PyRealSense Documentation

Release 2.2.0

Antoine Loriette

Sep 29, 2017

Contents

CHAPTER 1

Readme

Cross-platform ctypes/Cython wrapper to the librealsense library.

Prerequisites

- install librealsense and run the examples.
- install the dependencies: pyrealsense uses pycparser for extracting necessary enums and structures definitions from the librealsense API, Cython for wrapping the inlined functions in the librealsense API, and Numpy for generic data shuffling.
- Windows specifics: set environment variable PYRS_INCLUDES to the `rs.h` directory location and environment variable PYRS_LIBS to the librealsense binary location. You might also need to have `stdint.h` available in your path.

Installation

from PyPI - (OBS: not always the latest):

```
pip install pyrealsense
```

from source:

```
python setup.py install
```

Online Usage

```
## setup logging
import logging
logging.basicConfig(level = logging.INFO)

## import the package
import pyrealsense as pyrs

## start the service - also available as context manager
serv = pyrs.Service()

## create a device from device id and streams of interest
cam = serv.Device(device_id = 0, streams = [pyrs.stream.ColorStream(fps = 60)])

## retrieve 60 frames of data
for _ in range(60):
    cam.wait_for_frames()
    print(cam.color)

## stop camera and service
cam.stop()
serv.stop()
```

The server for Realsense devices is started with `pyrs.Service()` which will printout the number of devices available. It can also be started as a context with `with pyrs.Service():`.

Different devices can be created from the service `Device` factory. They are created as their own class defined by device id, name, serial, firmware as well as enabled streams and camera presets. The default behaviour create a device with `id = 0` and setup the color, depth, pointcloud, color_aligned_depth, depth_aligned_color and infrared streams.

The available streams are either native or synthetic, and each one will create a property that exposes the current content of the frame buffer in the form of `device.<stream_name>`, where `<stream_name>` is color, depth, points, cad, dac or infrared. To get access to new data, `Device.wait_for_frames` has to be called once per frame.

Offline Usage

```
## with connected device cam
from pyrealsense import offline
offline.save_depth_intrinsics(cam)
```

```
## previous device cam now offline
from pyrealsense import offline
offline.load_depth_intrinsics('610205001689')  # camera serial number
d = np.linspace(0, 1000, 480*640, dtype=np.uint16)
pc = offline.deproject_depth(d)
```

The module `offline` can store the rs_intrinsics and depth_scale of a device to disk by default in the user's home directory in the file `.pyrealsense`. This can later be loaded and used to deproject depth data into pointcloud, which is useful to store raw video file and save some disk memory.

Examples

There are 3 examples using different visualisation technologies: - still color with matplotlib - color and depth stream with opencv - pointcloud stream with VTK

Caveats

To this point, this wrapper is tested with:

- librealsense v1.12.1
- Ubuntu 16.04 LTS, Mac OS X 10.12.2 w/ SR300 camera
- Mac OS X 10.12.3 w/ R200 camera

The offline module only supports a single camera.

Build Status

Ubuntu Trusty, python 2 and 3:

Possible Pull Requests

- improvments to the documentation
- more functionality from `rs.h`
- boiler plate code (Qt example?)
- support for several cameras in offline module
- continuous integration for Windows and MacOs

Make sure to push to the `dev` branch.

CHAPTER 2

Code source docs

pyrealsense package

On import, you get access to the Service class which handles device creation.

pyrealsense.core module

pyrealsense.core.**Device**(*service*, *device_id=0*, *streams=None*, *depth_control_preset=None*, *ivcam_preset=None*)

Camera device, which subclass *DeviceBase* and create properties for each input streams to expose their data. It should be instantiated through *Service.Device()*.

Parameters

- **service** (*Service*) – any running service.
- **device_id** (*int*) – the device id as hinted by the output from `start()`.
- **streams** (`list` of *pyrealsense.stream.Stream*) – if None, all streams will be enabled with their default parameters (e.g *640x480@30FPS*)
- **depth_control_preset** (*int*) – optional preset to be applied.
- **ivcam_preset** (*int*) – optional preset to be applied with input value from *pyrealsense.constants.rs_ivcam_preset*.

