FaceUnity SDK使用说明

June 23th, 2016

文档版本

|  |  |
| --- | --- |
| 时间 | 版本 |
| 2016年4月25日 | V1.0 |
| 2016年5月13日 | V1.1 |
| 2016年6月24日 | V2.0 |
| 2016年6月24日 | V2.1 |
| 2016年6月24日 | V2.2 |
| 2016年11月15日 | V3.0 |
| 2016年12月30日 | V3.1 |

Contents

[1. FaceUnity SDK简介 1](#_Toc454541694)

[2. 使用DDE系列函数 2](#_Toc454541695)

[2.1 相关函数—按照调用顺序 2](#_Toc454541696)

[2.2 初始化 4](#_Toc454541697)

[2.2 运行识别面部和追踪表情 4](#_Toc454541698)

[2.3获得表情追踪结果 5](#_Toc454541699)

[3.使用easydde系列函数 7](#_Toc454541700)

[3.1 相关函数—按照调用顺序 7](#_Toc454541701)

[3.1 初始化 8](#_Toc454541702)

[3.2easydde\_run 识别面部 8](#_Toc454541703)

[3.3 easydde\_get\_data获得结果 8](#_Toc454541704)

[4. 表情参数结果数据 9](#_Toc454541705)

[4.1 结果参数expression 9](#_Toc454541706)

[4.2 头部旋转方向rotation 9](#_Toc454541707)

[4.3 数据可用性face\_confirmation\_failure\_stress 9](#_Toc454541708)

[4.4眼球方向pupil\_pos 10](#_Toc454541709)

[4.5定位点坐标landmarks 10](#_Toc454541710)

[联系 11](#_Toc454541711)

# 1.FaceUnitySDK简介

使用FaceUnity SDK有两种调用方式： dde系列函数，以及easydde系列函数。dde系列函数使用起来相对复杂，需要手动调用face detector，可以对多张脸进行识别跟踪；easydde系列函数使用方便，直接就能获得表情参数。

对于需要在原画面的人脸上叠加绘制其他内容的应用（比如贴纸，口红，描眉之类），建议使用ddear系列API获取三维模型而不是直接使用二维特征点。

# 2.使用DDE系列函数

## 2.1 相关函数—按照调用顺序

**#1 int dde\_setup(void\* p, void\* authdata,int authdata\_sz);**

p 指向数据文件data.bin的内容

authdata 指向用户签名指纹数据内容，存在authpack.h中，用于认证身份

authdata\_sz authdata的内容大小

用于初始化，导入SDK所需数据data.bin，引用authpack.h。这两个文件会与SDK一起提供。

在调用任何其他DLL函数的期间指针p指向的内容不能释放。

**#2 size\_t dde\_context\_size();**

return dde context的大小

用于初始化，配合malloc使用以创建dde的context。

**#3 void\* dde\_facedet\_create();**

return face detector的context

用于初始化，为face detector创建一个context并将其返回。

**#4 int dde\_facedet\_run(void\* detector,void\* img,int w,int h,int\* ret,int max\_faces,int rotation\_mode);**

detector face detector的context

img 用于识别的图像

w 图像宽度

h 图像高度

ret 保存识别到的每张脸的位置(x,y,w,h)即：x坐标、y坐标、宽、高

max\_faces 最多可以识别几张脸

rotation\_mode 图像的旋转方向：

正向为0，顺时针旋转90°、180°、270°分别为1、2、3

return 识别到几张脸

用于面部识别，返回识别到几张脸，并将每张脸在图像中的位置（每张脸四个int值）保存在ret中。

**#5 int hldde\_first(TWorkArea\* context,void\* img,int pitch,int w,int h, float\* bb);**

context dde的context

img 用于识别的图像

pitch 图像相邻两行之间所差字节数（例如RGBA图像为w\*4）

w 图像宽度

h 图像高度

bb 需要跟踪表情的脸部在整张图像中的位置：左上角坐标和右下角坐标

return 表情跟踪是否成功，成功时返回正整数，失败时返回非正数

用于表情追踪，需要在进行过面部识别并成功之后调用，只识别面部朝向正上方的情况。如果返回失败则需要重新进行面部识别。

**#6 int hldde\_first\_ex(TWorkArea\* context, void\* img,int pitch,int w,int h, float\* bb,int rotation\_mode,const float\* pfl);**

