Supplementary Information for

Combinatorial Inkjet Printing for Compositional Tuning of Metal-Halide Perovskite Thin Films

Hampus Näsström,[ab] Oleksandra Shargaieva,[a] Pascal Becker,[b] Florian Mathies,[a] Ivo Zizak,[b] Vincent R. F. Schröder[d], Emil J.W. List-Kratochvil,*[cd] Thomas Unold,*[b] and Eva Unger*[ae]

[a] *Young Investigator Group Hybrid Materials Formation and Scaling, Helmholtz-Zentrum Berlin für Materialien und Energie GmbH, Kekulestraße 5, 12489 Berlin, Germany. E-mail: eva.unger@helmholtz-berlin.de*
[b] *Department Structure and Dynamics of Energy Materials, Helmholtz-Zentrum Berlin für Materialien und Energie GmbH, Hahn-Meitner-Platz 1, 14109 Berlin, Germany. E-mail: unold@helmholtz-berlin.de*
[c] *Humboldt-Universität zu Berlin, Institut für Physik, Institut für Chemie & IRIS Adlershof, Zum Großen Windkanal 2, 12489 Berlin, Germany. E-mail: emil.list-kratochvil@hu-berlin.de*
[d] *Helmholtz-Zentrum Berlin für Materialien und Energie GmbH, Hahn-Meitner-Platz 1, 14109 Berlin, Germany.*
[e] *Lund University, Chemical Physics and Nano Lund, Sweden*

Table of figures

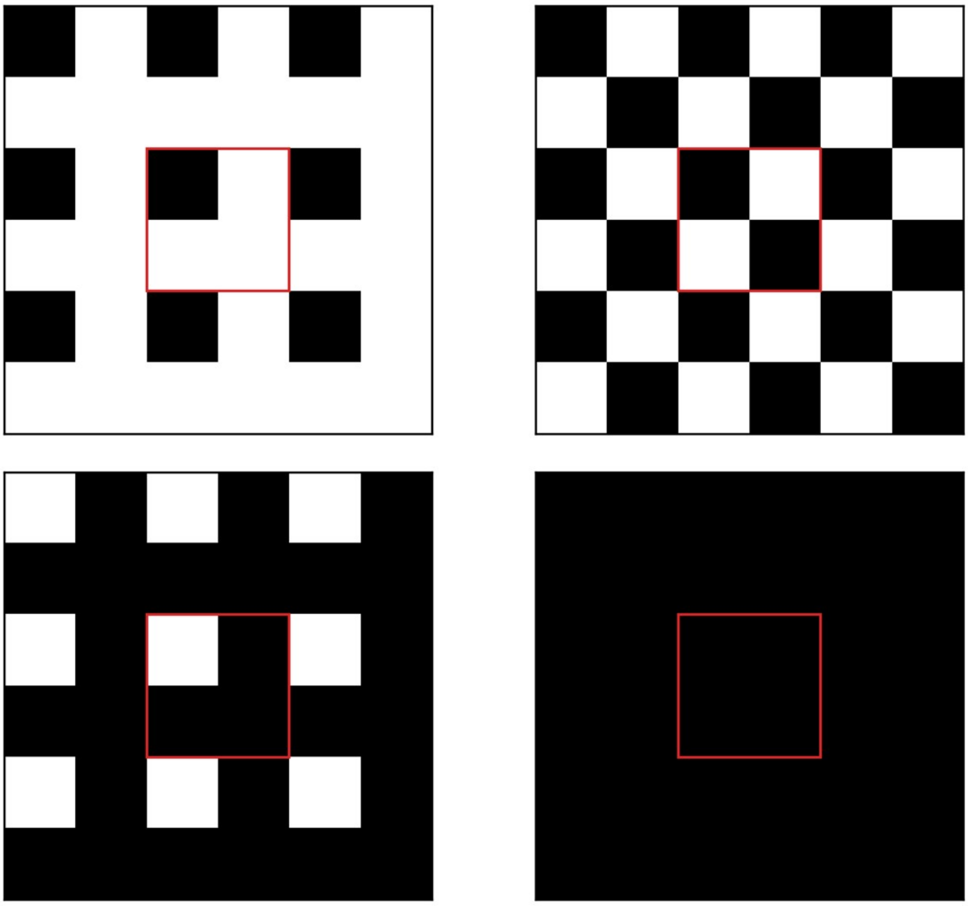


Fig. S1: A 3×3 tiling of the 2×2 base matrix generated from the algorithm in the main text. The base image is highlighted by a red square. The first base with no filling is omitted as it is just a blank image.

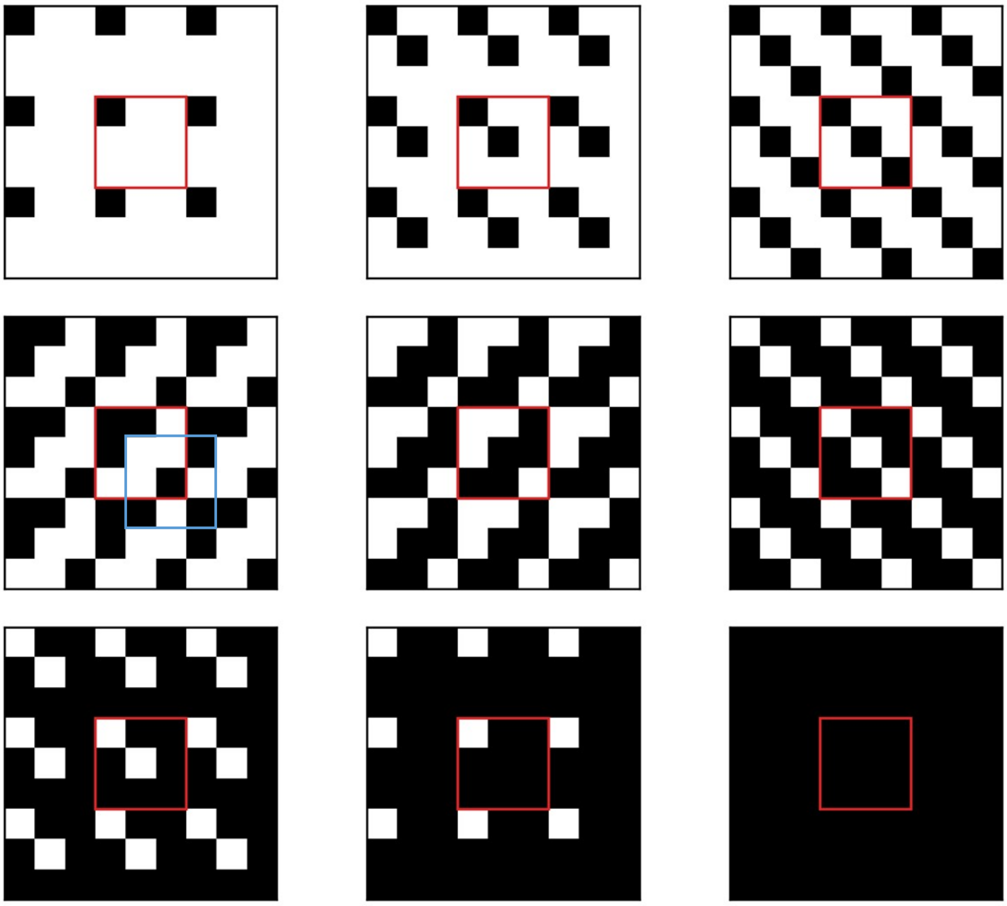


Fig. S2: A 3×3 tiling of the 3×3 base matrix generated from the algorithm in the main text. The base image is highlighted by a red square. The first base with no filling is omitted as it is just a blank image. The base matrix here for a filling of 4/9 and 5/9 (the inverse) appear different from the one in Scheme 1 of the main text but is identical in terms of drop distances and only corresponds to a rotation by 90° and a shift by one pixel in x and y, see blue square.

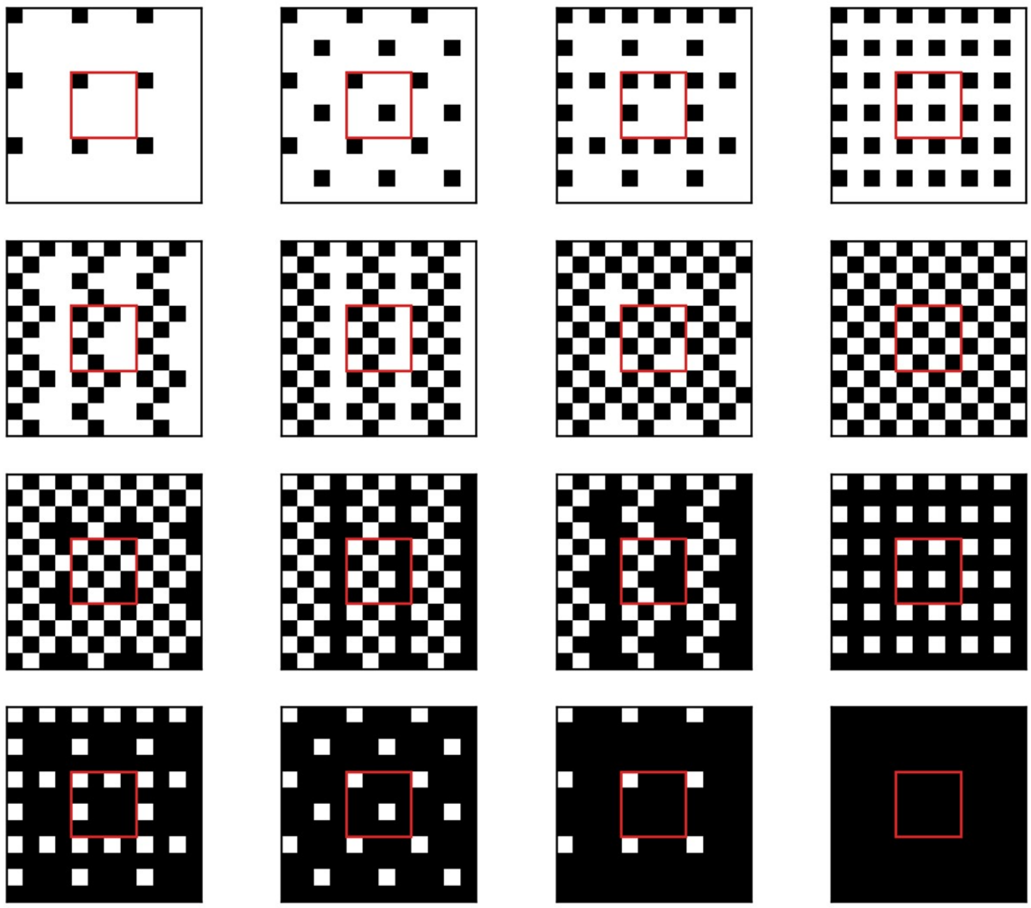


Fig. S3: A 3×3 tiling of the 4×4 base matrix generated from the algorithm in the main text. The base image is highlighted by a red square. The first base with no filling is omitted as it is just a blank image.

