Demo_Part_Generation

November 23, 2022

1 Demonstration of Generation Pipeline

[1]:
```
import time
import numpy as np
from matplotlib import pyplot as plt
import os
import subprocess
from sklearn import cluster
from utils import *
import pandas as pd
from analyze_pore_samples.plotting_utils import frame_tick,␣
↪set_axes_equal,animate_data, improve_pairplot
from analyze_pore_samples.analyze_porosity_clean import load_data,␣
↪analyze_boundaries, process_images, replace_boundary
from make_surface.generate_surfaces import generate_surface
from reconstruction.pipeline_clean import load_boundary, analyze_results,␣
↪trim_boundary, plot_image
```

```
/home/cmu/anaconda3/lib/python3.7/site-packages/skimage/io/manage_plugins.py:23:
UserWarning: Your installed pillow version is < 8.1.2. Several security issues
(CVE-2021-27921, CVE-2021-25290, CVE-2021-25291, CVE-2021-25293, and more) have
been fixed in pillow 8.1.2 or higher. We recommend to upgrade this library.
  from .collection import imread_collection_wrapper
```

```
use torch backend
use torch backend
```

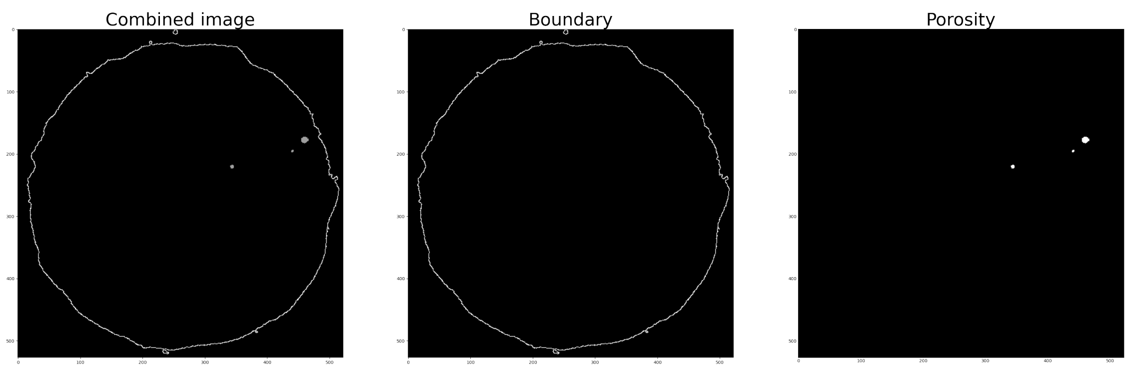
1.1 Load CT scan data

The first step in the process is to load the stack of .tiff images, and perform preprocessing. During this stage, the sample resolution and angle of rotation are used to convert from voxel units to physical measurements.

Several image stacks are used for further processing. `imstack_test` contains the combined surface roughness and pore_images, while `pore_stack` contains only the porosity, and `boundary_stack` only contains the boundary of the part.

[2]:
```
%matplotlib inline
folder_index = 0 # Index of part to be analyzed
frame_number = 200 # Number of frames to analyze, most parts have 2000 - 3000␣
 ↪frames depending on resolution
rootdir = './sample_dataset'
subfolders = [ f.path for f in os.scandir(rootdir) if f.is_dir() ]
voxelsizes = np.loadtxt('./sample_dataset/VoxelSize.txt', skiprows = 1, dtype=␣
 ↪'str')
voxelsizes = dict(voxelsizes)
voxelsize, imstack_test, _, _  = load_data(folder_index = folder_index, num =␣
 ↪frame_number)
pore_stack, boundary_stack = replace_boundary(imstack_test)

fig, ax = plt.subplots(1,3, dpi = 100, figsize = np.array([9,4])*5)

ax[0].imshow(imstack_test[:,:,0], cmap = 'gist_gray')
ax[0].set_title("Combined image", fontsize = 40)
ax[1].imshow(boundary_stack[:,:,0], cmap = 'gist_gray')
ax[1].set_title("Boundary", fontsize = 40)
ax[2].imshow(pore_stack[:,:,0], cmap = 'gist_gray')
ax[2].set_title("Porosity", fontsize = 40)
frame_tick()
plt.savefig('data_example.png')
plt.show()
```

```
199it [00:03, 64.89it/s]
```



[15]:
```
plt.figure(dpi = 300)
im2 = np.zeros(pore_stack[:,:,0].shape + (4,))
im2[:, :, 3] = pore_stack[:,:,0]>0
plt.imshow(im2)
plt.axis('off')
```

[15]: (-0.5, 521.5, 526.5, -0.5)

1.2 Calculate surface roughness details

Next, the 2-D surface roughness profile is extracted from the boundaries, by converting to polar coordinates and storing the fluctuation in radius as a function of r and z. This process is shown for a fragment of 500 frames below.

[3]:
```
%matplotlib inline
profilometry = analyze_boundaries(k = 0, limit = 500, data_info = (voxelsize, imstack_test))

theta = np.linspace(0, 2*np.pi, profilometry.shape[1])
z = np.linspace(0, profilometry.shape[0]*voxelsize,profilometry.shape[0])
thetas, zs = np.meshgrid(theta,z)
```

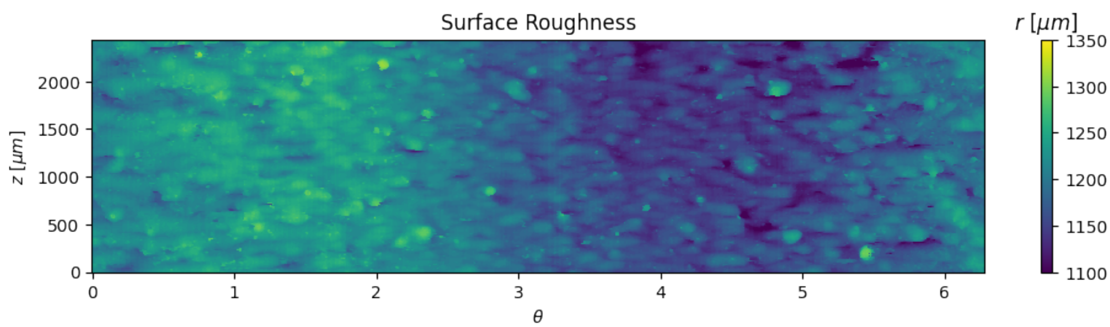
```
Processing boundaries …
```

```
499it [00:07, 63.21it/s]
```

```
processed 0 samples out of 500
29.57880187034607 time in seconds
```

1.2.1 Plot surface roughness profile

[4]:
```
plt.figure(dpi = 100, figsize = np.array([profilometry.shape[1], profilometry.
 ↪shape[0]])/200)
plt.pcolormesh(thetas,zs, profilometry*voxelsize, cmap = 'viridis',shading =␣
 ↪'auto', vmin = 1100, vmax = 1350)
clb = plt.colorbar()
clb.ax.set_title(r'$r$ $ [\mu m]$')
ax = plt.gca()
# ax.axes.set_aspect(profilometry.shape[1]/profilometry.shape[0])
# plt.imshow(profilometry, cmap = 'viridis', extent = [0, 2*np.pi, 0,␣
 ↪profilometry.shape[0]*voxelsize])
plt.axis()
plt.title('Surface Roughness')
plt.xlabel(r'$\theta$')
plt.ylabel(r'$z$ $ [\mu m]$ ')
plt.show()
```



1.3 Extract pore probability estimates and pore information

The individual pores are extracted from the overall part sample, and the relevant statistics are extracted. `all_pores` contains the list of pore objects, while `dict_properties` contains a pandas dataframe of pore properties. Additionally, information required for the reconstruction process is saved under `analyze_pore_samples/results/pore_properties`.

[5]:
```
dict_properties, all_pores = process_images(k = 0, frame_window = frame_number,␣
 ↪shift = frame_number//2, save = True, save_pores = True, data_info =␣
 ↪(voxelsize, imstack_test))
```

```
Processing pores, index = 0 out of 200 …
Pore processed: 0 pores out of 855 pores…
1904 could not broadcast input array from shape (27,75,57) into shape (27,5,57)
Pore processed: 50 pores out of 855 pores…
Saving pores  100
```

```
Pore processed: 100 pores out of 855 pores…
Pore processed: 150 pores out of 855 pores…
Saving pores  200
Pore processed: 200 pores out of 855 pores…
Pore processed: 250 pores out of 855 pores…
Saving pores  300
Pore processed: 300 pores out of 855 pores…
Pore processed: 350 pores out of 855 pores…
Saving pores  400
Pore processed: 400 pores out of 855 pores…
Pore processed: 450 pores out of 855 pores…
Saving pores  500
Pore processed: 500 pores out of 855 pores…
Pore processed: 550 pores out of 855 pores…
Saving pores  600
Pore processed: 600 pores out of 855 pores…
Pore processed: 650 pores out of 855 pores…
Saving pores  700
Pore processed: 700 pores out of 855 pores…
Pore processed: 750 pores out of 855 pores…
Saving pores  800
Pore processed: 800 pores out of 855 pores…
Pore processed: 850 pores out of 855 pores…
Added 855 pores, total is now 855 pores
Processing pores, index = 100 out of 200 …
Pore processed: 0 pores out of 12 pores…
Added 12 pores, total is now 867 pores
```

[9]:
```
dict_properties['rough']
```

[9]:
```
12.264418469165317
```

[9]:
```
import importlib, sys
del process_images
importlib.reload(sys.modules['analyze_pore_samples'])
del test_generator
importlib.reload(sys.modules['reconstruction'])
```

[9]:
```
<module 'reconstruction' (namespace)>
```

Here, we visualize the bivariate relationships present in the original part

[8]:
```
replacements= {'x_locs': 'X Location [$\mu m$]', 'y_locs': 'Y Location [$\mu␣
↪m$]', 'z_locs': 'Z location [$\mu m$]', 'maj_axis_l': 'Major Axis Length',␣
↪'vols': r'$\log_{10} $ Volume [$\mu m^3$]',
                     'anisotropies':'Anisotropy', 'orientations': r'Orientation,␣
↪$\theta$ [rad]', 'phis': r'Orientation, $\phi$ [rad]'}
```

```python
import seaborn as sns; sns.set(style="ticks", color_codes=True)
import pandas as pd
pores_df = pd.DataFrame(dict_properties)
pores_df= pores_df.drop(columns=pores_df.columns[0])
pores_df['vols'] = np.log10(pores_df['vols'])
pores_df = pores_df.drop(['maj_axis_l', 'surf_dist', 'rough', 'surf_angles'],␣
↪axis = 1)
g = sns.pairplot(pores_df, kind = 'kde', plot_kws = {'fill': True, 'alpha':0.
↪7}, palette = {'Ground Truth': '#ff7f0e', 'Reconstructed': '#1f77b4'})
improve_pairplot(g, replacements)
```

