Advances on Graph-Based Machine Learning Algorithms for Image Analysis

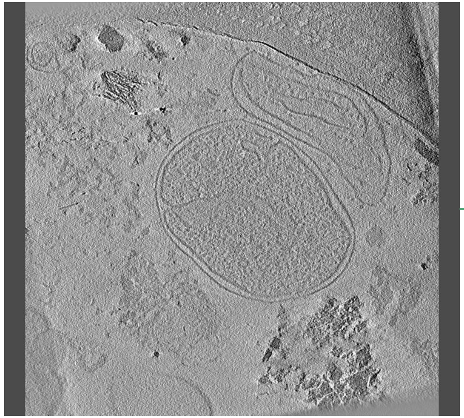
Talita Perciano

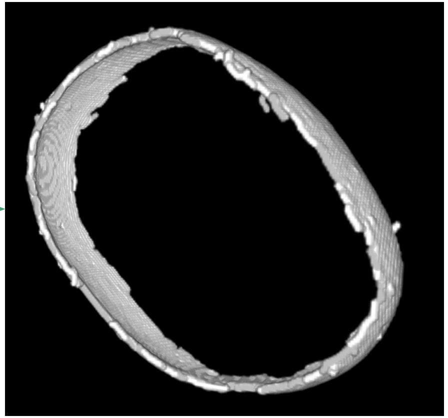
Data Analytics & Visualization Group
Computational Biosciences Group
Computational Research Division
Lawrence Berkeley National Laboratory

Collaborators

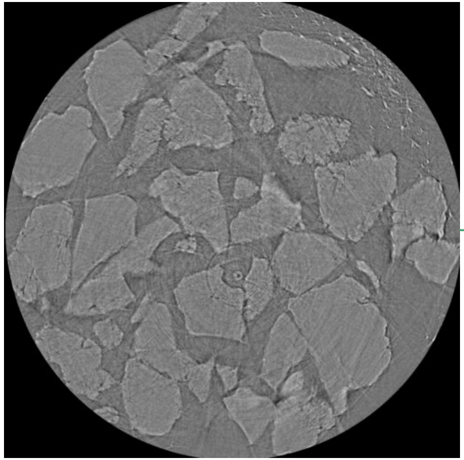
From images to knowledge... efficiently!

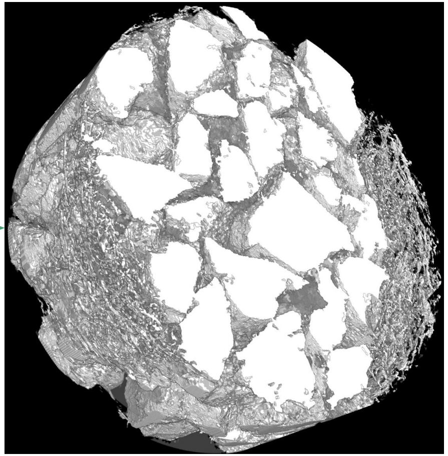
Cryo-ET





Micro-CT





Outline

1. Motivation
2. Basic concepts
3. Interactive Machine Learning for Tomogram Segmentation
    a. Electron Cryotomography
    b. Graph-based unsupervised segmentation
    c. Results
    d. Python code
4. Parallel Markov Random Fields
    a. Micro-Computed Tomography
    b. Markov Random Fields
    c. Results
5. Final Remarks

Motivation

Research under DOE mission science

- Large amount of research relies on image-based data
- Amount of data continues to increase
- Science questions are increasing in complexity and sophistication
- Opportunity to improve data analysis algorithms and software
- Enable accurate and deep understanding for decision-making
- Analysis bottlenecks: unsuitable data representation, optimization taking into account the veracity of the data, use physical constraints, consider multiple scales and dimensions, computational complexity

Example

The Transmission Electron Aberration-corrected Microscope (TEAM 0.5) at Berkeley Lab has been upgraded with a new detector that can capture atomic-scale images in millionths-of-a-second increments. (Credit: Thor Swift/Berkeley Lab)

The 4D Camera - Dynamic Diffraction Direct Detector

- Latest innovation in EM
- EM experiments: amount of information used among all the possible information generated as the microscope's beam interacts with samples
- 4D Camera: captures all!
- Fast, high-resolution microscopy => generating 4 terabytes of data per minute
- Atomic-scale images in millionths-of-a-second

3D images of platinum particles between 2-3 nanometers in diameter shown rotating in liquid under an electron microscope. Each nanoparticle has approximately 600 atoms. White spheres indicate the position of each atom in a nanoparticle. (Courtesy of IBS)

Basic Concepts

How and why graphs?

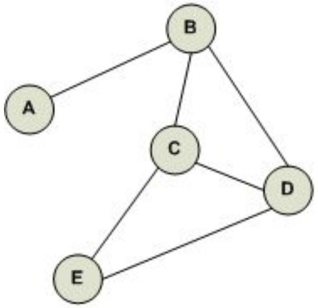
- **Discrete** and **mathematically** simple representation: efficiency and correctness
- Minimalistic representation: **flexibility**
- **Graph theory** is out there already!
- Allows for **structural representation**

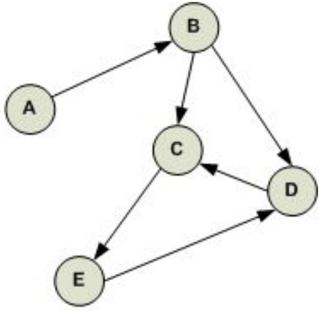
Graphs

A graph is a set of vertices and edges G={V,E}

V = {A, B, C, D, E} E = {AB, BC, BD, CD, CE, ED}

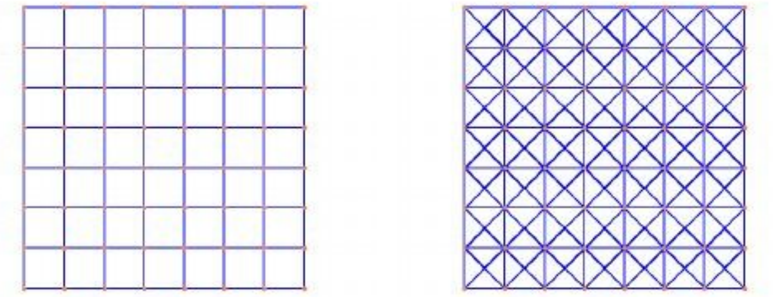
- **Node**: fundamental unit out of which graphs are formed
- **Edge**: gives relationship between vertices
- Important terms: adjacency, complete graph, subgraph, cliques, neighborhood
- Directed vs undirected?



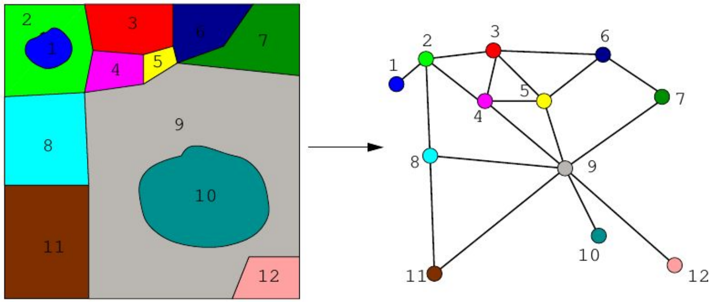


Graphs from images

Pixel-based graph



Region-based graph



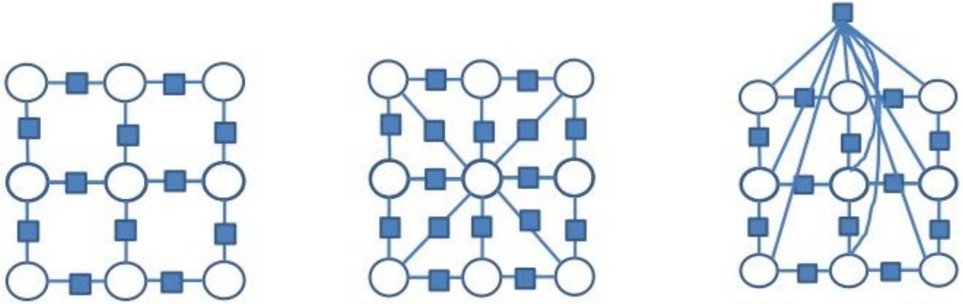
Important to notice: **nodes** and **neighborhood**

Markov Random Fields

Energy function with two terms:

1. Data term
2. Smoothness term

Usually we want to minimize this energy function to find the best "graph configuration" (with highest probability)



4-connected; pairwise MRF

$$E(x) = \sum_{i,j \in N_4} \theta_{ij}(x_i, x_j)$$

Order 2

"Pairwise energy"

higher(8)-connected; pairwise MRF

$$E(x) = \sum_{i,j \in N_8} \theta_{ij}(x_i, x_j)$$

Order 2

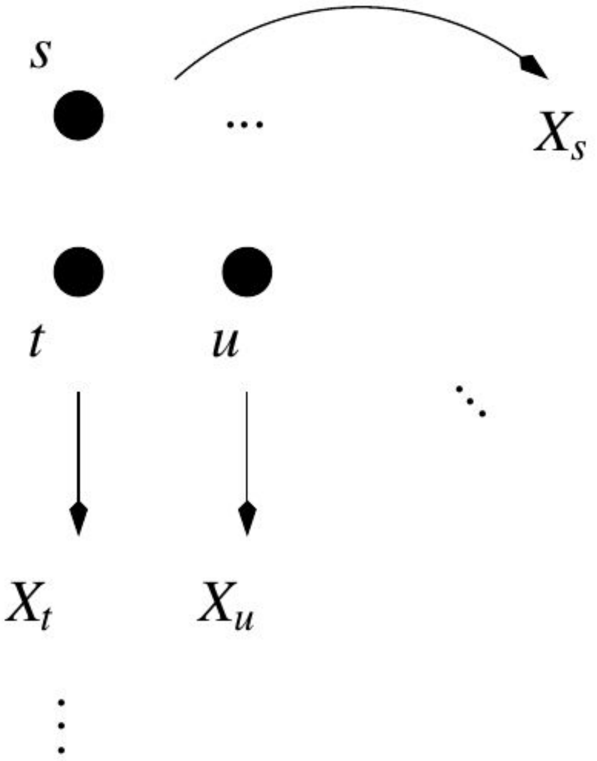
Higher-order RF

$$E(x) = \sum_{i,j \in N_4} \theta_{ij}(x_i, x_j) + \theta(x_1, \dots, x_n)$$

Order n

"higher-order energy"

Markov Random Fields



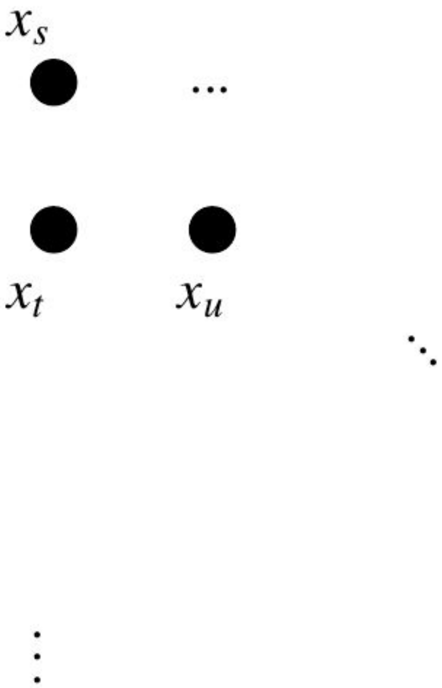
$\mathbb{L}$

$X_s = x_s$

$$X = (X_s; X_t; \ldots)$$

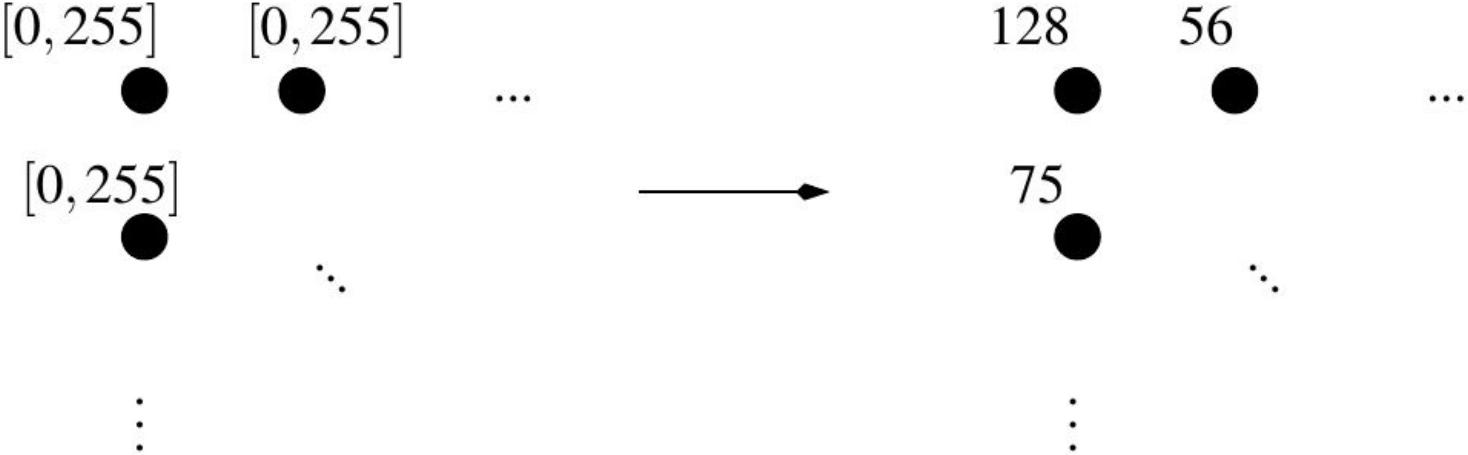
$$(X_s = x_s; X_t = x_t; \ldots)$$

Markov Random Fields



$$x = (x_s; x_t; \ldots)$$

Markov Random Fields



Markov Random Fields

The probability of the random value X_s taking the value x_s is denoted $P(X_s = x_s)$ and the joint probability is given by $P(X = x) = P(X_s = x_s; X_t = x_t; \dots)$.

The joint probability allows us to calculate the image likelihood and the conditional local probabilities provide a way to mesure the statistical connections between a gray level and the rest of the image. Those probabilities can be calculated through the Markovian hypothesis.

