SIGGRAPH Course 2014 — Skinning: Real-time Shape Deformation

Mesh Animation Decomposition and Compression

Binh H. Le and Zhigang Deng

University of Houston

With the development of advanced computer vision and tracking techniques, deformed mesh sequences can be soundly reconstructed by markerless performance capture [De Aguiar et al. 2008b; Vlasic et al. 2008; Vlasic et al. 2009; Stoll et al. 2010], or by motion capture with dense markers [Park and Hodgins 2006; De Aguiar et al. 2007]. With the maturity of these high-quality capture techniques, researchers in computer graphics community have also explored the concept of using example poses (i.e., a sequence of deformed mesh frames) for character skinning, rigging and animation, which has become increasingly practical and useful in entertainment practice. For example, proxy bones (or called *bones* for simplicity) and corresponding skinning weights can be automatically extracted from a set of example poses [James and Twigg 2005; Kavan et al. 2010; Le and Deng 2012], and the resulting bones and skinning weights can be potentially used for many applications including mesh animation compression, hardware-accelerated skinning animation, collision detection, animation editing, and so on [James and Twigg 2005; Kavan et al. 2010]. Furthermore, different from a set of disconnected proxy bones, a skeletal rigging model can also be automatically extracted from example poses [Schaefer and Yuksel 2007; Hasler et al. 2010; Le and Deng 2014]. Since the skeleton extracted from example poses is compatible with game engines and popular animation software such as Maya and Blender, it can be directly used for various animation editing, compression, and rendering applications, which could help to substantially reduce production cost in industry practice.

In linear blend skinning (LBS) model (or cage-based deformation), the deformed position of a surface vertex is influenced by a set of bones (or control points). Often, the number of bones that affect a vertex of the target mesh varies significantly, such as from one to tens of them. Meanwhile, with the availability and affordability of high-precision 3D scanning and acquisition devices, dense 3D mesh models with thousands or even millions of vertices have being commonly used in movie special effects, video games, and other entertainment industry practices. Therefore, skinning high-resolution 3D meshes on off-the-shelf computers has become an expensive computational task, in particular, for real-time graphics applications (such as video games) due to their real-time response requirement. Fortunately, modern GPU hardware can provide an unprecedented computing capability to handle massive parallel data processing. Researchers have strived to efficiently exploit the GPU computing power to accelerate skinning animation on GPU. In particular, one intensively studied solution is to impose the sparseness constraint on the skinning weights [James and Twigg 2005; Landreneau and Schaefer 2010; Le and Deng 2013] (that is, enforce no more than a fixed number of non-zero skinning weights for any vertex), while maximally retaining the visual quality of the reduced skinning animation. This process is called *skinning weight reduction and compression*.

In the remainder of the course note, we will first describe a number of recent example-based skinning decomposition techniques that automatically extract proxy bones (flexible or rigid) and corresponding skinning weights from a set of example poses (Section 1). Then, we will further describe recent skeleton extraction algorithms that automatically extract skeletal rigging models from example poses (Section 2). Finally, we will describe a number of skinning weight reduction and compression approaches to balance the trade-off between skinning efficiency and quality (Section 3).

1 Example-based Skinning Decomposition

A number of example-based skinning decomposition algorithms have been proposed in recent years, including skinning mesh animation [James and Twigg 2005] (called **SMA** for brevity in this note), fast and efficient skinning of animated meshes [Kavan et al. 2010] (called **FSD**), learning skeleton for shape and pose [Hasler et al. 2010] (called **LSSP**), and smooth skinning decomposition with rigid bones [Le and Deng 2012] (called **SSDR**). Since besides proxy bones and skinning weights, the LSSP method can further extract skeleton from animated meshes, we leave its discussion to the follow-up Section 2. In this section, we primarily focus on the methodologies of the SMA, FSD, and SSDR algorithms.

1.1 Problem Formulation

Example-based skinning decomposition (or called skinning decomposition from animated meshes) aims to solve the inverse problem of the LBS model, that is, automatically estimating proxy bones and skinning weights from a set of example poses. Suppose we have S example poses of a n-vertices model, where the coordinate of the i-th vertex in the t-th example pose is denoted as $\mathbf{v}'_{t,i}$, and the coordinate of the i-th vertex at the rest pose is denoted as $\mathbf{v}_i$, a skinning decomposition method takes the positions of vertices, $\{\mathbf{v}'_{t,i} : t = 1 \cdots S, i = 1 \cdots n\}$, as the input, and decomposes them to the bone transformations (assuming a total of m proxy bones) and the bone-vertex weight map, i.e. $\{w_{ij}\}$, $\{\mathbf{R}_{t,j}\}$, and $\{\mathbf{T}_{t,j}\}$ in the right hand side of the following LBS formula (Eq. (1)).

$$\mathbf{v}'_{t,i} = \sum_{j=1}^{m} w_{ij}(\mathbf{R}_{t,j}\mathbf{v}_i + \mathbf{T}_{t,j}) \tag{1}$$

The example poses $\mathbf{V}$ can be considered as a $n \times S$ matrix, where each element is a 3×1 vector representing the 3D coordinate of a vertex in an example pose. The output of the skinning decomposition algorithm is the bone-vertex influence map $\mathbf{W} = \{w_{ij}\}$ and the bone transformations $\mathbf{B} = \{\mathbf{R}_{t,j}, \mathbf{T}_{t,j}\}_{j=1}^{m}$. $\mathbf{W}$ is a sparse, non-negative $n \times m$ matrix, where w_{ij} denotes the influence of the j-th bone on the i-th vertex and the sum of elements in any row equals to 1. The transformation matrix of the j-th bone in the t-th example pose is $[\mathbf{R}_{t,j}|\mathbf{T}_{t,j}]$, where $\mathbf{R}_{t,j}$ denotes the 3×3 rotation matrix and $\mathbf{T}_{t,j}$ denotes the 3×1 translation vector.

In this way, skinning decomposition can be formulated as a constrained least squares optimization problem in terms of the following example pose reconstruction error:

$$\min_{\mathbf{W},\mathbf{R},\mathbf{T}} E = \min_{\mathbf{W},\mathbf{R},\mathbf{T}} \sum_{t=1}^{S}\sum_{i=1}^{n} \left\| \mathbf{v}'_{t,i} - \sum_{j=1}^{m} w_{ij}(\mathbf{R}_{t,j}\mathbf{v}_i + \mathbf{T}_{t,j}) \right\|^2 \tag{2a}$$

$$\text{Subject to: } w_{ij} \geq 0, \forall i, j \tag{2b}$$

$$\sum_{j=1}^{m} w_{ij} = 1, \forall i \tag{2c}$$

$$|\{w_{ij}|w_{ij} \neq 0\}| \leq K, \forall i \tag{2d}$$

The objective function E in Eq. (2a) is the squared sum of the reconstruction errors for all the vertices for all the example poses. E is minimized subject to three hard constraints: the *non-negativity* constraint, Eq. (2b); the *affinity* constraint, Eq. (2c); and the *sparseness* constraint (K is often set to 4 in practice), Eq. (2d), are imposed on the bone-vertex influences.

Note that the three methods (SMA, FSD, and SSDR) have a similar problem formulation in principle (Eq. (2)), but they differ in the following main aspect: The FSD method does not impose any constraint onto the rotation part of bone transformations, $\{\mathbf{R}_{t,j}\}$, while the SSDR and SMA methods can impose one additional *orthogonal* (rigid bone) constraint (Eq. (3)) to $\{\mathbf{R}_{t,j}\}$, although they handle the orthogonal constraint differently (specifically, the orthogonal constraint is treated as a soft constraint in the SMA method, while the orthogonal constraint is handled as a hard constraint in the SSDR method).

$$\mathbf{R}_{t,j}{}^{\mathsf{T}}\mathbf{R}_{t,j} = \mathbf{I}, \det \mathbf{R}_{t,j} = 1, \forall t, j \tag{3}$$

The orthogonal constraint in Eq. (3) ensures all the matrices, $\{\mathbf{R}_{t,j}\}$, in the rotation group (i.e. the special orthogonal group $SO(3, \mathbf{R})$ in the 3D Euclidean space).

1.2 Pipeline

The general pipeline of solving the above formulated optimization problem roughly consists of the following two steps:

- *Clustering-based bone initialization*, including necessary data preprocessing. As the first step, a clustering approach is applied to example poses at triangle or vertex level to obtain an initial estimation of proxy bones. Its outcomes (the initialized proxy bones) are either further refined through a follow-up iterative optimization process [Kavan et al. 2010; Le and Deng 2012] or are directly used as an input to the next estimation step [James and Twigg 2005]. In addition, as a preprocessing step, the FSD method [Kavan et al. 2010] introduces a technique to reduce the dimension of the problem space to improve its efficiency, based on the observation that many vertex trajectories in the example poses can be well approximated as linear combinations of a set of representative vertex trajectories, and thus these vertices (and their trajectories) can be bypassed during the optimization solving process (refer to [Kavan et al. 2010] for more technical details).

- *Optimization of skinning parameters*. In both the FSD and SSDR methods, based on the initialized bones, an iterative optimization strategy is used to alternatively refine the bone transformations and skinning weights while continuously reducing the reconstruction error, E in Eq. (2). In the SMA method, the initialized bone transformations are used to determine an optimal influence set for each vertex (in order to satisfy the above sparseness constraint) and then further solve the corresponding skinning weights for each vertex. An iterative optimization process is not employed in the SMA method.

1.3 Preliminaries

In recent example-based skinning decomposition approaches [James and Twigg 2005; Hasler et al. 2010; Kavan et al. 2010; Le and Deng 2012], *block coordinate descent* and *non-negative least squares solvers* are often employed or extended to solve the formulated optimization problems. For the sake of readability and completeness, in this course note we briefly describe preliminary knowledge of block coordinate descent and non-negative least squares with affinity constraint.

1.3.1 Block Coordinate Descent

Gradient Descent, also known as *steepest descent*, is one of the most widely known optimization methods [Snyman 2005]. Let $f(\mathbf{x})$ be a multivariable function that has continuous first partial derivatives on $\mathbb{R}^n$, and its gradient $\triangledown f(\mathbf{x})$ is a n-dimensional vector. The line search direction $\mathbf{d}$ in the gradient descent method is computed as $-\triangledown f(\mathbf{x})$. If $\mathbf{d}$ satisfies $\triangledown f(\mathbf{x})^T\mathbf{d} < 0$, $f(\mathbf{x})$ can be improved with a sufficiently small step size α.

Different from the above gradient descent method, **Coordinate Descent** method takes the search direction $\mathbf{d}$ as $\mathbf{e}_i$ (or $-\mathbf{e}_i$), where $\mathbf{e}_i = [0, \cdots, 0, 1, 0, \cdots, 0]$ has 1 at its *i*-th position and 0 at other positions [Snyman 2005]. In other words, at each step $f(x_1, \cdots, x_n)$ is minimized with respect to one variable x_i while the other variables are held fixed. There are different strategies to select the coordinates:

- *Gauss-Southwell.* At each iteration the coordinate that has the largest absolute value is selected as the search direction $\mathbf{d}$.
- *Cyclic coordinate.* The coordinate variables to be optimized are $x_1, \cdots, x_n, x_1, \cdots, x_n$, etc.
- *Double sweep.* The order of searched coordinate variables is $x_1, \cdots, x_n$, and then, $x_n, \cdots, x_1$, then repeat.
- *Cyclic with permutation.* The coordinate variable order is random in each cycle.
- *Random sampling.* Pick a random coordinate variable at each iteration.

The convergence properties of the coordinate descent method [Tseng 2001] are in general poorer than the gradient descent method, it can be a suitable and useful method if the optimization for each coordinate is simple to implement. Furthermore, if the minimization with respect to one coordinate variable is much cheaper to compute, then the overall efficiency of the coordinate descent method could be still high, specifically when high correlations among coordinate variables do not exist.

Block Coordinate Descent (BCD) method, as a natural extension of the coordinate descent method, minimizes the objective function with respect to a block of coordinates at each iteration [Bertsekas 1999]. In other words, the coordinates are partitioned into a number of blocks and, at each iteration, f is minimized with respect to one of the coordinate blocks while the other coordinates are held fixed.

Assume an objective function for minimization is defined as follows:

$$\min \ f(\mathbf{x}) := f(\mathbf{x}_1, \cdots, \mathbf{x}_m), \quad \text{where } \mathbf{x}_i \in \mathfrak{X}_i, \forall i = 1, \cdots, m, \text{and } \mathfrak{X} = \mathfrak{X}_1 \times \mathfrak{X}_2 \times \cdots \times \mathfrak{X}_m \tag{4}$$

The general process of the BCD method is executed in the following way.

1. Initialization: Choose any $\mathbf{x}^0 = (\mathbf{x}_1^0, \cdots, \mathbf{x}_m^0) \in \mathfrak{X}$.
2. Iteratively optimize f until convergence: For the $(r+1)$-th iteration $(r \geq 0)$, given $\mathbf{x}^r = (\mathbf{x}_1^r, \cdots, \mathbf{x}_m^r) \in \mathfrak{X}$, choose an index $s \in \{1, \cdots, m\}$ using one of the above block coordinate selection strategies, and compute $\mathbf{x}^{r+1}$ that satisfies the following:

$$\mathbf{x}_s^{r+1} = \arg\min_{\xi \in \mathfrak{X}_s} f(\mathbf{x}_1^r, \cdots, \mathbf{x}_{s-1}^r, \xi, \mathbf{x}_{s+1}^r, \cdots, \mathbf{x}_m^r) \tag{5}$$

$$\mathbf{x}_j^{r+1} = \mathbf{x}_j^r, \quad \forall j \neq s \tag{6}$$

BCD Convergence Theorem. Let f be continuously differentiable over $\mathfrak{X} := \times_{i=1}^m \mathfrak{X}_i$. Further, assume for each block i and $\mathbf{x} \in \mathfrak{X}$, the minimum of

$$\min_{\xi \in \mathfrak{X}_i} f(\mathbf{x}_1, \cdots, \mathbf{x}_{i-1}, \xi, \mathbf{x}_{i+1}, \cdots, \mathbf{x}_m)$$

is uniquely achieved. Every limit point of the BCD sequence $\{\mathbf{x}^k\}$ is a stationary point of f. [Proof is omitted for brevity.]

Corollary. If f is also convex (besides being continuously differentiable over $\mathfrak{X}$), then every limit point of the BCD sequence $\{\mathbf{x}^k\}$ is a global minimum. The BCD method may not converge (i.e. cycle indefinitely), if the number of coordinate blocks is > 2 and f is a non-convex function.

Two-blocks BCD Convergence Theorem [Grippo and Sciandrone 2000]. Let $f(\mathbf{x}) := f(\mathbf{x}_1, \mathbf{x}_2)$, where $\mathbf{x} \in \mathfrak{X}_1 \times \mathfrak{X}_2$, be continuously differentiable, and $\mathfrak{X}_1$, $\mathfrak{X}_2$ be closed and convex. Assume that both the BCD subproblems have solutions, and that the sequence $\mathfrak{X}$ has limit points. Then, every limit point of $\mathfrak{X}$ is stationary.

1.3.2 Non-negative Least Squares with Affinity Constraint

In example-based skinning decomposition, the skinning weights $\{w_{ij}\}$ are updated by fixing other variables (e.g., the skeleton, joint positions, bone transformations, and other) to find the optimized $\{w_{ij}\}$ subject to the non-negativity and the affinity constraints (Eq. (13)). Thus, often the building block for the skinning weights update is a constrained linear least squares as follows (Eq. (7)).

$$\begin{aligned} \mathbf{W}_i{}^{\mathsf{T}} &= \arg\min_{\mathbf{x}} \|\mathbf{A}\mathbf{x} - \mathbf{b}\|^2 \\ \text{Subject to: } &\mathbf{x} \geq \mathbf{0} \\ &\mathbf{J}\mathbf{x} = 1, \mathbf{J} = (1, 1, \ldots, 1) \end{aligned} \tag{7}$$

The above problem (Eq. (7)) is a particular case of the general linear least squares with equality and inequality constraints, in which the equality constraint could be in a form of $\mathbf{C}\mathbf{x} = \mathbf{d}$ and the inequality constraint could be in a form of $\mathbf{E}\mathbf{x} > \mathbf{f}$. One widely-used solution to the general linear least squares problem is the Active Set Method (ASM) [Gill et al. 1981].

The performance of the classical ASM solver is slow since it involves many iterations to find the true active set (i.e. a subset of constraints that are exactly satisfied). Each iteration needs to solve one unconstrained linear least squares problem corresponding to the current active set and then change one constraint from the active set to the inactive set or vice versa. Starting from an estimation of the active set, the algorithm stops when the true active set is found. Based on the original ASM solver [Gill et al. 1981; Bjorck 1996; Nocedal and Wright 2000], Le and Deng [2012] make the following two modifications (the new solver is called Fast Active Set Method (FASM) in this course note):

1. The first modification is to initialize a feasible solution only at the first run: $\mathbf{x} = (1/n)\mathbf{J}^{\mathsf{T}}$ and the corresponding active set $\mathcal{W}$. Later, the FASM algorithm uses the value of $\mathbf{x}$ from the previous run as the initial value at the next run. Note that since the main block coordinate descent algorithm (Algorithm 3) updates the bone-vertex influences at every iteration and converges to the optimal solution, it is assumed that the value of $\mathbf{x}$ and the active set will not be changed significantly at the next iteration. Thus, re-using the solution at the previous step for initialization would save unnecessary computational effort. Actually, a similar idea was successfully used for aircraft control allocation [Harkegard 2002].
2. The second modification is to pre-compute the cross products $\mathcal{A} = \mathbf{A}^{\mathsf{T}}\mathbf{A}$ and $\mathcal{B} = \mathbf{A}^{\mathsf{T}}\mathbf{b}$, since the solution of the unconstrained linear least squares $\mathbf{A}\mathbf{x} = \mathbf{b}$ is the same as that of $\mathbf{A}^{\mathsf{T}}\mathbf{A}\mathbf{x} = \mathbf{A}^{\mathsf{T}}\mathbf{b}$. Thus, the least squares $\mathbf{p} = \arg\min \|\mathbf{A}(\mathbf{x} + \mathbf{p}) - \mathbf{b}\|^2$ with equality constraint $\mathbf{J}\mathbf{p} = \mathbf{0}$ and active set constraint $p_i = 0, \forall i \in \mathcal{W}$ via $\mathcal{A}$ and $\mathcal{B}$, can be solved. By doing so, each iteration on the active set needs to solve the linear least squares with the size of the unknowns $\mathbf{x}$, regardless the number of equations. Specifically, the number of unknowns equals to the number of desired bones, which is typically much smaller than the number of equations (i.e., the number of input example poses). Note that both $\mathcal{A}$ and $\mathcal{B}$ can also be used to compute the Lagrange multipliers in Eq. (8).

As described in Algorithm 1, the FASM algorithm [Le and Deng 2012] first initializes a feasible solution $\mathbf{x} = (1/n)\mathbf{J}^{\mathsf{T}}$ and its corresponding active set $\mathcal{W}$ (line 3). At each iteration (line 5 to 25), it first solves the descent direction $\mathbf{p}$ for all p_i with i-th constraint active (line 6). Here the constraint $\mathbf{J}\mathbf{p} = \mathbf{0}$ is enforced to ensure the moving of $\mathbf{x}$ in the direction of $\mathbf{p}$ would satisfy the equality constraint $\mathbf{J}(\mathbf{x} + \alpha\mathbf{p}) = 1$. If $\mathbf{x} + \mathbf{p} \geq \mathbf{0}$, we move the current solution by $\mathbf{p}$, i.e. let $\mathbf{x} \leftarrow \mathbf{x} + \mathbf{p}$ (line 8), otherwise, the current solution is moved by $\mathbf{x} \leftarrow \mathbf{x} + \alpha\mathbf{p}$ with the maximum step $\alpha > 0$ and update the active set $\mathcal{W}$ (line 17 to line 23).

In line 9 to 15, the KKT conditions, Eq. (8) [Kuhn and Tucker 1951; Bertsekas 1999; Nocedal and Wright 2000], are used to check the optimum of the current solution $\mathbf{x}$. Using these KKT conditions, the *Lagrange multiplier* λ_i for each constraint $i \notin \mathcal{W}$ can be calculated. In this equation, each unknown λ_i corresponds to the inequality constraint $x_i \geq 0$ and the unknown $\mu \in \mathbb{R}$ corresponds to the equality constraint $\mathbf{J}\mathbf{x} = 1$. Since the equality constraint $\mathbf{J}\mathbf{x} = 1$ is always true, there are at most $n - 1$ active inequality constraints. Thus, the system of equation (Eq. (8)) is never under-determined. In addition, the solution $\mathbf{x}$ always satisfies all the constraints, and Eq. (8) is always consistent.

$$\begin{gathered} \begin{bmatrix} \mathbf{J}^{\mathsf{T}} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mu \\ \lambda \end{bmatrix} = \mathbf{A}^{\mathsf{T}}(\mathbf{A}\mathbf{x} - \mathbf{b}) \\ \lambda_i = 0, \forall i \notin \mathcal{W} \end{gathered} \tag{8}$$

Handling Equality Constraints

In the above Algorithm 1, a sub-problem of linear least squares with equality constraint $\mathbf{J}\mathbf{p} = 0$ (zero-sum) (line 6) needs to be solved. This problem can be solved by a direct elimination method, which removes the constraint by transforming the problem to a lower dimension, such as one unknown is set to be $p_1 = -\sum_{i=2}^{n} p_i$ as proposed in [Mohr and Gleicher 2003]. However, this

Algorithm 1 Active Set Algorithm for Linear Least-Squares with Equality and Inequality Constraints

```
Require: $\mathbf{A} \in \mathbb{R}^{m\times n}, \mathbf{b} \in \mathbb{R}^m$
Ensure: $\mathbf{x}^* = \arg\min \|\mathbf{Ax} - \mathbf{b}\|^2$ subject to $\mathbf{x} \geq \mathbf{0}$ and $\mathbf{Jx} = 1$
 1: $\mathcal{A} \leftarrow \mathbf{A}^\mathsf{T}\mathbf{A}$
 2: $\mathcal{B} \leftarrow \mathbf{A}^\mathsf{T}\mathbf{b}$
 3: Initialize a feasible solution $\mathbf{x} = 1/n\mathbf{J}^\mathsf{T}$ if necessary
 4: Initialize active (working) set $\mathcal{W} \leftarrow \{i|x_i = 0\}$
 5: loop
 6:     Solve $\mathbf{p} = \arg\min \|\mathbf{A}(\mathbf{x}+\mathbf{p}) - \mathbf{b}\|^2$ s.t. $\mathbf{Jp} = \mathbf{0}$ and $p_i = 0, \forall i \in \mathcal{W}$ via $\mathcal{A}$ and $\mathcal{B}$ by algorithm 2
 7:     if $\mathbf{x} + \mathbf{p} \geq \mathbf{0}$ then
 8:         $\mathbf{x} \leftarrow \mathbf{x} + \mathbf{p}$
 9:         Compute the Lagrange multipliers $\lambda$ asscociated with $\mathcal{W}$ by equation (8)
10:         if $\lambda_i \geq 0, \forall i$ then
11:             return $\mathbf{x}^* \leftarrow \mathbf{x}$ is the optimal solution
12:         else
13:             $i^* \leftarrow \arg\min_{i\in\mathcal{W}} \lambda_i$
14:             $\mathcal{W} \leftarrow \mathcal{W}\backslash i$
15:         end if
16:     else
17:         $\alpha \leftarrow -\max_{i\notin\mathcal{W}} x_i/p_i$
18:         if $\alpha \leq 0$ then
19:             return $\mathbf{x}^* \leftarrow \mathbf{x}$ is the optimal solution
20:         else
21:             $x_i \leftarrow x_i + \alpha p_i, \forall i$
22:             $\mathcal{W} \leftarrow \{i|x_i = 0\}$
23:         end if
24:     end if
25: end loop
```

solution would change the cost function and thus may break the convergence of the main algorithm. Instead, the FASM algorithm [Le and Deng 2012] uses an orthogonal projection to preserve the cost function [Bjorck 1996].

