

# Assignment 6: Skinning & Skeletal Animation

Handout date: 17.05.2024

Submission deadline: 14.06.2024 at 10:00

In this exercise, you will implement different methods for skeletal-based deformation. The main tasks are:

- Implement a handle selection tool to associate mesh vertices to each skeletal bone
- Implement forward kinematics (FK) to extract the transformation at each joint from the input bone parameters
- Compute the harmonic skinning weights
- Apply skeletal deformation with
  - Per-vertex Linear Blend Skinning (LBS)
  - Dual Quaternion Skinning (DQS)
  - Per-face Blend Skinning with Poisson stitching
- Context-aware skeletal deformation, where multiple example poses are provided to improve the deformation

**No template code is provided.** You are required to come up with your own solution from scratch. You can use previous assignments as an inspiration to start or use the default [example blank project](#). Please note that specifically for this assignment, **the clarity of your code will be taken into account**.

## 1. OVERVIEW, NOTATION, & BACKGROUND

1.1. **Input Data.** You are provided with the following data:

- The rest shape of a character (in `.off`, `.obj`)  $\mathcal{S} = (V, F)$ , where  $V \in \mathbb{R}^{n \times 3}$ ,  $F \in \mathbb{N}^{m \times 3}$
- The rigging (skeleton) of the character (in `.tgf`) contains the joints  $\mathcal{J} = \{j_1, \dots, j_{K+1}\} \in \mathbb{R}^3$  and the links (bones)  $\mathcal{B} = \{b_1, \dots, b_K\} \in \mathbb{R}^3$  between the joints, as a directed graph.
- For each bone  $b_k$ , we provide a bone handle  $H_k \subset V$  (in `.dmat`), i.e., a list of vertices associated with the bone. Note that the handles are disjoint, but need not cover the entire mesh surface, i.e.,  $H_i \cap H_j = \emptyset, \forall i \neq j$ , and  $\bigcup_{j=1}^K H_j \neq V$ . Note that, you are requested to design your own solution to select handles for the skeleton. The provided bone handles here are for reference only.
- Skeleton animation including a set of rotations (in `.dmat`). Note that all the rotations are represented in the quaternion format ( $[x, y, z, w]$  order). For each frame, a sequence of rotations  $P^{(l)} = \{\tilde{q}_1^{(l)}, \dots, \tilde{q}_K^{(l)}\}$  is given, where  $\tilde{q}_k^{(l)}$  gives the relative rotation of the  $k$ -th bone  $b_k$  (w.r.t. its source joint) for the  $l$ -th frame. Therefore, the `.dmat` file you load should be a  $\mathbb{R}^{KL \times 4}$  matrix, where  $K$  is the total number of bones, and  $L$  is the total number of frames. The rotations are given following the number of frames first, then the number of bones. It means

that the first  $K$  rotations are for the first frame, the following  $K$  for the second frame, and so on.

Given the deformation of the mesh skeleton (in the form of per-bone transformation  $R(b_k)$ ), we want to find the per-vertex  $R(v)$  or per-face  $R(f)$  transformation. Then  $R(v)$  or  $R(f)$  can help us to update the vertex positions and lead to a deformed shape  $\mathcal{S}' = (V', F)$ .

**Useful libigl functions:** `igl::readTGF` and `igl::readDMAT`. See the corresponding header files to know how to use them.

**1.2. Quaternions.** we can use a quaternion  $q = [x, y, z, w] \in \mathbb{R}^{4 \times 1}$  to represent a rotation in 3D:  $q = w + x\mathbf{i} + y\mathbf{j} + z\mathbf{k}$ , where  $\mathbf{i}, \mathbf{j}, \mathbf{k}$  are symbols that can be interpreted as unit-vectors pointing along the three spatial axes. We can rewrite  $q$  as  $q = [\cos \frac{\theta}{2}, \sin \frac{\theta}{2} \mathbf{v}]$ , encoding a rotation of  $\theta$  radians about the unit axis  $\mathbf{v}$ . More details can be found in [this paper \[1\]](#) or [this blog](#). Quaternion is a double cover of the rotation group  $\mathbb{SO}(3)$ : each rotation in  $\mathbb{SO}(3)$  can be represented by two quaternions  $q$  and  $-q$ . An intuitive way to understand it is rotating about  $\mathbf{v}$  by  $\theta$  and rotating about  $-\mathbf{v}$  by  $2\pi - \theta$  will give the same rotation, which corresponds to  $q$  and  $-q$ .

