impro Documentation

Release a

Sebastian Reinhard

Jul 02, 2019

Contents

Impro. is a package for data processing in super-resolution microscopy. It contains high perfomant GPU based visualization and algorithms for 2D and 3D data. Main features:

- Cuda accelerated 2D Alpha Shapes
- Automated image alignment via General Hough Transform
- Huge list of filters for localization datasets
- Customizable Opengl widget based on modelview perspective standard
- Pearson correlation

CHAPTER 1

References

A detailed explantation of the algorithm and it's functionality can be found at:

Reinhard S, Aufmkolk S, Sauer M, Doose S. Registration and Visualization of Correlative Super-Resolution Microscopy Data. Biophys J. 2019 Jun; 116(11) 2073-2078. doi:10.1016/j.bpj.2019.04.029. PMID: 31103233.

CHAPTER 2

Example

Basic usage example. Preprocessing is adapted to the dataset provided in Super-resolution correlator.

```
from impro.data.image_factory import ImageFactory
from impro.analysis.filter import Filter
from impro.analysis.analysis_facade import *

# Read and preprocess SIM image
image = ImageFactory.create_image_file(r"path_to_file.czi")

# Example for image preprocessing
image_array = image.data[:, 3] / 6
image_array = np.clip(image_array[0], 0, 255)
image_array = np.flipud(image_array)
image_array = (image_array).astype("uint8")[0:1400, 0:1400]
image_array = np.fliplr(image_array)

# Read dSTORM data
storm = ImageFactory.create_storm_file(r"path_to_file.txt")

# Preprocess dSTORM point data
indices = Filter.local_density_filter(storm.stormData, 100.0, 18)
storm_data = storm.stormData[indices]
# Render dSTORM data to image
im = create_alpha_shape(storm_data, 130)
col = int(im.shape[1]/200)
row = int(im.shape[0]/200)
source_points, target_points, overlay, results = find_mapping(sim, storm, n_row=row,␣
↪n_col=col)
source_points, target_points = error_management(results, source_points, target_points,
↪ n_row=row)
M = transform.estimate_transform("affine",source_points,target_points)
correlation_index = pearson_correlation(sim, cv2.cvtColor(storm, cv2.COLOR_RGBA2GRAY),
↪ M)
```

2.1 Analysis module

Collection of useful analysis tools. For easy acess use the *Analysis facade*

2.1.1 Analysis facade

Easy access to complex algorithmic concepts.

Author *Sebastian Reinhard*

Organization Biophysics and Biotechnology, Julius-Maximillians-University of Würzburg

Version 2019.06.26

impro.analysis.analysis_facade.**create_alpha_shape**(*storm_data: numpy.ndarray*, *alpha_value: float*, *px_size=32.2*, *line_width=1.0*) → numpy.ndarray

Facade function to render alpha shape of dSTORM data

Image is rendered from (0,0) to the maximal dimension of the dSTORM data.

Parameters

- **storm_data** (*np.ndarray (nx5)*) – Point Cloud data to be rendered
- **px_size** (*float*) – Pixel size for the rendered image in nanometer per pixel
- **alpha_value** (*float*) – Core value to compute the alpha shape in nanometer
- **line_width** (*float*) – Line width of the alpha shape lines in a.u.

Returns **image** – Rendered image

Return type np.array

Example

```
>>> points = np.random.randint(0,32000,(1000000, 2))
>>> image = create_alpha_shape(points, 130)
>>> image.shape
(994,994)
```

impro.analysis.analysis_facade.**create_storm**(*storm_data: numpy.ndarray*, *px_size=32.2*, *size=20*, *cluster=array(3.)*)

Facade function to render an image of dSTORM data.

Image is rendered from (0,0) to the maximal dimension of the dSTORM data.

Parameters

- **storm_data** (*np.array (nx5)*) – Point Cloud data to be rendered
- **px_size** (*float*) – Pixel size for the rendered image in nanometer per pixel
- **size** (*float*) – Size of the rendered points in nanometer
- **cluster** (*np.array (n)*) – Affiliation of the i th point to cluster[i] cluster

Returns **image** – Rendered image

Return type np.array

Example

```
>>> points = np.random.randint(0,32000,(1000000, 5))
>>> image = create_alpha_shape(points, 130)
>>> image.shape
(994,994)
```

impro.analysis.analysis_facade.**error_management**(*result_list: list*, *source_points*, *target_points*, *n_row=5*)

Test the results of th weighted GHT for missmatches

Parameters

- **result_list** (*list*) – results of the Weighted General Hough Transform
- **source_points** (*np.array*) – Source points of affine transformation
- **target_points** (*np.array*) – Target points of affine transformation
- **n_row** (*int*) – number of rows used for the weighted GHT

Returns **source_points, target_points** – Filtered source and target points for affine transformation

Return type np.array

impro.analysis.analysis_facade.**find_mapping**(*target: numpy.ndarray*, *source_color: numpy.ndarray*, *n_col=5*, *n_row=5*, *offset=0*) → numpy.ndarray

Facade function to find a mapping from source_color image to gray scale target image.