context dde的context

img 用于识别的图像

pitch 图像相邻两行之间所差字节数（例如RGBA图像为w\*4）

w 图像宽度

h 图像高度

bb 需要跟踪表情的脸部在整张图像中的位置：左上角坐标和右下角坐标

rotation\_mode 图像的旋转方向：

正向为0，顺时针旋转90°、180°、270°分别为1、2、3

pfl 摄像头的像素焦距，如果搞不清楚，建议值为

float pfl[1]={536.5f/480.f\*(float)h};

return 表情跟踪是否成功，成功时返回正整数，失败时返回非正数

用于表情追踪，与#5相似，但可以识别面部朝向上下左右四个方向的情况，利用rotation\_mode控制，同样需要在进行过面部识别并成功之后调用。如果返回失败需要重新进行面部识别。

**#7 int hldde\_next(TWorkArea\* context, void\* img,int pitch,int w,int h);**

context dde的context

img 用于识别的图像

pitch 图像相邻两行之间所差字节数（例如RGBA图像为w\*4）

w 图像宽度

h 图像高度

return 表情跟踪是否成功，成功时返回正整数，失败时返回非正数

用于表情追踪，需要在hldde\_first或hldde\_first\_ex成功的情况下调用，并且只要hldde\_next返回成功即可不断调用。如果失败需要重新进行面部识别。

**#8 float\* dde\_get(TWorkArea\* context,char\* name,int\* pdim);**

context dde的context

name 选择要获取的参数名字

pdim 返回参数的个数

return 表情识别的结果数据

用于获取表情参数，需要在表情追踪成功的情况下调用，即hldde\_first或hldde\_first\_ex或hldde\_next返回正数的情况。

## 2.2 初始化

调用初始化相关参数，下面的伪代码是例子：

**#include "authpack.h"**

**float**\* g\_global\_tables=NULL;

TWorkArea\* g\_dde\_context=NULL;

**void**\* fd\_ctx=NULL;

**void faceinit**(){

p = (**float**\*)ddev2.bin中数据;

dde\_setup((void\*)p,g\_auth\_package,sizeof(g\_auth\_package));

//创建dde的context和face detector的context

g\_dde\_context=(TWorkArea\*)**malloc**(dde\_context\_size());

fd\_ctx=dde\_facedet\_create();

}

在初始化阶段读取数据"ddev2.bin"全部内容，传给init\_global\_tables作为参数。还需要初始化detector和tracker，并保存它们的context。

## 2.2运行识别面部和追踪表情

仿照下面的例子进行面部识别和表情追踪：

bool face\_run(u32\* img\_raw,int w,int h){

int x, y, width, height;

if(g\_state==STATE\_NONE){

int n\_faces=dde\_facedet\_run(fd\_ctx,img\_raw,w,h,g\_face\_bb,1,1);**//(1)**

if(n\_faces != 0){

g\_state=STATE\_GOT\_FACERECT;

} else{

g\_state=STATE\_NONE;

}

}

if(g\_state==STATE\_GOT\_FACERECT){

int is\_valid;

int i;

g\_face\_bbf[0]=(float)g\_face\_bb[0];

g\_face\_bbf[1]=(float)g\_face\_bb[1];

g\_face\_bbf[2]=(float)g\_face\_bb[0]+(float)g\_face\_bb[2];

g\_face\_bbf[3]=(float)g\_face\_bb[1]+(float)g\_face\_bb[3];

is\_valid = hldde\_first\_ex(g\_dde\_context, img\_raw, w\*4, w, h, g\_face\_bbf, 1, pfl); **//(2)**

if(is\_valid != 0 && g\_face\_bbf[2] < w && g\_face\_bbf[3] < h){

g\_state=STATE\_GOT\_DDE;

}else {

g\_state=STATE\_NONE;