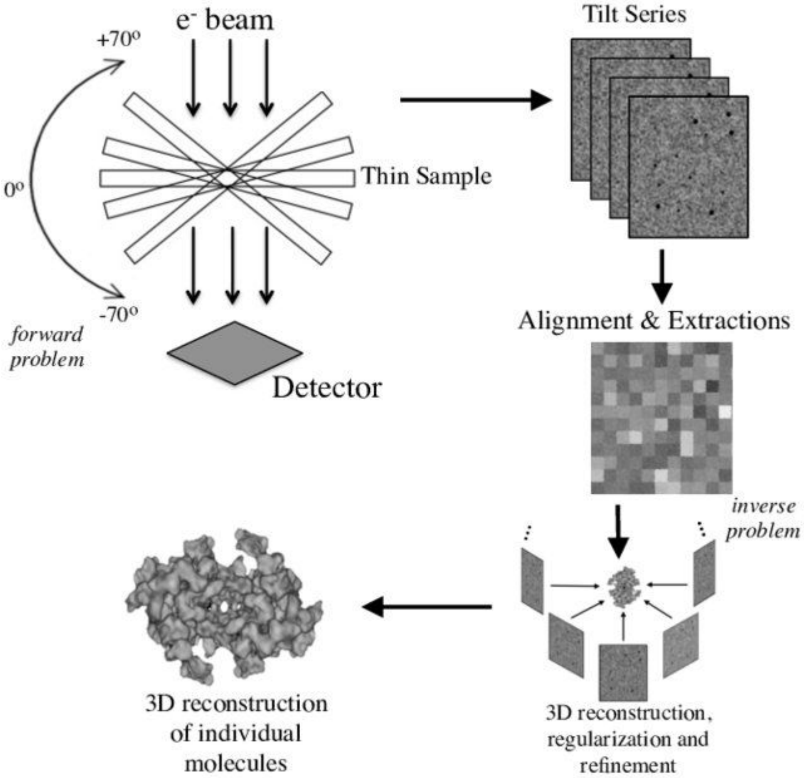
Interactive Machine Learning for Tomogram Segmentation

Electron Cryotomography - CryoET

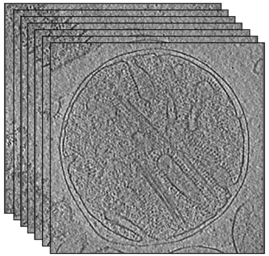
"An electron microscope is used to record a series of two-dimensional images as a biological sample held at cryogenic temperatures is tilted. Using computational methods, the two-dimensional images can be aligned to yield a three-dimensional (tomographic) reconstruction of the sample." Nature.com

Special type of CryoTEM. Samples are immobilized in non-crystalline ice and imaged under cryogenic conditions. Provides unique information on protein structure and interactions *in situ.*

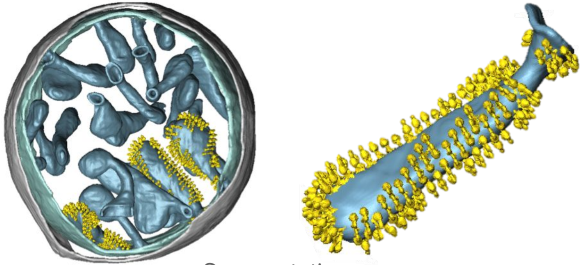
Electron Cryotomography - CryoET



Credit: Faisal Mahmood "An Extended Field-based method for Noise Removal from Electron Tomographic Reconstructions"



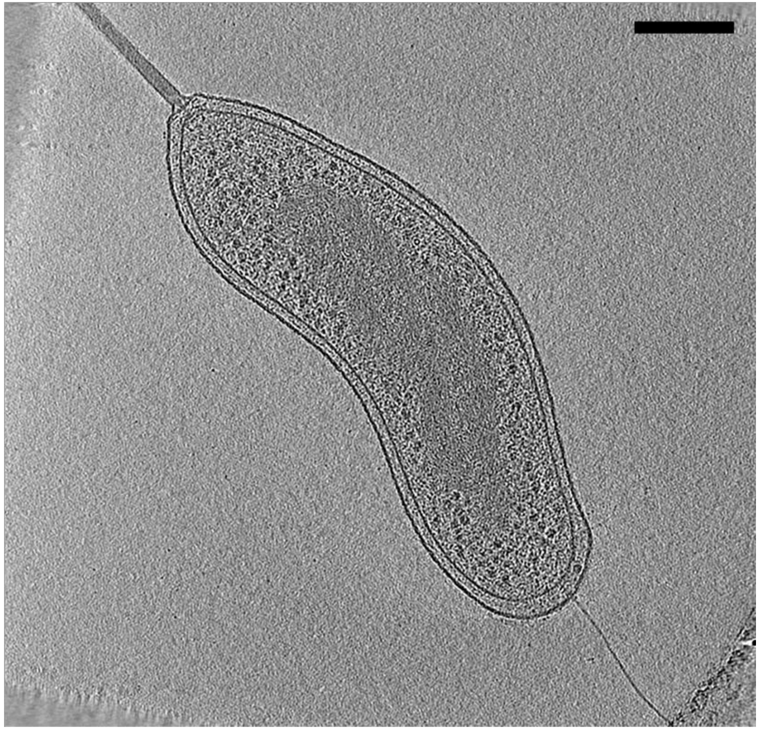
Tilt Series Collection



Segmentation

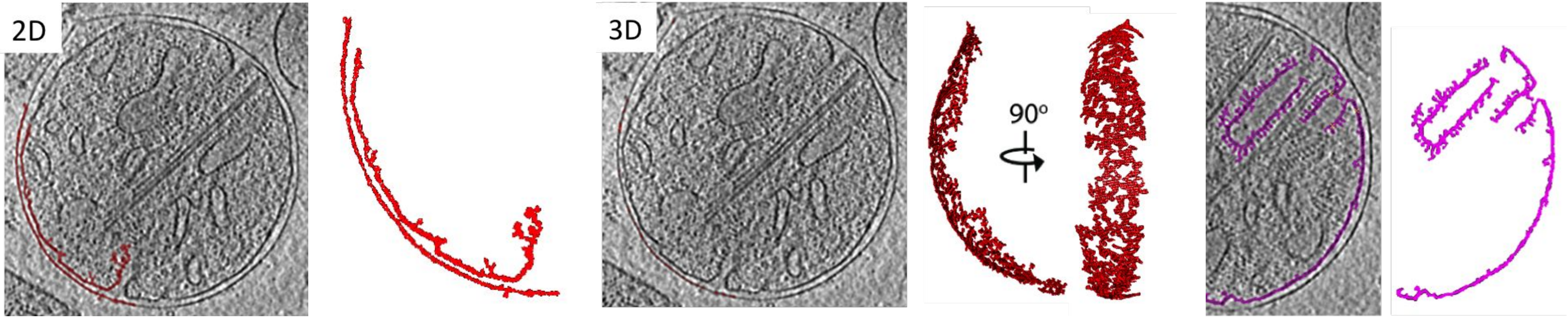
Electron Cryotomography - CryoET

- Unique details about specimens including subcellular organelles or structurally heterogeneous protein complexes
- Drug development through the study of drug liposome
- Because of the macromolecular resolution, used to study viruses and small cells



By Eikosi - Own work, CC BY-SA 4.0,
https://commons.wikimedia.org/w/index.php?curid=45409611

Issues with segmentation methods



1. Connections between inner and outer membrane prevents isolation of one membrane
2. Low SNR causes membranes to be rough/noisy
3. Variations in density results in holey membrane surface
4. Proteins and membranes can not be separated
5. Manual segmentation is the most effective method - **3 months of work**

Research goals

Algorithm that:

1. Detects and labels distinct cellular features
2. Distinguishes between proteins and membrane
3. Generated smooth surface for membranes, free from noise and artificial holes

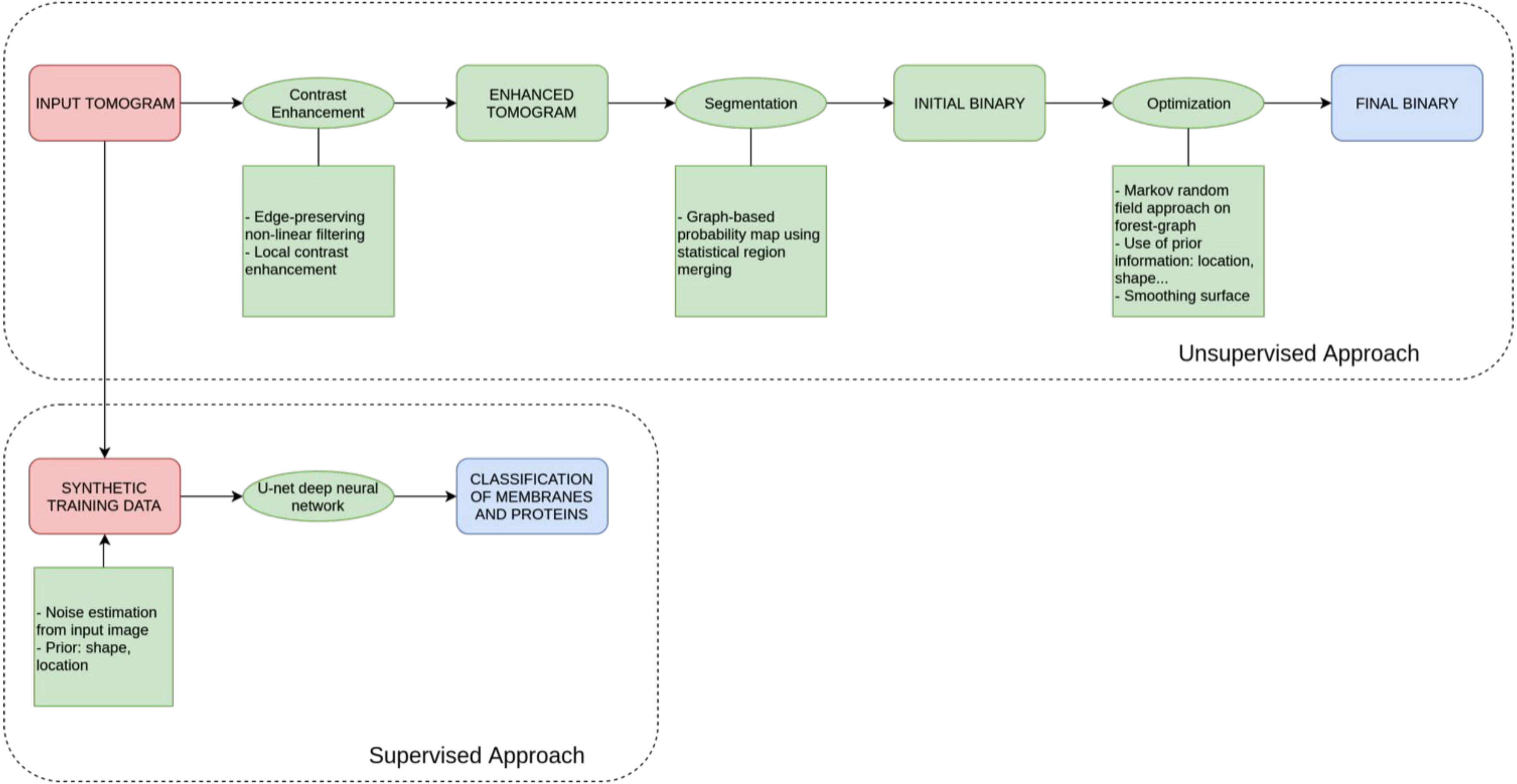
Approach:

1. Machine learning with user interaction

Novelties:

1. Using **prior knowledge** and **user input** to correct and direct segmentation
2. Not pixel based; **higher-level** (shape patterns) instead

General approach



INPUT TOMOGRAM

Contrast Enhancement

- Edge-preserving non-linear filtering
- Local contrast enhancement

ENHANCED TOMOGRAM

Segmentation

- Graph-based probability map using statistical region merging

INITIAL BINARY

Optimization

- Markov random field approach on forest-graph
- Use of prior information: location, shape...
- Smoothing surface

FINAL BINARY

Unsupervised Approach

SYNTHETIC TRAINING DATA

- Noise estimation from input image
- Prior: shape, location

U-net deep neural network

CLASSIFICATION OF MEMBRANES AND PROTEINS

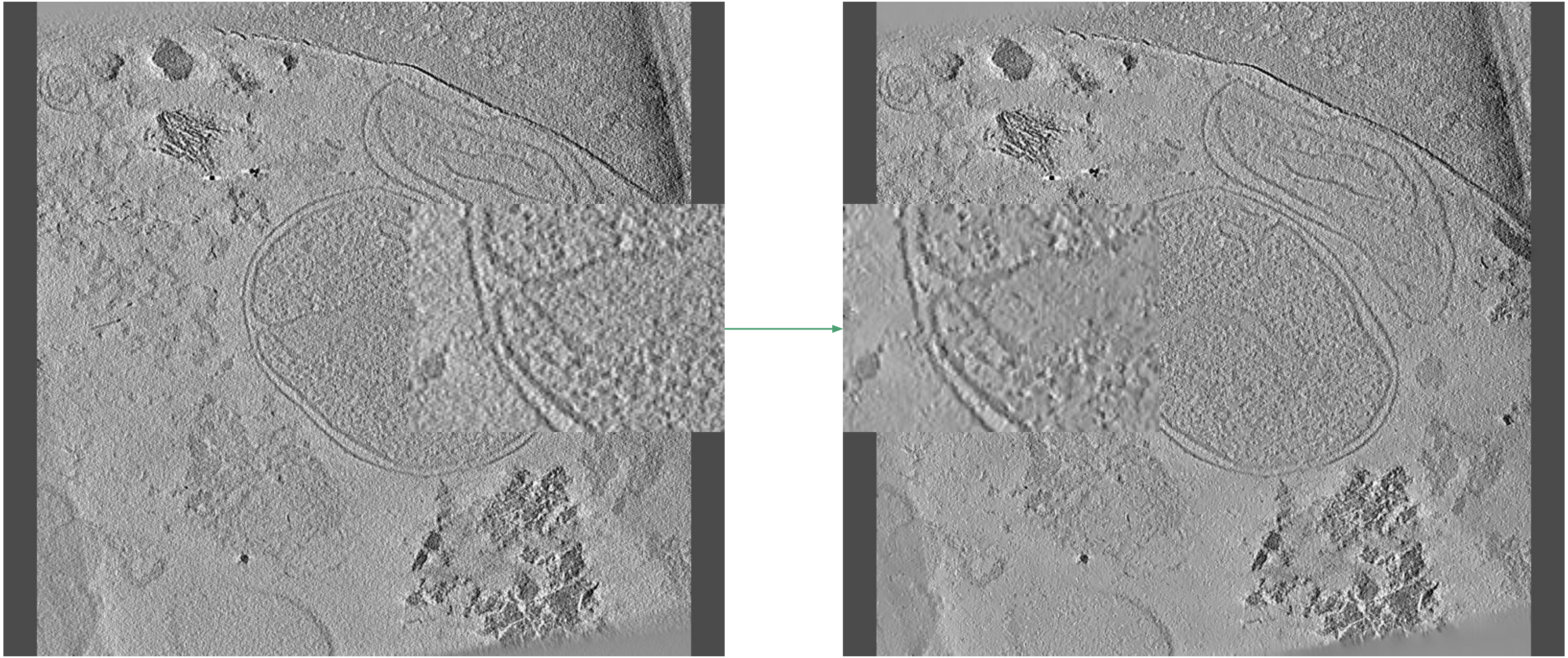
Supervised Approach

Non-local means denoising

The NLM algorithm replaces the value of a pixel by an average of a selection of other pixels values: small patches centered on the other pixels are compared to the patch centered on the pixel of interest, and the average is performed only for pixels that have patches close to the current patch. We estimate the noise standard deviation directly from the image. This algorithm performs well by reducing noise and restoring well textures that would be blurred by other denoising algorithms (resulting in preservation of valuable details).

Jacques Froment. Parameter-Free Fast Pixelwise Non-Local Means Denoising. Image Processing On Line, 2014, vol. 4, pp. 300-326. DOI: 10.5201/ipol.2014.120

Processing steps



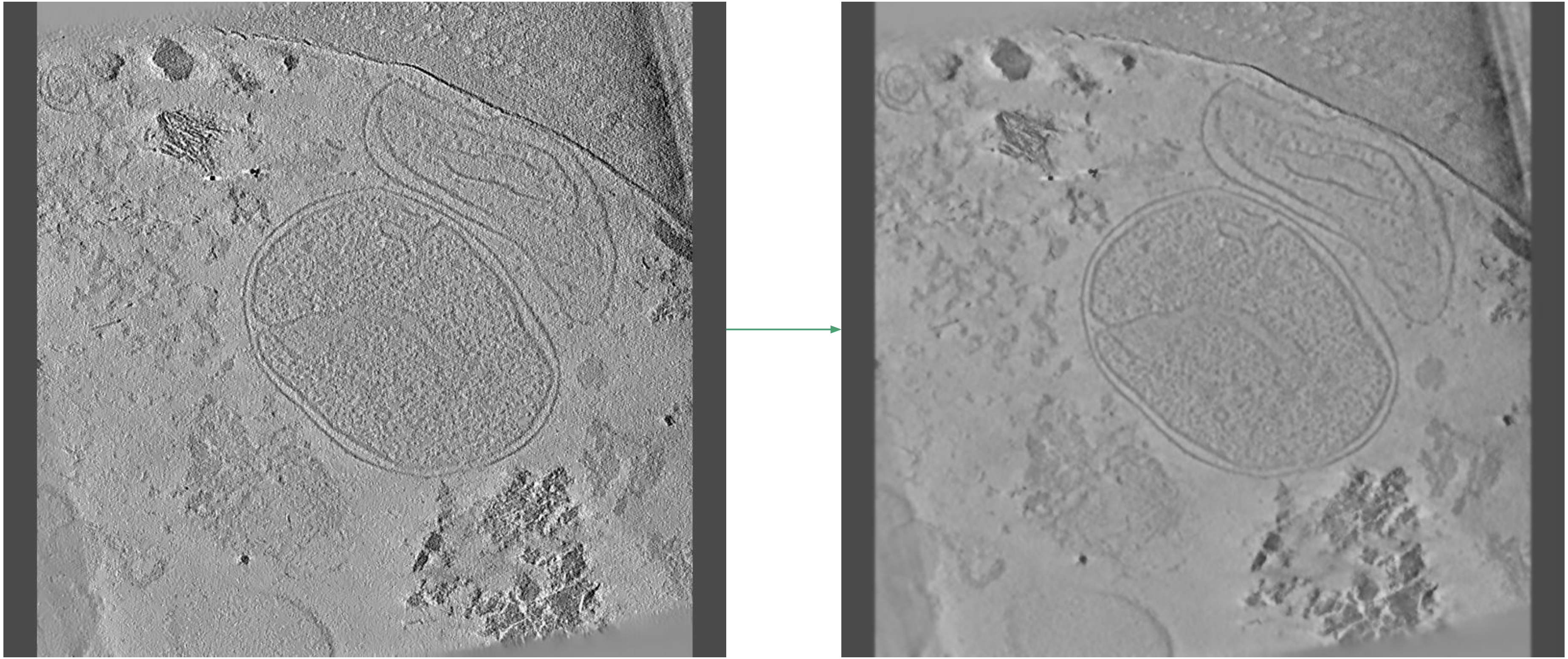
Non-local means filtering

Bilateral filter

This filter is an edge-preserving and noise reducing filter. It averages pixels based on their spatial closeness and radiometric similarity. In other words, it smooths homogeneous regions of the image and preserves details (such as borders of objects).