Suppose the goal is to solve for $\mathbf{x}'$ in the linear least squares with h equality constraints (Eq. (9)).

$$\min_{\mathbf{x}'} \left\|\mathbf{A}'\mathbf{x}' - \mathbf{b}'\right\|^2 \text{ subject to } \mathbf{C}'\mathbf{x}' = \mathbf{d}' \qquad \mathbf{x}' \in \mathbb{R}^k, \mathbf{C}' \in \mathbb{R}^{h\times k} \tag{9}$$

Direct elimination is performed via the following *QR decomposition*:

$$\mathbf{C}'^\mathsf{T} = \mathbf{QR} = [\underbrace{\mathbf{Q}_1}_{h} \mid \underbrace{\mathbf{Q}_2}_{k-h}] \begin{bmatrix}\mathbf{R}_1 \\ \mathbf{0}\end{bmatrix} \text{ subject to } \mathbf{QQ}^\mathsf{T} = \mathbf{I}$$

Let

$$\mathbf{Q}^\mathsf{T}\mathbf{x}' = \begin{bmatrix}\mathbf{y} \\ \mathbf{z}\end{bmatrix} \Leftrightarrow \mathbf{x}' = \mathbf{Q}\begin{bmatrix}\mathbf{y} \\ \mathbf{z}\end{bmatrix}$$

We have

$$\mathbf{d}' = \mathbf{C}'\mathbf{x}' = \mathbf{R}^\mathsf{T}\mathbf{Q}^\mathsf{T}\mathbf{x}' = \mathbf{R}^\mathsf{T}\begin{bmatrix}\mathbf{y} \\ \mathbf{z}\end{bmatrix} = \mathbf{R}_1{}^\mathsf{T}\mathbf{y}$$

and

$$\mathbf{A}'\mathbf{x}' = \mathbf{A}'\mathbf{QQ}^\mathsf{T}\mathbf{x}' = \mathbf{A}'[\mathbf{Q_1}|\mathbf{Q_2}]\begin{bmatrix}\mathbf{y} \\ \mathbf{z}\end{bmatrix} = \mathbf{A}'\mathbf{Q}_1\mathbf{y} + \mathbf{A}'\mathbf{Q}_2\mathbf{z}$$

Then, the problem (Eq. (9)) becomes an unconstrained linear least squares:

$$\begin{aligned} &\min_{\mathbf{z}} \left\| \mathbf{A}_2\mathbf{z} - (\mathbf{b}' - \mathbf{A}_1\mathbf{y}) \right\|^2 \\ \text{Where: } &\mathbf{A}_1 = \mathbf{A}'\mathbf{Q}_1 \\ &\mathbf{A}_2 = \mathbf{A}'\mathbf{Q}_2 \\ &\mathbf{R}_1^{\mathsf{T}}\mathbf{y} = \mathbf{d}' \\ &\mathbf{x}' = \mathbf{Q}\begin{bmatrix}\mathbf{y}\\ \mathbf{z}\end{bmatrix} \end{aligned}$$

The solution to this unconstrained linear least squares is:

$$\mathbf{z} = (\mathbf{A}_2{}^{\mathsf{T}}\mathbf{A}_2)^{-1}\mathbf{A}_2{}^{\mathsf{T}}(\mathbf{b}' - \mathbf{A}_1\mathbf{y}) \tag{10}$$

Applying the above calculations and transformations to the particular problem of linear least squares with equality constraints in line 6 of Algorithm 1 can further simplify all the equations and achieve Algorithm 2. Here $k = n - |\mathcal{W}|$ unknowns need to be solved, and each unknown corresponds to an inactive inequality constraint. The matrix $\mathbf{A}'$ contains all columns of $\mathbf{A}$ with indexes not in $\mathcal{W}$, $\mathbf{b}' = \mathbf{b} - \mathbf{A}'\mathbf{x}$, $\mathbf{C}' = \mathbf{J}^{\{k\}} = (1, 1, \ldots, 1) \in \mathbb{R}^k$, and $\mathbf{d}' = \mathbf{0}$.

The QR decomposition of $\mathbf{C}' = \mathbf{J}^{\{k\}}$ can be pre-computed as follows:

$$\begin{gathered} \mathbf{J}^{\{k\}^{\mathsf{T}}} = \mathbf{Q}^{\{k\}}\mathbf{R}^{\{k\}} = \left[\mathbf{Q}_1^{\{k\}} | \mathbf{Q}_2^{\{k\}}\right]\mathbf{R}^{\{k\}} \\ \underbrace{\begin{bmatrix}1\\1\\1\\1\\ \vdots\\1\end{bmatrix}}_{\mathbf{J}^{\{k\}^{\mathsf{T}}}} = \left[\begin{matrix} \underbrace{\begin{matrix}\alpha\\ \alpha\\ \alpha\\ \alpha\\ \vdots\\ \alpha\end{matrix}}_{\mathbf{Q}_1^{\{k\}}} & \underbrace{\begin{matrix}\alpha & \alpha & \alpha & \cdots & \alpha\\ \delta & \gamma & \gamma & \cdots & \gamma\\ \gamma & \delta & \gamma & \cdots & \gamma\\ \gamma & \gamma & \delta & \cdots & \gamma\\ \vdots & \vdots & \vdots & \ddots & \vdots\\ \gamma & \gamma & \gamma & \cdots & \delta\end{matrix}}_{\mathbf{Q}_2^{\{k\}} \in \mathbb{R}^{k\times(k-1)}} \end{matrix}\right] \underbrace{\begin{bmatrix}\rho\\0\\0\\0\\ \vdots\\0\end{bmatrix}}_{\mathbf{R}^{\{k\}}} \\ \text{Here: } \alpha = -1/\sqrt{k}, \delta = \gamma + 1, \gamma = \frac{-1}{k + \sqrt{k}}, \rho = -\sqrt{k} \end{gathered} \tag{11}$$

In Algorithm 2, the two matrix products $\mathbf{A}_2{}^{\mathsf{T}}\mathbf{A}_2$ and $\mathbf{A}_2{}^{\mathsf{T}}(\mathbf{b}' - \mathbf{A}_1\mathbf{y})$ in Eq. (10) can be represented via the pre-calculated matrices $\mathcal{A} = \mathbf{A}^{\mathsf{T}}\mathbf{A}$ and $\mathcal{B} = \mathbf{A}^{\mathsf{T}}\mathbf{b}$. The final simplified calculations are shown in lines 1 to 7 of Algorithm 2, where $\tilde{\mathcal{A}} = \mathbf{A}_2{}^{\mathsf{T}}\mathbf{A}_2$ and $\tilde{\mathcal{B}} = \mathbf{A}_2{}^{\mathsf{T}}(\mathbf{b}' - \mathbf{A}_1\mathbf{y})$.

Algorithm 2 Linear Least-Squares with Zero-Sum Constraint

```
Require: $\mathcal{A} \in \mathbb{R}^{n\times n}, \mathcal{B} \in \mathbb{R}^n, x \in \mathbb{R}^n, \mathcal{W} \subseteq \{1 \ldots n\}$
Ensure: $\mathbf{p} = \arg\min \|\mathbf{A}(\mathbf{x} + \mathbf{p}) - \mathbf{b}\|^2$ subject to $\mathcal{A} = \mathbf{A}^{\mathsf{T}}\mathbf{A}$,
    $\mathcal{B} = \mathbf{A}^{\mathsf{T}}\mathbf{b}, \mathbf{J}\mathbf{p} = 0$ and $p_i = 0, \forall i \in \mathcal{W}$
1: $\mathcal{B}' \leftarrow \mathcal{B} - \mathcal{A}\mathbf{x}$
2: $\mathcal{A}^{\overline{\mathcal{W}}} \leftarrow$ Rows and columns of $\mathcal{A}$ with indexes not in $\mathcal{W}$
3: $\mathcal{B}^{\overline{\mathcal{W}}} \leftarrow$ Rows of $\mathcal{B}'$ with indexes not in $\mathcal{W}$
4: Compute $\mathbf{Q}_2^{\{k\}} \in \mathbb{R}^{k\times(k-1)}$ by the QR decomposition in Eq. (11), where $k = n - |\mathcal{W}|$
5: $\tilde{\mathcal{A}} \leftarrow \mathbf{Q}_2^{\{k\}^{\mathsf{T}}}\mathcal{A}^{\overline{\mathcal{W}}}\mathbf{Q}_2^{\{k\}}$
6: $\tilde{\mathcal{B}} \leftarrow \mathbf{Q}_2^{\{k\}^{\mathsf{T}}}\mathcal{B}^{\overline{\mathcal{W}}}$
7: $\tilde{\mathbf{p}} \leftarrow \tilde{\mathcal{A}}^{-1}\tilde{\mathcal{B}}$
8: $\mathbf{p}^{\overline{\mathcal{W}}} \leftarrow \mathbf{Q}_2^{\{k\}}\tilde{\mathbf{p}}$
9: return $\mathbf{p} \leftarrow$ Corresponding rows of $\mathbf{p}^{\overline{\mathcal{W}}}$, other rows are $\mathbf{0}$
```

1.4 Clustering-based Bone Initialization

The purpose of this initialization step is to roughly estimate proxy bones from example poses. A clustering approach is typically employed for this step, for example, mean shift clustering in the SMA method, multiple source region growing in the FSD method, and K-means clustering in the SSDR method. For non-iterative skinning decomposition methods (e.g., the SMA method), it is critical for this initialization step to obtain robust bone transformations that are used as the input to the rest pipeline. While for iterative optimization based methods (e.g., the FSD method and the SSDR method), since the initialized bones and weights will be iteratively refined through the follow-up optimization process that blends away the differences between different initializations at the end, as noted by Kavan and colleagues [Kavan et al. 2010].

Mean Shift Clustering. In the SMA method [James and Twigg 2005], the 3×3 rotation matrix of a triangle j in each example pose, relative to the rest pose, is first extracted via polar decomposition of its deformation gradient [Sumner and Popović 2004], and then the rotation matrices of the same triangle in all the example poses are vectorized and concatenated to a high-dimensional data point, $\mathbf{z}_j \in \mathbb{R}^{9S}$. Then, given $\{\mathbf{z}_j\}$, a fixed bandwidth mean shift algorithm [Cheng 1995] is employed to adaptively cluster $\{\mathbf{z}_j\}$ to clusters based on the following sample point estimator:

$$f(\mathbf{z}) = \frac{1}{n}\sum_{j=1}^{n} k(\|\frac{\mathbf{z}-\mathbf{z}_j}{h}\|^2), \tag{12}$$

where h is the user-specified bandwidth of a spherically symmetric kernel function, $k(\centerdot)$. The computed gradient of f($\mathbf{z}$) is used in a hill climbing process to map each data point, $\mathbf{z}_j$, to the nearest stationary point, $\bar{\mathbf{z}}_j$. Based on each discovered $\bar{\mathbf{z}}_j$, its associated core triangles can be identified to compute the corresponding bone transformation. As noted by James and Twigg [2005], the main advantages of mean shift clustering for initial bone estimation include: (1) it is robust to outliers, and (2) the number of the resulting clusters does not need to be specified explicitly, but indirectly controlled via a physically meaningful, scaling parameter, h.

Multiple Source Region Growing. This clustering method is used in the FSD method [Kavan et al. 2010] to initialize the proxy bones. It starts by selecting P seed triangles uniformly distributed on the rest pose [Kavan et al. 2007], and each cluster is assumed to be contiguous and initialized with the vertices of the seed triangle; then, for the vertices immediately adjacent to each cluster, their prediction errors are computed based on the deformation gradients of the cluster, and the vertex with the minimum error is chosen to append to the cluster. This process repeats until all the vertices are assigned to certain clusters. To this end, a bone transformation is further computed based on the vertices in the same cluster. The main benefit of this clustering method is its efficiency.

K-Means Clustering. At the initialization step of the SSDR method [Le and Deng 2012], the deformation is assumed to be as rigid as possible [Alexa et al. 2000; Igarashi et al. 2005], that is, each vertex is influenced by exactly one bone and its bone-vertex weight is exactly 1. Then, the initialization problem becomes clustering n vertices into $|B| = m$ clusters (m is specified by users), and all the vertices in one cluster follows the same rigid transformation. In this way, the initialization of proxy bones can be straightforwardly converted to a K-means clustering problem: clustering n vertices into m clusters, and then for each cluster, solving for its rigid bone transformation based on the vertices in the same cluster.

1.5 Optimization of Skinning Parameters

Once the bones are initialized, the next step of skinning decomposition is to optimize the involved skinning parameters (that is, $\{\mathbf{R}_{t,j}\}$, $\{\mathbf{T}_{t,j}\}$ and $\mathbf{W}$). In the SMA method [James and Twigg 2005], for each surface vertex, its optimal set of influence bones is identified first, and then the skinning weights are solved per vertex using constrained least squares (details can be found at Section 3.3 of this course note and [James and Twigg 2005]). By contrast, both the FSD method and the SSDR method employ iterative optimization algorithms to refine the skinning parameters, and they share similar spirit in principle. In this note, we will only detail the iterative optimization algorithm used in the SSDR method that is based on the aforementioned BCD framework.

To solve the formulated constrained least squares problem (Eq. (2)), the SSDR method [Le and Deng 2012] employs a BCD algorithm with $m + 1$ blocks where one block is the bone-vertex influences $\mathbf{W}$ and the remaining blocks are m bone transformations. After a feasible solution is initialized, for each iteration, the objective function is optimized with respect to one block while the other blocks are held fixed. For convenience, the optimization of all the bone transformations optimization is done sequentially and grouped as one big block update step. Through iterations the two blocks are alternatively updated until the objective function converges to a local minimum. Finally, the rest pose is corrected by the linear least squares solver proposed by Kavan and colleagues [Kavan et al. 2010]. The whole process is detailed in Algorithm 3, where each update step (bone-vertex weights update or bone transformations update) decreases the objective function in order to ensure the convergence of the algorithm.

Algorithm 3 Smooth Skinning Decomposition with Rigid Bones (SSDR)

```
Require: $\mathbf{V} = \{\mathbf{v}'_{t,i}\}, \mathbf{P} = \{\mathbf{v}_i\}, m, K$
Ensure: $\mathbf{W} = \{w_{ij}\}, \mathbf{B} = \{\mathbf{R}_{t,j}|\mathbf{T}_{t,j}\}_{j=1}^{m}$ subject to Eq. (2)
1: Initialize bone transformations $\mathbf{B}$
2: repeat
3:     Update bone-vertex weight map $\mathbf{W}$
4:     Update bone transformations $\mathbf{B}$
5: until Convergence OR Maximum iterations are reached
6: Correct the rest pose $\mathbf{P}$
7: return $\mathbf{W}$, $\mathbf{B}$, and $\mathbf{P}$
```

1.5.1 Skinning Weights Update

In the SSDR method, the bone-vertex weight map $\mathbf{W}$ is updated (with fixed $\mathbf{B}$) by finding the optimized $\mathbf{W}$ subject to the non-negativity constraint, Eq. (2b), the affinity constraint, Eq. (2c), and the sparseness constraint, Eq. (2d). Specifically, $\mathbf{W}$ is optimized per vertex by solving the constrained least squares for each line $\mathbf{W}_i$ as follows:

$$\begin{aligned} {\mathbf{W}_i}^{\mathsf{T}} &= \arg\min_{x} \|\mathbf{A}\mathbf{x} - \mathbf{b}\|^2 \\ \text{Subject to: } &\mathbf{x} \geq \mathbf{0} \\ &\|\mathbf{x}\|_1 = 1 \\ &\|\mathbf{x}\|_0 \leq K \end{aligned} \tag{13}$$

Instead of selecting a subset of associated bones for each vertex prior to computing its approximation error [Mohr and Gleicher 2003; James and Twigg 2005; Kavan et al. 2010], the SSDR method associates K out of m bones for each vertex by first solving the linear least squares with non-negativity constraint ($\mathbf{x} \geq \mathbf{0}$) and affinity constraint ($\|\mathbf{x}\|_1 = 1$), and then removing $m - K$ bone weights with *least effect* to the solution. Finally, the linear solver is performed one more time with the remaining K bone weights.

e_{ij}, the effect of the bone j on the solution $\mathbf{W}_i$, is calculated as the displacement caused by setting the corresponding weight w_{ij} to 0. Specifically, the contribution of the bone transformation j to the approximation of vertex i is computed as follows:

$$e_{ij} = \|w_{ij}(\mathbf{R}_{t,j}\mathbf{v}_i + \mathbf{T}_{t,j})\|^2 \tag{14}$$

To solve the linear least squares problem with non-negativity and affinity constraints, the SSDR method extends and accelerates the Lawson and Hanson's active set method (ASM) with bound constraints [Schaefer and Yuksel 2007] (details of the modified fast ASM method can be found in Section 1.3.2). In contrast with several solutions for calculating the skinning weights, the fast ASM solver in the SSDR method [Le and Deng 2012] does not treat the affinity constraint as soft constraint as in [James and Twigg 2005], and does not take exponential computational time with respect to the number of bones as in [Kavan et al. 2010].

1.5.2 Bone Transformations Update

This update step solves the optimal bone rotations $\{\mathbf{R}_{t,j}\}$ and the bone translations $\{\mathbf{T}_{t,j}\}$ over the set of example poses by fixing the bone-vertex influence map $\mathbf{W}$ and minimizing the objective function in Eq. (2a). Since the bone transformations for every pose are independent, the minimization for each pose can be solved individually. For a pose t, the problem becomes finding the set of bone transformations to associate $\{\mathbf{v}_i\}$ to the vertices $\{\mathbf{v}'_{t,i}\}$ through minimization of the following objective function:

$$E^t = \min_{\mathbf{R}_{t,j}|\mathbf{T}_{t,j}, j=1\cdots m} \sum_{i=1}^{n} \left\| \mathbf{v}'_{t,i} - \sum_{j=1}^{m} w_{ij}(\mathbf{R}_{t,j}\mathbf{v}_i + \mathbf{T}_{t,j}) \right\|^2 \tag{15}$$

The above Eq. (15) can be solved by the Levenberg-Marquardt (LM) algorithm [Marquardt 1963]. However, the LM algorithm is inefficient since it requires many iterations as well as an initial guess that is sufficiently close to the optimal solution. Extending the closed-form solution to *Absolute Orientation* problem [Horn 1987; Kabsch 1978] to multiple bone transformations by adding the bone weights for each vertex [Müller et al. 2005; Zhu and Gortler 2007; Schaefer and Yuksel 2007] cannot provide the exact solution to the original problem in Eq. (15), as illustrated in the left side of Fig. 1.

To guarantee the non-increasing of the global objective function, Eq. (2a), the SSDR method updates the bone transformations one by one instead of updating all of them at once. For an example pose t, updating the transformation of bone $\hat{j}$ while keeping the remaining $m - 1$ bones fixed, can be solved by minimizing the following equation:

$$E^t_{\hat{j}} = \sum_{i=1}^{n} \left\| \mathbf{v}'_{t,i} - \sum_{j=1, j\neq\hat{j}}^{m} w_{ij}(\mathbf{R}_{t,j}\mathbf{v}_i + \mathbf{T}_{t,j}) - w_{i\hat{j}}(\mathbf{R}_{t,\hat{j}}\mathbf{v}_i + \mathbf{T}_{t,\hat{j}}) \right\|^2$$

Let $\mathbf{q}^t_i$ be the deformation residual of vertex i in example pose t caused by the remaining $m - 1$ bones,

$$\mathbf{q}^t_i = \mathbf{v}'_{t,i} - \sum_{j=1, j\neq\hat{j}}^{m} w_{ij}(\mathbf{R}_{t,j}\mathbf{v}_i + \mathbf{T}_{t,j}) \tag{16}$$

The problem of finding the optimal transformation then becomes:

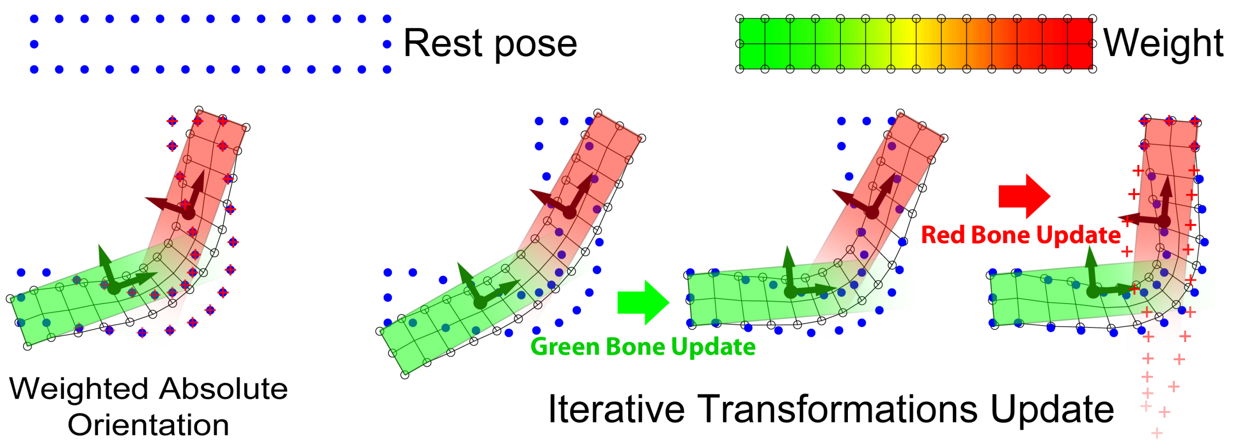
Rest pose
Weight
Weighted Absolute Orientation
Green Bone Update
Red Bone Update
Iterative Transformations Update

