libface Reference

PlayStation®Vita Programmer Tool Runtime Library Release 3.50

Document serial number: 000004892117

Document serial number: 000004892117

Table of Contents

Document serial number: 000004892117

Face Detection APIs

SceFaceDetectionDictPtr

Face detection dictionary data pointer type

Definition

```
#include <libface.h>
typedef void *SceFaceDetectionDictPtr;
```

Description

This defined type is used as a pointer to dictionary data used in face detection.

It is used for an argument type of the face detection execution functions `sceFaceDetection()` and `sceFaceDetectionLocal()`.

See Also

`sceFaceDetection()`,`sceFaceDetectionEx()`,`sceFaceDetectionLocal()`

SceFaceDetectionParam

Face detection (extended version) parameters

Definition

```
#include <libface.h>
typedef struct SceFaceDetectionParam {
        int version;
        int size;
        float magBegin;
        float magStep;
        float magEnd;
        int xScanStart;
        int yScanStart;
        int xScanStep;
        int yScanStep;
        float xScanOver;
        float yScanOver;
        float thresholdScore;
        int resultPrecision;
        int searchType;
} SceFaceDetectionParam;
```

Members

version	Version number of this structure (=2)
size	Size of this structure (=56)
magBegin	Input image scaling starting magnification
magStep	Input image scaling rate (recommended value: 0.841f)
magEnd	Input image scaling ending magnification (recommended value: 0.0f)
xScanStart	Face detection window horizontal start position [pixel]
yScanStart	Face detection window vertical start position [pixel]
xScanStep	Face detection window horizontal shift amount [pixel] (recommended value: 2)
yScanStep	Face detection window vertical shift amount [pixel] (recommended value: 2)
xScanOver	Face detection window horizontal overscan amount [%]
yScanOver	Face detection window vertical overscan amount [%]
thresholdScore	Face detection rate threshold score (recommended value: 0.50f)
resultPrecision	Face detection result position precision setting value
searchType	Setting to end face detection processing

Description

This datatype represents the parameters for face detection (extended version). It is used with the face detection (extended version) execution function `sceFaceDetectionEx()`. The default parameters can be set using the `sceFaceDetectionGetDefaultParam()` function.

For *version*, input 2. For *size*, set the size of this structure (=56 [byte]) in bytes. These values will be used for future extensions of this structure.

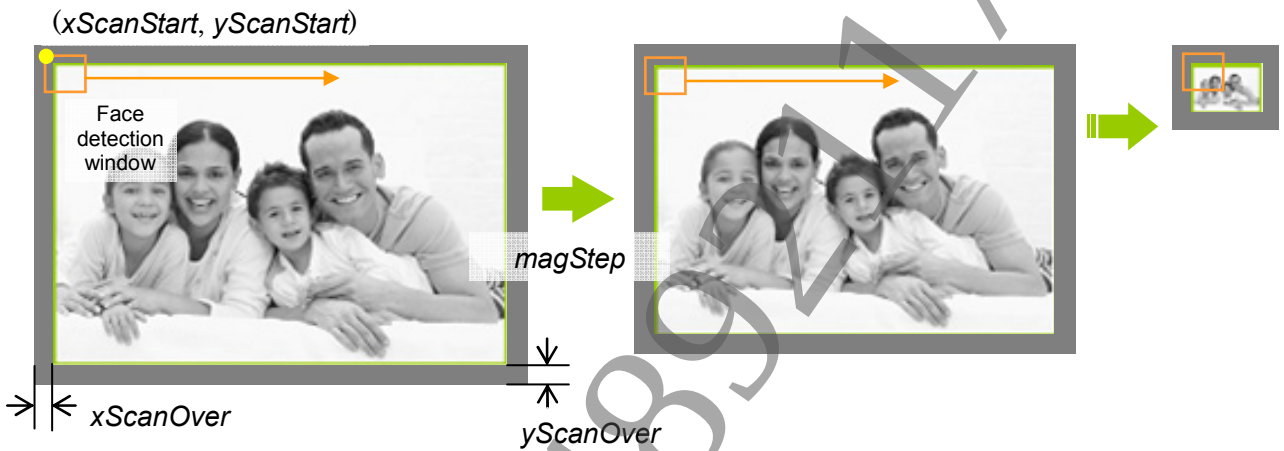
For *magBegin*, *magStep*, *magEnd*, *xScanStep*, *yScanStep*, *thresholdScore*, and *resultPrecision*, refer to the Description for `sceFaceDetection()`.

For *xScanStart* and *yScanStart*, set the start points of the face detection window. Normally, the start points are set to (0, 0), but for camera input, by setting *xScanStep* and *yScanStep* to (4, 4) while changing *xScanStart* and *yScanStart* to (0, 0) -> (2, 0) -> (0, 2) -> (2, 2) every frame, the face detection time can be parsed and executed chronologically

For *xScanOver* and *yScanOver*, set the ratio of the width of the face detection window for the portion outside of the face detection window to perform searching. The values that can be set for *xScanOver* and *yScanOver* are 0.0 to 0.5. For example, when *xScanOver* is set to 0.5, it may be possible to detect faces where half of the face is outside the input image.

For searchType, set either SCE_FACE_DETECT_SEARCH_ALL_FACE(0) or SCE_FACE_DETECT_SEARCH_FACE_NUM_LIMIT(1) according to the search purpose. When SCE_FACE_DETECT_SEARCH_ALL_FACE is set, detection processing will be carried out for all faces in the input image and results will be output accordingly. When SCE_FACE_DETECT_SEARCH_FACE_NUM_LIMIT is set, face detection processing will end when the number of faces specified to the *resultFaceArraySize* argument of sceFaceDetectionEx() in the input image is detected in order starting with the largest face. By setting a small value for *resultFaceArraySize* as appropriate, face detection processing can be completed in high speed.

Figure 1 Parameter Descriptions



See Also

sceFaceDetectionEx(), sceFaceDetectionGetDefaultParam()

SceFaceDetectionResult

Face detection results (detected face region)

Definition

```
#include <libface.h>
typedef struct SceFaceDetectionResult {
        float faceX;
        float faceY;
        float faceW;
        float faceH;
        float faceRoll;
        float facePitch;
        float faceYaw;
        float score;
} SceFaceDetectionResult;
```

Members

faceX	x coordinate of upper left corner of detected face region
faceY	y coordinate of upper left corner of detected face region
faceW	Width of detected face region
faceH	Height of detected face region
faceRoll	Planar (roll axis) rotation angle of detected face region in radians
facePitch	Upward or vertical (pitch axis) rotation angle of detected face region in radians
faceYaw	Sideways or horizontal (yaw axis) rotation angle of detected face region in radians
score	Detected face region score that shows the probability of face.

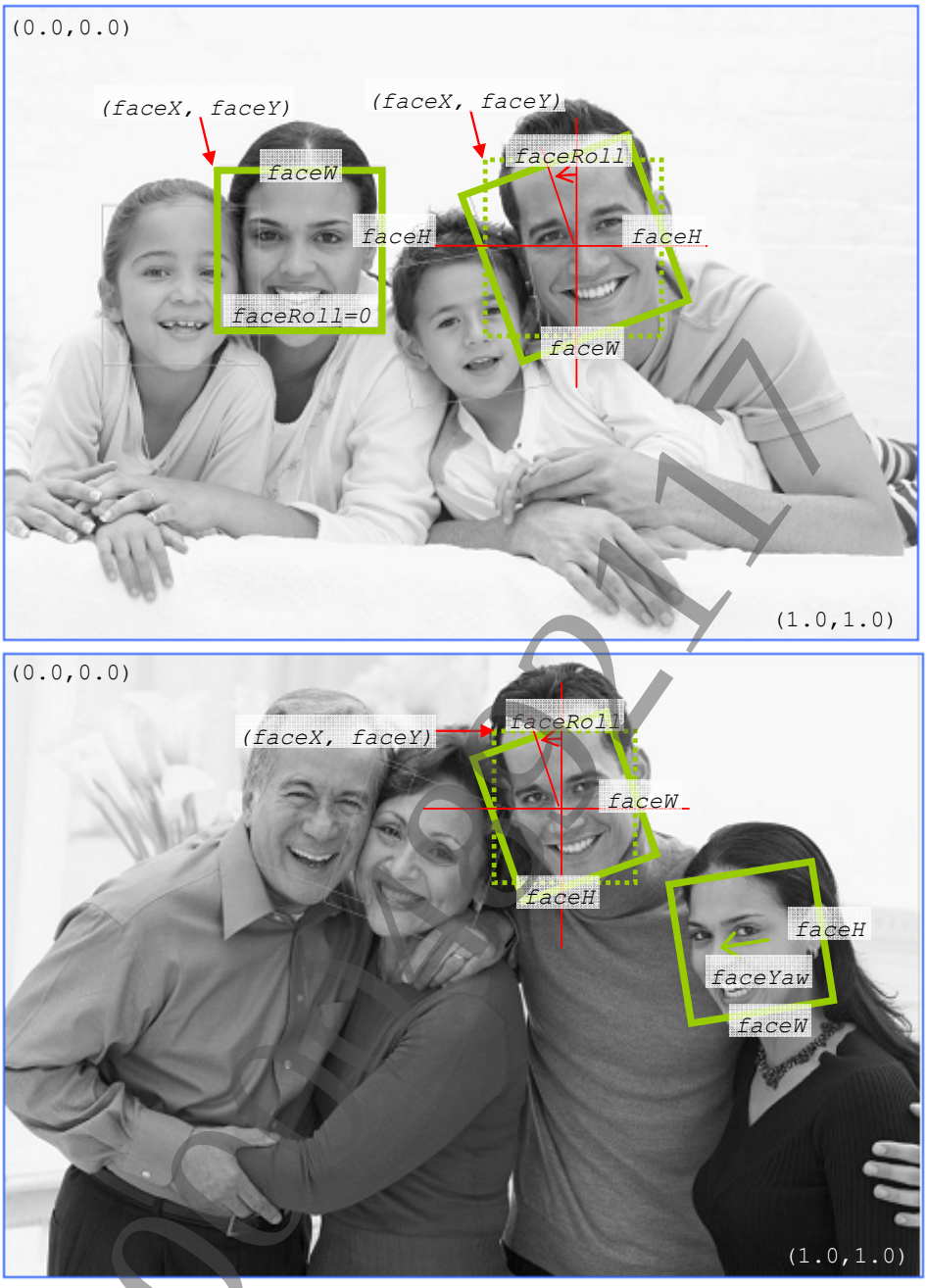
Description

This data type is output for face detection results. When a face detection execution function is called (`sceFaceDetection()`, `sceFaceDetectionEx()` or `sceFaceDetectionLocal()`), data for the number of faces that were detected is written to the memory area (`SceFaceDetectionResult` array) pointed to by the pointer that was set in the *`resultFaceArray`* argument of these functions.

Also, since these face detection results are used in the processing of `sceFaceParts()`, `sceFaceEstimatePoseRegion()`, and `sceFaceAttribute()`, set a pointer to detected face data (`SceFaceDetectionResult`) in the *`detectedFace`* argument of these functions.

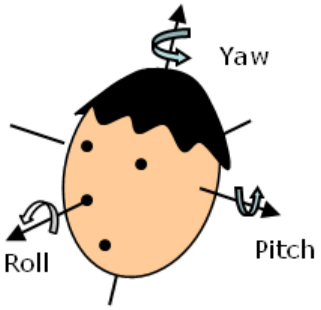
`faceX`, *`faceY`*, *`faceW`*, and *`faceH`* are represented in a coordinate system that is normalized by the width and height of the input image.

Figure 2 Face Detection Results



faceRoll, *facePitch*, and *faceYaw* are defined in a right-hand coordinate system, as shown in Figure 3.

Figure 3 Rotation Axes in Face Detection Results (Roll, Pitch, Yaw) and Their Directions



The following code converts the coordinates of the face area (*faceX*, *faceY*, *faceW*, *faceH*, *faceRoll*, *facePitch*, *faceYaw*) into coordinates on the input image.

```
void rotate(int* ox, int *oy, int ix, int iy, int cx, int cy, float radian)
{
    const float c = cosf(radian);
    const float s = sinf(radian);
    ix -= cx;
    iy -= cy;
    *ox = (int)(ix * c - iy * s + cx);
    *oy = (int)(ix * s + iy * c + cy);
}

SceFaceDetectionResult face;
const int x1 = (int)( face.faceX * image_width);
const int y1 = (int)( face.faceY * image_height);
const int x2 = (int)((face.faceX + face.faceW) * image_width);
const int y2 = (int)((face.faceY + face.faceH) * image_height);
const int cx = (int)((face.faceX + face.faceW / 2) * image_width);
const int cy = (int)((face.faceY + face.faceH / 2) * image_height);
int face_top_left_x, face_top_left_y;  // pixel position
int face_top_right_x, face_top_right_y;  // pixel position
int face_bottom_left_x, face_bottom_left_y;  // pixel position
int face_bottom_right_x, face_bottom_right_y;  // pixel position

rotate(&face_top_left_x, &face_top_left_y, x1, y1, cx, cy, -face.faceRoll);
rotate(&face_top_right_x, &face_top_right_y, x2, y1, cx, cy, -face.faceRoll);
rotate(&face_bottom_left_x, &face_bottom_left_y, x1, y2, cx, cy, -face.faceRoll);
rotate(&face_bottom_right_x, &face_bottom_right_y, x2, y2, cx, cy, -face.faceRoll);
```

See Also

sceFaceDetection(), sceFaceDetectionEx(), sceFaceDetectionLocal(), sceFaceParts(), sceFaceAllParts(), sceFaceEstimatePoseRegion(), sceFaceAttribute()

sceFaceDetection

Execute face detection

Definition

```
#include <libface.h>
int sceFaceDetection(
        const unsigned char *imgPtr,
        int width,
        int height,
        int rowstride,
        const SceFaceDetectionDictPtr detectDictPtr,
        float magBegin,
        float magStep,
        float magEnd,
        int xScanStep,
        int yScanStep,
        float thresholdScore,
        int resultPrecision,
        SceFaceDetectionResult resultFaceArray[],
        int resultFaceArraySize,
        int *resultFaceNum,
        void *workMemory,
        int workMemorySize
)
```

Arguments

imgPtr	Pointer to input image (8-bit grayscale)
width	Input image width [pixels]
height	Input image height [pixels]
rowstride	Input image data width
detectDictPtr	Pointer to face detection dictionary data
magBegin	Input image scaling starting magnification
magStep	Input image scaling rate (recommended value: 0.841f)
magEnd	Input image scaling ending magnification (recommended value: 0.0f)
xScanStep	Face detection window horizontal shift amount [pixels] (recommended value: 2)
yScanStep	Face detection window vertical shift amount [pixels] (recommended value: 2)
thresholdScore	Face detection rate adjustment value (recommended value: 0.50f)
resultPrecision	Face detection result position precision setting
resultFaceArray	Pointer to face detection result output area
resultFaceArraySize	Number of elements in *resultFaceArray*
resultFaceNum	Number of detected faces
workMemory	Pointer to work buffer
workMemorySize	Size of work buffer

Return Values

Returns `SCE_OK`(0) if processing is successful.

Returns one of the following error codes (a negative value) for errors.

Value	(Hexadecimal)	Description
`SCE_FACE_ERROR_NO_MEMORY`	0x808B0001	Not enough memory to run, or *`workMemory`* is NULL
`SCE_FACE_ERROR_INVALID_PARAM`	0x808B0002	Parameter is invalid
`SCE_FACE_ERROR_INVALID_DICT`	0x808B0003	Face detection dictionary data is invalid, or *`detectDictPtr`* is NULL

Description

This function performs face detection.

For *`detectDictPtr`*, set a pointer to the memory area where one of the face detection dictionary data files `SCE_FACE_DETECT_FRONTAL_DICT`, `SCE_FACE_DETECT_ROLL_DICT`, `SCE_FACE_DETECT_YAW_DICT`, `SCE_FACE_DETECT_ROLL_YAW_DICT`, or `SCE_FACE_DETECT_ROLL_YAW_PITCH_DICT` is read according to the requirements of the application.

The main difference in these face detection dictionaries is in the trade-off between the supported range of detectable face angles and processing time.

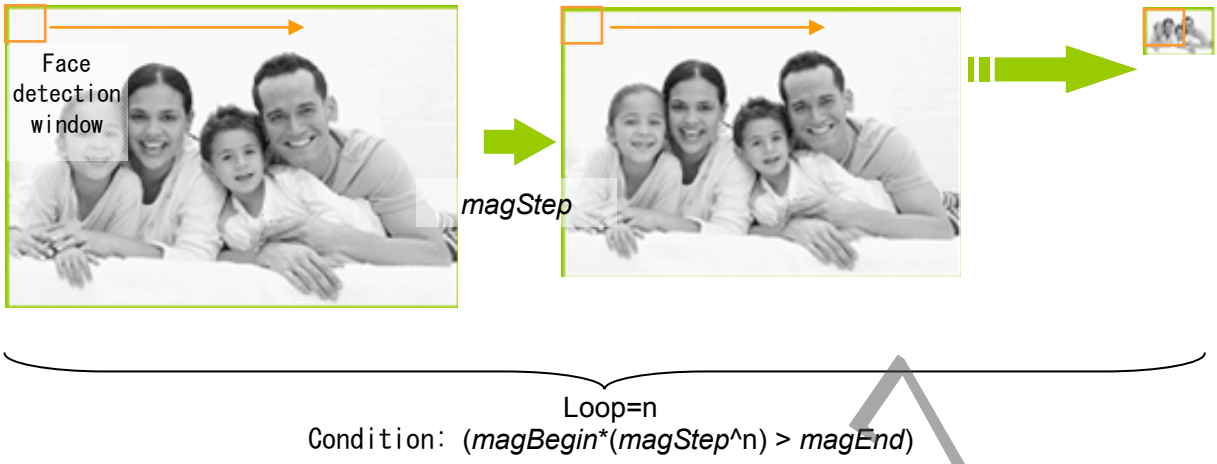
For *`magBegin`*, set a value not exceeding 1.0f as the starting magnification for input image scaling. The value 20/*`magBegin`* is the minimum face detection size [pixels] for the input image. Setting *`magBegin`* to the smallest possible value will shorten the processing time, but if it is too small, only large faces in the input image will be able to be detected. Since the *`magBegin`* value has a relatively large effect on processing time, be sure to adjust it to a suitable value to match the application use and image size.

For *`magEnd`*, set a value of 0.0f or more as the ending magnification for input image scaling. The value 20/*`magEnd`* is the maximum face detection size [pixels] in the input image. However, since the processing time generally does not change very much by changes in the *`magEnd`* setting, the recommended value is 0.0f, which can always detect a large face in one image regardless of the image size.

The recommended value for *`magStep`* is 0.841f. Faces of various sizes can be detected by searching for faces while multiplying *`magBegin`* by *`magStep`* until *`magEnd`* is reached. Although a reduction in the number of intermediate magnifications and the total processing time can be achieved by setting *`magStep`* to a smaller value (such as 0.75f), the detection precision will drop since face detection will become coarser according to the size.

The recommended value for *`xScanStep`* and *`yScanStep`* is 2 [pixels]. These arguments represent the positional coarseness of face detection processing, and although detection precision will increase if they are set to 1 [pixel], processing time will also increase. On the other hand, if they are set to 3 [pixels], processing time will decrease, but detection omissions may occur. *`xScanStep`* and *`yScanStep`* may also be set to different values.

Figure 4 Input Image Scaling



thresholdScore can be used to fine-tune the detection rate. Set a value from 0.0f to 1.0f. Although 0.50f is normally set, you should set a smaller value when it is better to have fewer detection omissions even if it means invalid detections will increase somewhat (non-face locations judged to be faces). On the contrary, you should set a larger value when it is better to decrease invalid detection even if it misses the faces. The processing time is not affected even if this parameter is changed.

For *resultPrecision*, select either `SCE_FACE_DETECT_RESULT_NORMAL`, `SCE_FACE_DETECT_RESULT_PRECISE`, or `SCE_FACE_DETECT_RESULT_FAST` to suit the application use.

Since there is a trade-off between detection precision and processing time as described earlier, adjust these parameters to set the optimum balance to suit the application use. To adjust the parameters, it is recommended that you perform test runs under conditions that are close to the actual application usage conditions.

For *resultFaceArray*, set a pointer to the `SceFaceDetectionResult` array for storing face detection results.

For *resultFaceArraySize*, set the number of array elements in *resultFaceArray*. The number of face detection results that are output will not exceed *resultFaceArraySize* so that a buffer overrun will not occur.

The number of detected faces is returned in *resultFaceNum*.

For *workMemory*, set the pointer to the allocated memory whose size is calculated by `sceFaceDetectionGetWorkingMemorySize()` or more.

For *workMemorySize*, set the size of *workMemory*.

When an `SCE_FACE_ERROR_NO_MEMORY` error occurs, 0 is always returned in *resultFaceNum*.

Notes

During the face detection processing, if the temporary storage area used internally for intermediate results becomes insufficient, the detection precision that is output may end up getting slightly worse.

Size of the internally-used work buffer may require more than the size of `sceFaceDetectionGetWorkingMemorySize()` if too many face detection results are extracted in the image.

If the input image shooting conditions were not ideal and the face parts are dark due to backlighting or insufficient light or if, on the other hand, the face is overexposed and is whitened out, or if the image contains a lot of noise, face detection rate may worsen even if processing is performed normally by the program. Adjust the input image quality as needed before calling this function.

Example

```
/* Read face recognition dictionary data file */
unsigned char detectDict[SCE_FACE_DETECT_FRONTAL_DICT_SIZE];
SceUID fd = sceIoOpen("host0:"SCE_FACE_DETECT_FRONTAL_DICT, SCE_O_RDONLY, 0);
sceIoRead(fd, detectDict, SCE_FACE_DETECT_FRONTAL_DICT_SIZE);
sceIoClose(fd);

unsigned char yImg[320*240];
SceFaceDetectionResult detectResult[16];
int i, numFace, ret;

/* Allocate work buffer for face detection */
int workSize = sceFaceDetectionGetWorkingMemorySize(320, 240, 320, detectDict);
void *workPtr = malloc(workSize);

/* Execute face detection */
ret = sceFaceDetection(
        yImg, 320, 240, 320,
        detectDict,
        0.4f, 0.841f, 0.0f, 2, 2, 0.5f,
        SCE_FACE_DETECT_RESULT_NORMAL,
        detectResult, 16, &numFace,
        workPtr, workSize);
if (ret != SCE_OK) {
        printf("sceFaceDetection() failed! (0x%08x)\n", ret);
        return;
}

/* Print result */
for (i = 0; i < numFace; i++) {
        printf("face[%d] x = %f y = %f w = %f h = %f\n", i,
              detectResult[i].faceX, detectResult[i].faceY,
              detectResult[i].faceW, detectResult[i].faceH);
}
```

See Also

SceFaceDetectionDictPtr, SCE_FACE_DETECT_FRONTAL_DICT,
SCE_FACE_DETECT_ROLL_DICT, SCE_FACE_DETECT_YAW_DICT,
SCE_FACE_DETECT_ROLL_YAW_DICT, SCE_FACE_DETECT_ROLL_YAW_PITCH_DICT,
SceFaceDetectionResult, sceFaceDetectionGetWorkingMemorySize()

sceFaceDetectionEx

Face detection (extended version) execution

Definition

```
#include <libface.h>
int sceFaceDetectionEx(
        const unsigned char *imgPtr,
        int width,
        int height,
        int rowstride,
        const SceFaceDetectionDictPtr detectDictPtr,
        const SceFaceDetectionParam *detectParam,
        SceFaceDetectionResult resultFaceArray[],
        int resultFaceArraySize,
        int *resultFaceNum,
        void *workMemory,
        int workMemorySize
)
```

Arguments

imgPtr	Pointer to input image (8-bit grayscale)
width	Input image width [pixels]
height	Input image height [pixels]
rowstride	Input image data width
detectDictPtr	Pointer to face detection dictionary data
detectParam	Pointer to face detection (extended version) parameters
resultFaceArray	Pointer to face detection result output area
resultFaceArraySize	Number of elements in *resultFaceArray*
resultFaceNum	Number of detected faces
workMemory	Pointer to work buffer
workMemorySize	Size of work buffer

Return Values

Returns SCE_OK(0) if processing is successful.

Returns one of the following error codes (a negative value) for errors.

Value	(Hexadecimal)	Description
SCE_FACE_ERROR_NO_MEMORY	0x808B0001	Not enough memory to run, or *workMemory* is NULL
SCE_FACE_ERROR_INVALID_PARAM	0x808B0002	Parameter is invalid
SCE_FACE_ERROR_INVALID_DICT	0x808B0003	Face detection dictionary data is invalid, or *detectDictPtr* is NULL

Description

This function is for performing face detection (extended version). With this function, it is possible to detect the specified number of faces in order starting from the largest face and including faces that may be partially cut off from the image.

For *imgPtr*, *width*, *height*, *rowstride*, *detectDictPtr*, *resultFaceArray*, *resultFaceArraySize*, *resultFaceNum*, *workMemory*, and *workMemorySize*, refer to the Description for `sceFaceDetection()`.

For *detectParam*, set the parameters for face detection (extended version) using `sceFaceDetectionGetDefaultParam()`, etc.

Notes

When parameters that can detect faces that are outside the image are set, the processing time will take longer than `sceFaceDetection()`.

