libface Overview

PlayStation®Vita Programmer Tool Runtime Library Release 3.50

Table of Contents

1 Library Overview 3
Function Overview and Features 3
Used Resources 4
Embedding into a Program 4
Related Files 4
Sample Programs 5
2 Using the Library 7
Initialization 7
Termination 7
Face Detection Procedure 7
Parts Detection and Detailed Parts Detection Procedure 11
Attribute Classification Procedure 13
Age Estimate Procedure 16
Face Identification Procedure 19
Face Fitting Procedure 22
Face Tracking Procedure 25
List of Functions 28
List of Face Recognition Dictionaries 29
3 Notes 31
Processing Time and Memory Consumption 31
Misrecognition 31
Library Implementation 31

1 Library Overview

Function Overview and Features

libface is a library for analyzing grayscale images and recognizing the faces of people whose pictures appear in those images. The following functions are provided by libface.

- Detects a face (or faces) in a grayscale image
- Detects the eyes, nose and mouth of a detected face
- Detects the outlines of the eyes, pupils, eyebrows, nostrils, nose line, outline of the lips, and the outline of a detected face
- Classifies a detected face by the degree of its smile, how wide the eyes are open, gender, age group (baby, child/adult, elderly), and whether or not glasses are worn
- Estimates the age for a detected face
- Identifies individuals from a detected face
- Tracks the detected face

The following terms are used in libface.

- Face recognition:
  Overall recognition of faces in an image. In libface, face recognition includes face detection, parts detection, and the classification of face attributes.
- Face detection:
  Detection of face regions within an image.
- Parts detection:
  Detection of the eyes, nose and mouth from detected face regions.
- Detailed parts detection:
  Detection of the outlines of the eyes, pupils, eyebrows, nostrils, nose line, outline of the lips, and the outline of the face from detected face regions.
- Face attribute classification:
  Determination of the degree of smile, how wide the eyes are open, whether it is male or female, age group (baby, child/adult, elderly), and whether or not glasses are worn from detected face regions.
- Age estimate:
  Age is estimated based on a detected face area. There are two types of age estimating functionality: "age range estimate" where the minimum/maximum age is estimated and "statistical age estimate" where the results of the estimated age range are accumulated to derive an age estimate probability distribution data to make a more precise age estimate.
- Face identification:
  Face features of a detected face area are quantified and individuals can be identified by calculating the similarity between face feature values.
- Fitting/Tracking a face:
  The outline shapes of the eyes, nose, mouth, and eyebrows from the detected face regions are fitted/tracked.

Used Resources

libface uses the following system resources.

Resource	Description
Work memory	When executing face detection function (at VGA image), the application explicitly allocates 322 KiB When executing parts detection function, the application explicitly allocates 164 KiB When executing detail parts detection function, the application explicitly allocates 179 KiB When executing face attribute classification function, the application explicitly allocates 152 KiB When executing face identification function, the application explicitly allocates 211 KiB When executing face fitting/tracking (QQVGA size), the application explicitly allocates 68 KiB
Thread	none

Embedding into a Program

Include libface.h in the source program.

Upon building the program, link libSceFace_stub.a or libSceFace_stub_weak.a.

Related Files

The following files are required to use libface.

Filename	Description
libface.h	Header file
libSceFace_stub.a	Stub library file
libSceFace_stub_weak.a	Weak import stub library file

The following face recognition dictionary data files are used by libface.

These files are located in target/sce_data/libface/.

Filename	Description
face_detect_frontal.fdt	Face detection dictionary data file (for frontal face)
face_detect_roll.fdt	Face detection dictionary data file (roll support)
face_detect_yaw.fdt	Face detection dictionary data file (yaw support)
face_detect_pitch.fdt	Face detection dictionary data file (pitch support)
face_detect_roll_yaw.fdt	Face detection dictionary data file (roll and yaw support)
face_detect_roll_yaw_pitch.fdt	Face detection dictionary data file (roll, yaw, and pitch support)
face_parts_roll.pdt	Parts detection dictionary data file (frontal face with roll support)
face_parts_roll_yaw.pdt	Parts detection dictionary data file (roll and yaw support)
face_parts_check.cdt	Parts detection result validity check dictionary data file
face_allparts.pdt	Detailed parts detection dictionary file
face_allparts_shape.dat	Shape correction dictionary file (used within detailed parts detection)
face_attrib_smile.adt	Smile attribute classification dictionary data file
face_attrib.adt	All attribute classification dictionary data file
face_age.adt	Age range estimate dictionary data file
face_identify.idt	Face identification dictionary data file
face_shape_frontal.shp	Face fitting/tracking dictionary data file

Sample Programs

The following libface sample programs are available for your reference.

sample_code/engines/api_libface/face_detection_still/

This sample shows the basic way of using libface for face detection. Face detection is executed on jpeg/png files.

sample_code/engines/api_libface/parts_detection_still/

This sample shows the basic way of using libface for face detection, part detection, and detailed part detection. It executes face detection on jpeg/png files and then part detection and detailed part detection on the detected face areas.

sample_code/engines/api_libface/attribute_classify_still/

This sample shows the basic way of using libface for face detection, part detection, and face attribute classification. It executes face detection on jpeg/png files and then part detection and face attribute classification on the detected face areas.

sample_code/engines/api_libface/face_identify_still/

This sample shows the basic way of using libface for face detection, part detection, and face identification. It executes face detection on jpeg/png files and then part detection and face feature extraction on the detected face areas. The extracted face features are saved to the file, and used to calculate the similarity of registered face feature data and perform personal identification.

sample_code/engines/api_libface/attribute_classify_camera/

This sample executes face detection, part detection, and face attribute classification on input from cameras.

sample_code/engines/api_libface/age_estimate_camera/

This sample executes face detection, parts detection, and age estimate on input from cameras. A face of one person is tracked and the accumulated age range estimate results are calculated to make a more specific age estimate.

sample_code/engines/api_libface/face_recognition_camera_local_search/

This sample executes high-speed face detection, part detection, and face attribute classification on input from cameras.

This sample performs high-speed execution by using two threads. One thread (sub thread) is used for face detection on the entire input image (global search), and the other thread (main thread) uses the results of the last global search to execute face detection only in the areas where faces are most likely to be found (local search).

sample_code/engines/api_libface/face_recognition_calib/

This sample performs calibration by gradually changing the camera input parameter (EV/ExposureValue) so that the parameter is optimized for face detection.

Performing calibration will improve the performance of recognition in unfavorable conditions for face detection, such as backlighting.

sample_code/engines/api_libface/shape_fitting_still/

This sample shows the basic way of using libface for face fitting. Face detection is carried out from a jpeg/png file, and parts detection and face fitting are executed on each detected region.

sample_code/engines/api_libface/shape_fitting_camera/

This sample executes face detection, parts detection, and face fitting/tracking on input from cameras.

Notes on the Samples

Assuming that the input image from the camera will be used only for face recognition, the camera should be set to the lowest resolution QQVGA (160x120) to achieve extremely fast performance. The smaller the image size is, the less possibility there will be of a data cache miss occurrence and the faster the entire program will run, including face recognition.

If the camera is set to high resolution, the image size, memory consumption and processing time will all increase.

Since a grayscale image is required for face recognition, libface uses `SCE_CAMERA_FORMAT_YUV420_PLANE` camera input to capture the YUV420 image, and libface directly uses the Y-image at the beginning of that data. The captured YUV420 image is converted to RGBA for displaying the camera image and the result of the face recognition by using a graphics shader.

A feature of self-portrait image detection is the assumption that a face will be large and near the center of the image. The parameters of the face recognition functions in the sample programs are set so the programs run quickly under those image conditions, so only large faces in an image will be detected. If the parameters are set to detect small faces, the face detection time will increase.

2 Using the Library

Initialization

Copy sdk/target/sce_module/libface.suprx to a location that can be accessed from a program with app0:sce_module/libface.suprx.

Before calling the library functions, load PRX using the following code.

```
sceSysmoduleLoadModule(SCE_SYSMODULE_FACE);
```

No initialization is necessary to use the libface library. However, since face recognition dictionary data is required to execute face recognition, it is recommended that the required face recognition dictionary data files be read in advance as part of initialization. Also, it is recommended to allocate the working memory area to execute face recognition, depending on the size of the input image and the type of dictionary.

Termination

libface has no termination function. If face recognition dictionary data is read, or working memory area is allocated as part of initialization, freeing the memory area should be performed as part of termination

Lastly, unload PRX using the following code.

