libgac

v0.1.0

Generated by Doxygen 1.8.17

Chapter 1

GAC - Gaze Analysis C Library

This is a pure C library to perform basic gaze analysis.

Features:

- Sample filtering with moving average
- Sample gap fill-in through linear interpolation (lerp)
- Fixation detection with I-DT algorithm
- Saccade detection with I-VT algorithm

Quick Start

Initialise the gaze analysis handler:

```
gac_t h;
gac_init( &h, NULL );
```

To parse gaze data for fixations and saccades, for each new sample do the following:

```
gac_sample_window_update( &h, sample.origin.x, sample.origin.y, sample.oridin.z,
        sample.point.x, sample.point.y, sample.point.z, sample.timestamp );
// check for fixation
res = gac_sample_window_fixation_filter( &h, &fixation );
if( res == true )
{
    // new fixation deteced -> do something with the data
    // fixation structures must be destroyed once they are no longer needed.
    gac_fixation_destroy( &fixation );
}
// check for saccade
res = gac_sample_window_saccade_filter( &h, &saccade );
if( res == true )
{
    // new saccade deteced -> do something with the data
    // saccade structures must be destroyed once they are no longer needed.
    gac_saccade_destroy( &saccade );
}
// remove samples from the sample window which are no longer used
gac_sample_window_cleanup( h );
```

At the end, destroy the gaze analysis handler:

```
gac_destroy( &h );
```

Basic Concept

The library provides several functions to work with gaze data. The easiest approach is to use the functions `gac_sample_window_*` as these maintain their own sample window and noise and gap filters can be configured through the `gac_filter_paramter_t` structure.

Alternatively it is possible to manually maintain a sample window and work with each filter individually. This means filter structures have to be created and destroyed manually and filtering has to be applied manually to a custom sample window. Refer to the API for more information.

3d vs 2d Data

All calculations are performed on 3d data. If only 2d data is available this library cannot be used (yet). The reason for this is that with 3d data it is possible to compute an accurate dispersion and velocity threshold based on the distance of the gaze origin to the gaze point. For 2d data the dispersion and velocity threshold would need to be estimated based on the measured data which is not (yet) supported by the library.

However, it is possible to provide 2d data alongside 3d data for each data sample which will propagated to fixation and saccade result structures. To add 2d data for each sample instead of the function `gac_sample_window_update()` use `gac_sample_window_update_screen()`.

If 2d data is not available it is possible to compute it from 3d data. `gac_sample_window_update()` does this automatically if the screen location is defined. To define the screen location use the function `gac_set_screen()`.

Sample annotations

Each sample has two fields available for custom data annotation:

- `trial_id`: expects an integer number and can be used to e.g. associate a data point to a trial.
- `label`: expects a string and can be used to e.g. describe the currently displayed stimuli.

The annotations are propagated to the fixation and saccade result structures.

Further, each sample has two additional timestamp fields for onset information of the annotations:

- `trial_onset`: the amount of milliseconds since the last change in the field `trial_id`.
- `label_onset`: the amount of milliseconds since the last change in the field `label`.

Building the library on Linux (Ubuntu)

In order to build the library the following packages are required:
```
sudo apt install build-essential
sudo apt install autoconf autogen libtool
```

To build the library use the command
```
make
```

To run tests use
```
make test
```

Build and run the example with the following commands:
```
cd example
make
make run
```

Building the library on Windows

Build the library on windows with `msys2`. Once installed start `msys2.exe`.

Some dependencies need to be installed. To do this type the following commands:

```
pacman -Syyu
pacman -Sy mingw-w64-x86_64-gcc
pacman -Sy autogen autoconf automake libtool
```

Finally, to build the library type

```
make
```

Build the example with the following commands:

```
cd example
make
```

To run the example make sure that the system knows the location of `msys2.dll` (either by adding the location to the PATH or by copying the file to the example folder). Run the example by starting `example.exe`.

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 4

Data Structure Documentation

4.1 gac_filter_fixation_t Struct Reference

```
#include <gac.h>
```

Data Fields

- bool is_heap
- double normalized_dispersion_threshold
- double duration_threshold
- bool is_collecting
- gac_queue_t window
- uint32_t new_samples
- double duration
- vec2 screen_point
- vec3 point

4.1.1 Detailed Description

The fixation filter structure holding filter parameters.

gac_filter_fixation_s

4.1.2 Field Documentation

4.1.2.1 duration

```
double gac_filter_fixation_t::duration
```

The fixation duration

4.1.2.2 duration_threshold

```
double gac_filter_fixation_t::duration_threshold
```

The duration threashold

4.1.2.3 is_collecting

```
bool gac_filter_fixation_t::is_collecting
```

A flag indicating whether a fixation is ongoing.

4.1.2.4 is_heap

```
bool gac_filter_fixation_t::is_heap
```

Flag to indicate whether the struct was allocated on the heap.

4.1.2.5 new_samples

```
uint32_t gac_filter_fixation_t::new_samples
```

Counter to keep track of new items in the parent queue.

4.1.2.6 normalized_dispersion_threshold

```
double gac_filter_fixation_t::normalized_dispersion_threshold
```

The pre-computed dispersion threshold at unit distance

4.1.2.7 point

```
vec3 gac_filter_fixation_t::point
```

The fixation point

4.1.2.8 screen_point

```
vec2 gac_filter_fixation_t::screen_point
```

The fixation screen point

4.1.2.9 window

```
gac_queue_t gac_filter_fixation_t::window
```

A pointer to the sample queue

The documentation for this struct was generated from the following file:

- include/gac.h

4.2 gac_filter_gap_t Struct Reference

```
#include <gac.h>
```

Data Fields

- bool is_heap
- bool is_enabled
- double max_gap_length
- double sample_period

4.2.1 Detailed Description

The gap fill-in filter structure.

gac_filter_gap_s

4.2.2 Field Documentation

4.2.2.1 is_enabled

```
bool gac_filter_gap_t::is_enabled
```

A flag indicating whether the filter is active or not

4.2.2.2 is_heap

```
bool gac_filter_gap_t::is_heap
```

Flag to indicate whether the struct was allocated on the heap.

4.2.2.3 max_gap_length

```
double gac_filter_gap_t::max_gap_length
```

The maximal allowed gap length to be filled-in

4.2.2.4 sample_period

```
double gac_filter_gap_t::sample_period
```

The sample period to compute the number of required fill-in samples

The documentation for this struct was generated from the following file:

- include/gac.h

4.3 gac_filter_noise_t Struct Reference

```
#include <gac.h>
```

Data Fields

- bool is_heap
- bool is_enabled
- gac_queue_t window
- uint32_t mid
- gac_filter_noise_type_t type

4.3.1 Detailed Description

The noise filter parameters.

gac_filter_noise_s

4.3.2 Field Documentation

4.3.2.1 is_enabled

```
bool gac_filter_noise_t::is_enabled
```

A flag indicating whether the noise filter is active or not

4.3.2.2 is_heap

```
bool gac_filter_noise_t::is_heap
```

Flag to indicate whether the struct was allocated on the heap.