Contact diameter

We assume that the droplet on the surface is a spherical section of height *h* formed by the intersection of a plane and a sphere with radius *r*. The height *h* and contact diameter *D* is then given from the contact angle θ by:

$$h = r\,(1 - \cos\theta),\ D = 2r\sin\theta$$

The volume *V* of the spherical section can be calculated by the disc method as:

$$V = \pi \int_{r-h}^{r} f(x)^2 dx$$

Where $f(x)$ is the function of a sphere in the upper half plane, i.e. the positive $\sqrt{r^2 - x^2}$. This gives:

$$V = \pi\left(r^2 h - \left(\frac{r^3 - (r-h)^3}{3}\right)\right)$$

Substituting in *h* from above and solving for *r* gives:

$$r = \sqrt[3]{\frac{3V}{\pi\left(2 - 3\cos\theta + (\cos\theta)^3\right)}}$$

Using the expression above, the contact diameter *D* can then be calculated as:

$$D = 2\sin\theta\sqrt[3]{\frac{3V}{\pi\left(2 - 3\cos\theta + (\cos\theta)^3\right)}}$$

From this equation we can calculate the minimum resolution for the contact diameters to overlap along the print direction and along the diagonal as a function of the drop volume *V* and contact angle θ. This is plotted for a drop volume of 30 pl in fig. S4 below.

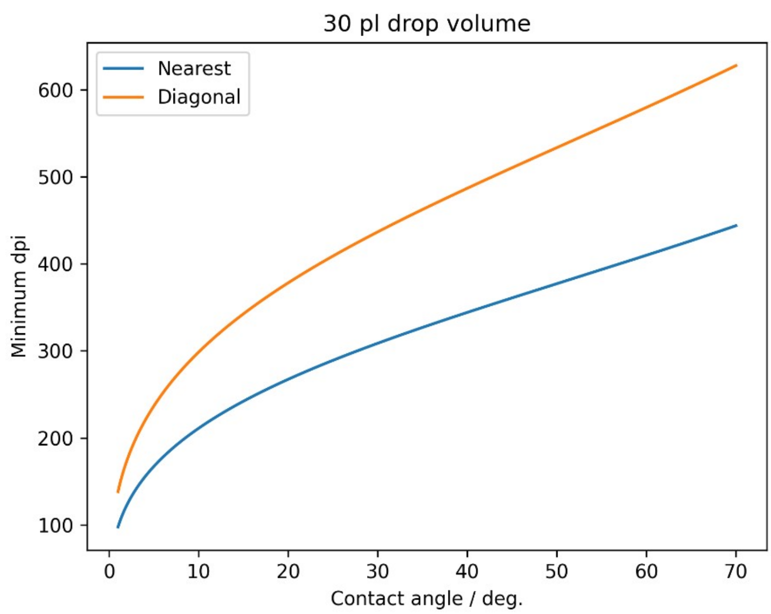


Fig. S4: The minimum resolution for the contact diameters to overlap along the print direction (nearest) and along the diagonal as a function of the contact angle θ for a drop volume $V = 30$ pl.

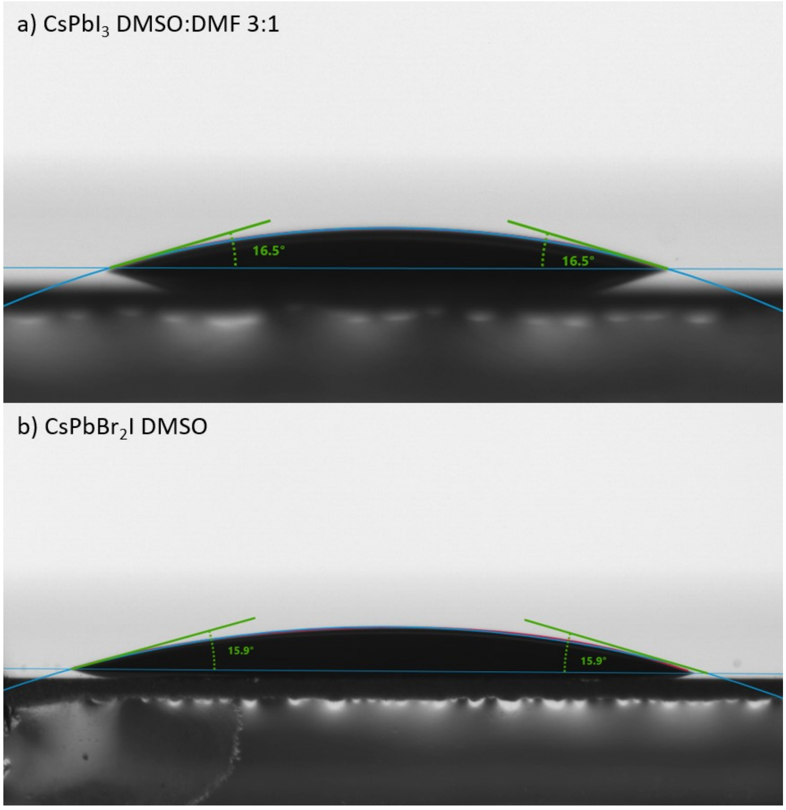


Fig. S5: The contact angle of 1 mol/l of a) $CsPbI_3$ in a 3:1 DMSO:DMF mixture and b) $CsPbBr_2I$ in DMSO on a uv ozone cleaned substrate of quartz glass.

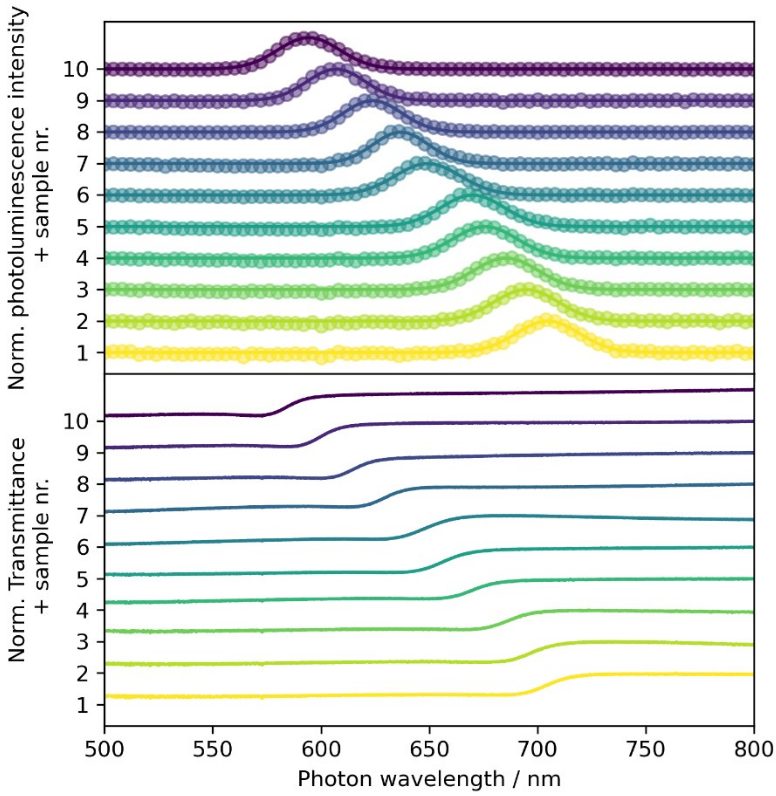


Fig. S6: The normalized PL and Transmittance spectra as a function of the photon wavelength for all 10 samples of the combinatorial library printed at 350 dpi.

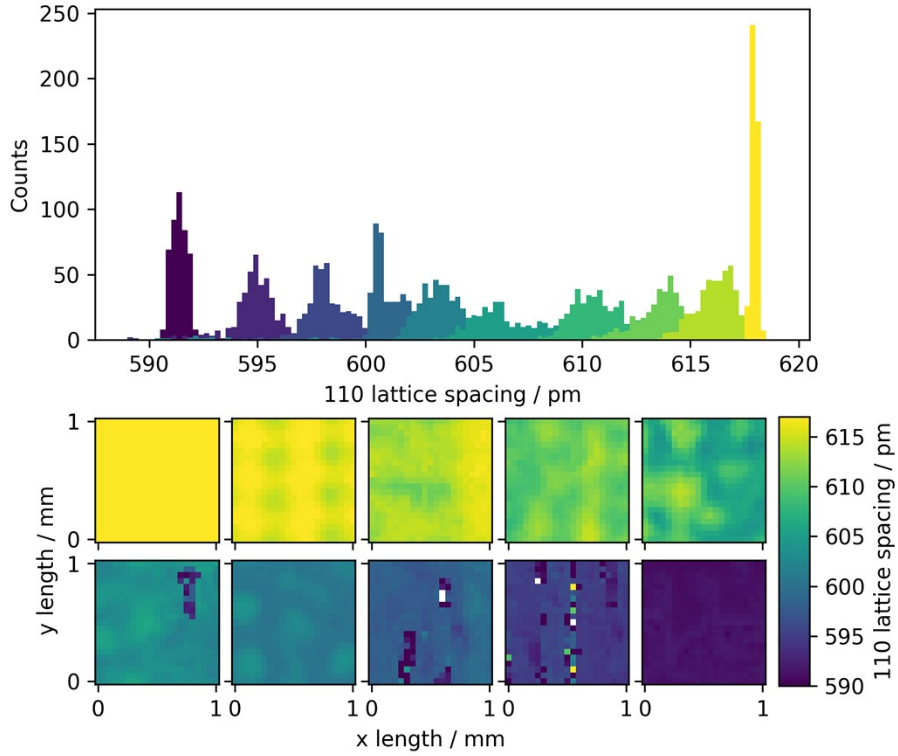


Fig. S7: Bottom, heatmap of the 110 lattice spacings mapped with 50 µm resolution in the centre of each of the samples for the library printed at 250 dpi. Top, histogram of the same lattice spacing's for each of the ten samples.

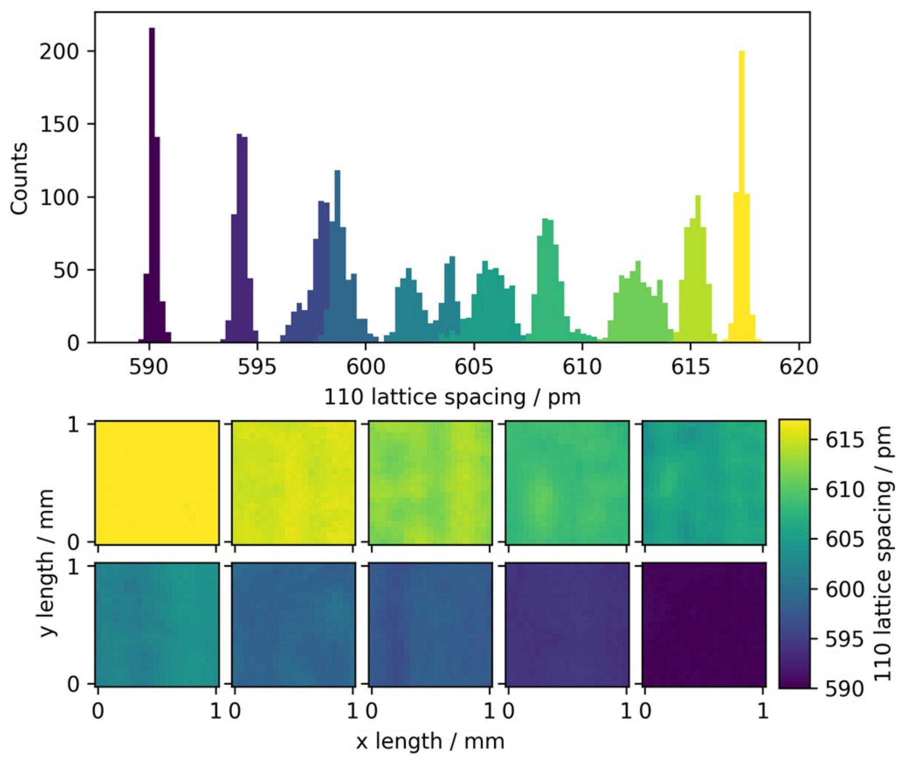


Fig. S8: Bottom, heatmap of the 110 lattice spacings mapped with 50 µm resolution in the centre of each of the samples for the library printed at 350 dpi. Top, histogram of the same lattice spacing's for each of the ten samples.