C. Tomasi and R. Manduchi. “Bilateral Filtering for Gray and Color Images.” IEEE International Conference on Computer Vision (1998) 839-846. DOI:10.1109/ICCV.1998.710815

Processing steps



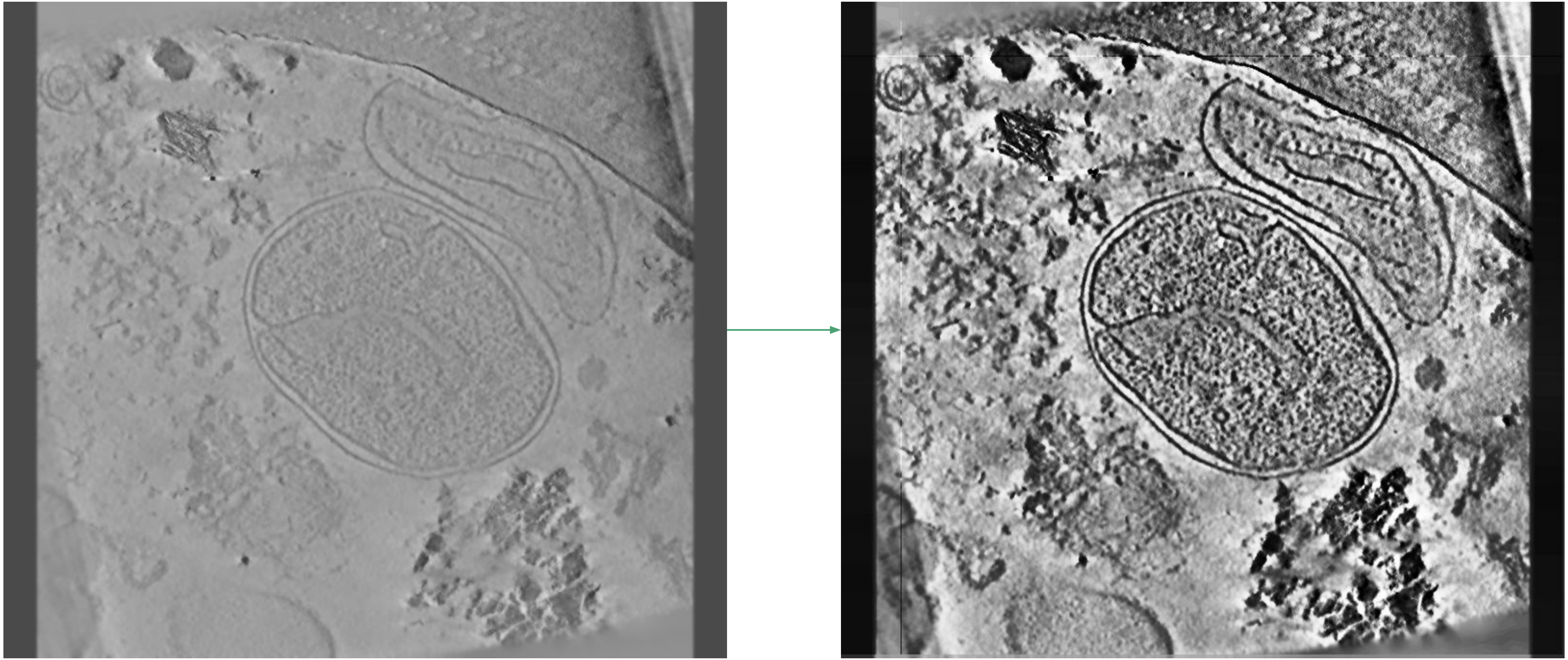
Bilateral filtering

Adaptive local contrast enhancement

This process applies a technique called Contrast Limited Adaptive Histogram Equalization (CLAHE). It uses histograms computed over different tile regions of the image. Local details can therefore be enhanced even in regions that are darker or lighter than most of the image.

Zuiderveld, Karel. “Contrast Limited Adaptive Histogram Equalization.” Graphic Gems IV. San Diego: Academic Press Professional, 1994. 474–485.

Processing steps



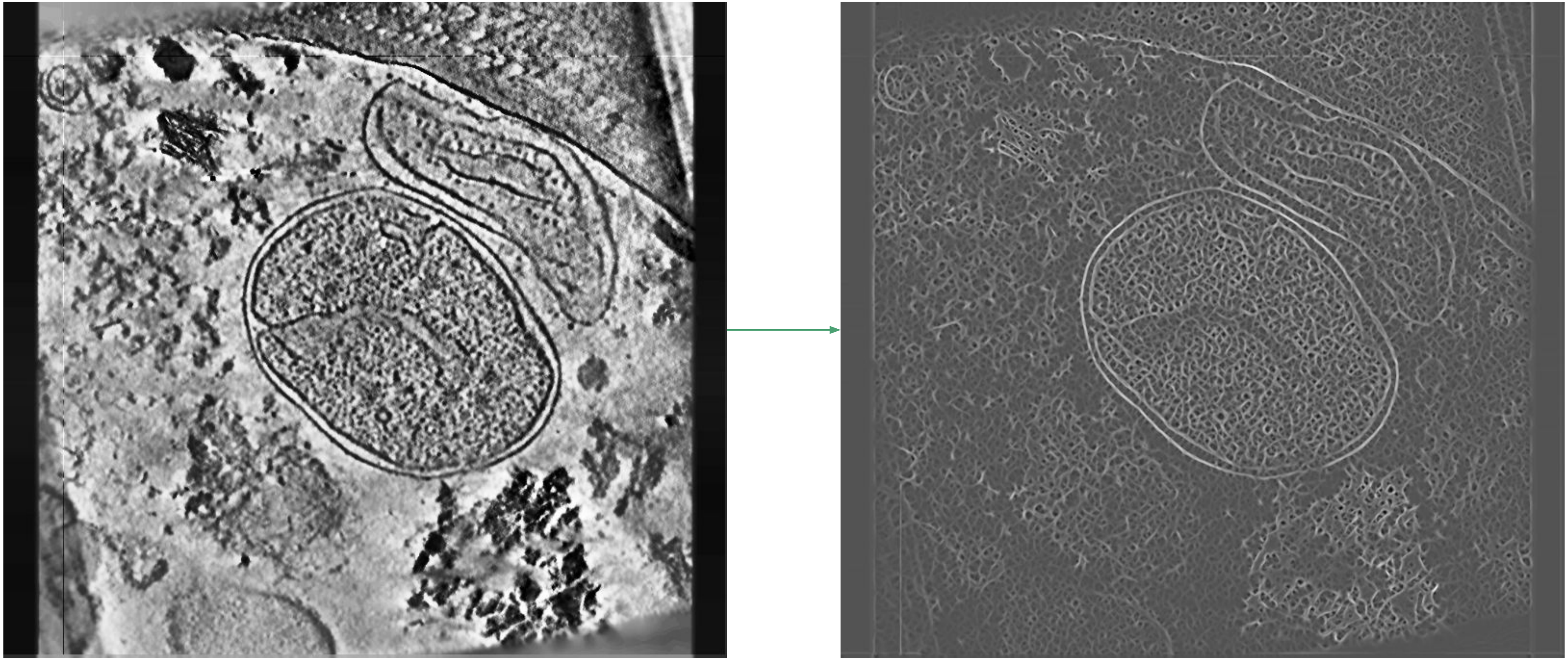
Adaptive local contrast enhancement

Ridge detection

We perform ridge detection through Hessian matrix calculation: we convolve the image with the second derivatives of a Gaussian kernel in different directions. Then we find the eigenvalues of the Hessian matrix, detecting ridge structure where the intensity changes perpendicular but not along the structure.

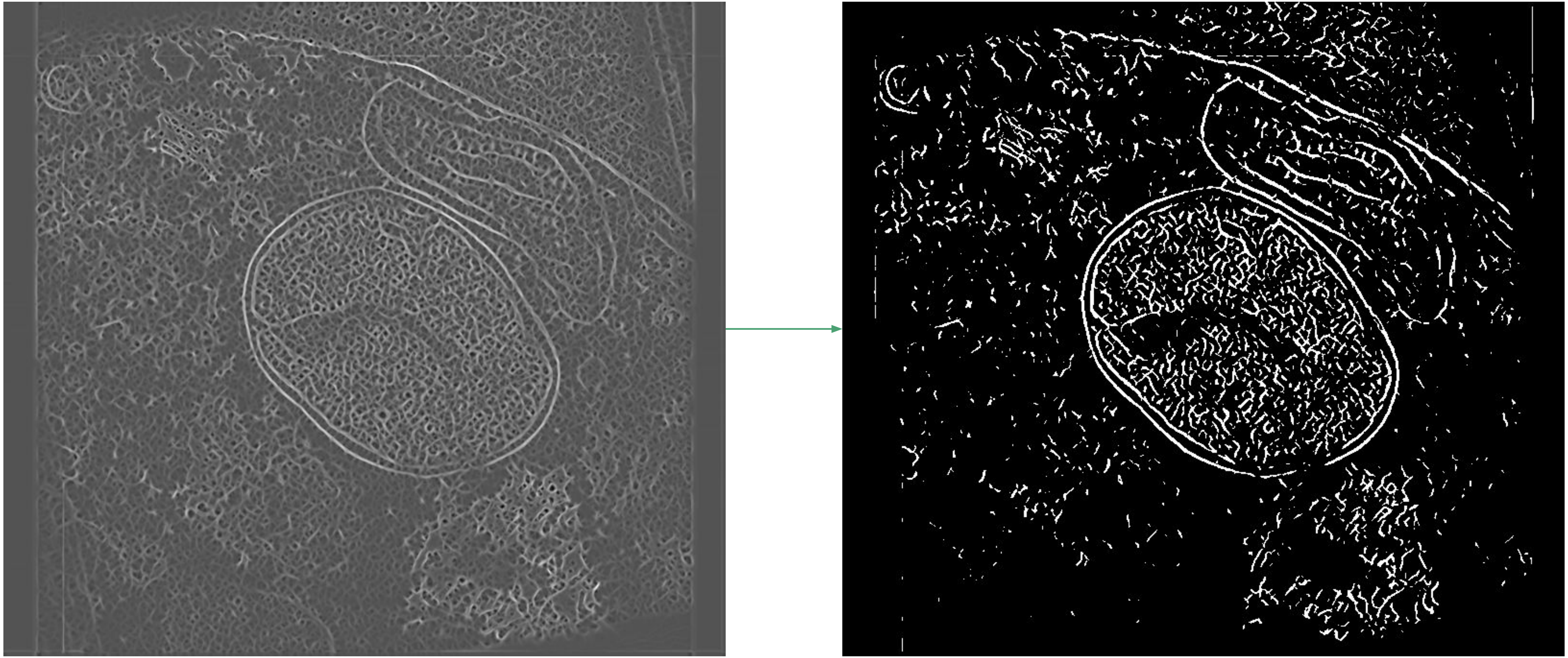
Ng, C. C., Yap, M. H., Costen, N., & Li, B. (2014, November). Automatic wrinkle detection using hybrid Hessian filter. In Asian Conference on Computer Vision (pp. 609-622). Springer International Publishing.
DOI:10.1007/978-3-319-16811-1_40

Processing steps



Ridge detection

Processing steps



Ridge detection

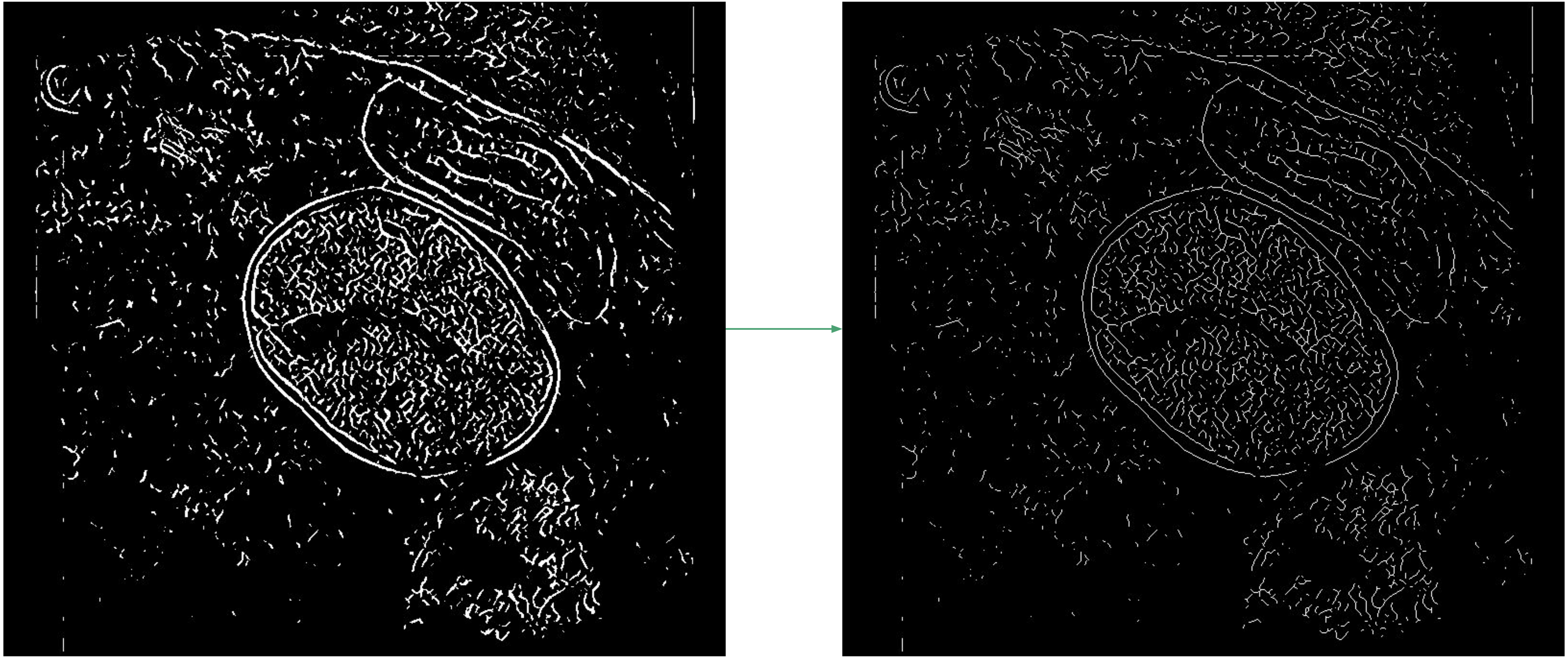
Skeletonization

The skeletonization process reduces binary objects to 1 pixel wide representations. The idea behind this process is to simplify connected components aiming feature extraction.

A fast parallel algorithm for thinning digital patterns, T. Y. Zhang and C. Y. Suen, Communications of the ACM, March 1984, Volume 27, Number 3.

T.-C. Lee, R.L. Kashyap and C.-N. Chu, Building skeleton models via 3-D medial surface/axis thinning algorithms. Computer Vision, Graphics, and Image Processing, 56(6):462-478, 1994.

