Volume 33 (2014), Number 2 pp. 1–13 COMPUTER GRAPHICS *forum*

ARAPLBS: Robust and efficient elasticity-based optimization of Weights and Skeleton Joints for Linear Blend Skinning with Parameterized Bones

J.-M. Thiery[1,2†] E. Eisemann[1‡]

[1]Deflt University of Technology, The Netherlands [2]LTCI, Telecom-ParisTech, Université Paris-Saclay, Paris, France

Abstract

We present a fast, robust and high-quality technique to skin a mesh with reference to a skeleton. We consider the space of possible skeleton deformations (based on skeletal constraints, or skeletal animations), and compute skinning weights based on an optimization scheme to obtain as-rigid-as-possible (ARAP) corresponding mesh deformations. We support stretchable-and-twistable bones (STBs) and spines by generalizing the ARAP deformations to stretchable deformers. Additionally, our approach can optimize joint placements. If wanted, a user can guide and interact with the results, which is facilitated by an interactive feedback, reached via an efficient sparsification scheme. We demonstrate our technique on challenging inputs (STBs and spines, triangle and tetrahedral meshes featuring missing elements, boundaries, self-intersections, or wire edges).

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Curve, surface, solid, and object representations

1. Introduction

Linear Blend Skinning (LBS) is a standard technique for skeleton-based animation and is used extensively in video games, movies, simulations, and virtual-reality systems. It is popular because artists use skeletons to animate characters and because of its efficiency: it only requires blending bone transformations using precomputed *skinning weights*.

Looking at typical meshes and skeletons produced by artists (Fig. 1, left), it is clear that automatic weight computation is challenging: **i:** Models can consist of disconnected parts, which might not be manifold, or orientable. **ii:** The skeletal topology might differ from the mesh topology (skeletons are conceived to facilitate animating, not to follow geometric constraints). **iii:** The skeleton can be complex.

Typically, skinning weights should fulfil some properties (e. g., sum to 1). Approaches based on (k-)harmonic or general PDEs ensure these on the mesh or on an embedding, while enforcing *smoothness*. Although smooth weights are necessary, they do not ensure high-quality deformations.

Our approach produces weights to optimize deformation quality. Building upon *as-rigid-as-possible* (ARAP) transformations to benefit from their properties (angle, edge length preservation), we construct a set of representative skeleton deformations, *exemplars*, sampled from the space of admissible transformations, and optimize the skinning weights jointly to achieve corresponding ARAP transformations. We can further specialize our weights for skeleton animations by adding them to our exemplars.

Our approach is general and robust. Although compati-

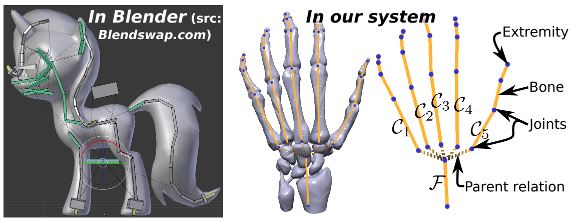


Figure 1: *We use standard skeleton definitions (left); composed of vertices (joints and extremities), and oriented segments (bones, or parent relations). E.g., $\mathcal{F}$ is parent of $\{\mathcal{C}_i\}$.*

† jean-marc.thiery@telecom-paristech.fr
‡ e.eisemann@tudelft.nl

ble with volumetric meshes, it does not rely on volumetric structures (which are sometimes impossible to generate with existing tools) for the weight computation on surface meshes. Our method handles challenging inputs, e. g., containing self-intersections, open boundaries, disconnected parts or wire-edges. It is even compatible with complex bones, e. g., stretchable-and-twistable bones (STBs) [JS11] and *spines*. Our solution can also optimize the skeleton joints w.r.t. the mesh for an optimal deformation quality, which can be a tedious task when performed manually.

Our method was conceived with efficiency and user-friendliness in mind, and we introduce a weight sparsification scheme, which results in a performance increase by several orders of magnitude. Hereby, the artist can adapt weights to influence the result during the optimization and obtains interactive feedback.

Our main technical contributions are:

- an efficient, robust, high-quality LBS definition based on a sampling of feasible skeleton deformations;
- a joint-placement algorithm to optimize the skeleton;
- a generalization of ARAP deformations for stretchable inputs: *as-rigid-as-desired* deformations.

1.1. Related Work

Several supporting structures can be used to deform a shape. Here, we focus on the most-related work on LBS weights optimization, and refer the reader to the excellent surveys [BS08, NS13, JDKL14] for further information.

Smoothness-based: Baran & Popovic' [BP07] produce so-called *Heat Bones Weights* (HBs) by solving an analogy to a volumetric heat-diffusion equation of the bones' basis functions. Here, the Voronoi cells of the bones act as the diffusion structure. Jacobson et al. [JBPS11] instead use a tetrahedrization to solve for a biharmonic diffusion of the bones' influence with positive and bounded variables, producing *Bounded Biharmonic Weights* (BBWs). In an improvement [JWS12], they prevent local maxima in the solution. Finally, Dionne et al. [DdL13] create a robust voxelization to skin poorly-meshed surfaces. While offering ways to drive the diffusion, volumetric solutions are generally more costly, due to the higher number of discrete elements, and currently difficult to obtain for many real-life meshes.

Elasticity-based: Kavan & Sorkine [KS12] were the first to combine the concepts of ARAP and LBS/DQS deformations. Their contribution is a joint-based deformer, which has two parameters influencing the mesh vertices: the amount of twist and the amount of swing that is applied by this deformer in the local basis attached to the child bone. These two parameters per deformer are optimized by minimizing the ARAP energy for representative deformations of the deformer, and all deformers are optimized one after the other. After that, all deformations of individual deformers are blended, using precomputed BBWs, which act as skinning weights. They argue that all complex deformations are essentially located around the joints, and since these complex deformations are handled by the twist and swing deformer individually, they can keep the BBWs, blending the various deformations, untouched. In contrast, our contribution is not a specific new deformer but an optimization of the blending (i.e., the skinning weights) of the deformers in order to obtain ARAP mesh deformations. As them, we can avoid the candy-wrapper effect. We achieve this by supporting specialized deformers such as STBs and spines, which is made feasible even for highly-stretchable controls by introducing as-rigid-as-desired transformations, which extend ARAP ones naturally. Our choice is motivated by the desire to encourage artistic flexibility, as each bone type can be specified individually. Further, we rely on standard bone definitons to facilitate the integration into existing frameworks. Additionally, our experience is that the required BBWs are difficult to obtain for many challenging input models. Our strategy provides adequate results where existing methods failed, which we illustrate with real-life models (`blendswap.com`).

Example-based methods approximate a *mesh sequence* via LBS [dATTHP08, KSO10, LD12]. Some, e. g., [LD14], output an automatically-created skeleton as well. Instead of skeleton deformations, one could try generating *mesh poses* to obtain LBS weights using such techniques. However, generating mesh poses capturing all degrees of freedom is challenging and seems feasible for watertight models only.

1.2. Technical Background

Skeleton Topology and Constraints A skeleton (Fig. 1) is composed of *(skeletal) vertices and segments*, which can be *bones* (affecting the mesh), or *virtual segments* (used to define parent-child relations only). A vertex linked to several segments is called a *joint*. Bones can have restrictions like maximal angles or imposed rotation axes.

Linear Blend Skinning (LBS) with parameterized bones [MTLT*88, LCF00] transforms a vertex i as

$$f : v_i \mapsto \sum_{j \in B(i)} w_{ij} (R_j(v_i) \cdot v_i + T_j(v_i)), \qquad (1)$$

where $R_j(v_i)$ is the rotation and $T_j(v_i)$ the translation applied by bone j on vertex i (these are constant for rigid bones – $(R_j(v_i), T_j(v_i)) = (R_j, T_j)\ \forall i$, but depend on v_i otherwise), $B(i)$ is the set of bones influencing i and w_{ij} the *weight* of bone j over i. We will refer to $\{B(i)\}$ as the *bone influence maps* and to $\{w_{ij}\}$ as the *weight maps*.

Typically, skinning weights should verify:

1. **Affinity**: $\sum_j w_{ij} = 1$; reproduces rigid transformations.
2. **Positivity**: $w_{ij} \geq 0$; prevents unnatural behavior.

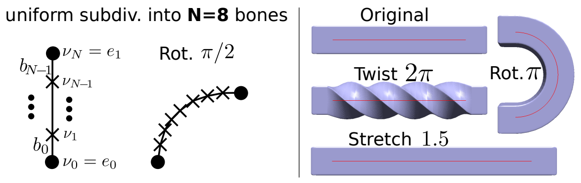


Figure 2: ***Left:*** *Approximation.* ***Right:*** *Continuous setting.*

3. **Sparsity**: only *"few"* $w_{ij} > 0$; leads to *"simpler"* controls and a faster rendering process *(ill-defined)*.
4. **Locality**: bones should have *"small"* influence zones; improves control over editing operations *(ill-defined)*.

Spines STBs [JS11] are a popular example of parameterized bones. Our approach is compatible with these, as well as a generalization, *spines*, described here. A *spine* **s** is a skeletal segment $[e_0, e_1]$ subdivided into infinitely-small bones undergoing the same transformation (Fig. 2). The name reflects its suitability to represent spines in models. Fixing e_0, its transformation combines a stretch $\sigma_\mathbf{s}$, affecting its length, and a rotation around an axis $a_\mathbf{s}$, such that the accumulated rotation from e_0 to e_1 amounts to $\theta_\mathbf{s}$. Similar to other bones, a spine **s** also has a rigid transformation applied to e_0, the *spine's base transformation* $(R_\mathbf{s}, t_\mathbf{s})$. In consequence, spines have 5 parameters: $(R_\mathbf{s}, t_\mathbf{s}, \sigma_\mathbf{s}, \theta_\mathbf{s}, a_\mathbf{s})$, and STBs are twist-restricted spines (i. e., $a_\mathbf{s} = \overrightarrow{e_0e_1}/||\overrightarrow{e_0e_1}||$ in their case).