***Figure 1:** Comparison of the Weighted Absolute Orientation solution [Horn 1987; Kabsch 1978] on the left and the iterative bone transformations update [Le and Deng 2012] on the right. The blue dots indicate vertices in the example pose. The red plus signs (+) indicate the target positions for the red bone transformation fitting. While the Weighted Absolute Orientation only provides an approximate solution due to the deformation of the target positions in the example pose, the iterative method on the right converges to the true optimum solution by calculating the target positions as the residual of the deformation caused by the remaining bones (the green bone). Image courtesy of [Le and Deng 2012]. Copyright ©ACM 2012.*

$$\{\mathbf{R}_{t,\hat{j}}, \mathbf{T}_{t,\hat{j}}\}^* = \arg\min_{\mathbf{R}_{t,\hat{j}}, \mathbf{T}_{t,\hat{j}}} \sum_{i=1}^{n} \left\| \mathbf{q}_i^t - w_{i\hat{j}} (\mathbf{R}_{t,\hat{j}} \mathbf{v}_i + \mathbf{T}_{t,\hat{j}}) \right\|^2 \tag{17}$$
$$\text{Subject to: } {\mathbf{R}_{t,\hat{j}}}^{\mathsf{T}} \mathbf{R}_{t,\hat{j}} = \mathbf{I}, \det \mathbf{R}_{t,\hat{j}} = 1$$

The problem, Eq. (17), is similar to the Weighted Absolute Orientation problem except the difference in the scope of the weights term, i.e. the weights only affect $\mathbf{v}_i$ but not $\mathbf{q}_i^t$. Similar to the work of [Horn 1987; Kabsch 1978], the SSDR method first removes the translation $\mathbf{T}_{t,\hat{j}}$ and then solve for the optimum rotation. This is done by translating all the vertices to bring the center of rotation (CoR) to the origin for each set of vertices as computed in Eq. (18). Note that this CoR is different from the CoR computed in [Horn 1987; Kabsch 1978], since the latter is simply the centroid of the two sets of vertices.

$$\overline{\mathbf{p}}_i = \mathbf{v}_i - \mathbf{p}_* \,, \qquad \overline{\mathbf{q}}_i^t = \mathbf{q}_i^t - w_{i\hat{j}} \mathbf{q}_*^t \tag{18a}$$
$$\text{Where: } \mathbf{p}_* = \frac{\sum_{i=1}^{n} w_{i\hat{j}}^2 \mathbf{v}_i}{\sum_{i=1}^{n} w_{i\hat{j}}^2} \,, \qquad \mathbf{q}_*^t = \frac{\sum_{i=1}^{n} w_{i\hat{j}} \mathbf{q}_i^t}{\sum_{i=1}^{n} w_{i\hat{j}}^2} \tag{18b}$$

The above translation removes $\mathbf{T}_{t,\hat{j}}$ from the optimization problem and leaves only the rotation $\mathbf{R}_{t,\hat{j}}$ to be minimized. Similar to [Kabsch 1978], the optimal rotation can be found by Singular Value Decomposition (SVD). First, two matrices $\mathbf{P} = [w_{1\hat{j}} \overline{\mathbf{p}}_1 \ldots w_{n\hat{j}} \overline{\mathbf{p}}_n]$ and $\mathbf{Q} = [\overline{\mathbf{q}}_1^t \ldots \overline{\mathbf{q}}_n^t]$ ($\mathbf{P}, \mathbf{Q} \in \mathbb{R}^{3 \times n}$) are constructed. Then, SVD is performed on $\mathbf{P}\mathbf{Q}^{\mathsf{T}}$ as follows:

$$\mathbf{P}\mathbf{Q}^{\mathsf{T}} = \sum_{i=1}^{n} w_{i\hat{j}} \overline{\mathbf{p}}_i {\overline{\mathbf{q}}_i^t}^{\mathsf{T}} = \mu \Sigma \vartheta^{\mathsf{T}} \tag{19}$$

Finally, the optimal rotation and translation can be obtained as follows:

$$\mathbf{R}_{t,\hat{j}} = \vartheta \mu^{\mathsf{T}} \,, \quad \mathbf{T}_{t,\hat{j}} = \mathbf{q}_*^t - \mathbf{R}_{t,\hat{j}} \mathbf{p}_* \tag{20}$$

The above bone transformation update process is described in Algorithm 4. The detailed proof for the optimal CoR in Eq. (18) and the optimal transformation in Eq. (20) can be found in the **Appendices** of the SSDR method [Le and Deng 2012].

Re-Initialization of Insignificant Bone Transformations

At the bone-vertex weights update step, some proxy bones maybe do not have major influence on any vertices (called "*insignificant bones*"). In such a case, their bone transformations cannot be accurately estimated. This is similar to the case of having less than 3 points in the Absolute Orientation problem, where the rotation cannot be uniquely determined. In addition, having those insignificant bones does not have major impact on the reconstruction error; thus, it is unnecessary to keep them. For this reason, its bone transformation needs to be randomly re-initialized if a bone is determined as an insignificant bone. The bone re-initialization can also prevent the iteration process to get stuck at a local minimum.

Specifically, in the SSDR method [Le and Deng 2012] the $\hat{j}$-th bone is identified as an insignificant bone if $\sum_{i=1}^{n} w_{i\hat{j}}^2 < \epsilon$. The threshold ϵ is empirically set to 3 since this is the minimum number of vertices required in the Absolute Orientation problem. If the j-th bone needs to be re-initialized, this bone is assigned to the vertex ι with the largest reconstruction error. Then, based on 20 nearest vertices of ι in the rest pose, denoted as $N(\iota)$, the bone transformations $\{\mathbf{R}_{t,j}|\mathbf{T}_{t,j}\}$ can be reinitialized by the Kabsch algorihtm [Kabsch 1978]. Note that re-initialization does not guarantee the strict decreasing of the objective function. As such, it could create a loop. To avoid such a potential loop, the SSDR method also allows the user to set the maximum number of re-initializations.

Algorithm 4 Update bone transformations $\mathbf{B}$

```
Require: $\mathbf{V} = \{\mathbf{v}'_{t,i}\}, \mathbf{P} = \{\mathbf{v}_i\}, \mathbf{W} = \{w_{ij}\}$
Ensure: $\mathbf{B} = \{\mathbf{R}_{t,j}|\mathbf{T}_{t,j}\}$ subject to Eq. (3)
 1: for $t = 1 \to S$ do
 2:     for $\hat{j} = 1 \to m$ do
 3:         if $\sum_{i=1}^{n} w_{i\hat{j}}^2 \geq \epsilon$ then
 4:             Calculate $\mathbf{q}_i, \forall i$ by Eq. (16)
 5:             Compute $\mathbf{v}_*$, $\mathbf{q}_*$, $\overline{\mathbf{v}}_i$ and $\overline{\mathbf{q}}_i$ $\forall i$ by Eq. (18)
 6:             Perform SVD by Eq. (19)
 7:             Compute rotation $\mathbf{R}_{t,\hat{j}}$ by Eq. (20)
 8:         else
 9:             Re-initialize insignificant bone transformations
10:         end if
11:     end for
12: end for
```

1.6 Comparisons and Discussion of Different Skinning Decomposition Methods

To compare and evaluate different skinning decomposition methods, a set of publicly available triangle mesh datasets are used, including 12 datasets from the “Deformation Transfer" [Sumner and Popović 2004], 2 datasets from the “Geometry Videos" [Briceño et al. 2003], 1 dataset (chickenCrossing) from the “Skinning Mesh Animations" [James and Twigg 2005], and 1 dataset (pjump) from the “Wavelet Compression" [Guskov and Khodakovsky 2004]. To quantitatively evaluate different methods, the error metric proposed in [Kavan et al. 2010] is chosen. Specifically, all the datasets are resized so that the rest poses are tightly enclosed by a unit sphere. Then, the error metric is calculated from the value E of the objective function in Eq. (2a) as follows:

$$E_{RMS} = 1000\sqrt{\frac{E}{3nS}}$$

The SMA method, the LSSP method, and the SSDR method were chosen for comparison, and all the three methods impose the orthogonal (rigid bone) constraint (Eq. (3)) on the bone rotation matrices, convex constraints and sparseness constraints on the bone-vertex weight map. To ensure a fair comparison, all the three methods ran on a single thread, without GPU-based acceleration. Note that [Kavan et al. 2010] was not included in this comparison because it cannot handle rigid bones.

Fig. 2 shows the comparison results on three models generated by SSDR, SMA, and LSSP. Due to the high deformability of the models, SMA fails to associate vertices into rigid bones and hefty distortion can be observed on the reconstructed poses, especially on the camel-collapse$_{11}$ model. Although LSSP can estimate the global deformation well, certain details are not correctly reconstructed, e.g., on the horse-collapse$_{10}$ model.

Table 1 quantitatively shows that the SSDR method clearly outperforms SMA and LSSP. Especially on the elastic models (i.e., camel-collapse$_{11}$, face-poses, horse-collapse$_3$, and pcow$_{24}$), the SSDR method generates significantly smaller errors than SMA. The reason is that SMA computes the bone transformations by first clustering the triangle rotation sequences into rigid transformation groups. Thus, it only works well if the mesh can be divided into nearly-rigid parts. The SSDR method also outperforms LSSP due to the following two limitations of LSSP: (1) LSSP employs the soft constraint for sparseness; and (2) LSSP does not consider the affinity constraint during the optimization, and the affinity constraint is only enforced through post-normalization. Also, even the LSSP method supports the sparseness of the bone-vertex weight map, it does not guarantee the maximum number of non-zero weights per vertex, which could be an issue for the implementation of hardware-accelerated skinning. Among the three approaches (SSDR, SMA, and LSSP), LSSP has the lowest performance due to its MATLAB implementation and the complexity of its clustering and optimization algorithm. In general, SSDR performs slightly slower than SMA. This is one limitation of the SSDR method.

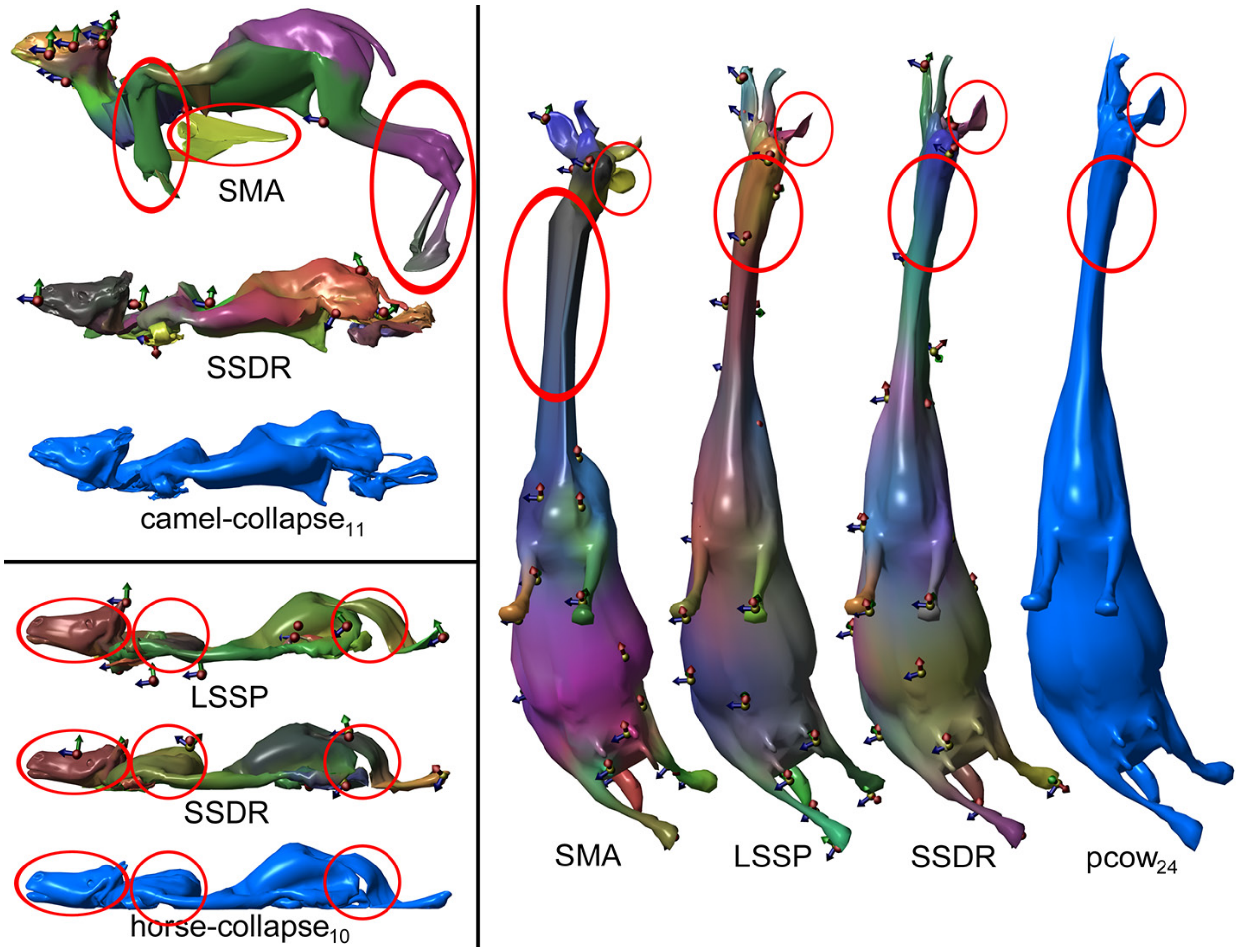

Figure 2: *Comparisons of the skinning decomposition results among the SSDR [Le and Deng 2012], SMA [James and Twigg 2005], and LSSP [Hasler et al. 2010]. LSSP is unable to run on the camel-collapse due to the large size of this dataset and SMA is unable to configure with 10 bones for the horse-collapse dataset. The example poses (ground-truth) are rendered in blue. Significant distortion areas are indicated in red circles. Image courtesy of [Le and Deng 2012]. Copyright ©ACM 2012.*

2 Example-based Skeleton Extraction

While LBS with unorganized proxy bones is convenient for efficient computing and compression, it poses some disadvantages on animation editing, compared to hierarchical bones (a.k.a. skeletal structure). Similar to the above skinning decomposition methods, example-based skeleton extraction methods aim to minimize the example pose reconstruction error (Eq. (2)) while handing one additional rotational constraint at the joints of the skeleton. These joint constraints connect the bones so that the joint becomes the center of rotation of two connected bones. Existing example-based skeleton extraction methods can be divided into two categories: *single-pass* method [Schaefer and Yuksel 2007; de Aguiar et al. 2008a; Hasler et al. 2010], and *multi-pass* method [Le and Deng 2014].

2.1 Single-pass Method

Due to the difficulty of adding the joint constraints, early skeleton extraction approaches [Schaefer and Yuksel 2007; de Aguiar et al. 2008a; Hasler et al. 2010] do not optimize the reconstruction error (Eq. (2)) explicitly. Instead, they perform a single pass consisting of three sequential steps:

- *Initialization*. First, skinning weights and unorganized bone transformations (proxy bones) are initialized. At this step, a clustering approach can be used to estimate rigid bone transformations [Schaefer and Yuksel 2007; de Aguiar et al. 2008a], and then, constrained least squares solvers can be used to compute skinning weights. An alternative yet more accurate solution is to perform a skinning decomposition [Hasler et al. 2010], which results in both skinning weights and bone transformations.
- *Skeleton reconstruction*. At this step, unorganized bones are linked together by constructing a tree-like structure, where each

edge of the tree corresponds to a bone and each node of the tree corresponds to the joint between two bones. This step can be formulated as the problem of finding a minimum spanning tree in a weighted graph, where each node represents one unorganized bone and the weight of each edge represents the cost of connecting two bones via a joint.

- *Joint positions calculation.* By convention, positions of the joints are computed in the rest pose. This step is formulated as finding the least squares deviations of the joint locations after applying bone transformations.

2.2 Initialization

The goal of this step is very similar to the goal of the aforementioned skinning decomposition problem, i.e. generating skinning weights and bone transformations from example poses. For this reason, techniques in section 1 can be employed. In addition, other possible strategies are also described below.

Segmentation/Spectral clustering. This strategy [de Aguiar et al. 2008a] includes two steps. At the segmentation step, seed vertices are selected as an evenly distributed subset of n vertices of the model (around 0.3% to 1.0% of n). Then, curvature-based segmentation [Yamauchi et al. 2005] is used to decompose the rest pose into surface patches where each patch contains one seed vertex. At the second step, *Spectral clustering* [Ng et al. 2001] is employed to further divide seed vertices into nearly-rigid components. In particular, the affinity matrix that encodes the distance between seed vertices is computed by the mutual Euclidean distance of seed vertices in all example poses. On one hand, the spectral clustering algorithm offers a robust initialization without manually specifying the number of clusters (or number of bones). On the other hand, it has higher computing time and memory complexities than the K-means clustering algorithm; thus, seed vertices are used to reduce the complexities.

Edge collapsing. Schaefer and Yuksel [2007] employ a bottom-up hierarchical clustering strategy. First, each mesh triangle is initialized as one cluster; clusters with adjacency edges are continuously merged together until the desired number of clusters is reached. The cluster merging is prioritized by the *rigid error function* that measures the per-cluster error if only rigid transformations are used to fit the cluster in the rest pose for all example poses. Compared to other non-hierarchical clustering strategies (e.g., K-means, mean shift, and spectral clustering), the edge collapsing strategy always generates clusters with connected triangles since the merging step only combines clusters with adjacency edges. This feature makes it more robust since the vertices supported by the same bone are typically in a connected region.

Skinning decomposition. After performing a spectral clustering to decompose the mesh into nearly-rigid parts, Hasler et al. [2010] run 10 iterations of block coordinate descent skinning decomposition to further refine bone transformations and skinning weights. The pipeline of these iterations are similar to [Le and Deng 2012], excepts for particular solvers used in each iteration. Specifically, Hasler et al. use L1-norm minimization to handle the weight sparseness constraints and use the Levenberg–Marquardt algorithm [Marquardt 1963] to handle the orthogonal bone transformations.

2.2.1 Skeleton Reconstruction

The skeletal topology is constructed based on both mesh connectivity and bone transformations. A weighted graph $\mathbb{G}$ with m nodes (corresponding to m bones) is constructed. The weight $g(j,k)$ between bone j and bone k is computed by Eq. (21), which puts a strong preference for having the joint (j,k) if the *joint location fitting error* (numerator) is small and the *blending of two bones* (denominator) is large. The joint fitting error is computed as the sum of squared difference of the *center of rotation* of two bones, $\mathbf{o}_{jk}$, after the two transformations of bone j and bone k. Here, the center of rotation $\mathbf{o}_{jk}$ between two bones j and k is defined at the rest pose, and its position is computed by [Anguelov et al. 2004] as described in follow-up section 2.2.2. Since $\mathbf{o}_{jk}$ is the point such that its locations after both bone transformations are most similar in all example poses; thus, the joint fitting error (the numerator in Eq. (21)) measures the quality of having the joint between bone j and bone k. The blending of two bones (the denominator in Eq. (21)) gives another measure on the relative position of the two bones, in which a larger blending means the two bones are more likely to share the common joint; and vice versa. As the result, $g(j,k)$ is large if bone j and bone k have a high probability of sharing the common joint.

$$g(j,k) = \frac{\sum_{t=1}^{S} \left\| \Big([\mathbf{R}_{t,j}|\mathbf{T}_{t,j}] - [\mathbf{R}_{t,k}|\mathbf{T}_{t,k}] \Big) \begin{bmatrix} \mathbf{o}_{jk} \\ 1 \end{bmatrix} \right\|_2^2}{\sum_{i=1}^{n} w_{ij} w_{ik}} \tag{21}$$

Finally, the skeleton $\mathbb{S}$ is determined as the *Minimum Spanning Tree* (MST) of $\mathbb{G}$ [Kruskal 1956]. In this writing, $(j,k) \in \mathbb{S}$ is used to denote (j,k) is an edge of the tree $\mathbb{S}$, that is, bone j and bone k share a common joint. This common joint is also denoted as (j,k), and the joint position (at the rest pose) is denoted as $\mathbf{o}_{jk}$, which is also the center of rotation of two bones j and k. In practice, the root of the skeleton is often manually specified. One simple way is to set the root as the joint that is the closest to the centroid of the rest pose.

2.2.2 Joint Constraints and Joint Positions Solver

Joint constraints. The constraints proposed by Anguelov et al. [2004] is used to make bone transformations rotate around common joints. Assuming the two bones j and k share the same joint (j,k). Let $\mathbf{o}_{jk} \in \mathbb{R}^3$ be the position of joint (j,k) (at the rest pose), which is also the center of rotation of two bones j and k. Intuitively, $\mathbf{o}_{jk}$ is the point such that its locations after both bone transformations are most similar in all frames. Thus, the joint fitting error in Eq. (22) will be close to zero.

$$\sum_{t=1}^{S} \left\| \Big([\mathbf{R}_{t,j}|\mathbf{T}_{t,j}] - [\mathbf{R}_{t,k}|\mathbf{T}_{t,k}] \Big) \begin{bmatrix} \mathbf{o}_{jk} \\ 1 \end{bmatrix} \right\|_2^2 \rightarrow \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{22}$$

Joint positions solver. Given the two bone transformations sequences $[\mathbf{R}_{t,j}|\mathbf{T}_{t,j}]$ and $[\mathbf{R}_{t,k}|\mathbf{T}_{t,k}]$, the common joint of bone j and bone k can be found by minimizing the joint fitting error in Eq. (22). However, many joints of articulated creatures are hinge joints, which only have one degree of freedom (rotation). For these type of joints, any point in the rotation axis will be the minimum solution to Eq. (22). There are two solutions to this problem.

Anguelov et al. [2004] add a soft constraint to minimize the distance between joint position to the centroid of two bones as follows:

$$\min_{\mathbf{o}_{jk}} \sum_{t=1}^{S} \left\| \Big([\mathbf{R}_{t,j}|\mathbf{T}_{t,j}] - [\mathbf{R}_{t,k}|\mathbf{T}_{t,k}] \Big) \begin{bmatrix} \mathbf{o}_{jk} \\ 1 \end{bmatrix} \right\|_2^2 + \mu \left\| \mathbf{g}_j + \mathbf{g}_k - 2\mathbf{o}_{jk} \right\|_2^2 , \tag{23}$$

where $\mathbf{g}_j$ and $\mathbf{g}_k$ denote the centroids of the vertices of the bones, and a user-specified parameter μ (typically set to a small number, e.g., 0.01) is used to control the trade-off between the joint fitting error and the joint position error.

The second approach [Schaefer and Yuksel 2007] uses a pseudo-inverse solver to solve the joint position in a subspace. It gives a slightly smaller approximation error than [Anguelov et al. 2004]. However, it tends to find the wrong subspace if the estimated bone rotations are non-robust.

