Qiber3D

Release 0.7.0

Hagen Eckert, Anna Jaeschke

Oct 24, 2021

CONTENTS

Automated quantification of fibrous networks

`Qiber3D` is an open software toolkit for analysing fibrous networks in 3D. This documentation is for version 0.7.0.

CHAPTER

ONE

SETUP

1.1 Binder

You can test `Qiber3D` without installing a Python environment on your computer. This is possible through the binder service, allowing us to start a fully set up JupyterLab in your browser. It is important to note that the provided environments do not have enough memory to work with full resolution image stacks. The included examples should still be helpful to start with `Qiber3D`.

To start the JupyterLab click here!

1.2 Library

You can install the `Qiber3D` library directly through the Python Package Index (PyPI). The use of a virtual environment is recommended.

```
$ pip install Qiber3D
```

If the stable version of Qiber3D on PyPI is missing a particular function, you can install the latest version directly from the GitHub repository.

```
$ pip install -U git+https://github.com/theia-dev/Qiber3D.git#egg=Qiber3D
```

1.3 Development and documentation

To work with the source code clone the repository from GitHub and install the requirements. If you add improvements to the code a pull request would be welcomed. The source code is accompanied by the documentation and a collection of test cases.

```
$ git clone https://github.com/theia-dev/Qiber3D.git
$ python3 -m venv Qiber3D_env
$ . Qiber3D_env/bin/activate
(Qiber3D_env) $ pip install --upgrade pip
(Qiber3D_env) $ pip install -r Qiber3D/requirements.txt
```

Building the documentation locally needs a few extra python packages. They can also be installed in the same virtual environment with the following command.

```
(Qiber3D_env) $ pip install -r Qiber3D/docs/requirements.txt
```

The HTML version of the documentation can then be built:

```
(Qiber3D_env) $ sphinx-build -b html Qiber3D/docs Qiber3D/docs_html
```

The tests are located under `Qiber3D/tests` can be started with:

```
(Qiber3D_env) $ python -m unittest discover -s Qiber3D/tests
```

CHAPTER

TWO

USAGE

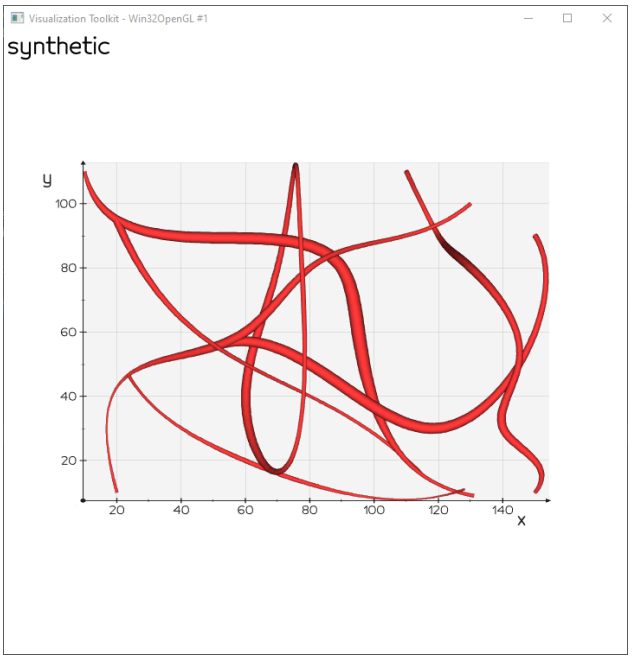
2.1 Visualization

To explore some of the possibilities of the *Qiber3D.Render* module we can use the synthetic network as example.

```
>>> from Qiber3D import IO
>>> net = IO.load.synthetic_network()
>>> print(net)
Input file: memory
  Number of fibers: 4 (clustered 2)
  Number of segments: 11
  Number of branch points: 5
  Total length: 1141.44
  Total volume: 4688.67
  Average radius: 0.936
  Cylinder radius: 1.143
  Bounding box volume: 806162
```

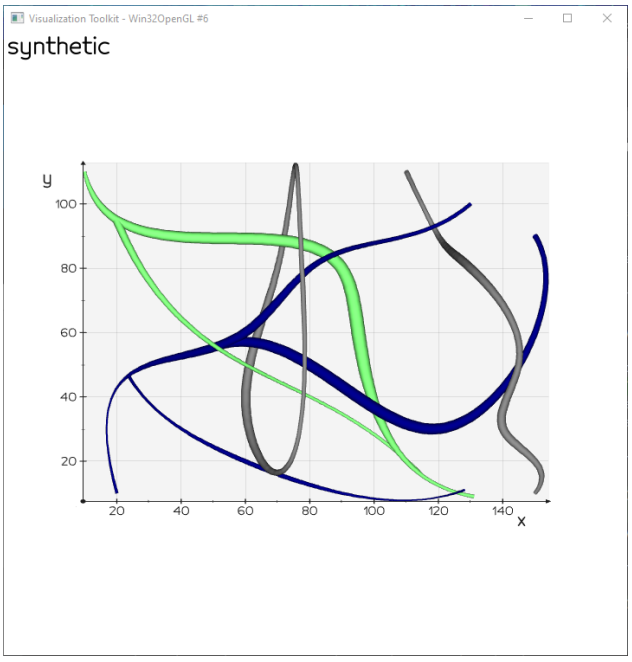
A loaded network can be visualized in different ways. Calling *render.show()* gives a quick view of the network.

```
net.render.show()
```

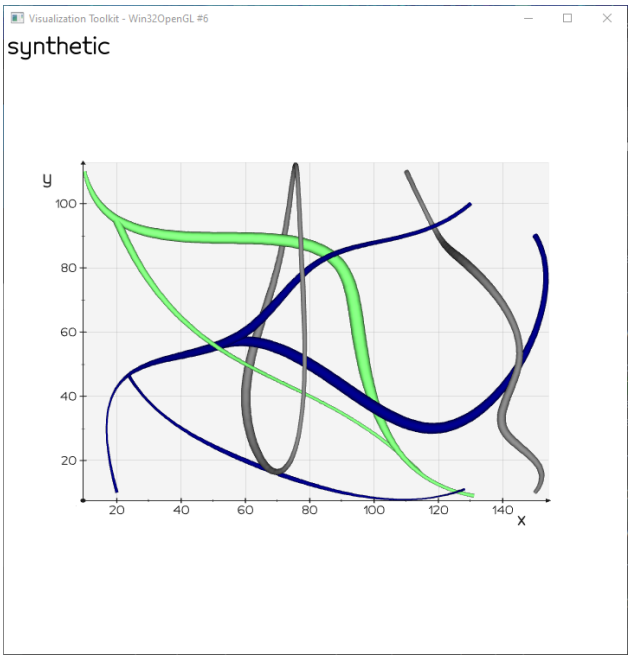


The `color_mode` parameter changes how the different segments (*`Qiber3D.Segment`*) are represented. Selecting `'fiber'` randomly colors fibers (*`Qiber3D.Fiber`*) that have at least one branch point. Fibers without a branch point are grey. With `'segment'` all segments are colored randomly. The full list of possible `color_mode` parameter is documented with *`Qiber3D.Render.show()`*.

```
net.render.show(color_mode='fiber')
```

