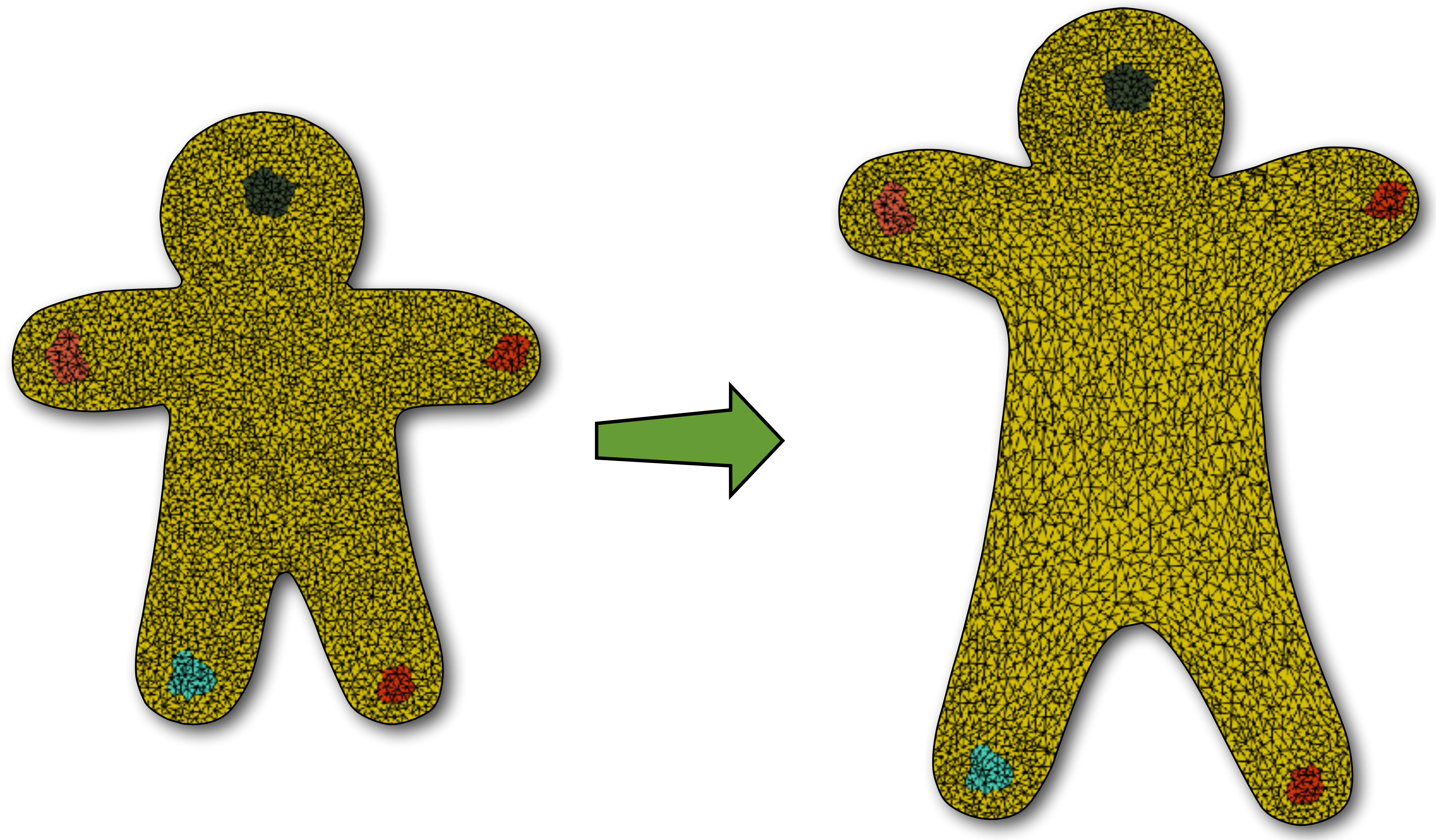
Geometric Modeling
Assignment 5: Shape Deformation

Acknowledgements: Olga Diamanti, Julian Panetta, Zhongshi Jiang

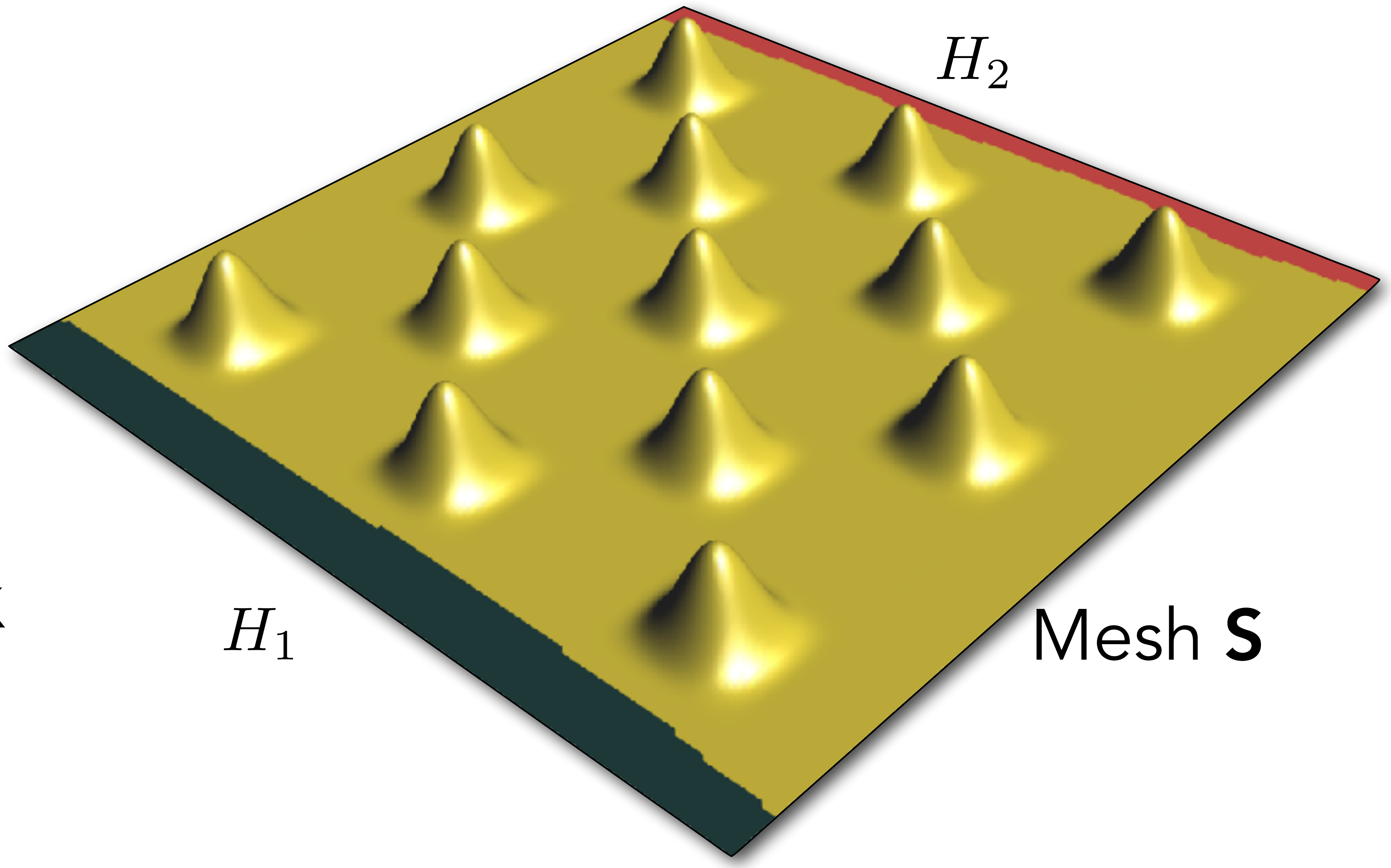
University of Victoria | Computer Science

Shape Deformation



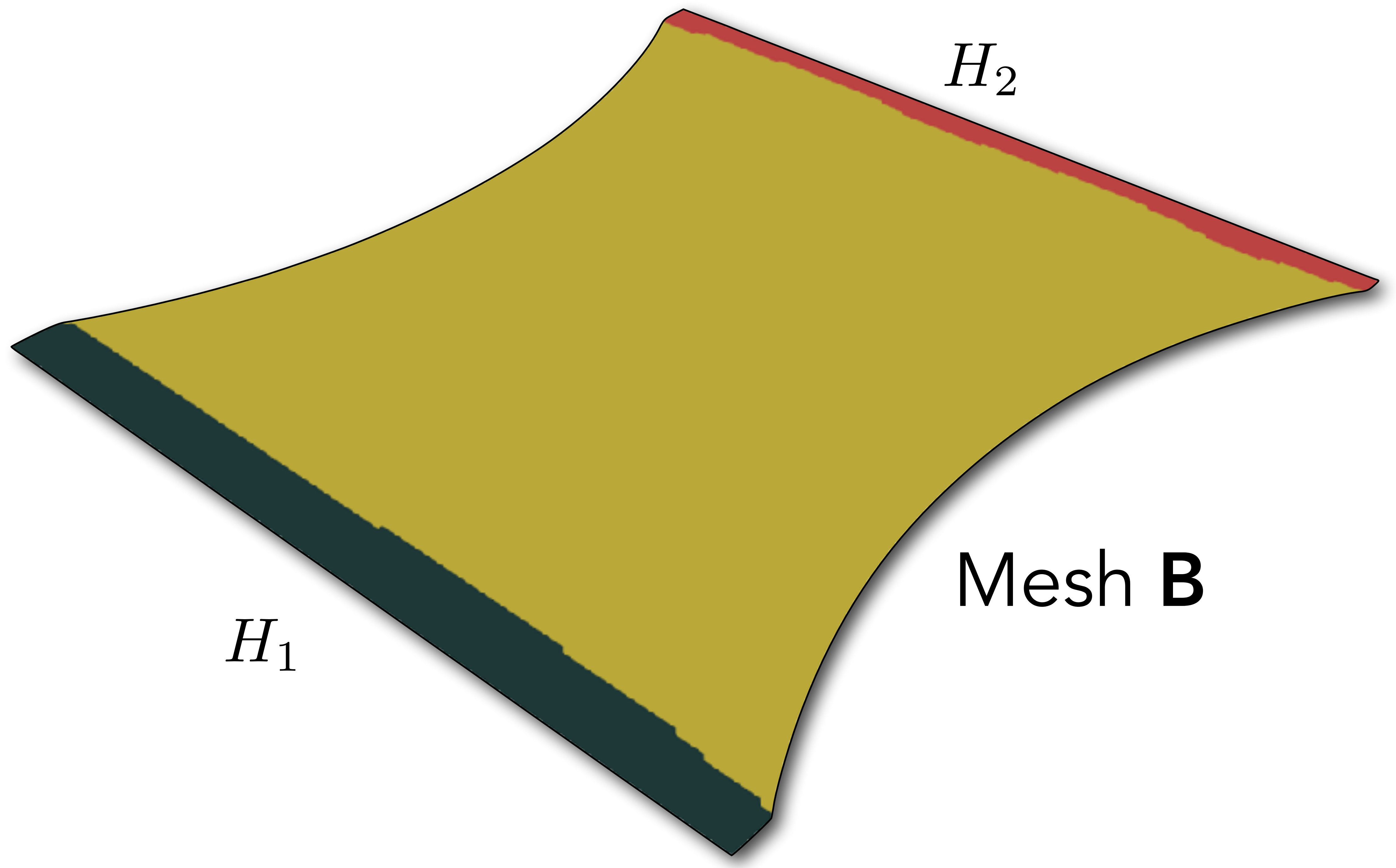
Step 1: Select and Deform Handle Regions