Parameters

- **target** (*np.array*) – Target image of affine transformation
- **source_color** (*np.array*) – Source image of affine transformation
- **n_col** (*int*) – number of collums to seperate the source image into
- **n_row** (*int*) – number of rows to seperate the source image into
- **offset** (*int*) – Start seperating the source image at (0+offset,0+offset). Offset is given in pixel

Returns

- **source_points, target_points** (*np.array*) – Source and target points for affine transformation
- **overlay** (*np.array*) – Target image overlayed with source image segments at the matching position
- **results** (*list*) – Results of the Weighted General Hough Transform

Example

```
>>> import cv2
>>> points = np.random.randint(0,32000,(1000000, 5))
>>> source = create_alpha_shape(points, 130)
>>> points = np.random.randint(0,120000,(1000000, 5))
>>> target = create_alpha_shape(points, 130)
>>> find_mapping(cv2.cvtColor(target, cv2.COLOR_RGBA2GRAY), source)
```

impro.analysis.analysis_facade.**pearson_correlation**(*target: numpy.ndarray*, *source: numpy.ndarray*, *map: numpy.ndarray*) → float

Compute pearson correlation index after alignment

Parameters

- **target** (*np.array*) – target image in gray scale
- **source** (*np.array*) – source image in gray scale
- **map** (*sklearn.transformation*) – computed transformation

Returns **source_points, target_points** – Filtered source and target points for affine transformation

Return type np.array

Example

```
>>> import skimage.transform
>>> source = np.ones((1000,1000))
>>> target = np.ones((1000,1000))
>>> source_points = np.array([[1.0,1.0],[500,500],[700,500]])
>>> target_points = source_points
>>> M = transform.estimate_transform("affine",source_points,target_points)
>>> pearson_correlation(target, source, M)
(1.0)
```

2.1.2 Alpha Shape

The objective of the alpha shape algorithm is to deliver a formal meaning for the geometric notation of 'shape', in the context of finite point sets. The straciatella ice cream example gives a visual explanation of the concept: Consider a ball of ice cream with chocolate pieces. The chocolate pieces represent the distribution of points in space (ice cream). The alpha shape algorithm now tries to eat as much ice cream as possible without touching any chocolate pieces, using an arbitrary predefined spoon size, the α-value. Choosing a very small spoon size results in all ice cream being eaten, a very big spoon size in no ice cream being eaten at all (Convex Hull). But a size in between creates a concave hull representing the shape of the distributed chocolate pieces, the desired alpha shape.

References

(1) Edelsbrunner, Herbert ; Mücke, Ernst P.: Three-dimensional Alpha Shapes. In: ACM Trans. Graph. 13 (1994), Januar, Nr. 1, 43–72. http: //dx.doi.org/10.1145/174462.156635. – DOI 10.1145/174462.156635. – ISSN 0730–0301

(2) Fischer, Kaspar: Introduction to Alpha Shapes. https: //graphics.stanford.edu/courses/cs268-11-spring/handouts/ AlphaShapes/as_fisher.pdf.

Author *Sebastian Reinhard*

Organization Biophysics and Biotechnology, Julius-Maximillians-University of Würzburg