When `SCE_FACE_DETECT_SEARCH_FACE_NUM_LIMIT` is set to the *searchType* member of `SceFaceDetectionParam`, face detection processing will end when the number of faces specified in *resultFaceArraySize* is detected in the input image in order starting with the largest face. By setting a small value for *resultFaceArraySize* as appropriate, face detection processing can be completed in high speed.

Examples

```
/* Read face recognition dictionary data file */
unsigned char detectDict[SCE_FACE_DETECT_FRONTAL_DICT_SIZE];
SceUID fd = sceIoOpen("host0:"SCE_FACE_DETECT_FRONTAL_DICT, SCE_O_RDONLY, 0);
sceIoRead(fd, detectDict, SCE_FACE_DETECT_FRONTAL_DICT_SIZE);
sceIoClose(fd);

unsigned char yImg[320*240];
SceFaceDetectionResult detectResult[16];
int i, numFace, ret;

/* Allocate work buffer for face detection */
int workSize = sceFaceDetectionGetWorkingMemorySize(320, 240, 320, detectDict);
void *workPtr = malloc(workSize);

/* Get/set parameters for face detection (extended version) */
SceFaceDetectionParam detectParam;
sceFaceDetectionGetDefaultParam(&detectParam);

/* Face detection (extended version) execution */
ret = sceFaceDetectionEx(
        yImg, 320, 240, 320,
        detectDict,
        &detectParam,
        detectResult, 16, &numFace,
        workPtr, workSize);
if (ret != SCE_OK) {
        printf("sceFaceDetection() failed! (0x%08x)\n", ret);
        return;
}

/* Print result */
for (i = 0; i < numFace; i++) {
        printf("face[%d] x = %f y = %f w = %f h = %f\n", i,
```

```
                    detectResult[i].faceX, detectResult[i].faceY,
                    detectResult[i].faceW, detectResult[i].faceH);
    }
```

See Also

SceFaceDetectionDictPtr, SceFaceDetectionParam, SCE_FACE_DETECT_FRONTAL_DICT, SCE_FACE_DETECT_ROLL_DICT, SCE_FACE_DETECT_YAW_DICT, SCE_FACE_DETECT_ROLL_YAW_DICT, SCE_FACE_DETECT_ROLL_YAW_PITCH_DICT, SceFaceDetectionResult, sceFaceDetectionGetDefaultParam(), sceFaceDetectionGetWorkingMemorySize()

sceFaceDetectionLocal

Execute fast face detection using local search

Definition

```
#include <libface.h>
int sceFaceDetectionLocal(
        const unsigned char *imgPtr,
        int width,
        int height,
        int rowstride,
        const SceFaceDetectionDictPtr detectDictPtr,
        float magStep,
        float xExpandRegion,
        float yExpandRegion,
        int xScanStep,
        int yScanStep,
        float thresholdScore,
        const SceFaceDetectionResult referenceFaceArray[],
        int referenceFaceArraySize,
        SceFaceDetectionResult resultFaceArray[],
        int resultFaceArraySize,
        int *resultFaceNum,
        void *workMemory,
        int workMemorySize
)
```

Arguments

imgPtr	Pointer to input image (8-bit grayscale)
width	Input image width [pixels]
height	Input image height [pixels]
rowstride	Input image data width
detectDictPtr	Pointer to face detection dictionary data
magStep	Input image scaling rate (recommended value: 0.841f)
xExpandRegion	Horizontal magnification rate of face region to reference (search range adjustment)
yExpandRegion	Vertical magnification rate of face region to reference (search range adjustment)
xScanStep	Face detection window horizontal shift amount [pixels] (recommended value: 1)
yScanStep	Face detection window vertical shift amount [pixels] (recommended value: 1)
thresholdScore	Face detection rate adjustment value (recommended value: 0.50f)
referenceFaceArray	Pointer to face detection result array to reference
referenceFaceArraySize	Number of faces in face detection results to reference
resultFaceArray	Pointer to face detection result output area
resultFaceArraySize	Number of elements in *resultFaceArray*
resultFaceNum	Number of detected faces
workMemory	Pointer to work buffer
workMemorySize	Size of work buffer

Return Values

Returns `SCE_OK`(0) if processing is successful.

Returns one of the following error codes (a negative value) for errors.

Value	(Hexadecimal)	Description
`SCE_FACE_ERROR_NO_MEMORY`	0x808B0001	Not enough memory to run, or *`workMemory`* is NULL
`SCE_FACE_ERROR_INVALID_PARAM`	0x808B0002	Parameter is invalid
`SCE_FACE_ERROR_INVALID_DICT`	0x808B0003	Face detection dictionary data is invalid, or *`detectDictPtr`* is NULL

Description

This function uses local search to perform fast face detection.

When face detection is performed on time-sequenced images such as video signals, the detection speed can be increased by executing the face search processing only in the vicinity of face regions detected in the previous frame (i.e. performing a local search).

The descriptions of arguments that are the same as those of `sceFaceDetection()` have been omitted. See the description of `sceFaceDetection()`.

For *`referenceFaceArray`*, specify the `SceFaceDetectionResult` array in which the results of face detection processing performed for the entire image by `sceFaceDetection()` are saved, or specify the `SceFaceDetectionResult` array in which the face detection results of the local search of the previous frame performed by `sceFaceDetectionLocal()` are saved. For *`numReferenceFace`*, specify the number of faces.

Referencing the face region positions and sizes specified in *`referenceFaceArray`* to narrow the range for the scaling and scanning operations used in face detection processing can significantly reduce the processing time compared with searching the entire image.

The role of *`magStep`* is the same as for `sceFaceDetection()` and the recommended value is 0.841f, but the number of scaling steps is limited internally to 3 (centered on the size of the face region to be referenced). If faster processing is required and the face size will hardly change at all, setting 0.0f will cause processing to be performed by scaling in only one step, which will increase speed but in this case, changes in face size will no longer be able to be handled.

For x*`ExpandFaceRegion`* and *`yExpandFaceRegion`*, specify a value of 1.0f or more as the magnification rate of the face region to reference. The smaller this value is, the narrower the search range will be, which can significantly reduce the processing time. However, if this value is too small, processing may not be able to deal with cases when faces move quickly and they are more likely to be missed.

The roles of *`xScanStep`* and *`yScanStep`* are also the same as for `sceFaceDetection()`. Detection errors are less likely to occur for 1 pixel and the position accuracy also increases, however, if faster processing is required, set one or both of the values for *`xScanStep`* and *`yScanStep`* to 2 pixels. In that case, detection errors may be more likely to occur comparing to setting 1 pixel in *`xScanStep`* and *`yScanStep`*.

For *`workMemory`*, set the pointer to the allocated memory whose size is calculated by `sceFaceDetectionGetWorkingMemorySize()` or more.

For *`workMemorySize`*, set the size of *`workMemory`*.

Since there is a trade-off between the search range and processing time as described above, adjust these parameters to set the optimum balance to suit the application use. To adjust the parameters, it is recommended that you perform test runs under conditions that are close to the actual application usage conditions.

Notes

These notes are omitted since they are the same as the notes for `sceFaceDetection()`. See the notes for `sceFaceDetection()`.

Example

```
/* Read face recognition dictionary data file */
unsigned char detectDict[SCE_FACE_DETECT_FRONTAL_DICT_SIZE];
SceUID fd = sceIoOpen("host0:"SCE_FACE_DETECT_FRONTAL_DICT, SCE_O_RDONLY, 0);
sceIoRead(fd, detectDict, SCE_FACE_DETECT_FRONTAL_DICT_SIZE);
sceIoClose(fd);

unsigned char yImg[320*240];
SceFaceDetectionResult detectResult[16];
SceFaceDetectionResult detectResult2[16];
int i, numFace, numFace2, ret;

/* Allocate work buffer for face detection */
int workSize = sceFaceDetectionGetWorkingMemorySize(320, 240, 320, detectDict);
void *workPtr = malloc(workSize);

/* Execute face detection */
ret = sceFaceDetection(
        yImg, 320, 240, 320,
        detectDict,
        0.4f, 0.841f, 0.0f, 2, 2, 0.5f,
        SCE_FACE_DETECT_RESULT_NORMAL,
        detectResult, 16, &numFace,
        workPtr, workSize);
if (ret != SCE_OK) {
        printf("sceFaceDetection() failed! (0x%08x)\n", ret);
        return;
}

/* Execute local search face detection */
ret = sceFaceDetectionLocal(
        yImg, 320, 240, 320,
        detectDict,
        0.841f, 1.1f, 1.1f, 1, 1, 0.5f,
        detectResult, numFace,
        detectResult2, 16, &numFace2,
        workPtr, workSize);
if (ret != SCE_OK) {
        printf("sceFaceDetectionLocal() failed! (0x%08x)\n", ret);
        return;
}
for (i = 0; i < numFace2; i++) {
        printf("face[%d] x = %f y = %f w = %f h = %f\n", i,
              detectResult2[i].faceX, detectResult2[i].faceY,
              detectResult2[i].faceW, detectResult2[i].faceH);
}
```

See Also

`SceFaceDetectionDictPtr`, `SCE_FACE_DETECT_FRONTAL_DICT`, `SCE_FACE_DETECT_ROLL_DICT`, `SCE_FACE_DETECT_YAW_DICT`, `SCE_FACE_DETECT_ROLL_YAW_DICT`, `SCE_FACE_DETECT_ROLL_YAW_PITCH_DICT`, `SceFaceDetectionResult`, `sceFaceDetectionGetWorkingMemorySize()`

sceFaceDetectionGetDefaultParam

Get face detection (extended version) default parameters

Definition

```
#include <libface.h>
int sceFaceDetectionGetDefaultParam(
        SceFaceDetectionParam *detectParam
)
```

Arguments

detectParam	Pointer to face detection dictionary data

Return Values

Stores recommended values of face detection parameters required for face detection (extended version) execution in *detectParam* and returns SCE_OK(0) if processing is successful.

Returns the following error code (a negative value) for errors.

Value	(Hexadecimal)	Description
SCE_FACE_ERROR_INVALID_PARAM	0x808B0002	Parameter is invalid. *detectParam* is NULL

Description

The face detection parameters required for face detection (extended version) execution are initialized at the standard settings by this function.

Each parameter will be initialized at the following recommended values (default values).

```
detectParam->version          = 2;
detectParam->size             = sizeof(SceFaceDetectionParam);
detectParam->magBegin         = 0.5f;
detectParam->magStep          = 0.841f;
detectParam->magEnd           = 0.0f;
detectParam->xScanStart       = 0;
detectParam->yScanStart       = 0;
detectParam->xScanStep        = 2;
detectParam->yScanStep        = 2;
detectParam->xScanOver        = 0.5f;
detectParam->yScanOver        = 0.5f;
detectParam->thresholdScore   = 0.5f;
detectParam->resultPrecision  = SCE_FACE_DETECT_RESULT_PRECISE;
detectParam->searchType       = SCE_FACE_DETECT_SEARCH_ALL_FACE;
```

See Also

SceFaceDetectionParam, sceFaceDetectionEx()

sceFaceDetectionGetWorkingMemorySize

Calculate the size of working memory for face detection

Definition

```
#include <libface.h>
int sceFaceDetectionGetWorkingMemorySize(
        int width,
        int height,
        int rowstride,
        const SceFaceDetectionDictPtr detectDictPtr,
)
```

Arguments

width	Input image width for face detection [pixels]
height	Input image height for face detection [pixels]
rowstride	Input image data width for face detection
detectDictPtr	Pointer to face detection dictionary data

Return Values

Returns the size of work buffer for face detection.

Returns 0 for errors.

Description

This function calculates the size of work buffer for face detection.

Allocates the memory with the size calculated by this function, then call `sceFaceDetection()`, `sceFaceDetectionEx()` or `sceFaceDetectionLocal()` with the pointer to the work buffer and its size.

If this function returns 0, the face detection dictionary data is invalid or parameters are invalid.

Notes

As executing the face detection, the size of internally used work buffer differs depending on the number of faces detected in the input image. It may need more than the size returned by this function, if too many face detection results are extracted in the image.

See Also

`sceFaceDetection()`,`sceFaceDetectionEx()`,`sceFaceDetectionLocal()`

SCE_FACE_DETECT_FRONTAL_DICT

Face detection dictionary data file (for frontal face)

Definition

```
#include <libface.h>
#define SCE_FACE_DETECT_FRONTAL_DICT "face_detect_frontal.fdt"
#define SCE_FACE_DETECT_FRONTAL_DICT_SIZE (43488)
```

Description

These constants represent the filename and size (in bytes) of dictionary data used in face detection.

The *detectDictPtr* argument of the face detection execution functions `sceFaceDetection()`, `sceFaceDetectionEx()` and `sceFaceDetectionLocal()` should be set with a pointer to the memory area where this file was read.

This dictionary only detects frontal faces.

Notes

Although this dictionary can only detect faces that are (more-or-less) directly facing the camera without any tilt, the processing time is shorter compared to other face detection dictionaries.

Choose the type of face detection dictionary that is best suited for the application.

Example

```
SceFaceDetectionDictPtr detectDict =
        (SceFaceDetectionDictPtr)malloc(SCE_FACE_DETECT_FRONTAL_DICT_SIZE);
SceUID fd = sceIoOpen("host0:"SCE_FACE_DETECT_FRONTAL_DICT, SCE_O_RDONLY, 0);
sceIoRead(fd, detectDict, SCE_FACE_DETECT_FRONTAL_DICT_SIZE);
sceIoClose(fd);

ret = sceFaceDetection(
        yImg, 320, 240, 320,
        detectDict,
        0.4f, 0.841f, 0.0f, 2, 2, 0.5f,
        SCE_FACE_DETECT_RESULT_NORMAL,
        detectResult, 16, &numFace,
        workMemory, workMemorySize);
```

See Also

```
SceFaceDetectionDictPtr,sceFaceDetection(),sceFaceDetectionEx(),
sceFaceDetectionLocal()
```

SCE_FACE_DETECT_ROLL_DICT

Face detection dictionary data file (roll rotation support)

Definition

```
#include <libface.h>
#define SCE_FACE_DETECT_ROLL_DICT "face_detect_roll.fdt"
#define SCE_FACE_DETECT_ROLL_DICT_SIZE (78648)
```

Description

These constants represent the filename and size (in bytes) of dictionary data used in face detection.

The *detectDictPtr* argument of the face detection execution functions `sceFaceDetection()`, `sceFaceDetectionEx()` and `sceFaceDetectionLocal()` should be set with a pointer to the memory area where this file was read.

This dictionary detects frontal faces and tilted faces (faces with a roll angle).

Notes

Although this dictionary can detect faces that are facing forward or tilted (with a roll angle), it cannot detect faces that are turned sideways (with a yaw angle) or upwards/downwards (with a pitch angle). The processing time is medium fast.

Choose the type of face detection dictionary that is best suited for the application.

Example

```
SceFaceDetectionDictPtr detectDict =
        (SceFaceDetectionDictPtr)malloc(SCE_FACE_DETECT_ROLL_DICT_SIZE);
SceUID fd = sceIoOpen("host0:"SCE_FACE_DETECT_ROLL_DICT, SCE_O_RDONLY, 0);
sceIoRead(fd, detectDict, SCE_FACE_DETECT_ROLL_DICT_SIZE);
sceIoClose(fd);

ret = sceFaceDetection(
        yImg, 320, 240, 320,
        detectDict,
        0.4f, 0.841f, 0.0f, 2, 2, 0.5f,
        SCE_FACE_DETECT_RESULT_NORMAL,
        detectResult, 16, &numFace,
        workMemory, workMemorySize);
```

See Also

```
SceFaceDetectionDictPtr,sceFaceDetection(),sceFaceDetectionEx(),
sceFaceDetectionLocal()
```

SCE_FACE_DETECT_YAW_DICT

Face detection dictionary data file (yaw rotation support)

Definition

```
#include <libface.h>
#define SCE_FACE_DETECT_YAW_DICT "face_detect_yaw.fdt"
#define SCE_FACE_DETECT_YAW_DICT_SIZE (134752)
```

Description

These constants represent the filename and size (in bytes) of dictionary data used in face detection.

The *detectDictPtr* argument of the face detection execution functions `sceFaceDetection()`, `sceFaceDetectionEx()` and `sceFaceDetectionLocal()` should be set with a pointer to the memory area where this file was read.

This dictionary detects frontal faces and faces that are turned sideways (faces with a yaw angle).

Notes

Although this dictionary can detect faces that are facing forward or turned sideways (with a yaw angle), it cannot detect faces that are tilted (with a roll angle) or turned upwards/downwards (with a pitch angle). The processing time is medium fast.

Choose the type of face detection dictionary that is best suited for the application.

Example

```
SceFaceDetectionDictPtr detectDict =
        (SceFaceDetectionDictPtr)malloc(SCE_FACE_DETECT_YAW_DICT_SIZE);
SceUID fd = sceIoOpen("host0:"SCE_FACE_DETECT_YAW_DICT, SCE_O_RDONLY, 0);
sceIoRead(fd, detectDict, SCE_FACE_DETECT_YAW_DICT_SIZE);
sceIoClose(fd);

ret = sceFaceDetection(
        yImg, 320, 240, 320,
        detectDict,
        0.4f, 0.841f, 0.0f, 2, 2, 0.5f,
        SCE_FACE_DETECT_RESULT_NORMAL,
        detectResult, 16, &numFace,
        workMemory, workMemorySize);
```

See Also

```
SceFaceDetectionDictPtr,sceFaceDetection(),sceFaceDetectionEx(),
sceFaceDetectionLocal()
```

SCE_FACE_DETECT_PITCH_DICT

Face detection dictionary data file (pitch rotation support)

Definition

```
#include <libface.h>
#define SCE_FACE_DETECT_PITCH_DICT "face_detect_pitch.fdt"
#define SCE_FACE_DETECT_PITCH_DICT_SIZE (124728)
```

Description

These constants represent the filename and size (in bytes) of dictionary data used in face detection.

The *detectDictPtr* argument of the face detection execution functions `sceFaceDetection()`, `sceFaceDetectionEx()` and `sceFaceDetectionLocal()` should be set with a pointer to the memory area where this file was read.

This dictionary detects frontal faces and faces that are turned upwards/downwards (faces with a pitch angle).

Notes

Although this dictionary can detect faces that are facing forward or turned upwards/downwards (with a pitch angle), it cannot detect faces that are turned sideways (with a yaw angle) or tilted (with a roll angle). The processing time is medium fast.

Choose the type of face detection dictionary that is best suited for the application.

Examples

```
SceFaceDetectionDictPtr detectDict =
        (SceFaceDetectionDictPtr)malloc(SCE_FACE_DETECT_PITCH_DICT_SIZE);
SceUID fd = sceIoOpen("host0:"SCE_FACE_DETECT_PITCH_DICT, SCE_O_RDONLY, 0);
sceIoRead(fd, detectDict, SCE_FACE_DETECT_PITCH_DICT_SIZE);
sceIoClose(fd);

ret = sceFaceDetection(
        yImg, 320, 240, 320,
        detectDict,
        0.4f, 0.841f, 0.0f, 2, 2, 0.5f,
        SCE_FACE_DETECT_RESULT_NORMAL,
        detectResult, 16, &numFace,
        workMemory, workMemorySize);
```

See Also

`SceFaceDetectionDictPtr`, `sceFaceDetection()`, `sceFaceDetectionEx()`, `sceFaceDetectionLocal()`

SCE_FACE_DETECT_ROLL_YAW_DICT

Face detection dictionary data file (roll and yaw rotation support)

Definition

```
#include <libface.h>
#define SCE_FACE_DETECT_ROLL_YAW_DICT "face_detect_roll_yaw.fdt"
#define SCE_FACE_DETECT_ROLL_YAW_DICT_SIZE (186444)
```

Description

These constants represent the filename and size (in bytes) of dictionary data used in face detection.

The *detectDictPtr* argument of the face detection execution functions `sceFaceDetection()`, `sceFaceDetectionEx()` and `sceFaceDetectionLocal()` should be set with a pointer to the memory area where this file was read.

This dictionary detects frontal faces, tilted faces (faces with a roll angle), and faces that are turned sideways (faces with a yaw angle).

Notes

Although this dictionary supports a wide angle range of detectable faces, the processing time is longer than for the other dictionaries described earlier. It does not support detection of faces turned at a large upward or downward angle and cannot output an accurate pitch angle.

Choose the type of face detection dictionary that is best suited for the application.

Example

```
SceFaceDetectionDictPtr detectDict =
        (SceFaceDetectionDictPtr)malloc(SCE_FACE_DETECT_ROLL_YAW_DICT_SIZE)
;
SceUID fd = sceIoOpen("host0:"SCE_FACE_DETECT_ROLL_YAW_DICT, SCE_O_RDONLY, 0);
sceIoRead(fd, detectDict, SCE_FACE_DETECT_ROLL_YAW_DICT_SIZE);
sceIoClose(fd);

ret = sceFaceDetection(
        yImg, 320, 240, 320,
        detectDict,
        0.4f, 0.841f, 0.0f, 2, 2, 0.5f,
        SCE_FACE_DETECT_RESULT_NORMAL,
        detectResult, 16, &numFace,
        workMemory, workMemorySize);
```

See Also

`SceFaceDetectionDictPtr`, `sceFaceDetection()`, `sceFaceDetectionEx()`, `sceFaceDetectionLocal()`

SCE_FACE_DETECT_ROLL_YAW_PITCH_DICT

Face detection dictionary data file (roll, yaw, and pitch rotation support)

Definition

```
#include <libface.h>
#define SCE_FACE_DETECT_ROLL_YAW_PITCH_DICT "face_detect_roll_yaw_pitch.fdt"
#define SCE_FACE_DETECT_ROLL_YAW_PITCH_DICT_SIZE (267508)
```

Description

These constants represent the filename and size (in bytes) of dictionary data used in face detection.

The *detectDictPtr* argument of the face detection execution functions `sceFaceDetection()`, `sceFaceDetectionEx()` and `sceFaceDetectionLocal()` should be set with a pointer to the memory area where this file was read.

This dictionary also supports faces that are tilted, turned sideways or facing upward or downward (faces with a roll angle, a yaw angle and a pitch angle).

Notes

Although this dictionary supports the widest angle range of detectable faces, the processing time is the longest. It also supports the output of the pitch angle.

Choose the type of face detection dictionary that is best suited for the application.

Example

```
SceFaceDetectionDictPtr detectDict = (SceFaceDetectionDictPtr)malloc(
                                SCE_FACE_DETECT_ROLL_YAW_PITCH_DICT_SIZE);
SceUID fd = sceIoOpen("host0:"SCE_FACE_DETECT_ROLL_YAW_PITCH_DICT,
                                                        SCE_O_RDONLY, 0);
sceIoRead(fd, detectDict, SCE_FACE_DETECT_ROLL_YAW_PITCH_DICT_SIZE);
sceIoClose(fd);

ret = sceFaceDetection(
        yImg, 320, 240, 320,
        detectDict,
        0.4f, 0.841f, 0.0f, 2, 2, 0.5f,
        SCE_FACE_DETECT_RESULT_NORMAL,
        detectResult, 16, &numFace,
        workMemory, workMemorySize);
```

See Also

`SceFaceDetectionDictPtr`, `sceFaceDetection()`, `sceFaceDetectionEx()`, `sceFaceDetectionLocal()`

Set Precision of Face Positions for Face Detection Results Macros

Constants used for setting the precision of face positions for face detection results

Definition

```
#include <libface.h>
#define SCE_FACE_DETECT_RESULT_NORMAL 0
#define SCE_FACE_DETECT_RESULT_PRECISE 1
#define SCE_FACE_DETECT_RESULT_FAST 2
```

Description

These constants are used for setting the precision of face positions for face detection results.
They can be specified for the *resultPrecision* argument of sceFaceDetection().

Macro	Value	Description
SCE_FACE_DETECT_RESULT_NORMAL	0	Face detection result position precision setting for normal use
SCE_FACE_DETECT_RESULT_PRECISE	1	Precision setting in which position accuracy takes precedence over processing time
SCE_FACE_DETECT_RESULT_FAST	2	Precision setting in which speed takes precedence over position accuracy

See Also

sceFaceDetection(),sceFaceDetectionEx(),sceFaceDetectionLocal()

Set Face Detection Ending Macros

Constants used for setting the end of face detection processing

Definition

```
#include <libface.h>
#define SCE_FACE_DETECT_SEARCH_ALL_FACE 0
#define SCE_FACE_DETECT_SEARCH_FACE_NUM_LIMIT 1
```

Description

These constants are used to set when face detection processing should end.

Specify one of the following values to the *searchType* member of SceFaceDetectionParam.

Macro	Value	Description
SCE_FACE_DETECT_SEARCH_ALL_FACE	0	Detect all faces in the input image
SCE_FACE_DETECT_SEARCH_FACE_NUM_LIMIT	1	Detect faces in the input image for the number of faces specified in the *resultFaceArraySize* argument of sceFaceDetectionEx() in order starting with the largest face

When SCE_FACE_DETECT_SEARCH_ALL_FACE is set to *searchType*, detection processing will be carried out for all faces in the input image. When SCE_FACE_DETECT_SEARCH_FACE_NUM_LIMIT is set to *searchType*, face detection processing will end when the number of faces specified in the *resultFaceArraySize* argument of sceFaceDetectionEx() is detected in the input image in order starting with the largest face. It may be possible to complete face detection processing in high speed by setting a small value for *resultFaceArraySize* as appropriate.