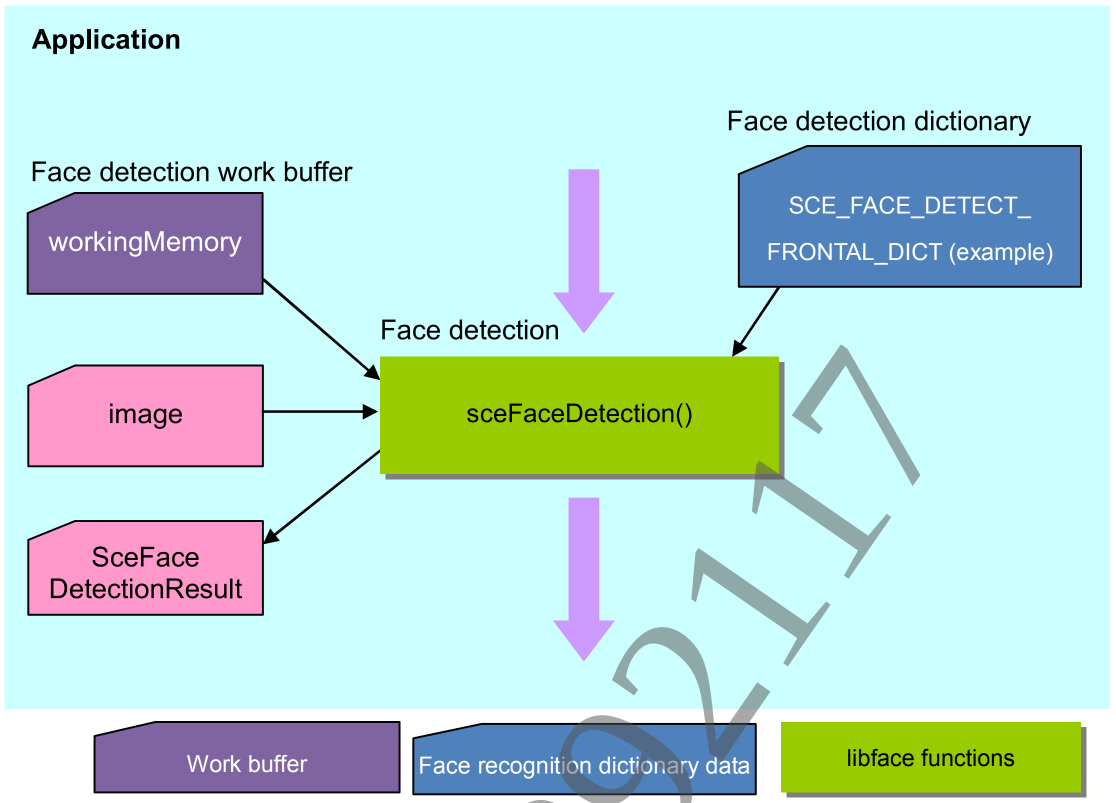
```
sceSysmoduleUnloadModule(SCE_SYSMODULE_FACE);
```

Face Detection Procedure

A grayscale image is required as the input data for face detection.

To execute face detection, select a face detection dictionary suitable for the application and pass a pointer to it as an argument. The face detection dictionary must be read onto memory in advance from a face detection dictionary data file. Also, the working memory area must be allocated depending on the size of the input image and the type of face detection dictionary.

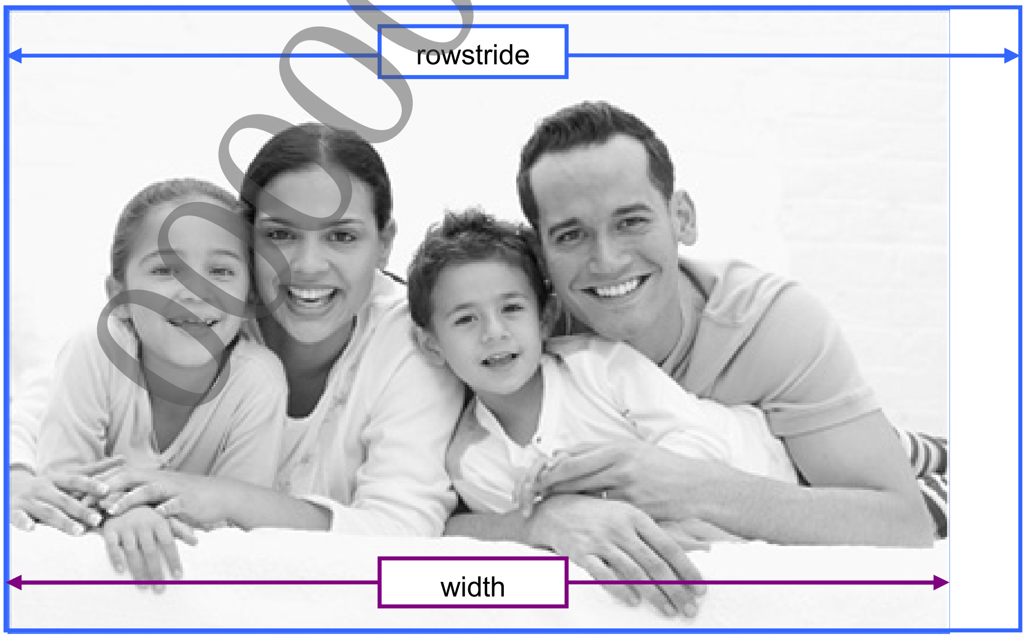
Figure 1 Face Detection Procedure


Application
Face detection dictionary
SCE_FACE_DETECT_ FRONTAL_DICT (example)
Face detection work buffer
workingMemory
Face detection
sceFaceDetection()
image
SceFace DetectionResult
Work buffer
Face recognition dictionary data
libface functions

(1) Prepare the input image

The input image for face recognition is an 8-bit grayscale image.

Figure 2 Input Image Width and Data Width



(2) Prepare the face detection dictionary data and the working memory area

Read the appropriate face detection dictionary data file onto memory.

The size of the working memory area for face detection can be obtained by setting the size of the input image and the face detection dictionary data as arguments to `sceFaceDetectionGetWorkingMemorySize()`. Allocate the memory area according to the size obtained.

(3) Prepare the output area for the face detection results

Prepare an array of the `SceFaceDetectionResult` structure as the area for saving the output results from face detection. The number of elements in this array can be set as desired to match the number of faces to be detected. For example, when it is sufficient to detect a maximum of 10 people's faces from one image, prepare a `SceFaceDetectionResult` structure array with 10 elements.

Note
If the number of faces to be detected is decreased, memory will be conserved but it will still take the same amount of time for face detection.

(4) Execute face detection

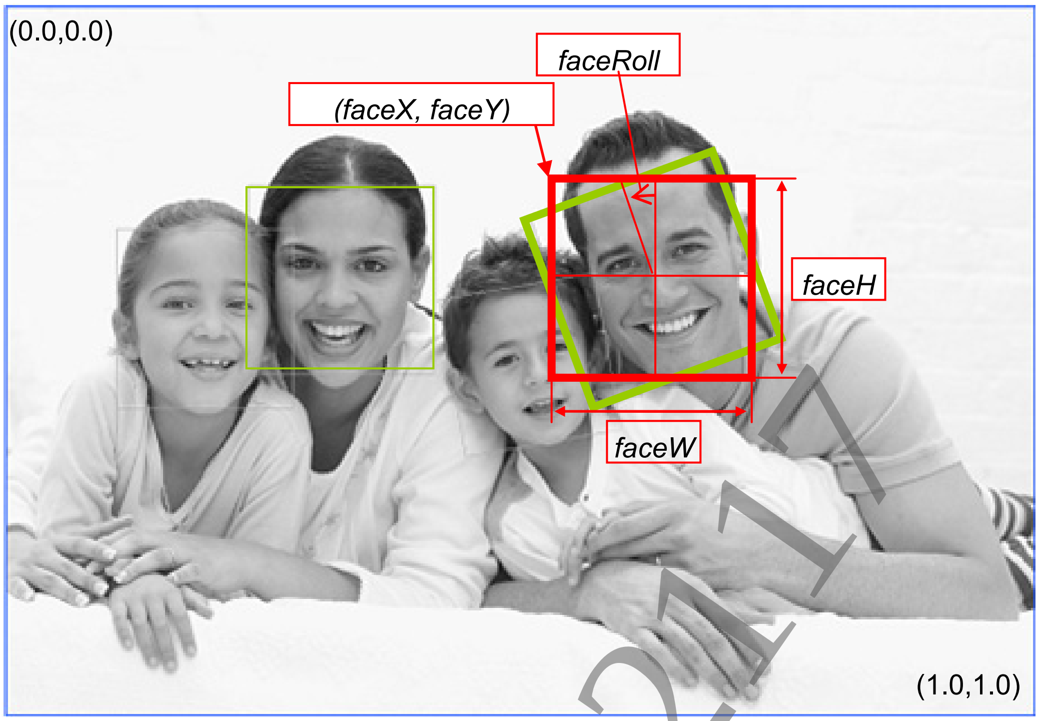
Set information about the input image, the address of the face detection dictionary data, address of the face detection result output area, number of elements in the face detection result output array, other face detection parameters, and the size and address of the working memory area as arguments and execute `sceFaceDetection()` or `sceFaceDetectionEx()`.

(5) Receive the face detection results

When `sceFaceDetection()` or `sceFaceDetectionEx()` completes successfully, the number of detected faces is returned in *`resultFaceNum`*. The individual face detection results are stored in each of the elements of the `SceFaceDetectionResult` structure array that was prepared in (3) and whose address was set in the argument *`resultFaceArray`*.

The face detection results include the face region position (*`faceX`*, *`faceY`*), size (*`faceW`*, *`faceH`*), and orientation (*`faceRoll`*, *`facePitch`*, *`faceYaw`*). The position and size are represented in a coordinate system in which the input image width and height are each 1.0, respectively.