A point p with parameter $u_p \in [0,1]$ (describing "which small bone" p is attached to) is transformed by a spine as

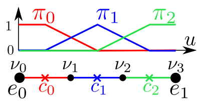
$$p \mapsto R_\mathbf{s} R_{\text{loc}}(u_p) \cdot p \quad + \quad R_\mathbf{s} \cdot t_{\text{loc}}(u_p) + t_\mathbf{s} \quad \text{, with} \tag{2}$$

$$R_{\text{loc}}(u) := Rot(a_\mathbf{s}, u\theta_\mathbf{s}), \text{ and} \tag{3}$$

$$\begin{aligned} t_{\text{loc}}(u) := {} & (\text{Id} - R_{\text{loc}}(u)) \cdot e_0 + \\ & \sigma_\mathbf{s}.(\sin(u\theta_\mathbf{s})/\theta_\mathbf{s} - u\cos(u\theta_\mathbf{s}))(\text{Id} - a_\mathbf{s} \cdot a_\mathbf{s}^{\text{T}}) \cdot \overrightarrow{e_0e_1} + \\ & \sigma_\mathbf{s}.(u \sin(u\theta_\mathbf{s}) + (\cos(u\theta_\mathbf{s}) - 1)/\theta_\mathbf{s}) a_\mathbf{s} \times \overrightarrow{e_0e_1} + \\ & u.(\sigma_\mathbf{s} - 1) R_{\text{loc}}(u) \cdot \overrightarrow{e_0e_1}. \end{aligned} \tag{4}$$

Given a spine **s** with vertices e_0, e_1, one needs to define the above parametrization $u_\mathbf{s} : \mathcal{V} \mapsto [0,1]$ on the mesh vertices $\mathcal{V}$. It is possible to use complex (or artistically-driven) definitions [JS11], but by default, we use a linear parameterization: $u_\mathbf{s}(v_i) = \max(0, \min(1, \overrightarrow{e_0v_i}^{\text{T}} \cdot \overrightarrow{e_0e_1}/||\overrightarrow{e_0e_1}||^2))$.

If a system only supports rigid bones, **s** can be converted into a set of them: **s** is cut into n bones of equal length with joints $\nu_0 \cdots \nu_n$, and for each ν_j its corresponding u-parameter u_j is set as j/n. We define a unity partition as piecewise-linear functions $\{\pi_j : [0,1] \mapsto [0,1]\}$, where $\pi_j(c_k) = \delta_j^k$, with $c_j = (u_j + u_{j+1})/2$, and impose $\pi_0(0) = \pi_n(1) = 1$ (the inset shows the case for 3 bones). The weight of a mesh vertex v_i (with weight $w_{i\mathbf{s}}$ w.r.t. the spine) w.r.t. bone k is defined as $w_{i\mathbf{s}} \pi_k(u_\mathbf{s}(v_i))$.



Note that Forstmann et al. [FOKGM07] introduce a spline-based deformer, which could perhaps be used in place of our spine deformer. However we are not aware of the existence of closed-form expressions describing both its deformation and its derivatives, as we require in our work.

As-rigid-as-possible (ARAP) transformations An ARAP transformation [SA07] f minimizes the energy:

$$\mathfrak{E}(f, \mathcal{R}) := \sum_{i \in \mathcal{V}} \Lambda_i \overbrace{\sum_{k \in V_1(i)} \lambda_{ik} || \underbrace{f(v_k) - f(v_i)}_{=: \bar{e}_{ik}} - \mathcal{R}(i) \cdot \underbrace{(v_k - v_i)}_{=: e_{ik}} ||^2}^{=: \mathfrak{E}_i(f, \mathcal{R}(i)), \text{ the rigidity of vertex } i \text{ assuming } \mathcal{R}(i)} \tag{5}$$

where $V_1(i)$ is the one ring neighborhood of i, $\mathcal{R}(i)$ a 3×3-matrix associated with i, Λ_i is a(n input) vertex weight and λ_{ik} a(n input) weight for the (directed) edge e_{ik} from vertex i to k to account for the local discretization of the mesh.

2. Overview

Input to our method is a polygonal mesh and a skeleton (with possible deformation restrictions). The output are skinning weights, which employed with LBS (Eq. 1) minimize the ARAP energy (Eq. 5) over a set of skeleton deformations. Fig. 3 illustrates the main steps of our algorithm.

During **initialization**, we produce a rough weight estimate based on the rest pose. These are then sparsified concurrently to localize influence and improve performance. The user can provide manual indications if wanted. Finally, we determine a set of skeleton deformations (*exemplars*) by sampling feasible poses (or/and using provided skeleton animations).

The **weight-optimization** stage defines skinning weights, which result in ARAP transformations for the exemplars, by minimizing a quadratic energy. The solution is found iteratively by optimizing transformation matrices and weights alternatingly. During each iteration, we add constraints to enforce weight positivity. Upon convergence, the optimal weights have been determined.

The **joint optimization** is optional, and used to refine the input skeleton's joint positions using a similar strategy.

Since weights (resp. joints) impact the joint (resp. weight) solver's setup, switching between joint and weight optimizations requires a matrix refactorization. However, successive iterations of the same solver can be performed without.

3. System Initialization

3.1. Weight initialization

For triangle meshes, we initialize our weights using a variant of HBs [BP07], which were chosen for robustness. The weight vector $w_{\cdot j}^{\text{HB}} := (w_{0j}, \ldots, w_{N-1j})^{\text{T}}$ for bone j is found by solving:

$$-\Delta \cdot w_{\cdot j}^{\text{HB}} + H \cdot w_{\cdot j}^{\text{HB}} = H \cdot \chi_{\cdot j}, \tag{6}$$

Δ denoting the Laplacian matrix, and $\chi_{\cdot j}$ the indicative function of the Voronoi cell of bone j. H is a diagonal matrix,

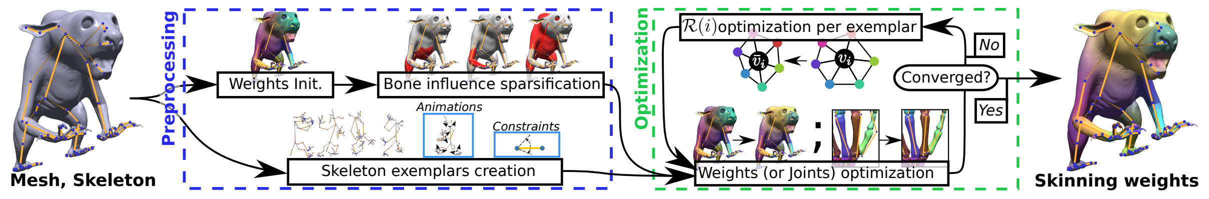

Mesh, Skeleton
Preprocessing
Weights Init.
Bone influence sparsification
Animations
Constraints
Skeleton exemplars creation
Optimization
$\mathcal{R}(i)$optimization per exemplar
v_i
v_i
No
Converged?
Yes
Weights (or Joints) optimization
Skinning weights

Figure 3: ***System overview:*** *Given a 3D mesh and skeleton, we output skinning weights. We produce skeleton* exemplars *respecting existing constraints such as angles range, and use frames of provided skeleton animations if desired. The weights are optimized such that the corresponding mesh deformations are ARAP. Optionally, the skeleton's joints can be optimized similarly.*

where $H_{ii} = \|v_i - p(i)\|^{-2}$, if the segment between v_i and its closest point on the skeleton $p(i)$ does not intersect the mesh (tested with a Kd-tree), otherwise $H_{ii} = 0$.

We introduce several modifications, to make $\{w^{\text{HB}}_{\cdot j}\}$ a good entry point for our algorithm. **i):** We add support for virtual segments. Fig. 4 (a) shows a partitioning using only skeleton bones. The zone of influence of the parent bone does not reflect the user's intent. We obtain a more natural transition by making the virtual segments part of the parent bone, thus extending their Voronoi cells (see Fig. 4 (b)). **ii):** The linear system (Eq. 6) can be non-invertible for challenging input. In this case, we simply set $H_{ii} = \|v_i - p(i)\|^{-2}$. These weights can be of poor quality (see Fig. 4 (c)), but will be optimized immediately (see Fig. 4 (d)).

3.2. Bone-Influence Reduction

We sparsify the bone influence maps *prior to the optimization*, while most other approaches enforce sparsity by adding the l_p (with $p \leq 1$) norm of the variables (e. g., [NVT*14]) to the energy. Our choice has two major advantages: **i)** The linear system becomes small and sparse, leading to interactive rates, and **ii)** constraints can be added during optimization with immediate feedback; an artist can use this information to control sparsity and locality.

In our approach, the bone influence map $B(i)$ of a vertex i is initially set to all bone indices. We prune bones from $\{B(i)\}$, while ensuring the following two *sparsification properties*: $(\mathbf{P_1})$: from the perspective of bone j, $w_{\cdot j}$ should have its topology preserved (i. e., no extremum is artificially introduced) to make edits consistent; $(\mathbf{P_2})$: from the perspective of vertex i, the set of bones in $B(i)$ should stay fully connected, which is meaningful for the ARAP optimization.

In practice, we process all weights $\{w_{ij}\}$ by increasing value, and remove j from $B(i)$ if: **i)** at least M bones still influence i, **ii)** $\sum_k w_{ik}$ stays larger than a threshold t, **iii)** w_{ij} is smaller than a portion of the largest weight ($w_{ij} < \alpha \max_k(w_{ik})$), and **iv)** no local minimum is introduced (for any adjacent vertex k for which $j \in B(k)$, $w_{ij} \leq w_{kj}$). In all examples, we used $(M,t,\alpha) := (1, 0.95, 0.05)$. Note that the third condition states that the bones' importance is *relative*.

After this pruning, we establish property $(\mathbf{P_2})$. We convert the skeleton to a graph, noted SG, by collapsing all virtual segments. Then, for each vertex i, we find the minimal spanning tree in each component of SG covering $B(i)$.

3.3. Exemplars creation

To define an exemplar (a deformation of the skeleton), we start with a *local* random admissible rotation matrix r_j for each bone j, hence, respecting existing constraints (Fig. 5). To avoid extreme deformations, for which it is unreasonable to optimize for ARAP deformations, we introduce default constraints. Typically, the roll axis (orthogonal to the bone) is picked randomly and the roll value taken in $[-60°, 60°]$. The twist value is picked in $[-10°, 10°]$. For a spine $\mathbf{s}$, we define its base transformation ($r_{\mathbf{s}}$) similarly, while the rotation axis ($a_{\mathbf{s}}$) is chosen randomly with a rotation angle ($\theta_{\mathbf{s}}$) in $[-60°, 60°]$. For a virtual segment j, we set r_j to Id.

