Supplementary Information

Contents

Image Analysis and Pair Distribution Function (PDF) Computation 3
Overview: Multi–Image Composition 3
Image Preprocessing 3
Contour Detection and Fiber Segmentation 4
Fiber Sub-Segmentation and Data Export 4
Pairwise Distance Computation 5
Pair Distribution Function (PDF) 5
Pseudo-3D Analysis via Z-Stack Slices 5
Supplementary Code 7
Python Code for Image Analysis and Pair Distribution Function Computation . . . 7

Image Analysis and Pair Distribution Function (PDF) Computation

Overview: Multi–Image Composition

The `Stack` function was used to merge multiple grayscale images into a single RGB composite. For visualization (Figure S1), each slice was assigned to a distinct color channel to highlight differences and overlaps between slices. For the quantitative analysis pipeline, the slices were also combined into an RGB image, but mapped onto the same channel intensity range to ensure consistent segmentation and to avoid channel-dependent contour-detection artifacts. This composite enables qualitative assessment of fiber continuity across depth.

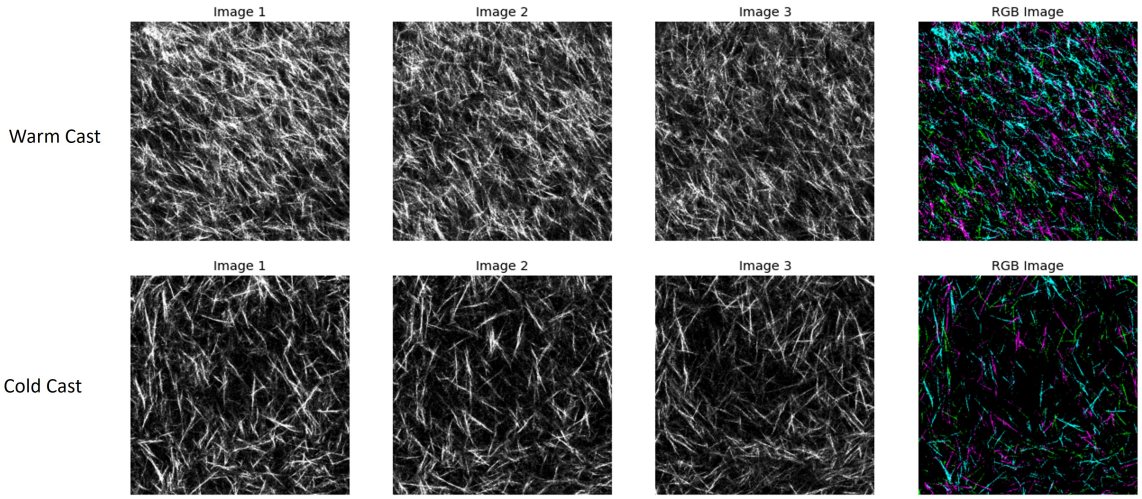


Figure S1: Stacked Image Analysis. *The figure illustrates the stacking function where three layers are merged. Each image is converted to grayscale and assigned a unique color representation. The final stacked image is generated by combining these processed layers, effectively visualizing structural differences across the input images. The image dimensions are 400 μm × 400 μm.*

This composite provides a visual reference for structural variations between slices that the subsequent quantitative analyses aim to resolve.

Image Preprocessing

All images were converted to grayscale and smoothed with a `Gaussian blur` to suppress high-frequency noise. Gaussian filtering is a standard preprocessing method to stabilize pixel intensities before thresholding [1, 2]. Otsu's global thresholding (`cv2.THRESH_OTSU`) [3, 4] was computed to obtain an objective estimate of the threshold range, after which a fixed threshold value (equal to or very close to the Otsu value) was applied to standardize binarization across images. Because the fibers were consistently brighter than the background, this produced binary masks with fibers as white structures on a black background.

Morphological `opening` (erosion followed by dilation) and `closing` (dilation followed by erosion) were then applied to remove short bright artifacts and fill small gaps, respectively [5, 6]. The effect of these preprocessing operations is illustrated in Figure S2.

