# MINFLUX Fiber Segmentation Pipeline: Algorithmic and Mathematical Description

## Contents

## 1 Overview and Notation

The goal of the MINFLUX fiber segmentation pipeline is to assign individual MINFLUX localizations to distinct filamentous fibers and to estimate per–fiber geometric properties such as length and effective width. The pipeline is segmentation–oriented: it partitions localizations into fiber labels rather than reconstructing detailed physical geometry.

At a high level, the pipeline proceeds as follows:

1. Project 3D localizations into 2D (XY) and construct a kernel density estimate (KDE) image on a pixel grid.
2. Enhance filamentous structures via a tubeness filter and segment a binary fiber mask in 2D.
3. Skeletonize the mask and convert the skeleton into a graph of nodes (endpoints and junctions) and segments.
4. Merge segments into longer fibers based on geometric adjacency and orientation criteria.
5. Map the 2D fiber centerlines into 3D by assigning averaged MINFLUX points (grouped by TraceID) to fibers, forming 3D tubes.
6. Optionally grow the 3D fibers using a cylinder–based extrapolation model.
7. Compute per–fiber length and width statistics.

Implementation overview:

- Global configuration (pixel size, thresholds, radii, etc.) is defined in `fiber_pipeline.config`.
- The top–level batch driver is `fiber_pipeline.run_folder`.

### Notation

- Raw MINFLUX localizations (per sample) are

$$\mathbf{p}_i = (x_i, y_i, z_i)^\top \in \mathbb{R}^3, \quad i = 1, \ldots, N,$$

with coordinates in micrometers (µm).

- When Trace IDs are available, each localization also carries an integer label

$$\text{TraceID}_i \in \{1, \dots, T\},$$

grouping points that belong to the same temporal trace or localization cluster.

- For a given TraceID $t$, the set of indices is

$$I_t = \{i \in \{1, \dots, N\} \mid \text{TraceID}_i = t\}.$$

- The averaged location for trace $t$ is

$$\bar{\mathbf{p}}_t = (\bar{x}_t, \bar{y}_t, \bar{z}_t)^\top = \frac{1}{|I_t|} \sum_{i \in I_t} \mathbf{p}_i,$$

with trace multiplicity $N_t = |I_t|$.

- All XY images are defined on a discrete pixel grid of size $H \times W$.

### Module reference for this section

- Global constants and configuration: `fiber_pipeline.config`.
- Top–level orchestration and folder traversal: `fiber_pipeline.run_folder`.

## 2 2D KDE Construction and Tubeness Enhancement

This section describes the construction of a 2D KDE image from XY localizations and the application of a tubeness filter to highlight filamentous structures.

Implementation:

- Module: `fiber_pipeline.kde_tubeness`
- Key functions:
  - `compute_kde_counts`
  - `normalize_to_8bit`
  - `tubeness_from_kde_gray8`

### 2.1 Bounding box and pixel mapping

Let

$$x_{\min} = \min_i x_i, \quad x_{\max} = \max_i x_i, \quad y_{\min} = \min_i y_i, \quad y_{\max} = \max_i y_i.$$

We choose a padding parameter PAD_UM $> 0$ (in µm) and define the continuous XY range

$$x_0 = x_{\min} - \text{PAD\_UM}, \quad x_1 = x_{\max} + \text{PAD\_UM},$$

$$y_0 = y_{\min} - \text{PAD\_UM}, \quad y_1 = y_{\max} + \text{PAD\_UM}.$$

The resolution is specified by PX_PER_UM $= P$ pixels per micrometer. The approximate width and height in pixels of the base grid are

$$W_0 = \lfloor (x_1 - x_0)P + 1.5 \rfloor, \quad H_0 = \lfloor (y_1 - y_0)P + 1.5 \rfloor.$$

Each localization $(x_i, y_i)$ is mapped to a discrete pixel coordinate $(u_i, v_i)$:

$$u_i = \mathrm{round}\big((x_i - x_0)P\big), \tag{1}$$

$$v_i = H_0 - 1 - \mathrm{round}\big((y_i - y_0)P\big). \tag{2}$$

The vertical coordinate $v$ is flipped so that increased $y$ corresponds to decreasing row index, consistent with standard image conventions.

The discrete count image $C \in \mathbb{R}^{H_0 \times W_0}$ is defined by

$$C[v, u] = \sum_{i=1}^{N} \mathbf{1}\big((u_i, v_i) = (u, v)\big), \tag{3}$$

where $\mathbf{1}(\cdot)$ is the indicator function.

## 2.2 Gaussian smoothing: approximate KDE

We interpret $C$ as a discretized approximation of an underlying spatial intensity field and further smooth it with a Gaussian kernel to obtain a KDE–like image.

Given a Gaussian standard deviation in micrometers $\sigma_{\mathrm{KDE}}$ (parameter `GAUSS_SIGMA_UM`), the corresponding pixel sigma is

$$\sigma_{\mathrm{px}} = \sigma_{\mathrm{KDE}} \cdot P.$$

We define a separable Gaussian kernel $G$ over the pixel coordinates:

$$G(\Delta u, \Delta v) = \frac{1}{2\pi\sigma_{\mathrm{px}}^2} \exp\left(-\frac{\Delta u^2 + \Delta v^2}{2\sigma_{\mathrm{px}}^2}\right),$$

normalized so that $\sum_{\Delta u, \Delta v} G(\Delta u, \Delta v) = 1$.

The blurred density (KDE image) $K$ is the convolution

$$K = C * G. \tag{4}$$

In the code, this is implemented numerically via `scipy.ndimage.gaussian_filter`.

