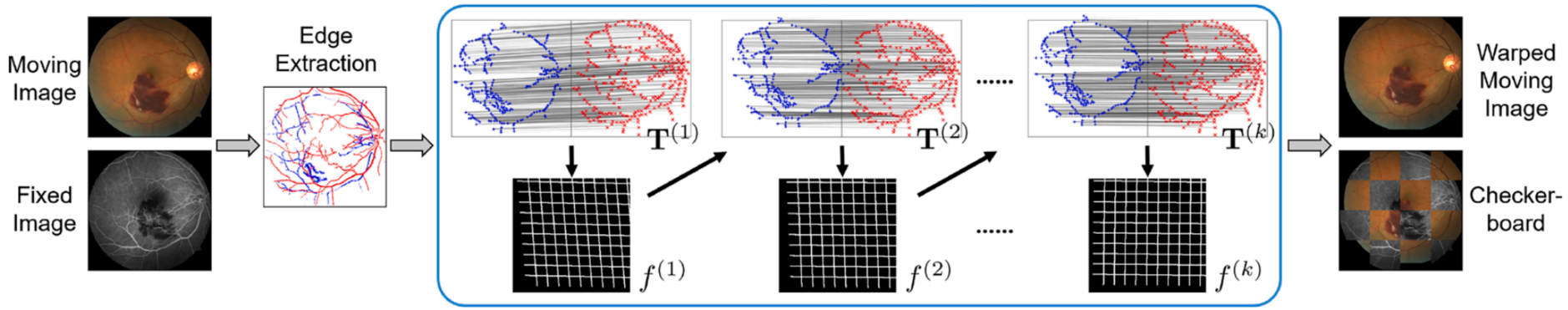
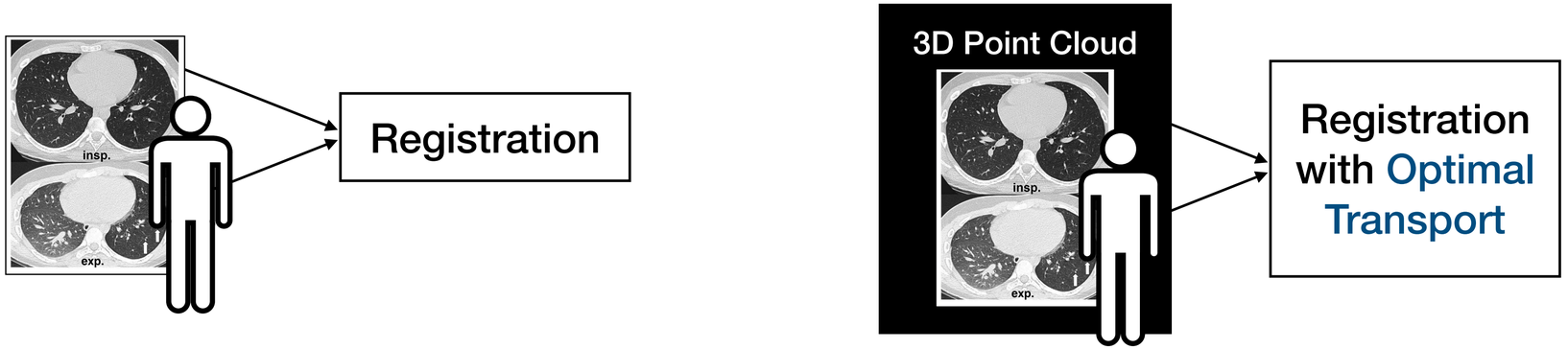
3D Intra-patient Point Cloud Registration with Iterative-Optimal Transport



Athena Mo | 03/20 Short Update

3D Intra-patient Point Cloud Registration with Iterative-Optimal Transport



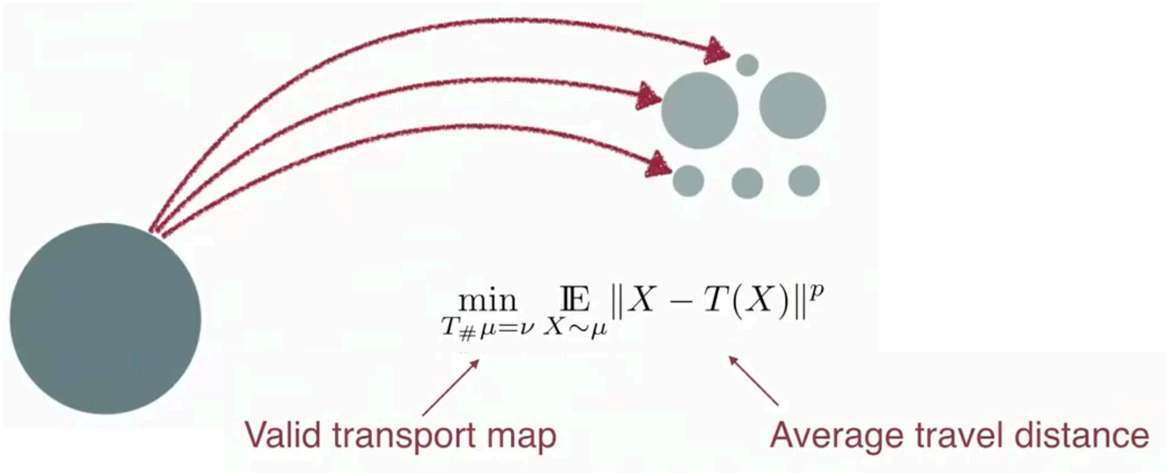
Why does this matter?