Figure 3 Face Detection Results



The following code can be used to convert the face regions in the face detection results to coordinates in 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);
```

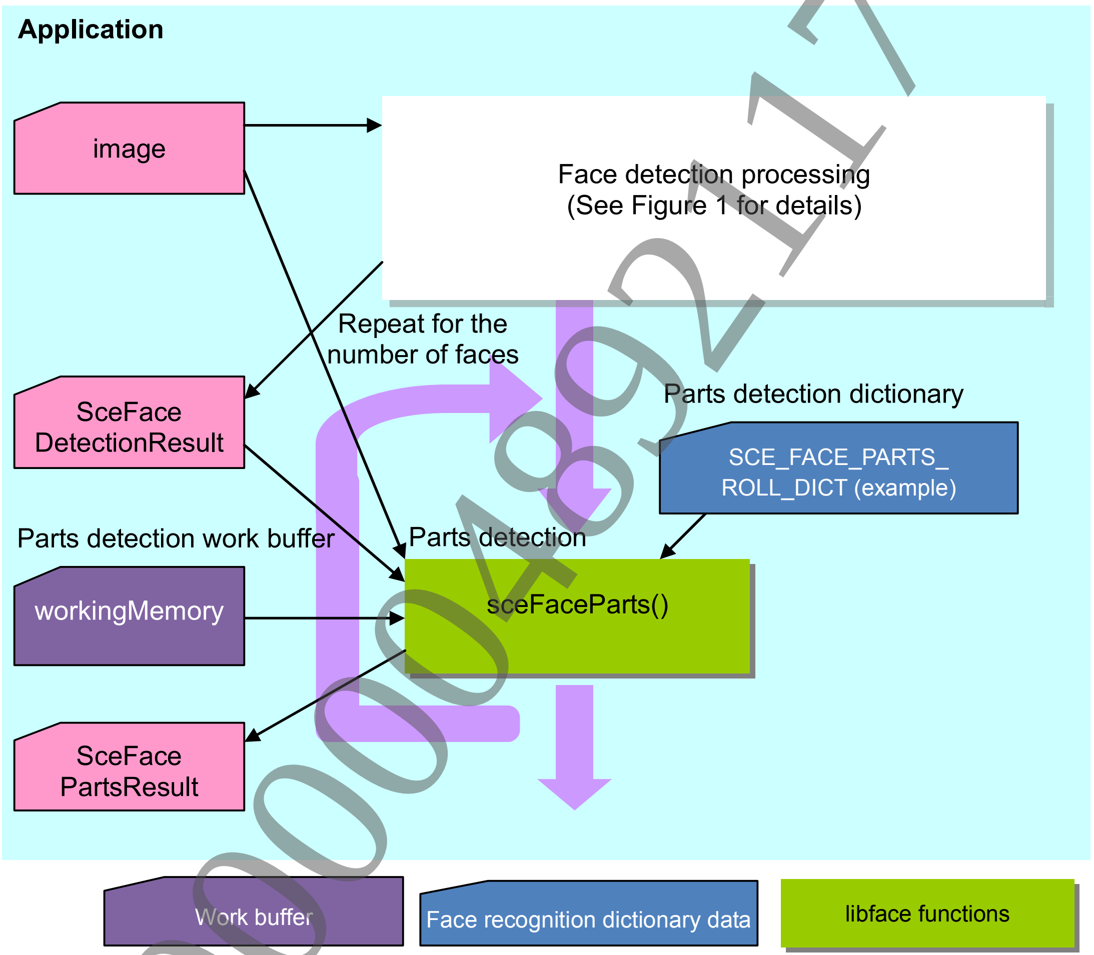
Parts Detection and Detailed Parts Detection Procedure

To perform parts detection, the face detection results and the input image used for face detection are required as input data.

The processing flow is as follows. First, perform face detection. Then perform parts detection processing for each individual face in the face detection results. Detailed parts detection can be performed the same way as parts detection.

To execute parts detection and detailed parts detection processing, select the parts detection dictionary and detailed parts detection dictionary suitable for the application and pass a pointer to it as an argument. The parts detection and the detailed parts detection dictionary must be read onto memory in advance from the parts detection and detailed parts detection dictionary data file.

Figure 4 Parts Detection Processing Procedure


Application
image
Face detection processing
(See Figure 1 for details)
Repeat for the number of faces
SceFace DetectionResult
Parts detection dictionary
SCE_FACE_PARTS_ROLL_DICT (example)
Parts detection work buffer
Parts detection
workingMemory
sceFaceParts()
SceFace PartsResult
Work buffer
Face recognition dictionary data
libface functions

(1) Execute face detection

Prepare the input image and face detection result output area and execute face detection with the face detection parameters set as arguments. For details about the procedure, refer to the "Face Detection Procedure" section.

(2) Prepare the parts detection and the detailed parts detection dictionary data, and working memory area

In case of performing parts detection, read the appropriate parts detection dictionary data file into the memory.

Similarly, in case of performing detailed parts detection, read the detailed parts detection dictionary data file into the memory.

The size of working memory area for parts detection can be obtained by calling `sceFacePartsGetWorkingMemorySize()` with the size of input image and the parts detection dictionary data as arguments.

Similarly, the size of working memory area for detailed parts detection can be obtained by calling `sceFaceAllPartsGetWorkingMemorySize()` with the size of input image and the detailed parts detection dictionary data as arguments.

Allocate the memory area according to the size obtained.

(3) Prepare the parts detection and the detailed parts detection result output area

Prepare an array of the `SceFacePartsResult` structure as the area for saving the output results from parts detection. This array must have `SCE_FACE_PARTS_NUM_MAX(4)` elements.

Similarly, prepare an array of the `SceFacePartsResult` structure as the area for saving the output results from detailed parts detection. This array must have `SCE_FACE_ALLPARTS_NUM_MAX(55)` elements.

(4) Execute parts detection and detailed parts detection

For parts detection, execute `sceFaceParts()` and pass it information about the input image, the parts detection dictionary data address, face detection result obtained in (1), address of the parts detection result output area, number of elements in the parts detection result array, other parts detection parameters, and the size and address of working memory area as arguments.

Similarly, for detailed parts detection, execute `sceFaceAllParts()` and pass it information about the input image, the detailed parts detection dictionary data address, face detection result obtained in (1), address of the detailed parts detection result output area, number of elements in the detailed parts detection result array, other detailed parts detection parameters, and the size and address of working memory area as arguments.

For the input image, use the same image that was used for face detection. The image area must not be cleared until all parts detection and detailed parts detection processing has finished.

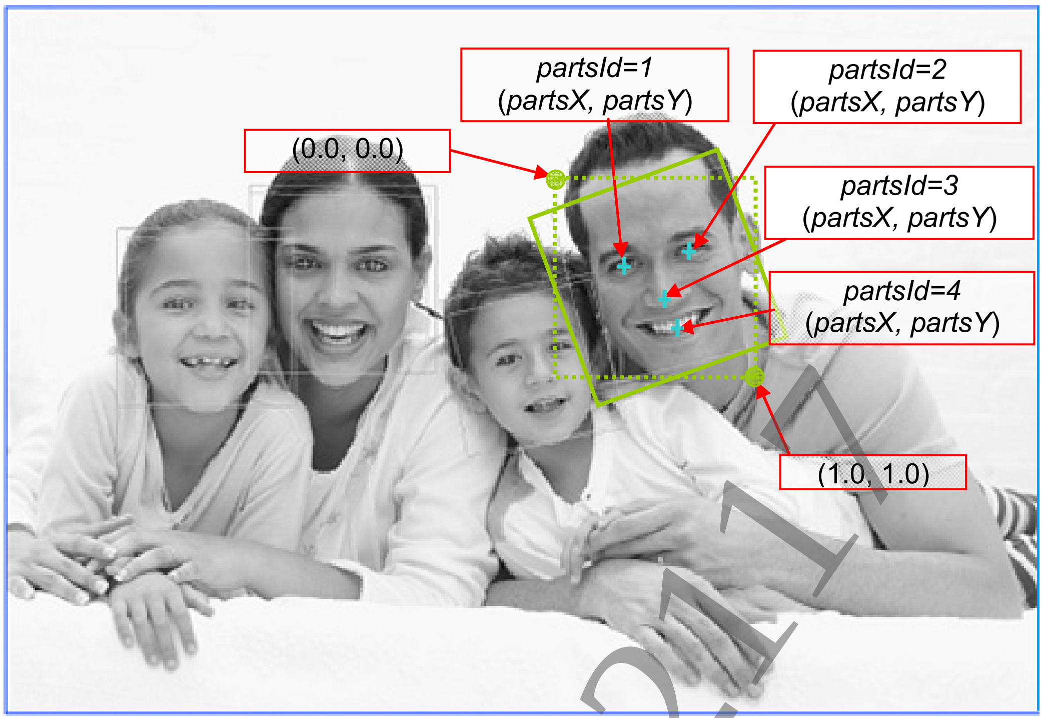
(5) Receive the parts detection and the detailed parts detection results

When `sceFaceParts()` completes successfully, the number of parts that were detected is returned in *`resultPartsNum`*. The detected positions of the right eye, left eye, nose and mouth are stored in the various elements of the `SceFacePartsResult` structure array prepared in (3) and whose address was set in the *`resultPartsArray`* argument.

Similarly, when `sceFaceAllParts()` completes successfully, the number of parts that were detected is returned in *`resultPartsNum`*. The detected positions of the detailed parts are stored in the various elements of the `SceFacePartsResult` structure array prepared in (3) and whose address was set in the *`resultPartsArray`* argument.

The type of part that was detected is stored in the *`partsId`* member of the `SceFacePartsResult` structure and the position of the part is stored in *`partsX`* and *`partsY`*. *`partsX`* and *`partsY`* are represented in a coordinate system in which the upper left corner of the face region within the face detection results is at (0.0, 0.0) and the lower right corner is at (1.0, 1.0).

Figure 5 Parts Detection Results



The following code can be used to convert the parts positions of the parts detection results 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;
```

Note
When the value stored in the *partsId* member of the `SceFacePartsResult` structure is `SCE_FACE_PARTS_ID_UNDEF`, it means that parts detection or detailed parts detection processing failed.