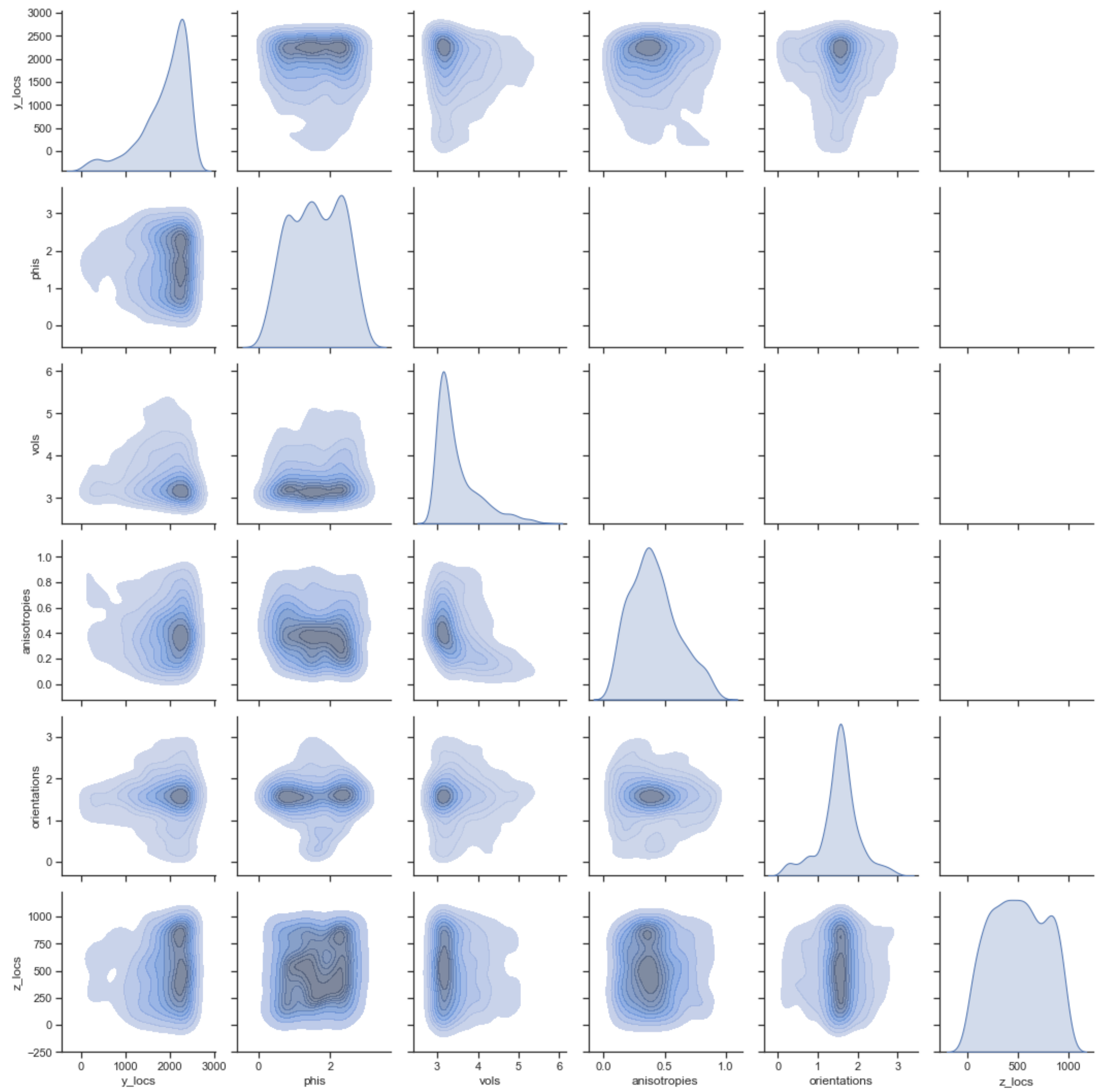
```
    ␣
↪---------------------------------------------------------------------------

    KeyboardInterrupt                         Traceback (most recent call last)

    Input In [8], in <cell line: 10>()
      8 pores_df['vols'] = np.log10(pores_df['vols'])
      9 pores_df = pores_df.drop(['maj_axis_l', 'surf_dist', 'rough',␣
↪'surf_angles'], axis = 1)
---> 10 g = sns.pairplot(pores_df, kind = 'kde', plot_kws = {'fill': True,␣
↪'alpha':0.7}, palette = {'Ground Truth': '#ff7f0e', 'Reconstructed': '#1f77b4'})
     11 improve_pairplot(g, replacements)


    File ~/anaconda3/envs/kymatioenv/lib/python3.8/site-packages/seaborn/
↪_decorators.py:46, in _deprecate_positional_args.<locals>.inner_f(*args,␣
↪**kwargs)
     36     warnings.warn(
     37         "Pass the following variable{} as {}keyword arg{}: {}. "
     38         "From version 0.12, the only valid positional argument "
    (…)
     43         FutureWarning
     44     )
     45 kwargs.update({k: arg for k, arg in zip(sig.parameters, args)})
---> 46 return f(**kwargs)


    File ~/anaconda3/envs/kymatioenv/lib/python3.8/site-packages/seaborn/
↪axisgrid.py:2147, in pairplot(data, hue, hue_order, palette, vars, x_vars,␣
↪y_vars, kind, diag_kind, markers, height, aspect, corner, dropna, plot_kws,␣
↪diag_kws, grid_kws, size)
   2145     from .distributions import kdeplot  # Avoid circular import
   2146     plot_kws.setdefault("warn_singular", False)
-> 2147     plotter(kdeplot, **plot_kws)
   2148 elif kind == "hist":
```

```
   2149     from .distributions import histplot  # Avoid circular import


    File ~/anaconda3/envs/kymatioenv/lib/python3.8/site-packages/seaborn/
↪axisgrid.py:1389, in PairGrid.map_offdiag(self, func, **kwargs)
   1387     self.map_lower(func, **kwargs)
   1388     if not self._corner:
-> 1389         self.map_upper(func, **kwargs)
   1390 else:
   1391     indices = []


    File ~/anaconda3/envs/kymatioenv/lib/python3.8/site-packages/seaborn/
↪axisgrid.py:1372, in PairGrid.map_upper(self, func, **kwargs)
   1361 """Plot with a bivariate function on the upper diagonal subplots.
   1362
   1363 Parameters
   (…)
   1369
   1370 """
   1371 indices = zip(*np.triu_indices_from(self.axes, 1))
-> 1372 self._map_bivariate(func, indices, **kwargs)
   1373 return self


    File ~/anaconda3/envs/kymatioenv/lib/python3.8/site-packages/seaborn/
↪axisgrid.py:1539, in PairGrid._map_bivariate(self, func, indices, **kwargs)
   1537     if ax is None:  # i.e. we are in corner mode
   1538         continue
-> 1539     self._plot_bivariate(x_var, y_var, ax, func, **kws)
   1540 self._add_axis_labels()
   1542 if "hue" in signature(func).parameters:


    File ~/anaconda3/envs/kymatioenv/lib/python3.8/site-packages/seaborn/
↪axisgrid.py:1579, in PairGrid._plot_bivariate(self, x_var, y_var, ax, func,␣
↪**kwargs)
   1577 kwargs.setdefault("hue_order", self._hue_order)
   1578 kwargs.setdefault("palette", self._orig_palette)
-> 1579 func(x=x, y=y, **kwargs)
   1581 self._update_legend_data(ax)


    File ~/anaconda3/envs/kymatioenv/lib/python3.8/site-packages/seaborn/
↪_decorators.py:46, in _deprecate_positional_args.<locals>.inner_f(*args,␣
↪**kwargs)
     36     warnings.warn(
```

```
     37         "Pass the following variable{} as {}keyword arg{}: {}. "
     38         "From version 0.12, the only valid positional argument "
    (…)
     43         FutureWarning
     44     )
     45 kwargs.update({k: arg for k, arg in zip(sig.parameters, args)})
---> 46 return f(**kwargs)


      File ~/anaconda3/envs/kymatioenv/lib/python3.8/site-packages/seaborn/
↪distributions.py:1783, in kdeplot(x, y, shade, vertical, kernel, bw, gridsize,␣
↪cut, clip, legend, cumulative, shade_lowest, cbar, cbar_ax, cbar_kws, ax,␣
↪weights, hue, palette, hue_order, hue_norm, multiple, common_norm,␣
↪common_grid, levels, thresh, bw_method, bw_adjust, log_scale, color, fill,␣
↪data, data2, warn_singular, **kwargs)
   1770     p.plot_univariate_density(
   1771         multiple=multiple,
   1772         common_norm=common_norm,
   (…)
   1778         **plot_kws,
   1779     )
   1781 else:
-> 1783     p.plot_bivariate_density(
   1784         common_norm=common_norm,
   1785         fill=fill,
   1786         levels=levels,
   1787         thresh=thresh,
   1788         legend=legend,
   1789         color=color,
   1790         warn_singular=warn_singular,
   1791         cbar=cbar,
   1792         cbar_ax=cbar_ax,
   1793         cbar_kws=cbar_kws,
   1794         estimate_kws=estimate_kws,
   1795         **kwargs,
   1796     )
   1798 return ax


      File ~/anaconda3/envs/kymatioenv/lib/python3.8/site-packages/seaborn/
↪distributions.py:1111, in _DistributionPlotter.plot_bivariate_density(self,␣
↪common_norm, fill, levels, thresh, color, legend, cbar, warn_singular,␣
↪cbar_ax, cbar_kws, estimate_kws, **contour_kws)
   1109 # Estimate the density of observations at this level
   1110 observations = observations["x"], observations["y"]
-> 1111 density, support = estimator(*observations, weights=weights)
   1113 # Transform the support grid back to the original scale
```

```
   1114 xx, yy = support


    File ~/anaconda3/envs/kymatioenv/lib/python3.8/site-packages/seaborn/
↪_statistics.py:189, in KDE.__call__(self, x1, x2, weights)
    187     return self._eval_univariate(x1, weights)
    188 else:
--> 189     return self._eval_bivariate(x1, x2, weights)


    File ~/anaconda3/envs/kymatioenv/lib/python3.8/site-packages/seaborn/
↪_statistics.py:180, in KDE._eval_bivariate(self, x1, x2, weights)
    177 else:
    179     xx1, xx2 = np.meshgrid(*support)
--> 180     density = kde([xx1.ravel(), xx2.ravel()]).reshape(xx1.shape)
    182 return density, support


    File ~/anaconda3/envs/kymatioenv/lib/python3.8/site-packages/scipy/stats/
↪kde.py:252, in gaussian_kde.evaluate(self, points)
    249 else:
    250     raise TypeError('%s has unexpected item size %d' %
    251                     (output_dtype, itemsize))
--> 252 result = gaussian_kernel_estimate[spec](self.dataset.T, self.weights[:
↪, None],
    253                                         points.T, self.inv_cov,␣
↪output_dtype)
    254 return result[:, 0]


    File _stats.pyx:586, in scipy.stats._stats.gaussian_kernel_estimate()


    File ~/anaconda3/envs/kymatioenv/lib/python3.8/site-packages/numpy/core/
↪_asarray.py:23, in asarray(a, dtype, order, like)
     19 def _asarray_dispatcher(a, dtype=None, order=None, *, like=None):
     20     return (like,)
---> 23 @set_array_function_like_doc
     24 @set_module('numpy')
     25 def asarray(a, dtype=None, order=None, *, like=None):
     26     """Convert the input to an array.
     27 
     28     Parameters
   (…)
     97 
     98     """
     99     if like is not None:
```

