1 Overview

This section summarizes the data, notation, and end-to-end pipeline used to simulate and learn patient-specific lung deformation from 4D–CT. All symbols introduced here are reused consistently in later sections. we denote

$$\mathbf{u}_{\text{true}} \colon \Omega \subset \mathbb{R}^3 \to \mathbb{R}^3$$

the displacement field computed by non-rigid image registration between a *reference* phase (typically end-exhalation, EE) and a *target* phase (e.g., end-inspiration, EI). At each location $\mathbf{x} \in \Omega$, the vectors $\mathbf{u}_{\text{true}}(\mathbf{x})$ specifies how the reference image must move to align with the target.

1.1 Data specification

The primary entities manipulated by our framework are listed in Table 1. Note that the *volumetric mesh is generated only once during preprocessing and reused for all respiratory phases*, whereas the displacement field is recomputed for each reference–target pair.

Table 1: Data entities used by the pipeline

Symbol / File	**Description**
$I_i \colon \Omega \to \mathbb{R}$	3D CT volume at respiratory phase i (0%, 10%, …, 90%), where 0% corresponds to EE and 50% to EI.
Segmentation mask	multi-class volume (same resolution as I_i) identifying lung parenchyma, airways, etc. Used to *define the computational domain* and extract the surface $\partial\Omega$.
Volumetric mesh $\mathcal{T} = (\mathbf{V}, \mathbf{E}, \mathbf{T})$	Generated once from the segmentation mask; provides vertices, edges, and tetrahedra for the mass–spring system.
Ground-truth displacement $\mathbf{u}_{\text{true}}$	Dense vector field obtained by solving the registration problem in Eq. (1). Interpolated (cubic or a first order (linear) B-spline in implementation)on $\partial\Omega$ to prescribe Dirichlet conditions.
Material labels $c(v)$ or intensities $I(v)$	Node-/element-wise features consumed by the neural network that predicts stiffness.

Segmentation vs. boundary conditions. A segmentation mask encodes *shape only*. First it yields the surface geometry $\partial\Omega$ (via marching cubes). Then the independently computed registration field $\mathbf{u}_{\text{true}}$ is sampled at each surface vertex to obtain the vector-valued data. Hence:

1. *Segmentation* $\longrightarrow$ domain Ω and boundary mesh;
2. *Registration* $\longrightarrow$ displacement vectors on Ω;
3. Sampling $\mathbf{u}_{\text{true}}$ on $\partial\Omega$ $\longrightarrow$ prescribed displacements $\mathbf{u}_i$.

The simulator never consumes the mask directly as a boundary condition.

Image registration

For a given pair $(I_{\text{ref}}, I_{\text{tar}})$, we estimate $\mathbf{u}_{\text{true}}$ by solving

$$\min_{\mathbf{u}} \int_\Omega \left| I_{\text{tar}}(\mathbf{x}) - I_{\text{ref}}(\mathbf{x} + \mathbf{u}(\mathbf{x})) \right|^2 \mathrm{d}\mathbf{x} + \lambda \int_\Omega \|\nabla \mathbf{u}(\mathbf{x})\|^2 \, \mathrm{d}\mathbf{x}, \tag{1}$$

with $\lambda \in [10^{-2}, 10^{-1}]$.The result has been processed and stored as data.

Table 2: End-to-end processing steps

Step	Input → Output	Key operation
1	$(I_{\text{ref}}, I_{\text{tar}}) \to \mathbf{u}_{\text{true}}$	Solve (1); saved as files.
2	Mask $\to \mathcal{T}$	Generate tetrahedral mesh (cached).
3	$k \to \mathbf{u}_{\text{sim}} = g(k)$	Forward spring–mass solve (Sect. **??**): enforce $\nabla_{\mathbf{q}} U(k, \mathbf{q}) = \mathbf{0}$, then compute $\mathbf{u}_{\text{sim}} = \mathbf{q} - \mathbf{X}$.
4	$\mathbf{u}_{\text{true}} \to k^*$	$k^* = \arg\min_k \left\| \mathbf{u}_{\text{true}} - \mathbf{u}_{\text{sim}}(k) \right\|^2$
5	$(I_i, k^*) \to \theta^*$	*Learning*: train ANN $k_\theta = f_\theta(I)$ via $\theta^* = \arg\min_\theta \mathbb{E}_{\text{cases}} \left\| \mathbf{u}_{\text{true}} - \mathbf{u}_{\text{sim}}\big(f_\theta(I)\big) \right\|^2$.

1.2 Processing pipeline

Level 1 (Forward). Given a stiffness field $k(\mathbf{x})$, solve for the simulated displacement:

$$g(k): \quad \frac{\partial U(k, \mathbf{q})}{\partial \mathbf{q}} = \mathbf{0}, \qquad \mathbf{u}_{\text{sim}} = \mathbf{q} - \mathbf{X}.$$

Here:

- Given the deformed nodal coordinates $\mathbf{q} \in \mathbb{R}^{3N}$, and the reference (initial) nodal coordinates $\mathbf{X} \in \mathbb{R}^{3N}$.
- $U(k, \mathbf{q})$ is the total potential energy (sum of all spring energies), where k is the stiffness field.
- $\nabla_{\mathbf{q}} U = \partial U / \partial \mathbf{q}$ is the gradient w.r.t. the nodal $\mathbf{q}$ forces; equilibrium makes the summarization zero.
- Get the simulated displacement field with $\mathbf{u}_{\text{sim}} = \mathbf{q} - \mathbf{X}$.

Level 2 So we need to find k s.t.:

$$k^* = \arg\min_k \left\| \mathbf{u}_{\text{true}} - \mathbf{u}_{\text{sim}}(k) \right\|^2,$$

Level 3 (Learning). Learn a mapping from image to stiffness:

$$\theta^* = \arg\min_\theta \mathbb{E}_{\text{cases}} \left\| \mathbf{u}_{\text{true}} - \mathbf{u}_{\text{sim}}\big(f_\theta(I)\big) \right\|^2,$$

with $k_\theta = f_\theta(I)$ the ANN prediction and gradients flowing through the differentiable solver $g(\cdot)$.