4.3.2.3 mid

```
uint32_t gac_filter_noise_t::mid
```

The mid-point counter

4.3.2.4 type

```
gac_filter_noise_type_t gac_filter_noise_t::type
```

The noise filter type

4.3.2.5 window

```
gac_queue_t gac_filter_noise_t::window
```

The noise filter window

The documentation for this struct was generated from the following file:

- include/gac.h

4.4 gac_filter_parameter_t Struct Reference

```
#include <gac.h>
```

Data Fields

- bool is_heap
- struct {
    double max_gap_length
    double sample_period
  } gap
- struct {
    gac_filter_noise_type_t type
    uint32_t mid_idx
  } noise
- struct {
    float velocity_threshold
  } saccade
- struct {
    double duration_threshold
    float dispersion_threshold
  } fixation

4.4.1 Detailed Description

The filter parameter structure to initialise the gaze analysis handeler.

gac_filter_parameter_s

4.4.2 Field Documentation

4.4.2.1 dispersion_threshold

```
float gac_filter_parameter_t::dispersion_threshold
```

The dispersion threshold in degrees.

4.4.2.2 duration_threshold

```
double gac_filter_parameter_t::duration_threshold
```

The duration threshold in milliseconds.

4.4.2.3 fixation

```
struct { ... } gac_filter_parameter_t::fixation
```

Fixation detection.

4.4.2.4 gap

```
struct { ... } gac_filter_parameter_t::gap
```

The gap filter parameter

4.4.2.5 is_heap

```
bool gac_filter_parameter_t::is_heap
```

Flag to indicate whether the struct was allocated on the heap.

4.4.2.6 max_gap_length

```
double gac_filter_parameter_t::max_gap_length
```

The maximal allowed gap length to be filled-in. Set to zero to disable gap fill-in filter.

4.4.2.7 mid_idx

```
uint32_t gac_filter_parameter_t::mid_idx
```

The mid index of the window. This is used to compute the length of the window: window_length = mid_idx ∗ 2 + 1. Set to zero to disable noise filtering.

4.4.2.8 noise

```
struct { ... } gac_filter_parameter_t::noise
```

Noise filter parameter

4.4.2.9 saccade

```
struct { ... } gac_filter_parameter_t::saccade
```

Saccade detection.

4.4.2.10 sample_period

```
double gac_filter_parameter_t::sample_period
```

The sample period to compute the number of required fill-in samples

4.4.2.11 type

```
gac_filter_noise_type_t gac_filter_parameter_t::type
```

The noise filter type.

4.4.2.12 velocity_threshold

```
float gac_filter_parameter_t::velocity_threshold
```

The velocity threshold in degrees per seconds.

The documentation for this struct was generated from the following file:

- include/gac.h

4.5 gac_filter_saccade_t Struct Reference

```
#include <gac.h>
```

Data Fields

- bool is_heap
- float velocity_threshold
- bool is_collecting
- uint32_t new_samples
- gac_queue_t window

4.5.1 Detailed Description

The saccade filter structure holding filter parameters.

gac_filter_saccade_s

4.5.2 Field Documentation

4.5.2.1 is_collecting

```
bool gac_filter_saccade_t::is_collecting
```

A flag indicating whether a saccade is ongoing

4.5.2.2 is_heap

```
bool gac_filter_saccade_t::is_heap
```

Flag to indicate whether the struct was allocated on the heap.

4.5.2.3 new_samples

```
uint32_t gac_filter_saccade_t::new_samples
```

Counter to keep track of new items in the parent queue.

4.5.2.4 velocity_threshold

```
float gac_filter_saccade_t::velocity_threshold
```

The velocity threshold

4.5.2.5 window

```
gac_queue_t gac_filter_saccade_t::window
```

A pointer to the sample queue

The documentation for this struct was generated from the following file:

- include/gac.h

4.6 gac_fixation_t Struct Reference

```
#include <gac.h>
```

Data Fields

- bool is_heap
- vec2 screen_point
- vec3 point
- double duration
- gac_sample_t first_sample

4.6.1 Detailed Description

A fixation sample.

gac_fixation_s

4.6.2 Field Documentation

4.6.2.1 duration

```
double gac_fixation_t::duration
```

The fixation duration in milliseconds.

4.6.2.2 first_sample

```
gac_sample_t gac_fixation_t::first_sample
```

The first sample of the fixation.

4.6.2.3 is_heap

```
bool gac_fixation_t::is_heap
```

Flag to indicate whether the struct was allocated on the heap.

4.6.2.4 point

```
vec3 gac_fixation_t::point
```

The fixation gaze point.

4.6.2.5 screen_point

```
vec2 gac_fixation_t::screen_point
```

The 2d fixation gaze point on the screen.

The documentation for this struct was generated from the following file:

- include/gac.h

4.7 gac_plane_t Struct Reference

```
#include <gac.h>
```

Data Fields

- bool is_heap
- vec3 p1
- vec3 p2
- vec3 p3
- vec3 e1
- vec3 e2
- vec3 norm
- mat4 m

4.7.1 Detailed Description

A genaral plane definition.

gac_plane_s

4.7.2 Field Documentation

4.7.2.1 e1

```
vec3 gac_plane_t::e1
```

The vector pointing from p1 to p2.

4.7.2.2 e2

```
vec3 gac_plane_t::e2
```

The vector pointing from p1 to p3.

4.7.2.3 is_heap

```
bool gac_plane_t::is_heap
```

Flag to indicate whether the struct was allocated on the heap.

4.7.2.4 m

```
mat4 gac_plane_t::m
```

Transformation matrix to transform a 3d gaze point to a 2d gaze point.

4.7.2.5 norm

```
vec3 gac_plane_t::norm
```

The normal of the screen surface.

4.7.2.6 p1

```
vec3 gac_plane_t::p1
```

A point on the plane 3d space.

4.7.2.7 p2

```
vec3 gac_plane_t::p2
```

A point on the plane 3d space.

4.7.2.8 p3

```
vec3 gac_plane_t::p3
```

A point on the plane 3d space.

The documentation for this struct was generated from the following file:

- include/gac.h

4.8 gac_queue_item_t Struct Reference

```
#include <gac.h>
```

Data Fields

- gac_queue_item_t ∗ next
- gac_queue_item_t ∗ prev
- void ∗ data

4.8.1 Detailed Description

A generic queue item.

gac_queue_item_s

4.8.2 Field Documentation

4.8.2.1 data

```
void* gac_queue_item_t::data
```

A pointer to the arbitrary data structure

4.8.2.2 next

```
gac_queue_item_t* gac_queue_item_t::next
```

A pointer to the next queue item (towards the head).

4.8.2.3 prev

```
gac_queue_item_t* gac_queue_item_t::prev
```

A pointer to the previous queue item (towards the tail).

The documentation for this struct was generated from the following file:

- include/gac.h

4.9 gac_queue_t Struct Reference

```
#include <gac.h>
```

Data Fields

- bool is_heap
- gac_queue_item_t ∗ tail
- gac_queue_item_t ∗ head
- uint32_t count
- uint32_t length
- void(∗ rm)(void ∗)

4.9.1 Detailed Description

A generic queue structure.

gac_queue_s

4.9.2 Field Documentation

4.9.2.1 count

```
uint32_t gac_queue_t::count
```

The number of occupied spaces.