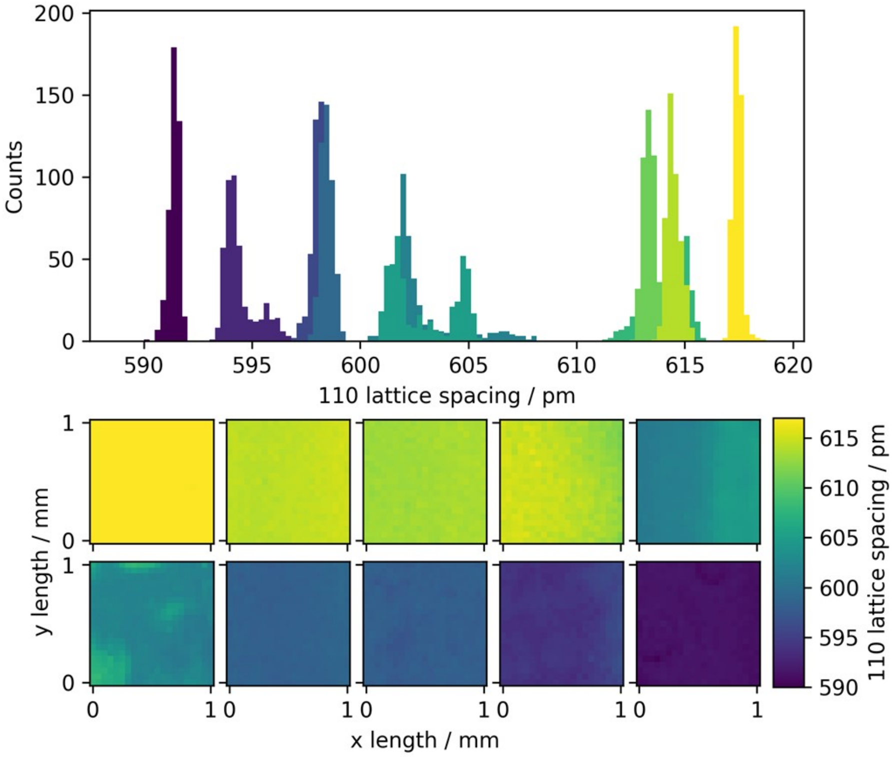


Fig. S9: Bottom, heatmap of the 110 lattice spacings mapped with 50 µm resolution in the centre of each of the samples for the library printed at 450 dpi. Top, histogram of the same lattice spacing's for each of the ten samples.

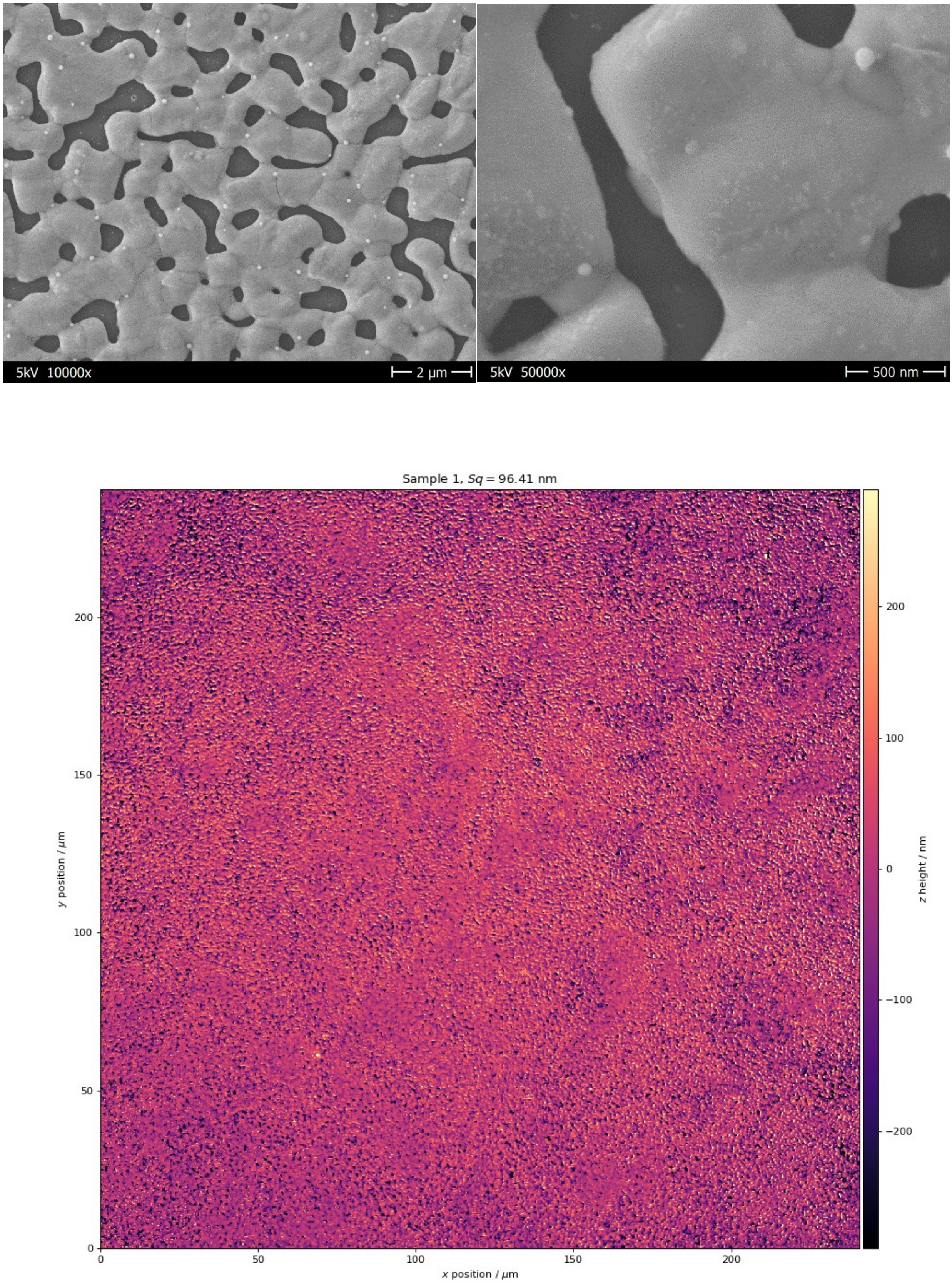
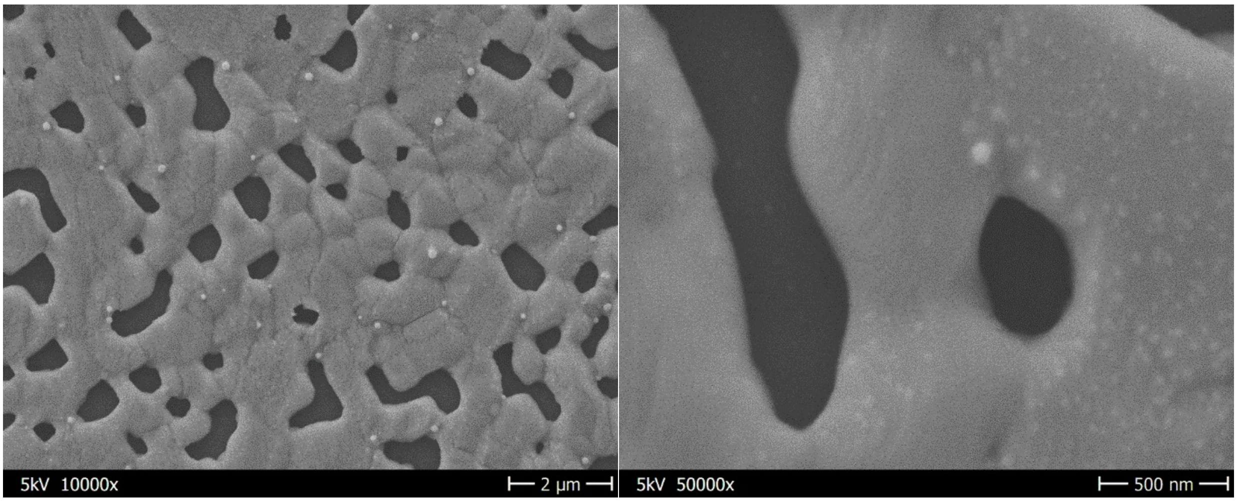


Fig. S10: SEM (top) and CLSM height profile (bottom) for sample 1 of the library printed at 350 dpi.



