Shape Modeling and Geometry Processing

Exercise 5 - Shape Deformation

INTERACTIVE GEOMETRY LAB

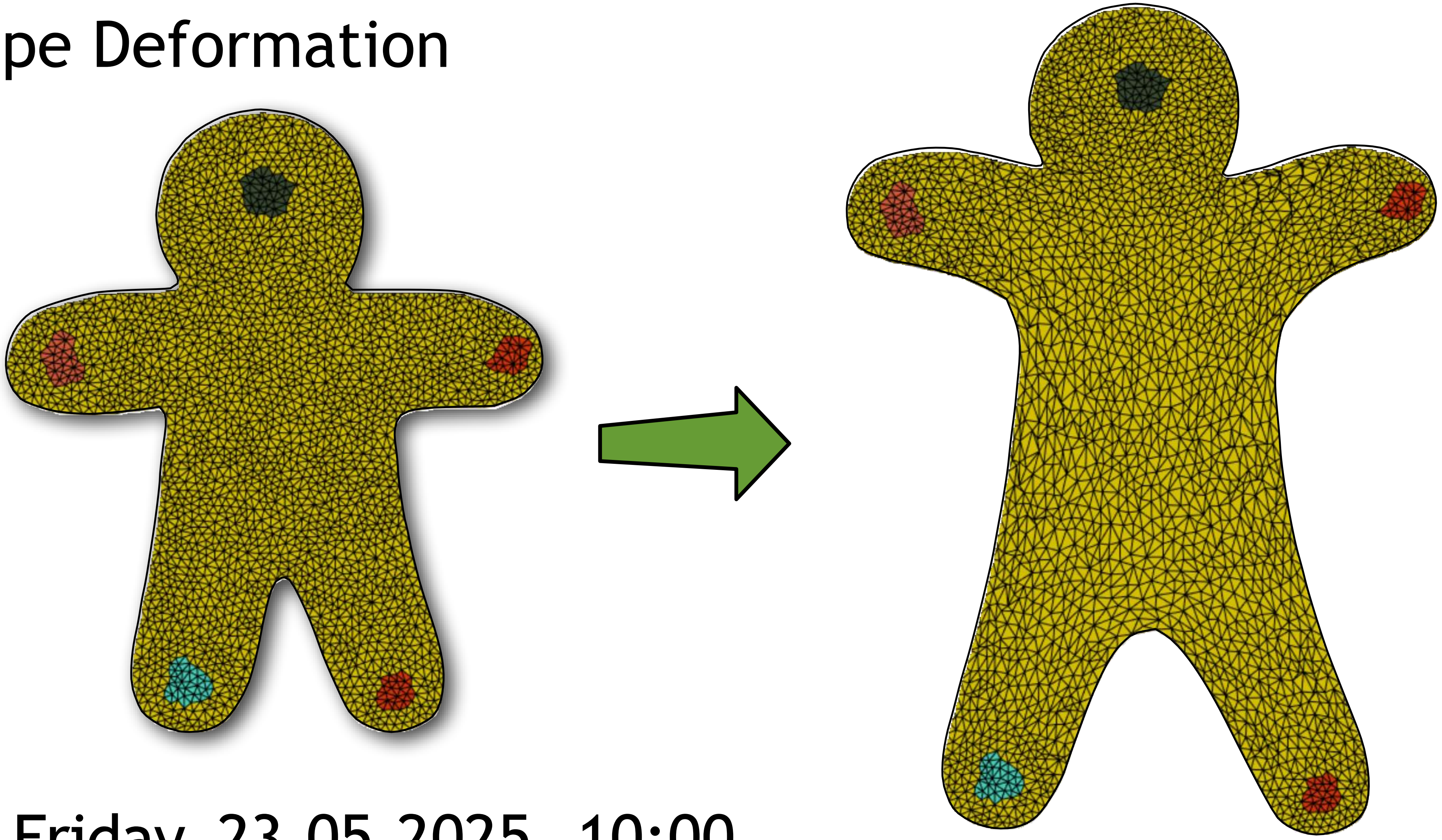
Annika Oehri

May 9, 2025

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

This exercise

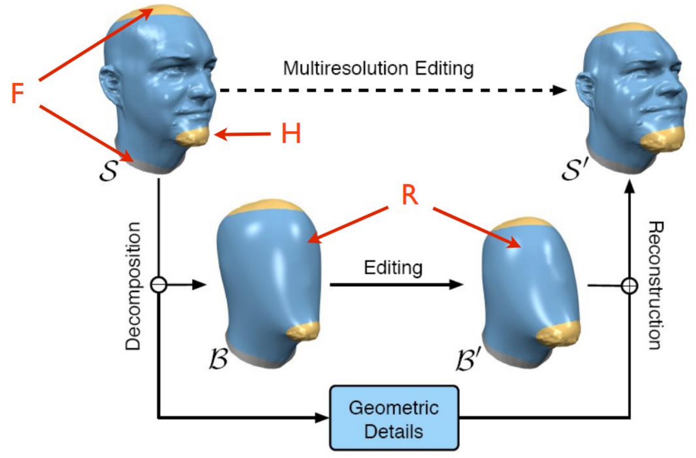
- Topic: Shape Deformation



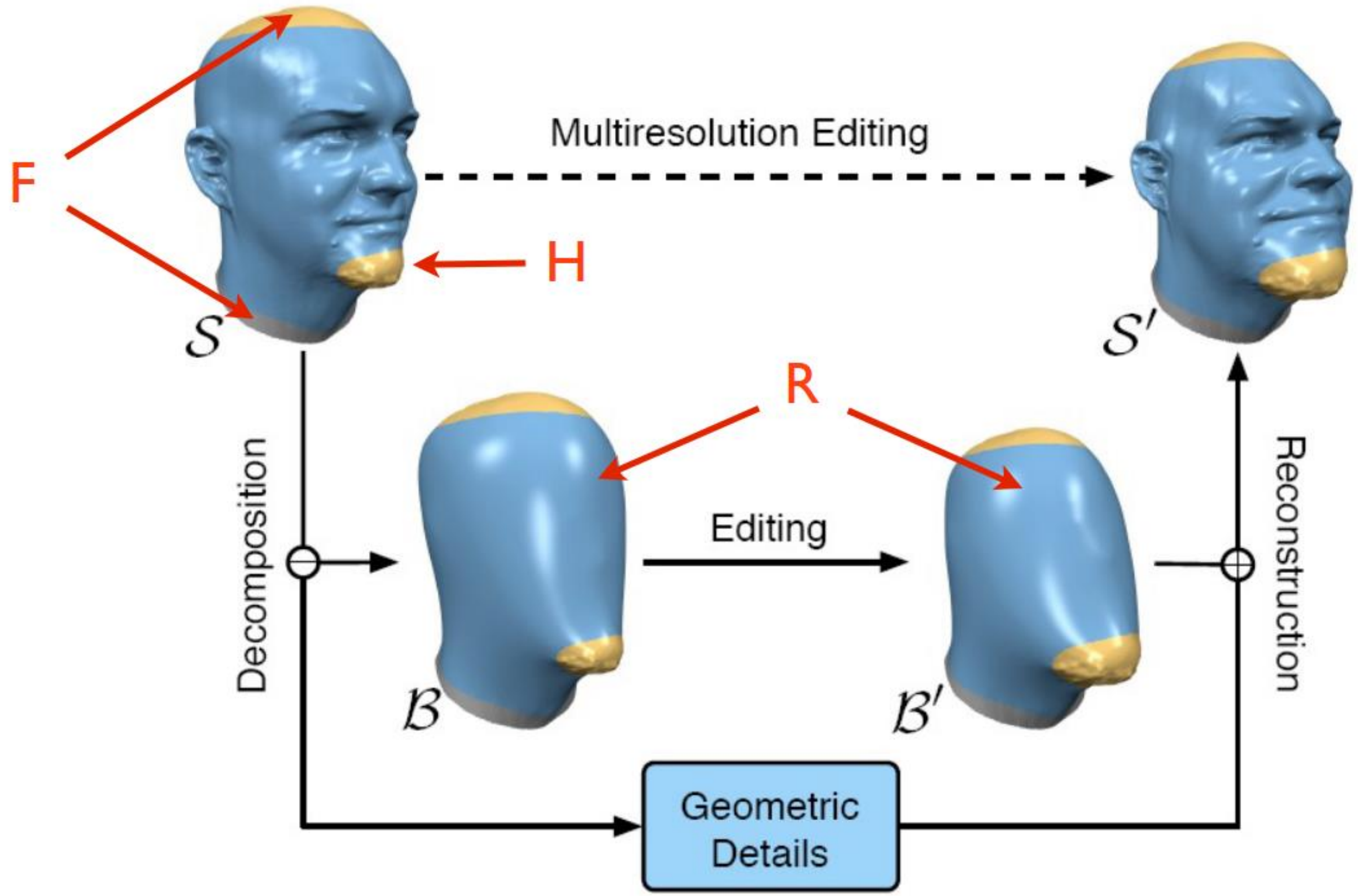
- Deadline: Friday, 23.05.2025, 10:00

Algorithm Overview

1. Select handle regions
2. Smoothing with handle regions fixed
3. Encode high-frequency information as local displacements
4. Deform the smoothed shape (by manipulating the handles)
5. Add local (high-freq) details back to the deformed shape