See Also

SceFaceDetectionParam, sceFaceDetectionEx()

Parts Detection and Detailed Parts Detection APIs

SceFacePartsDictPtr

Parts detection dictionary data pointer type

Definition

```
#include <libface.h>
typedef void *SceFacePartsDictPtr;
```

Description

This defined type is used as a pointer to dictionary data used in normal and detailed parts detection. It is used for an argument type of the function `sceFaceParts()` and `sceFaceAllParts()`.

See Also

`sceFaceParts()`,`sceFaceAllParts()`

SceFaceShapeDictPtr

Shape correction dictionary data pointer type

Definition

```
#include <libface.h>
typedef void *SceFaceShapeDictPtr;
```

Description

This defined type is used as a pointer to the shape correction dictionary data used in detailed parts detection.

It is used for an argument type of the function `sceFaceAllParts()`.

See Also

`sceFaceAllParts()`

SceFacePartsCheckDictPtr

Parts result validity check dictionary data pointer type

Definition

```
#include <libface.h>
typedef void *SceFacePartsCheckDictPtr;
```

Description

This defined type is used as a pointer to dictionary data used in parts result validity check.
It is used for an argument type of the function `sceFacePartsResultCheck()`.

See Also

`sceFacePartsResultCheck()`

SceFacePartsResult

Parts detection result

Definition

```
#include <libface.h>
typedef struct SceFacePartsResult {
        unsigned int partsId;
        float partsX;
        float partsY;
        float score;
} SceFacePartsResult;
```

Members

partsId	Type of detected part
partsX	x position of detected part
partsY	y position of detected part
score	Score of detected part

Description

This data type is output for parts detection results.

When `sceFaceParts()` or `sceFaceAllParts()` is called, detected parts information is written as an array of this structure in the area that was set in the *resultPartsArray* argument.

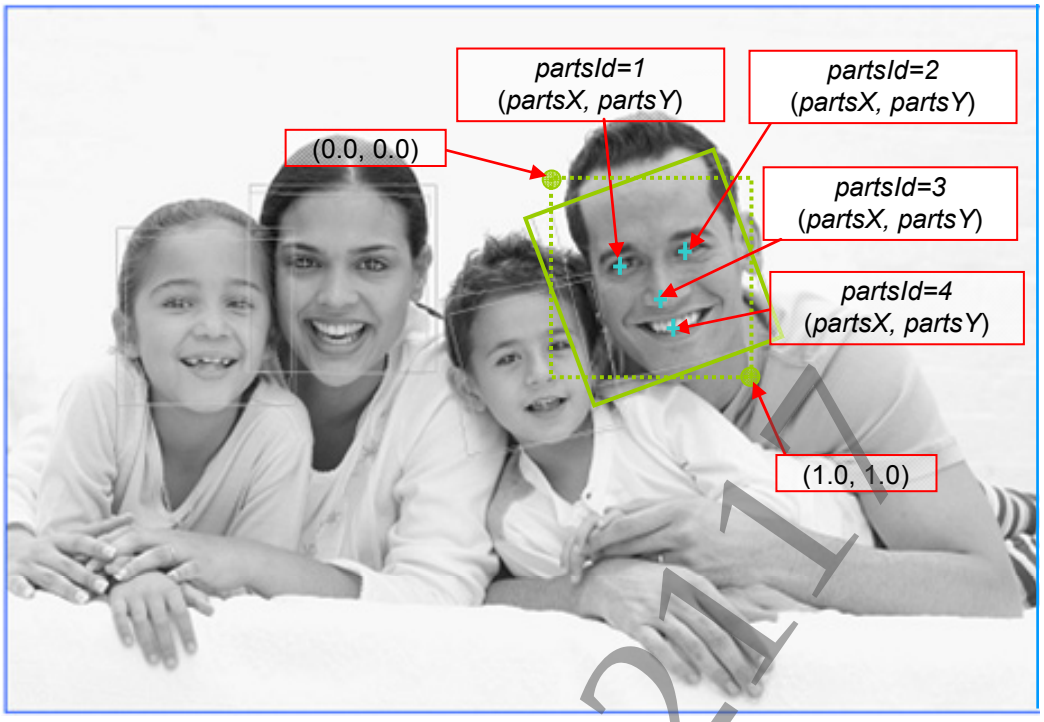
For *partsId*, a value indicated in "Part ID Macros for Parts Detection" will be stored for `sceFaceParts()` and a value indicated in "Part ID Macros for Detailed Part Detection" will be stored for `sceFaceAllParts()`.

partsX and *partsY* are represented in a coordinate system in which the upper left corner of the face region is set to (0.0, 0.0) and the lower right corner is set to (1.0, 1.0).

The following code can be used to convert to coordinates in the input image.

```
SceFaceDetectionResult face;
SceFacePartsResult parts;
int parts_x_of_image = (face.faceX + face.faceW * parts.partsX) * image_width;
int parts_y_of_image = (face.faceY + face.faceH * parts.partsY) * image_height;
```

Figure 5　Parts Detection Result



See Also

`sceFaceParts()`, `sceFaceAllParts()`, `sceFaceEstimatePoseRegion()`, `sceFaceAttribute()`

SceFacePose

Face pose based on parts detection result

Definition

```
#include <libface.h>
typedef struct SceFacePose {
        float faceRoll;
        float facePitch;
        float faceYaw;
} SceFacePose;
```

Members

faceRoll	Planar (roll axis) rotation angle of face in radians
facePitch	Upward or vertical (pitch axis) rotation angle of face in radians
faceYaw	Sideways or horizontal (yaw axis) rotation angle of face in radians

Description

This structure represents the face pose (rotation angle of the face), which is calculated based on the positions of the right eye, left eye, nose, and mouth.

When all four parts (right eye, left eye, nose and mouth) are detected by `sceFaceParts()`, and the parts detection results are entered in `sceFaceEstimatePoseRegion()`, then the face pose and the face region will be calculated and this structure type data will be written to the area set by the *resultFacePose* argument of `sceFaceEstimatePoseRegion()`.

See Also

`sceFaceParts()`, `sceFaceEstimatePoseRegion()`

SceFaceRegion

Face region based on parts detection result

Definition

```
#include <libface.h>
typedef struct SceFaceRegion {
        float faceRegionX[4];
        float faceRegionY[4];
} SceFaceRegion;
```

Members

faceRegionX	Array of x-coordinate of face region around top-left, top-right, bottom-right, and bottom-left.
faceRegionY	Array of y-coordinate of face region around top-left, top-right, bottom-right, and bottom-left.

Description

This structure represents the face region, which is calculated based on the positions of the right eye, left eye, nose, and mouth.

When all four parts (right eye, left eye, nose, and mouth) are detected by `sceFaceParts()`, and the parts detection results are entered in `sceFaceEstimatePoseRegion()`, then the face region will be calculated and this structure type data will be written to the area set by the *`resultFaceRegion`* argument of `sceFaceEstimatePoseRegion()`.

See Also

`sceFaceParts()`, `sceFaceEstimatePoseRegion()`

sceFaceParts

Execute parts detection

Definition

```
#include <libface.h>
int sceFaceParts(
        const unsigned char *imgPtr,
        int width,
        int height,
        int rowstride,
        const SceFacePartsDictPtr partsDictPtr,
        int xScanStep,
        int yScanStep,
        const SceFaceDetectionResult *detectedFace,
        SceFacePartsResult resultPartsArray[],
        int resultPartsArraySize,
        int *resultPartsNum,
        void* workMemory,
        int workMemorySize
)
```

Arguments

imgPtr	Pointer to input image (8-bit grayscale)
width	Input image width [pixels]
height	Input image height [pixels]
rowstride	Input image data width
partsDictPtr	Pointer to parts detection dictionary data
xScanStep	Parts detection window horizontal shift amount [pixels] (recommended value: 1)
yScanStep	Parts detection window vertical shift amount [pixels] (recommended value: 1)
detectedFace	Pointer to face detection result
resultPartsArray	Pointer to parts detection result output area
resultPartsArraySize	Number of array elements in *resultPartsArray*
resultPartsNum	Number of detected parts
workMemory	Pointer to work buffer
workMemorySize	Size of work buffer

Return Values

Returns SCE_OK(0) if processing is successful.

Returns one of the following error codes (a negative value) for errors.

Value	(Hexadecimal)	Description
SCE_FACE_ERROR_NO_MEMORY	0x808B0001	Not enough memory to run, or *workMemory* is NULL
SCE_FACE_ERROR_INVALID_PARAM	0x808B0002	Parameter is invalid
SCE_FACE_ERROR_INVALID_DICT	0x808B0003	Parts detection dictionary data is invalid, or *partsDictPtr* is NULL
SCE_FACE_ERROR_OUT_OF_RANGE	0x808B0005	Face angle is out of range

Description

This function detects the eyes, nose, and mouth parts from the face region that was obtained by face detection.

Generally, the input image that was used in face detection processing should be used directly for the input image that is set for *imgPtr*, and the image should not be updated or cleared until parts detection processing is completely finished and this function returns.

As long as the image does not change significantly - such as - in a movie, it does not necessarily have to be completely the same. However, since search processing is only performed over a limited range using the face regions and face angles that were obtained from the face detection results, the likelihood of detection errors occurring will increase for greater variations in image content.

For *partsDictPtr*, set a pointer to the memory area where the parts detection dictionary data file SCE_FACE_PARTS_ROLL_DICT or SCE_FACE_PARTS_ROLL_YAW_DICT was read.

Except for when a lightweight dictionary is selected to give precedence to faster speed over detection precision, these dictionaries should basically be matched with the face detection dictionary type.

For *detectedFace*, set a pointer to the data for a single face from the face detection results.

For *resultPartsArray*, set a pointer to the SceFacePartsResult array for storing the parts detection results. Normally, the number of elements in this SceFacePartsResult array is SCE_FACE_PARTS_NUM_MAX(4) and SCE_FACE_PARTS_NUM_MAX(4) should be set for *resultPartsArraySize*.

If parts detection is executed normally, SCE_FACE_PARTS_NUM_MAX(4) sets of parts detection result data will be output to the memory area set for *resultPartsArray* and the number of detected parts will be returned in *resultPartsNum*.

For *workMemory*, set the pointer to the allocated memory whose size is calculated by sceFacePartsGetWorkingMemorySize() or more.

For *workMemorySize*, set the size of *workMemory*.

For any parts that could not be detected normally, the *partsId* member of the SceFacePartsResult structure of the detection results will be SCE_FACE_PARTS_ID_UNDEF.

Notes

During the parts detection processing, if the temporary storage area used internally for intermediate results becomes insufficient, the detection precision that is output may end up getting slightly worse.

If the input image shooting conditions were not ideal and the face parts are dark due to backlighting or insufficient light or if, on the other hand, the face is overexposed and is whitened out, or if the image contains a lot of noise, the face detection rate may worsen even if processing is performed normally by the program. Adjust the input image quality as needed before calling this function.

Parts detection may also fail even if the causes mentioned above do not occur. This can happen when the face angle is large such as when the face is turned almost sideways so that one eye is barely visible or when part of the face is hidden by hair, eyeglasses, or some other object.

Note that in some situations, a program that runs normally may end up detecting an invalid position, and in that case, it will be impossible to determine whether it was an invalid detection from the return value, the number of parts detected, or the contents of the *partsId* member of the SceFacePartsResult structure in the detection results. Call sceFacePartsResultCheck() to check the validity of parts detection result.

Example

```
/* Read face recognition dictionary data file */
unsigned char detectDict[SCE_FACE_DETECT_FRONTAL_DICT_SIZE];
unsigned char partsDict[SCE_FACE_PARTS_ROLL_DICT_SIZE];
SceUID fd;
fd = sceIoOpen("host0:"SCE_FACE_DETECT_FRONTAL_DICT, SCE_O_RDONLY, 0);
sceIoRead(fd, detectDict, SCE_FACE_DETECT_FRONTAL_DICT_SIZE);
sceIoClose(fd);
fd = sceIoOpen("host0:"SCE_FACE_PARTS_ROLL_DICT, SCE_O_RDONLY, 0);
sceIoRead(fd, partsDict, SCE_FACE_PARTS_ROLL_DICT_SIZE);
sceIoClose(fd);

unsigned char yImg[320*240];
SceFaceDetectionResult detectResult[16];
SceFacePartsResult resultPartsArray[SCE_FACE_PARTS_NUM_MAX];
int i, j, numFace, numParts, ret;

/* Allocate work buffer for face detection */
int workSize = sceFaceDetectionGetWorkingMemorySize(320, 240, 320, detectDict);
void *workPtr = malloc(workSize);

/* Allocate work buffer for parts detection */
int workPartsSize = sceFacePartsGetWorkingMemorySize(320, 240, 320, partsDict);
void *workPartsPtr = malloc(workPartsSize);

/* Execute face detection */
ret = sceFaceDetection(
        yImg, 320, 240, 320,
        detectDict,
        0.4f, 0.841f, 0.0f, 2, 2, 0.5f,
        SCE_FACE_DETECT_RESULT_NORMAL,
        detectResult, 16, &numFace,
        workPtr, workSize);
if (ret != SCE_OK) {
        printf("sceFaceDetection() failed! (0x%08x)\n", ret);
        return;
}
for (i = 0; i < numFace; i++) {
        /* Execute parts detection */
        ret = sceFaceParts(
              yImg, 320, 240, 320,
              partsDict,
              1, 1, &detectResult[i],
              resultPartsArray, SCE_FACE_PARTS_NUM_MAX, &numParts,
              workPartsPtr, workPartsSize);
        if (ret != SCE_OK) {
              printf("sceFaceParts() failed! (0x%08x)\n", ret);
              return;
        }
        for (j = 0; j < numParts; j++) {
              printf("parts[%d] id = %d x = %f y = %f\n", j,
                     resultPartsArray[j].partsId,
                     resultPartsArray[j].partsX,
                     resultPartsArray[j].partsY);
        }
}
```

See Also

SceFacePartsDictPtr, SCE_FACE_PARTS_ROLL_DICT, SCE_FACE_PARTS_ROLL_YAW_DICT, SceFaceDetectionResult, SceFacePartsResult, sceFacePartsGetWorkingMemorySize()

sceFacePartsEx

Execute parts detection with feature to correct invalid parts detection

Definition

```
#include <libface.h>
int sceFacePartsEx(
        const unsigned char *imgPtr,
        int width,
        int height,
        int rowstride,
        const SceFacePartsDictPtr partsDictPtr,
        const SceFacePartsCheckDictPtr partsCheckDictPtr,
        int xScanStep,
        int yScanStep,
        const SceFaceDetectionResult *detectedFace,
        SceFacePartsResult resultPartsArray[],
        int resultPartsArraySize,
        int *resultPartsNum,
        void* workMemory,
        int workMemorySize
)
```

Arguments

imgPtr	Pointer to input image (8-bit grayscale)
width	Input image width [pixels]
height	Input image height [pixels]
rowstride	Input image data width
partsDictPtr	Pointer to parts detection dictionary data
partsCheckDictPtr	Pointer to parts detection result validity evaluation dictionary data
xScanStep	Parts detection window horizontal shift amount [pixels] (recommended value: 1)
yScanStep	Parts detection window vertical shift amount [pixels] (recommended value: 1)
detectedFace	Pointer to face detection result
resultPartsArray	Pointer to parts detection result output area
resultPartsArraySize	Number of array elements in *resultPartsArray*
resultPartsNum	Number of detected parts
workMemory	Pointer to work buffer
workMemorySize	Size of work buffer

Return Values

Returns SCE_OK(0) if processing is successful.

Returns one of the following error codes (a negative value) for errors.

Value	(Hexadecimal)	Description
SCE_FACE_ERROR_NO_MEMORY	0x808B0001	Not enough memory to run, or *workMemory* is NULL
SCE_FACE_ERROR_INVALID_PARAM	0x808B0002	Parameter is invalid
SCE_FACE_ERROR_INVALID_DICT	0x808B0003	Parts detection dictionary data is invalid Either *partsDictPtr* is NULL or the parts detection result validity evaluation dictionary data is invalid
SCE_FACE_ERROR_OUT_OF_RANGE	0x808B0005	Face angle is out of range

Description

This function detects the eyes, nose, and mouth parts from the face region that was obtained by face detection. Using the parts detection result validity evaluation dictionary, the function corrects invalid parts detection to output the most valid detection results for eyes, nose, and mouth parts.

Explanation of arguments that are the same as `sceFaceParts()` are omitted here. Refer to the Description of `sceFaceParts()` for details.

For *`partsCheckDictPtr`*, set the pointer to the memory area to which the parts detection result validity evaluation dictionary data file `SCE_FACE_PARTS_CHECK_DICT` was read. When specifying NULL to this argument, this function will behave in the same manner as `sceFaceParts()`.

Notes

Since this function uses the parts detection result validity evaluation dictionary to output the most valid detection results for eyes, nose, and mouth parts, processing time takes longer than `sceFaceParts()`.

When no combination of parts detection results is evaluated as valid, *`resultPartsNum`* stores 0. Since the validity of parts detection result is not evaluated in `sceFaceParts()`, parts detection results with invalid results may be output to *`resultPartsArray`* and *`resultPartsNum`* in the above case when using `sceFaceParts()`.

Examples

```
/* Read face recognition dictionary data file */
unsigned char detectDict[SCE_FACE_DETECT_FRONTAL_DICT_SIZE];
unsigned char partsDict[SCE_FACE_PARTS_ROLL_DICT_SIZE];
unsigned char partsCheckDict[SCE_FACE_PARTS_ROLL_DICT_SIZE];
SceUID fd;
fd = sceIoOpen("host0:"SCE_FACE_DETECT_FRONTAL_DICT, SCE_O_RDONLY, 0);
sceIoRead(fd, detectDict, SCE_FACE_DETECT_FRONTAL_DICT_SIZE);
sceIoClose(fd);
fd = sceIoOpen("host0:"SCE_FACE_PARTS_ROLL_DICT, SCE_O_RDONLY, 0);
sceIoRead(fd, partsDict, SCE_FACE_PARTS_ROLL_DICT_SIZE);
sceIoClose(fd);
fd = sceIoOpen("host0:"SCE_FACE_PARTS_CHECK_DICT, SCE_O_RDONLY, 0);
sceIoRead(fd, partsCheckDict, SCE_FACE_PARTS_CHECK_DICT_SIZE);
sceIoClose(fd);

unsigned char yImg[320*240];
SceFaceDetectionResult detectResult[16];
SceFacePartsResult resultPartsArray[SCE_FACE_PARTS_NUM_MAX];
int i, j, numFace, numParts, ret;

/* Allocate work buffer for face detection */
int workSize = sceFaceDetectionGetWorkingMemorySize(320, 240, 320, detectDict);
void *workPtr = malloc(workSize);

/* Allocate work buffer for parts detection */
int workPartsSize = sceFacePartsGetWorkingMemorySize(320, 240, 320, partsDict);
void *workPartsPtr = malloc(workPartsSize);

/* Execute face detection */
ret = sceFaceDetection(
        yImg, 320, 240, 320,
        detectDict,
        0.4f, 0.841f, 0.0f, 2, 2, 0.5f,
        SCE_FACE_DETECT_RESULT_NORMAL,
```

```
            detectResult, 16, &numFace
            workPtr, workSize);
    if (ret != SCE_OK) {
            printf("sceFaceDetection() failed! (0x%08x)\n", ret);
            return;
    }
    for (i = 0; i < numFace; i++) {
            /* Execute parts detection */
            ret = sceFacePartsEx(
                  yImg, 320, 240, 320,
                  partsDict, partsCheckDict,
                  1, 1, &detectResult[i],
                  resultPartsArray, SCE_FACE_PARTS_NUM_MAX, &numParts,
                  workPartsPtr, workPartsSize);
            if (ret != SCE_OK) {
                  printf("sceFaceParts() failed! (0x%08x)\n", ret);
                  return;
            }
            for (j = 0; j < numParts; j++) {
                  printf("parts[%d] id = %d x = %f y = %f\n", j,
                        resultPartsArray[j].partsId,
                        resultPartsArray[j].partsX,
                        resultPartsArray[j].partsY);
            }
    }
```

See Also

SceFacePartsDictPtr, SceFacePartsCheckDictPtr, SCE_FACE_PARTS_ROLL_DICT, SCE_FACE_PARTS_ROLL_YAW_DICT, SCE_FACE_PARTS_CHECK_DICT, SceFaceDetectionResult, SceFacePartsResult, sceFacePartsGetWorkingMemorySize()

sceFaceAllParts

Execute detailed parts detection

Definition

```
#include <libface.h>
int sceFaceAllParts(
        const unsigned char *imgPtr,
        int width,
        int height,
        int rowstride,
        const SceFacePartsDictPtr partsDictPtr,
        const SceFaceShapeDictPtr shapeDictPtr,
        int xScanStep,
        int yScanStep,
        const SceFaceDetectionResult *detectedFace,
        SceFacePartsResult resultPartsArray[],
        int resultPartsArraySize,
        int *resultPartsNum,
        void* workMemory,
        int workMemorySize
)
```

Arguments

imgPtr	Pointer to input image (8-bit grayscale)
width	Input image width [pixels]
height	Input image height [pixels]
rowstride	Input image data width
partsDictPtr	Pointer to detailed parts detection dictionary data
shapeDictPtr	Pointer to shape correction dictionary data
xScanStep	Detailed parts detection window horizontal shift amount [pixels] (recommended value: 1)
yScanStep	Detailed parts detection window vertical shift amount [pixels] (recommended value: 1)
detectedFace	Pointer to face detection result
resultPartsArray	Pointer to detailed parts detection result output area
resultPartsArraySize	Number of array elements in *resultPartsArray*
resultPartsNum	Number of detected parts
workMemory	Pointer to work buffer
workMemorySize	Size of work buffer

Return Values

Returns SCE_OK(0) if processing is successful.

Returns one of the following error codes (a negative value) for errors.

Value	(Hexadecimal)	Description
SCE_FACE_ERROR_NO_MEMORY	0x808B0001	Not enough memory to run, or *workMemory* is NULL
SCE_FACE_ERROR_INVALID_PARAM	0x808B0002	Parameter is invalid
SCE_FACE_ERROR_INVALID_DICT	0x808B0003	Detailed parts detection dictionary data is invalid, or *partsDictPtr* is NULL
SCE_FACE_ERROR_OUT_OF_RANGE	0x808B0005	Face angle is out of range

Description

This function detects detailed parts from the face area obtained in face detection

Generally, the input image that was used in face detection processing should be used directly for the input image that is set for *imgPtr*, and the image should not be updated or cleared until detailed parts detection processing is completely finished and this function returns.

As long as the image does not change significantly - such as - in a movie, it does not necessarily have to be completely the same. However, since search processing is only performed over a limited range using the face regions and face angles that were obtained from the face detection results, the likelihood of detection errors occurring will increase for greater variations in image content.

For *partsDictPtr*, set a pointer to the memory area where the detailed parts detection dictionary data files SCE_FACE_ALLPARTS_DICT was read.

For *shapeDictPtr*, set a pointer to the memory area where the shape correction detection dictionary data files SCE_FACE_ALLPARTS_SHAPE_DICT was read. If shape correction is not required, set NULL to *shapeDictPtr*.

For *detectedFace*, set a pointer to the data for a single face from the face detection results.

For *resultPartsArray*, set a pointer to the SceFacePartsResult array for storing the detailed parts detection results. Normally, the number of elements in this SceFacePartsResult array is SCE_FACE_ALLPARTS_NUM_MAX(55) and SCE_FACE_ALLPARTS_NUM_MAX(55) should be set for *resultPartsArraySize*.

If detailed parts detection is executed normally, SCE_FACE_ALLPARTS_NUM_MAX(55) sets of detailed parts detection result data will be output to the memory area set for *resultPartsArray* and the number of detected parts will be returned in *resultPartsNum*.

For *workMemory*, set the pointer to the allocated memory whose size is calculated by sceFaceAllPartsGetWorkingMemorySize() or more.

For *workMemorySize*, set the size of *workMemory*.

For any parts that could not be detected normally, the *partsId* member of the SceFacePartsResult structure of the detection results will be SCE_FACE_PARTS_ID_UNDEF.

Notes

During the detailed parts detection processing, if the temporary storage area used internally for intermediate results becomes insufficient, the detection precision that is output may end up getting slightly worse.

If the input image shooting conditions were not ideal and the face parts are dark due to backlighting or insufficient light or if, on the other hand, the face is overexposed and is whitened out, or if the image contains a lot of noise, the detection rate may worsen even if processing is performed normally by the program. Adjust the input image quality as needed before calling this function.

Detailed parts detection may also fail even if the causes mentioned above do not occur. This can happen when the face angle is large such as when the face is turned almost sideways so that one eye is barely visible or when part of the face is hidden by hair, eyeglasses or some other object.

Note that in some situations, a program that runs normally may end up detecting an invalid position, and in that case, it will be impossible to determine whether it was an invalid detection from the return value, the number of parts detected, or the contents of the *partsId* member of the SceFacePartsResult structure in the detection results.