Radiation therapy planning, Longitudinal disease tracking, 4D CT motion modelling, …

Why was the problem challenging?

→ large non-rigid deformations between EXP and INSP

→ intensity-based registration (e.g. ANTs) can lead to mismatch



Why Optimal Transport (OT)?

- Standard registration approaches…
  - point by point → computationally costly
  - hard one-to-one correspondences → less accurate when structures appear/disappear across phases
- Geometric-aware registration
  - minimizing the total work (squared Euclidean distance in 3D space)
    - Less computational cost
    - Better anatomical structure

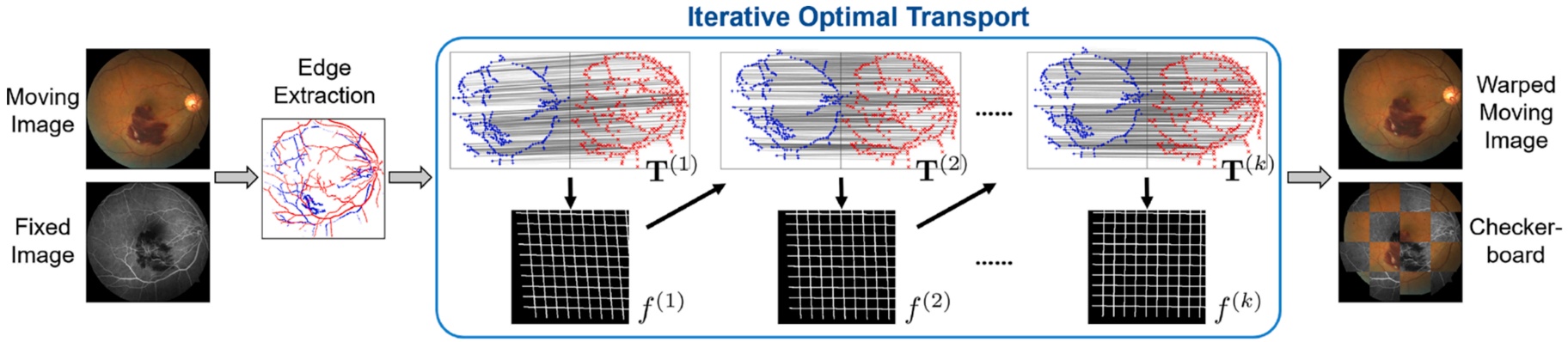
(Center for Intelligent Systems CIS EPFL, 2022)

Iterative optimal transport for multimodal image registration ★

Mengyu Li [a], Cheng Meng [b,*], Xiaodan Fan [c]

[a] *Department of Statistics and Data Science, Tsinghua University, Beijing, China*
[b] *Center for Applied Statistics, Institute of Statistics and Big Data, Renmin University of China, Beijing, China*
[c] *Department of Statistics and Data Science, The Chinese University of Hong Kong, Hong Kong, China*

(2025)



What is Iterative Optimal Transport (IOT)?

- IOT alternates between
  - "Given where the points currently are, what's the best soft matching?"
  - "Given that soft matching, what's the best polynomial warp?"
  - Repeat until neither the matching nor the warp changes

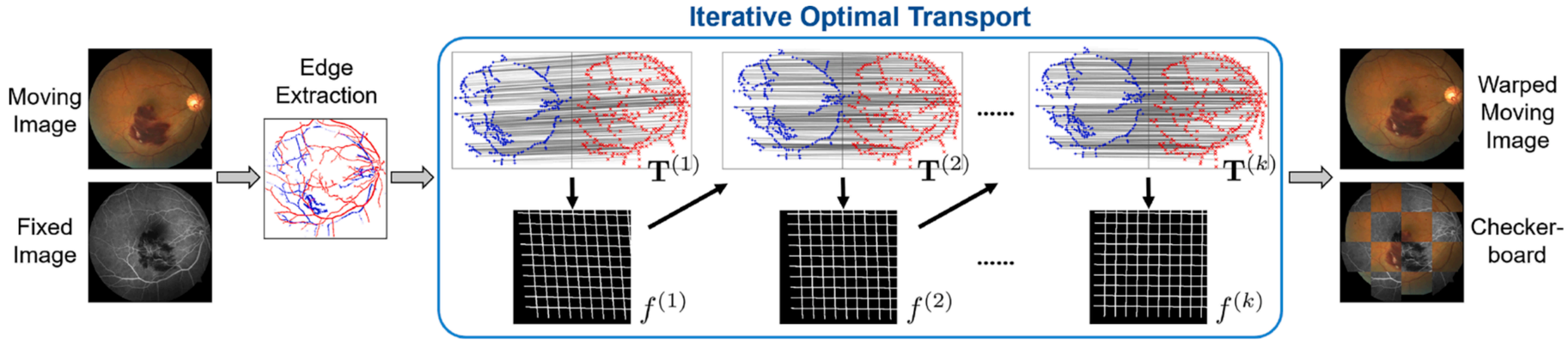
$$f(X) = \Phi(X) \mathbin{@} T$$

- **Input:** two sets of structural points
  - moving (EXP) and fixed (INSP)
  - sampled from edge/vessel structures
- **Output**: a polynomial transformation f that warps the moving points to align with the fixed points