2 Spring Mass System Approach

For the forward simulation (Level 1), we model the deformable tissue, such as the lung, as a Mass-Spring System (MSM). The system consists of a grid of particles (mass points) interconnected by a network of springs. This model approximates the continuous mechanics of the tissue. Usually we need three steps to update the system:

1. build the topology of the mass spring system
2. Compute the forces based on the previous topology and act on each particle
3. Integrate these forces over time to update the particle positions and velocities. Repeat the process for each time step until the simulation reaches a desired state or time.

This section includes how we assemble the whole system from the aspect of total potential energy coming from the defined-topology springs. The logic of the section is as follows: Firstly, we talk about the usual concepts of a mass-spring system, including how to compute the spring forces and how to update the system forward. Then we introduce the topology of the volumetric mesh, which is a tetrahedral mesh, and how to compute the forces and use them to update the whole system.

2.1 Introduction to Mass–Spring Systems

In this section we outline the classical pipeline for a mass–spring system (SMS) simulation in three logical steps:

1. **Build the topology of the mass–spring system.**
2. **From the energy perspective, compute forces on each particle.**
3. **Integrate these forces over time to update positions and velocities.**

Particle definitions We consider a 3D grid of N particles. Each particle i has

- **Mass** m_i. For simplicity we set $m_i = m$ for all i.
- **Position** $\mathbf{x}_i(t) \in \mathbb{R}^3$. Collectively $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_N) \in \mathbb{R}^{3N}$.
- **Velocity** $\mathbf{v}_i(t) = \dot{\mathbf{x}}_i(t)$.

1. Topology We connect particles by three types of springs based on their grid indices (i, j, k):

- **Structural springs:** between (i, j, k) and $(i+1, j, k)$, $(i, j+1, k)$, $(i, j, k+1)$.
- **Shear springs:** between diagonal neighbors, e.g. (i, j, k) and $(i+1, j+1, k)$, etc.
- **Bend (flexion) springs:** between particles two steps away, e.g. (i, j, k) and $(i+2, j, k)$.

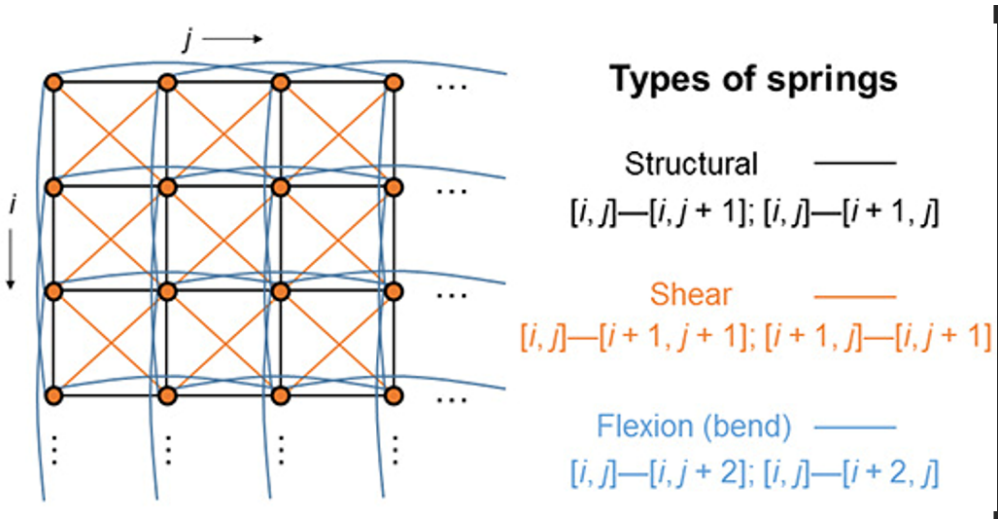

Figure 1: Example mass–spring topology for a thin cloth patch.

2. Forces from Energy Each spring contributes a potential energy

$$U_{\text{spring}}(\mathbf{p}, \mathbf{q}) = \tfrac{1}{2}\, k\big(\|\mathbf{p} - \mathbf{q}\| - L_0\big)^2,$$

where $\mathbf{p}, \mathbf{q}$ are its endpoint positions, k its stiffness, and L_0 the rest length. The total elastic energy is

$$U(\mathbf{x}) = \sum_{\text{springs } (i,j)} \tfrac{1}{2}\, k_{ij}\big(\|\mathbf{x}_i - \mathbf{x}_j\| - L_{0,ij}\big)^2.$$

The internal spring force on particle i is the negative energy gradient:

$$\mathbf{F}_i^{\text{int}} = -\frac{\partial U}{\partial \mathbf{x}_i} = \sum_{j \in \mathcal{N}(i)} k_{ij}\big(L_{0,ij} - \|\mathbf{x}_i - \mathbf{x}_j\|\big)\,\frac{\mathbf{x}_i - \mathbf{x}_j}{\|\mathbf{x}_i - \mathbf{x}_j\|}.$$

It's the same as the spring force in Hooke's law, where the force is proportional to the displacement from the rest length, and acts along the line connecting the two particles.:

Spring Forces Consider the internal force as the sum of forces from all springs connected to particle i. For a single spring connecting two particles at positions $\mathbf{p}$ and $\mathbf{q}$, with stiffness k and rest length L_0, the restoring force exerted on particle $\mathbf{p}$ is given by Hooke's Law:

$$\mathbf{F}_{\text{spring}} = k(L_0 - \|\mathbf{p} - \mathbf{q}\|)\frac{\mathbf{p} - \mathbf{q}}{\|\mathbf{p} - \mathbf{q}\|} \tag{2}$$

External forces (e.g. gravity $\mathbf{F}_i^{\text{ext}} = m\,\mathbf{g}$, contact, boundary constraints) are added to form the net force

$$\mathbf{F}_i(\mathbf{x}, \mathbf{v}) = \mathbf{F}_i^{\text{int}} + \mathbf{F}_i^{\text{ext}}.$$

3. System Update Particle motion obeys Newton's second law:

$$m_i\,\ddot{\mathbf{x}}_i(t) = \mathbf{F}_i\big(\mathbf{x}(t), \mathbf{v}(t)\big). \tag{3}$$

A simple explicit Euler scheme with time step Δt reads

$$\mathbf{v}_i(t + \Delta t) = \mathbf{v}_i(t) + \frac{\Delta t}{m_i}\,\mathbf{F}_i\big(\mathbf{x}(t), \mathbf{v}(t)\big), \quad \mathbf{x}_i(t + \Delta t) = \mathbf{x}_i(t) + \Delta t\;\mathbf{v}_i(t).$$

This loop (build topology–compute forces–integrate) is repeated each time step until the simulation reaches its target time or state. Here the right-hand sides are evaluated at time t. Explicit Euler is easy to implement but conditionally stable (requires small Δt).