Example

```
/* Read face recognition dictionary data file */
unsigned char detectDict[SCE_FACE_DETECT_FRONTAL_DICT_SIZE];
unsigned char allpartsDict[SCE_FACE_ALLPARTS_DICT_SIZE];
unsigned char shapeDict[SCE_FACE_ALLPARTS_SHAPE_DICT_SIZE];
```

```
SceUID fd;
fd = sceIoOpen("host0:"SCE_FACE_DETECT_FRONTAL_DICT, SCE_O_RDONLY, 0);
sceIoRead(fd, detectDict, SCE_FACE_DETECT_FRONTAL_DICT_SIZE);
sceIoClose(fd);
fd = sceIoOpen("host0:"SCE_FACE_ALLPARTS_DICT, SCE_O_RDONLY, 0);
sceIoRead(fd, allpartsDict, SCE_FACE_ALLPARTS_DICT_SIZE);
sceIoClose(fd);
fd = sceIoOpen("host0:"SCE_FACE_ALLPARTS_SHAPE_DICT, SCE_O_RDONLY, 0);
sceIoRead(fd, shapeDict, SCE_FACE_ALLPARTS_SHAPE_DICT_SIZE);
sceIoClose(fd);

unsigned char yImg[320*240];
SceFaceDetectionResult detectResult[16];
SceFacePartsResult resultAllPartsArray[SCE_FACE_ALLPARTS_NUM_MAX];
int i, j, numFace, numAllParts, ret;

/* Allocate work buffer for face detection */
int workSize = sceFaceDetectionGetWorkingMemorySize(320, 240, 320, detectDict);
void *workPtr = malloc(workSize);

/* Allocate work buffer for detailed parts detection */
int workAllPartsSize = sceFaceAllPartsGetWorkingMemorySize(
        320, 240, 320, allpartsDict);
void *workAllPartsPtr = malloc(workAllPartsSize);

/* Execute face detection */
ret = sceFaceDetection(
        yImg, 320, 240, 320,
        detectDict,
        0.4f, 0.841f, 0.0f, 2, 2, 0.5f,
        SCE_FACE_DETECT_RESULT_NORMAL,
        detectResult, 16, &numFace,
        workPtr, workSize);
if (ret != SCE_OK) {
        printf("sceFaceDetection() failed! (0x%08x)\n", ret);
        return;
}
for (i = 0; i < numFace; i++) {
        /* Execute detailed parts detection */
        ret = sceFaceAllParts(
              yImg, 320, 240, 320,
              allpartsDict,
              shapeDict,
              1, 1, &detectResult[i],
              resultAllPartsArray, SCE_FACE_ALLPARTS_NUM_MAX, &numAllParts,
              workAllPartsPtr, workAllPartsSize);
        if (ret != SCE_OK) {
              printf("sceFaceAllParts() failed! (0x%08x)\n", ret);
              return;
        }
        for (j = 0; j < numAllParts; j++) {
              printf("parts[%d] id = %d x = %f y = %f\n", j,
                    resultAllPartsArray[j].partsId,
                    resultAllPartsArray[j].partsX,
                    resultAllPartsArray[j].partsY);
        }
}
```

See Also

`SceFacePartsDictPtr`, `SCE_FACE_ALLPARTS_DICT`, `SceFaceDetectionResult`, `SceFacePartsResult`, `sceFaceAllPartsGetWorkingMemorySize()`

sceFaceEstimatePoseRegion

Estimate face pose and region from the positions of detected parts

Definition

```
#include <libface.h>
int sceFaceEstimatePoseRegion(
        int width,
        int height,
        const SceFaceDetectionResult *detectedFace,
        const SceFacePartsResult detectedPartsArray[],
        int detectedPartsNum,
        SceFacePose *resultFacePose,
        SceFaceRegion *resultFaceRegion
)
```

Arguments

width	Input image width [pixels]
height	Input image height [pixels]
detectedFace	Pointer to face detection result
detectedPartsArray	Pointer to parts detection result array
detectedPartsNum	Number of detected parts
resultFacePose	Pointer to facial pose estimation result output area
resultFaceRegion	Pointer to facial region estimation result output area

Return Values

Returns SCE_OK(0) if processing is successful.

Returns one of the following error codes (a negative value) if an error occurs.

Value	(Hexadecimal)	Description
SCE_FACE_ERROR_INVALID_PARAM	0x808B0002	Parameter is invalid
SCE_FACE_ERROR_IMPERF_PARTS	0x808B0004	Parts are incomplete

Description

This function calculates the face rotation angle and face region based on the positions of the right eye, left eye, nose, and mouth.

The calculation is performed from only the face and parts detection results without performing any image recognition.

The *faceRoll*, *facePitch*, and *faceYaw* members of the SceFacePose structure, which are output in *resultFacePose* are defined in a right-handed coordinate system as shown in Figure 3.

Notes

For this function to work, the input parts information must contain normal detection results of all four parts (left and right eyes, nose, and mouth). Otherwise, an SCE_FACE_ERROR_IMPERF_PARTS error will be returned.

Example

```
/* Read face recognition dictionary data file */
unsigned char detectDict[SCE_FACE_DETECT_FRONTAL_DICT_SIZE];
unsigned char partsDict[SCE_FACE_PARTS_ROLL_DICT_SIZE];
SceUID fd;
fd = sceIoOpen("host0:"SCE_FACE_DETECT_FRONTAL_DICT, SCE_O_RDONLY, 0);
sceIoRead(fd, detectDict, SCE_FACE_DETECT_FRONTAL_DICT_SIZE);
sceIoClose(fd);
fd = sceIoOpen("host0:"SCE_FACE_PARTS_ROLL_DICT, SCE_O_RDONLY, 0);
sceIoRead(fd, partsDict, SCE_FACE_PARTS_ROLL_DICT_SIZE);
sceIoClose(fd);

unsigned char yImg[320*240];
SceFaceDetectionResult detectResult[16];
SceFacePartsResult resultPartsArray[SCE_FACE_PARTS_NUM_MAX];
SceFacePose resultFacePose;
SceFaceRegion resultFaceRegion;
int i, numFace, numParts, ret;

/* Allocate work buffer for face detection */
int workSize = sceFaceDetectionGetWorkingMemorySize(320, 240, 320, detectDict);
void *workPtr = malloc(workSize);

/* Allocate work buffer for parts detection */
int workPartsSize = sceFacePartsGetWorkingMemorySize(320, 240, 320, partsDict);
void *workPartsPtr = malloc(workPartsSize);

/* Execute face detection */
ret = sceFaceDetection(
        detectDict,
        yImg, 320, 240, 320,
        0.4f, 0.841f, 0.0f, 2, 2, 0.5f,
        SCE_FACE_DETECT_RESULT_NORMAL,
        detectResult, 16, &numFace,
        workPtr, workSize);
if (ret != SCE_OK) {
        printf("sceFaceDetection() failed! (0x%08x)\n", ret);
        return;
}
for (i = 0; i < numFace; i++) {
        /* Execute parts detection */
        ret = sceFaceParts(
              yImg, 320, 240, 320,
              partsDict,
              1, 1, &detectResult[i],
              resultPartsArray, SCE_FACE_PARTS_NUM_MAX, &numParts,
              workPartsPtr, workPartsSize);
        if (ret != SCE_OK) {
              printf("sceFaceParts() failed! (0x%08x)\n", ret);
              return;
        }
        /* Estimate face angle and face region according to parts positions */
        ret = sceFaceEstimatePoseRegion(
              320, 240, &detectResult[i],
              resultPartsArray, numParts,
              &resultFacePose,
              &resultFaceRegion);
        if (ret != SCE_OK) {
              printf("sceFaceEstimatePoseRegion() failed! (0x%08x)\n", ret);
```

```
                return;
            }
            printf("roll = %f pitch = %f yaw = %f\n",
                resultFaceAngle.faceRoll,
                resultFaceAngle.facePitch,
                resultFaceAngle.faceYaw);
    }
```

See Also

SceFaceDetectionResult, SceFacePartsResult, SceFacePose

sceFacePartsResultCheck

Check the validity of parts detection result

Definition

```
#include <libface.h>
int sceFacePartsResultCheck(
        const SceFaceDetectionResult *detectedFace,
        const SceFacePartsResult detectedPartsArray[],
        int detectedPartsNum,
        const SceFacePartsCheckDictPtr partsCheckDictPtr,
        bool *good
)
```

Arguments

detectedFace	Pointer to face detection result
detectedPartsArray	Pointer to parts detection result array
detectedPartsNum	Number of detected parts
partsCheckDictPtr	Pointer to parts result validity check dictionary data
good	Validity flag of parts detection result check

Return Values

Returns SCE_OK(0) if processing is successful.

Returns one of the following error codes (a negative value) if an error occurs.

Value	(Hexadecimal)	Description
SCE_FACE_ERROR_INVALID_PARAM	0x808B0002	Parameter is invalid
SCE_FACE_ERROR_INVALID_DICT	0x808B0003	Parts result validity check dictionary data is invalid, or *partsCheckDictPtr* is NULL
SCE_FACE_ERROR_IMPERF_PARTS	0x808B0004	Parts are incomplete

Description

This function checks the validity of detected parts result: eyes, nose, and mouth.

For *detectedFace*, set a pointer to the data for a single face from the face detection results.

The parts detection results set in *detectedPartsArray* must contain normal detection results of all four parts (left and right eyes, nose, and mouth), and *detectedPartsNum* must be SCE_FACE_PARTS_NUM_MAX(4).

If the parts detection results that were set do not satisfy these conditions, an SCE_FACE_ERROR_IMPERF_PARTS error will be returned.

For *partsCheckDictPtr*, set a pointer to the memory area where the parts result validity check dictionary data file SCE_FACE_PARTS_CHECK_DICT was read.

Example

```
/* Read face recognition dictionary data file */
unsigned char detectDict[SCE_FACE_DETECT_FRONTAL_DICT_SIZE];
unsigned char partsDict[SCE_FACE_PARTS_ROLL_DICT_SIZE];
unsigned char partsCheckDict[SCE_FACE_PARTS_CHECK_DICT_SIZE];
SceUID fd;
fd = sceIoOpen("host0:"SCE_FACE_DETECT_FRONTAL_DICT, SCE_O_RDONLY, 0);
sceIoRead(fd, detectDict, SCE_FACE_DETECT_FRONTAL_DICT_SIZE);
sceIoClose(fd);
fd = sceIoOpen("host0:"SCE_FACE_PARTS_ROLL_DICT, SCE_O_RDONLY, 0);
sceIoRead(fd, partsDict, SCE_FACE_PARTS_ROLL_DICT_SIZE);
sceIoClose(fd);
fd = sceIoOpen("host0:"SCE_FACE_PARTS_CHECK_DICT, SCE_O_RDONLY, 0);
sceIoRead(fd, partsCheckDict, SCE_FACE_PARTS_CHECK_DICT_SIZE);
sceIoClose(fd);

unsigned char yImg[320*240];
SceFaceDetectionResult detectResult[16];
SceFacePartsResult resultPartsArray[SCE_FACE_PARTS_NUM_MAX];
int i, j, numFace, numParts, ret;

/* Allocate work buffer for face detection */
int workSize = sceFaceDetectionGetWorkingMemorySize(320, 240, 320, detectDict);
void *workPtr = malloc(workSize);

/* Allocate work buffer for parts detection */
int workPartsSize = sceFacePartsGetWorkingMemorySize(320, 240, 320, partsDict);
void *workPartsPtr = malloc(workPartsSize);

/* Execute face detection */
ret = sceFaceDetection(
        yImg, 320, 240, 320,
        detectDict,
        0.4f, 0.841f, 0.0f, 2, 2, 0.5f,
        SCE_FACE_DETECT_RESULT_NORMAL,
        detectResult, 16, &numFace,
        workPtr, workSize);
if (ret != SCE_OK) {
        printf("sceFaceDetection() failed! (0x%08x)\n", ret);
        return;
}
for (i = 0; i < numFace; i++) {
        /* Execute parts detection */
        ret = sceFaceParts(
              yImg, 320, 240, 320,
              partsDict,
              1, 1, &detectResult[i],
              resultPartsArray, SCE_FACE_PARTS_NUM_MAX, &numParts,
              workPartsPtr, workPartsSize);
        if (ret != SCE_OK) {
              printf("sceFaceParts() failed! (0x%08x)\n", ret);
              return;
        }
        for (j = 0; j < numParts; j++) {
              printf("parts[%d] id = %d x = %f y = %f\n", j,
                     resultPartsArray[j].partsId,
                     resultPartsArray[j].partsX,
                     resultPartsArray[j].partsY);
        }
```

```
            /* Check the validity of detected parts result */
            bool good;
            ret = sceFacePartsResultCheck(
                  &detectResult[i],
                  resultPartsArray, numParts,
                  partsCheckDict,
                  &good);
            printf("parts is: %s\n", good ? "good" : "ng!");
}
```

See Also

SceFacePartsCheckDictPtr, SCE_FACE_PARTS_CHECK_DICT, SceFaceDetectionResult, SceFacePartsResult

sceFacePartsGetWorkingMemorySize

Calculate the size of working memory for parts detection

Definition

```
#include <libface.h>
int sceFacePartsGetWorkingMemorySize(
        int width,
        int height,
        int rowstride,
        const SceFacePartsDictPtr partsDictPtr
)
```

Arguments

width	Input image width for parts detection [pixels]
height	Input image height for parts detection [pixels]
rowstride	Input image data width for parts detection
partsDictPtr	Pointer to parts detection dictionary data

Return Values

Returns the size of work buffer for parts detection.

Returns 0 for errors.

Description

This function calculates the size of the work buffer for parts detection.

Allocate the memory with the size returned by this function, then call `sceFaceParts()` with the pointer to the work buffer and its size to execute parts detection.

If this function returns 0, the parts detection dictionary data is invalid or parameters are invalid.

Notes

Work buffer for parts detection can be same buffer with the one for face detection, since the size of work buffer for parts detection is generally smaller than the one for the face detection (however, this can be done only when the input image that is used for parts detection is the same as the one used for face detection).

See Also

`sceFaceParts()`

sceFaceAllPartsGetWorkingMemorySize

Calculate the size of working memory for detailed parts detection

Definition

```
#include <libface.h>
int sceFaceAllPartsGetWorkingMemorySize(
        int width,
        int height,
        int rowstride,
        const SceFacePartsDictPtr partsDictPtr
)
```

Arguments

width	Input image width for detailed parts detection [pixels]
height	Input image height for detailed parts detection [pixels]
rowstride	Input image data width for detailed parts detection
partsDictPtr	Pointer to detailed parts detection dictionary data

Return Values

Returns the size of work buffer for detailed parts detection.

Returns 0 for errors.

Description

This function calculates the size of the work buffer for detailed parts detection.

Allocate the memory with the size returned by this function, then call `sceFaceAllParts()` with the pointer to the work buffer and its size to execute detailed parts detection.

If this function returns 0, the detailed parts detection dictionary data is invalid or parameters are invalid.

Notes

Work buffer for detailed parts detection can be same buffer with the one for face detection, since the size of work buffer for detailed parts detection is generally smaller than the one for face detection (however, this can be done only when the input image that is used for detailed parts detection is the same as the one used for face detection).

See Also

`sceFaceAllParts()`

SCE_FACE_PARTS_ROLL_DICT

Parts detection dictionary data file (roll rotation support)

Definition

```
#include <libface.h>
#define SCE_FACE_PARTS_ROLL_DICT "face_parts_roll.pdt"
#define SCE_FACE_PARTS_ROLL_DICT_SIZE (40644)
```

Description

These constants represent the filename and size (in bytes) of dictionary data used in parts detection.

The *partsDictPtr* argument of the parts detection execution function `sceFaceParts()` should be set with a pointer to the memory area where this file was read.

This dictionary supports frontal faces and tilted faces (faces with a roll angle).

Notes

Although the processing time for this dictionary is less than that for `SCE_FACE_PARTS_ROLL_YAW_DICT`, it will have more parts detection errors for a face that is turned sideways (face with a large yaw angle).

This dictionary is suitable when `SCE_FACE_DETECT_FRONTAL_DICT` or `SCE_FACE_DETECT_ROLL_DICT` was used as the face detection dictionary. Although it can also be used for face detection results that were obtained using another face detection dictionary, there may be more parts detection errors when the yaw angle of the detected face is large.

Example

```
SceFacePartsDictPtr partsDict =
        (SceFacePartsDictPtr)malloc(SCE_FACE_PARTS_ROLL_DICT_SIZE);
SceUID fd = sceIoOpen("host0:"SCE_FACE_PARTS_ROLL_DICT, SCE_O_RDONLY, 0);
sceIoRead(fd, partsDict, SCE_FACE_PARTS_ROLL_DICT_SIZE);
sceIoClose(fd);

ret = sceFaceParts(
        yImg, 320, 240, 320,
        partsDict,
        1, 1, &detectResult[i],
        resultPartsArray, SCE_FACE_PARTS_NUM_MAX, &numParts,
        workMemory, workMemorySize);
```

See Also

`SceFacePartsDictPtr`, `sceFaceParts()`

SCE_FACE_PARTS_ROLL_YAW_DICT

Parts detection dictionary data file (roll and yaw rotation support)

Definition

```
#include <libface.h>
#define SCE_FACE_PARTS_ROLL_YAW_DICT "face_parts_roll_yaw.pdt"
#define SCE_FACE_PARTS_ROLL_YAW_DICT_SIZE (193748)
```

Description

These constants represent the filename and size (in bytes) of dictionary data used in parts detection.

The *partsDictPtr* argument of the parts detection execution function `sceFaceParts()` should be set with a pointer to the memory area where this file was read.

This dictionary supports frontal faces, tilted faces (faces with a roll angle), and faces turned sideways (faces with a yaw angle).

Notes

Compared to `SCE_FACE_PARTS_ROLL_DICT`, there will be fewer parts detection errors in a face that is turned sideways (face with a large yaw angle) by this dictionary, however, the processing time will be greater.

This dictionary is suitable when `SCE_FACE_DETECT_YAW_DICT`, `SCE_FACE_DETECT_ROLL_YAW_DICT`, or `SCE_FACE_DETECT_ROLL_YAW_PITCH_DICT` was used as the face detection dictionary. Although it can also be used for face detection results that were obtained by using another face detection dictionary, unnecessary redundant processing will be performed internally in that case and processing time will be wasted.

Example

```
SceFacePartsDictPtr partsDict =
        (SceFacePartsDictPtr)malloc(SCE_FACE_PARTS_ROLL_YAW_DICT_SIZE);
SceUID fd = sceIoOpen("host0:"SCE_FACE_PARTS_ROLL_YAW_DICT, SCE_O_RDONLY, 0);
sceIoRead(fd, partsDict, SCE_FACE_PARTS_ROLL_YAW_DICT_SIZE);
sceIoClose(fd);

ret = sceFaceParts(
        yImg, 320, 240, 320,
        partsDict,
        1, 1, &detectResult[i],
        resultPartsArray, SCE_FACE_PARTS_NUM_MAX, &numParts,
        workMemory, workMemorySize);
```

See Also

`SceFacePartsDictPtr`, `sceFaceParts()`

SCE_FACE_ALLPARTS_DICT

Detailed parts detection dictionary data file (roll rotation support)

Definition

```
#include <libface.h>
#define SCE_FACE_ALLPARTS_DICT "face_allparts.pdt"
#define SCE_FACE_ALLPARTS_DICT_SIZE (1609168)
```

Description

These constants represent the filename and size (in bytes) of dictionary data used in detailed parts detection.

The *partsDictPtr* argument of the detailed parts detection execution function `sceFaceAllParts()` should be set with a pointer to the memory area where this file was read.

This dictionary supports frontal faces and tilted faces (faces with a roll angle).

Example

```
SceFacePartsDictPtr allpartsDict =
        (SceFacePartsDictPtr)malloc(SCE_FACE_ALLPARTS_DICT_SIZE);
SceUID fd = sceIoOpen("host0:"SCE_FACE_ALLPARTS_DICT, SCE_O_RDONLY, 0);
sceIoRead(fd, allpartsDict, SCE_FACE_ALLPARTS_DICT_SIZE);
sceIoClose(fd);

ret = sceFaceAllParts(
        yImg, 320, 240, 320,
        allpartsDict,
        shapeDict,
        1, 1, &detectResult[i],
        resultPartsArray, SCE_FACE_ALLPARTS_NUM_MAX, &numParts,
        workMemory, workMemorySize);
```

See Also

`SceFacePartsDictPtr`, `sceFaceAllParts()`

SCE_FACE_ALLPARTS_SHAPE_DICT

Shape correction dictionary data file for detailed parts detection (roll rotation support)

Definition

```
#include <libface.h>
#define SCE_FACE_ALLPARTS_SHAPE_DICT "face_allparts_shape.dat"
#define SCE_FACE_ALLPARTS_SHAPE_DICT_SIZE (55080)
```

Description

These constants represent the filename and size (in bytes) of dictionary data used in shape correction for detailed parts detection.

The *shapeDictPtr* argument of the detailed parts detection execution function `sceFaceAllParts()` should be set with a pointer to the memory area where this file was read.

This dictionary supports frontal faces and tilted faces (faces with a roll angle).

Notes

Using this dictionary to correct shape to stabilize the result for detailed parts detection is recommended. If it is not enough memory area to load this dictionary or it needs to gain a little more speed, shape correction feature can be disabled by setting NULL to the *shapeDictPtr* argument of `sceFaceAllParts()`.

Example

```
SceFaceShapeDictPtr shapeDict =
        (SceFacePartsDictPtr)malloc(SCE_FACE_ALLPARTS_SHAPE_DICT_SIZE);
SceUID fd = sceIoOpen("host0:"SCE_FACE_ALLPARTS_SHAPE_DICT, SCE_O_RDONLY, 0);
sceIoRead(fd, shapeDict, SCE_FACE_ALLPARTS_SHAPE_DICT_SIZE);
sceIoClose(fd);

ret = sceFaceParts(
        yImg, 320, 240, 320,
        allpartsDict,
        shapeDict,
        1, 1, &detectResult[i],
        resultPartsArray, SCE_FACE_ALLPARTS_NUM_MAX, &numParts,
        workMemory, workMemorySize);
```

See Also

`SceFaceShapeDictPtr`, `sceFaceAllParts()`

SCE_FACE_PARTS_CHECK_DICT

Parts result validity check dictionary data file (roll, yaw and pitch rotation support)

Definition

```
#include <libface.h>
#define SCE_FACE_PARTS_CHECK_DICT "face_parts_check.cdt"
#define SCE_FACE_PARTS_CHECK_DICT_SIZE (1868)
```

Description

These constants represent the filename and size (in bytes) of dictionary data used in parts detection result validity check.

The *partsCheckDictPtr* argument of the parts detection result validity check function `sceFacePartsResultCheck()` should be set with a pointer to the memory area where this file was read.

This dictionary is used to check the validity of the parts detection result which is detected from the face with roll, yaw and pitch rotation.

Notes

The parts result to check the validity is detected by either `SCE_FACE_PARTS_ROLL_DICT` or `SCE_FACE_PARTS_ROLL_YAW_DICT`.

Example

```
SceFacePartsCheckDictPtr partsCheckDict = (SceFacePartsCheckDictPtr)
malloc(SCE_FACE_PARTS_CHECK_DICT_SIZE);
SceUID fd = sceIoOpen("host0:"SCE_FACE_PARTS_CHECK_DICT, SCE_O_RDONLY, 0);
sceIoRead(fd, partsCheckDict, SCE_FACE_PARTS_CHECK_DICT_SIZE);
sceIoClose(fd);

/* Check the validity of detected parts result */
bool good;
ret = sceFacePartsResultCheck(
        &detectResult[i],
        resultPartsArray, numParts,
        partsCheckDict,
        &good);
```

See Also

`SceFacePartsCheckDictPtr`, `sceFacePartsResultCheck()`

Parts Detection and Detailed Parts Detection Macro

Constants used in parts detection and detailed parts detection

Definition

```
#include <libface.h>
#define SCE_FACE_PARTS_NUM_MAX 4
#define SCE_FACE_ALLPARTS_NUM_MAX 55
```

Description

These are the constants used in part detection and detailed part detection.

Value	Value	Description
SCE_FACE_PARTS_NUM_MAX	4	Maximum number of parts to detect
SCE_FACE_ALLPARTS_NUM_MAX	55	Maximum number of detailed parts to detect

See Also

sceFaceParts(),sceFaceAllParts()

Part ID Macros for Parts Detection

Parts ID constants used in parts detection

Definition

```
#include <libface.h>
#define SCE_FACE_PARTS_ID_UNDEF 0
#define SCE_FACE_PARTS_ID_R_EYE_CENTER 1
#define SCE_FACE_PARTS_ID_L_EYE_CENTER 2
#define SCE_FACE_PARTS_ID_NOSE_CENTER 3
#define SCE_FACE_PARTS_ID_MOUTH_CENTER 4
```

Description

These are the part ID constants used in part detection.

Macro	Value	Description
SCE_FACE_PARTS_ID_UNDEF	0	Undefined
SCE_FACE_PARTS_ID_R_EYE_CENTER	1	Center of right eye
SCE_FACE_PARTS_ID_L_EYE_CENTER	2	Center of left eye
SCE_FACE_PARTS_ID_NOSE_CENTER	3	Center of nose
SCE_FACE_PARTS_ID_MOUTH_CENTER	4	Center of mouth

When `sceFaceParts()` is called, the detection result of each part is output as an array of `SceFacePartsResult` datatype to the area specified in the *`resultPartsArray`* argument, where one of the above constants will be stored in each element's *`partsId`*.