4.9.2.2 head

```
gac_queue_item_t* gac_queue_t::head
```

A pointer to the tail to write to

4.9.2.3 is_heap

```
bool gac_queue_t::is_heap
```

Flag to indicate whether the struct was allocated on the heap.

4.9.2.4 length

```
uint32_t gac_queue_t::length
```

The number of total available spaces

4.9.2.5 rm

```
void( * gac_queue_t::rm) (void *)
```

The handler to remove data items

4.9.2.6 tail

```
gac_queue_item_t* gac_queue_t::tail
```

A pointer to the head of the queue to read from.

The documentation for this struct was generated from the following file:

- include/gac.h

4.10 gac_saccade_t Struct Reference

```
#include <gac.h>
```

Data Fields

- bool is_heap
- gac_sample_t first_sample
- gac_sample_t last_sample

4.10.1 Detailed Description

A saccade sample.

gac_saccade_s

4.10.2 Field Documentation

4.10.2.1 first_sample

```
gac_sample_t gac_saccade_t::first_sample
```

The first sample of the saccade.

4.10.2.2 is_heap

```
bool gac_saccade_t::is_heap
```

Flag to indicate whether the struct was allocated on the heap.

4.10.2.3 last_sample

```
gac_sample_t gac_saccade_t::last_sample
```

The last sample of the saccade.

The documentation for this struct was generated from the following file:

- include/gac.h

4.11 gac_sample_t Struct Reference

```
#include <gac.h>
```

Data Fields

- bool is_heap
- uint32_t trial_id
- vec2 screen_point
- vec3 point
- vec3 origin
- double timestamp
- double trial_onset
- double label_onset
- char * label

4.11.1 Detailed Description

The gaze data sample.

gac_sample_s

4.11.2 Field Documentation

4.11.2.1 is_heap

```
bool gac_sample_t::is_heap
```

Flag to indicate whether the struct was allocated on the heap.

4.11.2.2 label

```
char* gac_sample_t::label
```

Arbitrary label to annotate the sample.

4.11.2.3 label_onset

```
double gac_sample_t::label_onset
```

The time in milliseconds since the last change of label.

4.11.2.4 origin

```
vec3 gac_sample_t::origin
```

The gaze origin.

4.11.2.5 point

```
vec3 gac_sample_t::point
```

The gaze point.

4.11.2.6 screen_point

```
vec2 gac_sample_t::screen_point
```

The 2d gaze point on the screen.

4.11.2.7 timestamp

```
double gac_sample_t::timestamp
```

The sample timestamp.

4.11.2.8 trial_id

```
uint32_t gac_sample_t::trial_id
```

The ID of a ongoing trial.

4.11.2.9 trial_onset

```
double gac_sample_t::trial_onset
```

The time in milliseconds since the last change of trial ID.

The documentation for this struct was generated from the following file:

- include/gac.h

4.12 gac_screen_t Struct Reference

```
#include <gac.h>
```

Data Fields

- bool is_heap
- float width
- float height
- vec2 origin
- gac_plane_t plane

4.12.1 Detailed Description

Screen definition of the eye tracker.

gac_screen_s

4.12.2 Field Documentation

4.12.2.1 height

```
float gac_screen_t::height
```

The height of the screen.

4.12.2.2 is_heap

```
bool gac_screen_t::is_heap
```

Flag to indicate whether the struct was allocated on the heap.

4.12.2.3 origin

```
vec2 gac_screen_t::origin
```

The screen origin in 2d space.

4.12.2.4 plane

```
gac_plane_t gac_screen_t::plane
```

The underlying plane definition of the screen

4.12.2.5 width

```
float gac_screen_t::width
```

The width of the screen.

The documentation for this struct was generated from the following file:

- include/gac.h

4.13 gac_t Struct Reference

```
#include <gac.h>
```

Data Fields

- bool is_heap
- gac_queue_t samples
- gac_filter_fixation_t fixation
- gac_filter_gap_t gap
- gac_filter_saccade_t saccade
- gac_filter_noise_t noise
- gac_filter_parameter_t parameter
- gac_screen_t ∗ screen
- gac_sample_t ∗ last_sample

4.13.1 Detailed Description

The gaze analysis handler structure.

gac_s

4.13.2 Field Documentation

4.13.2.1 fixation

```
gac_filter_fixation_t gac_t::fixation
```

The fixation filter structure

4.13.2.2 gap

```
gac_filter_gap_t gac_t::gap
```

The gap filter structure

4.13.2.3 is_heap

```
bool gac_t::is_heap
```

Flag to indicate whether the struct was allocated on the heap.

4.13.2.4 last_sample

```
gac_sample_t* gac_t::last_sample
```

The last sample entered to the window. This remains even if the sample window is cleared.

4.13.2.5 noise

```
gac_filter_noise_t gac_t::noise
```

The noise filter structure

4.13.2.6 parameter

```
gac_filter_parameter_t gac_t::parameter
```

The parameters passed during configuration

4.13.2.7 saccade

```
gac_filter_saccade_t gac_t::saccade
```

The saccade filetr structure

4.13.2.8 samples

```
gac_queue_t gac_t::samples
```

The sample queue

4.13.2.9 screen

```
gac_screen_t* gac_t::screen
```

The screen information.

The documentation for this struct was generated from the following file:

- include/gac.h

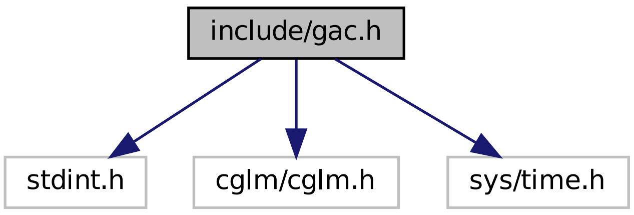
Chapter 5

File Documentation

5.1 include/gac.h File Reference

```
#include <stdint.h>
#include <cglm/cglm.h>
#include <sys/time.h>
```

Include dependency graph for gac.h:



Data Structures

- struct gac_sample_t
- struct gac_fixation_t
- struct gac_saccade_t
- struct gac_queue_item_t
- struct gac_queue_t
- struct gac_filter_fixation_t
- struct gac_filter_saccade_t
- struct gac_filter_noise_t
- struct gac_filter_gap_t
- struct gac_filter_parameter_t
- struct gac_plane_t
- struct gac_screen_t
- struct gac_t

Typedefs

- typedef enum gac_filter_noise_type_e gac_filter_noise_type_t

Enumerations

- enum gac_filter_noise_type_e { GAC_FILTER_NOISE_TYPE_AVERAGE, GAC_FILTER_NOISE_TYPE_MEDIAN }