- Draw vertex selection with mouse
- Move one handle H at a time to the deform mesh
- Leave some handles undeformed to fix vertices.
- Code provided for the picking/dragging



Step 2: Smooth Mesh

- Remove high-frequency details from unconstrained vertices.
- This smooth mesh will be deformed; details are added back afterward
- Smooth by solving a bi-laplacian system (minimize the Laplacian Energy)

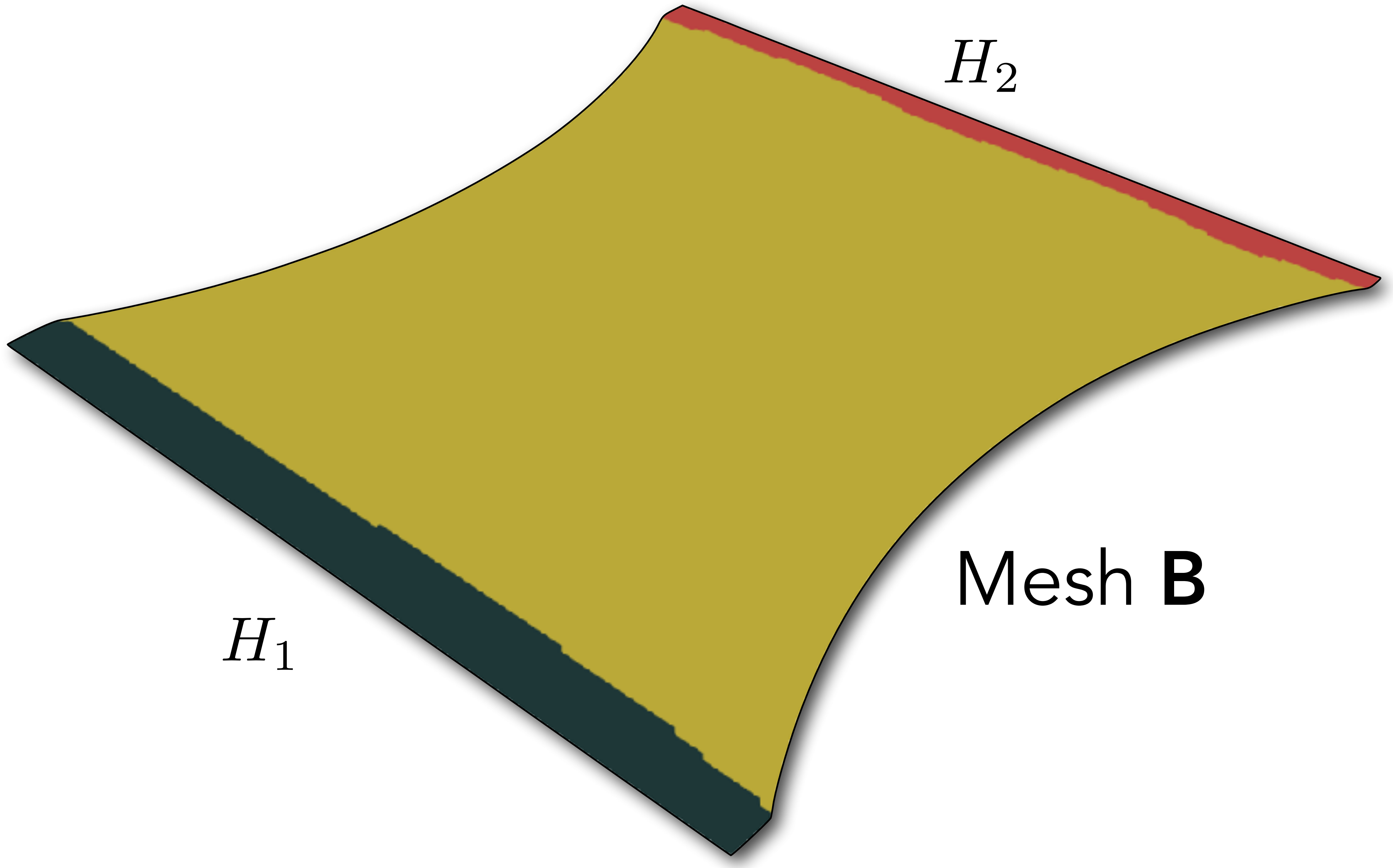


Step 2: Smooth Mesh

- Remove high-frequency details from unconstrained vertices.
- This smooth mesh will be deformed; details are added back afterward
- Smooth by solving a bi-laplacian system (minimize the Laplacian Energy):

$$\min_{\mathbf{v}} \mathbf{v}^T \mathbf{L}_\omega \mathbf{M}^{-1} \mathbf{L}_\omega \mathbf{v}$$

$$\text{s.t.} \quad \mathbf{v}_{H_i} = \underline{o_{H_i}} \;\; \forall i \xleftarrow{\text{Original vertex positions of handles}}$$



Original vertex positions of handles

University of Victoria | Computer Science

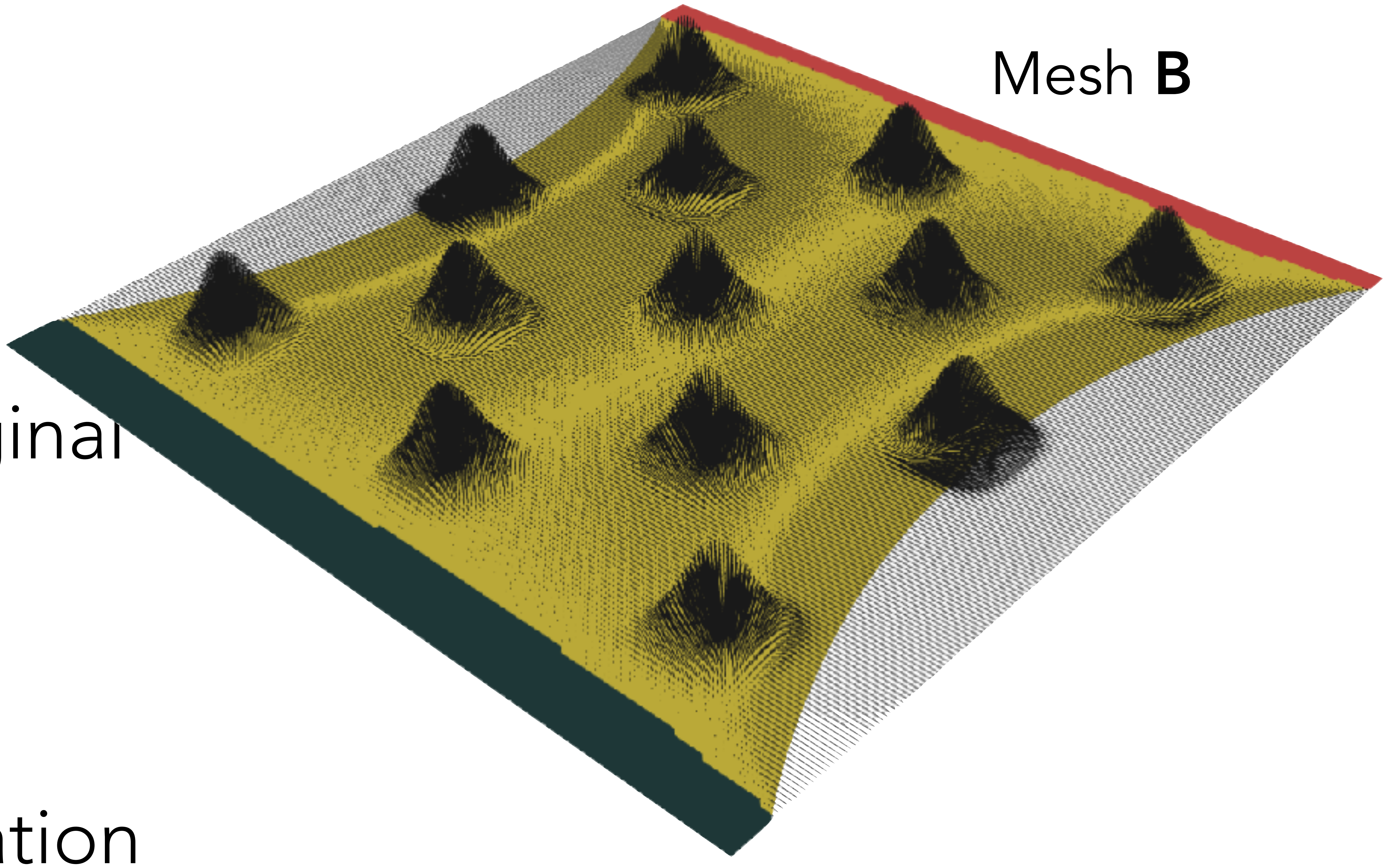
Step 3: Encode Displacements

- Compute the per-vertex displacements from B to S:

$$\mathbf{d}_i = \mathbf{v}_i^S - \mathbf{v}_i^B$$

- These represent the **details** of the original surface.