See Also

`sceFaceParts()`

Result Index Macros for Parts Detection

Constants representing the order by which each part is stored in the part detection results array

Definition

```
#include <libface.h>
#define SCE_FACE_PARTS_R_EYE_CENTER_INDEX 0
#define SCE_FACE_PARTS_L_EYE_CENTER_INDEX 1
#define SCE_FACE_PARTS_NOSE_CENTER_INDEX 2
#define SCE_FACE_PARTS_MOUTH_CENTER_INDEX 3
```

Description

These constants indicate the index for the results of part detection to be stored in an array.

Macro	Value	Description
SCE_FACE_PARTS_R_EYE_CENTER_INDEX	0	Index with the right eye as the center
SCE_FACE_PARTS_L_EYE_CENTER_INDEX	1	Index with the left eye as the center
SCE_FACE_PARTS_NOSE_CENTER_INDEX	2	Index with the nose as the center
SCE_FACE_PARTS_MOUTH_CENTER_INDEX	3	Index with the mouth as the center

When `sceFaceParts()` is called, the detection result of each part is output as an array of `SceFacePartsResult` datatype to the area specified in the *`resultPartsArray`* argument. When using the above constants, the detection result of the center of the right eye, for example, can be accessed as the `SCE_FACE_PARTS_R_EYE_CENTER_INDEX`-th element of the output array, as exemplified in the following code.

```
SceFacePartsResult parts[SCE_FACE_PARTS_NUM_MAX];
// Omitted: prepare the parameters and call sceFaceParts()
float reyeX = parts[SCE_FACE_PARTS_R_EYE_CENTER_INDEX].partsX;
float reyeY = parts[SCE_FACE_PARTS_R_EYE_CENTER_INDEX].partsY;
float reyeScore = parts[SCE_FACE_PARTS_R_EYE_CENTER_INDEX].score;
```

See Also

`SceFacePartsResult`, `sceFaceParts()`

Part ID Macros for Detailed Part Detection

Part ID constants to be used in detailed part detection

Definition

```
#include <libface.h>
#define SCE_FACE_PARTS_ID_UNDEF 0
#define SCE_FACE_PARTS_ID_ALL_BASE 100
#define SCE_FACE_PARTS_FACE_ID_ALL_00 100
#define SCE_FACE_PARTS_FACE_ID_ALL_01 101
#define SCE_FACE_PARTS_FACE_ID_ALL_02 102
#define SCE_FACE_PARTS_FACE_ID_ALL_03 103
#define SCE_FACE_PARTS_FACE_ID_ALL_04 104
#define SCE_FACE_PARTS_FACE_ID_ALL_05 105
#define SCE_FACE_PARTS_FACE_ID_ALL_06 106
#define SCE_FACE_PARTS_FACE_ID_ALL_07 107
#define SCE_FACE_PARTS_FACE_ID_ALL_08 108
#define SCE_FACE_PARTS_FACE_ID_ALL_09 109
#define SCE_FACE_PARTS_FACE_ID_ALL_10 110
#define SCE_FACE_PARTS_FACE_ID_ALL_11 111
#define SCE_FACE_PARTS_R_EYEBROW_ID_ALL_12 112
#define SCE_FACE_PARTS_R_EYEBROW_ID_ALL_13 113
#define SCE_FACE_PARTS_R_EYEBROW_ID_ALL_14 114
#define SCE_FACE_PARTS_R_EYE_ID_ALL_15 115
#define SCE_FACE_PARTS_R_EYE_ID_ALL_16 116
#define SCE_FACE_PARTS_R_EYE_ID_ALL_17 117
#define SCE_FACE_PARTS_R_EYE_ID_ALL_18 118
#define SCE_FACE_PARTS_R_EYE_ID_ALL_19 119
#define SCE_FACE_PARTS_R_EYE_ID_ALL_20 120
#define SCE_FACE_PARTS_R_EYE_ID_ALL_21 121
#define SCE_FACE_PARTS_R_EYE_ID_ALL_22 122
#define SCE_FACE_PARTS_R_PUPIL_ID_ALL_23 123
#define SCE_FACE_PARTS_L_EYEBROW_ID_ALL_24 124
#define SCE_FACE_PARTS_L_EYEBROW_ID_ALL_25 125
#define SCE_FACE_PARTS_L_EYEBROW_ID_ALL_26 126
#define SCE_FACE_PARTS_L_EYE_ID_ALL_27 127
#define SCE_FACE_PARTS_L_EYE_ID_ALL_28 128
#define SCE_FACE_PARTS_L_EYE_ID_ALL_29 129
#define SCE_FACE_PARTS_L_EYE_ID_ALL_30 130
#define SCE_FACE_PARTS_L_EYE_ID_ALL_31 131
#define SCE_FACE_PARTS_L_EYE_ID_ALL_32 132
#define SCE_FACE_PARTS_L_EYE_ID_ALL_33 133
#define SCE_FACE_PARTS_L_EYE_ID_ALL_34 134
#define SCE_FACE_PARTS_L_PUPIL_ID_ALL_35 135
#define SCE_FACE_PARTS_NOSE_ID_ALL_36 136
#define SCE_FACE_PARTS_NOSE_ID_ALL_37 137

#define SCE_FACE_PARTS_NOSTRIL_ID_ALL_40 140
#define SCE_FACE_PARTS_NOSTRIL_ID_ALL_41 141
#define SCE_FACE_PARTS_NOSTRIL_ID_ALL_42 142

#define SCE_FACE_PARTS_R_LIP_ID_ALL_45 145
#define SCE_FACE_PARTS_U_LIP_ID_ALL_46 146
#define SCE_FACE_PARTS_U_LIP_ID_ALL_47 147
#define SCE_FACE_PARTS_U_LIP_ID_ALL_48 148
#define SCE_FACE_PARTS_L_LIP_ID_ALL_49 149
#define SCE_FACE_PARTS_D_LIP_ID_ALL_50 150
#define SCE_FACE_PARTS_D_LIP_ID_ALL_51 151
#define SCE_FACE_PARTS_D_LIP_ID_ALL_52 152
```

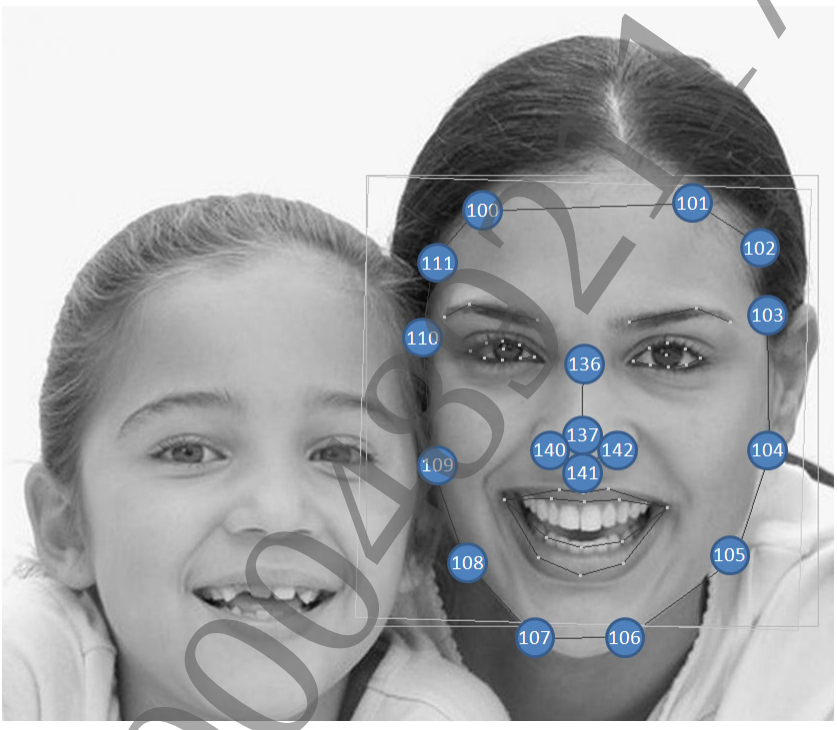
```
#define SCE_FACE_PARTS_UC_LIP_ID_ALL_53 153
#define SCE_FACE_PARTS_UC_LIP_ID_ALL_54 154
#define SCE_FACE_PARTS_UC_LIP_ID_ALL_55 155
#define SCE_FACE_PARTS_DC_LIP_ID_ALL_56 156
#define SCE_FACE_PARTS_DC_LIP_ID_ALL_57 157
#define SCE_FACE_PARTS_DC_LIP_ID_ALL_58 158
```

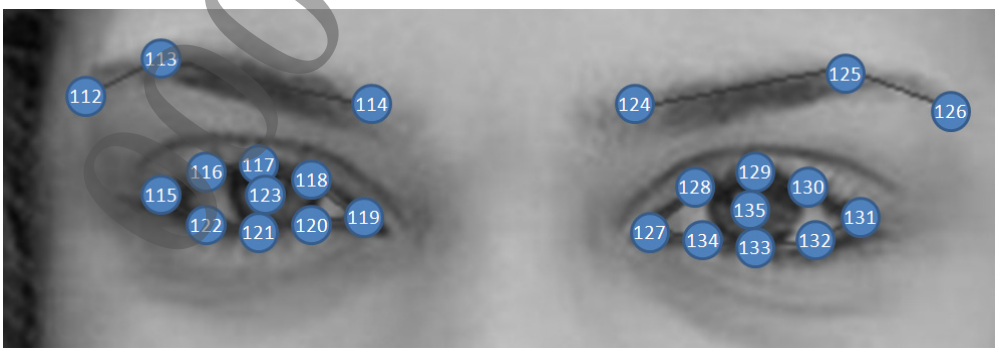
Description

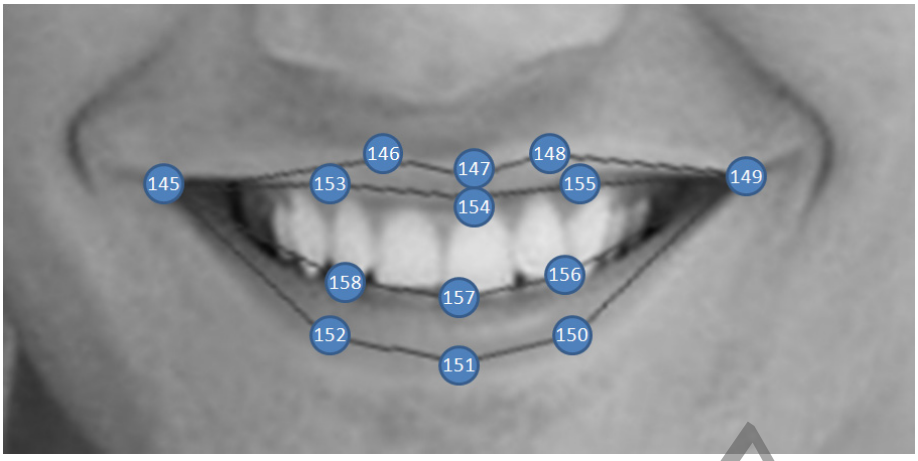
These are the part ID constants used in detailed part detection.

Macro	Value	Description
SCE_FACE_PARTS_ID_UNDEF	0	Undefined
SCE_FACE_PARTS_ID_ALL_BASE	100	Base of the detailed part ID
SCE_FACE_PARTS_FACE_ID_ALL_00	100	Facial outline 0
SCE_FACE_PARTS_FACE_ID_ALL_01	101	Facial outline 1
SCE_FACE_PARTS_FACE_ID_ALL_02	102	Facial outline 2
SCE_FACE_PARTS_FACE_ID_ALL_03	103	Facial outline 3
SCE_FACE_PARTS_FACE_ID_ALL_04	104	Facial outline 4
SCE_FACE_PARTS_FACE_ID_ALL_05	105	Facial outline 5
SCE_FACE_PARTS_FACE_ID_ALL_06	106	Facial outline 6
SCE_FACE_PARTS_FACE_ID_ALL_07	107	Facial outline 7
SCE_FACE_PARTS_FACE_ID_ALL_08	108	Facial outline 8
SCE_FACE_PARTS_FACE_ID_ALL_09	109	Facial outline 9
SCE_FACE_PARTS_FACE_ID_ALL_10	110	Facial outline 10
SCE_FACE_PARTS_FACE_ID_ALL_11	111	Facial outline 11
SCE_FACE_PARTS_R_EYEBROW_ID_ALL_12	112	Right eyebrow 0
SCE_FACE_PARTS_R_EYEBROW_ID_ALL_13	113	Right eyebrow 1
SCE_FACE_PARTS_R_EYEBROW_ID_ALL_14	114	Right eyebrow 2
SCE_FACE_PARTS_R_EYE_ID_ALL_15	115	Right eye 0
SCE_FACE_PARTS_R_EYE_ID_ALL_16	116	Right eye 1
SCE_FACE_PARTS_R_EYE_ID_ALL_17	117	Right eye 2
SCE_FACE_PARTS_R_EYE_ID_ALL_18	118	Right eye 3
SCE_FACE_PARTS_R_EYE_ID_ALL_19	119	Right eye 4
SCE_FACE_PARTS_R_EYE_ID_ALL_20	120	Right eye 5
SCE_FACE_PARTS_R_EYE_ID_ALL_21	121	Right eye 6
SCE_FACE_PARTS_R_EYE_ID_ALL_22	122	Right eye 7
SCE_FACE_PARTS_R_PUPIL_ID_ALL_23	123	Right pupil
SCE_FACE_PARTS_L_EYEBROW_ID_ALL_24	124	Left eyebrow 0
SCE_FACE_PARTS_L_EYEBROW_ID_ALL_25	125	Left eyebrow 1
SCE_FACE_PARTS_L_EYEBROW_ID_ALL_26	126	Left eyebrow 2
SCE_FACE_PARTS_L_EYE_ID_ALL_27	127	Left eye 0
SCE_FACE_PARTS_L_EYE_ID_ALL_28	128	Left eye 1
SCE_FACE_PARTS_L_EYE_ID_ALL_29	129	Left eye 2
SCE_FACE_PARTS_L_EYE_ID_ALL_30	130	Left eye 3
SCE_FACE_PARTS_L_EYE_ID_ALL_31	131	Left eye 4
SCE_FACE_PARTS_L_EYE_ID_ALL_32	132	Left eye 5
SCE_FACE_PARTS_L_EYE_ID_ALL_33	133	Left eye 6
SCE_FACE_PARTS_L_EYE_ID_ALL_34	134	Left eye 7
SCE_FACE_PARTS_L_PUPIL_ID_ALL_35	135	Left pupil
SCE_FACE_PARTS_NOSE_ID_ALL_36	136	Nose 0
SCE_FACE_PARTS_NOSE_ID_ALL_37	137	Nose 1
SCE_FACE_PARTS_NOSTRIL_ID_ALL_40	140	Nose 2
SCE_FACE_PARTS_NOSTRIL_ID_ALL_41	141	Nose 3
SCE_FACE_PARTS_NOSTRIL_ID_ALL_42	142	Nose 4

Macro	Value	Description
SCE_FACE_PARTS_R_LIP_ID_ALL_45	145	Mouth 0
SCE_FACE_PARTS_U_LIP_ID_ALL_46	146	Mouth 1
SCE_FACE_PARTS_U_LIP_ID_ALL_47	147	Mouth 2
SCE_FACE_PARTS_U_LIP_ID_ALL_48	148	Mouth 3
SCE_FACE_PARTS_L_LIP_ID_ALL_49	149	Mouth 4
SCE_FACE_PARTS_D_LIP_ID_ALL_50	150	Mouth 5
SCE_FACE_PARTS_D_LIP_ID_ALL_51	151	Mouth 6
SCE_FACE_PARTS_D_LIP_ID_ALL_52	152	Mouth 7
SCE_FACE_PARTS_UC_LIP_ID_ALL_53	153	Mouth 8
SCE_FACE_PARTS_UC_LIP_ID_ALL_54	154	Mouth 9
SCE_FACE_PARTS_UC_LIP_ID_ALL_55	155	Mouth 10
SCE_FACE_PARTS_DC_LIP_ID_ALL_56	156	Mouth 11
SCE_FACE_PARTS_DC_LIP_ID_ALL_57	157	Mouth 12
SCE_FACE_PARTS_DC_LIP_ID_ALL_58	158	Mouth 13







When `sceFaceAllParts()` is called, the detection result of each part is output as an array of `SceFacePartsResult` datatype to the area specified in the *`resultPartsArray`* argument, where one of the above constants will be stored in each element's *`partsId`*.

See Also

`SceFacePartsResult`, `sceFaceAllParts()`

Result Index Macros for Detailed Parts Detection

Constants representing the order by which each part is stored in the detailed part detection results array

Definition

```
#include <libface.h>
#define SCE_FACE_PARTS_FACE_ALL_00_INDEX 0
#define SCE_FACE_PARTS_FACE_ALL_01_INDEX 1
#define SCE_FACE_PARTS_FACE_ALL_02_INDEX 2
#define SCE_FACE_PARTS_FACE_ALL_03_INDEX 3
#define SCE_FACE_PARTS_FACE_ALL_04_INDEX 4
#define SCE_FACE_PARTS_FACE_ALL_05_INDEX 5
#define SCE_FACE_PARTS_FACE_ALL_06_INDEX 6
#define SCE_FACE_PARTS_FACE_ALL_07_INDEX 7
#define SCE_FACE_PARTS_FACE_ALL_08_INDEX 8
#define SCE_FACE_PARTS_FACE_ALL_09_INDEX 9
#define SCE_FACE_PARTS_FACE_ALL_10_INDEX 10
#define SCE_FACE_PARTS_FACE_ALL_11_INDEX 11
#define SCE_FACE_PARTS_R_EYEBROW_ALL_12_INDEX 12
#define SCE_FACE_PARTS_R_EYEBROW_ALL_13_INDEX 13
#define SCE_FACE_PARTS_R_EYEBROW_ALL_14_INDEX 14
#define SCE_FACE_PARTS_R_EYE_ALL_15_INDEX 15
#define SCE_FACE_PARTS_R_EYE_ALL_16_INDEX 16
#define SCE_FACE_PARTS_R_EYE_ALL_17_INDEX 17
#define SCE_FACE_PARTS_R_EYE_ALL_18_INDEX 18
#define SCE_FACE_PARTS_R_EYE_ALL_19_INDEX 19
#define SCE_FACE_PARTS_R_EYE_ALL_20_INDEX 20
#define SCE_FACE_PARTS_R_EYE_ALL_21_INDEX 21
#define SCE_FACE_PARTS_R_EYE_ALL_22_INDEX 22
#define SCE_FACE_PARTS_R_PUPIL_ALL_23_INDEX 23
#define SCE_FACE_PARTS_L_EYEBROW_ALL_24_INDEX 24
#define SCE_FACE_PARTS_L_EYEBROW_ALL_25_INDEX 25
#define SCE_FACE_PARTS_L_EYEBROW_ALL_26_INDEX 26
#define SCE_FACE_PARTS_L_EYE_ALL_27_INDEX 27
#define SCE_FACE_PARTS_L_EYE_ALL_28_INDEX 28
#define SCE_FACE_PARTS_L_EYE_ALL_29_INDEX 29
#define SCE_FACE_PARTS_L_EYE_ALL_30_INDEX 30
#define SCE_FACE_PARTS_L_EYE_ALL_31_INDEX 31
#define SCE_FACE_PARTS_L_EYE_ALL_32_INDEX 32
#define SCE_FACE_PARTS_L_EYE_ALL_33_INDEX 33
#define SCE_FACE_PARTS_L_EYE_ALL_34_INDEX 34
#define SCE_FACE_PARTS_L_PUPIL_ALL_35_INDEX 35
#define SCE_FACE_PARTS_NOSE_ALL_36_INDEX 36
#define SCE_FACE_PARTS_NOSE_ALL_37_INDEX 37
#define SCE_FACE_PARTS_NOSTRIL_ALL_40_INDEX 38
#define SCE_FACE_PARTS_NOSTRIL_ALL_41_INDEX 39
#define SCE_FACE_PARTS_NOSTRIL_ALL_42_INDEX 40
#define SCE_FACE_PARTS_R_LIP_ALL_45_INDEX 41
#define SCE_FACE_PARTS_U_LIP_ALL_46_INDEX 42
#define SCE_FACE_PARTS_U_LIP_ALL_47_INDEX 43
#define SCE_FACE_PARTS_U_LIP_ALL_48_INDEX 44
#define SCE_FACE_PARTS_L_LIP_ALL_49_INDEX 45
#define SCE_FACE_PARTS_D_LIP_ALL_50_INDEX 46
#define SCE_FACE_PARTS_D_LIP_ALL_51_INDEX 47
#define SCE_FACE_PARTS_D_LIP_ALL_52_INDEX 48
#define SCE_FACE_PARTS_UC_LIP_ALL_53_INDEX 49
#define SCE_FACE_PARTS_UC_LIP_ALL_54_INDEX 50
```

```
#define SCE_FACE_PARTS_UC_LIP_ALL_55_INDEX 51
#define SCE_FACE_PARTS_DC_LIP_ALL_56_INDEX 52
#define SCE_FACE_PARTS_DC_LIP_ALL_57_INDEX 53
#define SCE_FACE_PARTS_DC_LIP_ALL_58_INDEX 54
```

Description

These constants indicate the index for the results of part detection to be stored in an array.

Macro	Value	Description
SCE_FACE_PARTS_FACE_ALL_00_INDEX	0	Index of face outline 0
SCE_FACE_PARTS_FACE_ALL_01_INDEX	1	Index of face outline 1
SCE_FACE_PARTS_FACE_ALL_02_INDEX	2	Index of face outline 2
SCE_FACE_PARTS_FACE_ALL_03_INDEX	3	Index of face outline 3
SCE_FACE_PARTS_FACE_ALL_04_INDEX	4	Index of face outline 4
SCE_FACE_PARTS_FACE_ALL_05_INDEX	5	Index of face outline 5
SCE_FACE_PARTS_FACE_ALL_06_INDEX	6	Index of face outline 6
SCE_FACE_PARTS_FACE_ALL_07_INDEX	7	Index of face outline 7
SCE_FACE_PARTS_FACE_ALL_08_INDEX	8	Index of face outline 8
SCE_FACE_PARTS_FACE_ALL_09_INDEX	9	Index of face outline 9
SCE_FACE_PARTS_FACE_ALL_10_INDEX	10	Index of face outline 10
SCE_FACE_PARTS_FACE_ALL_11_INDEX	11	Index of face outline 11
SCE_FACE_PARTS_R_EYEBROW_ALL_12_INDEX	12	Index of right eyebrow 0
SCE_FACE_PARTS_R_EYEBROW_ALL_13_INDEX	13	Index of right eyebrow 1
SCE_FACE_PARTS_R_EYEBROW_ALL_14_INDEX	14	Index of right eyebrow 2
SCE_FACE_PARTS_R_EYE_ALL_15_INDEX	15	Index of right eye 0
SCE_FACE_PARTS_R_EYE_ALL_16_INDEX	16	Index of right eye 1
SCE_FACE_PARTS_R_EYE_ALL_17_INDEX	17	Index of right eye 2
SCE_FACE_PARTS_R_EYE_ALL_18_INDEX	18	Index of right eye 3
SCE_FACE_PARTS_R_EYE_ALL_19_INDEX	19	Index of right eye 4
SCE_FACE_PARTS_R_EYE_ALL_20_INDEX	20	Index of right eye 5
SCE_FACE_PARTS_R_EYE_ALL_21_INDEX	21	Index of right eye 6
SCE_FACE_PARTS_R_EYE_ALL_22_INDEX	22	Index of right eye 7
SCE_FACE_PARTS_R_PUPIL_ALL_23_INDEX	23	Index of right pupil
SCE_FACE_PARTS_L_EYEBROW_ALL_24_INDEX	24	Index of left eyebrow 0
SCE_FACE_PARTS_L_EYEBROW_ALL_25_INDEX	25	Index of left eyebrow 1
SCE_FACE_PARTS_L_EYEBROW_ALL_26_INDEX	26	Index of left eyebrow 2
SCE_FACE_PARTS_L_EYE_ALL_27_INDEX	27	Index of left eye 0
SCE_FACE_PARTS_L_EYE_ALL_28_INDEX	28	Index of left eye 1
SCE_FACE_PARTS_L_EYE_ALL_29_INDEX	29	Index of left eye 2
SCE_FACE_PARTS_L_EYE_ALL_30_INDEX	30	Index of left eye 3
SCE_FACE_PARTS_L_EYE_ALL_31_INDEX	31	Index of left eye 4
SCE_FACE_PARTS_L_EYE_ALL_32_INDEX	32	Index of left eye 5
SCE_FACE_PARTS_L_EYE_ALL_33_INDEX	33	Index of left eye 6
SCE_FACE_PARTS_L_EYE_ALL_34_INDEX	34	Index of left eye 7
SCE_FACE_PARTS_L_PUPIL_ALL_35_INDEX	35	Index of left pupil
SCE_FACE_PARTS_NOSE_ALL_36_INDEX	36	Index of nose 0
SCE_FACE_PARTS_NOSE_ALL_37_INDEX	37	Index of nose 1
SCE_FACE_PARTS_NOSTRIL_ALL_40_INDEX	38	Index of nose 2 (nose line 0)
SCE_FACE_PARTS_NOSTRIL_ALL_41_INDEX	39	Index of nose 3 (nose line 1)
SCE_FACE_PARTS_NOSTRIL_ALL_42_INDEX	40	Index of nose 4 (nose line 2)
SCE_FACE_PARTS_R_LIP_ALL_45_INDEX	41	Index of mouth 0 (right lip edge)
SCE_FACE_PARTS_U_LIP_ALL_46_INDEX	42	Index of mouth 1 (upper lip 0)
SCE_FACE_PARTS_U_LIP_ALL_47_INDEX	43	Index of mouth 2 (upper lip 1)
SCE_FACE_PARTS_U_LIP_ALL_48_INDEX	44	Index of mouth 3 (upper lip 2)

Macro	Value	Description
SCE_FACE_PARTS_L_LIP_ALL_49_INDEX	45	Index of mouth 4 (left lip edge)
SCE_FACE_PARTS_D_LIP_ALL_50_INDEX	46	Index of mouth 5 (lower lip 0)
SCE_FACE_PARTS_D_LIP_ALL_51_INDEX	47	Index of mouth 6 (lower lip 1)
SCE_FACE_PARTS_D_LIP_ALL_52_INDEX	48	Index of mouth 7 (lower lip 2)
SCE_FACE_PARTS_UC_LIP_ALL_53_INDEX	49	Index of mouth 8 (upper lip 3)
SCE_FACE_PARTS_UC_LIP_ALL_54_INDEX	50	Index of mouth 9 (upper lip 4)
SCE_FACE_PARTS_UC_LIP_ALL_55_INDEX	51	Index of mouth 10 (upper lip 5)
SCE_FACE_PARTS_DC_LIP_ALL_56_INDEX	52	Index of mouth 11(lower lip 3)
SCE_FACE_PARTS_DC_LIP_ALL_57_INDEX	53	Index of mouth 12 (lower lip 4)
SCE_FACE_PARTS_DC_LIP_ALL_58_INDEX	54	Index of mouth 13 (lower lip 5)

When `sceFaceAllParts()` is called, the detection result of each part is output as an array of `SceFacePartsResult` datatype to the area specified in the *`resultPartsArray`* argument. When using the above constants, the detection result of face outline 0, for example, can be accessed as the `SCE_FACE_PARTS_FACE_ALL_00_INDEX`-th element of the output array, as exemplified in the following code.

```
SceFacePartsResult allParts[SCE_FACE_ALLPARTS_NUM_MAX];
// Omitted: prepare the parameters and call sceFaceAllParts()
float allParts00X = allParts[SCE_FACE_PARTS_ALL_00_INDEX].partsX;
float allParts00Y = allParts[SCE_FACE_PARTS_ALL_00_INDEX].partsY;
float allParts00Score = allParts[SCE_FACE_PARTS_ALL_00_INDEX].score;
```

For details regarding which part of a face each detailed part corresponds to, refer to "Part ID Macros for Detailed Part Detection".