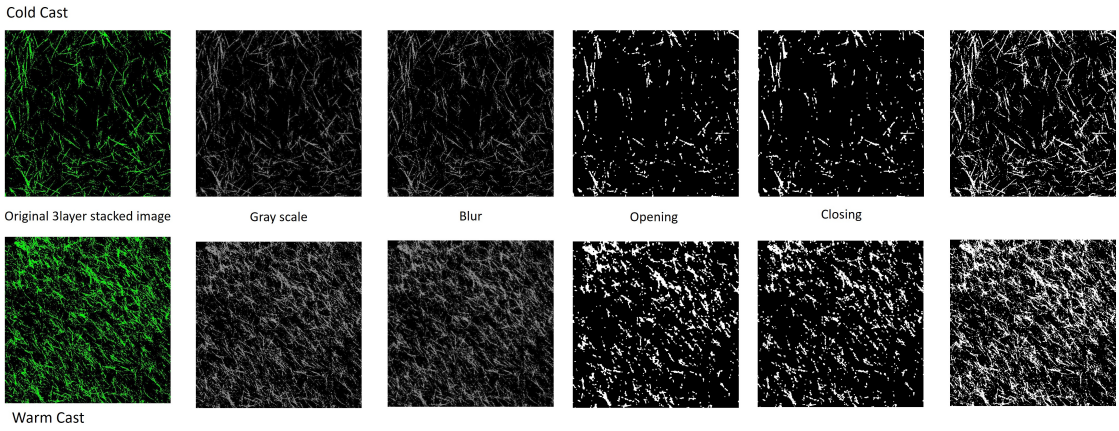


Figure S2: Morphological Processing of Fiber Structures. *Comparison of Cold Cast and Warm Cast fiber structures through sequential image processing steps, including grayscale conversion,* `Gaussian blur`*, thresholding (*`cv2.THRESH_OTSU`*), and morphological transformations (*`opening`* and `closing`*). Image dimensions: 400 µm × 400 µm.*

Contour Detection and Fiber Segmentation

After morphological cleanup, `cv2.findContours` was used to detect discrete fiber objects. Small contours below a minimum area threshold were discarded to remove fiber tails and noise fragments.

For each contour retained, key geometric features were extracted to characterize fiber properties. The contour centroid was computed using image moments via `cv2.moments`. The centroid coordinates were calculated as

$$C_x = \frac{M_{10}}{M_{00}}, \qquad C_y = \frac{M_{01}}{M_{00}},$$

where M_{ij} are spatial moments [2].

Next, we computed the convex hull of each contour, which provides the smallest convex polygon enclosing the fiber. The convex hull smooths local concavities and provides a simplified geometric boundary. We also fit an ellipse to each contour using `cv2.fitEllipse`, yielding an estimate of fiber major-axis length and in-plane orientation. These properties (centroid, convex hull, length, orientation) were stored in a structured table for downstream analysis.

In cases where one contour enclosed several touching or crossing fibers (aggregated fibers), an additional segmentation step was performed. Radial lines were projected from the centroid to vertices of the convex hull to partition the merged shape. This technique leverages convexity defects to identify concave boundaries that separate adjacent fibers, similar to methods used for splitting overlapping objects in microscopy (e.g., nuclei or particles) [7, 8]. This process isolated individual fibers from complex merged contours.

Fiber Sub-Segmentation and Data Export

To improve spatial accuracy, each fiber-containing contour was represented by multiple sub-segments rather than by a single centroid. For each contour, radial line elements were generated by connecting the centroid to the vertices of its convex hull. Along each of these line elements, a series of evenly spaced circular sampling points was placed, producing a set of sub-segment centers that approximate the spatial extent of the fiber region. Representing extended fibrillar structures as multiple point elements is consistent with spatial statistics approaches that convert line patterns into multi-point approximations for distance-based analysis [9, 10].