Processing steps



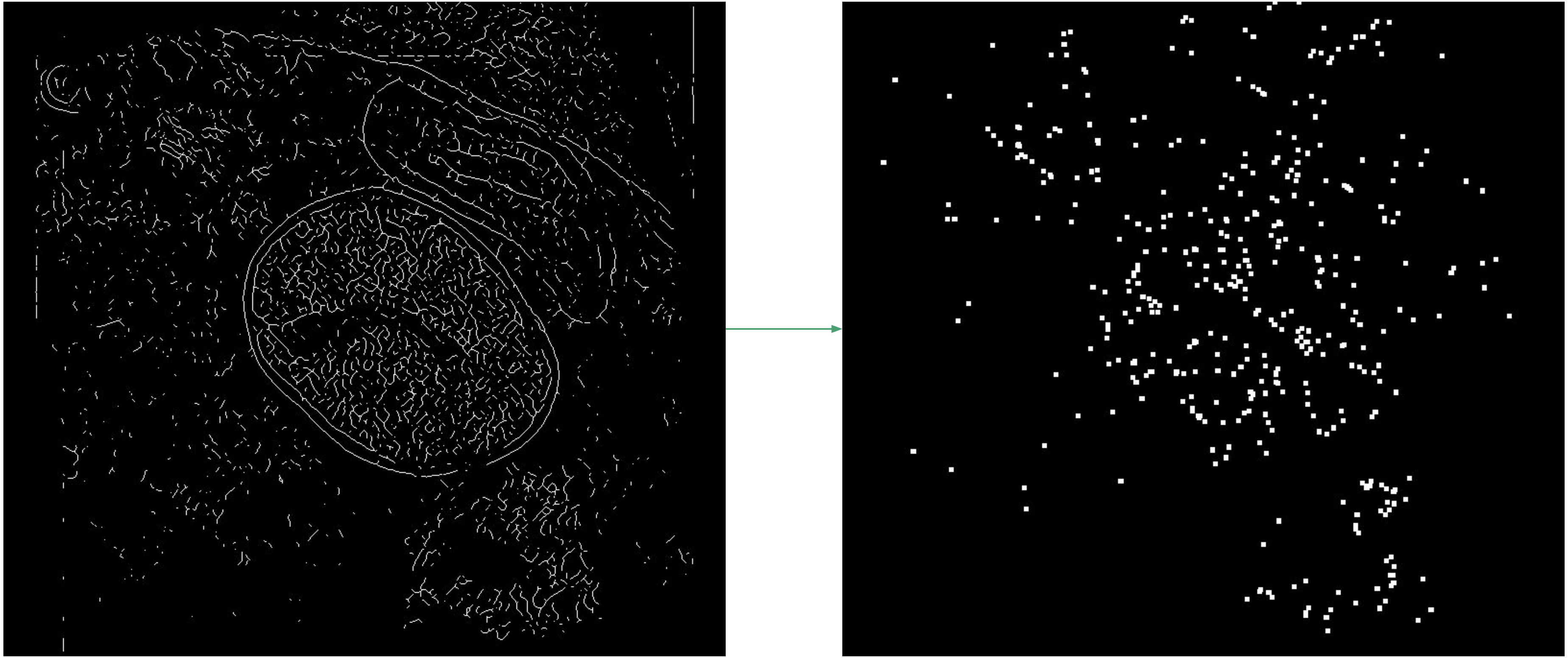
Skeletonization

Bifurcation detection

This step aims to simplify the skeleton by subdividing every connected component by detecting bifurcations. In the end of this process, every component in the image is a simple open curve. The bifurcations are detected using a process called morphological hit-or-miss, which finds a given configuration (in our case a possible bifurcation) in a binary image using the morphological erosion operator.

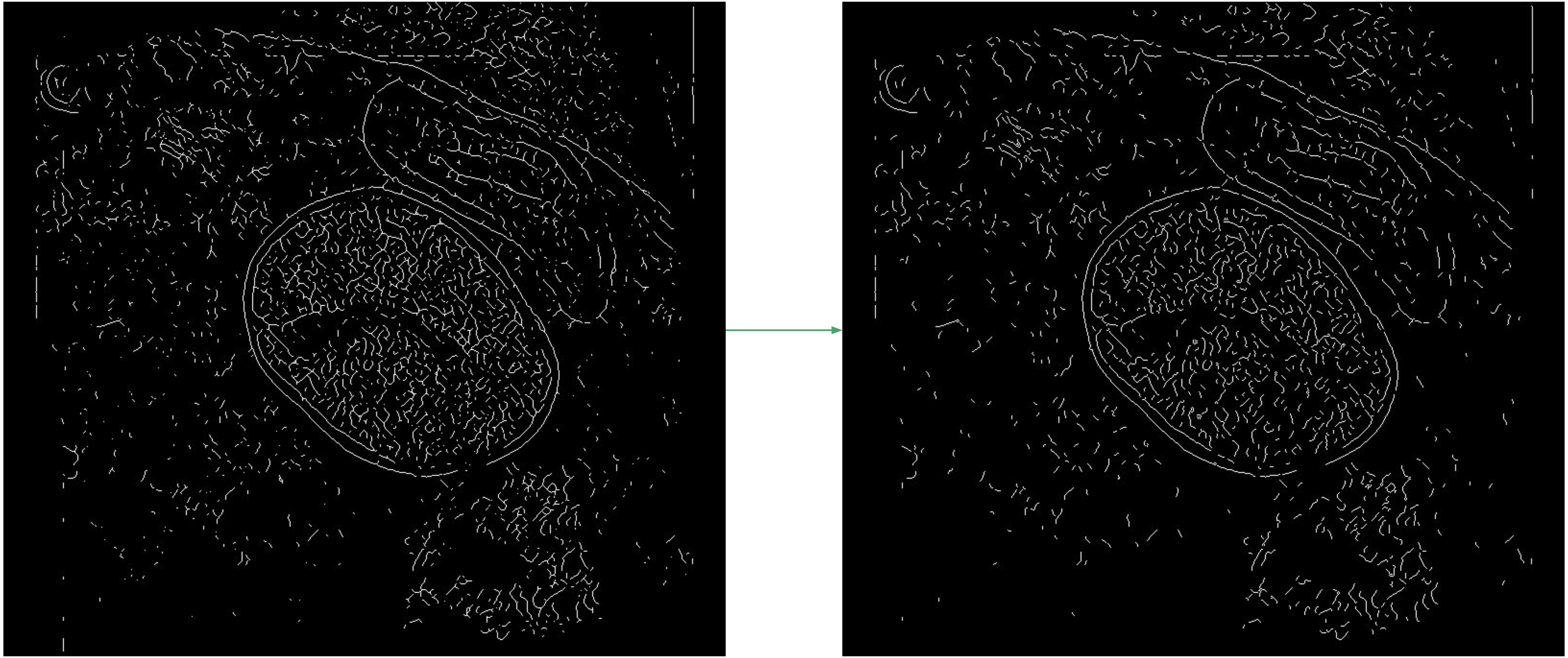
https://en.wikipedia.org/wiki/Hit-or-miss_transform

Processing steps



Bifurcation detection

Processing steps



Bifurcation detection

Geometric approximation

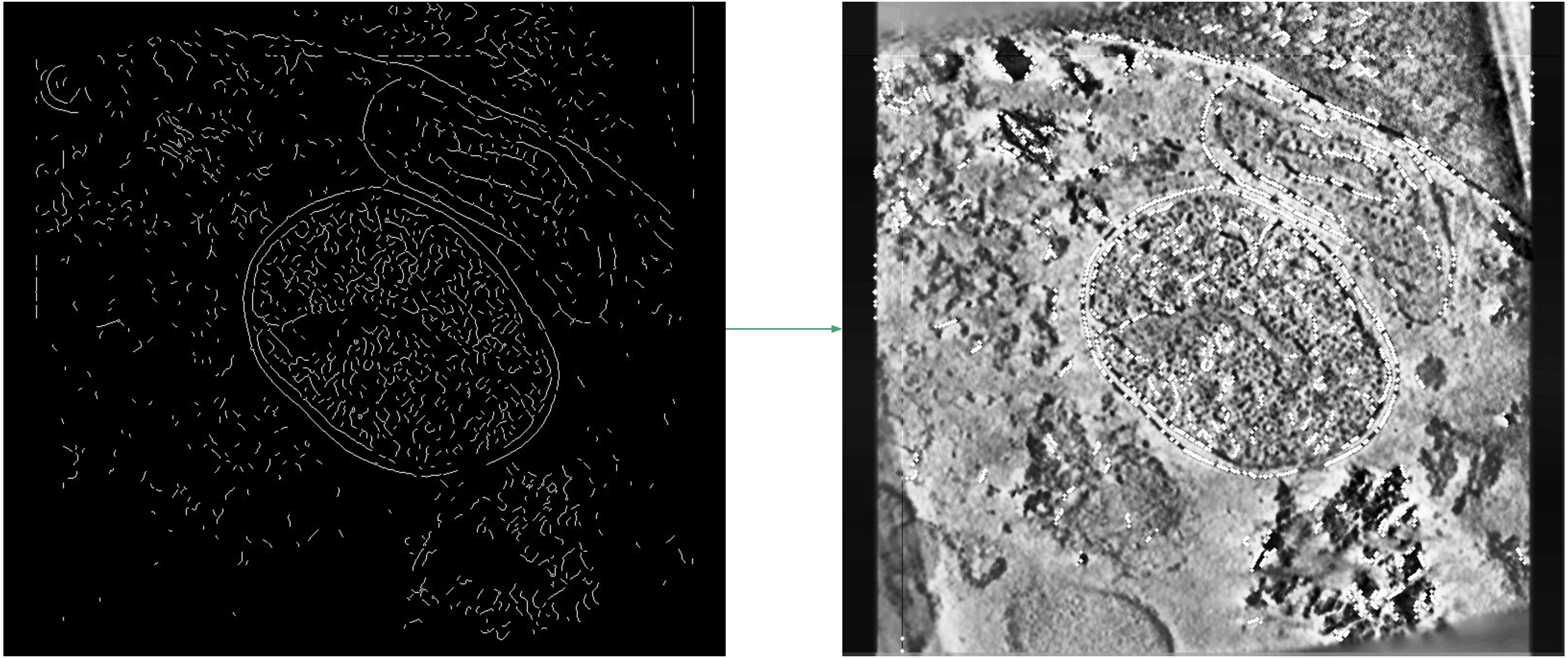
Now that the binary images contains components that are simple open curves, we go through a preprocessing for the graph construction step. Here, we approximate each curve by simple straight lines.

Formally, the algorithm approximates a curve/polygon with another curve/polygon with less vertices so that the distance between them is less or equal to the specified precision. The algorithm used is called Douglas-Peucker algorithm.

Prasad, Dilip K.; Leung, Maylor K.H.; Quek, Chai; Cho, Siu-Yeung (2012). "A novel framework for making dominant point detection methods non-parametric". Image and Vision Computing. 30 (11): 843–859. doi:10.1016/j.imavis.2012.06.010.

Wu, Shin-Ting; Marquez, Mercedes (2003). "A non-self-intersection Douglas-Peucker algorithm". 16th Brazilian Symposium on Computer Graphics and Image Processing (SIBGRAPI 2003). Sao Carlos, Brazil: IEEE. pp. 60–66. CiteSeerX 10.1.1.73.5773. doi:10.1109/SIBGRA.2003.1240992. ISBN 978-0-7695-2032-2.

Processing steps



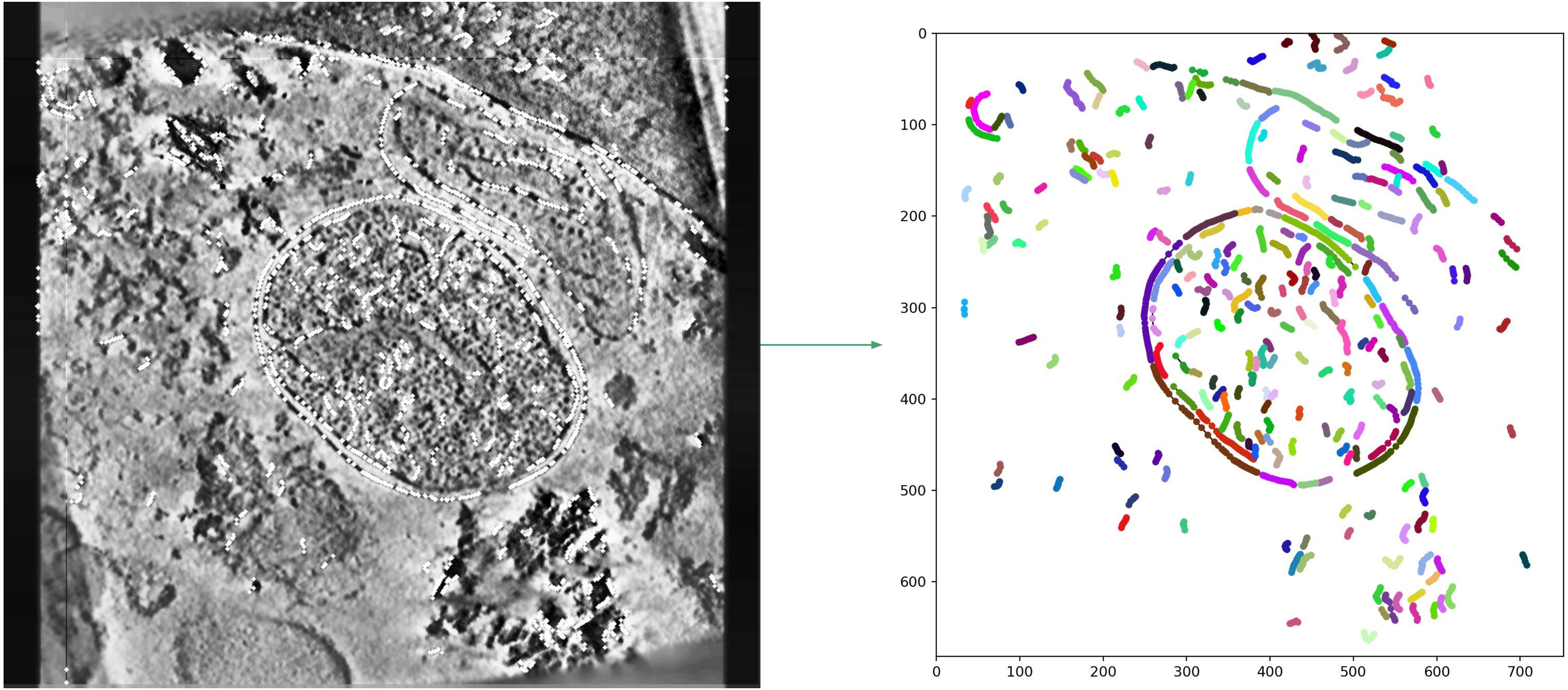
Approximation points

Low-level graph representation

In this step, we represent the structures in the image as a graph:

- Each node of the graph is a line segment obtained from the previous step
- Two nodes are connected if they are in the same curve
- With this process, we obtain what is called a forest (a collection of tree-like graphs)

Processing steps



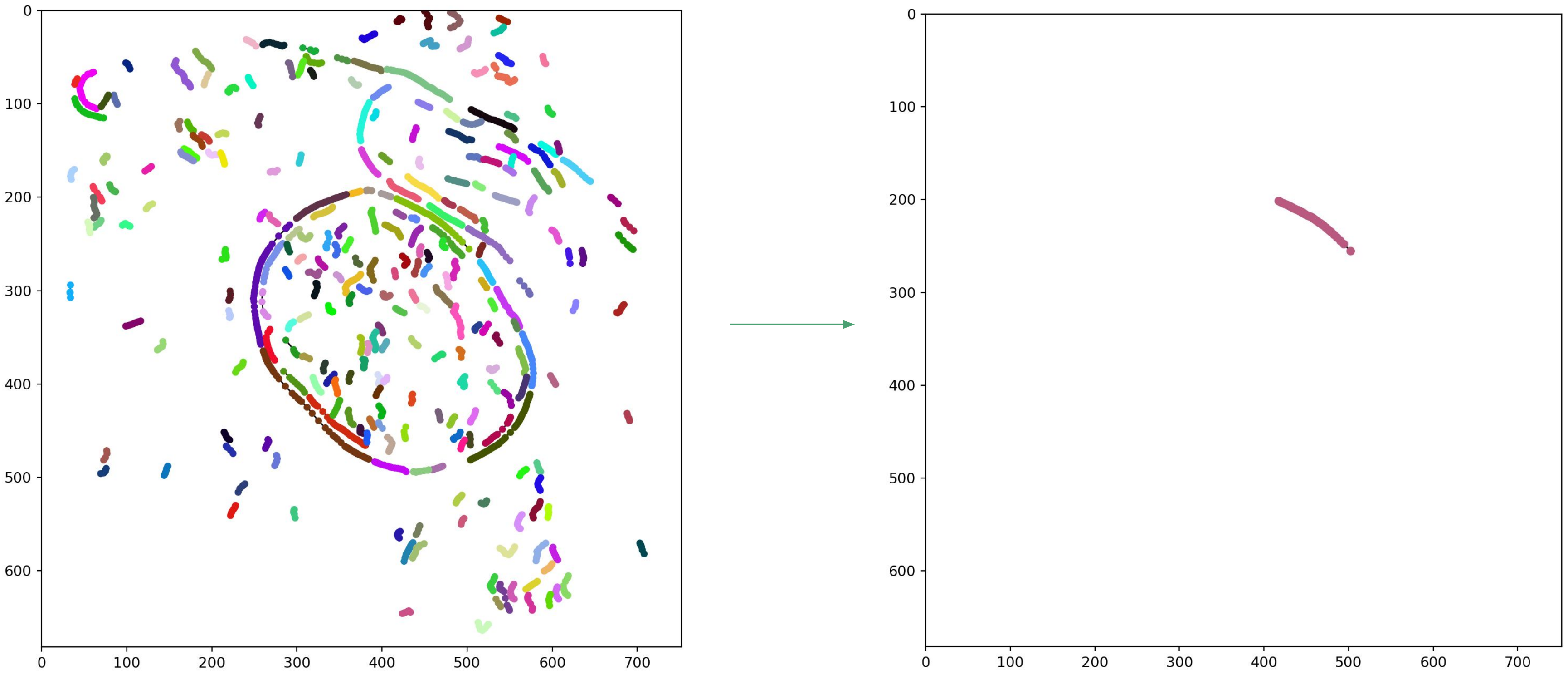
Low-level graph

Object reconstruction using MRF model

Outer membrane reconstruction process:

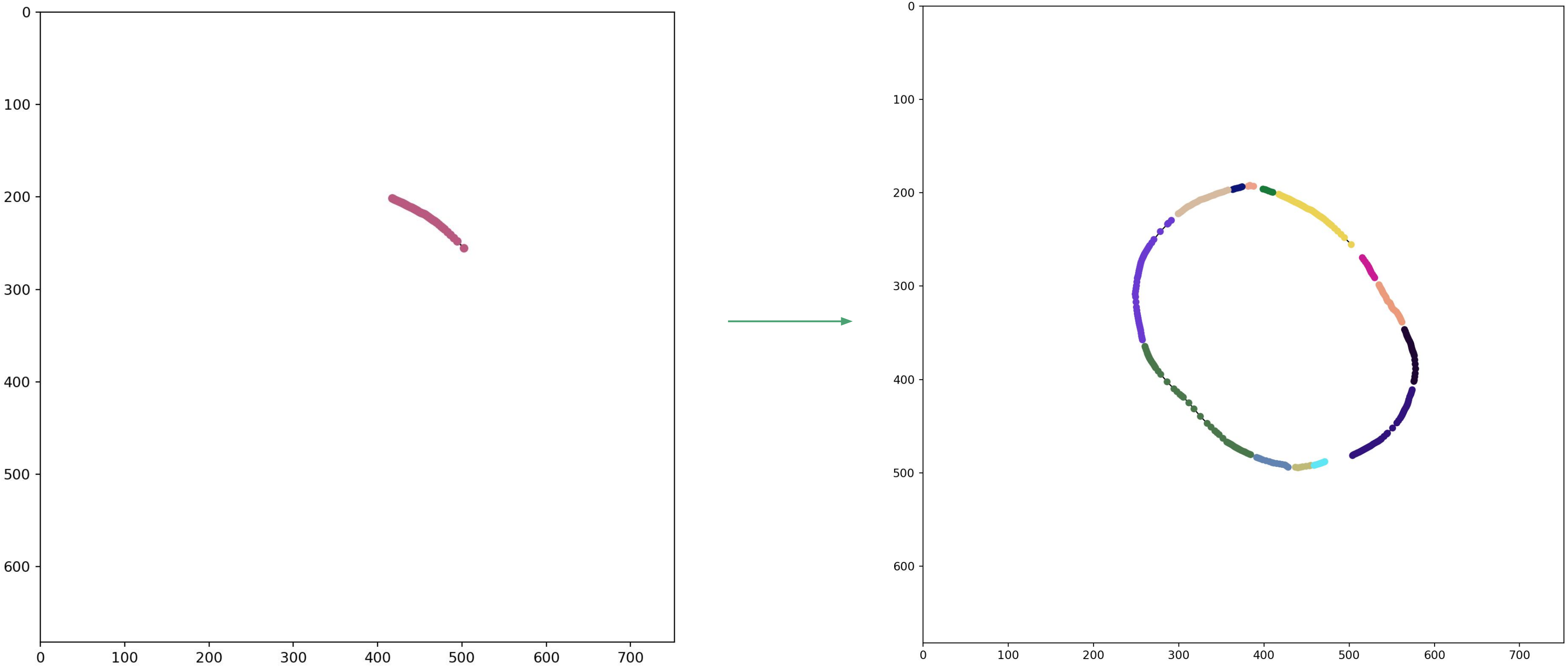
1) User chooses a starting point from the low-level graph
2) Algorithm reconstructs the object using prior information
   a) Curvature of the targeted feature
   b) Closeness between features

Processing steps



Initial step

Processing steps

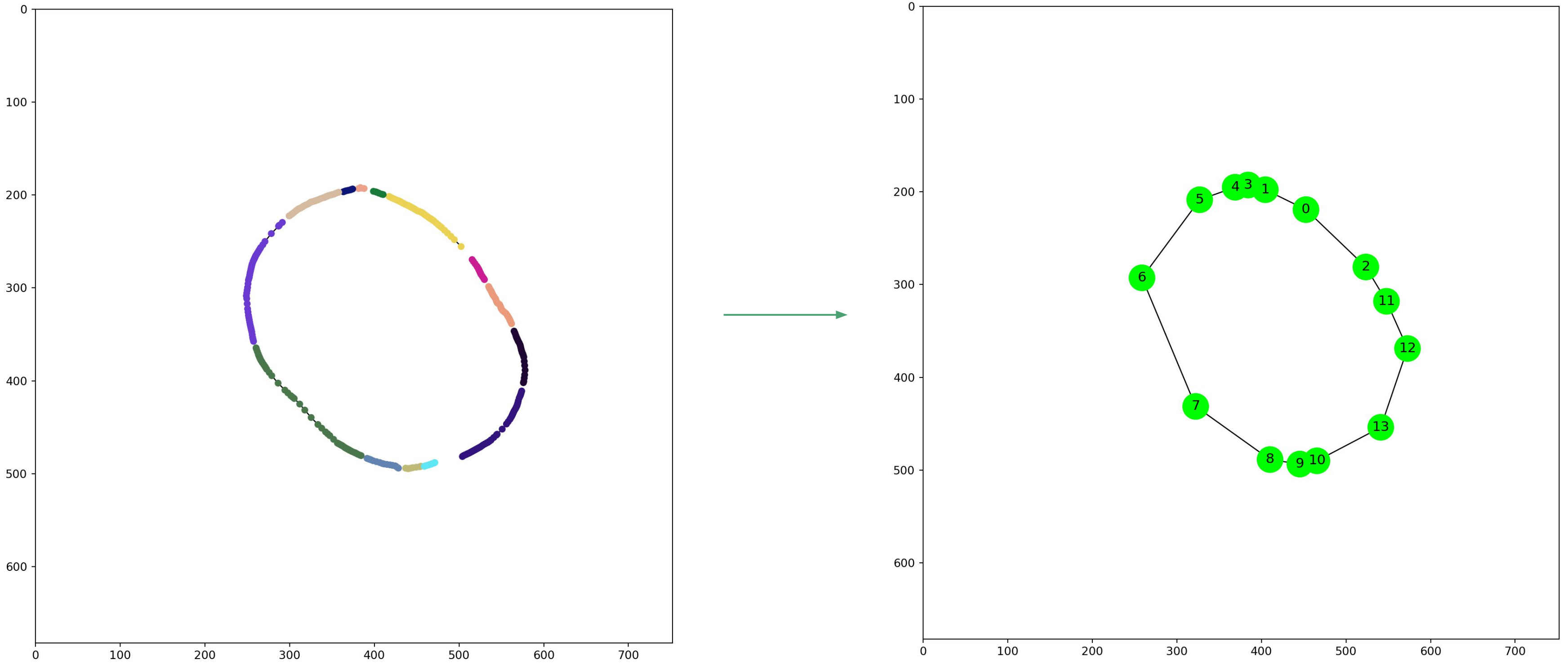


Reconstruction

High-level graph representation

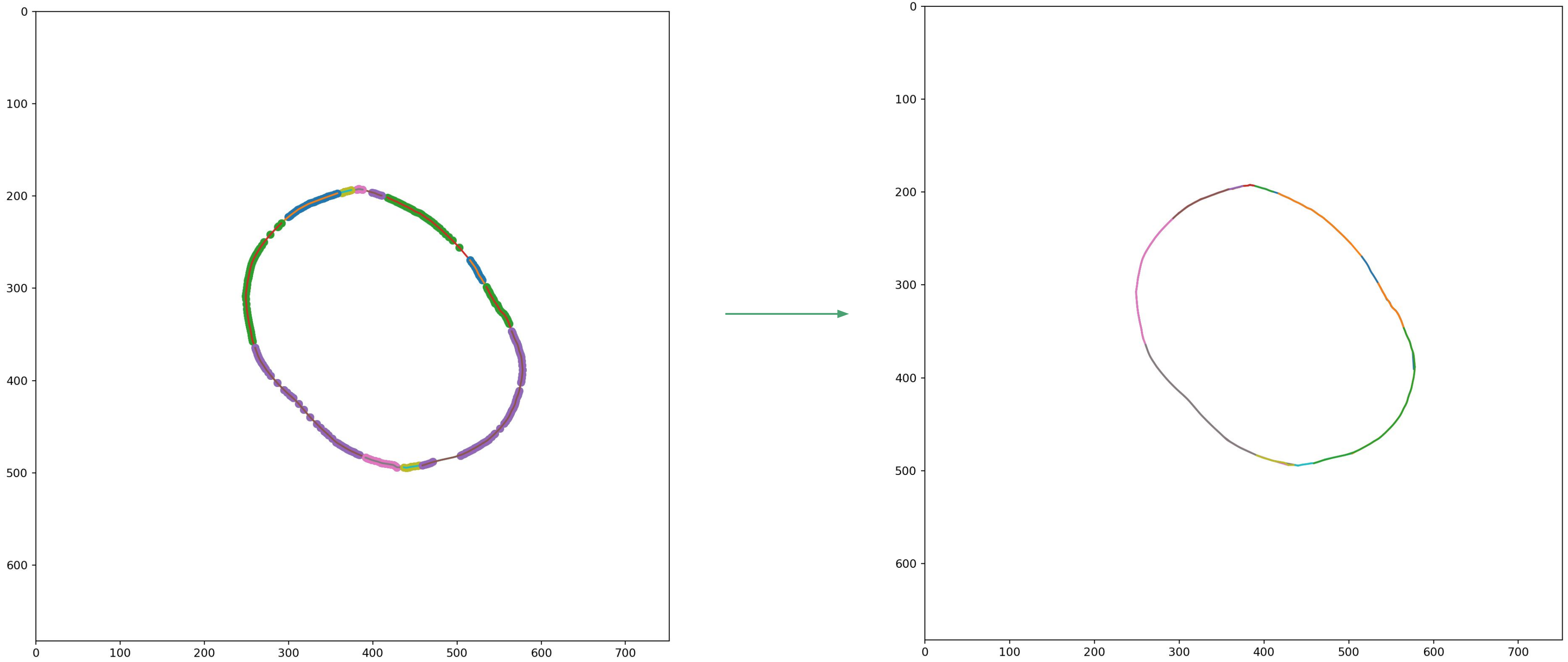
This time we represent the feature detected (outer-membrane) also as a graph. However, in this case, each node of the graph is a curve and nodes are connected to obtain the final approximation of the feature (mathematical interpolation).

Processing steps



High-level graph representation

Processing steps

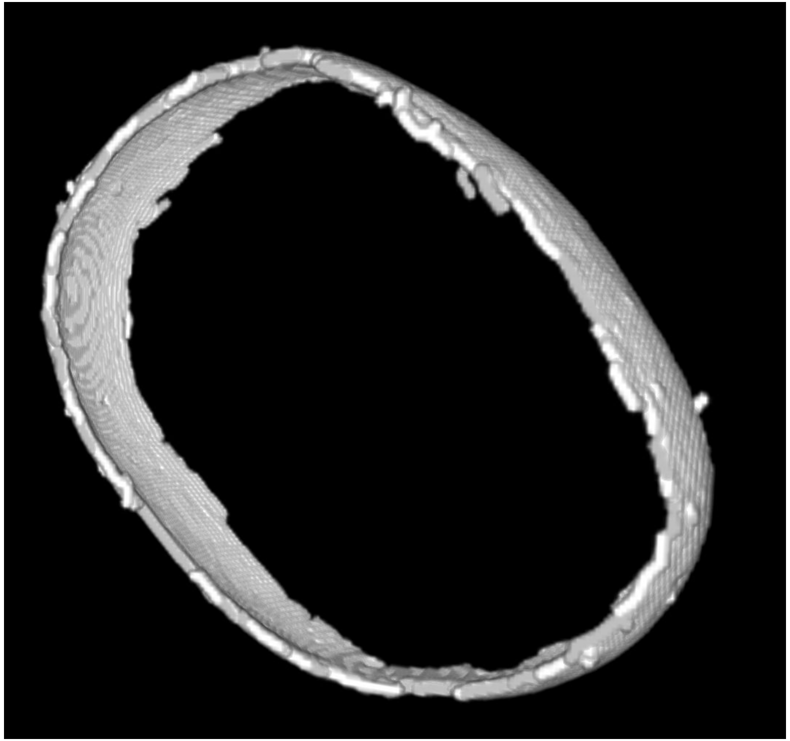


High-level graph representation

Surface reconstruction

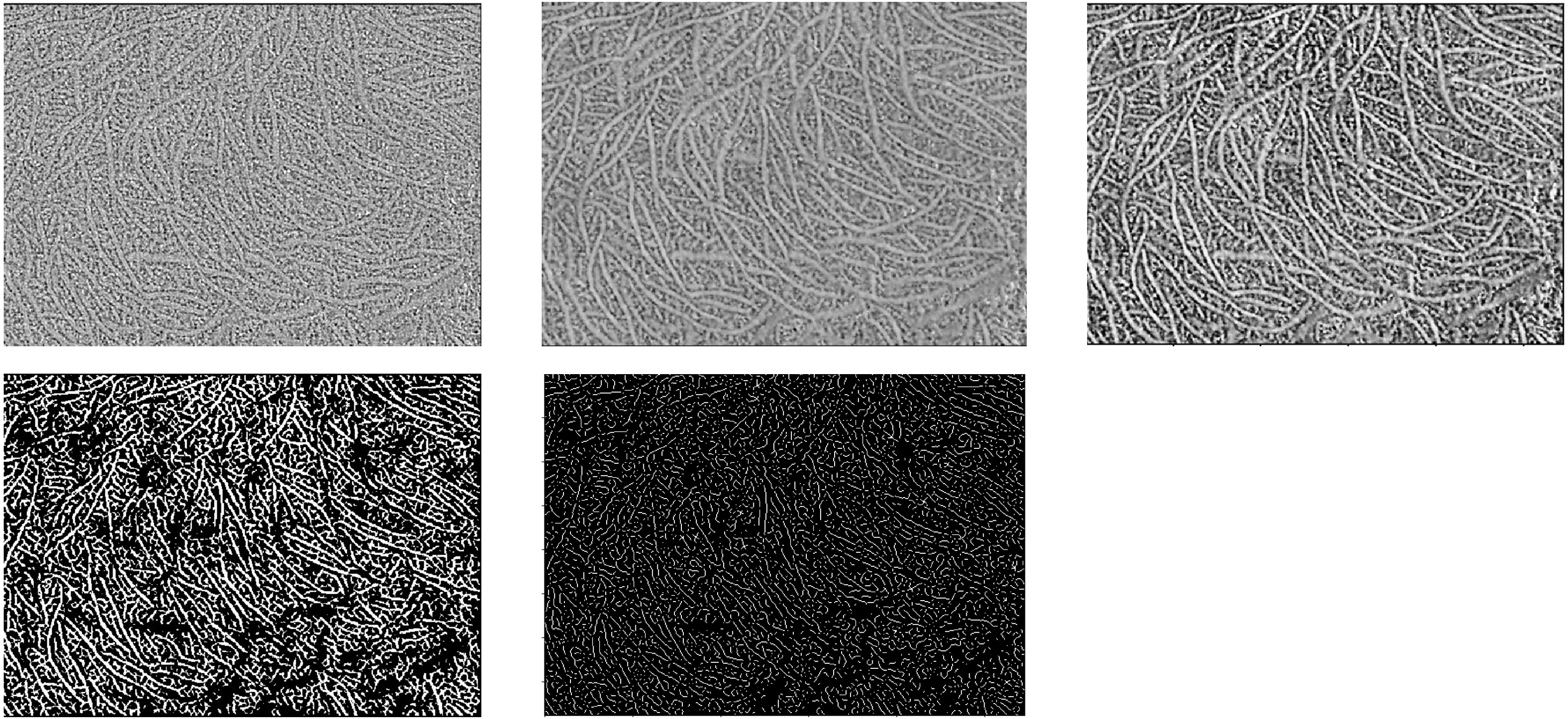
Based on the feature reconstructed in one slice, we now are able to reconstruct that same feature in 3D automatically also using prior information targeting smoothness and closeness.