See Also

`SceFacePartsResult`, `sceFaceAllParts()`

Attribute Classification and Age Estimate APIs

SceFaceAttribDictPtr

Attribute classification dictionary data pointer type

Definition

```
#include <libface.h>
typedef void *SceFaceAttribDictPtr;
```

Description

This defined type is used as a pointer to dictionary data used in attribute classification.
It is used for an argument type of the function `sceFaceAttribute()`.

See Also

`sceFaceAttribute()`

SceFaceAgeDictPtr

Age range estimate dictionary data pointer type

Definition

```
#include <libface.h>
typedef void *SceFaceAgeDictPtr;
```

Description

This defined type is used as a pointer to dictionary data used in age estimate.
It is used for an argument type of the function `sceFaceAgeRangeEstimate()`.

See Also

`sceFaceAgeRangeEstimate()`

SceFaceAttribResult

Attribute classification result

Definition

```
#include <libface.h>
typedef struct SceFaceAttribResult {
        unsigned int attribId;
        float score;
} SceFaceAttribResult;
```

Members

attribId	Attribute ID
score	Attribute classification score

Description

This data type is output for attribute classification results. When `sceFaceAttribute()` is called, classified result is written as an array of this structure in the area set in the *resultAttribArray* argument.

For *attribId*, one of the following values is set.

Macro	Value	Description
SCE_FACE_ATTRIB_ID_UNDEF	0	Undefined attribute
SCE_FACE_ATTRIB_ID_SMILE	1	Smiling face (0.0: Sorrowful, 20.0: Expressionless, 100.0: Big smile)
SCE_FACE_ATTRIB_ID_REYEOPEN	2	How open the right eye is [0.0 to 100.0 (from Closed to Opened wide)]
SCE_FACE_ATTRIB_ID_LEYEOPEN	3	How open the left eye is [0.0 to 100.0 (from Closed to Opened wide)]
SCE_FACE_ATTRIB_ID_GENDER	4	Gender [Under 50.0: Female, Over 50.0: Male]
SCE_FACE_ATTRIB_ID_ADULT	5	Adult [Under 50.0: Child, Over 50.0: Adult]
SCE_FACE_ATTRIB_ID_BABY	6	Baby [Under 50.0: Not a baby, Over 50.0: Baby]
SCE_FACE_ATTRIB_ID_ELDER	7	Elderly person [Under 50.0: Not elderly, Over 50.0: Elderly]
SCE_FACE_ATTRIB_ID_GLASS	8	Glasses [Under 50.0: No glasses, Over 50.0: Wearing glasses]

The value entered in *score* is output as a normalized value ranging from 0.0 to 100.0. If *attribId* is SCE_FACE_ATTRIB_ID_SMILE, SCE_FACE_ATTRIB_ID_REYEOPEN or SCE_FACE_ATTRIB_ID_LEYEOPEN, the value to be stored in *score* corresponds to what degree a person is smiling, or how wide the eyes are opened. Thus, if the value is in the middle range, a person can be evaluated as smiling or squinting their eyes, for example. For other attributes, the value stored in score does not represent a degree but the accuracy of the classification result; thus, make classification by using 50.0 as the point of reference and seeing whether the value is greater or lesser than that. The closer a value is to 0.0 or 100.0 indicates a more accurate classification result.

See Also

`sceFaceAttribute()`

SceFaceAgeRangeResult

Age range estimate result

Definition

```
#include <libface.h>
typedef struct SceFaceAgeRangeResult {
        unsigned char maxAge;
        unsigned char minAge;
} SceFaceAgeRangeResult;
```

Members

maxAge	Maximum age estimate
minAge	Minimum age estimate

Description

This data type is output for age range estimate results. When `sceFaceAgeRangeEstimate()` is called, estimate results are written as this structure in the area set in the *resultAge* argument.

For *minAge* and *maxAge*, respectively, an age from 0 to 70 will be stored. For example, *minAge*=10, *maxAge*=20 means the estimated age is from 10 to 20 years old. However, *maxAge*=70 may mean that the estimated maximum age is higher than 70.

Although the usual difference between *minAge* and *maxAge* is 10, the difference may exceed 20 under conditions that are difficult to make better estimates.

See Also

`sceFaceAgeRangeEstimate()`

SceFaceAgeDistrData

Age probability distribution data

Definition

```
#include <libface.h>
typedef struct SceFaceAgeDistrData {
        float score[SCE_FACE_AGE_BIN_SIZE];
        int numIntegrated;
} SceFaceAgeDistrData;
```

Members

score	Probability score for each age
numIntegrated	Number of integrations of age range estimate results

Description

This data type is output for age estimate probability distribution obtained by integrating multiple age range estimate results. When `sceFaceAgeRangeIntegrate()` is called, the age range estimate result set to *ageRange* will be integrated to this structure set using *ageDistrData*. By repeating age range estimate and integrating those results, estimate age probability distribution data will be accumulated in this structure.

For *score*, the probability score per age will be output as a `float` array. The age range for which probability can be calculated is between 0 and 80 (`SCE_FACE_AGE_BIN_SIZE(81)`).

For *numIntegrated*, the number of times age range estimate results have been integrated will be stored. When the value of *numIntegrated* is small, probability distribution may be unstable.

See Also

`sceFaceAgeRangeEstimate()`

sceFaceAttribute

Execute attribute classification

Definition

```
#include <libface.h>
int sceFaceAttribute(
        const unsigned char *imgPtr,
        int width,
        int height,
        int rowstride,
        const SceFaceAttribDictPtr attribDictPtr,
        const SceFaceDetectionResult *detectedFace,
        const SceFacePartsResult detectedPartsArray[],
        int detectedPartsNum,
        SceFaceAttribResult resultAttribArray[],
        int resultAttribArraySize,
        int *resultAttribNum,
        void* workMemory,
        int workMemorySize
)
```

Arguments

imgPtr	Pointer to input image (8-bit grayscale)
width	Input image width [pixels]
height	Input image height [pixels]
rowstride	Input image data width
attribDictPtr	Pointer to attribute classification dictionary data
detectedFace	Pointer to face detection results
detectedPartsArray	Pointer to parts detection results
detectedPartsNum	Number of detected parts
resultAttribArray	Pointer to attribute classification result output area
resultAttribArraySize	Number of array elements in *resultAttribArray*
resultAttribNum	Number of discriminated attributes
workMemory	Pointer to work buffer
workMemorySize	Size of work buffer

Return Values

Returns SCE_OK(0) if processing is successful.

Returns one of the following error codes (a negative value) for errors.

Value	(Hexadecimal)	Description
SCE_FACE_ERROR_NO_MEMORY	0x808B0001	Not enough memory to run, or *workMemory* is NULL
SCE_FACE_ERROR_INVALID_PARAM	0x808B0002	Parameter is invalid
SCE_FACE_ERROR_INVALID_DICT	0x808B0003	Attribute classification dictionary data is invalid, or *attribDictPtr* is NULL
SCE_FACE_ERROR_IMPERF_PARTS	0x808B0004	Parts are incomplete
SCE_FACE_ERROR_OUT_OF_RANGE	0x808B0005	Face angle is out of range

Description

This function uses face detection results and parts detection results for the eyes, nose, and mouth to perform attribute classification.

The input image that is set in *imgPtr* must be the same as the one used in parts detection.

For *attribDictPtr*, set a pointer to the memory area where the attribute classification dictionary data file SCE_FACE_ATTRIB_DICT or SCE_FACE_ATTRIB_SMILE_DICT was read.

The face detection results that are set in *detectedFace* must be the same results as those used as input to parts detection.

The parts detection results set in *detectedPartsArray* must contain normal detection results of all four parts (left and right eyes, nose and mouth), and *detectedPartsNum* must be SCE_FACE_PARTS_NUM_MAX(4).

If the parts detection results that were set do not satisfy these conditions and attribute classification cannot be performed, an SCE_FACE_ERROR_IMPERF_PARTS error will be returned.

For *resultAttribArray*, set a pointer to an array of the SceFaceAttribResult structure having SCE_FACE_ATTRIB_NUM_MAX(8) elements, and for *resultAttribArraySize*, set SCE_FACE_ATTRIB_NUM_MAX(8).

When attribute classification processing is performed normally, SCE_FACE_ATTRIB_NUM_MAX(8) is returned in *resultAttribNum*, and the attribute classification results are output as an SceFaceAttribResult array of SCE_FACE_ATTRIB_NUM_MAX(8) elements to the area specified by *resultAttribArray*.

However, if only smile attribute classification is performed using SCE_FACE_ATTRIB_SMILE_DICT, having just one element in *resultAttribArray* is sufficient, and if smile attribute classification is performed successfully, it is set to 1 in *resultAttribNum*.

For *workMemory*, set the pointer to the allocated memory whose size is calculated by sceFaceAttributeGetWorkingMemorySize() or more.

For *workMemorySize*, set the size of *workMemory*.

Notes

If the input image shooting conditions were not ideal and the face parts are dark due to backlighting or insufficient light or if, on the other hand, the face is overexposed and is whitened out, or if the image contains a lot of noise, attribute classification performance may worsen even if processing is performed normally by the program. Adjust the input image quality as needed before calling this function.

Attribute classification performance tends to get worse, the more the face is offset from the frontal direction (as the face angle increases).

Also, due to a cause of image quality etc., an offset from a real part position may end up being incorrectly recognized as the actual part position without an error occurring in the program, and the attribute classification result in that case may also differ from the true result.

Example

```
/* Read face recognition dictionary data file */
unsigned char detectDict[SCE_FACE_DETECT_FRONTAL_DICT_SIZE];
unsigned char partsDict[SCE_FACE_PARTS_ROLL_DICT_SIZE];
unsigned char attribDict[SCE_FACE_ATTRIB_SMILE_DICT_SIZE];
SceUID fd;
fd = sceIoOpen("host0:"SCE_FACE_DETECT_FRONTAL_DICT, SCE_O_RDONLY, 0);
sceIoRead(fd, detectDict, SCE_FACE_DETECT_FRONTAL_DICT_SIZE);
sceIoClose(fd);
fd = sceIoOpen("host0:"SCE_FACE_PARTS_ROLL_DICT, SCE_O_RDONLY, 0);
sceIoRead(fd, partsDict, SCE_FACE_PARTS_ROLL_DICT_SIZE);
sceIoClose(fd);
fd = sceIoOpen("host0:"SCE_FACE_ATTRIB_SMILE_DICT, SCE_O_RDONLY, 0);
sceIoRead(fd, attribDict, SCE_FACE_ATTRIB_SMILE_DICT_SIZE);
sceIoClose(fd);

unsigned char yImg[320*240];
SceFaceDetectionResult detectResult[16];
SceFacePartsResult resultPartsArray[SCE_FACE_PARTS_NUM_MAX];
SceFaceAttribResult resultAttribArray[SCE_FACE_ATTRIB_NUM_MAX];
int i, j, numFace, numParts, numAttrib, ret;

/* Allocate work buffer for face detection */
int workSize = sceFaceDetectionGetWorkingMemorySize(320, 240, 320, detectDict);
void *workPtr = malloc(workSize);

/* Allocate work buffer for parts detection */
int workPartsSize = sceFacePartsGetWorkingMemorySize(320, 240, 320, partsDict);
void *workPartsPtr = malloc(workPartsSize);

/* Allocate work buffer for attribute classification */
int workAttribSize = sceFaceAttributeGetWorkingMemorySize(
        320, 240, 320, attribDict);
void *workAttribPtr = malloc(workAttribSize);

/* Execute face detection */
ret = sceFaceDetection(
        yImg, 320, 240, 320,
        detectDict,
        0.4f, 0.841f, 0.0f, 2, 2, 0.5f,
        SCE_FACE_DETECT_RESULT_NORMAL,
        detectResult, 16, &numFace,
        workBuf, workSize);
if (ret != SCE_OK) {
        printf("sceFaceDetection() failed! (0x%08x)\n", ret);
        return;
}
for (i = 0; i < numFace; i++) {
        /* Execute parts detection */
        ret = sceFaceParts(
              yImg, 320, 240, 320,
              partsDict,
              1, 1, &detectResult[i],
              resultPartsArray, SCE_FACE_PARTS_NUM_MAX, &numParts,
              workPartsPtr, workPartsSize);
        if (ret != SCE_OK) {
              printf("sceFaceParts() failed! (0x%08x)\n", ret);
              return;
        }
```

```
            /* Execute attribute classification */
            ret = sceFaceAttribute(
                  yImg, 320, 240, 320,
                  attribDict,
                  &detectResult[i],
                  resultPartsArray, numParts,
                  resultAttribArray, SCE_FACE_ATTRIB_NUM_MAX, &numAttrib,
                  workAttribPtr, workAttribSize);
            if (ret != SCE_OK) {
                  printf("sceFaceAttribute() failed! (0x%08x)\n", ret);
                  return;
            }
            for (j = 0; j < numAttrib; j++) {
                  printf("facial attribute:   id = %d score = %f\n",
                         resultAttribArray[j].attribId,
                         resultAttribArray[j].score);
            }
      }
```

See Also

SceFaceAttribDictPtr, SCE_FACE_ATTRIB_SMILE_DICT, SCE_FACE_ATTRIB_DICT, SceFaceDetectionResult, SceFacePartsResult, SceFaceAttribResult, sceFaceAttributeGetWorkingMemorySize()

sceFaceAgeRangeEstimate

Execute age range estimate

Definition

```
#include <libface.h>
int sceFaceAgeRangeEstimate(
        const unsigned char *imgPtr,
        int width,
        int height,
        int rowstride,
        const SceFaceAgeDictPtr ageDictPtr,
        const SceFaceDetectionResult *detectedFace,
        const SceFacePartsResult detectedPartsArray[],
        int detectedPartsNum,
        SceFaceAgeRangeResult *resultAge,
        void* workMemory,
        int workMemorySize
)
```

Arguments

imgPtr	Pointer to input image (8-bit grayscale)
width	Input image width [pixels]
height	Input image height [pixels]
rowstride	Input image data width
ageDictPtr	Pointer to age estimate range dictionary data
detectedFace	Pointer to face detection result
detectedPartsArray	Pointer to parts detection results
detectedPartsNum	Number of detected parts
resultAge	Pointer to age range estimate result output area
workMemory	Pointer to work buffer
workMemorySize	Size of work buffer

Return Values

Returns SCE_OK(0) if processing is successful.

Returns one of the following error codes (a negative value) for errors.

Value	(Hexadecimal)	Description
SCE_FACE_ERROR_NO_MEMORY	0x808B0001	Not enough memory to run, or *workMemory* is NULL
SCE_FACE_ERROR_INVALID_PARAM	0x808B0002	Parameter is invalid
SCE_FACE_ERROR_INVALID_DICT	0x808B0003	Age range estimate dictionary data is invalid or *ageDictPtr* is NULL
SCE_FACE_ERROR_IMPERF_PARTS	0x808B0004	Parts are incomplete
SCE_FACE_ERROR_OUT_OF_RANGE	0x808B0005	Face angle is out of range

Description

This function uses face detection results and parts detection results for the eyes, nose, and mouth to perform an age range estimate.

The input image that is set in *imgPtr* must be the same as the one used in parts detection.

For *ageDictPtr*, set a pointer to the memory area where the age range estimate dictionary data file SCE_FACE_AGE_DICT was read.

The face detection results that are set in *detectedFace* must be the same results as those input to parts detection.

The parts detection results set in *detectedPartsArray* must contain normal detection results of all four parts (left and right eyes, nose, and mouth), and *detectedPartsNum* must be SCE_FACE_PARTS_NUM_MAX(4).

If the parts detection results that were set do not satisfy these conditions and attribute classification cannot be performed, an SCE_FACE_ERROR_IMPERF_PARTS error will be returned.

For *resultAge*, set the pointer to SceFaceAgeRangeResult.

For *workMemory*, set the pointer to the allocated memory whose size is calculated by sceFaceAgeGetWorkingMemorySize() or more.

For *workMemorySize*, set the size of *workMemory*.

Notes

If the input image shooting conditions were not ideal and the face parts are dark due to backlighting or insufficient light or if, on the other hand, the face is overexposed and is whitened out, or if the image contains a lot of noise, age range estimate performance may worsen even if processing is performed normally by the program. Adjust the input image quality as needed before calling this function.

Age range estimate performance tends to get worse, the more the face is offset from the frontal direction (as the face angle increases).

Also, due to a cause of image quality etc., an offset from a real part position may end up being incorrectly recognized as the actual part position without an error occurring in the program, and the age range estimate result in that case may also differ from the true result.

Examples

```
/* Read face recognition dictionary data file */
unsigned char detectDict[SCE_FACE_DETECT_FRONTAL_DICT_SIZE];
unsigned char partsDict[SCE_FACE_PARTS_ROLL_DICT_SIZE];
unsigned char ageDict[SCE_FACE_AGE_DICT_SIZE];
SceUID fd;
fd = sceIoOpen("host0:"SCE_FACE_DETECT_FRONTAL_DICT, SCE_O_RDONLY, 0);
sceIoRead(fd, detectDict, SCE_FACE_DETECT_FRONTAL_DICT_SIZE);
sceIoClose(fd);
fd = sceIoOpen("host0:"SCE_FACE_PARTS_ROLL_DICT, SCE_O_RDONLY, 0);
sceIoRead(fd, partsDict, SCE_FACE_PARTS_ROLL_DICT_SIZE);
sceIoClose(fd);
fd = sceIoOpen("host0:"SCE_FACE_AGE_DICT, SCE_O_RDONLY, 0);
sceIoRead(fd, ageDict, SCE_FACE_AGE_DICT_SIZE);
sceIoClose(fd);

unsigned char yImg[320*240];
SceFaceDetectionResult detectResult[16];
SceFacePartsResult resultPartsArray[SCE_FACE_PARTS_NUM_MAX];
SceFaceAgeRangeResult resultAge;
int i, j, numFace, numParts, ret;
```

```
/* Allocate work buffer for face detection */
int workSize = sceFaceDetectionGetWorkingMemorySize(320, 240, 320, detectDict);
void *workPtr = malloc(workSize);

/* Allocate work buffer for parts detection */
int workPartsSize = sceFacePartsGetWorkingMemorySize(320, 240, 320, partsDict);
void *workPartsPtr = malloc(workPartsSize);

/* Allocate work buffer for age range estimate */
int workAgeSize = sceFaceAgeGetWorkingMemorySize(320, 240, 320, ageDict);
void *workAgePtr = malloc(workAgeSize);

/* Execute face detection */
ret = sceFaceDetection(
        yImg, 320, 240, 320,
        detectDict,
        0.4f, 0.841f, 0.0f, 2, 2, 0.5f,
        SCE_FACE_DETECT_RESULT_NORMAL,
        detectResult, 16, &numFace,
        workBuf, workSize);
if (ret != SCE_OK) {
        printf("sceFaceDetection() failed! (0x%08x)\n", ret);
        return;
}
for (i = 0; i < numFace; i++) {
        /* Execute parts detection */
        ret = sceFaceParts(
              yImg, 320, 240, 320,
              partsDict,
              1, 1, &detectResult[i],
              resultPartsArray, SCE_FACE_PARTS_NUM_MAX, &numParts,
              workPartsPtr, workPartsSize);
        if (ret != SCE_OK) {
              printf("sceFaceParts() failed! (0x%08x)\n", ret);
              return;
        }
        /* Execute age range estimate */
        ret = sceFaceAgeRangeEstimate(
              yImg, 320, 240, 320,
              ageDict,
              &detectResult[i],
              resultPartsArray, numParts,
              &resultAge,
              workAgePtr, workAgeSize);
        if (ret != SCE_OK) {
              printf("sceFaceAgeRangeEstimate() failed! (0x%08x)\n", ret);
              return;
        }
        printf("facial age range:   (%d ~ %d)\n",
              resultAge.minAge,
              resultAge.maxAge);
        }
}
```

See Also

SceFaceAgeDictPtr, SCE_FACE_AGE_DICT, SceFaceDetectionResult, SceFacePartsResult, SceFaceAgeRangeResult, sceFaceAgeGetWorkingMemorySize()

sceFaceAgeRangeIntegrate

Integrate age range estimate results (statistical age estimate)

Definition

```
#include <libface.h>
int sceFaceAgeRangeIntegrate(
        const SceFaceAgeRangeResult *ageRange,
        SceFaceAgeDistrData *ageDistrData,
        int *ageResult
)
```

Arguments

ageRange	Pointer to the target age range estimate result
ageDistrData	Pointer to the target age estimate probability distribution data
ageResult	Pointer to the age estimated from the integrated result

Return Values

Returns SCE_OK(0) if processing is successful.

Returns the following error code (a negative value) for errors.

Value	(Hexadecimal)	Description
SCE_FACE_ERROR_INVALID_PARAM	0x808B0002	Parameter is invalid

Description

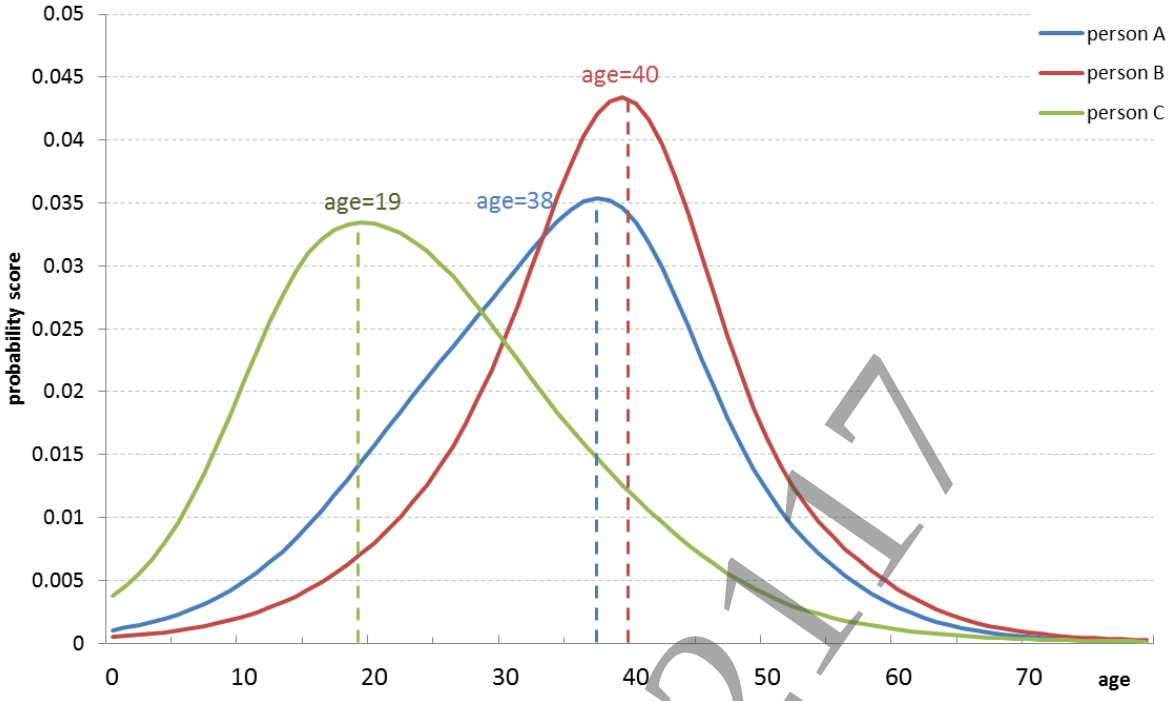
This function integrates the age range estimate result to the probability distribution data. When age estimate results for a person can be obtained sequentially, as in movie inputs, this function can be used to integrate those results and calculate the age estimate probability distribution to make a more precise statistical age estimate.