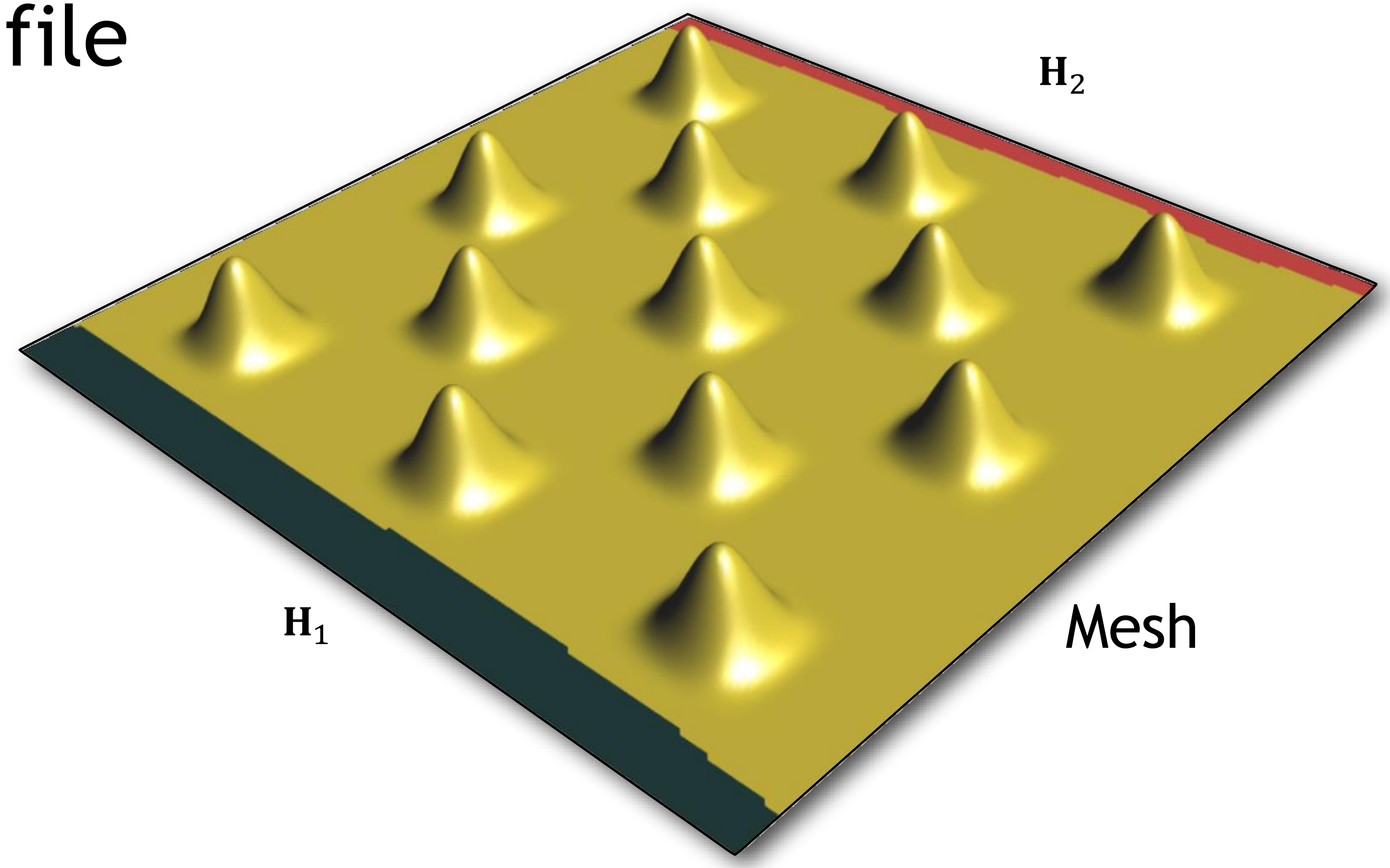


Algorithm Overview



Step 1: Select Handle Regions

- Select with mouse or load from file
- Move one handle at a time by clicking and dragging
- Rest of the handles stay fixed
- Code provided

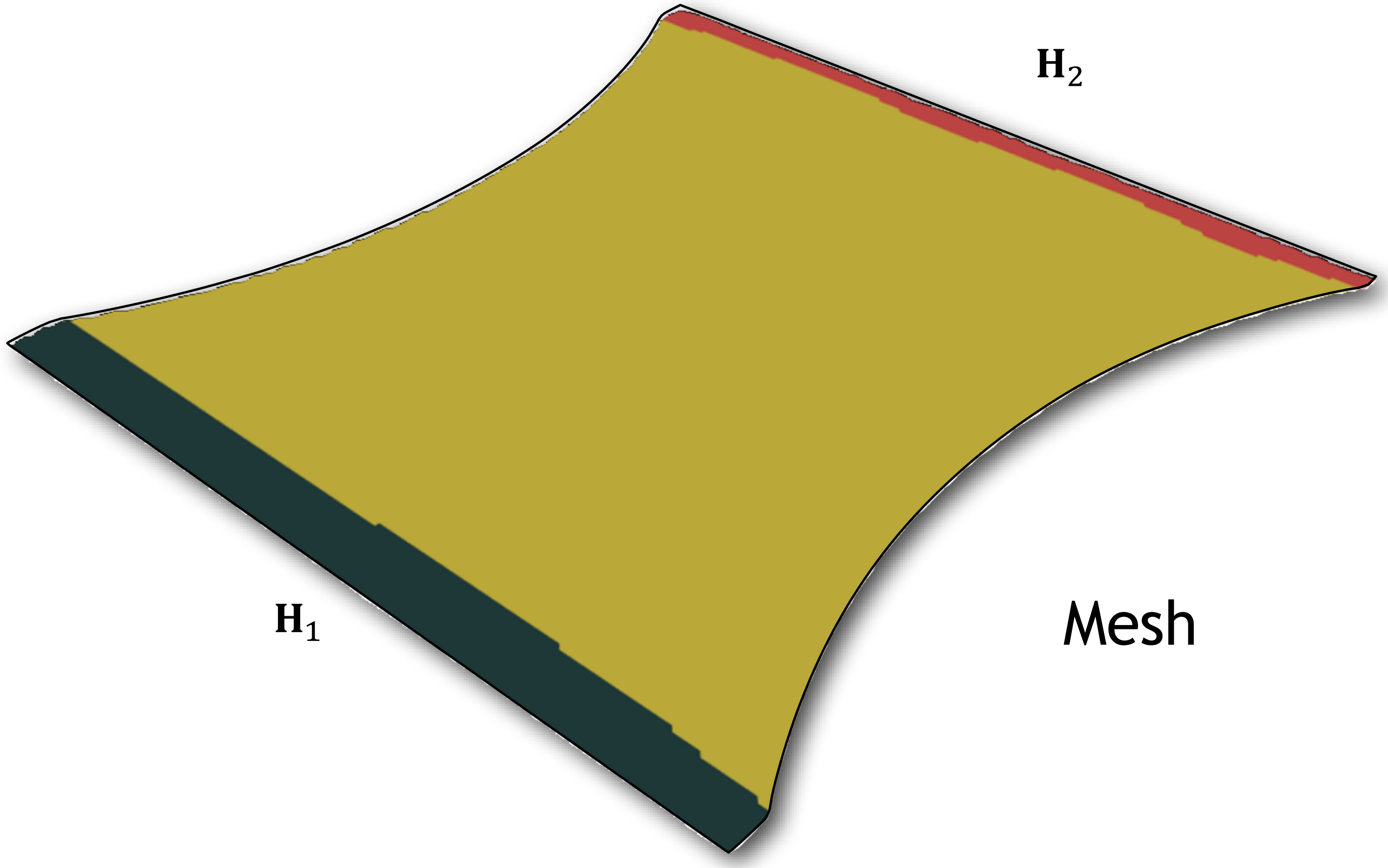


Step 2: Smoothing

- Remove high-frequency details with handles fixed
- Initially only the smoothed mesh will be deformed, and the details will be transferred later
- Solve a bi-Laplacian system
  - solution minimizes the Laplacian Energy