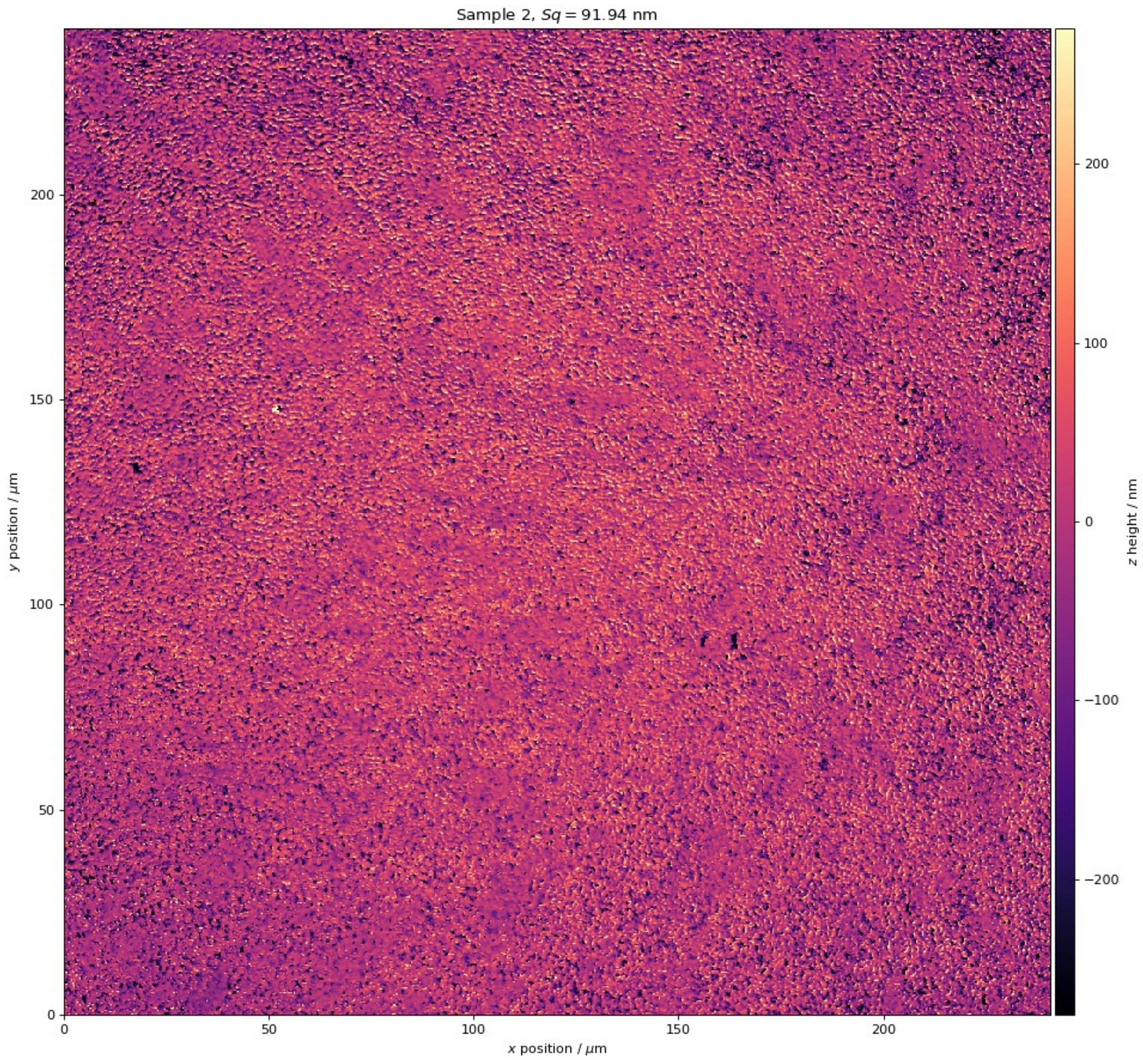
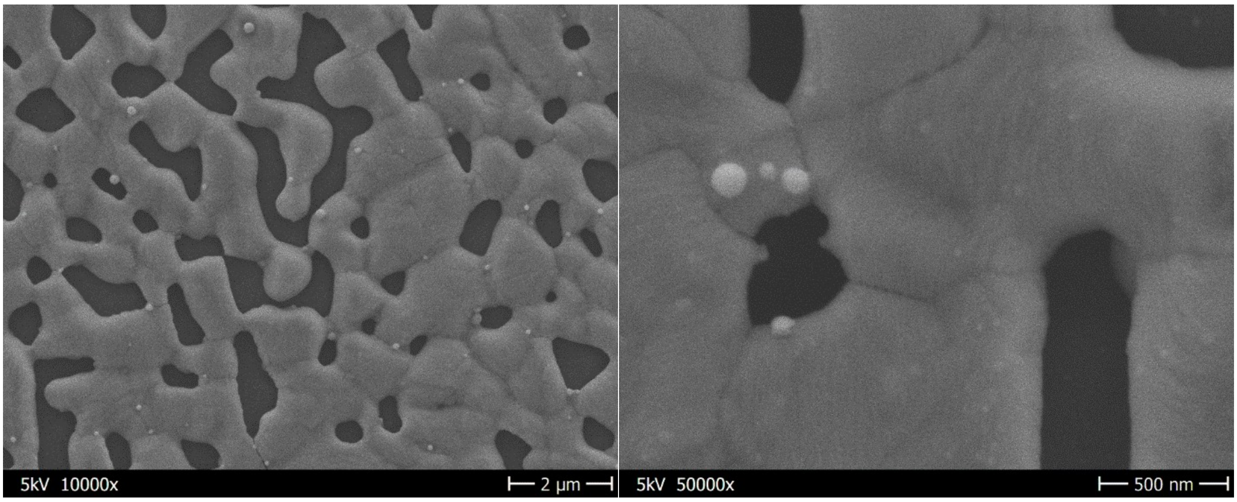


Fig. S11: SEM (top) and CLSM height profile (bottom) for sample 2 of the library printed at 350 dpi.



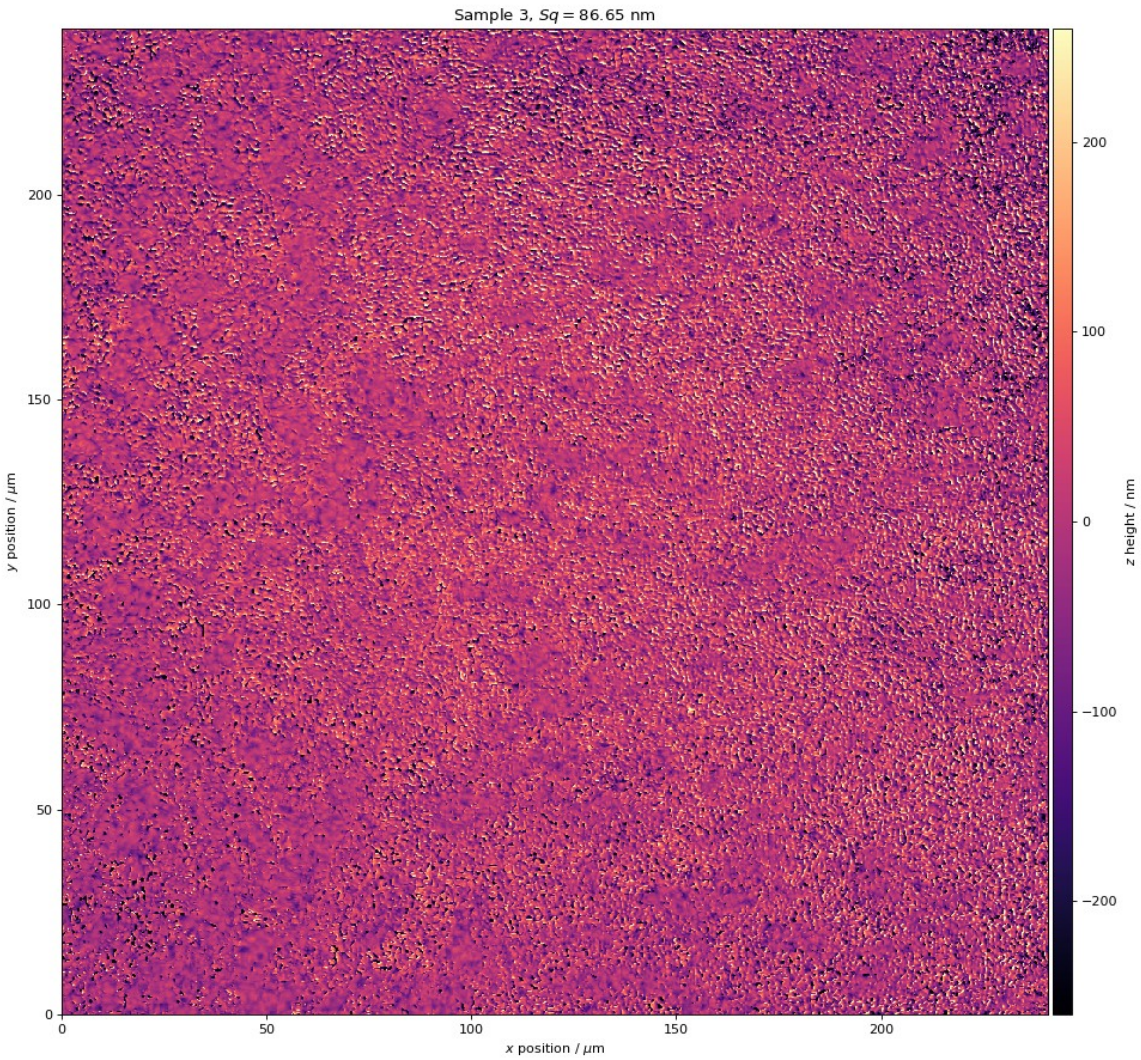


Fig. S12: SEM (top) and CLSM height profile (bottom) for sample 3 of the library printed at 350 dpi.

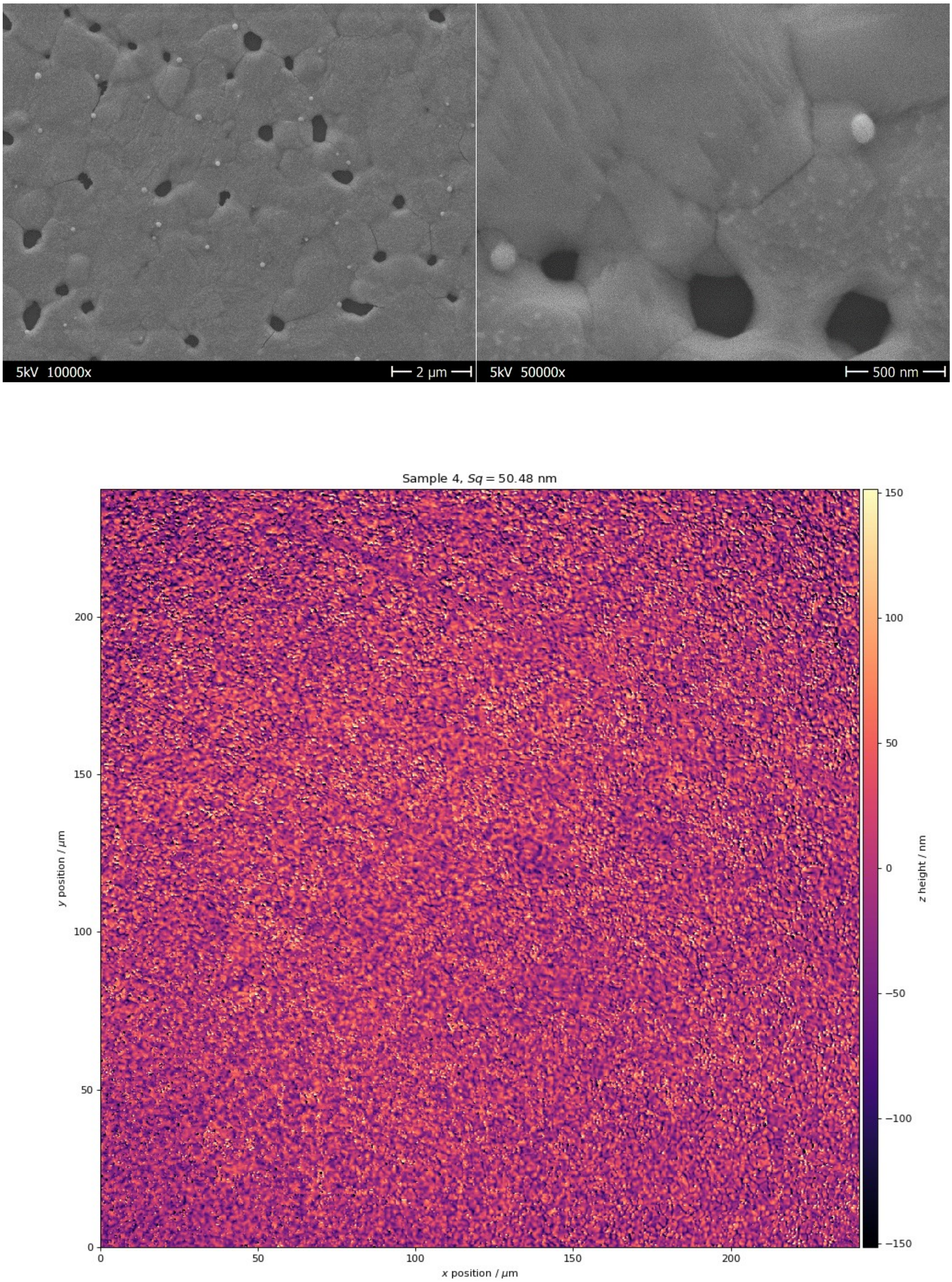
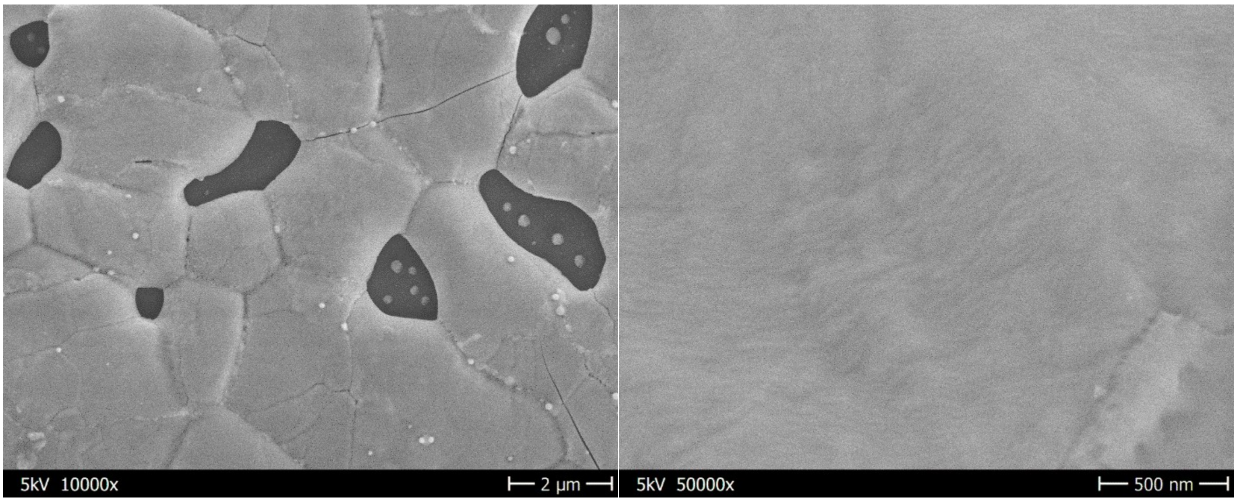


Fig. S13: SEM (top) and CLSM height profile (bottom) for sample 4 of the library printed at 350 dpi.