When the right eye, left eye, nose and mouth are all detected by `sceFaceParts()`, the face pose (roll, pitch, yaw) and the face region can be estimated from the parts positions by executing `sceFaceEstimatePoseRegion()` with those parts detection results set as arguments.

(6) Repeat the process for all faces

Repeat the processing of (4) and (5) for all faces that were detected in (1).

Attribute Classification Procedure

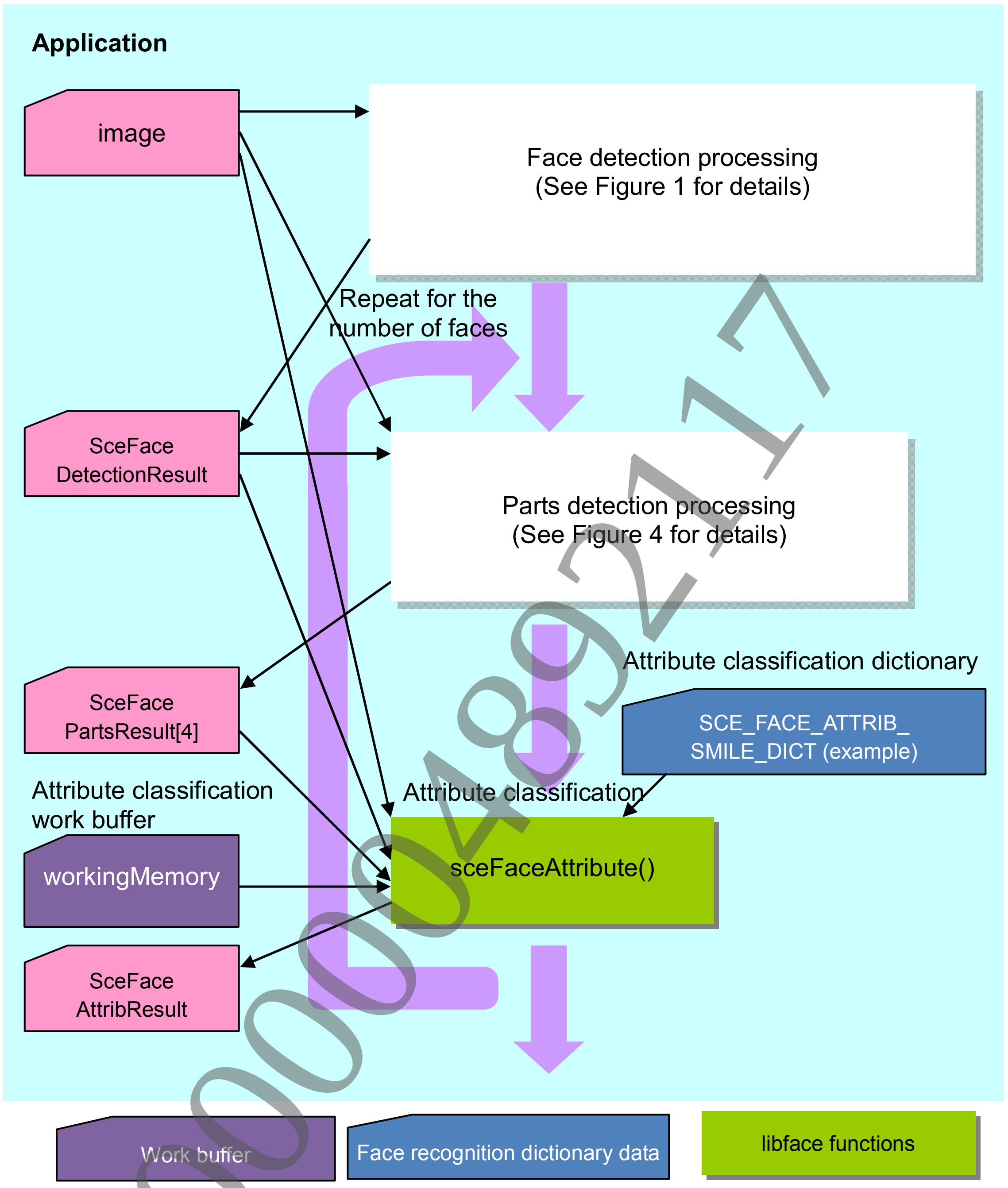
To perform attribute classification, the face detection results, parts detection results and the input image used for face and parts detection are required as input data.

The processing flow is as follows. First, perform face detection. Then for each individual face in the face detection results, perform parts detection processing and attribute classification.

When executing the attribute classification function, pass a pointer to the attribute classification dictionary as an argument. The attribute classification dictionary must be read onto memory in advance from the attribute classification dictionary data file.

Note that currently there are two types of attribute classification dictionaries: smile only, and smile/eye openness/gender/adult or child/baby/elder/glasses (all in one).

Figure 6 Attribute Classification Procedure



(1) Execute face detection

Prepare the input image and face detection result output area and execute face detection with the face detection parameters set as arguments. For details about the procedure, refer to the "Face Detection Procedure" section.

(2) Execute parts detection

Prepare the parts detection result output area and execute parts detection with the parts detection parameters set as arguments. For details about the procedure, refer to the "Parts Detection and Detailed Parts Detection Procedure" section.

(3) Prepare the attribute classification dictionary data, and working memory area

Read the appropriate attribute classification dictionary data file onto memory.

The size of the working memory area for attribute classification can be obtained by calling `sceFaceAttributeGetWorkingMemorySize()` with the size of input image and the attribute classification dictionary data as arguments. Allocate the memory area according to the size obtained.

(4) Prepare the attribute classification result output area

Prepare an array of the `SceFaceAttribResult` structure as the area for saving the output results from attribute classification. This array must have `SCE_FACE_ATTRIB_NUM_MAX(8)` elements. Just one element is sufficient, however, if only the smile attribute score will be obtained with `SCE_FACE_ATTRIB_SMILE_DICT`.

(5) Execute attribute classification

Execute `sceFaceAttribute()` and set information about the input image, the attribute classification dictionary data address, face detection result that was obtained in (1), parts detection result that was obtained in (2), attribute classification result output area address, number of elements in the attribute classification result array, and the size and address of working memory area as arguments.

For the input image, use the same image that was used for face detection and parts detection. The image area must not be cleared until all attribute classification has finished.

Note that for attribute classification to complete, the parts detection results must have detected all four parts (right eye, left eye, nose and mouth).

(6) Receive attribute classification results

When `sceFaceAttribute()` completes successfully, the number of attributes that were classified is returned in *`resultAttribNum`*. The attribute classification results are stored in the various elements of the `SceFaceAttribResult` structure prepared in (3) and whose address was set in the *`resultAttribArray`* argument. The type of attribute is stored in the *`attribId`* member of the `SceFaceAttribResult` structure and the classification result score is stored in *`score`*.

The smile attribute score, which reflects the degree of a smile, is output as a normalized value ranging from 0.0 to 100.0. A score of 20.0 represents an expressionless face. The closer the score is to 100.0, the bigger the smile, whereas, the closer the score is to 0.0, the more sorrowful or anguished the face is determined to be. The score of how wide the eyes are open will be each output as a normalized value between 0.0 and 100.0 representing the degree to which each eye is opened/closed. A value closer to 100.0 represents a widely-open eye, and a value closer to 0.0 represents a shut eye. Other attribute scores are output as normalized values between 0.0 and 100.0, but with 50.0 as the point of reference. For example, a gender classification score of less than 50.0 indicates a female, and a score of over 50.0 indicates a male. The classification scores for "baby", "adult/child", and "elderly" are similar; scores less than 50.0 indicate "not a baby", "not an adult" (is a child), and "not an elderly person", respectively. Scores over 50.0 indicate that the person is a baby, child, adult, or an elderly person, respectively. The same scoring system applies when classifying people according to whether they wear glasses.

(7) Repeat the process for all faces

Repeat the processing of (5) and (6) for all faces detected in (1).