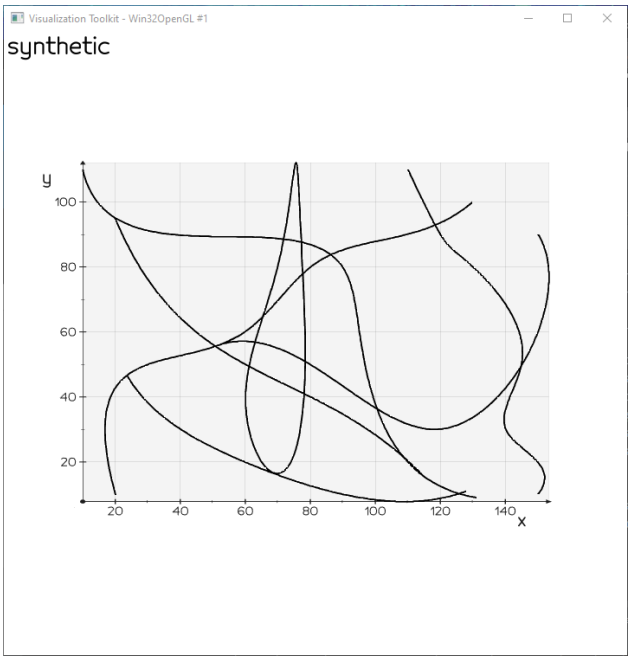



```
net.render.show(color_mode='segment')
```



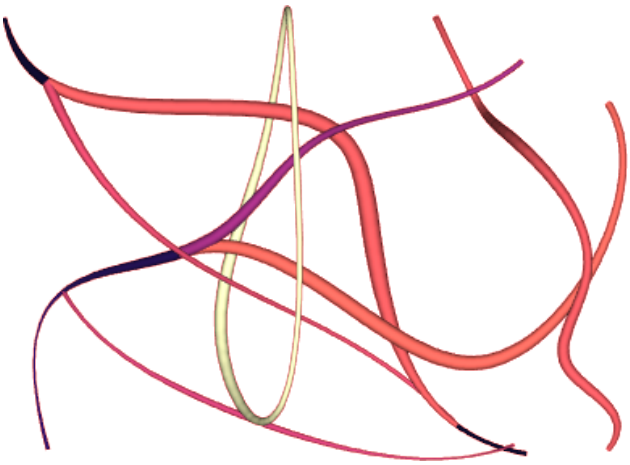
Sometimes it can be helpful to display just the reconstructed center-lines of a network. To archive this the parameter `object_type` can be set to `'line'`.

```
net.render.show(color_mode='flat', color=(0, 0, 0), object_type='line')
```



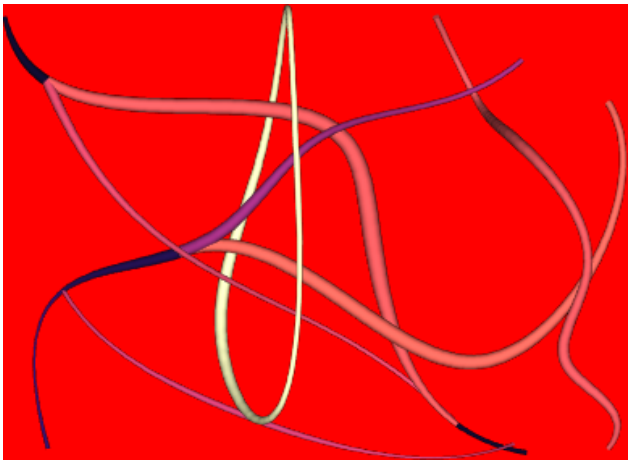
While interactive representations are helpful when inspecting a small number of networks, it is more effective to create different views of the network as rendered images. For this purpose *Qiber3D.Render.overview()* can be used. The syntax is very similar to *Qiber3D.Render.show()*, but now a `out_path` and the `image_resolution` can be set. If no `out_path` is set the file name is automatically chosen. An existing file will not be overwritten. Set `overwrite` to `True` to change this behaviour.

```
>>> net.render.overview(color_mode='segment_length', color_map='magma', background='red')
Qiber3D_render [INFO] New overview saved under: overview_segment_length_synthetic.png
>>> net.render.overview(color_mode='segment_length', color_map='magma', background='red')
Qiber3D_helper [WARNING] File exist: overview_segment_length_synthetic.png
```



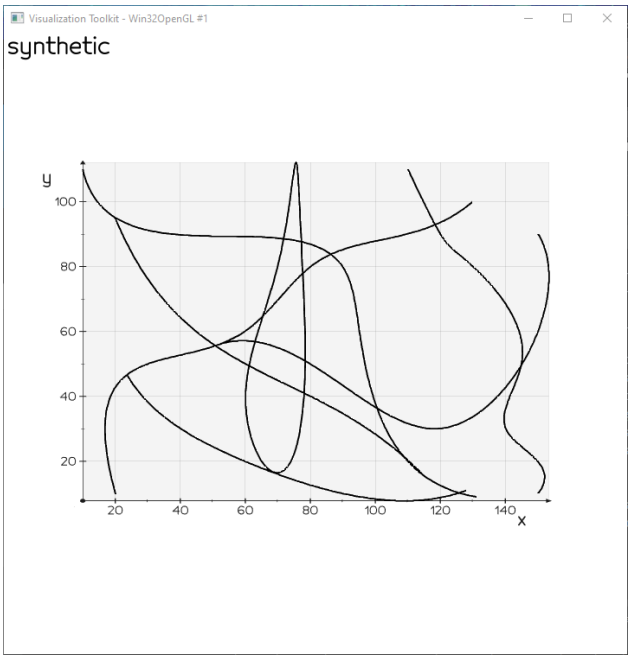
While we requested a red background, it is not visible in the resulting image. The reason for this behavior is, that for the background the alpha channel of the `.png` file comes into play. If a image without transperency is needed `rgba` can be set to `False`.

```
>>> net.render.overview(color_mode='segment_length', color_map='magma', background='red',
↪ rgba=False, overwrite=True)
Qiber3D_render [INFO] New overview saved under: overview_segment_length_synthetic.png
```



The last basic visualization option is to save an animation of the network as a `.mp4` movie. (`.gif` and `.webm` are also possible)

```
>>> net.render.animation(color_mode='segment', color_map='hsv', duration=4,␣
↪background=(1.0, 1.0, 1.0))
Qiber3D_render [INFO] Preparing animation
rendering: 100%|| 120/120 [00:06<00:00, 17.94frame/s]
Qiber3D_render [INFO] New animation saved under: animation_segment_synthetic.mp4
```



2.2 In- and output

A *Qiber3D.Network* can either be created from from an image stack, or an already reconstructed source like a `.mv3d` or `.swc` file. To load the network simply pass its path to *Qiber3D.Network.load()*. Based on the file suffix the corresponding part of *Qiber3D.IO.load* is used.

Note: The examples presented here use the corresponding image stacks from figshare doi:10.6084/m9.figshare.13655606. The *Qiber3D.helper.Example* class can be used to download the data easily.