**1.3. SLERP: Spherical linear interpolation.** Given two rotations  $R_a$  and  $R_b$ , how can we interpolate them to find intermediate rotations? One trivial solution is to interpolate the rotation matrices by  $R_t = R_a + t(R_b - R_a), t \in [0, 1]$ . Actually, this is what is used in the linear blend skinning technique (Sec. 5). However, there is no guarantee that  $R_t$  is a rotation matrix.

To address this issue, we can use quaternions to represent the rotations and use SLERP to interpolate the quaternion directly:

$$(1) \quad q_t = q_a (q_a^{-1} q_b)^t, \quad t \in [0, 1]$$

One can easily verify that  $q_0 = q_a, q_1 = q_b$  and for any  $t$ ,  $q_t$  is a quaternion that encodes an intermediate rotation. We will use SLERP to interpolate skeleton animation (Sec. 3).

**1.4. Averaging rotations.** Given a set of rotations  $R_1, \dots, R_n$  and corresponding weights  $w_1, \dots, w_n$ , how can we compute the average rotation? Slerp works well when  $n = 2$ , but during skeletal deformation, specifically per-face blend skinning with Poisson stitching, we need to average a set of rotations. See [this paper \[5\]](#) (Equation 12, 13) for a detailed discussion on finding the average rotation under the Frobenius norm.

**1.5. Dual Quaternions.** The ordinary quaternion can represent a 3D rotation. We need to use [dual quaternion](#) to represent general transformations that include both rotations and translations. A dual quaternion can be represented as  $q = q_r + q_d \epsilon$ , where  $\epsilon$  is the dual operator, and  $q_r, q_d$  are two ordinary quaternions. Specifically for a rigid 3D transformation including a rotation  $q_0$  (represented as an ordinary quaternion) and a translation  $t = (t_1, t_2, t_3)$ , it can equivalently be represented using a dual quaternion  $q = q_0 + \frac{\epsilon}{2} (t_1 \mathbf{i} + t_2 \mathbf{j} + t_3 \mathbf{k}) q_0$ . See [this paper \[3\]](#) and [Sec.3.2. of this paper \[2\]](#) for a more detailed discussion. We can interpolate the dual quaternions to propagate transformation from joints/bones to the mesh vertices (Sec. 6).

**1.6. Assignment Overview.** The goal of this assignment is to transfer the animation (defined as transformations w.r.t. the rest pose) from the bones to the corresponding mesh, where we “average” or “linearly” interpolate a set of transformations and assign it to the mesh. To properly formulate this problem, we need to clarify the following three key aspects:

- **Per-vertex v.s. per-face:** are the transformations transferred from bones to the mesh vertices or mesh faces? For the former case, the transferred transformations can be used to modify vertex positions and thus get the mesh animation directly. For the latter case, transforming each face individually can lead to broken mesh and thus a post-processing step called Poisson Stitching is required to fix the mesh.
- **How transformations (especially the rotational part) are represented?** As discussed above, there are different ways to represent the 3D rotations, which can lead to different results after interpolation.
- **How to interpolate and average rotations?** We can use Lerp (linear interpolation), Nlerp (normalized Lerp), and Slerp (spherical Lerp) to interpolate a set of rotations. Specifically, Lerp is a standard linear interpolation in the chosen rotation space, which is not guaranteed to be a rigid transformation. Nlerp can be regarded as projecting the Lerp results into the chosen rotation space (e.g., making it a orthonormal matrix if  $\mathbb{SO}(3)$  is chosen, or making it unit length if rotations are represented as quaternions). Slerp can be regarded as a linear interpolation of the angles, therefore it is independent of the rotation representations.

To summarize, we can combine different settings to transfer the animations from bones to the mesh. In this assignment, you are requested to particularly experiment three settings: (1) per-vertex +  $\mathbb{SO}(3)$  + Lerp (see Sec. 5) (2) per-vertex + quaternion representation + Nlerp (see Sec. 6); (3) per-face + quaternion representation + averaging rotations (see Sec. 7).

## 2. ROTATION REPRESENTATIONS: THEORETICAL QUESTIONS [3.5 points]

Consult [this paper \[1\]](#) and [this paper \[3\]](#) to understand the different kinds of rotation representations. You will learn about *Euler angles*, *rotation matrices*, *quaternions*, and *dual quaternions*.

