Direct Delta Mush Skinning and Variants

BINH HUY LE, SEED - Electronic Arts
JP LEWIS, Google AI

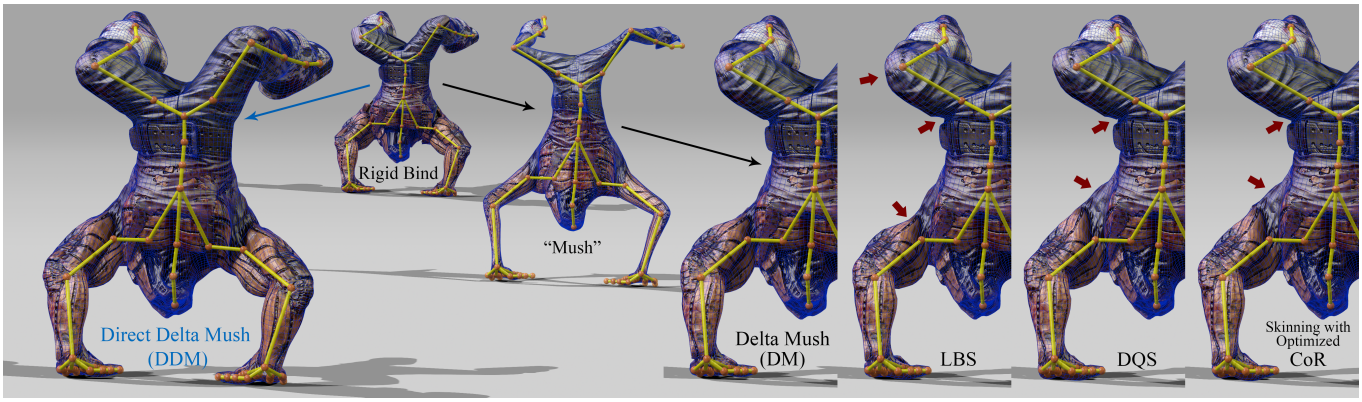


Fig. 1. The skinned model (left) is produced directly from the "unrigged" rigid bind model using our Direct Delta Mush algorithm. DDM can produce equivalent results to the Delta Mush algorithm but uses a direct local computation rather than the iterated global "mush" runtime smoothing of DM. The DM and DDM algorithms both provide greatly simplified authoring. They do not have the bulge and cleft artifacts common to other methods, which are prominent in the under-arm and hip regions (respectively) in this example (red arrows). DDM offers further advantages over DM, as described in the paper.

A significant fraction of the world's population have experienced virtual characters through games and movies, and the possibility of online VR social experiences may greatly extend this audience. At present, the skin deformation for interactive and real-time characters is typically computed using geometric skinning methods. These methods are efficient and simple to implement, but obtaining quality results requires considerable manual "rigging" effort involving trial-and-error weight painting, the addition of virtual helper bones, etc. The recently introduced Delta Mush algorithm largely solves this rig authoring problem, but its iterative computational approach has prevented direct adoption in real-time engines.

This paper introduces Direct Delta Mush, a new algorithm that simultaneously improves on the efficiency and control of Delta Mush while generalizing previous algorithms. Specifically, we derive a direct rather than iterative algorithm that has the same ballpark computational form as some previous geometric weight blending algorithms. Straightforward variants of the algorithm are then proposed to further optimize computational and storage cost with insignificant quality losses. These variants are equivalent to special cases of several previous skinning algorithms.

Our algorithm simultaneously satisfies the goals of reasonable efficiency, quality, and ease of authoring. Further, its explicit decomposition of rotational and translational effects allows independent control over bending versus twisting deformation, as well as a skin sliding effect.

Authors' addresses: Binh Huy Le, SEED - Electronic Arts, Redwood City, CA, bbinh85@gmail.com; JP Lewis, Google AI, San Francisco, CA, noisebrain@gmail.com.

0730-0301/2019/7-ART113 $15.00
https://doi.org/10.1145/3306346.3322982

CCS Concepts: • **Computing methodologies** → **Animation**.

Additional Key Words and Phrases: skinning, skeletal animation, delta mush, real time, deformation, character animation

ACM Reference Format:
Binh Huy Le and JP Lewis. 2019. Direct Delta Mush Skinning and Variants. *ACM Trans. Graph.* 38, 4, Article 113 (July 2019), 13 pages. https://doi.org/10.1145/3306346.3322982

1 INTRODUCTION

Typically characters are the main focus of any movie or game. Major characters are often humans or animals, and thus are articulated models with rigid bones underlying deformable flesh and skin. Other objects in the scene such as trees can deform and may also be represented with a similar underlying approach. A key focus in all these cases is getting the deformation right.

A character deformation method suitable for games and interactive applications such as animation should have the following characteristics: (1) speed, (2) quality, (3) simplicity of setup and authoring. Existing approaches to character deformation can be very broadly classified into geometric skinning and simulation approaches. Simulation approaches produce the highest quality but may be less suitable in terms of criteria (1) and (3). Regarding speed, simulation effects are not justified when nearly the same effect can be produced with a cheaper method. It should be remembered that character deformation is just one of many things that must be computed within the frame interval at typical frame rates of 24fps (movie animation), 60fps (games) or 120fps (VR). Other tasks include various rendering steps, gameplay AI, collision detection, other types of physics, etc. Simulation approaches are also not ideal in terms of simplicity. The rig may require constructing additional components

(such as interior structures) beyond the surface geometry, and when the simulation does not produce the desired deformation it may not be immediately obvious what should be changed.

Geometric skinning methods have been the predominant approach in movies and games to date. These approaches are simple to implement, very efficient, and map easily to GPU hardware. They fail on criterion (3) however. Authoring the desired deformation is difficult because the artist must specify blending weights that are only indirectly related to the desired deformed shapes. This task is further complicated in regions such as the shoulder and hips that are influenced by more than two bones, typically resulting in time-consuming trial and error exploration. Geometric skinning methods also have common artifacts, such as the bulges and clefts visible in Fig. 1, due to blending the potentially large movements resulting from rotating a vertex about distant joints. Further, there is not necessarily a single set of weights that is optimal for all poses, so the artist must visualize the results on a range of different poses when assessing a new set of weights. Lastly, even in a simple one degree of freedom case such as the elbow, the resulting deformation is not necessarily realistic, and artists may need to resort to introducing auxiliary "phantom" bones, a.k.a. helper bones, to better produce the desired deformation.

Delta Mush (DM) [Mancewicz et al. 2014] fits the bill for (1) and (3). Since its introduction in 2014, DM has become widely used in industry and has been incorporated in software such as Maya[1]. DM uses the following practical approach to geometric skinning: a) first start with low quality rig, typically using trivial rigid binding of geometry (i.e. each vertex moves rigidly with one bone, so all skinning weights are 0/1), and then b) smooth the geometry, which also converts the rigid articulation to smooth deformation – thus the term "mush". During the smoothing step, surface details are lost, so c) surface details at the rest pose (specifically the difference between the original shape and its smoothed version – thus the term "delta") are saved and added back on top of the smoothed deformed shape.

DM is immediately suitable for non-real-time applications such as movies, because it provides an easy way to produce a fairly high quality rig, free from most of the undesired collapsing and bulging artifacts occurring in previous skinning algorithms. However, it is expensive for interactive applications such as games. The reason is that the smoothing step is an iterative process (Laplacian smoothing) and this iteration needs to be done at every animation frame. In the games industry, a common practice is to "bake" the DM to a traditional skinning model, i.e., use DM to generate training poses which are then fit by solving for the skinning weights of the classical linear blend skinning (LBS) model [Le and Deng 2012]. This model fitting is slow, and the use of different training poses can produce significantly different results. As a consequence, artists have no interactive feedback, so DM also fails on criterion (3) when used in conjunction with a baking procedure to permit real-time use.

This paper introduces the Direct Delta Mush (DDM) algorithm. DDM addresses the above problems by mathematically re-expressing the DM calculation into a per-frame *direct* (rather than iterative) form, more analogous to the computation in traditional skeletal skinning models. Superficially, DDM has the same setup as DM but instead of smoothing the geometry at every frame, DDM only smooths a set of multi-weights [Merry et al. 2006; Wang and Phillips 2002] at the rest pose in a pre-computation step and caches these weights (§3.2).

DDM inherits the benefits of DM, in particular, the ability to use a trivial "rigid bind" rigging, while also offering greatly accelerated computation. It is thus the first algorithm to approximately satisfy all criteria (1)-(3). Specifically, DDM has the following advantages and benefits:

- Speed: DDM computation is in the ballpark of traditional skinning methods, and has a considerably reduced operation count relative to DM (Table 1). Our CPU-only implementation easily runs complex characters in real time.
- Quality: as with DM, DDM provides a somewhat more general class of deformations than is possible with traditional skinning methods (Section 3.3).
- Authoring: as with DM, reasonable results can be obtained with almost no effort. The time-consuming weight editing and helper bones required in skinning methods to obtain quality results is not needed in common cases. Unlike DM, our formulation also provides easy localized control over the extent of the smoothing or blending (Fig. 7).
- Generality: a straightforward modification allows DDM to emulate or specialize to several existing skinning algorithms (Section 3.5). DDM is thus a framework that generalizes several existing approaches.
- Skin sliding: DDM has an explicit decomposition into rotation and translation which enables a simple emulation of a skin-sliding effect (Figs. 8, 12 and accompanying Video), something that is not possible with existing geometric skinning algorithms.

2 RELATED WORK

For our purposes deformation can be classified into geometric, data-driven, and simulation-based methods. Geometric methods in turn can be described according to whether they use a direct (local) per-vertex calculation or a global computation involving all vertices, and according to the type of control parameterization, i.e. skeleton joint angles, cages, or other schemes. Our survey of related work will focus on methods suitable for real-time deformation, and in particular on direct skeleton-driven geometric techniques motivated by character animation.

Blend skinning methods include LBS [Magnenat-Thalmann et al. 1988], log-matrix blending [Alexa 2002; Magnenat-Thalmann et al. 2004], spherical blend skinning [Kavan and Žára 2005], dual quaternion skinning (DQS) [Kavan et al. 2008], and optimized centers of rotation skinning (CoR) [Le and Hodgins 2016]. These methods are simple to implement, fast, map conveniently to GPU hardware, and are implemented in most software packages. Although these skinning methods have known drawbacks as mentioned in the introduction, they are widely used in games and other interactive applications

[1] https://youtu.be/EaCktzhxbTA

Linear multi-weight schemes [Merry et al. 2006; Wang and Phillips 2002] extend classic skinning techniques by providing more than one weight per transform or "bone", thus allowing a bone to influence each coordinate of a vertex differently [Merry et al. 2006], or further allowing different coordinates to be influenced by different components of a bone's movement [Wang and Phillips 2002]. The resulting schemes have fewer artifacts than LBS, but the weights are no longer intuitive and must be determined by a fitting process – which means that some other source of example poses is needed. Despite the additional flexibility, these are still linear schemes having the consequent difficulties in emulating rotational effects.

Special purpose algorithms are used to solve the "elbow" situation, such as using dedicated twisting and stretching deformers [Jacobson and Sorkine 2011], joint-based deformers [Kavan and Sorkine 2012], and using extra helper joints to reduce the deformation artifacts [Le and Hodgins 2016; Mukai 2015; Mukai and Kuriyama 2016]. These are hard to generalize to characters with non-hierarchical skeletons and typically require extra authoring effort to setup the required extra weights or extra animation on helper joints.

Auto-generated skinning weights [Baran and Popović 2007; Jacobson et al. 2011, 2012b] solve the problem of doing manual weight-painting that is needed in traditional geometric skinning methods. This does not solve the collapsing or bulging artifacts found in most geometric skinning algorithms, however, so to obtain the highest quality results artists must still resort to introducing additional helper bones and manual weight adjustment.

Global geometric deformation improves quality by solving for the deformation of all vertices at once, often using a variant of a Laplace or Poisson equation [Wang et al. 2007], rather than using a direct per-vertex computation. This comes at a cost, however – even when the underlying linear system is pre-factorized, the resulting back substitution is poorly suited for parallel computation. Nonlinear effects can be introduced with iterative optimization that alternates global smoothing and local rotational computations [Sorkine and Alexa 2007]. The cost of global methods has been addressed with linear reduced models [Jacobson et al. 2012a; Wang et al. 2015].

Cage deformation methods [Jacobson et al. 2011; Joshi et al. 2007; Ju et al. 2005; Lipman et al. 2008] provide very general deformation that can be used on a variety of shapes. The cage provides extra degrees of freedom compared to a skeletal model parameterized by only the joint angles. This can be an advantage for cartoon characters, but is unnecessary for more realistic characters where the bones do not stretch.

