Access AKDK Application Software with Femto Bolt

Access AKDK Application Software with Femto Bolt

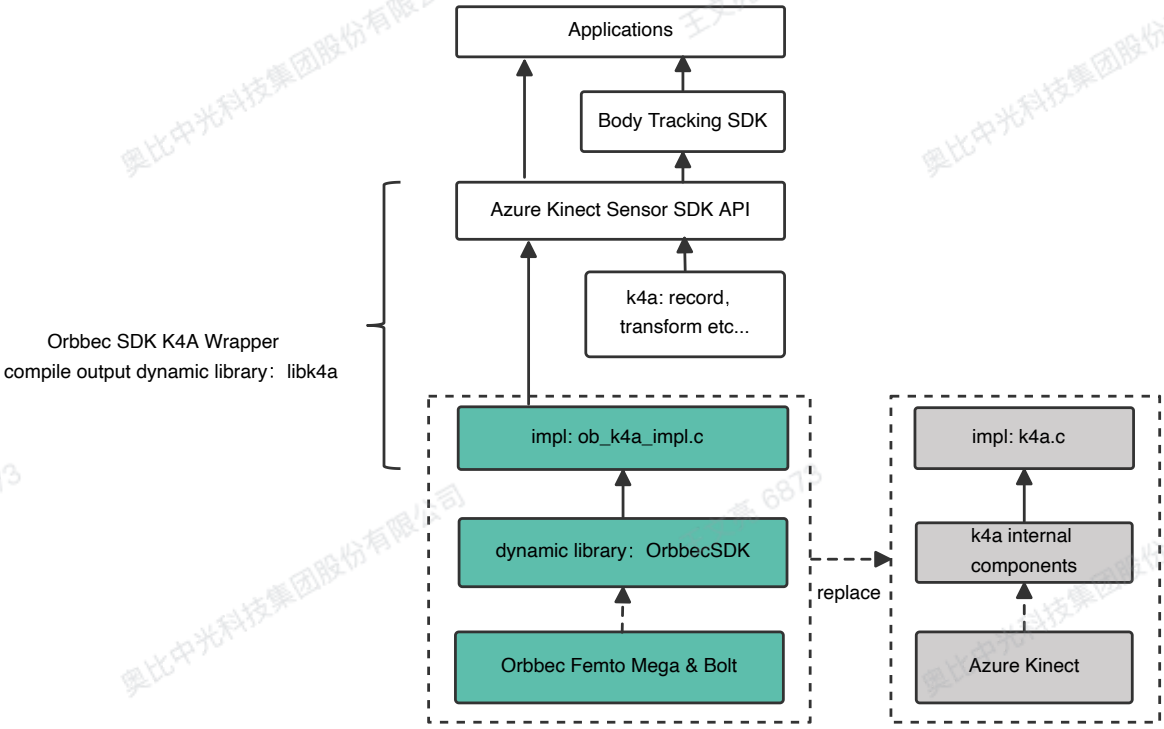
1. Overview

Orbbec SDK K4A Wrapper is designed and encapsulated based on Orbbec SDK, converting Orbbec SDK to Azure Kinect Sensor SDK interfaces. It mainly implements data stream reception, color parameter setting, D2C and point cloud functions, recording and playback, consistent APIs with Azure Kinect Sensor SDK, allowing users to quickly switch to Orbbec Femto Bolt and Orbbec Femto Mega cameras without modifying the application code.

Orbbec SDK K4A Wrapper open source link: https://github.com/orbbec/OrbbecSDK-K4A-Wrapper

The implementation principle of Orbbec SDK K4A Wrapper is as follows:

The characteristics of Orbbec SDK K4A Wrapper are as follows:



- Maintain the original interfaces of Azure Kinect Sensor SDK unchanged.
- Modify the implementation (impl) of Azure Kinect Sensor SDK C API, call Orbbec SDK internally to get video frames and control Femto Bolt and Femto Mega cameras.
- Coordinate transformation, D2C, C2D, point cloud reuse Azure Kinect Sensor SDK.

Orbbec SDK K4A Wrapper currently supports the following cameras:

Orbbec Femto Bolt:

Win10 x86/x64, Ubuntu18.04 x64, Ubuntu20.04 x64

Orbbec Femto Mega:

Win10 x86/x64, Ubuntu20.04 x64

2. How AKDK applications switch to Orbbec SDK K4A Wrapper

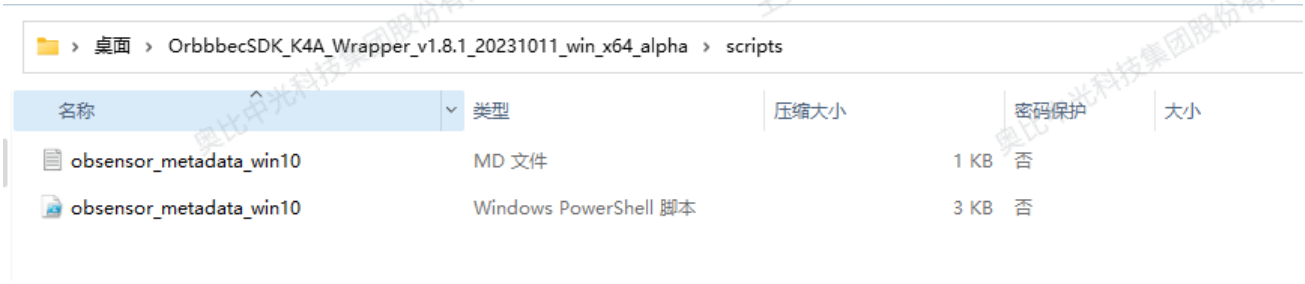
This chapter mainly shows how to implement code project switching and adaptation to Femto Bolt cameras by replacing Azure Kinect Sensor SDK library files and header files with Orbbec SDK K4A Wrapper under the user application code project.

For compiled application executables, you can also directly switch by replacing the libk4a library file linked by the application executable with Orbbec SDK K4A Wrapper.

2.1 Windows Platform Application

2.1.1 Environment Configuration: Modify Registry

Due to the Windows system mechanism, for UVC protocol devices, if you need to get timestamp and other metadata information, you need to register in the registry first. You can execute the obsensor_metadata_win10.ps1 script to complete automatic registration according to the obsensor_metadata_win10.md document guide.



2.1.2 Replace Header Files

Orbbec SDK K4A Wrapper is developed based on Azure Kinect Sensor SDK V1.4.1 version. The header files directly use the original K4A library header files. If the user project originally used Azure Kinect Sensor SDK V1.4.1 version, this step can be ignored.