- We will use these to add back (transformed) details after the deformation



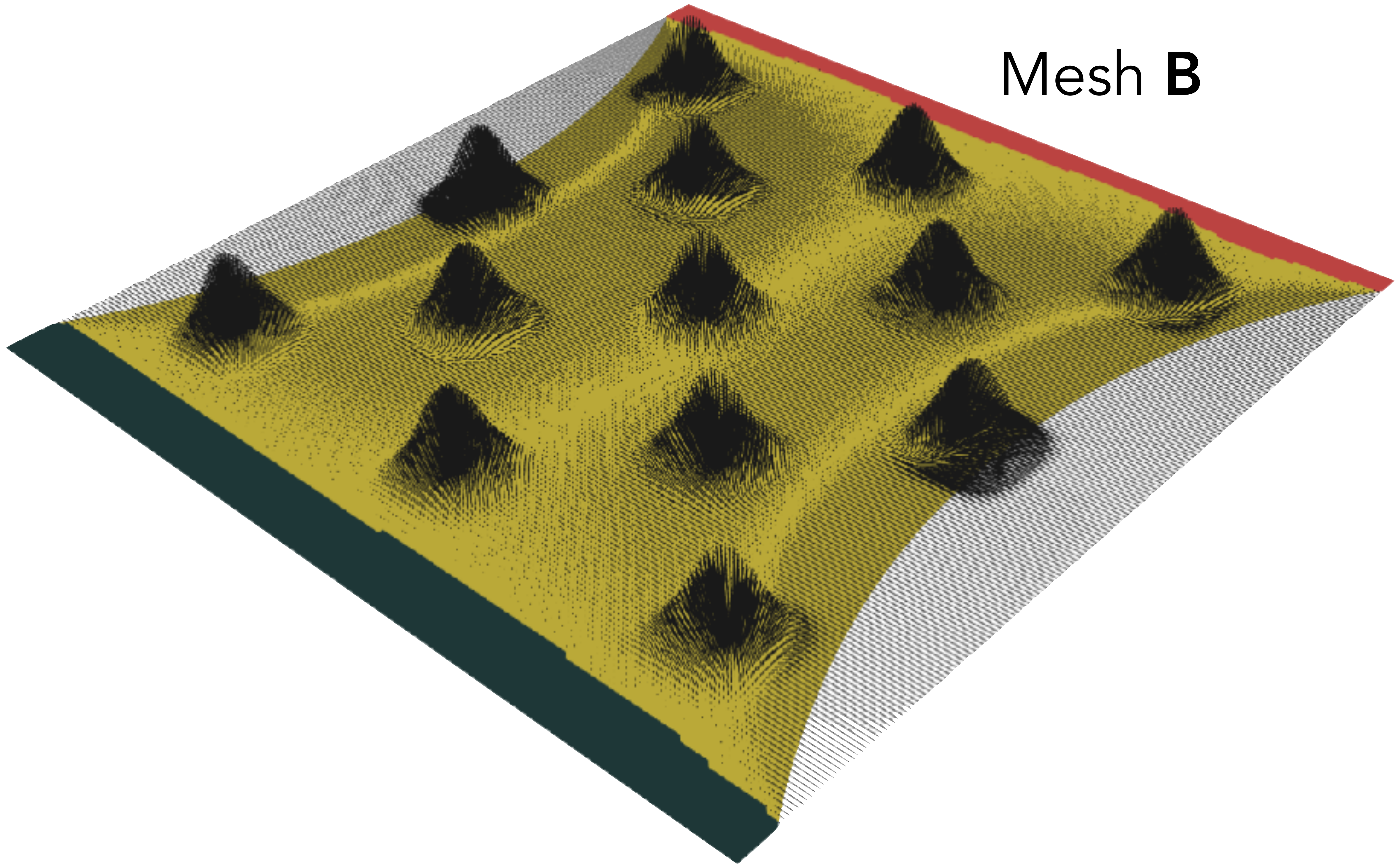
Step 3: Encode Displacements

$$\mathbf{d}_i = \mathbf{v}_i^S - \mathbf{v}_i^B$$

- We want these details to rotate with Mesh **B** as it is later deformed into Mesh **B'**

- To do this, we express the displacements in a **local frame on B,** which is then rotated to align with **B'**

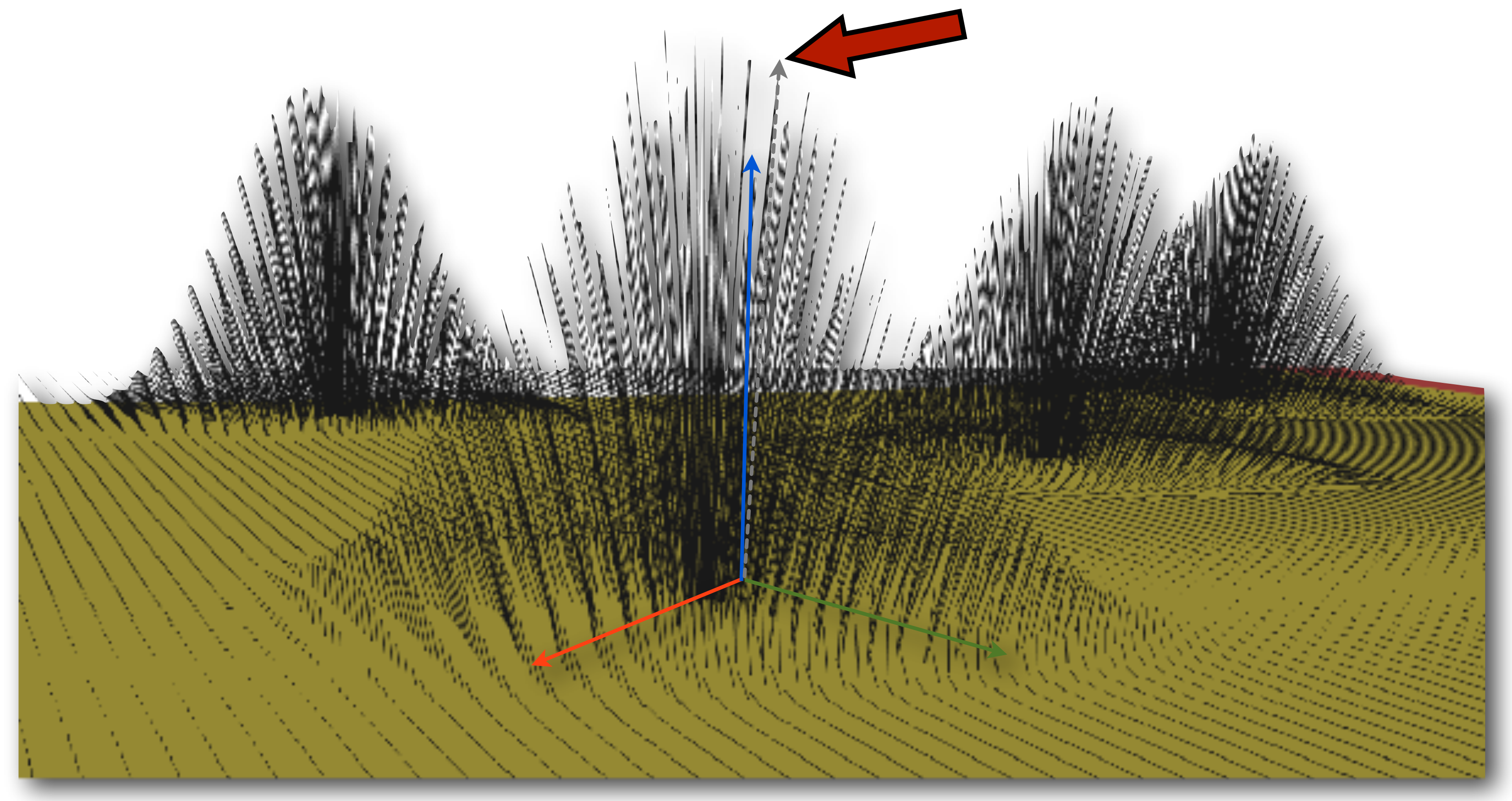
- We just need to define the basis vectors for this frame…