Baking, skinning decomposition, and general regression can automatically convert animated mesh sequences into skinned models suitable for real-time playback. A popular pipeline is to use these methods to bake complex rigs authored offline, fit training data from motion capture (mocap) [Park and Hodgins 2008], DM (Maya's Bake Deformer tool[2]) or muscle models [James and Twigg 2005; Le and Deng 2012, 2014; Mukai 2015; Mukai and Kuriyama 2016]. More generally, the approach of approximating training poses from mocap data or a given rig can be considered as a regression problem [Anguelov et al. 2005; Feng et al. 2008; Gao et al. 2016; Jones et al. 2016; Loper et al. 2014, 2015; Wang et al. 2007].

[2]https://youtu.be/5i-gxtXj1Ww

Corrective shapes-based methods such as Eigenskin [Kry et al. 2002] and pose space deformation [Lewis et al. 2000; Sloan et al. 2001] compute the deformed geometry by interpolating example shapes rather than using algorithmic deformation. These methods can provide high quality but require additional geometry that must be sculpted or scanned. For the purpose of body animation, these methods are orthogonal to our goal, since they are often employed as a corrective layer on top of algorithmically defined skinning such as ours.

Implicit skinning [Vaillant et al. 2013, 2014] provides a relatively cheap solution for local self-collision without requiring a full physics simulation. At the time of its publication the algorithm was capable of computing a single high-resolution character in real time, which is sufficient for animation editing and preview but still too expensive for applications involving many computational tasks beyond character deformation.

Simulation methods [Hahn et al. 2012, 2013; Ichim et al. 2017; Kadleček et al. 2016; Li et al. 2013; Liu et al. 2013; McAdams et al. 2011; Rémillard and Kry 2013; Saito et al. 2015; Smith et al. 2018] provide important additional effects such as collision, volume-preserving bulging, jiggling, skin sliding and wrinkling , Until recently simulation methods were only suitable for offline production, e.g. Maya Muscle[3] or Weta Digital's Tissue System[4]), however thanks to recent methods [Bouaziz et al. 2014; Brandt et al. 2018; Dinev et al. 2018; Liu et al. 2017; Xu and Barbič 2016] certain types of real-time simulation are now possible. These methods are particularly suitable for dynamic effects (jiggling) as well as types of quasi-static deformation that cannot be produced by cheap skinning methods.

Delta Mush (DM) [Mancewicz et al. 2014] stands out as a particularly simple and artist friendly method. It does not involve indirect and possibly non-intuitive concepts such as weights on transforms, but rather considers skinning simply as a problem of smoothing the low-frequency geometry while preserving detail. It is also versatile, as it can be applied either to "unrigged" rigidly articulated models or on top of other deformation. Lastly, DM has the distinction of being widely used in production. The major limitation of DM is its iterative computation with required global synchronizations between iterations, which significantly impacts efficiency.

Our algorithm, DDM, is inspired by DM but takes a different computational approach. In so doing it unifies and generalizes several existing geometric skinning approaches, while simultaneously improving on DM in several respects. The algorithm is described next.

3 DDM COMPUTING MODEL AND VARIANTS

The overall idea of our algorithm is, instead of computing the smooth geometry after applying bone transformations as in DM, we treat the bone transformations as unknown parameters and substitute them into the Laplacian smoothing equation. Because Laplacian smoothing can be represented by a series of linear matrix multiplications, it can be combined with linear bone transformations. Simplifying the equations yields the direct formulation.

[3]www.autodesk.com
[4]www.wetafx.co.nz

Following subsections detail this idea. First, the original DM algorithm is presented using our notation (§3.1). Then, the DDM formulation is presented (§3.2). A simple extension generalizes the achievable deformation and provides further control (§3.3). Finally, the precomputation for DDM is summarized (§3.4) and some variants are presented to show how our algorithm unifies and generalizes several existing approaches (§3.5).

3.1 Original Delta Mush Model

This section revises the original Delta Mush (DM) model [Mancewicz et al. 2014] and presents notations to setup DM on top of a Linear Blend Skinning (LBS) model [Magnenat-Thalmann et al. 1988].

Assume that our character model is represented by a polygonal mesh with n vertices. The homogeneous position of vertex $i = 1..n$ at the rest pose is $\mathbf{u}_i \in \mathbb{R}^4$, where the 4^{th} component is 1. For convenience, we concatenate all vectors $\mathbf{u}_i$ to a matrix $\mathbf{U} = [\mathbf{u}_1, \mathbf{u}_2, .., \mathbf{u}_n] \in \mathbb{R}^{4\times n}$.

The deformation of $\mathbf{U}$ is driven by a LBS model with m bones. The transformation of bone $j = 1..m$ is $\mathbf{M}_j \in \mathbb{R}^{4\times 4}$. Let w_{ij} be the weight of bone j on vertex i. The weights are required to be affine, i.e. $\sum_{j=1}^{m} w_{ij} = 1, \forall i$. Non-negativity and sparseness have no effect on our formulation. Note that we use the LBS model for the sake of generality, but rigid binding is more common in practice, i.e. each vertex is only assigned to one bone ($w_{ij} \in \{0, 1\}$). The skinned geometry $\mathbf{V} = [\mathbf{v}_1, \mathbf{v}_2, .., \mathbf{v}_n] \in \mathbb{R}^{4\times n}$ is computed as:

$$\mathbf{v}_i = \sum_{j=1}^{m} w_{ij}\mathbf{M}_j\mathbf{u}_i \ , \ i = 1..n \tag{1}$$

Layered on top of LBS model is a DM deformer with p Laplacian smoothing iterations, where the step size of each iteration is $\lambda > 0$. $p\lambda$ controls the amount of smoothness applied on the model and λ controls the precision of the smoothing. Let $\mathbf{L} \in \mathbb{R}^{n\times n}$ be the Laplacian matrix of the mesh. $\mathbf{L}$ is symmetric, positive semidefinite with zero row- and column-sums. Let $\bar{\mathbf{L}} = \mathbf{L}\mathbf{D}_\mathbf{L}{}^{-1}$ be the normalized Laplacian, where $\mathbf{D}_\mathbf{L}$ is the diagonal of $\mathbf{L}$ (note that this expression is transposed with respect to the usual spectral graph theory notation in order to be consistent with notation used in the remainder of the paper). The smooth rest pose $\tilde{\mathbf{U}} = [\tilde{\mathbf{u}}_1, \tilde{\mathbf{u}}_2, .., \tilde{\mathbf{u}}_n] \in \mathbb{R}^{4\times n}$ and the smooth skinned pose $\tilde{\mathbf{V}} = [\tilde{\mathbf{v}}_1, \tilde{\mathbf{v}}_2, .., \tilde{\mathbf{v}}_n] \in \mathbb{R}^{4\times n}$ can be computed with either explicit or implicit schemes [Desbrun et al. 1999]:

$$\text{Explicit: } \tilde{\mathbf{U}} = \mathbf{U}\mathcal{A} \ , \ \tilde{\mathbf{V}} = \mathbf{V}\mathcal{A} \ , \ \text{where: } \mathcal{A} = (\mathbb{I} - \lambda\bar{\mathbf{L}})^p \ , \tag{2a}$$

$$\text{Implicit: } \tilde{\mathbf{U}} = \mathbf{U}\mathcal{B} \ , \ \tilde{\mathbf{V}} = \mathbf{V}\mathcal{B} \ , \ \text{where: } \mathcal{B} = (\mathbb{I} + \lambda\bar{\mathbf{L}})^{-p} \ , \tag{2b}$$

$$\text{where } \mathbb{I} \text{ is the identity matrix}$$

Eq. (2) provides a consistent definition for both implicit and explicit methods. In practice, $\tilde{\mathbf{U}}$ and $\tilde{\mathbf{V}}$ are computed in an iterative process, i.e. explicitly compute $\mathbf{U}_k \leftarrow \mathbf{U}_{k-1}(\mathbb{I} - \lambda\bar{\mathbf{L}})$ or implicitly solve $\mathbf{U}_k(\mathbb{I} + \lambda\bar{\mathbf{L}}) = \mathbf{U}_{k-1}$, where $\mathbf{U}_0 = \mathbf{U}$ and $\mathbf{U}_p = \tilde{\mathbf{U}}$. The implicit method is unconditionally stable and it allows using a large step size λ while the explicit method is only stable with $\lambda \leq 1$. With small enough step size, both methods produce very similar results. In this work, we present the formulation and implement the implicit method. However, the formulation for the explicit method is exactly the same, with the only exception of using matrix $\mathcal{A}$ instead of $\mathcal{B}$.

To recover the surface details lost in smoothing, the delta $\mathbf{d}_i \in \mathbb{R}^3$ at vertex i is computed as the difference between the original rest pose $\mathbf{U}$ and the smooth version $\tilde{\mathbf{U}}$:

$$\begin{bmatrix}\mathbf{d}_i \\ 0\end{bmatrix} = \mathbf{u}_i - \tilde{\mathbf{u}}_i \tag{3}$$

The final deformed position $\mathbf{x}_i$ is computed by adding the delta after a local frame transformation $\mathbf{R}_i \in \mathbb{R}^{3\times 3}$ to the smooth skinned vertex:

$$\mathbf{x}_i = \tilde{\mathbf{v}}_i + \begin{bmatrix}\mathbf{R}_i\mathbf{d}_i \\ 0\end{bmatrix} \tag{4}$$

The local frame transformation $\mathbf{R}_i$ is expected to be a rotation matrix, i.e. $\mathbf{R}_i{}^\mathsf{T}\mathbf{R}_i = \mathbb{I}$ and $\det(\mathbf{R}_i) = 1$. In practice, $\mathbf{R}_i$ can be approximated from two surface tangent vectors [Mancewicz et al. 2014].

3.2 Direct Delta Mush Skinning

Because the original DM tries to preserve the delta $\mathbf{d}_i$ in the local (orthogonal) coordinate frame, we can directly find the local rigid transformation Γ_i to bring that coordinate frame from the rest pose to the deformed pose. Therefore, Γ_i should also bring $\tilde{\mathbf{u}}_i$ to $\tilde{\mathbf{v}}_i$. Mathematically, we have:

$$\mathbf{x}_i = \Gamma_i\mathbf{u}_i = \Gamma_i\tilde{\mathbf{u}}_i + \Gamma_i(\mathbf{u}_i - \tilde{\mathbf{u}}_i) = \Gamma_i\tilde{\mathbf{u}}_i + \begin{bmatrix}\mathbf{R}_i\mathbf{d}_i \\ 0\end{bmatrix}$$

which matches Eq. (4) if $\tilde{\mathbf{v}}_i = \Gamma_i\tilde{\mathbf{u}}_i$

From Eq. (2), we can view $\tilde{\mathbf{u}}_i$ and $\tilde{\mathbf{v}}_i$ as the weighted averaging of non-smooth geometries $\mathbf{U}$ and $\mathbf{V}$, i.e. $\tilde{\mathbf{u}}_i = \sum_{k=1}^{n} \mathcal{B}_{ki}\mathbf{u}_k$ and $\tilde{\mathbf{v}}_i = \sum_{k=1}^{n} \mathcal{B}_{ki}\mathbf{v}_k$. Intuitively, the Laplacian smoothing can be viewed as a per-vertex weighted averaging operator where the weights for vertex i are elements of the column $\mathcal{B}_i$.

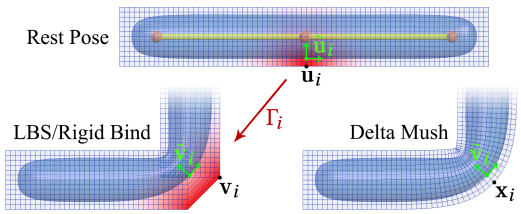


Fig. 2. The core idea of Direct Delta Mush schemes. The accumulated affect of iterated Laplacian smoothing is captured in a per-vertex weight mask (column B_i, red color gradient) that if applied to the undeformed geometry produces the equivalent of the iteratively smoothed result (inner blue geometry). These weights are used to solve for the local coordinate transformation Γ_i which is defined as the best rigid transformation to align the local patch (red regions) from the rest pose to the LBS deformed pose, a.k.a. a form of Weighted Procrustes problem. The delta between $\mathbf{u}_i$ and $\tilde{\mathbf{u}}_i$ is also transformed and added to the smoothed deformed position $\tilde{\mathbf{v}}_i$, producing the final vertex position $\mathbf{x}_i$.

Therefore, the transformation Γ_i can be seen as the best transformation to bring the set of all vertices $\mathbf{U}$ to $\mathbf{V}$ with weights $\mathcal{B}_i$ as illustrated in Fig. 2, which can be obtained by minimizing the objective function:

$$\min E(\Gamma) = \sum_{k=1}^{n} \mathcal{B}_{ki} \left\| \mathbf{v}_k - \Gamma \mathbf{u}_k \right\|_2^2 \tag{5}$$

To clarify the presentation, we will only summarize the main steps in the remainder of this section. Detailed derivations are presented in the Appendices.