Processing steps

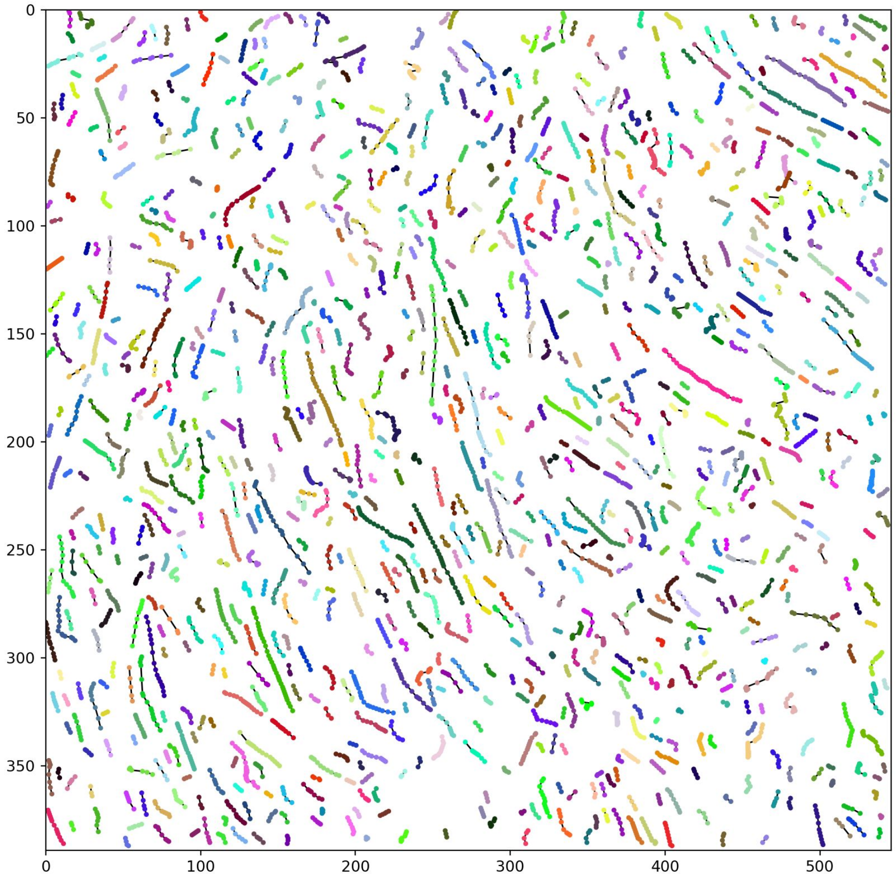


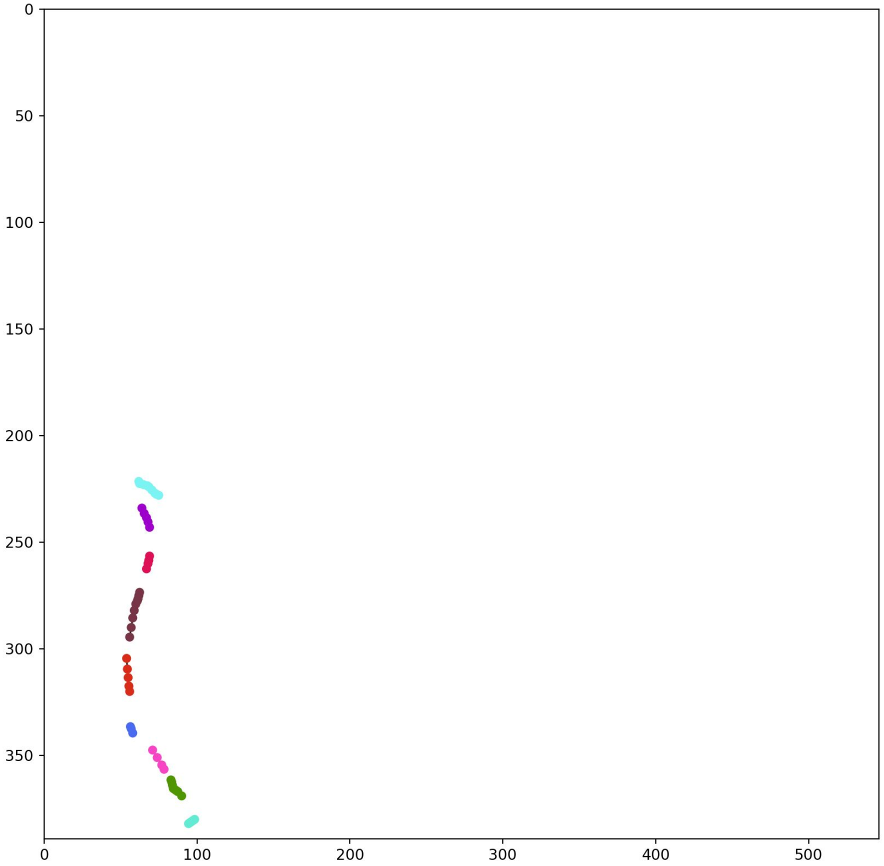
Surface reconstruction

Other applications - polyethylene

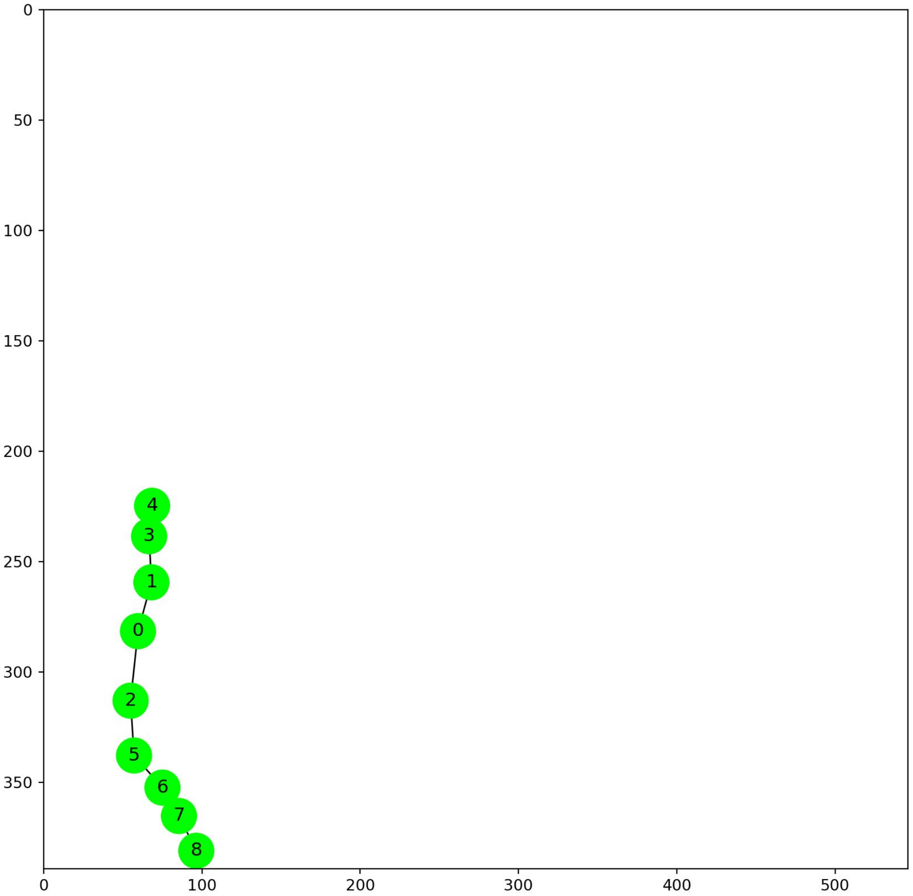


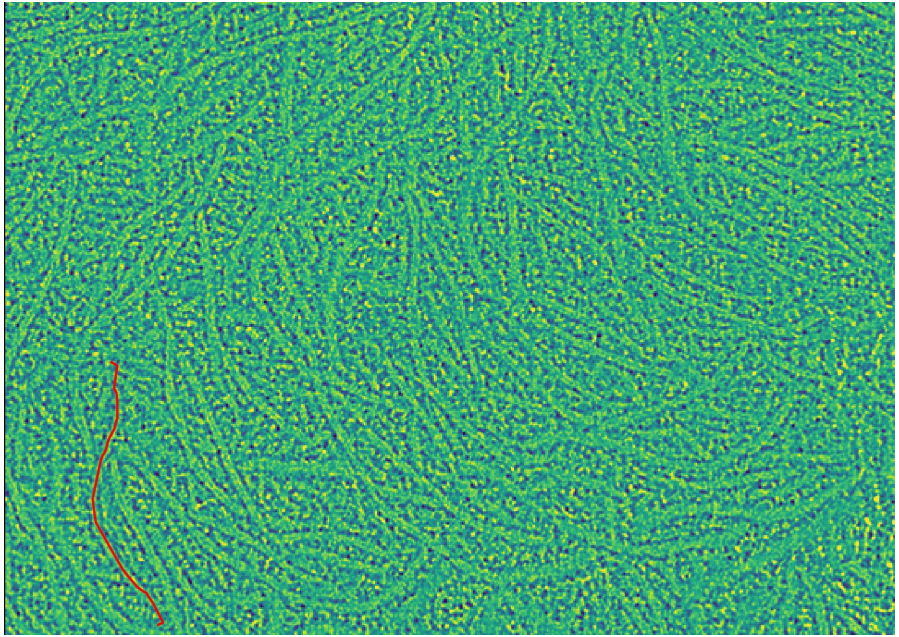
Low-level graph





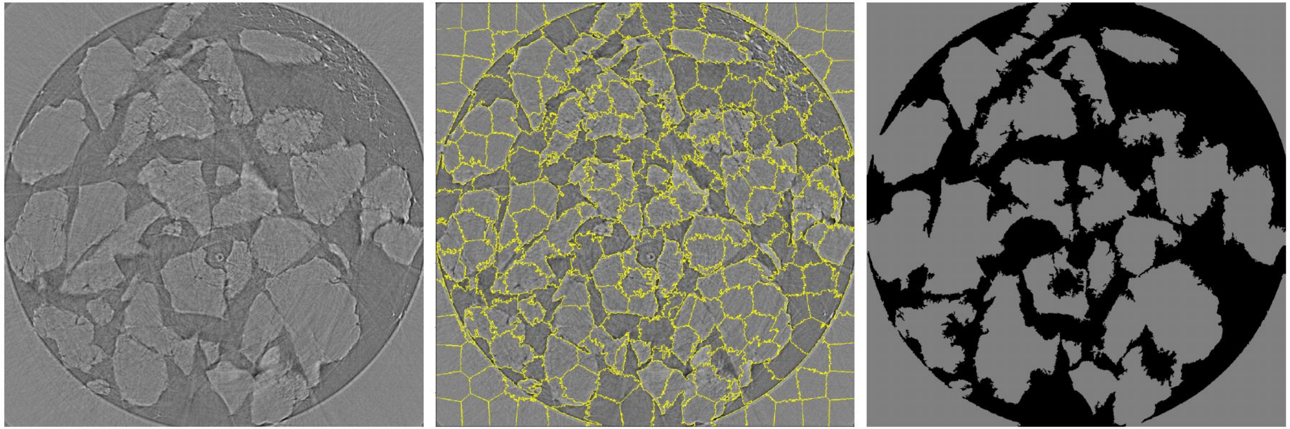
High-level graph and interpolation





Parallel Markov Random Fields

Problem: segmentation of 3D scientific images



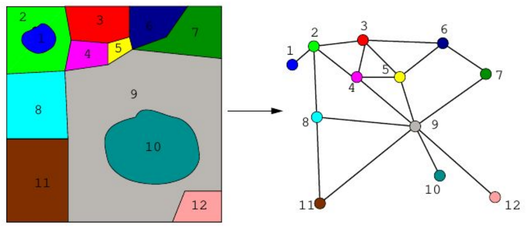
(a) Original data (b) Oversegmentation (c) PMRF segmentation

Fig. 1: Going from a raw image obtained by experiment to a segmented image suitable for quantitative analysis involves multiple processing stages.

Contributions

- Three different implementations of a Probabilistic Graphical Model optimization algorithm: C11-threads, OpenMP, and DPP
- In-depth study of shared-memory parallel performance of the three implementations
    - Analysis of hardware performance counters on multiple platforms
    - DPP implementation exhibits better runtime but less favorable scaling characteristics