- `microvascular_network.nd2` - FigID: 26211077 (1.06 GB)
- `microvascular_network.tif` - FigID: 30771817 (1.06 GB)
- `microvascular_network-C2.tif` - FigID: 30771877 (362 MB)
- `microvascular_network-C2-reduced.tif` - FigID: 31106104 (40MB)

Note: To directly download the files on the commandline `curl` can be used. Just replace `$(FigID)` with the number from the list above.

```
curl -X GET "https://api.figshare.com/v2/file/download/$(FigID)"
```

2.2.1 ND2 example

```
>>> import logging
>>> from Qiber3D import Network, config
>>> from Qiber3D.helper import Example, change_log_level
>>> config.log_level = change_log_level(logging.DEBUG)  # This will show 3D renderings␣
↪of the intermediate steps
>>> config.extract.nd2_channel_name = 'FITC'
>>> net_ex = Example.nd2()
>>> net = Network.load(net_ex)
Qiber3D_extract [INFO] Load image data from microvascular_network.nd2
Qiber3D_extract [INFO] Image voxel size: [1.230,1.230,2.500]
Qiber3D_extract [INFO] Median Filter (despeckle)
Qiber3D_extract [INFO] Z-Drop correction
Qiber3D_extract [INFO] Resample image to cubic voxels
Qiber3D_extract [INFO] Apply gaussian filter
Qiber3D_extract [INFO] Generate binary representation
Qiber3D_extract [INFO] Binary representation used a threshold of 4.9% (otsu)
Qiber3D_extract [INFO] Morph binary representation
Qiber3D_extract [INFO] reconstruct image
Qiber3D_reconstruct [INFO] Skeletonize image by thinning
Qiber3D_reconstruct [INFO] Euclidean distance transformation
Link up skeleton: 100%|| 13/13 [00:15<00:00,  1.17s/it]
Qiber3D_reconstruct [INFO] Build Qiber3D.Network from the raw graph
Qiber3D_reconstruct [INFO] Cleaning Network
Qiber3D_reconstruct [INFO] Smooth Segments
>>> print(net)
Input file: microvascular_network.nd2
  Number of fibers: 405 (clustered 109)
  Number of segments: 966
  Number of branch points: 327
  Total length: 43192.45
  Total volume: 5146410.65
  Average radius: 5.770
  Cylinder radius: 6.158
  Bounding box volume: 683663872
```

A reconstructed network can quickly be saved with *Qiber3D.Network.save()*. Without arguments, just the reconstructed network is saved. If the image stack at the different stages should be preserved set `save_steps to True. Saving the reconstruction steps can consume a lot of space.

```
>>> net.save(save_steps=True)
Qiber3D_core [INFO] Network saved to microvascular_network.qiber
```

The created `.qiber` can be loaded as any other supported file type.

```
>>> net = Network.load('microvascular_network.qiber')
```

2.2.2 TIFF example

The same example can be run using a *.tif* version of the same data.

Note: While the reduced dataset is the same subject, the reduced resolution and greyscale range will result in altered results.

```
>>> from Qiber3D import Network, config
>>> from Qiber3D.helper import Example
>>> config.extract.voxel_size = [1.2302, 1.2302, 2.5]
>>> config.extract.low_memory = True
>>> net_ex = Example.tiff()
>>> net = Network.load(net_ex, channel=1)
Qiber3D_extract [INFO] Load image data from microvascular_network.tif
Qiber3D_extract [INFO] Image voxel size: [1.230,1.230,2.500]
Qiber3D_extract [INFO] Median Filter (despeckle)
Qiber3D_extract [INFO] Z-Drop correction
Qiber3D_extract [INFO] Resample image to cubic voxels
Qiber3D_extract [INFO] Apply gaussian filter
Qiber3D_extract [INFO] Generate binary representation
Qiber3D_extract [INFO] Binary representation used a threshold of 4.9% (otsu)
Qiber3D_extract [INFO] Morph binary representation
Qiber3D_extract [INFO] reconstruct image
Qiber3D_reconstruct [INFO] Skeletonize image by thinning
Qiber3D_reconstruct [INFO] Euclidean distance transformation
Qiber3D_reconstruct [INFO] Build Qiber3D.Network from the raw graph
Qiber3D_reconstruct [INFO] Cleaning Network
Qiber3D_reconstruct [INFO] Smooth Segments
>>> print(net)
Input file: microvascular_network.tif
  Number of fibers: 406 (clustered 111)
  Number of segments: 971
  Number of branch points: 329
  Total length: 43241.93
  Total volume: 5150823.10
  Average radius: 5.775
  Cylinder radius: 6.158
  Bounding box volume: 683658224
```

A reconstructed network can saved as we did before with *Qiber3D.Network.save()*. The save function can be given a name for the save file.

```
>>> net.save('reconstructed_net.qiber')
Qiber3D_core [INFO] Network saved to reconstructed_net.qiber
```

2.2.3 Synthetic example

The settings for the different image filters can be accessed through the *Qiber3D.config* module. In next example the image to be reconstructed is already cleaned up and just needs to be binarized. As source we will use a rasterized version of the synthetic test network (:meth:'Qiber3D.IO.load.synthetic_network').

```
>>> from Qiber3D import config
>>> from Qiber3D import Network, IO
>>> syn_net = IO.load.synthetic_network()
# set up the resolution of the synthetic network
>>> voxel_resolution = 5
>>> config.extract.voxel_size = [1 / voxel_resolution] * 3
# switching off unneeded filters,
# as the synthetic network is already cleaned
>>> config.extract.morph.apply = False
>>> config.extract.median.apply = False
>>> config.extract.smooth.apply = False
>>> config.extract.z_drop.apply = False
# set a specific threshold to preserve radius
>>> config.extract.binary.threshold = 29
# export the synthetic network as .tif file
>>> syn_net.export('synthetic.tif', voxel_resolution=voxel_resolution, overwrite=True)
Qiber3D_render [INFO] Rasterizing network (voxel resolution : 5.00E+00 voxel/unit)
Qiber3D_core [INFO] Network exported to synthetic.tif.
# import .tif file
>>> net = Network.load('synthetic.tif')
Qiber3D_render [INFO] Rasterizing network (voxel resolution : 5.00E+00 voxel/unit)
Qiber3D_core [INFO] Network exported to synthetic.tif
Qiber3D_extract [INFO] Load image data from synthetic.tif
Qiber3D_extract [INFO] Image voxel size: [0.200,0.200,0.200]
Qiber3D_extract [INFO] Resample image to cubic voxels
Qiber3D_extract [INFO] Generate binary representation
Qiber3D_extract [INFO] Binary representation used a threshold of 29.0% (direct)
Qiber3D_extract [INFO] reconstruct image
Qiber3D_reconstruct [INFO] Skeletonize image by thinning
Qiber3D_reconstruct [INFO] Euclidean distance transformation
Qiber3D_reconstruct [INFO] Link up skeleton
Qiber3D_reconstruct [INFO] Build Qiber3D.Network from the raw graph
Qiber3D_reconstruct [INFO] Cleaning Network
Qiber3D_reconstruct [INFO] Smooth Segments
>>> print(net)
Input file: synthetic.tif
  Number of fibers: 4 (clustered 2)
  Number of segments: 11
  Number of branch points: 4
  Total length: 1120.84
  Total volume: 4665.63
  Average radius: 0.960
  Cylinder radius: 1.151
  Bounding box volume: 799821
```

Based on the synthetic network we can explore further export options of a *Qiber3D.Network*.

```
syn_net.export('synthetic.json')
Qiber3D_core [INFO] Network exported to synthetic.json
```