Returns A subclass of *DeviceBase* which class name includes the device serial number.

class pyrealsense.core.**DeviceBase**(*dev*, *device_id*, *name*, *serial*, *version*, *streams*)

Bases: `object`

Camera device base class which is called via the *Device()* factory. It exposes the main functions from librealsense.

apply_ivcam_preset(*preset*)

Provide access to several recommend sets of option presets for ivcam.

Parameters **preset** (*int*) – preset from (*pyrealsense.constants.rs_ivcam_preset*)

deproject_pixel_to_point(*pixel*, *depth*)
Deproject a 2d pixel to its 3d point coordinate by calling rsutil's rs_deproject_pixel_to_point under the hood.

Parameters

- **pixel** (*np.array*) – (x,y) coordinate of the point
- **depth** (*float*) – depth at that pixel

Returns (x,y,z) coordinate of the point

Return type point (np.array)

get_available_options()
Returns available options as a list of (DeviceOptionRange, value).

get_device_extrinsics(*from_stream*, *to_stream*)
Retrieve extrinsic transformation between the viewpoints of two different streams.

Parameters

- **from_stream** (*pyrealsense.constants.rs_stream*) – from stream.
- **to_stream** (*pyrealsense.constants.rs_stream*) – to stream.

Returns extrinsics parameters as a structure

Return type (*pyrealsense.extstruct.rs_extrinsics*)

get_device_modes()
Returns a generator that yields all possible streaming modes as StreamMode.

get_device_option(*option*)
Get device option.

Parameters **option** (*int*) – taken from *pyrealsense.constants.rs_option*.

Returns option value.

Return type (double)

get_device_option_description(*option*)
Get the device option description.

Parameters **option** (*int*) – taken from *pyrealsense.constants.rs_option*.

Returns option value.

Return type (str)

get_device_option_range_ex(*option*)
Get device option range.

Parameters **option** (*int*) – taken from *pyrealsense.constants.rs_option*.

Returns option range.

Return type (DeviceOptionRange)

get_device_options(*options*)
Get device options.

Parameters **option** (list of int) – taken from *pyrealsense.constants.rs_option*.

Returns options values.

Return type (iter of double)

get_frame_number(*stream*)
Retrieve the frame number for specific stream.

Parameters **stream** (*int*) – value from *pyrealsense.constants.rs_stream*.

Returns frame number.

Return type (double)

get_frame_timestamp(*stream*)
Retrieve the time at which the latest frame on a specific stream was captured.

Parameters **stream** (*int*) – stream id

Returns timestamp

Return type (long)

is_streaming()
Indicates if device is streaming.

Returns return value of *lrs.rs_is_device_streaming*.

Return type (bool)

poll_for_frame()
Check if new frames are available, without blocking.

Returns 1 if new frames are available, 0 if no new frames have arrived

Return type int

Raises utils.RealsenseError – in case librealsense reports a problem.

project_point_to_pixel(*point*)
Project a 3d point to its 2d pixel coordinate by calling rsutil's rs_project_point_to_pixel under the hood.

Parameters **point** (*np.array*) – (x,y,z) coordinate of the point

Returns (x,y) coordinate of the pixel

Return type pixel (np.array)

reset_device_options_to_default(*options*)
Reset device options to default.

Parameters **option** (list of int) – taken from *pyrealsense.constants.rs_option*.

set_device_option(*option*, *value*)
Set device option.

Parameters

- **option** (*int*) – taken from *pyrealsense.constants.rs_option*.
- **value** (*double*) – value to be set for the option.

set_device_options(*options*, *values*)
Set device options.

Parameters

- **option** (list of int) – taken from *pyrealsense.constants.rs_option*.

- **values** (list of double) – options values.

stop()
End data acquisition. :raises: utils.RealsenseError – in case librealsense reports a problem.

wait_for_frames()
Block until new frames are available.

Raises utils.RealsenseError – in case librealsense reports a problem.

class pyrealsense.core.**Service**
Bases: object

Context manager for librealsense service.

Device(**args*, ***kwargs*)
Factory function which returns a *Device*, also accepts optionnal arguments.

get_device_modes(*device_id*)
Generates all different modes for the device which *id* is provided.

Parameters **device_id** (*int*) – the device id as hinted by the output from *start()* or *get_devices()*.