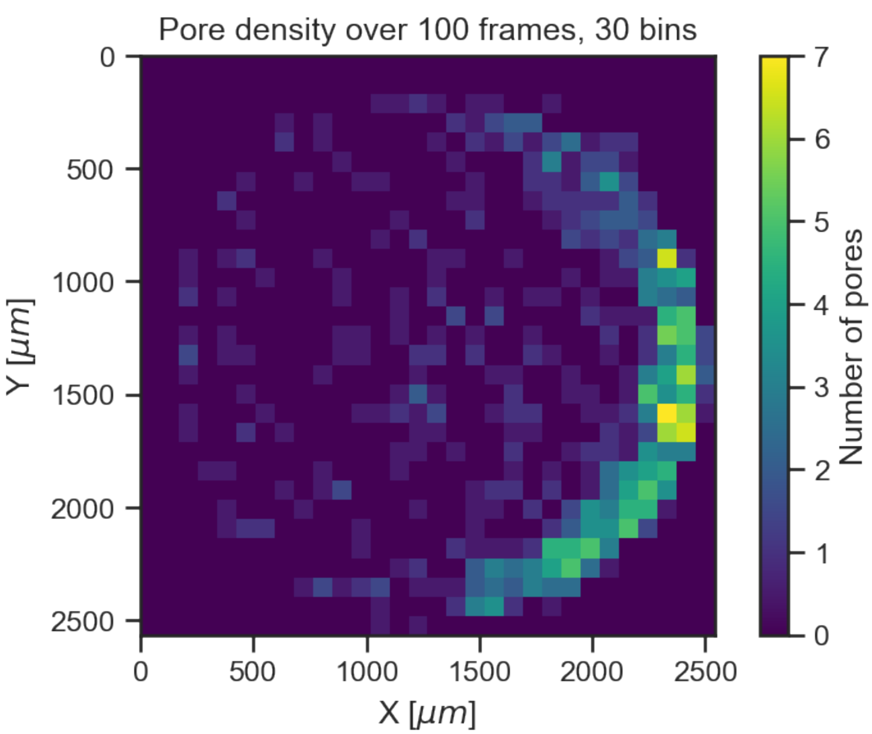
```
KeyboardInterrupt:
```



1.3.1 Coarse-grained probability matrix

The conditional probability of the pore properties, dependent on the cross-sectional area is used later on in the reconstruction process. These quantities are extracted from the original data, a sample density matrix for the number of pores in a given bin is shown below.

[8]:
```
%matplotlib inline
properties_folder = './analyze_pore_samples/results/pore_properties/
↪probability_matrices/'
n_bins = 30
window_size =100
plt.figure(dpi = 150)
prob_matrix_num =  np.load(properties_folder + str(n_bins) +␣
↪'_{}allprob_matrix_num.npy'.format(0))*(window_size/100)
plt.title("Pore density over 100 frames, 30 bins")

plt.imshow(prob_matrix_num, cmap = 'viridis', extent = [0, imstack_test.
↪shape[1]*voxelsize, imstack_test.shape[0]*voxelsize,0])
clb = plt.colorbar()
clb.ax.set_ylabel('Number of pores')
plt.xlabel(r"X [$\mu m$]")
plt.ylabel("Y [$\mu m$]")
plt.show()
```



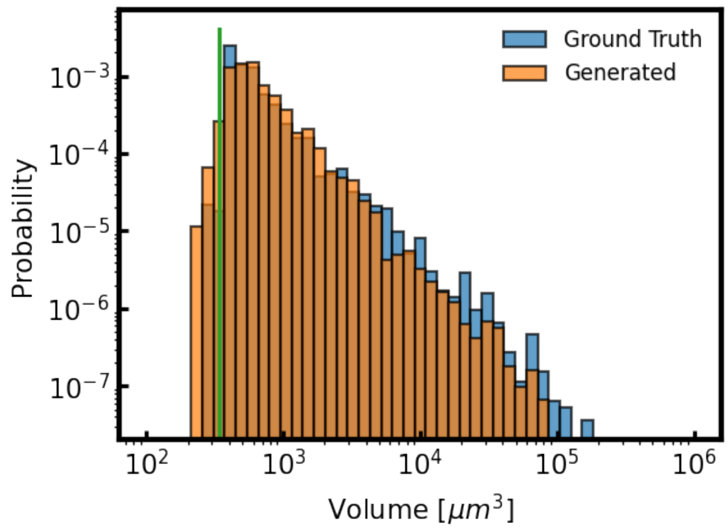
1.4 Generate samples from GAN

Next, a GAN model can be used to generate new pores from the existing distribution of pores. A pre-trained model, trained for 38 epochs on 165 000 pores is included with this repository. The statistics of the reconstruction for N generated and N extracted pores are shown below, where N is the number of pores extracted from the fragment studied above.

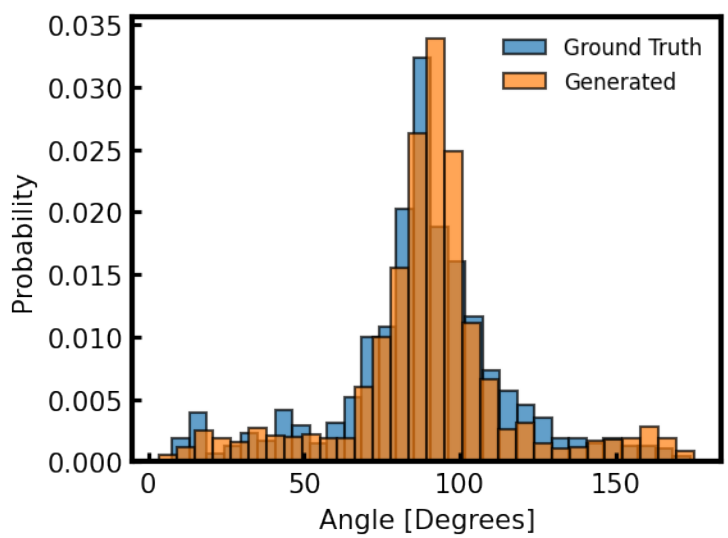
[3]:
```
%matplotlib inline
from reconstruction.pore_generate import test_generator
plt.close('all')
test_generator(num_samples = 2000, folder_index = 0, show = True)
```

```
Epoch,  38
cuda:0  will be used.

Removing previously generated pores
========================
0 pores processed
========================
========================
100 pores processed
========================
========================
200 pores processed
========================
========================
300 pores processed
========================
========================
400 pores processed
========================
========================
500 pores processed
========================
========================
600 pores processed
========================
========================
700 pores processed
========================
========================
800 pores processed
========================
========================
900 pores processed
========================
========================
1000 pores processed
========================
========================
1100 pores processed
========================
```

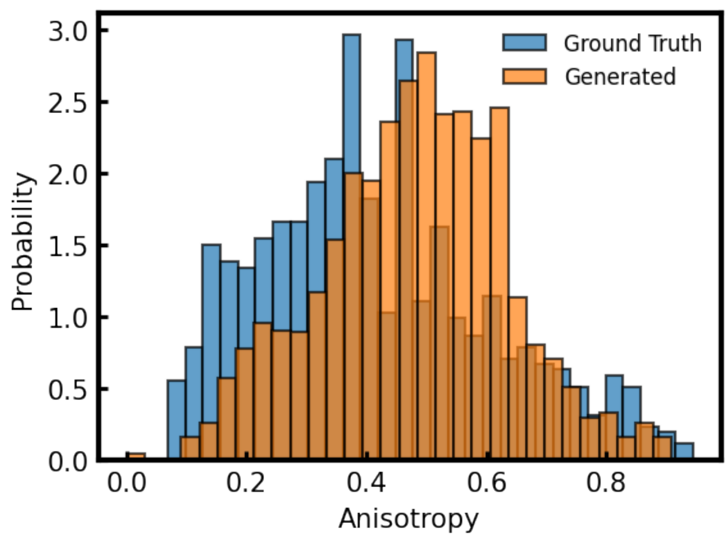
```
========================
1200 pores processed
========================
========================
1300 pores processed
========================
========================
1400 pores processed
========================
========================
1500 pores processed
========================
========================
1600 pores processed
========================
========================
1700 pores processed
========================
========================
1800 pores processed
========================
========================
1900 pores processed
========================
2000 pores generated, saved in  ./reconstruction/gan/saved_generated_pores/
<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>

<Figure size 432x288 with 0 Axes>
```