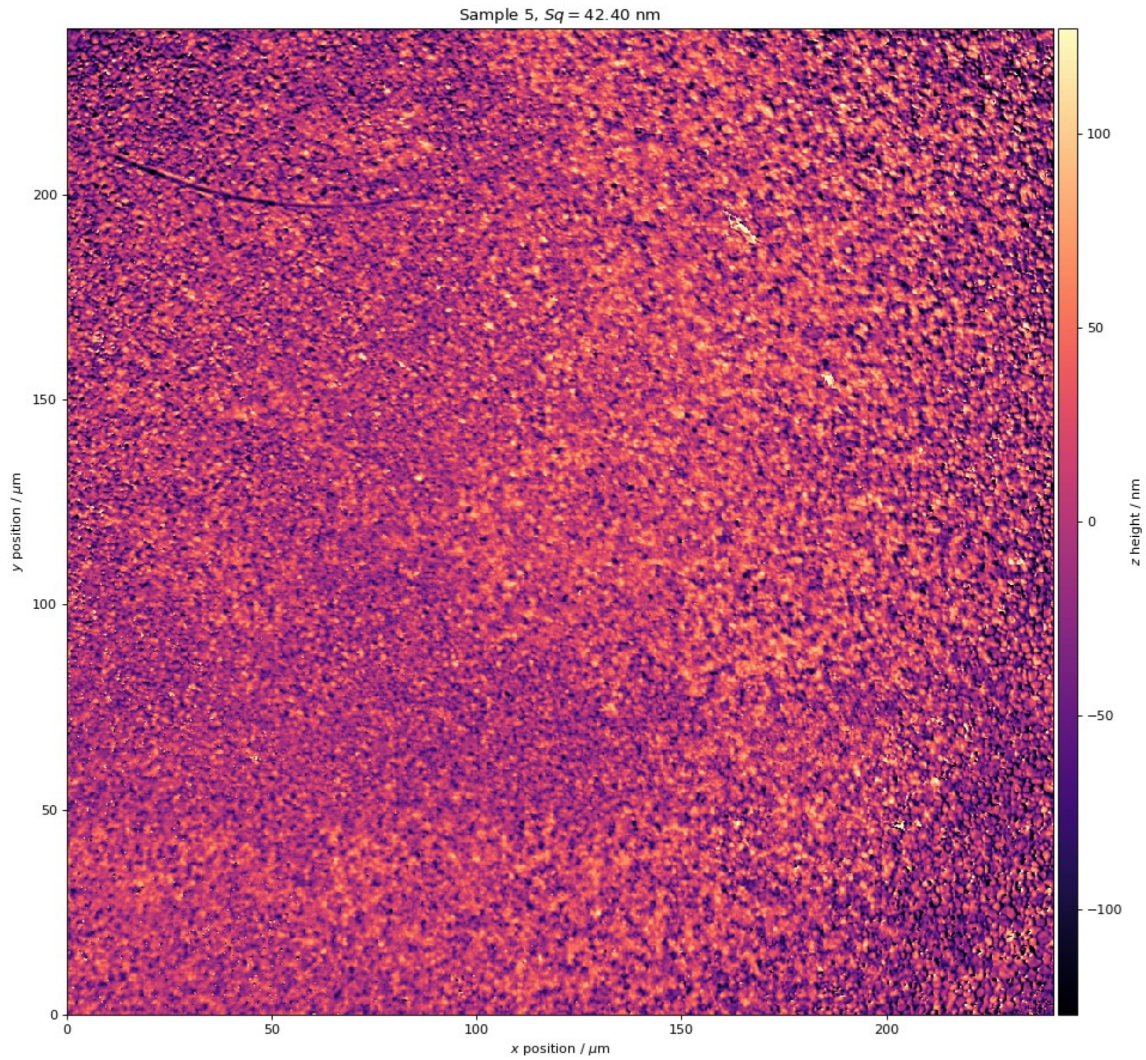
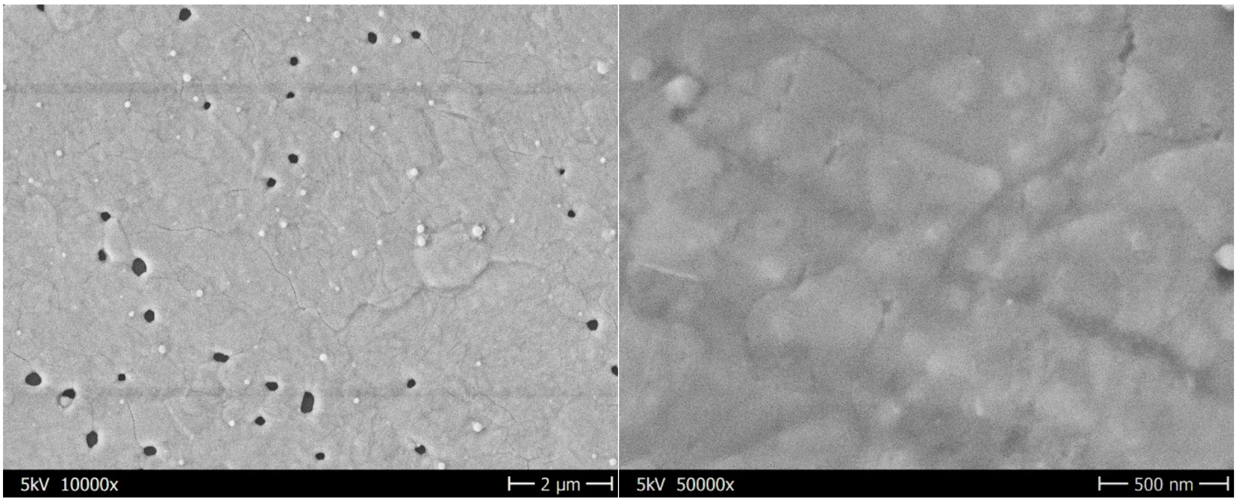


Fig. S14: SEM (top) and CLSM height profile (bottom) for sample 5 of the library printed at 350 dpi.



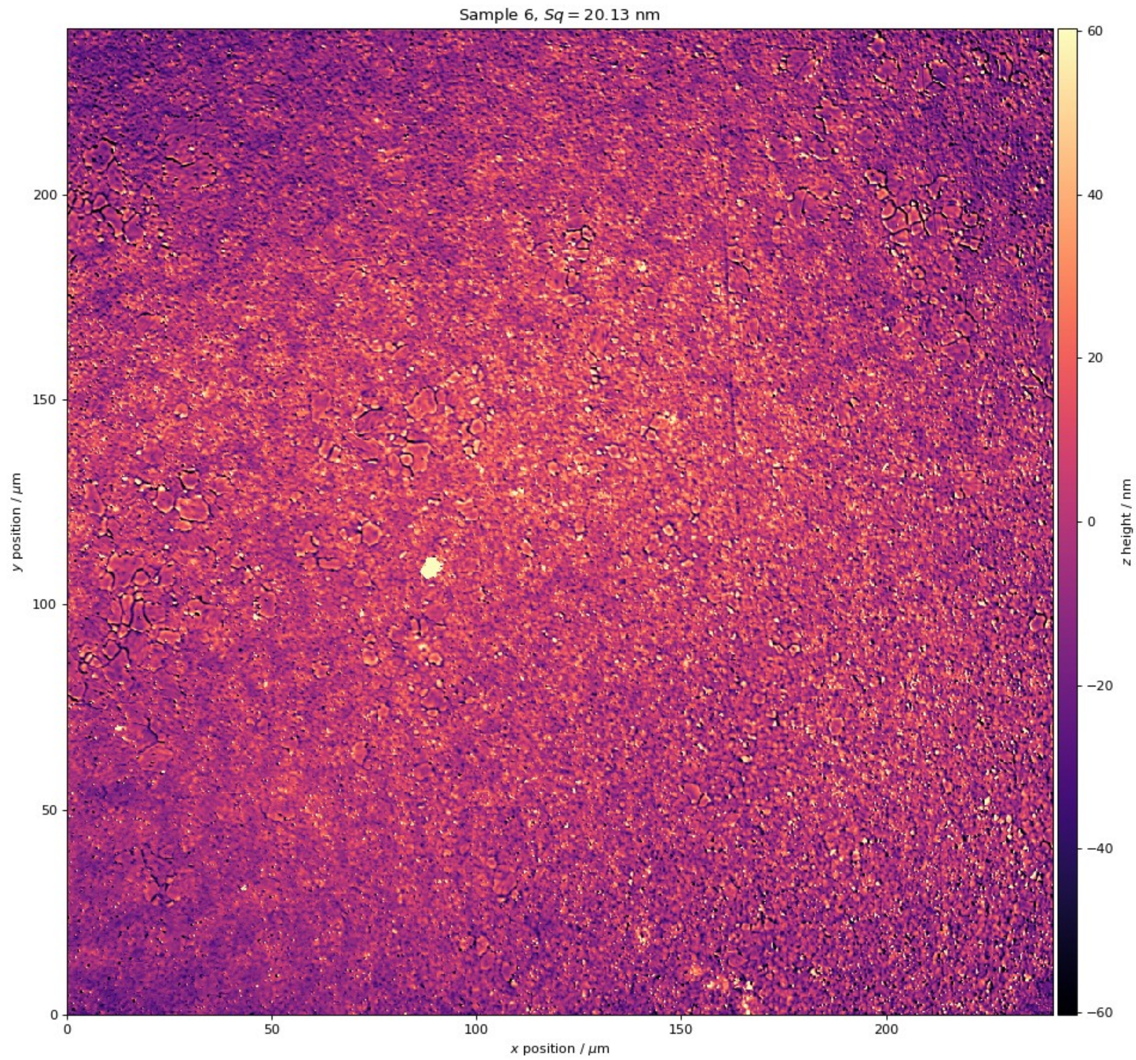
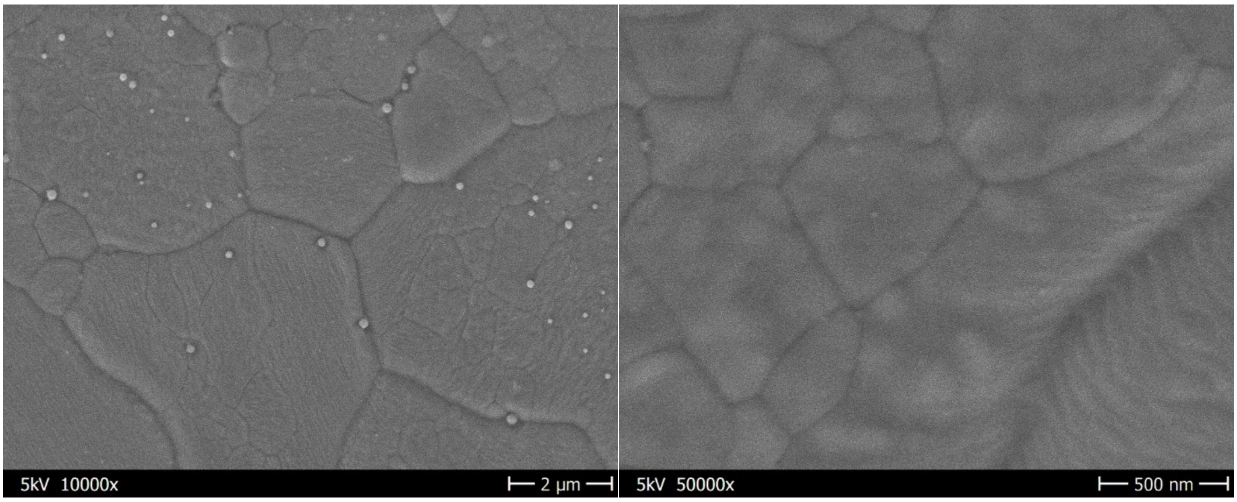