## 2.3 Normalization and tubeness measure

We normalize $K$ to the interval $[0, 1]$ by

$$K_{\mathrm{norm}} = \frac{K - \min K}{\max K - \min K + \varepsilon}, \tag{5}$$

$$I_{\mathrm{8bit}} = \mathrm{round}(255\, K_{\mathrm{norm}}), \tag{6}$$

with a small $\varepsilon > 0$ to avoid division by zero.

The tubeness filter (Sato vesselness) operates on the normalized float image $I_{\mathrm{norm}} = K_{\mathrm{norm}}$, interpreted as intensities on $\mathbb{Z}^2$.

For each pixel and scale $\sigma$, the Sato filter conceptually:

1. Smooths the image with a Gaussian of standard deviation $\sigma$.
2. Computes the Hessian matrix of second derivatives

$$H_\sigma(x,y) = \begin{bmatrix} \frac{\partial^2 I_\sigma}{\partial x^2} & \frac{\partial^2 I_\sigma}{\partial x \partial y} \\ \frac{\partial^2 I_\sigma}{\partial y \partial x} & \frac{\partial^2 I_\sigma}{\partial y^2} \end{bmatrix}.$$

3. Obtains the eigenvalues $\lambda_1, \lambda_2$ of $H_\sigma$ (ordered by $|\lambda_1| \geq |\lambda_2|$).
4. For bright tubular structures on a dark background, a vesselness response is constructed as a function of the eigenvalues (e.g. strong negative curvature in one direction and small curvature in the orthogonal one).

The resulting multi–scale vesselness $V(x,y)$ is typically a maximum or combination over scales:

$$V(x,y) = \max_{\sigma \in \Sigma} \text{Vesselness}_\sigma(x,y), \tag{7}$$

where $\Sigma$ is a finite set of scales.

In this pipeline, a simplified Sato filter is applied at one or several scales, implemented via `skimage.filters.sato`. Negative responses are clipped to zero:

$$V(x,y) = \max\big(\text{sato}(I_{\text{norm}})(x,y),\, 0\big).$$

## 3 2D Fiber Segmentation

This stage converts the tubeness image into a clean binary mask of fibers and then skeletonizes it.

Implementation:

- Module: `fiber_pipeline.segment_2d`
- Key functions:
  - `to_gray01`
  - `segment_fibers`
  - `find_endpoints`
  - `bridge_small_gaps`

### 3.1 Preprocessing: denoising and contrast enhancement

Let $I_{\text{tube}}(x,y)$ denote the tubeness image (rescaled to $[0,1]$). A denoised version $\tilde{I}$ is obtained by total variation (TV) regularization, e.g. via Chambolle's algorithm:

$$\tilde{I} = \arg\min_J \frac{1}{2}\|J - I_{\text{tube}}\|_2^2 + \lambda \text{TV}(J), \tag{8}$$

where $\text{TV}(\cdot)$ is the total variation seminorm; $\lambda > 0$ is a regularization parameter (controlled via `weight` in `denoise_tv_chambolle`).

Local contrast is further enhanced via CLAHE (Contrast Limited Adaptive Histogram Equalization), yielding

$$I_{\text{clahe}} = \text{CLAHE}(\tilde{I}).$$

## 3.2 Multi–scale ridge enhancement

We apply the Sato filter to $I_{\text{clahe}}$ at multiple scales $\sigma \in [\sigma_{\min}, \sigma_{\max}]$, discretized into $n_\sigma$ scales:

$$\Sigma = \{\sigma_k\}_{k=1}^{n_\sigma}, \quad \sigma_k = \sigma_{\min} + (k-1)\,\frac{\sigma_{\max} - \sigma_{\min}}{n_\sigma - 1}.$$

The vesselness image is

$$V(x,y) = \max_{\sigma \in \Sigma} \text{Vesselness}_\sigma\big(I_{\text{clahe}}(x,y)\big),$$

rescaled to $[0,1]$ via linear intensity scaling.

## 3.3 Hysteresis thresholding

We choose thresholds based on the empirical distribution of $V$:

$$T_H = \text{percentile}\big(V,\, p_{\text{high}}\big), \tag{9}$$
$$T_L = \alpha \cdot T_H, \tag{10}$$

where $p_{\text{high}}$ is parameter `hyst_high_pct`, and $\alpha$ is `hyst_low_ratio`.

Hysteresis thresholding defines a binary mask $M$ by:

1. Identify all pixels with $V(x,y) \geq T_H$ as strong ridges.
2. Include all pixels with $T_L \leq V(x,y) < T_H$ that are connected to a strong ridge by a path of pixels also above $T_L$.

Formally, $M$ is the union of all connected components of the set $\{V \geq T_L\}$ that intersect the set $\{V \geq T_H\}$.

## 3.4 Morphological cleanup

Let $M$ be the raw binary mask. We perform:

- removal of connected components smaller than a minimum size $s_{\min}$,

  $$M \leftarrow \text{RemoveSmallObjects}(M, s_{\min}),$$

- binary closing with a disk of radius $r_{\text{close}}$ to connect small gaps,

  $$M \leftarrow M \bullet B_{r_{\text{close}}},$$

  where $\bullet$ denotes morphological closing and $B_{r_{\text{close}}}$ is a disk structuring element,

- filling of holes up to area $A_{\text{hole}}$,

  $$M \leftarrow \text{RemoveSmallHoles}(M, A_{\text{hole}}).$$

### 3.5 Gap bridging

Optionally, we attempt to bridge small gaps between skeleton endpoints using geodesic shortest paths through high–vesselness regions.