Each sub-segment was assigned coordinates and metadata identifying its parent contour, and all sub-segment data were exported to a CSV file for the subsequent computation of pairwise distances. Figure S3 shows examples of labeled contours and their subdivided sampling points.

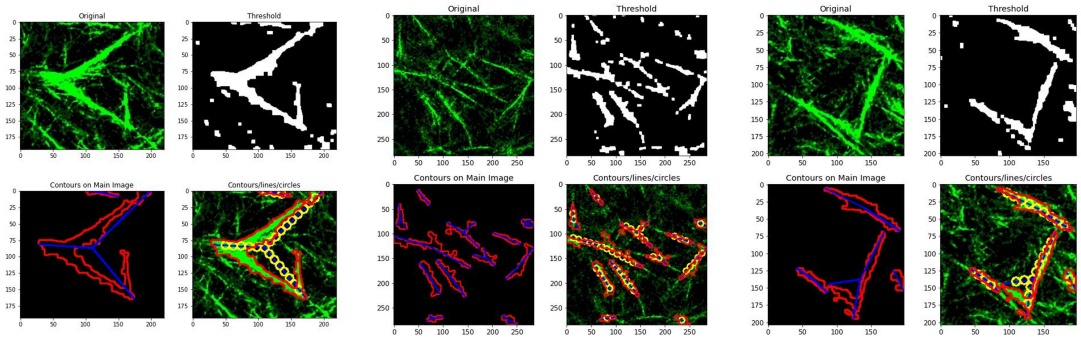


Figure S3: Fiber Detection and Contour Analysis. *Illustration of the step-by-step fiber detection process. The original image is thresholded to extract fiber structures, followed by contour detection using* `cv2.findContours`*. Identified contours are further analyzed to extract key properties, including the center of mass (using* `cv2.moments()`*), convex hull (using* `cv2.arcLength()`*), and fiber orientation (via ellipse fitting). Individual fibers are segmented into smaller circular regions, and their locations are labeled and saved as a CSV file. Pairwise distances between segmented fiber regions are calculated and stored separately. Image dimensions: 100 μm × 100 μm.*

Pairwise Distance Computation

With all fiber sub-segment coordinates collected, we computed pairwise Euclidean distances using NumPy's linear-algebra norm function (`np.linalg.norm`). Pairwise distance computation is the standard foundation for spatial pattern analysis in materials science and tissue imaging [11, 12]. For every pair of segments i and j belonging to different fibers, the distance

$$d_{ij} = \|x_i - x_j\|$$

was recorded, generating the set of inter-segment separations used to construct the pair distribution function.

Pair Distribution Function (PDF)

The spatial organization of fibers was quantified by constructing a pair-distribution-type function based on the inter-fiber distances. The set of distances was binned into radial intervals of width Δr to obtain a histogram $N(r)$ of pair counts as a function of separation. Because the number of possible pairs naturally increases with distance, the raw histogram was normalized by the corresponding radial sampling volume to reduce this geometric bias. The resulting normalized distance distribution acts as a PDF-like radial summary of the spatial relationships between fibers.

For a spatially random arrangement, the normalized distribution is expected to be relatively flat, whereas peaks or troughs indicate preferred spacings or avoidance between fibers [13, 14]. The final normalized distance profile was plotted as a function of radius to interpret characteristic fiber spacings and pairwise correlations.

Pseudo-3D Analysis via Z-Stack Slices

Three confocal z-stack images separated by approximately 16 μm were first combined into a single composite image to approximate a pseudo-3D representation of the fiber network. This slice aggregation approach is conceptually similar to stereological sampling, in which multiple thin 2D sections are used to infer aspects of a 3D structure [9, 15]. Because the slices are closely spaced, fibers appear in overlapping lateral positions, and combining them enhances the sampling of the underlying 3D organization.

Previous studies have shown that many structural metrics of fibrillar networks extracted from 2D sections reproduce the qualitative trends observed in full 3D reconstructions [16]. By aggregating the three adjacent axial planes into a single composite prior to distance analysis, we obtained a more representative sampling of the fiber network without requiring full volumetric imaging.