}

}else if(g\_state==STATE\_GOT\_DDE){

int is\_valid=hldde\_next(g\_dde\_context, img\_raw,w\*4,w,h);**//(3)**

if(is\_valid > 0){

g\_state=STATE\_GOT\_DDE;

}else{

g\_state=STATE\_NONE;

}

}

return g\_state==STATE\_GOT\_DDE;**//(4/5)**

}

步骤（识别每一帧图像时要调用此函数）：

**(1)**首先进行面部识别，得到面部在图像中的位置(g\_face\_bb)。

**(2)**面部识别成功后，可以进行第一次表情识别，如果表情识别成功，则返回成功(4)，否则返回失败(5)。

**(3)**第(2)步成功后，再进行表情追踪时可以直接进行hldde\_next，如果成功则返回成功(4)，否则退回失败(5)。

**(4)**表情识别成功，接下来可以使用dde\_get函数获得表情数据，识别下一帧时从(3)开始。

**(5)**表情识别失败，识别下一帧要退回第(1)步。

## 2.3获得表情追踪结果

仿照下面的例子获得表情追踪结果。

float\* face\_get(char\* name,int\* pdim){

return dde\_get(g\_dde\_context,name,pdim);

}

在上一步面部识别和表情追踪成功时，即可调用dde\_get函数获取表情追踪的结果，使用示例：expression\_data = face\_get((char\*)("expression"), &psize);

这其中"expression"即是要获得的数据类型，psize会得到数据格式大小，返回的expression\_data是float指针，dde\_get会直接分配好空间，也就是说，调用这个函数之前expression\_data=NULL即可，得到数据后也千万不要去free。

根据参数name的设定，可以返回不同类型的表情参数，参数的获取将在第四章节介绍。

# 3.使用easydde系列函数

## 3.1 相关函数—按照调用顺序

**#1 int easydde\_run(void\* img,int w,int h,int flags);**

Img 用于识别的图像

w 图像宽度

h 图像高度

flags 可选标志如下：

FLAG\_IMAGE\_FORMAT\_RGBA –0

FLAG\_IMAGE\_FORMAT\_NV21– 1

return 表情跟踪是否成功，成功时返回正整数，失败时返回非正数

函数包含了面部识别与表情追踪，使用时调用这一个函数即可。

flags标志一般设为FLAG\_IMAGE\_FORMAT\_RGBA；如果在Android项目中使用，可以在`Camera.Parameters`中调用`setPreviewFormat(ImageFormat.NV21)`，从`onPreviewFrame`中获得摄像头图像，并将标志设为FLAG\_IMAGE\_FORMAT\_NV21。

对于FLAG\_IMAGE\_FORMAT\_NV21模式，SDK内部其实只会访问图像最开头的灰度部分，所以这个模式也可以应用在YUV I420格式或者普通的灰度图上。

**#2 int easydde\_get\_data(float\* ret,int szret, char\* s);**

ret 保存得到的表情参数结果，需要先分配好足够的空间

szret 需要获取的表情参数结果值的数量

s 选择要获取的参数名字

return face detector的context

用于获取表情参数，需要在easydde\_run返回成功的情况下调用。

**#3 void easydde\_reset();**

用于强行停止表情识别，其他情况不需调用。

**#4 TWorkArea\* easydde\_get\_context();**

用于获取easydde系列API内部使用的context。混用easydde和其他API时会用到。

## 3.1 初始化

easydde的初始化只要直接读取ddev2.bin的数据即可：

**#include "authpack.h"**

**float**\* g\_global\_tables=NULL;

**void faceinit**(){

p = (**float**\*)(ddev2.bin中数据);

dde\_setup((void\*)p,g\_auth\_package,sizeof(g\_auth\_package));

}

## 3.2 easydde\_run 识别面部

直接调用easydde\_run(image, w, h, 0)即可，返回值1为可用，0和-1为不可用。

需要说明的，easydde\_run每运行一次，会顺时针转90度，也就是说，如果不知道人脸在图片中方向的话，只要将这个函数调用4次即可，并且可以通过调用easydde\_get\_rotation\_mode()函数来获得图片的方向。