- Find the header files in Orbbec SDK K4A Wrapper

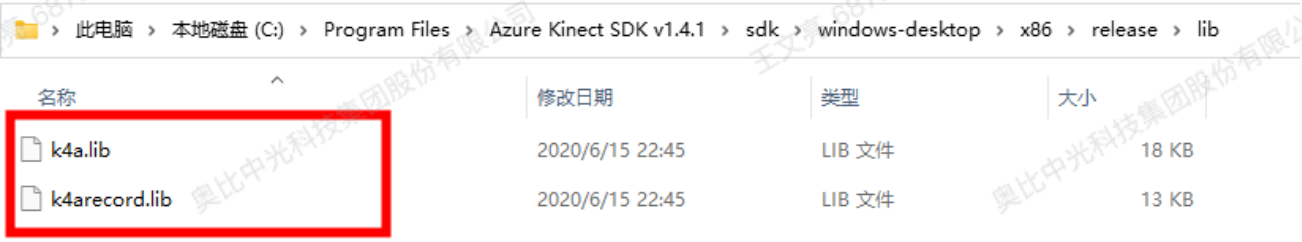
- Replace the corresponding header files in Azure Kinect Sensor SDK

2.1.3 Replace Library Files

Compile

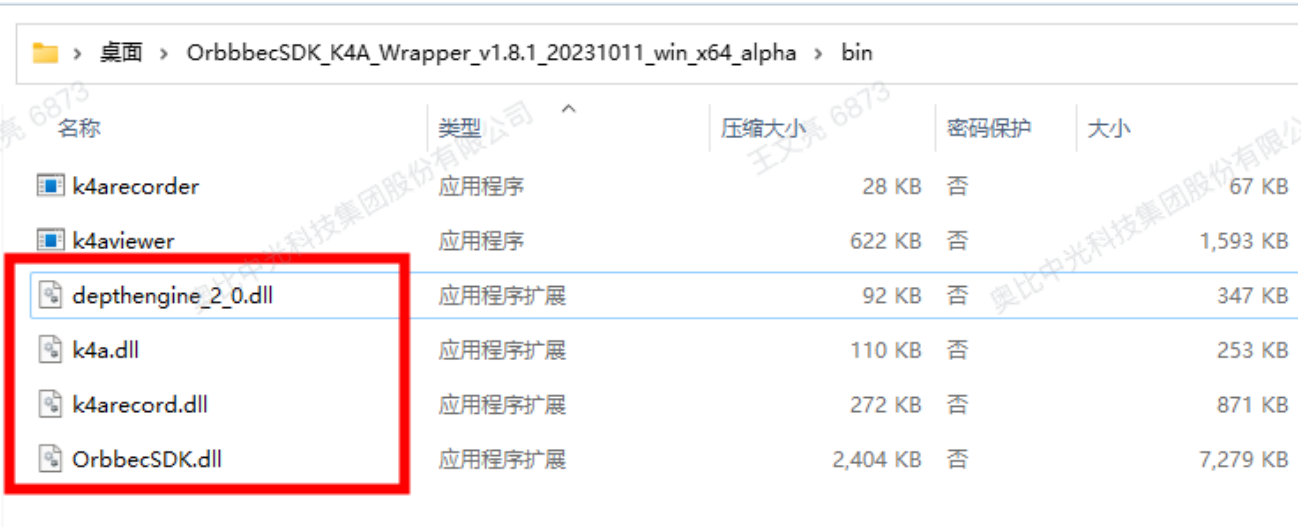
Copy the k4a.lib and k4arecord.lib library files from Orbbec SDK K4A Wrapper to the corresponding Azure Kinect Sensor SDK path.

Replace the corresponding k4a.lib and k4arecord.lib library files in Azure Kinect Sensor SDK to compile the application program with Orbbec SDK K4A Wrapper.

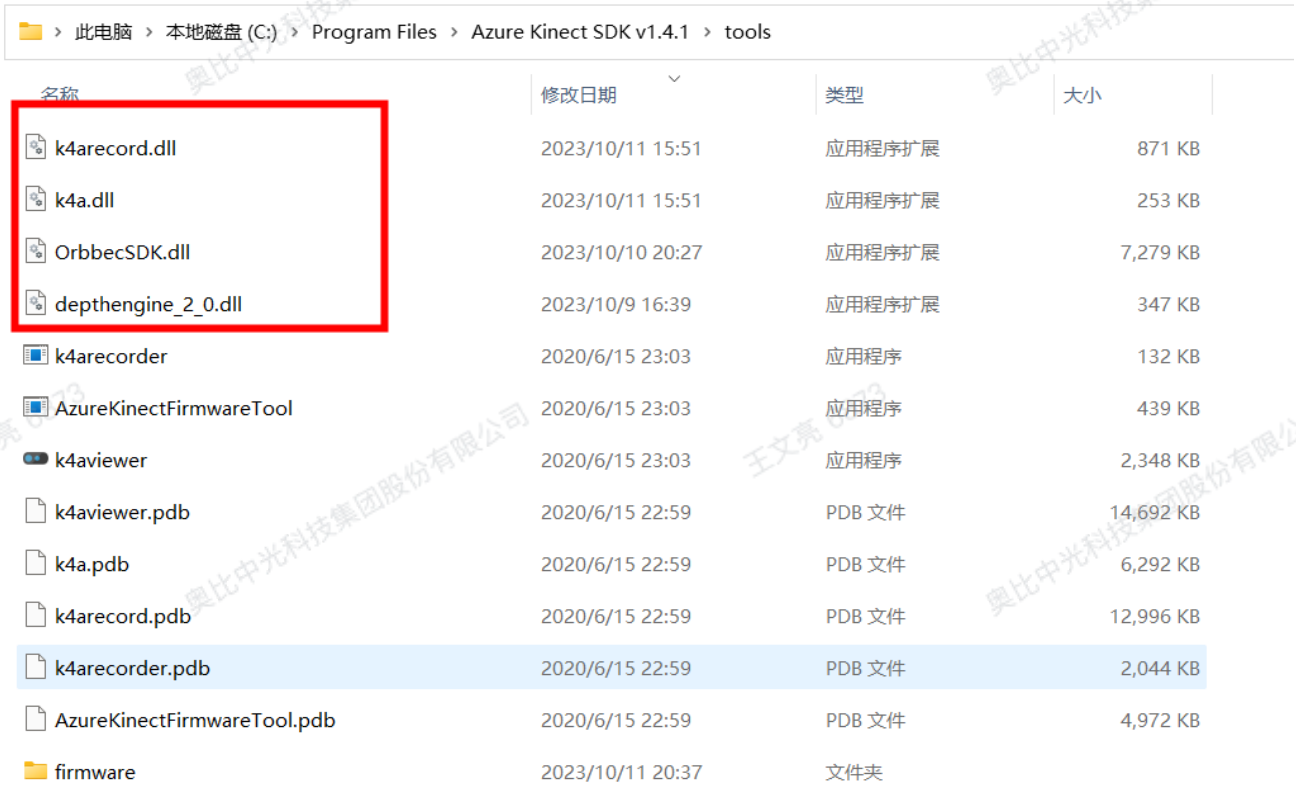


Run

Find k4a.dll, k4arecord.dll, OrbbecSDK.dll, depthengine_2_0.dll library files in Orbbec SDK K4A Wrapper