2.3 Multi-pass Strategy

$$\min_{\mathbb{S}, \mathbf{o}_{jk}, w_{ij}, [\mathbf{R}_{t,j}|\mathbf{T}_{t,j}]} E = E_D + \omega E_S + \lambda E_J \tag{24a}$$

$$\text{Where: } E_D = \frac{1}{nS} \sum_{i=1}^{n} \sum_{t=1}^{S} \left\| \sum_{j=1}^{m} w_{ij} [\mathbf{R}_{t,j}|\mathbf{T}_{t,j}] \begin{bmatrix} \mathbf{v}_i \\ 1 \end{bmatrix} - \mathbf{v}'_{t,i} \right\|_2^2 \tag{24b}$$

$$E_S = \sum_{j=1}^{m} \mathbf{w}_j{}^{\mathsf{T}} \mathbf{L} \mathbf{w}_j \tag{24c}$$

$$E_J = \frac{1}{S} \sum_{(j,k)\in\mathbb{S}} \sum_{t=1}^{S} \left\| \Big([\mathbf{R}_{t,j}|\mathbf{T}_{t,j}] - [\mathbf{R}_{t,k}|\mathbf{T}_{t,k}] \Big) \begin{bmatrix} \mathbf{o}_{jk} \\ 1 \end{bmatrix} \right\|_2^2 \tag{24d}$$

$$\text{Subject to: } w_{ij} \geq 0, \ \sum_{j=1}^{m} w_{ij} = 1, \ \left\| w_i \right\|_0 \leq K (=4), \forall i, j \tag{24e}$$

$$\mathbf{R}_{t,j}{}^{\mathsf{T}} \mathbf{R}_{t,j} = \mathbf{I}, \ \det \mathbf{R}_{t,j} = 1, \forall j, t \tag{24f}$$

The multi-pass approach proposed by Le and Deng [2014] minimizes the objective function in Eq. (24a) to find the optimized LBS model with skeletal constraints. The objective function E includes the following 3 terms:

- *Data fitting term* E_D (Eq. (24b)) minimizes the mesh reconstruction error. This term is the squared sum of the reconstruction errors for all the vertices for all the example poses.
- *Weight regularization term* E_S (Eq. (24c)) favors the smoothness of skinning weights and drives the removal of redundant bones (section 2.3.2). This term is derived from the fairness and smoothness conditions on the manifold, which is similar to [Kim et al. 2010]. $\mathbf{w}_j \in \mathbb{R}^n$ is the column vector of n skinning weights of bone j, and $\mathbf{L} \in \mathbb{R}^{n \times n}$ is a discrete Laplacian matrix of the input mesh, where L_{ik} represents the similarity between the weights of two neighboring vertices i and k.
- *Joint constraint term* E_J (Eq. (24d)) keeps any two connected bone transformations $\in \mathbb{S}$ rotate around their common joint. This soft constraint is derived from [Anguelov et al. 2004; Schaefer and Yuksel 2007]. If two bones j and k share the common joint (j, k), this constraint favors the rest-pose position of the joint, $\mathbf{o}_{jk}$, going to the same position after bone j transformation and bone k transformation (also refer to the explanation of Eq. (21) in section 2.2.1).

The minimization of E is also subject to the set of hard constraints. It includes *convex constraints* (non-negativity and affinity) and *sparseness constraints* (no more than 4 non-zero weights per vertex) on the skinning weights $\{w_{ij}\}$ (Eq. (24e)), and includes *orthogonal constraints* (Eq. (24f)) on the bone transformations $\{\mathbf{R}_{t,j}\}$ (all the bone transformations need to be rigid).

2.3.1 Adding Joint Constraints

To enforce the joint constraints while updating the rigid bone transformations, the solution to the Absolute Orientation problem with blending to relate two sets of points [Le and Deng 2012], is employed, while the joints are treated as additional points with a large weight (λ). When the parameter λ is increased, bones are constrained to rotate more strictly around the joints as illustrated in Fig. 3.

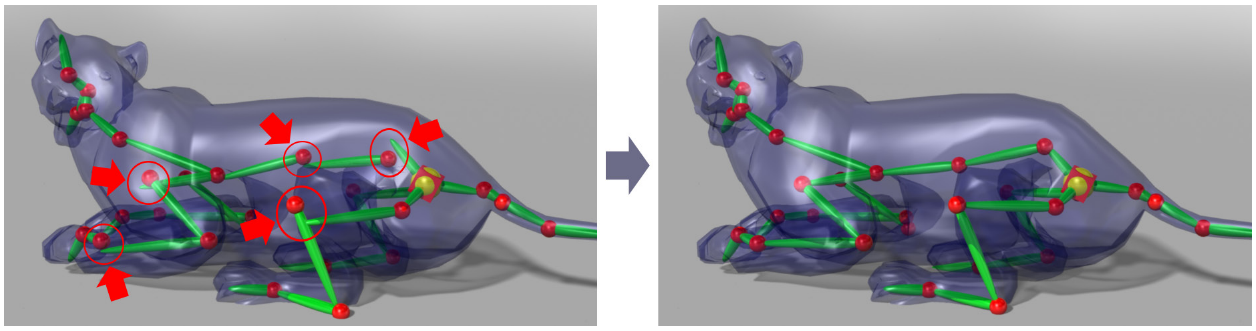

Figure 3: *(Left) Without the joint constraints, the bone transformations (bones) generated by [Le and Deng 2012] do not always rotate around the joints. (Right) With the soft joint constraints, bones rotate more strictly around the joints [Le and Deng 2014]. Since the bone transformations are alternatively updated with the joint locations, the resulting rigging model can converge to a local optimum with a good approximation of the input. Image courtesy of [Le and Deng 2014]. Copyright ©ACM 2014.*

At this bone transformations update step, the objective function (Eq. (24a)) needs to be minimized with respect to the bone transformations $[\mathbf{R}_{t,j}|\mathbf{T}_{t,j}]$. To constrain the rotation matrix $\mathbf{R}_{t,j}$ to be orthogonal, Le and Deng [2014] employ an optimization strategy for the rigid transformations with blending, where bones are updated one by one while keeping the rest of the bones fixed, described below.

The transformation of bone $\hat{j}$ at pose t is updated by minimizing the following objective function, Eq. (25a), which contains two parts. The first part corresponds to the data fitting term (Eq. (24b)) whose optimal solution brings the rest pose $\mathbf{v}_i$ to the residual $\mathbf{q}_i^t$ (Eq. (25b)). The second part corresponds to the joint constraints in Eq. (24d), which enforces $[\mathbf{R}_{t,\hat{j}}|\mathbf{T}_{t,\hat{j}}]$ to bring every joint $\mathbf{o}_{\hat{j}k}$ of bone $\hat{j}$ to its expected position $\Psi_k^t(\mathbf{o}_{\hat{j}k})$ after bone k transformation. Note that the weight smoothness term E_S in Eq. (24c) can be dropped since $[\mathbf{R}_{t,j}|\mathbf{T}_{t,j}]$ is not involved in E_S.

$$\min E_{\hat{j}}^t = \frac{1}{n}\sum_{i=1}^{n}\left\| w_{i\hat{j}}[\mathbf{R}_{t,\hat{j}}|\mathbf{T}_{t,\hat{j}}]\begin{bmatrix}\mathbf{v}_i\\1\end{bmatrix} - \mathbf{q}_i^t\right\|_2^2 + \lambda\sum_{(\hat{j},k)\in\mathbb{S}}\left\|[\mathbf{R}_{t,\hat{j}}|\mathbf{T}_{t,\hat{j}}]\begin{bmatrix}\mathbf{o}_{\hat{j}k}\\1\end{bmatrix} - \Psi_k^t(\mathbf{o}_{\hat{j}k})\right\|_2^2 \tag{25a}$$

$$\text{Where: } \mathbf{q}_i^t = \mathbf{v}'_{t,i} - \sum_{j=1, j\neq\hat{j}}^{m} w_{ij}[\mathbf{R}_{t,j}|\mathbf{T}_{t,j}]\begin{bmatrix}\mathbf{v}_i\\1\end{bmatrix} \tag{25b}$$

$$\Psi_k^t(\mathbf{o}_{\hat{j}k}) = [\mathbf{R}_{t,k}|\mathbf{T}_{t,k}]\begin{bmatrix}\mathbf{o}_{\hat{j}k}\\1\end{bmatrix} \tag{25c}$$

The optimal solution to minimize the objective function Eq. (25a) is the combination of the solution of rigid transformations with blending and the solution to the *Weighted Absolute Orientation* problem [Kabsch 1978]. The center of rotation $\mathbf{p}_*$ for the rest pose (Eq. (26a)) and the center of rotation $\mathbf{q}_*^t$ for pose t (Eq. (26b)) need to be computed first. Note that, in Eqs. (26a) and (26b), $\left|(\hat{j},k)\in\mathbb{S}\right|$ denotes the number of the edges that are connected with $\hat{j}$ in $\mathbb{S}$.

$$\mathbf{p}_* = \frac{\frac{1}{n}\sum_{i=1}^{n} w_{i\hat{j}}^2\mathbf{v}_i + \lambda\sum_{(\hat{j},k)\in\mathbb{S}}\mathbf{o}_{\hat{j}k}}{\frac{1}{n}\sum_{i=1}^{n} w_{i\hat{j}}^2 + \lambda\left|(\hat{j},k)\in\mathbb{S}\right|} \tag{26a}$$

$$\mathbf{q}_*^t = \frac{\frac{1}{n}\sum_{i=1}^{n} w_{i\hat{j}}\mathbf{q}_i^t + \lambda\sum_{(\hat{j},k)\in\mathbb{S}}\Psi_k^t(\mathbf{o}_{\hat{j}k})}{\frac{1}{n}\sum_{i=1}^{n} w_{i\hat{j}}^2 + \lambda\left|(\hat{j},k)\in\mathbb{S}\right|} \tag{26b}$$

Then, the center of rotation is subtracted from each vertex and each joint as follows:

$$\overline{\mathbf{p}}_i = \mathbf{v}_i - \mathbf{p}_*; \qquad \overline{\mathbf{o}}_{\hat{j}k} = \mathbf{o}_{\hat{j}k} - \mathbf{p}_* \tag{27a}$$
$$\overline{\mathbf{q}}_i^t = \mathbf{q}_i^t - w_{i\hat{j}}\mathbf{q}_*^t; \qquad \overline{\Psi}_k^t(\mathbf{o}_{\hat{j}k}) = \Psi_k^t(\mathbf{o}_{\hat{j}k}) - \mathbf{q}_*^t \tag{27b}$$

The points after subtraction are concatenated into matrices $\mathbf{P}, \mathbf{Q} \in \mathbb{R}^{3\times(n+|(\hat{j},k)\in\mathbb{S}|)}$ as follows:

$$\mathbf{P} = \left[\frac{w_{1\hat{j}}}{n}\overline{\mathbf{p}}_1 \dots \frac{w_{n\hat{j}}}{n}\overline{\mathbf{p}}_n \middle| \,\forall_{(\hat{j},k)\in\mathbb{S}}\lambda\overline{\mathbf{o}}_{\hat{j}k}\right] \tag{28a}$$
$$\mathbf{Q} = \left[\frac{1}{n}\overline{\mathbf{q}}_1^t \dots \frac{1}{n}\overline{\mathbf{q}}_n^t \middle| \,\forall_{(\hat{j},k)\in\mathbb{S}}\lambda\overline{\Psi}_k^t(\mathbf{o}_{\hat{j}k})\right] \tag{28b}$$

Finally, the optimal transformation of bone $\hat{j}$ at pose t, Eq. (29a), is computed by performing SVD on $\mathbf{PQ}^\mathsf{T}$.

$$\mathbf{R}_{t,\hat{j}} = \vartheta\mu^\mathsf{T}; \;\; \mathbf{T}_{t,\hat{j}} = \mathbf{q}_*^t - \mathbf{R}_{t,\hat{j}}\mathbf{p}_* \tag{29a}$$
$$\text{Where: } \mu\varsigma\vartheta^\mathsf{T} = \mathbf{PQ}^\mathsf{T} \tag{29b}$$

2.3.2 Skeleton Pruning

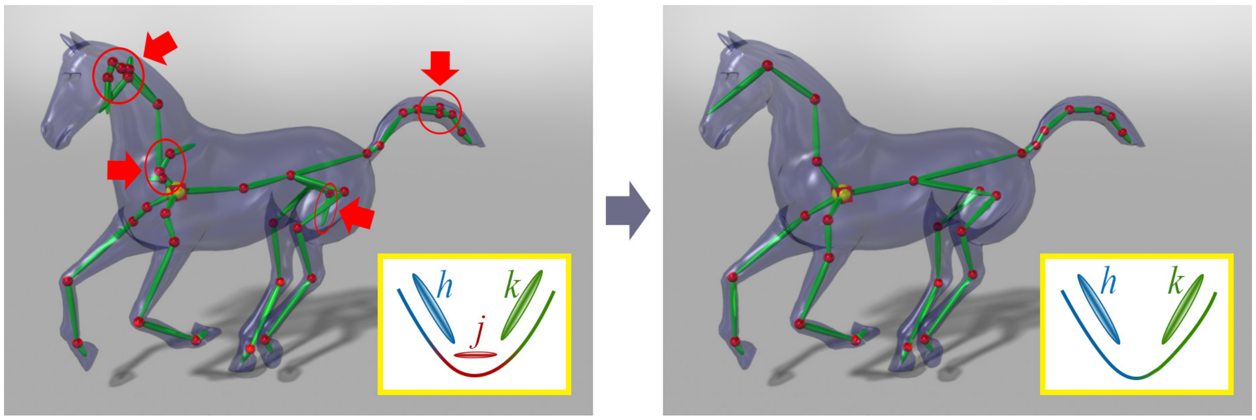

Figure 4: *The redundant bones in the left panel are pruned to achieve the neat skeleton in the right panel by [Le and Deng 2014]. As illustrated in the two yellow boxes, the redundant bone j is identified by utilizing the weight regularization term to force its weights degenerate. Image courtesy of [Le and Deng 2014]. Copyright ©ACM 2014.*

Since the initialization step (section 2.2) considers neither transformation blending nor the skeleton structure, it might generate some redundant bones. Typically, the redundant bones are located in highly deformable regions, e.g., around the joints, as they cause large approximation errors at the cluster initialization step. Later, after the LBS resolves the highly deformable regions, the redundant bones are no longer needed and thus should be removed. Unfortunately, since the redundant bones do not violate any conditions in the LBS formulation, accurately identifying them is difficult. For this reason, the solution optimized by a general data fitting with alternative skinning weights update and bone transformations update [Le and Deng 2012], would still retain the redundant bones, because keeping the redundant bones indeed helps to further reduce the LBS deformation error.

Instead of conducting a brute-force search for the redundant bones, Le and Deng [2014] utilize the weight regularization term to force their weights degenerate. An example of this strategy is illustrated in Fig. 4. As illustrated in the yellow box in the left panel, the bone j (red) is initialized at a potential joint between the bone h (blue) and the bone k (green). Although the blending between h and k can closely approximate the deformation, having the bone j is still a valid solution for the best approximation. By adding the weight regularization term, the minimized objective function (Eq. (24a)) makes the weights of j become very close to zero, in order to improve the smoothness of the skinning weights. As the result, the bone j becomes degenerate, and it can be removed. Specifically, the redundant bone j can be removed if the sum of its squared weights is smaller than 1% of the largest sum of the squared weights among all the bones. Mathematically, the bone j is removed if its weights satisfy the condition described in the following Eq. (30).

$$\text{Remove bone } j \text{ if } \sum_{i=1}^{n} {w_{ij}}^2 < 10^{-2} M \qquad (30)$$
$$\text{where: } M = \max_{k} \left\{ \sum_{i=1}^{n} {w_{ik}}^2 \right\}$$

2.4 Comparisons and Discussion of Different Skeleton Extraction Methods

9 test datasets (Table 2) are obtained from various publicly available sources: the **cat-poses**, **horse-poses**, **lion-poses**, and **horse-gallop** from [Sumner and Popović 2004]; the **hand** from [Utah 2013]; the **dance** and **cow** from [Briceño et al. 2003]; the **scape** was obtained from [Anguelov et al. 2005]; and the **samba** was obtained from [Vlasic et al. 2008].

Since the three main steps of a single-pass method are well-separated, its pipeline is efficient and simple to implement. However, the single-pass methods [Schaefer and Yuksel 2007; de Aguiar et al. 2008a; Hasler et al. 2010] have the following shortcomings: First, bones cannot be identified perfectly since the initialization step does not model connection between bones (skeletal structure). As such, this step would either require non-trivial model-specific parameter tuning or result in an un-robust skeleton with many redundant bones. Second, each step in this pipeline is performed on the output from the previous step; thus, each step does not model any constraints on the previous or next steps. For example, the clustering step does not model transformation blending. Likewise, after the joint locations are determined, joint constraints would change the bone transformations generated at the previous step. As an end result, errors could be significantly accumulated (e.g., from the root joint to leaf joints). Examples in Fig. 5 show two common limitations of these single-pass methods.

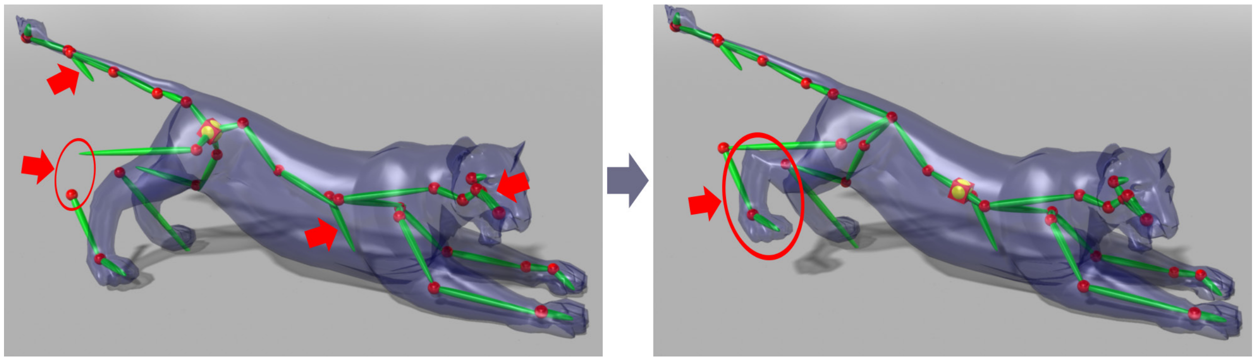

Figure 5: *Examples that show two common limitations of the single-pass skeleton extraction methods: (i) an over-estimated bone initialization may generate inaccurate and redundant bones (indicated by red arrows); (ii) an inaccurate estimation of bone transformations (indicated by red arrows with ellipse) causes noticeable deformation errors when the bones are connected by joints (see the change of the legs). The examples in this figure are generated using [Hasler et al. 2010]. Image courtesy of [Le and Deng 2014]. Copyright ©ACM 2014.*

Fig. 6 shows the skeletons extracted from 3 test datasets by the multi-pass approach [Le and Deng 2014] as well as their corresponding cluster initialization results. Despite generating over-estimated clusters at the initialization step, the multi-pass approach successfully pruned redundant bones, especially for the **dance** and **hand** models. Fig. 7 demonstrates the robustness and effectiveness of its skeleton pruning for handling different numbers of clusters from the initialization step. This approach can produce similar final skeletons with consistent structure on the trunk and legs, by pruning most of the redundant bones. Note that minor differences only appear on certain highly deformable regions such as the tail and feet.

Fig. 8 shows the comparisons among the multi-pass approach [Le and Deng 2014] and three state of the art single-pass approaches [Schaefer and Yuksel 2007; de Aguiar et al. 2008a; Hasler et al. 2010]. For a fair comparison, the same number of bones are used in the four methods.

Table 2 shows quantitative comparisons among all the four methods (*method I* - [Le and Deng 2014], *method II* - [Schaefer and Yuksel 2007], *method III* - [de Aguiar et al. 2008a], and *method IV* - [Hasler et al. 2010]). In term of the approximation power (measured by lower RMSE), [Le and Deng 2014] soundly outperforms the other three methods due to its iterative rigging. However, it is slower than both the method II and the method III, since it needs many iterations. Fig. 9 shows examples of visual distortion on the reconstructed **hand** poses with respect to different RMSE values.

3 Skinning Weight Reduction and Compression

In many linear surface deformation methods (including LBS and cage-based deformation), the deformed position of a surface vertex can be expressed as a weighted summation of control points or bones. Specifically, in cage-based deformation, a vertex $\mathbf{v}'_i$ on the

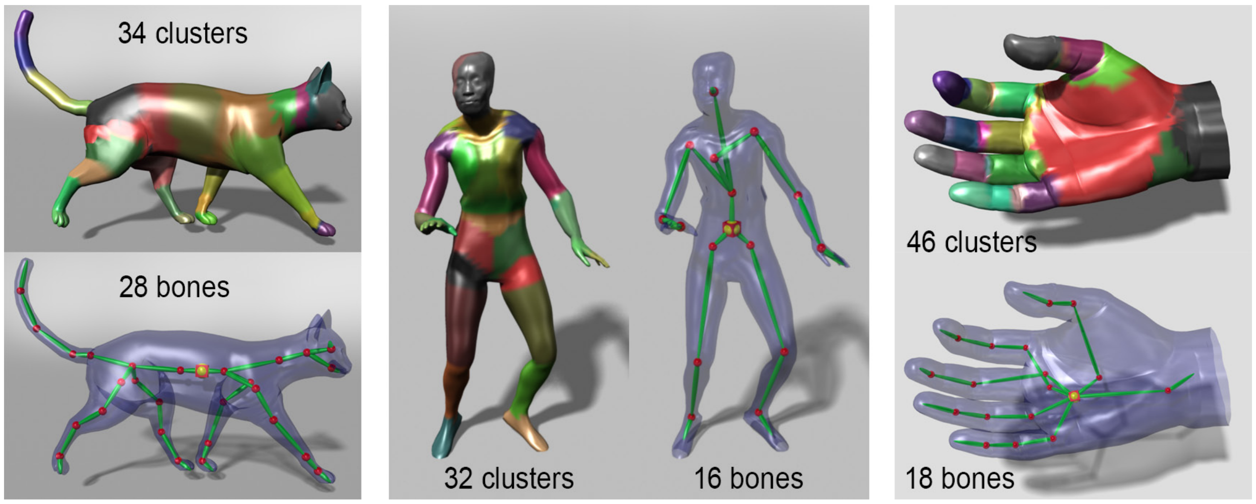

Figure 6: *The skeletons extracted from 3 datasets by the multi-pass skeleton extraction approach proposed by Le and Deng [2014] (from left to right):* ***cat-poses****,* ***dance****, and* ***hand****. The clustering results obtained at the initialization step are also illustrated via a color-coded scheme. Using the same set of parameters, this approach can robustly determine the optimal number of bones in the skeletons thanks to the skeleton pruning. Image courtesy of [Le and Deng 2014]. Copyright ©ACM 2014.*