Next, the local are converted to *global* transformations by traversing the skeleton and accumulating transformations,

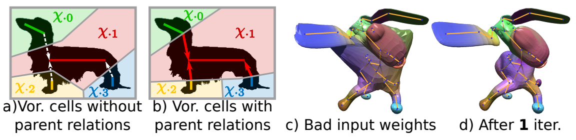

χ_0
χ_1
χ_2
χ_3
a)Vor. cells without parent relations
b) Vor. cells with parent relations
c) Bad input weights
d) After **1** iter.

Figure 4: *Bone Voronoi cells (a) and our modified cells (b). Diffusing cell indicators can result in poor initial weights (c), but our algorithm immediately improves upon those (d).*

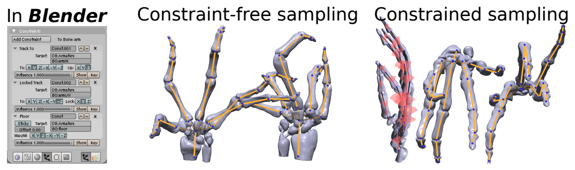


Figure 5: *Professional modelling solutions (e. g., Blender, Maya) offer restrictions on the skeleton. These constraints are respected by our exemplars and, thus, by our weights.*

while ensuring its connectivity. If the skeleton consists of several components, those can be treated in any order. *The set of all bone transformations constitutes the exemplar.*

4. Skinning Weight & Joint Optimization

In order to optimize the weights, we express the ARAP energy in terms of weights, by inserting Eq. 1 into Eq. 5, while enforcing affinity of the weights. It leads to a quadratic energy, whose minimization results in weights that lead to mesh deformations that best approximate an ARAP behavior. We follow traditional ARAP optimization strategies, and alternate between the matrix-optimization (referred to as ***local*** step, since $\{\mathcal{R}(i)\}$ are optimized independently) and the optimization of the weights (referred to as the ***global*** step, since they are jointly optimized). At each iteration of the optimization process, we inject additional *weight constraints* to steer the weight derivation. We also show how to express the ARAP energy in terms of skeleton joint positions, in order to optimize their location similarly.

4.1. $\mathcal{R}(i)$ refinement (*local* step)

To derive $\mathcal{R}(i)$ for each vertex and each exemplar, we need to consider Eq. 5 and optimize for

$$\mathcal{R}(i) = \underset{R \in \mathcal{T}^{\text{adm}}}{\operatorname{argmin}} \mathfrak{E}_i(f, R),$$

where $\mathcal{T}^{\text{adm}}$ is a set of *admissible transformations*. In the traditional ARAP case, $\mathcal{T}^{\text{adm}} := SO(3)$ (rotations in $\mathbb{R}^3$), for which the solution is obtained by projecting $Cov_i := \sum_{k \in V_1(i)} \lambda_{ik} \bar{e}_{ik} \cdot e_{ik}{}^{\mathrm{T}}$ onto SO(3) (see [SA07] for details).

As-rigid-as-desired deformations When the bones' length is preserved, aiming for ARAP is natural. However, spines and STBs contain stretch, and optimizing for rotations in this case makes no sense. Here, we present a generalization of ARAP optimizations for stretchable inputs.

We constrain the singular value σ_k (with associated vector ϑ_k, i. e., $Cov_i = U\Sigma V^{\mathrm{T}}$ and ϑ_k is the k^{th} column of V), based on an analysis of the stretch induced by the bones in the exemplar. Intuitively, if the bones influencing v_i are stretched, v_i's neighborhood should be as well, by an amount contained within the range of the stretch values of the bones. Given a bone $j \in B(i)$ with deformation Jacobian J (see Sec. A for formulas) at point v_i, we compute $\alpha_k^j := \|J \cdot \vartheta_k\|$, the stretch induced by bone j in direction ϑ_k. We then take $\sigma_k := \sum_{j \in B(i)} w_{ij} \alpha_k^j$, thus blending the stretch values induced by the various bones by the weights $\{w_{ij}\}$. Finally, $\mathcal{R}(i) = U \text{diag}(\sigma_1, \sigma_2, \sigma_3) V^{\mathrm{T}}$. If all bones are rigid, this construction becomes the traditional solution (i. e., their Jacobians are rotations, therefore $\alpha_k^j = 1 \; \forall j$, and $\sigma_k = 1$).

Our definition follows a mathematically-sound intuition, but also proved more efficient than optimizing for rotations,

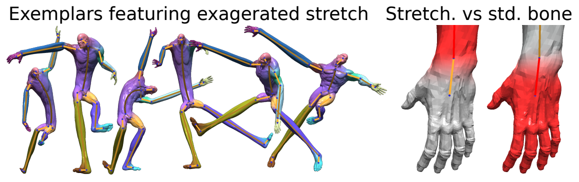


Figure 6: *Stretchable bones. Notice weight map differences for the arm (stretchable* $\sigma \in [0.5, 4.0]$*) and hand (std. bone).*

for similarities, for arbitrary matrices, or simply bounding the singular values by the stretch of the bones. Fig. 6 illustrates an optimization with highly-stretched-bone exemplars.

4.2. Skinning Weight Optimization (*global* step)

We now show how to optimize the weights to decrease the ARAP energy. Let $B(i) =: \{b_i^0, b_i^1, \cdots, b_i^{N_i-1}\}$ be the bones influencing vertex i. Let $\mathbf{w}$ be the vector concatenating the weights $\{w_{ij}\}$, ordered by increasing indices i and j, while ignoring the first weight of each vertex (which is redundant, as $w_{ib_i^0} := 1 - \sum_{j=1}^{N_i-1} w_{ib_i^j}$ due to the affinity condition):

$$\mathbf{w} := (w_{0b_0^1}, \cdots, w_{0b_0^{N_0-1}}, w_{1b_1^1}, \cdots, w_{1b_1^{N_1-1}}, \cdots)^{\mathrm{T}}. \qquad (7)$$

We first consider a single exemplar **ex** (for readability, we avoid indexing the various quantities by **ex**). Noting $\phi_i^j := R_{b_i^j}(v_i) \cdot v_i + T_{b_i^j}(v_i)$ the transformation of vertex i by its j^{th} influencing bone, allows us to write:

$$\bar{v}_i = \sum_{j=0}^{N_i-1} w_{ib_i^j} \phi_i^j = \sum_{j=1}^{N_i-1} w_{ib_i^j} (\phi_i^j - \phi_i^0) + \phi_i^0 =: \Phi_i \cdot \mathbf{w} + \phi_i^0.$$

Note that matrix $\Phi_i \in \mathbb{R}^{3,|\mathbf{w}|}$ is sparse and only contains entries organized in a block for vertex i. We deduce a similar expression for a transformed edge $\bar{e}_{ik} = \Phi_{ik} \cdot \mathbf{w} + \phi_{ik}$, where $\Phi_{ik} := \Phi_k - \Phi_i$ and $\phi_{ik} := (\phi_k^0 - \phi_i^0)$. This expression is inserted into the ARAP energy (Eq. 5) to obtain an expression in terms of weights, leading (for an exemplar **ex**) to:

$$\mathfrak{E}_{\mathbf{ex}}(\mathbf{w}, \mathcal{R}) = \mathbf{w}^{\mathrm{T}} \cdot \mathbf{A}(\mathbf{ex}) \cdot \mathbf{w} - 2\mathbf{B}(\mathbf{ex})^{\mathrm{T}} \cdot \mathbf{w} + \text{const}, \text{ with}$$

$$\mathbf{A}(\mathbf{ex}) := \sum_{i \in \mathcal{V}} \Lambda_i \sum_{k \in V_1(i)} \lambda_{ik} \Phi_{ik}{}^{\mathrm{T}} \cdot \Phi_{ik}$$

$$\mathbf{B}(\mathbf{ex}) := \sum_{i \in \mathcal{V}} \Lambda_i \sum_{k \in V_1(i)} \lambda_{ik} \Phi_{ik}{}^{\mathrm{T}} \cdot (\mathcal{R}(i) \cdot e_{ik} - \phi_{ik})$$

Multiple Exemplars For a reasonable deformation behavior, we typically use a few tens to hundreds of exemplars. Fortunately, the size and sparsity pattern of **A** (and the time for minimizing $\mathfrak{E}(\mathbf{w}, \mathcal{R})$) *are independent of the number of exemplars*. The ARAP energy in weight space is then

$$\mathfrak{E}(\mathbf{w}, \mathcal{R}) := \mathbf{w}^{\mathrm{T}} \cdot \mathbf{A} \cdot \mathbf{w} - 2\mathbf{B}^{\mathrm{T}} \cdot \mathbf{w} + \text{const} \qquad (8)$$

with $\mathbf{A} := \sum_{\mathbf{ex}} \mathbf{A}(\mathbf{ex})$ and $\mathbf{B} := \sum_{\mathbf{ex}} \mathbf{B}(\mathbf{ex})$. The optimal weights under the affinity constraint are thus given by

$$\mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmin}}\, \mathfrak{E}(\mathbf{w}, \mathcal{R}) = \mathbf{A}^{\dagger} \cdot \mathbf{B} \tag{9}$$

4.3. Skinning Weight Constraints

We introduce additional constraints to steer the optimization and improve quality. These *weight constraints* are defined in form of a supplementary error term added to $\mathfrak{E}(\mathbf{w}, \mathcal{R})$:

$$\mathfrak{E}(\mathbf{w}, \mathcal{R}) \leftarrow \mathfrak{E}(\mathbf{w}, \mathcal{R}) \quad + \sum_{(i,j,\hat{w}_{ij}) \in \text{Constraints}} c_i |w_{ij} - \hat{w}_{ij}|^2,$$

where $\hat{w}_{ij}$ is a value, which is softly enforced on weight w_{ij} and c_i a scaling factor to control its impact. In all our examples, we used $c_i := 100.\text{BBD}^2.\Lambda(i).\#\{\mathbf{ex}\}$, where BBD is the bounding box diagonal (to make value ranges compatible), the vertex weight $\Lambda(i)$ (to account for vertex density), and the number of exemplars $\#\{\mathbf{ex}\}$ (to account for the pose sampling). Adding/removing these constraints after the factorization of $\mathbf{A}$ is efficiently done using low-rank Cholesky-factorization updates [DH09]. In our experiments, the observed performance overhead was negligeable.

