Users' Guide

1.0 Company and product introduction

1. Company and product introduction

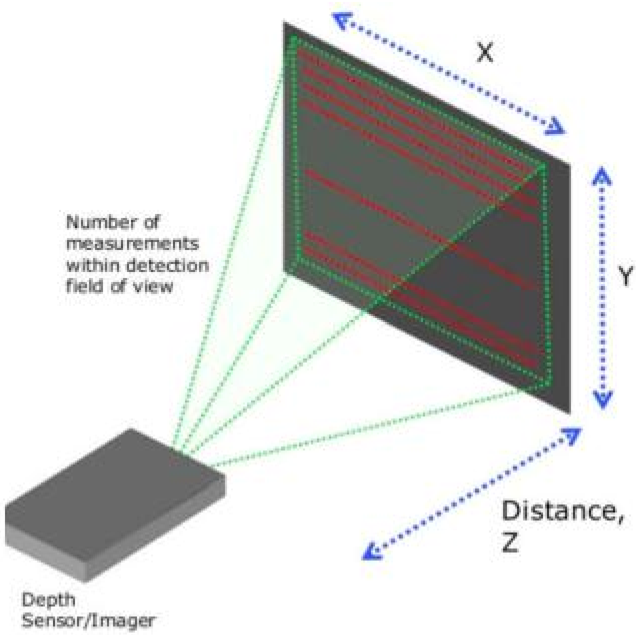
1.1 Introduction to Obi Zhongguang

Aobi Zhongguang is a 3D vision sensor technology solution provider integrating R&D, production and sales. It is the fourth company in the world that can mass produce 3D Sensors. It mainly provides hardware layer (physical layer) support for face recognition, gesture recognition, skeleton recognition, 3D measurement, environment perception, and 3D map reconstruction. For a complete set of solutions at the application layer, the company integrates the resources of partners, Provide support to customers.

ORBBEC®Astra series, P series, Deeyea series and Canglong depth cameras use structured light 3D imaging technology to obtain depth images of objects, while using color cameras to collect color images of objects. Among them, the Astra series, P series and Deeyea series are based on monocular structured light 3D imaging technology, and the Canglong series are based on binocular structured light 3D imaging technology. Support cross-platform development kit SDK.

1.1.1 3D Sensor

The 3D camera is characterized by not only being able to obtain a plane image, but also the depth information (Z axis) of the subject, that is, position and distance information. The 3D camera acquires the depth information, three-dimensional size and spatial information of environmental objects in real time, and provides basic technical support for "pain-type" application scenarios such as motion capture, three-dimensional modeling, VR/AR, indoor navigation and positioning.



What the 3D Sensor outputs is not a 3D model, but an RGB image plus a depth image.

1.1.2 3D and 2D visual contrast

***More feature information and higher recognition accuracy**

The 3D camera technology collects face image depth information during shooting, which can obtain more characteristic information and greatly improve the recognition accuracy on the basis of traditional face recognition technology. Compared with 2D face recognition systems, 3D face recognition can collect depth feature information such as the distance between the corners of the eyes, the tip of the nose, the alar point, the distance between the two temples, and the distance from the ear to the eye, and these parameters generally do not vary. As a person undergoes plastic surgery and changes his hairstyle, major changes occur, so 3D face recognition can continue to maintain a very high recognition accuracy rate when user characteristics change.

	3D	2D
FRA（错误接收率：系统接受错误对象概率）	0.047%	0.120%
FRR（错误拒绝率：系统拒绝真实对象概率）	0.103%	9.790%
姿态变化后识别率	100.00%	23.00%
头发遮挡后识别率	87.00%	50.00%
头部遮挡（帽子、头盔）后之别率	95.00%	<5.00%
弱光环境识别率	100%	0.00%

资料来源：新加坡出入境管理局、安信证券研究中心

***More accurate object segmentation and higher security**

2D images cannot segment objects correctly due to lack of depth information in complex scenes; 2D images can also be used for face recognition, but the 2D camera has not yet been used for face-swiping payment, because it is not safe enough. In fact, it means that flat information collection has a relatively large security risk and may be compromised by photos.

***Gesture recognition: a new way of interaction**

The key to gesture recognition lies in the 3D perception technology. The 3D camera acquires image depth information and recognizes user gestures through algorithm processing, so that users can control smart terminals in space, free users from touching the screen, and actively capture user gestures and actions. Recognition processing will become the next interactive pain point!

***Model reconstruction of 3D data**

The 3D camera can be used to create a "point cloud" of the surrounding environment. The point cloud data combined with the RBG information of the environment image can be used to restore the scene. After that, multiple applications such as distance measurement, virtual shopping, decoration, etc. can be derived on this basis, such as Virtual furniture placement, because the restored scene has in-depth information, the simulated furniture cannot continue to be pushed when it encounters an obstacle, which has a super sense of reality

1.1.3 Pros and cons of different 3D Sensing technical solutions

*** 3D structured light scheme**

As the name suggests, structured light is light with a special structure, such as discrete light spots, fringe light, and coded structured light. Projecting such a one-dimensional or two-dimensional image onto the object to be measured, based on the size and distortion of the image, can determine the surface shape of the object to be measured, that is, the depth information. For example, take a flashlight to illuminate a wall, stand close or stand far, the light spots on the wall are of different sizes, and the light spots will appear different ellipses when illuminating the wall from different angles. This is the basis of structured light. Because the energy density of the diffracted spot decreases at a certain distance, it is not suitable for long-distance depth information collection.

***TOF plan**

What TOF emits is not speckle, but a surface light source, so there will not be a lot of attenuation within a certain distance, and the depth accuracy will not decrease significantly as the distance increases. But the corresponding power consumption is higher, and the current depth image resolution is low.

***Binocular stereo vision**

Binocular stereo vision uses two or more cameras to collect images at the same time. By comparing the differences in the images obtained by these different cameras at the same time, algorithms are used to calculate depth information, thereby multi-angle three-dimensional imaging.

Program	Binocular	Structured light	TOF
Basic principles	Binocular matching Parallax algorithm	Laser speckle coding	Flight time (phase difference)
Laser light source	None (passive)	Speckle laser	Uniform surface laser
Power consumption	Low	medium	high
Image Resolution	Middle and high	medium	low
Precision	low	medium	Middle and high
Frame rate	low	medium	high
Low light performance	weak	good	good
Strong light performance	good	weak	medium
advantage	Mature technology, high resolution of plane information, low power consumption	Good anti-interference, long recognition distance	Strong light has good anti-interference and high power consumption
Highlight shortcomings	The algorithm has high complexity and poor real-time performance; it is not applicable in a dim	Under strong light conditions, it will be affected by light	Low depth image resolution;

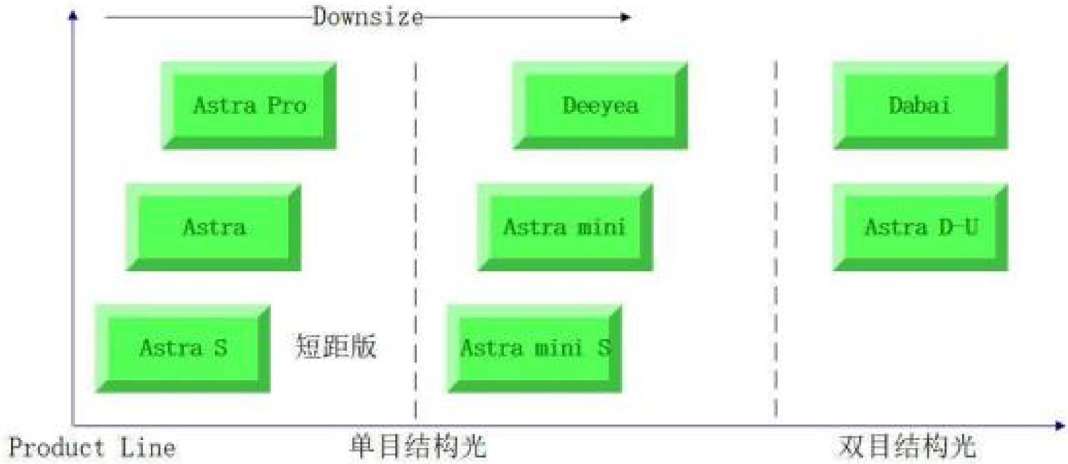
	environment; it cannot be used in scenes where the target feature texture changes are not obvious.		

To put it simply, the binocular has good outdoor performance, but it is not suitable for occasions where the texture changes are not obvious, such as white walls; TOF has better accuracy at long distances, but it is currently limited by low image resolution and high power consumption;

The comprehensive performance of structured light is currently the best among the three 3D sensing technologies, but the outdoor strong light interference is obvious and the accuracy is poor at long distances. Therefore, Aobi Zhongguang launched the second-generation product P1, Deeyea and the combination of binocular structured light in the 940nm band, Canglong No. 1, to improve the inherent defects of structured light as much as possible.

1.2 Orbbec product line and development resources

1.2.1 Orbbec Product Line



*Among the general standard products, Astra is the standard product, and the suffix mini means that the size is miniaturized; the suffix S is short, which means that the depth detection range is shorter; the suffix pro means that the RGB upgrade version supports higher resolution color images, and The color data stream is transmitted through the UVC protocol, which is different from other products.
*Deeyea is our latest generation single target product
*Astra D-U and Dabai are the latest binocular structured light products with the best outdoor performance

1.2.2 System requirements and development resources

系统要求 Android

- Android OS 4.4/ 5.1/6.0/7.1
- USB 2.0 或 3.0 （支持 host 接口）
- 建议 RK3288（四核 Cortex-A17，主频 1.8GHz）或以上
- 建议 2GB RAM 或以上
- 支持 LibUSB+LibUVC
- 支持 UVC 设备
- 支持 SELinux 权限访问 UVC 设备

系统要求 Windows

- Windows 7,10, 32-bit and 64-bit
- USB 2.0 或 3.0
- 双核，主频 2.2+GHz 或以上
- 建议 4GB RAM 或以上

Aobi Zhongguang provides customers with development code packages under Windows/Android/Linux/Unity/ROS systems and supports C++/Java language development. To obtain the SDK package, contact the staff.

Note:

The Astra SDK2.0 provided by Obi Zhongguang's Chinese and English official website, the SDK obtained through this channel is only for gesture and skeleton development and learning;

1.3 Orbbec product overview

1.3.1 Product naming rules

For the five products of Astra, Astra S, Astra mini, Astra mini S and Astra Pro, they are named with letters + serial numbers. The comparison of different letters and products is as follows.

product type	Logo letter
Astra	A
Astra S	B
Astra Pro	C
Astra mini	D
Astra mini S	E

By checking the device SN, you can intuitively judge the product type, chip generation and production date, as follows.