Fig. S15: SEM (top) and CLSM height profile (bottom) for sample 6 of the library printed at 350 dpi.



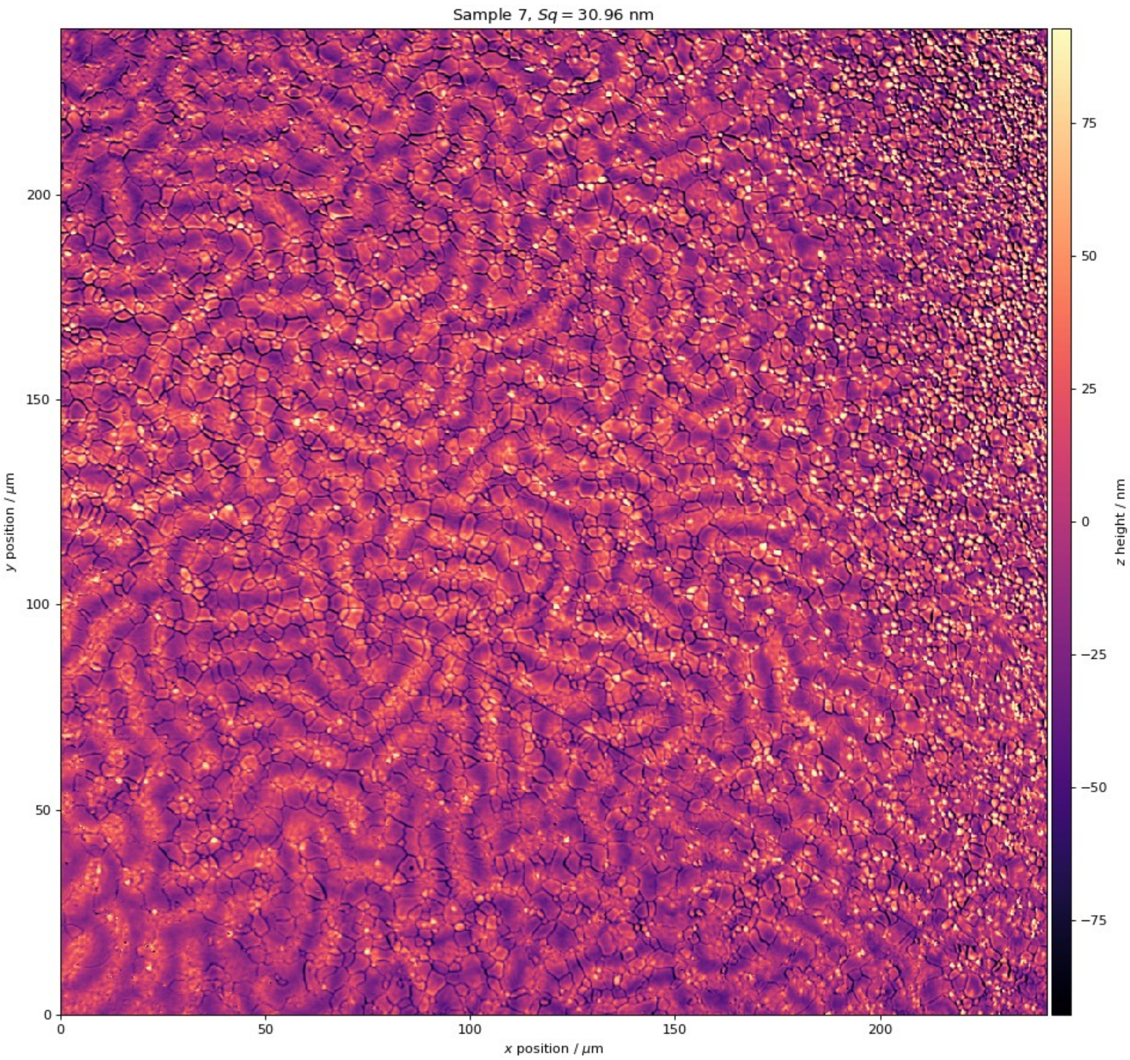
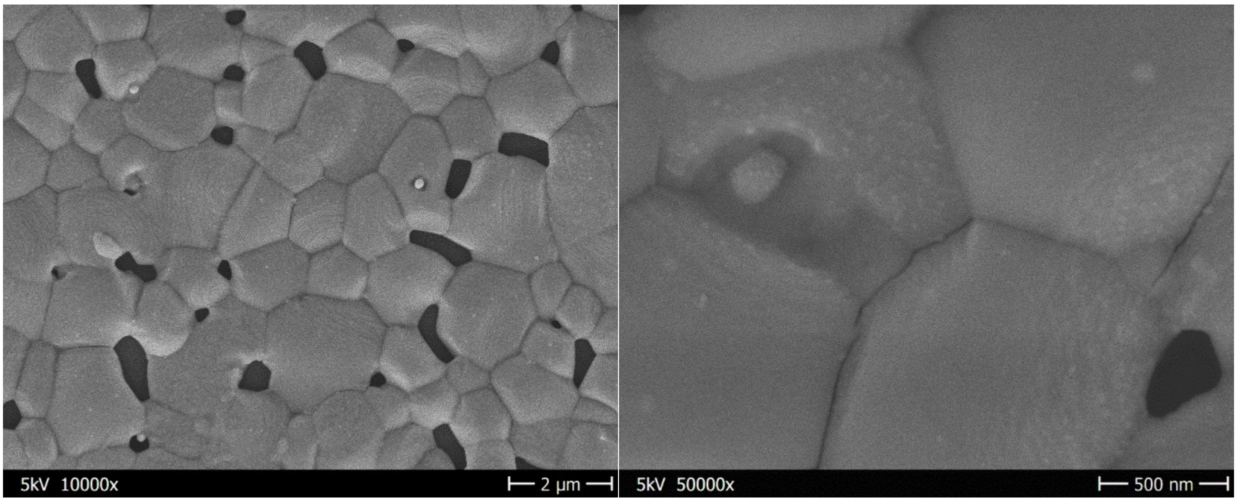


Fig. S16: SEM (top) and CLSM height profile (bottom) for sample 7 of the library printed at 350 dpi.



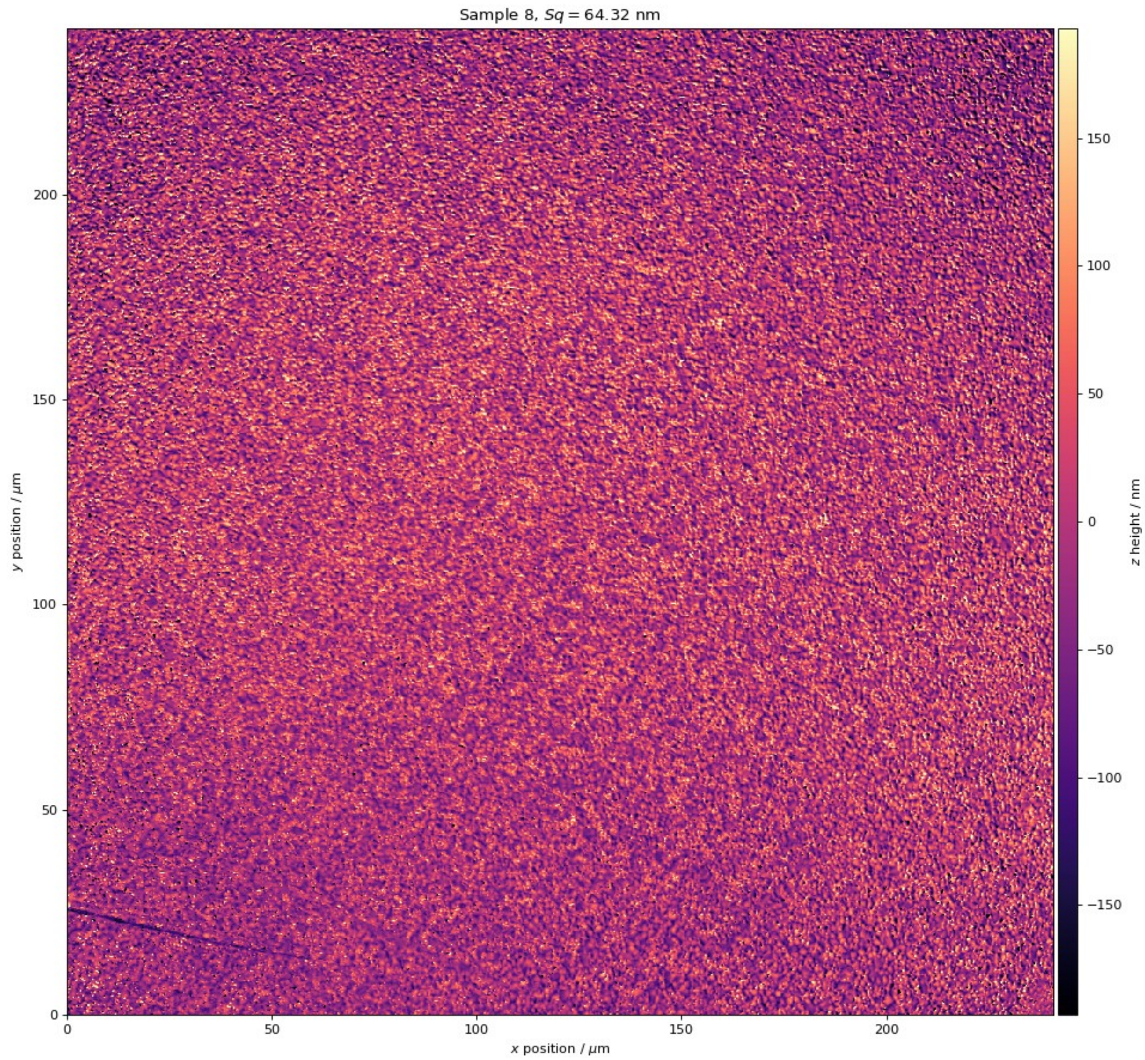
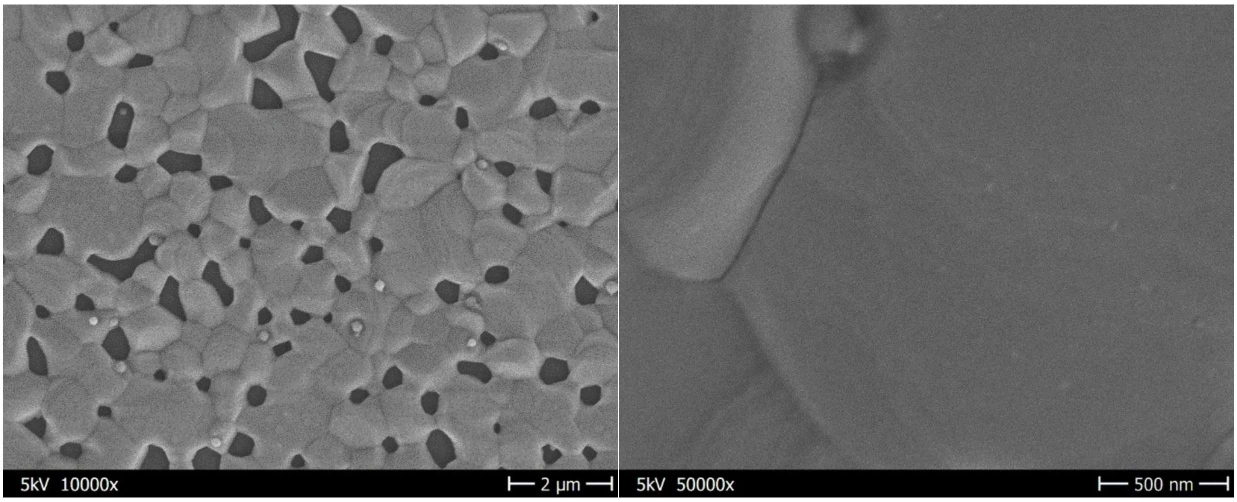