Iterative Optimal Transport

** $f(X) = \Phi(X) @ T$

f, polynomial transformation function
T, the N×M transport plan matrix



Degree 1 — *only affine transforms*
Degree 2 — *can model smooth, gradual bending*
Degree 3 — *can model more complex local deformations*

- Initialize with warped EXP = EXP, no movement
- For each λ:
  - Update T by solving unbalanced OT ($\lambda \rightarrow$ inf = OT), f fixed at f_{k-1}
  - Update f by solving **, T fixed at T_k
- Repeat until $\|T_k - T_{k-1}\|$ and $\|f_k - f_{k-1}\| <$ `stop_thr`

Why 3D point clouds?
- Enforces anatonomical focus
- Less computationally expensive

Algorithm 1 Iterative optimal transport.

Input: moving points $\{\boldsymbol{x}_i\}_{i=1}^n$, fixed points $\{\boldsymbol{y}_i\}_{i=1}^m$, polynomial order q, regularization parameters ε, λ
Initialize $f^{(0)}(\boldsymbol{x}_i) = \boldsymbol{x}_i$, $k = 0$
repeat
  Set $k = k + 1$.
  Update $\mathbf{T}^{(k)}$ by solving the problem

$$\min_{\mathbf{T}\in\mathbb{R}_+^{n\times m}} \sum_{i,j} \|f^{(k-1)}(\boldsymbol{x}_i) - \boldsymbol{y}_j\|^2 T_{ij} + \lambda\,\mathrm{KL}(\mathbf{T}\mathbf{1}_m \| n^{-1}\mathbf{1}_n) + \lambda\,\mathrm{KL}(\mathbf{T}^\top \mathbf{1}_n \| m^{-1}\mathbf{1}_m) \tag{5}$$

  with fixed $f^{(k-1)}$ using a maximization-minimization approach [27].
  Update $f^{(k)}$ by solving the problem

$$\min_{f\in\mathcal{H}} \sum_{i,j} \|f(\boldsymbol{x}_i) - \boldsymbol{y}_j\|^2 T_{ij}^{(k)} + \varepsilon \sum_i \|f(\boldsymbol{x}_i) - \boldsymbol{x}_i\|^2 \tag{6}$$

  with fixed $\mathbf{T}^{(k)}$ using a quasi-Newton method [28].
until Convergence
Output: $f^{(k)}$, $\mathbf{T}^{(k)}$

Degree 1 — *only affine transforms*
Degree 2 — *can model smooth, gradual bending*
Degree 3 — *can model more complex local deformations*

** f(X) = Φ(X) @ T

f, polynomial transformation function
T, the N×M transport plan matrix

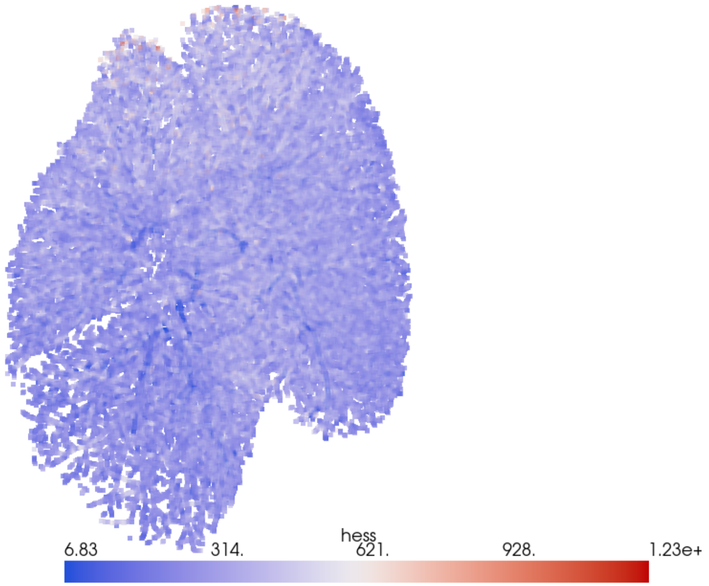
Example. $X = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 1 & 4 \\ 3 & 3 & 2 \end{bmatrix}, \qquad \mathbf{x}_1 = (1,2,3),\ \mathbf{x}_2 = (2,1,4),\ \mathbf{x}_3 = (3,3,2)$

$$\Phi_1 = \left[\begin{array}{ccc|c} x & y & z & 1 \\ \hline 1 & 2 & 3 & 1 \\ 2 & 1 & 4 & 1 \\ 3 & 3 & 2 & 1 \end{array}\right] \in R^{3\times 4}$$