Attach Constraints are set prior to the factorization and used to maintain a strong attachment for vertices with a maximal initial weight estimate. Hereby, we ensure that especially extremities of a skeleton are not ignored. In practice, we set $\hat{w}_{ij} = 1$ if **i)** w_{ij} is a strict local maximum at i (because a single attachment is sufficient), **ii)** $w_{ij} > 0.5$ (because only strong influences should be considered), and **iii)** j is visible from i (similar to [BP07]). Note that, when optimizing for a volume mesh with a skeleton located inside the shape, we only constrain the vertices *on the bones themselves* (similarly to [JBPS11]), which is compatible with the constraints previously listed for the surface case.

Neumann Constraints are set prior to the factorization, and ensure that weights fade out smoothly where their influence disappears. We set $\hat{w}_{ij} = 0$ if a neighbor k of i is not influenced by bone j (thus enforcing a null gradient of $w_{.j}$ on edge (i,k)).

Positivity Constraints are used to enforce weight positivity. In practice, we add after each iteration a constraint with $\hat{w}_{ij} = 0$, if a weight w_{ij} is negative (we test if $w_{ij} < -10^{-3}$) *and is a local minimum*. Hereby, we reduce negative weights drastically, while *introducing constraints sparsely*.

Mixed-Weight Constraints are optional and, if wanted by the user, used to bound areas of the mesh to following other strategies than ARAP by setting $\hat{w}_{ij}$ accordingly.

4.4. Joint Optimization

Artists tend to produce skeletons reflecting the desired degrees of freedom to best serve their animation needs, but fine-tuning the skeleton location is challenging. Consider a skeletal chain inside a cylinder (e. g., the arm of a character), here, the bones should lie close to the medial axis *to minimize stretch*. Similarly, joints would be best placed at the centers of the intended articulations. This section presents an automatic optimization of the skeletal joint positioning, which in turn leads to better skinning results as well.

It is important to note that exemplars are defined for the skeleton. Hence, when changing skeletal joints, an adaptation of the exemplars is needed. In Sec. 3.3, we constructed exemplars while ensuring that the skeleton stays connected at the joints. In this section, we will see that the previous sequential construction can be expressed in terms of a linear dependency, linking the joint positions and the bone translations (Sec. 4.4.1). In consequence, we can plug the resulting joint positions in the ARAP energy. The obtained expression allows us to optimize for the joint displacements $\{\delta J_k\}$ minimizing the ARAP energy (Sec. 4.4.2).

4.4.1. Algebraic Exemplar Construction

First, we express bone translations in terms of joint displacements, while maintaining the other parameters (e. g., rotations) of the exemplar. We first investigate a pair of rigid bones (i, j) connected by a joint k displaced by δJ_k, before addressing different components of the skeleton. Applying either one of their transformations on the joint should yield the same result to preserve connectivity. It reads:

$$R_i^{\mathbf{ex}} \cdot (J_k + \delta J_k) + (T_i^{\mathbf{ex}} + \delta T_i^{\mathbf{ex}}) = R_j^{\mathbf{ex}} \cdot (J_k + \delta J_k) + (T_j^{\mathbf{ex}} + \delta T_j^{\mathbf{ex}})$$

Observing that the skeleton transformations $\{R_i^{\mathbf{ex}}, T_i^{\mathbf{ex}}\}$ and $\{R_j^{\mathbf{ex}}, T_j^{\mathbf{ex}}\}$ in **ex** were constructed similarly, ($R_i^{\mathbf{ex}} \cdot J_k + T_i^{\mathbf{ex}} = R_j^{\mathbf{ex}} \cdot J_k + T_j^{\mathbf{ex}}$) (Sec. 3.3), we obtain the following *joint constraint*, which we note $JointConst(i, j, k)^{\mathbf{ex}}$:

$$\delta T_j^{\mathbf{ex}} - \delta T_i^{\mathbf{ex}} = (R_i^{\mathbf{ex}} - R_j^{\mathbf{ex}}) \cdot \delta J_k. \tag{10}$$

If spines are used, this construction is slightly more involved. For a spine $\mathbf{s}$ with extremities (e_0, e_1) and parameters $(R_{\mathbf{s}}, t_{\mathbf{s}}, \sigma_{\mathbf{s}}, \theta_{\mathbf{s}}, a_{\mathbf{s}})$, the deformation of e_0 will be $\bar{e}_0 = R_{\mathbf{s}} \cdot e_0 + t_{\mathbf{s}}$ (like previously) but the deformation of e_1 will be $\bar{e}_1 = R_{\mathbf{s}} Rot(a_{\mathbf{s}}, \theta_{\mathbf{s}}) \cdot e_1 + R_{\mathbf{s}} A(1) \cdot e_0 + R_{\mathbf{s}} B(1) \cdot e_1 + t_{\mathbf{s}}$ (by rewriting Eq. 4 as $t_{\text{loc}}(u) =: A(u) \cdot e_0 + B(u) \cdot e_1$). In short, the fact that $\bar{e}_1$ depends both on e_0 and on e_1 changes the right side of $JointConst(i, j, k)^{\mathbf{ex}}$ in case bone i (or j) is a spine and joint k is its positive extremity (i. e., e_1). $JointConst(i, j, k)^{\mathbf{ex}}$ then contains not only products of matrices by δJ_k, but also products of matrices by $\delta J_{k'}$, joint k' being the negative extremity of the spine (i. e., e_0).

Joint constraints are insufficient to express $\{\delta T_j^{\mathbf{ex}}\}$ in terms of $\{\delta J_k\}$: applying a constant displacement to a connected component would still preserve all joint constraints. We thus introduce auxiliary variables $\delta C_c^{\mathbf{ex}}$, one for each connected component c, which reflect an *average component displacement* . The resulting *component constraints*, noted

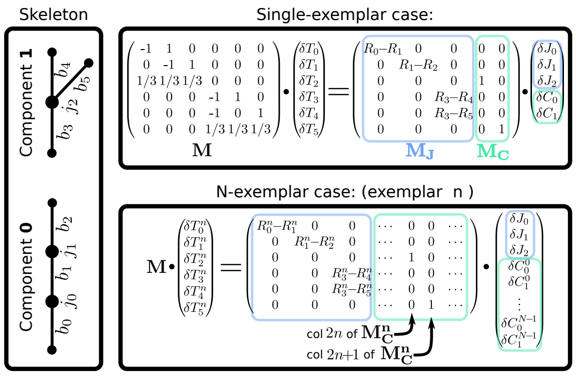


Figure 7: *Algebraic construction of the exemplars. All elements are 3×3-matrices (i. e., a scalar x represents $x.I_3$.)*

$CompConst(c)^{\mathbf{ex}}$, are given by:

$$\sum_{i \in Component(c)} \delta T_i^{\mathbf{ex}} / \#Component(c) = \delta C_c^{\mathbf{ex}}. \qquad (11)$$

Using joint and component constraints, we can define a linear relation to express the change of the skeletal-segment translations $\{\delta T_j^{\mathbf{ex}}\}$ for a set of joint displacements $\{\delta J_k\}$ and average component displacements $\{\delta C_c^{\mathbf{ex}}\}$. To setup the equation system, we process each connected component c of the skeleton successively. We then process each joint k in c successively as well. Let $\{s_0,\dots,s_n\}$ be k's adjacent skeletal segments, we then add a constraint $JointConst(s_0,s_i,k)^{\mathbf{ex}}$ for all $i>0$ (Eq. 10). Once all joint constraints of c are collected, we add the component constraint $CompConst(c)^{\mathbf{ex}}$ (Eq. 11) and proceed with the next connected component. Finally, the resulting set of equations can be rewritten as:

$$\delta \mathbf{T}^{\mathbf{ex}} =: \mathbf{M}^{-1} \cdot (\mathbf{M}_{\mathbf{J}}^{\mathbf{ex}}, \mathbf{M}_{\mathbf{C}}^{\mathbf{ex}}) \cdot \begin{bmatrix} \delta \mathbf{J} \\ \delta \mathbf{C} \end{bmatrix} \qquad (12)$$

where $\delta\mathbf{J} := (\delta J_0 \cdots \delta J_{n-1})^{\mathrm{T}}$ and $\delta\mathbf{C} := (\delta C_0^{\mathbf{ex}} \cdots \delta C_{m-1}^{\mathbf{ex}})^{\mathrm{T}}$ are the concatenations of the displacement variables for the n joints and m connected components. Fig. 7 illustrates the construction of $(\mathbf{M}, \mathbf{M}_{\mathbf{J}}^{\mathbf{ex}}, \mathbf{M}_{\mathbf{C}}^{\mathbf{ex}})$ on a toy example.

Multiple Exemplars For each exemplar, one auxiliary variable is added for each connected component, while the joint displacements are shared and globally defined. In consequence, each exemplar enlarges $\mathbf{M}_{\mathbf{C}}^{\mathbf{ex}}$. Fig. 7 (bottom) illustrates the extended form of $\mathbf{M}_{\mathbf{C}}^{\mathbf{ex}}$ and the corresponding displacement vector, which is similarly extended to hold the average component displacements over all exemplars.

4.4.2. Skeletal-Joint optimization (*global* step)

We now express the ARAP energy term as a function of the joint displacements. We first observe that $\delta\mathbf{T}^{\mathbf{ex}}$ leads to an offset $\delta\bar{v}_i^{\mathbf{ex}}$ for the LBS of a vertex i (Eq. 1):

$$\delta \bar{v}_i^{\mathbf{ex}} = \sum_{j \in B(i)} w_{ij} \delta T_j^{\mathbf{ex}}(v_i).$$

For rigid bones, the change of the translation applied to vertex i by bone j (i. e., $\delta T_j^{\mathbf{ex}}(v_i)$) is simply $\delta T_j^{\mathbf{ex}}$, and one can use Eq. 12 to express $\delta T_j^{\mathbf{ex}}(v_i)$ in terms of $(\delta\mathbf{J},\delta\mathbf{C})^{\mathrm{T}}$.