E1804300018					
E	18	4	30	3	0018
mini s	年	月	日	二代芯片	序列号
备注：	2代表一代芯片				

1.3.2 Product PID allocation table

	VID：0x2BC5			
No.	**model**	PID	UVC	**Remarks**
1	Astra	0x0401		**Monocular structured light**
2	Astra S	0x0402		**Monocular structured light**
3	Astra Pro	0x0403	0x0501	**Monocular structured light**
4	Astra mini	0x0404		**Monocular structured light**
5	Astra mini S	0x0407		**Monocular structured light**
6	Astra D-U	0x0608	0x0508	**Binocular structured light**
7	Deeyea	0x060b	0x050b	**Monocular structured light**
8	Astra Pro Plus	0x060d	0x050d	**Monocular structured light**
9	Dabai	0x060e	0x050e	**Binocular structured light**

Under Linux, you can check whether there is a corresponding device connected through lsusb.

1.3.3 Orbbec product parameters

Orbbec产品概览

项目	Astra	Astra S	Astra Pro	Astra mini	Astra mini S	Deeyae	Dabai	Astra D-U
Orbbec ASIC	✓	✓	✓	✓	✓	✓	✓	✓
独立ISP	×	×	✓	×	×	✓	✓	✓
接近传感器	HW	HW	HW	×	×	SW	×	×
外壳	✓	✓	✓	×	×	✓	×	×
麦克风	✓	✓	✓	×	×	×	×	×
深度范围（米）	0.6-8	0.4-2	0.6-8	0.6-8	0.4-2	0.25-1.5	0.3-3	0.3-3
Baseline	75mm	75mm	75mm	55mm	55mm	40mm	40mm	40mm
平均功耗（w）	<2.4	<2.4	<2.5	<2.4	<2.4	<2.5	1.6W	2.36W
彩色支持UVC	×	×	✓	×	×	✓	✓	✓
精度	±3mm@1m	±3mm@1m	±3mm@1m	±3mm@1m	±3mm@1m	±3mm@1m	TBD	±5mm@1m
深度分辨率	1280x1024@7fps 640x480@30fps 320x240@30fps 160x120@30fps	1280x1024@7fps 640x480@30fps 320x240@30fps 160x120@30fps	1280x1024@7fps 640x480@30fps 320x240@30fps 160x120@30fps	1280x1024@7fps 640x480@30fps 320x240@30fps 160x120@30fps	1280x1024@7fps 640x480@30fps 320x240@30fps 160x120@30fps	1280x800@30FPS 640x400@30FPS	640*400@30FPS 320*200@30FPS	640*400@30FPS 320*200@30FPS
深度FOV	H 58.4° V 45.5°	H 58.4° V 45.5°	H 58.4 V 45.5°	H 58.4° V 45.5°	H 58.4° V 45.5°	H67.9° V45.3°	H67.9° V45.3°	H67.9° V45.3°
彩色分辨率	1280x960@7fps 640x480@30fps	1280x960@7fps 640x480@30fps	1280x720@30fps 640x480@30fps 320x240@30fps	1280x960@7fps 640x480@30fps 320x240@30fps	1280x960@7fps 640x480@30fps 320x240@30fps	1920x1080@30FPS 1280x720@30FPS 640x480@30FPS	1920*1080@30FPS 1280*720@30FPS 640*480@30FPS	1920*1080@30FPS 1280*720@30FPS 640*480@30FPS
彩色FOV	H 63.1° V 49.4°	H 63.1° V 49.4°	H 66.1 V 40.2°	H 63.1° V 49.4°	H 63.1° V 49.4°	H71.5° V56.7°	H71° V43.7°	H59.9° V46.5°
接口方式	标准USB2.0	标准USB2.0	标准USB2.0	标准USB2.0	标准USB2.0	USB3.0 Type-C	USB 物理接口非标	USB 物理接口非标
结构尺寸（mm）	164.85*30*48.25	164.85*30*48.25	164.85*30*48.25	80*20*19.72	80*20*19.72	68.6*22.3*14.8	59.6* 17.4* 11.1	59.6* 17.4* 10.4
适用场景	室内	室内	室内	室内	室内	室内/室外 （50000Lux@1m）	室内/室外 （50000Lux@1m）	室内/室外 （50000Lux@1m）
备注	单目结构光	单目结构光	单目结构光	单目结构光	单目结构光	单目结构光	双目结构光	双目结构光

1.4 Comparison of Sensor Types of Orbbec Products

<table>
<tr><td>应用产品</td><td colspan="2">Astra/Astra S/Astra mini/Astra mini S</td></tr>
<tr><td>Sensor</td><td>MT9M001</td><td>MT9M114</td></tr>
<tr><td>Spec</td><td>
<table>
<tr><th>Parameter</th><th colspan="2">Typical Value</th></tr>
<tr><td colspan="2">Optical format</td><td>1/2-inch (5:4)</td></tr>
<tr><td colspan="2">Active imager size</td><td>6.66mm(H) x 5.32mm(V)</td></tr>
<tr><td colspan="2">Active pixels</td><td>1,280H x 1,024V</td></tr>
<tr><td colspan="2">Pixel size</td><td>5.2μm x 5.2μm</td></tr>
<tr><td colspan="2">Shutter type</td><td>Electronic rolling shutter (ERS)</td></tr>
<tr><td colspan="2">Maximum data rate/
master clock</td><td>48 MPS/48 MHz</td></tr>
<tr><td>Frame rate</td><td>SXGA
(1280 x 1024)</td><td>30 fps progressive scan;
programmable</td></tr>
<tr><td colspan="2">ADC resolution</td><td>10-bit, on-chip</td></tr>
<tr><td colspan="2">Responsivity</td><td>2.1 V/lux-sec</td></tr>
<tr><td colspan="2">Dynamic range</td><td>68.2dB</td></tr>
<tr><td colspan="2">SNR_{MAX}</td><td>45dB</td></tr>
<tr><td colspan="2">Supply voltage</td><td>3.0V–3.6V, 3.3V nominal</td></tr>
<tr><td colspan="2">Power consumption</td><td>325mW at 3.3V;
Standby 275μW</td></tr>
<tr><td colspan="2">Operating temperature</td><td>0°C to +70°C</td></tr>
<tr><td colspan="2">Packaging</td><td>48-pin CLCC</td></tr>
</table>
</td><td>
<table>
<tr><th>Parameter</th><th>Typical Value</th></tr>
<tr><td>Optical Format</td><td>1/6-inch</td></tr>
<tr><td>Active Pixels</td><td>1296 (H) × 976 (V) = 1.26 Mp</td></tr>
<tr><td>Pixel Size</td><td>1.9 μm × 1.9 μm</td></tr>
<tr><td>Color Filter Array</td><td>RGB Bayer</td></tr>
<tr><td>Shutter</td><td>Electronic Rolling Shutter (ERS)</td></tr>
<tr><td>Input Clock Range</td><td>6–54 MHz</td></tr>
<tr><td>Output MIPI Data Rate Maximum</td><td>768 Mb/s</td></tr>
<tr><td>Max. Frame Rate</td><td>30 fps Full Res
36.7 fps 720p
75 fps VGA
120 fps QVGA (Note 2)</td></tr>
<tr><td>Responsivity</td><td>2.24 V/Lux–sec (550 nm)</td></tr>
<tr><td>SNR_{MAX}</td><td>37 dB</td></tr>
<tr><td>Dynamic Range</td><td>70.8 dB</td></tr>
<tr><td>Supply Voltage
Digital
Analog
I/O
PLL
PHY</td><td>
1.7–1.95 V
2.5–3.1 V
1.7–1.95 V or 2.5–3.1 V
2.5–3.1 V
1.7–1.95 V</td></tr>
<tr><td>Power Consumption</td><td>135 mW (Note 1)</td></tr>
<tr><td>Operating Temperature Range (Ambient) – T_A</td><td>–30°C to 70°C</td></tr>
<tr><td>Chief Ray Angle</td><td>27.7°</td></tr>
<tr><td>Active Imager Size</td><td>2.46 mm (H) × 1.85 mm (V),
3.08 mm Diagonal</td></tr>
<tr><td>Package Options</td><td>Bare Die, CSP</td></tr>
</table>
</td></tr>
<tr><td>制造商</td><td>ON semiconductor</td><td>ON semiconductor</td></tr>
<tr><td>Shutter</td><td>Rolling shutter</td><td>Rolling shutter</td></tr>
<tr><td>Pixels</td><td>1-Megapixel</td><td>1.26-Megapixel</td></tr>
</table>

<table>
<tr><td>应用产品</td><td colspan="2">Astra Pro</td></tr>
<tr><td>Sensor</td><td>MT9M001</td><td>OV9712</td></tr>
<tr><td>Spec</td><td>
<table>
<tr><th>Parameter</th><th colspan="2">Typical Value</th></tr>
<tr><td colspan="2">Optical format</td><td>1/2-inch (5:4)</td></tr>
<tr><td colspan="2">Active imager size</td><td>6.66mm(H) x 5.32mm(V)</td></tr>
<tr><td colspan="2">Active pixels</td><td>1,280H x 1,024V</td></tr>
<tr><td colspan="2">Pixel size</td><td>5.2μm x 5.2μm</td></tr>
<tr><td colspan="2">Shutter type</td><td>Electronic rolling shutter (ERS)</td></tr>
<tr><td colspan="2">Maximum data rate/
master clock</td><td>48 MPS/48 MHz</td></tr>
<tr><td>Frame rate</td><td>SXGA
(1280 x 1024)</td><td>30 fps progressive scan;
programmable</td></tr>
<tr><td colspan="2">ADC resolution</td><td>10-bit, on-chip</td></tr>
<tr><td colspan="2">Responsivity</td><td>2.1 V/lux-sec</td></tr>
<tr><td colspan="2">Dynamic range</td><td>68.2dB</td></tr>
<tr><td colspan="2">SNR_{MAX}</td><td>45dB</td></tr>
<tr><td colspan="2">Supply voltage</td><td>3.0V–3.6V, 3.3V nominal</td></tr>
<tr><td colspan="2">Power consumption</td><td>325mW at 3.3V;
Standby 275μW</td></tr>
<tr><td colspan="2">Operating temperature</td><td>0°C to +70°C</td></tr>
<tr><td colspan="2">Packaging</td><td>48-pin CLCC</td></tr>
</table>
</td><td>
■ active array size: 1280 x 800
■ power supply:
- core: 1.5 VDC ±5%
- analog: 3.0 - 3.6 V
- I/O: 1.7 - 3.6 V
■ power requirements:
- active: 110 mW
- standby: 50 μA
■ temperature range:
- operating: -30°C to 70°C
- stable image: 0°C to 50°C
■ output formats: 10-bit RAW RGB data
■ lens size: 1/4"
■ lens chief ray angle: 25° non-linear
■ input clock frequency: 6 - 27 MHz
■ dynamic range: 69 dB
■ maximum image transfer rate:
- WXGA (1280x800): 30 fps
- HD 720p (1280x720): 30 fps
- VGA (640x480): 60 fps
■ sensitivity: 3300 mV/(lux-sec)
■ S/N ratio: 39 dB
■ maximum exposure interval: 826 x t_{ROW}
■ pixel size: 3 μm x 3 μm
■ well capacity: 13 Ke$^-$
■ dark current: 20 mV/s @ 60°C
■ image area: 3888 μm x 2430 μm
■ package/die dimensions:
- CSP2: 5415 μm x 4415 μm
- COB: 5430 μm x 4430 μm
</td></tr>
<tr><td>制造商</td><td>ON semiconductor</td><td>OmniVision</td></tr>
<tr><td>Shutter</td><td>Rolling shutter</td><td>Rolling shutter</td></tr>
<tr><td>Spec</td><td>1-MegaPixel</td><td>1-Megapixel</td></tr>
</table>