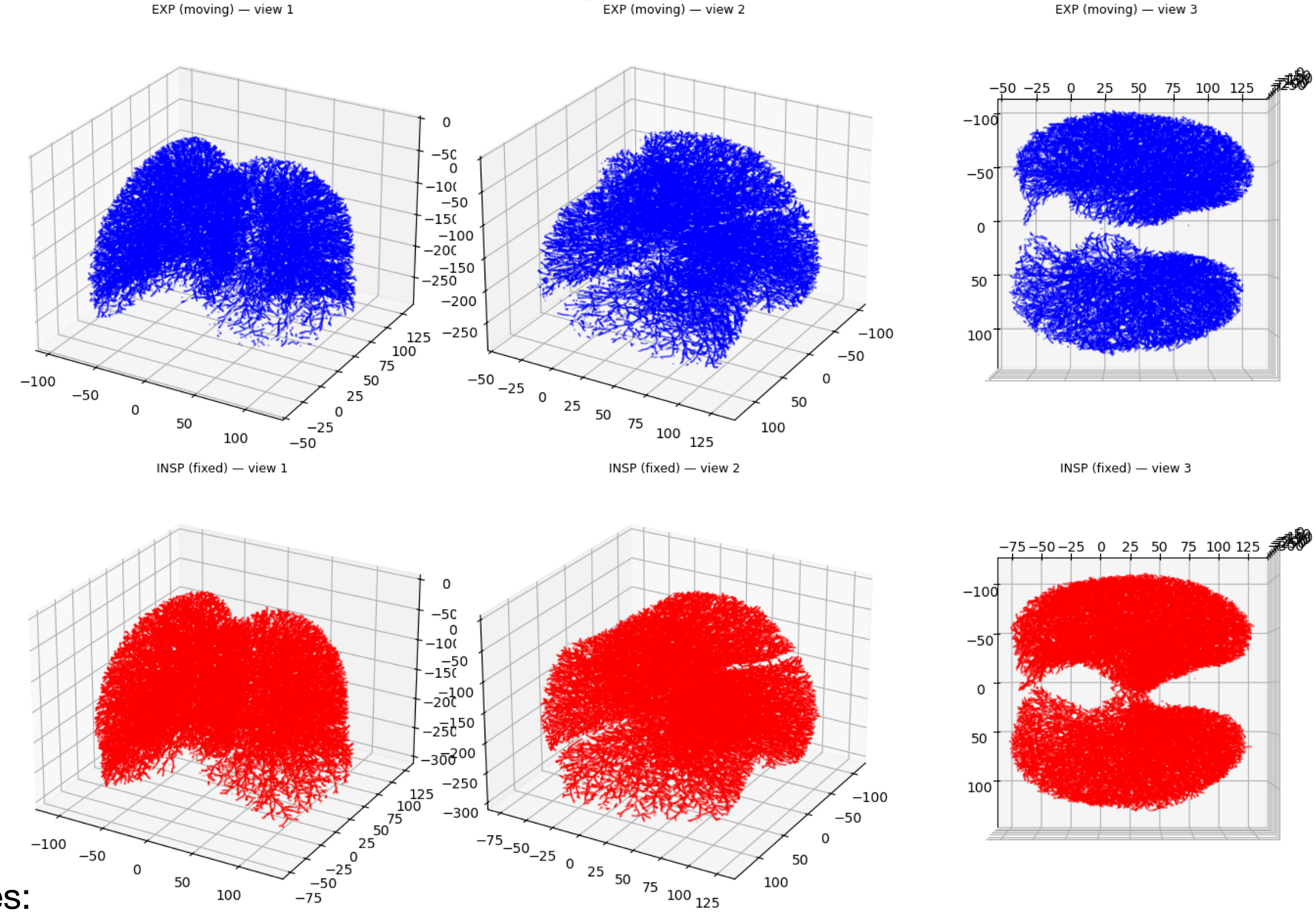
$$\Phi_2 = \left[\begin{array}{ccc|ccc|ccc|c} x & y & z & xy & xz & yz & x^2 & y^2 & z^2 & 1 \\ \hline 1 & 2 & 3 & 2 & 3 & 6 & 1 & 4 & 9 & 1 \\ 2 & 1 & 4 & 2 & 8 & 4 & 4 & 1 & 16 & 1 \\ 3 & 3 & 2 & 9 & 6 & 6 & 9 & 9 & 4 & 1 \end{array}\right] \in R^{3\times 10}$$

$$\Phi_3 = \left[\begin{array}{ccc|ccc|ccc|ccc|c} x & y & z & xy & xz & yz & x^2 & y^2 & z^2 & x^3 & y^3 & z^3 & 1 \\ \hline 1 & 2 & 3 & 2 & 3 & 6 & 1 & 4 & 9 & 1 & 8 & 27 & 1 \\ 2 & 1 & 4 & 2 & 8 & 4 & 4 & 1 & 16 & 8 & 1 & 64 & 1 \\ 3 & 3 & 2 & 9 & 6 & 6 & 9 & 9 & 4 & 27 & 27 & 8 & 1 \end{array}\right] \in R^{3\times 16}$$

Dataset - PVT1010




```
PolyData (0x1632768c0)
  N Cells:    113194
  N Points:   113194
  N Strips:   0
  X Bounds:   -9.743e+01, 1.202e+02
  Y Bounds:   -4.057e+01, 1.302e+02
  Z Bounds:   -2.750e+02, -1.587e+01
  N Arrays:   2
```



Built-in per-point quantities:

- `Hess`: Hessian matrix of the CT image intensity at each point → How tube-like is this point?
- `Raidus` (mm): filter run at multiple scales + physical size at point → How thick is tube?