Let $M$ be the binary mask. We first skeletonize:

$$S = \text{Skeletonize}(M),$$

and then compute its endpoints as pixels with exactly one 8–connected neighbor.

We define a cost image $C_{\text{cost}}$ based on vesselness:

$$C_{\text{cost}}(x, y) = 1 - \tilde{V}(x, y), \quad \tilde{V} = \text{Rescale}(V, [0, 1]).$$

Lower cost corresponds to higher vesselness (more likely fiber center).

For each endpoint pair $(p_i, p_j)$ within some Euclidean distance $d_{\max}$, we compute a shortest path

$$\gamma = \arg\min_{\gamma: p_i \to p_j} \sum_{(x,y)\in\gamma} C_{\text{cost}}(x, y).$$

If the path length $|\gamma|$ is bounded by some factor times the straight–line distance $\|p_i - p_j\|_2$ (e.g. $|\gamma| \leq \alpha \|p_i - p_j\|_2 + \beta$), we consider the path plausible and set $M(x, y) = 1$ for all $(x, y) \in \gamma$.

### 3.6 Skeletonization

Finally, we skeletonize the mask $M$ to obtain a one–pixel–wide centerline:

$$S = \text{Skeletonize}(M).$$

The binary skeleton $S$ is a subset of the pixel grid $\{0, \ldots, H - 1\} \times \{0, \ldots, W - 1\}$.

## 4 Skeleton Graph: Nodes and Segments

The skeleton $S$ is converted into a graph whose vertices are endpoint and junction nodes and whose edges are segments of centerline pixels between nodes.

Implementation:

- Module: `fiber_pipeline.skeleton_graph`
- Representative functions:
  - degree and node labeling logic (endpoints vs. junctions),
  - `junction_exits`, `endpoint_exits`,
  - segment extraction loop.

### 4.1 Node classification

Let $S(x, y) \in \{0, 1\}$ be the skeleton image. For each skeleton pixel $(r, c)$, we compute the number of 8–connected neighbors:

$$n(r, c) = \sum_{(\Delta r, \Delta c)\in\mathcal{N}_8} S(r + \Delta r, c + \Delta c),$$

where
$$\mathcal{N}_8 = \{(-1,-1),(-1,0),(-1,1),(0,-1),(0,1),(1,-1),(1,0),(1,1)\}.$$

Define its degree as
$$d(r,c) = n(r,c)\cdot S(r,c).$$

We classify skeleton pixels as:

- *endpoints*: $S(r,c)=1$ and $d(r,c)=1$,
- *junction pixels*: $S(r,c)=1$ and $d(r,c)\geq 3$,
- *simple chain pixels*: $S(r,c)=1$ and $d(r,c)=2$.

All connected components of junction pixels are merged into single *junction nodes*.

Let $J_k$ be the pixels in the $k$–th connected component of junction pixels. The node $v_k$ is defined by:
$$v_k = \left(\frac{1}{|J_k|}\sum_{(r,c)\in J_k} r,\ \frac{1}{|J_k|}\sum_{(r,c)\in J_k} c\right),$$
together with its set of pixel indices $J_k$.

Each endpoint pixel forms its own endpoint node.

We assign each skeleton pixel $(r,c)$ a node ID
$$\text{node\_id}(r,c)\in\{-1,0,1,\ldots,N_{\text{nodes}}-1\},$$
with $-1$ indicating non–node skeleton pixels.

## 4.2 Neighbor function

For each skeleton pixel $(r,c)$, define the neighbor set
$$\text{Nbrs}(r,c) = \{(r',c')\mid (r',c')\in\mathcal{N}_8(r,c),\ S(r',c')=1\},$$
where $\mathcal{N}_8(r,c)$ is the 8–connected neighborhood.

## 4.3 Segment extraction

A segment is defined as a maximal chain of skeleton pixels connecting two distinct nodes without passing through any other node.

For each node $n$ (junction or endpoint), we define *exits*: pixels adjacent to the node that lie on the skeleton but are outside the node's pixel set. For a junction node $n$ with pixel set $J_n$:
$$\text{Exits}(n) = \{(p_{\text{out}},p_{\text{anchor}})\},$$
where $p_{\text{anchor}}\in J_n$ is a node pixel and $p_{\text{out}}\in S\setminus J_n$ is a neighbor of $p_{\text{anchor}}$.

For an endpoint node $n$ with single pixel $p_{\text{E}}$, exits are simply the skeleton neighbors of $p_{\text{E}}$ that are not part of any node.

Starting from each exit $(p_{\text{out}},p_{\text{anchor}})$ of node $n_0$, we follow the chain via `Nbrs`:

1. Initialize the path as $[p_{\text{anchor}},p_{\text{out}}]$.

2. At each step, let $p_{\text{prev}}, p_{\text{cur}}$ be the last two pixels in the path. Consider neighbors $q \in \text{Nbrs}(p_{\text{cur}}) \setminus \{p_{\text{prev}}\}$:

   - If any $q$ is a node pixel (belongs to some node $n_1$), and $n_1 \neq n_0$, then we terminate the segment at node $n_1$.
   - If the only neighbor is a chain pixel not belonging to any node, we append it and continue.
   - If there are zero or multiple such chain neighbors, we stop (dead end or ambiguity).

Each such traversal yields a segment characterized by:

- node IDs $(u, v)$ at its ends,
- a sequence of interior skeleton pixels $\{p_k\}$ between anchors,
- anchor pixels $(\text{anchor}_u, \text{anchor}_v)$ at each node boundary.