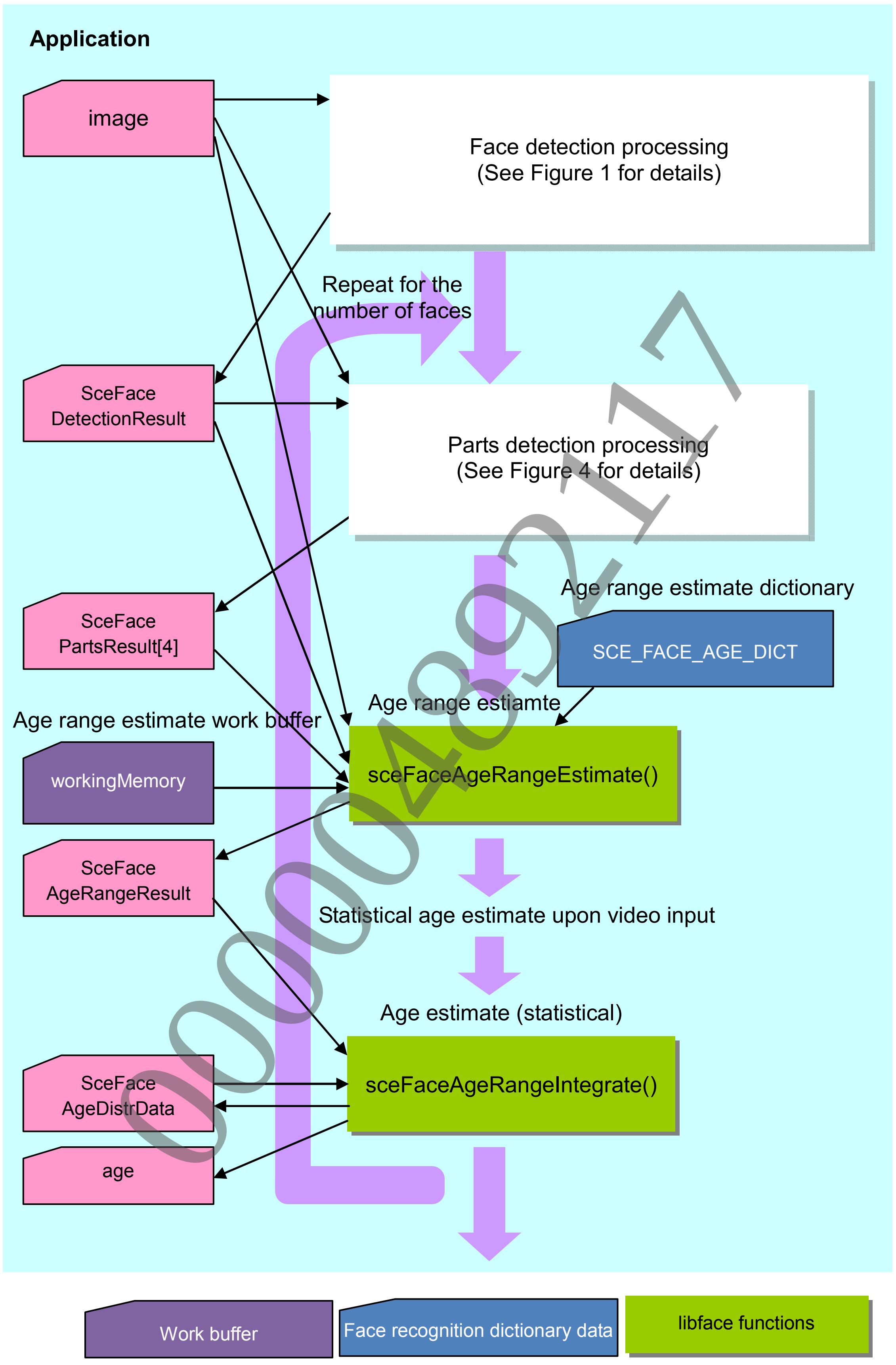
Age Estimate Procedure

To perform an age estimate, the face detection result, parts detection result, and the input image used for face and parts detection are required as input data.

The processing flow is as follows. First, perform face detection. Then for each individual face in the face detection result, perform parts detection processing and age range estimate. Moreover, for movie inputs, tracking the target face and executing age range estimate per frame will enable you to get a more precise, statistical age estimate.

When executing the age range estimate function, pass a pointer to the age range estimate dictionary as an argument. The age range estimate dictionary must be read onto memory in advance from the age range estimate dictionary data file.

Figure 7 Age Estimate Procedure


Application
image
Face detection processing
(See Figure 1 for details)
Repeat for the number of faces
SceFace DetectionResult
Parts detection processing
(See Figure 4 for details)
SceFace PartsResult[4]
Age range estimate dictionary
SCE_FACE_AGE_DICT
Age range estimate work buffer
Age range estiamte
workingMemory
sceFaceAgeRangeEstimate()
SceFace AgeRangeResult
Statistical age estimate upon video input
Age estimate (statistical)
sceFaceAgeRangeIntegrate()
SceFace AgeDistrData
age
Work buffer
Face recognition dictionary data
libface functions

(1) Execute face detection

Prepare the input image and face detection result output area, and execute face detection with the face detection parameters set as arguments. For details about the procedure, refer to the "Face Detection Procedure" section.

(2) Execute Parts Detection

Prepare the parts detection result output area, and execute parts detection with the parts detection parameters set as arguments. For details about the procedure, refer to the "Parts Detection and Detailed Parts Detection Procedure" section.

(3) Prepare the age range estimate dictionary, and working memory area

Read the age range estimate dictionary data file onto memory.

The size of the working memory area for age range estimate can be obtained by calling `sceFaceAgeGetWorkingMemorySize()` with the size of input image and the age range estimate dictionary as arguments. Allocate the memory area according to the size obtained.

(4) Prepare the age estimate result output area

Prepare an output area of the `SceFaceAgeRangeResult` structure as the area for saving the output results from age range estimate. For making statistical age estimate from movie inputs, also prepare an output area of the `SceFaceAgeDistrData` structure as the area for saving accumulated user age probability distribution data (make sure to zero-clear this area as its initialization value).

(5) Execute age range estimate

Set information about the input image, the age range estimate dictionary data address, face detection result that was obtained in (1), parts detection result that was obtained in (2), age range estimate result output area address, and the size and address of working memory area as arguments and execute `sceFaceAgeRangeEstimate()`.

For the input image, use the same image that was used for face detection and parts detection. The image area must not be cleared until age range estimate processing completes.

Note that for age range estimate to complete, the parts detection results must have detected all four parts (right eye, left eye, nose, and mouth).

(6) Receive age range estimate results

When `sceFaceAgeRangeEstimate()` completes successfully, the minimum and maximum age range will be stored in *`minAge`* and *`maxAge`*, respectively, of the `SceFaceAgeRangeResult` structure - for which its address was set to the *`resultAge`* argument in step (4).

The estimate range is normally 10 years. This means *`minAge`* and *`maxAge`* should have a difference of 10; however, this range may exceed 20 in cases when it is difficult to make an age estimate - for example, when the target face is under poor lighting or when a face is heavily made up with face paint.

(7) Integrate estimated age range results

To obtain a statistical age estimate from movie inputs, execute `sceFaceAgeRangeIntegrate()` and integrate the results of the age range estimate obtained per frame to the estimate age probability distribution data (`SceFaceAgeDistrData` structure prepared in step (4)). Repeat integration to obtain a precise age estimate; track the target face and integrate the result of the age range estimate with the same face's estimate age probability distribution data.

(8) Repeat the process for all faces

Repeat the processing from step (2) onwards for all faces detected in (1).

Face Identification Procedure

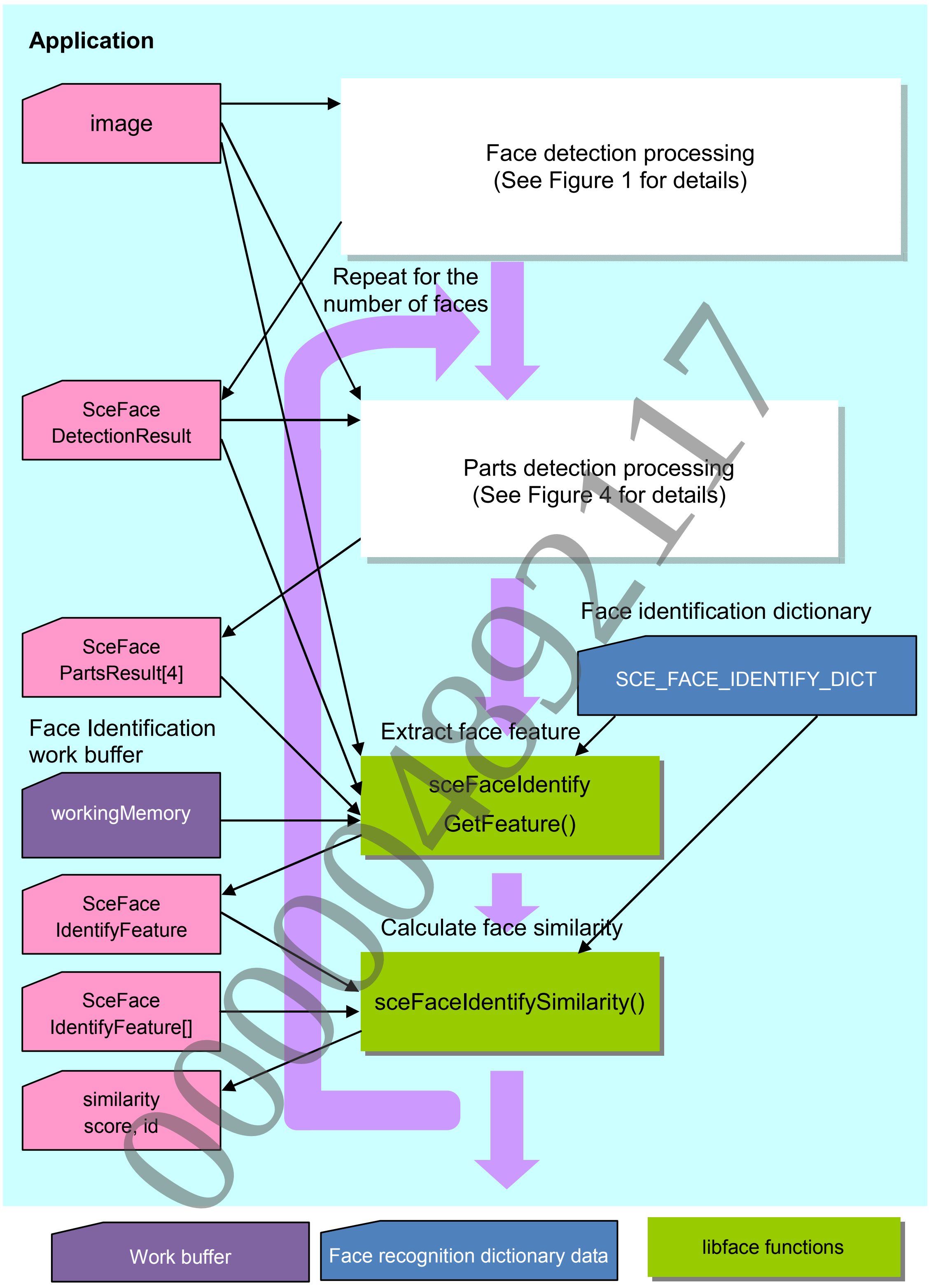
Face identification procedure consists of two phases: face feature calculation and face feature similarity calculation.