Version 2018.03.09

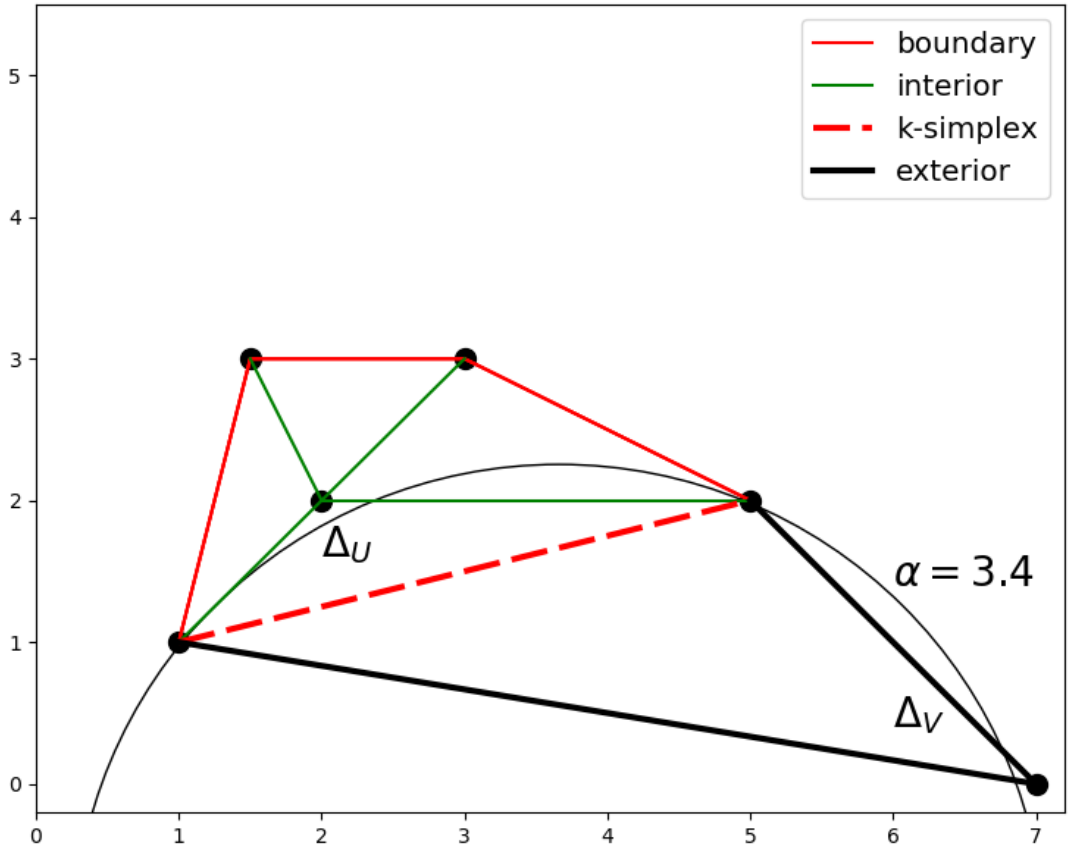

Fig. 1: Considering the red dotted line l, the limit a is defined as the minimum of the diameter of the circumcircles around $\triangle_U$ and $\triangle_V$. The limit b is the maximum of circumcircles around $\triangle_U$ and $\triangle_V$. Since the α-ball is smaller than b, but bigger than a, l is classified as boundary.

Example

```
>>> data = np.random.randint(0,32000,(1000000, 2))
>>> k_simplices = get_k_simplices(data[...,0:2])[0]
```

class impro.analysis.alpha_shape_gpu.**AlphaComplex**(*struct_ptr: int*, *indices: numpy.ndarray*, *points: numpy.ndarray*, *neighbors: numpy.ndarray*)

Bases: object

Derive 2D alpha complex from scipy.spatial Delaunay

Class to create a alpha complex structure on GPU.

Parameters

- **struct_ptr** (*int*) – pointer to allocated memmory for structure
- **indices** (*np.array*) – nx3 array containing point indices of the delaunay triangulation simplices
- **neighbors** (*np.array*) – nx3 array containing the indices of neighboring simplices
- **points** (*np.array*) – nx2 array of the points used for delaunay triangulation

get()

Returns nx5 of d=1 simplices containing: [index1, index2, dist, sigma1, sigma2] with sigma 1 < sigma 2

Return type numpy.array

memsize = 32

merge()

impro.analysis.alpha_shape_gpu.**get_k_simplices**(*points: numpy.ndarray*)

Parameters **points** (*np.array*) – nx2 array of points to use for alpha complex

Returns

- **alpha complex** (*mx5 array of d=1 simplices containing the upper and lower limits for a simplice to be interior/*)
- *on boundary of the alpha shape.*

2.1.3 Filter

Author *Sebastian Reinhard*

Organization Biophysics and Biotechnology, Julius-Maximillians-University of Würzburg

Version 2019.06.26

class impro.analysis.filter.**Filter**

Bases: object

static local_density_filter(*points*, *r*, *n*)

Get the indices of all points having a minimum of n neighbors in a radius of r

Parameters

- **points** (*np.array*) – Points to filter (X,Y,Z)

- **r** (*float*) – Search for neighbors in a radius of r
- **n** (*int*) – Required neighbors in radius r to pass the filter