Functions

- gac_t ∗ gac_create (gac_filter_parameter_t ∗parameter)
- void gac_destroy (gac_t ∗h)
- bool gac_init (gac_t ∗h, gac_filter_parameter_t ∗parameter)
- bool gac_get_filter_parameter (gac_t ∗h, gac_filter_parameter_t ∗parameter)
- bool gac_get_filter_parameter_default (gac_filter_parameter_t ∗parameter)
- bool gac_set_screen (gac_t ∗h, float top_left_x, float top_left_y, float top_left_z, float top_right_x, float top←_right_y, float top_right_z, float bottom_left_x, float bottom_left_y, float bottom_left_z)
- bool gac_filter_fixation (gac_filter_fixation_t ∗filter, gac_sample_t ∗sample, gac_fixation_t ∗fixation)
- gac_filter_fixation_t ∗ gac_filter_fixation_create (float dispersion_threshold, double duration_threshold)
- void gac_filter_fixation_destroy (gac_filter_fixation_t ∗filter)
- bool gac_filter_fixation_init (gac_filter_fixation_t ∗filter, float dispersion_threshold, double duration_threshold)
- bool gac_filter_fixation_step (gac_filter_fixation_t ∗filter, gac_sample_t ∗sample, gac_fixation_t ∗fixation)
- uint32_t gac_filter_gap (gac_filter_gap_t ∗filter, gac_queue_t ∗samples, gac_sample_t ∗sample)
- gac_filter_gap_t ∗ gac_filter_gap_create (double max_gap_length, double sample_period)
- void gac_filter_gap_destroy (gac_filter_gap_t ∗filter)
- bool gac_filter_gap_init (gac_filter_gap_t ∗filter, double max_gap_length, double sample_period)
- gac_sample_t ∗ gac_filter_noise (gac_filter_noise_t ∗filter, gac_sample_t ∗sample)
- gac_filter_noise_t ∗ gac_filter_noise_create (gac_filter_noise_type_t type, uint32_t mid_idx)
- void gac_filter_noise_destroy (gac_filter_noise_t ∗filter)
- bool gac_filter_noise_init (gac_filter_noise_t ∗filter, gac_filter_noise_type_t type, uint32_t mid_idx)
- gac_sample_t ∗ gac_filter_noise_average (gac_filter_noise_t ∗filter)
- bool gac_filter_saccade (gac_filter_saccade_t ∗filter, gac_sample_t ∗sample, gac_saccade_t ∗saccade)
- gac_filter_saccade_t ∗ gac_filter_saccade_create (float velocity_threshold)
- void gac_filter_saccade_destroy (gac_filter_saccade_t ∗filter)
- bool gac_filter_saccade_init (gac_filter_saccade_t ∗filter, float velocity_threshold)
- bool gac_filter_saccade_step (gac_filter_saccade_t ∗filter, gac_sample_t ∗sample, gac_saccade_←t ∗saccade)
- gac_fixation_t ∗ gac_fixation_create (vec2 ∗screen_point, vec3 ∗point, double duration, gac_sample_←t ∗first_sample)
- void gac_fixation_destroy (gac_fixation_t ∗fixation)
- bool gac_fixation_init (gac_fixation_t ∗fixation, vec2 ∗screen_point, vec3 ∗point, double duration, gac_←sample_t ∗first_sample)
- float gac_fixation_normalised_dispersion_threshold (float angle)
- gac_plane_t ∗ gac_plane_create (vec3 ∗p1, vec3 ∗p2, vec3 ∗p3)
- void gac_plane_destroy (gac_plane_t ∗plane)
- bool gac_plane_init (gac_plane_t ∗plane, vec3 ∗p1, vec3 ∗p2, vec3 ∗p3)
- bool gac_plane_intersection (gac_plane_t ∗plane, vec3 ∗origin, vec3 ∗dir, vec3 ∗intersection)
- bool gac_plane_point (gac_plane_t ∗plane, vec3 ∗point3d, vec2 ∗point2d)
- bool gac_queue_clear (gac_queue_t ∗queue)
- gac_queue_t ∗ gac_queue_create (uint32_t length)
- void gac_queue_destroy (gac_queue_t ∗queue)
- bool gac_queue_grow (gac_queue_t ∗queue, uint32_t count)

- bool gac_queue_init (gac_queue_t ∗queue, uint32_t length)
- bool gac_queue_pop (gac_queue_t ∗queue, void ∗∗data)
- bool gac_queue_push (gac_queue_t ∗queue, void ∗data)
- bool gac_queue_remove (gac_queue_t ∗queue)
- bool gac_queue_set_rm_handler (gac_queue_t ∗queue, void(∗rm)(void ∗))
- gac_saccade_t ∗ gac_saccade_create (gac_sample_t ∗first_sample, gac_sample_t ∗last_sample)
- void gac_saccade_destroy (gac_saccade_t ∗saccade)
- bool gac_saccade_init (gac_saccade_t ∗saccade, gac_sample_t ∗first_sample, gac_sample_t ∗last_sample)
- gac_sample_t ∗ gac_sample_create (vec2 ∗screen_point, vec3 ∗origin, vec3 ∗point, double timestamp, uint32_t trial_id, const char ∗label)
- gac_sample_t ∗ gac_sample_copy (gac_sample_t ∗sample)
- bool gac_sample_copy_to (gac_sample_t ∗dest, gac_sample_t ∗sample)
- void gac_sample_destroy (void ∗sample)
- bool gac_sample_init (gac_sample_t ∗sample, vec2 ∗screen_point, vec3 ∗origin, vec3 ∗point, double timestamp, uint32_t trial_id, const char ∗label)
- bool gac_sample_window_cleanup (gac_t ∗h)
- bool gac_sample_window_fixation_filter (gac_t ∗h, gac_fixation_t ∗fixation)
- bool gac_sample_window_saccade_filter (gac_t ∗h, gac_saccade_t ∗saccade)
- uint32_t gac_sample_window_update (gac_t ∗h, float ox, float oy, float oz, float px, float py, float pz, double timestamp, uint32_t trial_id, const char ∗label)
- uint32_t gac_sample_window_update_vec (gac_t ∗h, vec2 ∗screen_point, vec3 ∗origin, vec3 ∗point, double timestamp, uint32_t trial_id, const char ∗label)
- uint32_t gac_sample_window_update_screen (gac_t ∗h, float ox, float oy, float oz, float px, float py, float pz, float sx, float sy, double timestamp, uint32_t trial_id, const char ∗label)
- bool gac_samples_average_point (gac_queue_t ∗samples, vec3 ∗avg, uint32_t count)
- bool gac_samples_average_origin (gac_queue_t ∗samples, vec3 ∗avg, uint32_t count)
- bool gac_samples_average_screen_point (gac_queue_t ∗samples, vec2 ∗avg, uint32_t count)
- bool gac_samples_dispersion (gac_queue_t ∗samples, float ∗dispersion, uint32_t count)
- gac_screen_t ∗ gac_screen_create (vec3 ∗top_left, vec3 ∗top_right, vec3 ∗bottom_left)
- void gac_screen_destroy (gac_screen_t ∗screen)
- bool gac_screen_init (gac_screen_t ∗screen, vec3 ∗top_left, vec3 ∗top_right, vec3 ∗bottom_left)
- bool gac_screen_point (gac_screen_t ∗screen, vec3 ∗point3d, vec2 ∗point2d)

5.1.1 Typedef Documentation

5.1.1.1 gac_filter_noise_type_t

```
typedef enum gac_filter_noise_type_e gac_filter_noise_type_t
```

gac_filter_noise_type_e

5.1.2 Enumeration Type Documentation

5.1.2.1 gac_filter_noise_type_e

```
enum gac_filter_noise_type_e
```

The available noise filter types

Enumerator

GAC_FILTER_NOISE_TYPE_AVERAGE	Moving average filtering
GAC_FILTER_NOISE_TYPE_MEDIAN	[not implemented] Moving median filtering

5.1.3 Function Documentation

5.1.3.1 gac_create()

```
gac_t* gac_create (
            gac_filter_parameter_t * parameter )
```

Allocate the gaze analysis structure on the heap. This must be freed. If no parameter structure is provided default values are used. Refer to gac_init() for more information.