Returns: generator that yields all possible streaming modes as StreamMode.

get_devices()
Returns a generator that yields a dictionnary containing 'id', 'name', 'serial', 'firmware' and 'is_streaming' keys.

is_device_streaming(*device_id*)
Indicates if device is streaming.

Utility function which does not require to enumerate all devices or to initialize a Device object.

start()
Start librealsense service.

stop()
Stop librealsense service.

pyrealsense.stream module

class pyrealsense.stream.**CADStream**(*name='cad'*, *width=640*, *height=480*, *fps=30*, *color_format='rgb'*)
Bases: *pyrealsense.stream.Stream*

CAD stream from device, with default parameters.

class pyrealsense.stream.**ColorStream**(*name='color'*, *width=640*, *height=480*, *fps=30*, *color_format='rgb'*)
Bases: *pyrealsense.stream.Stream*

Color stream from device, with default parameters.

class pyrealsense.stream.**DACStream**(*name='dac'*, *width=640*, *height=480*, *fps=30*)
Bases: *pyrealsense.stream.Stream*

DAC stream from device, with default parameters.

class pyrealsense.stream.**DepthStream**(*name='depth'*, *width=640*, *height=480*, *fps=30*)
Bases: *pyrealsense.stream.Stream*

Depth stream from device, with default parameters.

class pyrealsense.stream.**InfraredStream**(*name='infrared'*, *width=640*, *height=480*, *fps=30*)
Bases: *pyrealsense.stream.Stream*

Infrared stream from device, with default parameters.

class pyrealsense.stream.**PointStream**(*name='points'*, *width=640*, *height=480*, *fps=30*)
Bases: *pyrealsense.stream.Stream*

Point stream from device, with default parameters.

class pyrealsense.stream.**Stream**(*name*, *native*, *stream*, *width*, *height*, *format*, *fps*)
Bases: object

Stream object that stores all necessary information for interaction with librealsense. See for possible combinations.

Parameters

- **name** (*str*) – name of stream which will be used to create a @property on *pyrealsense.core.DeviceBase*.
- **native** (*bool*) – whether the stream is native or composite
- **stream** (*int*) – from *pyrealsense.constants.rs_stream*
- **width** (*int*) – width
- **height** (*int*) – height
- **format** (*int*) – from *pyrealsense.constants.rs_format*
- **fps** (*int*) – fps

pyrealsense.offline module

pyrealsense.extlib module

This module loads rsutilwrapper and librealsense library.

pyrealsense.extstruct module

This module manually wraps structures defined in rs.h.

class pyrealsense.extstruct.**rs_context**
Bases: _ctypes.Structure

This is a placeholder for the context. It is only defined to hold a reference to a pointer.

class pyrealsense.extstruct.**rs_device**
Bases: _ctypes.Structure

This is a placeholder for the context. It is only defined to hold a reference to a pointer.

class pyrealsense.extstruct.**rs_error**
Bases: _ctypes.Structure

This is a 1-to-1 mapping to rs_error from librealsense.

The *_fields_* class variable is defined as follows:

- `message` (c_char_p): error message
- `function` (pointer(c_char)): function which caused the error
- `args` (c_char_p): arguments to the function which caused the error

class `pyrealsense.extstruct.`**`rs_extrinsics`**
Bases: `_ctypes.Structure`

This is a 1-to-1 mapping to rs_extrinsics from librealsense.

The *_fields_* class variable is defined as follows:

- `rotation` (c_float*9): column-major 3x3 rotation matrix
- `height` (c_float*3): 3 element translation vector, in meters

class `pyrealsense.extstruct.`**`rs_intrinsics`**
Bases: `_ctypes.Structure`

This is a 1-to-1 mapping to rs_intrinsics from librealsense.