Listing 1: synthetic.json (truncated)


```
{
  "meta": {
    "created": "2021-01-26T14:39:31.774967", "app_name": "Qiber3D", "app_version": "0.4.0
↪", "source": null,
    "name": "synthetic"
  }, "network": {
  "fiber": {
    "0": {
      "fid": 0, "volume": 1651.7148841392172, "average_radius": 0.8604967317483905,
      "cylinder_radius": 1.0789216328856772, "length": 451.65352873495203, "sid_list":␣
↪[0, 1, 2, 3, 4]
    }, "1": {
      "fid": 1, "volume": 1750.7734629144547, "average_radius": 0.9743524445038705,
      "cylinder_radius": 1.322617429184498, "length": 318.5749324139831, "sid_list": [8,␣
↪5, 6, 7]
    }, "2": {
      "fid": 2, "volume": 600.3195103585697, "average_radius": 1.128023071088283,
↪"cylinder_radius": 1.145703995907954,
      "length": 145.5753120928598, "sid_list": [9]
    }, "3": {
      "fid": 3, "volume": 685.8592471183374, "average_radius": 0.9674890076192243,
      "cylinder_radius": 0.9836495172449595, "length": 225.6339048471931, "sid_list":␣
↪[10]
    }
  }, "extractor_data": null, "bbox": [[10.0, 7.8, -2.7], [153.4, 112.1, 51.2]],
  "bbox_size": [143.4, 104.3, 53.900000000000006], "bbox_volume": 806161.8180000001,
  "center": [81.7, 59.949999999999996, 24.250000000000004], "volume": 4688.667104530579,
  "average_radius": 0.9359459559513582, "max_radius": 1.7082413328724901, "cylinder_
↪radius": 1.1434670669897804,
  "length": 1141.437678088988, "segment": {
    "0": {
      "sid": 0, "volume": 17.301298158883707, "average_radius": 0.3663121252534919,
      "cylinder_radius": 0.3701523254210267, "length": 40.19461319013881, "direction":␣
↪[3.6000000000000014, 36.6, 5.7],
      "point": [
        [20.0, 10.0, -0.0], [19.6, 11.6, -0.2], [19.2, 13.2, -0.3], [18.8, 14.7, -0.4],␣
↪[18.5, 16.2, -0.5],
        [18.2, 17.6, -0.6], [17.9, 19.0, -0.7], [17.7, 20.4, -0.7], [17.5, 21.7, -0.7],␣
↪[17.3, 23.0, -0.7],
```


```
syn_net.export('synthetic.xlsx')
Qiber3D_core [INFO] Network exported to synthetic.xlsx
```

```
syn_net.export('synthetic.csv')
Qiber3D_core [INFO] Network exported to synthetic.csv
```

Listing 2: synthetic.csv (truncated)

```
FID;SID;X;Y;Z;Radius
0;0;20.000;10.000;-0.000;0.500
0;0;19.600;11.600;-0.200;0.473
0;0;19.200;13.200;-0.300;0.448
0;0;18.800;14.700;-0.400;0.425
0;0;18.500;16.200;-0.500;0.404
0;0;18.200;17.600;-0.600;0.385
0;0;17.900;19.000;-0.700;0.367
0;0;17.700;20.400;-0.700;0.351
0;0;17.500;21.700;-0.700;0.337
0;0;17.300;23.000;-0.700;0.324
0;0;17.200;24.200;-0.700;0.313
0;0;17.000;25.400;-0.700;0.304
0;0;16.900;26.600;-0.700;0.296
```

```
syn_net.export('synthetic.mv3d')
Qiber3D_core [INFO] Network exported to synthetic.mv3d
```

Listing 3: synthetic.csv (truncated)

```
# MicroVisu3D file (created by Qiber3D 0.4.0)
# Number of lines   11
# Number of points  1404
# Number of inter.  5
#
# No        x        y        z        d
#
0        20.000        10.000        -0.000        1.000
0        19.600        11.600        -0.200        0.946
0        19.200        13.200        -0.300        0.897
0        18.800        14.700        -0.400        0.851
0        18.500        16.200        -0.500        0.809
0        18.200        17.600        -0.600        0.770
0        17.900        19.000        -0.700        0.735
```

```
syn_net.export('synthetic.x3d')
Qiber3D_core [INFO] Network exported to synthetic.x3d
```

Listing 4: synthetic.x3d (truncated)

```
<?xml version="1.0" encoding ="UTF-8"?>

<X3D profile="Immersive" version="3.0">
  <head>
    <meta name="filename" content="docs\x3d\synthetic_show.x3d"/>
    <meta name="generator" content="Visualization ToolKit X3D exporter v0.9.1"/>
    <meta name="numberofelements" content="11"/>
  </head>
  <Scene>
    <Background skyColor="1.0000000000000000e+00 1.0000000000000000e+00 1.
↪0000000000000000e+00"/>
```

(continues on next page)

(continued from previous page)

```
    <Viewpoint fieldOfView="5.23598790168762207e-01" position="8.18827030567863687e+01 6.
↪01443915117113974e+01 3.86125999329838692e+02" description="Default View" orientation=
↪"0.00000000000000000e+00 0.00000000000000000e+00 1.00000000000000000e+00 -0.
↪00000000000000000e+00" centerOfRotation="8.18827030567863687e+01 6.
↪01443915117113974e+01 2.42252174161411489e+01"/>
    <NavigationInfo type='"EXAMINE" "FLY" "ANY"' speed="4.00000000000000000e+00"␣
↪headlight="true"/>
    <DirectionalLight ambientIntensity="1.00000000000000000e+00" intensity="0.
↪00000000000000000e+00" color="1.00000000000000000e+00 1.00000000000000000e+00 1.
↪00000000000000000e+00"/>
    <Transform DEF="ROOT" translation="0.00000000000000000e+00 0.00000000000000000e+00 0.
↪00000000000000000e+00">
```

2.3 Logging

The amount of output can be changed. Using the *DEBUG* level the results of the extraction steps will be shown. For a silent operation choose the *ERROR* level. While using the python debugging module is more verbose, the log_level can also be set with the corresponding integer values. (*DEBUG*: 10, *INFO*: 20, *WARNING*: 30, *ERROR*: 40)

```
>>> import debugging
>>> from Qiber3D import Network, helper
>>> helper.change_log_level(logging.DEBUG)

>>> net = Network.load('tests/cases/network_example.tif')
```

CHAPTER

THREE

SOURCE DOCUMENTATION

Detailed documentation of *Qiber3D*

3.1 Network / Fiber / Segment

class Qiber3D.**Network**(*data*)

Class representing the complete network

Parameters **data** (*dict*) – metadata and segment data collection

Variables

- **segment** – directory of *Segment* forming the *Network*
- **fiber** – directory of *Fiber* forming the *Network*
- **average_radius** (*float*) – average radius
- **cylinder_radius** (*float*) – radius if segment is interpreted as single cylinder
- **average_diameter** (*float*) – average diameters
- **length** (*float*) – overall length
- **volume** (*int*) – overall volume modeled as truncated cones
- **number_of_fibers** – fiber count
- **vector** (*ndarray*) – vectors between points
- **direction** (*ndarray*) – vector pointing from *start* to *end*
- **bbox** (*float*) – bounding box corners
- **bbox_volume** – bounding box volume
- **center** (*ndarray*) – bounding box center
- **bbox_size** (*ndarray*) – bounding box size

export(*out_path='.'*, *overwrite=False*, *mode=None*, ***kwargs*)

Export the network data. Available file types: `.json`, `.qiber`, `.swc`, `.xlsx`, `.csv`, `.static`, `.tif` or `.mv3d`. For more details see `Qiber3D.io.IO`.

Parameters

- **out_path** (*str, Path*) – file or folder path where to save the network
- **overwrite** –
- **mode** –

- **kwargs** –

Returns

static load(*path*, ***kwargs*)
Load a *Network*. Available file types: `.tif`, `.nd2`, `.json`, `.qiber`, `.swc`, `.ntr`, `.csv`, or `.mv3d`. For more details see `Qiber3D.io.IO`.