3.2.1 Expanding and Regrouping. The objective function (5) is the sum of per-vertex quadratic terms with respect to vertex index k. Substituting $\mathbf{v}_k$ from the LBS equation (1) and expanding the function using the trace notation $\mathrm{tr}(\bullet)$ yields a sum of quadratic terms with respect to vertex index k and bone index j. Regrouping terms by the bone transformations $\mathbf{M}_j$ yields (please see more details in Appendix A):

$$E(\Gamma) = \mathrm{tr}\left(\Gamma \sum_{j=1}^{m} \Psi_{ij} \Gamma^{\mathsf{T}}\right) - 2\,\mathrm{tr}\left(\sum_{j=1}^{m} \mathbf{M}_j \Psi_{ij} \Gamma^{\mathsf{T}}\right) + \text{const.}\,, \tag{6a}$$

$$\text{where: } \Psi_{ij} = \sum_{k=1}^{n} \mathcal{B}_{ki} w_{kj} \mathbf{u}_k \mathbf{u}_k{}^{\mathsf{T}} \tag{6b}$$

This regrouping allows pre-computing and caching matrices $\{\Psi_{ij} \in \mathbb{R}^{4\times4} \mid j = 1..m\}$ from the rest pose (vertices $\mathbf{u}_k$) and scalar weights ($\mathcal{B}_{ki} w_{kj}$), which are constants with respect to bone transformations. During animation, when the skeleton pose is determined by all bone transformations $\{\mathbf{M}_j \mid j = 1..m\}$, we can compute E by pre-multiplying $\mathbf{M}_j$ to Ψ_{ij}. This regrouped function is much faster than the original function (5) because the calculation does not depend on the number of vertices n, but only depends on the number of bones $m << n$.

Note that the block matrix Ψ is sparse, i.e. many Ψ_{ij} are zero matrices as they correspond to pairs of (i, j) with $\sum_{k=1}^{n} \mathcal{B}_{ki} w_{kj} = 0$.

3.2.2 Decoupling Rotation and Translation. Similar to a Procrustes problem [Kabsch 1978; Wahba 1965], we need to separate the rotation and translation. Let:

$$\begin{bmatrix} \mathbf{P}_i & \mathbf{p}_i \\ \mathbf{p}_i{}^{\mathsf{T}} & 1 \end{bmatrix} = \sum_{j=1}^{m} \Psi_{ij}\,, \quad \begin{bmatrix} \mathbf{Q}_i & \mathbf{q}_i \\ \mathbf{p}_i{}^{\mathsf{T}} & 1 \end{bmatrix} = \sum_{j=1}^{m} \mathbf{M}_j \Psi_{ij}\,, \tag{7}$$

$$\text{where: } \mathbf{P}_i \in \mathbb{R}^{3\times3}, \mathbf{Q}_i \in \mathbb{R}^{3\times3}, \mathbf{p}_i \in \mathbb{R}^3, \mathbf{q}_i \in \mathbb{R}^3$$

Next, performing Eigen decomposition of the symmetric, positive definite matrix $\mathbf{P}_i - \mathbf{p}_i \mathbf{p}_i{}^{\mathsf{T}}$ gives:

$$\mathbf{P}_i - \mathbf{p}_i \mathbf{p}_i{}^{\mathsf{T}} = \mathbf{Z}_i \Lambda_i \mathbf{Z}_i{}^{\mathsf{T}}\,, \tag{8}$$

where: $\mathbf{Z}_i$ is an orthogonal matrix, Λ_i is a diagonal matrix

Letting $\Gamma = \begin{bmatrix} \Phi & \tau \\ 0 & 1 \end{bmatrix}$, where $\Phi \in \mathbb{R}^{3\times3}$ is the rotation matrix and $\tau \in \mathbb{R}^3$ is the translation vector, then substituting Eq. (7) and Eq. (8) into Eq. (6) yields (please see details in Appendix B):

$$E(\Gamma) = \left\| \Phi \mathbf{Z}_i \Lambda_i{}^{1/2} - (\mathbf{Q}_i - \mathbf{q}_i \mathbf{p}_i{}^{\mathsf{T}}) \mathbf{Z}_i \Lambda_i{}^{-1/2} \right\|_F^2 + \left\| (\mathbf{q}_i - \Phi \mathbf{p}_i) - \tau \right\|_2^2 + \text{const.} \tag{9}$$

Finally, comparing Eq. (5), we have the solution:

$$\mathbf{R}_i = \arg\min_{\Phi} \left\| \Phi \mathbf{Z}_i \Lambda_i{}^{1/2} - (\mathbf{Q}_i - \mathbf{q}_i \mathbf{p}_i{}^{\mathsf{T}}) \mathbf{Z}_i \Lambda_i{}^{-1/2} \right\|_F^2\,, \tag{10a}$$

$$\mathbf{t}_i = \mathbf{q}_i - \mathbf{R}_i \mathbf{p}_i \tag{10b}$$

Solving Eq. (10) provides the transformation $\begin{bmatrix} \mathbf{R}_i & \mathbf{t}_i \end{bmatrix}$ for vertex i. Eq. (10a) is a Procrustes problem which can be solved by Singular Value Decomposition [Kabsch 1978; Wahba 1965]. We will further discuss cheaper approximated solutions in §3.5. Note that $\mathbf{P}_i, \mathbf{Q}_i, \mathbf{p}_i$, and $\mathbf{q}_i$ can be quickly computed from the bone transformations $\{\mathbf{M}_j \mid j = 1..m\}$ given the precomputed Ψ_{ij} in Eq. (6b). We will also recap this precomputation step in §3.4.

3.3 Extension

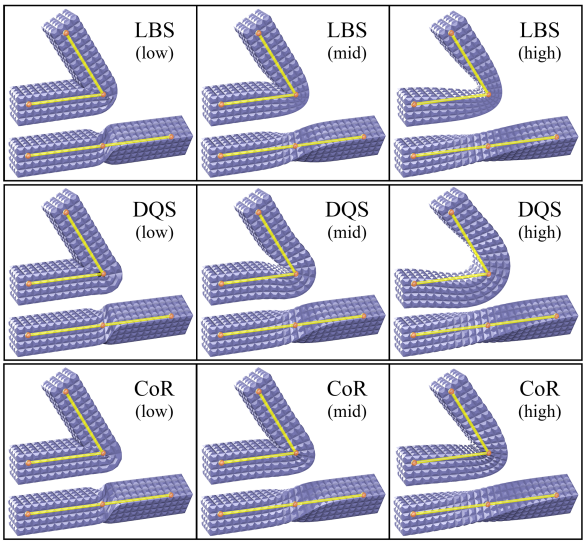


Fig. 3. None of the previous skinning methods, even LBS, is free of the "bulging" artifact. Each column visualizes deformation with different weight diffusion. The bulging is more visible with higher weight diffusion (right column). Notice how the center curves of the bars is offset from the bone segments (yellow).

DM provides an additional form of control not found in previous methods that we call *"negative bulging"*. In all other direct skeletal skinning methods, if the joint position is inside the undeformed shape it will necessarily remain inside the final deformed shape (Fig. 3). DM does not have this restriction (Fig. 4, lower right). While the behavior of the other methods is often what is desired, particularly in cases where the "bones" and joints are anatomically inspired, the flexibility afforded by DM effectively provides a type of level of detail for the skeletal control. For example, this type of deformation can allow deforming the hip joint without needing a corrective shape (Fig. 10, §4.0.3).

Combining the characteristics of both DM and other methods would give more control, in particular, this can mimic the skin sliding effect (Figs. 8 and 12), where the DM deformer configured with high smoothness creates strong influence while skeletal skinning configured with low smoothness (i.e. low weight diffusion) keeps

the center line close to the bone segments and the joint. This can be done without any extra computing or storage cost at run time by altering the precomputed Ψ_{ij} matrices so that they change the translation $\mathbf{t}_i$ but not the rotation $\mathbf{R}_i$ in Eq. (10).

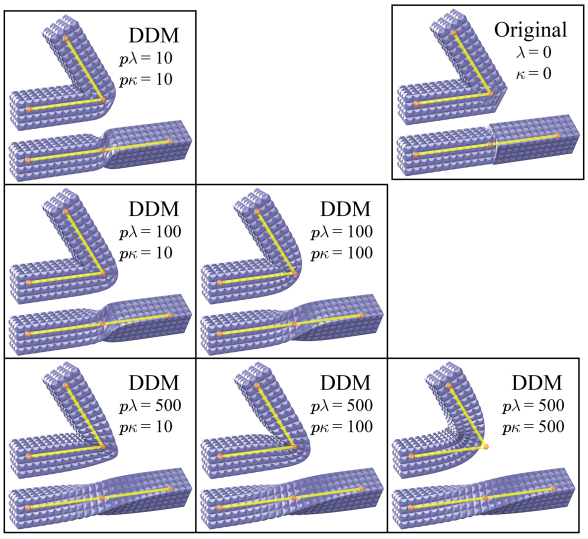


Fig. 4. Our extension: the original DM uses a single control for smoothness, which can only produce deformations similar to figures on the diagonal (with $p\lambda = p\kappa$). Our extension allows blending different rotation smoothness (controlled by $p\lambda$) and translation smoothness (controlled by $p\kappa$). This unique translation blending does not affect rotation (proven in Appendix C) which makes authoring less challenging. In the last row, changing $p\kappa$ does not change the twist deformation. The bottom left corner with $p\lambda = 500$, $p\kappa = 10$ is a good pattern for elbow deformation, which mimics the skin sliding on top of a knuckle.

Let $0 \le \alpha < 1$ be the blending weight between DM and skeletal skinning models. We can prove that for arbitrary affine skinning weights: w'_{ij}, where $\sum_{j=1}^{m} w'_{ij} = 1 \, \forall i$, using the convex combination Ω_{ij} (Eq. (11)) instead of Ψ_{ij} does not change $\mathbf{R}_i$ (Appendix C).

$$\Omega_{ij} = (1-\alpha)\Psi_{ij} + \alpha w'_{ij} \begin{bmatrix} \mathbf{p_i}\mathbf{p_i}^\mathsf{T} & \mathbf{p_i} \\ \mathbf{p_i}^\mathsf{T} & 1 \end{bmatrix} \tag{11}$$

We can utilize Laplacian smoothing, similar to Eq. (2), to construct the skinning weights:

$$w'_{ij} = \sum_{k=1}^{n} C_{ki} w_{kj}, \quad \text{where: } C = (\mathbb{I} + \kappa\bar{\mathbf{L}})^{-p} \tag{12}$$

We use the same number of iterations p for convenience of implementation, although we could theoretically use a different number. The step size κ should be set to $\kappa < \lambda$ to keep the optimization of $\mathbf{R}_i$ in Eq. (10a) well defined.

We also use a set of per-vertex weights to locally control the strength of smoothing. Let $\mathbf{D}_\lambda, \mathbf{D}_\kappa, \mathbf{D}_\alpha \in \mathbb{R}^{n\times n}$ be diagonal weight matrices, where each diagonal element represents the local strength at each vertex. We multiply these per-vertex weight matrices by the corresponding global parameters, i.e. replacing λ in Eq. 2 by $\lambda\mathbf{D}_\lambda$, replacing κ in Eq. 12 by $\kappa\mathbf{D}_\kappa$, and replacing α in Eq. 11 by $\alpha\mathbf{D}_\alpha$. Although this local control was not mentioned in the original paper [Mancewicz et al. 2014], it is implemented in Maya's Delta Mush weights tool.

3.4 Precomputation Summary

Eq. (11 rev.) precomputes all $\Omega_{ij} \in \mathbb{R}^{4\times4}$ matrices for the extended version of our algorithm (Section 3.3). Matrix $\mathcal{B}$ in Eq. (2b rev.) and matrix C in Eq. (12 rev.) are computed by implicit Laplacian smoothing of the mesh on multiple channels, where the channels at a vertex i are the $16m$ elements of $\{w_{ij}\mathbf{u}_i\mathbf{u}_i{}^\mathsf{T} \mid j = 1..m\}$ (for computing $\mathcal{B}$) or m elements of $\{w_{ij} \mid j = 1..m\}$ (for computing C).

$$\Omega_{ij} = (1 - \alpha(\mathbf{D}_\alpha)_{ii})\,\Psi_{ij} + \alpha(\mathbf{D}_\alpha)_{ii} w'_{ij} \begin{bmatrix} \mathbf{p_i}\mathbf{p_i}^\mathsf{T} & \mathbf{p_i} \\ \mathbf{p_i}^\mathsf{T} & 1 \end{bmatrix}, \tag{11 rev.}$$

$$\text{where: } \Psi_{ij} = \sum_{k=1}^{n} \mathcal{B}_{ki} w_{kj} \mathbf{u}_k \mathbf{u}_k{}^\mathsf{T}, \tag{6b rev.}$$

$$\mathcal{B} = (\mathbb{I} + \lambda\bar{\mathbf{L}}\mathbf{D}_\lambda)^{-p}, \tag{2b rev.}$$

$$\begin{bmatrix} \mathbf{P}_i & \mathbf{p}_i \\ \mathbf{p}_i{}^\mathsf{T} & 1 \end{bmatrix} = \sum_{j=1}^{m} \Omega_{ij}, \tag{7 rev.}$$

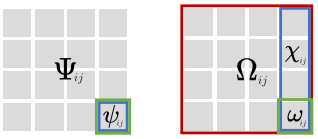
$$w'_{ij} = \sum_{k=1}^{n} C_{ki} w_{kj}, \quad C = (\mathbb{I} + \kappa\bar{\mathbf{L}}\mathbf{D}_\kappa)^{-p} \tag{12 rev.}$$

Note that matrices Ψ_{ij} and Ω_{ij} are symmetric by construction. Therefore, they can be stored with only 10 instead of 16 floats.