### Required Output:

- (1) A discussion comparing the four aforementioned different approaches. Explain the pros and cons of each method.
- (2) Explain how to switch between these rotation representations: how to go from Euler angles to rotation matrices, from rotation matrices to quaternions, and from quaternions with a translation  $\mathbf{t} = (t_x, t_y, t_z)$  to dual quaternions.

## 3. SKELETAL ANIMATION VISUALIZATION [1.5 point]

First, you will visualize the animation of the skeleton. You are provided with two types of rotations: a sequence of rotations and a target rotation. All the target rotations files are with suffix `_target`.

For sequences, you are given the set of rotations  $\{\tilde{R}_k^{(l)} |_{k=1}^K\}$  for each frame  $l$ . Note that  $\tilde{R}_k^{(l)}$  is the relative rotation w.r.t. its source joint node. We therefore need to recover the absolute rotation  $\hat{R}_k^{(l)}$  and absolute translation  $T_k^{(l)} \in \mathbb{R}^{3 \times 1}$  w.r.t. the root node using [Forward Kinematics](#). Then we can get

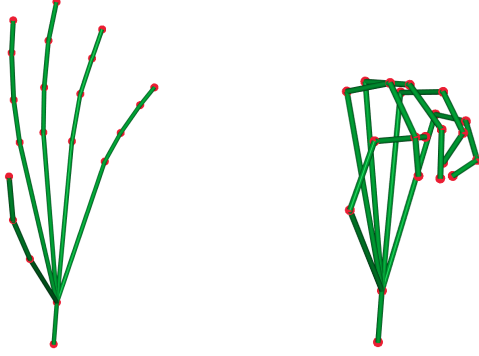


FIGURE 1. Skeleton visualization in rest pose and target pose for the hand. Rendered with the help of `libhedra`'s `line_cylinders` function.

the position of the joint  $b_k$  at the  $l$ -th frame as  $\mathbf{x}_k^{(l)} = \hat{R}_k^{(l)} \mathbf{x}_k + T_k^{(l)}$ , where  $\mathbf{x}_0$  is the position of the root node. Visualizing the skeleton graph with positions  $\{\mathbf{x}_k^{(l)}\}_{k=1}^K$  for all the frames produces an animation.

For target rotation, only the final pose is given, which means we need to compute the joint positions at each frame and then visualize the 3D graph of the skeleton. Specifically, for each bone  $b_k$ , we only have the quaternion rotation at the final pose  $q_k^{(L)}$ . We can first apply SLERP to interpolate the identity rotation (the initial pose) and  $q_k^{(L)}$  (the final pose) to get  $q_k^{(l)}$ . Note that these quaternions are again relative rotations w.r.t. the parent joint. We therefore need to recover the absolute transformation first to get the joint position as discussed above. No need to reimplement the same algorithm as for rotations matrices: you can directly use the provided function `igl::forward_kinematics` for quaternions after performing the SLERP.

**Required Output:** Visualize the animation of the input skeleton (in video or gif) from the two types of rotations (rotation matrices and quaternions).

**Useful libigl functions:** To visualize the skeleton, you can use `viewer.data().set_edges` and `viewer.data().set_points`. Set the option `viewer.data().show_overlay_depth = false`; to make the lines and points always visible. Some provided libigl functions like `igl::directed_edge_parents` or `igl::forward_kinematics` are relevant. To produce an animation, you can check the callback function `viewer.callback_pre_draw`.

**Known bug:** macOS does not support fully OpenGL anymore, therefore it is not possible to set edge line width directly for these OS (OpenGL's directive `glLineWidth` does not support values different from 1.0f). You can use a workaround by outputting cylinders instead. You can use `libhedra`'s `line_cylinders` function to generate these cylinders.

#### 4. COMPUTE HARMONIC SKINNING WEIGHTS [3 points]

To deform the rest shape  $\mathcal{S}$  to conform to a given skeleton pose, we need to compute skinning weights, i.e., associating a continuous scalar field  $w_k \in \mathbb{R}^{n \times 1}$  for each bone  $b_k$ . Specifically, each vertex can be