Returns **filt** – indices of entries passing the filter

Return type np.array

static values_filter(*value*, *minimum*, *maximum*)
Get the indices of an array values with value > minimum and value < maximum

Parameters

- **value** (*np.array*) – Array with values to filter
- **minimum** (*float*) – minimum value to pass the filter
- **maximum** (*float*) – maximum value to pass the filter

Returns **filt** – indices of entries passing the filter

Return type np.array

2.1.4 Hough Transform

The General Hough Transform (GHT) maps the orientation of edge points in a template image to a predefined origin (typically the middle pixel of the image). Comparing this map with an image gives each point a rating of likelihood for being a source point of that map.

Author *Sebastian Reinhard*

Organization Biophysics and Biotechnology, Julius-Maximillians-University of Würzburg

Version 2019.06.26

Example

```
>>> target = cv2.imread("path_to_file")
>>> template = cv2.imread("path_to_file")
>>> ght_target = GHTImage(target, blur=5, canny_lim=(130,180))
>>> H = HoughTransform()
>>> H.target = ght_target
>>> ght_template = TemplateImage(template)
>>> H.template = ght_template
>>> res = H.transform()
```

class impro.analysis.hough_transform_gpu.**GHTImage**(*image*, *blur=4*, *canny_lim=(130, 200)*)

Bases: object

GHT specific image preprocessing

o_image
Input image

Type np.array

image
Resized and blurred input image

Type np.array

canny
Canny Edge processed image

Type np.array

gradient
Gradient image

Type np.array

class impro.analysis.hough_transform_gpu.**HoughTransform**(*rotation_min=-10*, *rotation_max=10*)
Bases: object

Perform Gradient Weighted General Hough Transform on the Graphics Processing Unit

rotation_min
Start matching template at rotation_min

Type float

rotation_max
End matching template at rotation_max

Type float

template
Matching template image on target image

Type np.array

target
Matching template image on target image

Type np.array

weighted
Weight the GHT by Gradient density

Type bool

transform()
Perform GHT algorithm with given data

rotation_max

rotation_min

target

template

transform()

class impro.analysis.hough_transform_gpu.**Matrix**(*array*)
Bases: object

Wrapper class for matrix and matrixf struct on GPU:

get()

class impro.analysis.hough_transform_gpu.**TemplateImage**(*image*, ***kwargs*)
Bases: *impro.analysis.hough_transform_gpu.GHTImage*

Extend GHTImage by templte properties

o_image
Input image

Type np.array

image
Resized and blurred input image

Type np.array

canny
Canny Edge processed image

Type np.array

gradient
Gradient image

Type np.array

r_matrix_zero
Vector mapping of edge points to a predefined origin

Type np.array

r_matrix
Rotated r_matrix_zero

Type np.array

rotate_r_matrix(*angle*)
Rotate R-Matrix by angle rad

angle: float Angle to rotate matrix in rad

2.1.5 Module contents

2.2 Data module

2.2.1 Image Factory

Author *Sebastian Reinhard*

Organization Biophysics and Biotechnology, Julius-Maximillians-University of Würzburg

Version 2019.06.26

class impro.data.image_factory.**ImageFactory**
Bases: object

Interface to File readers

classmethod create_image_file(*path: str*)

classmethod create_storm_file(*path: str*)

2.2.2 Image

Author *Sebastian Reinhard*

Organization Biophysics and Biotechnology, Julius-Maximillians-University of Würzburg

Version 2019.06.26

class impro.data.image.**ImageReader**(*path*)
Bases: object

Read Image file

Possible formats are .czi, .lsm, .tiff, .png

path
Path to file

Type str

data
Image data as array

Type np.array

metaData
Meta Data of image file containing computional dimensions: {ShapeSizeX, ShapeSizeY, ShapeSizeZ, ShapeSizeC} and physical dimensions: SizeX(Pixel Size in X direction)...