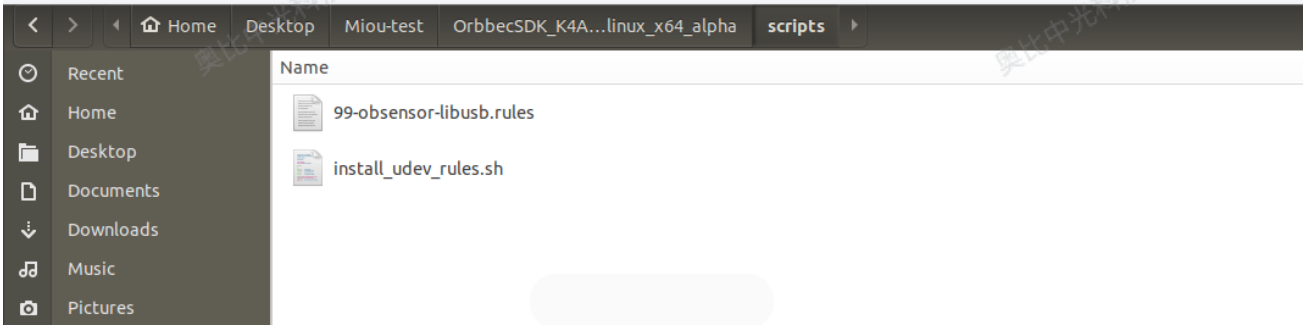
Copy the above library files to the original executable application path of Azure Kinect Sensor SDK, and replace the original k4a.dll, k4arecord.dll, depthengine_2_0.dll library files to run the application program.



2.2 Linux Platform Application

2.2.1 Environment Configuration: Install udev rules Configuration

By default, Linux systems require root permissions for direct access to USB devices, which can be solved through the rules configuration file. Orbbec SDK K4A Wrapper provides a 99-obsensor-libusb.rules configuration file and install_udev_rules.sh installation script to complete the installation by executing the install_udev_rules.sh script.



Execution method:

```
sudo chmod +x ./install.sh # Make sure the installation script is executable
sudo ./install.sh # Execute the script with sudo
```

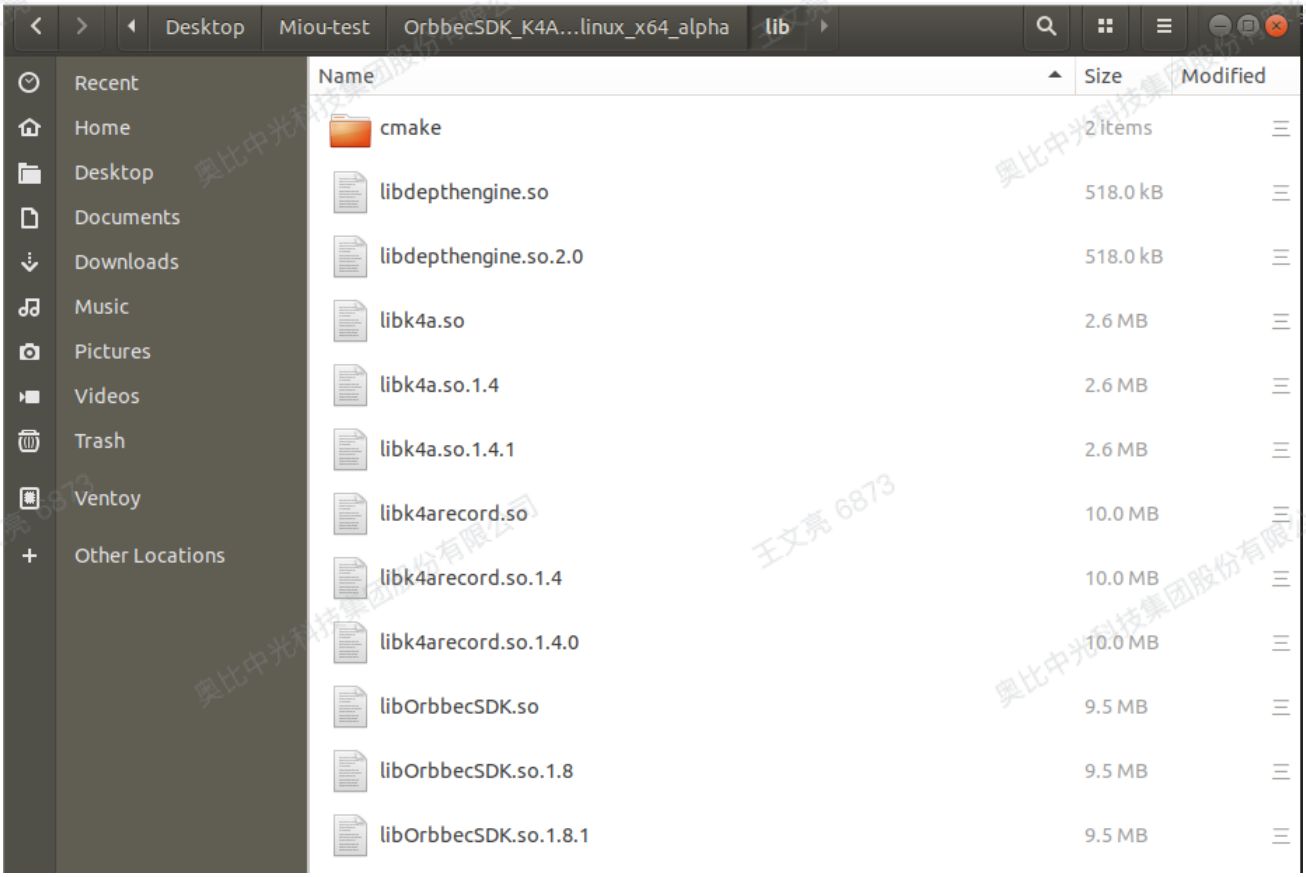
2.2.2 Replace Header Files

Orbbec SDK K4A Wrapper is developed based on Azure Kinect Sensor SDK V1.4.1 version. The header files directly use the original K4A library header files. If the user project originally used Azure Kinect Sensor SDK V1.4.1 version, this step can be ignored.