Parameters

parameter	An optional filter parameter structure.

Returns

A pointer to the allocated structure or NULL on failure.

5.1.3.2 gac_destroy()

```
void gac_destroy (
            gac_t * h )
```

Destroy the gaze analysis handler.

Parameters

h	A pointer to the gaze analysis handler.

5.1.3.3 gac_filter_fixation()

```
bool gac_filter_fixation (
            gac_filter_fixation_t * filter,
            gac_sample_t * sample,
            gac_fixation_t * fixation )
```

The fixation detection algorithm I-DT.

Parameters

filter	The gap filter structure holding the configuration parameters.
sample	The lastes sample
fixation	A location where a detected fixation is stored. This is only valid if the function returns true.

Returns

True if a fixation was detected, false otherwise.

5.1.3.4 gac_filter_fixation_create()

```
gac_filter_fixation_t* gac_filter_fixation_create (
            float dispersion_threshold,
            double duration_threshold )
```

Allocate a new fixation filter structure on the heap. This structure must be freed.

Parameters

dispersion_threshold	The dispersion thresholad in degrees.
duration_threshold	The duration threshold in milliseconds.

Returns

The allocated fixation filter structure or NULL on failure.

5.1.3.5 gac_filter_fixation_destroy()

```
void gac_filter_fixation_destroy (
            gac_filter_fixation_t * filter )
```

Destroy the fixation filter structure.

Parameters

filter	A pointer to the structure to destroy.

5.1.3.6 gac_filter_fixation_init()

```
bool gac_filter_fixation_init (
            gac_filter_fixation_t * filter,
            float dispersion_threshold,
            double duration_threshold )
```

Initialise a fixation filter structure.

Parameters

filter	The filter structure to initialise.
dispersion_threshold	The dispersion thresholad in degrees.
duration_threshold	The duration threshold in milliseconds.

Returns

True on success, false on failure.

5.1.3.7 gac_filter_fixation_step()

```
bool gac_filter_fixation_step (
            gac_filter_fixation_t * filter,
            gac_sample_t * sample,
            gac_fixation_t * fixation )
```

Internal function to compute the fixation detection algorithm I-DT. Do not use this function. INstead use either the function gac_sample_window_fixation_filter() or gac_filter_fixation().

Parameters

filter	The gap filter structure holding the configuration parameters.
sample	The lastes sample
fixation	A location where a detected fixation is stored. This is only valid if the function returns true.

Returns

True if a fixation was detected, false otherwise.

5.1.3.8 gac_filter_gap()

```
uint32_t gac_filter_gap (
            gac_filter_gap_t * filter,
            gac_queue_t * samples,
            gac_sample_t * sample )
```

Fill in gaps between the last sample and the current sample if any. The number of samples to be filled in depends on the sample period. To avoid filling up large gaps the gap filling is limited to a maximal gap length (in milliseconds). The sample passed to the function is added as well.

Parameters

filter	The gap filter structure holding the configuration parameters.
samples	The sample queue to be filled in
sample	The lastes sample

Returns

The number of samples added to the sample window.

5.1.3.9 gac_filter_gap_create()

```
gac_filter_gap_t* gac_filter_gap_create (
            double max_gap_length,
            double sample_period )
```

Allocate the filter gap structure on the heap. this needs to be freed.

Parameters

max_gap_length	The maximal gap length in milliseconds to fil-in. Larger gaps are ignored. If set to 0 the filter is disabled.
sample_period	The expected average sample period in milliseconds (1000 / sample_rate).

Returns

A pointer to the allocated filter gap structure.

5.1.3.10 gac_filter_gap_destroy()

```
void gac_filter_gap_destroy (
            gac_filter_gap_t * filter )
```

Destroy the gap filter structure.

Parameters

filter	A pointer to the structure to destroy.

5.1.3.11 gac_filter_gap_init()

```
bool gac_filter_gap_init (
            gac_filter_gap_t * filter,
            double max_gap_length,
            double sample_period )
```

Initialise a filter gap structure.

Parameters

filter	A pointer to the struct to be initialised.
max_gap_length	The maximal gap length in milliseconds to fil-in. Larger gaps are ignored. If set to 0 the filter is disabled.
sample_period	The expected average sample period in milliseconds (1000 / sample_rate).

Returns

True on success, false on failure.

5.1.3.12 gac_filter_noise()

```
gac_sample_t* gac_filter_noise (
            gac_filter_noise_t * filter,
            gac_sample_t * sample )
```

A noise filter. The filter consecutively collects samples into a window and returns a filtered value when the window is full, otherwise the passed sample is returned. The filter maintains its won sample window.

Parameters

filter	The filter parameters.
sample	The sample to add to the filter window.

Returns

A filtered sample if the filter window is full or the sample passed to the function otherwise.

5.1.3.13 gac_filter_noise_average()

```
gac_sample_t* gac_filter_noise_average (
            gac_filter_noise_t * filter )
```

A moving average noise filter. It computes the average sample point and origin from all samples in the filter window and assigns the timestamp of the median sample (the sample in the middle of the window) to the averaged sample.

Parameters

filter	The filter parameters

Returns

A new averaged sample if the filter window is full or the sample passed to the function otherwise.

5.1.3.14 gac_filter_noise_create()

```
gac_filter_noise_t* gac_filter_noise_create (
            gac_filter_noise_type_t type,
            uint32_t mid_idx )
```

Allocate the noise filter structure. This needs to be freed.

Parameters

type	The noise filter type.
mid_idx	The mid index of the window. This is used to compute the length of the window: window_length = mid_idx ∗ 2 + 1. If set to 0 the filter is disabled.

Returns

A pointer to the allocated structure or NULL on failure.

5.1.3.15 gac_filter_noise_destroy()

```
void gac_filter_noise_destroy (
            gac_filter_noise_t * filter )
```

Destroy the noise filter structure.

Parameters

filter	A pointer to the structure to destroy.

5.1.3.16 gac_filter_noise_init()

```
bool gac_filter_noise_init (
            gac_filter_noise_t * filter,
            gac_filter_noise_type_t type,
            uint32_t mid_idx )
```

Initialises a noise filter structure.

Parameters

filter	A pointer to the structure to initialise.
type	The noise filter type.
mid_idx	The mid index of the window. This is used to compute the length of the window: window_length = mid_idx ∗ 2 + 1. If set to 0 the filter is disabled.

Returns

True on success, false on failure.