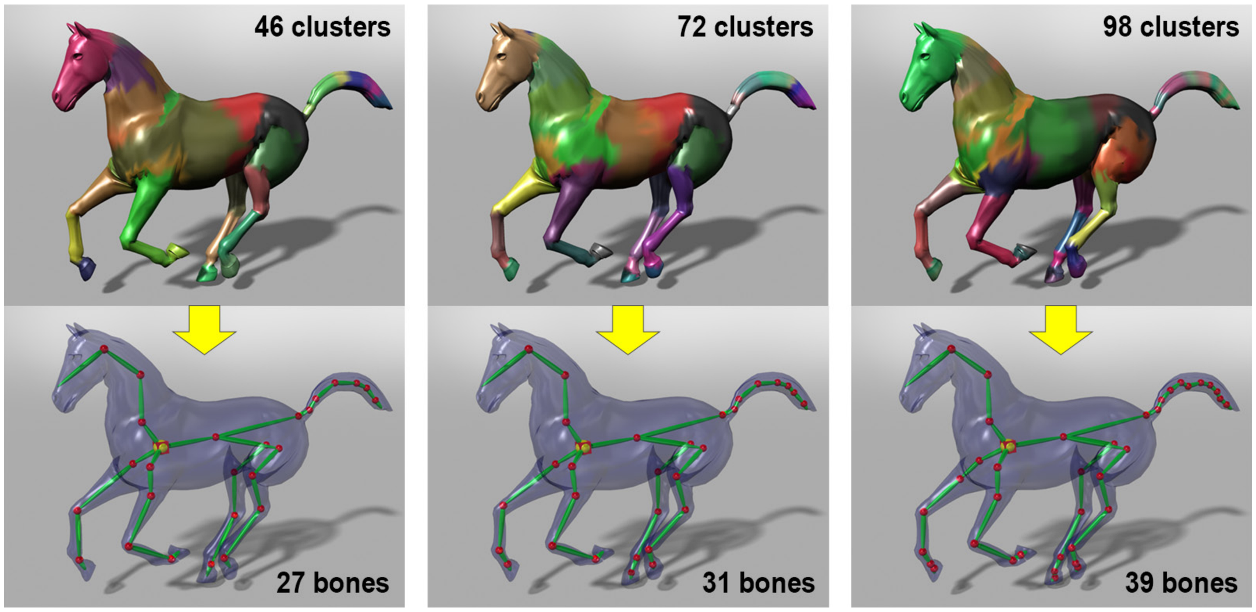

Figure 7: *The extracted* ***horse-gallop*** *skeletons by the multi-pass skeleton extraction approach proposed by Le and Deng [2014] with different cluster initializations. Despite very different numbers of initialized clusters, this approach can output similar final skeletons with consistent structure on the trunk and legs; minor differences on the tail and feet are due to the high deformations on these regions. Image courtesy of [Le and Deng 2014]. Copyright ©ACM 2014.*

target surface is computed as a weighted combination of the vertices $\mathbf{c}_j$ of the control mesh (i.e., cage), as described in Eq. (31).

$$\mathbf{v}_i' = \sum_j w_{ij}\mathbf{c}_j, \tag{31}$$

where w_{ij} denotes the influence (weight) of the j-th control point on vertex i on the surface. Similarly, in LBS case, a similar

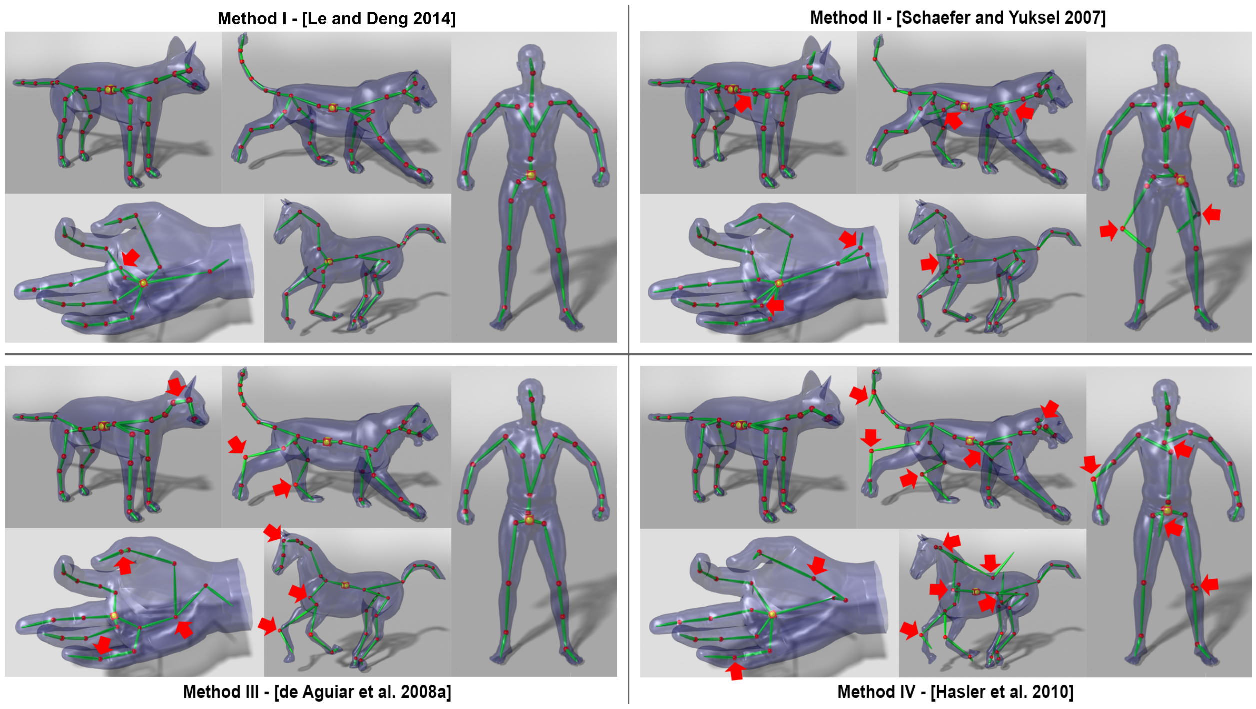
Method I - [Le and Deng 2014]
Method II - [Schaefer and Yuksel 2007]
Method III - [de Aguiar et al. 2008a]
Method IV - [Hasler et al. 2010]

Figure 8: *Comparisons between the multi-pass approach [Le and Deng 2014] and three state of the art single-pass approaches [Schaefer and Yuksel 2007; de Aguiar et al. 2008a; Hasler et al. 2010]. The five test datasets shown in this figure are (in the clockwise direction starting from the top-left corner):* ***cat-poses****,* ***lion-poses****,* ***scape****,* ***horse-gallop****, and* ***hand****. The issues in the results are indicated by red arrows. Image courtesy of [Le and Deng 2014]. Copyright ©ACM 2014.*

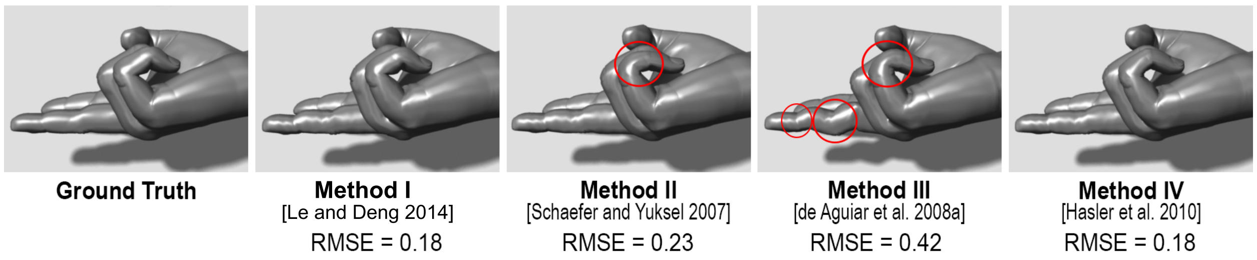

Figure 9: *Examples of visual distortion on the reconstructed* ***hand*** *poses with respect to different RMSE values (pay attention to the red circled areas). Fig. 8 shows the corresponding skeletons. Image courtesy of [Le and Deng 2014]. Copyright ©ACM 2014.*

weighted combination formula exists (Eq. (32)).

$$\mathbf{v}'_i = \sum_{j=1}^{m} w_{ij} [\mathbf{R}_j | \mathbf{T}_j] \mathbf{v}_i, \tag{32}$$

where $[\mathbf{R}_j|\mathbf{T}_j]$ denotes the transformation of the j-th bone, and $\mathbf{v}_i$ denotes the position of vertex i at the rest pose.

In both Eq. (31) and Eq. (32), the non-negativity and affinity constraints are often imposed onto $\{w_{ij}\}$, that is, $w_{ij} \geq 0$ and $\sum_{j=1}^{m} w_{ij} = 1$ (for any i). One important property of these deformation methods is *locality*: the deformed position of a vertex is often influenced by a small number of bones (or control points) that are in a local region. Furthermore, the weight w_{ij} typically decreases with respect to the Euclidean distance [Ju et al. 2005] and geodesic distance [Joshi et al. 2007] between the vertex and the bone (or control point). Therefore, many weights $\{w_{ij}\}$ for a particular vertex $\mathbf{v}'_i$ are often very small (or even close to zero), and only a few of them are significant to determine the final position of $\mathbf{v}'_i$ [Landreneau and Schaefer 2010]. As such, effectively exploiting the weight locality would not only make the reduction of weights for each vertex possible, but also make the efficient GPU implementation of the deformation and skinning algorithms feasible, since modern GPU structure efficiently supports a fixed number of operations per vertex at architectural level.

In recent years, a number of skinning weight reduction methods have been proposed, including:

- *k-largest weight reduction* that straightforwardly chooses the largest k weights from the original weight set;
- *smooth weight reduction* that optimizes the Laplacian of the reduced weights to match that of the original weight set [Landreneau and Schaefer 2010];
- *geometric weight reduction* that selects an optimal set of bones (or control points), having the smallest square errors when used to individually predict deformed position, and then compute the corresponding bone-vertex weights [James and Twigg 2005]; and
- *Poisson-based weight reduction* that optimizes a Poisson equation defined over the surface [Landreneau and Schaefer 2010].

The above weight reduction methods are designed to optimally retain the deformation quality while reducing the number of weights for each vertex to a small, fixed number. Often, this reduced number is set to 4 due to practical consideration of the efficient GPU implementation. However, *exceptional vertices* that are naturally associated with more than k bones or control points could exist on some 3D models (examples are shown in Fig. 10). As a result, the above skinning weight reduction methods could lead to noticeable visual artifacts on certain 3D models. The recent proposed *two-layer sparse compression* method [Le and Deng 2013] is designed to eliminate such noticeable visual artifacts as well as increase the runtime efficiency of dense-weight skinning models.

In follow-up course note, we first briefly describe the main methodology of the k-largest weight reduction, smooth weight reduction, geometric weight reduction, and Poisson-based weight reduction. Then, we describe the methodology of the two-layer sparse compression method in details. Lastly, we describe some comparison results among all the above skinning weight reduction and compression approaches.

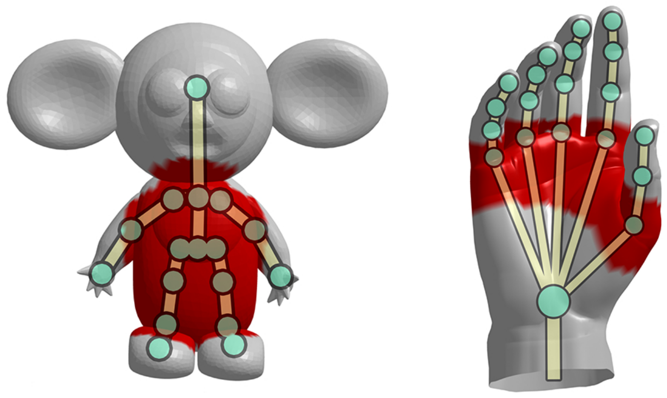

Figure 10: *Two examples of 3D models with exceptional vertices (red regions) that are naturally associated with more than k bones or control points. Image courtesy of [Le and Deng 2013]. Copyright ©ACM 2013.*

3.1 K-largest Weight Reduction

The basic idea of the K-largest weight reduction method is to retain k largest weights from the original weight set for each vertex and then renormalize the retained weights (to satisfy the affinity constraint). Its underlying rationale is that, due to the observation that the blending weight is in general proportional to the distance between the vertex and the control point or bone, a smaller weight in the original weight set generally implies a larger distance between the vertex and the control point or bone, and thus forcing them to zero would be justified to certain extent.

Mathematically, this weight reduction method can be described as follows:

$$\mathbf{v}'_i = \sum_{j=1}^{k} \hat{w}_{ij}[\mathbf{R}_j|\mathbf{T}_j]\mathbf{v}_i, \qquad \text{(for linear blend skinning)} \tag{33}$$

$$\mathbf{v}'_i = \sum_{j=1}^{k} \hat{w}_{ij}\mathbf{c}_j, \qquad \text{(for cage-based deformation)} \tag{34}$$

assuming $w_{ij}(\forall j = 1 \cdots k) \geq w_{il}$ $(\forall l > k)$ is valid, and $\hat{w}_{ij}$ is the renormalized version of w_{ij}. Despite its simplicity, however, with this method, the influence sets of bones (or control points) for neighboring vertices could be different, which could cause visual tears and normal discontinuity on 3D models at the end. One such example of cage-based deformation is shown in Fig. 11.

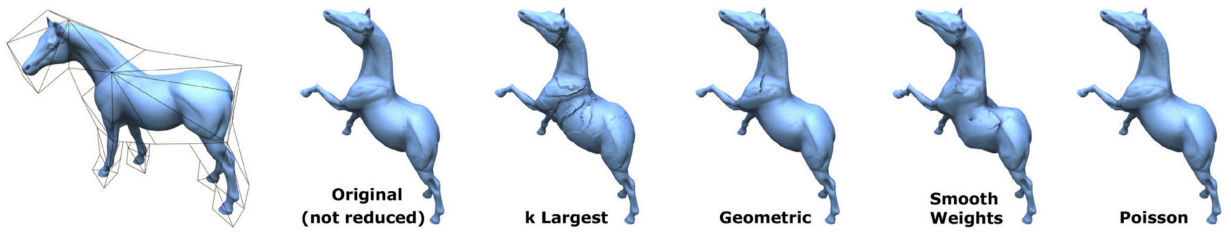

Figure 11: *(from left to right) The rest pose with control cage, original deformed surface, reduced to 8 per vertex using k-largest weights, geometric reduction, smooth weight reduction, and Poisson-based weight reduction. Image courtesy of [Landreneau and Schaefer 2010]. Copyright ©Eurographics Association 2010.*

3.2 Smooth Weight Reduction

The smooth weight reduction method [Landreneau and Schaefer 2010] does not utilize multiple example poses (just need a single pose), and it also assumes the influence set of k control points (or bones) for vertex i, denoted as $\mathfrak{B}_i$, is heuristically determined based on the order of their original weight values, which is the same as the above k-largest weight method.

The essential idea of this method is to find a set of smooth weights that satisfy the weights are zero outside of the influence set for a vertex, which can be achieved by optimizing the Laplacian of the reduced weights to match that of the original weight set [Landreneau and Schaefer 2010]. Specifically, in the case of cage-based deformation (or any generalized barycentric coordinate deformation), let $\bar{\rho}_j(\mathbf{v})$ be the barycentric coordinate function of the j-th control point of a cage, the goal of this weigh reduction method is to maximize the smoothness of this restricted influence function, $\rho_j(\mathbf{v})$, whose value is 0 if $j \notin \mathfrak{B}_i$, by minimizing the following objective function:

$$\sum_j (\triangledown^2 \rho_j - \triangledown^2 \bar{\rho}_j), \tag{35a}$$

$$\text{Subject to: } \sum_j \rho_j(\mathbf{v}) = 1, \tag{35b}$$

$$\sum_j \rho_j(\mathbf{v})\mathbf{c}_j = \mathbf{v}. \tag{35c}$$

In the above equations, the $\triangledown^2$ operator for polygonal meshes is a summation of cotan weights [Pinkall and Polthier 1993; Landreneau and Schaefer 2010]. Thus, solving the above minimization problem can be converted to the solving of a sparse, linear system of nk+4 equations (n is the total number of vertices on the mesh, and k is the number of retained weights for each vertex). If an additional non-negativity constraint is imposed on $\rho_j(\mathbf{v})$, then this problem can be solved using nonnegative least squares (NNLS) solvers [Lawson and Hanson 1974]. The work of [Landreneau and Schaefer 2010] also proposed an efficient, iterative solver that incrementally updates the weights of each vertex by assuming the weights of its one-ring neighbors are fixed.

This smooth weight reduction method can also be straightforwardly applied to the LBS model. In particular, $\mathfrak{B}_i$ denotes the influence set of k bones for vertex i, and the above Eq. (35c) also needs to be changed to the following form:

$$\sum_j \rho_j(\mathbf{v})[\mathbf{R}_j|\mathbf{T}_j]\mathbf{v} = \mathbf{v}' \tag{36}$$

Compared with the naive k-largest weight reduction method, the smooth weight reduction method can generate smoother results (less visible artifacts on the surface), as shown in the two examples in Fig. 11 and Fig. 12. However, we still can see tears and visual artifacts at certain regions on the 3D models, the main reason is that since only a single pose is utilized in this method, it does not have any clue on plausible deformations of the 3D model that may be generated by users [Landreneau and Schaefer 2010].

3.3 Geometric Weight Reduction

In order to sample the plausible deformations of the given 3D model, a set of user-provided example poses typically needs to be utilized. By employing a set of example poses, James and Twigg [2005] proposed a geometric weight reduction approach for LBS model, and later it has been applied to cage-based deformation [Landreneau and Schaefer 2010]. Specifically, in the LBS case, the crux of geometric weight reduction is to first determine an optimal set of influence bones for each vertex, and then compute the corresponding bone-vertex weights.

(1) *Identifying an optimal set of influence bones* for each vertex. In this approach, instead of directly solving a complex discrete model/variable section problem that does not have any yet known optimal polynomial solutions [Zou and Hastie 2005], it chooses the best k bones for vertex i that have the smallest square errors when used to individually predict the deformed positions [James and Twigg 2005]. Mathematically, let $\mathfrak{B}_i$ be the set of bones that maximally influence vertex i, and further assume $|\mathfrak{B}_i| \le k$ (no more than k bone-vertex weights for each vertex), $\mathfrak{B}_i$ can be obtained by solving the following minimization problem:

$$e_{ij} = \sum_{t=1}^{S} ||\mathbf{v}'_{t,i} - [\mathbf{R}_{t,j}|\mathbf{T}_{t,j}]\mathbf{v}_i||_2^2, \qquad j = 1 \cdots m \tag{37}$$

where $\mathbf{v}'_{t,i}$ denotes the deformed position of vertex i at the t-th example pose. To this end, based on the obtained $\{e_{ij}\}$ $(j = 1 \cdots m)$, the smallest k bones are chosen and included to $\mathfrak{B}_i$.

(2) *Computing bone-vertex weights* for each vertex. After $\mathfrak{B}_i$ is obtained, the corresponding bone-vertex weights $\{w_{ij}\}$ $(j = 1 \cdots k)$ for vertex i can be computed by a least squares approach. Mathematically, it can be formulated as the solving of the following constrained system of S equations:

$$\sum_{j \in \mathfrak{B}_i} ([\mathbf{R}_{t,j}|\mathbf{T}_{t,j}]\mathbf{v}_i) w_{ij} = \mathbf{v}'_{t,i}, \qquad t = 1 \cdots S \tag{38}$$

The above system of equations (Eq. (38)) can be converted to its equivalent matrix form as follows:

$$\mathbf{A}^{(i)}\mathbf{w}^{(i)} = \mathbf{b}^{(i)} \tag{39a}$$

$$\text{Subject to: } \sum_j w_{ij} = 1 \tag{39b}$$

To prevent the weight over-fitting issue, James and Twigg [2005] introduce another scaling parameter c (e.g., $c = 1/||\mathbf{b}^{(i)}||_2$) into the above Eq. (39), and also incorporate the affinity constraint (Eq. (39b)) through matrix/vector expansion as follows.

$$\begin{bmatrix} c\mathbf{A}^{(i)} \\ 1 \cdots 1 \end{bmatrix} \mathbf{w}^{(i)} = \begin{pmatrix} c\mathbf{b}^{(i)} \\ 1 \end{pmatrix} \quad \Leftrightarrow \quad \tilde{\mathbf{A}}^{(i)}\mathbf{w}^{(i)} = \tilde{\mathbf{b}}^{(i)} \quad (\text{subject to: } \mathbf{w}^{(i)} \ge 0) \tag{40}$$

The above Eq. (40) can be solved using various constrained least squares techniques, such as an efficient nonnegative least squares (NNLS) iterative solver proposed by Lawson and Hanson [1974]. To the end, $\{w_{ij}|j \in \mathfrak{B}_i\}$ needs to be normalized to satisfy the affinity constraint (Eq. (39b)).

In general, the geometric weight reduction method can produce better visual quality than the K-largest weight method as well as the smooth weight reduction method (for example, the silhouette can be better approximated); however, the geometric weight reduction method still fails to guarantee the normal continuities, which would still lead to noticeable visual artifacts on the surface (refer to cage-based deformation example in Fig. 11). Another linear blend skinning comparison example is shown in Fig. 12.

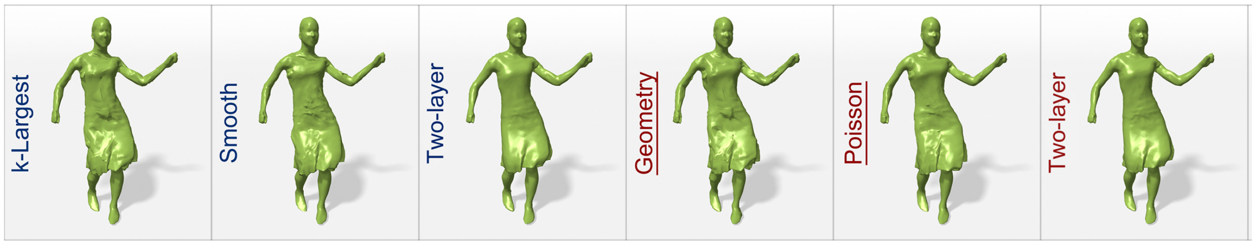

Figure 12: *The Samba model comparisons among selected skinning weight reduction methods, including k-largest weight reduction, smooth weight reduction [Landreneau and Schaefer 2010], geometric weight reduction [James and Twigg 2005], and Poisson weight reduction [Landreneau and Schaefer 2010], and two-layer sparse compression of dense skinning weights [Le and Deng 2013]. Image courtesy of [Le and Deng 2013]. Copyright ©ACM 2013.*

3.4 Poisson-based Weight Reduction

Directly ensuring the normal continuities of the weight-reduced deformations is often involved with the solving of a non-linear optimization problem, which is quite costly. With the aid of a set of example poses, the core idea of the Poisson-based weight reduction method [Landreneau and Schaefer 2010] is to minimize a Poisson equation defined over the surface [Yu et al. 2004]. Specifically, for cage-based deformation, let ϕ be an unknown function defined over the target surface, the Poisson equation is the optimal solution to the following problem:

$$\min_{\phi} \int_{\Omega} |\nabla\phi - g|^2 dA, \qquad \Leftrightarrow \qquad \min_{\phi} \int_{\Omega} |\nabla^2\phi - \nabla.g|^2 dA, \tag{41}$$

where Ω denotes the surface, g is a guidance field defined over the surface Ω, and ∇^2 is the Laplacian operator. Assuming g is also provided as a divergence of a scalar field ψ over the surface, Eq. (41) can be rewritten to the following form:

$$\min_{\phi} \int_{\Omega} |\nabla^2\phi - \nabla^2\psi|^2 dA, \qquad \Leftrightarrow \qquad \min_{\mathbf{w}} \sum_{\gamma=0}^{2}\sum_{t=1}^{S}\sum_{i=1}^{n} (\nabla^2(\sum_{j\in\mathfrak{B}_i} w_{ij}\mathbf{c}_{t,j}[\gamma]) - \nabla^2\mathbf{v}'_{t,i}[\gamma])^2, \tag{42}$$

where γ denotes a different component of the 3D position (x: $\gamma = 0$; y: $\gamma = 1$; z: $\gamma = 2$), and $\mathbf{c}_{t,j}$ denotes the position of the j-th control point of the t-th example.