Segments between the same pair of nodes $(u, v)$ are deduplicated, keeping the longest chain.

## 5 Orientation and Segment Merging in 2D

Segments are grouped into fibers using geometric constraints: adjacency in the skeleton graph, near–neighbor tests, and angular similarity.

Implementation:

- Module: `fiber_pipeline.merge_segments`
- Representative logic:
  - per–segment orientation computation,
  - adjacency (*share–a–junction* and near–neighbor),
  - mutual best–neighbor merging loop,
  - post–processing branch splitting.

### 5.1 Segment orientation (axial)

Let $c_u$ and $c_v$ be the centroids of nodes $u$ and $v$ for a segment $s$. The segment orientation $\theta_s$ (in degrees) is defined by

$$\begin{aligned} \theta_s &= \text{orientation\_0\_180}(c_u, c_v) && (11) \\ &= \left( \text{atan2}(c_{v,y} - c_{u,y},\ c_{v,x} - c_{u,x}) \cdot \frac{180}{\pi} + 180 \right) \bmod 180. && (12) \end{aligned}$$

This is an *axial* orientation: the direction is considered equivalent with its opposite, so angles lie in $[0, 180)$.

### 5.2 Acute angle difference

Given two axial angles $\theta_a, \theta_b \in [0, 180)$, the acute difference is

$$\Delta(\theta_a, \theta_b) = \min\left(|\theta_a - \theta_b|,\ 180^\circ - |\theta_a - \theta_b|\right), \tag{13}$$

which lies in $[0, 90^\circ]$. This is implemented by the function often named `acute_difference` or similar.

### 5.3 Base adjacency (share–a–junction)

Two segments $s_i$ and $s_j$ are *adjacent at a junction* if they share a junction node, i.e. if there exists a junction $n$ such that both segments are incident on $n$. Formally,

$$s_i \sim_{\mathrm{J}} s_j \quad \Longleftrightarrow \quad \exists n \text{ (junction) } : \; n \in \{u_i, v_i\} \cap \{u_j, v_j\}.$$

This relation is symmetric and is used to initialize an adjacency graph between segments.

### 5.4 Near–neighbor adjacency

Beyond strict junction adjacency, segments can be considered adjacent based on proximity of their endpoints and nearby junctions.

**Endpoint–endpoint proximity.** Let $E_i$ be the set of endpoints for segment $s_i$, i.e. anchors that belong to endpoint nodes. For each endpoint $p \in E_i$, we consider other endpoints $q \in E_j$ from different segments. If

$$\|p - q\|_2 \le R_{\text{near}},$$

with $R_{\text{near}}$ in pixels (`NEAR_RADIUS_PX`), we declare $s_i$ and $s_j$ adjacent.

**Endpoint–junction proximity.** For an endpoint $p \in E_i$ and a junction $n$ with centroid $c_n$, if

$$\|p - c_n\|_2 \le R_{\text{near}},$$

we connect $s_i$ to all segments incident on junction $n$.

These checks expand the adjacency from purely graph–based to geometric vicinity.

### 5.5 Group orientation via double–angle trick

During merging, segments are aggregated into groups (candidate fibers). Let $G$ be a group and $\mathcal{S}_G$ the set of its segments. Their individual orientations are $\{\theta_s\}_{s \in \mathcal{S}_G}$.

To average orientations on an axial domain, we use the *double–angle* representation:

$$z_G = \frac{1}{|\mathcal{S}_G|} \sum_{s \in \mathcal{S}_G} \exp\left(i\, 2\theta_s \frac{\pi}{180}\right). \tag{14}$$

The group orientation is then

$$\theta_G = \frac{1}{2} \arg(z_G) \cdot \frac{180}{\pi}. \tag{15}$$

If $z_G \approx 0$ (e.g. orientations cancel), a fallback such as the average of $\theta_s$ can be used.

### 5.6 Mutual best–neighbor merging

We maintain an array of group IDs (initially one segment per group). At each iteration:

1. Compute group orientation $\theta_G$ for each group.
2. For each group $G$, assemble the set of adjacent groups $\mathcal{N}(G)$ by looking at adjacency between their segments.
3. For each neighboring pair $(G, H)$, compute:

- angular difference $\Delta(\theta_G, \theta_H)$,
- minimal anchor distance between any segment in $G$ and any segment in $H$.

4. Define *candidate neighbors* for $G$ as those groups $H \in \mathcal{N}(G)$ satisfying

$$\Delta(\theta_G, \theta_H) < \Theta_{\max},$$

where $\Theta_{\max}$ is `ANGLE_THRESH_DEG`.

5. Among candidates, choose the best neighbor best$(G)$ as the one with minimal angle difference and, in case of ties, minimal anchor distance.

6. Merge only mutual best neighbors: if best$(G) = H$ and best$(H) = G$, then merge $G$ and $H$ into a single group (via union–find).

7. Update group IDs, compress labels, and repeat until no merges occur or a maximum iteration count (`MAX_ITERS`) is reached.

This strategy prevents chains of merges where merging would propagate arbitrarily through the adjacency graph. Groups only merge if both consider each other their most compatible neighbor.

## 5.7 Branch splitting at junctions

After merging, some fibers may still form branches at junctions (more than two arms with the same label meeting at a junction). A post–processing step enforces at most two arms per fiber at each junction.