Parameters **path** (*str, Path*) – file path to input file

Returns *Qiber3D.Network*

save(*out_path='.'*, *overwrite=False*, *save_steps=False*)
Save network to file.

Parameters

- **out_path** (*str, Path*) – file or folder path where to save the network
- **overwrite** (*bool*) – allow file overwrite
- **save_steps** (*bool*) – add extraction steps to the saved file

Returns path to saved file

Return type Path

class Qiber3D.**Fiber**(*network*, *fiber_id*, *segment_ids*)
Class representing the large elements in a network

Parameters

- **network** (*Network*) – overarching network
- **fiber_id** (*int*) – unique fiber identifier
- **segment_ids** (*list*) – list of segment identifier forming the **Fiber**

Variables

- **fid** – unique fiber identifier
- **segment** – directory of *Segment* forming the *Fiber*
- **average_radius** (*float*) – average radius
- **cylinder_radius** (*float*) – radius if segment is interpreted as single cylinder
- **average_diameter** (*float*) – average diameters
- **length** (*float*) – overall length
- **volume** (*float*) – overall volume modeled as truncated cones
- **graph** (*nx.Graph*) – *Fiber* represented as networkx graph

class Qiber3D.**Segment**(*point*, *radius*, *segment_index*)
Class representing the small element in a network

Parameters

- **point** (*ndarray*) – ordered list of points forming the Segment
- **radius** (*ndarray*) – radii (same order as *point*)
- **segment_index** (*int*) – unique identifier

Variables

- **sid** – unique segment identifier

- **point** (*ndarray*) – ordered points forming the Segment
- **x** (*ndarray*) – ordered list of x coordinates
- **y** (*ndarray*) – ordered list of y coordinates
- **z** (*ndarray*) – ordered list of z coordinates
- **radius** (*ndarray*) – radii in same order as *point* (also available as **r**)
- **average_radius** (*float*) – average radius
- **cylinder_radius** (*float*) – radius if segment is interpreted as single cylinder
- **diameter** (*ndarray*) – diameters in same order as *point* (also available as **d**)
- **average_diameter** (*float*) – average diameters
- **start** (*tuple*) – start point coordinates
- **end** (*tuple*) – end point coordinates
- **vector** (*ndarray*) – vectors between points
- **direction** (*ndarray*) – vector pointing from *start* to *end*
- **length** (*float*) – length from *start* to *end*
- **volume** (*float*) – Segment volume modeled as truncated cones

3.2 config

Qiber3D.config.**app_author = 'Anna Jaeschke; Hagen Eckert'**
str: Authors

Qiber3D.config.**app_name = 'Qiber3D'**
Name of the app

Qiber3D.config.**core_count = 0**
int: CPU core count (0 = autodetect)

class Qiber3D.config.**extract**
Bases: object

Parameter to extract a *Qiber3D.Network* from an image stack

class binary
Bases: object

Binarization

threshold = 'Otsu'
(float or string): binarization threshold in percent, or name of the following auto threshold methods - Otsu, Isodata, Li, Mean, Minimum, Triangle, Yen (examples)

invert = False
bool: invert the input image (extracted structures should be high and background low)

class isotropic_resampling
Bases: object

Rescaling to cubic voxels

target = 'z'
string: select if the z resolution or the xy resolution should be adjust to form cubic voxels

low_memory = False
bool: reduce the memory footprint by using less precision when possible

class median
Bases: object

Median filter - despeckle

apply = True
bool: apply the median filter to the image

footprint = None
array: If set, this 3D binary array is used instead of the size parameter to describe the neighborhood

size = 3
(int or array): size of the neighborhood cuboid, if an int is given the all three axis have the same length

class morph
Bases: object

Morphological dilation and erosion

apply = True
bool: apply the dilation and erosion to the image

iterations = 5
int: number of iterations

remove_vol = 100
float: remove islands with volume smaller than volume smaller than remove_vol - in (voxel_size units)^3

nd2_channel_name = 0
str or int: channel name or index when importing from nd2 files

save_steps = True
bool: save extraction steps compressed in memory

class smooth
Bases: object

Gaussian filter

apply = True
bool: apply a gaussian filter and erosion to the image

sigma = 2.0
float: standard deviation for Gaussian kernel in voxel

truncate = 2.0
float: truncate the filter after this many standard deviations

class teasar
Bases: object

TEASER reconstruction - for parameter explanation see kimimaro

const = 0

dust_threshold = 600

max_paths = 50

pdrf_exponent = 4

pdrf_scale = 100000

scale = 1

soma_acceptance_threshold = 3500

soma_detection_threshold = 1100

soma_invalidation_const = 300

soma_invalidation_scale = 1.0

class thinning
Bases: object

Thinning based reconstruction

distance_voxel_overlap = 15
int: Overlap in voxel for the low memory euclidean distance transformation

sliver_threshold = 6
int: segments with less than sliver_threshold voxels will be treated specially (minimum of 6)

voxel_per_point = 10
float: distance between interpolated points (every segment will have at least five points)

use_teasar = False
bool: if True use the kimimaro TEASAR implementation for reconstruction

voxel_size = None
list(float): size of a voxel in each axis

class z_drop
Bases: object

Z-Drop - Intensity attenuation correction

apply = True
bool: apply the intensity attenuation correction to the image

class Qiber3D.config.**figure**
Bases: object

Settings for matplotlib based figures

dpi = 300
int: figure resolution

format = '.pdf'
str: default figure export format

grid_color = (0.2, 0.2, 0.2)
tuple(float): grid rgb color tuple for directions figure

grid_lw = 0.25
float: grid line width for directions figure

Qiber3D.config.**log_level = 20**
int: default logging level. Use Qiber3D.helper.change_log_level() to change it on the fly

class Qiber3D.config.**render**
Bases: object

Settings for rendering images and animations

animation_height = 720
int: vertical resolution for animations

background = (0.0, 0.0, 0.0)
tuple(float): background color as RGB values (0-1)

color = (0.9, 0.2, 0.2)
tuple(floats): standard foreground color

ffmpeg_path = 'ffmpeg'
str: path to local installation of ffmpeg

image_resolution = 3840
int: horizontal resolution for images

notebook = False
bool: switch to adapt render pipeline for jupyter notebooks (automatically set)

rgba = True
bool: allow transparency for images

Qiber3D.config.**url = 'https://github.com/theia-dev/Qiber3D'**
str: git url

Qiber3D.config.**version = '0.7.0'**
str: version

Qiber3D.config.**version_number = (0, 7, 0)**
tuple(int): version number

3.3 Figure

class Qiber3D.**Figure**(*network*)
Bases: object

Generate matplotlib figures

Parameters **network** – a *Qiber3D.Network*

directions(*out_path='.'*, *overwrite=False*, *grid=True*, *color_map='RdYlGn'*, *mode='fine'*, *bins=None*)
Plot the dominat directions of a *Qiber3D.Network*

Parameters

- **out_path** (*str, Path*) – file or folder path where to save the network, if *None* show the plot.
- **overwrite** (*bool*) – allow file overwrite
- **grid** (*bool*) – show a grid
- **color_map** (*str*) – name of a matplotlib color map
- **mode** (*str*) – *'fine'*: use the vectors between every point, *'main'*: use the vectors between the start and end point of a segment
- **bins** (*int*) – number of bins

Returns path to saved file

Return type Path

histogram(*out_path='.'*, *overwrite=False*, *attribute='length'*, *bins=None*)
Plot a histogram for a specific attribute of *Qiber3D.Fiber*.