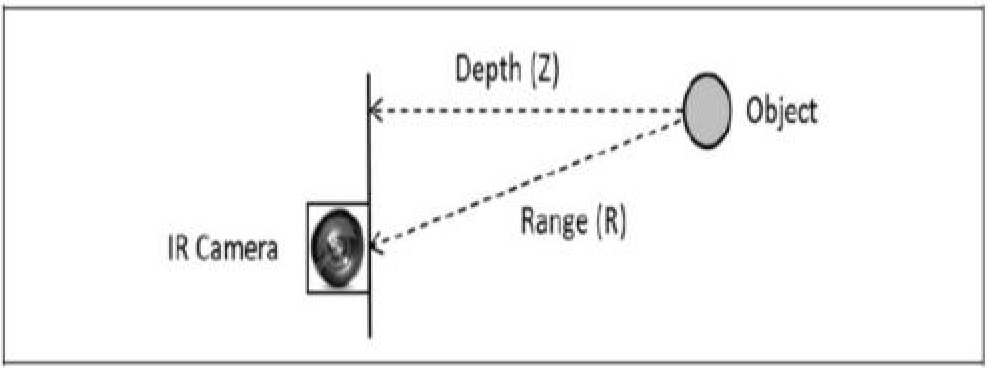
<table>
<tr><td>应用产品</td><td colspan="2">Deeyea</td></tr>
<tr><td>Sensor</td><td>OV9282</td><td>OV5695</td></tr>
<tr><td>Spec</td><td>■ active array size: 1296 x 816
■ power supply:
- core: 1.2V (nominal)
- analog: 2.8V (nominal)
- I/O: 1.8V (nominal)
■ power requirements:
- active: 156 mW
- standby: 150 μA
- XSHUTDOWN: 150 μA
■ temperature range:
- operating: -30°C to +85°C junction temperature
- stable image: 0°C to +50°C junction temperature
■ output interfaces:
2-lane MIPI serial output and DVP parallel output
■ output formats: 8/10-bit RAW
■ lens size: 1/4"
■ input clock frequency: 6 - 27 MHz
■ lens chief ray angle:
- OV9281: 9° linear
- OV9282: 26.78° non-linear
■ max S/N ratio: 38 dB
■ dynamic range: 68 dB
■ maximum image transfer rate:
- 1280 x 800: 120 fps
■ sensitivity:
13000 mV/μW.cm⁻².sec) @ 850 nm
6500 mV/μW.cm⁻².sec) @ 940 nm
■ scan mode: progressive
■ minimum exposure time: 1 row period
■ maximum exposure time:
frame length - 12 row periods, where frame length is set by registers {0x380E, 0x380F}
■ pixel size: 3 μm x 3 μm
■ dark current: 80e⁻/sec @ 50°C junction temperature
■ image area: 3896 μm x 2453 μm
■ package dimensions:
- OV9281 CSP: 5237 μm x 4463 μm
- OV9282 RW: 5252 μm x 4478 μm</td><td>■ active array size: 2592 x 1944
■ power supply:
- core: 1.14 - 1.26V (1.2V nominal)
- analog: 2.7 - 3.0V (2.8V nominal)
- I/O: 1.7 - 1.9V (1.8V nominal)
■ power requirements:
- active: 155 mW
- standby: 61 μW
- XSHUTDN: 36 μW
■ temperature range:
- operating: -30°C to +70°C junction temperature
- stable image: -20°C to +60°C junction temperature
■ output interface:
2-lane MIPI serial output
■ output formats: 10-bit RGB RAW
■ lens size: 1/4"
■ lens chief ray angle: 31.08° non-linear
■ input clock frequency: 6 - 27 MHz
■ maximum image transfer rate:
- 5MP (2592x1944): 30 fps
- quad HD (2560x1440): 30 fps
- 1080p (1920x1080): 60 fps
- 720p (1280x720): 60 fps
- VGA (640x480): 120 fps
■ pixel size: 1.4 μm x 1.4 μm
■ dark current: 15 e⁻/sec @ 60°C junction temperature
■ image area: 3684 μm x 2763 μm
■ dimensions:
- COB: 5022 μm x 3933 μm
- RW: 5072 μm x 3983 μm</td></tr>
<tr><td>制造商</td><td>OmniVision</td><td>Omni Vision</td></tr>
<tr><td>Shutter</td><td>Global shutter</td><td>Rolling shutter</td></tr>
<tr><td>Pixel</td><td>1-Megapixel</td><td>5-Megapixel</td></tr>
</table>

<table>
<tr><td>应用产品</td><td colspan="2">Astra D-U</td></tr>
<tr><td>Sensor</td><td>OV9282</td><td>AR0330</td></tr>
<tr><td>Spec</td><td>■ active array size: 1296 x 816
■ power supply:
- core: 1.2V (nominal)
- analog: 2.8V (nominal)
- I/O: 1.8V (nominal)
■ power requirements:
- active: 156 mW
- standby: 150 μA
- XSHUTDOWN: 150 μA
■ temperature range:
- operating: -30°C to +85°C junction temperature
- stable image: 0°C to +50°C junction temperature
■ output interfaces:
2-lane MIPI serial output and DVP parallel output
■ output formats: 8/10-bit RAW
■ lens size: 1/4"
■ input clock frequency: 6 - 27 MHz
■ lens chief ray angle:
- OV9281: 9° linear
- OV9282: 26.78° non-linear
■ max S/N ratio: 38 dB
■ dynamic range: 68 dB
■ maximum image transfer rate:
- 1280 x 800: 120 fps
■ sensitivity:
13000 mV/μW.cm⁻².sec) @ 850 nm
6500 mV/μW.cm⁻².sec) @ 940 nm
■ scan mode: progressive
■ minimum exposure time: 1 row period
■ maximum exposure time:
frame length - 12 row periods, where frame length is set by registers {0x380E, 0x380F}
■ pixel size: 3 μm x 3 μm
■ dark current: 80e⁻/sec @ 50°C junction temperature
■ image area: 3896 μm x 2453 μm
■ package dimensions:
- OV9281 CSP: 5237 μm x 4463 μm
- OV9282 RW: 5252 μm x 4478 μm</td><td><table>
<tr><th>Parameter</th><th>Typical Value</th></tr>
<tr><td>Optical Format</td><td>1/3-inch (6.0 mm)
Entire Array: 6.09 mm
Still Image: 5.63 mm (4:3)
HD Image: 5.82 mm (16:9)</td></tr>
<tr><td>Active Pixels</td><td>2304 (H) × 1536 (V): (Entire Array)
5.07mm (H) × 3.38mm (V)
2048 (H) × 1536 (V) (4:3, Still Mode)
2304 (H) × 1296 (V) (16:9, HD Mode)</td></tr>
<tr><td>Pixel Size</td><td>2.2 × 2.2 μm</td></tr>
<tr><td>Color Filter Array</td><td>RGB Bayer</td></tr>
<tr><td>Shutter Type</td><td>ERS and GRR</td></tr>
<tr><td>Input Clock Range</td><td>6–27 MHz</td></tr>
<tr><td>Output Clock Maximum</td><td>196 Mp/s (4-lane HiSPi or MIPI)</td></tr>
<tr><td>Output Video – 4-lane HiSPi</td><td>2304 × 1296 at 60 fps
< 450 mW (V_{CM} 0.2 V, 196 MP/s)
2304 × 1296 at 30 fps
< 300 mW (V_{CM} 0.2 V, 98 MP/s)</td></tr>
<tr><td>Responsivity</td><td>2.0 V/lux–sec</td></tr>
<tr><td>SNR_{MAX}</td><td>39 dB</td></tr>
<tr><td>Dynamic Range</td><td>69.5 dB</td></tr>
</table></td></tr>
<tr><td>制造商</td><td>OmniVision</td><td>ON semiconductor</td></tr>
<tr><td>Shutter</td><td>Global shutter</td><td>ERS(electronic rolling shutter)
GRR(global reset release)</td></tr>
<tr><td>Pixel</td><td>1-Megapixel</td><td>3.15-Megapixel</td></tr>
</table>

1.5 3D Sensor common term explanation

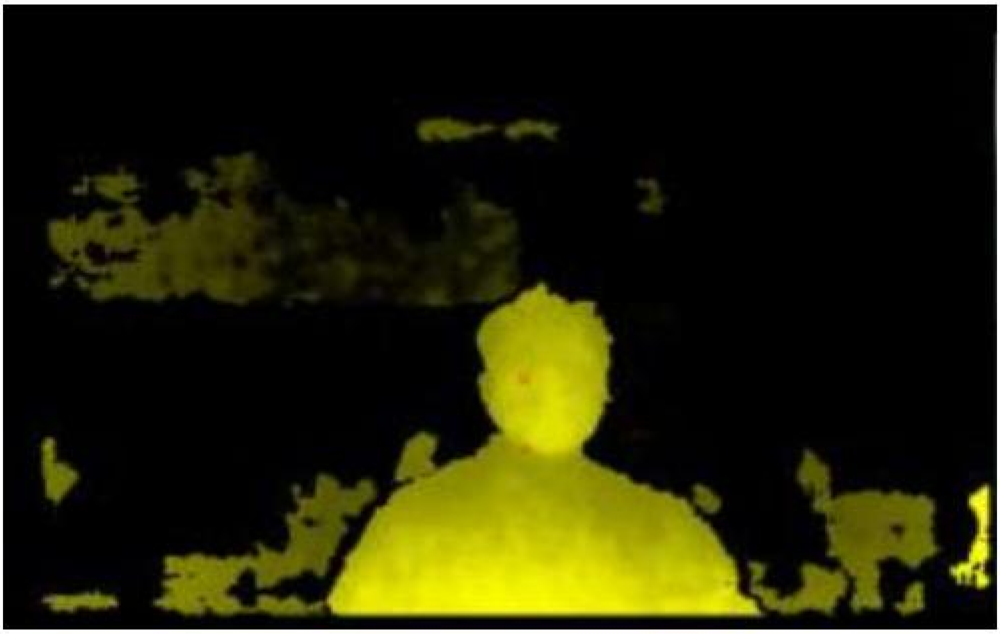
1.5.1 Depth

Compared with traditional 2D cameras, 3D cameras have added depth information. The so-called depth value Depth refers to the vertical distance from the measured object to the camera plane, that is, the Z value in the figure below, not R.