To perform face identification, the face detection results, parts detection results and the input image used for face and parts detection are required as input data.

The processing flow is as follows. First, perform face detection, and for each individual face in the face detection results, perform parts detection. Then, extract the face feature of the each individual face in the parts detection results and perform similarity calculation.

When executing the face identification function, pass a pointer to the face identification dictionary as an argument. The face identification dictionary must be read onto memory in advance from the face identification dictionary data file.

Figure 8 Face Identification Procedure


Application
image
Face detection processing
(See Figure 1 for details)
Repeat for the number of faces
SceFace DetectionResult
Parts detection processing
(See Figure 4 for details)
Face identification dictionary
SCE_FACE_IDENTIFY_DICT
SceFace PartsResult[4]
Face Identification work buffer
workingMemory
Extract face feature
sceFaceIdentify GetFeature()
SceFace IdentifyFeature
Calculate face similarity
sceFaceIdentifySimilarity()
SceFace IdentifyFeature[]
similarity score, id
Work buffer
Face recognition dictionary data
libface functions

(1) Execute face detection

Prepare the input image and face detection result output area and execute face detection with the face detection parameters set as arguments. For details about the procedure, refer to the "Face Detection Procedure" section.

(2) Execute parts detection

Prepare the parts detection result output area and execute parts detection with the parts detection parameters set as arguments. For details about the procedure, refer to the "Parts Detection and Detailed Parts Detection Procedure" section.

(3) Prepare the face identification dictionary data, and working memory area

Read the face identification dictionary data file into the memory.

The size of working memory area for face identification can be obtained by calling `sceFaceIdentifyGetWorkingMemorySize()` with the size of input image and the face identification dictionary data as arguments. Allocate the memory area according to the size obtained.

(4) Prepare the face feature and similarity score output area

Prepare a `SceFaceIdentifyFeature` structure as the area for face feature. To calculate the similarity between the target face and the registered faces, prepare an array of `float` as the output result area for similarity score. This array must have the same number of elements with the number of array elements in registered face feature. This output area for similarity scores is not necessary when obtaining only the most similar of the registered faces.

(5) Prepare registered face feature (for comparison)

Prepare registered face feature data to compare the similarity as an array of `SceFaceIdentifyFeature` in memory.

(6) Execute face feature calculation

Execute `sceFaceIdentifyGetFeature()` and set information about the input image, the face identification dictionary data address, face detection result that was obtained in (1), parts detection result that was obtained in (2), face feature data output area address, and the size and address of the working memory area as arguments.

For the input image, use the same image that was used for face detection and parts detection. The image area must not be cleared until all face identification has finished.

Note that the parts detection results must have detected all four parts (right eye, left eye, nose and mouth).

(7) Receive face feature data

When `sceFaceIdentifyGetFeature()` completes successfully, the face feature data for the target face is returned in the `SceFaceIdentifyFeature` structure prepared in (4) and whose address was set in the *`resultFeature`* argument.

(8) Execute face feature similarity calculation

Execute `sceFaceIdentifySimilarity()` and set the face feature data for the target face that was obtained in (7), the address and the number of elements of registered face feature data array, the face identification dictionary data address, the address of array index and score to determine the most similar face feature in registered face feature data, and the address to the output result area of all similarity score of each face feature data.

To only obtain the most similar of the registered faces, the address to the output result area of all similarity scores can be set to NULL.

(9) Receive similarity score

When `sceFaceIdentifySimilarity()` completes successfully, the array index and score to the most similarity face feature data are returned in the area whose address was set as arguments *`maxScoreId`* and *`maxScore`*. If the address to the output result area of all similarity scores is set, the similarity score between the target face feature data and each registered face feature data is output.

(10) Repeat the process for all faces

Repeat the processing of (6), (7), (8) and (9) for all faces detected in (1).

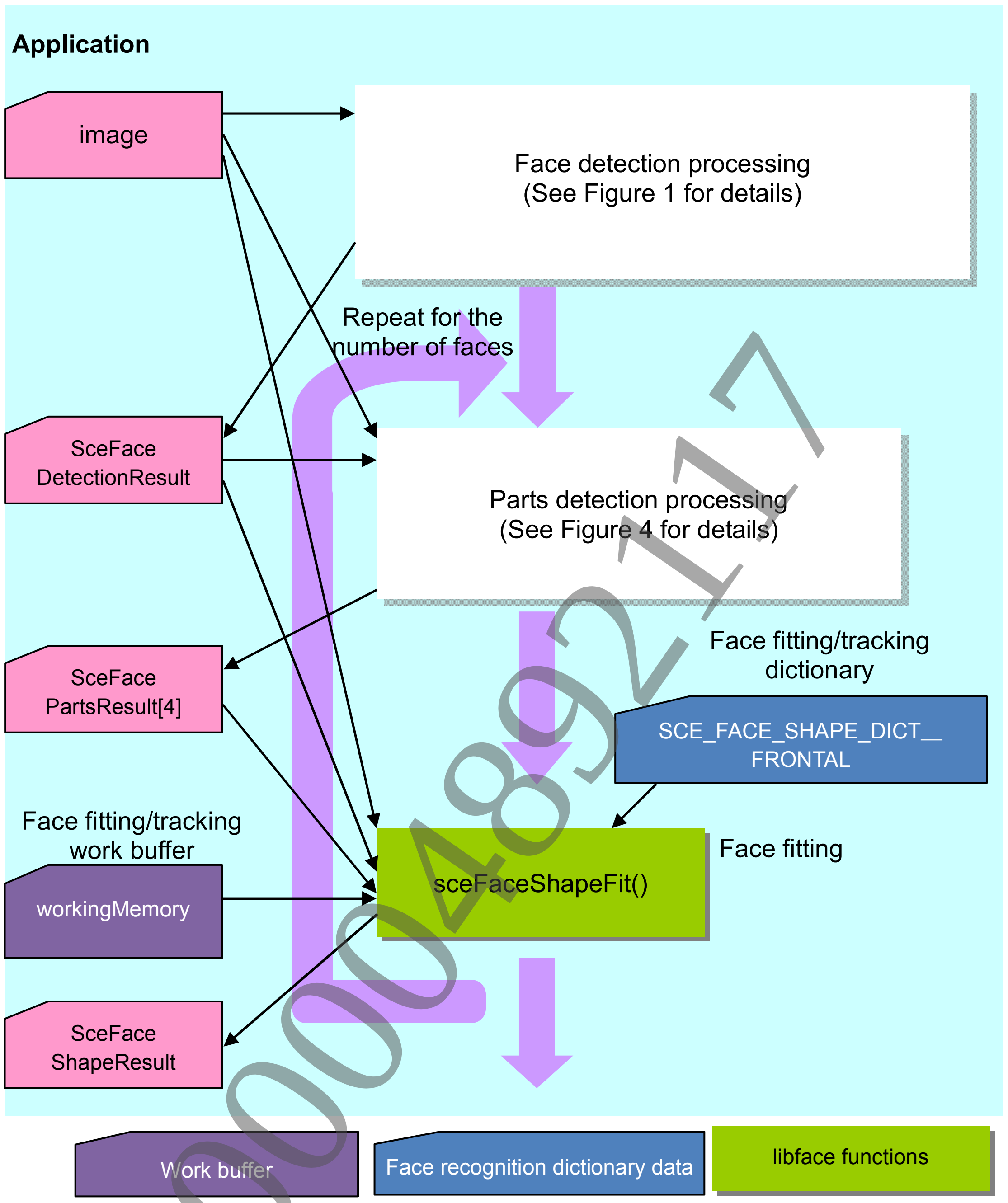
Face Fitting Procedure

To perform face fitting, the face detection result, parts detection result, and the input image used for face and parts detection are required as input data.

The processing flow is as follows. First, perform face detection, and for each individual face in the face detection results, perform parts detection. Then, perform face fitting for each individual face.

When executing the face fitting function, pass a pointer to the face fitting/tracking dictionary as an argument. The face fitting/tracking dictionary must be read onto memory in advance from the face fitting/tracking dictionary data file.