Fig. S17: SEM (top) and CLSM height profile (bottom) for sample 8 of the library printed at 350 dpi.



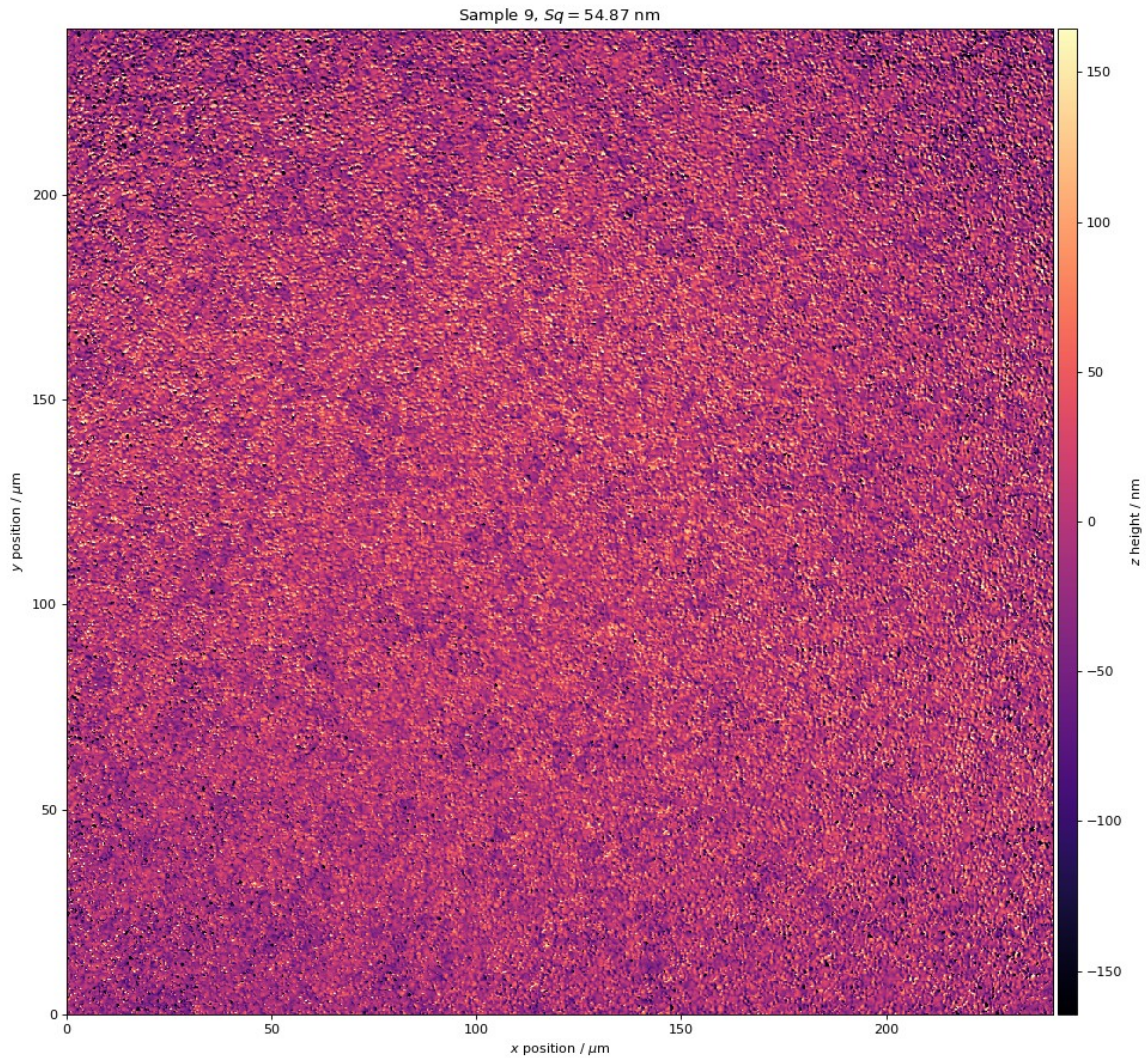
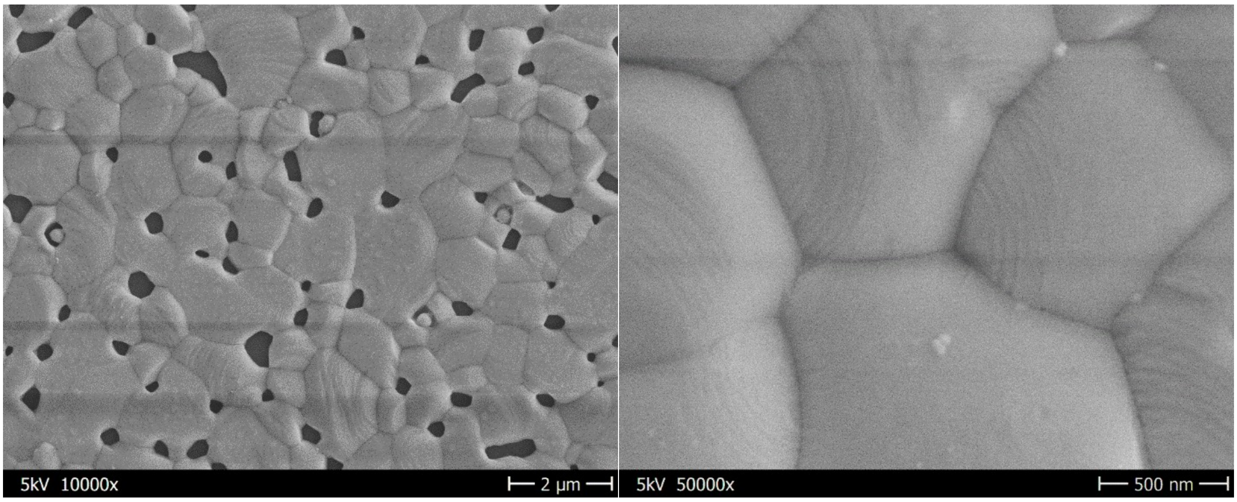


Fig. S18: SEM (top) and CLSM height profile (bottom) for sample 9 of the library printed at 350 dpi.



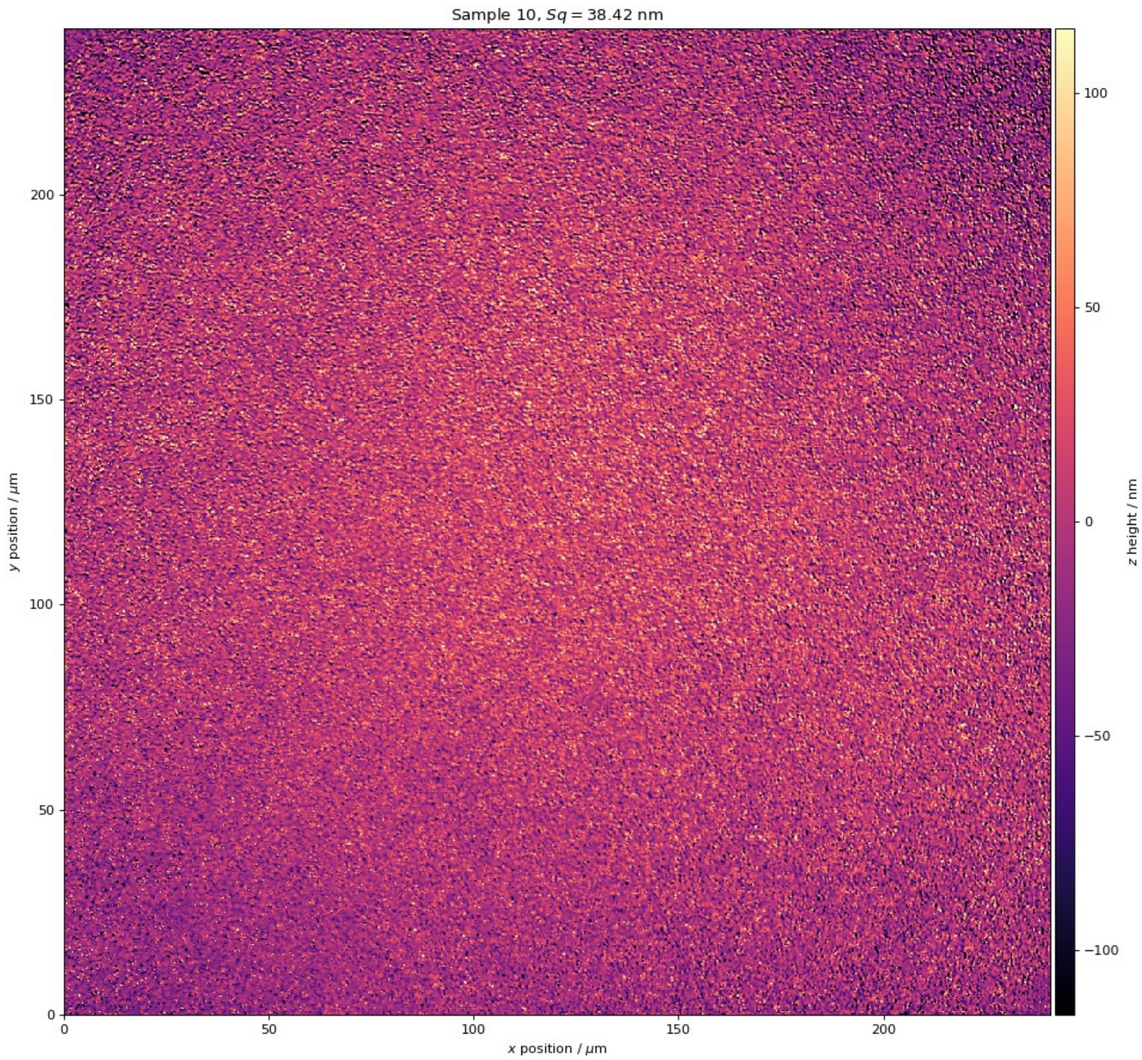


Fig. S19: SEM (top) and CLSM height profile (bottom) for sample 10 of the library printed at 350 dpi.