Supplementary Code

Python Code for Image Analysis and Pair Distribution Function Computation

The custom Python code used for image analysis and pair distribution function computation is provided below. The same analysis workflow was applied across the analyzed datasets, with dataset-specific differences limited primarily to the input image files and file paths.

Code Usage

In this script, `A` and `E` are used as identifiers for the two datasets being compared. For example, `A` can represents the ColFn sample and `E` can represents the ColFnCell sample under the cold cast condition. These identifiers can be replaced with other sample names or experimental groups when applying the code to different datasets.

To run the code with new data, the user should replace the image file paths in `image_paths_A` and `image_paths_E` with the locations of the corresponding input images. The output file paths should also be replaced with valid locations where the stacked images, CSV files, processed images, and PDF plots will be saved.

The code first estimates the image threshold using Otsu's thresholding method. The manually entered threshold should remain close to the Otsu-determined value and may be adjusted slightly, when necessary, to improve fiber identification and counting.

```
def Stack(image_paths, output_path, threshold=128):
    from PIL import Image
    import numpy as np
    import matplotlib.pyplot as plt


    # Load images and convert to grayscale
    images = [Image.open(img).convert('L') for img in image_paths]

    # Convert images to numpy arrays and threshold to create binary images
    image_arrays = [(np.array(img) > threshold).astype(np.uint8) for img in images]

    # Create an empty RGB image
    rgb_image = np.zeros((*image_arrays[0].shape, 3), dtype=np.uint8)

    # Define colors for each image
    colors = [(0, 255, 0), (0, 255, 0), (0, 255, 0)]
    assert len(colors) >= len(image_paths), "Not enough colors defined for the number of
    images"

    # Assign each image to a different color channel
    for img, color in zip(image_arrays, colors):
        for c in range(3):
            rgb_image[..., c] += img * color[c]

    # Create a subplot
    fig, axs = plt.subplots(1, len(images) + 1, figsize=(5*(len(images) + 1), 5))

    # Display the initial images
    for i, img in enumerate(images):
        axs[i].imshow(img, cmap='gray')
        axs[i].set_title('Image {}'.format(i+1))
        axs[i].axis('off')

    # Display the resulting image
```

```
    axs[-1].imshow(rgb_image)
    axs[-1].set_title('RGB Image')
    axs[-1].axis('off')

    # Convert the RGB array back to an image
    rgb_img = Image.fromarray(rgb_image)

    # Save the image to a filen
    rgb_img.save(output_path)

    plt.show()


# Load images
image_paths_A = [
    r"path/to/input/image_z0.jpg",
    r"path/to/input/image_z1.jpg",
    r"path/to/input/image_z2.jpg"
]

output_path_A = r"path/to/output/stack-A.jpg"


Stack(image_paths_A, output_path_A)

image_paths_E = [
    r"path/to/input/image_z0.jpg",
    r"path/to/input/image_z1.jpg",
    r"path/to/input/image_z2.jpg"
]

output_path_E = r'C:\Users\javad\Desktop\DATA FOR PAPER\CODES\Image processing result\Stack
    \stack-E.jpg'

Stack(image_paths_E, output_path_E)









import cv2
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import os

micrometers_per_pixel = 1280 / 3072

#Info of function for different images:

# Location-Image-Path:
Image_Path_A = output_path_A
Image_Path_E = output_path_E

# Location-CSV-Path:
CSV_Path_A = r"output/sample_A/fibrils.csv"
CSV_Path_E = r"output/sample_E/fibrils.csv"