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

$$s.t. \quad v_{H_i} = \boxed{o_{H_i}}$$ Original positions on S

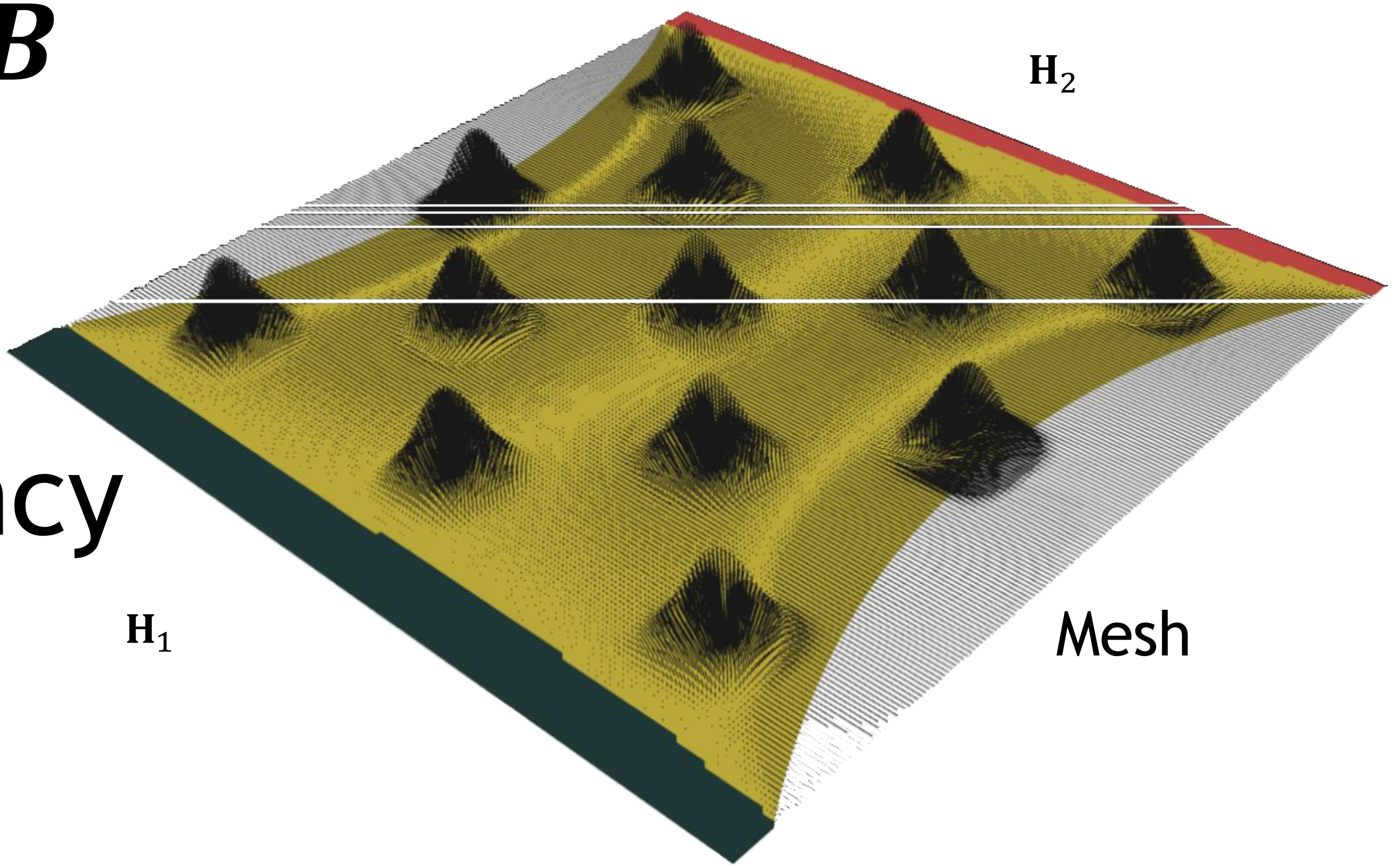


Mesh

Step 3: Encode Displacements

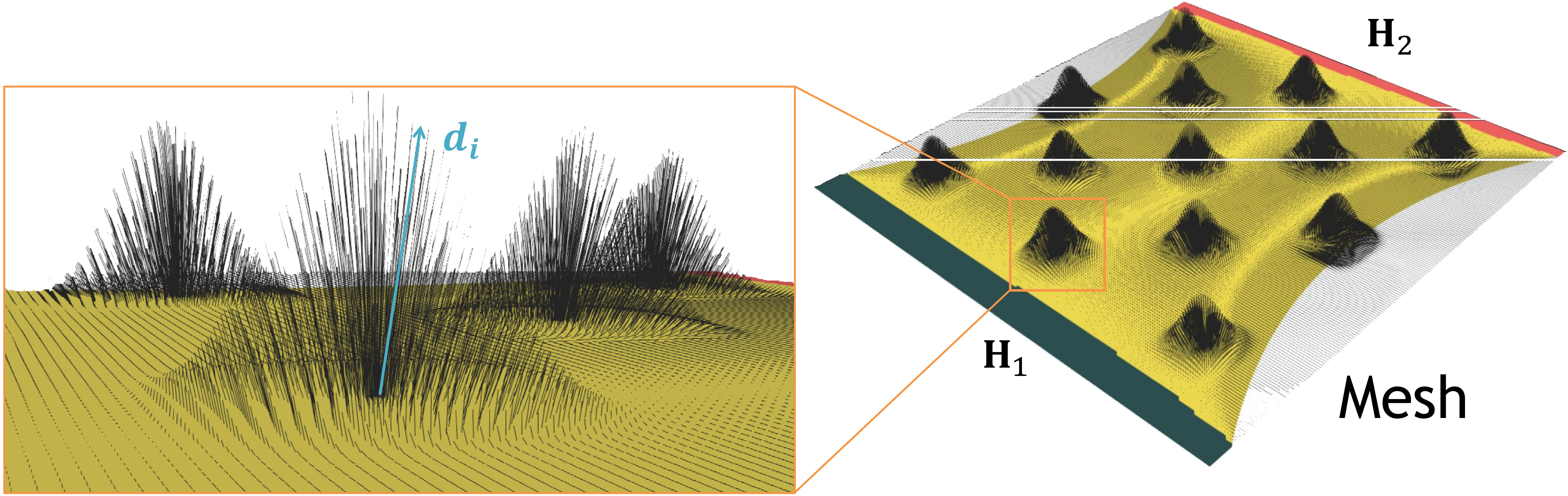
Per-vertex displacement from $\boldsymbol{B}$ to $\boldsymbol{S}$

- $\boldsymbol{d_i = v_i^S - v_i^B}$
- $\boldsymbol{d_i}$ represent the high frequency details
- will be added back after deformation



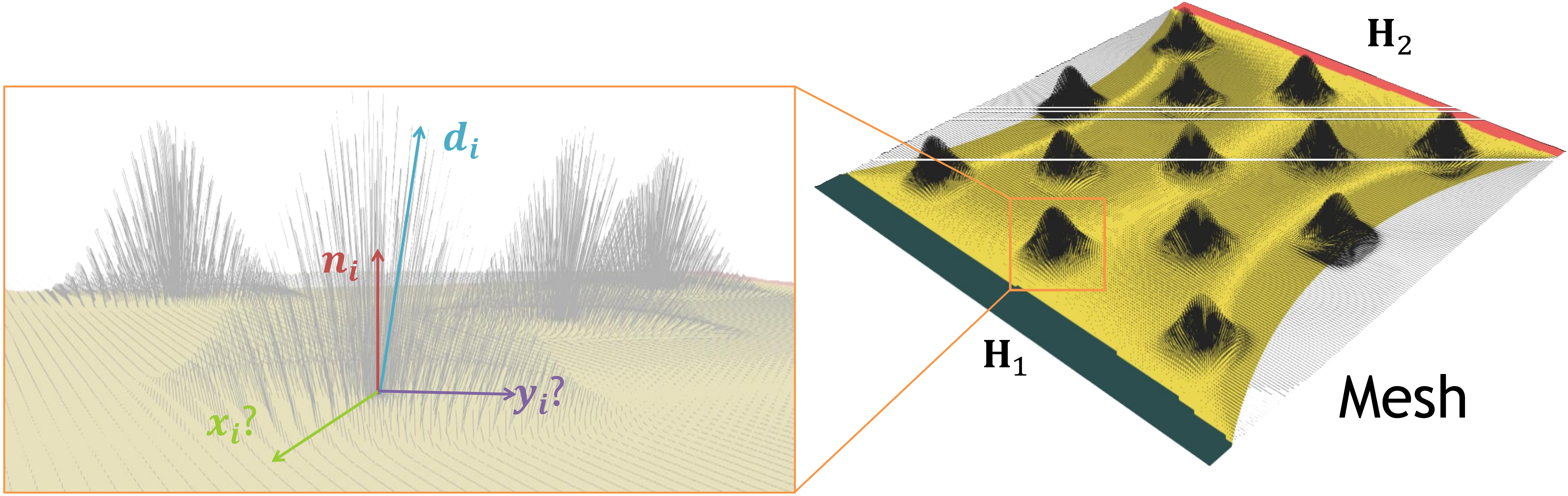
Step 3: Encode Displacements

Represent $\boldsymbol{d_i}$ in a local frame



Step 3: Encode Displacements

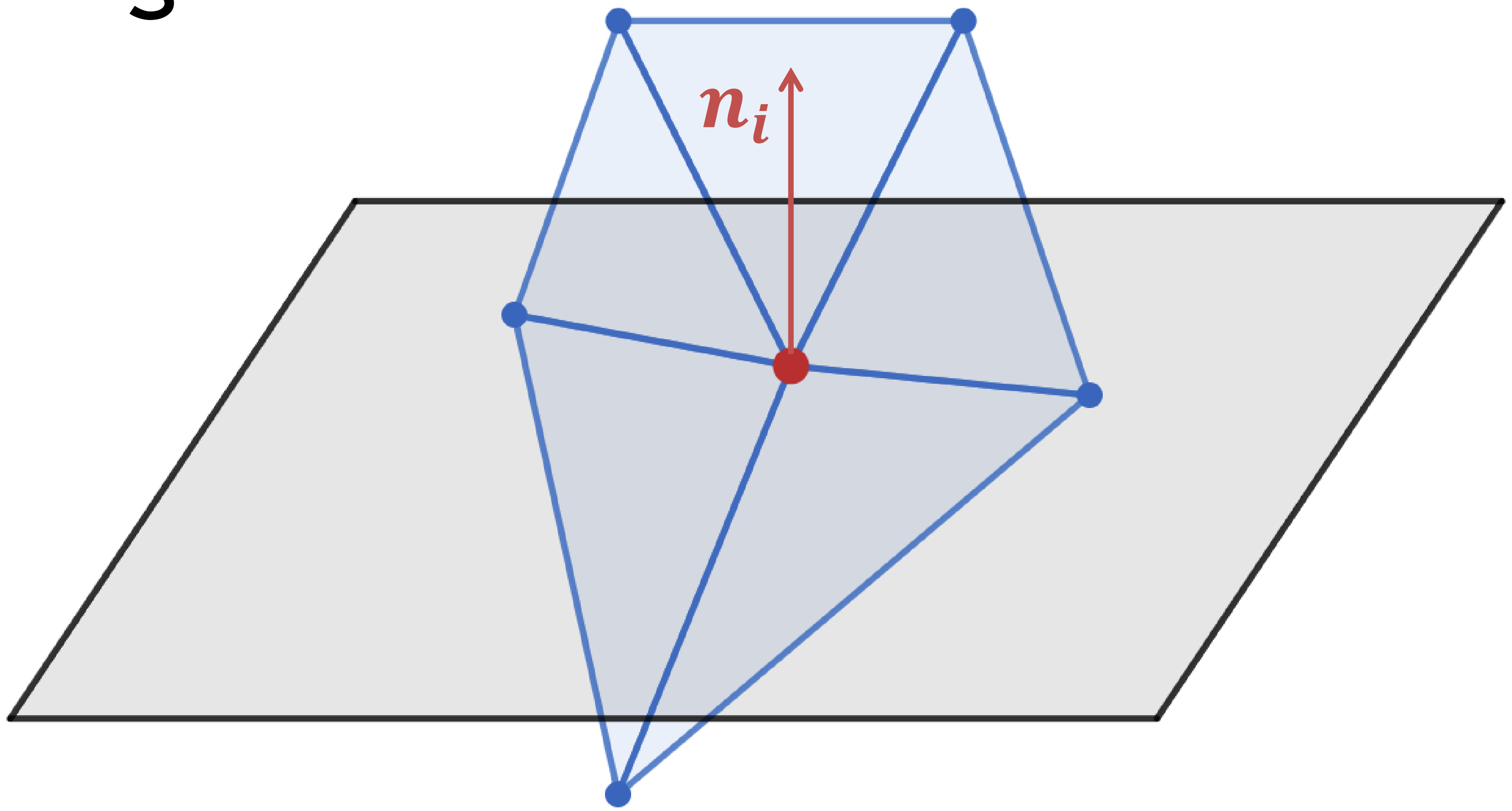
Represent $\boldsymbol{d}_i$ in a local frame