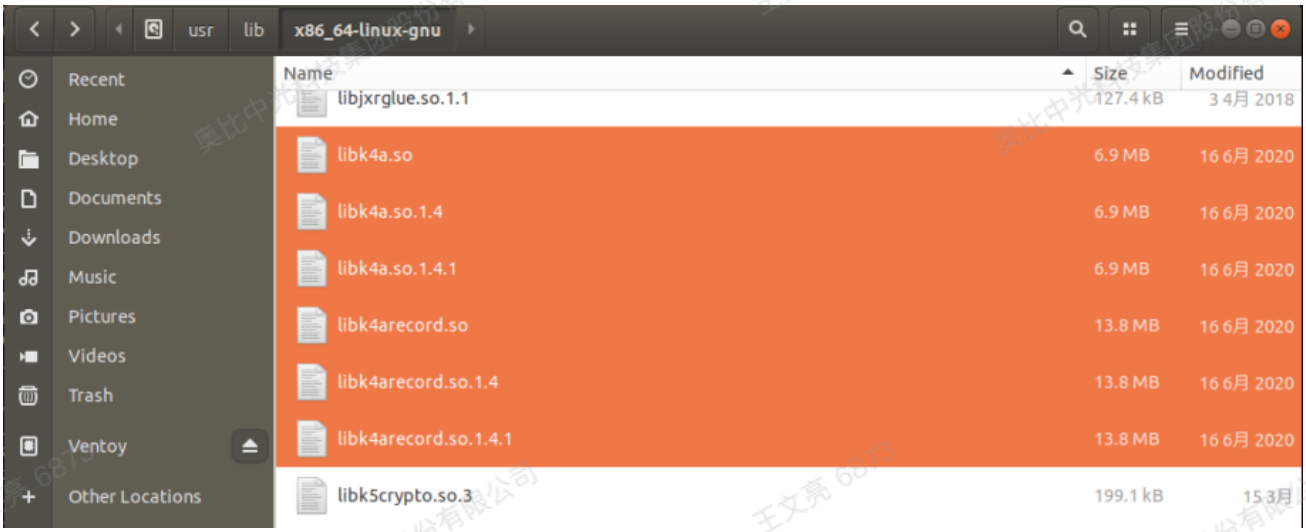
Find the original header files of Orbbec SDK K4A Wrapper and replace the corresponding header files under Azure Kinect Sensor SDK.

2.2.3 Replace Library Files

Find the library files in Orbbec SDK K4A Wrapper, including: libdepthengine, libk4a, libk4arecord, libOrbbecSDK.so.



Copy the above library files to the /usr/lib/x86_64-linux-gnu directory to replace the Azure Kinect Sensor SDK libraries to compile and use the application program with Orbbec SDK K4A Wrapper.



Note:

On Linux platforms, the depth engine library requires OpenGL. If the application layer also uses OpenGL and makes cross-thread calls to OpenGL instances, context conflicts may occur, causing problems during depth engine initialization and inability to start the depth data stream properly.

If the application layer uses OpenGL rendering, the application layer needs to switch the Context under OpenGL:
(https://www.khronos.org/opengl/wiki/OpenGL_and_multithreading)

The following is our solution using the glfw library:

a. Call glfwMakeContextCurrent(NULL) before starting the stream.

b. Then start the stream.

c. After the stream is finished, glfwMakeContextCurrent(currentContext).

3. AKDK User Skeleton Algorithm Adaptation

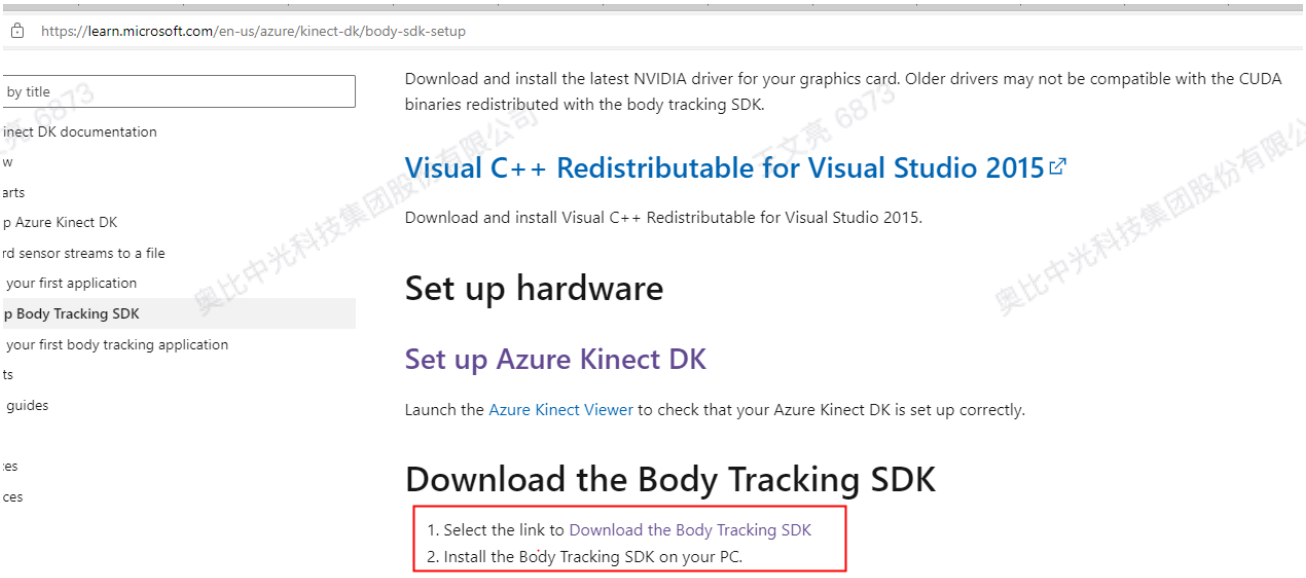
The following uses the Azure Kinect Body Tracking SDK Sample as an example to describe how to obtain skeleton data by using Orbbec Femto Bolt camera data after replacing Azure Kinect Sensor SDK through Azure Kinect Body Tracking SDK (hereinafter referred to as K4ABT). (Users can switch by directly replacing the library files without recompiling)