```

```
# Domain to save the info-output_prefix:
output_prefix_A= r"output/sample_A/"
output_prefix_E= r"output/sample_E/"
#Set thereshoulds
manual_thresh_A=48
manual_thresh_E=48
#location to save the plots: save_PDF_PLOT
Loc_A= r"output/sample_A/Afiltered.jpg"
Loc_E= r"output/sample_E/Efiltered.jpg"

Oloc_A= r"output/sample_A/AOrientation.jpg"
Oloc_E= r"output/sample_E/EOrientation.jpg"


indfiber=r"output/processing_steps/Singlefibers/"


def process_image(image_path, output_prefix,manual_thresh,title,save_PLOT,CSV_Path):



    # Load image and convert to grayscale
    img = cv2.imread(image_path)
    img1 = img.copy()

    cv2.imshow('original', img1)
    height, width, _ = img1.shape
    print(f"The size of the image is: {width} x {height}")
    print("The size of the image is:",width *micrometers_per_pixel ,height*
    micrometers_per_pixel)
    gray = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
    blur = cv2.GaussianBlur(gray, (5, 5), 0)
    ret, thresh = cv2.threshold(blur, 0, 255, cv2.THRESH_BINARY+ cv2.THRESH_OTSU)
    print("the Otsu threshoulding value:",ret)
    ret, thresh = cv2.threshold(blur, manual_thresh, 255, cv2.THRESH_BINARY)
    img2 = img1.copy()

    cv2.imwrite(r"output/processing_steps/orig.jpg",img)
    cv2.imwrite(r"output/processing_steps/gray.jpg",gray)
    cv2.imwrite(r"output/processing_steps/blur.jpg",blur)
    cv2.imwrite(r"output/processing_steps/thresh.jpg",thresh)


    kernel = np.ones((3, 3), np.uint8)
    opening = cv2.morphologyEx(thresh, cv2.MORPH_OPEN, kernel, iterations=2)
    closing = cv2.morphologyEx(opening, cv2.MORPH_CLOSE, kernel, iterations=2)
    cv2.imshow('thereshould', closing)
    contours, hierarchy = cv2.findContours(closing, cv2.RETR_EXTERNAL, cv2.
    CHAIN_APPROX_SIMPLE)

    cv2.imwrite(r"output/processing_steps/opening.jpg",opening)
    cv2.imwrite(r"output/processing_steps/closing.jpg",closing)

    min_area = round(100/micrometers_per_pixel) # Set a minimum area threshold in UM*UM
    unit
    print("The min area of considered fiber is(um^2):",min_area * micrometers_per_pixel)
    filtered_contours = []

    # Filter contours based on the minimum area
```

```
    for cnt in contours:
        if cv2.contourArea(cnt) >= min_area:
            filtered_contours.append(cnt)


    # Prepare a DataFrame to store fibril information
    fibril_data = []

    # For each filtered contour
    for i, cnt in enumerate(filtered_contours):
        img4 = img.copy()
        blank_image = np.zeros(img4.shape, dtype=np.uint8)
        img4_with_contour = cv2.drawContours(blank_image, [cnt], -1, (0, 0, 255), 2)


        # Compute the center of the contour
        M = cv2.moments(cnt)
        cX = int(M["m10"] / M["m00"])
        cY = int(M["m01"] / M["m00"])
        center = np.array([cX, cY])
        # Draw a yellow circle at the center
        radius = 5  # Adjust the size of the circle as needed
        cv2.circle(img4_with_contour, (cX, cY), radius, (0, 255, 255), -1)  # Yellow color
        # Save the image to a file instead of displaying it
        filename = f"{indfiber}Fiber_{i+1}.jpg"
        cv2.imwrite(filename, img4_with_contour)
        # Check if directory exists and create if needed
        #if not os.path.exists(indfiber):
          #  os.makedirs(indfiber)

        # Save the image to a file
        filename = os.path.join(indfiber, f"Fiber_{i+1}.jpg")
        cv2.imwrite(filename, img4_with_contour)

        # Compute the convex hull of the contour
        epsilon = 0.1*cv2.arcLength(cnt,True)
        simplified_cnt = cv2.approxPolyDP(cnt, epsilon, True)
        hull = cv2.convexHull(simplified_cnt)

        # Draw contour in red color on the blank image
        img4_with_contour = cv2.drawContours(blank_image, [cnt], -1, (0, 0, 255), 2)

        # For each point in the convex hull
        for j in range(len(hull)):
            # Calculate the distance from the center to the hull point
            dist = np.linalg.norm(center - hull[j][0])

            # Draw the fibril line in blue color
            cv2.line(img4_with_contour, (cX, cY), tuple(hull[j][0]), (255, 0, 0), 2)