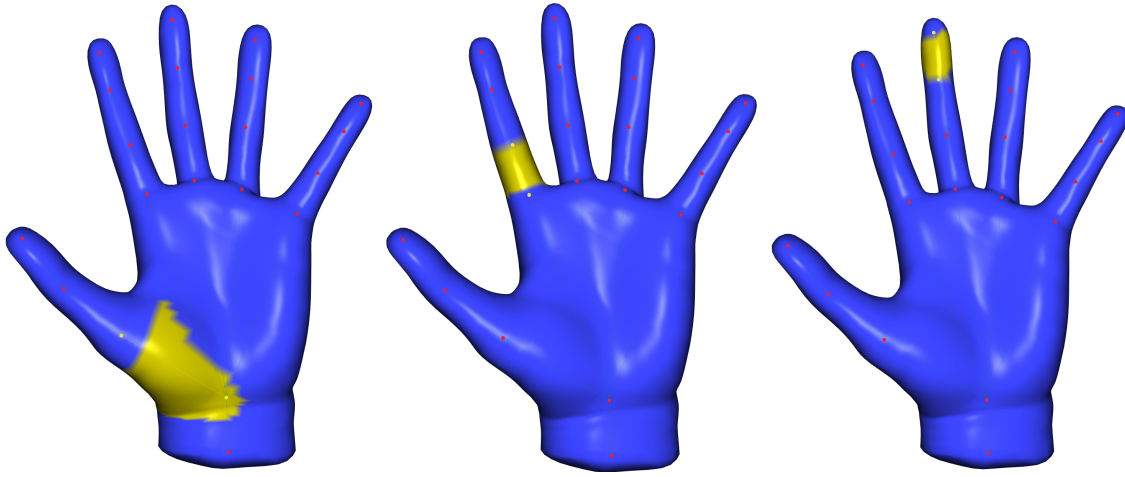


FIGURE 2. Handle selection

triggered to move by bones. The percentage of the influences from the  $k$ -th bone  $b_k$  is specified by the skinning weight  $w_k$ .

We follow [6] (see a more detailed discussion in Sec. 2.2) to compute harmonic skinning weights: the vertices that "belong" to the bone handle  $H_k$  are assigned the value  $w_k(v) = 1$ , while vertices belong to other bone handles are assigned the value  $w_k(v) = 0$ . All other vertices are assigned values  $w_k(v) \in [0, 1]$  obtained as discrete harmonic functions over the mesh. Specifically, we can compute these values for each mesh vertex by solving the Laplace equation:

$$(2) \quad Lw_k = 0 \quad \text{subject to} \quad w_k(v) = 1 \forall v \in H_k, w_k(v) = 0 \forall v \in H_j \text{ where } j \neq k$$

We can compute such a skinning weight  $w_k$  for each bone  $b_k$ , where  $k = 1, \dots, K$ . Note that  $w_k(v)$  is defined on mesh vertices  $v$ . We can define the skinning weights  $h_k(f)$  for each triangle  $f$  by averaging the values of  $w_k$  at the three vertices of  $f$ . We will use  $w_k(v)$  and  $h_k(f)$  for per-vertex and per-face blend skinning later.

**Sanity Check**  $\sum_{k=1}^K w_k(v) = 1 \forall v$  since  $w_1, \dots, w_K$  are a partition of unity over the mesh.

**4.1. First Task - handle selection.** To trigger the mesh deformation by its skeleton deformation, we need to attach to each bone  $b_k$  a set of vertices  $v \in H_k$  such that their weight is  $w_k(v) = 1$ . These weights act as the Dirichlet boundary conditions (fixed value). In this assignment, we only need to associate a small set of vertices on the mesh to each bone and compute the harmonic skinning weights automatically. You are requested to design a method for handle selection and store the bone handle in  $H_k$  for each bone  $b_k$ . We provide you with an example of handles for reference. Invent your own solution. Here are two possibilities, but you can come up with a more sophisticated approach:

- Manual selection: use the Lasso selection tool (see assignment 5) to select handle vertices for each bone.