Step 3: Encode Displacements

Represent $\boldsymbol{d}_i$ in a local frame

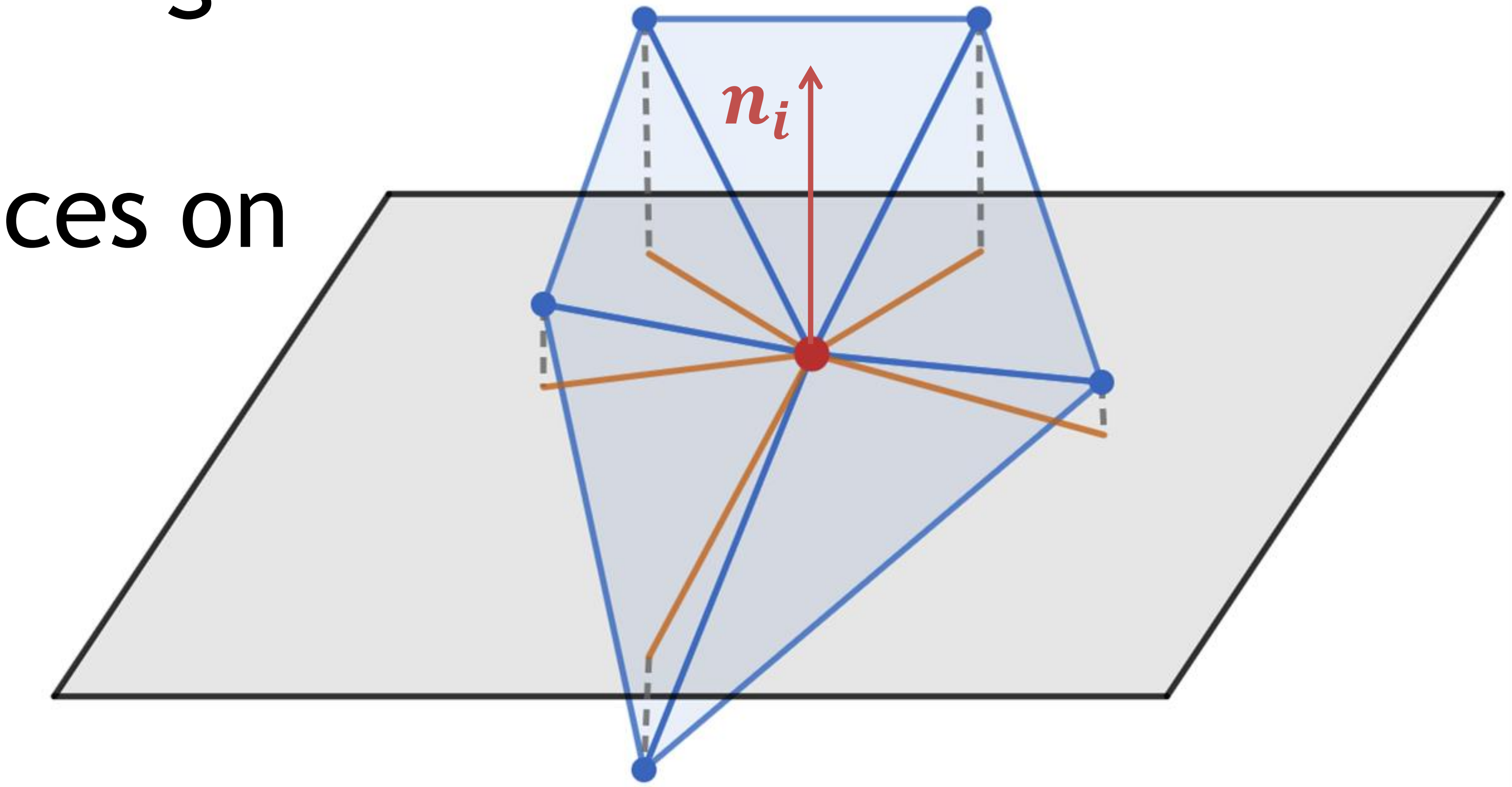
- Compute the normal $\boldsymbol{n}_i$ and tangent plane at the vertex $\boldsymbol{v}_i$



Step 3: Encode Displacements

Represent $\boldsymbol{d_i}$ in a local frame

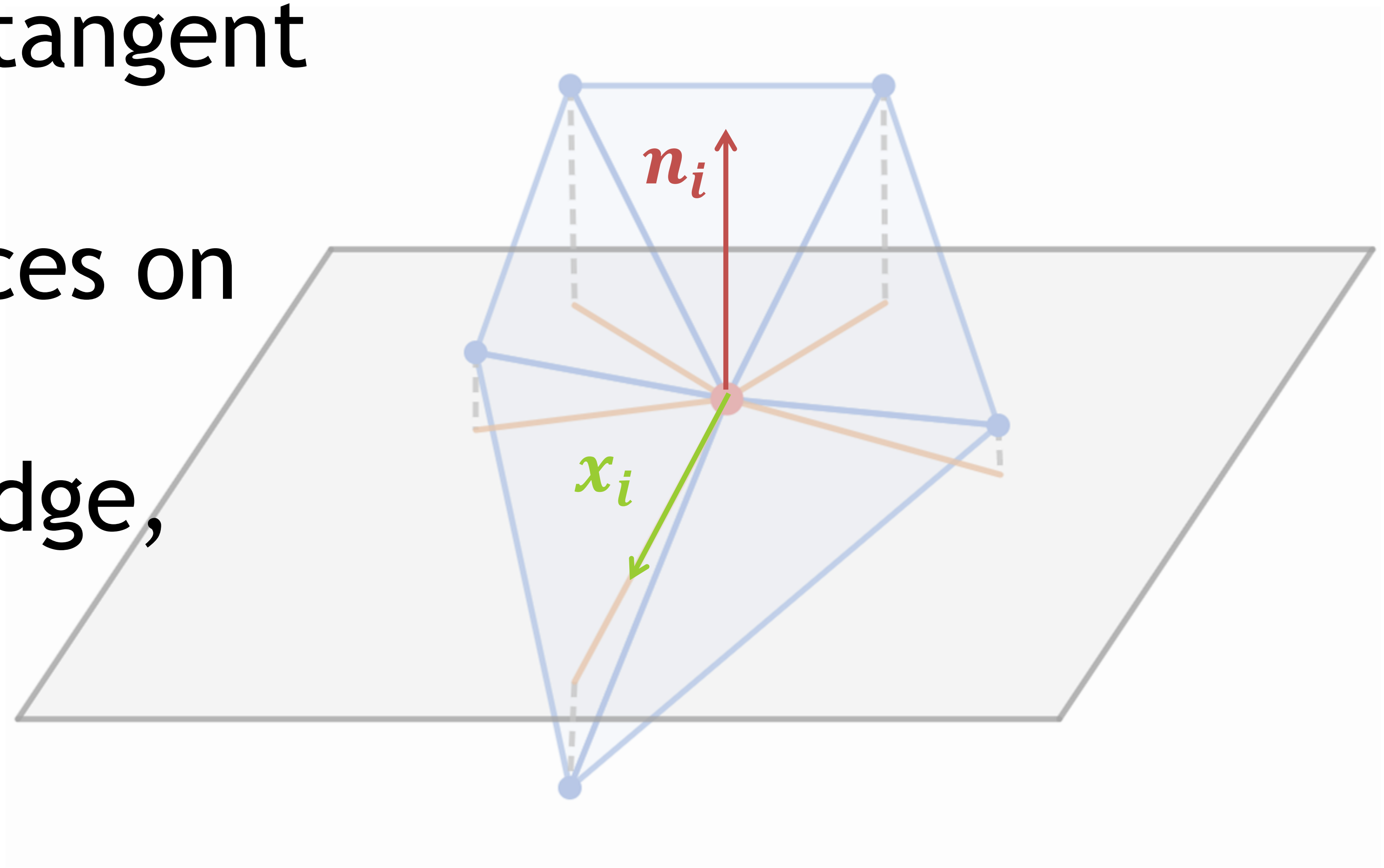
- Compute the normal $\boldsymbol{n_i}$ and tangent plane at the vertex $\boldsymbol{v_i}$
- Project all neighboring vertices on the tangent plane



Step 3: Encode Displacements

Represent $\boldsymbol{d}_i$ in a local frame

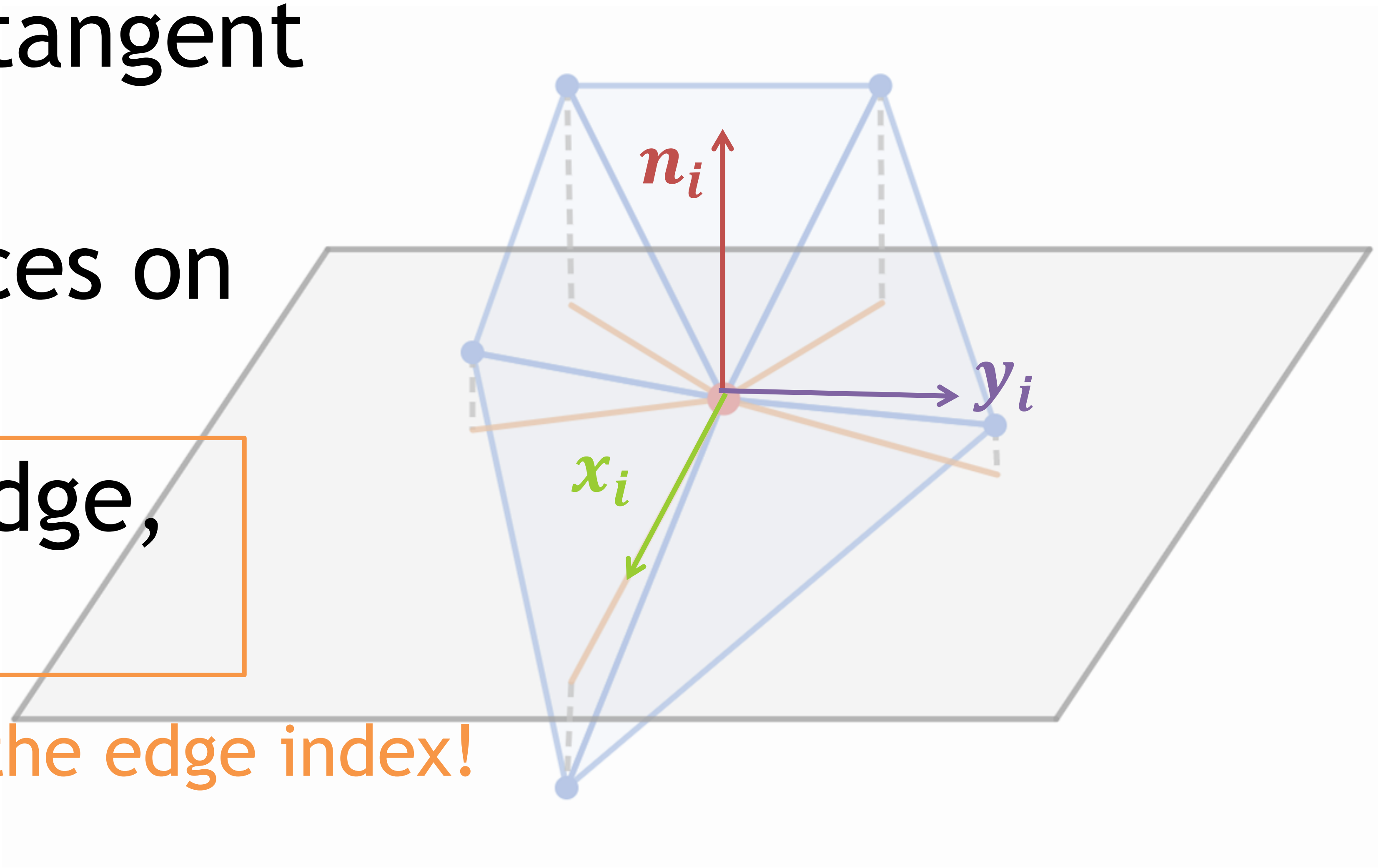
- Compute the normal $\boldsymbol{n}_i$ and tangent plane at the vertex $\boldsymbol{v}_i$
- Project all neighboring vertices on the tangent plane
- Find the longest projected edge, normalize it and set it as $\boldsymbol{x}_i$



Step 3: Encode Displacements

Represent $\boldsymbol{d}_i$ in a local frame

- Compute the normal $\boldsymbol{n}_i$ and tangent plane at the vertex $\boldsymbol{v}_i$
- Project all neighboring vertices on the tangent plane
- Find the longest projected edge, normalize it and set it as $\boldsymbol{x}_i$
- Compute $\boldsymbol{y}_i = \boldsymbol{n}_i \times \boldsymbol{x}_i$