For a spine $\mathbf{s}$ with extremities (e_0,e_1) and parameters $(R_{\mathbf{s}}, t_{\mathbf{s}}, \sigma_{\mathbf{s}}, \theta_{\mathbf{s}}, a_{\mathbf{s}})$, more care is needed. Looking at the induced transformation (Eq. 2), the change of the translation applied to vertex i by $\mathbf{s}$ ($\delta T_{\mathbf{s}}^{\mathbf{ex}}(v_i)$) is **not** simply $\delta t_{\mathbf{s}}$, because $t_{\mathrm{loc}}(u)$ depends on e_0 and e_1 as well. As before, we rewrite $t_{\mathrm{loc}}(u) =: A(u)\cdot e_0 + B(u)\cdot e_1$, leading to $\delta T_{\mathbf{s}}^{\mathbf{ex}}(v_i) = R_{\mathbf{s}} A(u(v_i))\cdot \delta e_0 + R_{\mathbf{s}} B(u(v_i))\cdot \delta e_1 + \delta t_{\mathbf{s}}$. Using Eq. 12 to express $\delta t_{\mathbf{s}}$ in terms of $(\delta\mathbf{J},\delta\mathbf{C})^{\mathrm{T}}$, $\delta T_j^{\mathbf{ex}}(v_i)$ can be expressed in terms of $(\delta\mathbf{J},\delta\mathbf{C})^{\mathrm{T}}$ for spines and STBs as well.

Consequently, we can also compute the effect on an edge $\bar{e}_{ik}^{\mathbf{ex}}$ and obtain again an offset $\delta\bar{e}_{ik}^{\mathbf{ex}}$, which is given by:

$$\delta \bar{e}_{ik}^{\mathbf{ex}} = \sum_{j \in B(k)} w_{kj} \delta T_j^{\mathbf{ex}}(v_k) - \sum_{j \in B(i)} w_{ij} \delta T_j^{\mathbf{ex}}(v_i) =: \delta\Omega_{(i,k)}^{\mathbf{ex}} \cdot \begin{bmatrix} \delta \mathbf{J} \\ \delta \mathbf{C} \end{bmatrix},$$

where $\delta\Omega_{(i,k)}^{\mathbf{ex}}$ is a horizontal vector of block matrices. Combining this result with Eq. 5 yields:

$$\mathfrak{E}(\begin{bmatrix} \delta \mathbf{J} \\ \delta \mathbf{C} \end{bmatrix}, \mathcal{R}) = \sum_{\mathbf{ex}} \sum_{i \in \mathcal{V}} \Lambda_i \sum_{k \in V_1(i)} \lambda_{ik} || \delta \bar{e}_{ik}^{\mathbf{ex}} - (\mathcal{R}(i)^{\mathbf{ex}} \cdot e_{ik} - \bar{e}_{ik}^{\mathbf{ex}}) ||^2$$

This last equation expresses the ARAP energy in terms of $(\delta\mathbf{J},\delta\mathbf{C})^{\mathrm{T}}$. Because it is not always positive definite, we regularize it and minimize: $\mathfrak{E}((\delta\mathbf{J},\delta\mathbf{C})^{\mathrm{T}}, \mathcal{R}) + \|\mathcal{E}_{\mathcal{J}} \cdot (\delta\mathbf{J},\delta\mathbf{C})^{\mathrm{T}}\|^2$, where $\mathcal{E}_{\mathcal{J}}$ is a block-diagonal matrix penalizing large offsets. The quadric to minimize is finally:

$$\mathfrak{E}_{\mathfrak{J}} := \begin{bmatrix} \delta \mathbf{J} \\ \delta \mathbf{C} \end{bmatrix}^{\mathrm{T}} \cdot \mathbf{A}_{\mathcal{J}} \cdot \begin{bmatrix} \delta \mathbf{J} \\ \delta \mathbf{C} \end{bmatrix} - 2\mathbf{B}_{\mathcal{J}}{}^{\mathrm{T}} \cdot \begin{bmatrix} \delta \mathbf{J} \\ \delta \mathbf{C} \end{bmatrix} + \text{const, with}$$

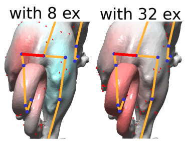
$$\mathbf{A}_{\mathcal{J}} := \sum_{\mathbf{ex}} \sum_{i \in \mathcal{V}} \Lambda_i \sum_{k \in V_1(i)} \lambda_{ik} \delta\Omega_{(i,k)}^{\mathbf{ex}}{}^{\mathrm{T}} \cdot \delta\Omega_{(i,k)}^{\mathbf{ex}} \; + \; \mathcal{E}_{\mathcal{J}}{}^{\mathrm{T}} \cdot \mathcal{E}_{\mathcal{J}}$$

$$\mathbf{B}_{\mathcal{J}} := \sum_{\mathbf{ex}} \sum_{i \in \mathcal{V}} \Lambda_i \sum_{k \in V_1(i)} \lambda_{ik} \delta\Omega_{(i,k)}^{\mathbf{ex}}{}^{\mathrm{T}} \cdot (\mathcal{R}(i)^{\mathbf{ex}} \cdot e_{ik} - \bar{e}_{ik}^{\mathbf{ex}})$$

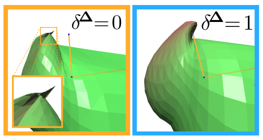
In our implementation, we penalize *tangential joint displacement* more severely than others (noting $\{b_k \in \mathbb{R}^3\}$ the 3D bones adjacent to the joint, we set the corresponding block matrix to $\varepsilon_1 \mathrm{Id} + \varepsilon_2 \sum b_k \cdot b_k{}^{\mathrm{T}} / |b_k|$). ε_1 can be defined per joint as a **joint-material** scalar function $\varepsilon_1 : \mathcal{J} \mapsto \mathbb{R}^+$ (e. g., to avoid specific joints displacements). After each iteration, each exemplar **ex** is updated using Eq. 12, as well as $\{\mathcal{R}(i)^{\mathbf{ex}}\}$. $\mathbf{A}_{\mathcal{J}}$ is factorized once and several iterations can be performed without updating it, until weights are changed.

5. Algorithmic Details

Vertex and Edge ARAP Weights While Λ_i and λ_{ik} are often defined via cotangent weights [PP93], it can lead to artifacts, as negative values can occur for obtuse triangles. We use barycentric cell-area weights instead, which are positive.



Positivity While we *penalize* negativity (Sec.4.3), we do not *ensure* positivity. However, negative weights are mostly naturally avoided. The inset illustrates this point; *without positivity constraints*, negative weights (in blue) occur when using 8 exemplars but are drastically diminished when using 32.



Local maxima Some previous works (e.g., [JWS12]) detect and suppress non-global extrema. In our solution, small local extrema can be observed, but a one-step Laplacian filtering (including normalization) after each optimization iteration is sufficient to avoid most. However, attach constraints can lead to artifacts (inset, orange) in challenging configurations (here, a large bone animates ***Dog***'s nose). A fully automatic solution is to add a harmonic (null Laplacian) term (inset, blue) to $\mathfrak{E}$:

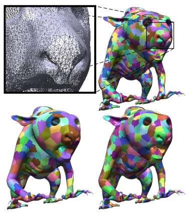
$$\mathfrak{E}(\mathbf{w},\mathcal{R}) \leftarrow \mathfrak{E}(\mathbf{w},\mathcal{R}) + \delta^{\Delta}\sum_{i,j} c_i|\Delta_i(w_{\cdot j})|^2 , \qquad (13)$$

δ^{Δ} balancing both terms. Taking the traditional cotangent formula discretizing the Laplacian operator, and expressing $w_{\cdot j}$ in terms of $\mathbf{w}$ (Eq. 7), we formulate $\Delta_i(w_{\cdot j})$ as $\Delta_i(w_{\cdot j}) =: A^{\Delta}_{ij}\cdot\mathbf{w} + b^{\Delta}_{ij}$. The updates to the quadratic and linear parts of $\mathfrak{E}$, corresponding to Eq. 13, are:

$$\mathbf{A} \leftarrow \mathbf{A} + \delta^{\Delta}\sum_{i,j} c_i {A^{\Delta}_{ij}}^{\mathrm{T}}\cdot A^{\Delta}_{ij} \ , \ \mathbf{B} \leftarrow \mathbf{B} + \delta^{\Delta}\sum_{i,j} c_i b^{\Delta}_{ij} {A^{\Delta}_{ij}}^{\mathrm{T}} .$$

Note, that this solution comes at a price: $\mathbf{A}$ is less sparse ($\approx 5\times$ in our experiments) and more costly to factorize ($\approx 10\times$), and each optimization of $\mathbf{w}^*$ (Eq. 9) is ($\approx 10\times$) slower than before. Unless otherwise noted, we did not use this step.

Vertex-Clustering Acceleration For each exemplar and iteration, updating $\{\mathcal{R}(i)\}$ requires computing numerous costly $3\times 3-$matrix SVDs. As noted in [JBK*12], the transformations (and thus $\{\mathcal{R}(i)\}$) should ultimately be *smooth*. Consequently, the step can be accelerated by clustering.



To avoid clustering effects in the output, we compute several clusterings and associate each exemplar with a random one (we use 10 clusterings on the positions, with 1000 to 5000 clusters). To obtain uniform clusterings despite anisotropy in the input mesh, we propose a variant of k-means++ [AV07], and select vertex v_i as n^{th} seed s_n with relative probability $\Lambda_i d^2(v_i,\{s_j\}_{j<n})$. The inset shows three uniform clusterings of various sizes.

For each exemplar, the covariance of a cluster cl is computed as $\sum_{i\in cl}\Lambda_i\sum_{k\in V_1(i)}\lambda_{ik}\bar{e}_{ik}\cdot {e_{ik}}^{\mathrm{T}} =: U\cdot\Sigma\cdot V^{\mathrm{T}}$ (SVD), and the stretch σ_k in direction ϑ_k is set to

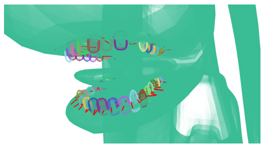
$$\sigma_k := \sum_{i\in cl}\sum_{j\in B(i)}\Lambda_i w_{ij}\|J_{ij}\cdot\vartheta_k\| / \sum_{i\in cl}\sum_{j\in B(i)}\Lambda_i w_{ij},$$

(J_{ij} being the Jacobian of bone j at point v_i in the exemplar). Finally, we set $\mathcal{R}(cl) = U\cdot\mathrm{diag}(\sigma_1,\sigma_2,\sigma_3)\cdot V^{\mathrm{T}}$, and each vertex i in the cluster cl is associated with this matrix: $\mathcal{R}(i) = \mathcal{R}(cl), \forall i\in cl$.