1.5.2 Depth Map

Different from RGB images, each pixel in the depth map saves the depth value data of objects in the field of view from the camera plane. The depth raw data is usually a 16-bit unsigned int type, and the unit can be specified through the SDK, usually 1mm, that is, each pixel in the depth map stores 16-bit unsigned integer data, the unit is 1mm. In order to display the depth data visually, it is usually converted into a grayscale display, as shown in the figure below, different gray levels represent different depth values.、



1.5.3 Commonly used terms

the term	description
Baseline	**1) Monocular structured light: the distance between the imaging center of the infrared camera and the projection center of the infrared projector** **2) Binocular structured light: the distance between the imaging centers of the left and right infrared cameras**
Depth	**The depth video stream is just like the color video stream, except that each pixel has a value representing the distance from the camera instead of color information**
FOV	**Field of view, used to describe the angle range of the camera imaging a given scene, there are three main types: horizontal field of view (H FOV), vertical field of view (V FOV) and diagonal field of view (D FOV)**
Depth processor	**Depth calculation processor, dedicated ASIC chip used to implement depth calculation algorithm and output depth image, such as MX400, MX6000**
IR camera	**Infrared camera, or infrared camera**
RGB camera	**Color camera, or color camera**
LDMP/LDM	**IR projector, also called infrared laser projector, structured light projector, etc., used to emit structured light patterns**
Depth camera	**Depth camera, including depth imaging module and color imaging module. The depth imaging module is generally composed of an infrared projector, an infrared camera and a depth cloud computing processor. The color imaging module generally refers to a color camera**

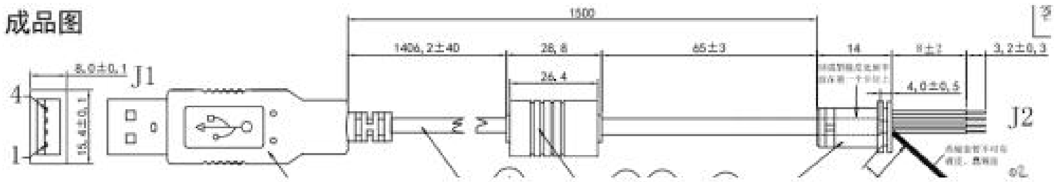
2.0 Product integrated design

2. Product integration design

2.1 Hardware integration design

2.1.1 Product physical interface

*All Obi Zhongguang products are USB interfaces, and most of them are standard Type-A interfaces.



*Deeyea is a Type-C interface, and the detailed interface definition can be found in the specification.

*P1 is a micro-USB interface, and the 20PIN additional interface is customized, which can be open to common users and does not provide wires.

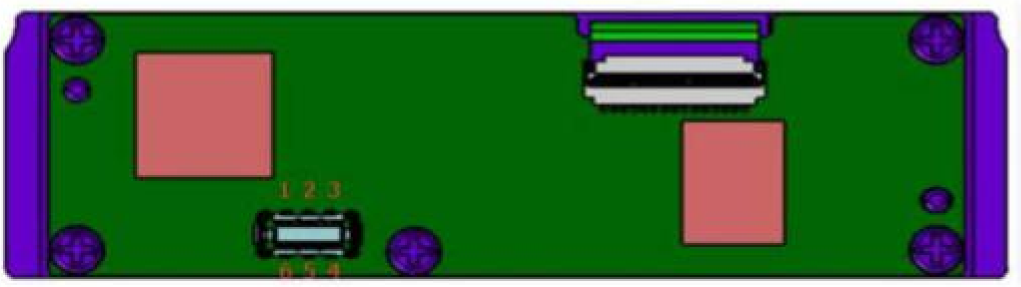
PIN	Name	Wire's color	Description
1	VBUS	Red	Power positive 5V
2	D-	White	USB cable negative
3	D+	Green	USB cable positive
4	ID	none	Divided into two interfaces: A and B A: Connect to ground B: Not connected to ground
5	GND	Black	Power negative

The function of the MIcro USB corresponding interface

Pin	名称
1	VBUS
2	VBUS
3	VBUS
4	GND
5	GND
6	D+
7	D-
8	GND
9	GND
10	GND

The function of the 20pin connector corresponding interface

*The physical interface of Astra D-U is a 6PIN B2B connector, model CPB9506-0113E, which transmits data via USB protocol



POSITION	NAME
1	VBUS
2	D+
3	D-
4	GND
5	ID
6	GND

2.2 Structural integration design

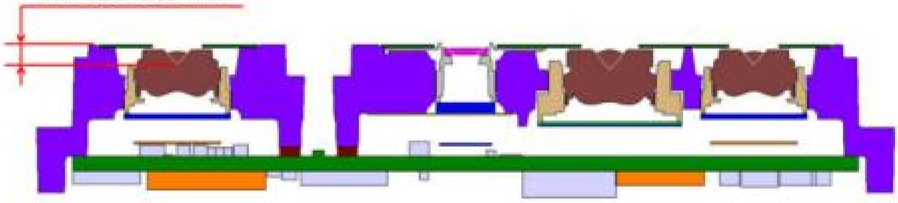
2.2.1 Structural seal design reference

For details, see the description of the structural seal design in the product datasheet.

2.2.2 Absolute distance reference point

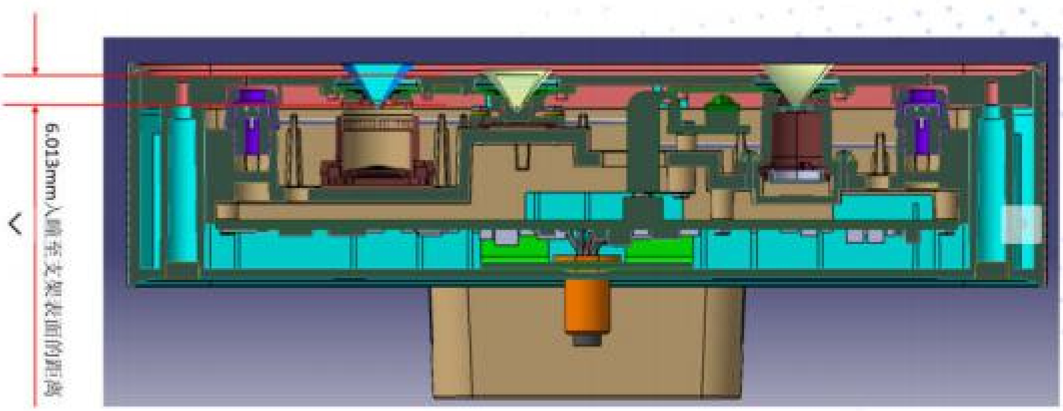
1) Astra D-U

The distance between the entrance pupil of Astra D-U and the surface of the stent is 0.6mm. When testing the absolute distance, the outer surface of the stent is often used as a reference.



2) Astra S

The distance from the entrance pupil of Astra S to the inner surface of the housing is 6.013mm. When testing the absolute distance, the housing surface is often used as a reference.



2.2.1 Thermal Design Reference

The company's miniaturized products (Astra mini, Deeyea, Astra D-U, etc.) need to consider heat dissipation when embedded in the design. For details, see the heat dissipation design reference document.

2.3 Driver installation and system adaptation

2.3.1 Windows system driver installation and diagnosis

1) System requirements

Windows 7, 8 and 10 (x86, x86-64); x86-based processor @ 1.8+ GHz

2) Drive import

Before using the Windows system for testing, you need to install the driver. The driver file is in the SDK/drivers directory. After the driver is installed correctly, you can view the device through the device manager. If you encounter an exception, please refer to the "Astra Driver Installation and Device Diagnosis Guide" in the SDK to handle the driver exception.

2.3.2 Android system adaptation

1) System requirements

The Android system does not need to install additional product drivers.

- Android OS 4.4/ 5.1/6.0/7.1
- USB 2.0 或 3.0 （支持 host 接口）
- 建议 RK3288（四核 Cortex-A17，主频 1.8GHz）或以上
- 建议 2GB RAM 或以上
- 支持 LibUSB+LibUVC
- 支持 UVC 设备
- 支持 SELinux 权限访问 UVC 设备

2) Test method

Install the NiViewer.apk in the SDK and apply it to the corresponding device, you can view the depth map and color map of the device.

The following points need to be noted:

*Orbbec devices are divided into UVC and non-UVC devices. For UVC devices, such as Deeyea, Canglong, etc., color images cannot be viewed through NiViewer_for_android.apk; UVC is a common USB video capture device protocol standard, and Android system cameras can be used directly Open to view color images, you can use libuvc or Android system camera during development.

* Different Orbbec devices have different resolutions. Use NiViewer_for_android.apk to select the corresponding resolution. For example, Deeyea, P1, etc. need to select 640*400 resolution, while Orbbec Pro A needs to select 640*480, otherwise the depth map preview is not normal.

3) Exception handling

***Android devices cannot recognize Camera**

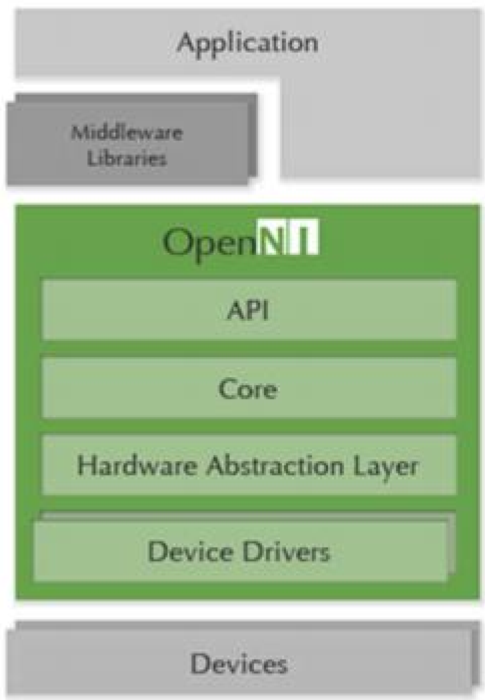
Common problems that are not recognized by Android devices can be solved through the following steps:

a) Insert the device into Windows, install the driver SensorDriver, and check whether the device is normally loaded in the device manager. If it fails to load normally, install "Driver Installation and Diagnosis Guide" to troubleshoot the driver;

b) Use NiViewer.exe on the Windows side to test whether the device can output pictures normally and troubleshoot the device
c) If it is normal, reconnect the Android device, connect adb, and use dumpsys USB to check whether the Orbbec device node can be loaded normally. The Vendor ID of the Orbbec device is 2bc5.
d) If it fails to load normally, it is usually judged to be a USB interface hardware circuit or power supply problem on the device side. Orbbec devices generally need 5V 500mA power supply, and the instantaneous current peak value can reach 700mA.