For LBS case, the above minimization equation (Eq. (42)) need to be modified to the following form (often the non-negativity constraint is also applied to w_{ij} for LBS model):

$$\min_{\mathbf{w}} \sum_{\gamma=0}^{2}\sum_{t=1}^{S}\sum_{i=1}^{n} (\nabla^2(\sum_{j\in\mathfrak{B}_i} w_{ij}([\mathbf{R}_{t,j}|\mathbf{T}_{t,j}]\mathbf{v}_i)[\gamma]) - \nabla^2\mathbf{v}'_{t,i}[\gamma])^2. \tag{43}$$

The above equations (Eq. (42) and Eq. (43)), as constrained sparse quadratic equations, can be solved using Lagrange multipliers solver or the NNLS solver [Lawson and Hanson 1974] if the non-negativity constraint is enforced to $\{w_{ij}\}$ [Landreneau and Schaefer 2010].

As shown in the comparison example in Fig. 11, in general the Poisson-based weight reduction method can produce smoother (and less noticeable artifacts) than the other three weight reduction methods (i.e., k-largest weight reduction, smooth weight reduction, and geometric weight reduction). Fig. 13 shows another comparison example among the four weight reduction methods. However, for certain complex deformations (e.g, the Samba model in Fig. 12), we still can observe noticeable visual artifacts by the Poisson-based weight reduction method, especially compared with the two-layer sparse compression method (elaborated below).

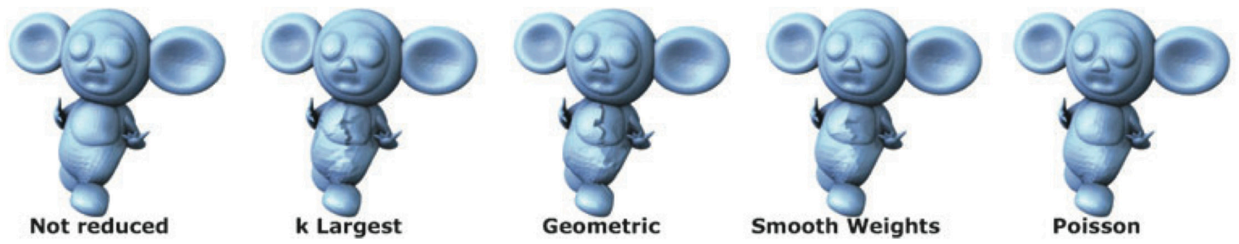

Figure 13: *Weight reduction comparisons on the Pinocchio model. (from left to right) The original deformed surface, reduced to 5 per vertex using k-largest weights, geometric reduction, smooth weight reduction, and Poisson-based weight reduction. Image courtesy of [Landreneau and Schaefer 2010]. Copyright ©Eurographics Association 2010.*

3.5 Two-layer Sparse Compression for Skinning Weight Reduction

The above weight reduction methods can greatly speed up the skinning efficiency (in particular, on the GPU); however, the visual quality of resulting skinning models is degraded noticeably, especially for 3D models with exceptional vertices. To balance the trade-off between the degraded visual quality and skinning efficiency, Le and Deng [2013] developed a two-layer sparse compression approach for dense-weight skinning models. It can be applied to both linear blend skinning and cage-based deformation. Its goal is to speed up the skinning efficiency significantly, with insignificant loss of its visual quality. The relation among this approach ("compressed"), the original skinning model ("dense LBS"), and the aforementioned skinning weight reduction approaches ("sparse LBS") can be illustrated in Fig. 14.

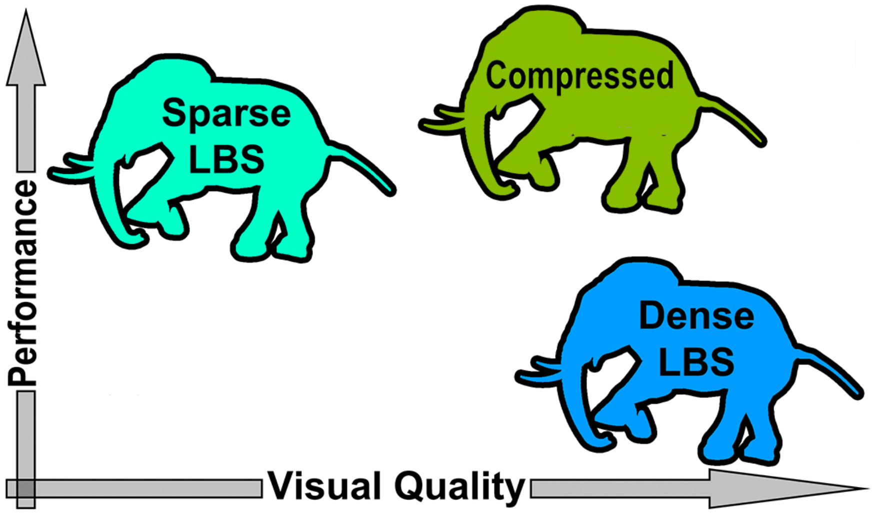

Figure 14: *The two-layer sparse compressed skinning model (green elephant) can achieve the good balance between visual quality and skinning efficiency. Image courtesy of [Le and Deng 2013]. Copyright ©ACM 2013.*

This approach constructs an effective two-layer blend model to reduce the computation of linear blend skinning (LBS) with dense weights. Specifically, its master bone blending layer blends the transformations of the original control bones (called *master bones*) and caches the results as virtual bone transformations. Then, its virtual bone blending layer blends the virtual bone transformations in a similar way to produce vertex transformations. At the virtual bone blending layer, each vertex transformation is blended by no more than two virtual bones. Compared with various existing (aforementioned) weight reduction techniques, it can achieve substantially smaller approximation errors than state of the art weight reduction techniques, given the same total number of bones. Fig. 16 illustrates the skinning quality and efficiency comparisons among the original dense-weight LBS, sparse LBS, and the two-layer sparse compression model.

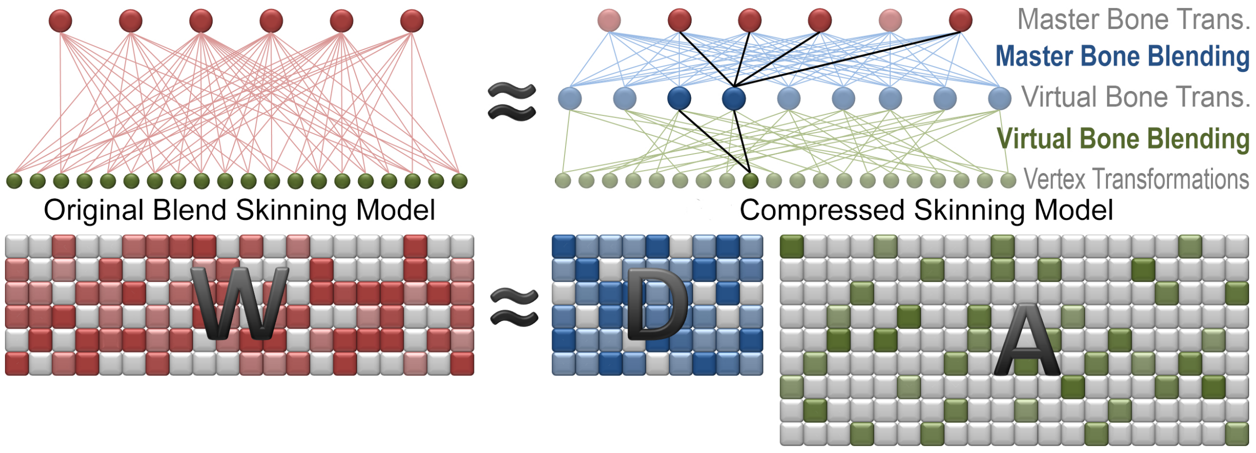

Figure 15: *A conventional blend skinning model (top left) with a dense weight matrix* $\mathbf{W}$ *(bottom left) is approximated as a two-layer blending with virtual bones (top right). This is equivalent to factorizing* $\mathbf{W}$ *into a sparse dictionary* $\mathbf{D}$ *and a matrix of sparse coefficients* $\mathbf{A}$ *(bottom right). Each column of* $\mathbf{D}$ *has at most* c *non-zero elements, while each column of* $\mathbf{A}$ *has at most 2 non-zero elements. Image courtesy of [Le and Deng 2013]. Copyright ©ACM 2013.*

3.5.1 Problem Formulation

Let $\mathbf{W} \in \mathbb{R}^{m \times n}$ be the weight matrix of an input skinning model with n vertices and m bones, as illustrated at the top left of Fig. 15, and let $\mathbf{w}_i$ denote the i-th column of $\mathbf{W}$ (or the original weights of the i-th vertex). The two-layer sparse compression approach compresses the original skinning model using a two-layer blending scheme with virtual bones (the top right of Fig. 15). At the first layer, a.k.a. *master bone blending*, it calculates and caches the transformations of q virtual bones by blending the transformations of m original bones (called *master bones*). Note that typically $q \ll n$. At the second layer, a.k.a. *virtual bone blending*, it calculates the position of each vertex by blending the transformations of the virtual bones and applying the resultant transformation to the vertex.

It also imposes a sparseness constraint on each blending layer to make the model friendly to parallel implementation on graphics hardware. Specifically, at the master bone blending layer, it allows at most c blending operations for each virtual bone; at the virtual bone blending layer, it allows at most 2 blending operations for each vertex (refer to Fig. 15).

Let $\mathbf{d}_j \in \mathbb{R}^m$ be the blending weights of the j-th virtual bone, we can represent all the master bone blending weights as a sparse matrix $\mathbf{D} = [\mathbf{d}_1, \ldots, \mathbf{d}_q] \in \mathbb{R}^{m \times q}$. Similarly, let $\mathbf{a}_i \in \mathbb{R}^q$ be the blending weights of the i-th vertex, it can represent all the virtual bone blending weights as another sparse matrix $\mathbf{A} = [\mathbf{a}_1, \ldots, \mathbf{a}_n] \in \mathbb{R}^{q \times n}$, where each column $\mathbf{a}_i$ has at most two non-zero elements. With the above matrix representations, the blend skinning compression problem can be elegantly reformulated as a sparse coding problem in which the original matrix $\mathbf{W}$ needs to be factorized into $\mathbf{D}$ (i.e., dictionary) and $\mathbf{A}$ (i.e., coefficients), as illustrated at the bottom of Fig. 15. Each column $\mathbf{d}_j$ is called an *atom* of the dictionary. This sparse coding problem can then be formulated as minimizing the following quadratic error function:

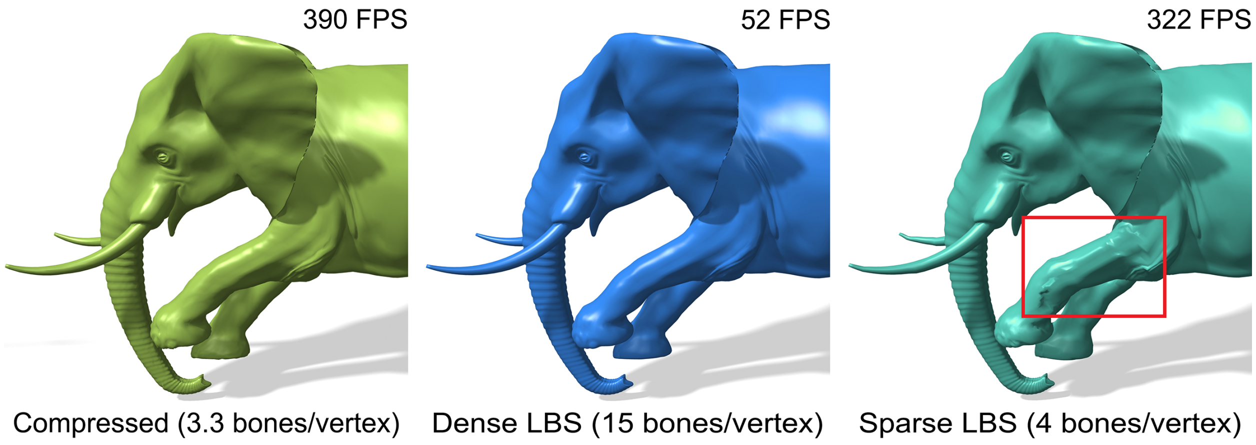
390 FPS
52 FPS
322 FPS
Compressed (3.3 bones/vertex)
Dense LBS (15 bones/vertex)
Sparse LBS (4 bones/vertex)

Figure 16: *The two-layer sparse compression model can compress Linear Blend Skinning (LBS) model with dense weights and generate a fast and compact model without sacrificing the quality of skinning, compared with dense-weight LBS model. Image courtesy of [Le and Deng 2013]. Copyright ©ACM 2013.*

$$\min_{\mathbf{D},\mathbf{A}} \Delta_{\mathbf{W}}{}^2 = \min_{\mathbf{D},\mathbf{A}} \frac{1}{mn} \|\mathbf{DA} - \mathbf{W}\|_F^2 \tag{44a}$$

$$\text{Subject to: } card(\mathbf{a}_i) \le 2, \forall i \tag{44b}$$

$$card(\mathbf{d}_i) \le c, \forall i \tag{44c}$$

In the above equations, $card(\mathbf{x})$ denotes the cardinality (number) of non-zero elements in vector $\mathbf{x}$. The constraint in Eq. (44b) enforces that at most two virtual bones can influence any particular vertex. Since the number of vertices is typically large, minimizing the number of blending operations (empirically choose it as two) at the virtual bone blending layer would be the most effective way to reduce computational cost. As shown in Eq. (44c), in order to maintain the approximation power of this model, it also needs to keep the sparseness of dictionary atoms (i.e., the sparseness of $\mathbf{D}$) to be no less than that of the original weights. As suggested by Le and Deng [2013], slightly increasing the sparseness of atoms could expand its approximation power. For this reason, c is empirically chosen as $\max_{i=1\ldots n} card(\mathbf{w}_i) + 1$. Note that any tuning on c can be performed to further optimize performance with different input data.

Compared with the aforementioned weight reduction methods (i.e., directly imposing a sparseness constraint on skinning weights), this two-layer sparse blending model essentially expands the linear blending space from *c-1* bone blending operations per vertex to $2c$ bone blending operations per vertex. Furthermore, virtual bones in this model can cache similar blending of master bones and save significant computational cost. The virtual bones can also be adaptively distributed in accordance with deformation complexity, e.g., the vertices on highly deformed regions could employ more virtual bones than those on other regions, as illustrated in Fig. 17.

3.5.2 Sparse Decomposition

Sparse decomposition (also called *sparse approximation*) is the estimation of a sparse vector that satisfies a linear system of equations, given multi-dimensional observed data and a dictionary matrix. Sparse decomposition techniques have been widely used in many applications including image processing, audio processing, biology, document analysis, and data compression and denoising. Here *sparsity* implies many zeros in a vector or a matrix.

Let $\mathbf{x} \in \mathbb{R}^n$ be a data point in a dataset called $\mathbf{X}$, assume $\mathbf{x}$ can be approximated as follows:

$$\mathbf{x} \approx \mathbf{Da}, \tag{45}$$

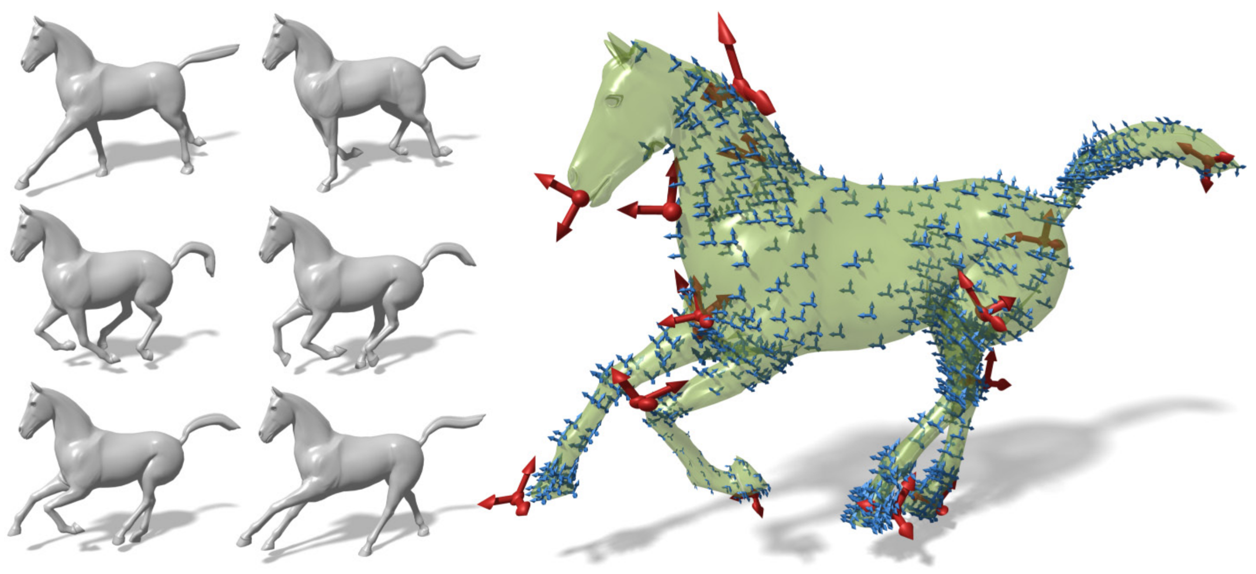

Figure 17: *Some example poses of an animated mesh sequence (left) and its corresponding two-layer compressed blend skinning model (right). Master bone transformations are illustrated in red, and virtual bone transformations are illustrated in blue. We place each virtual bone at a vertex with the largest sparse coefficient. The two-layer sparse compression model distributes virtual bones adaptively so that more virtual bones are employed for highly deformed regions (e.g., the legs or the tail). Image courtesy of [Le and Deng 2013]. Copyright ©ACM 2013.*

where $\mathbf{D} \in \mathbb{R}^{n \times m}$ denotes a *dictionary* matrix and $\mathbf{a} \in \mathbb{R}^m$ is a sparse vector. If $\mathbf{a}$ has at most k nonzero entries, it is called *k-sparse*. The dictionary $\mathbf{D}$ is *overcomplete* if $n < m$ (fewer rows than columns). The columns of $\mathbf{D}$ are called the *atoms* of the dictionary. The goal of sparse decomposition is to find a dictionary $\mathbf{D}$ so that each $\mathbf{x} \in \mathbf{X}$ has a sparse representation.

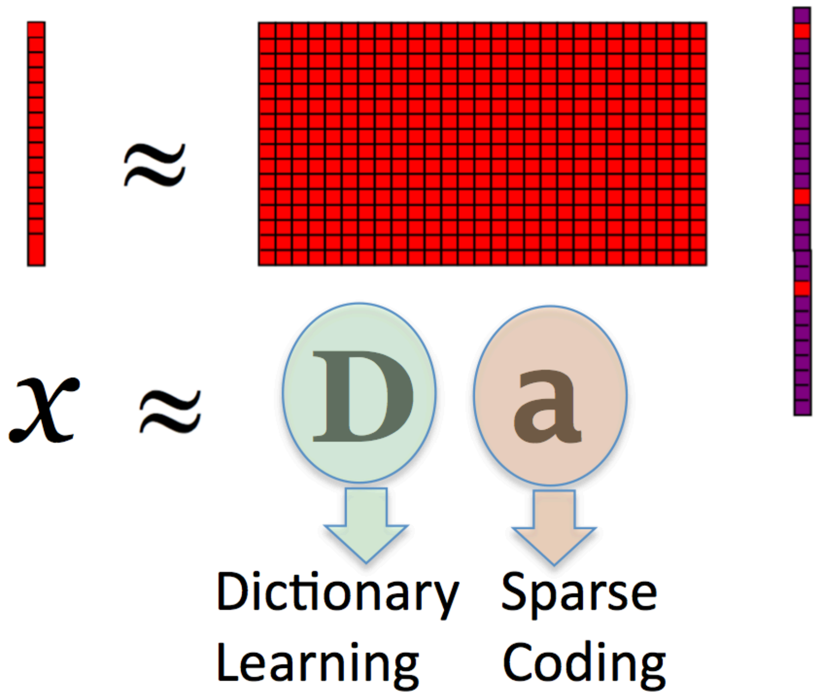

Figure 18: *Illustration of the sparse decomposition problem: it consists of two sub-problems - sparse coding and dictionary learning.*

As illustrated in Fig. 18, the sparse decomposition problem can be further divided to two subproblems: *sparse coding* and *dictionary learning*. Solving the optimal sparse vector α given the dictionary $\mathbf{D}$ is called *sparse coding*, while solving the optimal dictionary $\mathbf{D}$

given the sparse vector α is called *dictionary learning*.

Sparse coding. Now let's first look at the sparse coding model. It can be formulated as the minimization of the following objective function:

$$\min_{\mathbf{a}\in\mathbb{R}^m} \frac{1}{2}\|\mathbf{x}-\mathbf{D}\mathbf{a}\|_2^2 + \lambda\varphi(\mathbf{a}), \tag{46}$$

where the first term is called *data fitting* term, and the second term is called *regularization* term. The regularization term φ can be one of the following forms:

- l_2 norm, $\|\mathbf{a}\|_2^2 \triangleq \sum_{i=1}^m a_i^2$,
- l_1 norm, $\|\mathbf{a}\|_1 \triangleq \sum_{i=1}^m |a_i|$, and
- l_0 norm, $\|\mathbf{a}\|_0 \triangleq \#\{i|a_i \neq 0\}$.

The above l_0 norm and l_1 norm are also called *sparsity-inducing* norms. Over the years, a number of algorithms have been designed to solve the above sparse coding problem, including matching pursuit [Mallat and Zhang 1993], orthogonal matching pursuit [Tropp and Gilbert 2007; Cai and Wang 2011], LASSO algorithm [Mairal et al. 2010], projected gradient descent algorithm [Lin 2007], and so on. Since l_0 norm regularization is more relevant to the skinning weight reduction and compression, in the course note, we will focus on the sparse coding problem with l_0 norm regularization.

The sparse coding problem with l_0 norm regularization can often be reformulated as the following minimization problem.

$$\min_{\mathbf{a}\in\mathbb{R}^m} \frac{1}{2}\|\mathbf{x}-\mathbf{D}\mathbf{a}\|_2^2 \qquad \text{subject to} \quad \|\mathbf{a}\|_0 \leq L \tag{47}$$

Two classical algorithms (matching pursuit [Mallat and Zhang 1993] and orthogonal matching pursuit [Tropp and Gilbert 2007; Cai and Wang 2011]) can be used to solve the above minimization problem (Eq. (47)). The matching pursuit algorithm is described in **Algorithm** 5. A simple example of matching pursuit algorithm [Cao 2014] is illustrated in Fig. 19.