2.2 Topology for volumetric mesh

Intersection Points & Coefficient Matrix

In each tetrahedral element $\mathcal{V}_k$, we locate six intersection points $q_j[\mathbf{?}, \mathbf{?}]$ by ray-casting from the **barycenter** x_b along the three anisotropy axes to the faces. At the same time we build a 4×6 coefficient matrix C^k whose entries let us reconstruct any intersection q_j from the four vertex positions.

1. **Barycenter**

   where x_i are the four vertex coordinates.

$$x_b \;=\; \frac{1}{4}\sum_{i=1}^{4} x_i \tag{2.22}$$

2. **Point-in-triangle test & barycentric coords**

   A traced point q_j on face $\Delta_{i_1 i_2 i_3}$ is inside if and only if

$$S_{\Delta_{i_1 i_2 i_3}} \;=\; S_{\Delta_{q_j\, i_2\, i_3}} + S_{\Delta_{i_1\, q_j\, i_3}} + S_{\Delta_{i_1\, i_2\, q_j}}. \tag{2.23}$$

   Then its local (area) coordinates on that triangle are

$$\xi = \frac{S_{\Delta_{q_j\, i_2\, i_3}}}{S_{\Delta_{i_1 i_2 i_3}}}, \quad \eta = \frac{S_{\Delta_{q_j\, i_1\, i_3}}}{S_{\Delta_{i_1 i_2 i_3}}}, \quad 1 - \xi - \eta = \frac{S_{\Delta_{i_1\, i_2\, q_j}}}{S_{\Delta_{i_1 i_2 i_3}}}. \tag{2.24}$$

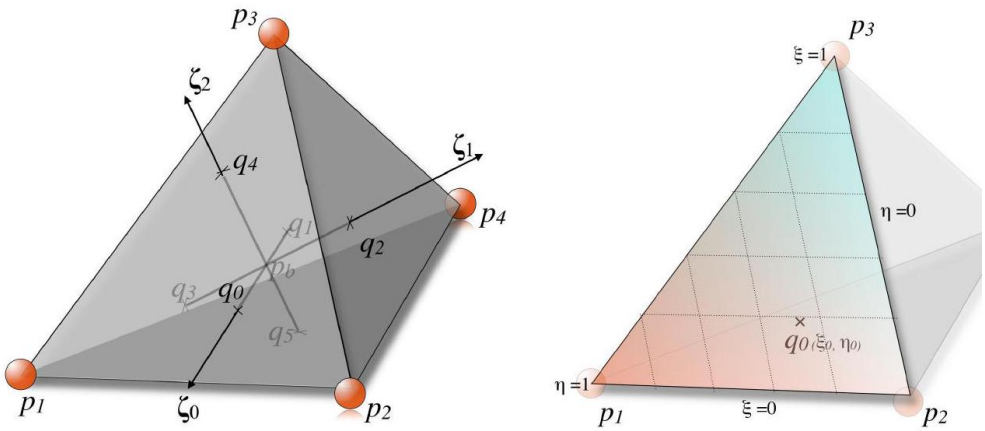

Figure 2: Intersection points in a tetrahedral volume element: The tetrahedron with three axes of anisotropy set at the barycenter and the six intersection points that they define (a), a triangular face of the element containing an intersection point and the coefficients ξ_0 and η_0 related to the intersection point. Note that ξ increases with the cyan color gradient starting from $\xi = 0$ at the line segment (p_1, p_2) and is equal to $\xi = 1$ at p_3, while η increases along the orange color gradient starting from $\eta = 0$ at (p_2, p_3) until it reaches $\eta = 1$ at p_1 (b).

3. **Building the coefficient matrix C^k**

   For each intersection q_j we evaluate the four linear shape-functions N_i of the tetrahedron's nodes $i = 1 \dots 4$. On the face containing q_j, those coincide with the barycentric coordinates:

$$\begin{cases} N_{i_1}(q_j) = 1 - \xi - \eta, \\ N_{i_2}(q_j) = \xi, \\ N_{i_3}(q_j) = \eta, \\ N_{i_4}(q_j) = 0, \end{cases}$$

   where $\{i_1, i_2, i_3\}$ are the face nodes and i_4 is the opposite vertex. We then set

$$C^k_{ij} \;=\; N_i(q_j),$$

   assembling a 4×6 matrix whose j-th column holds the four shape-function values at q_j.

4. **Updating intersections**

   At runtime, once the current vertex positions x^t_i are known, each intersection moves as

$$x^t_j \;=\; \sum_{i=1}^{4} C^k_{ij}\, x^t_i \tag{2.25}$$

   reproducing the straight-sided mapping of a linear tetrahedron.

   In the implementation the six q_j and the corresponding C_k are updated each step to remain exact under large deformation.

2.3 Internal Forces

Internal ("deformation") forces in each tetrahedron are computed by **three axial springs** along the anisotropy axes, plus **three torsion springs** coupling each pair of axes. See Fig. 3. The angle α^t_{lm} between the axes ζ_{l} and ζ_{m} can be given by...

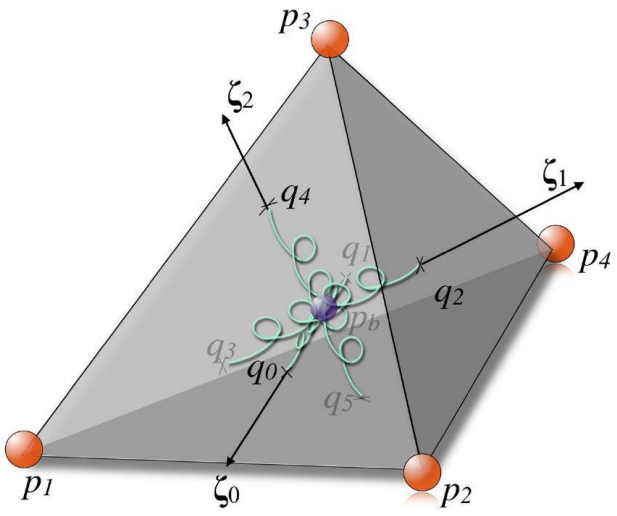

Figure 3: A tetrahedron with three axial springs (in cyan) along the axes of anisotropy and three torsion springs in the barycenter of the tetrahedron (in violet).