            # Display the image with the contour and diameter
            #cv2.imshow('Fiber ' + str(i+1), img4_with_contour)



        # Compute the distance from the origin of the image
        distance_from_origin = np.linalg.norm(center)
        # Compute the orientation of the fibril
        if len(cnt) >= 5:  # fitEllipse requires at least 5 points
```

```
            (x, y), (MA, ma), angle = cv2.fitEllipse(cnt)
            angle = abs(angle) % 90
        # For each point in the convex hull
        for j in range(len(hull)):
            # Calculate the distance from the center to the hull point
            dist = np.linalg.norm(center - hull[j][0])

            if dist < 10:
                angle = None
            # Draw the fibril line
            cv2.line(img2, (cX, cY), tuple(hull[j][0]), (255, 0, 0), 2)  # fibril line in blue color

            # Draw non-overlapping circles along the fibril line
            p1, p2 = center, hull[j][0]  # start and end points of the fibril
            fibril_length = dist

            # Adjust the step size to separate the circles

            circle_diameter = round(5/micrometers_per_pixel) # Set your desired circle diameter in UM unit
            step_size = circle_diameter  # No more +1, so circles will be touching
            num_circles = int(fibril_length // step_size)

            # Calculate the unit vector along the fibril line
            unit_vector = (p2 - p1) / fibril_length

            # Draw circles with centers along the fibril line
            circle_color = (0, 255, 255)  # Yellow color
            circle_radius = circle_diameter // 2

            for k in range(num_circles):
                fibril_center = p1 + (step_size * k) * unit_vector
                fibril_center = tuple(map(int, fibril_center))
                cv2.circle(img2, fibril_center, circle_radius, circle_color, 2)

                # Save fibril information to the DataFrame
                fibril_data.append({
                    "Fibril": i+1,
                    "Circle": k+1,
                    "Center": (fibril_center[0]*micrometers_per_pixel, fibril_center[1]*micrometers_per_pixel),
                    "distance_from_origin(um)": distance_from_origin* micrometers_per_pixel,
                    "Orientation": angle,
                    "Diameter(um)": circle_diameter* micrometers_per_pixel,  # converting the diameter from pixels to micrometers
                    "Fibril_length(um)": fibril_length * micrometers_per_pixel  # converting the fibril_length from pixels to micrometers
                })

    fibril_data_df = pd.DataFrame(fibril_data)
    fibril_data_df = fibril_data_df.dropna()
    fibril_data_df.to_csv(CSV_Path, index=False)



    # Prepare a DataFrame to store pairwise distances
    print("The circles diameter is(um):",circle_diameter * micrometers_per_pixel)
```

```
    pairwise_distances = []

    # Calculate pairwise distances between circle centers
    for i in range(len(fibril_data)):
        for j in range(i + 1, len(fibril_data)):
            fibril1 = fibril_data[i]
            fibril2 = fibril_data[j]

            # Ignore if the circles belong to the same fiber
            if fibril1["Fibril"] == fibril2["Fibril"]:
                continue

            # Calculate the distance between circle centers
            center1 = np.array(fibril1["Center"])
            center2 = np.array(fibril2["Center"])
            distance = np.linalg.norm(center1 - center2)

            # Save the distance to the DataFrame
            pairwise_distances.append({
                "Circle1": (fibril1["Fibril"], fibril1["Circle"]),
                "Circle2": (fibril2["Fibril"], fibril2["Circle"]),
                "Distance": distance
            })


    # Save pairwise distances to a CSV file
    pairwise_distances_df = pd.DataFrame(pairwise_distances)
    pairwise_distances_df.to_csv(output_prefix + "pairwise_distances.csv", index=False)

    # Compute the histogram of distances
    lower_limit_distance = 0 #pixel
    upper_limit_distance = 480 #pixel
    distances = pairwise_distances_df["Distance"].values