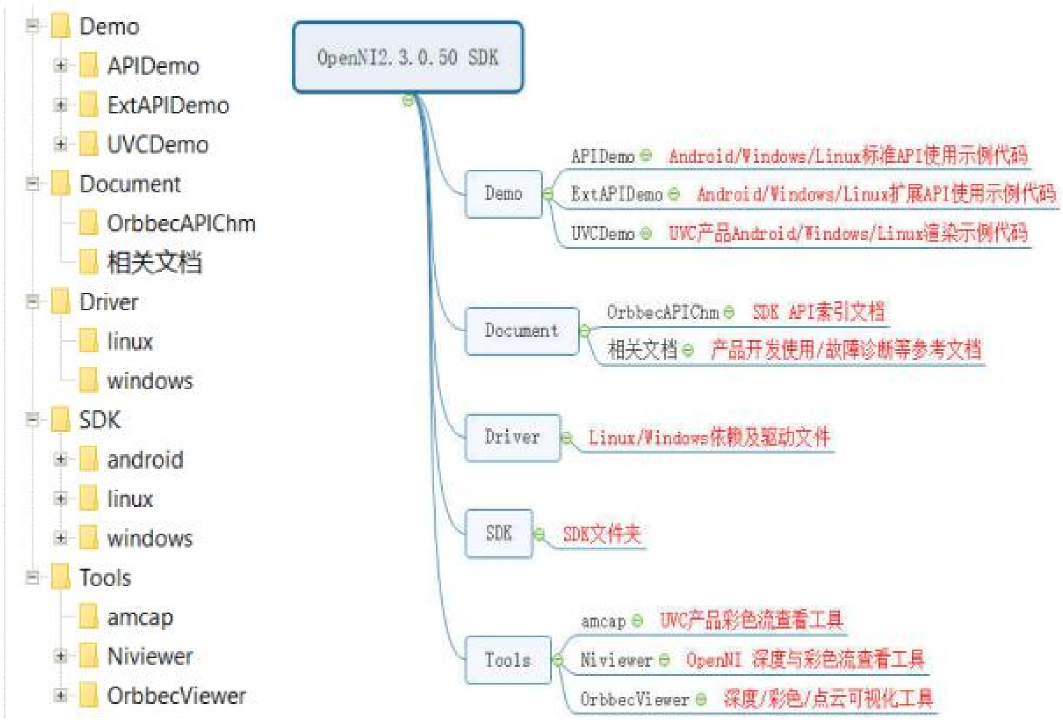
e) If the device node is found, it can basically be judged as a system compatibility problem, and confirm whether the system supports USB host. For systems above Android 6.0, you need to confirm that the Selinux permission is closed, and use getenforce to check whether the status is Permissive; if not, use setenforce 0 to close it under administrator permissions.
f) If you still cannot use the device normally, you can open the NiViewer application and grab the logfile for specific analysis.

3.0 SDK introduction and use
3. SDK introduction and use
3.1 Introduction to SDK
The OpenNI2.3 series SDK provided by Obi Zhongguang is developed based on OpenNI2. OpenNI2 (Open Natural Interaction) is a multi-language, cross-platform framework that defines the interface between applications, middleware and 3D sensing devices. Aobi Zhongguang's full range of products fully support the OpenNI protocol, which means that any application developed using OpenNI can be seamlessly connected to the company's products.
When using OpenNI2.3.0.50SDK, it is assumed that the user has a certain understanding of OpenNI. If you have not contacted OpenNI related development before, you can refer to the official OpenNI2 help document. The overall framework of OpenNI2 is shown below:



*The top layer is about OpenNI2 applications such as NITE gesture recognition, body motion detection, etc.
* Next is the unified interface provided by OpenNI2. The header file corresponding to these interfaces is OpenNI.h
* OpenNI Core is the core part of OpenNI2. The structure implementation in OpenNI.h is in this part; a unified API is provided for the Driver layer for the development and extension of the Driver. The header file corresponding to this part of the API is OniDriverAPI.h
* The bottom layer is hardware drivers or third-party libraries

3.2 OpenNI2.3.0.50 SDK directory structure



3.2 SDK project environment and configuration

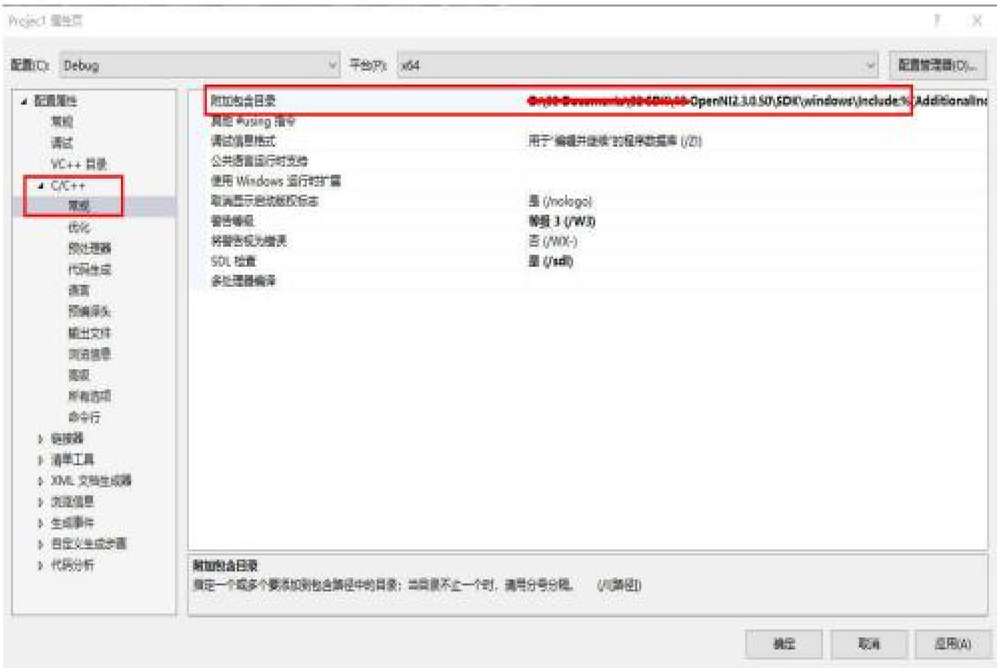
3.2.1 Windows engineering environment configuration

(1) Scope of application

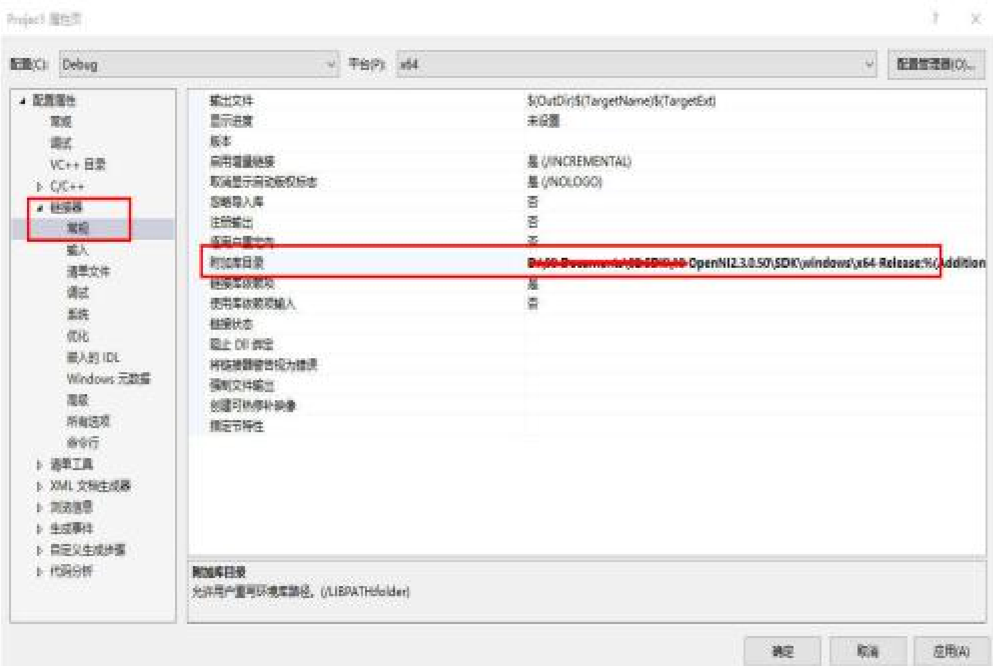
The company's SDK is suitable for X86/X64 Windows7 and above platforms

(2) Development platform construction

The recommended Windows development platform is Visual Studio 2013 or above. For the Visual Studio platform, you can build a development environment through the following steps:
* Create or open a C++ project
* Select "Project-Properties", add the \SDK\windows\Include directory to the header file dependency path, and note that the property configuration (x86/x64) is consistent with the runtime

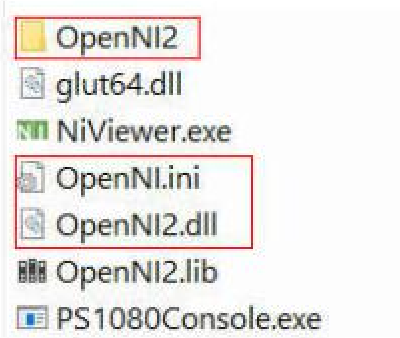


*Select the "Linker-General" project and add the directory where the OpenNI.lib file corresponding to x86 or x64 is located,



*Add the OpenNI.lib item in the "linker-input" directory, complete the environment configuration, and start editing your own code.

* Note that when compiling and running the project, copy the OpenNI folder, ini and dll files in the x64-release or x86-release folder to the directory where the exe file is located, otherwise it will crash or report an error



(3) SimpleCode description

Under Windows system, different sample codes are provided for applications such as standard API and extended API in SDK. For specific operations and instructions, please check the readme file in the corresponding folder

Steps to get the depth stream:

(1) Initialization of OpenNI, this method must be called, and then Videostream, Device, etc. can be used. Status rc = OpenNI::initialize();

(2) Open a device, where ANY_DEVICE is the first device connected in the open system Device device;rc = device.open(ANY_DEVICE);

(3) After turning on the device, obtain the sensor information, and then create a video stream based on the sensor information

if (device.getSensorInfo(SENSOR_DEPTH) != NULL)

(4) Create a depth video stream rc = depth.create(device, SENSOR_DEPTH);

(5) Call the start method to start collecting depth information rc = depth.start();

(6) Use polling method to obtain code stream information

VideoFrameRef frame;

VideoStream* pStream = &depth;

rc = OpenNI::waitForAnyStream(&pStream, 1, &changedStreamDummy, SAMPLE_READ_WAIT_TIMEOUT);

(7) Read one frame of data

rc = depth.readFrame(&frame);

(8) Obtain the depth value of the center point of the depth image

DepthPixel* pDepth = (DepthPixel*)frame.getData();

float x,y,z;

CoordinateConverter coorvert;

int middleIndex = (frame.getHeight()+1)*frame.getWidth()/2;

coorvert.convertDepthToWorld(depth, frame.getWidth()/4, frame.getHeight()/4, pDepth[middleIndex], &x, &y, &z);

(9) Turn off related equipment after stopping video capture

depth.stop();

depth.destroy();

device.close();

OpenNI::shutdown();

3.2.2 Linux environment configuration

(1) Scope of application

Under the Linux platform, our company provides SDK packages for Intel x86/x64 and ARM32/64. When using, please pay attention to platform consistency. For the ARM platform, depending on the processing capabilities, two SDK packages are provided, the normal version and the without-filter (no software filtering) version. If the normal version freezes, the SDK without software filtering can be used.