The *_fields_* class variable is defined as follows:

- `width` (c_int): width of the image in pixels
- `height` (c_int): height of the image in pixels
- `ppx` (c_float): horizontal coordinate of the principal point of the image, as a pixel offset from the left edge
- `ppy` (c_float): vertical coordinate of the principal point of the image, as a pixel offset from the top edge
- `fx` (c_float): focal length of the image plane, as a multiple of pixel width
- `fy` (c_float): focal length of the image plane, as a multiple of pixel height
- `model` (c_int): distortion model of the image
- `coeffs` (c_float*5): distortion coefficients

pyrealsense.utils module

This modules creates utility classes to objects that do not exists in RS API, as well as a wrapper for RS error and its pretty printing.

class `pyrealsense.utils.`**`DeviceOptionRange`**(*option*, *min*, *max*, *step*, *default*)
Bases: `tuple`

`default`
Alias for field number 4

`max`
Alias for field number 2

`min`
Alias for field number 1

`option`
Alias for field number 0

`step`
Alias for field number 3

exception pyrealsense.utils.**RealsenseError**(*function*, *args*, *message*)
Bases: exceptions.Exception

Error thrown during the processing in case the processing chain needs to be exited. Will printout the error message as received from librealsense.

class pyrealsense.utils.**StreamMode**(*stream*, *width*, *height*, *format*, *fps*)
Bases: tuple

format
Alias for field number 3

fps
Alias for field number 4

height
Alias for field number 2

stream
Alias for field number 0

width
Alias for field number 1

pyrealsense.utils.**pp**(*fun*, **args*)
Wrapper for printing char pointer from ctypes.

pyrealsense.constants module

This module extract the RS_API_VERSION to which pyrealsense is binded and wraps several enums from rs.h into classes with the same name.

class pyrealsense.constants.**rs_capabilities**
Bases: object

RS_CAPABILITIES_ADAPTER_BOARD = 7

RS_CAPABILITIES_COLOR = 1

RS_CAPABILITIES_COUNT = 9

RS_CAPABILITIES_DEPTH = 0

RS_CAPABILITIES_ENUMERATION = 8

RS_CAPABILITIES_FISH_EYE = 4

RS_CAPABILITIES_INFRARED = 2

RS_CAPABILITIES_INFRARED2 = 3

RS_CAPABILITIES_MOTION_EVENTS = 5

RS_CAPABILITIES_MOTION_MODULE_FW_UPDATE = 6

name_for_value = {0: 'RS_CAPABILITIES_DEPTH', 1: 'RS_CAPABILITIES_COLOR', 2: 'RS_CAPABILITIES_I

class pyrealsense.constants.**rs_distortion**
Bases: object

RS_DISTORTION_COUNT = 4

RS_DISTORTION_FTHETA = 3

RS_DISTORTION_INVERSE_BROWN_CONRADY = 2

RS_DISTORTION_MODIFIED_BROWN_CONRADY = 1

RS_DISTORTION_NONE = 0

name_for_value = {0: 'RS_DISTORTION_NONE', 1: 'RS_DISTORTION_MODIFIED_BROWN_CONRADY', 2: 'R

class pyrealsense.constants.**rs_format**

Bases: object

RS_FORMAT_ANY = 0

RS_FORMAT_BGR8 = 6

RS_FORMAT_BGRA8 = 8

RS_FORMAT_COUNT = 14

RS_FORMAT_DISPARITY16 = 2

RS_FORMAT_RAW10 = 11

RS_FORMAT_RAW16 = 12

RS_FORMAT_RAW8 = 13

RS_FORMAT_RGB8 = 5

RS_FORMAT_RGBA8 = 7

RS_FORMAT_XYZ32F = 3

RS_FORMAT_Y16 = 10

RS_FORMAT_Y8 = 9

RS_FORMAT_YUYV = 4

RS_FORMAT_Z16 = 1

name_for_value = {0: 'RS_FORMAT_ANY', 1: 'RS_FORMAT_Z16', 2: 'RS_FORMAT_DISPARITY16', 3: 'RS_FO

class pyrealsense.constants.**rs_ivcam_preset**

Bases: object

RS_IVCAM_PRESET_BACKGROUND_SEGMENTATION = 2

RS_IVCAM_PRESET_COUNT = 11

RS_IVCAM_PRESET_DEFAULT = 8

RS_IVCAM_PRESET_FACE_ANALYTICS = 5

RS_IVCAM_PRESET_FACE_LOGIN = 6

RS_IVCAM_PRESET_GESTURE_RECOGNITION = 3

RS_IVCAM_PRESET_GR_CURSOR = 7

RS_IVCAM_PRESET_IR_ONLY = 10

RS_IVCAM_PRESET_LONG_RANGE = 1

RS_IVCAM_PRESET_MID_RANGE = 9

RS_IVCAM_PRESET_OBJECT_SCANNING = 4

RS_IVCAM_PRESET_SHORT_RANGE = 0

name_for_value = {0: 'RS_IVCAM_PRESET_SHORT_RANGE', 1: 'RS_IVCAM_PRESET_LONG_RANGE', 2: 'R