3.5 Run Time Variants

Our algorithm encompasses or emulates several previous methods as special cases. Fig. 5 demonstrates the effects of different variants using non-hierarchical bones. Note that the differences are less noticeable with a hierarchical skeleton setup. Table 1 summarizes storage and computation cost for different variants. Also note that the storage cost reflects the artistic design of the character (i.e. how many bones are used to influence each vertex) and has nothing to do with sparseness.

Table 1. Storage and computation cost of different variants. Numbers in parenthesizes [ ] denote the number of floats per stored element. Stored matrices are symmetric so we only need to store upper (or lower) triangles.



Variant	Storage		Special Computation
	Per-weight	Per-vertex	
v0	$\Omega_{ij}[10]$	$\mathbf{u}_i[3]$	SVD 3×3
v1	$\Omega_{ij}[10]$	$\mathbf{u}_i[3]$, $\mathbf{P}_i - \mathbf{p}_i\mathbf{p}_i{}^\mathsf{T}[6]$	Inverse 3×3
v2	$\chi_{ij}[3]$, $\omega_{ij}[1]$, $\psi_{ij}[1]$	$\mathbf{u}_i[3]$, $\mathbf{p}_i[3]$	Quaternion blend
v3	$\chi_{ij}[3]$, $\omega_{ij}[1]$, $\psi_{ij}[1]$	$\mathbf{u}_i[3]$, $\mathbf{p}_i[3]$	-
v4	$\omega_{ij}[1]$, $\psi_{ij}[1]$	$\mathbf{u}_i[3]$, $\mathbf{p}_i[3]$	Quaternion blend
v5	$\omega_{ij}[1]$	$\mathbf{u}_i[3]$	-

*3.5.1 Variant **v0** (full model).* We can directly solve for the rotation $\mathbf{R}_i$ in Eq. (10a) using Singular Value Decomposition (SVD) [Everson 1997]. Note that matrices $\mathbf{Z}_i$ and Λ_i cancel out so we only need to compute $\begin{bmatrix}\mathbf{Q}_i & \mathbf{q}_i\\ \mathbf{p}_i^\mathsf{T} & 1\end{bmatrix}$:

$$\begin{aligned}\mathbf{R}_i &= \mathcal{U}_i\mathcal{V}_i^\mathsf{T},\\ \text{where: } \mathcal{U}_i\mathcal{S}_i\mathcal{V}_i^\mathsf{T} &= (\mathbf{Q}_i-\mathbf{q}_i\mathbf{p}_i^\mathsf{T})\mathbf{Z}_i{\Lambda_i}^{-1/2}(\mathbf{Z}_i{\Lambda_i}^{1/2})^\mathsf{T}\\ &= \mathbf{Q}_i-\mathbf{q}_i\mathbf{p}_i^\mathsf{T},\\ \begin{bmatrix}\mathbf{Q}_i & \mathbf{q}_i\\ \mathbf{p}_i^\mathsf{T} & 1\end{bmatrix} &= \sum_{j=1}^{m}\mathbf{M}_j\Omega_{ij} \qquad \text{(7 rev.)}\end{aligned}$$

*3.5.2 Variant **v1** (no SVD).* Performing SVD for every vertex is costly. Instead, we can first solve Eq. (10a) without the orthogonal constraint:

$$\begin{aligned}\mathbf{R}_i' &= (\mathbf{Q}_i-\mathbf{q}_i\mathbf{p}_i^\mathsf{T})\mathbf{Z}_i{\Lambda_i}^{-1/2}(\mathbf{Z}_i{\Lambda_i}^{1/2})^\mathsf{T}\left(\mathbf{Z}_i{\Lambda_i}^{1/2}(\mathbf{Z}_i{\Lambda_i}^{1/2})^\mathsf{T}\right)^{-1}\\ &= (\mathbf{Q}_i-\mathbf{q}_i\mathbf{p}_i^\mathsf{T})(\mathbf{P}_i-\mathbf{p}_i\mathbf{p}_i^\mathsf{T})^{-1}\end{aligned}$$

The rotation matrix can be approximated by taking the inverse transpose followed by determinant normalization, which is equivalent to transforming two orthogonal tangent vectors at i using $\mathbf{R}_i'$ [Tarini et al. 2014], and thus, it is equivalent to the implementation of the original DM [Mancewicz et al. 2014]. The final rotation at vertex i is computed as:

$$\begin{aligned}\mathbf{R}_i &= \frac{1}{\det((\mathbf{R}_i')^{-\mathsf{T}})}(\mathbf{R}_i')^{-\mathsf{T}}\\ &= \frac{\det(\mathbf{Q}_i-\mathbf{q}_i\mathbf{p}_i^\mathsf{T})}{\det(\mathbf{P}_i-\mathbf{p}_i\mathbf{p}_i^\mathsf{T})}(\mathbf{Q}_i-\mathbf{q}_i\mathbf{p}_i^\mathsf{T})^{-\mathsf{T}}(\mathbf{P}_i-\mathbf{p}_i\mathbf{p}_i^\mathsf{T}),\\ \text{where: } \begin{bmatrix}\mathbf{Q}_i & \mathbf{q}_i\\ \mathbf{p}_i^\mathsf{T} & 1\end{bmatrix} &= \sum_{j=1}^{m}\mathbf{M}_j\Omega_{ij},\ \begin{bmatrix}\mathbf{P}_i & \mathbf{p}_i\\ \mathbf{p}_i^\mathsf{T} & 1\end{bmatrix} = \sum_{j=1}^{m}\Omega_{ij} \qquad \text{(7 rev.)}\end{aligned}$$

Because **v1** does not compute an exact rotation matrix, the deformation is distorted. In the extreme case, the deformation is inverted if $\det\mathbf{R}_i' < 0$ as shown in the second pose (S shape) of DDM **v1** in Fig. 5.

Note that $\det(\mathbf{Q}_i-\mathbf{q}_i\mathbf{p}_i^\mathsf{T})(\mathbf{Q}_i-\mathbf{q}_i\mathbf{p}_i^\mathsf{T})^{-\mathsf{T}}$ is the cofactor matrix of $\mathbf{Q}_i-\mathbf{q}_i\mathbf{p}_i^\mathsf{T}$. The determinant $\det(\mathbf{Q}_i-\mathbf{q}_i\mathbf{p}_i^\mathsf{T})$ cancels out in this calculation. The constant matrix $\frac{\mathbf{P}_i-\mathbf{p}_i\mathbf{p}_i^\mathsf{T}}{\det(\mathbf{P}_i-\mathbf{p}_i\mathbf{p}_i^\mathsf{T})}$ can be cached for better runtime performance.

*3.5.3 Variant **v2** and **v3**.* Let ψ_{ij} be the [4, 4]-element of matrix Ψ_{ij}. ψ_{ij} is the homogeneous component that corresponds to the contribution of each bone to the rotation. We can directly compute $\mathbf{R}_i$ by blending the rotation components of bone transformations, either in quaternion space (**v2**) or in linear space (**v3**). Let $\mathbf{M}_j^\mathbf{R}\in\mathbb{R}^{3\times3}$ and $\mathbf{M}_j^\mathbf{t}\in\mathbb{R}^3$ be the rotation part and translation part of $\mathbf{M}_j$, respectively. Let $\mathbf{M}_j^\mathbf{q}\in\mathbb{R}^4$ be the corresponding unit quaternion of $\mathbf{M}_j^\mathbf{t}\in\mathbb{R}^3$. Let $\begin{bmatrix}\chi_{ij}^\mathsf{T} & \omega_{ij}\end{bmatrix}^\mathsf{T}$ be the 4-th column of matrix Ω_{ij}, where $\chi_{ij}\in\mathbb{R}^3$ and $\omega_{ij}\in\mathbb{R}$. The transformation $\begin{bmatrix}\mathbf{R}_i & \mathbf{t}_i\end{bmatrix}^\mathsf{T}$ for vertex i is:

$$\begin{aligned}\mathbf{R}_i &= \begin{cases}\text{rotationMatrix}\left(\sum_{j=1}^{m}\psi_{ij}\mathbf{M}_j^\mathbf{q}\right) & \text{(DDM v2)}\\ \frac{1}{\det\left(\sum_{j=1}^{m}\psi_{ij}\mathbf{M}_j^\mathbf{R}\right)}\sum_{j=1}^{m}\psi_{ij}\mathbf{M}_j^\mathbf{R} & \text{(DDM v3)}\end{cases},\\ \mathbf{t}_i &= \sum_{j=1}^{m}\begin{bmatrix}\mathbf{M}_j^\mathbf{R} & \mathbf{M}_j^\mathbf{t}\end{bmatrix}\begin{bmatrix}\chi_{ij}\\ \omega_{ij}\end{bmatrix}-\mathbf{R}_i\sum_{j=1}^{m}\chi_{ij}\end{aligned}$$

The benefit of directly blending the rotation is that it only requires storing a single weight ψ_{ij} instead of 6 numbers (in the top left corner of the symmetric matrix Ψ_{ij}). However, this representation cannot propagate changes from bone translations to the local skin rotation. This limitation is illustrated in the first and second pose (S shapes) of DDM **v2** and **v3** in Fig. 5. In these poses, we only lift, i.e. translate, one joint. For a skeleton with hierarchical bones, this manipulation is typically not allowed. In the third pose (C shape), DDM **v3** produces a deformation artifact due to the linear interpolation of rotation matrices.

For DDM **v3**, normalization with determinant $\det\left(\sum_{j=1}^{m}\psi_{ij}\mathbf{M}_j^\mathbf{R}\right)$ can be avoided by normalizing the transformed difference vector $\mathbf{R}_i\mathbf{d}_i$ in Eq. (4). By construction, Eqs. (2b), (6b) and (7) result in $\tilde{\mathbf{u}}_i=\mathbf{p}_i$ and $\tilde{\mathbf{v}}_i=\mathbf{q}_i=\sum_{j=1}^{m}\begin{bmatrix}\mathbf{M}_j^\mathbf{R} & \mathbf{M}_j^\mathbf{t}\end{bmatrix}\begin{bmatrix}\chi_{ij}\\ \omega_{ij}\end{bmatrix}$. Substituting into Eq. (4) yields a direct calculation of the deformed vertex position:

$$\begin{aligned}\mathbf{x}_i &= \sum_{j=1}^{m}\begin{bmatrix}\mathbf{M}_j^\mathbf{R} & \mathbf{M}_j^\mathbf{t}\end{bmatrix}\begin{bmatrix}\chi_{ij}\\ \omega_{ij}\end{bmatrix}+\frac{\|\mathbf{d}_i\|_2}{\|\mathbf{R}_i\mathbf{d}_i\|_2}\mathbf{R}_i\mathbf{d}_i,\\ \text{where: } \mathbf{R}_i &= \sum_{j=1}^{m}\psi_{ij}\mathbf{M}_j^\mathbf{R},\quad \mathbf{d}_i=\mathbf{u}_i-\mathbf{p}_i\end{aligned}$$

*3.5.4 Variant **v4**.* Similarly, ω_{ij}, the [4, 4]-element of matrix Ω_{ij}, is the homogeneous part that corresponds to the contribution from each bone to the translation. We can directly blend the translation and compute the transformation for vertex i:

$$\begin{aligned}\mathbf{R}_i &= \text{rotationMatrix}\left(\sum_{j=1}^{m}\psi_{ij}\mathbf{M}_j^\mathbf{q}\right),\\ \mathbf{t}_i &= \sum_{j=1}^{m}\omega_{ij}\begin{bmatrix}\mathbf{M}_j^\mathbf{R} & \mathbf{M}_j^\mathbf{t}\end{bmatrix}\begin{bmatrix}\mathbf{p}_i\\ 1\end{bmatrix}-\mathbf{R}_i\mathbf{p}_i,\\ \text{where: } \mathbf{p}_i &= \sum_{j=1}^{m}\chi_{ij} \text{ is cached}\end{aligned}$$

Note that if $\psi_{ij}=\omega_{ij}$, i.e. $\lambda=\kappa$ or $\alpha=0$, **v4** is equivalent to skinning with optimized centers of rotation (CoR) [Le and Hodgins 2016], where $\mathbf{p}_i$ is the center of rotation of vertex i.

*3.5.5 Variant **v5**.* ω_{ij} can be directly used as a single skinning weight for LBS or DQS. The practical application of this variant is replacing the skinning weight solver (bake skinning) using the original DM as the training data[5], which produces more robust results with instant feedback.