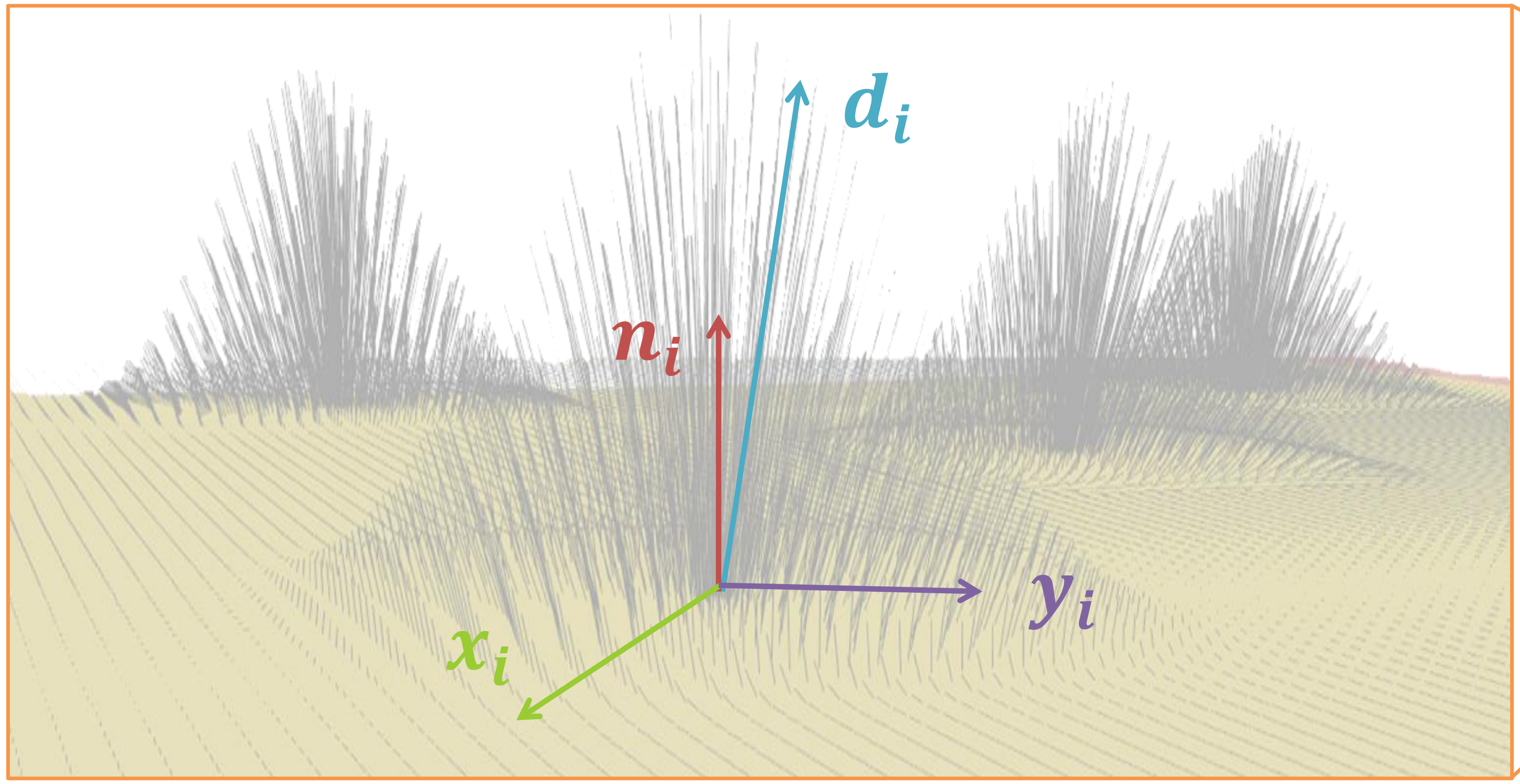
Save the edge index!

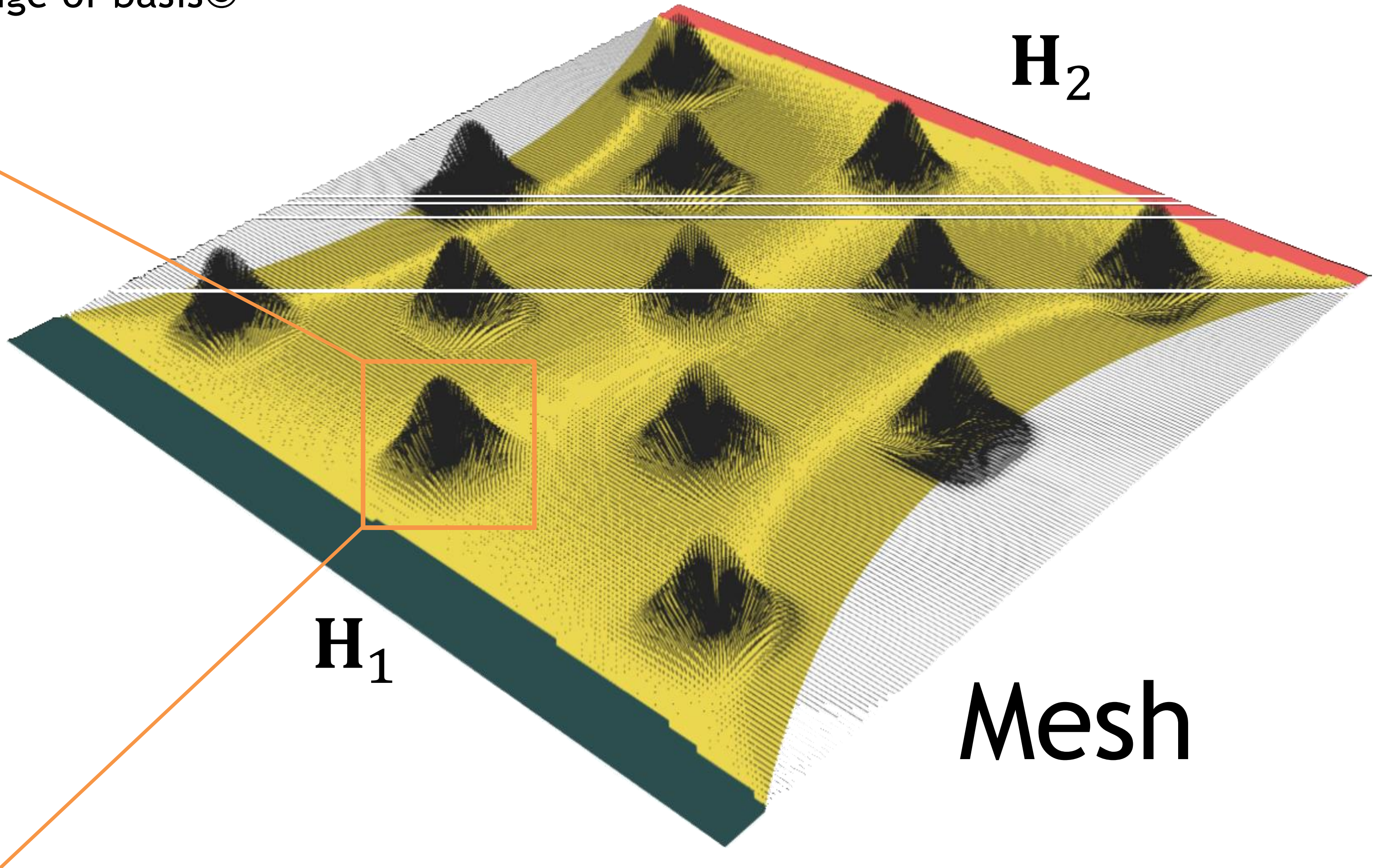
Step 3: Encode Displacements

Represent $\boldsymbol{d_i}$ in a local frame

$$\boldsymbol{d_i} = \boldsymbol{d_i^x}\boldsymbol{x_i} + \boldsymbol{d_i^y}\boldsymbol{y_i} + \boldsymbol{d_i^n}\boldsymbol{n_i}$$

Change of basis☺





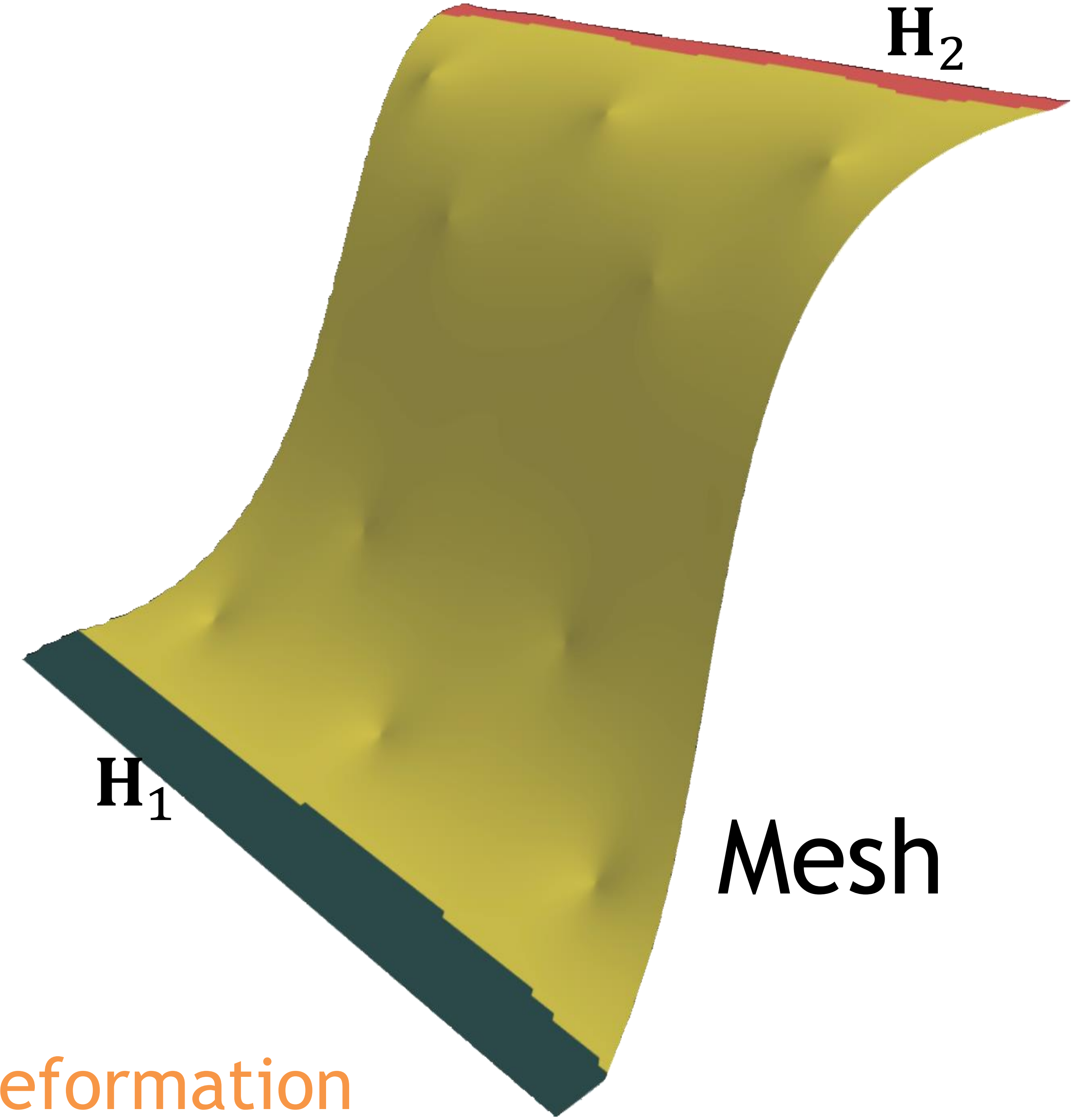
Step 4: Deform

Deform the mesh by manipulating the handles

- Solve for the deformed shapes $\boldsymbol{B}'$
- Solve similar bi-Laplacian system but with fixed **new** handle positions

$$\min_{v} v^T L_\omega M^{-1} L_\omega v$$
$$s.t. \quad v_{H_i} = \boxed{o_{H_i}}$$