Our Modifications to Paper

```
[CONFIG]
N_POINTS = 1000 [=300 @Paper]
Lambda = 1e-3, 5e-3, 2e-2, 1e-1, 5e-1
Degree = 2,3
```

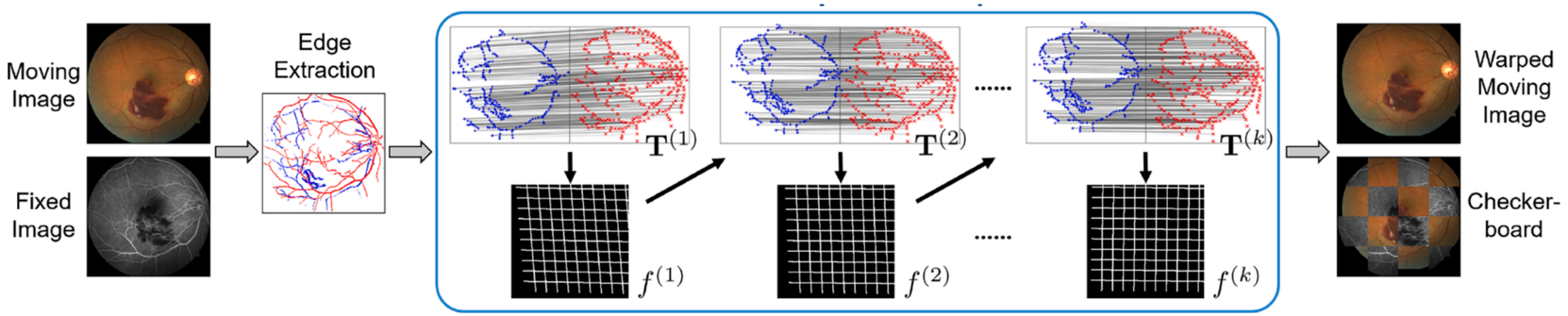
3D IOT Point Cloud Reg

↓

Load VTK Mesh via PyVista

↓

Point sampling + Normalization



Paper Method

- `edge_points()` randomly select edges
  - remove points closest to its nearest neighbor until n=300
- Normalize points

Our Method

- `foerstner_3d()` from 2D Förstner detector
  - for each point, find k-nearest neighbours using cKDTree
- Normalization → unit-variance normalization

↓

IOT

Our Method

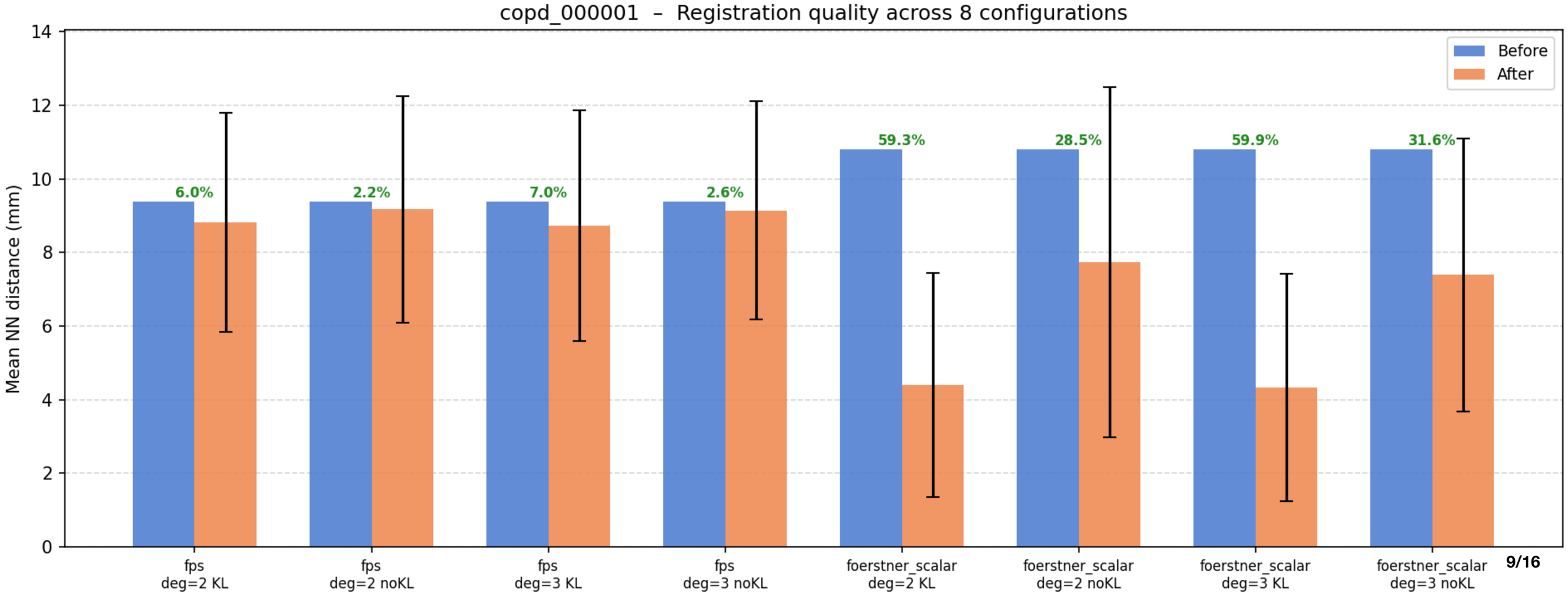
- Option to drop KL when finding T

Results

```
[CONFIG]
N_POINTS = 1000 [=300 @Paper]
Lambda = 1e-3, 5e-3, 2e-2, 1e-1, 5e-1
Degree = 2,3
```