class pyrealsense.constants.**rs_option**

Bases: object

RS_OPTION_COLOR_BACKLIGHT_COMPENSATION = 0

RS_OPTION_COLOR_BRIGHTNESS = 1

RS_OPTION_COLOR_CONTRAST = 2

RS_OPTION_COLOR_ENABLE_AUTO_EXPOSURE = 10

RS_OPTION_COLOR_ENABLE_AUTO_WHITE_BALANCE = 11

RS_OPTION_COLOR_EXPOSURE = 3

RS_OPTION_COLOR_GAIN = 4

RS_OPTION_COLOR_GAMMA = 5

RS_OPTION_COLOR_HUE = 6

RS_OPTION_COLOR_SATURATION = 7

RS_OPTION_COLOR_SHARPNESS = 8

RS_OPTION_COLOR_WHITE_BALANCE = 9

RS_OPTION_COUNT = 68

RS_OPTION_F200_ACCURACY = 13

RS_OPTION_F200_CONFIDENCE_THRESHOLD = 16

RS_OPTION_F200_DYNAMIC_FPS = 17

RS_OPTION_F200_FILTER_OPTION = 15

RS_OPTION_F200_LASER_POWER = 12

RS_OPTION_F200_MOTION_RANGE = 14

RS_OPTION_FISHEYE_AUTO_EXPOSURE_ANTIFLICKER_RATE = 62

RS_OPTION_FISHEYE_AUTO_EXPOSURE_MODE = 61

RS_OPTION_FISHEYE_AUTO_EXPOSURE_PIXEL_SAMPLE_RATE = 63

RS_OPTION_FISHEYE_AUTO_EXPOSURE_SKIP_FRAMES = 64

RS_OPTION_FISHEYE_ENABLE_AUTO_EXPOSURE = 60

RS_OPTION_FISHEYE_EXPOSURE = 56

RS_OPTION_FISHEYE_EXTERNAL_TRIGGER = 59

RS_OPTION_FISHEYE_GAIN = 57

RS_OPTION_FISHEYE_STROBE = 58

RS_OPTION_FRAMES_QUEUE_SIZE = 65

RS_OPTION_HARDWARE_LOGGER_ENABLED = 66

RS_OPTION_R200_AUTO_EXPOSURE_BOTTOM_EDGE = 43

RS_OPTION_R200_AUTO_EXPOSURE_BRIGHT_RATIO_SET_POINT = 38

RS_OPTION_R200_AUTO_EXPOSURE_KP_DARK_THRESHOLD = 41

RS_OPTION_R200_AUTO_EXPOSURE_KP_EXPOSURE = 40

RS_OPTION_R200_AUTO_EXPOSURE_KP_GAIN = 39

RS_OPTION_R200_AUTO_EXPOSURE_LEFT_EDGE = 44

RS_OPTION_R200_AUTO_EXPOSURE_MEAN_INTENSITY_SET_POINT = 37

RS_OPTION_R200_AUTO_EXPOSURE_RIGHT_EDGE = 45

RS_OPTION_R200_AUTO_EXPOSURE_TOP_EDGE = 42

RS_OPTION_R200_DEPTH_CLAMP_MAX = 34

RS_OPTION_R200_DEPTH_CLAMP_MIN = 33

RS_OPTION_R200_DEPTH_CONTROL_ESTIMATE_MEDIAN_DECREMENT = 46

RS_OPTION_R200_DEPTH_CONTROL_ESTIMATE_MEDIAN_INCREMENT = 47

RS_OPTION_R200_DEPTH_CONTROL_LR_THRESHOLD = 55

RS_OPTION_R200_DEPTH_CONTROL_MEDIAN_THRESHOLD = 48

RS_OPTION_R200_DEPTH_CONTROL_NEIGHBOR_THRESHOLD = 54

RS_OPTION_R200_DEPTH_CONTROL_SCORE_MAXIMUM_THRESHOLD = 50

RS_OPTION_R200_DEPTH_CONTROL_SCORE_MINIMUM_THRESHOLD = 49

RS_OPTION_R200_DEPTH_CONTROL_SECOND_PEAK_THRESHOLD = 53

RS_OPTION_R200_DEPTH_CONTROL_TEXTURE_COUNT_THRESHOLD = 51

RS_OPTION_R200_DEPTH_CONTROL_TEXTURE_DIFFERENCE_THRESHOLD = 52

RS_OPTION_R200_DEPTH_UNITS = 32

RS_OPTION_R200_DISPARITY_MULTIPLIER = 35

RS_OPTION_R200_DISPARITY_SHIFT = 36

RS_OPTION_R200_EMITTER_ENABLED = 31

RS_OPTION_R200_LR_AUTO_EXPOSURE_ENABLED = 28

RS_OPTION_R200_LR_EXPOSURE = 30

RS_OPTION_R200_LR_GAIN = 29

RS_OPTION_SR300_AUTO_RANGE_ENABLE_LASER = 19