For each junction node $n$:

1. Let $\mathcal{S}_n$ be the set of segments incident on $n$.

2. Partition $\mathcal{S}_n$ by current group label: for each label $\ell$, collect

$$\mathcal{S}_{n,\ell} = \{s \in \mathcal{S}_n \mid \text{group\_id}(s) = \ell\}.$$

3. For each label $\ell$ with $|\mathcal{S}_{n,\ell}| \geq 3$:

   (a) Compute *local* orientation $\theta_{s,n}$ of each segment $s \in \mathcal{S}_{n,\ell}$ at node $n$. This orientation is defined using the node anchor and the first pixel *away* from the node along the segment path.

   (b) Among all pairs $(s_a, s_b) \in \mathcal{S}_{n,\ell} \times \mathcal{S}_{n,\ell}$, select the pair minimizing $\Delta(\theta_{s_a,n}, \theta_{s_b,n})$ (smoothest continuation).

   (c) Keep this pair assigned to label $\ell$; reassign all remaining segments in $\mathcal{S}_{n,\ell}$ to new, distinct labels.

This ensures that each fiber label passes through a junction in at most two directions (closely aligned arms), with other arms forming separate fibers.

## 6 Mapping 2D Skeleton to Physical XY Units

To relate the 2D segmentation back to physical coordinates, we must invert the pixel mapping used in the KDE.

Implementation:

- Module: `fiber_pipeline.tubes_3d`
- Key helper: `pix_to_um`

### 6.1 Inverse mapping

Recall that in the base grid $(H_0, W_0)$, pixel coordinates $(u, v)$ are related to physical coordinates $(x, y)$ by

$$u \approx (x - x_0)P, \quad v \approx H_0 - 1 - (y - y_0)P.$$

The upsampled tubeness and segmentation steps operate on a grid scaled by `UPSCALE_FACTOR` $= U$, so that upsampled pixel coordinates are

$$u' = Uu, \quad v' = Uv.$$

Conversely,

$$u = \frac{u'}{U}, \quad v = \frac{v'}{U}.$$

Substituting into the original mapping, we obtain the inverse transform from upsampled pixel $(v', u')$ to physical XY:

$$x = x_0 + \frac{u'}{UP}, \tag{16}$$

$$y = y_0 + \frac{H_0 - 1 - \frac{v'}{U}}{P}. \tag{17}$$

Thus the function `pix_to_um` takes a skeleton pixel in upsampled coordinates and returns the corresponding $(X_\mu, Y_\mu)$ in micrometers.

## 7 3D Tube Centerlines from Averaged MINFLUX Points

This stage uses the 2D segmentation and skeleton to assign averaged MINFLUX points to fibers in 3D and to construct 3D tube centerlines.

Implementation:

- Module: `fiber_pipeline.tubes_3d`
- Key logic:
  - reading and averaging `RawLocs_IDs.csv` by `TraceID`,
  - `build_segment_polyline_um`,
  - `resample_polyline`,
  - assignment of averaged points to fibers via KD–tree,
  - construction of 3D centerline samples.

### 7.1 TraceID averaging

Let $\{\mathbf{p}_i\}_{i=1}^N$ with coordinates $(x_i, y_i, z_i)$ and Trace ID $t_i$ be loaded from the `RawLocs_IDs.csv` file.

For each Trace ID $t$, define:

$$I_t = \{i \mid t_i = t\}, \quad \bar{\mathbf{p}}_t = \frac{1}{|I_t|} \sum_{i \in I_t} \mathbf{p}_i.$$

We gather

$$\bar{\mathbf{p}}_t = (\bar{x}_t, \bar{y}_t, \bar{z}_t)^\top,$$

and store count $N_t = |I_t|$.

### 7.2 2D fiber polylines in µm

For each segment $s$, we construct a polyline in physical XY units:

1. Start with the pixel path

$$[\text{anchor}_u,\, p_1,\, p_2,\, \ldots,\, p_k,\, \text{anchor}_v],$$

where the $p_i$ are interior skeleton pixels.

2. Remove consecutive duplicate pixels if any.

3. Map each pixel $(v', u')$ to $(x, y)$ via the inverse mapping in Section 6.

Thus each segment $s$ has a discrete polyline

$$\mathbf{c}_s(j) = (x_{s,j}, y_{s,j})^\top, \quad j = 1, \ldots, M_s,$$

in micrometers.

### 7.3 Arc–length resampling

To regularize sampling along the fibers, each polyline is resampled at approximately uniform arc length step $\Delta s > 0$ (`CENTER_STEP`).

Let $\mathbf{c}_s(1), \ldots, \mathbf{c}_s(M_s)$ be the original points. Define segment lengths

$$\ell_j = \|\mathbf{c}_s(j+1) - \mathbf{c}_s(j)\|_2, \quad j = 1, \ldots, M_s - 1,$$

and cumulative lengths

$$L_0 = 0, \quad L_j = \sum_{k=1}^{j} \ell_k, \quad j = 1, \ldots, M_s - 1.$$

We construct a set of target arc lengths

$$T = \{t_m = m\Delta s \mid m = 0, 1, \ldots, \lfloor L_{M_s-1}/\Delta s \rfloor\}.$$

Each target $t_m$ falls within some segment $[L_j, L_{j+1}]$; we interpolate linearly between $\mathbf{c}_s(j)$ and $\mathbf{c}_s(j+1)$:

$$\mathbf{c}_s^*(t_m) = \mathbf{c}_s(j) + \frac{t_m - L_j}{L_{j+1} - L_j}\big(\mathbf{c}_s(j+1) - \mathbf{c}_s(j)\big).$$

The resulting resampled polyline is

$$\{\mathbf{c}_s^*(t_m)\}_m,$$

with approximately uniform spacing along arc length.