2.3.1 Axial Springs

- **Axis vectors**: Along axis $\ell \in \{1, 2, 3\}$, let the two intersection points be $q_{\ell,1}$ and $q_{\ell,2}$. Their current axis vector is
$$\zeta_\ell^t = x_{q_{\ell,1}}^t - x_{q_{\ell,2}}^t.$$
- **Initial length** (at $t = 0$):
$$l_\ell^0 = \|\zeta_\ell^0\| = \| x_{q_{\ell,1}}^0 - x_{q_{\ell,2}}^0 \|. \tag{2.30}$$
- **Unit direction**:
$$\hat{\zeta}_\ell^t = \frac{\zeta_\ell^t}{\|\zeta_\ell^t\|}. \tag{2.31}$$
- **Hooke's law** (linear axial force):
$$\boxed{f_{\ell,\,\text{axial}}^t = -\,k_\ell \big(\|\zeta_\ell^t\| - \|\zeta_\ell^0\|\big)\,\hat{\zeta}_\ell^t} \tag{2.35}$$
where k_ℓ is the stiffness constant.

2.3.2 Torsion Springs

To capture bending resistance between each pair of anisotropy axes in a tetrahedron, we introduce **torsion springs**. These springs penalize deviations of the angles between axes from their rest values.

1. Angle between two axes For any two axes ℓ and m, the angle is
$$\alpha_{\ell m}^t = \arccos\big(\hat{\zeta}_\ell^t \cdot \hat{\zeta}_m^t\big), \quad \alpha_{\ell m}^0 = \alpha_{\ell m}^{t=0} \tag{2.32}$$
where $\hat{\zeta}_\ell^t$ and $\hat{\zeta}_m^t$ are the unit–direction vectors at time t, and $\alpha_{\ell m}^0$ is the **rest angle**, measured in the undeformed configuration.

2. Decomposing the torsion force At each intersection point on axis ℓ, the net torsion force $f_{\ell,1}$ splits into three orthogonal components:

$$f_{\ell,1} = f_S(\zeta_\ell, \alpha_{\ell m}, \alpha_{\ell n})\,\hat{\zeta}_\ell + f_\tau(\zeta_\ell, \alpha_{\ell m}, \alpha_{\ell n})\,\hat{\zeta}_m + f_\tau(\zeta_\ell, \alpha_{\ell m}, \alpha_{\ell n})\,\hat{\zeta}_n, \tag{2.33}$$

with $f_{\ell,2} = -\,f_{\ell,1}$, and $\{m, n\}$ are the other two axes.

- **Axial** component f_S acts along $\hat{\zeta}_\ell$.
- **Torsional** components f_τ lie in the planes $(\hat{\zeta}_\ell, \hat{\zeta}_m)$ and $(\hat{\zeta}_\ell, \hat{\zeta}_n)$.

Expressions for f_S and f_τ We derive both from simple spring energies: $f_S = -\frac{\mathrm{d}U_S}{\mathrm{d}\|\zeta_\ell\|}$.

In a conservative spring model, the force along a single coordinate x is the negative derivative of its potential energy:

$$F(x) = -\frac{\mathrm{d}}{\mathrm{d}x}\,U(x).$$

Here our "coordinate" is the current length $\|\zeta_\ell\|$, so the axial force magnitude is

$$f_S = -\frac{\mathrm{d}}{\mathrm{d}\|\zeta_\ell\|}\,U_S.$$

1. **Axial term**: Define $U_S = \frac{1}{2}\,k_\ell\big(\|\zeta_\ell^t\| - \|\zeta_\ell^0\|\big)^2$. Then

$$f_S = -\frac{\mathrm{d}}{\mathrm{d}\|\zeta_\ell\|}\Big[\tfrac{1}{2}\,k_\ell(\|\zeta_\ell\| - \|\zeta_\ell^0\|)^2\Big] = -\,k_\ell\big(\|\zeta_\ell^t\| - \|\zeta_\ell^0\|\big),$$

   and the vector is $\mathbf{f}_S = f_S\,\hat{\zeta}_\ell$.

2. **Torsional terms**: Define $U_\tau = \frac{1}{2}\sum_{p\in\{m,n\}} k_{\ell p}\left(\alpha_{\ell p}^t - \alpha_{\ell p}^0\right)^2$. Differentiating with respect to each angle gives

$$f_\tau(\zeta_\ell, \alpha_{\ell m}, \alpha_{\ell n}) = -\,k_{\ell m}\left(\alpha_{\ell m}^t - \alpha_{\ell m}^0\right),$$

   and similarly for (ℓ, n).

3. Linear torsion-spring model

$$\boxed{f_{\ell\to m}^t = -\,k_{\ell m}\left(\alpha_{\ell m}^t - \alpha_{\ell m}^0\right)\hat{\zeta}_m^t,} \quad f_{m\to\ell}^t = -\,f_{\ell\to m}^t. \tag{2.40–2.41}$$

4. Cosine-approximation (small-angle) When axes remain near orthogonal, $\alpha_{\ell m}^t - \alpha_{\ell m}^0 \approx (\hat{\zeta}_\ell^t \cdot \hat{\zeta}_m^t) - (\hat{\zeta}_\ell^0 \cdot \hat{\zeta}_m^0)$. Thus

$$\boxed{f_{\ell\to m}^t = -\,k_{\ell m}\left((\hat{\zeta}_\ell^t \cdot \hat{\zeta}_m^t) - (\hat{\zeta}_\ell^0 \cdot \hat{\zeta}_m^0)\right)\hat{\zeta}_m^t,} \quad f_{m\to\ell}^t = -\,k_{\ell m}\left((\hat{\zeta}_\ell^t \cdot \hat{\zeta}_m^t) - (\hat{\zeta}_\ell^0 \cdot \hat{\zeta}_m^0)\right)\hat{\zeta}_\ell^t. \tag{2.44–2.45}$$

Assembly Each tetrahedron contributes:

- 6 axial-spring forces, and
- 6 torsion-spring forces,

which are then distributed to the four vertices via the shape-function coefficients C^k and summed with any body forces before time integration.