Figure 9 Face Fitting Procedure


Application
image
Face detection processing
(See Figure 1 for details)
Repeat for the number of faces
SceFace DetectionResult
Parts detection processing
(See Figure 4 for details)
SceFace PartsResult[4]
Face fitting/tracking dictionary
SCE_FACE_SHAPE_DICT__FRONTAL
Face fitting/tracking work buffer
workingMemory
sceFaceShapeFit()
Face fitting
SceFace ShapeResult
Work buffer
Face recognition dictionary data
libface functions

(1) Execute face detection

Prepare the input image and face detection result output area and execute face detection with the face detection parameters set as arguments. For details about the procedure, refer to the "Face Detection Procedure" section.

(2) Execute parts detection

Prepare the parts detection result output area and execute parts detection with the parts detection parameters set as arguments. For details about the procedure, refer to the "Parts Detection and Detailed Parts Detection Procedure" section.

(3) Prepare the face fitting/tracking dictionary, and working memory

Read the appropriate face fitting/tracking dictionary data file onto the memory.

The size of working memory area for face fitting/tracking can be obtained by calling `sceFaceShapeGetWorkingMemorySize()` with the size of input image and the face fitting/tracking dictionary as arguments. Allocate the memory area according to the size obtained.

(4) Prepare the face fitting result output area

Prepare an array of the `SceFaceShapeResult` structure as the area for saving the output results of face fitting. The number of elements of this array can be set according to the number of faces you want to detect. For example, if it is sufficient to detect 10 faces maximum from one image, prepare an array of the `SceFaceShapeResult` structure with 10 elements.

(5) Execute face fitting

Set information about the input image, the face fitting/tracking dictionary data address, address of the output area for face fitting results, score threshold values for evaluating face fitting losses, face detection result that was obtained in (1), parts detection result that was obtained in (2), and the size and address of the working memory area as arguments and execute `sceFaceShapeFit()`.

For the input image, use the same image that was used for face detection and parts detection. The image area must not be cleared until face fitting processing completes.

Note that for face fitting to complete, the parts detection results must have detected all four parts (right eye, left eye, nose, and mouth).

(6) Receive face fitting results

When `sceFaceShapeFit()` completes successfully, the results of face fitting will be stored in each element of the `SceFaceShapeResult` structure array - for which its address was specified to the *`shape`* argument in step (4). The following is an explanation of each element.

`pointX` and *`pointY`* store coordinates of face shape data, and *`score`* stores the score of the face fitting result.

The score for face fitting is output as a normalized value from 0.0 to 100.0 corresponding to the validity of the face shape result. A value closer to 100.0 indicates higher accuracy. A value around 60 is usually output; a value of 50 or lower indicates the process did not yield usable results. This benchmark evaluation can be controlled using the *`lostThreshold`* argument of `sceFaceShapeFit()`. `sceFaceShapeFit()` will return `SCE_FACE_ERROR_IMPERF_SHAPE` when *`score`* is lower than *`lostThreshold`*, and when most of the target face is concealed or out of the screen frame. For points evaluated as concealed or out of the screen frame, 1 is stored to *`isLost`*. In all other cases, 0 will be output.

As other parameters that are output as results: *`rectCenterX`* and *`rectCenterY`* indicate the position of the face, *`rectWidth`* and *`rectHeight`* indicate the size of the face, and *`faceRoll`*, *`facePitch`*, and *`faceYaw`* indicate the orientation of the face.

(7) Repeat the process for all faces

Repeat the process from step (5) and onwards for all faces detected in step (1).

Face Tracking Procedure

For face tracking, the current frame, the result of face fitting for the immediately-preceding frame and the input image used for it, are required as input data.

The processing flow is as follows. First, perform face detection, and for each individual face in the face detection results, perform parts detection. Then, perform face fitting for each individual face. Execute tracking on the succeeding frame based on the result of face fitting of the preceding frame.

When executing the face fitting/tracking function, pass a pointer to the face fitting/tracking dictionary as an argument. The face fitting/tracking dictionary must be read onto memory in advance from the face fitting/tracking dictionary data file.

For face tracking, the recommended input image size is QQVGA (160 x 120).

(1) Execute face detection

Prepare the input image and face detection result output area and execute face detection with the face detection parameters set as arguments. For details about the procedure, refer to the "Face Detection Procedure" section.

(2) Execute parts detection

Prepare the parts detection result output area and execute parts detection with the parts detection parameters set as arguments. For details about the procedure, refer to the "Parts Detection and Detailed Parts Detection Procedure" section.

(3) Prepare the face fitting/tracking dictionary, and working memory

Depending on the purpose, read the appropriate face fitting/tracking dictionary data file onto memory.

The size of working memory area for face fitting/tracking can be obtained by calling `sceFaceShapeGetWorkingMemorySize()` with the size of input image and the face fitting/tracking dictionary as arguments. Allocate the memory area according to the size obtained.

(4) Prepare the face fitting result output area

Prepare an array of the `SceFaceShapeResult` structure as the area for saving the output results of face fitting. The number of elements of this array can be set according to the number of faces you want to detect. For example, if it is sufficient to detect 10 faces maximum from one image, prepare an array of the `SceFaceShapeResult` structure with 10 elements.

(5) Execute face fitting

Set information about the input image, the face fitting/tracking dictionary data address, address of the output area for face fitting results, score threshold values for evaluating face fitting losses, face detection result that was obtained in (1), parts detection result that was obtained in (2), and the size and address of the working memory area as arguments and execute `sceFaceShapeFit()`.

For the input image, use the same image that was used for face detection and parts detection. The image area must not be cleared until face fitting processing completes.

Note that for face fitting to complete, the parts detection results must have detected all four parts (right eye, left eye, nose, and mouth).

(6) Receive face fitting results

When `sceFaceShapeFit()` completes successfully, the results of face fitting will be stored in each element of the `SceFaceShapeResult` structure array - for which its address was specified to the *`shape`* argument in step (4). The following is an explanation of each element.

`pointX` and *`pointY`* store coordinates of face shape data, and *`score`* stores the score of the face fitting result.

The score for face fitting is output as a normalized value from 0.0 to 100.0 corresponding to the validity of the face shape result. A value closer to 100.0 indicates higher accuracy. A value around 60 is usually output; a value of 50 or lower indicates the process did not yield usable results. This benchmark evaluation can be controlled using the *`lostThreshold`* argument of `sceFaceShapeFit()`. `sceFaceShapeFit()` will return `SCE_FACE_ERROR_IMPERF_SHAPE` when *`score`* is lower than *`lostThreshold`*, and when most of the target face is concealed or out of the screen frame. For points evaluated as concealed or out of the screen frame, 1 is stored to *`isLost`*. In all other cases, 0 will be output.

As other parameters that are output as results: *`rectCenterX`* and *`rectCenterY`* indicate the position of the face, *`rectWidth`* and *`rectHeight`* indicate the size of the face, and *`faceRoll`*, *`facePitch`*, and *`faceYaw`* indicate the orientation of the face.

(7) Execute face tracking

When the above steps complete successfully, tracking will be possible on the succeeding input image based on the immediately-preceding frame image and the obtained face shape data result.

Set information about the current input image and the image immediately-preceding it, address of the face fitting/tracking dictionary data, address of the data area storing the result of face fitting for the immediately-preceding input image, score threshold values for evaluating face fitting losses, and the size and address of the working memory area as arguments and execute `sceFaceShapeTrack()`.

Do not clear image area of the current input image and the image immediately-preceding it until all processing for face fitting completes.

When execution completes successfully, the data of the face fitting result for the immediately-preceding frame will be updated to the face shape result of the current input image.