6. Skinning Tools

User interaction should be an integral part of skinning to allow for subjective decisions. We introduce brushing tools to easily indicate additional constraints interactively if wanted.

The **bone-influence brush** increases/decreases a bone j's influence *prior to the factorization* by adding/removing j to/from $\{B(i)\}$. Hereby, its locality is easily adjusted.



The **wire-edge brush** links disconnected vertices *prior to the initialization* and can be used to link disconnected components, which are common in artist-designed 3D models. We add wire edges $i\rightarrow j$ connecting vertices i and j with the following ARAP weight: $\lambda_{ij}{:=}\sum_{k\in V_1(i)}\lambda_{ik}*\min\left(1/2,|V_1'(i)|/|V_1(i)|\right)/|V_1'(i)|$, where $V_1(i)$ is the set of original vertices adjacent to i and $V_1'(i)$ is the set of vertices newly-connected through wire-edges. The idea is to establish a soft connection between i and j, i. e., a portion of the weights adjacent to i is attributed to the new edges. To prioritize the original connectivity, their sum is bound to half the sum of the original weights.

Note that HBs [BP07] do not support wire-edges. We construct the weight vector $w_{\cdot j}^{\mathrm{HBM}}$ for bone j by solving for:

$$\left[\begin{array}{c} -\Delta + H \\ \hline D_{\mathrm{WE}} \end{array}\right]\cdot w_{\cdot j}^{\mathrm{HBM}} = \left[\begin{array}{c} H\cdot\chi_{\cdot j} \\ \hline 0 \end{array}\right],$$

D_{WE} being a matrix for which each line corresponds to a wire-edge linking two vertices k and l, with d_{kl} in k^{th} column and $-d_{kl}$ in l^{th} column, thus adding $d_{kl}^2(w_{kj}-w_{lj})^2$ in the energy (softly enforcing the weights of the two vertices to be similar). We used $d_{kl} = 10^5.\exp(-||v_k - v_l||^2/\mathrm{BBD}^2)$ for all results. Since these weights can be negative, we set $w_{ij} = \max(w_{ij}^{\mathrm{HB}}, w_{ij}^{\mathrm{HBM}})$ and finally normalize them as a post-process ($w_{ij}\leftarrow w_{ij}/\sum_k w_{ik}$) to initialize our weights. This strategy proved more robust than trying to introduce the wire-edges in the discretization of the Laplacian.

The **weight-constraint brush** can be used *during the optimization* to enforce weight values (Sec. 4.3, **mixed-weight constraints**). Note that brushes enforcing weight derivatives locally can also be created by constraining a linear combination of weights instead of weight values (e. g., a "null Laplacian" brush, see Eq. 13). We demonstrate such user interaction on ***Dog*** in the accompagnying video.

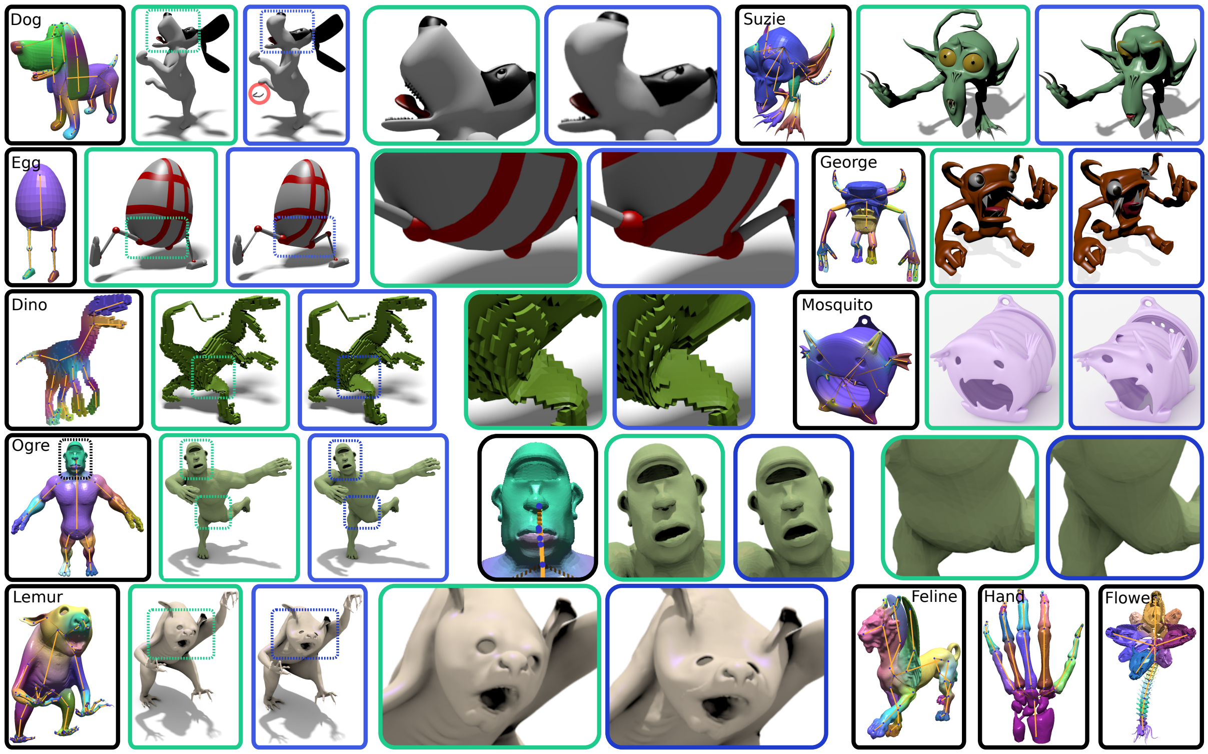


Figure 8: ***Results*** *produced in Blender. The use of our weights (green) is compared to HBs (blue) on the same input (rest pose mesh/skeleton, deformed skeleton).* ***Dog***, ***Suzie***, ***Egg***, ***George***, ***Dino***, ***Mosquito*** *and* ***Lemur*** *are courtesy of* `blendswap.com`.

7. Results and Discussion

We implemented our work in C++, using Cholmod to efficiently solve linear systems. All results are produced with a laptop system (Intel Core2 Duo, 2.5 GHz, 4GB of memory).

Models To illustrate practicality, we use models produced by real artists and provided via `blendswap.com`. They exhibit disconnected components, tiny features, non-manifold structures, boundaries and elongated triangles, self-intersections, and complex topology. The used skeletons are complex; partially disconnected, with various bones types, potential deformation constraints, and scales (i. e., from main limbs to fingers for ***Lemur*** and ***George***).

The main results of our approach are illustrated in Fig. 8 and compared to HBs [BP07], the only other technique among the ones we tried, which could give results on all examples. Our weights were computed automatically, with two exceptions: For ***Dog***, we illustrate that artistic choices remain possible, by applying the **weight-constraint brush** on the tip of the nose to enforce a large influence of 1, resulting in an apparent local stretch. For ***Suzie***, some *attach constraints* were manually removed during the session because the teeth of the upper jaw were attached to the bone of the lower jaw. These interactions took about 15 secs each.

MODEL (#V)	#B	INIT.	#U/V	IT.	INT.
Dog (11K)	29	7.6	4.1	0.7	≈ 15
Suzie (32K)	32	5.6	1.6	0.8	≈ 15
Egg (6K)	7	0.36	0.8	0.14	no
George (14K)	36	4.1	2.7	0.52	no
Dino (7K)	25	1.2	1.6	0.23	no
Mosquito (27K)	15	51.7	5.8	2	no
Mosquito (27K)	15	6.5	0.9	0.5	≈ 60
Ogre (26K)	16	4.4	1.7	0.78	no
Lemur (43K)	53	19.6	2.43	1.55	no

Table 1: *Weights computation timings (ref. Fig. 8), using* 80 *exemplars.* #V*: vertices.* #B*: bones.* INIT.*: Initialization in s.* #U/V*: (average) nb of unknowns per vertex.* IT.*: (average) iteration in s.* INT.*: Performed interaction in s.*

Timings Our solution is fast enough for interactive work sessions (see video), even though we optimize jointly all bone weight maps and use 80 exemplars in our setup. We report timings in Tab. 1. For all models, except ***Mosquito***, (with $\approx 5K - 40K$ vertices), the preprocessing takes $\approx 0.4 - 20s$, while an iteration takes $\approx 0.2 - 1.6s$. The joint-optimization cost depends on the skeleton topology and took $\approx 5 - 10s$ to factorize the linear system and $1 - 4s$ per iteration. Presented results use 10 to 40 iterations. These timings

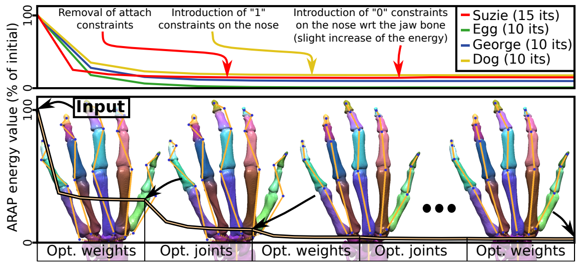


Figure 9: *Profile of the ARAP energy. We illustrate the impact of the brushes on **Dog** and **Suzie**. The bottom row shows the convergence of the ARAP energy when optimizing weights and joint positions on **Hand** (25 iterations in total).*

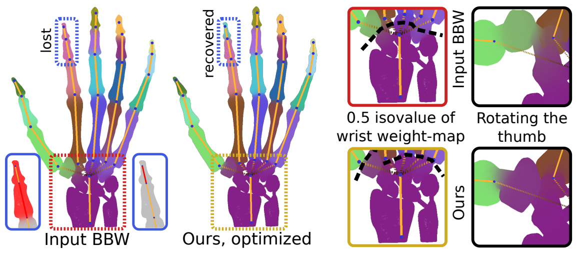


Figure 10: *Comparison with Bounded Biharmonic weights.*

could only be obtained thanks to our sparsification scheme, which decreases the number of variables drastically. For example, on the ***Lemur***, with 43K vertices, 53 bones and 80 exemplars, our sparsification led to 2.43 unknowns per vertex on average, and as a result each iteration took 1.55s. On the contrary, not using the sparsification led to failure after 45 mins (26 mins to fill the linear system and 19 to factorize it) with a CHOLMOD error "problem too large" (and it would have been even worse, had we used a volumetric structure, as the number of variables would have been increased).