<Figure size 432x288 with 0 Axes>

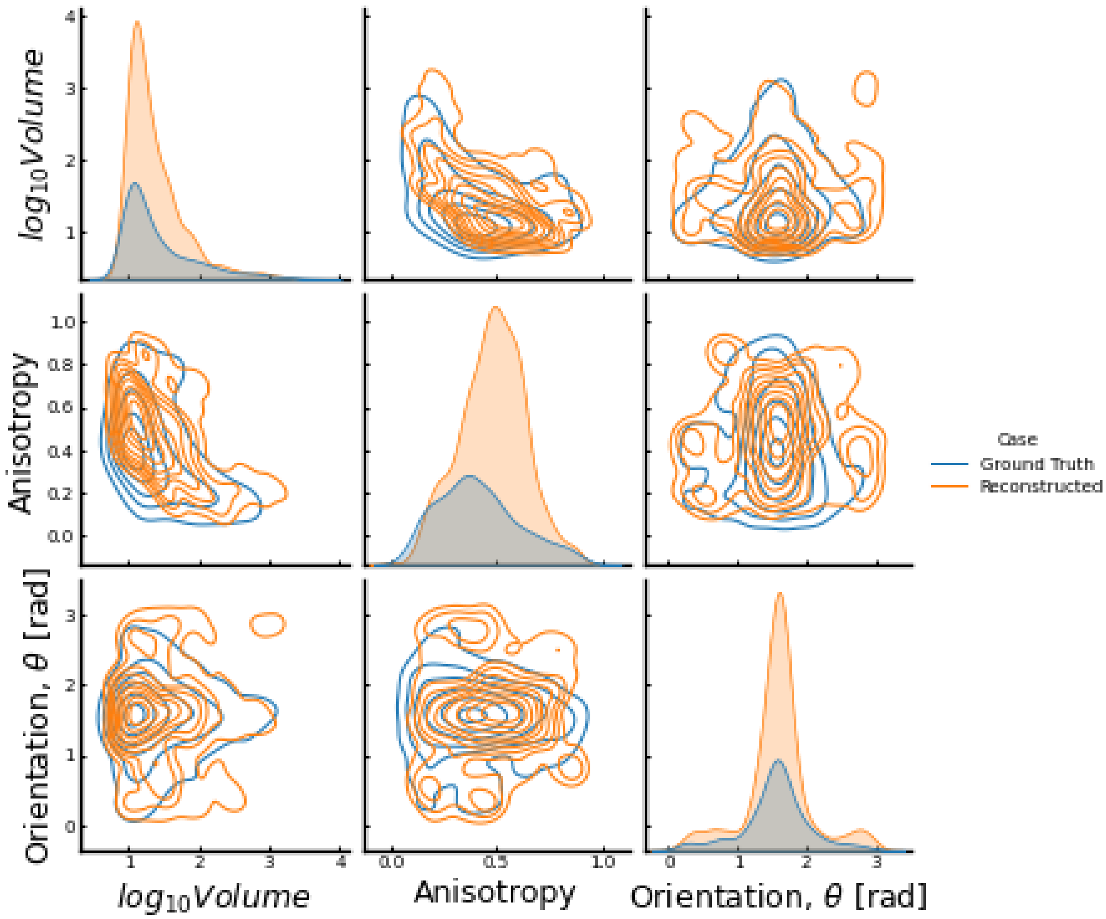


<Figure size 432x288 with 0 Axes>



<Figure size 432x288 with 0 Axes>

<Figure size 1200x900 with 0 Axes>




```
<Figure size 432x288 with 0 Axes>
```

1.5 Generate surface roughness

Following the generation of new individual pores, the next step is to generate the surface roughness, using the MST based micro-canonical model. This process is GPU-intensive and takes 10-15 minutes, so a sample generated roughness profile is also included with this repository. `torch<=1.7.1` is required to run the optimization process. To run the generation process, set the variable `use_saved` to `False`.

[10]:
```
import torch
print(torch.__version__, "Torch version")
use_saved = True
total_size = 2000 # number of boundary frames to generate -- this should be␣
↪close to the total number of frames in the part
profilometry = analyze_boundaries(k = 0, limit = total_size)
if not use_saved:
```

```
    generated_boundary, xmean, ymean, dict_image = generate_surface(0, im =␣
↪profilometry)
else:
    generated_boundary = torch.load('./make_surface/results/
↪sample_number_0original_folder_0/modelC_krec0_start1.pt')['tensor_opt']
    xmean = np.loadtxt('make_surface/results/sample_number_0original_folder_0/
↪xmean0')
    ymean = np.loadtxt('make_surface/results/sample_number_0original_folder_0/
↪ymean0')
im_opt = np.squeeze(np.array(generated_boundary.detach().cpu()+xmean+ymean))
```

```
1.7.1 Torch version
Processing boundaries …
```

```
1999it [00:31, 62.66it/s]
```

```
processed 0 samples out of 2000
processed 500 samples out of 2000
processed 1000 samples out of 2000
processed 1500 samples out of 2000
140.21099090576172 time in seconds
```

[ ]:

```
```

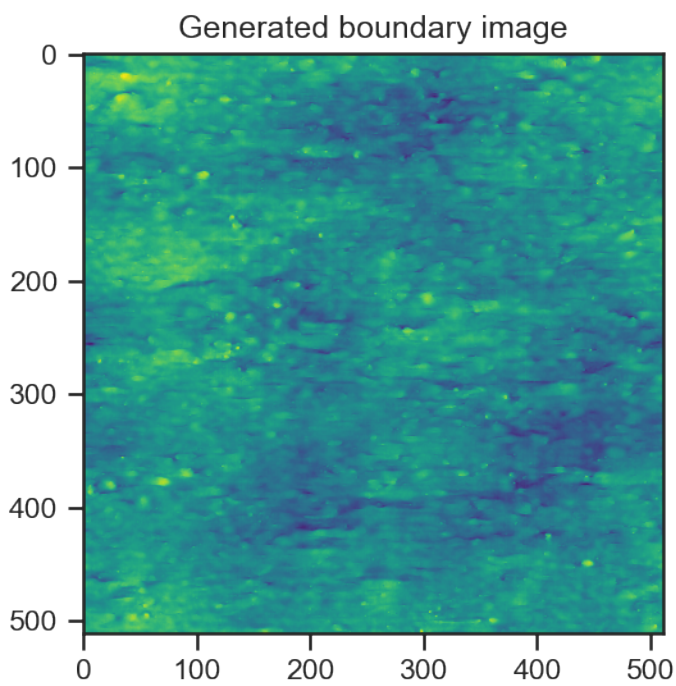
[11]:

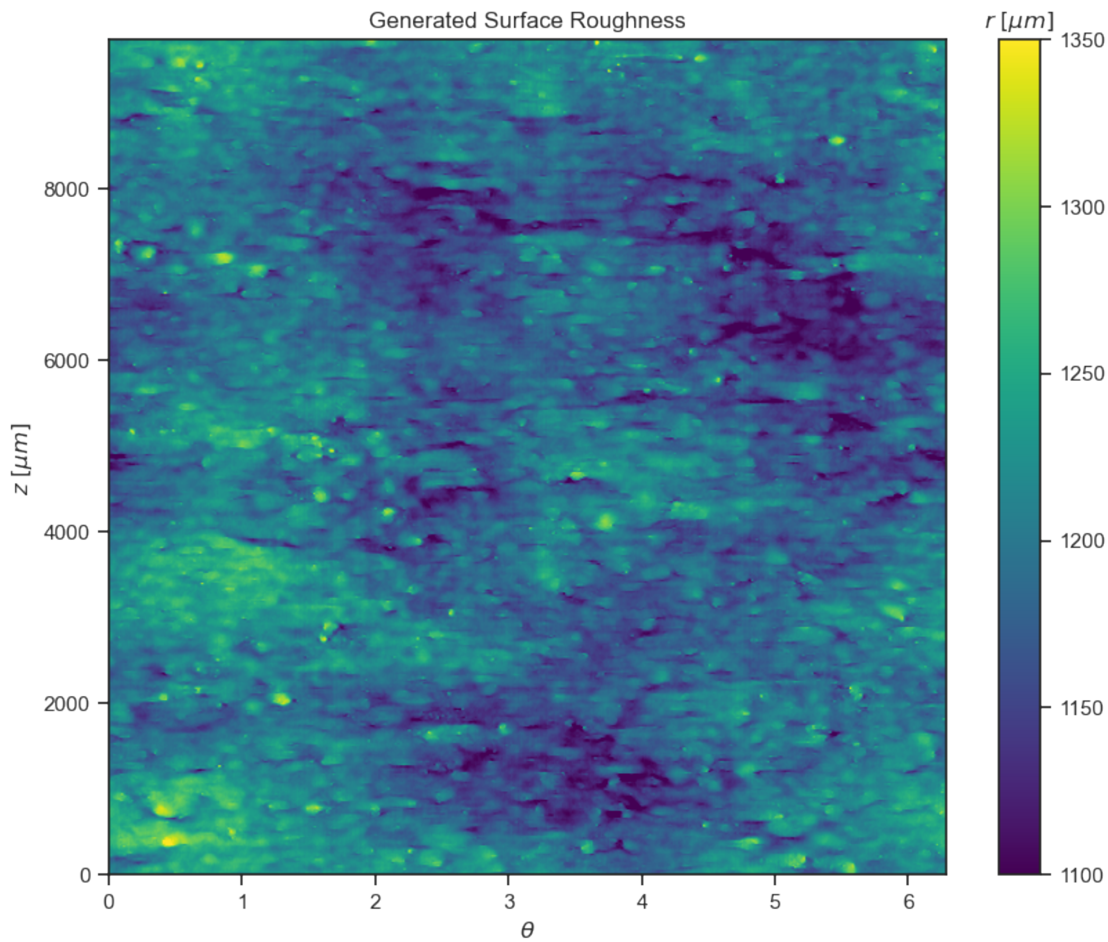
```
plt.figure(dpi = 150)
plt.imshow(im_opt, cmap = 'viridis')
plt.title("Generated boundary image")
plt.show()

from skimage.transform import resize
full_im_opt = im_opt
new_im = resize(full_im_opt, (2000, profilometry.shape[1]), order = 3)
full_image = new_im*(np.max(profilometry) - np.min(profilometry)) + np.
↪min(profilometry)

theta = np.linspace(0, 2*np.pi, full_image.shape[1])
z = np.linspace(0, full_image.shape[0]*voxelsize,full_image.shape[0])
thetas, zs = np.meshgrid(theta,z)
plt.figure(dpi = 100, figsize = np.array([full_image.shape[1], full_image.
↪shape[0]])/250)
plt.pcolormesh(thetas,zs, full_image*voxelsize, cmap = 'viridis',shading =␣
↪'auto', vmin = 1100, vmax = 1350)
clb = plt.colorbar()
clb.ax.set_title(r'$r$ $ [\mu m]$')
ax = plt.gca()
plt.axis()
plt.title('Generated Surface Roughness')
plt.xlabel(r'$\theta$')
```