    distances = distances[(distances >= lower_limit_distance) & (distances <=
    upper_limit_distance)]

    bins = np.linspace(0, np.max(distances), 50)
    hist, bin_edges = np.histogram(distances, bins=bins)

    # Calculate the bin width and the volume of the shells
    bin_width = bins[1] - bins[0]
    shell_volumes = 4 * np.pi * (bin_edges[1:]**2 - bin_edges[:-1]**2) * bin_width

    # Normalize the histogram to obtain the pair distribution function (PDF)
    pdf = hist / shell_volumes

    # Save the PDF data to a CSV file
    pdf_df = pd.DataFrame({"Distance": bins[:-1], "PDF": pdf})
    pdf_df.to_csv(output_prefix+"pdf.csv", index=False)

    # Plot the pair distribution function
    plt.plot(bins[:-1], pdf, marker="o")
    plt.xlabel("Distance (μm)", fontsize=16, fontweight="bold")
    plt.ylabel("Pair Distribution Function")
    plt.title(title)
    plt.savefig(save_PLOT)
    plt.show()
    img3 = img2.copy()
```

```
    # Draw contours in red color
    img3_with_contours = cv2.drawContours(img3.copy(), filtered_contours, -1, (0, 0, 255),
    2)

    # Display the original image with contours
    cv2.imshow('contoures on the main image', img3_with_contours)

    img4 = img3.copy()
    #*********************************************************************
    # Create a blank image
    blank_img4 = np.zeros(img4.shape, dtype=np.uint8)

    # Draw contours in red color on the blank image
    img4_with_contours = cv2.drawContours(blank_img4, filtered_contours, -1, (0, 0, 255),
    2)

    # For each filtered contour
    for i, cnt in enumerate(filtered_contours):
        # Compute the center of the contour
        M = cv2.moments(cnt)
        cX = int(M["m10"] / M["m00"])
        cY = int(M["m01"] / M["m00"])
        center = np.array([cX, cY])

        # Compute the convex hull of the contour
        epsilon = 0.1*cv2.arcLength(cnt,True)
        simplified_cnt = cv2.approxPolyDP(cnt, epsilon, True)
        hull = cv2.convexHull(simplified_cnt)

        # For each point in the convex hull
        for j in range(len(hull)):
            # Calculate the distance from the center to the hull point
            dist = np.linalg.norm(center - hull[j][0])

            # Draw the fibril line in blue color
            cv2.line(img4_with_contours, (cX, cY), tuple(hull[j][0]), (255, 0, 0), 2)

    # Display the image with contours and diameters
    cv2.imshow('contoures', img4_with_contours)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
    fig = plt.figure(figsize=(10,10))  # Adjust as necessary

    ax1 = fig.add_subplot(2,2,1)  # Here 2,2 is the number of rows and columns of images
    ax1.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))  # Convert BGR to RGB for correct
    display
    ax1.title.set_text('Original')

    ax2 = fig.add_subplot(2,2,2)
    ax2.imshow(closing, cmap='gray')
    ax2.title.set_text('Threshold')

    ax3 = fig.add_subplot(2,2,3)
    ax3.imshow(cv2.cvtColor(img4_with_contours, cv2.COLOR_BGR2RGB))  # Convert BGR to RGB
    for correct display
    ax3.title.set_text('Contours on Main Image')

    ax4 = fig.add_subplot(2,2,4)
    ax4.imshow(cv2.cvtColor(img3_with_contours, cv2.COLOR_BGR2RGB))  # Convert BGR to RGB
```

```
    for correct display
    ax4.title.set_text('Contours/lines/circles')

    plt.savefig(output_prefix + "combined_image.jpg")
    plt.show()
    return pdf_df




pdf_df_A=process_image(Image_Path_A,
    output_prefix_A,
    manual_thresh_A,
    "Col",
    Loc_A,
    CSV_Path_A

)

pdf_df_E=process_image(Image_Path_E,
    output_prefix_E,
    manual_thresh_E,
    "colfncell",
    Loc_E,
    CSV_Path_E

)



Npdf_A=pdf_df_A["PDF"]/pdf_df_A["PDF"].max()
Npdf_E=pdf_df_E["PDF"]/pdf_df_E["PDF"].max()



plt.plot(pdf_df_A["Distance"], Npdf_A, marker="o",color='purple' ,label="Col-Fn")
plt.plot(pdf_df_E["Distance"], Npdf_E, marker="o",color='blue' ,label="Col-Fn-Cell")
plt.xlabel("Distance (μm)", fontsize=10, fontweight="bold")
plt.ylabel("Pair Distribution Function",fontsize=10, fontweight="bold")
#plt.xlim([-300,400])
plt.legend()
plt.title('Cold cast')
plt.savefig(r"output/N-pdf.jpeg", dpi=600, bbox_inches="tight")
plt.show()
```