2.4 Axis of Anisotropy

In the original paper, the axis of the tetra is set instead of calculated from the data. Consider our problem, we have the deformation direction from the 4D-CT image registratioin. So here we will give how to set the axis of anisotropy based on the deformation direction.

2.4.1 Get the deformation direction

Firstly, the image registration gives us a displacement vector field $\mathbf{u}_i$ at each node X_i of the tetrahedral mesh. It's interpolated to get the displacement at each node. For each node, it's a one 3-vector. Then we can have the ground truth position of each node $x_i = X_i + u_i$. Within each tetra (with local nodes 0,1,2,3): we have:

$$D_m = [X_1 - X_0, X_2 - X_0, X_3 - X_0], d_x = [x_1 - X_0, x_2 - X_0, x_3 - X_0]$$

Both are 3×3 matrices built purely from known X_i and x_i.

Assume that within this tetra the mapping $X \to x$ is affine: $x = \alpha X + \beta$, where α is a constant 3×3 matrix (the deformation gradient) and β is a translation.

Write this equation for all four vertices $X_i \to x_i$: $x_i = \alpha \cdot X_i + \beta$, for $i = 0, 1, 2, 3$

Eliminate β by subtracting we can have:

$x_1 - x_0 = \alpha \cdot (X_1 - X_0)$, $x_2 - x_0 = \alpha \cdot (X_2 - X_0)$, $x_3 - x_0 = \alpha \cdot (X_3 - X_0)$.

Pack these three edge-differences into matrices

$$d_x = [x_1 - x_0, x_2 - x_0, x_3 - x_0], \quad D_m = [X_1 - X_0, X_2 - X_0, X_3 - X_0].$$

We can rewrite the above equations as:

$$d_x = \alpha \cdot D_m.$$

Because D_m is invertible (non-degenerate tetra), we can expressα: $\alpha = d_x \cdot D_m^{-1}$

2.4.2 Set the anisotropy axes from α

The transformation α could be factorized via SVD:

Factor $\alpha = U \cdot \Sigma \cdot V^T$, where $\Sigma = \text{diag}(\lambda_0, \lambda_1, \lambda_2)$, and $\lambda_0 \geq \lambda_1 \geq \lambda_2$, Columns v_0, v_1, v_2 of V are orthonormal principal-stretch directions in the reference frame.

With this, we can assign the anisotropy axes as follows:

- $e_0 = v_0$ (largest stretch direction),
- $e_1 = v_1$ (second stretch direction),
- $e_2 = v_2$ (third stretch direction).

The axes e_0, e_1, e_2 are orthonormal, and the eigenvalues $\lambda_0, \lambda_1, \lambda_2$ are the principal stretches along these axes.

Note that, in right handed co-coordinate axis, $(e_0 \times e_1)$ is the direction of e_2. So if $(e_0 \times e_1) \cdot e_2 < 0$, swap e_1 and e_2 to keep e_0, e_1, e_2 form a right-hand basis.

2.4.3 Handling Degenerate or Near-Rigid Tetrahedra

In the extraction of anisotropy axes via the SVD , two pathological scenarios can undermine numerical stability and physical fidelity: element inversion or severe compression, and near-rigid motion. We therefore introduce a descriptive two-branch fallback strategy.

1. Inverted or Highly Compressed Elements When an element inverts ($\det(\alpha) \leq 0$) or one principal stretch becomes negligible compared to the largest ($\lambda_2 \ll \lambda_0$), the SVD directions lose their intended meaning and may fluctuate wildly. In this case we replace the SVD axes with a stable, displacement-based frame:

1. Compute the mean nodal displacement

$$\bar{\mathbf{u}} = \tfrac{1}{4}\big(\mathbf{u}_0 + \mathbf{u}_1 + \mathbf{u}_2 + \mathbf{u}_3\big).$$

This vector represents the overall deformation trend of the tetrahedron. And the tetra deform mainly along this direction.

2. If $\|\bar{\mathbf{u}}\| > \varepsilon$, define the first axis by normalizing:

$$\mathbf{e}_0 = \frac{\bar{\mathbf{u}}}{\|\bar{\mathbf{u}}\|}.$$

This ensures $\mathbf{e}_0$ aligns with the dominant displacement direction.

3. To obtain a second orthogonal direction, select a reference edge $\mathbf{r} = \mathbf{X}_1 - \mathbf{X}_0$ and remove its component along $\mathbf{e}_0$:

$$\mathbf{e}_1 = \frac{\mathbf{r} - (\mathbf{r} \cdot \mathbf{e}_0)\,\mathbf{e}_0}{\left\|\mathbf{r} - (\mathbf{r} \cdot \mathbf{e}_0)\,\mathbf{e}_0\right\|}.$$

This guarantees $\mathbf{e}_1 \perp \mathbf{e}_0$.

4. Define the third axis by the right-hand rule:

$$\mathbf{e}_2 = \mathbf{e}_0 \times \mathbf{e}_1.$$

If instead $\|\bar{\mathbf{u}}\| \leq \varepsilon$, the element is effectively stationary and we retain the previous axes to prevent introducing noise.

2. Nearly Rigid Elements When all principal stretches are close to unity ($|\lambda_i - 1| < \varepsilon$ for $i = 0, 1, 2$), the tetrahedron undergoes almost pure rigid-body motion. In this regime the SVD directions are well defined in theory but even a slight numerical difference will cause the direction to constantly shift slightly. Here we simply preserve the last computed $\{\mathbf{e}_0, \mathbf{e}_1, \mathbf{e}_2\}$ until a significant deformation occurs.

Here, $\{\lambda_0, \lambda_1, \lambda_2\}$ are the singular values of α sorted so that $\lambda_0 \geq \lambda_1 \geq \lambda_2$, and ε is a small threshold (e.g. 10^{-6}).