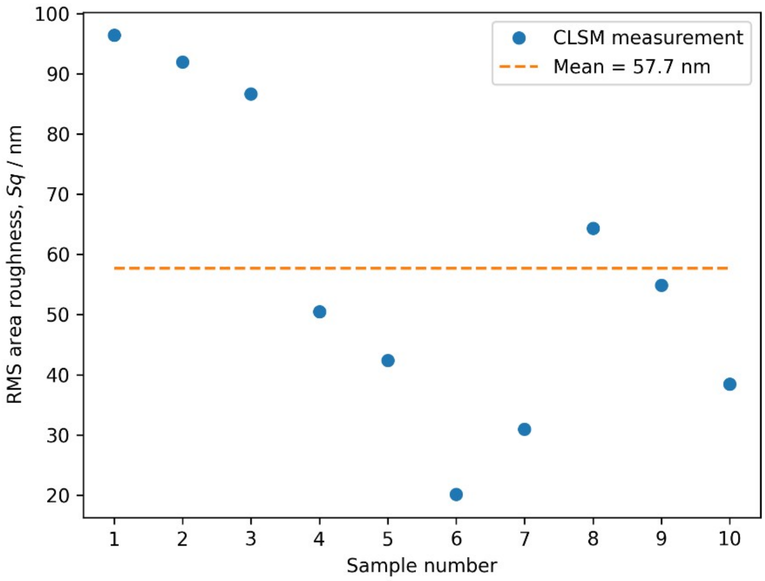


Fig. S20: Root mean square area roughness, *Sq*, for all samples of the library printed at 350 dpi.

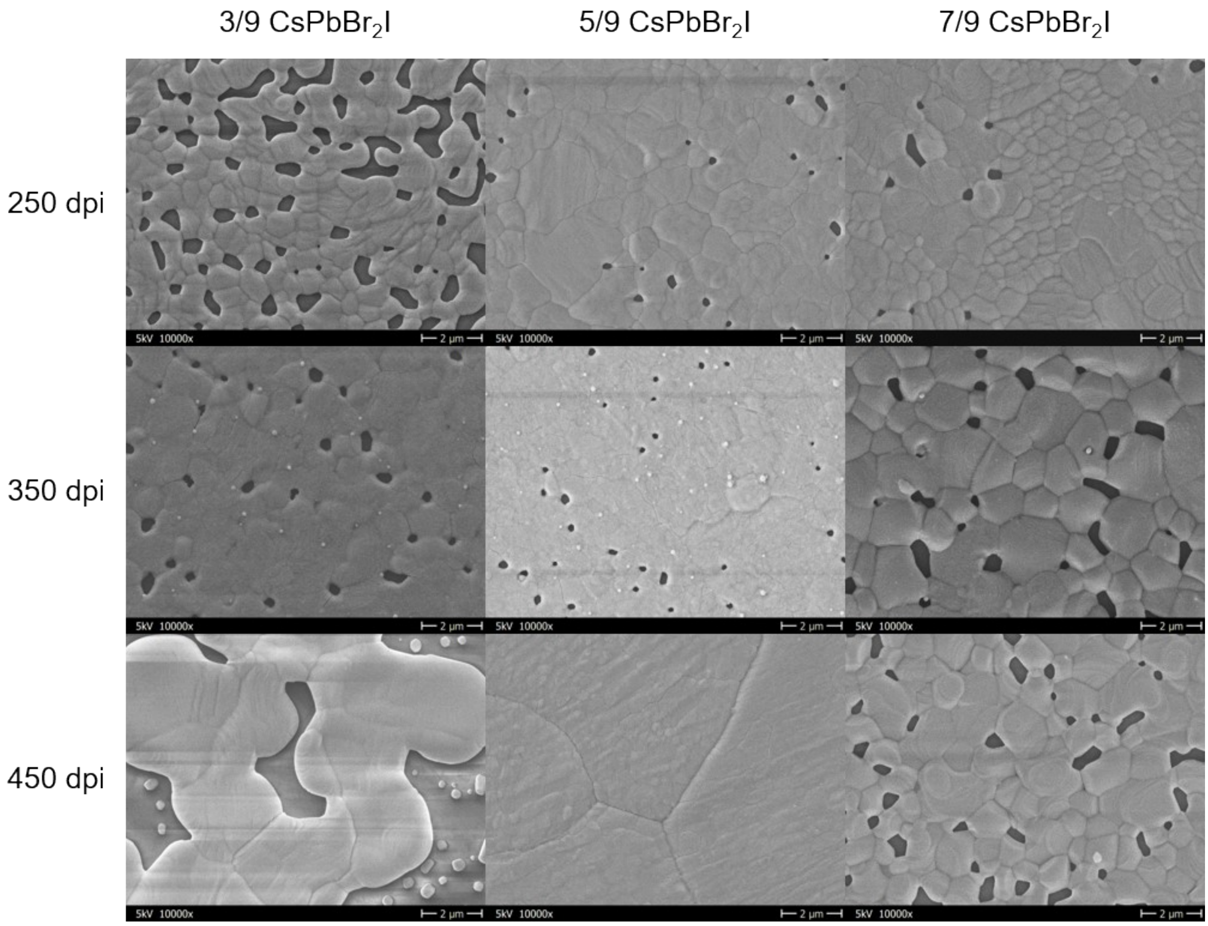


Fig. S21: From left to right, SEM pictures of the samples with a filling level of 3/9, 5/9 and 7/9 of the $CsPbBr_2I$ ink for, top to bottom, printing resolutions of 250, 350 and 450 dpi corresponding to droplet pitches of 102, 73 and 56 µm respectively.

```python
import sys
from typing import Tuple, List

import numpy as np
from itertools import combinations
from scipy.special import comb

"""
Code for generating base matrices to use for combinatorial inkjet printing.
Please cite this publication if used.
"""


def combs(a: np.ndarray, r: int) -> np.ndarray:
    """
    Return successive r-length combinations of elements in the array a.
    Should produce the same output as array(list(combinations(a, r))), but
    faster.
    From:
    https://stackoverflow.com/questions/16003217/n-d-version-of-itertools-
combinations-in-numpy

    :param a: Array with elements to make combinations from
    :type a: numpy.ndarray
    :param r: Number of elements to combine
    :type r: int
    :return: 2d array of all combinations
    :rtype: numpy.ndarray
    """
    a = np.asarray(a)
    dt = np.dtype([('', a.dtype)] * r)
    b = np.fromiter(combinations(a, r), dt)
    return b.view(a.dtype).reshape(-1, r)
```

```python
def droplet_position_optimization(n: int, k: int) -> Tuple[List[int], List[int]]:
    """
    Function for generating a base matrix with side n and filling k.

    :param n: Side length of base matrix
    :type n: int
    :param k: Filling of the base matrix
    :type k: int
    :return: Tuple of two lists of the optimal droplet positions,
             one for the x position and one for the corresponding y positions
    :rtype: Tuple[List[int], List[int]]
    """
    x, y = np.indices((n, n))
    pos = combs(np.arange(n ** 2), k)
    x_base_pos = x.flatten()[pos]
    y_base_pos = y.flatten()[pos]
    points = comb(n ** 2, k, exact=True)
    x_pos = np.zeros((points, k * 9))
    y_pos = np.zeros((points, k * 9))
    count = 0
    for y_rep in [-1, 0, 1]:
        for x_rep in [-1, 0, 1]:
            roi = slice(k * count, k * (count + 1))
            x_pos[:, roi] = x_base_pos + (x_rep * n)
            y_pos[:, roi] = y_base_pos + (y_rep * n)
            count += 1
    x_pos = np.tile(x_pos, (k, 1, 1))
    y_pos = np.tile(y_pos, (k, 1, 1))
    distances = np.sqrt(
        np.power(x_pos - np.broadcast_to(x_base_pos, (k * 9, points, k)).T, 2)
        + np.power(y_pos - np.broadcast_to(y_base_pos, (k * 9, points, k)).T, 2)
    )
    distances[np.where(distances < 1)] = (2 * n) ** 2
    min_dist = np.min(distances, axis=2)
    worst_point = np.argmin(min_dist, axis=0)
    if np.max(min_dist[worst_point, np.arange(points)]) > 1:
        best_option_index = np.argmax(min_dist[worst_point, np.arange(points)])
    else:
        worst_min_dist = np.min(min_dist, axis=0)
        min_dist_copy = min_dist
        min_dist_copy[np.where(min_dist != worst_min_dist)] = 0
        best_option_index = np.argmin(np.sum(min_dist_copy, axis=0))
    return x_base_pos[best_option_index], y_base_pos[best_option_index]
```

```python
def generate_bases(size: int) -> List[np.ndarray]:
    """
    Function for generating all base matrices for a certain size.

    :param size: Side length of the base matrix
    :type size: int
    :return: A list of the base matrices as size*size numpy arrays
    :rtype: List[numpy.ndarray]
    """
    if size > 5:
        print('For size above 4 the algorithm takes a long time.')
    high = int(np.ceil((size ** 2 + 1) / 2)) - 1
    bases = [np.zeros((size, size), dtype=bool)]
    for fill in range(high):
        best_x, best_y = droplet_position_optimization(size, fill + 1)
        base = np.zeros((size, size), dtype=bool)
        for idx in range(len(best_x)):
            base[best_y[idx], best_x[idx]] = True
        bases.append(base)
    remaining = size ** 2 + 1 - len(bases)
    for idx in range(remaining):
        bases.append(np.invert(bases[remaining - idx - 1]))
    return bases
```

```python
if __name__ == "__main__":
    import os
    import matplotlib.pyplot as plt
    from matplotlib import patches
    from PIL import Image

    base_size = 0
    if len(sys.argv) != 3:
        print("Please provide 2 arguments\n" +
              "    1st: Size of the base\n" +
              "    2nd: Location to save images")
    try:
        base_size = int(sys.argv[1])
    except ValueError:
        print("Unable to convert first argument to integer.")
        sys.exit(-1)

    save_dir = sys.argv[2]
    if not os.path.isdir(save_dir):
        print('"%s" is not a directory' % save_dir)
        sys.exit(-1)
    test_bases = generate_bases(base_size)

    fig, axs = plt.subplots(ncols=base_size, nrows=base_size, figsize=(6, 5))
    for base_idx, test_base in enumerate(test_bases):
        img = Image.fromarray(test_base.T)
        img.save(
            os.path.join(
                save_dir,
                'base_%d_fill_%d_of_%d.png' % (base_size,
                                                base_idx,
                                                len(test_bases))
            ),
            bits=1, optimize=True
        )
        if base_idx > 0:
            ax = axs[(base_idx - 1) // base_size, (base_idx - 1) % base_size]
            ax.imshow(np.tile(test_base, [3, 3]), cmap='binary', vmin=0, vmax=1)
            ax.set_xticks([])
            ax.set_yticks([])
            ax.add_patch(
                patches.Rectangle((base_size - 0.5, base_size - 0.5),
                                  base_size, base_size, edgecolor='C3',
                                  fill=False)
            )

    fig.tight_layout()
    fig.savefig(os.path.join(save_dir, 'base_%d.png') % base_size, dpi=300)
```