Bibliography

[1] S. Misra, H. Li, J. He, Machine learning for subsurface characterization, Gulf Professional Publishing, 2019.

[2] R. C. Gonzalez, Digital image processing, Pearson education india, 2009.

[3] C. Sha, J. Hou, H. Cui, A robust 2d otsu's thresholding method in image segmentation, Journal of Visual Communication and Image Representation 41 (2016) 339–351.

[4] N. Otsu, A threshold selection method from gray-level histograms, IEEE Transactions on Systems, Man, and Cybernetics 9 (1) (1979) 62–66. doi:10.1109/TSMC.1979.4310076.

[5] N. Singh, V. Tewari, P. Biswas, C. Pareek, L. Dhruw, Image processing algorithms for in-field cotton boll detection in natural lighting conditions, Artificial Intelligence in Agriculture 5 (2021) 142–156.

[6] P. Soille, et al., Morphological image analysis: principles and applications, Vol. 2, Springer, 1999.

[7] S. Suzuki, et al., Topological structural analysis of digitized binary images by border following, Computer vision, graphics, and image processing 30 (1) (1985) 32–46.

[8] Z. Zhang, Q. Li, W. Song, P. Wei, J. Guo, A novel concavity based method for automatic segmentation of touching cells in microfluidic chips, Expert Systems with Applications 202 (2022) 117432.

[9] L. A. Yates, B. W. Brook, J. C. Buettel, Spatial pattern analysis of line-segment data in ecology, Ecology 103 (2) (2022) e03597.

[10] A. Baddeley, E. Rubak, R. Turner, Spatial point patterns: methodology and applications with R, Vol. 1, CRC press Boca Raton, 2016.

[11] G. Vasiukov, T. Novitskaya, M.-F. Senosain, A. Camai, A. Menshikh, P. Massion, A. Zijlstra, S. Novitskiy, Integrated cells and collagen fibers spatial image analysis, Frontiers in bioinformatics 1 (2021) 758775.

[12] J. Illian, A. Penttinen, H. Stoyan, D. Stoyan, Statistical analysis and modelling of spatial point patterns, John Wiley & Sons, 2008.

[13] S. Torquato, et al., Random heterogeneous materials: microstructure and macroscopic properties, Vol. 16, Springer, 2002.

[14] J. Wilhelm, E. Frey, Radial distribution function of semiflexible polymers, Physical review letters 77 (12) (1996) 2581.

[15] V. Howard, M. Reed, Unbiased stereology: three-dimensional measurement in microscopy, Garland Science, 2004.

[16] Y.-l. Yang, L. M. Leone, L. J. Kaufman, Elastic moduli of collagen gels can be predicted from two-dimensional confocal microscopy, Biophysical journal 97 (7) (2009) 2051–2060.