3.1 Windows Platform

3.1.1 Skeleton Installation Package Download and Installation

Complete the installation according to Microsoft's installation documentation:
https://learn.microsoft.com/en-us/azure/kinect-dk/body-sdk-setup

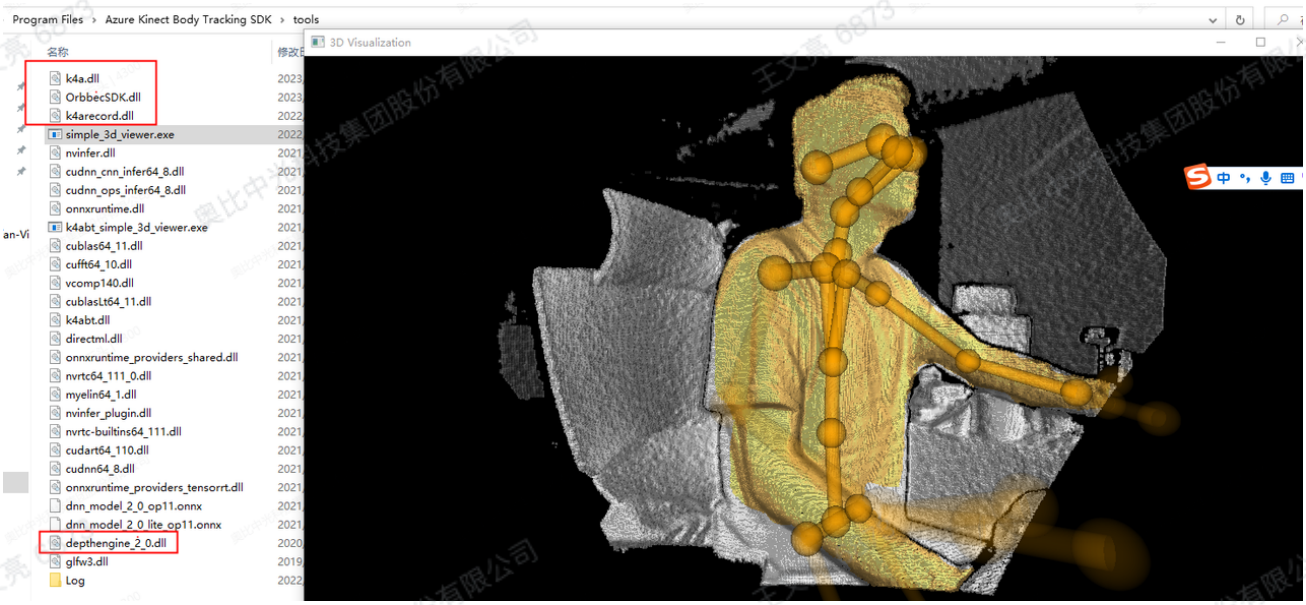
Download Azure Kinect Body Tracking SDK 1.1.2.msi, then execute the file to complete the installation.



3.1.2 Demonstrate Skeleton Algorithm Effects

Complete the environment configuration (metadata registration) according to Chapter 2, then replace the following libraries of Azure kinect Sensor SDK in the installation directory with Orbbec SDK K4A Wrapper libraries (k4a.dll, OrbbecSDK.dll, k4arecord.dll, depthengine_2_0.dll), then run simple_3d_viewer.exe with administrator rights.

The effect of the skeleton algorithm is as follows:



3.1.3 Skeleton Algorithm Secondary Development (sample compilation)

1. Download Azure Kinect Samples

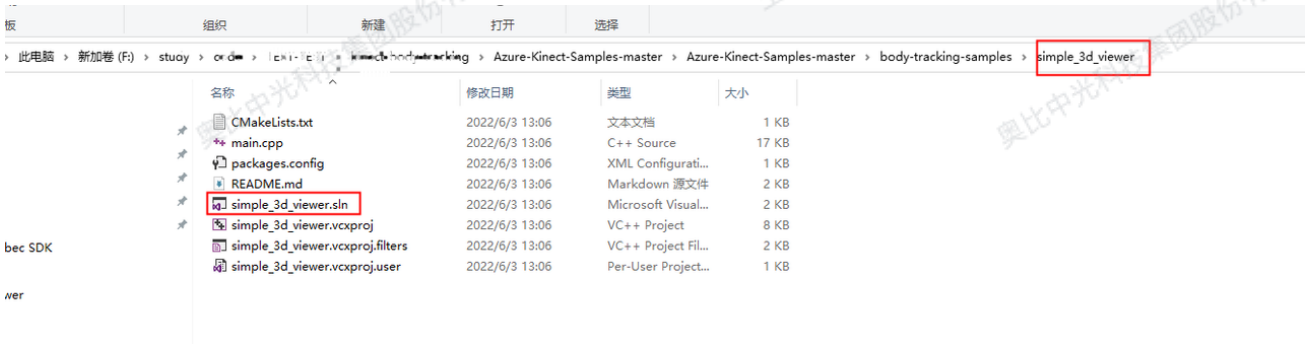
URL: https://github.com/microsoft/Azure-Kinect-Samples

Clone code:

```
git clone https://github.com/microsoft/Azure-Kinect-Samples.git
```

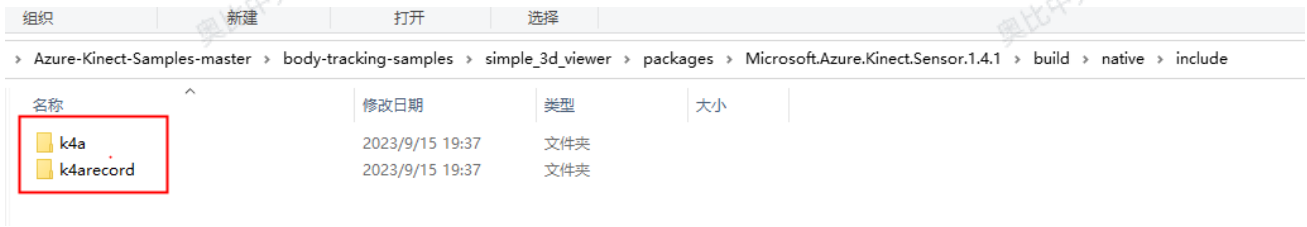
2. Open Microsoft skeleton sample with Visual Studio

Microsoft skeleton sample only supports opening with visual studio. Use VS2019 to open the following projects.