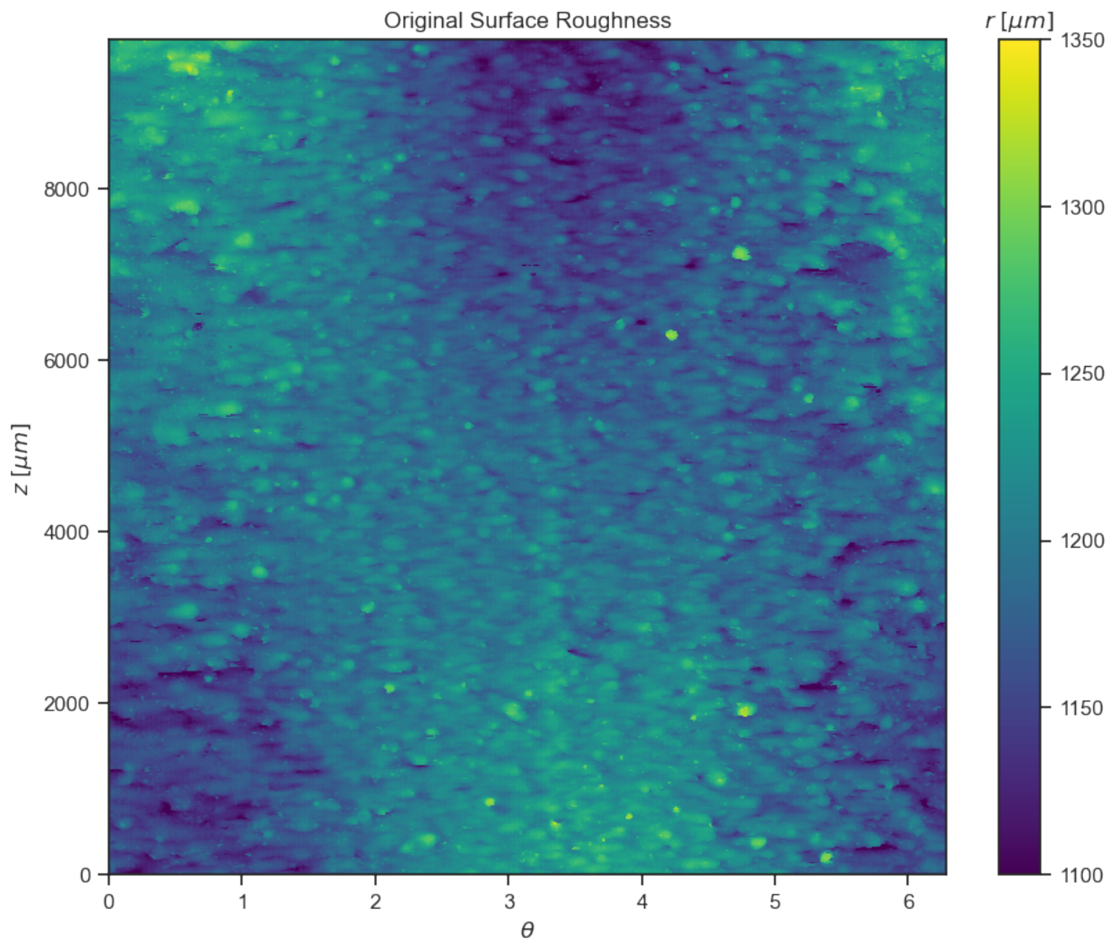
```
plt.ylabel(r'$z$ $ [\mu m]$ ')
plt.show()
```

```
plt.figure(dpi = 100, figsize = np.array([full_image.shape[1], full_image.
 ↪shape[0]])/250)
plt.pcolormesh(thetas,zs, profilometry*voxelsize, cmap = 'viridis',shading =␣
 ↪'auto', vmin = 1100, vmax = 1350)
clb = plt.colorbar()
clb.ax.set_title(r'$r$ $ [\mu m]$')
ax = plt.gca()
plt.axis()
plt.title('Original Surface Roughness')
plt.xlabel(r'$\theta$')
plt.ylabel(r'$z$ $ [\mu m]$ ')
plt.show()
```

```
generated_boundary = load_boundary(num_frames = 500, start = 0, pore_part_shape␣
 ↪= (imstack_test.shape[0], imstack_test.shape[1]), return_profile = False)
```

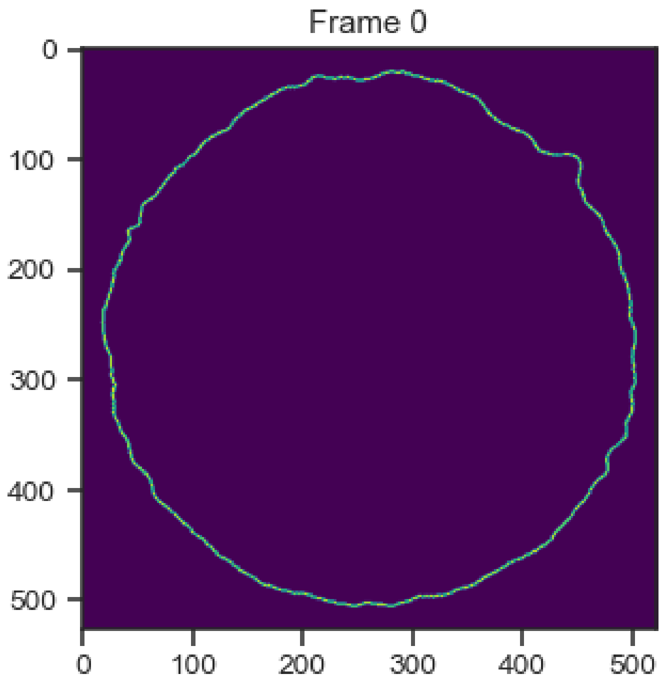



Generated Surface Roughness
$r\,[\mu m]$
1350
1300
1250
1200
1150
1100
$z\,[\mu m]$
8000
6000
4000
2000
0
0
1
2
3
4
5
6
θ

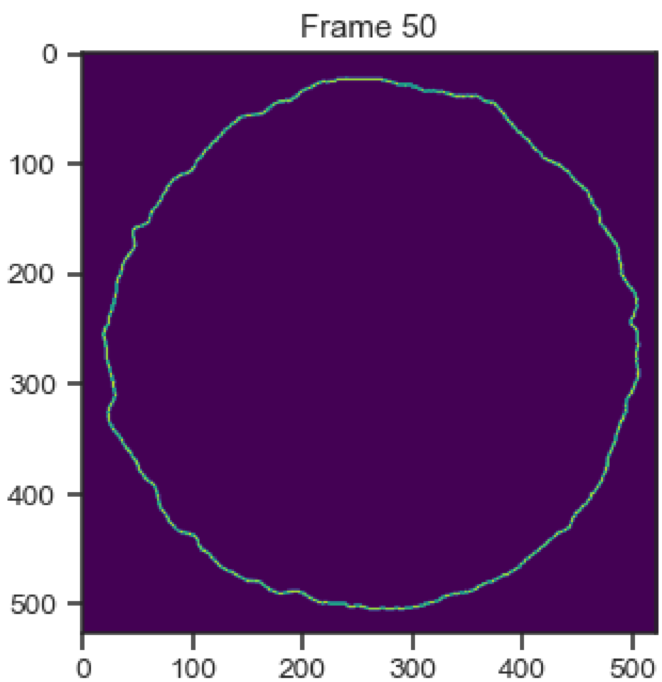



```
0    2.976135e-17
Name: max, dtype: float64
0    2.349059e-17
Name: min, dtype: float64
0
```

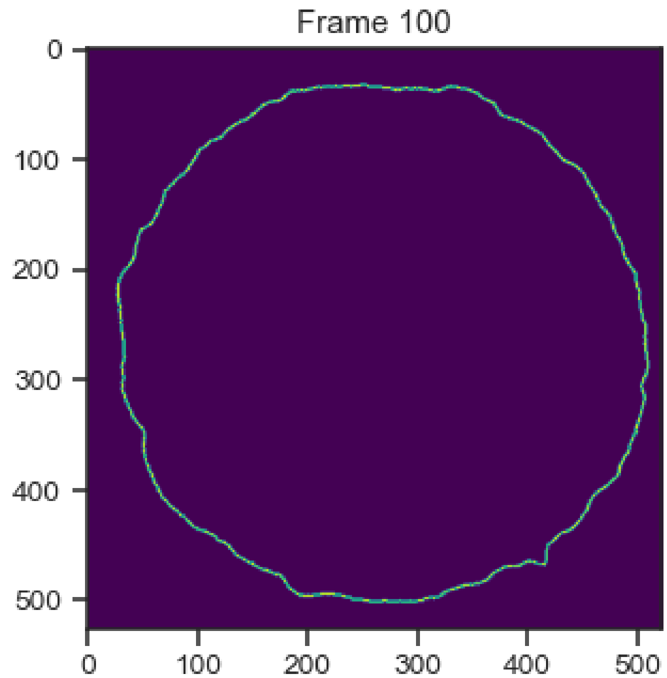
[12]:
```
for i in range(10):
    plt.figure(dpi = 90)

    plt.title('Frame {}'.format(i*50))
    plt.imshow(generated_boundary[:,:,i*50], cmap = 'viridis')
    plt.show()
```

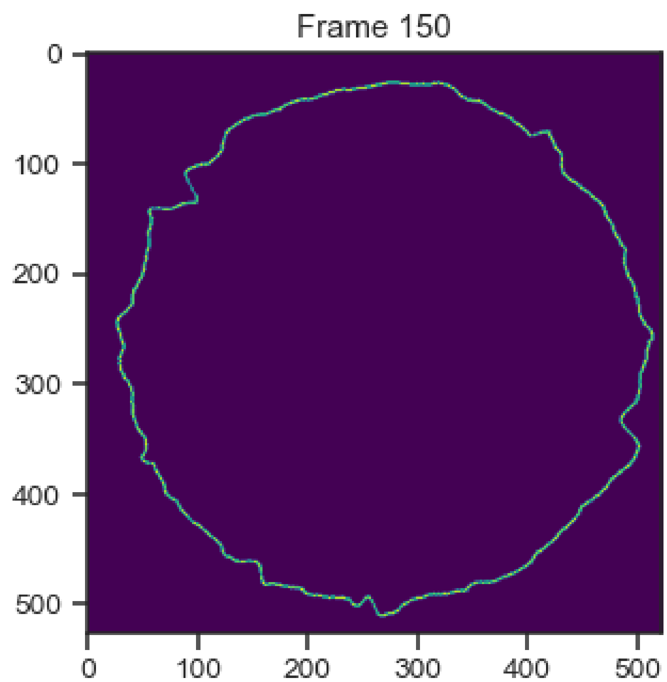


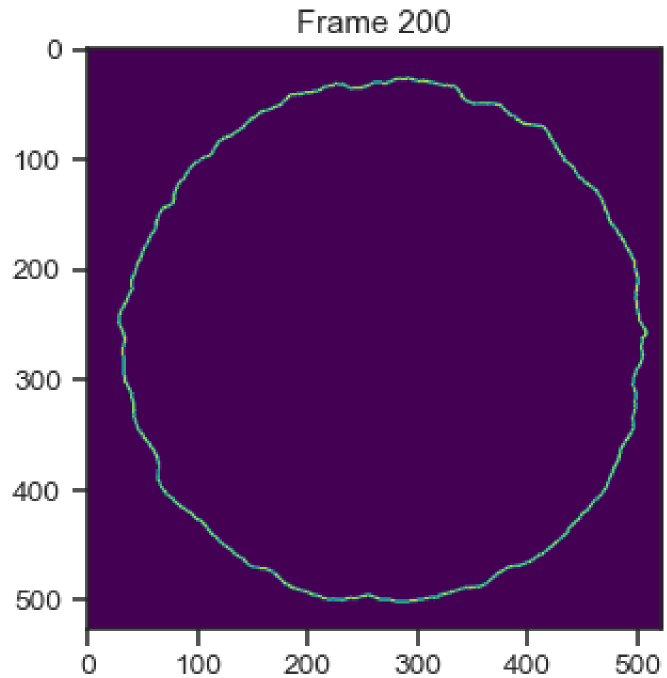
Frame 0
0
100
200
300
400
500
0
100
200
300
400
500