[5] https://youtu.be/5i-gxtXj1Ww

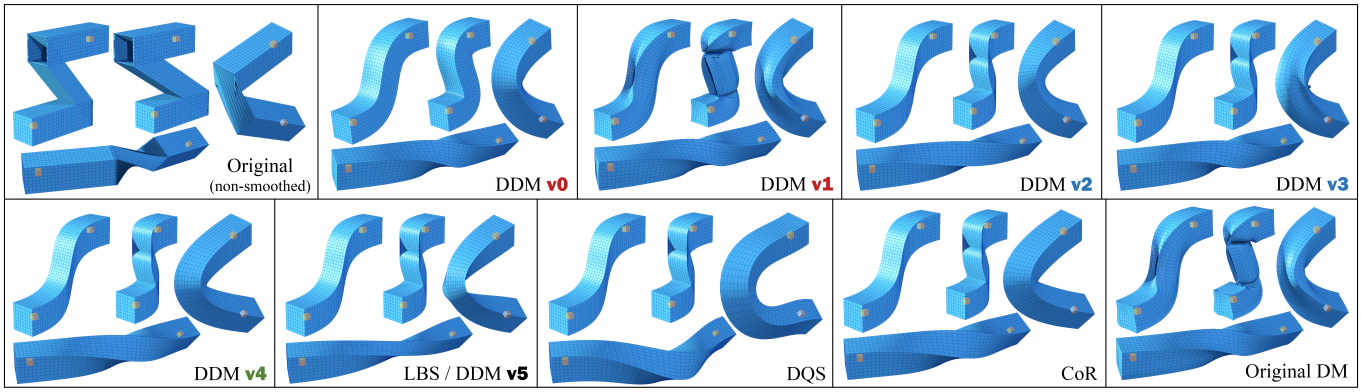


Fig. 5. Different variants of our DDM. The bar's deformation is controlled by two disconnected bones (yellow). Variants with higher number use less resources and produce more deformation artifacts. Run time deformations of some DDM variants are equivalent to previous skinning models: DDM **v1** is equivalent to the original DM, DDM **v4** is equivalent to CoR, and DDM **v5** is equivalent to LBS.

4 RESULTS AND COMPARISONS

This section presents our results and compares them with the results from four other methods: *Linear Blend Skinning* (LBS) [Magnenat-Thalmann et al. 1988], *Dual Quaternion Skinning* (DQS) [Kavan et al. 2008], Skinning with Optimized *Centers of Rotation* (CoR) [Le and Hodgins 2016], and *Delta Mush* (DM) [Mancewicz et al. 2014]. We acquired two models with skinning weights from Mixamo[6] and subdivided them using Maya to further distinguish the quality of different methods on high resolution models. We used two sets of skinning weights for the first three methods compared: the subdivided skinning weights from Mixamo (MW), and Bounded Biharmonic Weights (BBW) [Jacobson et al. 2012b]. DM and our *Direct Delta Mush* (DDM) both use a rigid bind, in which each vertex is associated to only the single bone with the largest skinning weights from Mixamo's models.

Table 2. The test models, parameters, and pre-computation time of our method. Test models are shown at their rest pose with color coded smoothing weights. n denotes the number of vertices, m denotes the number of bones. Per-vertex smoothing weights $\mathbf{D}_\lambda$ and $\mathbf{D}_\kappa$ are set proportional to the distance from the vertex to the bone to approximately represent the radius of each body part. Per-vertex blending weights $\mathbf{D}_\alpha$ of vertices on the hip area are set to lower values than on other vertices.

Pumpkin Hulk	**Brute**
$\mathbf{D}_\lambda = \mathbf{D}_\kappa$ $\mathbf{D}_\alpha$ 1 0	$\mathbf{D}_\lambda = \mathbf{D}_\kappa$ $\mathbf{D}_\alpha$ 1 0
$n = 20604$, $m = 65$	$n = 55886$, $m = 67$
Global parameters: $p = 20$, $\lambda = 20$, $\kappa = 1$, $\alpha = 0.9$	
precomputation time = 2.2s	precomputation time = 9.7s

[6]https://www.mixamo.com

Table 2 shows statistics of the models, our parameters and pre-computation times. Pre-computation times are on the order of seconds for our C++, CPU-only implementation using the sparse LU solver provided by Eigen[7]. The Laplacian matrix calculation and matrix factorization are done in serial, while each iteration of the implicit step is solved in parallel. Better precomputation performance can be achieved by more powerful solvers or careful GPU implementation.

The input models with disconnected geometry pieces were handled by adding edges that connect nearby vertices with consideration to the binding bones (Fig. 6).

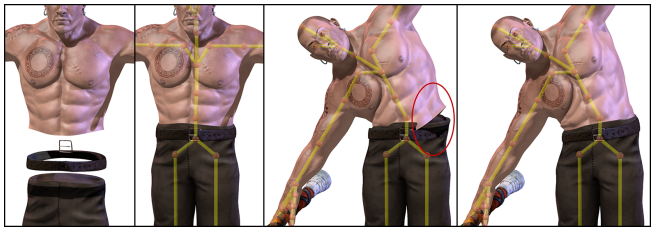


Fig. 6. Our trivial solution to handle models with multiple geometry parts. From left to right: (1) visualization of multiple parts of our character model, (2) the model at rest pose, (3) applying per-part mesh Laplacians breaks the deformation at boundaries of parts, (4) our fix by adding entries to the Laplacian matrix connecting close vertices associated to the same bone.

Table 3 shows a simple performance benchmark on different DDM variants. Note that the storage memory layouts of **v2**, **v3** and **v4** (Table 1) pose a major challenge for effective optimization because the computation includes mixed size matrices, vectors, and scalars. The compiler and the numerical library (Eigen) might not be able to take advantage of vectorization and parallelism such as SIMD instructions in these cases, so the presented results may include significant overhead that would need to be investigated with low-level programming. In contrast, the simple 4×4 matrix blocks in the storage layout of **v0** and **v1** can take full advantage

[7]http://eigen.tuxfamily.org

of special implementations for small size matrices. We also believe that the benchmark on *Brute* model is more reliable because the size of this model is larger, i.e. 55K vertices versus the 20K vertices of the *Pumpkin Hulk* model. The relative compute times shown in Table 3 are surprisingly lightweight compared to vanilla LBS skinning. This could be explained by the fact that on modern architectures (both CPU and GPU) compute times are dominated by memory access rather than FLOPS [Seo et al. 2011].

Table 3. Runtime performance comparison between variants. Numbers in this table show the average time to deform one vertex. Time is measured in microseconds on a single-threaded CPU. Note that **v5** is equivalent to LBS.

	v0	**v1**	**v2**	**v3**	**v4**	**v5**
Pumpkin Hulk	5.5	4.7	7.1	7.3	5.9	3.3
Brute	3.2	2.3	2.9	3.2	2.6	1.6

4.0.1 Local Control. Using per-vertex smoothing weights $\mathbf{D}_\lambda$ and $\mathbf{D}_\kappa$ allows handling parts with different scales in the same model, for example small fingers versus the larger arms or legs, as shown in Fig. 7. In general we expect that body parts with larger radii should deform more smoothly due to the larger amount of flesh and tissue. This principle is used to automatically calculate $\mathbf{D}_\lambda$ and $\mathbf{D}_\kappa$ as shown in Table 2. Fine tuning of the weights by skilled artists could further improve the skinning deformation quality, however all the results shown in the paper required no manual weight tuning.

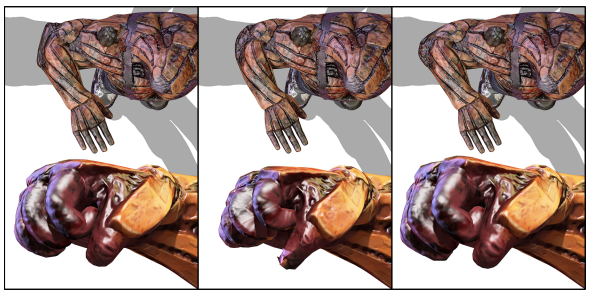


Fig. 7. Local control for different body part sizes. Left: DDM with five iterations and a global step size of 0.5 did not produce smooth deformation on the elbow. Middle: DDM with 100 iterations and a global step size of 0.5 produced good elbow deformation but it over-smoothed the finger deformations. Right: DDM with per-vertex weights (with parameters showed in Table 2) produces pleasing deformation for both elbow and fingers.

4.0.2 Independent Rotation and Translation Smoothness Control. Fig. 8 shows realistic elbow deformations produced by our DDM. Our extension (§3.3) allows simultaneously combining high rotation smoothness ($p\lambda$) with low translation smoothness ($p\kappa$). By changing the global parameters $< \lambda, \kappa, \alpha >$, we can quickly explore different deformation behavior and pick the most suitable set of parameters. Note that this is a much smaller space of parameters to explore than in traditional skinning algorithms, which require painting and adjustment of per-vertex skinning weights to obtain reasonable quality results. Thanks to the *independence* of rotation and translation smoothness, we do not need to try combinations of them. Notice how DDM keeps the same twist deformation while fixing $p\lambda$ and changing $p\kappa$ or α. More importantly, this independence removes the need to sacrifice good bending deformation in order to improve twisting deformation, as shown with other three skinning methods.

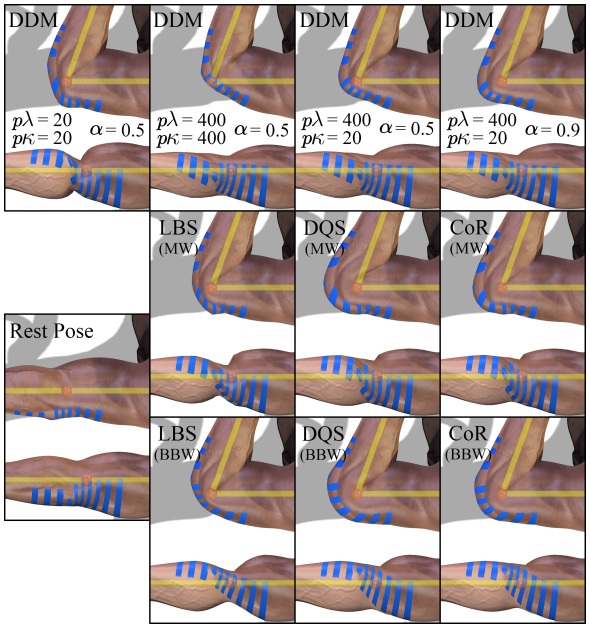


Fig. 8. Our DDM with different configurations (top row): using higher rotation smoothing step size λ produces smoother deformation when twisting the joint, while using lower translation smoothing step size κ produces more rigid deformation when bending the joint, similar to other skinning methods. Adjusting the blending weight α results in simultaneous rigid bend and smooth twist and mimics the sliding and stretching of skin on top of a knuckle. Notice how DDM with $\alpha = 0.5$ or $\alpha = 0.9$ produces more uniform skin stretching. Because our translation blending (§3.3) does not affect rotation, the smoothing effect on twisting does not change with different translation smoothing step size or blending weight (last three figures on the top row). Comparisons with other methods are shown in the bottom row. Without the independent control provided by DDM, other methods have to compromise between bending deformation quality or twisting deformation quality, typically resulting in suboptimal quality for both.

4.0.3 Hip Joint Deformation. Hip region is extremely difficult to handle using only geometric skinning, since unnatural "clefts" or creases extending into the torso commonly occur (Fig. 10). As a result corrective shapes have traditionally been added to fix this problem. Because the special negative bulging of DM is very similar to desired hip deformation (§3.3), we blend more of this effect in the hip region, or equivalently, blend less of translation, i.e. lowering $\mathbf{D}_\alpha$ at the hip region as shown in Table 2. For a quick setup, we implemented an interface that allows setting a blending value at each joint which is then automatically propagated to associated vertices. The results shown in Fig. 10 clearly show that our blending has fixed the cleft problem.

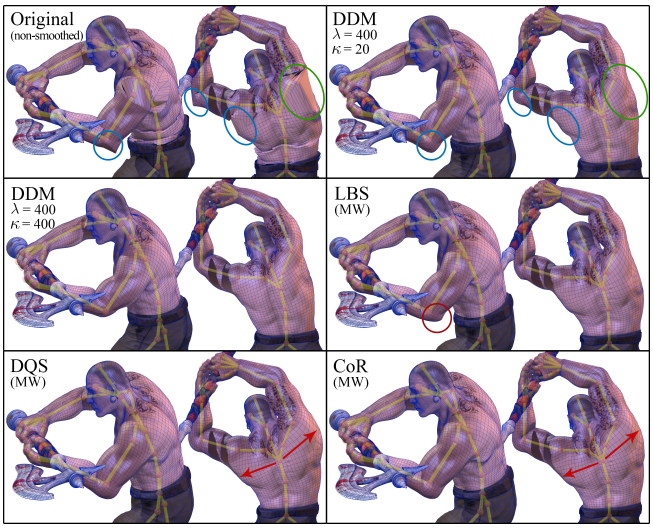


Fig. 9. Further comparison of DDM and other methods on the upper body of a detailed character. DDM is the first direct skinning method that can avoid the indicated bulging artifacts. Also see the accompanying video.