8 Configurations for

- **Sampler:** farthest point OR `foerstner_3d`
- **Degree:** 2 OR 3
- **KL:** with OR without



Results

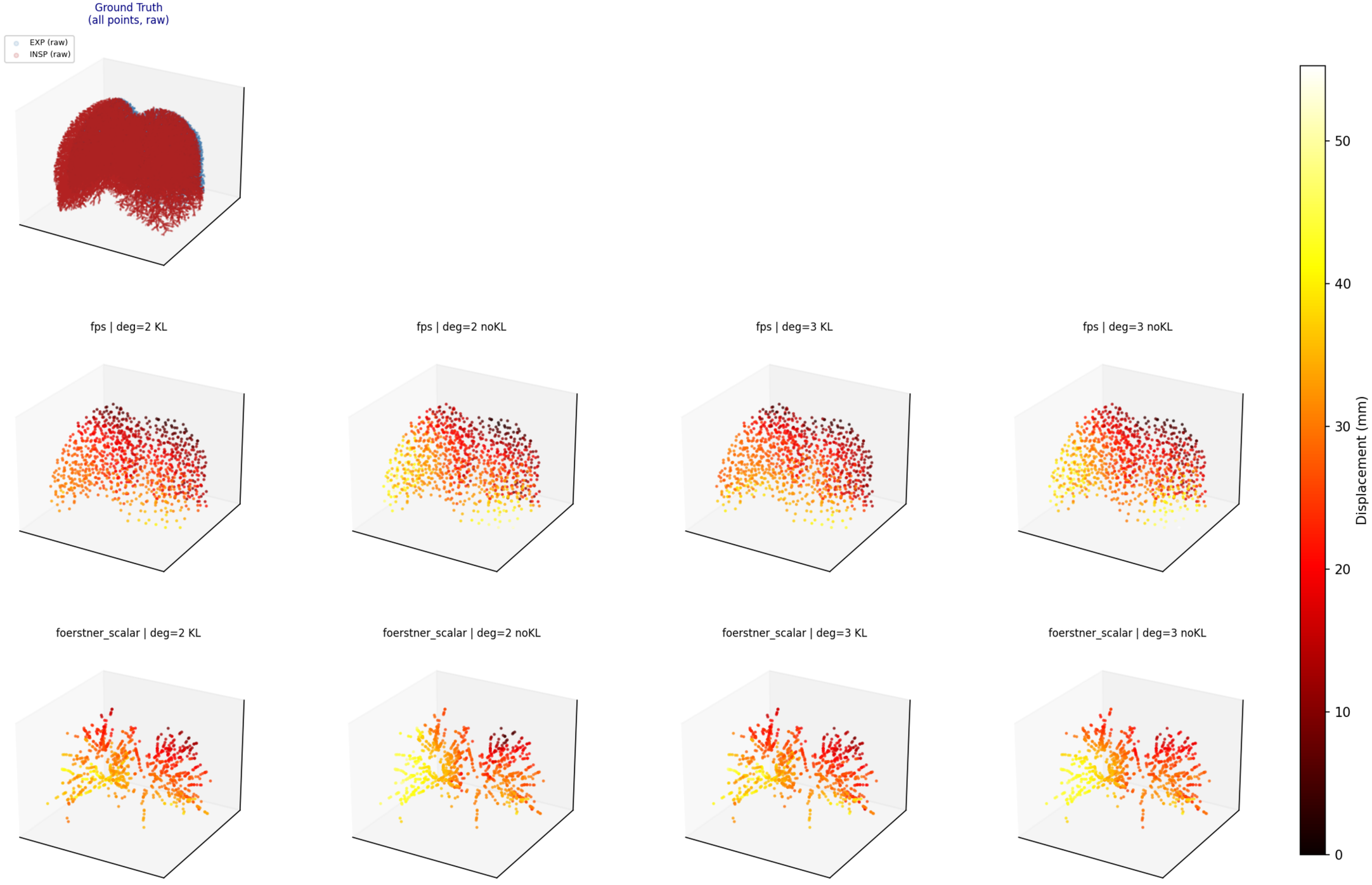
```
[CONFIG]
N_POINTS = 1000 [=300 @Paper]
Lambda = 1e-3, 5e-3, 2e-2, 1e-1, 5e-1
Degree = 2,3
```

Sampler	Deg	KL	reg_m	Pre-mean	Post-mean ± std	Improv.%
Farthest-Point Sampling (FPS)						
FPS	2	✓	0.02	9.382	8.823 ± 2.973	6.0%
FPS	2	×	N/A	9.382	9.175 ± 3.082	2.2%
FPS	3	✓	0.02	9.382	8.729 ± 3.128	7.0%
FPS	3	×	N/A	9.382	9.140 ± 2.966	2.6%
Förstner (radius-weighted)						
Förstner	2	✓	0.005	10.812	4.402 ± 3.050	59.3%
Förstner	2	×	N/A	10.812	7.733 ± 4.759	28.5%
Förstner	3	✓	0.02	10.812	**4.335 ± 3.099**	**59.9%**
Förstner	3	×	N/A	10.812	7.391 ± 3.710	31.6%