Type np.array

Example

```
>>> file = ImageReader("path_to_file.czi")
>>> file.parse()
>>> file.data.shape
(4,103,2430,2430)
```

parse(*calibration_px=1.0*)
Read data/Metadata from file

reset_data()
resets data

set_calibration(*px: float*)
Set pixel size manually

class impro.data.image.**LocalisationReader**(*path: str*)
Bases: object

Read RapidSTORM files

Rewritten class to read STORM/dSTORM data from a text file in a numpy array. Improved performance and logic. Can transform the localizations if a matrix is given. Recives metadata from file header.

path
Path to Rapidstorm .txt file

Type str

stormData
Restructured data: X, Y, Precision, Z, Emission, Frame #todo switch Precision and Z

Type np.array

size
X,Y size of data in nanometer

Type np.array

Example

```
>>> file = LocalisationReader("path_to_Rapidstormfile.txt")
>>> file.parse()
>>> data = file.stormData
>>> data.shape
(100000, 6)
```

parse()
Read dSTORM data/Metadata from file

reset_data()
Resets dSTORM data

transformAffine(*path=None*, *src=None*, *dst=None*)
Transform dSTORM data affine with Landmarks file

Parameters

- **path** (*str, optional*) – path to file containing source points in column 1,2 (starting by 0) and target points in column 3,4
- **src** (*(,2) ndarray*) – X,Y coordinates of source points
- **dst** (*(,2) ndarray*) – X,Y coordinates of destination points

class impro.data.image.**StormReader**(*input_loc_path*)
Bases: object

Internal Class. Find dSTORM columns with regular expressions

clear()

get_header_info()

readfile()

save_loc_file(*path*, *file*)

2.2.3 Module contents

2.3 Render module

2.3.1 Common module

Common module

Buffer

A couple of buffer objects for GPU rendering.

class impro.render.common.Buffer.**Framebuffer**
Bases: object

build(*renderbuf*, *texbuf*)

delete()

class impro.render.common.Buffer.**Renderbuffer**
Bases: object

build(*width*, *height*)

delete()

class impro.render.common.Buffer.**Texturebuffer**
Bases: object

build(*width*, *height*)

delete()

Objects

A couple of mesh objects for GPU rendering.

class impro.render.common.Objects.**Cube**
Bases: object

class impro.render.common.Objects.**Quad**
Bases: object

class impro.render.common.Objects.**Surface**
Bases: object

class impro.render.common.Objects.**Texture**
Bases: object

class impro.render.common.Objects.**UnitCube**
Bases: object

Surface

A couple of texture mappings for GPU rendering.

class impro.render.common.Surface.**CSurface**
Bases: object

build_surface(*width*, *height*, *numComponents*, *numTargets*)

class impro.render.common.Surface.**Surface**
Bases: object

class impro.render.common.Surface.**Volume**(*width*, *height*, *depth*)
Bases: object

Textures

class impro.render.common.Textures.**Create2DTexture**
Bases: object

set_texture(*data*, *interpolation*)

class impro.render.common.Textures.**Create3DTexture**
Bases: object

set_texture(*data*)

class impro.render.common.Textures.**CreateNoise**
Bases: object

get_texture_handle()

set_texture(*width*, *height*)

class impro.render.common.Textures.**CreateVolumeTexture**
Bases: object

set_texture(*data*)

Module contents

2.3.2 Render

Render multiple objects on GPU(points, image2D, image3D)

Author *Sebastian Reinhard*

Organization Biophysics and Biotechnology, Julius-Maximillians-University of Würzburg

Version 2019.06.26

2.3.3 Shader

Create and compile a program from vertex, fragment and geometry shader

Author *Sebastian Reinhard*

Organization Biophysics and Biotechnology, Julius-Maximillians-University of Würzburg

Version 2019.06.26

class impro.render.shader.**shader**(*filename*)
Bases: object

Parameters **filepath** – path to vertex fragment and (optional) geometry shaders. should all have the same name

Variables **m_program** – reference number to program

attribute_loc(*name*)

create_shader(*shader*, *type*)
Compile OpenGL shaders string and check for shaders errors :param shader: String of shaders :param type: Type of shaders can be: GL_VERTEX_SHADER, GL_FRAGMENT_SHADER, GL_GEOMETRY_SHADER :return: Shader ID

enums

link_program()
Link program :raises RuntimeError:

load_shader(*filename*)
Load shaders string from file :param filename: Full path to shaders :return: File as string

set_uniform(*name*, *data*)

uniform_loc(*name*)

validate_program()
Validate program :raises RuntimeError:

2.3.4 Module contents

2.4 Impro

2.4.1 impro package

Subpackages

impro.libs package

Submodules

impro.libs.czifile module

Read image and metadata from Carl Zeiss(r) ZISRAW (CZI) files.

CZI is the native image file format of the ZEN(r) software by Carl Zeiss Microscopy GmbH. It stores multidimensional images and metadata from microscopy experiments.