New positions after deformation



Mesh

Step 4: Deform

Where does this system come from?

Usually, we try to minimize some energy in deformation (comparable to the distortion energies in parameterization). Ours could be something like this:

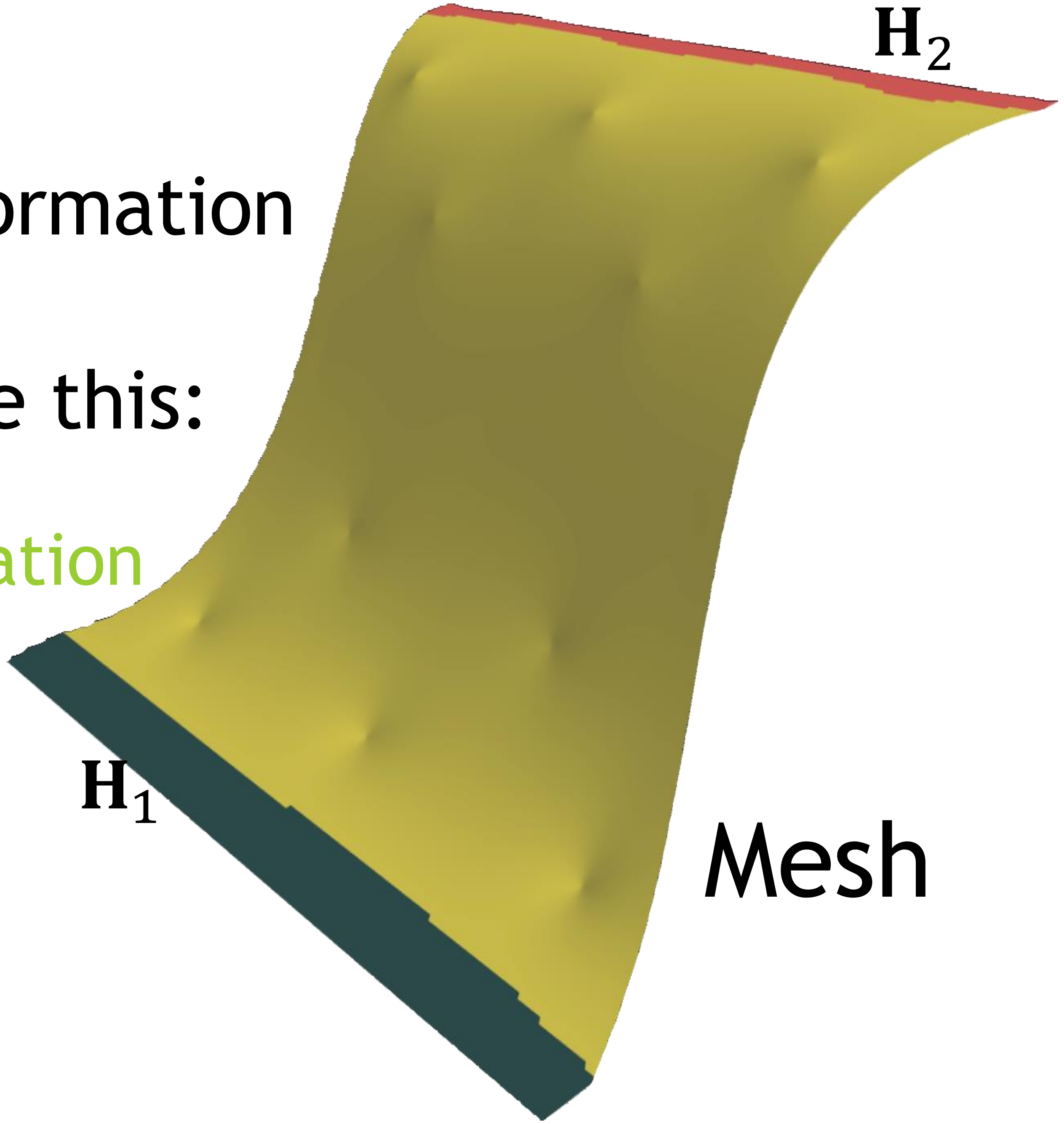
$$E = \sum_{v \in V} A_v \underbrace{\boxed{\|l_v}}_{} \boxed{- l'_v} \|^2$$

Original laplacian

Laplacian after deformation

+ fulfill handle constraints

→get bi-Laplacian after derivative to minimize energy



Mesh

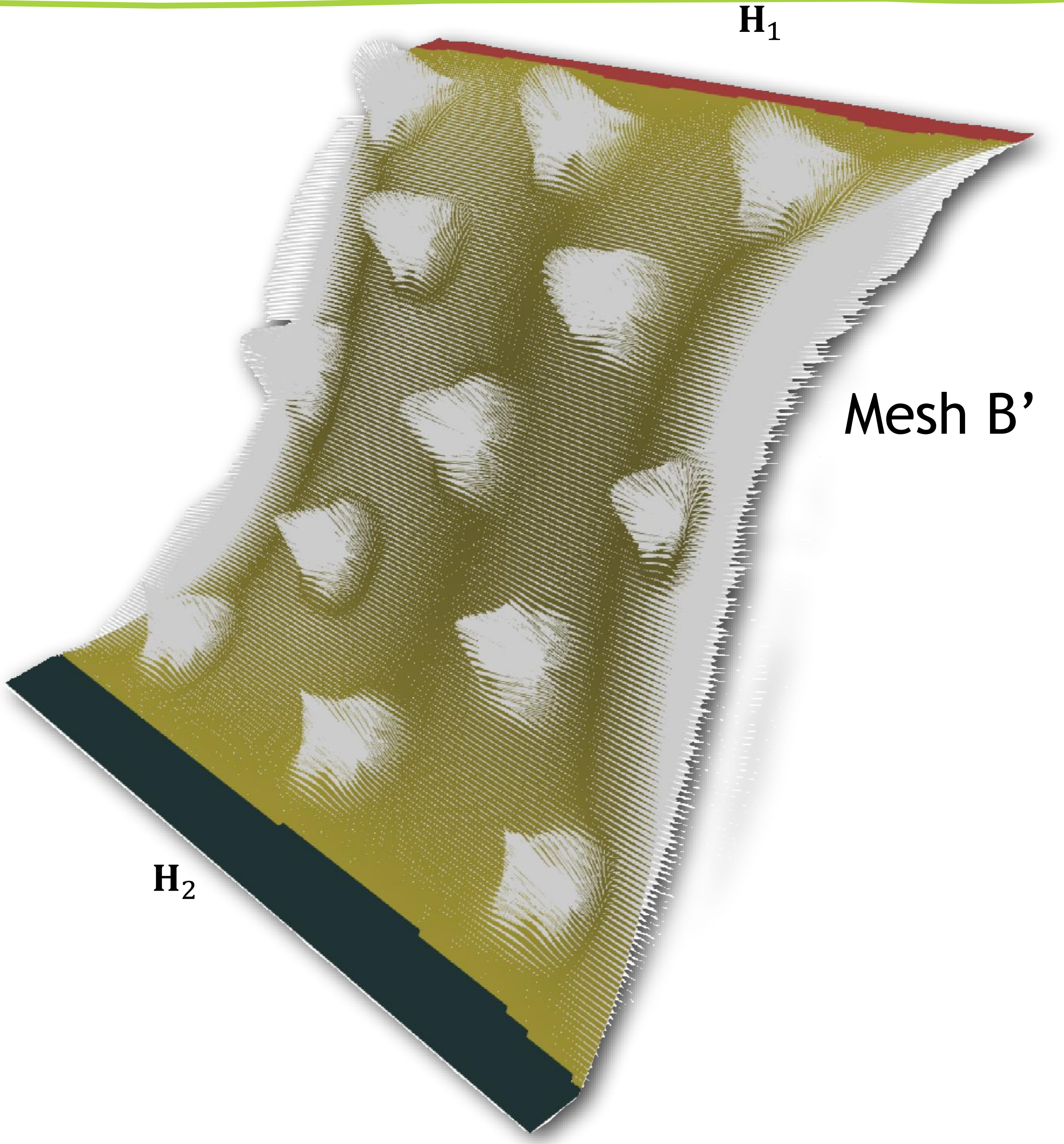
Step 5: Add local detail

Compute the local frame on $\boldsymbol{B'}$

- Calculate normal $\boldsymbol{n'_i}$
- Use the <u>same</u> edge as before but on $\boldsymbol{B'}$ to define $\boldsymbol{x'_i}$
- Compute $\boldsymbol{y'_i} = \boldsymbol{n'_i} \times \boldsymbol{x'_i}$