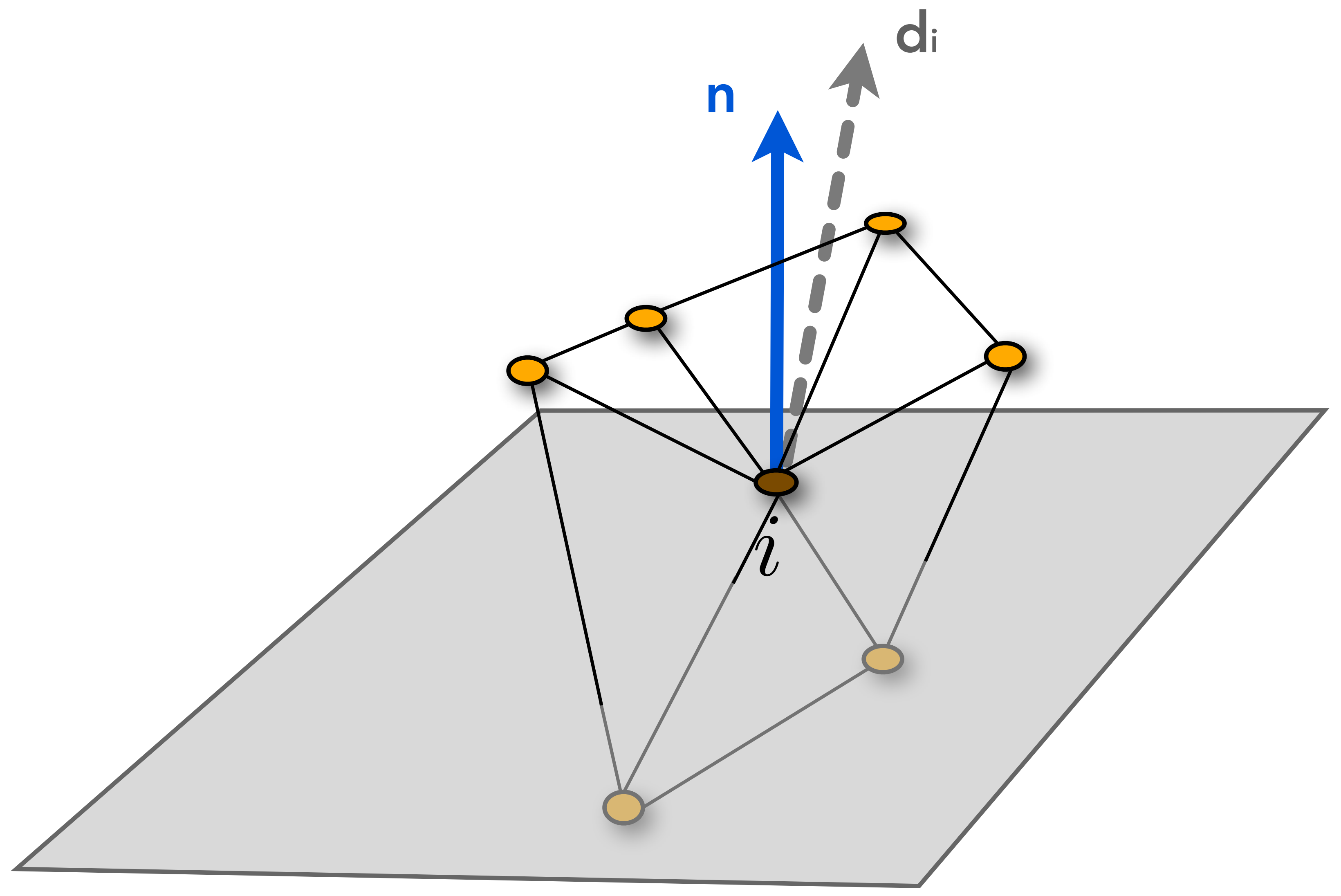


Step 3: Encode Displacements

- Construct the local frame for vertex i:

    1. Calculate normal $\mathbf{n}_i$ (for surface **B**)

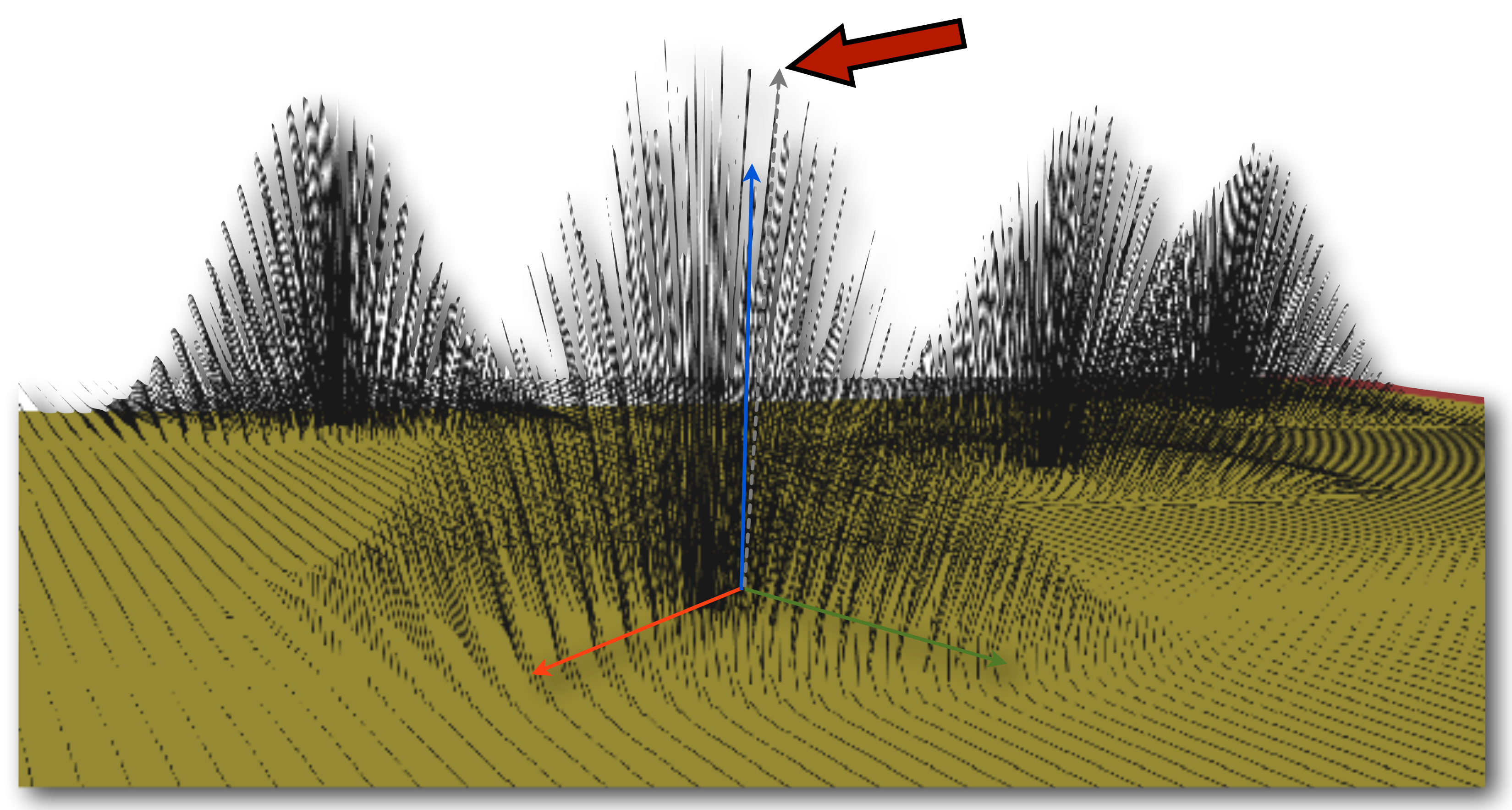


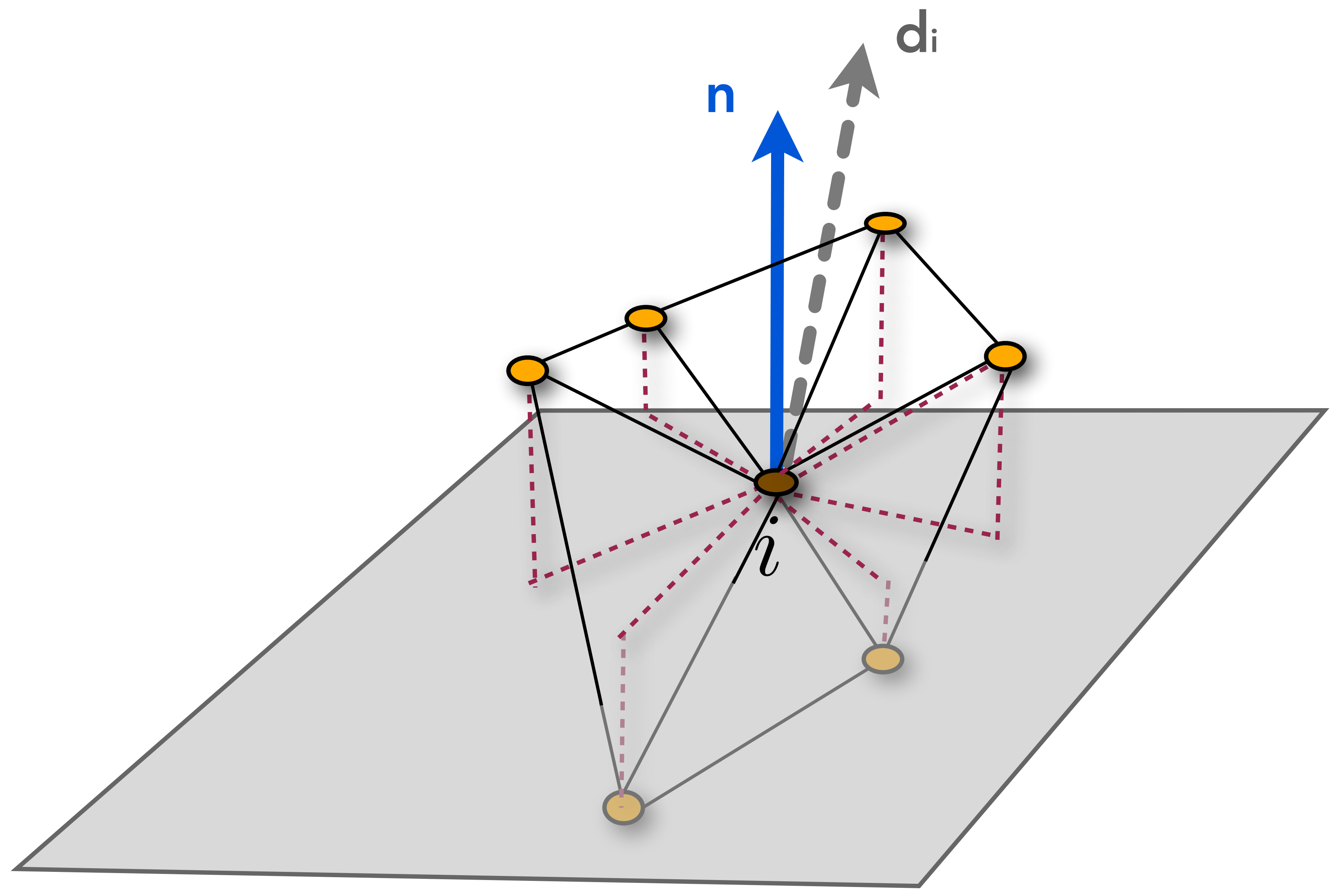


University of Victoria | Computer Science

Step 3: Encode Displacements

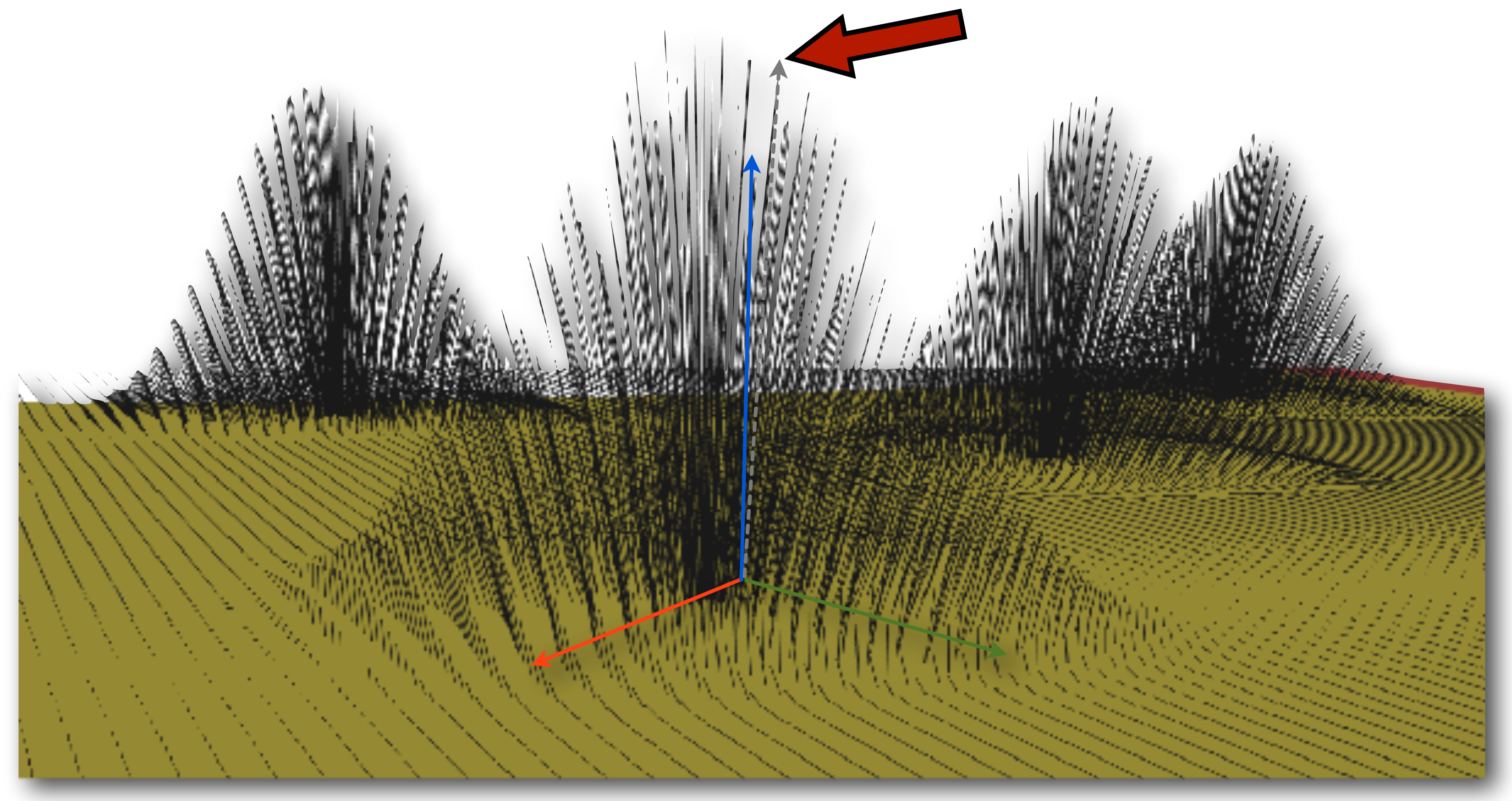
- Construct the local frame for vertex i:
  1. Calculate normal $\mathbf{n}_i$ (for surface **B**)
  2. Project all neighboring vertices to the tangent plane (perpendicular to $\mathbf{n}_i$)

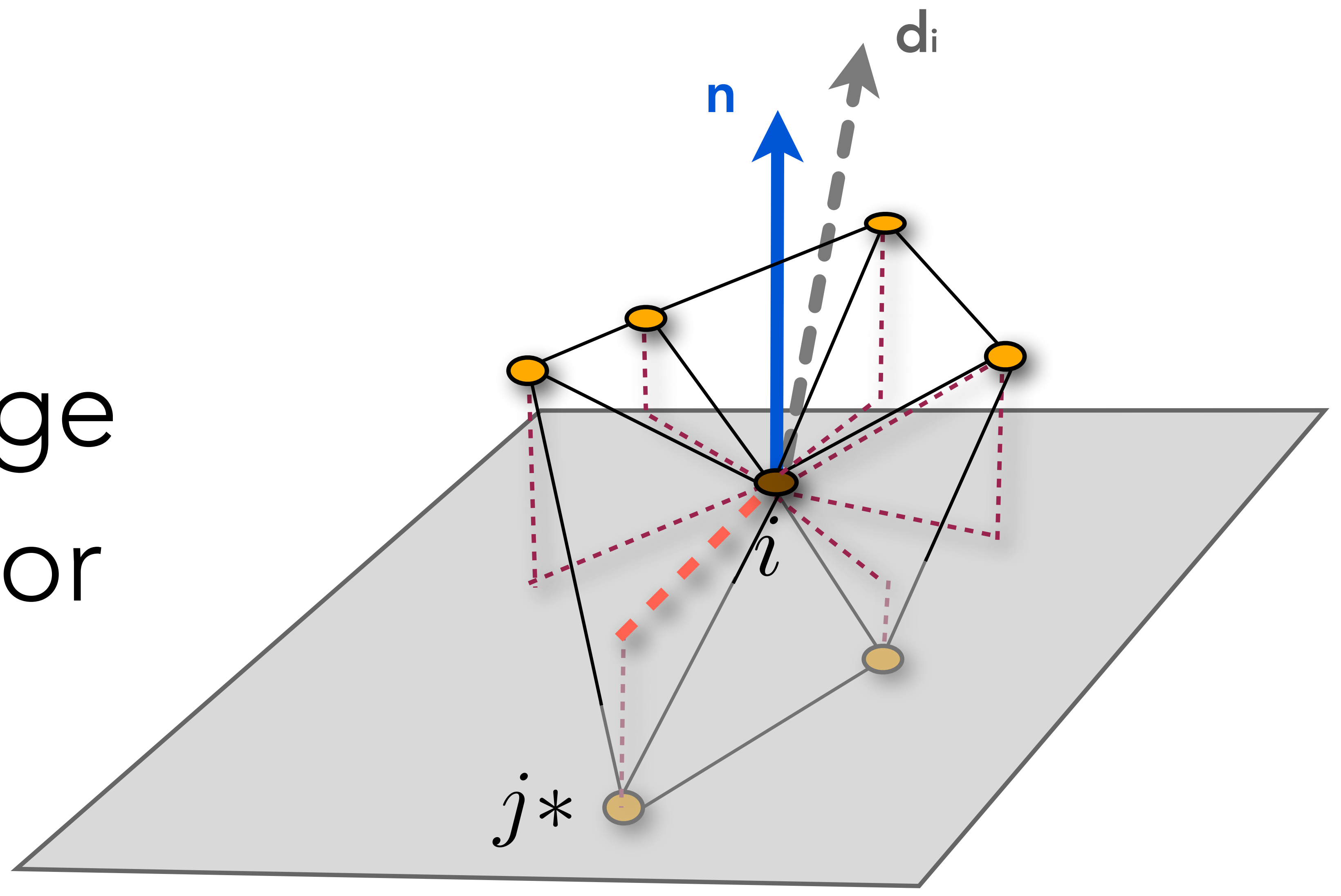




Step 3: Encode Displacements

- Construct the local frame for vertex i:
    1. Calculate normal $\mathbf{n_i}$ (for surface **B**)
    2. Project all neighboring vertices to the tangent plane (perpendicular to $\mathbf{n_i}$)
    3. Find neighbor j* for which projected edge (i, j) is longest. Normalize this edge vector and call it $\mathbf{x_i}$.