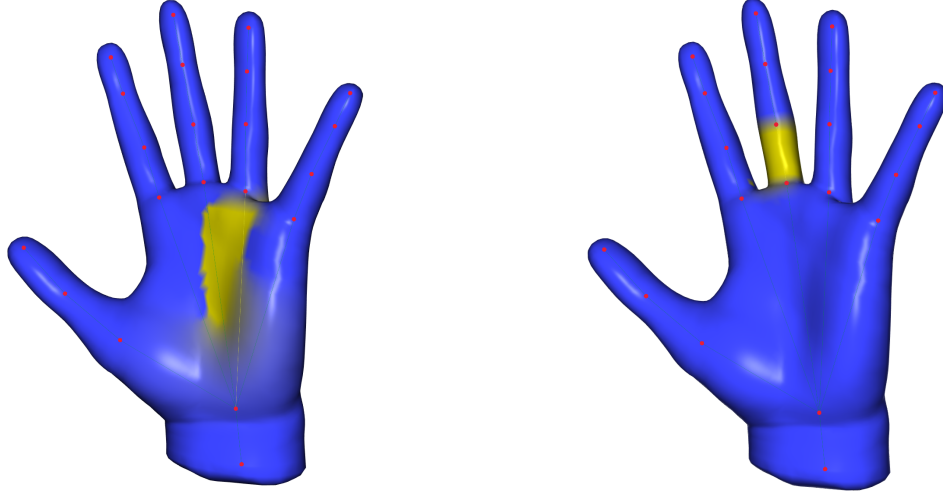


FIGURE 3. Harmonic weights

- Geometric selection: for each bone, add all the vertices that are closer to the bone than 1.5 times the distance between the bone and the closest vertex of the mesh. You can also create a cylinder around the bone and include only vertices which are close enough.

**Required Output:** Visualize the bone handles on the mesh by coloring the mesh vertices according to the bone ID. Show the coloring for 3 different bones. Compare your technique with the provided solution.

4.2. **Second task - harmonic skinning weights.** Solve the Laplacian equation (2) to compute the weights  $w_k$  for the whole mesh.

**Required Output** Visualize the skinning weight function  $w_k$  of bones on the input mesh by coloring the mesh. Show the coloring for 3 different bones.

**Useful libigl functions:** `igl::slice_into`, `igl::slice`.

## 5. PER-VERTEX LINEAR BLEND SKINNING [4 points]

Now we already know the absolute rotation  $R_k^{(l)}$  and translation  $T_k^{(l)}$  of the  $k$ -th bones at  $l$ -th frame. We can propagate the transformation from the bones to the mesh vertices using the per-vertex skinning weight  $w_k$  we computed:

$$(3) \quad \mathbf{v}_i^{(l)} = \sum_{k=1}^K w_k(i) (R_k^{(l)} \mathbf{v}_i^{(0)} + T_k^{(l)})$$

where  $\mathbf{v}_i^{(l)}$  is the 3D position of the  $i$ -th vertex on the mesh at  $l$ -th frame;  $w_k(i)$  gives the skinning weight of the  $k$ -th bone on the  $i$ -th vertex,  $\mathbf{v}_i^{(0)}$  is the vertex position on the rest pose. This equation means that we linearly combine the transformation on the bones based on the skinning weights and apply the new transformation to the vertex to get its new position. Follow this equation to compute the

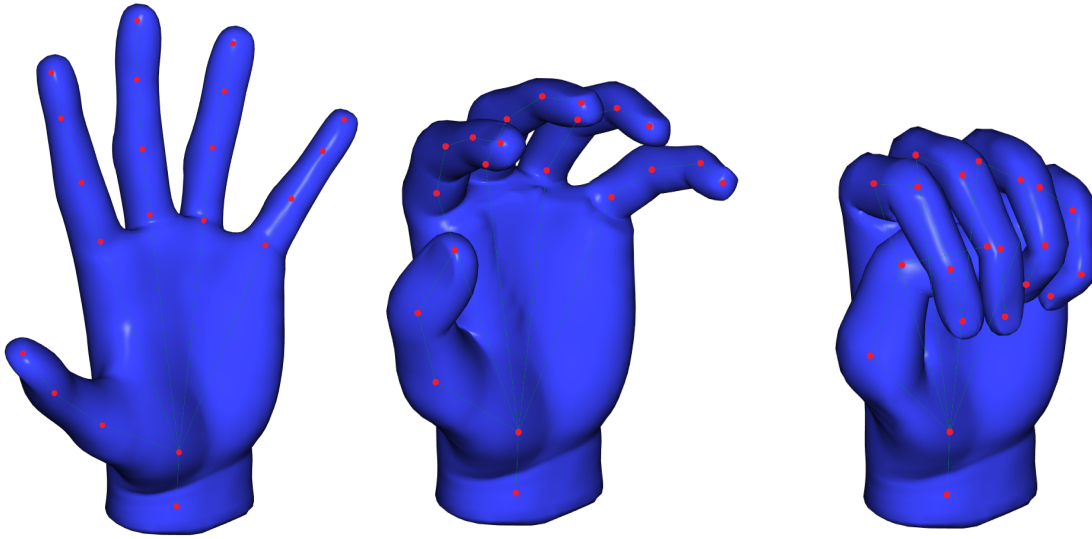


FIGURE 4. Linear blend skinning

new position for each vertex and each frame would give us an mesh animation enabled by the skeletal animation.