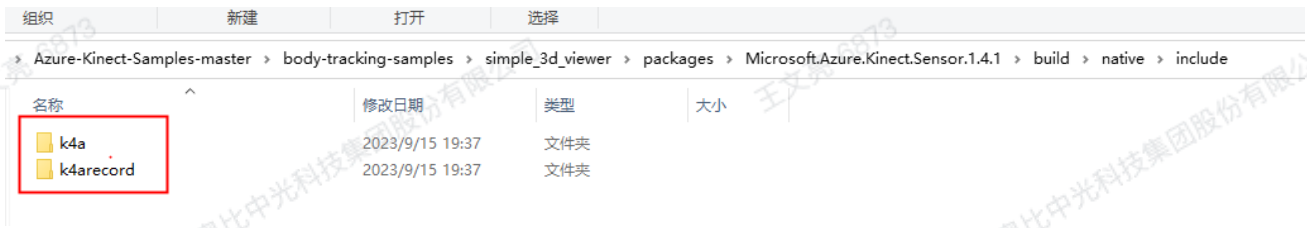


3. Use Orbbec SDK K4A Wrapper header files and library files to replace Azure Kinect Sensor SDK header files and library files.（k4a.dll, OrbbecSDK.dll, k4arecord.dll, depthengine_2_0.dll)。