### 7.4 Assignment of averaged points to fibers in 2D

Let $\mathcal{C} = \{\mathbf{c}_k\}_{k=1}^{M}$ denote all resampled centerline samples across all segments/fibers, and let $g(k)$ be the fiber label (group ID) for sample $k$. We store their XY coordinates as:

$$\mathbf{c}_k = (X_k, Y_k)^\top \in \mathbb{R}^2.$$

For each averaged TraceID point

$$\bar{\mathbf{q}}_t = (\bar{x}_t, \bar{y}_t)^\top,$$

we assign a fiber label using nearest–neighbor search in XY:

$$k^* = \arg\min_k \|\bar{\mathbf{q}}_t - \mathbf{c}_k\|_2.$$

If the distance satisfies

$$d_t = \|\bar{\mathbf{q}}_t - \mathbf{c}_{k^*}\|_2 \le r_{\text{tube}},$$

where $r_{\text{tube}}$ is `TUBE_RADIUS`, then we set

$$\text{assigned\_gid}(t) = g(k^*).$$

Otherwise, we declare the point unassigned:

$$\text{assigned\_gid}(t) = -1.$$

### 7.5 3D centerline construction

For each fiber label $f$ (segment group), consider:

- its resampled XY centerline $\{\mathbf{c}_{f,j}\}$,
- the set of averaged points assigned to this fiber,

$$\mathcal{T}_f = \{t \mid \text{assigned\_gid}(t) = f\}.$$

For each centerline sample $\mathbf{c}_{f,j}$:

1. We collect indices of nearby averaged points in XY within radius $r_{\text{search}}$,

$$\mathcal{N}_{f,j} = \{t \in \mathcal{T}_f \mid \|\bar{\mathbf{q}}_t - \mathbf{c}_{f,j}\|_2 \le r_{\text{search}}\},$$

   where $r_{\text{search}}$ is `SEARCH_RADIUS`.

2. We further restrict to those within $r_{\text{tube}}$:

$$\mathcal{N}'_{f,j} = \{t \in \mathcal{N}_{f,j} \mid \|\bar{\mathbf{q}}_t - \mathbf{c}_{f,j}\|_2 \le r_{\text{tube}}\}.$$

3. If $|\mathcal{N}'_{f,j}|$ is less than some minimum (`MIN_PTS_PER_SAMPLE`), we drop this centerline sample for 3D.

4. Otherwise, we estimate a robust Z coordinate as a median:

$$z_{f,j} = \text{median}\{\bar{z}_t \mid t \in \mathcal{N}'_{f,j}\}.$$

5. We compute the XY centroid

$$\bar{\mathbf{q}}_{f,j} = \frac{1}{|\mathcal{N}'_{f,j}|} \sum_{t \in \mathcal{N}'_{f,j}} (\bar{x}_t, \bar{y}_t)^\top$$

   and a shift vector

$$\mathbf{s}_{f,j} = \bar{\mathbf{q}}_{f,j} - \mathbf{c}_{f,j}.$$

   If $\|\mathbf{s}_{f,j}\|_2 \le s_{\max}$ (`MAX_XY_SHIFT`), we move the centerline sample fully to $\bar{\mathbf{q}}_{f,j}$; otherwise, we limit the shift magnitude to $s_{\max}$:

$$\tilde{\mathbf{c}}_{f,j} = \mathbf{c}_{f,j} + \min\left(1, \frac{s_{\max}}{\|\mathbf{s}_{f,j}\|_2 + \varepsilon}\right) \mathbf{s}_{f,j}.$$

The resulting 3D centerline sample is

$$\tilde{\mathbf{C}}_{f,j} = \left(\tilde{\mathbf{c}}_{f,j},\ z_{f,j}\right) \in \mathbb{R}^3.$$

## 8 Cylinder Growth in 3D

The cylinder growth stage refines fiber labels by growing cylinders along local principal directions at fiber ends and reassigning nearby points.

Implementation:

- Module: `fiber_pipeline.grow_cylinders`
- Key pieces:
  - `first_pc` for principal component computation,
  - `grow_fiber_once` for a single fiber,
  - global loop over fibers.

### 8.1 Initial fiber length and ranking

We work on the averaged points $\bar{\mathbf{p}}_t$ (TraceID level) and initial labels assigned_gid($t$), keeping only points with non–negative labels:

$$\mathcal{T}_{\text{valid}} = \{t \mid \text{assigned\_gid}(t) \ge 0\}.$$

For each fiber label $f$, let

$$\mathcal{I}_f = \{t \in \mathcal{T}_{\text{valid}} \mid \text{assigned\_gid}(t) = f\}.$$

If $|\mathcal{I}_f| < N_{\min}$ (`MIN_PTS_FIBER`), the fiber is not grown.

We collect the 3D points for fiber $f$:

$$P_f = \{\bar{\mathbf{p}}_t \mid t \in \mathcal{I}_f\}.$$

The global principal component axis $\mathbf{v}_f \in \mathbb{R}^3$ and mean $\boldsymbol{\mu}_f$ are obtained via PCA or SVD:

$$\boldsymbol{\mu}_f = \frac{1}{|\mathcal{I}_f|} \sum_{t \in \mathcal{I}_f} \bar{\mathbf{p}}_t,$$

$$P_f^{\text{centered}} = \{\bar{\mathbf{p}}_t - \boldsymbol{\mu}_f \mid t \in \mathcal{I}_f\},$$

$$\mathbf{v}_f = \arg\max_{\|\mathbf{v}\|=1} \sum_{t\in\mathcal{I}_f} \big((\bar{\mathbf{p}}_t - \boldsymbol{\mu}_f)\cdot\mathbf{v}\big)^2.$$

In practice, SVD is used to compute $\mathbf{v}_f$ as the first right singular vector.