Algorithm 5 Matching Pursuit Algorithm

```
Require: a data point $\mathbf{x} \in \mathbb{R}^n$, a dictionary matrix $\mathbf{D} \in \mathbb{R}^{n\times m}$, the sparsity number $L$.
Ensure: a sparse vector $\mathbf{a}$ that satisfy Eq. (47).
1: Initialization: $\mathbf{a} = \mathbf{0}$, residual $\mathbf{r} = \mathbf{x}$.
2: repeat
3:     select the element with maximum correlation with $\mathbf{r}$: $\hat{\imath} = \arg\max_{i=1\cdots m} |\mathbf{d}_i^T\mathbf{r}|$.
4:     update the coefficients: $\mathbf{a}_{\hat{\imath}} = \mathbf{a}_{\hat{\imath}} + \mathbf{d}_{\hat{\imath}}^T\mathbf{r}$.
5:     update the residual: $\mathbf{r} = \mathbf{r} - (\mathbf{d}_{\hat{\imath}}^T\mathbf{r})\mathbf{d}_{\hat{\imath}}$.
6: until $\|\mathbf{a}\|_0 \geq L$
7: return $\mathbf{a}$
```

In the above basic (nonorthogonal) matching pursuit algorithm, the dictionary atoms are not mutually orthogonal vectors. Therefore, subtracting subsequent residuals from the previous one can reintroduce components that are not orthogonal to the span of the previously included atoms. In the orthogonal matching pursuit algorithm (**Algorithm** 6), the residual is always orthogonal to the dictionary atoms that are already selected.

Algorithm 6 Orthogonal Matching Pursuit Algorithm

```
Require: a data point $\mathbf{x} \in \mathbb{R}^n$, a dictionary matrix $\mathbf{D} \in \mathbb{R}^{n\times m}$, the sparsity number $L$.
Ensure: a sparse vector $\mathbf{a}$ that satisfy Eq. 47.
1: Initialization: $\mathbf{a} = \mathbf{0}$, residual $\mathbf{r} = \mathbf{x}$, active set $\omega = \varnothing$.
2: repeat
3:     select the element with maximum correlation with $\mathbf{r}$: $\hat{\imath} = \arg\max_{i=1\cdots m} |\mathbf{d}_i^T\mathbf{r}|$.
4:     update the active set: $\omega = \omega \cup \hat{\imath}$.
5:     update the coefficients: $\mathbf{a}_\omega = (\mathbf{D}_\omega^T\mathbf{D}_\omega)^{-1}\mathbf{D}_\omega^T\mathbf{r}$.
6:     update the residual: $\mathbf{r} = \mathbf{r} - \mathbf{D}_\omega\mathbf{a}_\omega$.
7: until $\|\mathbf{a}\|_0 \geq L$
8: return $\mathbf{a}$
```

Dictionary learning. A number of dictionary learning algorithms have been proposed, including K-SVD algorithm for dictionary learning with l_0 norm regularization [Aharon et al. 2006], and online dictionary learning for l_1 norm regularization [Mairal et al.

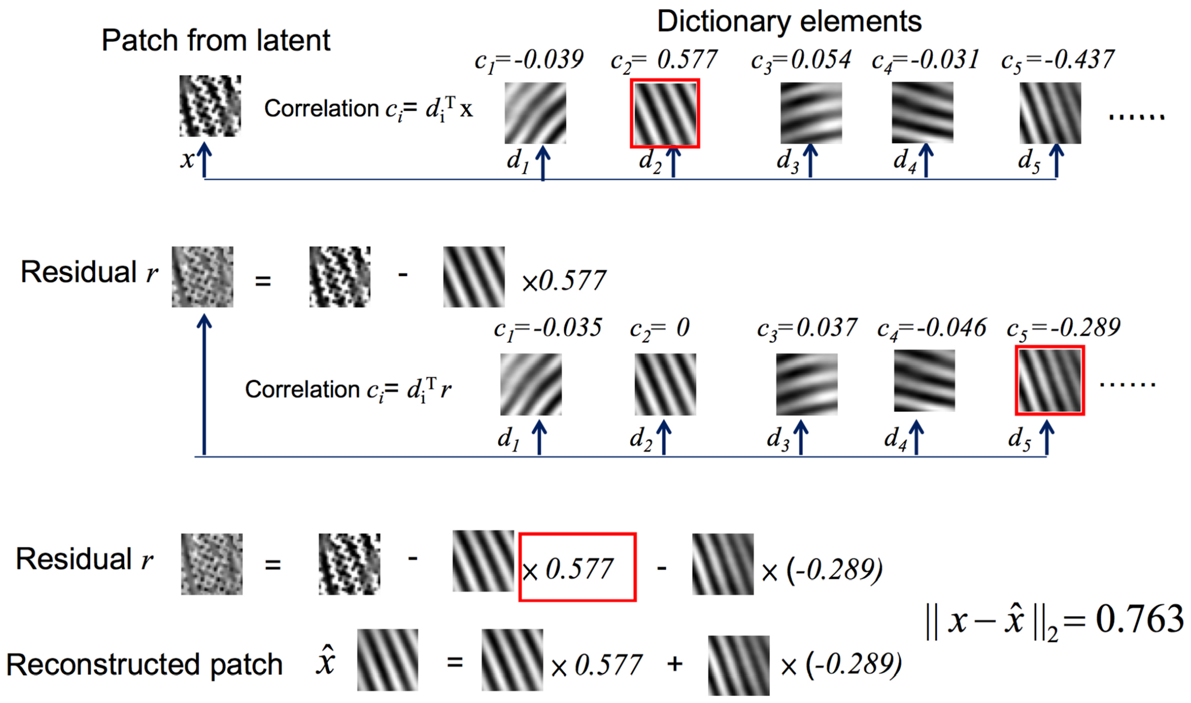
Patch from latent
Dictionary elements
Correlation $c_i = d_i^T x$
c_1=-0.039 c_2= 0.577 c_3=0.054 c_4=-0.031 c_5=-0.437
x
d_1 d_2 d_3 d_4 d_5
Residual r = - ×0.577
Correlation $c_i = d_i^T r$
c_1=-0.035 c_2= 0 c_3=0.037 c_4=-0.046 c_5=-0.289
Residual r = - × 0.577 - × (-0.289)
Reconstructed patch $\hat{x}$ = × 0.577 + × (-0.289)
$\| x - \hat{x} \|_2 = 0.763$

Figure 19: *An illustrative example of matching pursuit algorithm. Image courtesy of [Cao 2014].*

2010]. In this course note, we briefly describe the K-SVD algorithm below. The K-SVD algorithm iteratively perform sparse coding step and the dictionary update step until convergence.

(1) At the sparse coding stage, it minimizes the following objective function (assuming $\mathbf{D}$ is given).

$$\min_{\mathbf{A}} \|\mathbf{X} - \mathbf{DA}\|_F^2 \quad \forall j, \text{s. t. } \|\mathbf{a}_j\|_0 \leq L \tag{48}$$

Since coefficients of each example can be updated independently at this step, for the j-th example, $\mathbf{x}_i$, we just need to solve an ordinary sparse coding problem (using the above described matching pursuit algorithm or orthogonal matching pursuit algorithm), described below.

$$\min_{\mathbf{a}} \|\mathbf{X}_j - \mathbf{Da}\|_2^2 \quad \text{s. t. } \|\mathbf{a}\|_0 \leq L \tag{49}$$

(2) At the dictionary learning stage, it minimizes the following objective function (assuming $\mathbf{A}$ is given).

$$\min_{\mathbf{D}} \|\mathbf{X} - \mathbf{DA}\|_F^2 \quad \forall j, \text{s. t. } \|\mathbf{a}_j\|_0 \leq L \tag{50}$$

Its strategy is to update each atom d_k sequentially. For the k-th atom, the following minimization equation is solved.

$$\min_{\mathbf{d}_k} \|\mathbf{d}_k \mathbf{a}_k^T - \mathbf{E}_k\|_F^2, \quad \text{where the residual } \mathbf{E}_k = \sum_{i \neq k} (\mathbf{d}_i \mathbf{a}_i^T - \mathbf{X}) \tag{51}$$

The above Eq. (51) can be further solved with SVD [Aharon et al. 2006], as follows:

$$\mathbf{E}_k = \mathbf{U}\Lambda\mathbf{V} \quad \Rightarrow \quad \mathbf{d}_k = \mathbf{u}_1 \tag{52}$$

3.5.3 Solving of Sparse Matrix Factorization

Given the above formulated sparse matrix factorization problem (Eq. (44)), the next step of this approach is to efficiently solve this matrix factorization problem with sparseness constraints. A variety of general matrix factorization solvers have been proposed [Lee and Seung 1999; Mairal et al. 2010], including handle non-negativity constraint [Hoyer 2004]. Le and Deng [2013] proposed an iterative solver to speed up the above weight matrix factorization with sparseness constraints, described below in **Algorithm** 7.

Algorithm 7 Skinning Weight Matrix Factorization with Sparseness Constraints

```
Require: a weight matrix $\mathbf{W} = [\mathbf{w}_1, \dots, \mathbf{w}_n] \in \mathbb{R}^{m \times n}$, an error threshold $\varepsilon$ OR a maximum number of virtual bones $\Sigma$.
Ensure: $\mathbf{D} = [\mathbf{d}_1, \dots, \mathbf{d}_q] \in \mathbb{R}^{m \times q}$ and $\mathbf{A} = [\mathbf{a}_1, \dots, \mathbf{a}_n] \in \mathbb{R}^{q \times n}$ s.t. Eq. (44)
 1: Initialize a dictionary with $q = 2$ atoms: $\mathbf{D} = \{\mathbf{d}_1, \mathbf{d}_2\}$
 2: Initialize coefficients $\mathbf{A}$ according to $\mathbf{D}$
 3: repeat
 4:     for $t = 1 \to \kappa(\Delta_W, q)$ do
 5:         Find vertex $p$ with the largest approximation error
 6:         Add $\mathbf{w}_p$ to the dictionary
 7:         Update coefficients $\mathbf{A}$ from vertex $p$
 8:     end for
 9:     repeat
10:         Update dictionary $\mathbf{D}$
11:         for $i = 1 \to n$ do
12:             Update coefficients $\mathbf{A}$ from vertex $i$
13:         end for
14:     until Convergence
15: until $\Delta_W < \varepsilon$ OR $q = \Sigma$
16: return $\mathbf{D}$ and $\mathbf{A}$
```

The pipeline of the above algorithm 7 can be described as follows: First, a minimum dictionary with 2 atoms and its corresponding coefficients are initialized (line 1 and line 2). Then, atoms are sequentially added to the dictionary (line 4 to line 8) along with jointly optimizing the dictionary and coefficients (line 9 to line 14) until error $\Delta_W < \varepsilon$ or the size of the dictionary $q = \Sigma$.

(1) Initialization (line 1 and line 2). The first atom of the dictionary using the weights of a vertex with the largest ℓ^2-norm (i.e., $\mathbf{d}_1 = \arg\max_{\mathbf{w}_i} \|\mathbf{w}_i\|_2$). The second atom is initialized as the weights of another vertex with the smallest dot product to $\mathbf{d}_1$ (i.e., $\mathbf{d}_2 = \arg\min_{\mathbf{w}_i} \{\mathbf{w}_i \cdot \mathbf{d}_1\}$). The coefficients $\mathbf{A}$ can be solved per-vertex (column by column) by linear least squares with two unknowns (corresponding to two atoms) as follows: $\mathbf{a}_i = \arg\min_{\mathbf{x}} \|\mathbf{D}\mathbf{x} - \mathbf{w}_i\|_2^2$.

(2) Adding atoms (line 4 to line 8). In order to minimize Δ_W, the weights of a vertex p that has the largest error to the dictionary is always added as follows:

$$\mathbf{D} \leftarrow [\mathbf{D}, \mathbf{w}_p] \text{ s.t. } p = \arg\max_i \|\mathbf{D}\mathbf{a}_i - \mathbf{w}_i\|_2^2 \tag{53}$$

One straightforward way to add atoms would be to add atoms to the dictionary one by one and then perform a joint dictionary-coefficients optimization. However, it is inefficient; instead, the algorithm 7 adds a batch of $\kappa(\Delta_W, q)$ atoms (line 4) before each joint optimization. $\kappa(\Delta_W, q)$ is computed based on the current approximation error $\Delta_W(\mathbf{D}, \mathbf{A})$ (refer to Eq. (44a)) and the current dictionary size q (Eq. (54)). Note that when each atom is added without dictionary optimization, coefficients still need to be updated according to the added atom (line 7).

$$\kappa(\Delta_W, q) = \min\{\left(\frac{\Delta_W}{\varepsilon} - 1\right) q + 1, \Sigma - q\} \tag{54}$$

(3) Dictionary update (line 10). At this step, an online dictionary update algorithm with warm restarts [Mairal et al. 2010] is employed to do dictionary update due to its efficiency and simplicity. Specifically, Φ and Γ are first precomputed as follows:

$$\Phi = \sum_{i=1}^{n} \mathbf{a}_i \mathbf{a}_i^{\mathsf{T}} = [\phi_1, \dots, \phi_m] \in \mathbb{R}^{q \times q} \tag{55a}$$

$$\Gamma = \sum_{i=1}^{n} \mathbf{w}_i \mathbf{a}_i^{\mathsf{T}} = [\gamma_1, \dots, \gamma_m] \in \mathbb{R}^{m \times q} \tag{55b}$$

Then, each atom (column) $\mathbf{d}_j$ of the dictionary can be efficiently updated as follows:

$$\mathbf{d}_j \leftarrow \frac{1}{a_{jj}} (\gamma_j - \mathbf{D}\phi_j) + \mathbf{d}_j \tag{56}$$

After each atom update, the sparseness constraint in Eq. (44c) is enforced by keeping the c largest elements of vector d_j, while setting the others to be 0. Finally, the resulting d_j is normalized to satisfy the affinity constraint by dividing d_j by the sum of all the elements.

(4) Coefficients update (line 7 and line 12). The challenge of coefficients update for each vertex is to find two optimum dictionary atoms (i.e., virtual bones) that contribute to the vertex most. If the two optimum atoms are determined, the coefficients update becomes a trivial least squares problem with two unknowns. Assuming the affinity constraint is imposed, and r and s are the two identified optimum bones for updating coefficients of vertex i, then this least square problem can be described as follows, which can be solved straightforwardly:

$$\min_{\substack{(\mathbf{a}_i)_r \\ (\mathbf{a}_i)_s}} \|\mathbf{d}_r(\mathbf{a}_i)_r + \mathbf{d}_s(\mathbf{a}_i)_s - \mathbf{w}_i\|_2^2 \text{ s.t. } (\mathbf{a}_i)_r + (\mathbf{a}_i)_s = 1 \tag{57}$$

The work of [Le and Deng 2013] introduces a fast coefficients update method, by assuming skinning weights are typically smooth across neighboring vertices, that is, two neighboring vertices on a mesh might share the same optimum virtual bones. If the two optimum atoms for vertex i are updated, it is expected the two atoms might be the optimal ones for its neighboring vertices. Thus, the coefficients update step can be executed as a preorder graph traversal: the update process starts at a vertex with two known optimum dictionary atoms and use these atoms as candidates to update the coefficients of its neighboring vertices; then the same process is repeated for these neighboring vertices. Specifically, this recursive coefficients update can be implemented as a depth-first search on a mesh edge-based graph (Algorithm 8).

Algorithm 8 CoefficientsUpdate(vertex i, candidate atoms $\mathbf{d}_r$, $\mathbf{d}_s$)

```
Require: vertex $i$, candidate atoms $\mathbf{d}_r$, $\mathbf{d}_s$
1: if the linear combination of $\mathbf{d}_r$ and $\mathbf{d}_s$ improves error ${E_i}^2$ then
2:     Update $\mathbf{a}_i$ and ${E_i}^2 (= \|\mathbf{D}\mathbf{a}_i - \mathbf{w}_i\|_2^2)$ by linear least squares
3:     for all $j \in \mathcal{N}(i)$ do
4:         Find $\{r', s'\} \subset \{r, s, \rho_j, \sigma_j\}$ s.t. the linear combination of $\mathbf{d}_{r'}$ and $\mathbf{d}_{s'}$ best approximates $\mathbf{w}_j$
5:         CoefficientsUpdate($j$, $\mathbf{d}_{r'}$, $\mathbf{d}_{s'}$)
6:     end for
7: end if
```

(5) Dictionary-coefficients optimization (line 9 to line 14). The joint dictionary-coefficients optimization problem is solved by a block coordinate descent approach with warm restarts [Nocedal and Wright 2000], that is, alternatively updating the dictionary (line 10) and coefficients (line 11 to line 13). With the warm restarts, its alternative update process can converge within a small number of iterations, typically within 3 iterations.

3.5.4 Factorization with Example Poses

The above process does not utilize any example poses for skinning weight compression and reduction. However, if example poses are available, it can also utilize them for a better approximation. In order to utilize the example poses, it needs to find a compression model that best approximates given example poses, instead of a compression model that best approximates given dense skinning weights. Specifically, it minimizes a quadratic error function on all the example poses given corresponding bone transformations, by replacing the approximation error on skinning weights with the approximation error on example poses in the above sparse weight matrix factorization (Algorithm 7).

Given the example poses and bone transformations, the optimized linear blend skinning (LBS) weights $\mathbf{W}^* = [\mathbf{w}_1^*, \ldots, \mathbf{w}_n^*] \in \mathbb{R}^{m \times n}$ is first solved using linear least squares with equality constraint ($\sum_{j=1}^{m} (\mathbf{w}_i^*)_j = 1$) and inequality constraint ($\mathbf{w}_i^* \geq 0$) [Le and Deng 2012]. Note that the optimized weights $\mathbf{W}^*$ are not constrained to be sparse.

Then, the approximation error on weight matrix ${\Delta_W}^2$ (Eq. (44)) is replaced with the approximation error on example poses ${\Delta_E}^2$ (Eq. (58a)). The approximation error for each vertex ${E_i}^2$ (Eq. (58b)) is used to find new atoms for the dictionary (line 5 in Algorithm 7). Here, the approximation error is normalized by subtracting the lower bound ${E_i^*}^2$ (i.e., the approximation error with the optimized LBS weights $\mathbf{w}_i^*$, refer to Eq. (58c)) from it. Due to this modification, ${E_i}^2$ in Algorithm 8 is obtained through Eq. (58b), instead of the original $\|\mathbf{D}\mathbf{a}_i - \mathbf{w}_i\|_2^2$. Also, the optimized weights $\mathbf{W}^*$ (instead of $\mathbf{W}$) are used to pre-compute matrix Γ in Eq. (55b).

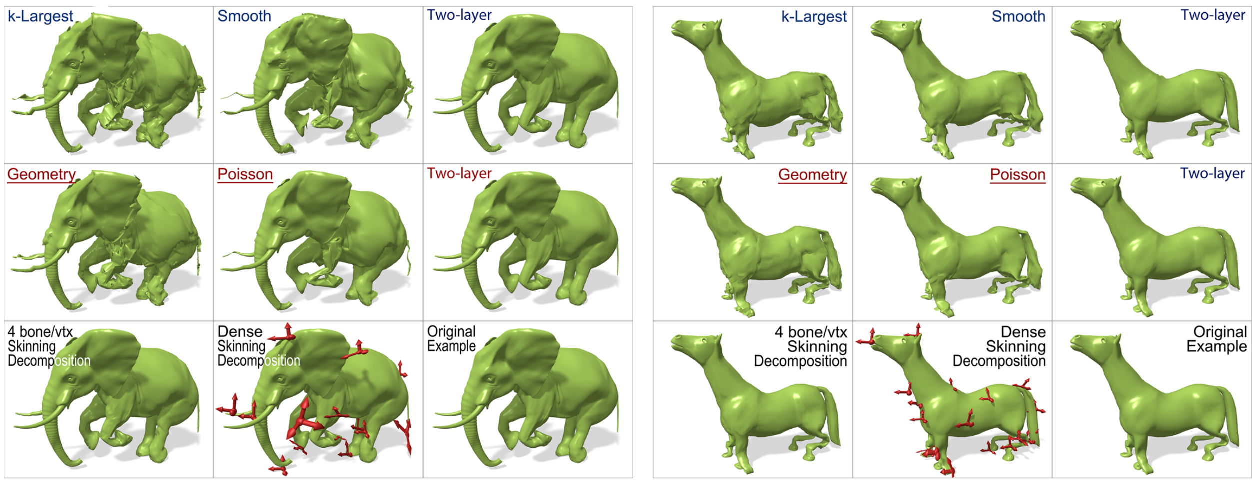

Figure 20: *Qualitative comparisons among the two-layer sparse compression method [Le and Deng 2013], k-largest weight reduction, smooth weight reduction [Landreneau and Schaefer 2010], geometric weight reduction [James and Twigg 2005], and Poisson weight reduction [Landreneau and Schaefer 2010]). The used models are elephant-gallop with 15 bones (top-left) and horse-collapse with 20 bones (top-right). In addition, "4 bones/vtx skinning decomposition" denotes the skinning model with 4 bones per vertex that is directly computed by the smooth skinning decomposition approach [Le and Deng 2012]. Image courtesy of [Le and Deng 2013]. Copyright ©ACM 2013.*

$$\Delta_E{}^2 = \frac{1}{3Sn}\sum_{i=1}^{n} E_i{}^2 \tag{58a}$$

$$\text{Where: } E_i{}^2 = \sum_{t=1}^{S}\left\|\sum_{j=1}^{m}(\mathbf{D}\mathbf{a}_i)_j[\mathbf{R}_{t,j}|\mathbf{T}_{t,j}]\mathbf{v}_i - \mathbf{v}'_{t,i}\right\|_2^2 - E_i^{*2} \tag{58b}$$

$$E_i^{*2} = \sum_{t=1}^{S}\left\|\sum_{j=1}^{m}(\mathbf{w}_i^*)_j[\mathbf{R}_{t,j}|\mathbf{T}_{t,j}]\mathbf{v}_i - \mathbf{v}'_{t,i}\right\|_2^2 \tag{58c}$$

3.6 Comparisons and Discussion of Weight Reduction and Compression Methods

Fig. 20 and Fig. 12 qualitatively compare all the above weight reduction and compression methods. The used 3D models are provided by [Sumner and Popović 2004; Vlasic et al. 2008]. All the methods reduce the skinning models with dense weights (obtained via the smooth skinning decomposition method [Le and Deng 2012]) to 4 bones per vertex. The methods noted with blue do not utilize example poses, while the methods noted with red utilize example poses. As shown in these two figures, all the methods reduce the skinning models with dense weights to 4 bones per vertex. We can further see that the weight reduction methods utilizing the example poses always give better approximations than the same methods without utilizing the example poses. In general, the results by the Poisson-based weight reduction method is smoother and more pleasing than those by the k-largest weight reduction, smooth weight reduction, and geometric weight reduction methods. Meanwhile, without noticeable visual distortions, the two-layer sparse compression method [Le and Deng 2013] approximates the original skinning models significantly better than the other four weight reduction methods (i.e., smooth, Poisson, k-largest, and geometric). In particular, when the example poses are utilized, the two-layer sparse compression method, with only 4 bone-blending operations per vertex, can compress and approximate the original models as good as dense-weight skinning models. This approximation is even better than the employed original skinning decomposition approach [Le and Deng 2012] with 4 bones per vertex, e.g. in the case of the elephant-gallop model.