5.1.3.17 gac_filter_saccade()

```
bool gac_filter_saccade (
            gac_filter_saccade_t * filter,
            gac_sample_t * sample,
            gac_saccade_t * saccade )
```

The saccade detection algorithm I-VT.

Parameters

filter	The filter parameters
sample	The lastes sample
saccade	A location where a detected saccade is stored. This is only valid if the function returns true.

Returns

True if a saccade was detected, false otherwise.

5.1.3.18 gac_filter_saccade_create()

```
gac_filter_saccade_t* gac_filter_saccade_create (
            float velocity_threshold )
```

Allocate a new saccade filter structure on the heap. This needs to be freed.

Parameters

velocity_threshold	The velocity threshold in degrees per second.

Returns

A pointer to the allocated filter structure or NUll on failure.

5.1.3.19 gac_filter_saccade_destroy()

```
void gac_filter_saccade_destroy (
            gac_filter_saccade_t * filter )
```

Destroy the saccade filter structure.

Parameters

filter	A pointer to the structure to destroy.

5.1.3.20 gac_filter_saccade_init()

```
bool gac_filter_saccade_init (
            gac_filter_saccade_t * filter,
            float velocity_threshold )
```

Initialise a saccade filter structure.

Parameters

filter	A pointer to the filter structure to initialise.
velocity_threshold	The velocity threshold in degrees per second.

Returns

True on success, false on failure.

5.1.3.21 gac_filter_saccade_step()

```
bool gac_filter_saccade_step (
            gac_filter_saccade_t * filter,
            gac_sample_t * sample,
            gac_saccade_t * saccade )
```

Internal function to compute the I-VT algorithm. Do not use this function. Instead use either the function gac_sample_window_saccade_filter() or gac_filter_saccade().

Parameters

filter	The filter parameters
sample	The lastes sample
saccade	A location where a detected saccade is stored. This is only valid if the function returns true.

Returns

True if a saccade was detected, false otherwise.

5.1.3.22 gac_fixation_create()

```
gac_fixation_t* gac_fixation_create (
            vec2 * screen_point,
            vec3 * point,
            double duration,
            gac_sample_t * first_sample )
```

Allocate a new fixation structure on the heap. This structure must be freed.

Parameters

screen_point	The fixation screen point.
point	The fixation point.
duration	The duration of the fixation.
first_sample	The first sample in the fixation.

Returns

The allocated fixation structure or NULL on failure.

5.1.3.23 gac_fixation_destroy()

```
void gac_fixation_destroy (
            gac_fixation_t * fixation )
```

Destroy a fixation structure.

Parameters

fixation	A pointer to the fixation structure to destroy.

5.1.3.24 gac_fixation_init()

```
bool gac_fixation_init (
            gac_fixation_t * fixation,
            vec2 * screen_point,
            vec3 * point,
            double duration,
            gac_sample_t * first_sample )
```

Initialise a fixation structure.

Parameters

fixation	The fixation structure to initialise.
screen_point	The fixation screen point.
point	The fixation point.
duration	The duration of the fixation.
first_sample	The first sample in the fixation.

Returns

True on success, false on failure.

5.1.3.25 gac_fixation_normalised_dispersion_threshold()

```
float gac_fixation_normalised_dispersion_threshold (
            float angle )
```

Compute a dispersion threashold assuming a unit distance. To get the actual dispersion threshold multiply this by the distance of the gaze origin to the gaze point.

Parameters

angle	The angel in degrees for which the dispersion threshold is computetd. Usual values range from 0.5 to 1 degree.

Returns

The normalized dispersion threshold.

5.1.3.26 gac_get_filter_parameter()

```
bool gac_get_filter_parameter (
            gac_t * h,
            gac_filter_parameter_t * parameter )
```

Get the filter parameters.

Parameters

h	A pointer to the gaze analysis structure to initialise.
parameter	A location where the filter parameter values can be stored.

Returns

True on success, false on failure.

5.1.3.27 gac_get_filter_parameter_default()

```
bool gac_get_filter_parameter_default (
            gac_filter_parameter_t * parameter )
```

Get the default filter parameter values.

Parameters

parameter	A location where the filter parameter values can be stored.

Returns

True on success, false on failure.

5.1.3.28 gac_init()

```
bool gac_init (
            gac_t * h,
            gac_filter_parameter_t * parameter )
```

Initialise the gaze analysis structure.

If no parameter structure is provided the following default values are set:

- fixation.dispersion_threshold = 0.5;
- fixation.duration_threshold = 100;
- saccade.velocity_threshold = 20;
- noise.mid_idx = 1;
- noise.type = GAC_FILTER_NOISE_TYPE_AVERAGE;
- gap.max_gap_length = 50;
- gap.sample_period = 16.67;

Parameters

h	A pointer to the gaze analysis structure to initialise.
parameter	An optional filter parameter structure.

Returns

True on success, false on failure.

5.1.3.29 gac_plane_create()

```
gac_plane_t* gac_plane_create (
             vec3 * p1,
             vec3 * p2,
             vec3 * p3 )
```

Allocate a plane in 3d space. This need to be freed with `gac_plane_destroy()`.

Parameters

p1	The 3d coordinates of a point in 3d space.
p2	The 3d coordinates of a point in 3d space.
p3	The 3d coordinates of a point in 3d space.

Returns

A pointer to the allocated plane or NULL on failure.

5.1.3.30 gac_plane_destroy()

```
void gac_plane_destroy (
             gac_plane_t * plane )
```

Destroy a plane structure.

Parameters

plane	A pointer to the plane structure to destroy.

5.1.3.31 gac_plane_init()

```
bool gac_plane_init (
             gac_plane_t * plane,
             vec3 * p1,
             vec3 * p2,
             vec3 * p3 )
```

Initialise a plane in 3d space.

Parameters

plane	A pointer to the plane structure to initialise.
p1	The 3d coordinates of a point in 3d space.
p2	The 3d coordinates of a point in 3d space.
p3	The 3d coordinates of a point in 3d space.

Returns

True on succes and false on failure.

5.1.3.32 gac_plane_intersection()

```
bool gac_plane_intersection (
            gac_plane_t * plane,
            vec3 * origin,
            vec3 * dir,
            vec3 * intersection )
```

Compute the 3d intersection point with a plane.

Parameters

plane	A pointer to the plane structure.
origin	The origin of the gaze.
dir	The gaze direction.
intersection	A location to store the intersection point. This is only valid if the function returns true.

Returns

True if an intersection was found, false otherwise.

5.1.3.33 gac_plane_point()

```
bool gac_plane_point (
            gac_plane_t * plane,
            vec3 * point3d,
            vec2 * point2d )
```

Transform a 3d gaze point into a 2d point on a plane. This only works for 3d points which coincide with the plane. To compute an intersection use the function gac_plane_intersection().

Parameters

plane	A pointer to the plane structure.
point3d	The 3d point to transform.
point2d	A location where the 2d point will be stored. This is only valid if the function returns true.

Returns

True on success, false otherwise.

5.1.3.34 gac_queue_clear()

```
bool gac_queue_clear (
             gac_queue_t * queue )
```

Remove all data items from the queue. The queue remove handler is used to free the data.

Parameters

queue	The queue to clear

Returns

True on success, false on failure.

