Assignment 5: Shape Deformation

Handout date: 06.05.2022
Submission deadline: 27.05.2022 at 10:00

Goal of this exercise

In this exercise, you will implement an algorithm to interactively deform 3D models. You will construct a two-level multi-resolution surface representation and use naive Laplacian editing to deform it.

1. Multiresolution mesh editing

For this task, you will compute a mesh deformation based on the rotations and translations applied interactively to a subset of its vertices via the mouse. Let H be the set of "handle" vertices that the user can manipulate (or leave fixed). We want to compute a deformation for the remaining vertices, denoted as R.

Let $\mathcal{S}$ be our input surface, represented as a triangle mesh. We want to compute a new surface that contains:

- the vertices in H translated/rotated using the user-provided transformation t, and
- the vertices in R properly deformed using the algorithm described next.

The algorithm is divided in three phases (see Figure 1):

- removing high-frequency details,
- deforming the smooth mesh, and
- transferring high-frequency details to the deformed surface.

1.1. **Selecting the handles.** A minimal, lasso-based interface for selecting vertices has been implemented for you. To use it, enable the `SELECT` mouse mode from the menu (or hit 'S'), click somewhere **on** the mesh and drag with your mouse to draw a stroke around the area of the mesh you want to select as a handle. The vertices inside the stroked region are saved in the `selected_v` variable. You have the options to: (a) accept the selected vertices as a new handle region (only if the vertices are not assigned to a handle already) by hitting the relevant button on the menu or key 'A' (Fig. 2(a)), (b) discard the selection and make a new one by drawing another stroke somewhere on the mesh. Once a selection is accepted, you can add additional handles by drawing more strokes.

As selections are accepted, their vertices are saved in the `handle_vertices` variable. We also store the handle index for each vertex in `handle_id` (-1 if the vertex belongs to no handle). The handle region centroids are stored in `handle_centroids`.

The selected handles can be transformed by selecting the appropriate mouse mode (`TRANSLATE` / `ROTATE`, shortcuts: ALT+'T', ALT+'R') and dragging with the mouse. While handles are dragged, the updated handle vertex positions are stored in `handle_vertex_positions`.

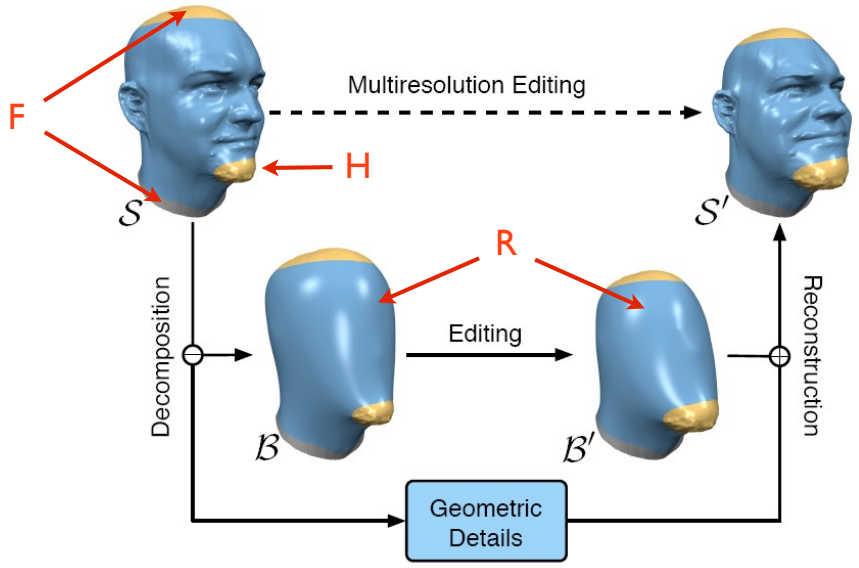
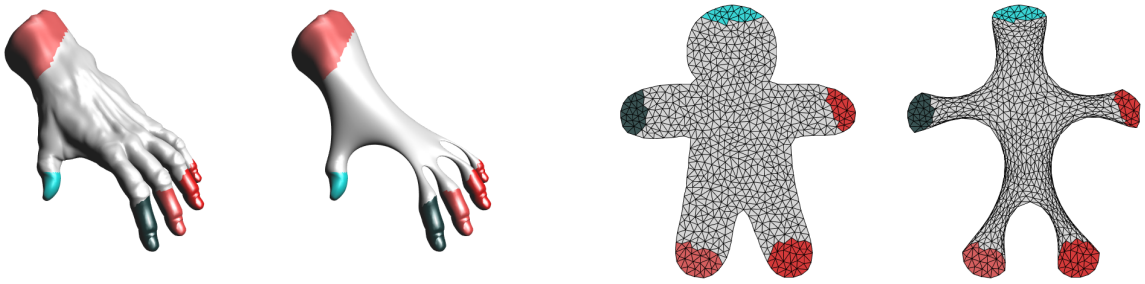


FIGURE 1. Algorithm overview



(A) *hand* mesh.