FPS: 2–7% over baseline
(regardless of degree or KL setting)

Förstner with UOT-KL: ~60% improvement

- No-KL OT is faster, but worse in post-mean
- Large standard deviation → (future direction) motion-weighted sampling


Ground Truth
(all points, raw)
EXP (raw)
INSP (raw)
fps | deg=2 KL
fps | deg=2 noKL
fps | deg=3 KL
fps | deg=3 noKL
foerstner_scalar | deg=2 KL
foerstner_scalar | deg=2 noKL
foerstner_scalar | deg=3 KL
foerstner_scalar | deg=3 noKL
Displacement (mm)
0
10
20
30
40
50

Results

```
[CONFIG]
N_POINTS = 1000 [=300 @Paper]
Lambda = 1e-3, 5e-3, 2e-2, 1e-1, 5e-1
Degree = 2,3
```

Comparison	t-statistic	p-value	Sig.
(1) Sampler: FPS vs. Förstner (radius-weighted)			
deg= 2, KL	32.356	1.11×10^{-157}	***
deg= 2, noKL	7.975	4.15×10^{-15}	***
deg= 3, KL	30.141	1.81×10^{-142}	***
deg= 3, noKL	11.539	5.29×10^{-29}	***
(2) Transport: UOT-KL vs. Balanced-OT (noKL)			
FPS, deg= 2	−3.277	1.09×10^{-3}	**
FPS, deg= 3	−3.701	2.27×10^{-4}	***
Förstner, deg= 2	−20.950	4.52×10^{-81}	***
Förstner, deg= 3	−25.086	4.28×10^{-108}	***
(3) Polynomial degree: deg= 2 vs. deg= 3			
FPS, KL	0.934	3.50×10^{-1}	ns
FPS, noKL	0.444	6.57×10^{-1}	ns
Förstner, KL	1.261	2.08×10^{-1}	ns
Förstner, noKL	3.628	3.00×10^{-4}	***

***p < 0.001; **p < 0.01; *p < 0.05; ns = not significant.

- Förstner significantly outperforms FPS in all four matched comparisons (all p < 0.001, ***)
- Negative → KL consistently produces lower NN distances
  - effect is stronger for Förstner than fps
- degree 2 vs. 3 is not significant in 3 out of 4 comparison

General tasks:

- Perform pipeline on multiple subjects
- Compare with existing works (& baselines):
  - ANTs, coherent point drift (CPD), iterative closest point (ICP), PointNet, DINOreg, etc.

IOT Paper's table (on brain CT paired with brain SPECT from Atlas Dataset)

Table 2
Average rank (with standard deviation in parentheses) based on RMSE of each method on the *CT* dataset in the cases of **C1–C3**, without or with noise. The smaller the better.

Method	C1		C2		C3	
	w/o noise	w/ noise	w/o noise	w/ noise	w/o noise	w/ noise
Open3D-ICP	**3.7** (1.0)	5.3 (1.5)	4.3 (0.9)	4.7 (0.9)	4.5 (0.7)	5.0 (1.0)
Probreg-Filter	8.3 (1.7)	9.8 (1.8)	6.5 (2.8)	5.5 (2.6)	6.6 (2.7)	5.6 (3.2)
Probreg-GMM	4.6 (1.7)	4.5 (1.6)	**3.4** (1.0)	**2.7** (0.9)	**2.6** (1.2)	***2.3*** (1.3)
ANTs-SyN	6.0 (0.8)	5.3 (1.6)	7.8 (1.0)	7.9 (0.5)	8.2 (1.5)	8.5 (1.4)
CR	10.5 (0.7)	9.8 (1.2)	9.2 (1.8)	10.1 (1.0)	9.0 (1.5)	8.0 (1.9)
RGF	5.4 (2.6)	5.3 (2.7)	4.4 (2.1)	4.7 (1.6)	4.5 (2.3)	4.8 (2.4)
DASC	9.3 (1.1)	8.7 (0.8)	10.6 (0.5)	10.4 (0.5)	10.3 (0.8)	10.5 (0.7)
RIFT	9.1 (1.0)	8.8 (1.2)	8.5 (1.7)	8.5 (1.7)	7.4 (2.7)	7.7 (2.4)
DINOreg	5.6 (1.2)	**4.4** (1.9)	7.0 (1.0)	7.1 (0.5)	7.8 (1.5)	8.1 (1.0)
IOT_1	***1.1*** (0.3)	***1.6*** (1.5)	***1.4*** (0.7)	***2.0*** (1.5)	**2.7** (1.4)	**2.4** (1.4)
IOT_2	**2.4** (0.8)	**2.5** (1.0)	**2.9** (1.4)	**2.4** (1.1)	***2.4*** (1.1)	**3.1** (0.9)

* The top-3 results of each case are in bold, and the best result is in italics.
* The median initial RMSEs for **C1–C3** are 14.64, 39.66, and 40.95, respectively.

https://www.med.harvard.edu/aanlib/home.html

[#1] Temporal Neural OT on Point Cloud Sequences

- **Input**: ~~EXP (moving) & INSP (fixed) 3D point clouds~~ Sequence of 3D point clouds at times t_0, t_1, t_2, ... t_n
- **Output**: ~~Polynomial transformation T~~ $T(t_i \rightarrow t_j)$ for any pair (i, j), including pairs not seen during training.