We define scalar coordinates along $\mathbf{v}_f$:

$$t_t = (\bar{\mathbf{p}}_t - \boldsymbol{\mu}_f)\cdot\mathbf{v}_f.$$

The initial fiber length estimate is

$$L_f^{\text{init}} = \max_{t\in\mathcal{I}_f} t_t - \min_{t\in\mathcal{I}_f} t_t.$$

Fibers are ranked by decreasing $L_f^{\text{init}}$; longer fibers are allowed to "steal" points from shorter fibers during growth.

## 8.2 End slices and local principal direction

For a given fiber $f$, we consider each end (max and min along $t_t$). Let:

$$t_{\max} = \max_{t\in\mathcal{I}_f} t_t, \quad t_{\min} = \min_{t\in\mathcal{I}_f} t_t.$$

**Max end.** We define the end slice set

$$\mathcal{I}_f^{\max} = \{t \in \mathcal{I}_f \mid t_t \geq t_{\max} - \Delta_{\text{end}}\},$$

where $\Delta_{\text{end}}$ is `END_SLICE_LEN`. If $|\mathcal{I}_f^{\max}| < N_{\text{end}}$ (`MIN_PTS_END`), we do not grow at this end.

We compute local PCA:

$$\boldsymbol{\mu}_f^{\max} = \frac{1}{|\mathcal{I}_f^{\max}|} \sum_{t\in\mathcal{I}_f^{\max}} \bar{\mathbf{p}}_t,$$

and a local principal direction $\mathbf{v}_f^{\max}$ (first PC).

We then orient $\mathbf{v}_f^{\max}$ to be consistent with outward growth:

$$d = \mathbf{v}_f^{\max}\cdot\mathbf{v}_f.$$

For the max end, we want $\mathbf{v}_f^{\text{out}}$ to point toward increasing $t_t$. Thus,

$$\mathbf{v}_f^{\text{out}} = \begin{cases} \mathbf{v}_f^{\max}, & d \geq 0, \\ -\mathbf{v}_f^{\max}, & d < 0. \end{cases}$$

**Min end.** Similarly, for the min end:

$$\mathcal{I}_f^{\min} = \{t \in \mathcal{I}_f \mid t_t \leq t_{\min} + \Delta_{\text{end}}\},$$

and $\mathbf{v}_f^{\text{out}}$ is oriented such that it points toward decreasing $t_t$ (i.e. opposite to $\mathbf{v}_f$).

## 8.3 Radius estimation in the end slice

For each end slice, we express the points in local coordinates relative to $\boldsymbol{\mu}_f^{\mathrm{end}}$ and $\mathbf{v}_f^{\mathrm{out}}$:

$$\tilde{\mathbf{p}}_t = \bar{\mathbf{p}}_t - \boldsymbol{\mu}_f^{\mathrm{end}},$$

$$\tau_t = \tilde{\mathbf{p}}_t \cdot \mathbf{v}_f^{\mathrm{out}}, \quad \mathbf{r}_t = \tilde{\mathbf{p}}_t - \tau_t \, \mathbf{v}_f^{\mathrm{out}}.$$

Here $\tau_t$ is the coordinate along the local axis, and $\mathbf{r}_t$ is the radial offset. The radial distances are

$$r_t = \|\mathbf{r}_t\|_2.$$

We estimate a characteristic radius $r_{\mathrm{end}}$ from the distribution of $r_t$ (e.g. a high percentile such as 95th) and cap it by the global maximum radius `MAX_RADIUS`:

$$r_{\mathrm{end}} = \min\big(\mathrm{quantile}(r_t, q),\, r_{\max}\big),$$

where $q$ is a chosen quantile and $r_{\max}$ is `MAX_RADIUS`.

We also define the tip position (furthest along $\mathbf{v}_f^{\mathrm{out}}$):

$$\tau_{\mathrm{tip}} = \max_{t \in \mathcal{I}_f^{\mathrm{end}}} \tau_t.$$

## 8.4 Extension interval and candidate region

We define an extension interval along the axis:

$$\tau_{\mathrm{start}} = \tau_{\mathrm{tip}}, \quad \tau_{\mathrm{stop}} = \tau_{\mathrm{tip}} + \Delta_{\mathrm{ext}},$$

where $\Delta_{\mathrm{ext}}$ is `EXTEND_STEP`. The mid–point is $\tau_{\mathrm{mid}} = (\tau_{\mathrm{start}} + \tau_{\mathrm{stop}})/2$. In 3D, the mid extension center is

$$\mathbf{m}_{\mathrm{mid}} = \boldsymbol{\mu}_f^{\mathrm{end}} + \tau_{\mathrm{mid}} \, \mathbf{v}_f^{\mathrm{out}}.$$

To avoid scanning all points, we define a bounding sphere of radius

$$R_{\mathrm{sphere}} = \sqrt{\left(\frac{\Delta_{\mathrm{ext}}}{2}\right)^2 + r_{\mathrm{end}}^2}$$

around $\mathbf{m}_{\mathrm{mid}}$. Candidate points $j$ are those for which

$$\|\bar{\mathbf{p}}_j - \mathbf{m}_{\mathrm{mid}}\|_2 \le R_{\mathrm{sphere}}.$$