Parameters

- **out_path** (*str, Path*) – file or folder path where to save the network, if *None* show the plot.
- **overwrite** (*bool*) – allow file overwrite
- **attribute** (*str*) – one of ['length', 'volume', 'average_diameter', 'average_radius', 'cylinder_radius']
- **bins** (*int*) – number of bins

Returns path to saved file

Return type Path

z_drop(*out_path='.'*, *overwrite=False*)
Plot the z_drop correction.

Parameters

- **out_path** (*str, Path*) – file or folder path where to save the network, if *None* show the plot.
- **overwrite** (*bool*) – allow file overwrite

Returns path to saved file

Return type Path

3.4 Render

class Qiber3D.**Render**(*network*)
Bases: object

Generate 3D representations

Parameters **network** (*Qiber3D.Network*) –

Variables

- **storage** (*Qiber3D.helper.NumpyMemoryManager*) – storage for rasterized data
- **raster** (*np.ndarray*) – rasterized representation of the network
- **raster_resolution** (*float*) – voxel per length unit
- **raster_offset** (*tuple(float)*) – distance between raster image stack origin and network orgin

animation(*out_path='.'*, *overwrite=False*, *duration=3*, *fps=30*, *height=None*, *color_mode='fiber'*, *color_map='jet'*, *color=None*, *background=None*, *object_type=None*, *segment_list=None*, *rgba=False*, *zoom=None*)
Animate a network by rotate the camera around it. Saves as h264 `.mp4` file by default. Supports also `.webm` and `.gif` as target.

Parameters

- **out_path** (*str, Path*) – file or folder path where to save the network, if *None* show the plot.
- **overwrite** (*bool*) – allow file overwrite
- **duration** (*float*) – animation duration in seconds
- **fps** (*int*) – frames per second

- **height** (*int*) – height of the animation (16:9 format)
- **color_mode** (*str*) – sets the way to color the network choose one of [‘flat’, ‘fiber’, ‘fiber_length’, ‘fiber_volume’, ‘segment’, ‘segment_length’, ‘segment_volume’, ‘fiber_segment_ratio’]
- **color_map** (*str*) – name of a matplotlib colormap
- **color** (*tuple(float)*) – color if color_mode is *‘flat’*
- **background** (*tuple(float)*) – background color
- **object_type** (*str*) – when set to *‘line’* render center line
- **segment_list** (*tuple*) – limit the visualisation to certain segment (use sid)
- **rgba** (*bool*) – allow transparency in saved file (for `.gif` and `.webm`)
- **zoom** (*float*) – zoom by rendering a larger image and cutting it down afterwards (must be > 1.0)

Returns path to saved file

Return type Path

compare(*color_mode='flat'*, *color_map='jet'*, *color=None*, *object_type=None*, *segment_list=None*)

Visualize extraction steps (original image, z-drop image, binary image, reconstruction) at once. The parameter influence just the reconstructed network.

Parameters

- **color_mode** (*str*) – sets the way to color the network choose one of [‘flat’, ‘fiber’, ‘fiber_length’, ‘fiber_volume’, ‘segment’, ‘segment_length’, ‘segment_volume’, ‘fiber_segment_ratio’]
- **color_map** (*str*) – name of a matplotlib colormap
- **color** (*tuple(float)*) – color if color_mode is *‘flat’*
- **object_type** (*str*) – when set to *‘line’* render center line
- **segment_list** (*tuple*) – limit the visualisation to certain segment (use sid)

export_image_stack(*out_path='.'*, *overwrite=False*, *voxel_resolution=None*, *segment_list=None*)

Export a network as TIFF image stack.

Parameters

- **out_path** (*str, Path*) – file or folder path where to save the network, if *None* show the plot.
- **overwrite** (*bool*) – allow file overwrite
- **voxel_resolution** (*float*) – number of voxels per unit length
- **segment_list** (*tuple*) – limit the visualisation to certain segment (use sid)

Returns path to saved file

Return type Path

export_intensity_projection(*image=None*, *out_path='.'*, *overwrite=False*, *color_map='bone'*, *mode='max'*, *axis='z'*, *voxel_resolution=None*, *segment_list=None*)

Create a intensity projection of the rasterized network.

Parameters

- **out_path** (*str, Path*) – file or folder path where to save the network, if *None* show the plot.
- **overwrite** (*bool*) – allow file overwrite
- **mode** (*str*) – projection mode, choose between *'max'* or *'average'*
- **axis** (*str*) – project along this axis (x, y, z)
- **voxel_resolution** (*float*) – number of voxels per unit length
- **segment_list** (*tuple*) – limit the visualisation to certain segment (use sid)

Returns path to saved file

Return type Path

export_x3d(*out_path='.'*, *overwrite=False*, *color_mode='flat'*, *color_map='jet'*, *color=None*, *object_type=None*, *segment_list=None*, *azimuth=None*, *elevation=None*, *roll=None*)

Parameters

- **out_path** (*str, Path*) – file or folder path where to save the network, if *None* show the plot.
- **overwrite** (*bool*) – allow file overwrite
- **color_mode** (*str*) – sets the way to color the network choose one of ['flat', 'fiber', 'fiber_length', 'fiber_volume', 'segment', 'segment_length', 'segment_volume', 'fiber_segment_ratio']
- **color_map** (*str*) – name of a matplotlib colormap
- **color** (*tuple(float)*) – color if color_mode is *'flat'*
- **object_type** (*str*) – when set to *'line'* render center line
- **segment_list** (*tuple*) – limit the visualisation to certain segment (use sid)
- **azimuth** (*float*) – change camera azimuth
- **elevation** (*float*) – change camera elevation
- **roll** (*float*) – roll camera

Returns path to saved file

Return type Path

overview(*out_path='.'*, *overwrite=False*, *image_resolution=None*, *color_mode='flat'*, *color_map='jet'*, *color=None*, *background=None*, *object_type=None*, *segment_list=None*, *azimuth=None*, *elevation=None*, *roll=None*, *rgba=None*, *axes=0*)

Parameters

- **out_path** (*str, Path*) – file or folder path where to save the network, if *None* show the plot.
- **overwrite** (*bool*) – allow file overwrite
- **image_resolution** (*int*) – image width
- **color_mode** (*str*) – sets the way to color the network choose one of ['flat', 'fiber', 'fiber_length', 'fiber_volume', 'segment', 'segment_length', 'segment_volume', 'fiber_segment_ratio']

- **color_map** (*str*) – name of a matplotlib colormap
- **color** (*tuple(float)*) – color if color_mode is *'flat'*
- **background** (*tuple(float)*) – background color
- **object_type** (*str*) – when set to *'line'* render center line
- **segment_list** (*tuple*) – limit the visualisation to certain segment (use sid)
- **azimuth** (*float*) – change camera azimuth
- **elevation** (*float*) – change camera elevation
- **roll** (*float*) – roll camera
- **rgba** (*bool*) – allow transparency in saved file
- **axes** – vedo axis selection (Documentation)

Returns path to saved file

Return type Path

classmethod save_3d_image(*image*, *out_path='.'*, *overwrite=False*, *image_resolution=None*, *binary=False*, *threshold=None*, *spacing=None*, *color=None*, *background=None*, *azimuth=None*, *elevation=None*, *roll=None*, *rgba=None*)

Render thresholded image stack to file.