The PMRF process



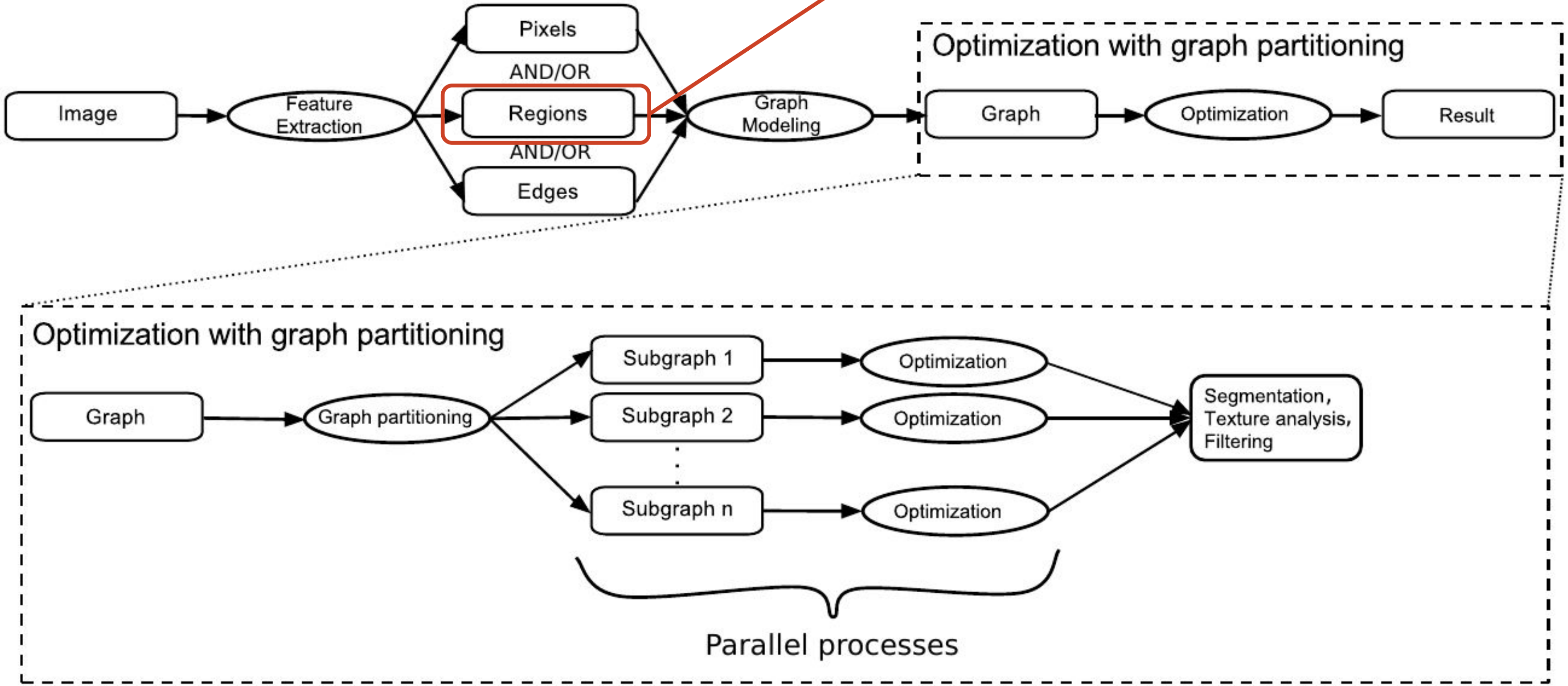


Image → Feature Extraction → Pixels AND/OR Regions AND/OR Edges → Graph Modeling

Optimization with graph partitioning

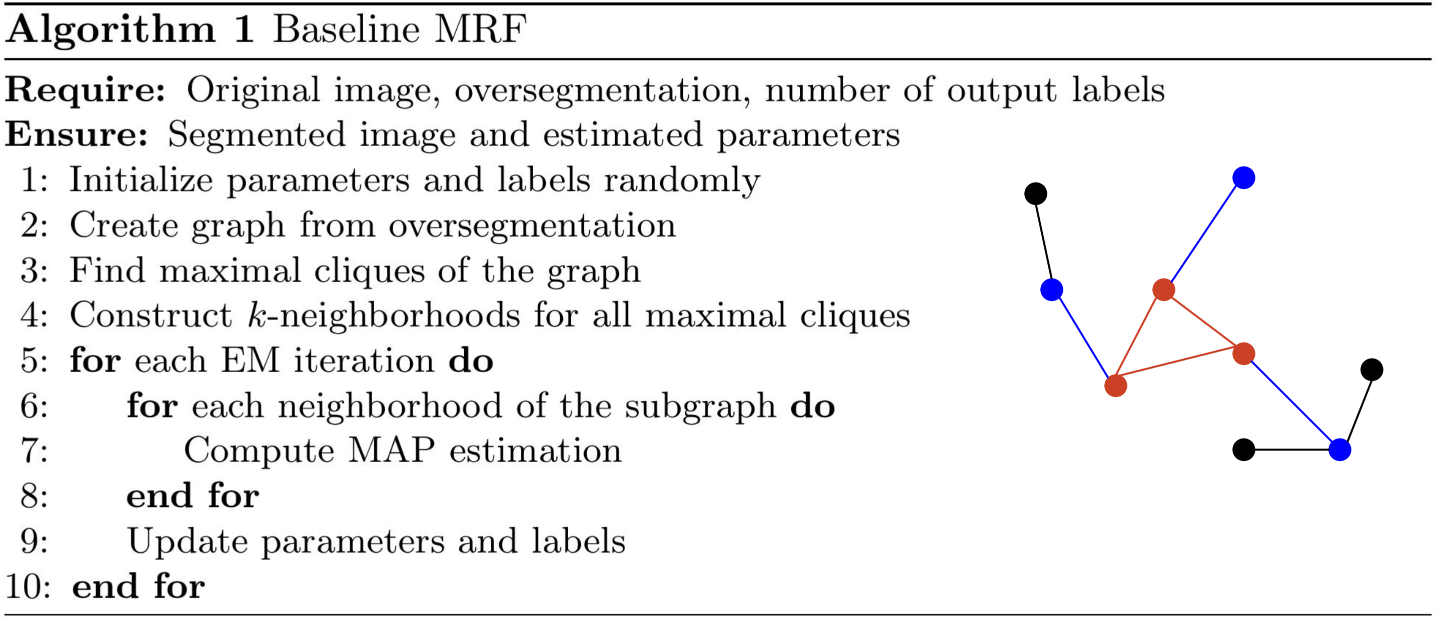
Graph → Optimization → Result

Optimization with graph partitioning

Graph → Graph partitioning → Subgraph 1, Subgraph 2, ..., Subgraph n → Optimization → Segmentation, Texture analysis, Filtering

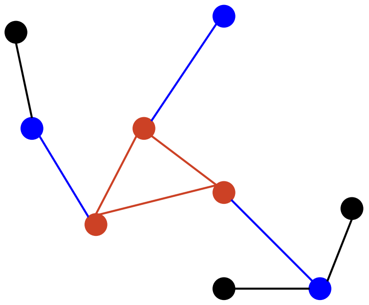
Parallel processes

Baseline MRF

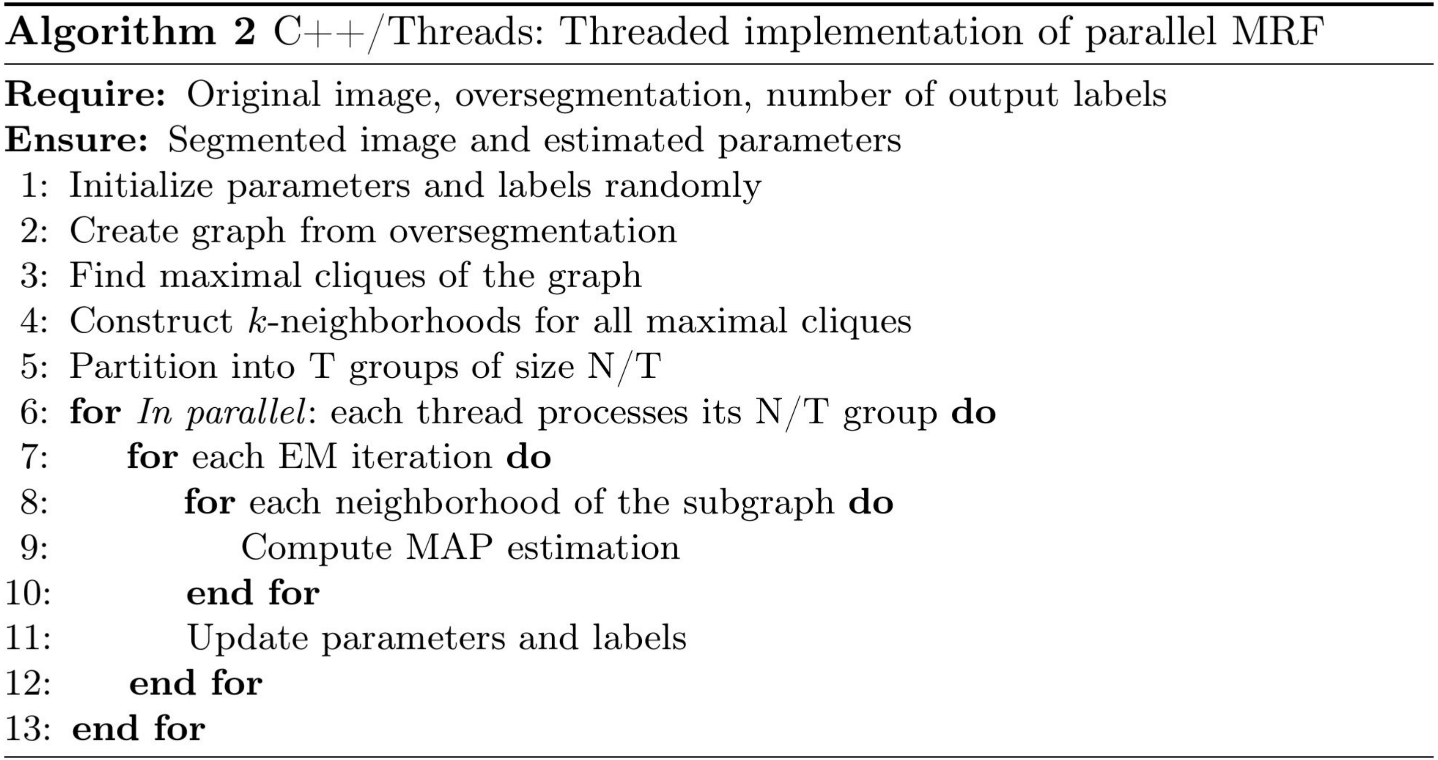


Algorithm 1 Baseline MRF

```
Require: Original image, oversegmentation, number of output labels
Ensure: Segmented image and estimated parameters
Initialize parameters and labels randomly
Create graph from oversegmentation
Find maximal cliques of the graph
Construct $k$-neighborhoods for all maximal cliques
for each EM iteration do
    for each neighborhood of the subgraph do
        Compute MAP estimation
    end for
    Update parameters and labels
end for
```



C++/Threads PMRF



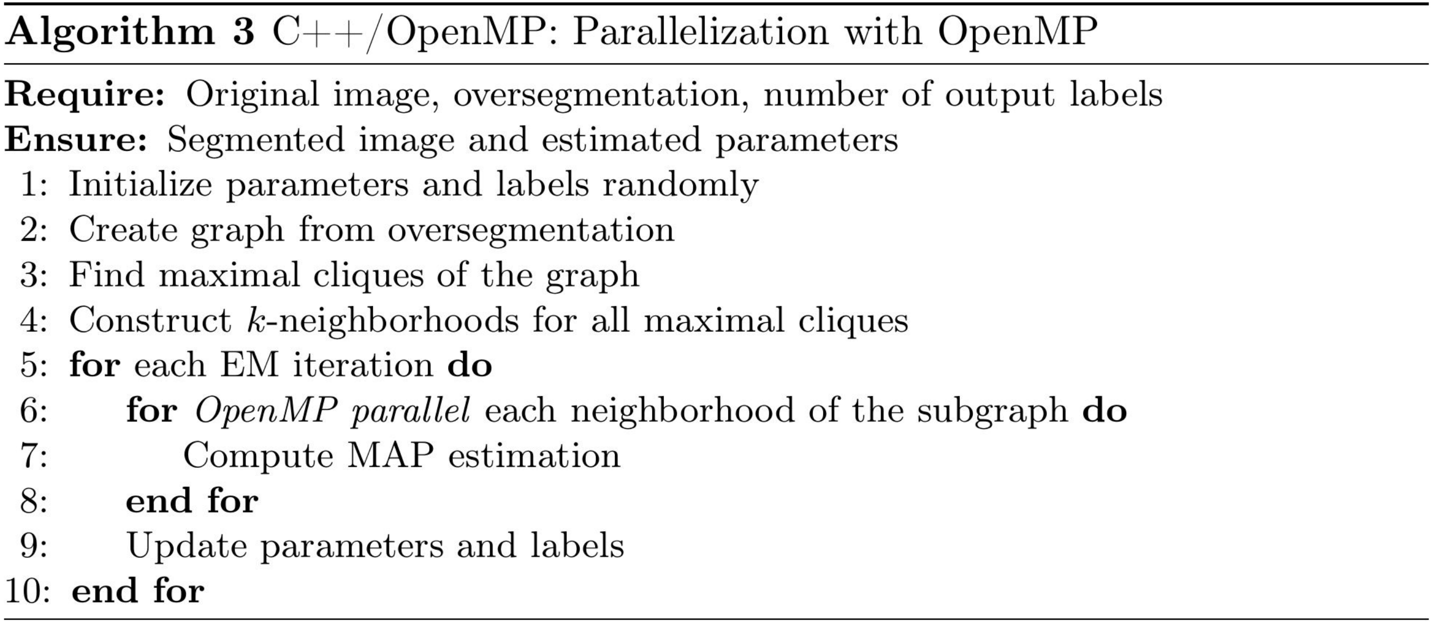
Algorithm 2 C++/Threads: Threaded implementation of parallel MRF

Require: Original image, oversegmentation, number of output labels

Ensure: Segmented image and estimated parameters

```
Initialize parameters and labels randomly
Create graph from oversegmentation
Find maximal cliques of the graph
Construct $k$-neighborhoods for all maximal cliques
Partition into T groups of size N/T
for In parallel: each thread processes its N/T group do
    for each EM iteration do
        for each neighborhood of the subgraph do
            Compute MAP estimation
        end for
        Update parameters and labels
    end for
end for
```

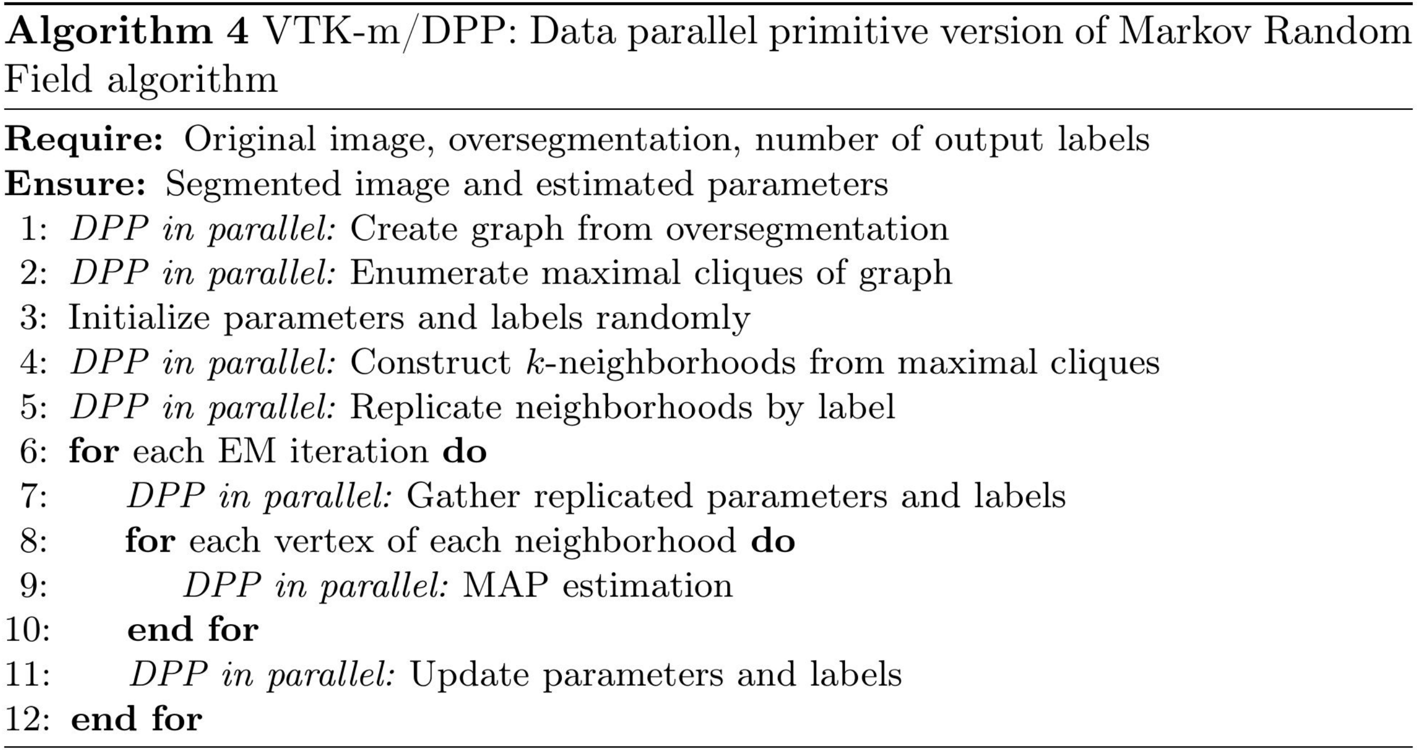
C++/OpenMP PMRF



Algorithm 3 C++/OpenMP: Parallelization with OpenMP

```
Require: Original image, oversegmentation, number of output labels
Ensure: Segmented image and estimated parameters
Initialize parameters and labels randomly
Create graph from oversegmentation
Find maximal cliques of the graph
Construct $k$-neighborhoods for all maximal cliques
for each EM iteration do
    for OpenMP parallel each neighborhood of the subgraph do
        Compute MAP estimation
    end for
    Update parameters and labels
end for
```

VTK-m/DPP PMRF



Algorithm 4 VTK-m/DPP: Data parallel primitive version of Markov Random Field algorithm

```
Require: Original image, oversegmentation, number of output labels
Ensure: Segmented image and estimated parameters
DPP in parallel: Create graph from oversegmentation
DPP in parallel: Enumerate maximal cliques of graph
Initialize parameters and labels randomly
DPP in parallel: Construct $k$-neighborhoods from maximal cliques
DPP in parallel: Replicate neighborhoods by label
for each EM iteration do
    DPP in parallel: Gather replicated parameters and labels
    for each vertex of each neighborhood do
        DPP in parallel: MAP estimation
    end for
    DPP in parallel: Update parameters and labels
end for
```

Experiment and Results

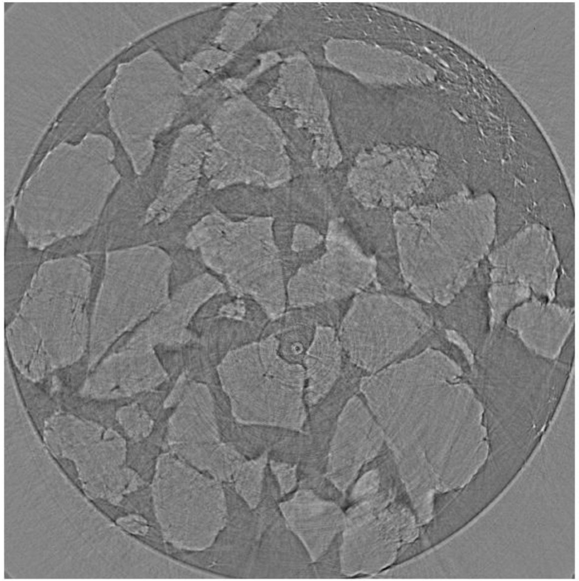
We aim to answer two primary questions:

1. How well the different implementations perform on a single-socket study
   a. What are the key performance characteristics for each version?
2. Collect hardware performance counters to understand how well each implementation vectorizes and makes use of the memory hierarchy
   a. What are the factors that lead to these performance characteristics?