University of Victoria | Computer Science

Step 3: Encode Displacements

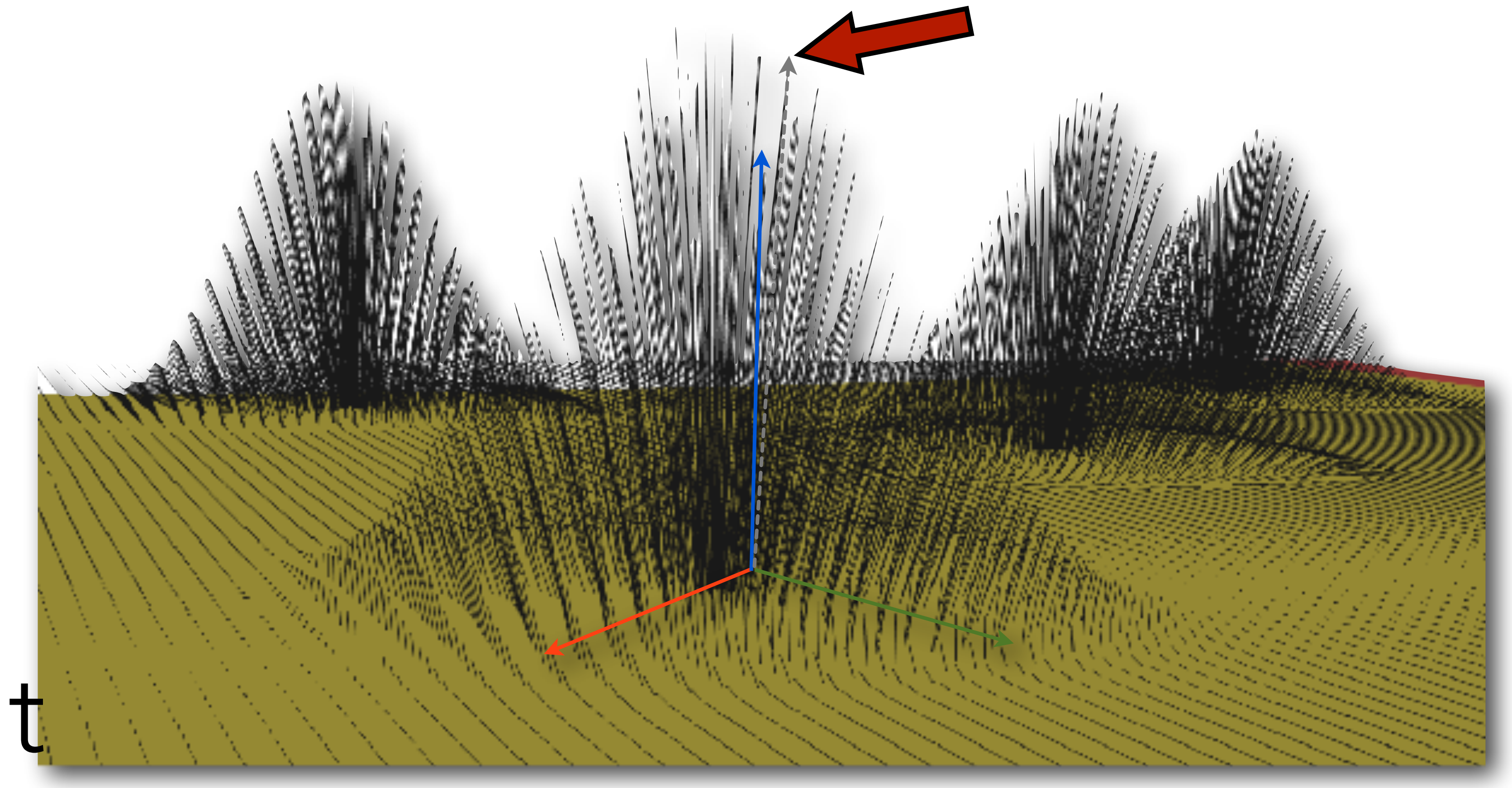
- Construct the local frame for vertex i:

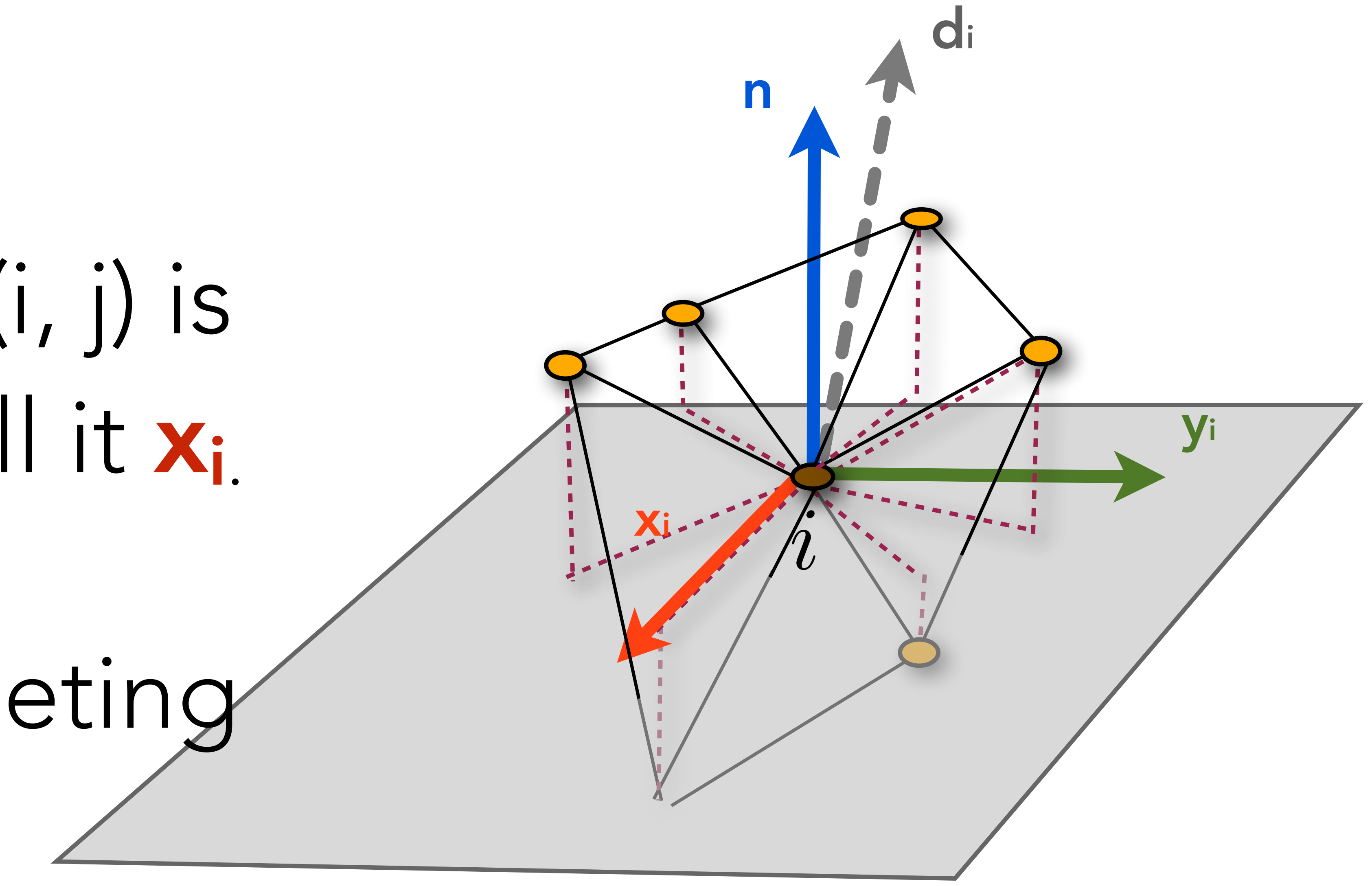
  1. Calculate normal $\mathbf{n}_i$ (for surface **B**)

  2. Project all neighboring vertices to the tangent plane (perpendicular to $\mathbf{n}_i$)

  3. Find neighbor j* for which projected edge (i, j) is longest. Normalize this edge vector and call it $\mathbf{x}_i$.

  4. Construct $\mathbf{y}_i$ using the cross product, completing orthonormal frame ($\mathbf{x}_i$, $\mathbf{y}_i$, $\mathbf{n}_i$)





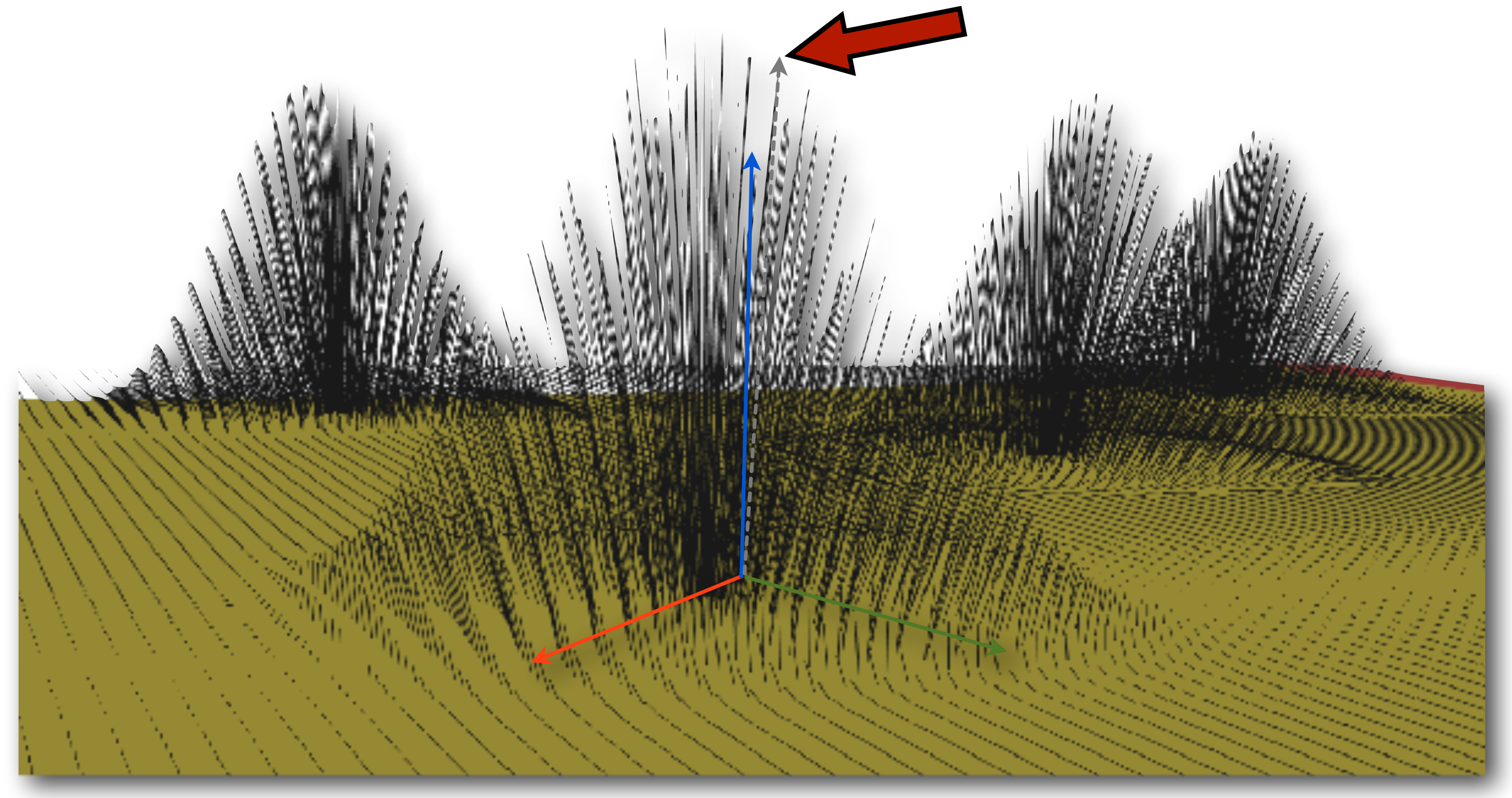
University of Victoria | Computer Science