(2) Linux environment configuration instructions

Before using OpenNI2 SDK for related development on the Linux platform, please execute the following commands to install and configure the related environment. Take the x64 platform as an example, in order to run related programs, you first need to install the following programs:

```
$sudo apt-get install cmake
$sudo apt-get install libusb-1.0-0-dev
$sudo apt-get install libudev-dev
$sudo apt-get install libglfw3-dev
$sudo apt-get install build-essential freeglut3 freeglut3-dev
```

Check the udev version, the Orbbec driver needs to rely on libudev.so.1, you can execute the following command to find, if you don't find it, you need to manually link libudev.so.x.x to libudev.so.1

```
$ldconfig -p | grep libudev.so.1
$cd /lib/x86_64-linux-gnu
$sudo ln -s libudev.so.x.x.x libudev.so.1
```

Copy the SDK package to a certain path, decompress it, and compile the routine.

```
$cd Demo/APIDemo/example/build ----- Common code examples
$cmake ..
$make
$cd Demo/ExtAPIDemo/linux-x64/ExtendedAPI ---Extended Api code example
$make
$cd Demo/UVCDemo/Linux/OrbbecStreamSample-Linux ---uvc code example
```

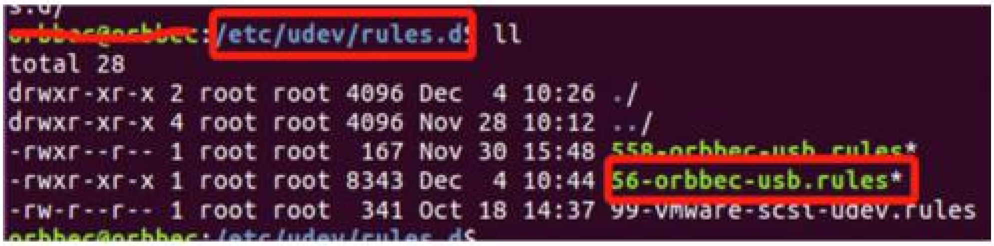
Plug in the camera to run the routine program.

```
$lsusb ---Confirm whether the device is recognized
$./DepthRead --- Run the corresponding routine
```

(3) udev description

In Linux system, in order to ensure that the camera device can be recognized and opened normally, the corresponding udev should be added. The specific operations are as follows:

```
$cd Driver/linux
$ ./ create_udev_rules
```

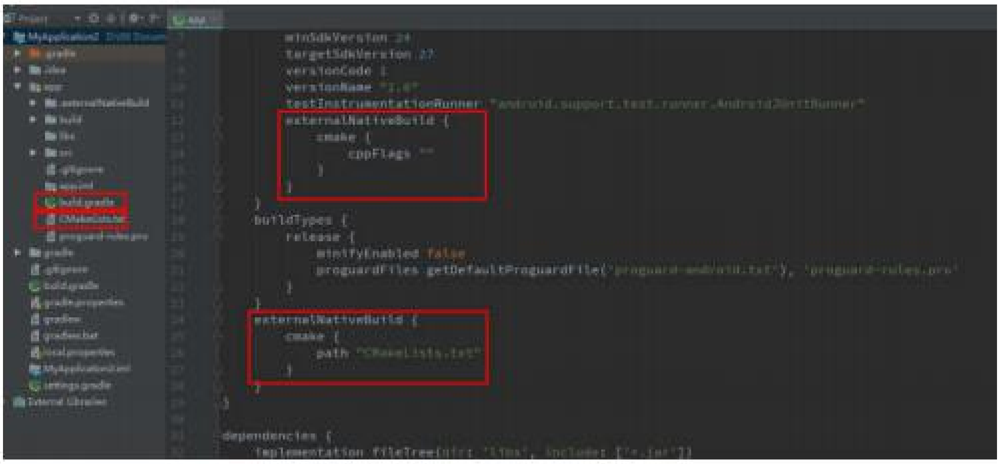


(4) SimpleCode description
Under the Linux system, different sample codes are provided for applications such as the standard API and extended API in the SDK. For specific operations and instructions, please refer to the routine compilation steps in the Linux environment configuration. For the in-depth flow steps, please refer to Windows.

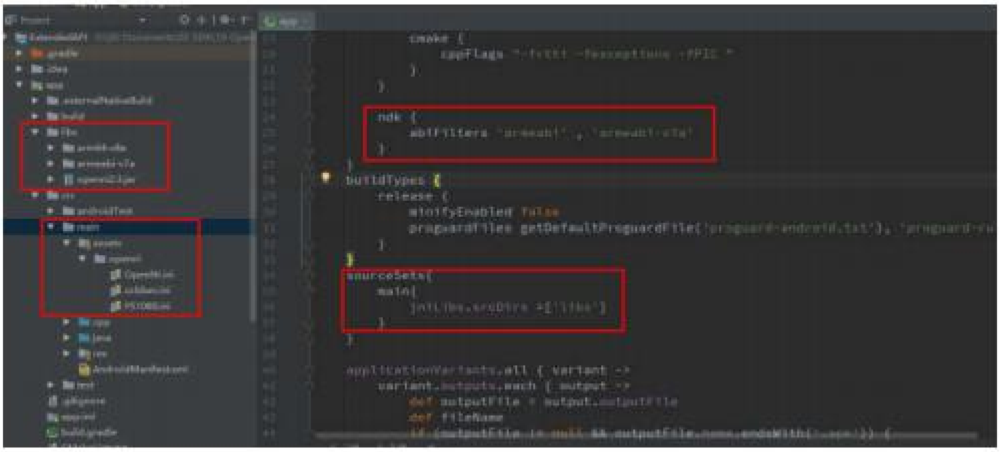
3.2.3 Android project environment configuration

(1) Scope of application
Under the Android platform, the company's SDK is suitable for Android 4.4.2 and above platforms
(2) Android environment configuration instructions
For Android SDK development, please build an Android Studio development environment. You can build a pure Java environment for development, or build a Java project that supports C++ for development, configure NDK, and call C++ API through JNI.
Before Android2.2, JNI was implemented through NDK+Android.mk+Application.mk, and later versions were implemented using ndk+Cmake. The following is introduced
The configuration process of Android Studio NDK environment.
* Create a new Android Studio that supports C++, check Include C++ support, and click Next until the creation is complete.

*It can be seen from the directory structure that there are several differences compared to ordinary Android projects



*Add library and resource files in SDK, configure gradle file



*Modify the cmake file and configure the created cpp file, you can call the C++ API in the Java project, and the ExtAPIDemo project file can be tested for detailed project configuration and API registration calls.

```
# You can define multiple libraries, and CMake builds it for you.
# Gradle automatically packages shared libraries with your APK.

add_library(OpenNI2
            SHARED
            IMPORTED
)

set_target_properties(OpenNI2
                      PROPERTIES IMPORTED_LOCATION
                      ../../../../libs/armeabi-v7a/libOpenNI2.so)

add_library( # Sets the name of the library.
             IRUtils

             # Sets the library as a shared library.
             SHARED

             # Provides a relative path to your source file(s).
             # Associated headers in the same location as their source
             # file are automatically included.
             src/main/cpp/IRUtils.cpp )
```

```
find_library( # Sets the name of the path variable.
              log-lib

              # Specifies the name of the NDK library that
              # you want CMake to locate.
              log )

# Specifies libraries CMake should link to your target library. You
# can link multiple libraries, such as libraries you define in the
# build script, prebuilt third-party libraries, or system libraries.

target_link_libraries( # Specifies the target library.
                       IRUtils
                       OpenNI2
                       # Links the target library to the log library
                       # included in the NDK.
                       ${log-lib} )
```

(3) SimpleCode description

Get the depth flow steps

1. Create the OpenNIHelper class

```
private OpenNIHelper mOpenNIHelper;
mOpenNIHelper = new OpenNIHelper(this);
```

2. Register the device to open the listener

```
mOpenNIHelper.requestDeviceOpen(this);
```

3. Implement the DeviceOpenListener interface abstract functions onDeviceOpened, onDeviceOpenFailed@Override

```
public void onDeviceOpened(UsbDevice device) {
init(device);
mStream = VideoStream.create(mDevice, SensorType.IR);
mVideoModes = mStream.getSensorInfo().getSupportedVideoModes();
for (VideoMode mode : mVideoModes) {
int X = mode.getResolutionX();
int Y = mode.getResolutionY();
int fps = mode.getFps();
Log.d(TAG, " support resolution: " + X + " x " + Y + " fps: " + fps + ", (" +
```

```
mode.getPixelFormat() + ")");
if (X == mWidth && Y == mHeight && mode.getPixelFormat() == PixelFormat.RGB888 &&
fps == 30) {
mStream.setVideoMode(mode);
Log.v(TAG, " setmode");
}
}
startThread();
}
@Override
public void onDeviceOpenFailed(String msg) {
showAlertAndExit("Open Device failed: " + msg);
}
```

4、Complete the device initialization in init(device) and open the OpenNI deviceprivate void

```
init(UsbDevice device) {
OpenNI.setLogAndroidOutput(true);
OpenNI.setLogMinSeverity(0);
OpenNI.initialize();
List<DeviceInfo> opennilist = OpenNI.enumerateDevices();
if (opennilist.size() <= 0) {
Toast.makeText(this, " openni enumerateDevices 0 devices", Toast.LENGTH_LONG).show();
return;
}
mDevice = null;
//Find device ID
for (int i = 0; i < opennilist.size(); i++) {
if (opennilist.get(i).getUsbProductId() == device.getProductId()) {
mDevice = Device.open();
break;
}
}
if (mDevice == null) {
Toast.makeText(this, " openni open devices failed: " + device.getDeviceName(),
Toast.LENGTH_LONG).show();
return;
}
}
```

5、startThread（）Start the thread to read the video stream

```
void startThread() {
mInit_Ok = true;
m_thread = new Thread() {
@Override
public void run() {
List<VideoStream> streams = new ArrayList<VideoStream>();
```

```
streams.add(mStream);
mStream.start();
while (!mExit) {
try {
OpenNI.waitForAnyStream(streams, 2000);
} catch (TimeoutException e) {
e.printStackTrace();
continue;
}
synchronized (m_sync) {
mGLView.update(mStream);
}
}
}
};
m_thread.start();
}
```

3.2.4 Instructions for using .ini configuration file

(1) OpenNI.ini configuration instructions

OpenNI2.3.0.50 provides a wealth of debugging logs. When you turn on the corresponding level of log, you can view the current depth and color flow information on the screen or in a file. This function can be realized by modifying the OpenNI.ini file.

```
[Log]
; 0 - Verbose; 1 - Info; 2 - Warning; 3 - Error. Default - None
;Verbosity=0    // 去掉注释符 ;, 修改为 Verbosity=0 查看最详细调试信息
;LogToConsole=1 //  去掉注释符 ;,修改为 LogToConsole=1 将调试信息输出到控制台
;LogToFile=1    // 去掉注释符 ;,修改为 LogToFile=1 可以保持 log 到文件
;LogToAndroidLog=1   // 去掉注释符 ;,修改 LogToAndroidLog=1 输出 Log 到 logcat

[Device]
;Override=
;RecordTo=

[Drivers]
; Location of the drivers, relative to OpenNI shared library location. When not provided, "OpenNI2/Drivers" will be used.
;Repository=OpenNI2/Drivers

; List of drivers to load, separated by commas. When not provided, OpenNI will try to load each shared library in Repository.
;List=
```

(2) Orbbec.ini configuration instructions

OpenNI2.3.0.50 provides a flexible external configuration function. By modifying the orbbec.ini file, the software library sets the working status of the camera by parsing the configuration file.