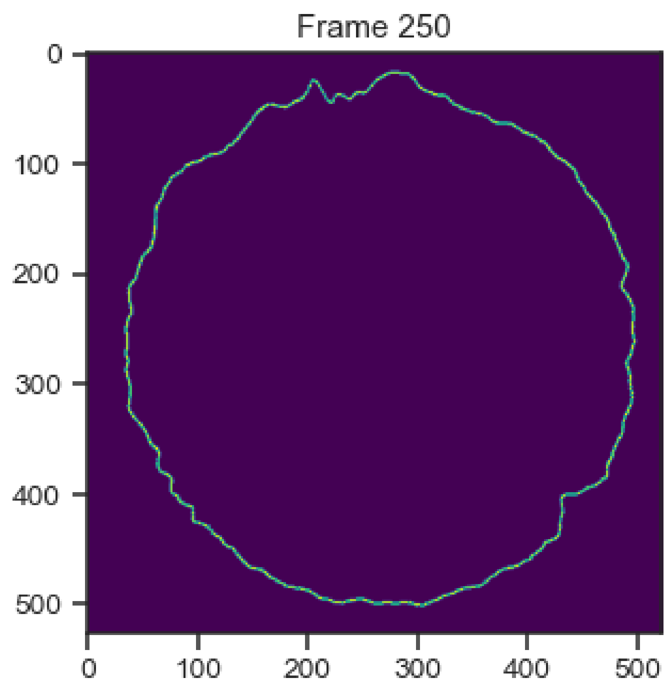


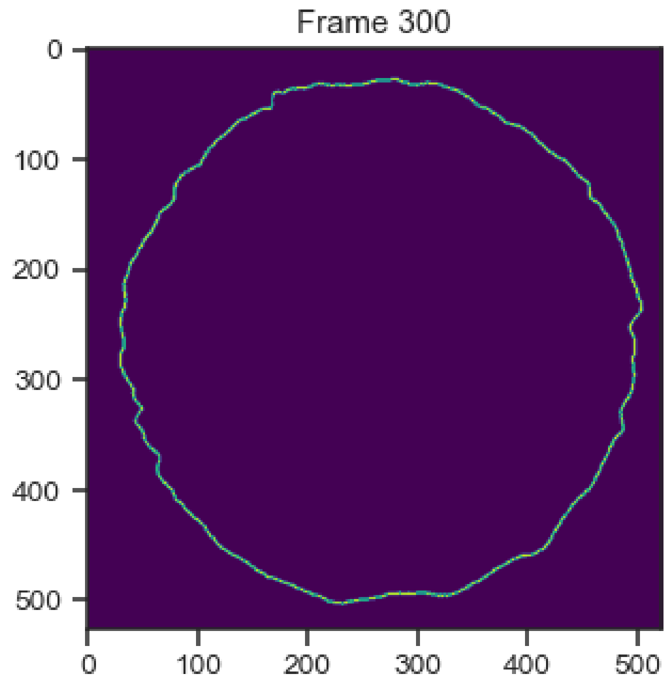
Frame 50
0
100
200
300
400
500
0
100
200
300
400
500

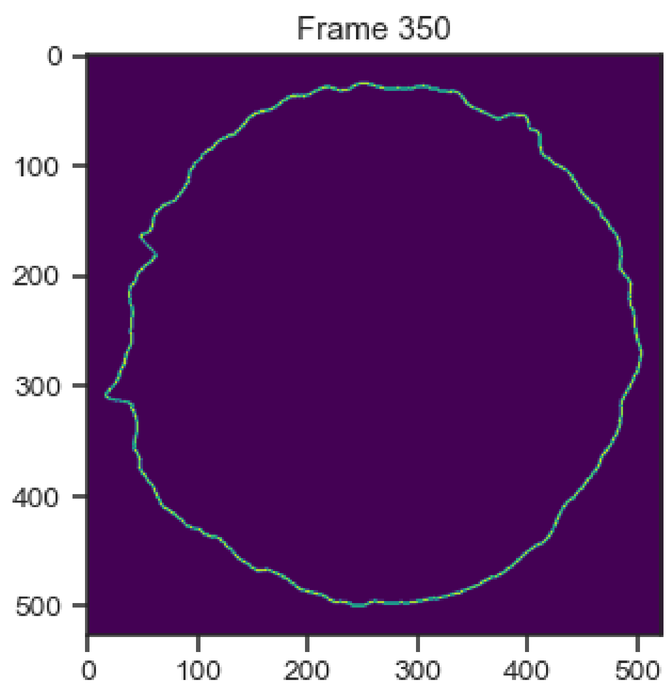

Frame 100
0
100
200
300
400
500
0
100
200
300
400
500

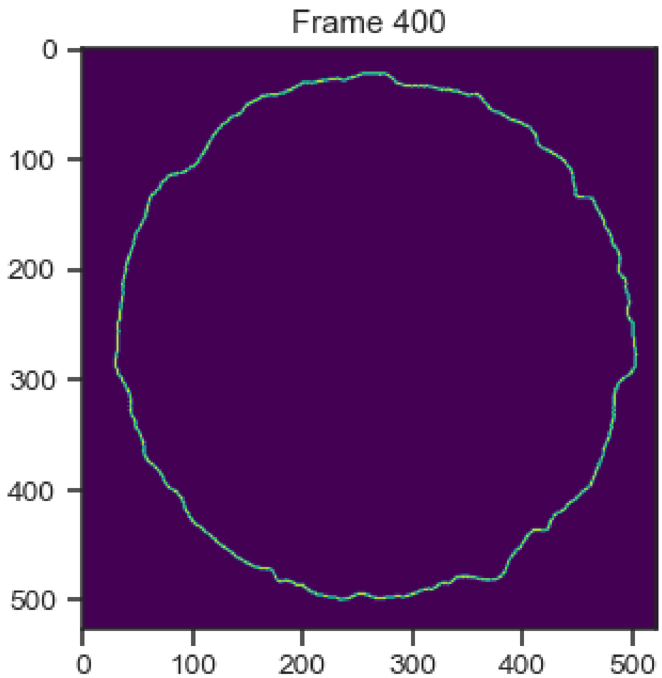

Frame 150
0
100
200
300
400
500
0
100
200
300
400
500

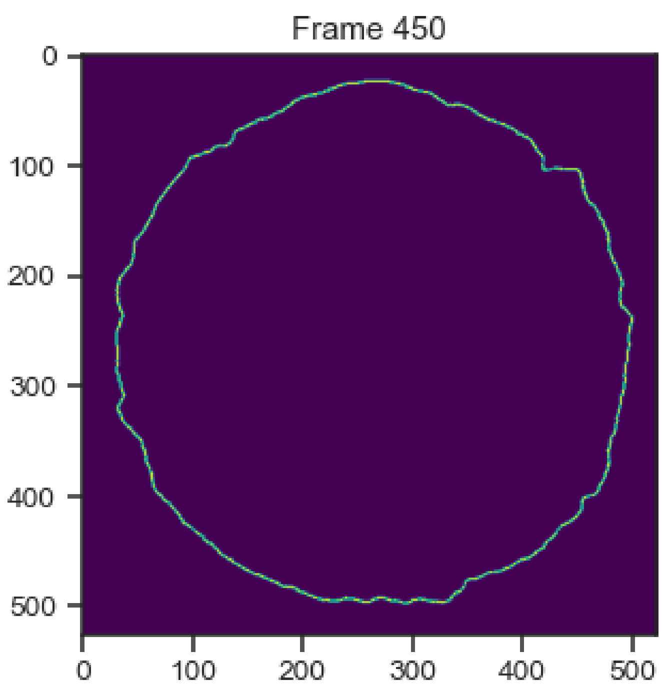

Frame 200
0
100
200
300
400
500
0
100
200
300
400
500


Frame 250
0
100
200
300
400
500
0
100
200
300
400
500

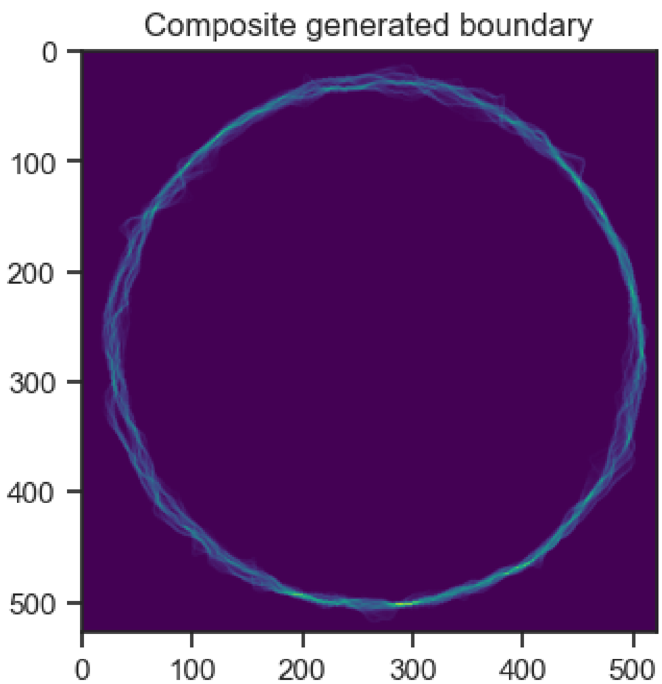

Frame 300
0
100
200
300
400
500
0
100
200
300
400
500

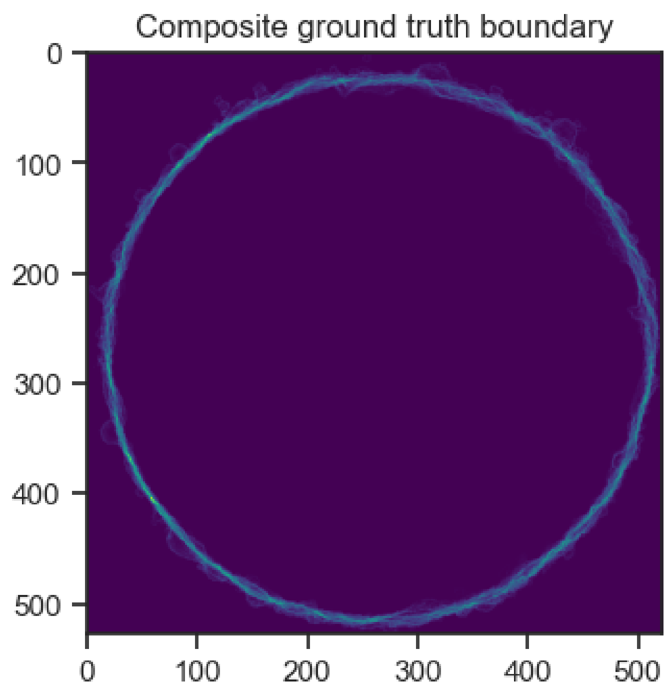

Frame 350
0
100
200
300
400
500
0
100
200
300
400
500


Frame 400
0
100
200
300
400
500
0
100
200
300
400
500


Frame 450
0
100
200
300
400
500
0
100
200
300
400
500

[13]:
```
plt.figure(dpi = 100)
plt.title("Composite generated boundary")
plt.imshow(np.sum(generated_boundary[:,:,:frame_number],axis=2), cmap =␣
 ↪'viridis')
plt.show()
plt.figure(dpi = 100)

plt.title("Composite ground truth boundary")
plt.imshow(np.sum(boundary_stack[:,:,:frame_number],axis=2), cmap = 'viridis')
plt.show()
```