5.1.3.35 gac_queue_create()

```
gac_queue_t* gac_queue_create (
             uint32_t length )
```

Allocate a new queue structure. This needs to be freed.

Parameters

length	The length of the queue.

Returns

The allocated queue structure.

5.1.3.36 gac_queue_destroy()

```
void gac_queue_destroy (
             gac_queue_t * queue )
```

Destroy a queue, all ist items and all data inside the items.

Parameters

queue	A pointer to the queue to destroy

5.1.3.37 gac_queue_grow()

```
bool gac_queue_grow (
            gac_queue_t * queue,
            uint32_t count )
```

Grow the queue.

Parameters

queue	A pointer to the queue to grow.
count	The number of spaces to add.

Returns

True on success, false on failure.

5.1.3.38 gac_queue_init()

```
bool gac_queue_init (
            gac_queue_t * queue,
            uint32_t length )
```

Initialise a queue structure.

Parameters

queue	A pointer to the queue to initialise.
length	The length of the queue

Returns

True on success, false on failure.

5.1.3.39 gac_queue_pop()

```
bool gac_queue_pop (
            gac_queue_t * queue,
            void ** data )
```

Remove a the data from the head of the queue and link the the now free space to the tail of the queue.

Parameters

queue	A pointer to the queue.
data	An optional location to store the popped data.

Returns

True on success, false on failure.

5.1.3.40 gac_queue_push()

```
bool gac_queue_push (
            gac_queue_t * queue,
            void * data )
```

Add a new item to the tail of the queue. If no more space is available, the queue is grown by one.

Parameters

queue	A pointer to the queue.
data	The data sample to be added to the tail of the queue.

Returns

True on success, false on failure.

5.1.3.41 gac_queue_remove()

```
bool gac_queue_remove (
            gac_queue_t * queue )
```

The same as gac_queue_pop() but also freeing the data item with the configured remove handler.

Parameters

queue	A pointer to the queue.

Returns

True on success, false on failure.

5.1.3.42 gac_queue_set_rm_handler()

```
bool gac_queue_set_rm_handler (
            gac_queue_t * queue,
            void(*)(void *) rm )
```

Set a remove handler which will be called whenever an item is removed from the queue.

Parameters

queue	A pointer to the queue.
rm	The renmove handler.

Returns

True on success, false on failure.

5.1.3.43 gac_saccade_create()

```
gac_saccade_t* gac_saccade_create (
            gac_sample_t * first_sample,
            gac_sample_t * last_sample )
```

Allocate a new saccade structure on the heap. This needs to be freed.

Parameters

first_sample	The first sample of the saccade, holding the source point.
last_sample	The last sample of the saccade, holding the target point.

Returns

The allocated saccade structure on success or NULL on failure.

5.1.3.44 gac_saccade_destroy()

```
void gac_saccade_destroy (
            gac_saccade_t * saccade )
```

Destroy a saccade structure.

Parameters

saccade	A pointer to the saccade structure to destroy.

5.1.3.45 gac_saccade_init()

```
bool gac_saccade_init (
            gac_saccade_t * saccade,
            gac_sample_t * first_sample,
            gac_sample_t * last_sample )
```

Initialise a saccade structure.

Parameters

saccade	A pointer to the saccade structure to initialise.
first_sample	The first sample of the saccade, holding the source point.
last_sample	The last sample of the saccade, holding the target point.

Returns

True on success, false on failure.

5.1.3.46 gac_sample_copy()

```
gac_sample_t* gac_sample_copy (
            gac_sample_t * sample )
```

Create a deep copy of a sample. This needs to be freed with gac_sample_destroy().

Parameters

sample	The sample to copy

Returns

A pointer to the new sample or NULL.

5.1.3.47 gac_sample_copy_to()

```
bool gac_sample_copy_to (
            gac_sample_t * dest,
            gac_sample_t * sample )
```

Deep copy of a sample to a target. This needs to be freed with gac_sample_destroy().

Parameters

dest	The location where the sample will be copied to.
sample	The sample to copy

Returns

A pointer to the new sample or NULL.

5.1.3.48 gac_sample_create()

```
gac_sample_t* gac_sample_create (
            vec2 * screen_point,
            vec3 * origin,
            vec3 * point,
            double timestamp,
            uint32_t trial_id,
            const char * label )
```

Allocate a new sample structure on the heap. This needs to be freed.

Parameters

screen_point	The 2d screen gaze point vector.
origin	The gaze origin vector.
point	The gaze point vector.
timestamp	The timestamp of the sample.
trial_id	The ID of the ongoing trial.
label	An optional arbitrary label annotating the sample.

Returns

The allocated sample structure or NULL on failure.

5.1.3.49 gac_sample_destroy()

```
void gac_sample_destroy (
            void * sample )
```

Destroy a sample structure.

Parameters

sample	A pointer to the structure to be destroyed.

5.1.3.50 gac_sample_init()

```
bool gac_sample_init (
            gac_sample_t * sample,
            vec2 * screen_point,
            vec3 * origin,
            vec3 * point,
            double timestamp,
            uint32_t trial_id,
            const char * label )
```

Initialise a sample structure.

Parameters

sample	The sample structure to initialise.
screen_point	The 2d screen gaze point vector.
origin	The gaze origin vector.
point	The gaze point vector.
timestamp	The timestamp of the sample.
trial_id	The ID of the ongoing trial.
label	An optional arbitrary label annotating the sample.

Returns

True on success, false on failure.

5.1.3.51 gac_sample_window_cleanup()

```
bool gac_sample_window_cleanup (
            gac_t * h )
```

Cleanup the sample window. This removes all sample data from the sample window which is no longer used for the gaze analysis.

Parameters

h	A pointer to the gaze analysis handler.

Returns

True on success, false on failure.

5.1.3.52 gac_sample_window_fixation_filter()

```
bool gac_sample_window_fixation_filter (
            gac_t * h,
            gac_fixation_t * fixation )
```

The fixation detection algorithm I-DT. This acts on the sample window managed by the functions gac_sample_window_update() and gac_sample_window_cleanup().

Parameters

h	A pointer to the gaze analysis handler.
fixation	A location where a detected fixation is stored. This is only valid if the function returns true.

Returns

True if a fixation was detected, false otherwise.

5.1.3.53 gac_sample_window_saccade_filter()

```
bool gac_sample_window_saccade_filter (
            gac_t * h,
            gac_saccade_t * saccade )
```

The saccade detection algorithm I-VT. This acts on the sample window managed by the functions gac_sample_window_update() and gac_sample_window_cleanup().

Parameters

h	A pointer to the gaze analysis handler.
saccade	A location where a detected saccade is stored. This is only valid if the function returns true.

Returns

True if a saccade was detected, false otherwise.

5.1.3.54 gac_sample_window_update()

```
uint32_t gac_sample_window_update (
            gac_t * h,
            float ox,
            float oy,
            float oz,
            float px,
            float py,
            float pz,
            double timestamp,
            uint32_t trial_id,
            const char * label )
```

Update the sample window with a new sample. If noise filtering is enabled the filtered data is added to the sample window and the raw sample is dismissed. If gap filtering is enabled, sample gaps are filled-in with interpolated data samples.

