Geometric Modeling
Assignment 5: Shape Deformation

Acknowledgements: Olga Diamanti, Julian Panetta
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):

\[
\min_{v} v^T L_\omega M^{-1} L_\omega v \\
\text{s.t. } v_{H_i} = o_{H_i} \ \forall i
\]
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

\[ d_i = v^S_i - v^B_i \]

• 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 $n_i$ (for surface $B$)
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 \( n_i \) (for surface \( B \))

2. Project all neighboring vertices to the tangent plane (perpendicular to \( n_i \))

3. Find neighbor \( j^* \) for which projected edge \( (i, j) \) is longest. Normalize this edge vector and call it \( x_i \).
Step 3: Encode Displacements

• Construct the local frame for vertex i:

1. Calculate normal $n_i$ (for surface $B$)

2. Project all neighboring vertices to the tangent plane (perpendicular to $n_i$)

3. Find neighbor $j^*$ for which projected edge $(i, j)$ is longest. Normalize this edge vector and call it $x_i$.

4. Construct $y_i$ using the cross product, completing orthonormal frame ($x_i, y_i, n_i$)
Step 3: Encode Displacements

• Decompose the displacement vectors in the frame’s basis:

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

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

$$\min_v v^T L_{\omega} M^{-1} L_{\omega} v$$

s.t. $v_{H_i} = t(o_{H_i}) \quad \forall i$

Transformed vertex positions at handles.
Step 5: Add Transformed Detail

- Compute for each vertex $v_i$ the new local frame on $B'$
- Calculate normal $n'_i$ (for surface $B'$)
- Compute new frame basis $(x'_i, y'_i, n'_i)$ as before, but using the same edge $(i, j^*)$ as chosen for $B$ (so frames are compatible).
- Construct the transformed displacement vectors:
  $$d'_i = d^x_i x'_i + d^y_i y'_i + d^m_i n'_i$$
Step 5: Add Transformed Detail

• Compute for each vertex $v_i$ the new local frame on $B'$

• Calculate normal $n_i'$ (for surface $B'$)

• Compute new frame basis $(x_i', y_i', n_i')$ as before, but using the same edge (i, j*) as chosen for $B$ (so frames are compatible).

• Construct the transformed displacement vectors:
  $$d_i' = d_i^x x_i' + d_i^y y_i' + d_i^n n_i'$$

• Add add them to $B'$ in to form $S'$
Solving the Bi-Laplacian System

\[
\min_v v^T L_\omega M^{-1} L_\omega v
\]

s.t. \( v_{H_i} = o_{H_i} \ \forall i \)

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

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

\[
A = L_\omega M^{-1} L_\omega = \begin{bmatrix} A_{ff} & A_{fc} \\ A_{cf} & A_{cc} \end{bmatrix}, \quad b = 0 = \begin{bmatrix} b_f \\ b_c \end{bmatrix}
\]

Instead of \( Av = b \), solve \( A_{ff} v_f = b - A_{fc} v_c \)

- (So an efficient, sparse Cholesky factorization can be used)
Pre-factoring the System

- `solver.compute()` 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).

- For real-time performance—and full credit on the assignment—you must recompute the factorization only when necessary (not while the user is interactively transforming handles).

```c++
Eigen::SimplicialCholesky<SparseMatrixType, Eigen::RowMajor> solver;
solver.compute(A_ff); // Factorize the free part of the system
```