(8) Check face tracking results and evaluate whether tracking can be continued

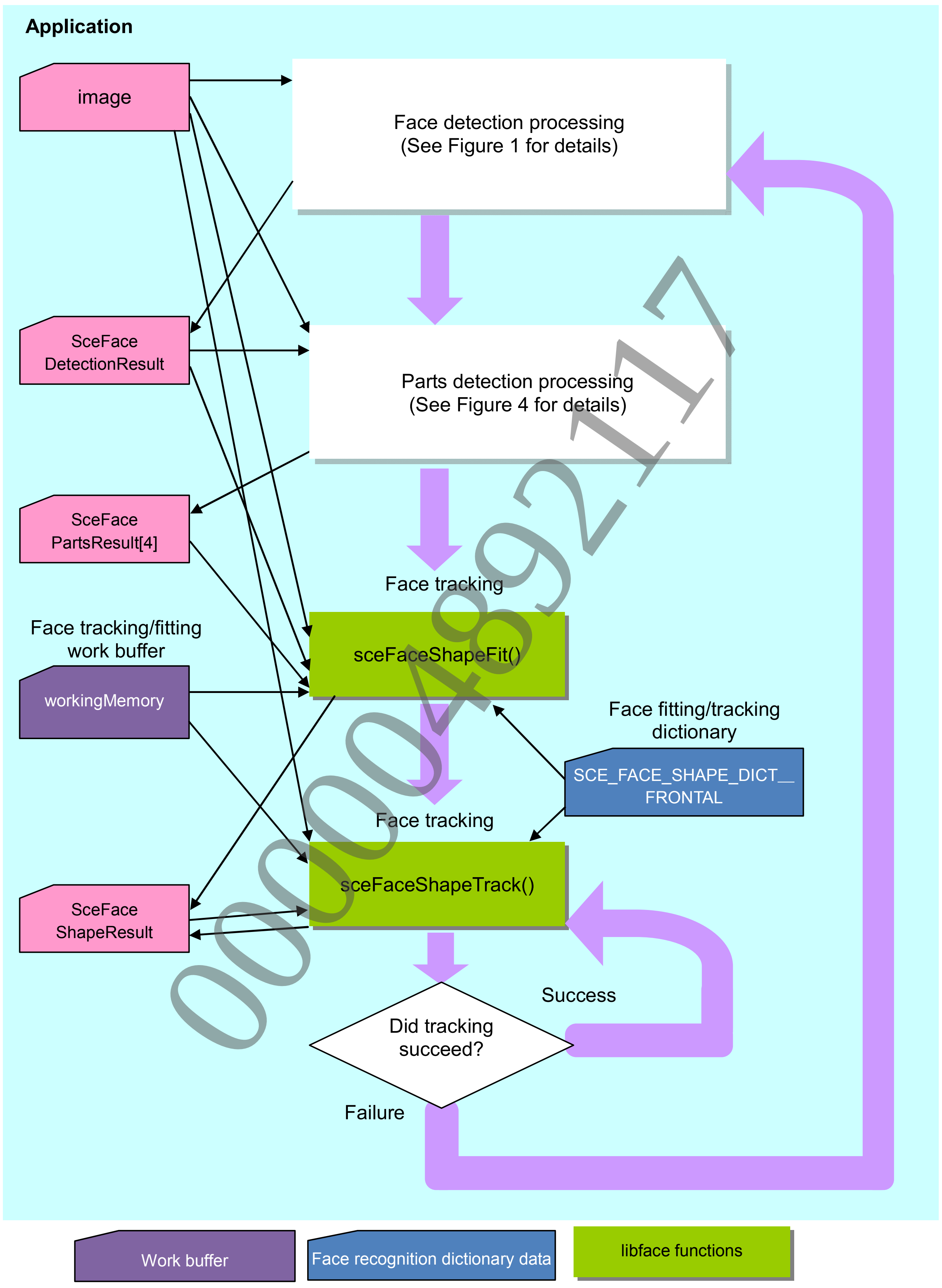
When `sceFaceShapeTrack()` completes successfully, this indicates face tracking succeeded, and face shape data will be output in the same manner described in step (5). Replace the processed frame as the immediately-preceding frame, set the next input frame as the current frame and execute step (6) to execute face tracking on the next frame. As long as face tracking continues to succeed, face tracking can be continued by repeating (6) and (7) on the subsequently input frames.

When `sceFaceShapeTrack()` returns the `SCE_FACE_ERROR_IMPERF_SHAPE` error, this indicates face tracking failed. Processing from face detection (step (1)) must be carried out for the next frame.

(9) Repeat for all faces

Repeat the process from step (2) and onwards for all faces detected in step (1).

Figure 10 Face Tracking Procedure



List of Functions

libface functions can be divided into five types: face detection, parts detection, attribute classification/age estimate, face identification, and face fitting/tracking.

Functions Related to Face Detection

API	Description
`sceFaceDetection()`	Executes face detection
`sceFaceDetectionEx()`	Executes face detection (extended version)
`sceFaceDetectionLocal()`	Executes fast face detection using a local search
`sceFaceDetectionGetDefaultParam()`	Obtains the face detection (extended version) default parameters
`sceFaceDetectionGetWorkingMemorySize()`	Obtains the size of the face detection working memory

Functions Related to Parts Detection

API	Description
`sceFaceParts()`	Executes parts detection
`sceFaceAllParts()`	Executes detailed parts detection
`sceFaceEstimatePoseRegion()`	Estimates face pose and face region according to detected parts positions
`sceFacePartsResultCheck()`	Checks validity of the parts detection result
`sceFacePartsGetWorkingMemorySize()`	Obtains the size of the parts detection working memory
`sceFaceAllPartsGetWorkingMemorySize()`	Obtains the size of the detailed parts detection working memory

Functions Related to Attribute Classification/Age Estimate

API	Description
`sceFaceAttribute()`	Executes attribute classification
`sceFaceAgeRangeEstimate()`	Executes age range estimate
`sceFaceAgeRangeIntegrate()`	Integrates age range estimate results (statistical age estimate)
`sceFaceAttributeGetWorkingMemorySize()`	Obtains the size of the attribute classification working memory
`sceFaceAgeGetWorkingMemorySize()`	Obtains the size of the age range estimate working memory

Functions Related to Face Identification

API	Description
`sceFaceIdentifyGetFeature()`	Executes face feature calculation
`sceFaceIdentifySimilarity()`	Executes face feature similarity calculation
`sceFaceIdentifyGetWorkingMemorySize()`	Obtains the size of the face identification working memory

Functions Related to Face Fitting/Tracking

API	Description
`sceFaceShapeFit()`	Executes face fitting
`sceFaceShapeTrack()`	Executes face tracking
`sceFaceShapeGetWorkingMemorySize()`	Obtains the size of the face fitting/tracking working memory

List of Face Recognition Dictionaries

To reduce memory consumption, libface provides face detection dictionary data in multiple files so that an application needs only to read and use the dictionaries that it requires.

More than one recognition processing dictionary is provided for face detection and parts detection, and the main difference between these dictionaries is in the supported range of face angles. Two different types of attribute classification dictionaries are provided. Note that as the supported face angle gets wider, memory consumption and processing time also tend to increase, so you should choose the proper dictionary according to the required specifications of your application.

Face Detection Dictionaries

Macro / Filename	Description
`SCE_FACE_DETECT_FRONTAL_DICT` `"face_detect_frontal.fdt"`	Face detection dictionary data file (for frontal face)
`SCE_FACE_DETECT_ROLL_DICT` `"face_detect_roll.fdt"`	Face detection dictionary data file (roll support)
`SCE_FACE_DETECT_YAW_DICT` `"face_detect_yaw.fdt"`	Face detection dictionary data file (yaw support)
`SCE_FACE_DETECT_PITCH_DICT` `"face_detect_pitch.fdt"`	Face detection dictionary data file (pitch support)
`SCE_FACE_DETECT_ROLL_YAW_DICT` `"face_detect_roll_yaw.fdt"`	Face detection dictionary data file (roll and yaw support)
`SCE_FACE_DETECT_ROLL_YAW_PITCH_DICT` `"face_detect_roll_yaw_pitch.fdt"`	Face detection dictionary data file (roll, yaw, and pitch support)

Parts Detection and Detailed Parts Detection Dictionaries

Macro / Filename	Description
`SCE_FACE_PARTS_ROLL_DICT` `"face_parts_roll.pdt"`	Parts detection dictionary data file (frontal face with roll support)
`SCE_FACE_PARTS_ROLL_YAW_DICT` `"face_parts_roll_yaw.pdt"`	Parts detection dictionary data file (roll and yaw support)
`SCE_FACE_PARTS_CHECK_DICT` `"face_parts_check.cdt"`	Parts result validity check dictionary data file (roll, yaw, and pitch support)
`SCE_FACE_ALLPARTS_DICT` `"face_allparts.pdt"`	Detailed parts detection dictionary data file (roll support)
`SCE_FACE_ALLPARTS_SHAPE_DICT` `"face_allparts_shape.pdt"`	Shape correction dictionary data file (roll support)

Attribute Classification Dictionaries/Age Range Estimate Dictionaries

Macro / Filename	Description
`SCE_FACE_ATTRIB_SMILE_DICT` `"face_attrib_smile.adt"`	Smile attribute classification dictionary data file
`SCE_FACE_ATTRIB_DICT` `"face_attrib.adt"`	All Attribute classification dictionary data file
`SCE_FACE_AGE_DICT` `"face_age.adt"`	Age range estimate dictionary data file

Face Identification Dictionaries

Macro / Filename	Description
`SCE_FACE_IDENTIFY_DICT` `"face_identify.idt"`	Face identification dictionary data file

Face Fitting/Tracking Dictionaries

Macro / Filename	Description
`SCE_FACE_SHAPE_DICT_FRONTAL` `"face_shape_frontal.shp"`	Face fitting/tracking dictionary data file (frontal face)

3 Notes

Processing Time and Memory Consumption

Memory consumption can be known by calling the working memory size obtaining function, such as `sceFaceDetectionGetWorkingMemorySize()`. Inside of libface, it is designed to work only in the working memory area, so that no dynamic memory allocation is performed. Infrequently, depending on the contents of the input image, the size of the working area needed may be more than the size obtained by `sceFaceDetectionGetWorkingMemorySize()`, and face recognition functions may return `SCE_FACE_ERROR_NO_MEMORY`. This may occur if there are many faces in the input image, or if many misrecognized faces are detected in the input image. In this case, the error may be fixed by allocating a larger size of the working memory area than the size obtained by `sceFaceDetectionGetWorkingMemorySize()`.

Also, processing time may vary according to the image content even if all configuration parameters are the same. Try to avoid creating situations that are known in advance that require some fixed amount of processing time.

Misrecognition

Detection omissions or misdetection may occur during face detection or parts detection. There may be a misfit of a face shape. Also, attribute classification errors and face recognition errors (the detected face cannot be recognized as the target person's face, or may be recognized as another person's face) may occur depending on the state of the input image or photographic conditions of the face. The occurrence rate of such recognition errors could be reduced by input image adjustments or parameter adjustments as properties of image recognition. However, it is fundamentally impossible to reduce them to zero. Therefore, it is recommended that you add a method of handling errors in the application so that even if some recognition errors do occur, they will not cause problems.

Library Implementation

The current implementation of libface does not require the use of special hardware resources such as the GPU, or DMAC. Also, libface does not require the use of special software resources such as exception handlers, interrupt handlers, threads, I/O files.