**Required Output** Visualize the animation of the mesh using per-vertex Linear Blend Skinning.

## 6. DUAL QUATERNION SKINNING [2 points]

Here we try another method to interpolate the transformations among the bones. We first represent both rotations  $R_k^{(l)}$  and translations  $T_k^{(l)}$  using a dual quaternion  $q_k^{(l)}$ . To get the transformation for each vertex, we linearly interpolate the dual quaternions based on the skinning weights and normalize the resulting transformation:

$$(4) \quad q_i^{(l)} = \frac{\sum_{k=1}^K w_k(i) q_k^{(l)}}{\left\| \sum_{k=1}^K w_k(i) q_k^{(l)} \right\|}$$

The normalization (known as Nlerp) forces the result of the linear blend to be a unit dual quaternion, which leads to a rigid transformation on the mesh vertices. However, you should not use the equation as is, because averaging quaternions on different hemispheres does not go through the geodesic on the  $\mathbb{S}^3$ . Instead, when adding two quaternions together, you should always check if they are on the same hemisphere, if not, negate one of them.

**Required Output** Visualize the animation of the mesh using Dual Quaternion Skinning and compare it with the previous methods.

**Useful libigl functions** You can use `igl::dqs` as a starting point. However, `igl::dqs` does not deal with the double-cover problem and directly using it will lead to artifacts.

## 7. PER-FACE BLEND SKINNING [4 points]

**7.1. Per-face Rotation Blend.** We can use the per-face skinning weights  $h_k$  to assign a blended rotation from the bones to each face. Instead of using linear blend, we use averaging rotations [5] as discussed in Section 1.4. Specifically, we can compute the average rotation for each face  $f$  as:

$$(5) \quad \tilde{q}_f^{(l)} = \arg \max_{q \in \mathbb{S}^3} q^T \left( \sum_{k=1}^K h_k(f) q_k^{(l)} q_k^{(l)T} \right) q$$

where  $q_k^{(l)}$  is the  $4 \times 1$  vector representing the quaternion of the absolute rotation of the bone  $b_k$  at  $l$ -th frame w.r.t the rest pose. After obtaining the per-face rotation  $\tilde{q}_f^{(l)}$  for the  $f$ -th face, we can apply it to the face triangle on the rest shape to get its new position in  $l$ -th frame. However, since each face is transformed independently, it is not likely we can get a fully connected mesh. We therefore need to apply Poisson Stitching to the transformed triangles to get the deformed mesh.

**7.2. Poisson Stitching.** The idea behind Poisson Stitching is simple: we would like to solve for new positions for the vertices, such that the deformation gradient agrees with the face transformation  $\tilde{q}_f^{(l)}$  we just computed:

$$(6) \quad G \begin{pmatrix} \mathbf{v}_1^{(l)} \\ \dots \\ \mathbf{v}_n^{(l)} \end{pmatrix} = \begin{pmatrix} \tilde{R}_1^{(l)} \\ \dots \\ \tilde{R}_m^{(l)} \end{pmatrix}$$

where  $G \in \mathbb{R}^{3m \times n}$  is the gradient operator which assigns a constant gradient to each face from a per-vertex function,  $\mathbf{v}_i^{(l)}, i = 1, \dots, n$  is the to-be-solved vertex position for the  $i$ -th vertex on  $l$ -th frame,  $\tilde{R}_f^{(l)}, f = 1, \dots, m$  is the face deformation gradient, which is the matrix form of the blended face rotations  $\tilde{q}_f^{(l)}$  in this case.

Since the gradients are translation-invariant, we also add Dirichlet boundary conditions to the vertices associated with the root bone  $b_1$ :  $\mathbf{v}_i^{(l)} = \mathbf{v}_i^{(0)}, \forall i \in H_1$ .