## 8.5 Acceptance criteria and label reassignment

For each candidate point $j$ with current label $\ell_j$ (possibly $-1$ for noise), we compute local coordinates w.r.t. the end slice frame:

$$\tilde{\mathbf{p}}_j = \bar{\mathbf{p}}_j - \boldsymbol{\mu}_f^{\mathrm{end}},$$

$$\tau_j = \tilde{\mathbf{p}}_j \cdot \mathbf{v}_f^{\mathrm{out}}, \quad \mathbf{r}_j = \tilde{\mathbf{p}}_j - \tau_j \, \mathbf{v}_f^{\mathrm{out}}, \quad r_j = \|\mathbf{r}_j\|_2.$$

We accept $j$ into fiber $f$ if:

1. $\tau_j \in (\tau_{\mathrm{start}}, \tau_{\mathrm{stop}}]$ (point lies in the extension slab),
2. $r_j \le r_{\mathrm{end}}$ (point is inside the cylinder),

3. either $\ell_j = -1$ (noise/unassigned) or $f$ has higher rank (is longer) than the current fiber $\ell_j$.

If accepted, we set $\ell_j \leftarrow f$. This is done in a single growth step. Growth is applied iteratively:

- For each global iteration, we sweep fibers in order of decreasing $L_f^{\text{init}}$.
- For each fiber $f$, we recompute $P_f$, $t_t$, and end slices, and call the growth procedure at both ends.
- If in an iteration no point changes its label, we declare convergence and stop.

The final labels are stored as fiber_grow_id($t$) for each averaged point and merged back into the raw localization table by joining on `TraceID`.

## 9 Fiber Length and Width Metrics

This section describes the computation of per–fiber length and width.

Implementation:

- Module: `fiber_pipeline.metrics`
- The code there:
  - aggregates per–fiber 3D centerline and/or point sets,
  - computes lengths from centerlines,
  - computes widths from radial statistics,
  - writes `fiber_metrics.csv` with columns: `fiber_id`, `fiber_length_um`, `fiber_width_um`.

### 9.1 Fiber length

For a given fiber $f$, suppose we have a 3D centerline represented by an ordered sequence

$$\{\mathbf{C}_{f,1}, \mathbf{C}_{f,2}, \dots, \mathbf{C}_{f,M_f}\}, \quad \mathbf{C}_{f,j} \in \mathbb{R}^3.$$

We estimate the fiber length as the polyline arc length:

$$L_f = \sum_{j=1}^{M_f-1} \|\mathbf{C}_{f,j+1} - \mathbf{C}_{f,j}\|_2. \tag{18}$$

In cases where a full 3D centerline is not available, a fallback based on 2D centerline length or PCA–based length can be used.

### 9.2 Fiber width (effective diameter)

A typical approach is to characterize the radial distribution of points around the fiber's main axis. Let $\{\bar{\mathbf{p}}_t \mid \text{fiber\_grow\_id}(t) = f\}$ be the points assigned to fiber $f$ in 3D. We compute a principal axis $\mathbf{v}_f$ and center $\boldsymbol{\mu}_f$ as in Section 8.

For each point:

$$\tilde{\mathbf{p}}_t = \bar{\mathbf{p}}_t - \boldsymbol{\mu}_f, \quad \tau_t = \tilde{\mathbf{p}}_t \cdot \mathbf{v}_f, \quad \mathbf{r}_t = \tilde{\mathbf{p}}_t - \tau_t \mathbf{v}_f, \quad r_t = \|\mathbf{r}_t\|_2.$$

The width can then be defined as

$$W_f = 2\, R_f, \tag{19}$$

where $R_f$ is a robust radial scale, e.g. a chosen percentile (such as the 95th) of $\{r_t\}$ or a similar statistic. The exact choice may be configured in the code.

Thus the final metrics per fiber are:

$$\texttt{fiber\_length\_um}(f) = L_f, \quad \texttt{fiber\_width\_um}(f) = W_f.$$

## 10 Implementation Map Summary

For convenience, we summarize the mapping from conceptual steps to modules and main functions:

- **Configuration and entry point**
  - `fiber_pipeline.config`: all hyperparameters (`PX_PER_UM`, thresholds, radii, etc.).
  - `fiber_pipeline.run_folder`: scans input folder, orchestrates the pipeline for each sample, creates per–sample result directories.
- **2D KDE & tubeness** (Section 2)
  - `kde_tubeness.compute_kde_counts`
  - `kde_tubeness.normalize_to_8bit`
  - `kde_tubeness.tubeness_from_kde_gray8`
- **2D segmentation** (Section 3)
  - `segment_2d.to_gray01`
  - `segment_2d.segment_fibers`
  - `segment_2d.find_endpoints`
  - `segment_2d.bridge_small_gaps`
- **Skeleton graph** (Section 4)
  - `skeleton_graph` module: degree computation, endpoint/junction masks, connected components of junctions, node/segment construction, neighbors, `junction_exits`, `endpoint_exits`.
- **Segment merging** (Section 5)
  - `merge_segments`: per–segment orientation, adjacency, mutual best–neighbor loop, branch splitting.
- **2D→3D mapping and tubes** (Section 7)
  - `tubes_3d.pix_to_um`
  - `tubes_3d.build_segment_polyline_um`
  - `tubes_3d.resample_polyline`
  - `tubes_3d` assignment of averaged points to fibers and 3D centerline construction.
- **Cylinder growth** (Section 8)

  - `grow_cylinders.first_pc`
  - `grow_cylinders.grow_fiber_once`
  - global cylinder growth loop updating `fiber_grow_id`.

- **Metrics** (Section 9)
  - `metrics` module: aggregates per–fiber points/centerlines to compute `fiber_length_um` and `fiber_width_um`, outputs `fiber_metrics.csv`.