The following are the key settings of the configuration file

```
;---------------- Sensor Default Configuration -------------------
[Device]
; Mirroring. 0 - Off (default), 1 - On
;Mirror=1

; FrameSync. 0 - Off (default), 1 - On
;FrameSync=1

; Stream Data Timestamps. 0 - milliseconds, 1 - microseconds (default)
;HighResTimestamps=1

; Stream Data Timestamps Source. 0 - Firmware (default), 1 - Host
;HostTimestamps=0

; A filter for the firmware log. Default is determined by firmware.
;FirmwareLogFilter=0

; Automatic firmare log retrieval. 0 - Off (default), or the number of milliseconds between log retrievals operations.
;FirmwareLogInterval=1000

; Print firmware log to console when automatic firmware log retrieval is on. 0 - Off (default), 1 - On
;FirmwareLogPrint=1

; Is APC enabled. 0 - Off, 1 - On (default)
;APCEnabled=1

; USB interface to be used. 0 - FW Default, 1 - ISO endpoints (default on Windows), 2 - BULK endpoints (default on Linux/Mac/Android machines)
UsbInterface=2
```

The item in the symbol [] indicates the current configuration device, which can configure the device Device, depth camera Depth, color camera Image and infrared camera IR items, while retaining the configuration items Depth.Cropping and IR.Cropping for depth map and color map cropping .
Take the configuration item in the above figure [Device] as an example. If you want to modify a specific item, you need to remove the semicolon used in the comment, and then select the corresponding configuration value according to the description. The following describes common configuration items.
* [Device]:
Mirror——device mirroring mode selection;
FrameSync——Depth and color frame synchronization settings. It should be noted that only non-UVC devices support this configuration item.
* [Depth]:
outputFormat——Select the depth raw data format of the output
Mirror——Set the depth map mirror mode
Resolution-depth image resolution setting
FPS-deep frame rate selection
Registration-Set RGBD hardware alignment
HoleFilter——Set the filter mode. For models with inconsistent depth and color resolution, by setting HoleFilter=2,
The black hole on the depth map after RGBD alignment can be removed.
* [Image]:
outputFormat——Select the output color raw data format
Mirror——Set the color image mirroring mode
The rest of the configuration items are similar to Depth, but it should be noted that the settings under the Image configuration item are only effective for color cameras that are not set by UVC. UVC cameras, such as Deeyea, Astra D-U, etc., cannot configure color cameras through this item.
* [IR]:
outputFormat——Select the output NIR image data format
Mirror——Set IR image mirror

Resolution——Select IR image resolution

It should be noted that since the Depth image is calculated based on the IR image, it is necessary to modify the corresponding options in the IR configuration item to modify the resolution and frame rate of the Depth image.

4.0 Introduction to Common APIs

4. Introduction to commonly used APIs

In the Document folder of the SDK package, the OrbbecSDK_v2.3.chm file is provided, which summarizes all the APIs supported in the SDK, and provides interface introduction and usage instructions. You can also select c++.chm or openni2java.chm to view the C++ or Java API description according to the development language. The document directory structure is as follows:

4.1 Introduction to Standard API

The standard API is a native API provided by the OpenNI open source software framework, including commonly used classes such as Device, VideoStream, and VideoMode.

There are 4 main classes required to obtain a deep video stream.

1. openni::OpenNI provides a static API entry point. It provides access to the device, device related events, version and error information. Of course, you must first make sure you connect the device.

2. openni::Device provides an interface for sensor device connection system (personal understanding is to access and control sensors through the Device class). Before it is created, the OpenNI class needs to be initialized. Device can access streams (Streams).

3. Openni::VideoStream extracts a video stream from a device (Device), and needs to obtain video frame references (VideoFrameRefs).

4. openni::VideoFramRef extracts a video frame from the related source data. This is obtained from a specific stream.

In addition to these main classes, there are many classes and structures used to hold some special types of data. The Rocorder class is used to store OpenNI video streams to files. There is also the Listener class to monitor events generated by the OpenNI and Stream classes.

The video stream can obtain data in two ways: polling and events. The following specifically introduces each commonly used class

4.1.1 OpenNI class

1) Introduction

The OpenNI class provides a static entry for library functions. Each OpenNI2.0 application needs to use the OpenNI class to initialize the SDK and device drivers. It is also possible to generate many device connection and disconnection events, and provide the same function of accessing data streams in a polling manner.

2) Basic access to equipment

The OpenNI class provides the OpenNI::initialize() method to initialize library functions and scan all available sensor devices in the system. All applications using OpenNI should call this method before using other APIs. Once the initialization method is completed, it is possible to create Device objects and use these objects to interact with the real sensor hardware. The OpenNI::enumerateDevices() method returns a list of available sensor devices connected to the system. When the application is ready to exit, the OpenNI::shutdown() method must be called to shut down all drivers and exit correctly.

3) The polling access interface of the basic access stream of video

streams is implemented by the OpenNI::waitForStream() method. One of the parameters of this method is the list of streams. When called, it will be locked until the stream in the list has new data available or it times out. Then return a status code (status code) and point to the specific data stream

4) Event-driven access to equipment

The OpenNI class provides a framework for accessing devices in an event driven manner. OpenNI defines three events: device connection event (onDeviceConnected), device disconnection event (onDeviceDisconnected), and device state change event (onDeviceStateChanged). The device connection event is generated when a new device is connected and available through OpenNI, and the device disconnection event is generated when a device is removed from the system. The device status change event is generated when the device setting is changed. The following methods can be used to add or remove Listener classes from the event handling list:
OpenNI::addDeviceConnectedListener()//Add device connection event listener
OpenNI::addDeviceDisconnectedListener()//Add device disconnect event listener
OpenNI::addDeviceStateChangedListener()//Add device state change event listener
OpenNI::removeDeviceConnectedListener()//Remove the device connection event listener
OpenNI::removeDeviceDisconnectedListener()//Remove the device disconnection event listener
OpenNI::removeDeviceStateChangedListener()//Remove the device state change event listener
All three events provide a pointer to the OpenNI::DeviceInfo object. This object is used to obtain the details and identification of the device submitted by the event. In addition, the device state change event also provides a pointer to the DeviceState object, which is used to view the new state information of the device. Event-driven access to the real video stream through the VideoStream class

5) Error message

There are many methods in the SDK that return a value of type "Status". When an error occurs, Status will contain a record or code displayed to the user. The OpenNI::getExtendedError() method returns more readable information about the error.

6) Version information

The version information of the API is obtained by the OpenNI::getVersion() method. This method returns the version information of the API currently used by the application

4.1.2 Device class

1 Introduction

The openni::Device class provides an abstract interface for a physical hardware device, and can also provide an interface for simulating a hardware device through an ONI record file derived from a physical device.

The Device object is used to connect and configure the underlying file or hardware device, and create a stream from the device. The Device object provides an interface for device configuration query and modification, which can be used to enable RGBD alignment or frame synchronization. Before creating and initializing the video stream, the Device object must be constructed and pointed to the physical device.

2) Equipment connection conditions

Before the device class can be connected to the physical hardware device, the device must be physically and correctly connected to the host, and the driver must be installed. If the connection is an ONI file, the ONI record file must be available when the system is running the application, and the application has sufficient permissions to access it.

Of course, the openni::OpenNI::initialize() method needs to be called before contacting the device. This will initialize the driver and let the API know that the device is connected.

3) Basic operation

*Constructor

The constructor of the Device class has no parameters, nor does it contact the physical hardware device. Just simply create the object.

* Device::open()

This method is used to connect to physical hardware devices. The open() method has a parameter, the URI (Uniform Resource Identifier) of the device, and the method returns a status code indicating whether it is successful.

The simplest usage is to use the constant openni::ANY_DEVICE as the URI of the device. Using this constant will make the system connect to all hardware devices, but it is only suitable for situations where only one device is connected.

If multiple sensors are connected, you must first call OpenNI::enumerateDevices() to get the list of available devices. Then find the device you are looking for, get the URI by calling DeviceInfo::getUri(), and use the output of this method as Device::open()

Then you can open the corresponding device.

If you open the file, the parameter is the path of the ONI file.

*Device::close()

The close() method is used to close the hardware device. By convention, all open devices must be closed. This will separate the hardware device and the driver so that subsequent applications can open the device again. Device::isValid()isValid() method is used to determine whether the device is correctly connected to the device object

*Device::setImageRegistrationMode()

Orbbec products will generate depth stream and color image stream at the same time, which are generated by different physical cameras (RGB camera and IR camera). The actual position of the physical camera is different, resulting in a difference in the depth and field of view of the color

picture, which makes the images obtained from different streams of the same device object different. The geometric relationship and distance between the two cameras are known to the device object, and the two images can be consistent and superimposed on each other through mathematical transformations, for example, each pixel of the depth map is superimposed on the color image on. This process is alignment (each pixel is superimposed on another image).
Orbbec equipment supports hardware calculations and can calibrate data in the module ISP. At the same time, there is a flag on the hardware to turn it on or off.
The device object provides the isImageRegistrationSupported() method to test whether the connected device supports the alignment function.
If supported, getImageRegistrationMode() can be used to query the status of this function, SetImageRegistrationMode() can set it. The openni::ImageRegistrationMode enumeration provides the following values for set or get:
IMAGE_REGISTRATION_OFF – Hardware registration features are disabled
IMAGE_REGISTRATION_DEPTH_TO_IMAGE – The depth image is transformed to have the same apparent vantage point as the RGB image, the depth image is transformed and superimposed on the color image
It should be noted that the FOV of the two sensors does not overlap in part. This causes some depth maps to not be displayed in the results.
*Device::setDepthColorSyncEnabled()
When both depth and color image streams are available, slight difference in frame rate or frame arrival time may cause frame synchronization problems.