The peculiar geometry of ***Mosquito*** led to poor input weights, poorly sparsified influence maps and large computation times (51.7s for the initialization, 2s per iteration). We report in Tab.1 an additional session where we further removed bones from far-away regions using the **bone-influence brush**, which resulted in better timings and visually-undistinguishable results. In our opinion, it reflects that such easy interactions should be offered to an artist.

Fig. 9 illustrates the convergence of the ARAP energy on several examples. As written earlier, we removed a few *attach constraints* on ***Suzie*** (first interaction on the red plot). For the session in Fig. 9, we also chose to enforce a zero value on the nose w.r.t. the jaw bone. While this results in a slight increase of the energy (see red plot), artistic choices should ultimately prevail and be made possible.

Volumetric structures have been used to obtain *shape-aware* weights [JBPS11], and to replace favorably poorly-shaped input surface meshes [DdL13]. Computing volumetric structures allows us to consider objects as volumes instead of simple shells [CPSS10], and results obtained using a volumetric weight optimization are typically better than the ones obtained by performing a surfacic optimization directly on the input meshes. Our solution is compatible with volumetric structures, but we purposely avoided *relying on* them, as we could not obtain volumetrizations of most of the input surface meshes shown in Fig. 8 with existing tools and techniques. Additionally, it allows us to present our results based on the setup usually used by modelers and CG artists.

One fundamental issue when volumetrizing surfaces is, that only surfaces that are 2−boundaries of Euclidean 3D volumes can be used as input. Consider the ***Dog***: The teeth were modelled independently of the mouth. Because some intersect with the mouth and the tongue, any watertight constrained Delaunay triangulation (CDT) sews the mouth shut and renders the animation impossible. Please note, that this problem does not relate to volumetrizing poorly-shaped surfaces, and that solutions like [JKSH13] still rely on a CDT. This problem occurs as well for [DdL13]: while the voxelization *allows them to bind together disconnected colocated elements*, it *prevents them from separating colocated elements, which are intended to be separated*. Sacht et al. [SJP*13] tackled this problem, by unwrapping the input surface (using iterative smoothing) before the CDT. However, this approach is not sure to succeed and disconnected components are smoothed independently: intersections between separate parts are not always resolved and the parts might not be correctly placed w.r.t. each other.

Another fundamental issue is that artists may choose to place the kinetics skeleton outside the mesh. In Fig. 15, a volumetrization of the surface would contain merely half the bones of the kinetics skeleton (see also ***Mosquito*** in Fig. 8). Note that volumetrizing the entire space would falsely connect the legs to the spine bone due to their proximity.

Comparison with BBWs For most meshes, we could obtain a valid tetrahedrization only after manually tuning the surface and/or the tetrahedrization. However, these changes required modifying the input mesh structure (causing issues for, e. g., textures) or the skeleton's geometry (which should only be driven by the artist's intentions, not avoidance of issues). For ***Feline***, we could only obtain BBWs after refining the tetrahedra in the wings, resulting in 140 mins of computation (using the original implementation [JP*14]), while we delivered weights of similar quality in 1 min (see Fig. 8).

Our method works on volumes as shown in Fig. 10, where BBWs were used for the initialization. BBWs robustness issues are illustrated by the lost influence of the index fingertip. Tiny features led to small tetrahedra causing numerical issues, even after ensuring the skeleton's local containment within the mesh. These difficulties are more likely to arise for models with thin structures, requiring a careful design of

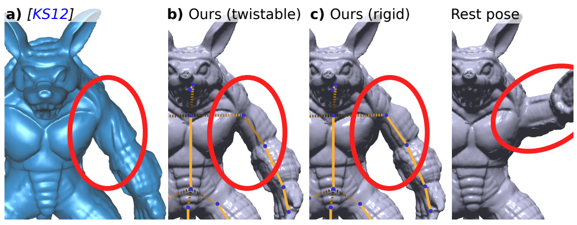


Figure 11: *a): deformer of [KS12] with BBWs (image courtesy of [KS12]). b,c): various bone types with our weights.*

the used tetrahedral mesh. Using weight constraints (Sec. 6), the missing influence was recovered after a few iterations of our algorithm. Note that our solution applied to the surface led to high-quality skinning in less than 1 min (see Fig. 8).

Similarly, the extreme aspect ratios of the triangles in ***Egg*** prevented the solver from succeeding directly. We were able to derive BBWs for ***Egg*** after manual tuning of the mesh (fixing self-intersections) and the tetrahedrization, but the BBWs performed similarly to [BP07] (where only *smoothness* not *deformation quality* is considered). This is also the conclusion reached by [KS12] and [JDKL14].

Comparison with Kavan & Sorkine As indicated, this method is related to ours, but there are several important differences, the main one being that we optimize different variables: They optimize for their joint-based deformers, and then blend the transformations induced by the various deformers using input BBWs, whereas we take as input common bone-based deformers and introduce a blending scheme that favors ARAP deformations. Simple on-the-fly user interactions, such as the ones we demonstrate and the integration of existing skeleton animations in the optimization process, are not possible for their approach. They rely on volumetric structures and BBWs, which can be difficult to obtain from surface meshes. Their deformer is optimized to avoid the candy-wrapper artifact under twist and the bulging effect under rotation. We support spines and STBs that avoid these artifacts as well, and we make it possible to define each bone type individually, including highly-stretched bones, to offer increased artistic freedom. Fig. 11 depicts visual differences between our weights and deformers (two kinds: twistable and rigid) and theirs on a simple example. While differences exist, both results are arguably of similar quality.

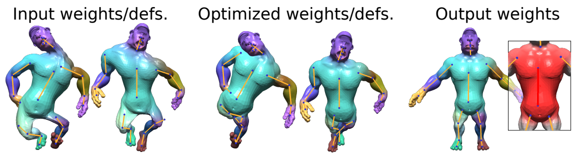


Figure 12: *Disconnected skeleton. Notice the torso's map.*

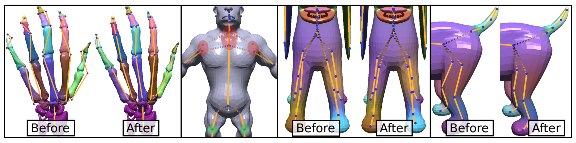


Figure 13: *Joint optimization on several examples.*

Disconnected skeletons We support disconnected skeletons (Fig. 12). Here, the skeleton constraints are not enough to create plausible exemplars, which are key to steering our optimization. By alternating between weight and connected-component optimizations (see Sec. 4.4), we recover the lost constraints and map the initially-generated exemplars to a set of useful ones, which drive our optimization process. To keep the example simple, joints were fixed.

Joint Optimization We assist users in placing joints in locations, which naturally resemble articulations, even when mesh and skeleton topologies do not match (Fig. 13, left, ***Hand***). Our optimization does not rely on a special type of weight: it can be used in conjunction with other definitions. Differences in displacements between shoulders, neck, knees and elbows on ***Ogre*** (Fig. 13, middle) illustrate the difficulty of placing the joints in non-trivial areas such as shoulders. Note that even though symmetry is not enforced in our system, the optimized skeleton tends to become symmetric naturally (see legs of ***Dog*** as well as fingers of the ***Hand*** in Fig. 13), and place the joints near plausible articulations in order to minimize stretch over the deformation exemplars.

Spines Our solution is fully compatible with parameterized bones (Sec.1.2) and can produce weights, which typically cannot be obtained without knowledge of user intent. To model a spine, an artist would typically use many bones and paint weights exhibiting a banding pattern. While small bones might mimic vertebras, large rotations result in a high stretch and implausible transformations – regardless of the weight definition. As a result, automatically-produced weights are sub-optimal, and the influence of many bones is lost (Fig. 14, left). Using a spine deformer (Fig. 14 right), our approach delivers high-quality weights. The spine can be converted into many bones in a postprocess (Sec. 1.2) and

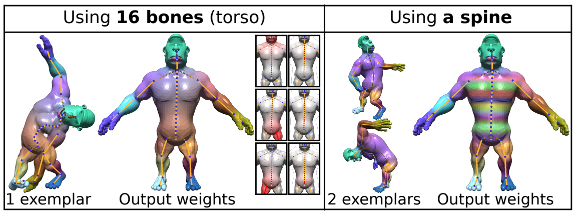


Figure 14: ***Left:*** *16 small bones* ***Right:*** *a spine used for the torso and converted back to 16 bones.*

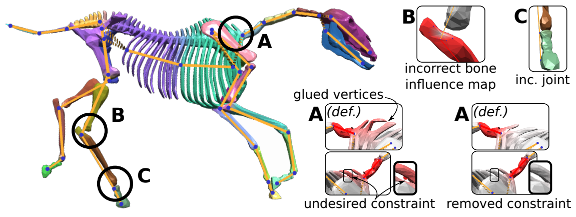


Figure 15: *Failure cases. Closeup from adapted view.*

leads to the expected banding pattern automatically. Note, that the torso's upper-bone weights blend naturally with the ones of arms and neck. To the best of our knowledge, no other automatic technique produces similar results, in particular because no optimization procedure has been proposed for the specific case of parameterized bones such as spines.

Limitations Our approach has limitations. We believe the major being the definition of the initial *attach constraints*. In Fig. 15, the geometric complexity led to several local maxima in the input maps, and thus to undesired attach constraints. Fig. 15(A) shows the use of an undesired constraint. Since nearby vertices are glued together, a strong distortion occurs when manipulating the skeleton. Fortunately, with our system's feedback, a user can easily and swiftly detect and correct such issues. Removing the constraint improves the weights after 2 or 3 iterations. In principle, it should be feasible to detect and edit such constraints after each iteration, to be even more independent of the initial weights.