Parameters

- **image** (*np.ndarray*) – image stack
- **out_path** (*str, Path*) – file or folder path where to save the network, if *None* show the plot.
- **overwrite** (*bool*) – allow file overwrite
- **image_resolution** (*int*) – image width
- **binary** (*bool*) – if the image stack is already binary
- **threshold** (*float*) – threshold for volume creation in percent
- **spacing** (*tuple(float)*) – spacing in all three axis
- **color** (*tuple(float)*) – color for rendering (0.0-1.0)
- **background** (*tuple(float)*) – background color
- **azimuth** (*float*) – change camera azimuth
- **elevation** (*float*) – change camera elevation
- **roll** (*float*) – roll camera
- **rgba** (*bool*) – allow transparency in saved file

Returns path to saved file

Return type Path

show(*color_mode='flat'*, *color_map='jet'*, *color=None*, *object_type=None*, *segment_list=None*)

Visualize a network interactively.

Parameters

- **color_mode** (*str*) – sets the way to color the network choose one of [‘flat’, ‘fiber’, ‘fiber_length’, ‘fiber_volume’, ‘segment’, ‘segment_length’, ‘segment_volume’, ‘fiber_segment_ratio’]
- **color_map** (*str*) – name of a matplotlib colormap
- **color** (*tuple(float)*) – color if color_mode is *‘flat’*
- **object_type** (*str*) – when set to *‘line’* render center line
- **segment_list** (*tuple*) – limit the visualisation to certain segment (use sid)

static show_3d_image(*image*, *name*, *binary=False*, *spacing=None*)

Visualize a image stack interactively. The threshold can be altered in the opened window.

Parameters

- **image** (*np.ndarray*) – image stack
- **name** (*str*) – display name
- **binary** (*bool*) – if the image stack is already binary
- **spacing** (*tuple(float)*) – spacing in all three axis

3.5 Filter

class Qiber3D.**Filter**

Bases: object

static length(*net*, *length*)

Parameters

- **net** (Qiber3D.Network) – Original Network
- **length** –

Returns Filtered network

Return type *Qiber3D.Network*

static volume(*net*, *volume*)

Parameters

- **net** (Qiber3D.Network) – Original Network
- **volume** –

Returns Filtered network

Return type *Qiber3D.Network*

static volume_ratio(*net*, *ratio=0.5*, *min_volume=None*)

Parameters **net** (Qiber3D.Network) – Original Network

3.6 IO

class Qiber3D.IO.**load**(*path*, ***kwargs*)

Bases: object

Returns a new *Qiber3D.Network* from file.

Supports: .qiber, .json, .mv3d, .tif, .nd2, .swc, .ntr

Parameters

- **path** (*str, Path*) – file path to load
- **kwargs** – key-word arguments are passed down to the individual IO functions

static binary(*path*)

Create a *Qiber3D.Network* from a .qiber file, created by *Qiber3D.Network.save()*

Parameters **path** (*str, Path*) – file path to load

Returns *Qiber3D.Network*

static image(*path*, *channel=None*, *voxel_size=None*)

Create a *Qiber3D.Network* from a image file.

Parameters

- **path** (*str, Path*) – file path to load
- **channel** (*int, str*) – either index or name of image channel
- **voxel_size** (*tuple(float)*) – physical size of voxel in (x,y,z)

Returns *Qiber3D.Network*

static json(*path*, *data=None*)

Create a *Qiber3D.Network* from a .json file.

Parameters

- **path** (*str, Path*) – file path to load
- **data** (*dict*) – load network from a dict representation of the .json file directly

Returns *Qiber3D.Network*

static mv3d(*path*)

Create a *Qiber3D.Network* from a .mv3d file.

Parameters **path** (*str, Path*) – file path to load

Returns *Qiber3D.Network*

static nd2(*path*, *channel=None*)

Create a *Qiber3D.Network* from a .nd2 file.

Parameters

- **path** (*str, Path*) – file path to load
- **channel** (*int, str*) – either index or name of image channel

Returns *Qiber3D.Network*

static network(*net*, *scale=1*, *input_path=None*, *segment_list=None*)

Create a new *Qiber3D.Network* from a *Qiber3D.Network*.

Parameters

- **net** (Qiber3D.Network) – original network
- **scale** (*float*) – scale all points and radii by this value
- **input_path** (*str, Path*) – set this path as new imput path of the returning network
- **segment_list** (*tuple*) – limit the new network to this list of segments (sid)

Returns *Qiber3D.Network*

static ntr(*path*)
Create a *Qiber3D.Network* from a `.ntr` file.

Parameters **path** (*str, Path*) – file path to load

Returns *Qiber3D.Network*

static swc(*path*, *allowed_types=None*)
Create a *Qiber3D.Network* from a `.swc` file.

Parameters

- **path** (*str, Path*) – file path to load
- **allowed_types** (*tuple*) – limit the returned network to these SEGMENT_TYPEs

Returns *Qiber3D.Network*

classmethod synthetic_network()
Create the synthetic test network.

Returns *Qiber3D.Network*

class Qiber3D.IO.**export**(*net*, *out_path='.'*, *overwrite=False*, *mode=None*, ***kwargs*)
Bases: `object`

Export a *Qiber3D.Network* to file. Selecting the appropriate format based on the file suffix.

Supports: `.qiber`, `.json`, `.mv3d`, `.tif`, `.nd2`, `.swc`, `.csv`, `.tsv`, `.xlsx`, `.x3d`

Parameters

- **net** (Qiber3D.Network) – network to export
- **out_path** (*str, Path*) – file or folder path where to save the network
- **overwrite** (*bool*) – allow file overwrite
- **mode** (*str*) – select the file format ignoring the file suffix. Choose from [‘binary’, ‘json’, ‘mv3d’, ‘x3d’, ‘swc’, ‘xlsx’, ‘csv’, ‘tsv’, ‘tif’]
- **kwargs** – key-word arguments are passed down to the individual IO functions

Returns path to saved file

Return type Path

static binary(*net*, *out_path='.'*, *overwrite=False*, *save_steps=False*)
Export *Qiber3D.Network* as binary file (`.qiber`).

Parameters

- **net** (Qiber3D.Network) – network to export
- **out_path** (*str, Path*) – file or folder path where to save the network
- **overwrite** (*bool*) – allow file overwrite
- **save_steps** (*bool*) – save extraction steps image stacks

Returns path to saved file

Return type Path

static csv(*net*, *out_path='.'*, *overwrite=False*, *separator=';'*)
Export *Qiber3D.Network* as `.csv` file.

Parameters

- **net** (Qiber3D.Network) – network to export
- **out_path** (*str, Path*) – file or folder path where to save the network
- **overwrite** (*bool*) – allow file overwrite
- **separator** (*str*) – char to separate values

Returns path to saved file

Return type Path

static json(*net*, *out_path='.'*, *overwrite=False*)
Export *Qiber3D.Network* as `.json` file.

Parameters

- **net** (Qiber3D.Network) – network to export
- **out_path** (*str, Path*) – file or folder path where to save the network
- **overwrite** (*bool*) – allow file overwrite

Returns path to saved file

Return type Path

static mv3d(*net*, *out_path='.'*, *overwrite=False*)
Export *Qiber3D.Network* as `.mv3d` file.

Parameters

- **net** (Qiber3D.Network) – network to export
- **out_path** (*str, Path*) – file or folder path where to save the network
- **overwrite** (*bool*) – allow file overwrite

Returns path to saved file

Return type Path

static swc(*net*, *out_path='.'*, *overwrite=False*, *multiple_files=False*)
Export *Qiber3D.Network* as `.swc` file.