For *ageRange*, set the pointer to the age estimate result obtained using sceFaceAgeRangeEstimate().

For *ageDistrData*, set the pointer to SceFaceAgeDistrData. The age range estimate result set to *ageRange* will be integrated with this data.

The highest probable age estimate derived from the integrated age estimate probability distribution data will be output to *ageResult*.

Figure 6 Age Estimate Probability Distribution Data



Notes

At the beginning of the statistical age estimate process, prepare a zero-cleared age estimate probability data and set it to `ageDistrData`. Subsequently, do not change `ageDistrData` and call this function repeatedly with the pointer to the newly-obtained age range estimate result set to `ageRange` each time.

In the beginning, the age estimate output to `ageResult` will be unstable and vary every time. However, this variance will become smaller as the process is repeated and it will become possible to obtain a highly probable age estimate.

It is also possible to carry out statistical age estimate for multiple faces in parallel. To do so, prepare an age estimate probability distribution data per face and integrate each person's age range estimate to the person's age estimate distribution data.

Examples

```
/* Read face recognition dictionary data file */
unsigned char ageDict[SCE_FACE_AGE_DICT_SIZE];
SceUID fd;
fd = sceIoOpen("host0:"SCE_FACE_AGE_DICT, SCE_O_RDONLY, 0);
sceIoRead(fd, ageDict, SCE_FACE_AGE_DICT_SIZE);
sceIoClose(fd);

unsigned char yImg[320*240];
SceFaceDetectionResult detectResult;
SceFacePartsResult resultPartsArray[SCE_FACE_PARTS_NUM_MAX];
SceFaceAgeRangeResult resultAge;
SceFaceAgeDistrData ageDistrData;
int numParts, age, ret;

/* Initialize age estimate probability distribution data */
memset(&ageDistData, 0, sizeof(SceFaceAgeDistrData));

/* Allocate work buffer for age range estimate */
int workAgeSize = sceFaceAgeGetWorkingMemorySize(320, 240, 320, ageDict);
void *workAgePtr = malloc(workAgeSize);

/* Execute age range estimate */
ret = sceFaceAgeRangeEstimate(
        yImg, 320, 240, 320,
        ageDict,
        &detectResult,
        resultPartsArray, numParts,
        &resultAge,
        workAgePtr, workAgeSize);
if (ret != SCE_OK) {
        printf("sceFaceAgeRangeEstimate() failed! (0x%08x)\n", ret);
        return;
}
printf("facial age range:  (%d ~ %d)\n",
        resultAge.minAge,
        resultAge.maxAge);
}

/* Integrate age range estimate result */
ret = sceFaceAgeRangeIntegrate(
        &resultAge,
        &ageDistrData,
        &age);
printf("facial age:  %d\n", age);
```

See Also

SceFaceAgeRangeResult,SceFaceAgeDistrData

sceFaceAttributeGetWorkingMemorySize

Calculate the size of working memory for attribute classification

Definition

```
#include <libface.h>
int sceFaceAttributeGetWorkingMemorySize(
        int width,
        int height,
        int rowstride,
        const SceFaceAttribDictPtr attribDictPtr,
)
```

Arguments

width	Input image width for attribute classification [pixels]
height	Input image height for attribute classification [pixels]
rowstride	Input image data width for attribute classification
attribDictPtr	Pointer to attribute classification dictionary data

Return Values

Returns the size of work buffer for attribute classification.

Returns 0 for errors.

Description

This function calculates the size of the work buffer for attribute classification.

Allocate the memory with the size returned by this function, then call `sceFaceAttribute()` with the pointer to the work buffer and its size to execute attribute classification.

If this function returns 0, the attribute classification dictionary data is invalid or parameters are invalid.

Notes

The work buffer for attribute classification can be the same buffer as the one for face detection, since the size of work buffer for attribute classification is generally smaller than the one for face detection (however, this can be done only when the input image that is used for attribute classification is the same as the one used for face detection).

See Also

`sceFaceAttribute()`

sceFaceAgeGetWorkingMemorySize

Calculate the size of working memory for age range estimate

Definition

```
#include <libface.h>
int sceFaceAgeGetWorkingMemorySize(
        int width,
        int height,
        int rowstride,
        const SceFaceAgeDictPtr ageDictPtr
)
```

Arguments

width	Input image width for age range estimate [pixels]
height	Input image height for age range estimate [pixels]
rowstride	Input image data width for age range estimate
ageDictPtr	Pointer to age range estimate dictionary data

Return Values

Returns the size of work buffer for age range estimate.

Returns 0 for errors.

Description

This function calculates the size of the work buffer for age range estimate.

Allocate the memory with the size returned by this function, then call `sceFaceAgeRangeEstimate()` with the pointer to the work buffer and its size to execute age range estimate.

If this function returns 0, the age range estimate dictionary data is invalid or parameters are invalid.

Notes

The work buffer for age range estimate can be the same buffer as that of face detection, since the size of the work buffer for age range estimate is generally smaller than the one for face detection (however, this can be done only when the input image that is used for age range estimate is the same as the one used for the face detection).

See Also

`sceFaceAgeRangeEstimate()`

SCE_FACE_ATTRIB_SMILE_DICT

Smile attribute classification dictionary data file

Definition

```
#include <libface.h>
#define SCE_FACE_ATTRIB_SMILE_DICT "face_attrib_smile.adt"
#define SCE_FACE_ATTRIB_SMILE_DICT_SIZE (25064)
```

Description

These constants represent the filename and size (in bytes) of dictionary data used in attribute classification.

The *attribDictPtr* argument of the attribute classification execution function sceFaceAttribute() should be set with a pointer to the memory area where this file was read.

Using this dictionary enables classification processing to be performed to determine the degree of a smile.

Example

```
SceFaceAttribDictPtr attribDict =
        (SceFaceAttribDictPtr)malloc(SCE_FACE_ATTRIB_SMILE_DICT_SIZE);
SceUID fd = sceIoOpen("host0:"SCE_FACE_ATTRIB_SMILE_DICT, SCE_O_RDONLY, 0);
sceIoRead(fd, attribDict, SCE_FACE_ATTRIB_SMILE_DICT_SIZE);
sceIoClose(fd);

ret = sceFaceAttribute(
        yImg, 320, 240, 320,
        attribDict,
        &detectResult[i],
        resultPartsArray, numParts,
        resultAttribArray, SCE_FACE_ATTRIB_NUM_MAX, &numAttrib,
        workMemory, workMemorySize);
```

See Also

SceFaceAttribDictPtr, sceFaceAttribute()

SCE_FACE_ATTRIB_DICT

All attribute classification dictionary data file

Definition

```
#include <libface.h>
#define SCE_FACE_ATTRIB_DICT "face_attrib.adt"
#define SCE_FACE_ATTRIB_DICT_SIZE (124368)
```

Description

These constants represent the filename and size (in bytes) of dictionary data used in attribute classification.

The *attribDictPtr* argument of the attribute classification execution function sceFaceAttribute() should be set with a pointer to the memory area where this file was read.

Using this dictionary enables classification processing to be performed to determine all attributes (degree of smile, degree of eye opening, gender, age groups (baby, child or adult, or elderly person), glasses).

Example

```
SceFaceAttribDictPtr attribDict =
        (SceFaceAttribDictPtr)malloc(SCE_FACE_ATTRIB_DICT_SIZE);
SceUID fd = sceIoOpen("host0:"SCE_FACE_ATTRIB_DICT, SCE_O_RDONLY, 0);
sceIoRead(fd, attribDict, SCE_FACE_ATTRIB_DICT_SIZE);
sceIoClose(fd);

ret = sceFaceAttribute(
        yImg, 320, 240, 320,
        attribDict,
        &detectResult[i],
        resultPartsArray, numParts,
        resultAttribArray, SCE_FACE_ATTRIB_NUM_MAX, &numAttrib,
        workMemory, workMemorySize);
```

See Also

SceFaceAttribDictPtr, sceFaceAttribute()

SCE_FACE_AGE_DICT

Age range estimate dictionary data file

Definition

```
#include <libface.h>
#define SCE_FACE_AGE_DICT "face_age.adt"
#define SCE_FACE_AGE_DICT_SIZE (921304)
```

Description

These constants represent the filename and size (in bytes) of dictionary data used in age range estimate.

The *ageDictPtr* argument of the age range estimate execution function `sceFaceAgeRangeEstimate()` should be set with a pointer to the memory area where this file was read.

Using this dictionary enables age range estimate within a range of 10 years.

Examples

```
SceFaceAgeDictPtr ageDict =
        (SceFaceAgeDictPtr)malloc(SCE_FACE_AGE_DICT_SIZE);
SceUID fd = sceIoOpen("host0:"SCE_FACE_AGE_DICT, SCE_O_RDONLY, 0);
sceIoRead(fd, ageDict, SCE_FACE_AGE_DICT_SIZE);
sceIoClose(fd);

ret = sceFaceAgeRangeEstimate(
        yImg, 320, 240, 320,
        ageDict,
        &detectResult[i],
        resultPartsArray, numParts,
        &resultAge,
        workMemory, workMemorySize);
```

See Also

`SceFaceAgeDictPtr`, `sceFaceAgeRangeEstimate()`

Attribute Classification and Age Estimate Macro

Constants used in attribute classification and age estimate

Definition

```
#include <libface.h>
#define SCE_FACE_ATTRIB_NUM_MAX 8
#define SCE_FACE_AGE_BIN_SIZE 81
```

Description

This is the constant used in attribute classification.

Value	Value	Description
SCE_FACE_ATTRIB_NUM_MAX	8	Maximum number of attribute to classify
SCE_FACE_AGE_BIN_SIZE	81	Age range of age estimate probability distribution data

See Also

sceFaceAttribute(),sceFaceAgeRangeEstimate(),sceFaceAgeRangeIntegrate()

Face Identification APIs

SceFaceIdentifyDictPtr

Face identification dictionary data pointer type

Definition

```
#include <libface.h>
typedef void *SceFaceIdentifyDictPtr;
```

Description

This defined type is used as a pointer to dictionary data used in face identification.

It is used for an argument type of the functions `sceFaceIdentifyGetFeature()` and `sceFaceIdentifySimilarity()`.

See Also

`sceFaceIdentifyGetFeature()`,`sceFaceIdentifySimilarity()`

SceFaceIdentifyFeature

Face feature data for face identification

Definition

```
#include <libface.h>
typedef struct SceFaceIdentifyFeature {
        unsigned char data[SCE_FACE_IDENTIFY_FEATURE_SIZE];
} SceFaceIdentifyFeature;
```

Members

data Face feature data

Description

This data type represents the face feature data used in face identification.

When `sceFaceIdentifyGetFeature()` is called, the calculated face feature data is written as this structure type in the area set in the *resultFeature* argument.

See Also

`sceFaceIdentifyGetFeature(),sceFaceIdentifySimilarity()`

sceFaceIdentifyGetFeature

Calculate face feature data for face identification

Definition

```
#include <libface.h>
int sceFaceIdentifyGetFeature(
        const unsigned char *imgPtr,
        int width,
        int height,
        int rowstride,
        const SceFaceIdentifyDictPtr identifyDictPtr,
        const SceFaceDetectionResult *detectedFace,
        const SceFacePartsResult detectedPartsArray[],
        int detectedPartsNum,
        SceFaceIdentifyFeature *resultFeature,
        void* workMemory,
        int workMemorySize
)
```

Arguments

imgPtr	Pointer to input image (8-bit grayscale)
width	Input image width [pixels]
height	Input image height [pixels]
rowstride	Input image data width
identifyDictPtr	Pointer to face identification dictionary data
detectedFace	Pointer to face detection results
detectedPartsArray	Pointer to parts detection results
detectedPartsNum	Number of detected parts
resultFeature	Pointer to face feature data output area
workMemory	Pointer to work buffer
workMemorySize	Size of work buffer

Return Values

Returns SCE_OK(0) if processing is successful.

Returns one of the following error codes (a negative value) for errors.

Value	(Hexadecimal)	Description
SCE_FACE_ERROR_NO_MEMORY	0x808B0001	Not enough memory to run, or *workMemory* is NULL
SCE_FACE_ERROR_INVALID_PARAM	0x808B0002	Parameter is invalid
SCE_FACE_ERROR_INVALID_DICT	0x808B0003	Face identification dictionary data is invalid, or *identifyDictPtr* is NULL
SCE_FACE_ERROR_IMPERF_PARTS	0x808B0004	Parts are incomplete
SCE_FACE_ERROR_OUT_OF_RANGE	0x808B0005	Face angle is out of range

Description

This function uses face detection results and parts detection results for the eyes, nose, and mouth to calculate face feature data for face identification.

The input image that is set in *imgPtr* must be the same as the one used in parts detection.

For *identifyDictPtr*, set a pointer to the memory area where the face identification dictionary data file SCE_FACE_IDENTIFY_DICT was read.

The face detection results that are set in *detectedFace* must be the same results as those input to parts detection.

The parts detection results set in *detectedPartsArray* must contain normal detection results of all four parts (left and right eyes, nose, and mouth), and *detectedPartsNum* must be SCE_FACE_PARTS_NUM_MAX(4).

If the parts detection results that were set do not satisfy these conditions and calculating the face feature cannot be performed, an SCE_FACE_ERROR_IMPERF_PARTS error will be returned.

For *resultFeature*, set a pointer to SceFaceIdentifyFeature type data.

When the face feature data calculation is performed normally, the face feature data is output to the area specified by *resultFeature*.

For *workMemory*, set the pointer to the allocated memory whose size is calculated by sceFaceIdentifyGetWorkingMemorySize() or more.

For *workMemorySize*, set the size of *workMemory*.

Notes

If the input image shooting conditions were not ideal and the face parts are dark due to backlighting or insufficient light or if, on the other hand, the face is overexposed and is whitened out, or if the image contains a lot of noise, the quality of face feature data may worsen even if processing is performed normally by the program. Adjust the input image quality as needed before calling this function.

The quality of face feature data tends to get worse, the more the face is offset from the frontal direction (as the face angle increases).

Also, due to a cause of image quality etc., an offset from a real part position may end up being incorrectly recognized as the actual part position without an error occurring in the program, and the face feature data calculation result in that case may also differ from the true result.

Example

```
/* Read face recognition dictionary data file */
unsigned char detectDict[SCE_FACE_DETECT_FRONTAL_DICT_SIZE];
unsigned char partsDict[SCE_FACE_PARTS_ROLL_DICT_SIZE];
unsigned char identifyDict[SCE_FACE_IDENTIFY_DICT_SIZE];
SceUID fd;
fd = sceIoOpen("host0:"SCE_FACE_DETECT_FRONTAL_DICT, SCE_O_RDONLY, 0);
sceIoRead(fd, detectDict, SCE_FACE_DETECT_FRONTAL_DICT_SIZE);
sceIoClose(fd);
fd = sceIoOpen("host0:"SCE_FACE_PARTS_ROLL_DICT, SCE_O_RDONLY, 0);
sceIoRead(fd, partsDict, SCE_FACE_PARTS_ROLL_DICT_SIZE);
sceIoClose(fd);
fd = sceIoOpen("host0:"SCE_FACE_IDENTIFY_DICT, SCE_O_RDONLY, 0);
sceIoRead(fd, identifyDict, SCE_FACE_IDENTIFY_DICT_SIZE);
sceIoClose(fd);

unsigned char yImg[320*240];
SceFaceDetectionResult detectResult[16];
SceFacePartsResult resultPartsArray[SCE_FACE_PARTS_NUM_MAX];
```

```
SceFaceIdentifyFeature resultFeature;
int i, j, numFace, numParts, ret;

/* Allocate work buffer for face detection */
int workSize = sceFaceDetectionGetWorkingMemorySize(320, 240, 320, detectDict);
void *workPtr = malloc(workSize);

/* Allocate work buffer for parts detection */
int workPartsSize = sceFacePartsGetWorkingMemorySize(320, 240, 320, partsDict);
void *workPartsPtr = malloc(workPartsSize);

/* Allocate work buffer for face feature calculation */
int workIdentifySize = sceFaceIdentifyGetWorkingMemorySize(
        320, 240, 320, identifyDict);
void *workIdentifyPtr = malloc(workIdentifySize);

/* Execute face detection */
ret = sceFaceDetection(
        yImg, 320, 240, 320,
        detectDict,
        0.4f, 0.841f, 0.0f, 2, 2, 0.5f,
        SCE_FACE_DETECT_RESULT_NORMAL,
        detectResult, 16, &numFace,
        workPtr, workSize);
if (ret != SCE_OK) {
        printf("sceFaceDetection() failed! (0x%08x)\n", ret);
        return;
}
for (i = 0; i < numFace; i++) {
        /* Execute parts detection */
        ret = sceFaceParts(
              yImg, 320, 240, 320,
              partsDict,
              1, 1, &detectResult[i],
              resultPartsArray, SCE_FACE_PARTS_NUM_MAX, &numParts,
              workPartsPtr, workPartsSize);
        if (ret != SCE_OK) {
              printf("sceFaceParts() failed! (0x%08x)\n", ret);
              return;
        }
        /* Calculate face feature data */
        ret = sceFaceIdentifyGetFeature(
              yImg, 320, 240, 320,
              identifyDict,
              &detectResult[i],
              resultPartsArray, numParts,
              &resultFeature,
              workIdentifyPtr, workIdentifySize);
        if (ret != SCE_OK) {
              printf("sceFaceIdentifyGetFeature() failed! (0x%08x)\n", ret);
              return;
        }
}
```

See Also

SceFaceIdentifyDictPtr, SCE_FACE_IDENTIFY_DICT, SceFaceDetectionResult, SceFacePartsResult, SceFaceIdentifyFeature, sceFaceIdentifyGetWorkingMemorySize()

sceFaceIdentifySimilarity

Calculate the similarity of face feature data for face identification

Definition

```
#include <libface.h>
int sceFaceIdentifySimilarity(
        const SceFaceIdentifyFeature *extractedFeature,
        const SceFaceIdentifyFeature registeredFeatureArray[],
        int registeredFeatureNum,
        const SceFaceIdentifyDictPtr identifyDictPtr,
        float *maxScore,
        int *maxScoreId,
        float resultScoreArray[]
)
```

Arguments

extractedFeature	Pointer to face feature data
registeredFeatureArray	Pointer to array of registered face feature data
registeredFeatureNum	Number of elements in *registeredFeatureArray*
identifyDictPtr	Pointer to face identification dictionary data
maxScore	Highest similarity score
maxScoreId	Index number of registered face feature data with the highest similarity
resultScoreArray	Pointer to area to output similarity scores

Return Values

Returns SCE_OK(0) if processing is successful.

Returns one of the following error codes (a negative value) for errors.

Value	(Hexadecimal)	Description
SCE_FACE_ERROR_INVALID_PARAM	0x808B0002	Parameter is invalid
SCE_FACE_ERROR_INVALID_DICT	0x808B0003	Face identification dictionary data is invalid, or *identifyDictPtr* is NULL

Description

This function compares the similarity between face feature data of a certain face and face features registered in advance for comparison, and calculates similarities.

For *extractedFeature*, set a pointer to the face feature data to be calculated similarity.

For *registeredFeatureArray*, set a pointer to an array of registered face feature data, and set the number of elements in *registeredFeatureArray* to *registeredFeatureNum*.

For *identifyDictPtr*, set a pointer to the memory area where the face identification dictionary data file SCE_FACE_IDENTIFY_DICT was read.

For *resultScoreArray*, set a pointer to an area of sizeof(float)*registeredFeatureNum or more to output the similarity scores, or NULL(0). When NULL(0) is set to *resultScoreArray*, the index number of the registered face with the most similar face feature data is stored in **maxScoreId* and the score in **maxScore*. When a pointer to an output area is set to *resultScoreArray*, all the similarity score (in comparison with the registered face feature data) are output to the specified data.

The maximum similarity score that can be output by this function is +100.0. The threshold value recommended for general-purpose use in face identification is +5.0.

Example

```
/* Read face recognition dictionary data file */
unsigned char identifyDict[SCE_FACE_IDENTIFY_DICT_SIZE];
SceUID fd;
fd = sceIoOpen("host0:"SCE_FACE_IDENTIFY_DICT, SCE_O_RDONLY, 0);
sceIoRead(fd, identifyDict, SCE_FACE_IDENTIFY_DICT_SIZE);
sceIoClose(fd);

/* Read registered face feature data file */
SceFaceIdentifyFeature registeredFeature[10];
fd = sceIoOpen("host0:FaceReg.dat", SCE_O_RDONLY, 0);
sceIoRead(fd, registeredFeature, sizeof(registeredFeature));
sceIoClose(fd);

SceFaceIdentifyFeature resultFeature;
int i, numFace, ret;

/* Execute face detection (omitted) */
for (i = 0; i < numFace; i++) {
        /* Execute parts detection (omitted) */
        /* Calculate face feature data (omitted) */
        /* Calculate similarity of face feature data */
        float maxScore;
        int maxScoreId;
        ret = sceFaceIdentifySimilarity(
              &resultFeature,
              registeredFeature, 10,
              identifyDict,
              &maxScore, &maxScoreId, NULL);
        if (maxScore > 5.0f) {
              printf("id:%d is found\n", maxScoreId);
        } else {
              printf("unknown face is found\n");
        }
}
```

See Also

SceFaceIdentifyDictPtr,SCE_FACE_IDENTIFY_DICT,SceFaceIdentifyFeature

sceFaceIdentifyGetWorkingMemorySize

Calculate the size of working memory for face feature calculation

Definition

```
#include <libface.h>
int sceFaceIdentifyGetWorkingMemorySize(
        int width,
        int height,
        int rowstride,
        const SceFaceIdentifyDictPtr identifyDictPtr,
)
```

Arguments

width	Input image width for face feature calculation [pixels]
height	Input image height for face feature calculation [pixels]
rowstride	Input image data width for face feature calculation
identifyDictPtr	Pointer to face identification dictionary data

Return Values

Returns the size of work buffer for face feature calculation.

Returns 0 for errors.

Description

This function calculates the size of the work buffer for face feature calculation.

Allocate the memory with the size returned by this function, then call `sceFaceIdentifyGetFeature()` with the pointer to the work buffer and its size to execute face feature calculation.

If this function returns 0, the face identification dictionary data is invalid or parameters are invalid.

Notes

The internal work buffer for face feature calculation can be same buffer with the one for face detection, since the size of the work buffer for the feature calculation is generally smaller than the one for face detection (however, this can be done only when the input image that is used for face feature calculation is the same as the one used for face detection).

See Also

`sceFaceIdentifyGetFeature()`

SCE_FACE_IDENTIFY_DICT

Face identification dictionary data file

Definition

```
#include <libface.h>
#define SCE_FACE_IDENTIFY_DICT "face_identify.idt"
#define SCE_FACE_IDENTIFY_DICT_SIZE (154736)
```

Description

These constants represent the filename and size (in bytes) of dictionary data used in face identification.

The *identifyDictPtr* argument of the face feature data calculation function `sceFaceIdentifyGetFeature()` and similarity of face feature data calculation function `sceFaceIdentifySimilarity()` should be set with a pointer to the memory area where this file was read.

This dictionary is used for both the face feature data calculation and similarity calculation.

Example

```
SceFaceIdentifyDictPtr identifyDict =
        (SceFaceIdentifyDictPtr)malloc(SCE_FACE_IDENTIFY_DICT_SIZE);
SceUID fd = sceIoOpen("host0:"SCE_FACE_IDENTIFY_DICT, SCE_O_RDONLY, 0);
sceIoRead(fd, identifyDict, SCE_FACE_IDENTIFY_DICT_SIZE);
sceIoClose(fd);

ret = sceFaceIdentifyGetFeature(
        yImg, 320, 240, 320,
        identifyDict,
        &detectResult[i],
        resultPartsArray, numParts,
        resultFeature,
        workMemory, workMemorySize);

ret = sceFaceIdentifySimilarity(
        &resultFeature,
        registeredFeature, 10,
        identifyDict,
        &maxScore, &maxScoreId, NULL);
```

See Also

```
SceFaceIdentifyDictPtr,sceFaceIdentifyGetFeature(),
sceFaceIdentifySimilarity()
```

Face Identification Macro

Constants used in face identification

Definition

```
#include <libface.h>
#define SCE_FACE_IDENTIFY_FEATURE_SIZE 4096
```

Description

This is the constant used in face identification.

