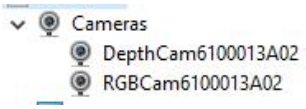
3D Camera SDK Docs

Camera connection

If you can't preview 3D-camera, please open the Device Manager to check if the device is connected properly. If the device is connected successfully, two cameras will appear as shown below.



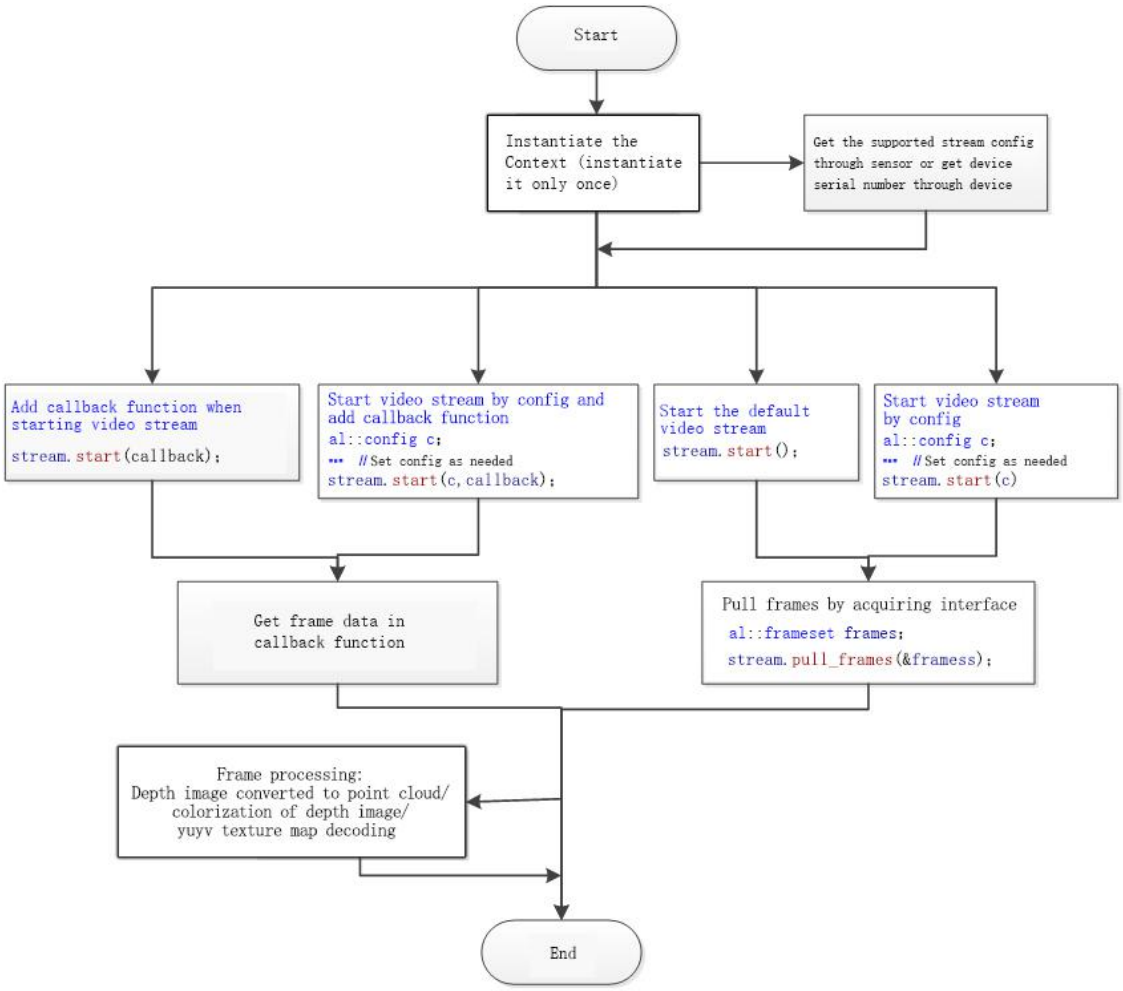
SDK overview

The current version of the 3D-camera SDK only supports Win8 and Win10.
3D-camera SDK includes two interfaces, C and C++, which can be selected as needed.
Please include 3DCamera.h to call C interface, include 3DCamera.hpp to call C++ interface.

Add static library 3DCamera.lib to compile it, add 3DCamera.dll to run it.

AilookViewer is a demo software developed with this SDK, VisualStudio2013 or above and Qt 5 need to be installed before running it. The C++ interface is introduced as follows.

Interface calling flowchart



Class and interface details

Context

```
al::context ctx;
```

You need to instantiate the class first, and make sure to instantiate it only once. Get the device or open the stream through this class.