Use the new local frame to compute

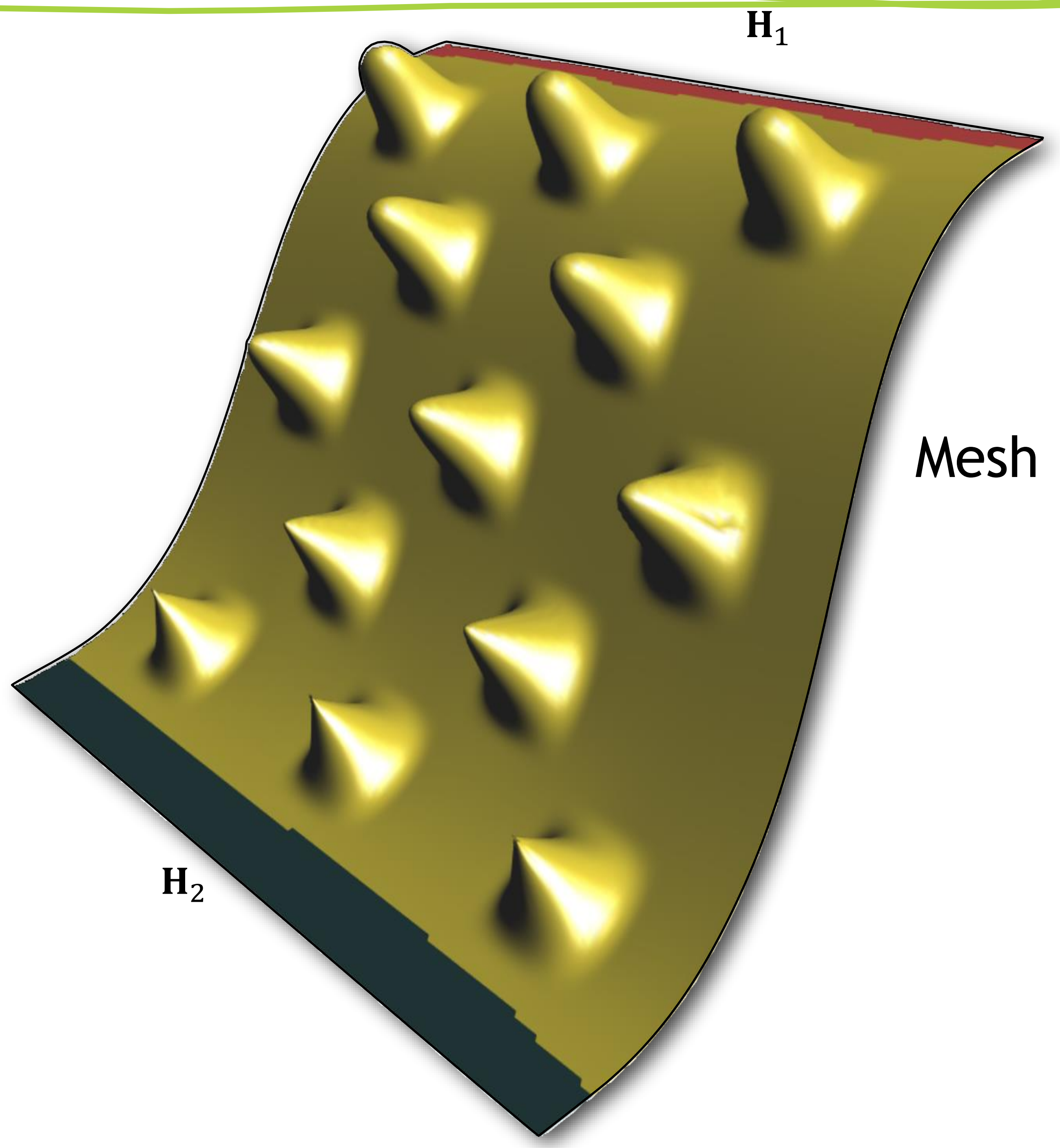
$$\boldsymbol{d'_i} = \color{green}{\boldsymbol{d_i^x}}\boldsymbol{x'_i} + \color{purple}{\boldsymbol{d_i^y}}\boldsymbol{y'_i} + \color{red}{\boldsymbol{d_i^n}}\boldsymbol{n'_i}$$



Step 5: Add local detail

Add local detail to $\boldsymbol{B}'$ to get the deformed shape:

$$\boldsymbol{S}' = \boldsymbol{B}' + \boldsymbol{d}'$$



How to solve a constrained system

$$\boxed{\begin{aligned} &\min_{v} v^T L_\omega M^{-1} L_\omega v \\ &s.t. \quad v_{H_i} = o_{H_i} \end{aligned}}$$

- Positions are imposed as hard constraints
- could be done using Lagrange multipliers (similar to assignment 4)
- but in this assignment, we will use substitution

(Disclaimer: these two approaches do not yield exactly the same results, but for our intents and purposes we can ignore this subtle difference)

Substitution example

- Rough idea can be easily seen by small example:
- $\begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 3 \\ 0 & 1 & 2 \end{bmatrix} \begin{bmatrix} x \\ 3 \\ y \end{bmatrix} = \begin{bmatrix} 3 \\ 7 \\ 10 \end{bmatrix}$
- $\begin{bmatrix} 1x + 1y \\ \cancel{1x + 3y} \\ 0x + 2y \end{bmatrix} \cancel{=} \begin{bmatrix} 3 - 3 \\ \cancel{7 - 6} \\ 10 - 3 \end{bmatrix}$

+ Additionally, you ignore the constrained row (here, the second)

Constrained solving through substitution

$$\boxed{\begin{aligned} &\min_{v} v^T L_\omega M^{-1} L_\omega v \\ &s.t. \quad v_{H_i} = o_{H_i} \end{aligned}}$$

$$A = \mathbf{L}_\omega \mathbf{M}^{-1} \mathbf{L}_\omega = \begin{bmatrix} A_{ff} & A_{fc} \\ A_{cf} & A_{cc} \end{bmatrix}$$

$$\begin{bmatrix} A_{ff} & A_{fc} \end{bmatrix} \begin{bmatrix} v_f \\ v_c \end{bmatrix} = 0 \Rightarrow A_{ff} v_f = -A_{fc} v_c$$

To do this reshuffling, igl::slice and igl::slice_into might become your best friend!

Pre-factoring the bi-Laplacian

```
//PickingPlugin.h
Eigen::SimplicialCholesky<SparseMatrixType, Eigen::RowMajor > solver;
solver.compute (BiLaplacian_ff); // the interior part of the (almost) bi-laplacian
```

- Factorization is the bottleneck of the solve → O(n^3)!
- Prefactorization is crucial to achieve real-time performance
- Should only be performed when a new handle is defined

Deformation Transfer

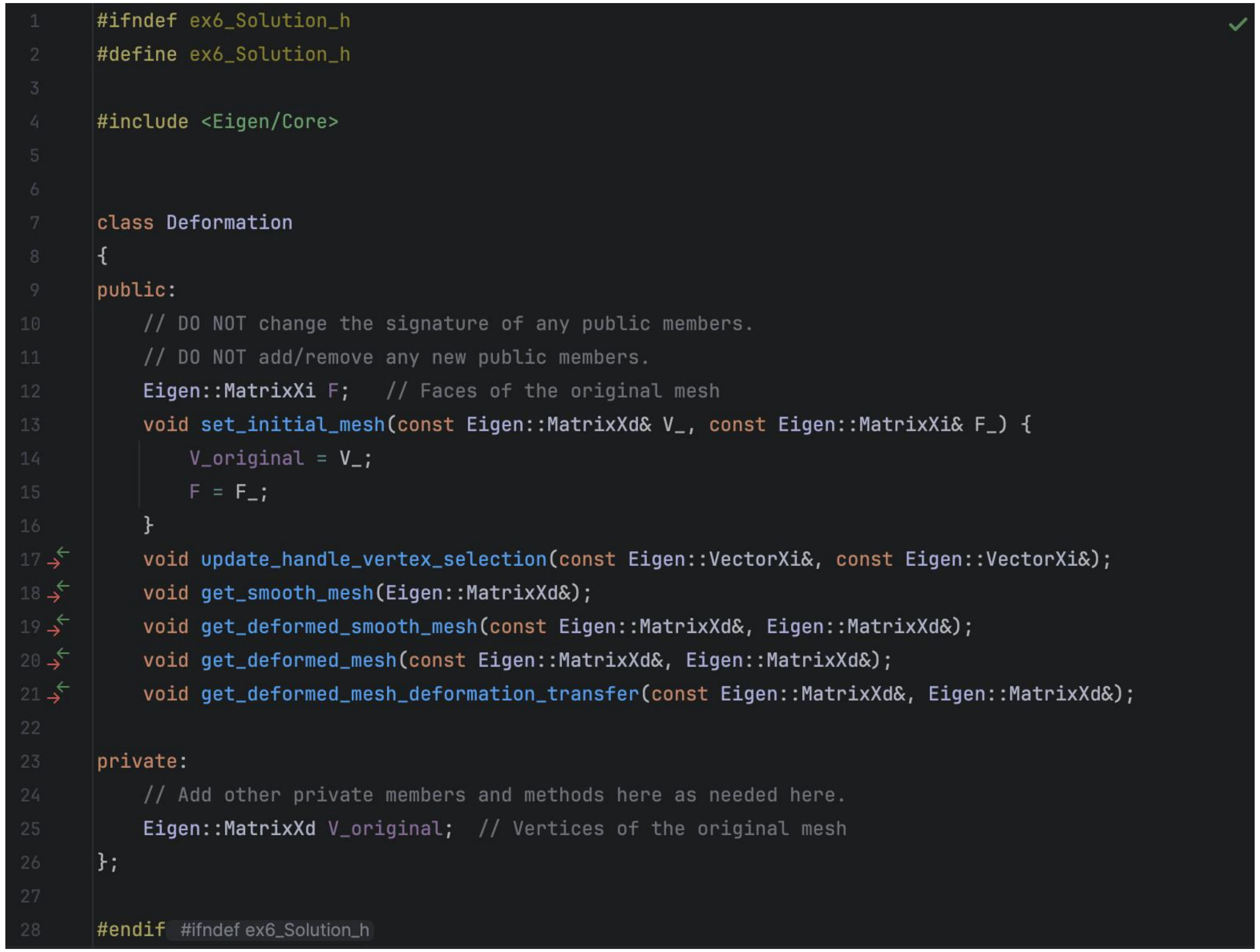
- Recall Multi-resolution:
  - $S = B + d$: base + details
  - $B \rightarrow B'$: deform base shape
  - $S' = B' + d'$: add rotation-invariant displacement back
- Deformation transfer:
  - $B \rightarrow B'$: already encodes the deformation
  - Solve for S' such that "the deformation from S to S'" is equivalent to "the deformation from B to B'" (Eq. (14) in the paper)

Provided Code

- Enables basic picking and dragging of handles
- You will fill it in with your deformation code in ***Deformation*** class (deformation.cpp/h)
- Shortcuts:
    - ’S’: select
    - ’A’: accept selection
    - ALT+’T’: translation, ALT+‘R’: rotation

Implementation Guidelines

- No modification on the signature of any public member is allowed
- Minimize changes to main.cpp
- Changes on private members are allowed