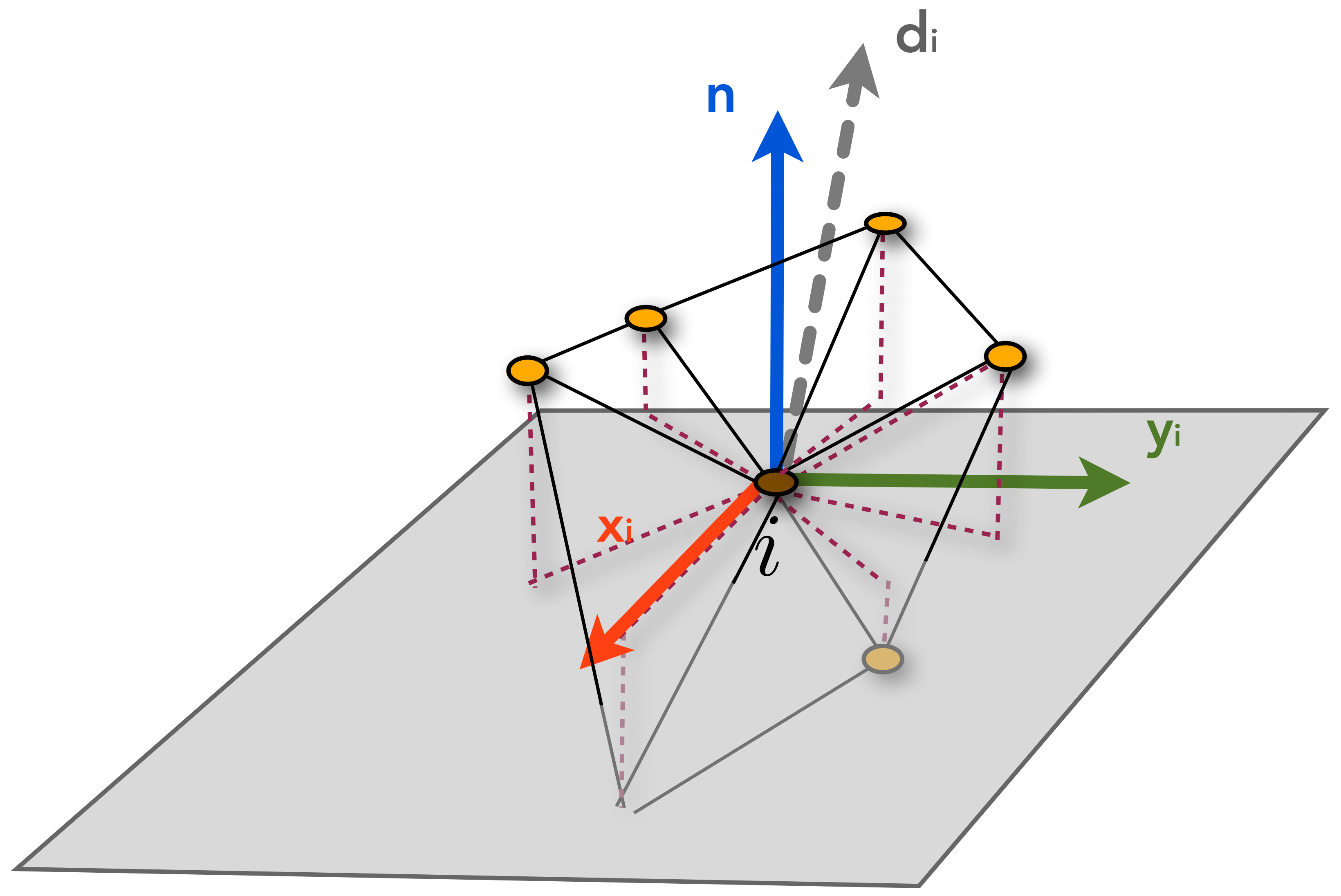
Step 3: Encode Displacements

- Decompose the displacement vectors in the frame's basis:

$$\mathbf{d}_i = d_i^{\mathbf{x}} \mathbf{x}_i + d_i^{\mathbf{y}} \mathbf{y}_i + d_i^{\mathbf{n}} \mathbf{n}_i$$

- (The basis is orthonormal, so you can do this just with inner products.)





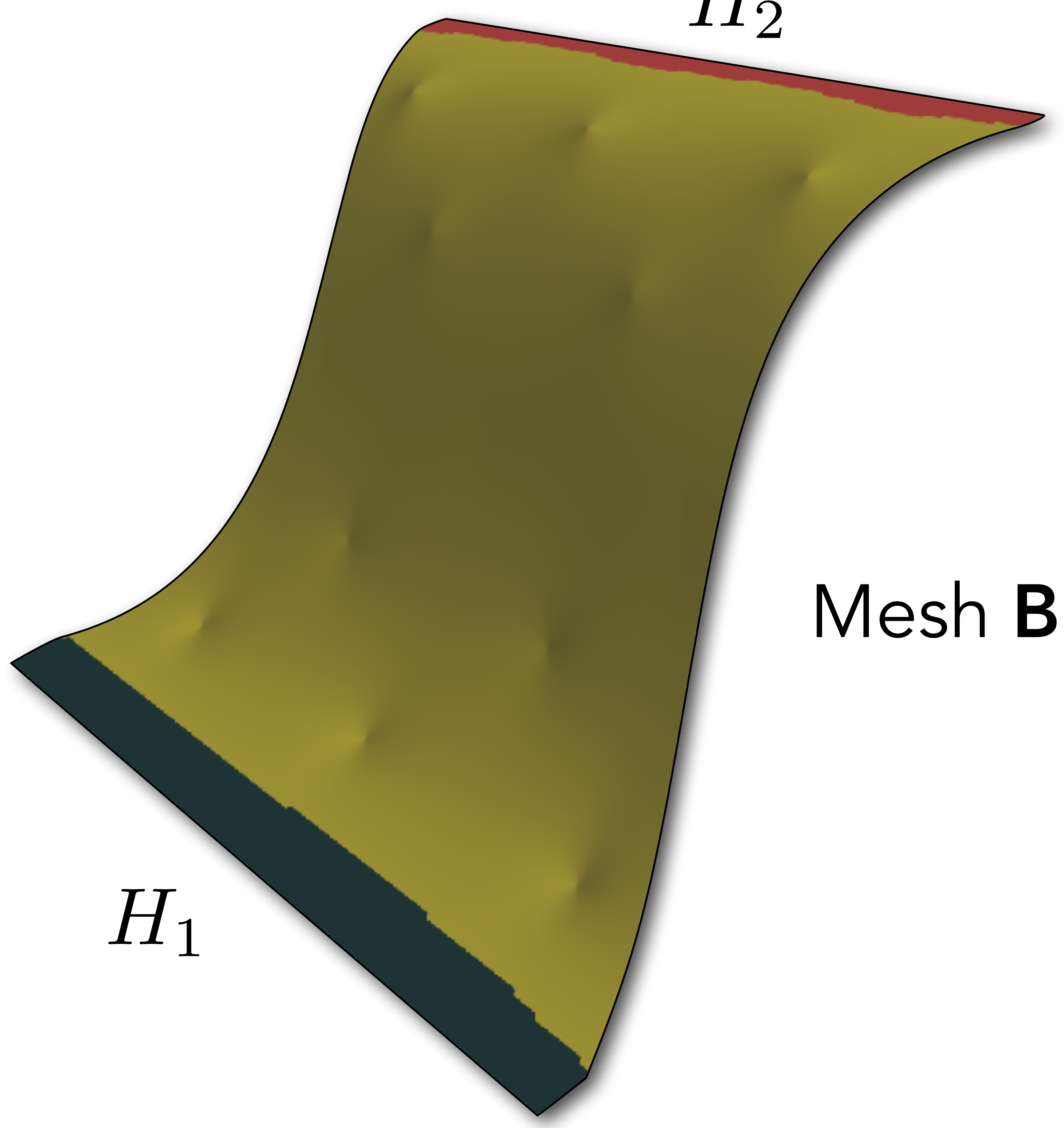
University of Victoria | Computer Science

Step 4: Deform **B**

- User manipulates the handles
- You solve for the deformed smooth mesh, **B′**
- Solve bi-laplacian system again, using the new handle position as constraints:

$$\min_{\mathbf{v}} \mathbf{v}^T \mathbf{L}_\omega \mathbf{M}^{-1} \mathbf{L}_\omega \mathbf{v}$$
$$\text{s.t.} \quad \mathbf{v}_{H_i} = \underline{t(o_{H_i})} \; \forall i$$

Transformed vertex positions at handles.