STREAM

- Method 1: Start the default data stream (output a depth camera stream and color camera stream)

```
al::streaming stream(ctx);
al::streaming_ctrl ctrl = stream.start();
```

- Method 2: You can configure the parameters of the stream through *config* and start the video stream through *config*. You can set the required resolution, format and framerate. If multiple devices work at the same time, you must set the device serial number. (For the supported stream settings, please refer to the Sensor introduction)

```
al::config c;
c.enable_stream(STREAM_DEPTH, 640, 400, FORMAT_Z16, 2);
c.enable_stream(STREAM_COLOR, 800, 600, FORMAT_MJPG, 15);
c.enable_device(serial_number);
al::streaming_ctrl ctrl = stream.start(c);
```

Stream format description:

```
FORMAT_Z16        //Depth image format of depth camera (16-bit value, 0.1mm per unit)
FORMAT_Y16        //Left and right original infrared image format of depth camera
FORMAT_MJPG       //Image format of color camera
```

There are two methods to get frame data:

- Method 1: When calling *start* without setting callback function, the frame data is pulled from acquiring interface. If the acquisition succeeds, return value is *true*.

```
al::frameset frames;
bool ret = stream.pull_frames(&frames);
if (ret)
{
    auto color = frames.get_color_frame();
    auto depth = frames.get_depth_frame();
}
```

- Method 2: Get the image through the callback function (set the callback function when calling *start*)

```
 auto callback = [&](const al::frame& frame)
 {
    if (al::frameset fs = frame.as<al::frameset>()){
        // All synchronized stream will arrive in a single frameset
```

```
    }
    else{
        // Get single frame
    }
};
al::streaming_ctrl ctrl= stream.start(callback);
```

DEVICE

A device represents a 3D Camera, and a device contains a depth sensor and an RGB sensor. The sensors under the device can be enumerated and the device information can be obtained through the interface in the device class.

```
//Get the device list
al::device_list devicelist = ctx.query_device_info_list();

//Get device serial number, firmware version, algorithm version and other information
std::string serial_number = device.get_info(CAMERA_INFO_SERIAL_NUMBER));

//Get sensor list
std::vector<sensor> sensors = device.query_sensors();

//Send algorithmic control commands
//Parameters are integers, parameters are separated by spaces and passed as string.
char param[100] = { 0 };
sprintf_s(param, "%d %d", value1, value2);
device.execute_commond(COMMAND_TYPE_Z_RANGE, param);
```

Supported commands are as follows:

```
COMMAND_TYPE_Z_RANGE,                 //set depth range [min max]
COMMAND_TYPE_CONTRAST,                //set reliability threshold by contrast [threshold]
COMMAND_TYPE_AUTO_EXPOSURE_MODE,      //auto exposure mode [mode]
COMMAND_TYPE_AUTO_EXPOSURE_AREA,      //auto exposure area setting by top-left point and
                                       bottom-right point, coordinate of which normalized to
                                       0~100) [top_left_x, top_left_y, bottom_right_x,
                                       bottom_right_y]
```

```
//wifi on/off setting
device.enable_wifi(false);
```

SENSOR

Get the stream profiles supported by the sensor and modify some parameters of the sensor through the interface in the *sensor* class.

```
// Get the format/resolution/frame rate information available for the sensor
std::vector<al::stream_profile> profiles = sensor.get_stream_profiles();
for (auto profile : profiles)
{
    int width = profile.width();
    int height = profile.height();
    ...
}
// Set sensor parameters such as exposure time and gain
if (sensor.supports(OPTION_GAIN))
{
    //Get option range
    al::option_range r = sensor.get_option_range(OPTION_GAIN);
    //Get current option
    currGain = sensor.get_option(OPTION_GAIN);
    //Set new option
    sensor.set_option(OPTION_GAIN, 1);
}
```

Note: The maximum exposure time is related to the frame rate, so get the actual maximum exposure value in *al::stream_profile*.

Process

yuyv decoding

```
//yuyv frame converted to rgb frame
al::yuy_decoder yuv_decoder;
auto rgb_frame = yuv_decoder.process(color);
```

Colorized depth image

```
//depth frame converted to rgb frame
al::colorizer colorize;
auto depth_frame = colorize.process(depth); //update view
```

Point cloud processing

```
//Get camera parameters
sensor_intrinsics color_intrinsics =
color.get_profile().as<al::video_stream_profile>().get_intrinsics();
sensor_extrinsics extrinsics =
color.get_profile().as<al::video_stream_profile>().get_extrinsics();
sensor_intrinsics depth_intrinsics =
depth.get_profile().as<al::video_stream_profile>().get_intrinsics();
//Generate point cloud by depth data and parameters
al::Pointcloud pc;
pc.generatePoints((unsigned short *)depth.get_data(), depth.get_width(),
depth.get_height(), 10, (al::st_intrinsics *)&depth_intrinsics, (al::st_extrinsics
*)&extrinsics, (al::st_intrinsics *)&color_intrinsics);
//After processing, the vertex, texture mapping coordinates and normal of the point cloud can be
obtained, and export_to_file can be called to save the file.
std::vector<st_point> vertices = pc.getVertices();
std::vector<st_texture> textures = pc.getTexcoords();
std::vector<st_normal> normals = pc.getNormals();
pc.export_to_file(…);
```

Depth map filtering

```
//filterSize is the filter value and must be an odd number.
 static int MeidianBlur(T *depthData, int width, int height, int filterSize)
```