**Hint** This is similar to what you have done for deformation transfer in Assignment 5.

**Required Output** Visualize the animation of the mesh using per-face LBS. Discuss the different methods.

## 8. USING CONTEXT FROM EXAMPLES [4 points]

We can further incorporate additional information about the shape's characteristics from the provided examples (i.e., the same shape in different poses) to achieve better shape deformation. Specifically, for the input rest shape  $\mathcal{S}_0 = (V_0, F)$ , we also provide some examples of the same shape in different poses:  $\mathcal{S}_j = (V_j, F), j = 1, \dots, J$  with pose skeleton parameters  $P_j$ . Note that these shapes are in the same triangulation but with different vertex positions to encode different poses.



**Overview** In the following, we will discuss (1) how to unpose an example shape; (2) how to define the shape differences between examples and the rest shape; (3) how to assign weights to different examples, and how to add the displacements extracted from the examples to improve the deformation. Similarly, we will discuss the per-vertex and per-face setting separately.

### 8.1. Per-vertex context-aware deformation.

8.1.1. *Unposing the example shapes (per-vertex).* It is relatively direct to unpose an example shape to the rest pose when considering per-vertex defined deformations. Specifically, for an example shape  $S_j = (V_j, F)$  with known vertex positions  $V_j$  and pose parameters (rotations)  $P_j$ , we can assume its unposed shape has vertex positions  $\bar{V}_j$ , which satisfies  $\mathbf{v}_{j,i} = T(P_j, w_k(i))\bar{\mathbf{v}}_{j,i}$ , where  $\mathbf{v}_{j,i}$  is the position of the  $i$ -th vertex on the  $j$ -th example, and  $\bar{\mathbf{v}}_{j,i}$  is the to-be-solved position after unposing. One can easily solve for the new positions via  $\bar{\mathbf{v}}_{j,i} = T^{-1}(P_j, w_k(i))\mathbf{v}_{j,i}$ .

8.1.2. *Extracting deformation nuances from the examples.* Now  $\bar{S}_j$  is directly comparable to the rest shape  $S_0$  since they are in the same pose space with rotations factored out. We then defined the per-vertex displacements  $\delta_{j,i} = \bar{\mathbf{v}}_{j,i} - \mathbf{v}_{0,i}$  from the unposed example to the rest shape.

8.1.3. *Context-aware deformation.* For an arbitrary pose with parameters  $P$ , we can compute the new deformed positions by adding the context from examples to the standard LBS:

$$(7) \quad \mathbf{v}_i = T(P, w_k(i)) \left( \mathbf{v}_i^{(0)} + \sum_{j=1}^J a_j(P) \delta_{j,i} \right)$$

where  $a_j(P)$  assigns a weight for the  $j$ -th example based on how similar the pose  $P$  is to the example, and note that  $a_j$  is the same for all the vertices. The only question left is how to compute such weight functions  $a_j(\cdot)$ . It is natural to have the following constraints on  $a_j(P)$ :

- (1)  $\sum_{j=1}^J a_j(P) = 1$ , as a linear interpolation among the pose space formed by the  $J$  examples
- (2)  $\forall j, a_j(P_j) = 1$  and  $a_j(P_i) = 0$  if  $i \neq j$ , which means for a pose  $P = P_j$  that has the same parameter as the provided example, the deformed shape computed from Eq. (7) should be the same as the provided example.
- (3)  $a_j(P)$  is continuous in the pose space to produce smooth animation, i.e.,  $a_j$  is a continuous function w.r.t.  $P$ .

To satisfy the continuity constraint, we can express  $a_j$  using  $J$  radial basis functions (RBF)  $\{\phi(\|P - P_j\|)\}_{j=1}^J$ , that centered at  $P_j$  respectively. Therefore, we can express  $a_j(P) = \sum_{t=1}^J c_{j,t} \phi(\|P - P_t\|)$ . The  $J$  unknowns  $c_{j,t}$  can be easily solved from the  $J$  constraints:  $a_j(P_j) = 1$  and  $a_j(P_i) = 0$  if  $i \neq j$ . To satisfy the sum-to-one constraint, we just need to rescale  $a_j(P)$ :

$$(8) \quad a_j(P) \leftarrow \frac{a_j(P)}{\sum_{j=1}^J a_j(P)}.$$

See Sec.5.2. in [this paper](#) [4] for more discussions. Now for an arbitrary pose  $P$  we can compute a better deformation using the provided examples following Eq. (7).

**Required Output** (1) Visualize 3 of all the unposed examples  $\bar{S}_j = (\bar{V}_j, F)$  that you will use; (2) Visualize the animation using context from examples and compare it to standard LBS.