Similarly, difficult configurations can make the initialization of the bone influence maps $\{B(i)\}$ suboptimal. Fig. 15(B) shows an example; as the bone crosses the input mesh at a vertex v_i, its weight is initialized to one, while the bone, which should have been attached, was removed from $B(i)$ (close-up). Nonetheless, these cases are easy to detect and rare: they arise only if no existing solution could correctly initialize the weights. This operation is also simply inevitable, if one desires to obtain interactive rates. These, in turn, make it possible to control locality and sparsity, and to edit weights during optimization. This last feature very important and critical for industrial productions.

Our joint-placement strategy has also limitations. While deformation quality should drive the optimization, it can prove insufficient. Fig. 15(C) shows a failure case. Our method will not move the joint to the middle of the leg because the model is disconnected and no stretch can be observed when rotating the joint.

Finally, as mentioned, using many small bones in the middle of a thick surface (Fig. 14, left), instead of using a spine deformer, leads to suboptimal weights.

8. Conclusion

We presented a practical, robust, fast, high-quality skinning solution, which is general and handles many complicated cases that are difficult (or impossible) to treat with existing solutions. We showed that sampling the space of admissible deformations of a skeleton results in meaningful weights, i. e., which minimize an ARAP energy. Furthermore, motion constraints (often available and easy to define), or even existing skeleton animations, are naturally integrated. We extended support to stretchable inputs such as stretchable-and-twistable bones, or spines. We have also shown how to optimize joint positions. The results illustrate that our approach achieves good solutions automatically for various cases and introduced skinning tools to resolve ambiguous situations. Our fast solution makes it possible to apply such user input during the optimization with interactive feedback.

In the future, we plan to investigate a continuous formulation to avoid exemplar sampling. To increase performance further, a GPU implementation is promising, as well as a solver, optimizing weights and joint positions simultaneously without re-factorizations.

Acknowledgments We thank the authors of [JBPS11] for sharing their code [JP*14] and Daniele Panozzo for his support in compiling the library, as well as Ladislav Kavan & Olga Sorkine for providing results of their technique [KS12] and helpful discussion. We also thank Noura Faraj for producing the various tetrahedral meshes used in this work, the artists of Blendswap, Keenan Crane and Alec Jacobson for sharing their meshes, as well as AimAtShapes for providing the watertight models (***Armadillo***, ***Feline***, ***Hand***).

Appendix A: Deformation Jacobian of Spines

For a spine with transformation given by Eq. 2, its Jacobian at point p with parameter u_p is

$$J_p = R_{\mathbf{s}} \cdot [R_{\text{loc}}(u_p) + (R'_{\text{loc}}(u_p) \cdot p + t'_{\text{loc}}(u_p)) \cdot \overrightarrow{\nabla} u^{\mathrm{T}}(p)]. \quad (14)$$

In Eq.14, $\overrightarrow{\nabla} u(p)$ is needed. Our work uses a linear parametrization, which is is not differentiable at $u = 0$ and 1. Moreover, such parameterizations are generally defined on the mesh only (e. g., hand-painted, or resulting from a diffusion on the mesh), and their gradients do not have a closed-form expression. We thus estimate the gradient $\overrightarrow{\nabla} u_i$ at vertex i (while keeping it aligned with the bone) as

$$\overrightarrow{\nabla} u_i := \underset{g \,|\, g \times \overrightarrow{e_0 e_1} = \vec{0}}{\operatorname{argmin}} \sum_{k \in V_1(i)} \lambda_{ik} ||g^{\mathrm{T}} \cdot e_{ik} - (u(v_k) - u(v_i))||^2.$$

Noting $a_{\mathbf{s}[\times]}$ the 3×3-matrix such that $a_{\mathbf{s}[\times]} \cdot v = a_{\mathbf{s}} \times v$, $\forall v \in \mathbb{R}^3$, the derivatives of $R_{\text{loc}}(u)$ and $t_{\text{loc}}(u)$ are given by:

$$R'_{\text{loc}}(u) = \theta_{\mathbf{s}} \cos(u\theta_{\mathbf{s}}) a_{\mathbf{s}[\times]} + \theta_{\mathbf{s}} \sin(u\theta_{\mathbf{s}})(a_{\mathbf{s}} \cdot a_{\mathbf{s}}^{\mathrm{T}} - \mathrm{Id})$$

$$\begin{aligned} t'_{\text{loc}}(u) = & -R'_{\text{loc}}(u) \cdot e_0 + \\ & \sigma_{\mathbf{s}}.(\cos(u\theta_{\mathbf{s}}) + u\theta_{\mathbf{s}} \sin(u\theta_{\mathbf{s}}))(\mathrm{Id} - a_{\mathbf{s}} \cdot a_{\mathbf{s}}^{\mathrm{T}}) \cdot \overrightarrow{e_0 e_1} + \\ & \sigma_{\mathbf{s}}.(u\theta_{\mathbf{s}} \cos(u\theta_{\mathbf{s}}) - \sin(u\theta_{\mathbf{s}})) a_{\mathbf{s}} \times \overrightarrow{e_0 e_1} + \\ & (\sigma_{\mathbf{s}} - 1) R_{\text{loc}}(u) \cdot \overrightarrow{e_0 e_1} + \\ & u.(\sigma_{\mathbf{s}} - 1) R'_{\text{loc}}(u) \cdot \overrightarrow{e_0 e_1}. \end{aligned}$$

References

[AV07] Arthur D., Vassilvitskii S.: k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms* (2007), pp. 1027–1035. 8

[BP07] Baran I., Popović J.: Automatic rigging and animation of 3d characters. In *ACM Transactions on Graphics* (2007), vol. 26, p. 72. 2, 3, 6, 8, 9, 11

[BS08] Botsch M., Sorkine O.: On linear variational surface deformation methods. *IEEE Transactions on Visualization and Computer Graphics 14*, 1 (2008), 213–230. 2

[CPSS10] Chao I., Pinkall U., Sanan P., Schröder P.: A simple geometric model for elastic deformations. *ACM transactions on graphics (TOG) 29*, 4 (2010), 38. 10

[dATTHP08] de Aguiar E., Theobalt C., Thrun S., H.-P. S.: Automatic conversion of mesh animations into skeleton-based animations. 2

[DdL13] Dionne O., de Lasa M.: Geodesic voxel binding for production character meshes. In *Proceedings of the 12th ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2013), SCA '13. 2, 10

[DH09] Davis T. A., Hager W. W.: Dynamic supernodes in sparse cholesky update/downdate and triangular solves. *ACM Transactions on Mathematical Software (TOMS) 35*, 4 (2009), 27. 6

[FOKGM07] Forstmann S., Ohya J., Krohn-Grimberghe A., McDougall R.: Deformation styles for spline-based skeletal animation. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2007), SCA '07. 3

[JBK*12] Jacobson A., Baran I., Kavan L., Popović J., Sorkine O.: Fast automatic skinning transformations. *ACM Transactions on Graphics (proceedings of SIGGRAPH) 31*, 4 (2012), 77:1–77:10. 8

[JBPS11] Jacobson A., Baran I., Popovic J., Sorkine O.: Bounded biharmonic weights for real-time deformation. *ACM Transactions on Graphics 30*, 4 (2011), 78. 2, 6, 10, 12

[JDKL14] Jacobson A., Deng Z., Kavan L., Lewis J.: Skinning: Real-time shape deformation. In *ACM SIGGRAPH 2014 Courses* (2014). 2, 11

[JKSH13] Jacobson A., Kavan L., Sorkine-Hornung O.: Robust inside-outside segmentation using generalized winding numbers. *ACM Transactions on Graphics 32*, 4 (2013), 33. 10

[JP*14] Jacobson A., Panozzo D., et al.: libigl: A simple C++ geometry processing library, 2014. http://libigl.github.io/libigl/. 10, 12

[JS11] Jacobson A., Sorkine O.: Stretchable and twistable bones for skeletal shape deformation. *ACM Transactions on Graphics 30*, 6 (2011), 165. 2, 3

[JWS12] Jacobson A., Weinkauf T., Sorkine O.: Smooth shape-aware functions with controlled extrema. In *Computer Graphics Forum* (2012), vol. 31, pp. 1577–1586. 2, 8

[KS12] Kavan L., Sorkine O.: Elasticity-inspired deformers for character articulation. *ACM Transactions on Graphics 31*, 6 (2012), 196. 2, 11, 12

[KSO10] Kavan L., Sloan P.-P., O'Sullivan C.: Fast and efficient skinning of animated meshes. In *Computer Graphics Forum* (2010), vol. 29, pp. 327–336. 2

[LCF00] Lewis J. P., Cordner M., Fong N.: Pose space deformation: a unified approach to shape interpolation and skeleton-driven deformation. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques* (2000), pp. 165–172. 2

[LD12] Le B. H., Deng Z.: Smooth skinning decomposition with rigid bones. *ACM Transactions on Graphics 31*, 6 (2012), 199. 2

[LD14] Le B. H., Deng Z.: Robust and accurate skeletal rigging from mesh sequences. *ACM Transactions on Graphics 33*, 4 (2014), 84. 2

[MTLT*88] Magnenat-Thalmann N., Laperrire R., Thalmann D., et al.: Joint-dependent local deformations for hand animation and object grasping. In *Proceedings on Graphics interface 88* (1988). 2

[NS13] Nieto J. R., Susín A.: *Cage Based Deformations: A Survey*. Springer Netherlands, 2013. doi:10.1007/978-94-007-5446-1_3. 2

[NVT*14] Neumann T., Varanasi K., Theobalt C., Magnor M., Wacker M.: Compressed manifold modes for mesh processing. In *Computer Graphics Forum* (2014), vol. 33, pp. 35–44. 4

[PP93] Pinkall U., , Polthier K.: Computing discrete minimal surfaces and their conjugates. *Experimental Mathematics 2* (1993), 15–36. 7

[SA07] Sorkine O., Alexa M.: As-rigid-as-possible surface modeling. In *Symposium on Geometry processing* (2007), vol. 4, pp. 109–116. 3, 5

[SJP*13] Sacht L., Jacobson A., Panozzo D., Schüller C., Sorkine-Hornung O.: Consistent volumetric discretizations inside self-intersecting surfaces. *Computer Graphics Forum (proceedings of EUROGRAPHICS/ACM SIGGRAPH Symposium on Geometry Processing) 32*, 5 (2013), 147–156. 10