Parameters

- **net** (Qiber3D.Network) – network to export
- **out_path** (*str, Path*) – file or folder path where to save the network
- **overwrite** (*bool*) – allow file overwrite
- **multiple_files** (*bool*) – save each fiber as separate swc file

Returns path to saved file

Return type Path

static tif(*net*, *out_path='.'*, *overwrite=False*, *voxel_resolution=None*, *segment_list=None*)
Export *Qiber3D.Network* as `.tif` image stack.

Parameters

- **net** (Qiber3D.Network) – network to export
- **out_path** (*str, Path*) – file or folder path where to save the network, if *None* show the plot.
- **overwrite** (*bool*) – allow file overwrite
- **voxel_resolution** (*float*) – number of voxels per unit length
- **segment_list** (*tuple*) – limit the visualisation to certain segment (use sid)

Returns path to saved file

Return type Path

static x3d(*net*, *out_path='.'*, *overwrite=False*, *color_mode='flat'*, *color_map='jet'*, *color=None*, *object_type=None*, *segment_list=None*, *azimuth=None*, *elevation=None*, *roll=None*)

Export *Qiber3D.Network* as `.x3d` file.

Parameters

- **net** (Qiber3D.Network) – network to export
- **out_path** (*str, Path*) – file or folder path where to save the network
- **overwrite** (*bool*) – allow file overwrite
- **color_mode** (*str*) – sets the way to color the network choose one of [‘flat’, ‘fiber’, ‘fiber_length’, ‘fiber_volume’, ‘segment’, ‘segment_length’, ‘segment_volume’, ‘fiber_segment_ratio’]
- **color_map** (*str*) – name of a matplotlib colormap
- **color** (*tuple(float)*) – color if color_mode is *‘flat’*
- **object_type** (*str*) – when set to *‘line’* render center line
- **segment_list** (*tuple*) – limit the visualisation to certain segment (use sid)
- **azimuth** (*float*) – change camera azimuth
- **elevation** (*float*) – change camera elevation
- **roll** (*float*) – roll camera

Returns path to saved file

Return type Path

static xlsx(*net*, *out_path='.'*, *overwrite=False*)

Export *Qiber3D.Network* as Excel file (`.xlsx`).

Parameters

- **net** (Qiber3D.Network) – network to export
- **out_path** (*str, Path*) – file or folder path where to save the network
- **overwrite** (*bool*) – allow file overwrite

Returns path to saved file

Return type Path

3.7 Extractor

class Qiber3D.**Extractor**(*input_path*, *channel=None*, *voxel_size=None*)
Bases: object

Create a *Qiber3D.Network* from an image stack with *get_network()*. Apply filter as set in *Qiber3D.config*.

Parameters

- **input_path** (*str, Path*) – file path to input file
- **channel** (*int, str*) – channel name or index
- **voxel_size** (*list(float)*) – size of a voxel in each axis

Variables

- **storage** (*Qiber3D.helper.NumpyMemoryManager*) – storage for extraction steps
- **processing_data** (*dict*) – extra information on the image processing
- **z_spacing** (*float*) – voxel size along z-axis
- **xy_spacing** (*float*) – voxel size along x/y-axis

Ivar:net generated network

get_network()

Returns Network from prepared image stack

Return type *Qiber3D.Network*

prepare()
Load and prepare an image stack for reconstruction.

reconstruct()
Reconstruct the network from the prepared image stack.

3.8 Reconstruct

class Qiber3D.**Reconstruct**
Bases: object

classmethod clean(*net*, *sliver_threshold=6*)
Clean a *Qiber3D.Network* from small sliver.

Parameters

- **net** (*Qiber.Network*) – network to smooth
- **sliver_threshold** (*int*) – treat smaller segments

Returns

classmethod create_base_network(*image*, *low_memory=False*, *distance_voxel_overlap=15*)
Create a unoptimized *Qiber3D.Network* from a binary image.

Parameters

- **image** (*np.ndarray*) – binary image stack
- **low_memory** (*bool*) – split the image for the euclidean distance transformation

- **distance_voxel_overlap** (*int*) – overlap for the low memory euclidean distance transformation in voxel

Returns *Qiber3D.Network*

classmethod get_network(*image*, *scale=1.0*, *input_path=None*, *sliver_threshold=6*, *voxel_per_point=10.0*, *low_memory=False*, *distance_voxel_overlap=15*)

Create a cleaned and smoothed *Qiber3D.Network* from a binary image.

Parameters

- **image** (*np.ndarray*) – binary image stack
- **scale** (*float*) – ratio between voxel size and length unit
- **input_path** (*str, Path*) – set this path as new imput path of the returning network
- **sliver_threshold** (*int*) – treat smaller segments
- **voxel_per_point** (*float*) – distance between interpolated points
- **low_memory** (*bool*) – low memory mode
- **distance_voxel_overlap** (*int*) – overlap for the low memory euclidean distance transformation in voxel

Returns *Qiber3D.Network*

classmethod smooth(*net*, *voxel_per_point=10*)

Smooth a *Qiber3D.Network* in place with a third order spline interpolation.

Parameters

- **net** (*Qiber.Network*) – network to smooth
- **voxel_per_point** (*float*) – distance between interpolated points

Returns *Qiber3D.Network*

3.9 helper

class Qiber3D.helper.**NumpyMemoryManager**(*compressor='blosclz'*, *storage=None*, *meta=None*)

Bases: object

Stores numpy arrays compressed in memory in a dictionary like structure.

Parameters

- **compressor** (*str*) – a compression algorithm suporrted by blosc
- **storage** (*dict*) – prefilled storage dictionary
- **meta** (*dict*) – prefilled meta dictionary

Variables **compression_ratio** (*float*) – compression ratio

classmethod load(*in_path=None*, *fileobj=None*)

Create a *NumpyMemoryManager* from either a file or a file object.

Parameters

- **in_path** (*str*) – path to load
- **fileobj** – a opened file object

Returns *NumpyMemoryManager*

save(*out_path*)
Save a *NumpyMemoryManager* to file.

Parameters **out_path** (*str, Path*) – file or folder path where to save the *NumpyMemoryManager*

class Qiber3D.helper.**LookUp**(**arg*, ***kw*)
Bases: object

Two way lookup dictionary

class Qiber3D.helper.**PointLookUp**(*points=()*, *places=3*, *convert='float'*)
Bases: object

Two way lookup dictionary between points and point IDs.

Parameters

- **points** (*list*) – list of initial points
- **places** (*int*) – round points to number of places if convert is *'float'*
- **convert** (*str*) – convert points to *'float'* or *'int'*

add_points(*points*)
Add multiple points at once.

Parameters **points** (*list*) – list of points to add

class Qiber3D.helper.**Example**
Bases: object

classmethod load_example(*ex_name*)
Download examples files from figshare.

Parameters **ex_name** (*str*) – name of the example (see Example.ex_list)

Returns Path

classmethod nd2()
Short form of *Example.load_example()* called with microvascular_network.nd2

Returns Path

classmethod tiff()
Short form of *Example.load_example()* called with microvascular_network.tif

Returns Path

classmethod tiff_c2()
Short form of *Example.load_example()* called with microvascular_network-C2.tif

Returns Path

classmethod tiff_c2_red()
Short form of *Example.load_example()* called with microvascular_network-C2-reduced.tif

Returns Path

- genindex

PYTHON MODULE INDEX

INDEX

J

L

M

N

O

P

Q

R

S

T

U

V

X

Z