a. Replace header files

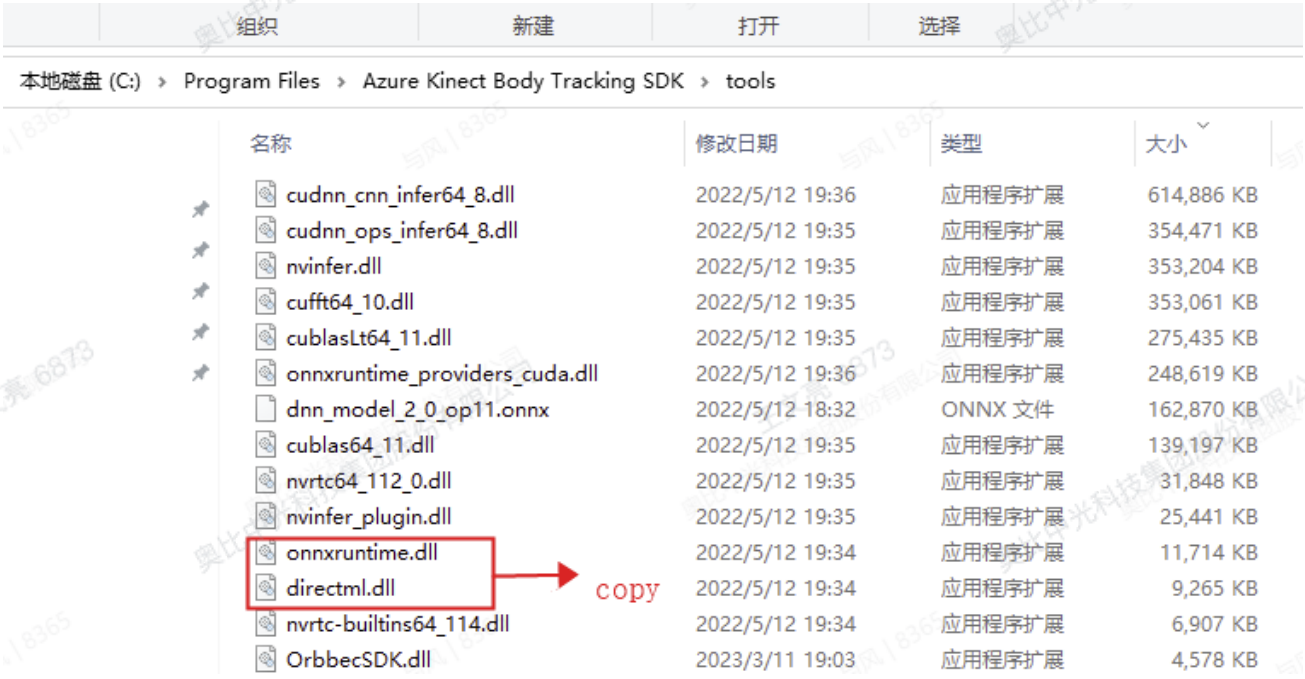


b. Replace library files

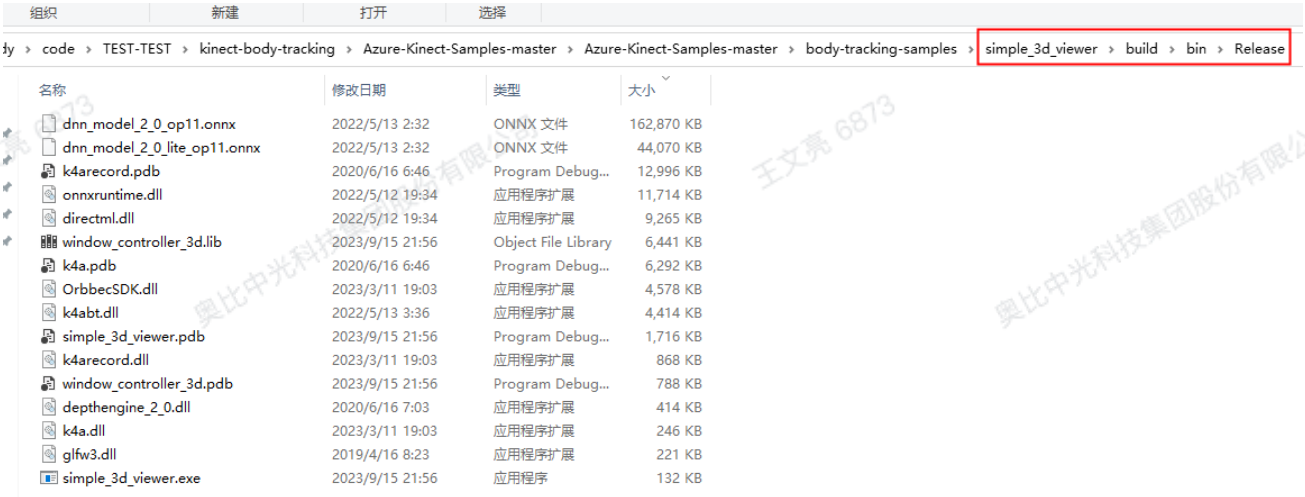


4. Compile & Run

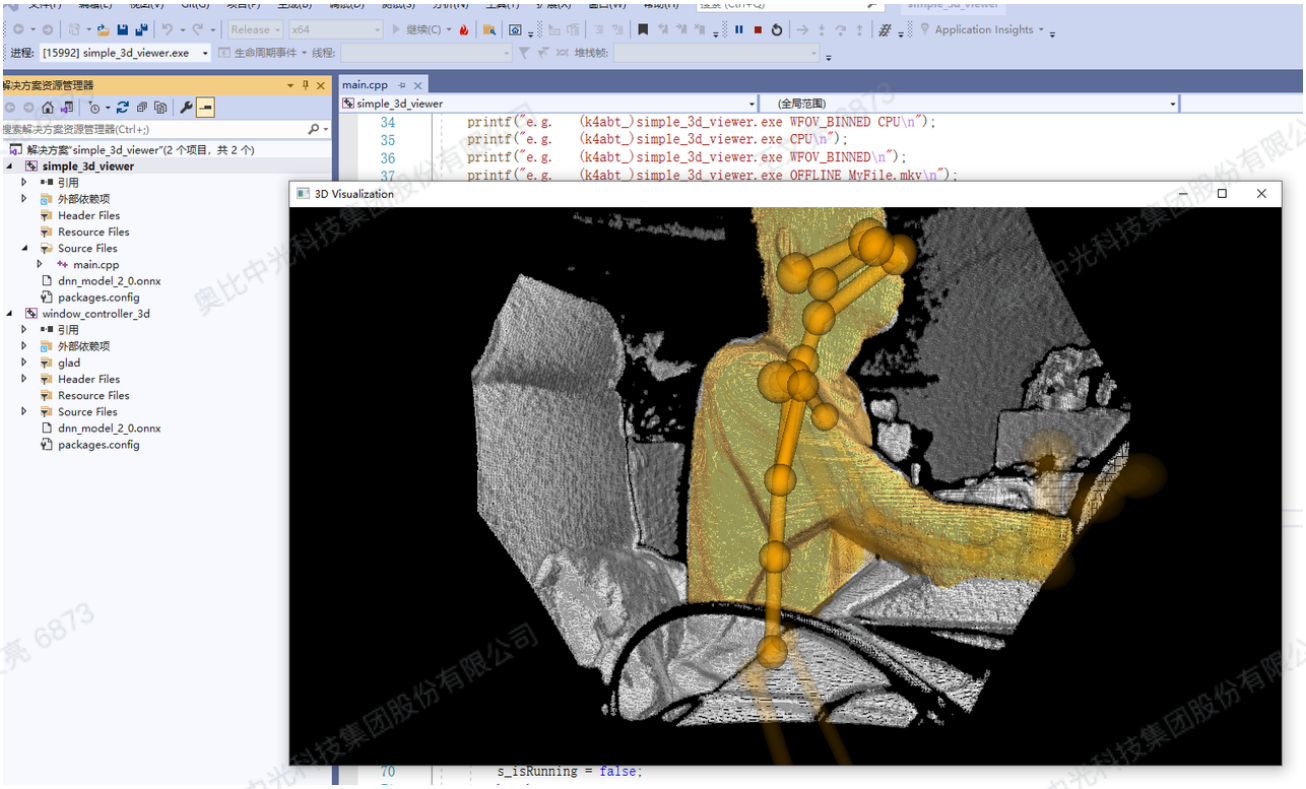
a. The simple_3d_viewer.exe example relies on the following 2 libraries, which are copied from the installation package to the bin directory where the compiled application is generated.



b. Copy the dependent libraries to the running directory:



c. Compilation running interface:



3.2 Running Kinect Skeleton Algorithm on Linux

Since Microsoft Azure Kinect Sensor SDK libraries only provide installation on Ubuntu 18.04, it is recommended to complete the following on Ubuntu 18.04 system.

3.2.1 Install Azure Kinect Sensor SDK

Refer to the Linux installation instructions section of Microsoft's installation documentation to complete the installation. Document link:

Azure Kinect Sensor SDK download | Microsoft Learn

Installation instructions:

```
curl -sSL -O https://packages.microsoft.com/config/ubuntu/18.04/packages-microsoft-prod.deb
sudo dpkg -i packages-microsoft-prod.deb
rm packages-microsoft-prod.deb
sudo apt-get update
sudo apt-get install libk4a1.4-dev
sudo apt-get install k4a-tools
```

3.2.2 Install Azure Kinect Body Tracking SDK

Refer to the Linux installation instructions section of Microsoft's installation documentation to complete the installation. Document link:

Azure Kinect Body Tracking SDK download | Microsoft Learn

The following instructions were executed during installation of Azure Kinect Sensor SDK and do not need to be repeated:

Install libk4abt:

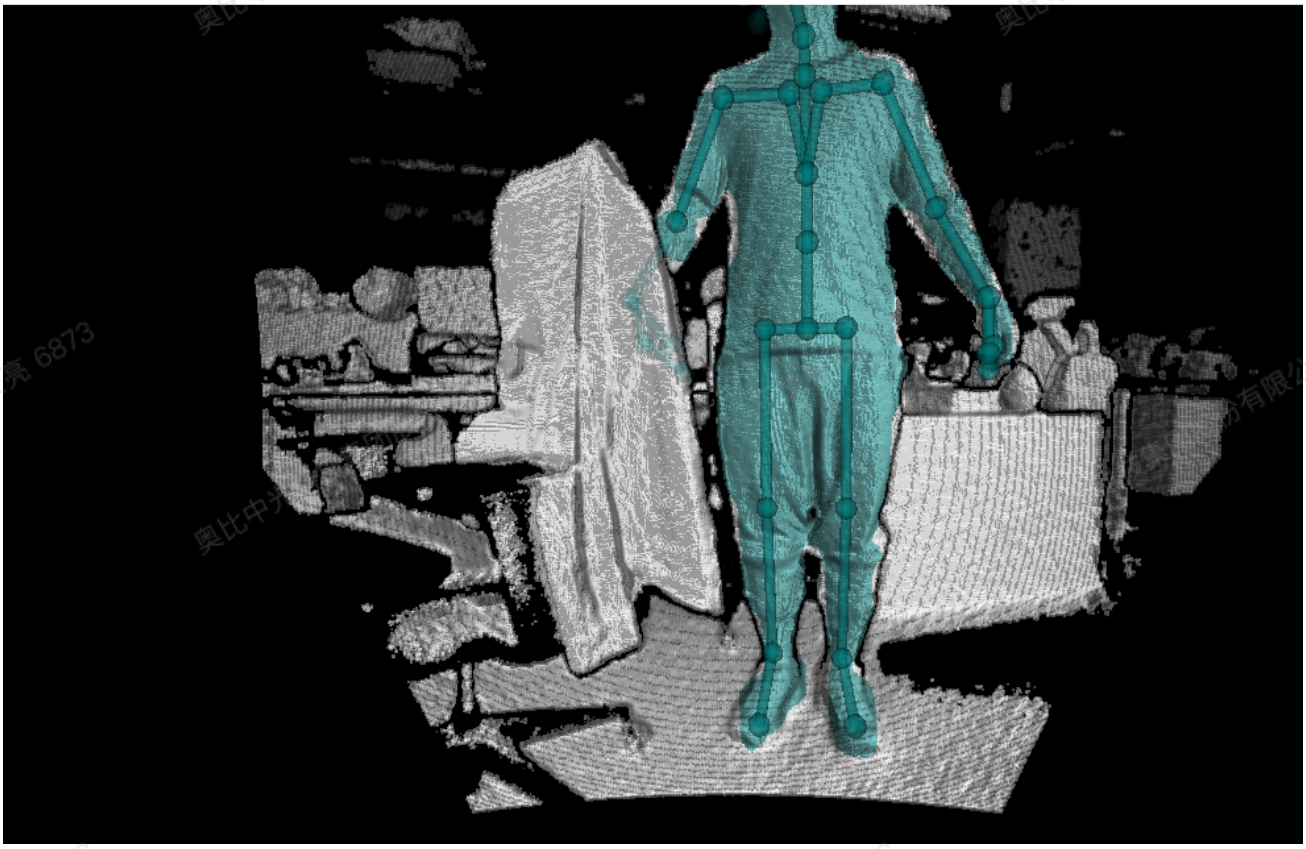
```
curl -sSL -O https://packages.microsoft.com/config/ubuntu/18.04/packages-microsoft-prod.deb
sudo dpkg -i packages-microsoft-prod.deb
rm packages-microsoft-prod.deb
sudo apt-get update
sudo apt install libk4abt1.1-dev
```

3.2.3 Use Orbbec SDK K4A Wrapper to Replace Azure Kinect Sensor SDK Library

Complete the environment configuration (udev rules script installation) according to Chapter 2, then replace the Azure Kinect Sensor SDK library files with the library files in the Orbbec SDK K4A Wrapper package (libk4a.so, libOrbbecSDK.so, depthengine2). After replacing and connecting the Orbbec camera, enter the simple_3d_viewer command in the terminal to see the following running effect:

```
[I20230919 09:17:42.261677 21304 FrameBufferManager.cpp:54] Frame buffer released=0.351989MB, total usage: {al
.3681MB, max limit=2048MB}
[I20230919 09:17:42.275681 21304 FrameBufferManager.hpp:58] FrameBufferManager created! @class libobsensor::Fr
anager<class libobsensor::ColorFrame>, obj addr:1961987970448, frame obj total size:0.351989MB
[I20230919 09:17:42.275681 21304 FrameBufferManager.cpp:123] ColorFrame bufferManager created!
[I20230919 09:17:42.275681 21304 FrameBufferManager.cpp:33] New frame buffer allocated=0.351989MB, total usage
ed=13.7201MB, max limit=2048MB}
[I20230919 09:17:42.288677 19828 FrameTimestampAdjuster.cpp:51] updateBaseTimeStamp:
[I20230919 09:17:42.288677 19828 FrameTimestampAdjuster.cpp:52]         srcTimestamp=0, prevSrcTsp_=0, tspDecr
```

3D Visualization



4. Differences between Orbbec SDK K4A Wrapper and Azure Kinect Sensor SDK

1. Functional points with differences

No.	Function Point	Orbbec SDK K4A Wrapper	Azure Kinect Sensor SDK	Impact on Application

1	Recording	`c++ typedef struct _k4a_record_configuration_t { /** * The timestamp offset of the start of the recording. All recorded timestamps are offset by this value such that * the recording starts at timestamp 0. This value can be used to synchronize timestamps between 2 recording files. */ uint64_t start_timestamp_offset_usec; } k4a_record_configuration_t;`	`c++ typedef struct _k4a_record_configuration_t { /** * The timestamp offset of the start of the recording. All recorded timestamps are offset by this value such that * the recording starts at timestamp 0. This value can be used to synchronize timestamps between 2 recording files. */ uint32_t start_timestamp_offset_usec; } k4a_record_configuration_t;`	Need to replace and recompile the header file of Orbbec SDK K4a Wrapper

2. Unimplemented interfaces in Orbbec SDK K4A Wrapper (return empty value or exception state)

No.	Azure Kinect Sensor SDK Interface Meaning	Impact of Differences

1	`c++ k4a_result_t k4a_set_allocator(k4a_memory_allocate_cb_t allocate, k4a_memory_destroy_cb_t free)` Pass in external user-defined memory manager for SDK internal memory application	User cannot use their custom memory manager for SDK internal use. Basic functioning of SDK itself is barely affected.
2	`c++ void k4a_capture_set_temperature_c(k4a_capture_t capture_handle, float temperature_c)` Set temperature information for capture	User cannot modify this value, impacting storage of custom values
3	`c++ float k4a_capture_get_temperature_c(k4a_capture_t capture_handle)` Get temperature information for capture	User cannot get this value, impacting algorithms or applications that rely on this value
4	`c++ void k4a_image_set_exposure_usec(k4a_image_t image_handle, uint64_t exposure_usec)` Set exposure value for image	User cannot modify this value, impacting storage of custom values

5	`c++ void k4a_image_set_white_balance(k4a_image_t image_handle, uint32_t white_balance)` Set white balance value for image	
6	`c++ void k4a_image_set_iso_speed(k4a_image_t image_handle, uint32_t iso_speed)` Set ISO speed for image	
7	`c++ uint64_t k4a_image_get_exposure_usec(k4a_image_t image_handle)` Get exposure value for image	User cannot get this value, impacting algorithms or applications that rely on this value
8	`c++ uint32_t k4a_image_get_white_balance(k4a_image_t image_handle)` Get white balance value for image	
9	`c++ uint32_t k4a_image_get_iso_speed(k4a_image_t image_handle)` Read ISO speed for image	

10	`c++ k4a_result_t k4a_device_get_sync_jack(k4a_device_t device_handle, bool *sync_in_jack_connected, bool *sync_out_jack_connected)` Get sync cable connection status for device	User application cannot rely on this interface to determine multi-camera sync cable linkage status