Experiment and Results

Datasets: experimental dataset generated at the ALS beamline 8.3.2 containing cross-sections of a geological sample

1. Sandstone2K: 2580 x 2610 x 500
2. Sandstone5K: 5160 x 5220 x 500



Performance and Scalability

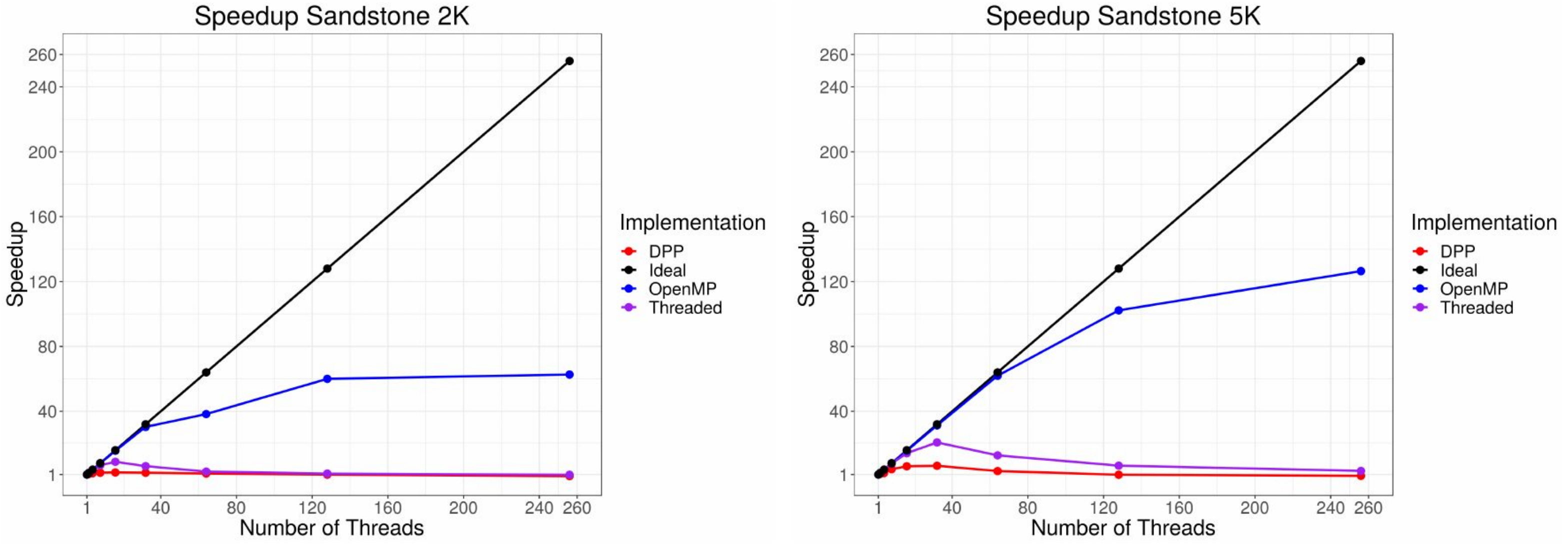


Fig. 2: Speedup of the Sandstone2K and Sandstone5K datasets on Cori. The horizontal axis is the concurrency level and the vertical axis measures the speedup.

Performance and Scalability

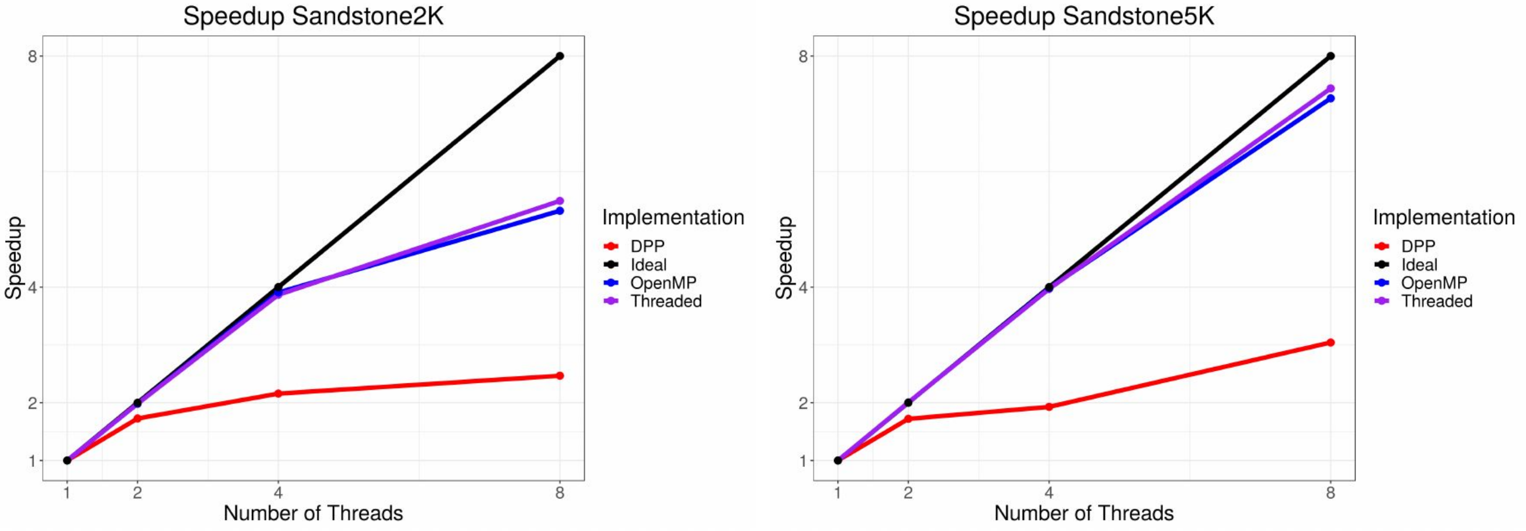


Fig. 3: Speedup of the Sandstone2K and Sandstone5K datasets on the Ivy Bridge platform. The horizontal axis is the concurrency level and the vertical axis measures the speedup.

Hardware performance counters

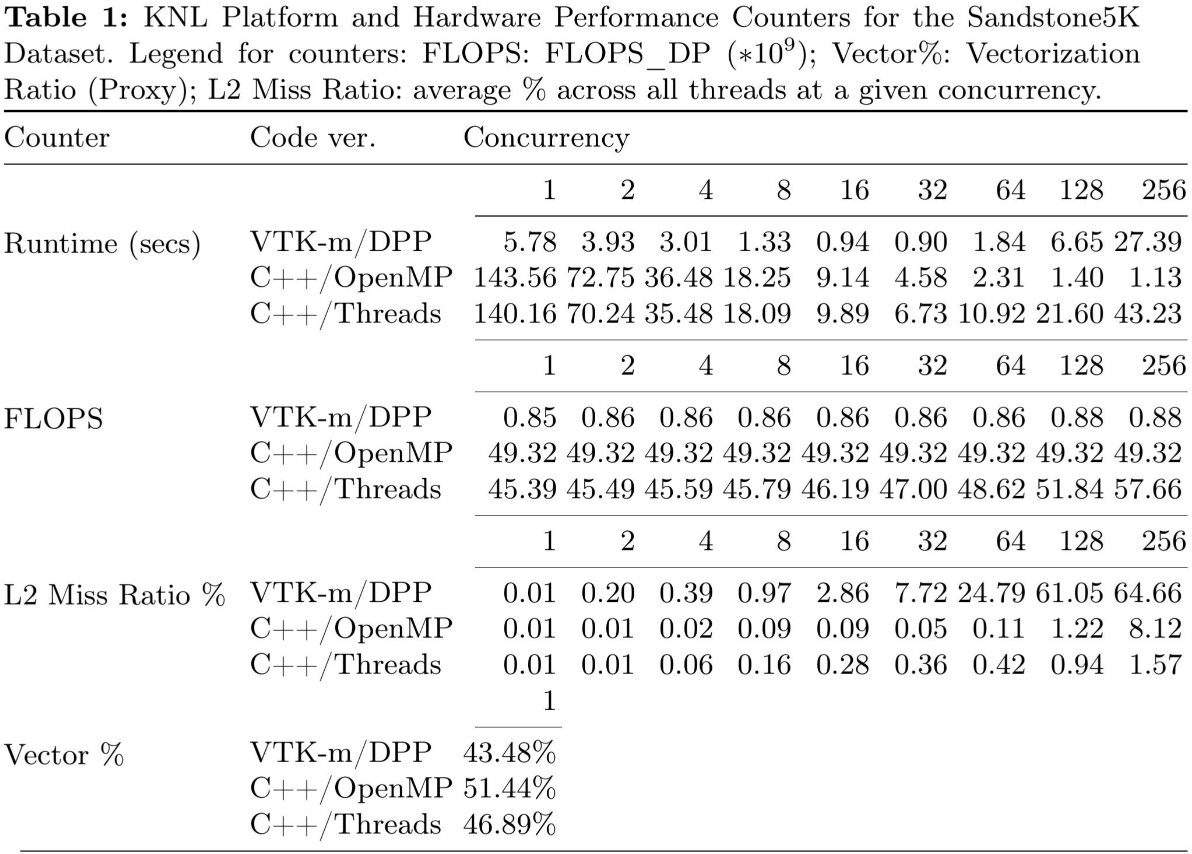


Table 1: KNL Platform and Hardware Performance Counters for the Sandstone5K Dataset. Legend for counters: FLOPS: FLOPS_DP ($*10^9$); Vector%: Vectorization Ratio (Proxy); L2 Miss Ratio: average % across all threads at a given concurrency.

Counter	Code ver.	Concurrency								
		1	2	4	8	16	32	64	128	256
Runtime (secs)	VTK-m/DPP	5.78	3.93	3.01	1.33	0.94	0.90	1.84	6.65	27.39
	C++/OpenMP	143.56	72.75	36.48	18.25	9.14	4.58	2.31	1.40	1.13
	C++/Threads	140.16	70.24	35.48	18.09	9.89	6.73	10.92	21.60	43.23
		1	2	4	8	16	32	64	128	256
FLOPS	VTK-m/DPP	0.85	0.86	0.86	0.86	0.86	0.86	0.86	0.88	0.88
	C++/OpenMP	49.32	49.32	49.32	49.32	49.32	49.32	49.32	49.32	49.32
	C++/Threads	45.39	45.49	45.59	45.79	46.19	47.00	48.62	51.84	57.66
		1	2	4	8	16	32	64	128	256
L2 Miss Ratio %	VTK-m/DPP	0.01	0.20	0.39	0.97	2.86	7.72	24.79	61.05	64.66
	C++/OpenMP	0.01	0.01	0.02	0.09	0.09	0.05	0.11	1.22	8.12
	C++/Threads	0.01	0.01	0.06	0.16	0.28	0.36	0.42	0.94	1.57
		1								
Vector %	VTK-m/DPP	43.48%								
	C++/OpenMP	51.44%								
	C++/Threads	46.89%								

Hardware performance counters

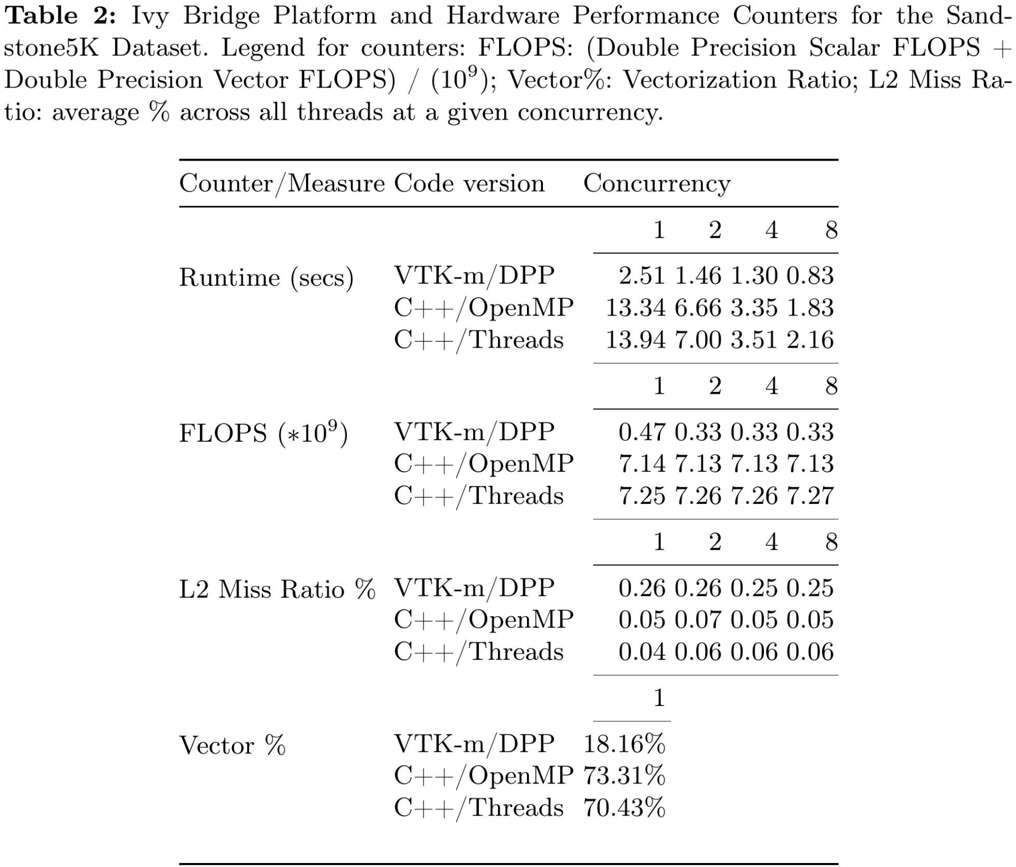


Table 2: Ivy Bridge Platform and Hardware Performance Counters for the Sandstone5K Dataset. Legend for counters: FLOPS: (Double Precision Scalar FLOPS + Double Precision Vector FLOPS) / (10^9); Vector%: Vectorization Ratio; L2 Miss Ratio: average % across all threads at a given concurrency.

Counter/Measure	Code version	Concurrency			
		1	2	4	8
Runtime (secs)	VTK-m/DPP	2.51	1.46	1.30	0.83
	C++/OpenMP	13.34	6.66	3.35	1.83
	C++/Threads	13.94	7.00	3.51	2.16
		1	2	4	8
FLOPS ($*10^9$)	VTK-m/DPP	0.47	0.33	0.33	0.33
	C++/OpenMP	7.14	7.13	7.13	7.13
	C++/Threads	7.25	7.26	7.26	7.27
		1	2	4	8
L2 Miss Ratio %	VTK-m/DPP	0.26	0.26	0.25	0.25
	C++/OpenMP	0.05	0.07	0.05	0.05
	C++/Threads	0.04	0.06	0.06	0.06
		1			
Vector %	VTK-m/DPP	18.16%			
	C++/OpenMP	73.31%			
	C++/Threads	70.43%			

Key findings

1. The VTK-m/DPP code is executing far fewer floating point instructions
2. Vectorization ratios
    a. KNL: comparable vectorization ratios (43% - 51%)
    b. Ivy Bridge: 70% for the C++/OpenMP and C++/Threads; 18% for the VTK-m/DPP implementation
        i. Differences in the code itself
        ii. Variation in how the compiler auto-vectorizes
3. Scalability
    a. VTK-m/DPP (KNL): decreasing runtime up to 32 cores, along with increase in the L2 Cache Miss ratio
    b. C++/Threads (KNL): decreasing runtime up to 32 cores, after which point the runtime increases significantly -> C++/OpenMP presents better results most likely because of the highly optimized OpenMP loop parallelization
    c. On the Ivy Bridge platform all implementations exhibit better scalability: large L3 cache that is shared across all cores

Conclusion and Future Work

- Understand the performance characteristics of three different approaches for doing shared-memory parallelization of a PGM optimization code
- Improve throughput of scientific analysis tools in light of increasing sensor and detector resolution
- We expected that the VTK-m/DPP implementation was running faster because of better vectorization… not true! It executes many fewer instructions
- This study is timely, shedding light on the performance characteristics of a non-trivial, data-intensive code implemented with three different methodologies
- Future: pressing deeper into the topic of platform portability: OpenMP version to emit GPU code

Thanks!

tperciano@lbl.gov