**8.2. (Optional) Per-face context-aware deformation** [6]. *This section is optional. Its completion can lead up to two bonus points.*

**8.2.1. Unposing the example shapes (per-face).** We can similarly unpose the provided examples in a per-face fashion. For each triangle  $f \in F$ , we can compute a rotation from the rest shape  $S_0$  to the example pose  $S_j$ . We then apply the *inverse* of the rotation to each triangle on the example pose  $S_j$  and use the Poisson Stitching technique (discussed in Sec. 7.2) to get the unposed shape  $\bar{S}_j$ .

**8.2.2. Extracting deformations nuances from examples.** Now all the unposed shapes  $\bar{S}_j$  are in the same pose as the rest shape  $S_0$  where the rotations due to pose have been factored out. We can then compute the deformation gradient  $T_{j,f}$ , of the  $f$ -th face on the  $j$ -th unposed example  $\bar{S}_j$  w.r.t. the rest shape  $S_0$ . The full representation of the characteristic behavior present in the  $j$ -th example  $S_j$  is the provided skeleton pose  $P_j$  and computed relative transformations  $\{T_{j,1}, \dots, T_{j,m}\}$ . You can regard  $T_{j,f}$  as "displacement" (represented by a transformation) defined on faces, in a similar way to  $\delta_{j,v}$  the "displacement" (represented by a displacement vector) defined on vertices as introduced in Sec. 8.1.2.

**8.2.3. Context-aware deformation.** Similarly to what is discussed in Sec. 8.1.3, we would like to use the face "displacements",  $T_{j,f}$ , to improve the deformations. The key difference is that the vertex "displacements"  $\delta_{j,i}$  are used to modify the deformed vertex positions directly, but now we would like to use  $T_{j,f}$  to modify the face transformations  $R(f)$ . A direct solution is:

$$(9) \quad R(f)' = R(f) \sum_{j=1}^J a_j(P) T_{j,f}$$

where  $R(f)$  is the standard per-face LBS as discussed in Sec. 7,  $a_j(P)$  is similar weight function as computed in Sec. 8.1.3.

As discussed in Sec.2.2. in [this paper](#) [6], the linear blend of  $T_{j,f}$  may lead to visual artifacts since it might contain rotational components. Therefore, [6] proposes to decompose  $T_{j,f} = Q_{j,f} S_{j,f}$  into a rotational part  $Q_{j,f}$  and a skew component  $S_{j,f}$ , leading to a better modification:

$$(10) \quad R(f)' = R(f) \left( \oplus_{j=1}^J a_j(P) Q_{j,f} \right) \left( \sum_{j=1}^J a_j(P) S_{j,f} \right)$$

where  $\oplus$  represents a linear combination of the rotations in log-quaternion space (see Eq. (5)).

With the improved per-face transformations  $R(f)$  based on the context from the examples, we can apply the Poisson Stitching to the transformed faces to get the final deformed shapes.

**Required Output** (1) Visualize all the unposed examples  $\bar{S}_j = (\bar{V}_j, F)$ ; (2) Visualize the animation using context from examples using Eq. (9); (3) Visualize the animation using context from examples

using Eq. (10); (4) Analyze the results between using Eq. (9) and using Eq. (10), and compare the per-face context deformation to the per-vertex context deformation.

#### REFERENCES

- [1] F Sebastian Grassia. Practical parameterization of rotations using the exponential map. *Journal of graphics tools*, 3(3):29–48, 1998.
- [2] Ladislav Kavan, Steven Collins, Jiří Žára, and Carol O’Sullivan. Skinning with dual quaternions. In *Proceedings of the 2007 symposium on Interactive 3D graphics and games*, pages 39–46, 2007.
- [3] Ben Kenwright. A beginners guide to dual-quaternions: what they are, how they work, and how to use them for 3d character hierarchies. 2012.
- [4] Tsuneya Kurihara and Natsuki Miyata. Modeling deformable human hands from medical images. In *Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 355–363, 2004.
- [5] F Landis Markley, Yang Cheng, John L Crassidis, and Yaakov Oshman. Averaging quaternions. *Journal of Guidance, Control, and Dynamics*, 30(4):1193–1197, 2007.
- [6] Ofir Weber, Olga Sorkine, Yaron Lipman, and Craig Gotsman. Context-aware skeletal shape deformation. In *Computer Graphics Forum*, volume 26, pages 265–274. Wiley Online Library, 2007.