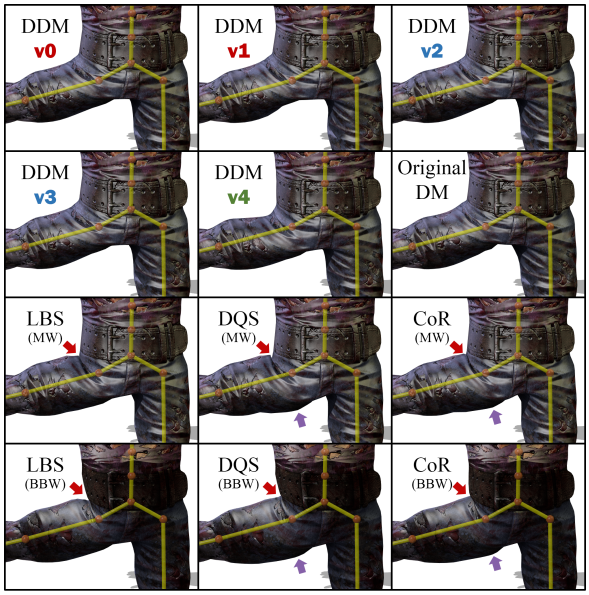


Fig. 10. A comparison of deformation at the hip joint: DM-based methods, including original DM and four variants of our DDM (**v0**, **v1**, **v2**, and **v3**), produce no clefts or fold-over due to the negative bulging. The clefts of other methods are indicated by red arrows. Notice that DDM **v4** has only one single weight for translation so it "cheats" by expanding the inner thigh (purple arrow) as happens in quaternion-based methods (DQS and CoR). The bends of DQS and CoR are more extreme so they create budges (purple arrows). The original DM is setup with 100 iterations and step size of 0.5.

4.0.4 Surface Detail Preservation. Detail preservation is important for preserving the look and feel of the character. This generally requires locally rigid transformation in order to minimize shearing and differential scaling. DQS and CoR achieve this by performing interpolation in quaternion space, thereby keeping transformation in SO(3), but this can result in bulging artifacts (Fig. 9). This is easiest to notice in the chest area while raising the arm and clavicle, because the arm/shoulder rotation carries over to chest, causing outward motion. Because DM is bulge-free, this property can be easily incorporated with DDM by appropriate parameter choices.

4.0.5 Volume Preservation. In general terms, the volume preservation of DDM is better than LBS, but not as good as DQS. The situation can be analyzed by considering the separate effects of the "delta" and "mush" components of DM and DDM. The deformation of the deltas applies locally rigid transformation, thus the mesh details do not lose volume. On the other hand, the mush deformation is linear, and so volume loss can result. However, consider the effect of the Laplacian smoothing that shrinks the mesh to the mush. With a large shrinkage, i.e. high smoothness, the mush may already have low volume in the rest pose, i.e. a rod-like geometry, in which case the volume cannot reduce much further under deformation (Fig. 11). Volume will be approximately preserved in this case. This behavior is very similar to the shrinkage from the mesh to the centers of rotation that reduces the volume loss [Le and Hodgins 2016]. In more critical cases, volume loss can be compensated using additional joints.

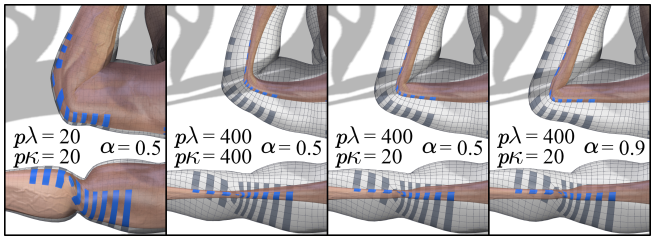


Fig. 11. Volume preservation behavior of the model (gray wireframed) depends on the volume of the mush (color shaded) at the rest pose. Leftmost: with low smoothness $p\lambda$, the mush does not shrink much, but some of this remaining volume can be lost with deformation. Middle and right: with high smoothness $p\lambda$, the mush shrinks to a rod-like shape with near zero volume and this cannot be further reduced by deformation. Bending deformation can be controlled by changing the translation smoothness $p\kappa$ and its blending weight α.

4.0.6 Skin Sliding. Consider the arm and upper back as shown in Fig. 12. Raising the arm above the shoulder should pull the skin on the back upward, causing a small upward movement extending even to the lower mid back[8] In classical skinning approaches however, such long-range interaction has to be limited to avoid the bulging artifact – the large rotational motion of a vertex about a distant joint is exactly the cause of this artifact. As seen in Figs. 9 and 12, DDM can simulate long-range skin sliding without bulging artifacts.

[8]This can be verified by pressing (for example) your right hand on your left mid-back and then raising your left arm straight above your head.

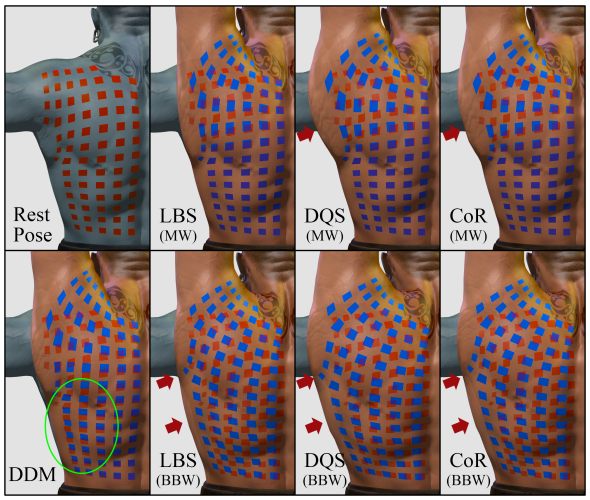


Fig. 12. DDM can produce a skin sliding effect on the mid-lower back without the accompanying bulges caused by long-range weights in other methods.

5 DISCUSSION AND CONCLUSION

This paper introduced *Direct Delta Mush*, the first geometric skinning algorithm that simultaneously provides reasonable quality and easy authoring while having computational costs in the same ballpark as existing geometric skinning methods. Our algorithm provides the same simplicity and quality as Delta Mush, an algorithm widely used in industry, and in particular produces reasonable results with virtually no manual "rigging" effort. However, DDM introduces a direct calculation analogous to that in traditional skinning algorithms rather than the iterative computation required in DM. The computation and storage requirements are similar to that in multi-weight geometric skinning approaches, plus an additional 3×3 SVD or matrix inverse.

The extended DDM algorithm further provides a general skinning framework that encompasses several previous algorithms as special cases. An explicit separation of rotation and translation allows simple skin-sliding effects to be produced, something not possible in previous geometric skinning approaches.

DDM shares the common limitations of most geometric skinning methods. These include the inability to handle self-collision, the lack of secondary effects, and insufficient volume preservation. Though the weights in DDM are sparse, there are somewhat more weights than in classic methods such as LBS. In practice, we found factors of between 2-10x more weights on the models we used. This relatively modest increase has been more than compensated by the massive improvements in computing power since the times when geometric skinning methods were first introduced. An additional limitation is the need for a short precomputation step, typically on the order of several seconds.

While DDM is attractive in terms of both versatility and efficiency, it has not been incorporated in real game engines. We leave this important evaluation for future work. In principle this integration may not pose major challenges, since DM is well known and the run-time computational form of DDM has similarities to LBS and DQS skinning. On the other hand, real-world game engines are extremely complex (consisting of millions of lines of code) and in some cases historical assumptions can appear in many places in the whole animation pipeline.[9]

REFERENCES

Marc Alexa. 2002. Linear Combination of Transformations. *ACM Trans. Graph.* 21, 3 (July 2002), 380–387. https://doi.org/10.1145/566654.566592

Dragomir Anguelov, Praveen Srinivasan, Daphne Koller, Sebastian Thrun, Jim Rodgers, and James Davis. 2005. SCAPE: Shape Completion and Animation of People. *ACM Trans. Graph.* 24, 3 (July 2005), 408–416. https://doi.org/10.1145/1073204.1073207

Ilya Baran and Jovan Popović. 2007. Automatic Rigging and Animation of 3D Characters. *ACM Trans. Graph.* 26, 3, Article 72 (July 2007).

Sofien Bouaziz, Sebastian Martin, Tiantian Liu, Ladislav Kavan, and Mark Pauly. 2014. Projective Dynamics: Fusing Constraint Projections for Fast Simulation. *ACM Trans. Graph.* 33, 4, Article 154 (July 2014), 11 pages. https://doi.org/10.1145/2601097.2601116

Christopher Brandt, Elmar Eisemann, and Klaus Hildebrandt. 2018. Hyper-reduced Projective Dynamics. *ACM Trans. Graph.* 37, 4, Article 80 (July 2018), 13 pages. https://doi.org/10.1145/3197517.3201387

Mathieu Desbrun, Mark Meyer, Peter Schröder, and Alan H. Barr. 1999. Implicit Fairing of Irregular Meshes Using Diffusion and Curvature Flow. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '99)*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 317–324. https://doi.org/10.1145/311535.311576

Dimitar Dinev, Tiantian Liu, Jing Li, Bernhard Thomaszewski, and Ladislav Kavan. 2018. FEPR: Fast Energy Projection for Real-time Simulation of Deformable Objects. *ACM Trans. Graph.* 37, 4, Article 79 (July 2018), 12 pages. https://doi.org/10.1145/3197517.3201277

Richard Everson. 1997. Orthogonal, but not Orthonormal, Procrustes Problems. In *Advances in Computational Mathematics*.

Wei-Wen Feng, Byung-Uck Kim, and Yizhou Yu. 2008. Real-time Data Driven Deformation Using Kernel Canonical Correlation Analysis. *ACM Trans. Graph.* 27, 3, Article 91 (Aug. 2008), 9 pages. https://doi.org/10.1145/1360612.1360690

Lin Gao, Yu-Kun Lai, Dun Liang, Shu-Yu Chen, and Shihong Xia. 2016. Efficient and Flexible Deformation Representation for Data-Driven Surface Modeling. *ACM Trans. Graph.* 35, 5, Article 158 (July 2016), 17 pages. https://doi.org/10.1145/2908736

Fabian Hahn, Sebastian Martin, Bernhard Thomaszewski, Robert Sumner, Stelian Coros, and Markus Gross. 2012. Rig-space Physics. *ACM Trans. Graph.* 31, 4, Article 72 (July 2012), 8 pages. https://doi.org/10.1145/2185520.2185568

Fabian Hahn, Bernhard Thomaszewski, Stelian Coros, Robert W. Sumner, and Markus Gross. 2013. Efficient Simulation of Secondary Motion in Rig-space. In *Proceedings of the 12th ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. 165–171. https://doi.org/10.1145/2485895.2485918

Alexandru-Eugen Ichim, Petr Kadleček, Ladislav Kavan, and Mark Pauly. 2017. Phace: Physics-based Face Modeling and Animation. *ACM Trans. Graph.* 36, 4, Article 153 (July 2017), 14 pages. https://doi.org/10.1145/3072959.3073664

Alec Jacobson, Ilya Baran, Ladislav Kavan, Jovan Popović, and Olga Sorkine. 2012a. Fast Automatic Skinning Transformations. *ACM Trans. Graph.* 31, 4, Article 77 (July 2012), 10 pages. https://doi.org/10.1145/2185520.2185573

Alec Jacobson, Ilya Baran, Jovan Popović, and Olga Sorkine. 2011. Bounded Biharmonic Weights for Real-time Deformation. *ACM Trans. Graph.* 30, 4, Article 78 (July 2011), 8 pages. https://doi.org/10.1145/2010324.1964973

Alec Jacobson and Olga Sorkine. 2011. Stretchable and Twistable Bones for Skeletal Shape Deformation. *ACM Trans. Graph.* 30, 6, Article 165 (Dec. 2011), 8 pages. https://doi.org/10.1145/2070781.2024199

Alec Jacobson, Tino Weinkauf, and Olga Sorkine. 2012b. Smooth Shape-Aware Functions with Controlled Extrema. *Comput. Graph. Forum* 31, 5 (Aug. 2012), 1577–1586. https://doi.org/10.1111/j.1467-8659.2012.03163.x

Doug L. James and Christopher D. Twigg. 2005. Skinning Mesh Animations. *ACM Trans. Graph.* 24, 3 (July 2005), 399–407. https://doi.org/10.1145/1073204.1073206

Ben Jones, Nils Thuerey, Tamar Shinar, and Adam W. Bargteil. 2016. Example-based Plastic Deformation of Rigid Bodies. *ACM Trans. Graph.* 35, 4, Article 34 (July 2016), 11 pages. https://doi.org/10.1145/2897824.2925979

Pushkar Joshi, Mark Meyer, Tony DeRose, Brian Green, and Tom Sanocki. 2007. Harmonic Coordinates for Character Articulation. *ACM Trans. Graph.* 26, 3, Article 71 (July 2007). https://doi.org/10.1145/1276377.1276466

Tao Ju, Scott Schaefer, and Joe Warren. 2005. Mean Value Coordinates for Closed Triangular Meshes. *ACM Trans. Graph.* 24, 3 (July 2005), 561–566. https://doi.org/10.1145/1073204.1073229

[9]For example, the the number of bone influences (weights) per vertex is often limited to 4 or 8.