Value	Value	Description
SCE_FACE_IDENTIFY_FEATURE_SIZE	4096	Data size of the face feature data

See Also

SceFaceIdentifyFeature, sceFaceIdentifyGetFeature()

Face Fitting/Tracking APIs

SceFaceShapeModelDictPtr

Face shape model dictionary data pointer type

Definition

```
#include <libface.h>
typedef void *SceFaceShapeModelDictPtr;
```

Description

This defined type is used as a pointer to the face shape model dictionary data used for face fitting/tracking.

It is used for an argument type of the `sceFaceShapeFit()` and `sceFaceShapeTrack()` functions.

See Also

`sceFaceShapeFit()`,`sceFaceShapeTrack()`

SceFaceShapeResult

Face shape data for face identification

Definition

```
#include <libface.h>
typedef struct tagSceFaceShapeResult {
        int modelID;
        int pointNum;
        float pointX[SCE_FACE_SHAPE_POINT_NUM_MAX];
        float pointY[SCE_FACE_SHAPE_POINT_NUM_MAX];
        int isLost[SCE_FACE_SHAPE_POINT_NUM_MAX];
        float fourPointX[SCE_FACE_PARTS_NUM_MAX];
        float fourPointY[SCE_FACE_PARTS_NUM_MAX];
        float rectCenterX;
        float rectCenterY;
        float rectWidth;
        float rectHeight;
        float faceRoll;
        float facePitch;
        float faceYaw;
        float score;
        unsigned char data[4096];
} SceFaceShapeResult;
```

Members

modelID	Face shape data's ID
pointNum	Number of points in face shape data
pointX	Array of each point's x coordinate in face shape data
pointY	Array of each point's y coordinate in face shape data
isLost	Array of flags indicating loss for each points in face shape data
fourPointX	x coordinates for the 4 points of a face
fourPointY	y coordinates for the 4 points of a face
rectCenterX	Center x coordinate of the face rectangular area
rectCenterY	Center y coordinate of the face rectangular area
rectWidth	Width of the face rectangular area
rectHeight	Height of the face rectangular area
faceRoll	Planar (roll axis) rotation angle of face in radians
facePitch	Upward or vertical (pitch axis) rotation angle of face in radians
faceYaw	Sideways or horizontal (yaw axis) rotation angle of face in radians
score	Validity score of detected face shape
data	Data area for internal processing

Description

This data type represents the face fitting/tracking result. When calling `sceFaceShapeFit()` or `sceFaceShapeTrack()`, the calculated face shape result will be stored in the form of this structure type to the area set by the *shape* argument.

For *modelID*, the face shape data's ID will be output. For *pointNum*, the number of points in the face shape data will be output.

For *pointX*/*pointY*, the x/y coordinate of each point in the face shape data will be output as an array with a *pointNum* number of elements.

For *isLost*, the flag indicating the loss of each point in the face shape data will be stored as an array of *pointNum* number of elements.

For *fourPointX*/*fourPointY*, the x/y coordinates of the 4 points of a face will be stored as an array. These 4 points are the right eye, left eye, tip of the nose, and mouth, and the coordinates are stored to the array in this order.

The x/y coordinates of the center point of the rectangle area covering the face is represented by *rectCenterX*/*rectCenterY*. This rectangular area's width and height are represented by *rectWidth* and *rectHeight*.

2D coordinate values and width/height values are values normalized by the input image's width/height; for example, in coordinates, the upper left corner of the input image will correspond to (X, Y) = (0, 0) and the lower right will correspond to (1.0, 1.0).

For *faceRoll*, *facePitch*, and *faceYaw*, the rotation angle of the planar (roll axis), upward or vertical (pitch axis), and sideways or horizontal (yaw axis), respectively, will be output (in radians) as parameters of the face pose.

score represents the validity of the detected face shape. A value between 0 and 100 will be output. A higher value means higher validity. A score of 50 or lower may mean that the detected face shape is off.

Notes

Currently, only 1 ID is defined for *modelID*. Moreover, in current performance, the calculation precision of *faceRoll* is higher compared to *facePitch* and *faceYaw*.

See Also

`sceFaceShapeFit(),sceFaceShapeTrack()`

sceFaceShapeFit

Face fitting

Definition

```
#include <libface.h>
int sceFaceShapeFit(
        const unsigned char *imgPtr,
        int width,
        int height,
        int rowstride,
        const SceFaceShapeModelDictPtr shapeDictPtr,
        SceFaceShapeResult *shape,
        float lostThreshold,
        const SceFaceDetectionResult *detectedFace,
        const SceFacePartsResult detectedPartsArray[],
        int detectedPartsNum,
        void* workMemory,
        int workMemorySize
)
```

Arguments

imgPtr	Pointer to input image (8-bit grayscale)
width	Input image width [pixels]
height	Input image height [pixels]
rowstride	Input image data width
shapeDictPtr	Pointer to face fitting/tracking dictionary data
shape	Face shape data
lostThreshold	Validity threshold values for face shape data
detectedFace	Pointer to face detection result
detectedPartsArray	Pointer to parts detection result
detectedPartsNum	Number of detected parts
workMemory	Pointer to the work buffer
workMemorySize	Size of work buffer

Return Values

Returns SCE_OK(0) if processing is successful.

Returns one of the following error codes (a negative value) for errors.

Value	(Hexadecimal)	Description
SCE_FACE_ERROR_NO_MEMORY	0x808B0001	Not enough memory to run, or *workMemory* is NULL
SCE_FACE_ERROR_INVALID_PARAM	0x808B0002	Parameter is invalid
SCE_FACE_ERROR_INVALID_DICT	0x808B0003	Face fitting/tracking dictionary data is invalid or *shapeDictPtr* is NULL
SCE_FACE_ERROR_IMPERF_PARTS	0x808B0004	Parts are incomplete
SCE_FACE_ERROR_IMPERF_SHAPE	0x808B0006	Face shape is out of range

Description

This function executes face fitting using the face detection result and the eyes, nose, and mouth parts detection results.

The input image set to *imgPtr* must be the same as that used for parts detection.

For *shapeDictPtr*, set the pointer to the memory area to which the face fitting/tracking dictionary data file SCE_FACE_SHAPE_DICT_FRONTAL was read.

For *shape*, set a pointer to data of the SceFaceShapeResult type.

When face fitting completes successfully, the face shape result will be output to the area specified by *shape*.

For *lostThreshold*, set a value of SCE_FACE_SHAPE_SCORE_LOST_THRES_MIN or more and SCE_FACE_SHAPE_SCORE_LOST_THRES_MAX or less. The recommended value is SCE_FACE_SHAPE_SCORE_LOST_THRES_DEFAULT or a value within the range of plus or minus 5 of it. If the value of *score* for *shape* subcedes the value of *lostThreshold*, the SCE_FACE_ERROR_IMPERF_SHAPE error will return. Basically, the higher the *lostThreshold* value, the less likely that an incorrect face shape will be output; however, if the output *score* value of *shape* is low (despite the detected face shape being correct) because of the face pose or because the face is wearing glasses, or because of individual differences, set the threshold to a lower value.

The face detection result set to *detectedFace* must be the same as that used for the input of parts detection.

The parts detection result set to *detectedPartsArray* must be the result of a process in which the eyes, nose, and mouth were all successfully detected, and *detectedPartsNum* must be SCE_FACE_PARTS_NUM_MAX(4).

If the set parts detection result does not meet these conditions and face feature cannot be calculated, SCE_FACE_ERROR_IMPERF_PARTS will return.

For *workMemory*, set a pointer to a memory area of a size calculated by sceFaceIdentifyGetWorkingMemorySize() or more.

For *workMemorySize*, set the size of the memory area allocated in *workMemory*.

Notes

If the input image shooting conditions were not ideal and the face parts are dark due to backlighting or insufficient light or if, on the other hand, the face is overexposed and is whitened out, or if the image contains a lot of noise, the quality of face shape data may worsen even if processing is performed normally by the program. Adjust the input image quality as needed before calling this function.

The quality of face shape data tends to get worse, the more the face is offset from the frontal direction (as the face angle increases).

Also, due to a cause of image quality etc., an offset from a real part position may end up being incorrectly recognized as the actual part position without an error occurring in the program, and the face shape data calculation result in that case may also differ from the true result.

Examples

```
/* Read face recognition dictionary data file */
unsigned char detectDict[SCE_FACE_DETECT_FRONTAL_DICT_SIZE];
unsigned char partsDict[SCE_FACE_PARTS_ROLL_DICT_SIZE];
unsigned char shapeDict[SCE_FACE_SHAPE_DICT_FRONTAL_SIZE];
SceUID fd;
fd = sceIoOpen("host0:"SCE_FACE_DETECT_FRONTAL_DICT, SCE_O_RDONLY, 0);
sceIoRead(fd, detectDict, SCE_FACE_DETECT_FRONTAL_DICT_SIZE);
sceIoClose(fd);
fd = sceIoOpen("host0:"SCE_FACE_PARTS_ROLL_YAW_DICT, SCE_O_RDONLY, 0);
sceIoRead(fd, partsDict, SCE_FACE_PARTS_ROLL_YAW_DICT_SIZE);
sceIoClose(fd);
fd = sceIoOpen("host0:"SCE_FACE_SHAPE_DICT_FRONTAL, SCE_O_RDONLY, 0);
sceIoRead(fd, shapeDict, SCE_FACE_SHAPE_DICT_FRONTAL_SIZE);
sceIoClose(fd);

unsigned char yImg[320*240];
SceFaceDetectionResult detectResult[16];
SceFacePartsResult resultPartsArray[SCE_FACE_PARTS_NUM_MAX];
SceFaceShapeResult resultShape[16];
int i, j, numFace, numParts, ret;
float lostThreshold = SCE_FACE_SHAPE_SCORE_LOST_THRES_DEFAULT;

/* Allocate work buffer for face detection */
int workSize = sceFaceDetectionGetWorkingMemorySize(320, 240, 320, detectDict);
void *workPtr = malloc(workSize);

/* Allocate work buffer for parts detection */
int workPartsSize = sceFacePartsGetWorkingMemorySize(320, 240, 320, partsDict);
void *workPartsPtr = malloc(workPartsSize);

/* Allocate work buffer for face fitting */
int workShapeSize = sceFaceShapeGetWorkingMemorySize(
        320, 240, 320, shapeDict, 320, 240, false);
void *workShapePtr = malloc(workShapeSize);

/* Execute face detection */
ret = sceFaceDetection(
        yImg, 320, 240, 320,
        detectDict,
        0.4f, 0.841f, 0.0f, 2, 2, 0.5f,
        SCE_FACE_DETECT_RESULT_NORMAL,
        detectResult, 16, &numFace,
        workPtr, workSize);
if (ret != SCE_OK) {
        printf("sceFaceDetection() failed! (0x%08x)\n", ret);
        return;
}
for (i = 0; i < numFace; i++) {
        /* Execute parts detection */
        ret = sceFaceParts(
              yImg, 320, 240, 320,
              partsDict,
              1, 1, &detectResult[i],
              resultPartsArray, SCE_FACE_PARTS_NUM_MAX, &numParts,
              workPartsPtr, workPartsSize);
        if (ret != SCE_OK) {
              printf("sceFaceParts() failed! (0x%08x)\n", ret);
              return;
```

```
            }
            /* Face fitting */
            ret = sceFaceShapeFit(
                  yImg, 320, 240, 320,
                  shapeDict,
                  &resultShape[i], lostThreshold,
                  &detectResult[i],
                  resultPartsArray, numParts,
                  workShapePtr, workShapeSize);
            if (ret != SCE_OK) {
                  printf("sceFaceShapeFit () failed! (0x%08x)\n", ret);
                  return;
            }
      }
```

See Also

SceFaceShapeModelDictPtr, SceFaceDetectionResult, SceFacePartsResult, SceFaceShapeResult, sceFaceShapeGetWorkingMemorySize()

sceFaceShapeTrack

Face tracking

Definition

```
#include <libface.h>
int sceFaceShapeTrack(
        const unsigned char *imgPtrCur,
        const unsigned char *imgPtrPrv,
        int width,
        int height,
        int rowstride,
        const SceFaceShapeModelDictPtr shapeDictPtr,
        SceFaceShapeResult *shape,
        float lostThreshold,
        void* workMemory,
        int workMemorySize
)
```

Arguments

imgPtrCur	Pointer to the current input image (8-bit gray scale)
imgPtrPrv	Pointer to the input image of the immediately-preceding frame (8-bit gray scale)
width	Input image width [pixels]
height	Input image height [pixels]
rowstride	Input image data width
shapeDictPtr	Pointer to the face fitting/tracking dictionary data file
shape	Face shape data
lostThreshold	Validity threshold values for face shape data
workMemory	Pointer to the work buffer
workMemorySize	Size of work buffer

Return Values

Returns SCE_OK(0) if processing is successful.

Returns one of the following error codes (a negative value) for errors.

Value	(Hexadecimal)	Description
SCE_FACE_ERROR_NO_MEMORY	0x808B0001	Not enough memory to run, or *workMemory* is NULL
SCE_FACE_ERROR_INVALID_PARAM	0x808B0002	Parameter is invalid
SCE_FACE_ERROR_INVALID_DICT	0x808B0003	Face fitting/tracking dictionary data is invalid or *shapeDictPtr* is NULL
SCE_FACE_ERROR_IMPERF_SHAPE	0x808B0006	Face shape is out of range

Description

This function executes face tracking based on the face shape result of the immediately-preceding frame for movie inputs

Before executing this function, the face shape data of the immediately-preceding frame calculated by `sceFaceShapeFit()` or `sceFaceShapeTrack()` must be set to *`shape`*.

The input images set to *`imgPtrCur`* and *`imgPtrPrv`* must have the same *`width`* and *`height`*.

For *`shapeDictPtr`*, set the pointer to the memory area to which the face fitting/tracking dictionary data file `SCE_FACE_SHAPE_DICT_FRONTAL` was read.

For *`shape`*, set a pointer to data of the `SceFaceShapeResult` type.

As mentioned above, the face shape data of the immediately-preceding *`imgPtrPrv`* frame calculated by `sceFaceShapeFit()` or `sceFaceShapeTrack()` must be set to *`shape`*. After this function's execution, the face shape result of the current input frame (*`imgPtrCur`*) will be updated with the data of *`shape`*.

For *`lostThreshold`*, set a value of `SCE_FACE_SHAPE_SCORE_LOST_THRES_MIN` or more and `SCE_FACE_SHAPE_SCORE_LOST_THRES_MAX` or less. The recommended value is `SCE_FACE_SHAPE_SCORE_LOST_THRES_DEFAULT` or a value within the range of plus or minus 5 of it. If the value of *`score`* for *`shape`* subcedes the value of *`lostThreshold`*, the `SCE_FACE_ERROR_IMPERF_SHAPE` error will return. Basically, the higher the *`lostThreshold`* value, the less likely that an incorrect face shape will be output; however, if the output *`score`* value of *`shape`* is low (despite the detected face shape being correct) because of the face pose or because the face is wearing glasses, or because of individual differences, set the threshold to a lower value.

For *`workMemory`*, set a pointer to a memory area of a size calculated by `sceFaceShapeGetWorkingMemorySize()` or more.

For *`workMemorySize`*, set the size of the memory area allocated in *`workMemory`*.

Notes

If the input image shooting conditions were not ideal and the face parts are dark due to backlighting or insufficient light or if, on the other hand, the face is overexposed and is whitened out, or if the image contains a lot of noise, the quality of face shape data may worsen even if processing is performed normally by the program. Adjust the input image quality as needed before calling this function.

In current specifications, the quality of face shape data tends to get worse in cases such as, when glasses are worn, or the more the face is offset from the frontal direction (as the face angle increases).

Examples

```
/* Read face recognition dictionary data file */
unsigned char detectDict[SCE_FACE_DETECT_FRONTAL_DICT_SIZE];
unsigned char partsDict[SCE_FACE_PARTS_ROLL_DICT_SIZE];
unsigned char shapeDict[SCE_FACE_SHAPE_DICT_FRONTAL_SIZE];
SceUID fd;
fd = sceIoOpen("host0:"SCE_FACE_DETECT_FRONTAL_DICT, SCE_O_RDONLY, 0);
sceIoRead(fd, detectDict, SCE_FACE_DETECT_FRONTAL_DICT_SIZE);
sceIoClose(fd);
fd = sceIoOpen("host0:"SCE_FACE_PARTS_ROLL_YAW_DICT, SCE_O_RDONLY, 0);
sceIoRead(fd, partsDict, SCE_FACE_PARTS_ROLL_YAW DICT_SIZE);
sceIoClose(fd);
fd = sceIoOpen("host0:"SCE_FACE_SHAPE_DICT_FRONTAL, SCE_O_RDONLY, 0);
sceIoRead(fd, shapeDict, SCE_FACE_SHAPE_DICT_FRONTAL_SIZE);
sceIoClose(fd);

unsigned char yImg[160*120];
unsigned char yImgPrv[160*120];
SceFaceDetectionResult detectResult[16];
SceFacePartsResult resultPartsArray[SCE_FACE_PARTS_NUM_MAX];
SceFaceShapeResult resultShape[16];
int i, j, numFace, numParts, ret;
float lostThreshold = SCE_FACE_SHAPE_SCORE_LOST_THRES_DEFAULT;

/* Allocate work buffer for face detection */
int workSize = sceFaceDetectionGetWorkingMemorySize(320, 240, 320, detectDict);
void *workPtr = malloc(workSize);

/* Allocate work buffer for parts detection */
int workPartsSize = sceFacePartsGetWorkingMemorySize(320, 240, 320, partsDict);
void *workPartsPtr = malloc(workPartsSize);

/* Allocate work buffer for face fitting/tracking */
int workShapeSize = sceFaceShapeGetWorkingMemorySize(
        160, 120, 160, shapeDict, 160, 120,true);
void *workShapePtr = malloc(workShapeSize);

/* Execute face detection */
ret = sceFaceDetection(
        yImg, 160, 120, 160,
        detectDict,
        0.4f, 0.841f, 0.0f, 2, 2, 0.5f,
        SCE_FACE_DETECT_RESULT_NORMAL,
        detectResult, 16, &numFace,
        workPtr, workSize);
if (ret != SCE_OK) {
        printf("sceFaceDetection() failed! (0x%08x)\n", ret);
        return;
}
for (i = 0; i < numFace; i++) {
        /* Execute parts detection */
        ret = sceFaceParts(
              yImg, 160, 120, 160,
              partsDict,
              1, 1, &detectResult[i],
              resultPartsArray, SCE_FACE_PARTS_NUM_MAX, &numParts,
              workPartsPtr, workPartsSize);
        if (ret != SCE_OK) {
              printf("sceFaceParts() failed! (0x%08x)\n", ret);
```

```
                return;
            }
            /* Face fitting */
            ret = sceFaceShapeFit(
                yImg, 160, 120, 160,
                shapeDict,
                &resultShape[i], lostThreshold,
                &detectResult[i],
                resultPartsArray, numParts,
                workShapePtr, workShapeSize);
            if (ret != SCE_OK) {
                printf("sceFaceShapeFit () failed! (0x%08x)\n", ret);
                return;
            }
    }

    /* Store current frame as previous frame */
    memcpy(yImgPrv, yImg, 160*120);
    /* Input new frame */
    cameraUpdate(&yImg);

    for (i = 0; i < numFace; i++) {
            /* Face tracking */
            ret = sceFaceShapeTrack(
                yImg, yImgPrv, 160, 120, 160,
                shapeDict,
                &resultShape[i], lostThreshold,
                workShapePtr, workShapeSize);
            if (ret != SCE_OK) {
                printf("sceFaceShapeTrack () failed! (0x%08x)\n", ret);
                return;
            }
    }
```

See Also

SceFaceShapeModelDictPtr, SceFaceDetectionResult, SceFacePartsResult, SceFaceShapeResult, sceFaceShapeGetWorkingMemorySize()

sceFaceShapeGetWorkingMemorySize

Calculate the size of working memory for face fitting/tracking

Definition

```
#include <libface.h>
int sceFaceShapeGetWorkingMemorySize(
        int width,
        int height,
        int rowstride,
        const SceFaceShapeModelDictPtr shapeDictPtr,
        int maxFaceWidth,
        int maxFaceHeight,
        bool isVideoInput
)
```

Arguments

width	Width of the input image for face fitting/tracking [pixels]
height	Height of the input image for face fitting/tracking [pixels]
rowstride	Data width of the input image for face feature calculation
shapeDictPtr	Pointer to the face fitting/tracking dictionary data
maxFaceWidth	Maximum width [pixels] of the target face for face fitting/tracking
maxFaceHeight	Maximum height [pixels] of the target face for face fitting/tracking
isVideoInput	Flag to indicate whether the input image is a movie or still image. Specify true for movie and false for a still image.

Return Values

Returns the size of work buffer for face fitting/tracking calculation.

Returns 0 for errors.

Description

This function calculates the size of the work buffer for face fitting/tracking calculation.

Allocate the memory with the size returned by this function, then call `sceFaceShapeFit()` and `sceFaceShapeTrack()` with the pointer to the memory area and its size to execute face fitting/tracking.

If this function returns 0, the face shape dictionary data is invalid or parameters are invalid.

Notes

The internal work buffer for face fitting/tracking calculation can be the same buffer with the one for face detection, since the size of work buffer for face fitting/tracking calculation is generally smaller than the one for face detection (however, this can be done only when the input image that is used for face fitting/tracking calculation is the same as the one used for face detection).

See Also

`sceFaceIdentifyGetFeature()`

SCE_FACE_SHAPE_DICT_FRONTAL

Face fitting/tracking dictionary data file

Definition

```
#include <libface.h>
#define SCE_FACE_SHAPE_DICT_FRONTAL "face_shape_frontal.shp"
#define SCE_FACE_SHAPE_DICT_FRONTAL_SIZE (44560)
```

Description

These constants represent the filename and size (in bytes) of dictionary data used in face fitting/tracking.

The *shapeDictPtr* argument of the face fitting function `sceFaceShapeFit()` and the face tracking function `sceFaceShapeTrack()` should be set with a pointer to the memory area where this file was read.

This dictionary is used for both the face fitting and face tracking.

Examples

```
SceFaceShapeModelDictPtr shapeDict =
        (SceFaceShapeModelDictPtr)malloc(SCE_FACE_SHAPE_DICT_FRONTAL_SIZE);
SceUID fd = sceIoOpen("host0:"SCE_FACE_SHAPE_DICT_FRONTAL, SCE_O_RDONLY, 0);
sceIoRead(fd, shapeDict, SCE_FACE_SHAPE_DICT_FRONTAL SIZE);
sceIoClose(fd);
float threshold = SCE_FACE_SHAPE_SCORE_LOST_THRES_DEFAULT;

ret = sceFaceShapeFit(
        yImg, 160, 120, 160,
        shapeDict,
        &shape, threshold,
        &detectResult[i],
        resultPartsArray, numParts,
        workMemory, workMemorySize);

ret = sceFaceShapeTrack(
        yImg, yImgPrevious, 160, 120, 160,
        shapeDict,
        &shape, threshold,
        workMemory, workMemorySize);
```

See Also

SceFaceShapeModelDictPtr,sceFaceShapeFit(),sceFaceShapeTrack()

Face Fitting/Tracking Macro

Constants used for face fitting/tracking

Definition

```
#include <libface.h>

#define SCE_FACE_SHAPE_POINT_NUM_MAX (46)
#define SCE_FACE_SHAPE_POINT_NUM_FRONTAL (46)

#define SCE_FACE_SHAPE_MODEL_NUM_MAX (1)
#define SCE_FACE_SHAPE_MODEL_ID_FRONTAL (0)

#define SCE_FACE_SHAPE_SCORE_LOST_THRES_MIN (40)
#define SCE_FACE_SHAPE_SCORE_LOST_THRES_MAX (100)
#define SCE_FACE_SHAPE_SCORE_LOST_THRES_DEFAULT (55)
```

Description

Constants used in face identification.

Macro	Value	Description
SCE_FACE_SHAPE_POINT_NUM_MAX	46	Maximum value for the number of points in a face shape model
SCE_FACE_SHAPE_POINT_NUM_FRONTAL	46	Number of points for a frontal face shape model
SCE_FACE_SHAPE_MODEL_NUM_MAX	1	Number of face shape model definitions
SCE_FACE_SHAPE_MODEL_ID_FRONTAL	0	Frontal face shape model ID
SCE_FACE_SHAPE_SCORE_LOST_THRES_MIN	40	Minimum threshold score for face shape result evaluation
SCE_FACE_SHAPE_SCORE_LOST_THRES_MAX	100	Maximum threshold score for face shape result evaluation
SCE_FACE_SHAPE_SCORE_LOST_THRES_DEFAULT	55	Standard threshold scores to be used for face shape result evaluation

See Also

SceFaceShapeModelDictPtr,sceFaceShapeFit(),sceFaceShapeTrack()

Return Codes

Return Codes

List of return codes returned by libface

Definition

Value	(Hexadecimal)	Description
SCE_OK	0	Processing completed successfully
SCE_FACE_ERROR_NO_MEMORY	0x808B0001	Not enough memory to run
SCE_FACE_ERROR_INVALID_PARAM	0x808B0002	Parameter is invalid
SCE_FACE_ERROR_INVALID_DICT	0x808B0003	Dictionary data is invalid
SCE_FACE_ERROR_IMPERF_PARTS	0x808B0004	Parts are incomplete
SCE_FACE_ERROR_OUT_OF_RANGE	0x808B0005	Face angle is out of range
SCE_FACE_ERROR_IMPERF_SHAPE	0x808B0006	Face shape is out of range