Table 3 quantitatively compare all the above skinning weight reduction and compression methods. In this comparison, if example poses are not utilized, all the three methods (k-largest, smooth, and two-layer) only perform reduction on the skinning weight matrix. Two sets of input skinning models, generated by skinning decomposition [Le and Deng 2012], are used: skinning models with 8 bones per vertex and with dense bone-vertex weights. Using the three different methods, the skinning models with 8 bones per vertex are reduced to 4 bones per vertex, and the skinning models with dense weights are reduced to 4 bones per vertex and 8 bones per vertex, respectively. As shown in this table, regardless whether example poses are used, the two-layer sparse compression method [Le and Deng 2013] can substantially outperform the other four weight reduction methods in terms of approximation error. In most cases, the two-layer sparse compression method are even better than the employed original skinning decomposition approach [Le and Deng 2012] with the same number of bones.

References

AHARON, M., ELAD, M., AND BRUCKSTEIN, A. 2006. k-svd: An algorithm for designing overcomplete dictionaries for sparse representation. *Signal Processing, IEEE Transactions on 54*, 11, 4311–4322.

ALEXA, M., COHEN-OR, D., AND LEVIN, D. 2000. As-rigid-as-possible shape interpolation. In *Proceedings of SIGGRAPH 2000*, 157–164.

ANGUELOV, D., KOLLER, D., PANG, H.-C., SRINIVASAN, P., AND THRUN, S. 2004. Recovering articulated object models from 3D range data. In *UAI'04: Proc. of Conference on Uncertainty in Artificial Intelligence*, 18–26.

ANGUELOV, D., SRINIVASAN, P., KOLLER, D., THRUN, S., RODGERS, J., AND DAVIS, J. 2005. Scape: Shape completion and animation of people. *ACM Trans. Graph. 24*, 3 (July), 408–416.

BERTSEKAS, D. P. 1999. *Nonlinear Programming*, 2nd ed. Athena Scientific, Sept.

BJORCK, A. 1996. *Numerical Methods for Least Squares Problems*. Philadelphia.

BRICEÑO, H. M., SANDER, P. V., MCMILLAN, L., GORTLER, S., AND HOPPE, H. 2003. Geometry videos: a new representation for 3d animations. In *SCA'03: Proc. of Symposium on Computer Animation*, 136–146.

CAI, T. T., AND WANG, L. 2011. Orthogonal matching pursuit for sparse signal recovery with noise. *Information Theory, IEEE Transactions on 57*, 7, 4680–4688.

CAO, K., 2014. http://www.cse.msu.edu/∼cse902/s14/ppt/sparse%20coding%20and%20dictionary%20learning.pdf.

CHENG, Y. 1995. Mean shift, mode seeking, and clustering. *Pattern Analysis and Machine Intelligence, IEEE Transactions on 17*, 8, 790–799.

DE AGUIAR, E., THEOBALT, C., STOLL, C., AND SEIDEL, H.-P. 2007. Marker-less deformable mesh tracking for human shape and motion capture. In *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*, IEEE, 1–8.

DE AGUIAR, E., THEOBALT, C., THRUN, S., AND SEIDEL, H.-P. 2008. Automatic Conversion of Mesh Animations into Skeleton-based Animations. vol. 27.

DE AGUIAR, E., STOLL, C., THEOBALT, C., AHMED, N., SEIDEL, H.-P., AND THRUN, S. 2008. Performance capture from sparse multi-view video. *ACM Transactions on Graphics (TOG) 27*, 3, 98.

GILL, P., MURRAY, W., AND WRIGHT, M. 1981. *Practical Optimization*. Academic Press, London, UK.

GRIPPO, L., AND SCIANDRONE, M. 2000. On the convergence of the block nonlinear gauss–seidel method under convex constraints. *Operations Research Letters 26*, 3, 127–136.

GUSKOV, I., AND KHODAKOVSKY, A. 2004. Wavelet compression of parametrically coherent mesh sequences. In *SCA'04: Proc. of Symposium on Computer Animation*, 183–192.

HARKEGARD, O. 2002. Efficient active set algorithms for solving constrained least squares problems in aircraft control allocation. In *Proc. of the 41st IEEE Conference on Decision and Control*, IEEE, vol. 2, 1295–1300.

HASLER, N., THORMÄHLEN, T., ROSENHAHN, B., AND SEIDEL, H.-P. 2010. Learning skeletons for shape and pose. In *Proc. I3D*.

HORN, B. K. P. 1987. Closed-form solution of absolute orientation using unit quaternions. *Journal of the Optical Society of America 4*, 4.

HOYER, P. O. 2004. Non-negative matrix factorization with sparseness constraints. *The Journal of Machine Learning Research 5*, 1457–1469.

IGARASHI, T., MOSCOVICH, T., AND HUGHES, J. F. 2005. As-rigid-as-possible shape manipulation. *ACM Trans. Graph. 24*, 3, 1134–1141.

JAMES, D. L., AND TWIGG, C. D. 2005. Skinning mesh animations. *ACM Trans. Graph. 24*, 3, 399–407.

JOSHI, P., MEYER, M., DEROSE, T., GREEN, B., AND SANOCKI, T. 2007. Harmonic coordinates for character articulation. *ACM Trans. Graph. 26*, 3, 71.

JU, T., SCHAEFER, S., AND WARREN, J. 2005. Mean value coordinates for closed triangular meshes. *ACM Trans. Graph. 24*, 3, 561–566.

KABSCH, W. 1978. A discussion of the solution for the best rotation to relate two sets of vectors. *Acta Crystallographica Section A 34*, 827–828.

KAVAN, L., MCDONNELL, R., DOBBYN, S., ŽÁRA, J., AND O'SULLIVAN, C. 2007. Skinning arbitrary deformations. In *Proceedings of the 2007 symposium on Interactive 3D graphics and games*, ACM, 53–60.

KAVAN, L., SLOAN, P., AND O'SULLIVAN, C. 2010. Fast and efficient skinning of animated meshes. *Comput. Graph. Forum 29*, 2, 327–336.

KIM, B.-U., FENG, W.-W., AND YU, Y. 2010. Real-time data driven deformation with affine bones. *The Visual Computer 26*, 6-8, 487–495.

KRUSKAL, J. B. 1956. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proc. of the American Mathematical Society 7*, 1 (Feb.), 48–50.

KRY, P. G., JAMES, D. L., AND PAI, D. K. 2002. Eigenskin: real time large deformation character skinning in hardware. In *Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 153–159.

KUHN, H. W., AND TUCKER, A. W. 1951. Nonlinear programming. 418–492.

LANDRENEAU, E., AND SCHAEFER, S. 2010. Poisson-based weight reduction of animated meshes. *Comput. Graph. Forum 29*, 6, 1945–1954.

LAWSON, C. L., AND HANSON, R. J. 1974. *Solving least squares problems*, vol. 161. SIAM.

LE, B. H., AND DENG, Z. 2012. Smooth skinning decomposition with rigid bones. *ACM Trans. Graph. 31*, 6.

LE, B. H., AND DENG, Z. 2013. Two-layer sparse compression of dense-weight blend skinning. *ACM Trans. Graph. 32*, 4.

LE, B. H., AND DENG, Z. 2014. Robust and accurate skeletal rigging from mesh sequences. *ACM Trans. Graph. 33*, 4.

LEE, D. D., AND SEUNG, H. S. 1999. Learning the parts of objects by non-negative matrix factorization. *Nature 401*, 6755, 788–791.

LIN, C.-J. 2007. Projected gradient methods for nonnegative matrix factorization. *Neural computation 19*, 10, 2756–2779.

MAIRAL, J., BACH, F., PONCE, J., AND SAPIRO, G. 2010. Online learning for matrix factorization and sparse coding. *The Journal of Machine Learning Research 11*, 19–60.

MALLAT, S. G., AND ZHANG, Z. 1993. Matching pursuits with time-frequency dictionaries. *Signal Processing, IEEE Transactions on 41*, 12, 3397–3415.

MARQUARDT, D. W. 1963. An algorithm for least-squares estimation of nonlinear parameters. *SIAM Journal on Applied Mathematics 11*, 2, 431–441.

MOHR, A., AND GLEICHER, M. 2003. Building efficient, accurate character skins from examples. *ACM Trans. Graph. 22*, 3 (July), 562–568.

MÜLLER, M., HEIDELBERGER, B., TESCHNER, M., AND GROSS, M. 2005. Meshless deformations based on shape matching. *ACM Trans. Graph. 24*, 3 (July), 471–478.

NG, A. Y., JORDAN, M. I., AND WEISS, Y. 2001. On spectral clustering: Analysis and an algorithm. In *ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS*, MIT Press, 849–856.

NOCEDAL, J., AND WRIGHT, S. 2000. *Numerical Optimization*. Springer.

PARK, S. I., AND HODGINS, J. K. 2006. Capturing and animating skin deformation in human motion. In *ACM Transactions on Graphics (TOG)*, vol. 25, ACM, 881–889.

PINKALL, U., AND POLTHIER, K. 1993. Computing discrete minimal surfaces and their conjugates. *Experiment. Math. 2*, 1, 15–36.

SCHAEFER, S., AND YUKSEL, C. 2007. Example-based skeleton extraction. In *Proc. SGP*.

SNYMAN, J. 2005. *Practical mathematical optimization: an introduction to basic optimization theory and classical and new gradient-based algorithms*, vol. 97. Springer.

STOLL, C., GALL, J., DE AGUIAR, E., THRUN, S., AND THEOBALT, C. 2010. Video-based reconstruction of animatable human characters. In *ACM Transactions on Graphics (TOG)*, vol. 29, ACM, 139.

SUMNER, R., AND POPOVIĆ, J. 2004. Deformation transfer for triangle meshes. *ACM Trans. Graph. 23*, 3, 399–405.

TROPP, J. A., AND GILBERT, A. C. 2007. Signal recovery from random measurements via orthogonal matching pursuit. *Information Theory, IEEE Transactions on 53*, 12, 4655–4666.

TSENG, P. 2001. Convergence of a block coordinate descent method for nondifferentiable minimization. *Journal of optimization theory and applications 109*, 3, 475–494.

UTAH, 2013. The utah 3D animation repository.

VLASIC, D., BARAN, I., MATUSIK, W., AND POPOVIĆ, J. 2008. Articulated mesh animation from multi-view silhouettes. *ACM Transactions on Graphics (TOG) 27*, 3, 97.

VLASIC, D., PEERS, P., BARAN, I., DEBEVEC, P., POPOVIĆ, J., RUSINKIEWICZ, S., AND MATUSIK, W. 2009. Dynamic shape capture using multi-view photometric stereo. *ACM Transactions on Graphics (TOG) 28*, 5, 174.

YAMAUCHI, H., GUMHOLD, S., ZAYER, R., AND SEIDEL, H.-P. 2005. Mesh segmentation driven by gaussian curvature. *The Visual Computer 21*, 8-10, 659–668.

YU, Y., ZHOU, K., XU, D., SHI, X., BAO, H., GUO, B., AND SHUM, H.-Y. 2004. Mesh editing with poisson-based gradient field manipulation. In *ACM Transactions on Graphics (TOG)*, vol. 23, ACM, 644–651.

ZHU, Y., AND GORTLER, S. J. 2007. 3D deformation using moving least squares. Tech. Rep. Tr-10-07, Harvard University, Cambridge, MA.

ZOU, H., AND HASTIE, T. 2005. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society: Series B (Statistical Methodology) 67*, 2, 301–320.

Dataset$_{[\text{\# of bones}]}$	Approximation error E_{RMS}			Execution time (minutes)		
	SMA	LSSP	SSDR	SMA	LSSP	SSDR
camel-collapse$_{11}$	125.3 (4)	-	5.4(1.7)	13.8	-	7.4
camel-collapse$_{20}$	-	-	4(1.4)	-	-	15.1
camel-gallop$_{10}$	-	-	8.1(2.8)	-	-	6.1
camel-gallop$_{20}$	-	-	3.7(1.5)	-	-	13.9
camel-gallop$_{29}$	17.6 (1.9)	-	3(1.3)	25.2	-	21.9
camel-poses$_{10}$	-	-	10.1(3.9)	-	-	1
camel-poses$_{25}$	8.3 (1.8)	-	2.7(1.2)	9.4	-	4.6
cat-poses$_{5}$	-	-	26.3(12.5)	-	-	0.2
cat-poses$_{7}$	-	21.2(9.3)	21.2(10.3)	-	199	0.2
cat-poses$_{9}$	-	-	13.7(6.1)	-	-	0.3
cat-poses$_{10}$	-	12.6(5.6)	13(5.7)	-	240.8	0.3
cat-poses$_{12}$	-	-	8.6(3.8)	-	-	0.6
cat-poses$_{15}$	-	10.7(6.1)	6.5(2.7)	-	302.8	0.7
cat-poses$_{20}$	-	-	4.7(2)	-	-	1
cat-poses$_{25}$	8.5 (3.1)	6.2(3.3)	3.4(1.4)	0.7	371.7	1.5
chickenCrossing$_{10}$	-	20.9(14.9)	21.8(20.4)	-	1091.2	4.9
chickenCrossing$_{20}$	-	10.1(9.7)	8.7(5.7)	-	1128.9	14.8
chickenCrossing$_{28}$	12.5 (4.2)	6.2(5.1)	8.1(5.4)	14.1	1165.4	24
elephant-gallop$_{10}$	-	-	8.3(3.5)	-	-	15.7
elephant-gallop$_{20}$	-	-	4.3(2.3)	-	-	27.5
elephant-gallop$_{27}$	5.6 (1.9)	-	2.7(1.3)	56.1	-	53.6
elephant-poses$_{10}$	-	-	8.2(4.2)	-	-	3.2
elephant-poses$_{21}$	5.8 (2.2)	-	3.2(1.5)	29.4	-	8.1
face-poses$_{9}$	-	-	12(5.7)	-	-	2
face-poses$_{27}$	-	-	7(3.6)	-	-	7
face-poses$_{36}$	37.6 (8.5)	-	-	3.6	-	-
flamingo-poses$_{5}$	-	-	16.6(6.1)	-	-	0.8
flamingo-poses$_{10}$	-	-	4.8(2.2)	-	-	2
flamingo-poses$_{23}$	5.7 (1.7)	-	1.7(0.8)	16.3	-	5.1
horse-collapse$_{3}$	139.5 (5.1)	51(3.1)	26.5(4.1)	3.3	856.6	0.8
horse-collapse$_{10}$	-	15.5(2.6)	7.4(2.1)	-	802	2.6
horse-collapse$_{20}$	-	6(2)	5(1.6)	-	1088.5	5.7
horse-gallop$_{10}$	-	28.1(7.1)	8.3(3.6)	-	754.1	2.4
horse-gallop$_{20}$	-	15.7(5.3)	3.6(1.8)	-	859.9	5.4
horse-gallop$_{33}$	9.5 (1.5)	12.5(4.6)	2.2(1.1)	3.8	911	9.8
horse-poses$_{10}$	-	18.8(7.7)	9(3.9)	-	392.2	0.4
horse-poses$_{20}$	-	20.6(7.8)	3.8(1.8)	-	461.8	1.4
horse-poses$_{30}$	-	2.8(1.4)	2.2(0.9)	-	339.8	2.3
horse-poses$_{42}$	4.7 (1.4)	2.2(1.2)	-	2.1	475.1	-
lion-poses$_{10}$	-	22.1(11)	11.3(5)	-	201.3	0.2
lion-poses$_{21}$	62.8 (5.7)	7.7(3.9)	4.4(2.2)	0.6	360.2	0.8
pcow$_{10}$	-	16.5(13.2)	14.4(11.9)	-	549.2	2.4
pcow$_{24}$	24.8 (13.2)	7.2(6.7)	5.7(4.8)	3.8	564.5	8.9
pdance$_{10}$	-	8(4.9)	6.3(3.4)	-	2177.7	5.8
pdance$_{24}$	3.8 (1.6)	3.4(2.3)	1.3(0.8)	22	2446.8	28.3
pjump$_{10}$	-	-	10(6.5)	-	-	14.3
pjump$_{20}$	-	-	6.7(4.7)	-	-	42.7
pjump$_{30}$	-	-	5.3(3.9)	-	-	72.9
pjump$_{40}$	15.3 (6.7)	-	4.5(3.4)	30.5	-	104.1
pkanga$_{10}$	-	44.5(11.5)	17(7)	-	247.1	1.5
pkanga$_{20}$	-	32(8.5)	8.9(4.5)	-	308.5	3.2
pkanga$_{30}$	-	26.2(7.6)	7.8(4.1)	-	320.1	5.3
pkanga$_{39}$	134.1 (15.4)	22.4(7)	6.7(4.2)	1.6	360.7	7.6

Table 1: *Rigid bone skinning decomposition results by the SMA [James and Twigg 2005], LSSP [Hasler et al. 2010], and the SSDR algorithm [Le and Deng 2012]. The number of bones is denoted as the subscript of the dataset name. All the running times were measured on the same computer with a 2GHz single core CPU. Note that the number of bones is automatically estimated in SMA, thus its results are not available for many specific numbers of bones. The results of LSSP are also not available for models with more than 10K vertices due to the large memory requirement in the Self-Tuning Spectral Clustering algorithm at the initialization step. The results after rank-5 EigenSkin correction [Kry et al. 2002] are also reported in the parentheses.*

Dataset	N	F	B	Method I		Method II		Method III		Method IV	
				Time	RMSE	Time	RMSE	Time	RMSE	Time	RMSE
cat-poses	7207	9	28	5.8	**0.25**	0.1	0.68	6.9	1.04	17.2	0.63
horse-poses	8431	10	27	7.7	**0.21**	0.2	0.54	6.2	1.24	20.0	0.75
lion-poses	5000	9	30	4.1	**0.27**	0.1	0.83	4.0	1.62	11.7	1.14
horse-gallop	8431	48	27	41.9	**0.22**	0.8	0.44	33.3	1.10	80.3	0.88
hand	7997	43	18	65.1	**0.18**	0.6	0.23	20.0	0.42	41.9	**0.18**
dance	7061	201	16	148.7	**0.22**	2.5	0.76	61.8	0.78	168.0	0.53
scape	12500	70	23	252.1	**0.42**	1.7	1.03	60.7	1.18	410.4	1.24
samba	9971	175	22	348.2	**0.56**	3.3	1.29	95.1	1.57	296.0	1.79
cow	2904	204	11	72.3	**1.52**	1.0	5.41	16.0	5.61	47.9	5.58

Table 2: *Quantitative comparisons among all the four methods. The reported RMSE is normalized by the bounding volume diagonal [Schaefer and Yuksel 2007; Hasler et al. 2010]. Specifically,* $RMSE = 100 \times \sqrt{E_D}/d$*, where* E_D *is the data fitting error in Eq.* (24b)*, and* d *is the diagonal of the bounding box of the rest pose. The error is computed on the output using the joint rotation representation to strictly enforce the joint constraints. The running time (in minutes) was recorded on the same off-the-shelf computer with an Intel Xeon E5405 2.0GHz CPU. All the methods in this comparison were implemented in C++ with single thread.*

Name$_{[m]}$	8 to 4 bones/vertex						Dense to 4 bones/vertex						4 b/v	Dense to 8 bones/vertex						8 b/v	Dense
	k-Largest	Smooth	Two-layer	Geometric	Poisson	Two-layer	k-Largest	Smooth	Two-layer	Geometric	Poisson	Two-layer	SD	k-Largest	Smooth	Two-layer	Geometric	Poisson	Two-layer	SD	SD
samba$_{10}$	10.6	11.5	**5**	9.4	10.5	**4.8**	11.8	12.7	**4.8**	10.5	11.6	**4.7**	5.7	5.1	5.3	**4.8**	4.8	5	**4.8**	4.9	4.8
samba$_{20}$	5.7	6.5	**2.3**	5	5.8	**2.1**	7.6	8.3	**2.4**	6.8	7.6	**2**	2.7	3.8	4.4	**2**	3.1	3.7	**1.9**	2.2	1.9
camel-gallop$_{15}$	8.3	9.7	**1.6**	6	8	**1.4**	13.1	14.9	**1.5**	9.5	11.9	**1.3**	2.3	5	6.5	**1.3**	3.2	4.2	**1.3**	1.6	1.3
camel-gallop$_{30}$	8.1	9.4	**1**	4.9	7.6	**0.8**	13.8	15.5	**2**	8.3	10.6	**0.8**	1.3	7.5	9.2	**0.7**	3.7	5.4	**0.4**	0.8	0.3
elephant-gallop$_{15}$	11.5	13.8	**2.1**	9.1	12.4	**1.9**	18.7	21.2	**1.8**	13.7	16.9	**1.6**	3.2	9	10.9	**1.7**	5.2	7.1	**1.6**	2	1.6
elephant-gallop$_{30}$	4.5	5.4	**1**	3.3	4.8	**0.8**	13	15.2	**1.5**	9.1	11.3	**0.7**	1.5	6.6	8.3	**0.7**	4.2	5.8	**0.5**	0.9	0.4
horse-gallop$_{15}$	9.5	10.3	**2.6**	7	8.2	**2.3**	20.7	23.5	**2.4**	10.1	11.5	**2**	3.2	5.9	6.9	**1.9**	3.9	4.7	**1.9**	2.5	1.9
horse-gallop$_{30}$	5.4	5.7	**1.2**	3.8	4.5	**1**	18.1	20.8	**3.4**	12.3	14.7	**1.6**	1.8	9.6	11.6	**1.6**	5.3	6.7	**0.8**	0.9	0.5
camel-collapse$_{20}$	10.6	12.4	**1.8**	6.2	8.4	**1.5**	15.5	17.4	**1.9**	10.3	13.4	**1.4**	2.8	7.2	9.5	**1.2**	3.6	5.4	**1.1**	1.5	1.1
camel-collapse$_{40}$	6.9	7.6	**1.6**	4	5.8	**1.2**	15.1	16.4	**3.1**	9.8	12.5	**1.5**	2.1	9.2	11	**1.5**	4.5	6.5	**0.8**	1.1	0.5
horse-collapse$_{20}$	12	13.4	**3**	7.2	9	**2.1**	17.9	20.3	**3.5**	11	13.5	**2.1**	3.2	8.4	10.5	**1.9**	4.2	5.5	**1.4**	1.9	1.3
horse-collapse$_{40}$	8.2	8.4	**2.8**	4.4	5.6	**2**	17.8	20	**4.8**	10.1	12.6	**2.5**	2.3	10.9	12.8	**2.9**	4.9	6.4	**1.3**	1.3	0.6

Table 3: *Comparison of the fitting errors on example poses* (E) *among the two-layer sparse compression method, k-largest weight reduction, geometric weight reduction method [James and Twigg 2005], the smooth weight reduction method [Landreneau and Schaefer 2010], and the Poisson weight reduction method [Landreneau and Schaefer 2010]. "SD" denotes the employed original skinning decomposition method [Le and Deng 2012]. "b/v" denotes bones per vertex. The methods noted with blue do not utilize example poses, while the methods noted with red utilize example poses. The numbers in blue denote results without utilizing example poses, while the numbers in red with underline denote results with utilizing example poses. The number of bones* m *is shown as the subscript of the input model name. For the sake of convenience, all the errors in this table are multiplied by 1000.*