Author Christoph Gohlke

Organization Laboratory for Fluorescence Dynamics, University of California, Irvine

Version 2017.09.12

Requirements

- CPython 3.6 64-bit
- Numpy 1.13
- Scipy 0.19
- Tifffile.py 2017.09.12
- Czifle.pyx 2017.07.20 (for decoding JpegXrFile and JpgFile images)

Revisions

2017.09.12 Require tifffile.py 2017.09.12

2017.07.21 Use multi-threading in CziFile.asarray to decode and copy segment data. Always convert BGR to RGB. Remove bgr2rgb options. Decode JpegXR directly from byte arrays.

2017.07.13 Add function to convert CZI file to memory-mappable TIFF file.

2017.07.11 Add 'out' parameter to CziFile.asarray. Remove memmap option from CziFile.asarray (backwards incompatible). Change spline interpolation order to 0 (backwards incompatible). Make axes return a string. Require tifffile 2017.07.11.

2015.08.17 Require tifffile 2015.08.17.

2014.10.10 Read data into a memory mapped array (optional).

2013.12.04 Decode JpegXrFile and JpgFile via _czifle extension module. Attempt to reconstruct tiled mosaic images.

2013.11.20 Initial release.

Notes

The API is not stable yet and might change between revisions.

The file format design specification [1] is confidential and the licence agreement does not permit to write data into CZI files.

Only a subset of the 2012 specification is implemented in the initial release. Specifically, multifile images are not yet supported.

Tested on Windows with a few example files only.

References

(1) ZISRAW (CZI) File Format Design specification Release Version 1.2.2. CZI 07-2016/CZI-DOC ZEN 2.3/DS_ZISRAW-FileFormat.pdf (confidential). Documentation can be requested at <http://microscopy.zeiss.com/microscopy/en_us/downloads/zen.html>

(2) CZI The File Format for the Microscope | ZEISS International <http://microscopy.zeiss.com/microscopy/en_us/products/microscope-software/ zen-2012/czi.html>

Examples

```
>>> with CziFile('test.czi') as czi:
...     image = czi.asarray()
>>> image.shape
(3, 3, 3, 250, 200, 3)
>>> image[0, 0, 0, 0, 0]
array([10, 10, 10], dtype=uint8)
```

impro.libs.czifile.**imread**(*filename*, **args*, ***kwargs*)

Return image data from CZI file as numpy array.

'args' and 'kwargs' are arguments to the CziFile.asarray function.

Examples

```
>>> image = imread('test.czi')
>>> image.shape
(3, 3, 3, 250, 200, 3)
>>> image.dtype
dtype('uint8')
```

class impro.libs.czifile.**CziFile**(*arg*, *multifile=True*, *filesize=None*, *detectmosaic=True*)

Bases: object

Carl Zeiss Image (CZI) file.

header

Global file metadata such as file version and GUID.

Type FileHeaderSegment

metadata
Global image metadata in UTF-8 encoded XML format.

Type etree.ElementTree.Element

All attributes are read-only.

asarray(*resize=True*, *order=0*, *out=None*, *max_workers=None*)
Return image data from file(s) as numpy array.

Parameters

- **resize** (*bool*) – If True (default), resize sub/supersampled subblock data.
- **order** (*int*) – The order of spline interpolation used to resize sub/supersampled subblock data. Default is 0 (nearest neighbor).
- **out** (*numpy.ndarray, str, or file-like object; optional*) – Buffer where image data will be saved. If numpy.ndarray, a writable array of compatible dtype and shape. If str or open file, the file name or file object used to create a memory-map to an array stored in a binary file on disk.
- **max_workers** (*int*) – Maximum number of threads to read and decode subblock data. By default up to half the CPU cores are used.

attachment_directory
Attribute whose value is computed on first access.

attachments()
Return iterator over all Attachment segments in file.

axes
Attribute whose value is computed on first access.

close()

dtype
Attribute whose value is computed on first access.

filtered_subblock_directory
Attribute whose value is computed on first access.

metadata
Attribute whose value is computed on first access.

save_attachments(*directory=None*)
Save all attachments to files.

segments(*kind=None*)
Return iterator over Segment data of specified kind.

Parameters **kind** (*bytestring or sequence thereof*) – Segment id(s) as listed in SEGMENT_ID. If None (default), all segments are returned.

shape
Attribute whose value is computed on first access.

start
Attribute whose value is computed on first access.

subblock_directory
Attribute whose value is computed on first access.

subblocks()
Return iterator over all SubBlock segments in file.

Module contents

Module contents

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

i

Index

G

H

I

L

M

O

P

Q

R

S

T

U

V

W