第一次调用、图片大小内容发生改变或是结果无效时，需要调用easydde\_reset()重置一下。

## 3.3 easydde\_get\_data获得结果

使用方法：easydde\_get\_data(expression\_data, 46, "expression");

与上一章中方法不同的是，这里介绍数据的float指针要先分配好空间，并且也要把数据大小传个easydde\_get\_data。

参数name的设定与上一章中方法相同，将在第四章节介绍。

# 4.表情参数结果数据

使用dde\_get和easydde\_get\_data函数获得表情结果，其中参数name的使用在ddeface.h中有描述，常用的数据可以如下使用：

easydde\_get\_data(expression\_data, 46,"expression");

easydde\_get\_data(rotation\_data, 4, "rotation");

easydde\_get\_data(failure\_data, 2,"face\_confirmation\_failure\_stress");

easydde\_get\_data(pupil\_pos, 2, "pupil\_pos");

easydde\_get\_data(landmarks, 150, "landmarks");

easydde\_get\_data(focal\_length, 1, "focal\_length");

如果不想把数组大小写死，可以这样做：

int n\_expressions=easydde\_get\_data(NULL, 0, "expression");

float\* expression=(float\*)malloc(sizeof(float)\*n\_expressions);

easydde\_get\_data(expression, n\_expressions, "expression");

## 4.1 结果参数expression

大小：46个，内容参见“Expression Guide.pdf”。

这就是用来表示面部表情的46个参数。其中第5-8和11-14八个描述眼珠方向的参数值是空的，需要通过获取眼球位置(pupil\_pos)来得到。

## 4.2 头部旋转方向rotation

大小：4个，即四位数（顺序：x, y, z, w），表示头部的旋转方向。

## 4.3 数据可用性face\_confirmation\_failure\_stress

大小：1个，表示得到的数据是否可用。

failure\_data[0] > 2时，忽视结果；

failure\_data[0] > 10时，reset。

easydde系列函数在failure\_data值大于10时会自动重置。

## 4.4眼球方向pupil\_pos

大小：2个，两个向量分别为x轴方向和y轴方向。

可直接将结果放进参数中缺少的部分，如：

//set pupil position

expression\_data[6] = expression\_data[7] = pupil\_pos[0];

expression\_data[10] = expression\_data[11] = -pupil\_pos[0];

expression\_data[12] = expression\_data[13] = pupil\_pos[1];

expression\_data[4] = expression\_data[5] = -pupil\_pos[1];

## 4.5定位点坐标landmarks

大小：150。75个点，每个点2个float值。

# 5.使用ddear系列函数

如前所述，ddear系列函数主要用于往原画面的人脸上叠加其他内容。这一系列API会提供一个3Dmesh，根据tracker输出的刚体变换和投影变换绘制后即可完全贴合视频中的人脸。需要叠加绘制的内容可以作为贴图贴在这个mesh上，也可以在mesh上选取特定顶点定位。

**#1 int ddear\_get\_static\_data(float\*\* ppuv,short\*\* ppebo,int\* pn\_vertices,int\* pn\_triangles);**

|  |  |
| --- | --- |
| ppuv | 调用后会在\*ppuv里面返回一个float\*的指针，指向所有顶点的uv纹理坐标。数据格式是每个顶点两个float。 |
| ppebo | 调用后会在\*ppebo里面返回一个short\*的指针，指向mesh的EBO (Element Array Buffer)数据。数据格式是每个三角形3个short。 |
| pn\_vertices | 调用后会在\*pn\_vertices里面返回mesh的顶点个数 |
| pn\_triangles | 调用后会在\*pn\_triangles里面返回mesh的三角形个数 |

用于获取mesh数据中的静态部分。返回的数据不会改变，只需要在ddear\_init\_global\_tables之后调用一次即可。返回的指针均不需释放。

**#2 int ddear\_get\_vertices(TWorkArea\* context, float\*\* ppv,float\* pmatrix);**

|  |  |
| --- | --- |
| context | Face tracking API的context |
| ppv | 调用后会在\*ppv里面返回一个float\*的指针，指向mesh的顶点数据。数据格式是每个顶点3个float。返回的指针指向context内部，不需释放。 |
| pmatrix | 指向一个float[16]的数组，调用后会在数组里返回对应于人脸当前位置的刚体变换矩阵。 |