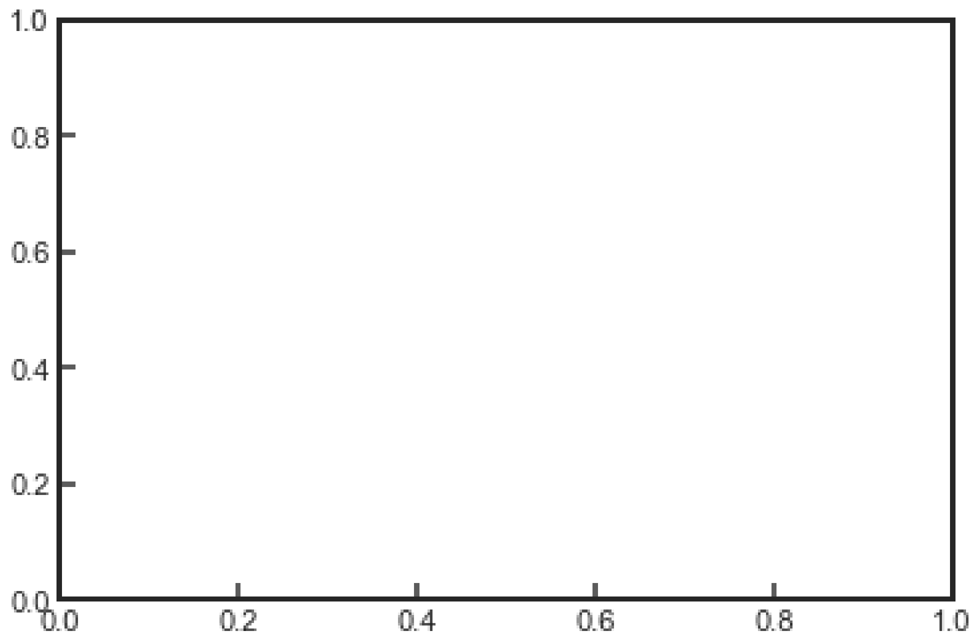
2 Reconstruction of part sample

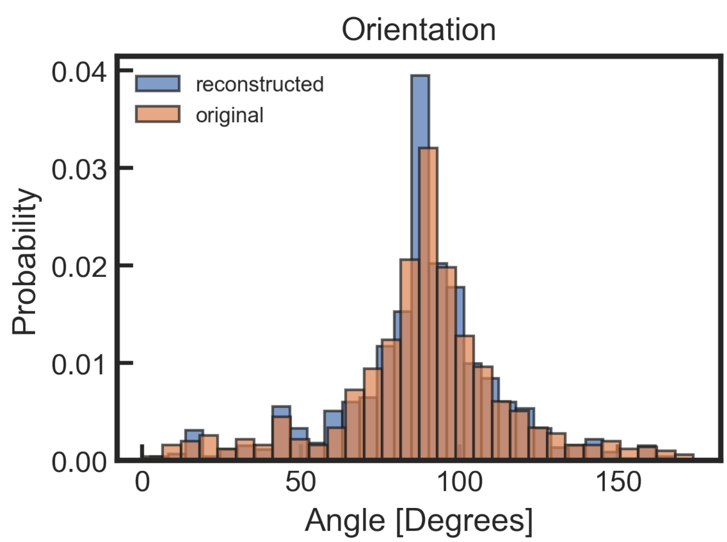
Now that the boundary and additional pore samples have been generated, the next step is to stochastically recombine these using the priors extracted from the original dataset. This process can be done on a basis of either cartesian or polar coordinates, but is demonstrated in this notebook for the cartesian case. Below, the statistics of the reconstruction will be shown. Since the reconstruction is based on the statistics extracted from the part segment, if larger sections of the part segment are analyzed for the statistics, the statistics of the generated part will be more accurate.

[14]:
```
from reconstruction.pipeline_clean import replace_sampling, analyze_results
%matplotlib inline
pore_part = np.zeros((imstack_test.shape[0], imstack_test.shape[1],␣
 ↪frame_number), dtype  = 'uint8')
voxelsize, imstack_test, _, _  = load_data(folder_index = 0, num = frame_number)
pore_part,_ = replace_sampling(pore_part, generated_boundary[:,:,:
 ↪frame_number], n_bins = 30)
print(str(len(_)) + " pores inserted")
folder_index = 0
analyze_results(imstack_test[:,:,:frame_number], pore_part, fname =␣
 ↪'reconstruction/full/statistics_part_' + str(folder_index) +'/', voxelsize =␣
 ↪voxelsize)
```

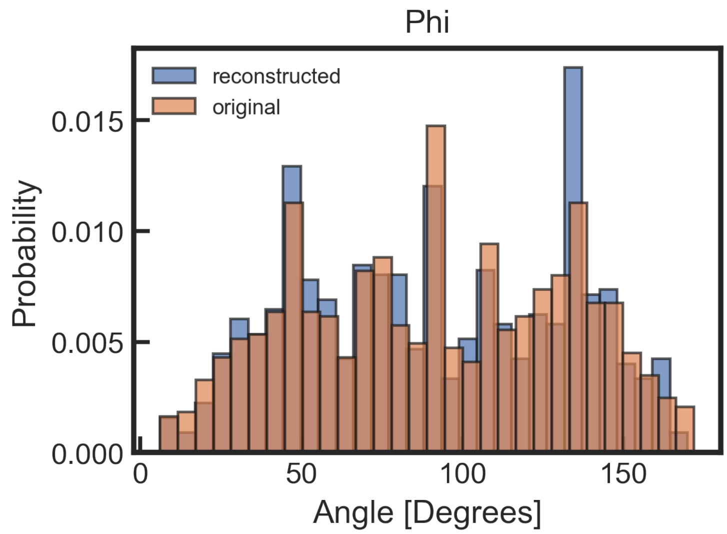
```
199it [00:03, 63.51it/s]
/home/cmu/anaconda3/envs/kymatioenv/lib/python3.8/site-
packages/scipy/stats/_continuous_distns.py:8810: RuntimeWarning: invalid value
encountered in true_divide
  self._hpdf = self._hpdf / float(np.sum(self._hpdf * self._hbin_widths))
```

```
continue 6 activated: Collision
continue 6 activated: Collision
1 1 cmerged
continue 7 activated: Collision
continue 6 activated: Collision
continue 6 activated: Collision
continue 6 activated: Collision
continue 6 activated: Collision
operands could not be broadcast together with shapes (3,2,2) (3,4,2)
continue 4 activated: indexing exception
could not broadcast input array from shape (3,4,3) into shape (3,1,3)
continue 8 activated: Insertion process failed
continue 2 activated: pore on back surface
continue 2 activated: pore on back surface
could not broadcast input array from shape (5,3,3) into shape (5,1,3)
continue 8 activated: Insertion process failed
continue 6 activated: Collision
operands could not be broadcast together with shapes (4,2,2) (4,3,2)
continue 4 activated: indexing exception
continue 6 activated: Collision
continue 2 activated: pore on back surface
continue 2 activated: pore on back surface
continue 2 activated: pore on back surface
1 1 cmerged
continue 7 activated: Collision
continue 6 activated: Collision
1 1 cmerged
continue 7 activated: Collision
continue 6 activated: Collision
862 pores inserted
FINISHED RECONSTRUCTION
```

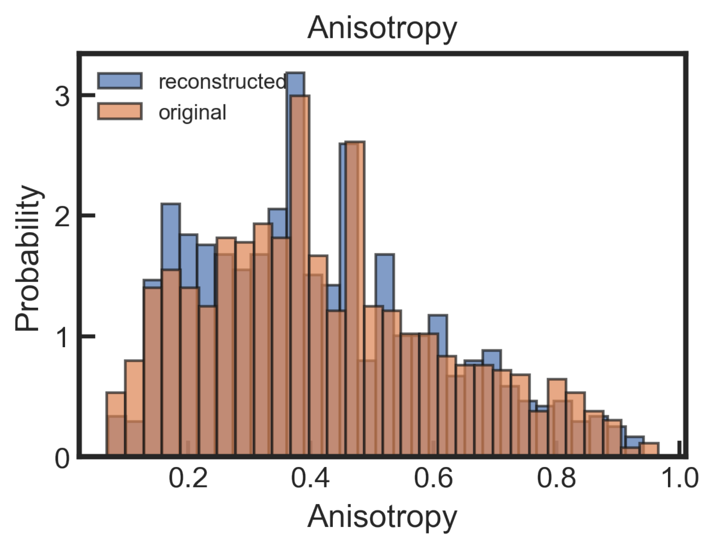




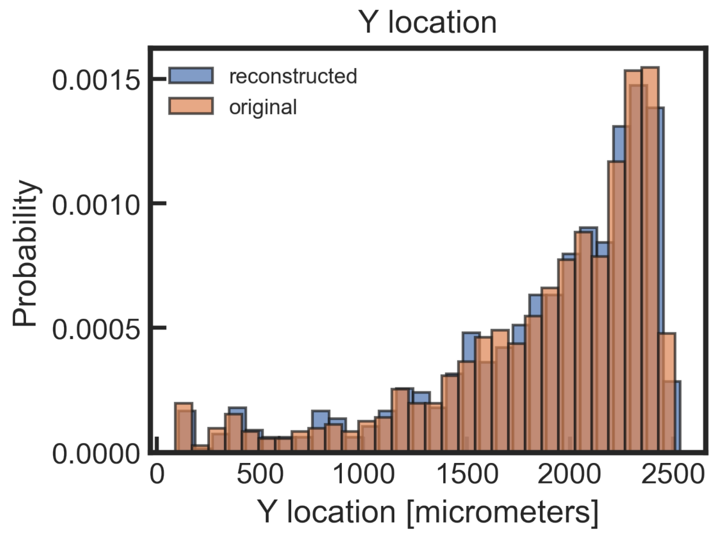

```
<Figure size 432x288 with 0 Axes>
```