In the Taichi field for tetra:

```
anisotropy_axes = ti.Vector.field(3, dtype=ti.f32, shape=(num_tetrahedra, 3))
anisotropy_axes[i,0] = e0
anisotropy_axes[i,1] = e1
anisotropy_axes[i,2] = e2
```

2.5 Simplified Volume Preservation (Barycentric Volume Springs)(Removed)

To control tetrahedral volume without full tensors, we use **barycentric springs** [Eqns. 2.76–2.77]:

1. **Current barycenter**:

$$x_b^t = \frac{1}{4} \sum_{i=1}^{4} x_i^t.$$

2. **Radial vectors**: $\xi_j^t = x_b^t - x_j^t$, with lengths $\|\xi_j^t\|$.

3. **Rest lengths** $\|\xi_j^0\|$ computed at $t = 0$.

4. **Total length error**:

$$\Delta L = \sum_{j=1}^{4} \|\xi_j^t\| \; - \; \sum_{j=1}^{4} \|\xi_j^0\|.$$

5. **Barycentric spring force** on node j:

$$f_j^t = -\,k_s\,\Delta L\,\frac{\xi_j^t}{\|\xi_j^t\|} \; - \; c\,(v_j^t - v_b^t),$$

where k_s is the bulk-modulus-based stiffness, c a damping coefficient, and v_b^t the barycenter velocity.

6. **Adaptive stiffness update** (LMS, Eq. 2.81):

$$k_s^{t+\Delta t} = k_s^t + \mu\, \Delta V \sum_{j=1}^{4} \|\xi_j^t\|,$$

clamped to $[k_{\min}, k_{\max}]$, with $\Delta V = V^t - V^0$ the volume error.

This completes the fully-spring-based model:

1. mesh topology & intersection (Eqs. 2.22–2.25),
2. internal forces via axial+torsion springs (Eqs. 2.30–2.44–2.45),
3. volume control via barycentric springs.

Below is a fully narrative, paper-style presentation. Every formula is preserved, with descriptive text to guide the reader through each step.

3 Static Equilibrium Solver

In modeling lung deformation from 4D-CT data, the motion between respiratory phases is slow that inertial effects can be neglected. Each phase can therefore be treated as a quasi-static problem. The core objective of the static equilibrium solver is to find a configuration of nodal positions $\mathbf{q}$ that exactly balances the internal elastic forces generated by the mass–spring network against any externally applied loads. In compact form, this balance has the form of a system of equations:

$$\mathbf{g}_{\text{int}}(\mathbf{q}) \;+\; \mathbf{F}_{\text{ext}} \;=\; \mathbf{0}.$$

Here, $\mathbf{g}_{\text{int}}(\mathbf{q}) \in \mathbb{R}^{3N}$ is the gradient of the total elastic energy with respect to nodal coordinates, and $\mathbf{F}_{\text{ext}} \in \mathbb{R}^{3N}$ is the assembled vector of all external forces.

3.1 Boundary Conditions

In our lung simulation, the surface displacements extracted from 4D-CT act as kinematic anchors, while physiological pressures or tractions drive the tissue response. We therefore split our boundary treatment into two complementary types: one that enforces exactly-known nodal positions, and one that applies distributed surface loads. Below we introduce each in turn before detailing their mathematical implementation.

3.1.1 Dirichlet Boundary Γ_D

On the Dirichlet boundary Γ_D, nodal positions are prescribed by image-registration data. Denoting by N_D the number of such boundary vertices, their deformed coordinates satisfy

$$\mathbf{q}_D \;=\; \mathbf{X}_D \;+\; \mathbf{u}_i \quad \in \mathbb{R}^{3N_D}.$$

In this expression:

- $\mathbf{X}_D \in \mathbb{R}^{3N_D}$ contains the reference (undeformed) positions of the boundary nodes.
- $\mathbf{u}_i \in \mathbb{R}^{3N_D}$ is the displacement field at phase i, interpolated from the 4D-CT registration.
- N_D is the total number of vertices on Γ_D.

These constraints are enforced by fixing $\Delta\mathbf{q}_D = 0$ in the Newton–Raphson updates (see Section 5).

3.1.2 Neumann Boundary Γ_N

Nodes on the Neumann boundary Γ_N are subjected to surface tractions (for example, pressure acting normal to the lung surface). The equivalent nodal force at node i is obtained by

$$[\mathbf{F}_{\text{ext}}]_i \;=\; \int_{\Gamma_N} N_i(\mathbf{s})\,\mathbf{t}(\mathbf{s})\,\mathrm{d}S, \quad i = 1, \ldots, N,$$

where:

- The integral is taken over the entire Neumann boundary Γ_N.
- $\mathbf{F}_{\text{ext}} \in \mathbb{R}^{3N}$ is the vector of external nodal forces.
- $N_i(\mathbf{s})$ is the finite-element shape function associated with node i, evaluated at surface point $\mathbf{s}$.
- $\mathbf{t}(\mathbf{s})$ denotes the traction vector at $\mathbf{s}$, typically the product of a pressure difference and the outward unit normal.

3.1.3 Energy–Volume Conjugacy and Pressure Derivation

A uniform pressure p applied to Γ_N can be shown to arise naturally as the volumetric derivative of the network's elastic energy:

$$p \;=\; \pm\frac{\mathrm{d}U}{\mathrm{d}V}.$$

Pressure is simply the rate at which the system's elastic energy changes when the volume is changed, because equilibrium demands that the work done by pressure exactly balances the additional stored energy for any uniform expansion or contraction. If we consider one tiny, isotropic puff that changes the volume by δV, the requirement that the extra work the pressure would do is exactly balanced by the extra elastic energy stored forces p to equal ($\pm$) the derivative of energy with respect to volume

We can derive this relationship as follows:

1. Total Elastic Energy: The energy stored in all springs is

$$U(q) = \sum_{(i,j)\in E} \frac{1}{2}\,k_{ij}\big(\|q_i - q_j\| - L_{ij}^0\big)^2,$$

where E is the set of springs, k_{ij} their stiffnesses, and L_{ij}^0 their rest lengths.

2. Mesh Volume. The current volume is the sum of signed tetrahedral volumes:

$$V(q) = \sum_{t\in\mathcal{T}} V_t(q).$$

3. External Work: When the mesh expands by δV against pressure p, the work done by the pressure is

$$W_{\text{ext}} \;=\; -\,p\,\delta V,$$

where the negative sign reflects work performed by the system on its surroundings when volume increases.