RS_OPTION_SR300_AUTO_RANGE_ENABLE_MOTION_VERSUS_RANGE = 18

RS_OPTION_SR300_AUTO_RANGE_LOWER_THRESHOLD = 27

RS_OPTION_SR300_AUTO_RANGE_MAX_LASER = 24

RS_OPTION_SR300_AUTO_RANGE_MAX_MOTION_VERSUS_RANGE = 21

RS_OPTION_SR300_AUTO_RANGE_MIN_LASER = 23

RS_OPTION_SR300_AUTO_RANGE_MIN_MOTION_VERSUS_RANGE = 20

RS_OPTION_SR300_AUTO_RANGE_START_LASER = 25

RS_OPTION_SR300_AUTO_RANGE_START_MOTION_VERSUS_RANGE = 22

RS_OPTION_SR300_AUTO_RANGE_UPPER_THRESHOLD = 26

RS_OPTION_TOTAL_FRAME_DROPS = 67

name_for_value = {0: 'RS_OPTION_COLOR_BACKLIGHT_COMPENSATION', 1: 'RS_OPTION_COLOR_BRIG

class pyrealsense.constants.**rs_stream**

Bases: object

RS_STREAM_COLOR = 1

RS_STREAM_COLOR_ALIGNED_TO_DEPTH = 7

RS_STREAM_COUNT = 12

RS_STREAM_DEPTH = 0

RS_STREAM_DEPTH_ALIGNED_TO_COLOR = 9

RS_STREAM_DEPTH_ALIGNED_TO_INFRARED2 = 11

RS_STREAM_DEPTH_ALIGNED_TO_RECTIFIED_COLOR = 10

RS_STREAM_FISHEYE = 4

RS_STREAM_INFRARED = 2

RS_STREAM_INFRARED2 = 3

RS_STREAM_INFRARED2_ALIGNED_TO_DEPTH = 8

RS_STREAM_POINTS = 5

RS_STREAM_RECTIFIED_COLOR = 6

name_for_value = {0: 'RS_STREAM_DEPTH', 1: 'RS_STREAM_COLOR', 2: 'RS_STREAM_INFRARED', 3: 'RS_

CHAPTER 3

Examples

with Matplotlib

```
import logging
logging.basicConfig(level=logging.INFO)

import matplotlib.pyplot as plt

import pyrealsense as pyrs

with pyrs.Service()
    dev = pyrs.Device()

    dev.wait_for_frame()
    plt.imshow(dev.colour)
    plt.show()
```

with OpenCV

```
import logging
logging.basicConfig(level=logging.INFO)

import time
import numpy as np
import cv2
import pyrealsense as pyrs


with pyrs.Service() as serv:
    with serv.Device() as dev:

        dev.apply_ivcam_preset(0)
```

```
        cnt = 0
        last = time.time()
        smoothing = 0.9
        fps_smooth = 30

        while True:

            cnt += 1
            if (cnt % 10) == 0:
                now = time.time()
                dt = now - last
                fps = 10/dt
                fps_smooth = (fps_smooth * smoothing) + (fps * (1.0-smoothing))
                last = now

            dev.wait_for_frames()
            c = dev.color
            c = cv2.cvtColor(c, cv2.COLOR_RGB2BGR)
            d = dev.depth * dev.depth_scale * 1000
            d = cv2.applyColorMap(d.astype(np.uint8), cv2.COLORMAP_RAINBOW)

            cd = np.concatenate((c, d), axis=1)

            cv2.putText(cd, str(fps_smooth)[:4], (0, 50), cv2.FONT_HERSHEY_SIMPLEX, 2,
↪ (0, 0, 0))

            cv2.imshow('', cd)
            if cv2.waitKey(1) & 0xFF == ord('q'):
                break
```