用于获取动态的mesh顶点数据。必须在context里面有可用的人脸数据，也就是每帧hldde\_next或easydde\_run成功后进行调用，否则可能崩溃。

这里取到的mesh顶点数据与dde\_get/easydde\_get到的landmarks\_3d数据使用同一块地址，如果需要同时使用，要先把数据存到其他地址。

## 5.1指针指向的数据大小

在用ddear\_get\_static\_data或ddear\_get\_vertices获得了数据之后可能会需要用memcpy传到其他地方或用glBufferSubData之类的方式更新到VBO里。这时会需要用到每个指针指向的数据大小。注意这些都可以通过\*pn\_vertices和\*pn\_triangles算出来：

|  |  |
| --- | --- |
| 指针 | 指向的数据大小（字节数） |
| \*ppuv | \*pn\_vertices \* 2\*sizeof(float) |
| \*ppebo | \*pn\_triangles \* 3\*sizeof(short) |
| \*ppv | \*pn\_vertices \* 3\*sizeof(float) |

## 5.2 投影变换

为了让mesh的绘制结果符合摄像头中的人脸，需要让渲染时投影矩阵的FOV和摄像头本身相符。FOV的值可以从tracker中获取。

首先需要通过dde\_get或者easydde\_get来获取focal\_length。focal\_length是一个float[1]的数组，里面存放的是tracker内部计算的摄像头焦距。然后就可以通过以下公式计算出纵向FOV的弧度值：

FOV=atan2(h\*0.5, focal\_length[0])\*2.f;

其中h为图片高度。

# 6.使用easymultiface系列函数

easymultiface系列函数主要用于多脸追踪。

## 6.1 相关函数—按照调用顺序

**#1 int easymultiface\_set\_max\_faces(int n\_max\_faces);**

把最大同时跟踪的脸数设到n\_max\_faces。n\_max\_faces必须在2到8之间（包含2和8）。注意设得太大会影响性能。

**#2 int easymultiface\_run(int\* p\_invalidation\_mask, const void\* img, int stride,int w, int h,int flags);**

p\_invalidation\_mask接收每个脸是否已经消失的mask。如果p\_invalidation\_mask&(1<<i) 非0，表示这一帧经过之后系统发现第i号脸消失了。注意，在同一帧里可能已经有一个新脸又占用了第i号的位置，所以不能用函数的返回值来判断这件事。

img 用于识别的图像

stride 图像两行之间相隔的字节数。如果不知道可以填0，表示使用默认值。

w 图像宽度

h 图像高度

flags 可选标志如下：

FLAG\_IMAGE\_FORMAT\_RGBA –0

FLAG\_IMAGE\_FORMAT\_NV21– 1

return 接收每个脸当前帧是否获得了合法跟踪结果的mask。如果返回值&(1<<i) 非0，表示第i号脸在这一帧里产生了合法的跟踪结果数据。

函数包含了面部识别与表情追踪，使用时调用这一个函数即可。

flags标志一般设为FLAG\_IMAGE\_FORMAT\_RGBA；如果在Android项目中使用，可以在`Camera.Parameters`中调用`setPreviewFormat(ImageFormat.NV21)`，从`onPreviewFrame`中获得摄像头图像，并将标志设为FLAG\_IMAGE\_FORMAT\_NV21。

easymultiface\_run返回的脸的编号在帧间是有对应的。如果一个人在第n帧是第2号脸，那么这人在第n+1帧应该还是第2号脸。但脸的编号未必是从0开始连续排的。比如有A和B两个人在画面中同时出现，A是0号脸，B是1号脸，那么在A从画面中离开之后就会出现1号脸有数据而0号脸没数据的情况。

**#3 TWorkArea\* easymultiface\_get\_context(int face\_id);**

获取第face\_id号脸的context。这个context可以传给dde\_get，用于获取对应脸的数据。

# 联系

如发现文档中的错误或问题，请与我们联系。