Orbbec provides a frame synchronization function for non-UVC devices. In order to obtain two frames within a determined maximum time range, the maximum value is usually less than the two-frame interval.
To enable or disable this function, use setDepthColorSyncEnable().
However, it should be noted that all UVC devices cannot perform frame synchronization settings through this API.

4.1.3 VideoStream class

1 Introduction

The video stream class created by the device class encapsulates all data streams. The user can start, stop and configure the data stream, and can also configure the parameters of the stream level (as opposed to the device level).

2) Basic functions of video streaming

* Create and initialize video stream
Calling the default constructor of the video stream will create an empty uninitialized video stream object. Before use, this object must be initialized by calling VideoStream::create(). The create() method requires an initialized device object. Once created, you can call the VideoStream::start() method to generate a data stream. The VideoStream::stop() method will stop the data stream.
*Data reading based on polling
Once the video stream is created, you can read the data through the VideoStream::readFrame() method. If there is new data available, this method will return a reference (VideoFrameRef) that

can access the latest video frame generated by the video stream.
If there is no new frame available, it will lock until a new frame is available.
It should be noted that if you read from the record acyclically, the program will always be stuck in this method after the last frame is read.
* Event-based data reading
Data can be read from the video stream in an event driven manner.
First, you need to create an inherited class of VideoStream::Listener, which should implement the method onNewFrame(). Once you have created this class and instantiated it, you can add listeners via the VideoStream::addListener() method. When a new frame arrives, the onNewFrame() method of the custom listener class is called. Then you can call readFrame() to read.

3) Get information about the video stream

* Sensor Info (SensorInfo) and video mode (VideoMode)
Sensor information and video mode can always track the information of the video stream. The video mode encapsulates the frame rate, resolution and pixel format of the video stream.
The sensor information contains the type of sensor that generates the video stream and the list of video mode objects for each stream. By traversing this list, we can determine all the available Energy mode.
Use VideoStream::getSensorInfo to get the sensor information object of the current stream
* Field of View
This function determines the field of view of the sensor. Use getHorizonFieldOfView() and getVerticalFieldOfView() methods to determine the field of view, and the returned value is radians.
* Min and Max PixelValues
In depth streams, it is often useful to know the maximum and minimum possible values for a pixel. Use the getMinPixelValue() and getMaxPixleValue() methods to get this information.

4) Configure the video stream

* Video Mode (Video Mode)
You can set the frame rate, resolution, and pixel type of a given stream. To set these, use the setVideoMode() method. Before that, you first need to obtain the sensor information (SensorInfo) of the configured video stream, and then you can select an available video mode.
* Cropping
If the sensor supports cropping, the video stream provides a way to control it. Use VideoStream::isCroppingSupported() method to determine whether it is supported.
If supported, use setCropping() to enable cropping and set the specific configuration of cropping. The ResetCropping() method is used to turn off cropping again. The getCropping() method is used to obtain the current cropping settings.
* Mirroring
Mirroring, as the name suggests, is to make what the video stream shows looks like in a mirror. To enable or disable mirroring, use the VideoStream::setMirroringEnable() method. Set true to enable, set false to disable. Use getMirroringEnable() to get the current setting.
* General Properties
At the firmware level, most sensor settings are stored as address/value pairs (address/value pairs). So it can be directly operated by the setProperty and getProperty methods. These

methods are used internally by the SDK to implement cropping, mirroring, and so on. And they are usually not used frequently by applications, because most of the useful properties are encapsulated in more friendly methods
Up.

4.1.4 VideoFrameRef Class

1 Introduction

The video frame reference class encapsulates all the data of a single frame read from the video stream. It provides access to a basic array containing frame data (metadata, frames required for work). The video frame reference object is obtained from the VideoStream::readFrame() method. The video frame reference data can be obtained from an infrared camera, an RGB camera or a depth camera. The getSensorType() method is used to determine the sensor type that generated this frame. It will return the sensor type.

2) Access frame data

The VideoFrameRef::getDate() method returns a pointer directly to the frame data. The type is void, so that the data type of each pixel can be indexed correctly.

3) Cropping data

The data frame reference knows the crop settings of the video stream, so it can be used to determine the origin of the crop box, the size of the crop box and whether the frame is enabled for cropping. The implementation method is as follows: getCropOriginX(), getCropOriginY(), getCroppingEnable(). If the cropping function is enabled, the cropping frame size is equal to the frame size.

4) Timestamp (Timestamp)

Each frame of data has a time stamp. This value is based on the number of microseconds since any value of 0. Different from the time difference between two frames, all streams of the same device use the same 0 value, so the difference in timestamp can be used to compare frames of different streams. In OpenNI 2.0, the 0 value of the timestamp is the arrival time of the first frame of data. However, there is no guarantee that it will be the same every time, so the program code should use timestamp increments. The time stamp value itself should not be used as an absolute time point.

5) Frame Index (FrameIndex)

In addition to the time stamp, the frame also provides consecutive frame index numbers. This is useful in determining how many frames are between two known frames. If the frame synchronization is enabled, the frame numbers of the corresponding frames should be consistent. If there is no synchronization, the frame numbers will not necessarily match. In this case, it is more effective to use the time stamp to determine the position of the relevant frame.

5) Video Mode

VideoFrameRef::getVideoMode() is used to determine the video mode of the sensor that generated the current frame. Information includes pixel format, resolution, and frame rate.

6) Array Stride

The span of the array containing the frames can be obtained with getStrideBytes(). It will return the size of each row of the array, in bytes. Mainly used to index two-dimensional image data.

4.1.5 Other support categories

1 Introduction

In addition to the main classes mentioned above, OpenNI also has a series of supporting classes. These classes mainly serve to encapsulate data and are mentioned in the chapters of other main classes.

2) Sensor configuration class

The device information (DeviceInfo) records all the configuration of the device, including device name, URI, USB VID/PID descriptor and supplier.

3) Sensor Info (SensorInfo)

This class stores all the configurations of the sensor, and the sensor here is only one of the three sensors. A device has multiple sensors.

4) Video mode (VideoMode)

This class stores the resolution, frame rate and pixel format. Used for video streaming settings and viewing settings, referenced by the video frame to view these settings, and sensor information provides a list of video modes.

5) Camera setting (CameraSetting)

The settings of the RGB camera of non-UVC devices are stored, and auto white balance and auto exposure can be enabled or disabled.

4.2 Introduction to Extension API

Orbbec has been extended on the basis of the general OpenNI2 function. It is an API interface that can realize functions such as reading device serial number, obtaining device type, using flash to save data, reading camera calibration parameters, IR camera parameter adjustment, and LDP control. The extended API is defined in the Device class. If users need to use these extended functions, they need to ensure that the OpenNI2 initialize is successful and the device is successfully opened before calling the extended API. The specific API instructions are as follows:

4.2.1 Get device model

```
char devType[32];
int dataSize = sizeof(devType);
memset(devType, 0, dataSize);
g_Device.getProperty(openni::OBEXTENSION_ID_DEVICETYPE, (uint8_t *)&devType, &dataSize);
```

4.2.2 Get device serial number

```
char serNumber[12];
int dataSize = sizeof(serNumber);
memset(serNumber, 0, dataSize);
g_Device.getProperty(openni::OBEXTENSION_ID_SERIALNUMBER, (uint8_t *)&serNumber,
&dataSize);
```

4.2.3 Set LDP switch

```
int dataSize = 4;
int ldp_en = enable;
g_Device.setProperty(openni::OBEXTENSION_ID_LDP_EN, (uint8_t *)&ldp_en, dataSize);
```

4.2.4 Set LDM switch

```
int dataSize = 4;
int laser_en = enable;
g_Device.setProperty(openni::OBEXTENSION_ID_LASER_EN, (uint8_t *)&laser_en, dataSize);
```

4.2.5 Set and get IR exposure value

```
int exposure = 0;
int dataSize = 4;
g_device.getProperty(openni::OBEXTENSION_ID_IR_EXP, (uint8_t *)&exposure, &dataSize);
printf("ir exposure value : 0x%x\n", exposure);
exposure += 256;
g_device.setProperty(openni::OBEXTENSION_ID_IR_EXP, (uint8_t *)&exposure, dataSize);
```

4.2.6 Set and get IR gain

```
int gain = 0;
int dataSize = 4;
g_device.getProperty(openni::OBEXTENSION_ID_IR_GAIN, (uint8_t *)&gain, &dataSize);
printf("ir gain value : 0x%x\n", gain);
gain++;
g_device.setProperty(openni::OBEXTENSION_ID_IR_GAIN, (uint8_t *)&gain, dataSize);
```

4.2.7 Setting, saving and obtaining camera calibration parameters

```
typedef struct OBCameraParams
{
float l_intr_p[4];                //[fx,fy,cx,cy]
float r_intr_p[4];       //[fx,fy,cx,cy]
```

```
float r2l_r[9];                    //[r00,r01,r02;r10,r11,r12;r20,r21,r22]
float r2l_t[3];                    //[t1,t2,t3]
float k[5];                          //[k1,k2,p1,p2,k3]
int is_mirror;
} OBCameraParams;
OBCameraParams m_CamParams = { 0 };
int dataSize = sizeof(OBCameraParams);
m_CamParams.l_intr_p[0] = 577.318970;
m_CamParams.l_intr_p[1] = 577.318970;
m_CamParams.l_intr_p[2] = 308.729004;
m_CamParams.l_intr_p[3] = 269.143005;
m_CamParams.r_intr_p[0] = 517.447998;
m_CamParams.r_intr_p[1] = 517.447998;
m_CamParams.r_intr_p[2] = 305.432007;
m_CamParams.r_intr_p[3] = 250.410995;
m_CamParams.r2l_r[0] = 0.999972;
m_CamParams.r2l_r[1] = -0.005735;
m_CamParams.r2l_r[2] = 0.004735;
m_CamParams.r2l_r[3] = 0.005736;
m_CamParams.r2l_r[4] = 0.999983;
m_CamParams.r2l_r[5] = -0.000298;
m_CamParams.r2l_r[6] = -0.004733;
m_CamParams.r2l_r[7] = 0.000325;
m_CamParams.r2l_r[8] = 0.999989;
m_CamParams.r2l_t[0] = -25.147900;
m_CamParams.r2l_t[1] = 0.015202;
m_CamParams.r2l_t[2] = -0.648167;
m_CamParams.k[0] = -0.077348;
m_CamParams.k[1] = 0.208761;
m_CamParams.k[2] = -0.196780;
m_CamParams.k[3] = 0.000617;
m_CamParams.k[4] = 0.001059;
m_CamParams.is_mirror = 0;
```

```
g_Device.setProperty(openni::OBEXTENSION_ID_CAM_PARAMS, (uint8_t *)&m_CamParams,
dataSize);
g_Device.getProperty(openni::OBEXTENSION_ID_CAM_PARAMS, (uint8_t *)&m_CamParams,
&dataSize);
```