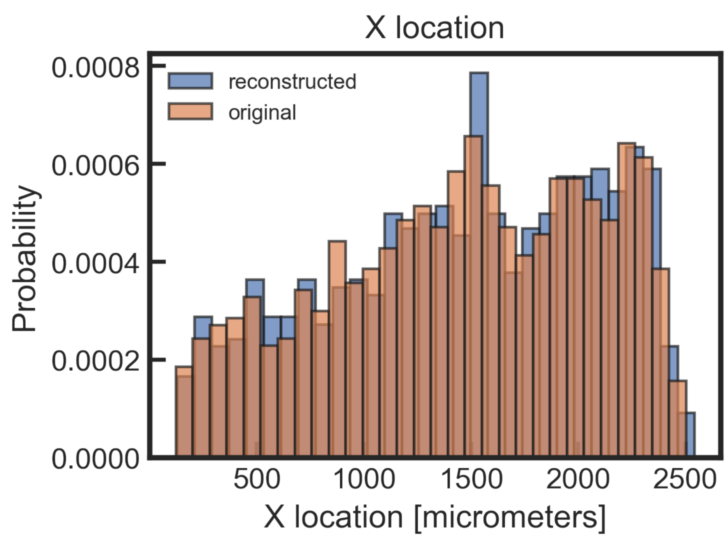
<Figure size 432x288 with 0 Axes>



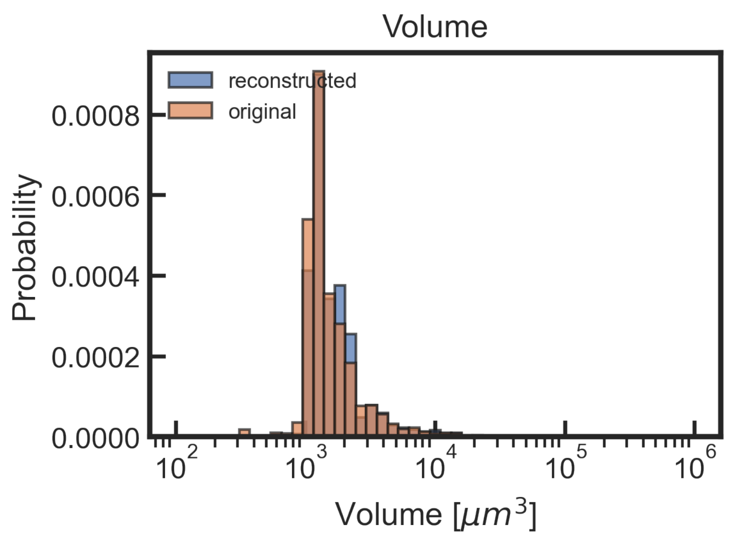
<Figure size 432x288 with 0 Axes>



<Figure size 432x288 with 0 Axes>



<Figure size 432x288 with 0 Axes>




```
Pores in the original sample: 883
Pores in the new sample: 822
```

```
<Figure size 432x288 with 0 Axes>
```

We then save the pore sample as ordered png images for visualization purposes. As in the original dataset, pores are represented by an intensity value of 159, while the boundary has an intensity value of 255.

[15]:
```
n_bins = 30
total_frame = 0
trim_boundary(generated_boundary, pore_part)
plot_image(images = 2*pore_part, shell = generated_boundary, title = './
 ↪reconstruction/full/images_generated_part_{}_bins_{}/sample/'.
 ↪format(folder_index, n_bins) , global_idx =total_frame)
```

```
SAVING TO ./reconstruction/full/images_generated_part_0_bins_30/sample/
```

```
<Figure size 432x288 with 0 Axes>
```

[16]:
```
np.sum(prob_matrix_num)*2
```

[16]: 855.0

We can also visualize the generated part sample in 3-D. To do so, we make use of the Fiji distribution of ImageJ to stack the segmented images together. Running these cells will open an instance of ImageJ to construct and record the segment. It is important that the ImageJ window remains at the foreground during this process. An installation of Fiji is required for this to work, a .zip file is provided of the linux installation. For other installations, see the insstructions here.

[17]:
```python
import subprocess

command = "./Fiji.app/ImageJ-linux64  -macro ./plotting_tools/create_video.ijm␣
↪"+ os.path.join(os.getcwd(),"reconstruction/full/
↪images_generated_part_{}_bins_{}/sample/".format(folder_index, n_bins))
print(command)
subprocess.run(command, shell = True)
```

```
./Fiji.app/ImageJ-linux64  -macro ./plotting_tools/create_video.ijm /home/cmu/Po
rosityGenerator/reconstruction/full/images_generated_part_0_bins_30/sample/
```

```
OpenJDK 64-Bit Server VM warning: ignoring option PermSize=128m; support was
removed in 8.0
OpenJDK 64-Bit Server VM warning: Using incremental CMS is deprecated and will
likely be removed in a future release
3D [dev] 1.6.0-scijava-2-pre11-daily-experimental daily


nFrames = 1
```

```
Error while executing the main() method of class 'net.imagej.Main':
java.lang.NullPointerException
        at net.imagej.legacy.IJ1Helper.setVisible(IJ1Helper.java:313)
        at net.imagej.legacy.ui.LegacyUI.show(LegacyUI.java:132)
        at org.scijava.ui.DefaultUIService.showUI(DefaultUIService.java:158)
        at org.scijava.ui.DefaultUIService.showUI(DefaultUIService.java:143)
        at org.scijava.AbstractGateway.launch(AbstractGateway.java:110)
        at net.imagej.Main.main(Main.java:55)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at
sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAcces
sorImpl.java:43)
        at java.lang.reflect.Method.invoke(Method.java:498)
        at net.imagej.launcher.ClassLauncher.launch(ClassLauncher.java:279)
        at net.imagej.launcher.ClassLauncher.run(ClassLauncher.java:186)
        at net.imagej.launcher.ClassLauncher.main(ClassLauncher.java:87)
X11Util.Display: Shutdown (JVM shutdown: true, open (no close attempt): 1/1,
reusable (open, marked uncloseable): 0, pending (open in creation order): 1)
X11Util: Open X11 Display Connections: 1
X11Util: Open[0]: NamedX11Display[:0, 0x7f0260037530, refCount 1, unCloseable
false]
```

[17]:
```
CompletedProcess(args='./Fiji.app/ImageJ-linux64  -macro
./plotting_tools/create_video.ijm /home/cmu/PorosityGenerator/reconstruction/ful
l/images_generated_part_0_bins_30/sample/', returncode=0)
```

2.1 Visualization of 3-D generated part

Note: this cell should be re-ran after running other cells in order to see the output.

[18]:
```
# Display
import ipywidgets as widgets
from ipywidgets import interact


from PIL import Image, ImageSequence
im = Image.open("./reconstruction/full/images_generated_part_{}_bins_{}/sample/
↪Movie.gif".format(folder_index, n_bins))

generated_frames = []
index = 1
for frame in ImageSequence.Iterator(im):
    generated_frames.append(np.array(frame))
    index += 1


%matplotlib notebook
show_images = generated_frames
fig, ax = plt.subplots(dpi=150)
im = plt.imshow(generated_frames[0], cmap='gray')
plt.title("Generated Sample")
plt.xticks([])
plt.yticks([])
loop_num = np.arange(len(show_images))
@interact(degree = (loop_num[0], loop_num[-1]*2))
def show(degree):
    im.set_array(show_images[degree//2])
    fig.canvas.draw_idle()
show(150)
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

```
interactive(children=(IntSlider(value=179, description='degree', max=358), Output()), _dom_class
```

We can also visualize the ground truth pores in 3-D. To do so, we make use of the Fiji distribution of ImageJ to stack the segmented images together. Running these cells will open an instance of ImageJ to construct and record the segment. It is important that the ImageJ window remains at the foreground during this process until it closes.

[19]:
```
%matplotlib inline


title = 'analyze_pore_samples/results/images_part_{}_{}_frame_segment/sample/'.
↪format(folder_index, frame_number)
name = 'images_part_{}_{}_frame_segment/sample'.format(folder_index,␣
↪frame_number)
plot_image(images = pore_stack, shell = boundary_stack, title = title ,␣
↪global_idx =0)

command = "./Fiji.app/ImageJ-linux64  -macro ./plotting_tools/create_video.ijm␣
↪"+ os.path.join(os.getcwd(),title)

print(command)
subprocess.run(command, shell = True)
```

```
SAVING TO analyze_pore_samples/results/images_part_0_200_frame_segment/sample/
./Fiji.app/ImageJ-linux64  -macro ./plotting_tools/create_video.ijm /home/cmu/Po
rosityGenerator/analyze_pore_samples/results/images_part_0_200_frame_segment/sam
ple/
```

```
OpenJDK 64-Bit Server VM warning: ignoring option PermSize=128m; support was
removed in 8.0
OpenJDK 64-Bit Server VM warning: Using incremental CMS is deprecated and will
likely be removed in a future release
3D [dev] 1.6.0-scijava-2-pre11-daily-experimental daily


nFrames = 1

Error while executing the main() method of class 'net.imagej.Main':
java.lang.NullPointerException
        at net.imagej.legacy.IJ1Helper.setVisible(IJ1Helper.java:313)
        at net.imagej.legacy.ui.LegacyUI.show(LegacyUI.java:132)
        at org.scijava.ui.DefaultUIService.showUI(DefaultUIService.java:158)
        at org.scijava.ui.DefaultUIService.showUI(DefaultUIService.java:143)
        at org.scijava.AbstractGateway.launch(AbstractGateway.java:110)
        at net.imagej.Main.main(Main.java:55)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at
sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAcces
sorImpl.java:43)
        at java.lang.reflect.Method.invoke(Method.java:498)
        at net.imagej.launcher.ClassLauncher.launch(ClassLauncher.java:279)
        at net.imagej.launcher.ClassLauncher.run(ClassLauncher.java:186)
        at net.imagej.launcher.ClassLauncher.main(ClassLauncher.java:87)
X11Util.Display: Shutdown (JVM shutdown: true, open (no close attempt): 1/1,
reusable (open, marked uncloseable): 0, pending (open in creation order): 1)
```

```
X11Util: Open X11 Display Connections: 1
X11Util: Open[0]: NamedX11Display[:0, 0x7f0fc8037530, refCount 1, unCloseable
false]
```

[19]:
```
CompletedProcess(args='./Fiji.app/ImageJ-linux64  -macro
./plotting_tools/create_video.ijm /home/cmu/PorosityGenerator/analyze_pore_sampl
es/results/images_part_0_200_frame_segment/sample/', returncode=1)
```

```
<Figure size 432x288 with 0 Axes>
```

2.2 Visualization of 3-D ground truth fragment

Note: this cell should be re-ran after running other cells in order to see the output.

[20]:
```
# Display
import ipywidgets as widgets
from ipywidgets import interact
%matplotlib notebook

from PIL import Image, ImageSequence
im = Image.open(title + "Movie.gif".format(folder_index, frame_number))

generated_frames = []
index = 1
for frame in ImageSequence.Iterator(im):
    generated_frames.append(np.array(frame))
    index += 1


%matplotlib notebook
show_images = generated_frames
fig, ax = plt.subplots(dpi=150)
im = plt.imshow(generated_frames[0], cmap='gist_gray')
plt.title("Ground Truth Sample")
plt.xticks([])
plt.yticks([])
loop_num = np.arange(len(show_images))
@interact(degree = (loop_num[0], loop_num[-1]*2))
def show(degree):
    im.set_array(show_images[degree//2])
    fig.canvas.draw_idle()
show(150)
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

```
interactive(children=(IntSlider(value=179, description='degree', max=358), Output()), _dom_class
```