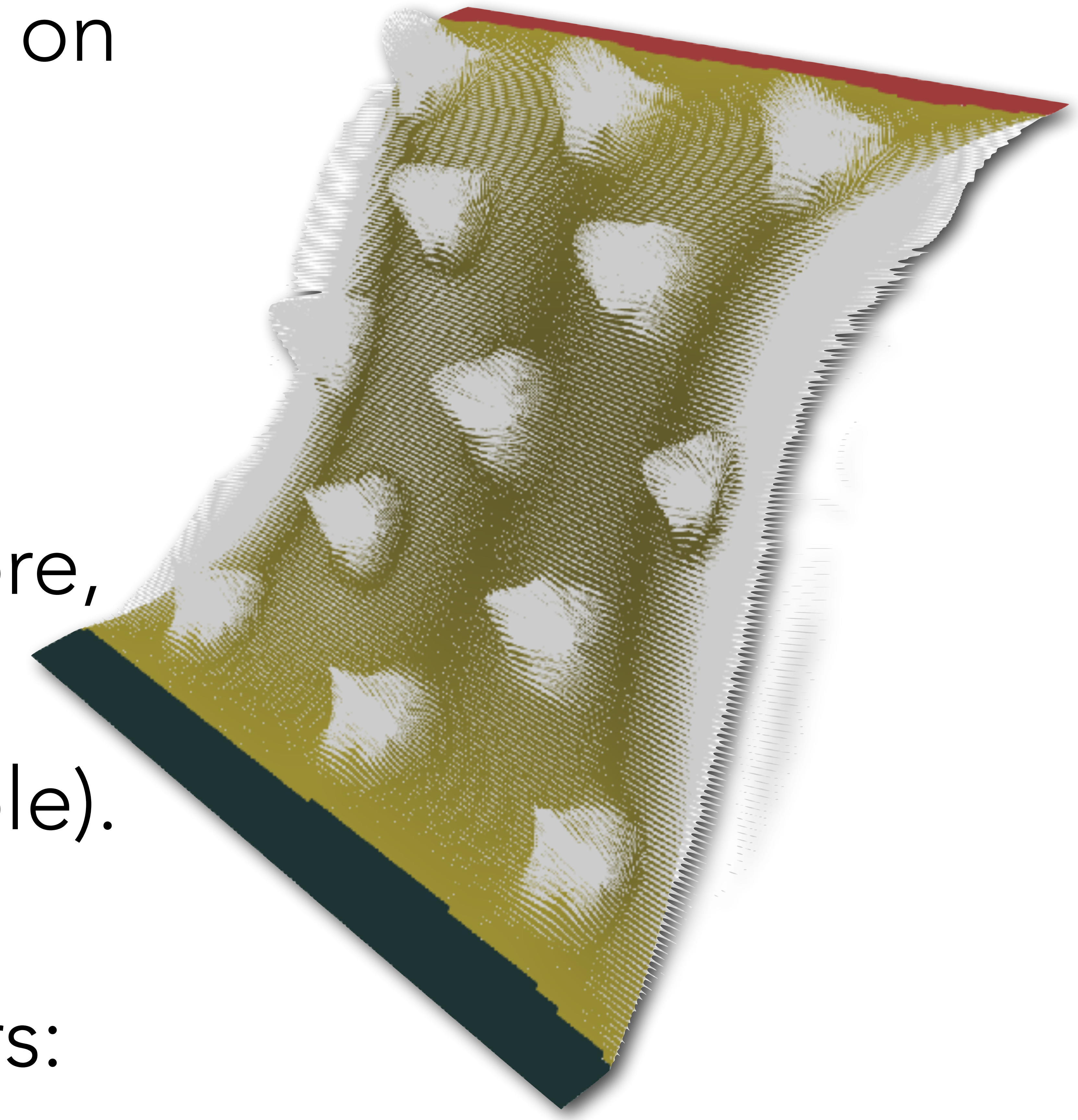


University of Victoria | Computer Science

Step 5: Add Transformed Detail

- Compute for each vertex v_i the new local frame on **B′**
  - Calculate normal $\mathbf{n}_i'$ (for surface **B′**)
  - Compute new frame basis ($\mathbf{x}_i'$, $\mathbf{y}_i'$, $\mathbf{n}_i'$) as before, but using the **same edge** (i, j*) as chosen for **B** (so frames are compatible).
- Construct the transformed displacement vectors:

$$\mathbf{d}_i' = d_i^{\mathbf{x}}\mathbf{x}_i' + d_i^{\mathbf{y}}\mathbf{y}_i' + d_i^{\mathbf{n}}\mathbf{n}_i'$$



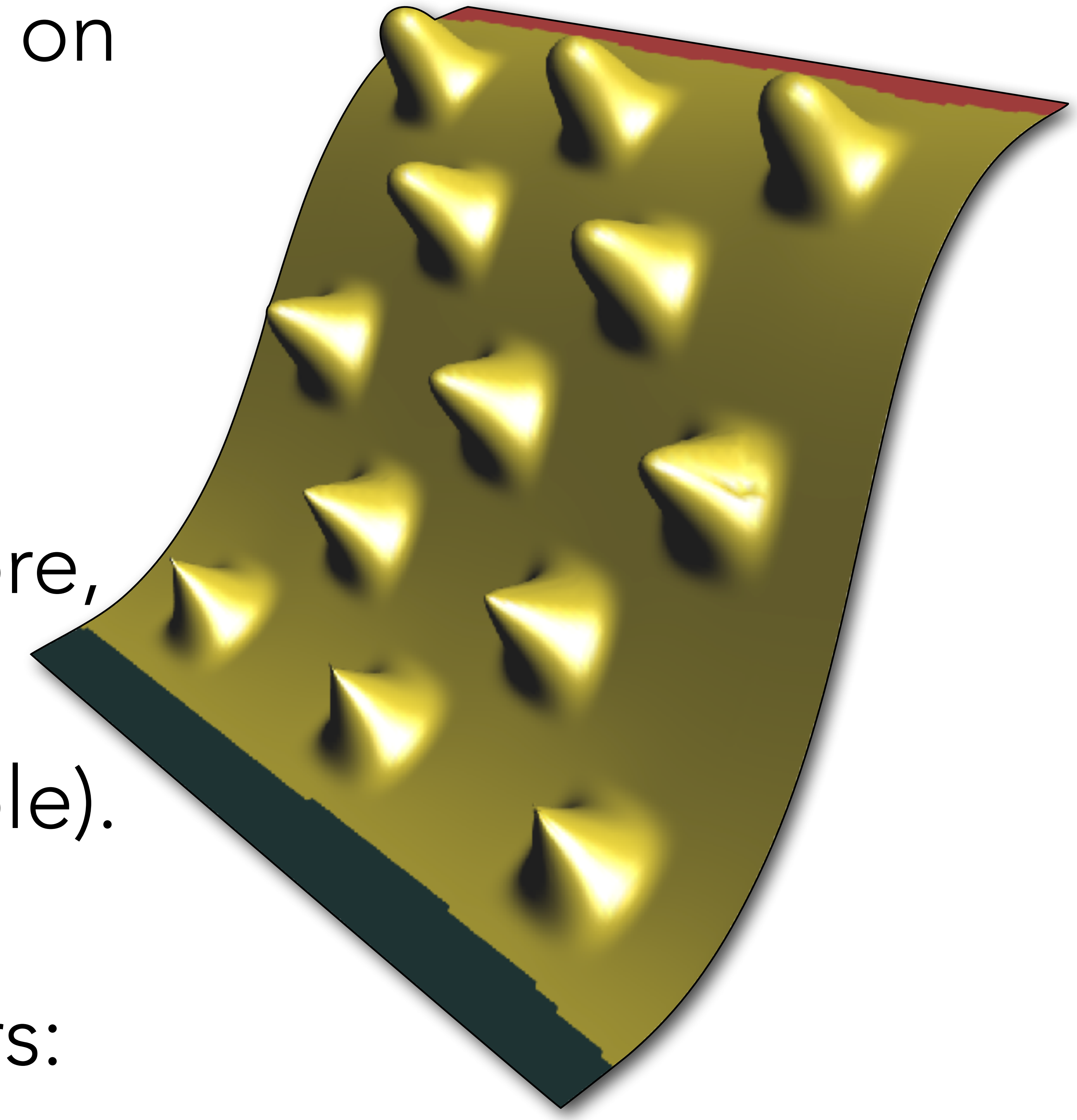
University of Victoria | Computer Science

Step 5: Add Transformed Detail

- Compute for each vertex v_i the new local frame on **B'**
  - Calculate normal $\mathbf{n}_i'$ (for surface **B'**)
  - Compute new frame basis ($\mathbf{x}_i'$, $\mathbf{y}_i'$, $\mathbf{n}_i'$) as before, but using the **same edge** (i, j*) as chosen for **B** (so frames are compatible).
- Construct the transformed displacement vectors:

$$\mathbf{d}_i' = d_i^{\mathbf{x}} \mathbf{x}_i' + d_i^{\mathbf{y}} \mathbf{y}_i' + d_i^{\mathbf{n}} \mathbf{n}_i'$$

- Add add them to **B'** in to form **S'**



Solving the Bi-Laplacian System

$$\min_{\mathbf{v}} \mathbf{v}^T \mathbf{L}_\omega \mathbf{M}^{-1} \mathbf{L}_\omega \mathbf{v}$$

$$\text{s.t.} \quad \mathbf{v}_{H_i} = o_{H_i} \ \forall i$$

- The positions of handle vertex must be imposed as **hard constraints**

- This can be done with the row/column removal trick from last assignment:

$$A = \mathbf{L}_\omega \mathbf{M}^{-1} \mathbf{L}_\omega = \begin{bmatrix} A_{ff} & A_{fc} \\ A_{cf} & A_{cc} \end{bmatrix}, \quad \mathbf{b} = \mathbf{0} = \begin{bmatrix} b_f \\ b_c \end{bmatrix}$$

Instead of $A\mathbf{v} = \mathbf{b}$, solve $\quad A_{ff}\mathbf{v}_f = \mathbf{b} - A_{fc}\mathbf{v}_c$

- (So an efficient, sparse Cholesky factorization can be used)

Pre-factoring the System

```
from sksparse.cholmod import cholesky
factor = cholesky(A)
x = factor(b)
```

- sp.sparse.linalg.spsolve() is **slow**. After the factorization is computed, solving for different vectors is fast.
- Factorization needs to be recomputed only when the matrix A_{ff} changes (when new handles are drawn).
- Additional Tricks: vectorize the code with numpy.einsum, or use numba.jit