W. Kabsch. 1978. A discussion of the solution for the best rotation to relate two sets of vectors. *Acta Crystallographica Section A* 34 (1978), 827–828. https://doi.org/10.1107/S0567739478001680

Petr Kadleček, Alexandru-Eugen Ichim, Tiantian Liu, Jaroslav Křivánek, and Ladislav Kavan. 2016. Reconstructing Personalized Anatomical Models for Physics-based Body Animation. *ACM Trans. Graph.* 35, 6, Article 213 (Nov. 2016), 13 pages. https://doi.org/10.1145/2980179.2982438

Ladislav Kavan, Steven Collins, Jiří Žára, and Carol O'Sullivan. 2008. Geometric Skinning with Approximate Dual Quaternion Blending. *ACM Trans. Graph.* 27, 4, Article 105 (Nov. 2008), 23 pages. https://doi.org/10.1145/1409625.1409627

Ladislav Kavan and Olga Sorkine. 2012. Elasticity-inspired Deformers for Character Articulation. *ACM Trans. Graph.* 31, 6, Article 196 (Nov. 2012), 8 pages. https://doi.org/10.1145/2366145.2366215

Ladislav Kavan and Jiří Žára. 2005. Spherical Blend Skinning: A Real-time Deformation of Articulated Models. In *Proceedings of the 2005 Symposium on Interactive 3D Graphics and Games (I3D '05)*. ACM, 9–16. https://doi.org/10.1145/1053427.1053429

Paul G. Kry, Doug L. James, and Dinesh K. Pai. 2002. EigenSkin: Real Time Large Deformation Character Skinning in Hardware. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA '02)*. ACM, New York, NY, USA, 153–159. https://doi.org/10.1145/545261.545286

Binh Huy Le and Zhigang Deng. 2012. Smooth Skinning Decomposition with Rigid Bones. *ACM Trans. Graph.* 31, 6, Article 199 (Nov. 2012), 10 pages. https://doi.org/10.1145/2366145.2366218

Binh Huy Le and Zhigang Deng. 2014. Robust and Accurate Skeletal Rigging from Mesh Sequences. *ACM Trans. Graph.* 33, 4, Article 84 (July 2014), 10 pages. https://doi.org/10.1145/2601097.2601161

Binh Huy Le and Jessica K. Hodgins. 2016. Real-time Skeletal Skinning with Optimized Centers of Rotation. *ACM Trans. Graph.* 35, 4, Article 37 (July 2016), 10 pages. https://doi.org/10.1145/2897824.2925959

J. P. Lewis, Matt Cordner, and Nickson Fong. 2000. 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*. 165–172. https://doi.org/10.1145/344779.344862

Duo Li, Shinjiro Sueda, Debanga R. Neog, and Dinesh K. Pai. 2013. Thin Skin Elastodynamics. *ACM Trans. Graph.* 32, 4, Article 49 (July 2013), 10 pages. https://doi.org/10.1145/2461912.2462008

Yaron Lipman, David Levin, and Daniel Cohen-Or. 2008. Green Coordinates. *ACM Trans. Graph.* 27, 3, Article 78 (Aug. 2008), 10 pages. https://doi.org/10.1145/1360612.1360677

Libin Liu, KangKang Yin, Bin Wang, and Baining Guo. 2013. Simulation and Control of Skeleton-driven Soft Body Characters. *ACM Trans. Graph.* 32, 6, Article 215 (Nov. 2013), 8 pages. https://doi.org/10.1145/2508363.2508427

Tiantian Liu, Sofien Bouaziz, and Ladislav Kavan. 2017. Quasi-Newton Methods for Real-Time Simulation of Hyperelastic Materials. *ACM Trans. Graph.* 36, 3, Article 116a (May 2017). https://doi.org/10.1145/2990496

Matthew Loper, Naureen Mahmood, and Michael J. Black. 2014. MoSh: Motion and Shape Capture from Sparse Markers. *ACM Trans. Graph.* 33, 6, Article 220 (Nov. 2014), 13 pages. https://doi.org/10.1145/2661229.2661273

Matthew Loper, Naureen Mahmood, Javier Romero, Gerard Pons-Moll, and Michael J. Black. 2015. SMPL: A Skinned Multi-person Linear Model. *ACM Trans. Graph.* 34, 6, Article 248 (Oct. 2015), 16 pages. https://doi.org/10.1145/2816795.2818013

N. Magnenat-Thalmann, F. Cordier, Hyewon Seo, and G. Papagianakis. 2004. Modeling of bodies and clothes for virtual environments. In *International Conference on Cyberworlds 2004.* 201–208.

N. Magnenat-Thalmann, R. Laperrière, and D. Thalmann. 1988. Joint-dependent Local Deformations for Hand Animation and Object Grasping. In *Proceedings on Graphics Interface '88*. Canadian Information Processing Society, Toronto, Ont., Canada, Canada, 26–33. http://dl.acm.org/citation.cfm?id=102313.102317

Joe Mancewicz, Matt L. Derksen, Hans Rijpkema, and Cyrus A. Wilson. 2014. Delta Mush: Smoothing Deformations While Preserving Detail. In *Proceedings of the Fourth Symposium on Digital Production (DigiPro '14)*. ACM, New York, NY, USA, 7–11. https://doi.org/10.1145/2633374.2633376

Aleka McAdams, Yongning Zhu, Andrew Selle, Mark Empey, Rasmus Tamstorf, Joseph Teran, and Eftychios Sifakis. 2011. Efficient Elasticity for Character Skinning with Contact and Collisions. *ACM Trans. Graph.* 30, 4, Article 37 (July 2011), 12 pages. https://doi.org/10.1145/2010324.1964932

Bruce Merry, Patrick Marais, and James Gain. 2006. Animation Space: A Truly Linear Framework for Character Animation. *ACM Trans. Graph.* 25, 4 (Oct. 2006), 1400–1423. https://doi.org/10.1145/1183287.1183294

Tomohiko Mukai. 2015. Building Helper Bone Rigs from Examples. In *Proceedings of the 19th ACM Symposium on Interactive 3D Graphics and Games*. 77–84. https://doi.org/10.1145/2699276.2699278

Tomohiko Mukai and Shigeru Kuriyama. 2016. Efficient Dynamic Skinning with Low-rank Helper Bone Controllers. *ACM Trans. Graph.* 35, 4, Article 36 (July 2016), 11 pages. https://doi.org/10.1145/2897824.2925905

Sang Il Park and Jessica K. Hodgins. 2008. Data-driven Modeling of Skin and Muscle Deformation. *ACM Trans. Graph.* 27, 3, Article 96 (Aug. 2008), 6 pages. https://doi.org/10.1145/1360612.1360695

Olivier Rémillard and Paul G. Kry. 2013. Embedded Thin Shells for Wrinkle Simulation. *ACM Trans. Graph.* 32, 4, Article 50 (July 2013), 8 pages. https://doi.org/10.1145/2461912.2462018

Shunsuke Saito, Zi-Ye Zhou, and Ladislav Kavan. 2015. Computational Bodybuilding: Anatomically-based Modeling of Human Bodies. *ACM Trans. Graph.* 34, 4, Article 41 (July 2015), 12 pages. https://doi.org/10.1145/2766957

Jaewoo Seo, Geoffrey Irving, J. P. Lewis, and Junyong Noh. 2011. Compression and Direct Manipulation of Complex Blendshape Models. *ACM Trans. Graph.* 30, 6, Article 164 (Dec. 2011), 10 pages. https://doi.org/10.1145/2070781.2024198

Peter-Pike J. Sloan, Charles F. Rose, III, and Michael F. Cohen. 2001. Shape by Example. In *Proceedings of the 2001 ACM Symposium on Interactive 3D Graphics*. 135–143. https://doi.org/10.1145/364338.364382

Breannan Smith, Fernando De Goes, and Theodore Kim. 2018. Stable Neo-Hookean Flesh Simulation. *ACM Trans. Graph.* 37, 2, Article 12 (March 2018), 15 pages. https://doi.org/10.1145/3180491

Olga Sorkine and Marc Alexa. 2007. As-rigid-as-possible Surface Modeling. In *Proceedings of the Fifth Eurographics Symposium on Geometry Processing*. 109–116. http://dl.acm.org/citation.cfm?id=1281991.1282006

Marco Tarini, Daniele Panozzo, and Olga Sorkine-Hornung. 2014. Accurate and Efficient Lighting for Skinned Models. *Computer Graphics Forum (proceedings of EUROGRAPHICS issue)* 33, 2 (2014), 421–428.

Rodolphe Vaillant, Loïc Barthe, Gaël Guennebaud, Marie-Paule Cani, Damien Rohmer, Brian Wyvill, Olivier Gourmel, and Mathias Paulin. 2013. Implicit Skinning: Real-time Skin Deformation with Contact Modeling. *ACM Trans. Graph.* 32, 4, Article 125 (July 2013), 12 pages. https://doi.org/10.1145/2461912.2461960

Rodolphe Vaillant, Gäel Guennebaud, Loïc Barthe, Brian Wyvill, and Marie-Paule Cani. 2014. Robust Iso-surface Tracking for Interactive Character Skinning. *ACM Trans. Graph.* 33, 6, Article 189 (Nov. 2014), 11 pages. https://doi.org/10.1145/2661229.2661264

G. Wahba. 1965. A Least Squares Estimate of Satellite Attitude. *SIAM Rev.* 7, 3 (1965), 409–409.

Robert Y. Wang, Kari Pulli, and Jovan Popović. 2007. Real-time Enveloping with Rotational Regression. *ACM Trans. Graph.* 26, 3, Article 73 (July 2007). https://doi.org/10.1145/1276377.1276468

Xiaohuan Corina Wang and Cary Phillips. 2002. Multi-weight Enveloping: Least-squares Approximation Techniques for Skin Animation. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. 129–138. https://doi.org/10.1145/545261.545283

Yu Wang, Alec Jacobson, Jernej Barbič, and Ladislav Kavan. 2015. Linear Subspace Design for Real-time Shape Deformation. *ACM Trans. Graph.* 34, 4, Article 57 (July 2015), 11 pages. https://doi.org/10.1145/2766952

Hongyi Xu and Jernej Barbič. 2016. Pose-space Subspace Dynamics. *ACM Trans. Graph.* 35, 4, Article 35 (July 2016), 14 pages. https://doi.org/10.1145/2897824.2925916

A EXPANDING AND REGROUPING

Let $\Delta_i = \text{diag}(\mathcal{B}_i) \in \mathbf{R}^{n\times n}$ be the diagonal matrix constructed from column i of $\mathcal{B}$, i.e. $(\Delta_i)_{kk} = \mathcal{B}_{ki}, \forall k$. $||\bullet||_F$ denote the Frobenius norm and $\text{tr}(\bullet)$ denote the trace of the matrix. With these conventions Eq. (5) becomes:

$$\begin{aligned} E(\Gamma) &= \left\| (\mathbf{V} - \Gamma\mathbf{U})\,\Delta_i{}^{1/2} \right\|_F^2 \\ &= \text{tr}(\mathbf{V}\Delta_i\mathbf{V}^\mathsf{T}) - 2\,\text{tr}(\mathbf{V}\Delta_i\mathbf{U}^\mathsf{T}\Gamma^\mathsf{T}) + \text{tr}(\Gamma\mathbf{U}\Delta_i\mathbf{U}^\mathsf{T}\Gamma^\mathsf{T}) \end{aligned} \tag{13}$$

As the first term $\text{tr}(\mathbf{V}\Delta_i\mathbf{V}^\mathsf{T})$ is a constant (with respect to Γ), we will only need to compute the last two terms:

- The second term $\text{tr}(\mathbf{V}\Delta_i\mathbf{U}^\mathsf{T}\Gamma^\mathsf{T})$: substituting $\mathbf{v}_k$ from the skinning equation (1) and regrouping terms by bone transformation $\mathbf{M}_j$ yields:

$$\mathbf{V}\Delta_i\mathbf{U}^\mathsf{T} = \sum_{k=1}^{n}\mathbf{v}_k\mathcal{B}_{ki}\mathbf{u}_k{}^\mathsf{T} = \sum_{k=1}^{n}\Big(\sum_{j=1}^{m}w_{kj}\mathbf{M}_j\mathbf{u}_k\Big)\mathcal{B}_{ki}\mathbf{u}_k{}^\mathsf{T}$$
$$= \sum_{j=1}^{m}\mathbf{M}_j\Big(\sum_{k=1}^{n}\mathcal{B}_{ki}w_{kj}\mathbf{u}_k\mathbf{u}_k{}^\mathsf{T}\Big) = \sum_{j=1}^{m}\mathbf{M}_j\Psi_{ij} \tag{14}$$

- The third term $\mathrm{tr}(\Gamma\mathbf{U}\Delta_i\mathbf{U}^\mathsf{T}\Gamma^\mathsf{T})$: using the affinity $\sum_{j=1}^{m}w_{kj} = 1, \forall k$ yields:

$$\mathbf{U}\Delta_i\mathbf{U}^\mathsf{T} = \sum_{k=1}^{n}\mathbf{u}_k\mathcal{B}_{ki}\mathbf{u}_k{}^\mathsf{T}\underbrace{\left(\sum_{j=1}^{m}w_{kj}\right)}_{1}$$
$$= \sum_{j=1}^{m}\left(\sum_{k=1}^{n}\mathcal{B}_{ki}w_{jk}\mathbf{u}_k\mathbf{u}_k{}^\mathsf{T}\right) = \sum_{j=1}^{m}\Psi_{ij} \tag{15}$$

Substituting Eq. (14) and Eq. (15) to Eq. (13) yields Eq. (6).