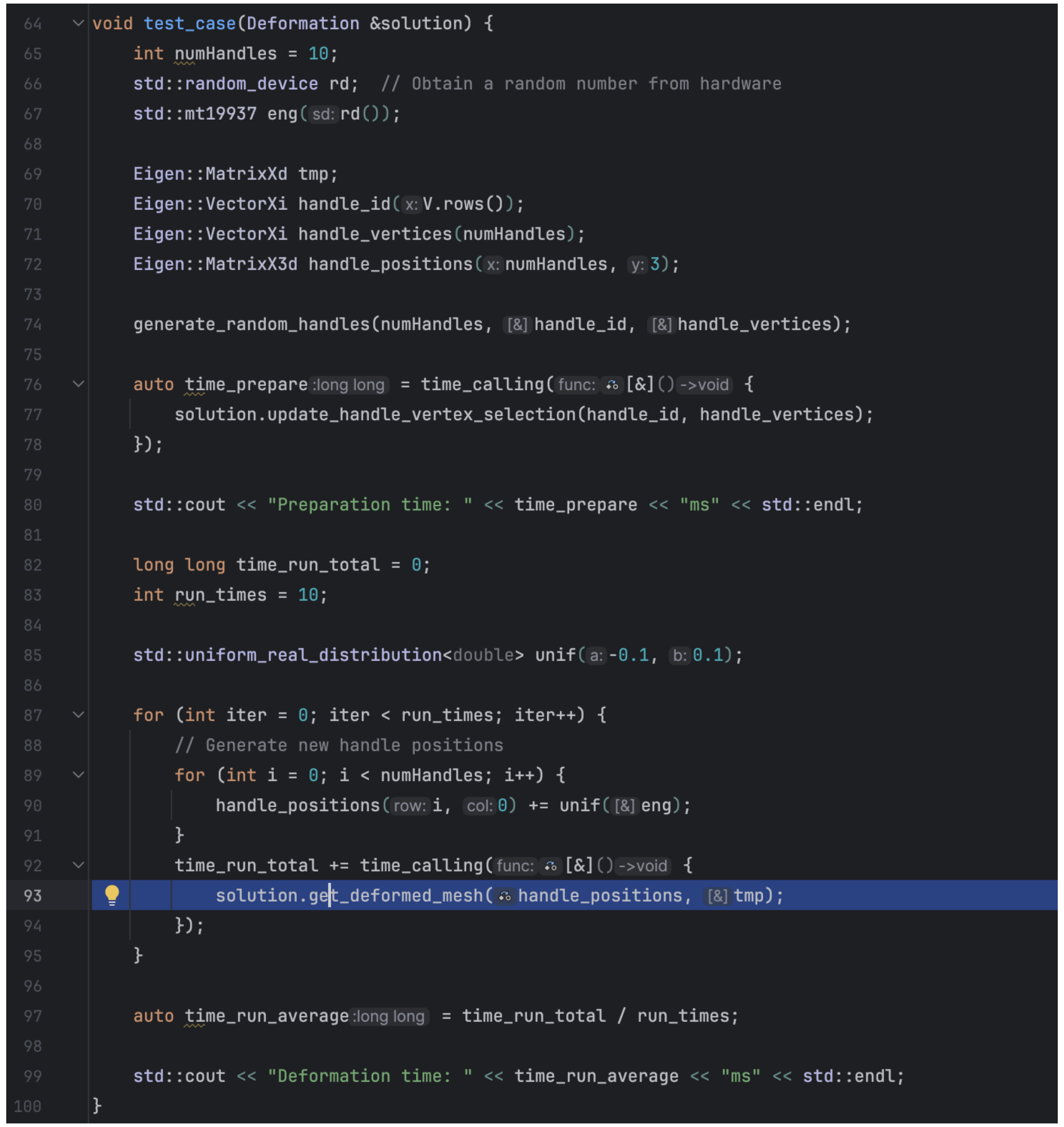
```
#ifndef ex6_Solution_h
#define ex6_Solution_h

#include <Eigen/Core>


class Deformation
{
public:
    // DO NOT change the signature of any public members.
    // DO NOT add/remove any new public members.
    Eigen::MatrixXi F;   // Faces of the original mesh
    void set_initial_mesh(const Eigen::MatrixXd& V_, const Eigen::MatrixXi& F_) {
        V_original = V_;
        F = F_;
    }
    void update_handle_vertex_selection(const Eigen::VectorXi&, const Eigen::VectorXi&);
    void get_smooth_mesh(Eigen::MatrixXd&);
    void get_deformed_smooth_mesh(const Eigen::MatrixXd&, Eigen::MatrixXd&);
    void get_deformed_mesh(const Eigen::MatrixXd&, Eigen::MatrixXd&);
    void get_deformed_mesh_deformation_transfer(const Eigen::MatrixXd&, Eigen::MatrixXd&);

private:
    // Add other private members and methods here as needed here.
    Eigen::MatrixXd V_original;  // Vertices of the original mesh
};

#endif  #ifndef ex6_Solution_h
```

Deformation.h

Implementation Guidelines

- Why?

- Your implementation's efficiency (and/or correctness) will be tested offline

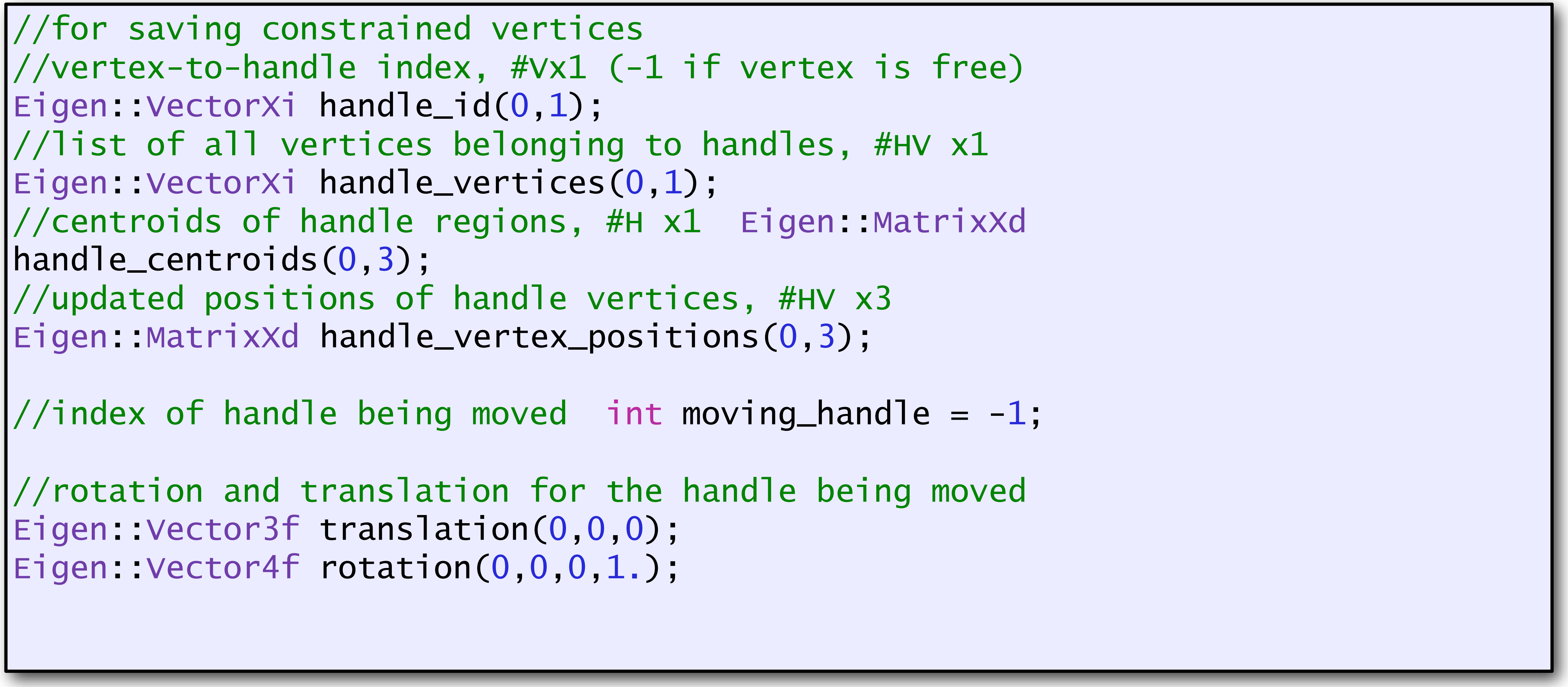


```cpp
void test_case(Deformation &solution) {
    int numHandles = 10;
    std::random_device rd;  // Obtain a random number from hardware
    std::mt19937 eng(rd());

    Eigen::MatrixXd tmp;
    Eigen::VectorXi handle_id(V.rows());
    Eigen::VectorXi handle_vertices(numHandles);
    Eigen::MatrixX3d handle_positions(numHandles, 3);

    generate_random_handles(numHandles, handle_id, handle_vertices);

    auto time_prepare = time_calling([&]() {
        solution.update_handle_vertex_selection(handle_id, handle_vertices);
    });

    std::cout << "Preparation time: " << time_prepare << "ms" << std::endl;

    long long time_run_total = 0;
    int run_times = 10;

    std::uniform_real_distribution<double> unif(-0.1, 0.1);

    for (int iter = 0; iter < run_times; iter++) {
        // Generate new handle positions
        for (int i = 0; i < numHandles; i++) {
            handle_positions(i, 0) += unif(eng);
        }
        time_run_total += time_calling([&]() {
            solution.get_deformed_mesh(handle_positions, tmp);
        });
    }

    auto time_run_average = time_run_total / run_times;

    std::cout << "Deformation time: " << time_run_average << "ms" << std::endl;
}
```

Efficiency test example

Provided Code

- Picking infrastructure



```
//for saving constrained vertices
//vertex-to-handle index, #Vx1 (-1 if vertex is free)
Eigen::VectorXi handle_id(0,1);
//list of all vertices belonging to handles, #HV x1
Eigen::VectorXi handle_vertices(0,1);
//centroids of handle regions, #H x1  Eigen::MatrixXd
handle_centroids(0,3);
//updated positions of handle vertices, #HV x3
Eigen::MatrixXd handle_vertex_positions(0,3);

//index of handle being moved  int moving_handle = -1;

//rotation and translation for the handle being moved
Eigen::Vector3f translation(0,0,0);
Eigen::Vector4f rotation(0,0,0,1.);
```

Provided Code

- While handle is being dragged

```
void get_new_handle_locations()
```

updates all handle vertex positions based on rotation and translation

stores them in handle_vertex_positions

- Replace solve() with your code

```
bool solve(igl::Viewer& viewer, bool update_constraints)
{
  igl::slice_into(handle_vertex_positions, handle_vertices, 1, V);

  /* etc. update variables*/  return true;
 };
```

- Turn on for easier debugging

```
#define UPDATE_ONLY_ON_UP
```

Must be OFF during demo (want to see deformation while mouse moves)

Questions?

Thank you!