表示瞳孔的参数坐标。该坐标本身不具有语义，一般直接从跟踪结果中获取。

is_valid：一个int，表示该 avatar 信息是否有效。该值为0的情况下系统不会处理对应 avatar 信息。

输入输出支持

仅输入

备注

该输入模式仅能配合 avatar 道具使用，加载人脸 AR 类道具会导致异常。

该输入模式会简化对传入图像数据的处理，在 avatar 应用情境下性能较高。此外，对于 avatar 的控制更加灵活，可以允许用户自由操控 avatar，如拖动 avatar 转头、触发特定表情等。

P2A 相关接口

```
/**
\brief Bind items to an avatar, already bound items won't be unbound
\param avatar_item is the avatar item handle
\param p_items points to a list of item handles to be bound to the avatar
\param n_items is the number of item handles in p_items
\param p_contracts points to a list of contract handles for authorizing items
\param n_contracts is the number of contract handles in p_contracts
\return the number of items newly bound to the avatar
*/
FUNAMA_API int fuAvatarBindItems(int avatar_item, int* p_items,int n_items, int* p_contracts,int
n_contracts);
/**
```

```
\brief Unbind items from an avatar
\param avatar_item is the avatar item handle
\param p_items points to a list of item handles to be unbound from the avatar
\param n_items is the number of item handles in p_items
\return the number of items unbound from the avatar
*/
FUNAMA_API int fuAvatarUnbindItems(int avatar_item, int* p_items,int n_items);

//
FUNAMA_API int fuBindItems(int item_src, int* p_items,int n_items);
FUNAMA_API int fuUnbindAllItems(int item_src);

/**
\brief Get the expire time of an avatar item
\param data - the pointer to the item data, the item must be type of "p2a_avatar"
\param sz - the data size, we use plain int to avoid cross-language compilation issues
\return x - this avatar will expire in 'x' days
        -1 - invalid expire time, detail will be print on console
*/
FUNAMA_API const int fuGetAvatarExpireTime(void* data,int sz);
```

废弃接口

```
/**
\brief Set the quality-performance tradeoff.
\param quality is the new quality value.
       It's a floating point number between 0 and 1.
       Use 0 for maximum performance and 1 for maximum quality.
       The default quality is 1 (maximum quality).
*/
FUNAMA_API void fuSetQualityTradeoff(float quality);

/**
\brief Turn off the camera
*/
FUNAMA_API void fuTurnOffCamera();

/**
\brief Generalized interface for rendering a list of items.
    This function needs a GLES 2.0+ context.
\param out_format is the output format
\param out_ptr receives the rendering result, which is either a GLuint texture handle or a
memory buffer
    Note that in the texture cases, we will overwrite *out_ptr with a texture we generate.
\param in_format is the input format
\param in_ptr points to the input image, which is either a GLuint texture handle or a memory
buffer
\param w specifies the image width
\param h specifies the image height
\param frameid specifies the current frame id.
    To get animated effects, please increase frame_id by 1 whenever you call this.
\param p_items points to the list of items
```

```
\param n_items is the number of items
\param p_masks indicates a list of masks for each item, bitwisely work on certain face
\return a GLuint texture handle containing the rendering result if out_format isn't
FU_FORMAT_GL_CURRENT_FRAMEBUFFER
*/
FUNAMA_API int fuRenderItemsMasked(
    int out_format,void* out_ptr,
    int in_format,void* in_ptr,
    int w,int h,int frame_id, int* p_items,int n_items, int* p_masks);

/**
\brief Get the camera image size
\param pret points to two integers, which receive the size
*/
FUNAMA_API void fuGetCameraImageSize(int* pret);
```