4. Change in Elastic Energy. The uniform volumetric increase makes nodal displacements δq:

$$\delta U = \nabla_q U \;\cdot\; \delta q.$$

so we can have

$$\delta q \;=\; \frac{\partial q}{\partial V}\,\delta V.$$

5. Work–Energy Balance: Static equilibrium requires that the spring-energy increase plus external work sum to zero:

$$\delta U + W_{\text{ext}} = 0 \quad\Longrightarrow\quad \frac{\mathrm{d}U}{\mathrm{d}V}\,\delta V \;-\; p\,\delta V = 0.$$

Dividing by δV yields

$$p \;=\; \pm \frac{\mathrm{d}U}{\mathrm{d}V}.$$

Hence, pressure emerges as the derivative of total spring energy with respect to volume, avoiding the introduction of new parameters.

3.2 System Partitioning

To solve for unknown nodal positions while enforcing Dirichlet constraints, vectors are partitioned into free (F) and fixed (D) components:

$$\mathbf{q} = \begin{bmatrix} \mathbf{q}_F \\ \mathbf{q}_D \end{bmatrix}, \quad \mathbf{F}_{\text{ext}} = \begin{bmatrix} \mathbf{F}_{F,\text{ext}} \\ \mathbf{F}_{D,\text{ext}} \end{bmatrix}.$$

- $\mathbf{q}_F \in \mathbb{R}^{3N_F}$ collects the unknown free-node coordinates, with $N_F = N - N_D$.
- $\mathbf{F}_{F,\text{ext}} \in \mathbb{R}^{3N_F}$ contains the forces acting on these free nodes.
- $\mathbf{F}_{D,\text{ext}} \in \mathbb{R}^{3N_D}$ represents reaction forces at the fixed nodes, used later for post-processing.

3.3 Total Potential Energy and Residual

3.3.1 Internal Potential Energy

Within each tetrahedral element $e \in \mathcal{T}$, all spring edges (i, j) contribute to the total energy:

$$U(k, \tilde{\mathbf{q}}) = \sum_{e \in \mathcal{T}} \sum_{(i,j) \in e} \frac{1}{2}\, k_{ij}\, \|\tilde{\mathbf{q}}_i - \tilde{\mathbf{q}}_j\|^2.$$

Taking the gradient with respect to $\tilde{\mathbf{q}}$ yields the internal force vector

$$\mathbf{g}_{\text{int}}(\tilde{\mathbf{q}}) = \nabla_{\tilde{\mathbf{q}}}\, U(k, \tilde{\mathbf{q}}) \;\in\; \mathbb{R}^{3N}.$$

3.3.2 Discretization of Uniform Pressure

When $\mathbf{t}(\mathbf{s}) = -\,p\,\mathbf{n}(\mathbf{s})$ on Γ_N, the nodal forces assemble via:

1. Face Force Approximation. Each triangle f with area A_f and normal $\mathbf{n}_f$ contributes

$$\mathbf{F}_f \approx -\,p\,A_f\,\mathbf{n}_f.$$

2. Lumped Nodal Force. Splitting $\mathbf{F}_f$ equally among its three vertices gives

$$\mathbf{F}_i^{(f)} = -\,\frac{p\,A_f}{3}\,\mathbf{n}_f.$$

3. Global Assembly. Summation over all faces incident on node i results in

$$\mathbf{F}_{\text{ext},i} = \sum_{f \ni i} \mathbf{F}_i^{(f)}.$$

4. Extraction of Free-DOF Vector. Collecting only those entries with $i \notin \Gamma_D$ produces $\mathbf{F}_{F,\text{ext}}$.

3.3.3 Equilibrium Residual

After including the external forces, the total residual vector for the system is defined as:

$$\mathbf{r}(\mathbf{q}) = \mathbf{g}_{\text{int}}(\mathbf{q}) + \mathbf{F}_{\text{ext}} \in \mathbb{R}^{3N}.$$

The system is in static equilibrium if this residual is zero. For our partitioned system, the residual for the free degrees of freedom must be zero:

$$\mathbf{r}_F = \mathbf{g}_F + \mathbf{F}_{F,\text{ext}} = \mathbf{0} \quad \in \mathbb{R}^{3N_F}.$$

3.4 Partitioned Linear System

After assembling the tangent stiffness matrix K and residual $\mathbf{r}$ at the current configuration $\mathbf{q}$, we enforce Dirichlet constraints and solve only for the free degrees of freedom via a direct matrix decomposition (e.g. Cholesky or LU) in Taichi.

Static equilibrium of the linear-spring network can be written as

$$\mathbf{g}_{\text{int}}(\mathbf{q}) + \mathbf{F}_{\text{ext}} = \mathbf{0}, \quad \text{with} \quad \mathbf{g}_{\text{int}}(\mathbf{q}) = K\,\mathbf{q} \quad \text{(constant stiffness } K\text{)}.$$

Partitioning into free (F) and Dirichlet (D) components:

$$\underbrace{\begin{bmatrix} K_{FF} & K_{FD} \\ K_{DF} & K_{DD} \end{bmatrix}}_{K} \begin{bmatrix} \mathbf{q}_F \\ \mathbf{q}_D \end{bmatrix} + \begin{bmatrix} \mathbf{F}_{F,\text{ext}} \\ \mathbf{F}_{D,\text{ext}} \end{bmatrix} = \mathbf{0}.$$

Since $\mathbf{q}_D$ is known, the free–free block yields the reduced system:

$$K_{FF}\,\mathbf{q}_F = -\Big(K_{FD}\,\mathbf{q}_D + \mathbf{F}_{F,\text{ext}}\Big).$$

3.4.1 Direct Solve via Matrix Decomposition

Form the right-hand side

$$\mathbf{b} \;=\; -\big(K_{FD}\,\mathbf{q}_D + \mathbf{F}_{F,\text{ext}}\big),$$

then solve the $(3N_F) \times (3N_F)$ linear system

$$K_{FF}\,\mathbf{q}_F = \mathbf{b}$$

in Taichi by one of:

1. Cholesky decomposition (for SPD K_{FF}):

$$K_{FF} = L\,L^T, \quad L\,y = \mathbf{b}, \quad L^T\,\mathbf{q}_F = y.$$

2. LU decomposition (general K_{FF}):

$$P\,K_{FF} = L\,U, \quad L\,y = P\,\mathbf{b}, \quad U\,\mathbf{q}_F = y.$$

A concise Taichi pseudocode sketch:

```
# assemble K_FF, K_FD, F_F_ext, q_D
b = -(K_FD @ q_D + F_F_ext)
# factorize and solve
L = ti.linalg.cholesky(K_FF)                # or LU via ti.linalg.lu()
y = L.solve(b)
q_F = L.T.solve(y)
# now q_F contains the free-node positions
```

3.4.2 Reaction Forces (Optional)

If needed, compute reaction forces at Dirichlet nodes via

$$\boldsymbol{\lambda} = -\,\mathbf{r}_D \;-\; K_{DF}\,\Delta\mathbf{q}_F \;\in\; \mathbb{R}^{3N_D}.$$

With this approach, the heavy linear algebra is offloaded to Taichi's optimized matrix routines, and only the $(3N_F) \times (3N_F)$ system is factorized and solved at each step.“‘

Why is $\Delta\mathbf{q}_D = 0$: Even though the **target** Dirichlet boundary does change from one breathing phase to the next, each phase is still treated as a *separate* equilibrium solve in which the boundary is held *exactly* at its prescribed shape. Concretely:

1. **Phase loop vs. sparse solver loop**

   In the static equilibrium solver, we have two nested loops:

   (1) Outer "phase" loop: we step through phases $i = 1, 2, \ldots$, and for each phase we load the new surface displacement field $q_D^{(i)} = X_D + u_i$ as the **fixed** boundary.

   (2) Inner Sparse Solver: within that phase, we enforce $\Delta q_D = 0$ at **every** sparse solver iteration, so that the Dirichlet nodes never move away from $q_D^{(i)}$.

2. **Implication for K_{FD}**

   When we write the partitioned linear system

$$\begin{bmatrix} K_{FF} & K_{FD} \\ K_{DF} & K_{DD} \end{bmatrix} \begin{bmatrix} \Delta q_F \\ \Delta q_D \end{bmatrix} = - \begin{bmatrix} r_F \\ r_D \end{bmatrix},$$

we are imposing $\Delta q_D = 0$. That instantly kills off the $K_{FD}\,\Delta q_D$ term in the free-DOF equation, leaving

$$K_{FF}\,\Delta q_F = -\,r_F.$$

   So, for **each** static solve we never actually need to assemble or apply the K_{FD} block.

3. When could K_{FD} matter?

   If we chose to **ramp** the boundary—e.g. start at zero displacement and gradually enforce $q_D^{(i)}$ over a series of sub-steps—then in each sub-step we'd have a **known** nonzero Δq_D, and we would need to include

$$-\,K_{FD}\,\Delta q_D$$

on the right-hand side for the free nodes. In our lung-phase workflow, however, the boundary jump from the previous anatomical phase to the next is **applied all at once** at the top of the solve, so the Newton iterations themselves always see $\Delta q_D = 0$.

In summary, even though the value of q_D is different from phase to phase, within each equilibrium solve it is a **hard** (fully enforced) Dirichlet condition. We therefore do **not** need to include or worry about K_{FD} in the free-DOF solver.

Think of it as a two-stage update:

1. **Apply the new boundary shape:** Before the sparse solver iteration, we overwrite the surface nodes with the prescribed $q_D = X_D + u_i$. In other words, we "push" them to the target position based on the image data.

2. **Hold them fixed in the Newton step:** During the solve we enforce

$$\Delta q_D = 0,$$

so the solver never tries to move those nodes again. All of the incremental updates go into Δq_F, and the K_{FD} coupling term drops out. So because we've already updated the boundary positions from the image data, their "present update" in the solver step is zero.

3.5 Note: The Role of External Forces in All-Dirichlet Systems

In the special case where external forces are only applied to nodes that are already constrained by Dirichlet conditions, these forces do not directly influence the solution of the free nodes ($\mathbf{q}_F$). Instead, they contribute to the **reaction forces** at the boundary.
In physics, this force can be expressed using an augmented potential function with Lagrange multipliers $\boldsymbol{\lambda}$, which represent the reaction forces:

$$\Pi(\mathbf{q}, \boldsymbol{\lambda}) = U(k, \mathbf{q}) + \mathbf{F}_{\text{ext}}^{\top}\mathbf{q} + \boldsymbol{\lambda}^{\top}\big(\mathbf{q}_D - \mathbf{X}_D - \mathbf{u}_i\big).$$

Solving the saddle-point system derived from Π shows that if $\mathbf{F}_{F,\text{ext}} = \mathbf{0}$ (i.e., no external forces on free nodes), the linear system for the free DOFs degenerates to:

$$\boxed{\mathbf{K}_{FF}\,\Delta\mathbf{q}_F = -\,\mathbf{g}_F}$$

In this specific scenario, the external forces $\mathbf{F}_{D,\text{ext}}$ are entirely balanced by the reaction forces $\boldsymbol{\lambda}$.

3.6 Linear Subproblem and Final Output

- **Linear Subproblem**: In the general case, the equation to solve at each iteration is

  $$\mathbf{K}_{FF}\,\Delta\mathbf{q}_F = -\big[\mathbf{g}_F + \mathbf{F}_{F,\text{ext}}\big],$$

  which is efficiently solved using the Conjugate Gradient method.

- **Simulated Displacement**: After convergence, the final displacement field is

  $$\mathbf{u}_{\text{sim}} = \mathbf{q} - \mathbf{X}.$$

- **Reaction Forces (Optional)**: The reaction forces at the Dirichlet boundary can be calculated for analysis:

  $$\boldsymbol{\lambda} = -(\mathbf{g}_D(\mathbf{q}) + \mathbf{F}_{D,\text{ext}})$$

4 Pipeline

4.0.1 Questions

1. the output from fem
   2. the iteration is slow maybe we need Liu's method to speed up
   3. taichi decorator kernel and func maybe different!!!
   4. Use Unet now. Compare the results with the image registration results.