B DECOUPLING ROTATION AND TRANSLATION

Substituting Eq. (7) into Eq. (6) and expanding $\Gamma = \begin{bmatrix}\Phi & \tau\\ 0 & 1\end{bmatrix}$ yields:

$$E(\Gamma) = \mathrm{tr}\left(\begin{bmatrix}\Phi & \tau\\ 0 & 1\end{bmatrix}\begin{bmatrix}\mathbf{P}_i & \mathbf{p}_i\\ \mathbf{p}_i{}^\mathsf{T} & 1\end{bmatrix}\begin{bmatrix}\Phi & \tau\\ 0 & 1\end{bmatrix}^\mathsf{T}\right)$$
$$- 2\,\mathrm{tr}\left(\begin{bmatrix}\mathbf{Q}_i & \mathbf{q}_i\\ \mathbf{p}_i{}^\mathsf{T} & 1\end{bmatrix}\begin{bmatrix}\Phi & \tau\\ 0 & 1\end{bmatrix}^\mathsf{T}\right) + \text{const.}$$
$$= \mathrm{tr}\left(\Phi\mathbf{P}_i\Phi^\mathsf{T}\right) + 2\,\mathrm{tr}\left(\Phi\mathbf{p}_i\tau^\mathsf{T}\right) + \mathrm{tr}\left(\tau\tau^\mathsf{T}\right) + 1$$
$$- 2\left(\mathrm{tr}\left(\mathbf{Q}_i\Phi^\mathsf{T}\right) + \mathrm{tr}\left(\mathbf{q}_i\tau^\mathsf{T}\right) + 1\right) + \text{const.}$$
$$= \text{const.} - 2\,\mathrm{tr}\left(\mathbf{Q}_i\Phi^\mathsf{T}\right)$$
$$+ \mathrm{tr}\left(\Phi\mathbf{P}_i\Phi^\mathsf{T}\right) - 2\,\mathrm{tr}\left((\mathbf{q}_i - \Phi\mathbf{p}_i)\tau^\mathsf{T}\right) + \mathrm{tr}\left(\tau\tau^\mathsf{T}\right)$$

Adding and subtracting $\mathrm{tr}\left((\mathbf{q}_i - \Phi\mathbf{p}_i)(\mathbf{q}_i - \Phi\mathbf{p}_i)^\mathsf{T}\right) = \mathrm{tr}\left(\mathbf{q}_i\mathbf{q}_i{}^\mathsf{T}\right) - 2\,\mathrm{tr}\left(\mathbf{q}_i\mathbf{p}_i{}^\mathsf{T}\Phi^\mathsf{T}\right) + \mathrm{tr}\left(\Phi\mathbf{p}_i\mathbf{p}_i{}^\mathsf{T}\Phi^\mathsf{T}\right)$ yields:

$$E(\Gamma) = \text{const.} - 2\,\mathrm{tr}\left(\mathbf{Q}_i\Phi^\mathsf{T}\right) - \mathrm{tr}\left(\mathbf{q}_i\mathbf{q}_i{}^\mathsf{T}\right) + 2\,\mathrm{tr}\left(\mathbf{q}_i\mathbf{p}_i{}^\mathsf{T}\Phi^\mathsf{T}\right)$$
$$- \mathrm{tr}\left(\Phi\mathbf{p}_i\mathbf{p}_i{}^\mathsf{T}\Phi^\mathsf{T}\right) + \mathrm{tr}\left(\Phi\mathbf{P}_i\Phi^\mathsf{T}\right)$$
$$+ \mathrm{tr}\left((\mathbf{q}_i - \Phi\mathbf{p}_i)(\mathbf{q}_i - \Phi\mathbf{p}_i)^\mathsf{T}\right) - 2\,\mathrm{tr}\left((\mathbf{q}_i - \Phi\mathbf{p}_i)\tau^\mathsf{T}\right) + \mathrm{tr}\left(\tau\tau^\mathsf{T}\right)$$
$$= \text{const.} - 2\,\mathrm{tr}\left((\mathbf{Q}_i - \mathbf{q}_i\mathbf{p}_i{}^\mathsf{T})\Phi^\mathsf{T}\right) + \mathrm{tr}\left(\Phi(\mathbf{P}_i - \mathbf{p}_i\mathbf{p}_i{}^\mathsf{T})\Phi^\mathsf{T}\right)$$
$$+ \mathrm{tr}\left((\mathbf{q}_i - \Phi\mathbf{p}_i - \tau)(\mathbf{q}_i - \Phi\mathbf{p}_i - \tau)^\mathsf{T}\right)$$
$$= \mathrm{tr}\left(\Phi(\mathbf{P}_i - \mathbf{p}_i\mathbf{p}_i{}^\mathsf{T})\Phi^\mathsf{T}\right) - 2\,\mathrm{tr}\left((\mathbf{Q}_i - \mathbf{q}_i\mathbf{p}_i{}^\mathsf{T})\Phi^\mathsf{T}\right)$$
$$+ \|(\mathbf{q}_i - \Phi\mathbf{p}_i) - \tau\|_2^2 + \text{const.}$$

Substituting Eq. (8) into the above equation yields Eq. (9) as follows:

$$E(\Gamma) = \mathrm{tr}\left(\Phi\mathbf{Z}_i\Lambda_i\mathbf{Z}_i{}^\mathsf{T}\Phi^\mathsf{T}\right) - 2\,\mathrm{tr}\left((\mathbf{Q}_i - \mathbf{q}_i\mathbf{p}_i{}^\mathsf{T})\mathbf{Z}_i\Lambda_i{}^{-1/2}\Lambda_i{}^{1/2}\mathbf{Z}_i{}^\mathsf{T}\Phi^\mathsf{T}\right)$$
$$+ \|(\mathbf{q}_i - \Phi\mathbf{p}_i) - \tau\|_2^2 + \text{const.}$$
$$= \left\|\Phi\mathbf{Z}_i\Lambda_i{}^{1/2} - (\mathbf{Q}_i - \mathbf{q}_i\mathbf{p}_i{}^\mathsf{T})\mathbf{Z}_i\Lambda_i{}^{-1/2}\right\|_F^2$$
$$+ \|(\mathbf{q}_i - \Phi\mathbf{p}_i) - \tau\|_2^2 + \text{const.}$$

C PROOF: USING Ω_{ij} DOES NOT CHANGE $\mathbf{R}_i$

We denote symbols that are computed from Ω_{ij} with asterisks. Using the affinity $\sum_{j=1}^{m}w'_{ij} = 1$ and substituting Ω_{ij} in Eq. (11) for Ψ_{ij} in Eq. (7) and Eq. (10) yields:

$$\begin{bmatrix}\mathbf{P}_i^* & \mathbf{p}_i^*\\ \mathbf{p}_i^{*\mathsf{T}} & 1\end{bmatrix} = \sum_{j=1}^{m}\Omega_{ij} = (1-\alpha)\sum_{j=1}^{m}\Psi_{ij} \quad + \alpha\sum_{j=1}^{m}w'_{ij}\begin{bmatrix}\mathbf{p}_i\mathbf{p}_i{}^\mathsf{T} & \mathbf{p}_i\\ \mathbf{p}_i{}^\mathsf{T} & 1\end{bmatrix}$$
$$= (1-\alpha)\begin{bmatrix}\mathbf{P}_i & \mathbf{p}_i\\ \mathbf{p}_i{}^\mathsf{T} & 1\end{bmatrix} + \alpha\begin{bmatrix}\mathbf{p}_i\mathbf{p}_i{}^\mathsf{T} & \mathbf{p}_i\\ \mathbf{p}_i{}^\mathsf{T} & 1\end{bmatrix}$$
$$= \begin{bmatrix}(1-\alpha)\mathbf{P}_i + \alpha\mathbf{p}_i\mathbf{p}_i{}^\mathsf{T} & \mathbf{p}_i\\ \mathbf{p}_i^\mathsf{T} & 1\end{bmatrix}$$
$$\Rightarrow \mathbf{p}_i^* = p_i\ ,\ \mathbf{P}_i^* = (1-\alpha)\mathbf{P}_i + \alpha\mathbf{p}_i\mathbf{p}_i{}^\mathsf{T}$$
$$\Rightarrow \mathbf{P}_i^* - \mathbf{p}_i^*\mathbf{p}_i^{*\mathsf{T}} = (1-\alpha)\mathbf{P}_i + \alpha\mathbf{p}_i\mathbf{p}_i{}^\mathsf{T} - \mathbf{p}_i\mathbf{p}_i{}^\mathsf{T} = (1-\alpha)(\mathbf{P}_i - \mathbf{p}_i\mathbf{p}_i{}^\mathsf{T})$$
$$= \mathbf{Z}_i\left((1-\alpha)\Lambda_i\right)\mathbf{Z}_i{}^\mathsf{T}$$

Letting $\begin{bmatrix}\Xi & \zeta\\ 0 & 1\end{bmatrix} = \sum_{j=1}^{m}w'_{ij}\mathbf{M}_j$, substituting Ω_{ij} for Ψ_{ij} in Eq. (7) and Eq. (10) yields:

$$\begin{bmatrix}\mathbf{Q}_i^* & \mathbf{q}_i^*\\ \mathbf{p}_i^{*\mathsf{T}} & 1\end{bmatrix} = \sum_{j=1}^{m}\mathbf{M}_j\Omega_{ij}$$
$$= (1-\alpha)\sum_{j=1}^{m}\mathbf{M}_j\Psi_{ij} + \alpha\sum_{j=1}^{m}w'_{ij}\mathbf{M}_j\begin{bmatrix}\mathbf{p}_i\mathbf{p}_i{}^\mathsf{T} & \mathbf{p}_i\\ \mathbf{p}_i{}^\mathsf{T} & 1\end{bmatrix}$$
$$= (1-\alpha)\begin{bmatrix}\mathbf{Q}_i & \mathbf{q}_i\\ \mathbf{p}_i{}^\mathsf{T} & 1\end{bmatrix} + \alpha\begin{bmatrix}(\Xi\mathbf{p}_i + \zeta)\mathbf{p}_i{}^\mathsf{T} & \Xi\mathbf{p}_i + \zeta\\ \mathbf{p}_i{}^\mathsf{T} & 1\end{bmatrix}$$
$$= \begin{bmatrix}(1-\alpha)\mathbf{Q}_i + \alpha(\Xi\mathbf{p}_i + \zeta)\mathbf{p}_i{}^\mathsf{T} & (1-\alpha)\mathbf{q}_i + \alpha(\Xi\mathbf{p}_i + \zeta)\\ \mathbf{p}_i{}^\mathsf{T} & 1\end{bmatrix}$$
$$\Rightarrow \mathbf{Q}_i^* - \mathbf{q}_i^*\mathbf{p}_i^{*\mathsf{T}} = (1-\alpha)\mathbf{Q}_i + \alpha(\Xi\mathbf{p}_i + \zeta)\mathbf{p}_i{}^\mathsf{T}$$
$$- (1-\alpha)\mathbf{q}_i\mathbf{p}_i{}^\mathsf{T} - \alpha(\Xi\mathbf{p}_i + \zeta)\mathbf{p}_i{}^\mathsf{T}$$
$$= (1-\alpha)(\mathbf{Q}_i - \mathbf{q}_i\mathbf{p}_i{}^\mathsf{T})$$

Substituting $\mathbf{P}_i^* - \mathbf{p}_i^*\mathbf{p}_i^{*\mathsf{T}}$ and $\mathbf{Q}_i^* - \mathbf{q}_i^*\mathbf{p}_i^{*\mathsf{T}}$ in Eq. (10) yields the new objective function:

$$\mathbf{R}_i^* = \arg\min_{\Phi}\left\|(1-\alpha)^{1/2}\left(\Phi\mathbf{Z}_i\Lambda_i{}^{1/2} - (\mathbf{Q}_i - \mathbf{q}_i\mathbf{p}_i{}^\mathsf{T})\mathbf{Z}_i\Lambda_i{}^{-1/2}\right)\right\|_F^2$$

which differs from the objective function in Eq. (10a) only by the constant scale $(1-\alpha)^{1/2}$, therefore, $\mathbf{R}_i^* = \mathbf{R}_i$.