with VTK

```
import time
import threading

import numpy as np

import vtk
import vtk.util.numpy_support as vtk_np


import pyrealsense as pyrs
serv = pyrs.Service()
serv.start()
cam = serv.Device()


class VTKActorWrapper(object):
    def __init__(self, nparray):
        super(VTKActorWrapper, self).__init__()

        self.nparray = nparray

        nCoords = nparray.shape[0]
```

```
        nElem = nparray.shape[1]

        self.verts = vtk.vtkPoints()
        self.cells = vtk.vtkCellArray()
        self.scalars = None

        self.pd = vtk.vtkPolyData()
        self.verts.SetData(vtk_np.numpy_to_vtk(nparray))
        self.cells_npy = np.vstack([np.ones(nCoords,dtype=np.int64),
                                    np.arange(nCoords,dtype=np.int64)]).T.flatten()
        self.cells.SetCells(nCoords,vtk_np.numpy_to_vtkIdTypeArray(self.cells_npy))
        self.pd.SetPoints(self.verts)
        self.pd.SetVerts(self.cells)

        self.mapper = vtk.vtkPolyDataMapper()
        self.mapper.SetInputDataObject(self.pd)

        self.actor = vtk.vtkActor()
        self.actor.SetMapper(self.mapper)
        self.actor.GetProperty().SetRepresentationToPoints()
        self.actor.GetProperty().SetColor(0.0,1.0,0.0)

    def update(self, threadLock, update_on):
        thread = threading.Thread(target=self.update_actor, args=(threadLock, update_
↪on))
        thread.start()

    def update_actor(self, threadLock, update_on):
        while (update_on.is_set()):
            time.sleep(0.01)
            threadLock.acquire()
            cam.wait_for_frames()
            self.nparray[:] = cam.points.reshape(-1,3)
            self.pd.Modified()
            threadLock.release()


class VTKVisualisation(object):
    def __init__(self, threadLock, actorWrapper, axis=True,):
        super(VTKVisualisation, self).__init__()

        self.threadLock = threadLock

        self.ren = vtk.vtkRenderer()
        self.ren.AddActor(actorWrapper.actor)

        self.axesActor = vtk.vtkAxesActor()
        self.axesActor.AxisLabelsOff()
        self.axesActor.SetTotalLength(1, 1, 1)
        self.ren.AddActor(self.axesActor)

        self.renWin = vtk.vtkRenderWindow()
        self.renWin.AddRenderer(self.ren)

        ## IREN
        self.iren = vtk.vtkRenderWindowInteractor()
        self.iren.SetRenderWindow(self.renWin)
        self.iren.Initialize()
```

```
        self.style = vtk.vtkInteractorStyleTrackballCamera()
        self.iren.SetInteractorStyle(self.style)

        self.iren.AddObserver("TimerEvent", self.update_visualisation)
        dt = 30 # ms
        timer_id = self.iren.CreateRepeatingTimer(dt)

    def update_visualisation(self, obj=None, event=None):
        time.sleep(0.01)
        self.threadLock.acquire()
        self.ren.GetRenderWindow().Render()
        self.threadLock.release()


def main():
    update_on = threading.Event()
    update_on.set()

    threadLock = threading.Lock()

    cam.wait_for_frames()
    pc = cam.points.reshape(-1,3)
    actorWrapper = VTKActorWrapper(pc)
    actorWrapper.update(threadLock, update_on)

    viz = VTKVisualisation(threadLock, actorWrapper)
    viz.iren.Start()
    update_on.clear()


main()
cam.stop()
serv.stop()
```

- genindex

Python Module Index

p

Index

P

R

S

W