$$f(X) = \Phi(X) \mathbin{@} T \quad \rightarrow \quad f(x, t\_src, t\_tgt) = \Phi(x) \mathbin{@} T(t\_src, t\_tgt)$$

f, polynomial transformation function
T, the N×M transport plan matrix

T(t_src, t_tgt), output by a small neural network (an MLP or transformer)

Loss function - satisfy consistency constraints:

- Inverse consistency: $T(t_i \rightarrow t_j) = T(t_j \rightarrow t_i)^{-1}$
- Transitivity: $T(t_0 \rightarrow t_2) \approx T(t_0 \rightarrow t_1) \circ T(t_1 \rightarrow t_2)$
- Identity at same time: $T(t \rightarrow t) = \text{identity}$

Lambda (λ) from latent distance $||z(t_i) - z(t_j)||$:

- Have to run IOT for each lambda → time-consuming
- If t_i and t_j are far apart, allow larger λ
- Encode each point cloud at time t into a low-dimensional vector z(t) → larger distance (small λ)

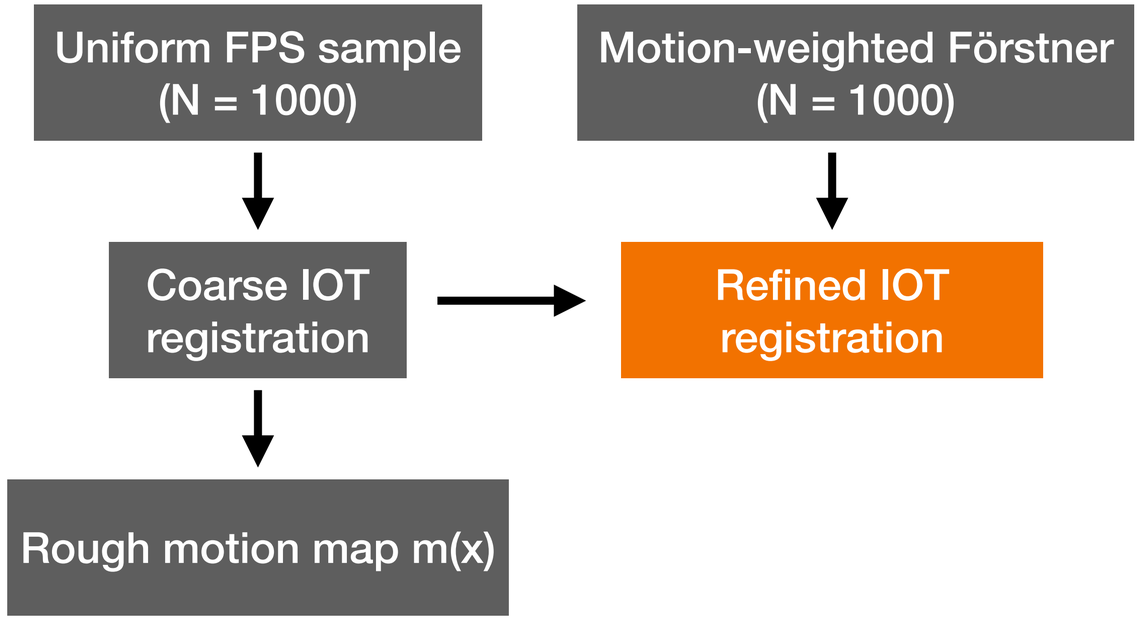
[#2] Motion-weighted Sampling

- **Förstner:** picks points based on local geometric structure (cornerness) + Vessel size (radius scalar)
  - has no knowledge of where the lung moves (large std)

High motion region

$$\text{Score}(x) = \text{cornerness}(x) \quad * \quad \text{radius}(x) \quad * \quad m(x)$$

Anatomical meaningful



For each point x,
average predicted displacement across time pairs
→ scalar motion score per point

Summary

What we did

application of IOT for 3D intra-patient lung CT point cloud registration (EXP + INSP)
original paper only tested on 2D multimodal brain and retina images
tested different samplers

What we found

Sampler contributes significantly Förstner+KL achieves ~60% improvement vs. 2–7% for FPS
UOT-KL is necessary dropping KL significantly drops performance, especially for Förstner
Large std spatially non-uniform registration → motivates motion-weighted sampling

Point sampling + Normalization

Paper Method

- `edge_points()` from Belongie et al.
  - randomly selected edge pixels
  - remove points closest to its nearest neighbor until n=300
  - paper input 2D edge map
- `norm_data()`
  - subtract the center
  - divide by sqrt(Σ||x_i - mean||^2 / N) applied identically to all axis

Our Method

- `foerstner_3d()` from 2D Förstner detector
  - for each point, find k-nearest neighbours using cKDTree
  - 3×3 structure tensor from vector to each neighbourhood -> `S = nbrs.T @ nbrs`
  - compute cornerness score
    - `w_i = det(S_i) / trace(S_i)^3`
    - higher the better
  - select top n points
- Normalization
  - subtract per-axis mean + divide by per-axis standard deviation for x, y, z
  - output -> unit-variance normalization