Parameters

h	A pointer to the gaze analysis handler.
ox	The x coordinate of the gaze origin.
oy	The y coordinate of the gaze origin.
oz	The z coordinate of the gaze origin.
px	The x coordinate of the gaze point.
py	The y coordinate of the gaze point.
pz	The z coordinate of the gaze point.
timestamp	The timestamp of the sample.
trial_id	The ID of the ongoing trial.
label	An optional arbitrary label annotating the sample.

Returns

The number of new samples added to the window.

5.1.3.55 gac_sample_window_update_screen()

```
uint32_t gac_sample_window_update_screen (
            gac_t * h,
            float ox,
            float oy,
            float oz,
            float px,
            float py,
            float pz,
            float sx,
            float sy,
            double timestamp,
            uint32_t trial_id,
            const char * label )
```

Update the sample window with a new sample. If noise filtering is enabled the filtered data is added to the sample window and the raw sample is dismissed. If gap filtering is enabled, sample gaps are filled-in with interpolated data samples.

Parameters

h	A pointer to the gaze analysis handler.
ox	The x coordinate of the gaze origin.

Parameters

oy	The y coordinate of the gaze origin.
oz	The z coordinate of the gaze origin.
px	The x coordinate of the gaze point.
py	The y coordinate of the gaze point.
pz	The z coordinate of the gaze point.
sx	The x coordinate of the screen gaze point.
sy	The y coordinate of the screen gaze point.
timestamp	The timestamp of the sample.
trial_id	The ID of the ongoing trial.
label	An optional arbitrary label annotating the sample.

Returns

The number of new samples added to the window.

5.1.3.56 gac_sample_window_update_vec()

```
uint32_t gac_sample_window_update_vec (
            gac_t * h,
            vec2 * screen_point,
            vec3 * origin,
            vec3 * point,
            double timestamp,
            uint32_t trial_id,
            const char * label )
```

Update sample window with a new sample.

Parameters

h	A pointer to the gaze analysis handler.
screen_point	The 2d screen gaze point
origin	The gaze origin.
point	The gaze point.
timestamp	The timestamp of the sample.
trial_id	The ID of the ongoing trial.
label	An optional arbitrary label annotating the sample.

Returns

The number of new samples added to the window.

5.1.3.57 gac_samples_average_origin()

```
bool gac_samples_average_origin (
            gac_queue_t * samples,
            vec3 * avg,
            uint32_t count )
```

Compute the average gaze origin of samples in the sample window.

Parameters

samples	A pointer to the sample window.
avg	A location to store the average gaze origin. This is only valid if the function returns true.
count	The number of samples to perform the computation on, starting by the queue tail (newest first). If 0 is passed, all samples are included.

5.1.3.58 gac_samples_average_point()

```
bool gac_samples_average_point (
            gac_queue_t * samples,
            vec3 * avg,
            uint32_t count )
```

Compute the average gaze point of samples in the sample window.

Parameters

samples	A pointer to the sample window.
avg	A location to store the average gaze point. This is only valid if the function returns true.
count	The number of samples to perform the computation on, starting by the queue tail (newest first). If 0 is passed, all samples are included.

5.1.3.59 gac_samples_average_screen_point()

```
bool gac_samples_average_screen_point (
            gac_queue_t * samples,
            vec2 * avg,
            uint32_t count )
```

Compute the average screen gaze point of samples in the sample window.

Parameters

samples	A pointer to the sample window.
avg	A location to store the average gaze point. This is only valid if the function returns true.
count	The number of samples to perform the computation on, starting by the queue tail (newest first). If 0 is passed, all samples are included.

5.1.3.60 gac_samples_dispersion()

```
bool gac_samples_dispersion (
             gac_queue_t * samples,
             float * dispersion,
             uint32_t count )
```

Compute the gaze point dispersion of samples in the sample window.

Parameters

samples	A pointer to the sample window.
dispersion	A location to store the dispersion value. This is only valid if the function returns true.
count	The number of samples to perform the computation on, starting by the queue tail (newest first). If 0 is passed, all samples are included.

5.1.3.61 gac_screen_create()

```
gac_screen_t* gac_screen_create (
             vec3 * top_left,
             vec3 * top_right,
             vec3 * bottom_left )
```

Allocate the screen structure. This needs to be freed with `gac_screen_destroy()`. The screen is defined through the top left, the top right and the bottom left point of the screen in 3d space. The width, the height, and the bottom right point of the screen are computed based on these three points. Make sure to provide points that describe a rectangle for this to make sense.

Parameters

top_left	The 3d coordinates of the top left screen point.
top_right	The 3d coordinates of the top right screen point.
bottom_left	The 3d coordinates of the bottom left screen point.

Returns

A pointer to the allocated screen structure or NULL on failure.

5.1.3.62 gac_screen_destroy()

```
void gac_screen_destroy (
             gac_screen_t * screen )
```

Destroy a screen structure.

Parameters

screen	A pointer to the screen structure to destroy

5.1.3.63 gac_screen_init()

```
bool gac_screen_init (
             gac_screen_t * screen,
             vec3 * top_left,
             vec3 * top_right,
             vec3 * bottom_left )
```

Initialise a screen structure through the top left, the top right and the bottom left point of the screen in 3d space. The width, the height, and the bottom right point of the screen are computed based on these three points. Make sure to provide points that describe a rectangle for this to make sense.

Parameters

screen	A pointer to the screen structure to initialise.
top_left	The 3d coordinates of the top left screen point.
top_right	The 3d coordinates of the top right screen point.
bottom_left	The 3d coordinates of the bottom left screen point.

Returns

True on succes and false on failure.

5.1.3.64 gac_screen_point()

```
bool gac_screen_point (
             gac_screen_t * screen,
             vec3 * point3d,
             vec2 * point2d )
```

Transform a 3d gaze point into a normalized 2d point on the screen. (0, 0) represents the top left corner of the screen and (1, 1) represents the bottom right corner.

Parameters

screen	A pointer to the screen structure.
point3d	The 3d point to transform.
point2d	A location where the 2d point will be stored. This is only valid if the function returns true.

Returns

True on success, false otherwise.

5.1.3.65 gac_set_screen()

```
bool gac_set_screen (
             gac_t * h,
             float top_left_x,
             float top_left_y,
             float top_left_z,
             float top_right_x,
             float top_right_y,
             float top_right_z,
             float bottom_left_x,
             float bottom_left_y,
             float bottom_left_z )
```

Configure the screen position in 3d space. This allows to compute 2d gaze point coordinates.

Parameters

h	A pointer to the gaze analysis handler.
top_left_x	The x coordinate of the top left screen corner.
top_left_y	The y coordinate of the top left screen corner.
top_left_z	The z coordinate of the top left screen corner.
top_right_x	The x coordinate of the top right screen corner.
top_right_y	The y coordinate of the top right screen corner.
top_right_z	The z coordinate of the top right screen corner.
bottom_left↩_x	The x coordinate of the bottom left screen corner.
bottom_left↩_y	The y coordinate of the bottom left screen corner.
bottom_left↩_z	The z coordinate of the bottom left screen corner.

Returns

True on success, false on failure.

Index