(B) Smoothed *hand*.

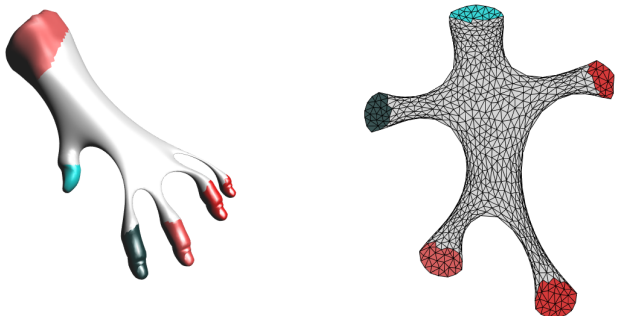
(C) *woody* mesh.

(D) Smoothed *woody*.

FIGURE 2. Step 1: Removal of high-frequency details.

1.2. **Removal of high-frequency details.** We remove the high-frequency details from the vertices R in $\mathcal{S}$ by minimizing the thin-plate energy, which involves solving a bi-Laplacian system arising from the quadratic energy minimization:

$$\min_{\mathbf{v}} \mathbf{v}^T \mathbf{L}_\omega \mathbf{M}^{-1} \mathbf{L}_\omega \mathbf{v}$$
$$\text{subject to } \mathbf{v}_H = \mathbf{o}_H,$$



(A) Deformed/ Smoothed *hand*.

(B) Deformed/ Smoothed *woody*.

FIGURE 3. Step 2: Deformation of the smoothed mesh.

where $\mathbf{o}_H$ are the handle H's vertex positions, $\mathbf{L}_\omega$ is the cotan Laplacian of $\mathcal{S}$, and $\mathbf{M}$ is the mass matrix of $\mathcal{S}$.

Notice that $\mathbf{L}_\omega$ is the symmetric matrix consisting of the cotangent weights ONLY (without the division by Voronoi areas). In other words, it evaluates an "integrated" Laplacian rather than an "averaged" laplacian when applied to a vector of vertices. The inverse mass matrix appearing in the formula above then applies the appropriate rescaling so that the laplacian operator can be applied again (i.e., so that the Laplacian value computed at each vertex can be interpreted as a piecewise linear scalar field whose Laplacian can be computed).

This optimization will produce a mesh similar to the one in Figure 2. Note that the part of the surface that we want to deform is now free of high-frequency details. We call this mesh $\mathcal{B}$.

1.3. **Deforming the smooth mesh.** The new deformed mesh is computed similarly to the previous step, by solving the minimization:

$$\min_{\mathbf{v}} \mathbf{v}^T \mathbf{L}_\omega \mathbf{M}^{-1} \mathbf{L}_\omega \mathbf{v}$$
$$\text{subject to } \mathbf{v}_H = t(\mathbf{o}_H),$$

where $t(\mathbf{o}_H)$ are the new handle vertex positions after applying the user's transformation. We call this mesh $\mathcal{B}'$. See Figure 3 for an example.

1.4. **Transferring high-frequency details to the deformed surface.** The high-frequency details on the original surface are extracted from $\mathcal{S}$ and transferred to $\mathcal{B}'$. We first encode the high-frequency details of $\mathcal{S}$ as displacements w.r.t. $\mathcal{B}$.

We define an orthogonal reference frame on every vertex v of $\mathcal{B}$ using:

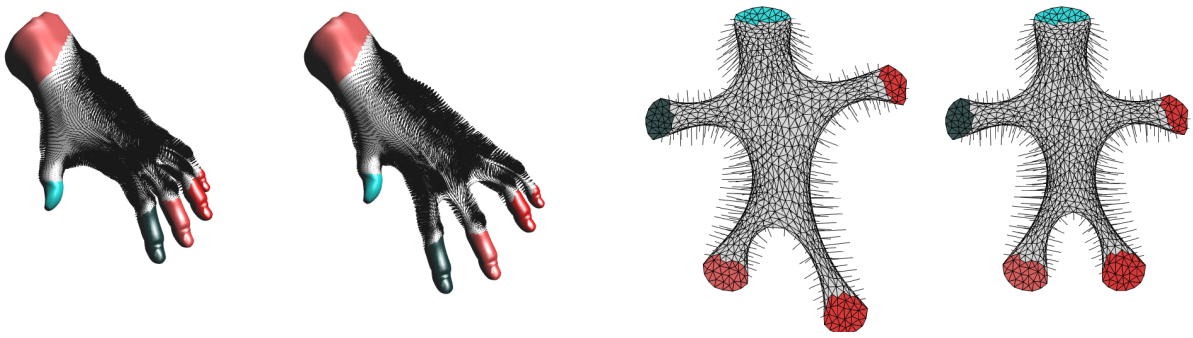
(1) The unit vertex normal

(2) The normalized projection of one of v's outgoing edges onto the tangent plane defined by the vertex normal. A stable choice is the edge whose projection onto the tangent plane is longest.
(3) The cross-product between (1) and (2)

For every vertex v, we compute the displacement vector that takes v from $\mathcal{B}$ to $\mathcal{S}$ and represent it as a vector in v's reference frame.

For every vertex of $\mathcal{B}'$, we also construct a reference frame using the normal and the SAME outgoing edge we selected for $\mathcal{B}$ (not the longest in $\mathcal{B}'$; it is important that the edges used to build both reference frames are the same). We can now use the displacement vector components computed in the previous paragraph to define transferred displacement vectors in the new reference frames of $\mathcal{B}'$. See Figure 4 for an example.

Applying the transferred displacements to the vertices of $\mathcal{B}'$ generates the final deformed mesh $\mathcal{S}'$. See Figure 5 for an example.



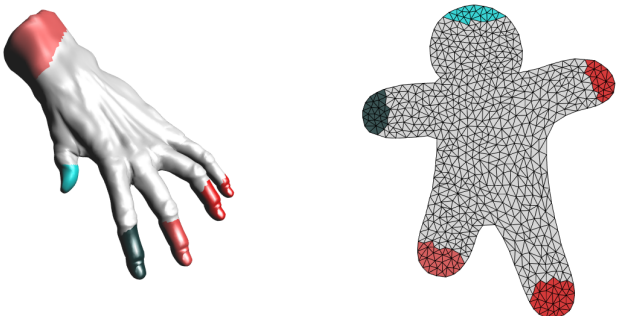
(A) Displacements on $\mathcal{B}$ for *hand*.
(B) Displacements on $\mathcal{B}'$ for *hand*.
(C) Displacements on $\mathcal{B}$ for *woody*.
(D) Displacements on $\mathcal{B}'$ for *woody*.

FIGURE 4. Step 3: Transfer high-frequency details to the deformed surface.

1.5. **Real-time performance.** To achieve real-time performance, you **must** prefactor the sparse bi-Laplacian matrix appearing in both linear systems. After the user specifies vertex sets H and F, you can factorize the matrix $\mathbf{L}_\omega\mathbf{M}^{-1}\mathbf{L}_\omega$ (using a Cholesky "LL^T" factorization, see Eigen::SimplicialCholesky) and then re-use the factorization to solve both linear systems efficiently. Additional subtasks that only need to be computed once (when the handles are selected) should also not be repeated unnecessarily. This is a mandatory part of the exercise; if your implementation does not achieve interactive frame-rates (10+ fps) on all provided meshes it will not receive the full score.

Required output of this section:

- Provide screenshots for 4 different deformed meshes. For each example, provide a rendering of $\mathcal{S}$, $\mathcal{B}$, $\mathcal{B}'$ and $\mathcal{S}'$.



(A) Final *hand* mesh.

(B) Final *woody* mesh.

FIGURE 5. Step 4: Final Deformation Results.

2. DEFORMATION TRANSFER

The high-frequency detail transfer method that you implemented in part 1 of this assignments may lead to self-intersections when the surface can locally not be properly approximated by a height field. In this task you will therefore replace the detail transfer from section 1.4 with the "deformation transfer" method, as explained in sections 3, 5 and 6 of the paper *"Deformation Transfer for Detail-Preserving Surface Editing"*, by Botsch et al. 2006. To compute the 3-by-3 source deformation gradient $\mathbf{S}_j$ you should use Equation 15 from the paper using the vertex positions $\mathbf{q}_i$ and $\mathbf{q}'_i$ of the meshes $\mathcal{B}$ and $\mathcal{B}'$.